



Ingeniería de los Computadores

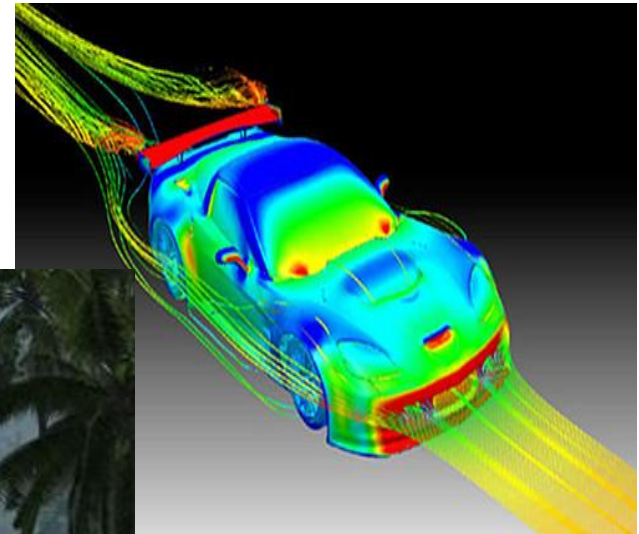
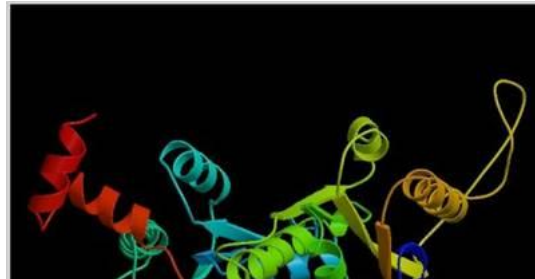
Unidad 1. Introducción al paralelismo

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

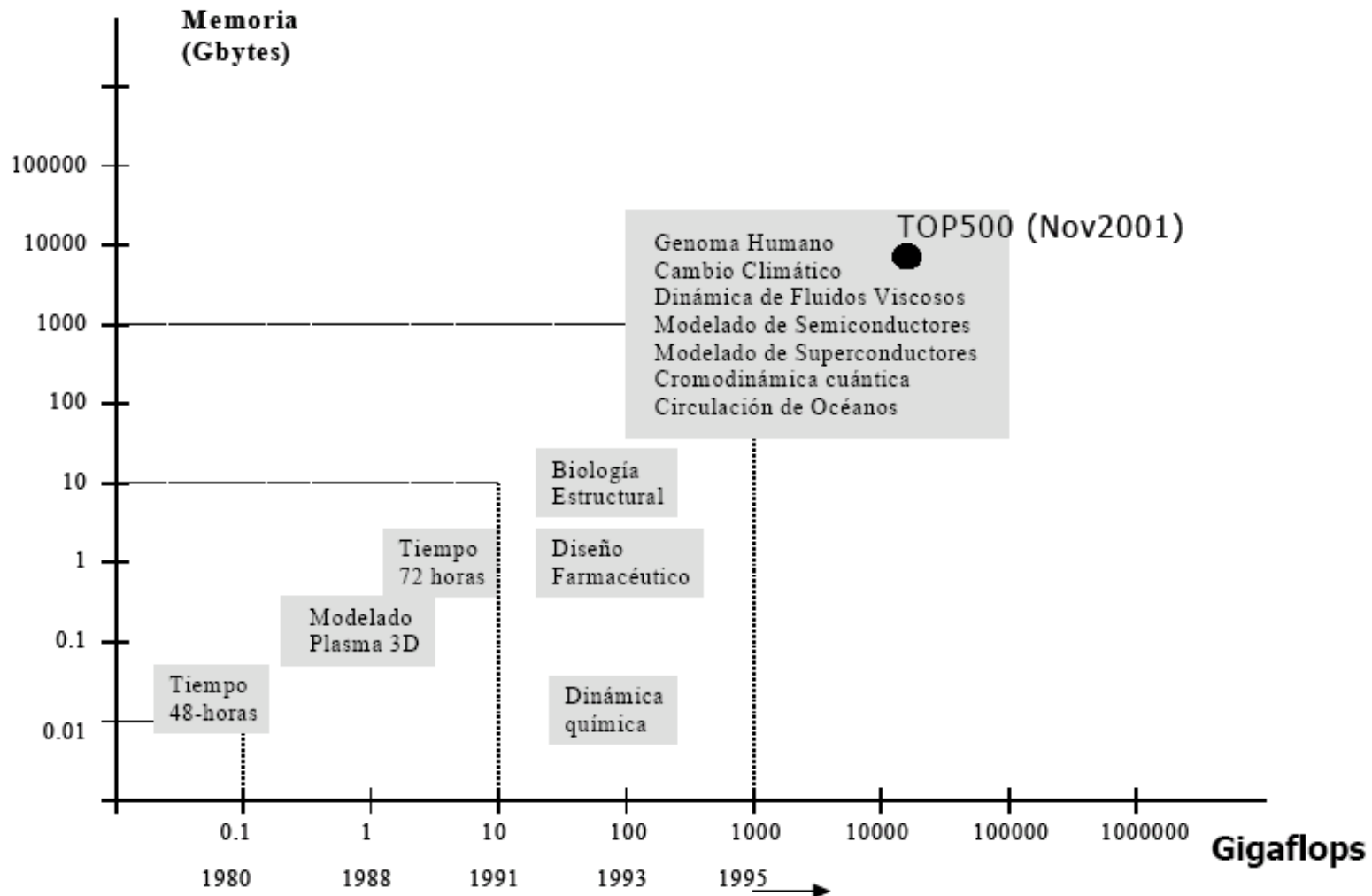
- Evolución de la informática ↔ Necesidades computacionales



Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

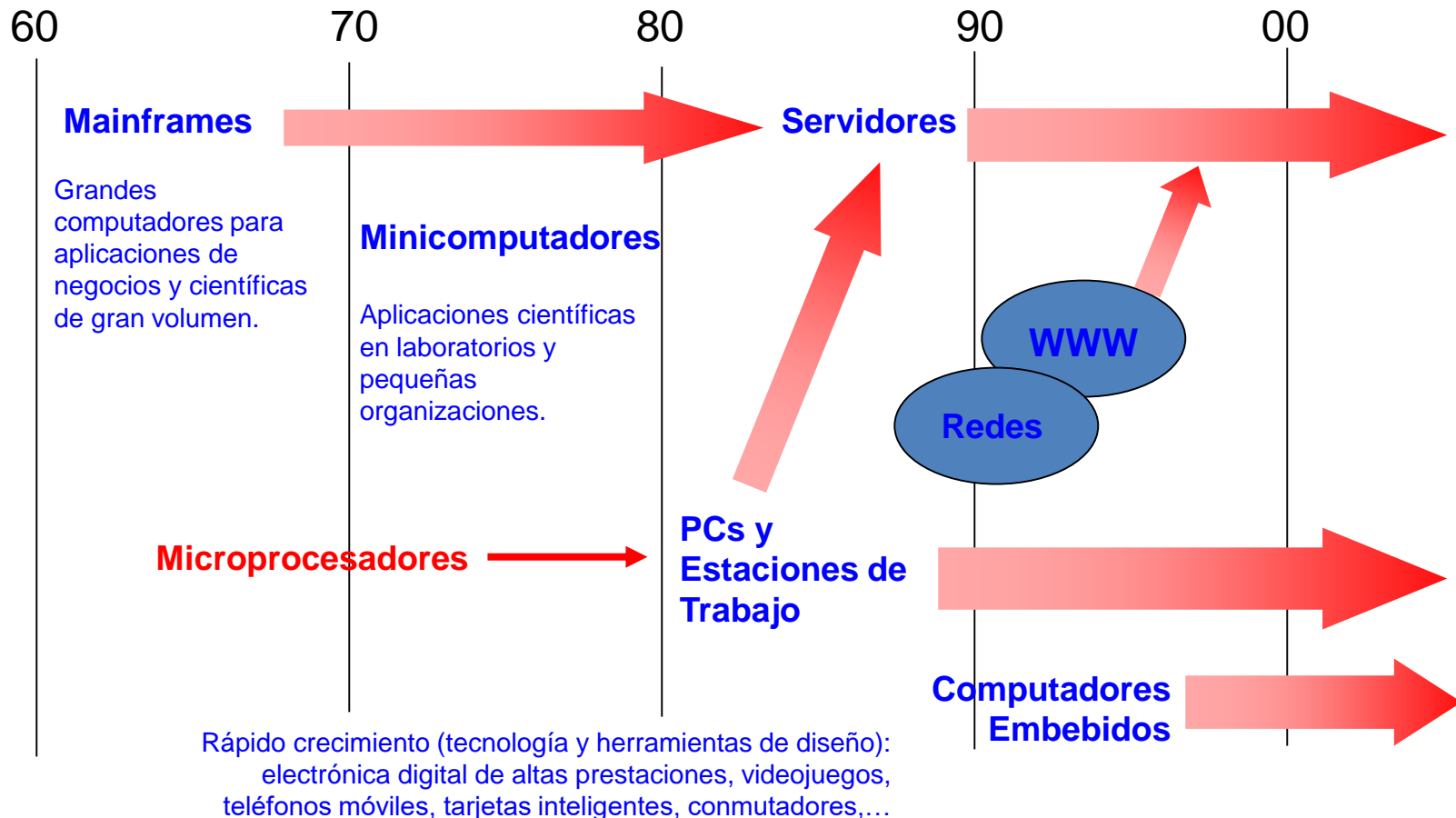
¿Qué?



Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

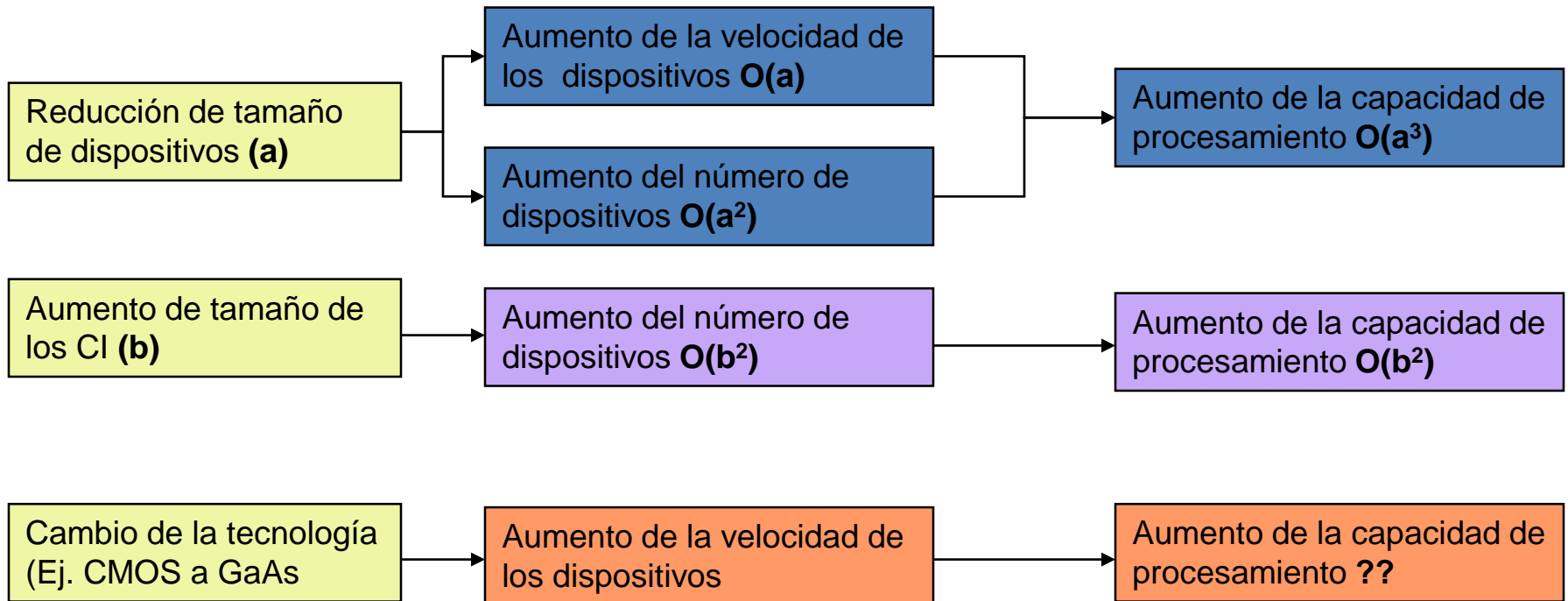


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Mejora de prestaciones
 - Avances en tecnologías (límites físicos: calor, ruido, etc.)



Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Mejora de prestaciones
 - Avances en arquitecturas
 - **Paralelismo:**
 - Segmentación de cauces.
 - Repetición de elementos: Utilizar varias unidades funcionales, procesadores, módulos de memoria, etc. para distribuir el trabajo.
 - **Localidad:** Acercar datos e instrucciones al lugar donde se necesitan para que el acceso a los mismos sea lo más rápido posible (jerarquía de memoria).

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Desarrollo de las arquitecturas de computadores - Objetivo

MAYOR CAPACIDAD COMPUTACIONAL

Iniciativas mayor peso software

- Repertorio de instrucciones (RISC, CISC, ...)
- Arquitecturas VLIW
- Extensiones SIMD (MMX, SSE, 3DNOW, ...)

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Desarrollo de las arquitecturas de computadores - Objetivo

MAYOR CAPACIDAD COMPUTACIONAL

Iniciativas mayor peso hardware

- Arquitecturas segmentadas
- Arquitecturas vectoriales
- Arquitecturas superescalares
- Arquitecturas paralelas o de alto rendimiento

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Arquitectura de Computadores

“Conjunto de instrucciones, recursos y características del procesador que son visibles al software que se ejecuta en el mismo. Por tanto, la arquitectura determina el software que el procesador puede ejecutar directamente, y esencialmente define las especificaciones a las que debe ajustarse la microarquitectura” [Ortega, 2005]

- En Ingeniería de Computadores veremos:
 - Arquitecturas superescalares
 - Arquitecturas paralelas: multicomputadores y multiprocesadores

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- **Ámbito de la arquitectura de computadores**
 - El lenguaje máquina del computador, la microarquitectura del procesador y la interfaz para los programas en lenguaje máquina (lenguaje máquina y arquitectura concreta del procesador).
 - Los elementos del computador y como interactúan (es decir la arquitectura concreta del computador, la estructura y organización).
 - La interfaz que se ofrece a los programas de alto nivel y los módulos que permiten controlar el funcionamiento del computador (sistema operativo y la arquitectura abstracta del computador).
 - Los procedimientos cuantitativos para evaluar los sistemas (benchmarking).
 - Las alternativas posibles y las tendencias en su evolución

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Niveles estructurales de Bell y Newell
 - Descripción del computador mediante una aproximación por capas.
 - Cada capa utiliza los servicios que proporciona la del nivel inferior.
 - Propone 5 niveles:
 - De componente
 - Electrónico
 - Digital
 - Transferencia entre registros (RT)
 - Procesador-Memoria-Interconexión (PMS)

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Niveles de interpretación de Levy
 - Contemplan al computador desde un punto de vista funcional.
 - Constituido por una serie de máquinas virtuales superpuestas.
 - Cada máquina interpreta las instrucciones de su nivel, proporcionando servicios a la máquina de nivel superior y aprovechando los de la máquina de nivel inferior.
 - Se distinguen 5 niveles:
 - Aplicaciones
 - Lenguajes de alto nivel
 - Sistema Operativo
 - Instrucciones máquina
 - Microinstrucciones
- Estos niveles son similares a los niveles funcionales de Tanenbaum

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Niveles de abstracción de un computador

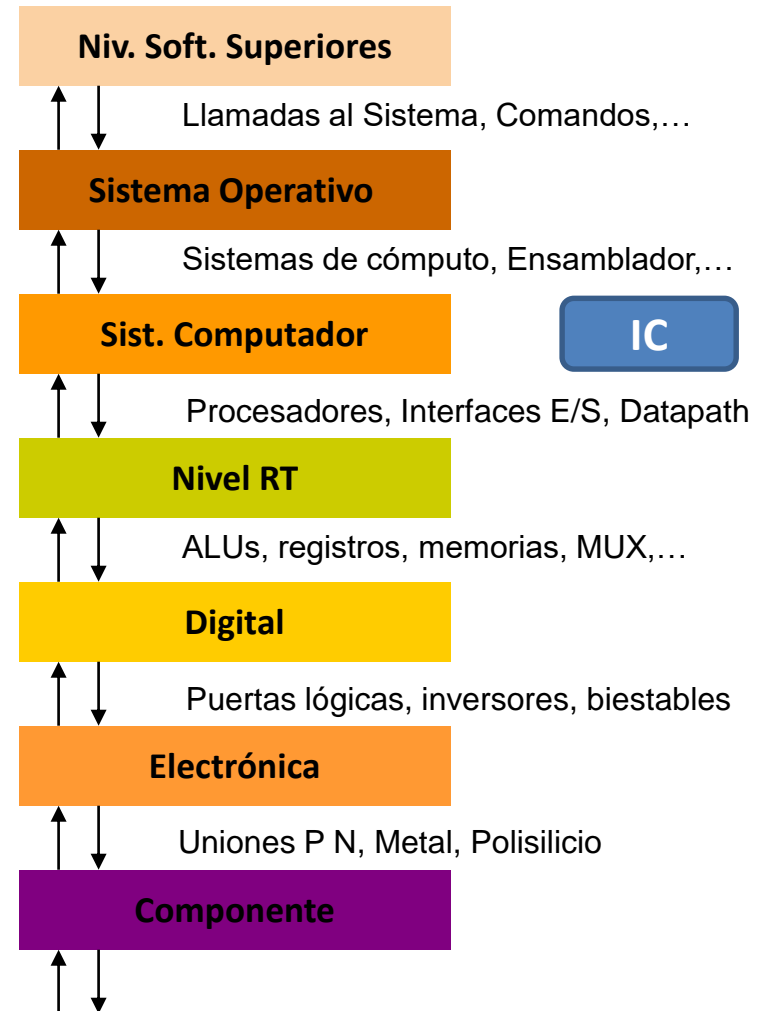
Integra la orientación **estructural** de los niveles de Bell y Newell y el punto de vista **funcional** de los niveles de Levy y Tanenbaum.

SW

ARQUITECTURA

TECNOLOGÍA

HW



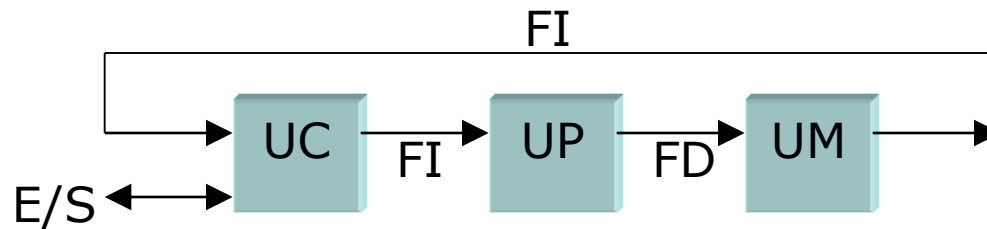
Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

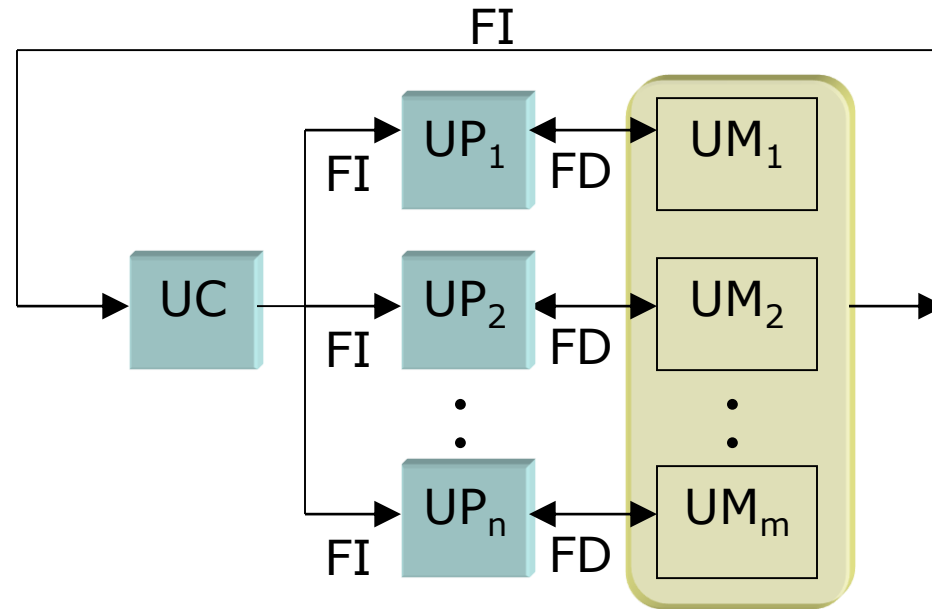
¿Qué?

- Taxonomía de Flynn

- SISD



- SIMD

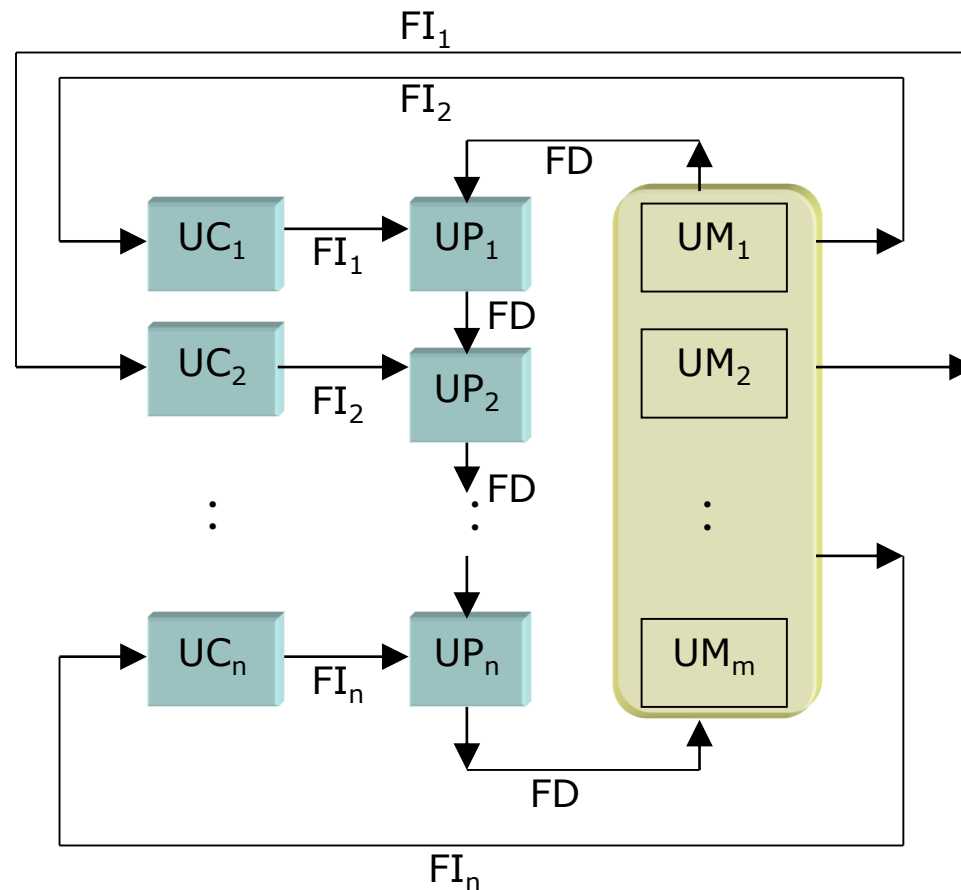


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Taxonomía de Flynn
 - MISD

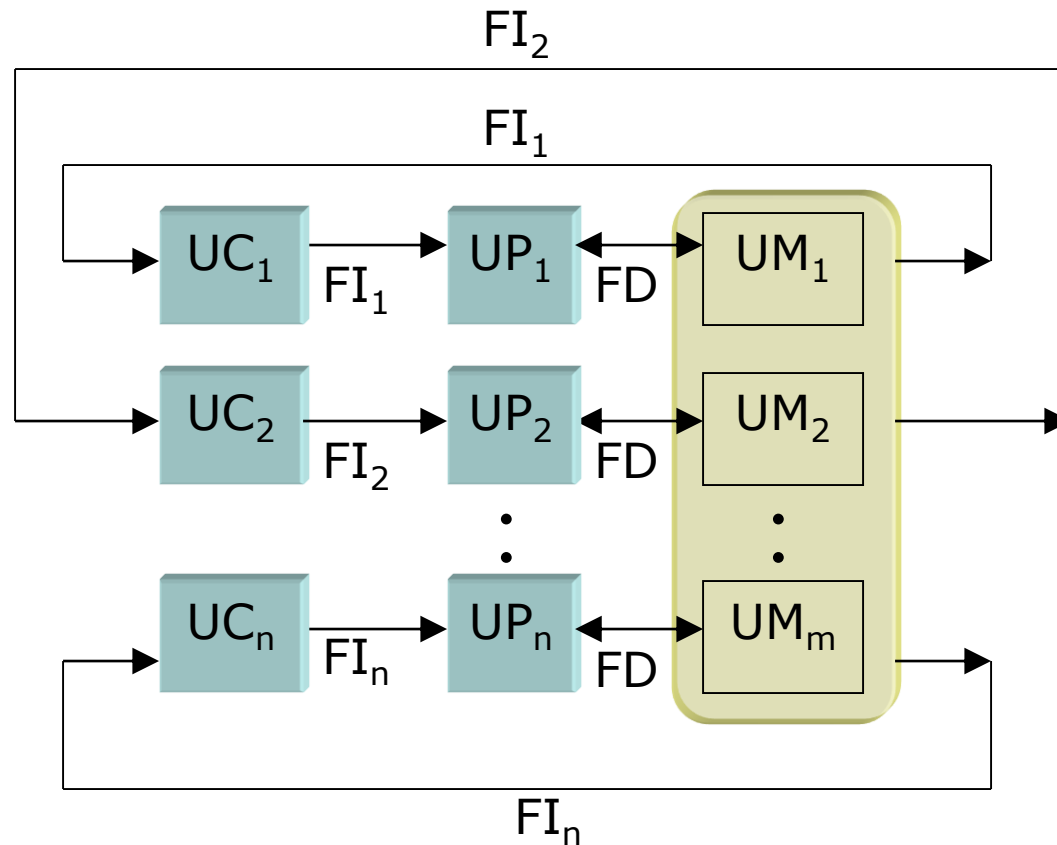


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Taxonomía de Flynn
 - MIMD

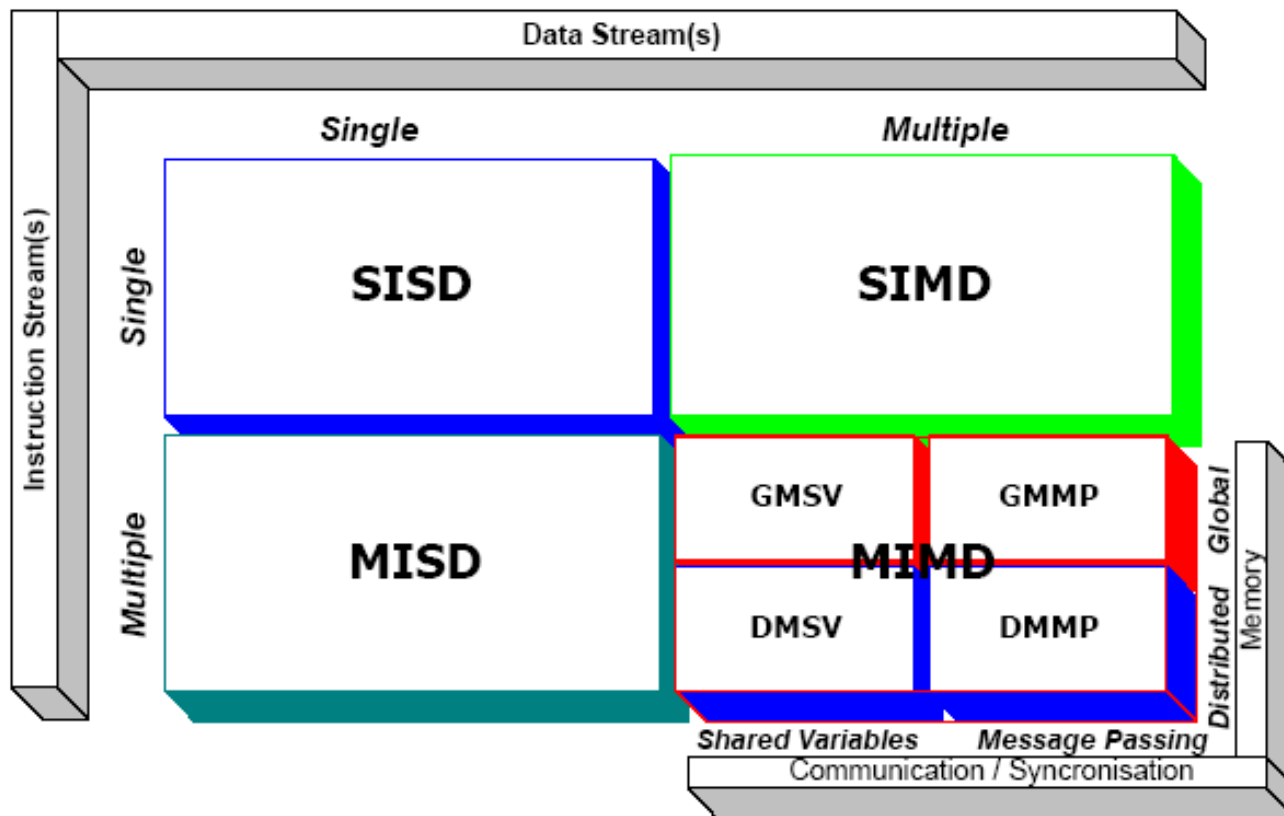


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Taxonomía de Flynn-Johnson



Tipos de paralelismo

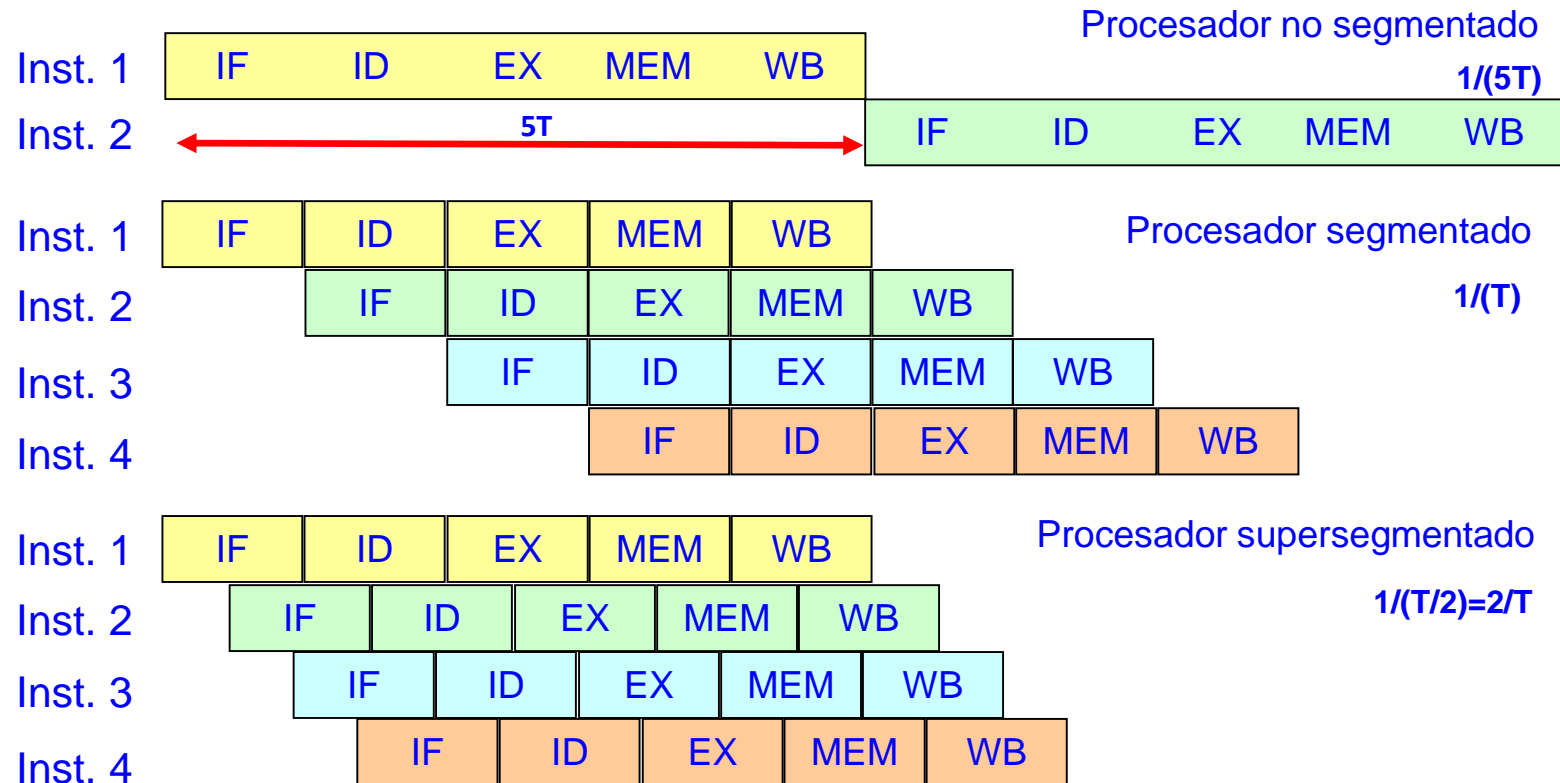
- Tipos de paralelismo
 - **Paralelismo de datos:** La misma función, instrucción, etc. se ejecuta en paralelo pero en cada una de esas ejecuciones se aplica sobre un conjunto de datos distinto.
 - **Paralelismo funcional:** Varias funciones, tareas, instrucciones, etc. (iguales o distintas) se ejecutan en paralelo.
 - Nivel de instrucción (ILP) – se ejecutan en paralelo las instrucciones de un programa. Granularidad fina.
 - Nivel de bucle o hebra (Thread) – se ejecutan en paralelo distintas iteraciones de un bucle o secuencias de instrucciones de un programa. Granularidad fina/media.
 - Nivel de procedimiento (Proceso) – distintos procedimientos que constituyen un programa se ejecutan simultáneamente. Grano medio.
 - Nivel de programa – la plataforma ejecuta en paralelo programas diferentes que pueden corresponder, o no, a una misma aplicación. Granularidad gruesa.

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Segmentación: ILP



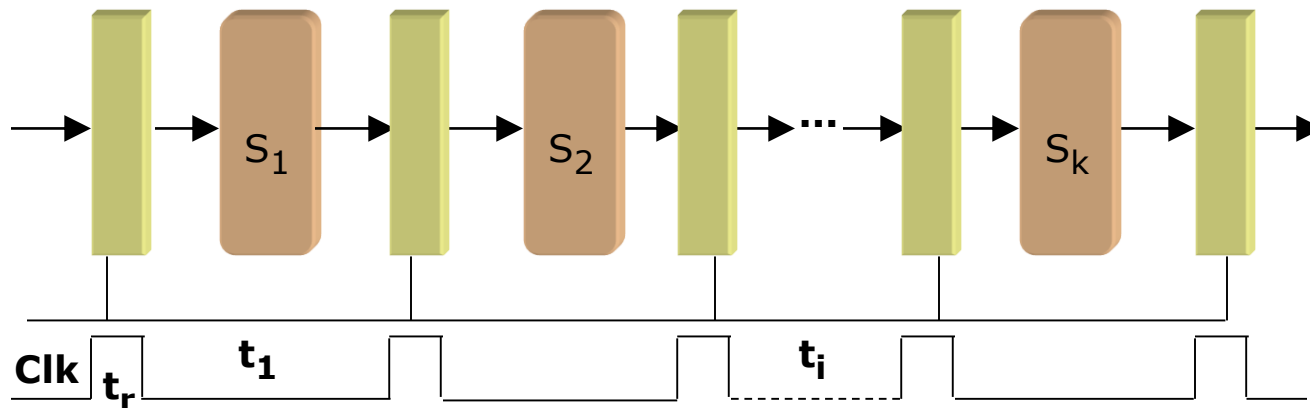
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Segmentación

- Identificación de fases en el procesamiento de una tarea.
- Rediseño para implementar cada fase de forma independiente al resto.
- Paralelismo por etapas (el sistema procesa varias tareas al mismo tiempo aunque sea en etapas distintas).
- Se aumenta el número de tareas que se completan por unidad de tiempo.



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Segmentación

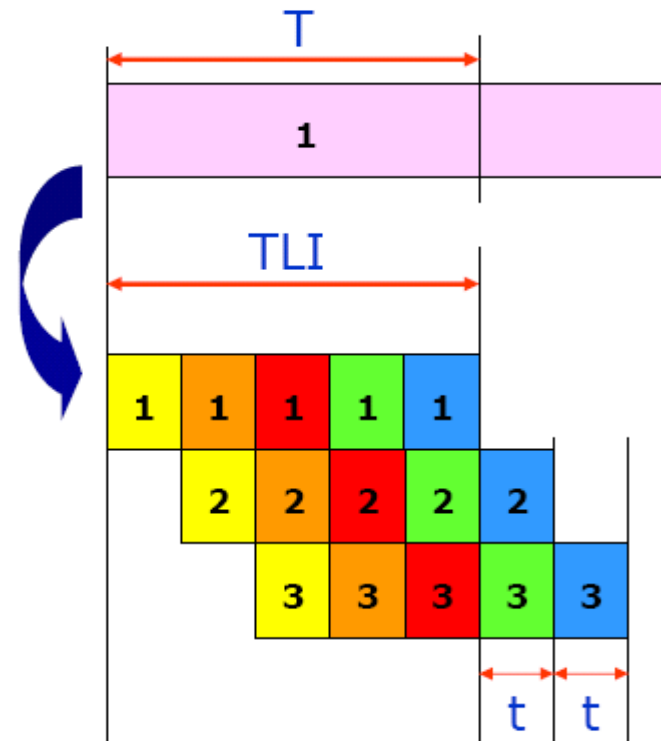
Ganancia. Suponemos que TLI (tiempo de latencia de inicio) = T , tiempo que tarda en ejecutarse una operación en una unidad sin segmentar.

TLI = $k \cdot t$, siendo k el nº de etapas del cauce, y t la duración de cada etapa

$$G_k = \frac{T_1}{T_k} = \frac{k \cdot n \cdot t}{k \cdot t + (n-1) \cdot t} =$$
$$= \frac{k \cdot n}{k + n - 1}$$

$$\lim_{n \rightarrow \infty} G_k = k$$

Normalmente, ¿ $TLI > T$ ó $TLI < T$?



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Ganancia real

$$G_k = \frac{T_{\text{sin_segmentar}}}{T_{\text{segmentado}}} = \frac{n \times T}{\text{TLI} + (n - 1) \times t}$$

$$G_{\text{max}} = \lim_{n \rightarrow \infty} \frac{n \times T}{(k \times t) + (n - 1) \times t} = \frac{T}{t} < k$$

↑
 $\text{TLI} = kt$

↑
 $T < \text{TLI} = kt$

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Tipos de riesgos (detección del cauce)
 - **Riesgos de datos.** Se producen por dependencias entre operandos y resultados de instrucciones distintas.
 - **Riesgos de control.** Se originan a partir de instrucciones de salto condicional que, según su resultado, determinan la secuencia de instrucciones que hay que procesar tras ellas.
 - **Riesgos estructurales o colisiones.** Se producen cuando instrucciones diferentes necesitan el mismo recurso al mismo tiempo

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Riesgos de datos



RAW (Read After Write)

```
R2 := R1 + R2  
R1 := R2 + R3
```

The diagram shows a red arrow pointing from the $R2$ in the first line to the $R2$ in the second line, indicating a data dependency.



WAR (Write After Read)

```
R2 := R1 + R2  
R1 := R2 + R3
```

The diagram shows a red arrow pointing from the $R1$ in the first line to the $R1$ in the second line, indicating a data dependency.



WAW (Write After Write)

```
R2 := R1 + R2  
R2 := R4 + R3
```

The diagram shows a red arrow pointing from the $R2$ in the first line to the $R2$ in the second line, indicating a data dependency.

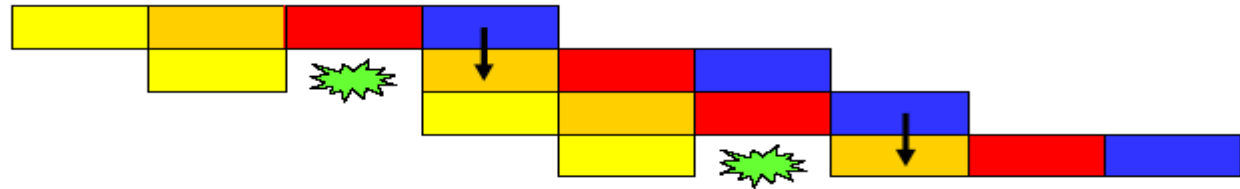
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

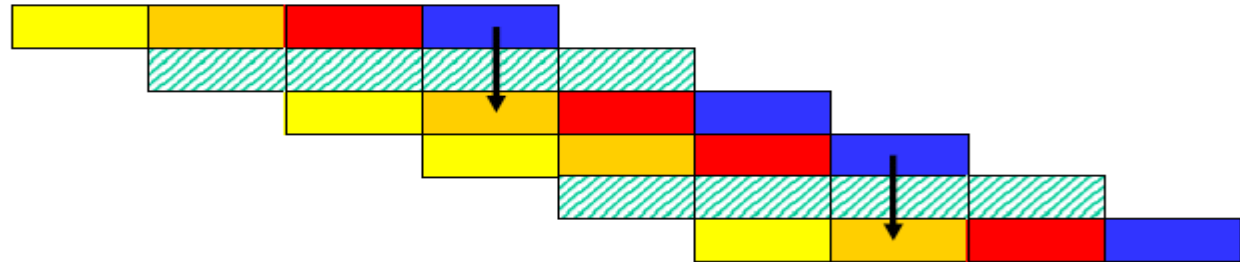
ILP

- Soluciones a los riesgos de datos
 - Reorganización de código (intercambio de instrucciones e inserción de NOP)

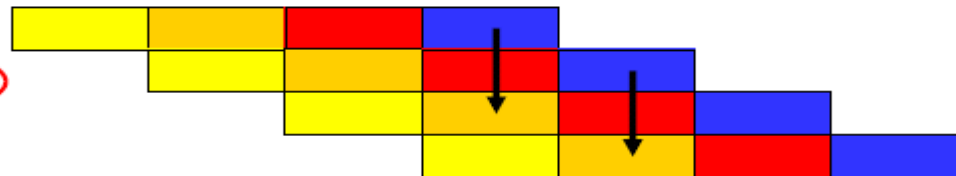
$R3 = R3 + R1$
 $R5 = R5 - R3$
 $R4 = R4 + R1$
 $R6 = R6 - R4$



$R3 = R3 + R1$
nop
 $R5 = R5 - R3$
 $R4 = R4 + R1$
nop
 $R6 = R6 - R4$



$R3 = R3 + R1$
 $R4 = R4 + R1$
 $R5 = R5 - R3$
 $R6 = R6 - R4$

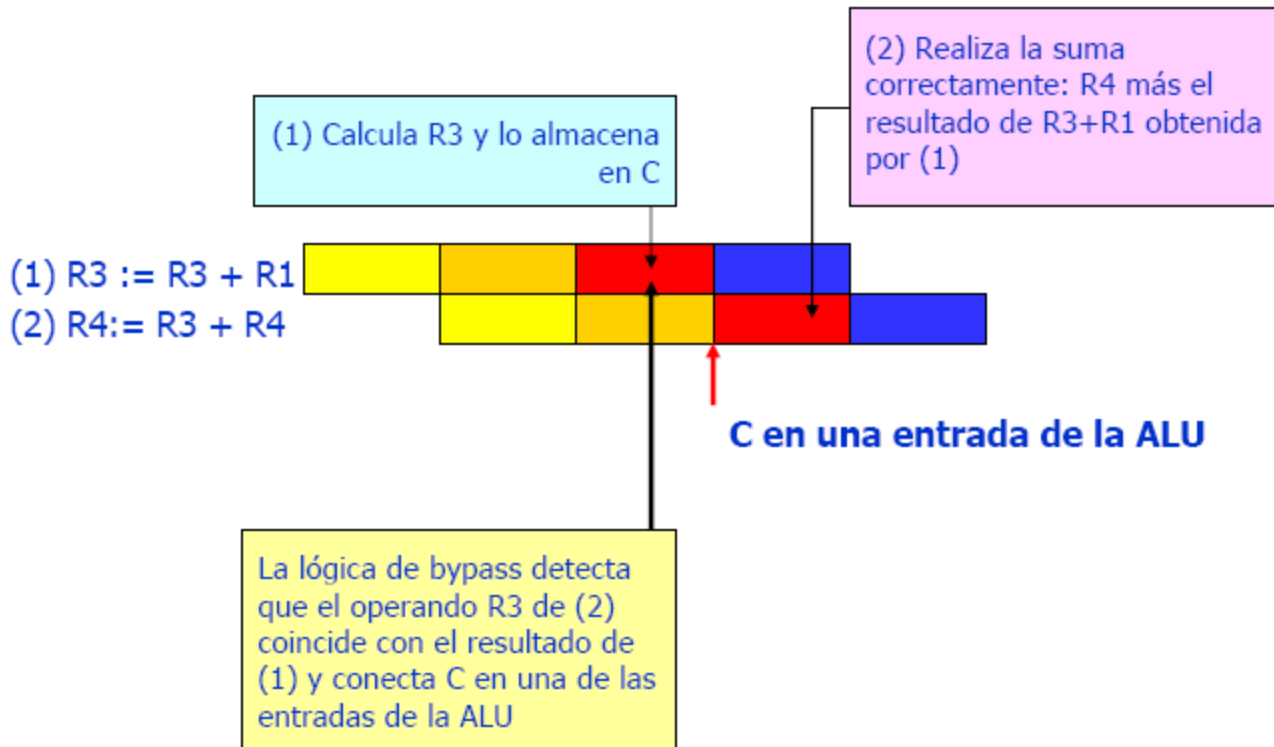


Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Soluciones a los riesgos de datos
 - Forwardings

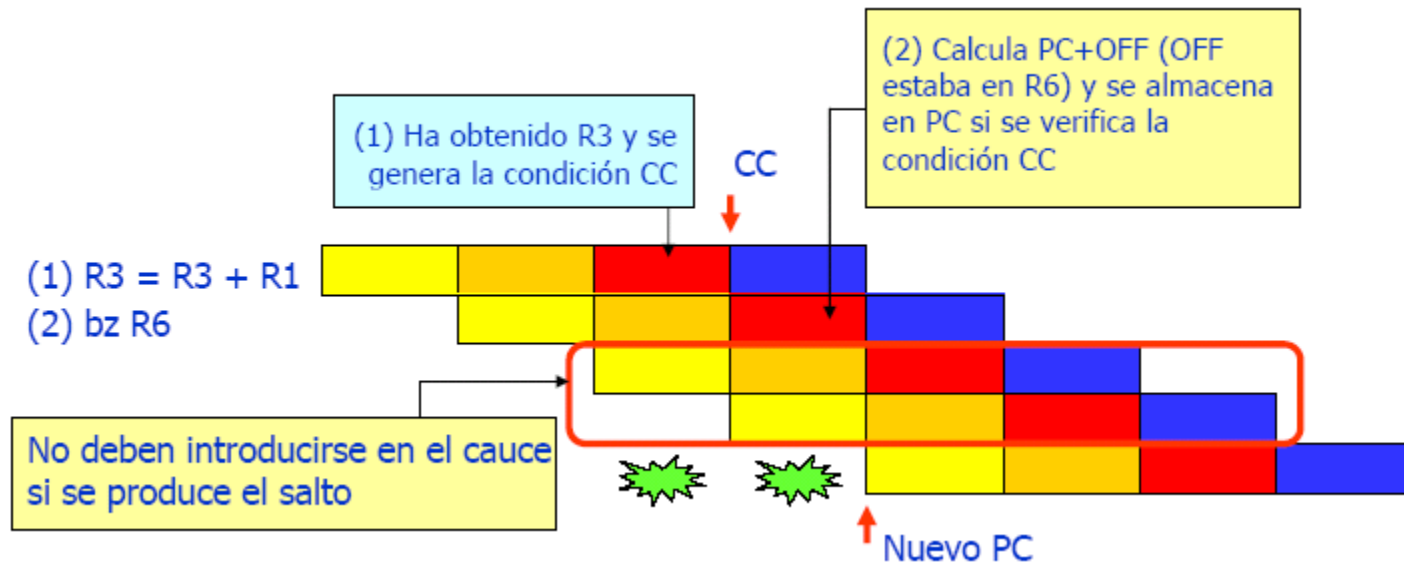


Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Soluciones a los riesgos de control
 - Abortar operaciones

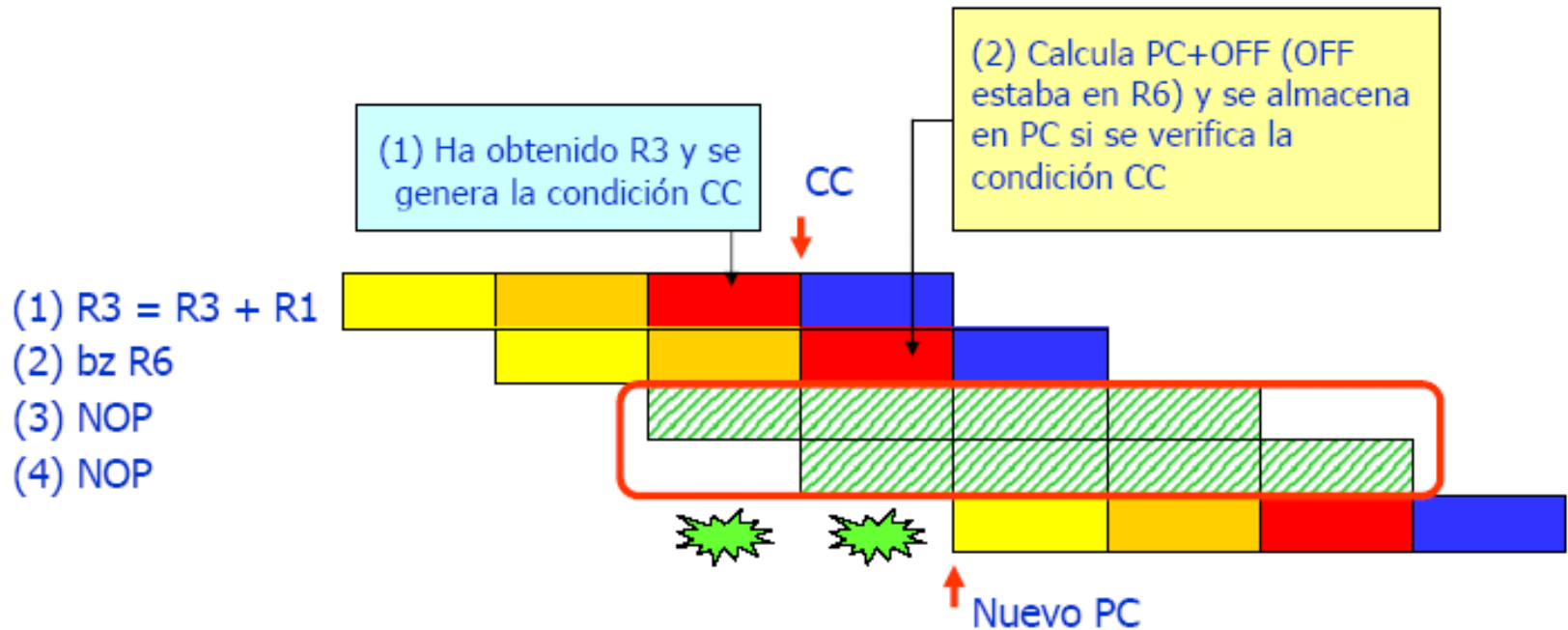


Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Soluciones a los riesgos de control
 - Bloqueos o uso de NOP

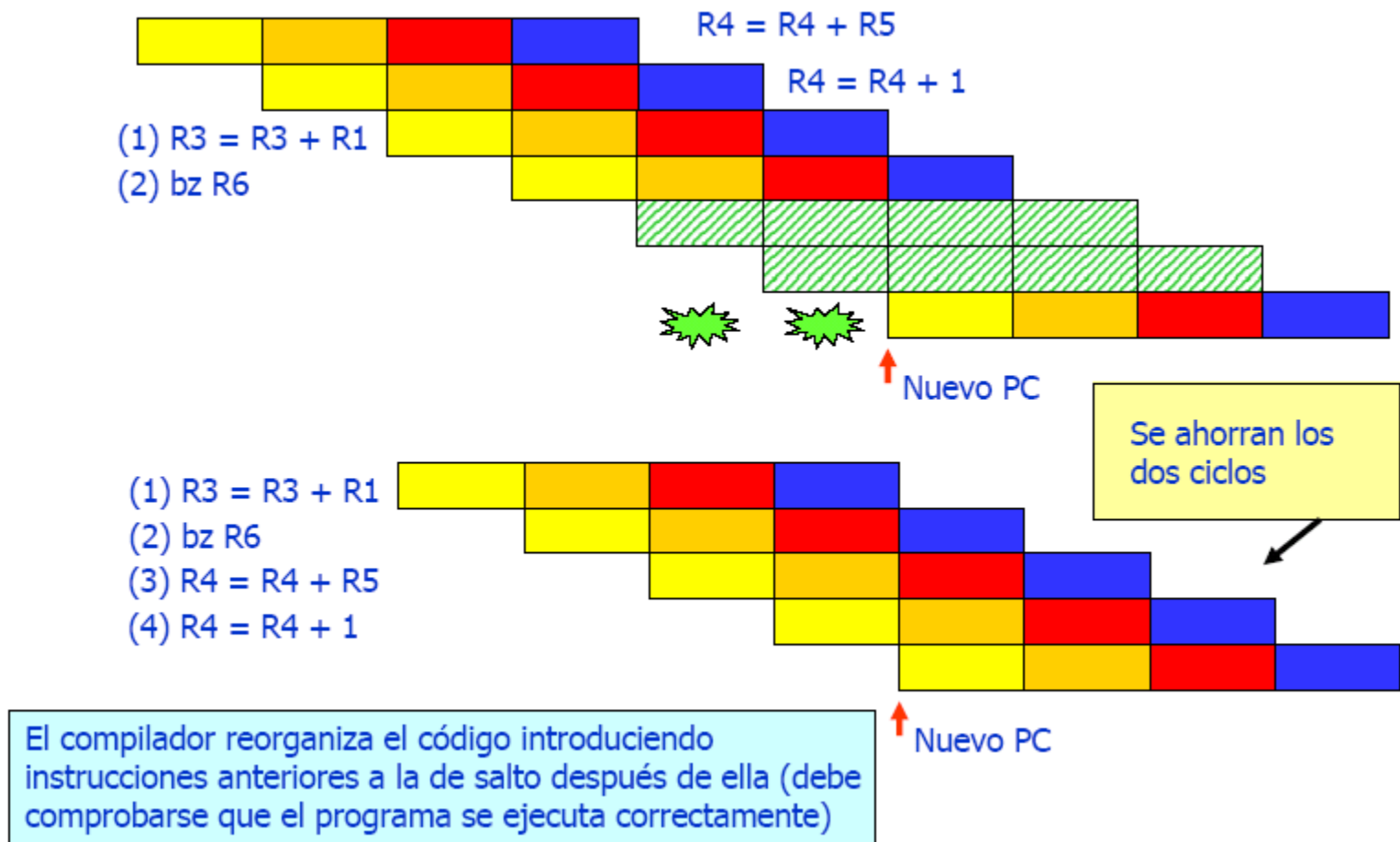


Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Soluciones a los riesgos de control
 - Delayed branch



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

- Tiempo de ejecución de un programa
 - Tiempo de CPU (usuario y sistema)
 - Tiempo de E/S (comunicaciones, acceso a memoria, visualización, etc.)

$$\text{Tiempo de CPU (T}_{\text{CPU}}\text{)} = \text{Ciclos_del_Programa} \times T_{\text{CICLO}} = \frac{\text{Ciclos_del_Programa}}{\text{Frecuencia_de_Reloj}}$$

$$\text{Ciclos por Instrucción (CPI)} = \frac{\text{Ciclos_del_Programa}}{\text{Número_de_Instrucciones (NI)}}$$

$$T_{\text{CPU}} = \text{NI} \times \text{CPI} \times T_{\text{CICLO}}$$

$$T_{\text{CICLO}} = 1/F$$

F=frecuencia

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

- Tiempo para arquitecturas capaces de emitir a ejecución varias instrucciones por unidad de tiempo

$$T_{\text{CPU}} = NI \times (CPE / IPE) \times T_{\text{CICLO}}$$

CPI

CPE = ciclos entre inicio de emisión de instrucciones.

IPE = instrucciones que pueden emitirse (empezar la ejecución) cada vez que se produce ésta.

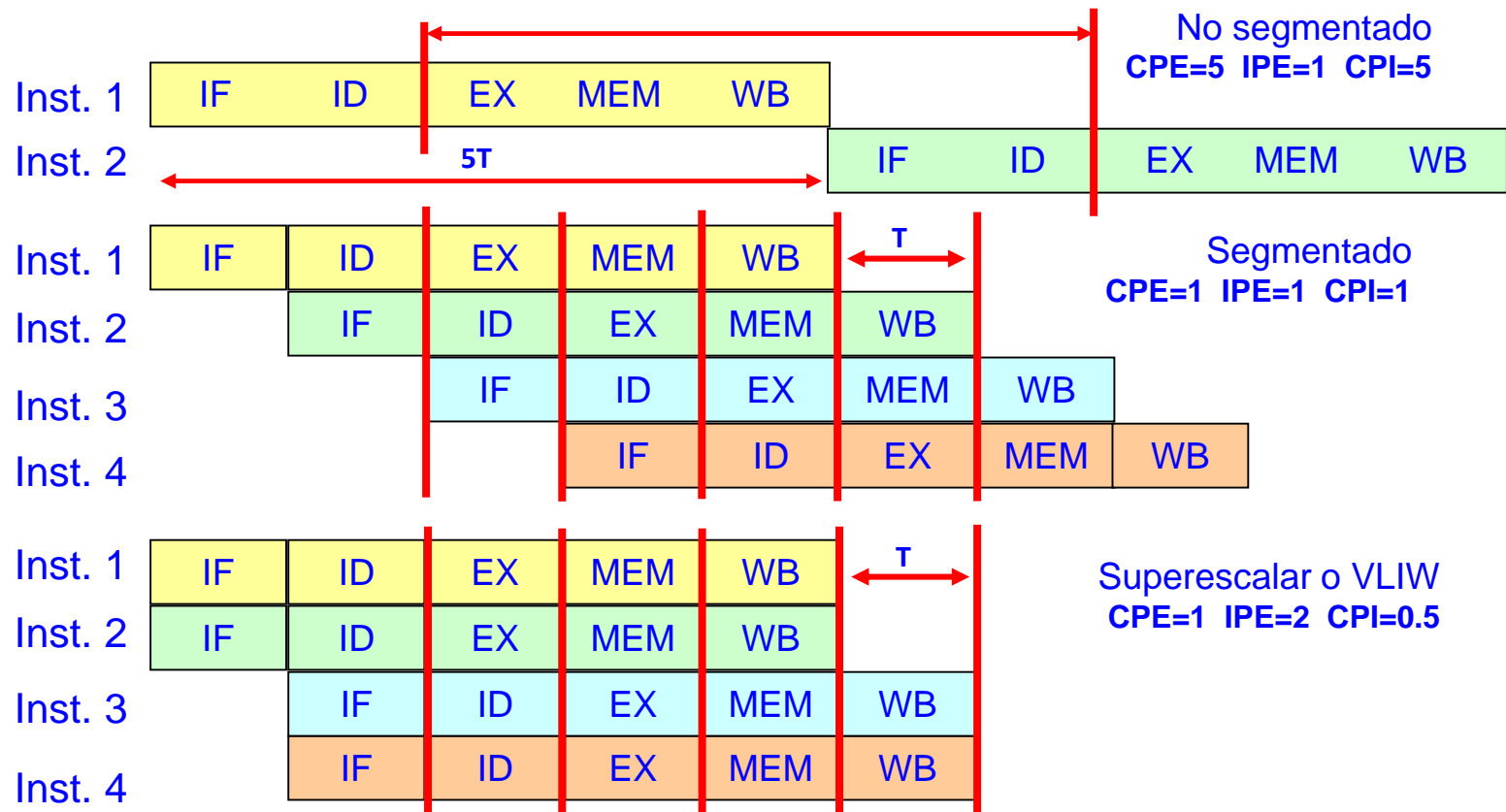
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

Ejemplo:

$$\text{CPI} = \text{CPE} / \text{IPE}$$



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

- Procesadores que codifican varias operaciones en una instrucción (VLIW)

$$T_{\text{CPU}} = \underbrace{(\text{Noper} / \text{Op_instr})}_{\text{NI}} \times \text{CPI} \times T_{\text{CICLO}}$$

Noper = número de operaciones que realiza el programa.

Op_instr = número de operaciones que puede codificar una instrucción.

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

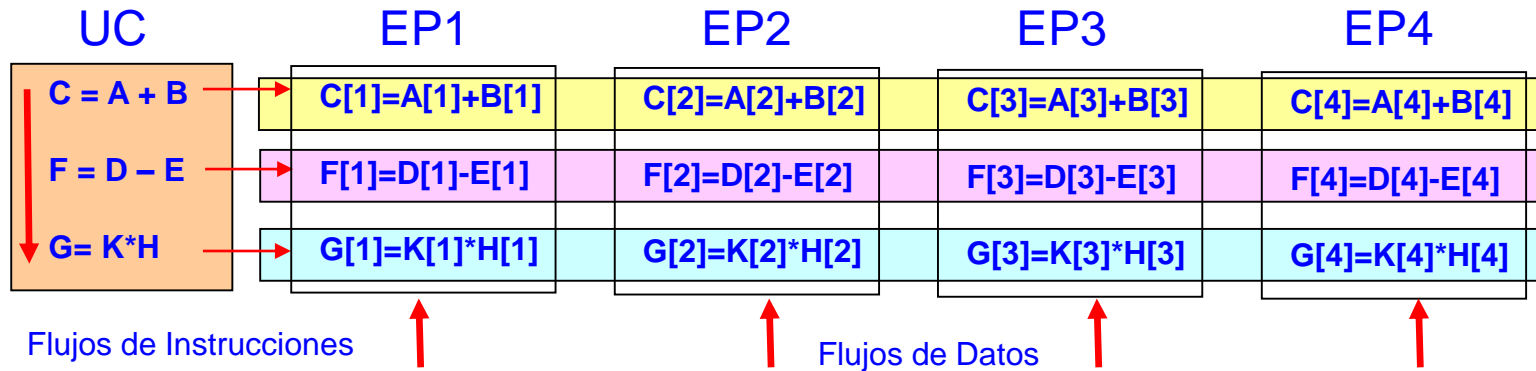
Ejemplo:

$$NI = Noper / Op_instr$$

Ejemplo paralelismo datos

Procesador Matricial

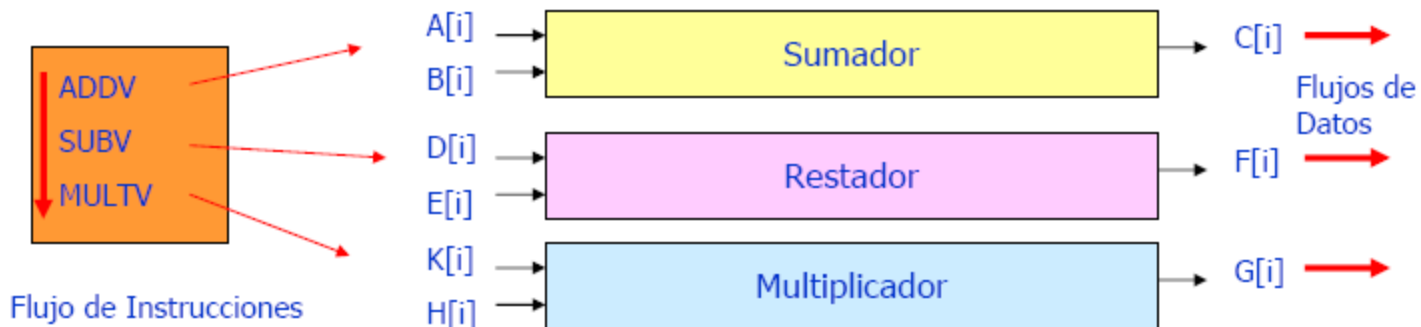
$$Noper=12 \quad Op_instr=4 \quad NI=3$$



Ejemplo paralelismo instrucciones

Procesador Vectorial

$$Noper=12 \quad Op_instr=4 \quad NI=3$$



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

- Medidas de rendimiento:

- Ganancia

$$G_P = \frac{T_1}{T_P}; \quad G_P \leq P$$

- Eficiencia

$$E_P = \frac{G_p}{P}; \quad E_P \leq 1$$

- Productividad

Unidad 1. Introducción al paralelismo

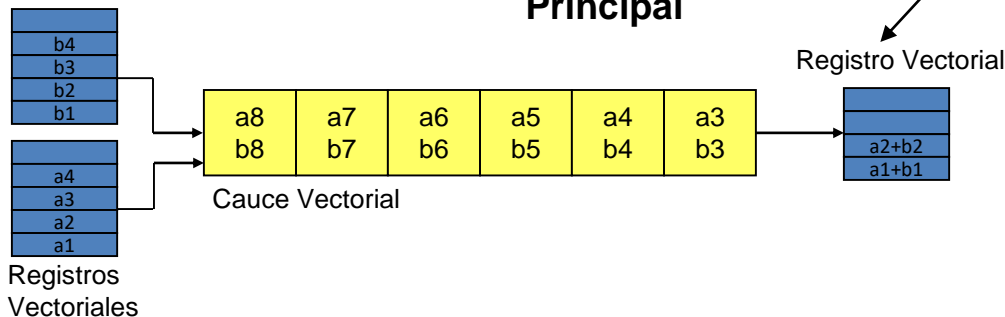
1.3 Arquitecturas vectoriales

Vectoriales

- Arquitecturas vectoriales: ILP y paralelismo de datos

El procesamiento de instrucciones está segmentado y se utilizan múltiples unidades funcionales.

Paralelismo de datos: cada instrucción vectorial codifica una operación sobre todos los componentes del vector.



Unidades funcionales segmentadas

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Arquitectura orientada al procesamiento de vectores (suma de vectores, productos escalares, etc.)
- Repertorio de instrucciones especializado
- Características
 - Cálculo de los componentes del vector de forma independiente (buenos rendimientos)
 - Cada operación vectorial codifica gran cantidad de cálculos (se reduce el número de instrucciones y se evitan riesgos de control)
 - Se optimiza el uso de memoria (entrelazado de memoria y organizaciones S y C)

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

Ejemplo: Sumar dos vectores de 100 elementos.

Pseudo-código escalar

for i:= 1 to 100 do c(i)=b(i)+a(i)

Ensamblador escalar (con bucle de 100 iteraciones)

```
LOADI R5, BASEa  
LOADI R6, BASEb  
LOADI R7, BASEc  
LOADI R1, 0  
INI ADDRI R5, R5, 1  
ADDRI R6, R6, 1  
ADDRI R7, R7, 1  
ADDMR R8, R5, R6  
STORE R7, R8  
INC R1  
COMP R1, 100  
JUMP NOT.EQUAL INI
```

Pseudo-código vectorial

c(1:100:1) = a(1:100:1) + b(1:100:1)

Ensamblador vectorial

```
ADDV c, a, b, 1, 100
```

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

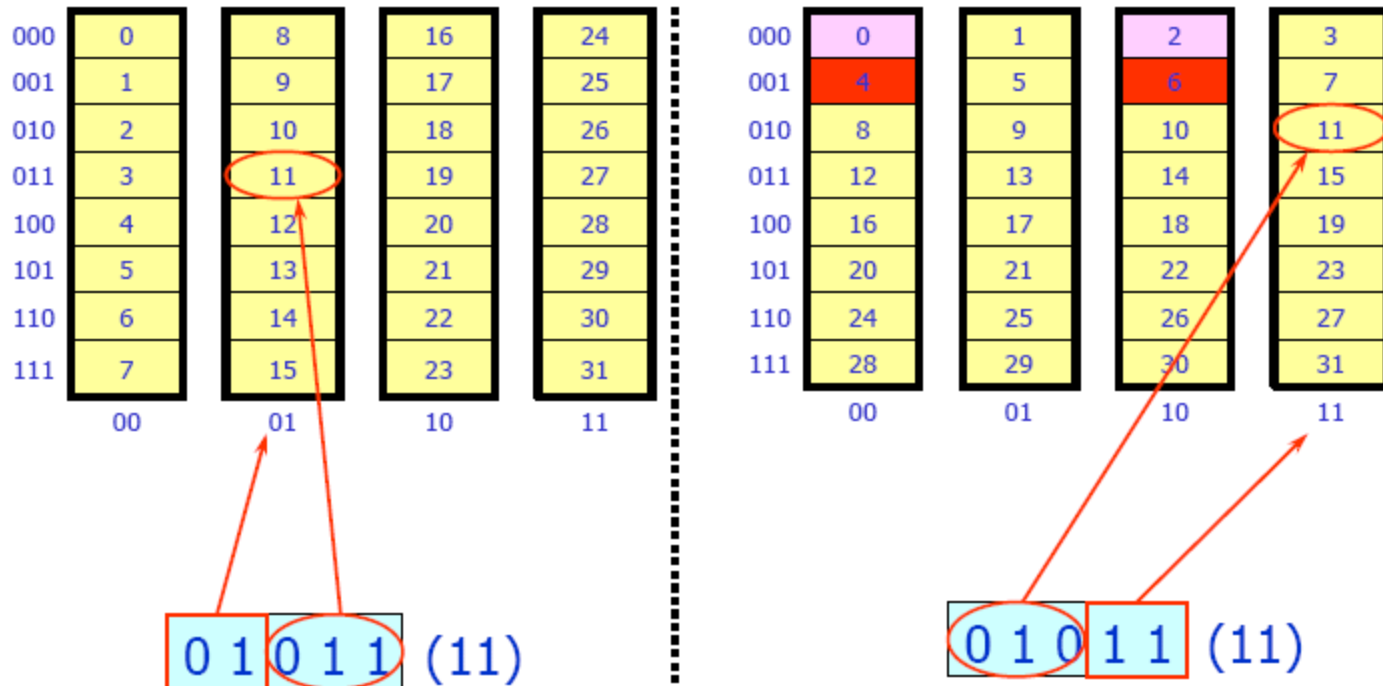
Vector instruction	Operands	Function
ADDV	V1, V2, V3	Add elements of V2 and V3, then put each result in V1.
ADDSV	V1, F0, V2	Add F0 to each element of V2, then put each result in V1.
SUBV	V1, V2, V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS	V1, V2, F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV	V1, F0, V2	Subtract elements of V2 from F0, then put each result in V1.
MULTV	V1, V2, V3	Multiply elements of V2 and V3, then put each result in V1.
MULTSV	V1, F0, V2	Multiply F0 by each element of V2, then put each result in V1.
DIVV	V1, V2, V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS	V1, V2, F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV	V1, F0, V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1, R1	Load vector register V1 from memory starting at address R1.
SV	R1, V1	Store vector register V1 into memory starting at address R1.
LVWS	V1, (R1, R2)	Load V1 from address at R1 with stride in R2, i.e., $R1+i \cdot R2$.
SVWS	(R1, R2), V1	Store V1 from address at R1 with stride in R2, i.e., $R1+i \cdot R2$.
LVI	V1, (R1+V2)	Load V1 with vector whose elements are at $R1+V2(i)$, i.e., V2 is an index.
SVI	(R1+V2), V1	Store V1 with vector whose elements are at $R1+V2(i)$, i.e., V2 is an index.
CVI	V1, R1	Create an index vector by storing the values $0, 1 \cdot R1, 2 \cdot R1, \dots, 63 \cdot R1$ into V1.
S_V	V1, V2	Compare (EQ, NE, GT, LT, GE, LE) the elements in V1 and V2. If condition is true put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S_SV performs the same compare but using a scalar value as one operand.
S_SV	F0, V1	
POP	R1, VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MOVI2S	VLR, R1	Move contents of R1 to the vector-length register.
MOV2I	R1, VLR	Move the contents of the vector-length register to R1.
MOV2S	VM, F0	Move contents of F0 to the vector-mask register.
MOV2F	F0, VM	Move contents of vector-mask register to F0.

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Entrelazado de memoria



$2^5=32$ direcciones de memoria

$2^2=4$ módulos de $2^3=8$ posiciones

Entrelazado Superior

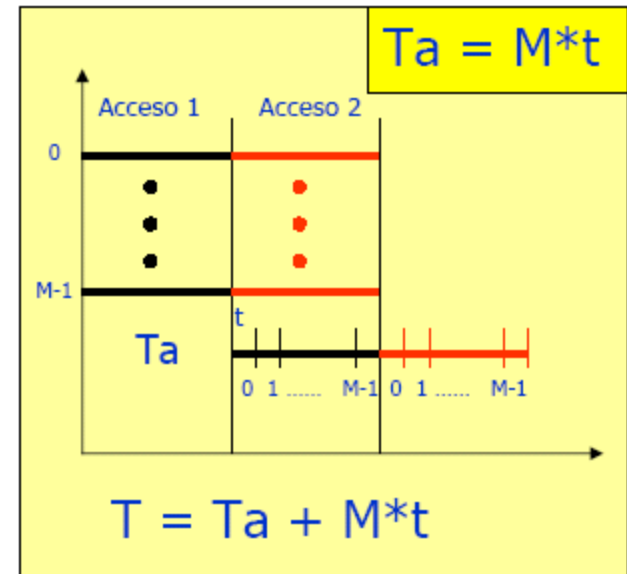
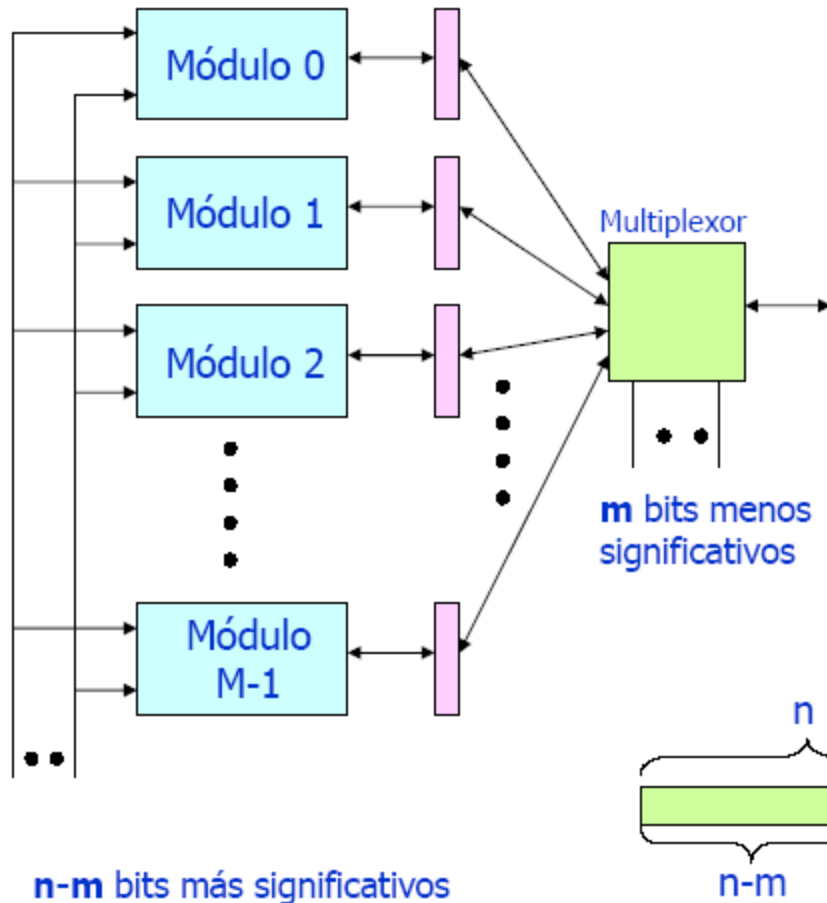
Entrelazado Inferior

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Acceso a memoria simultáneo o tipo S



Con Entrelazado Inferior

$N = 2^n$ direcciones

$M = 2^m$ módulos

$2^{(n-m)}$ direcciones/módulo

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

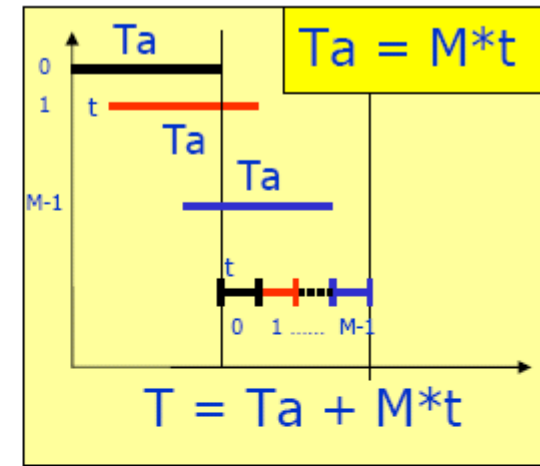
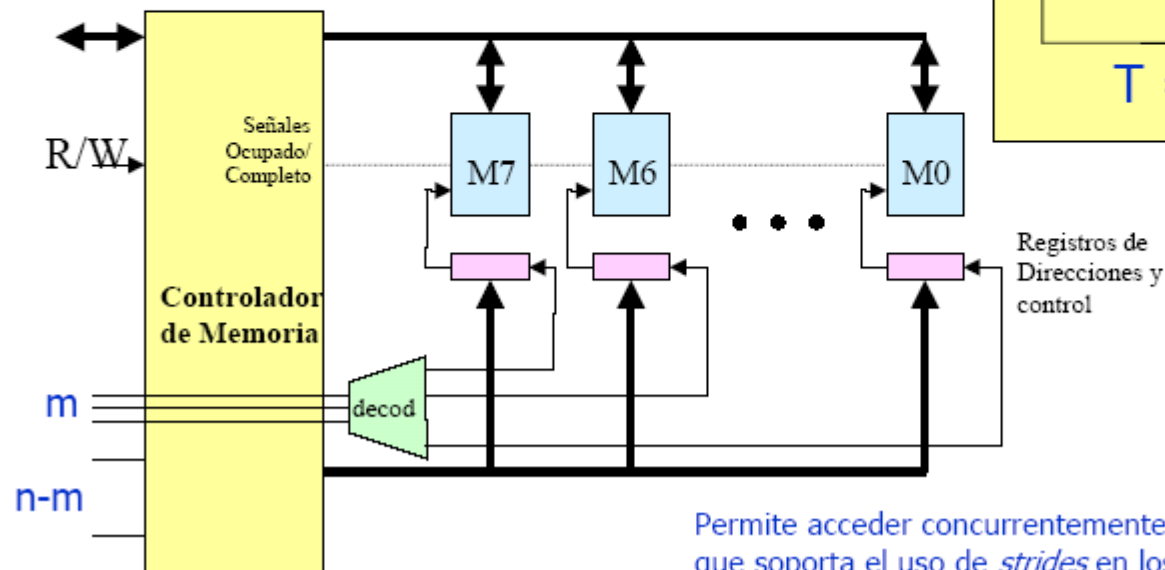
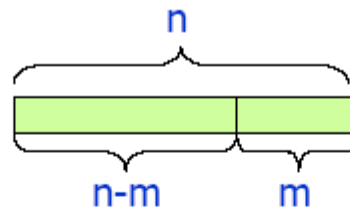
- Acceso a memoria concurrente o tipo C

Con Entrelazado Inferior

$N=2^n$ direcciones

$M=2^m$ módulos

$2^{(n-m)}$ direcciones/módulo



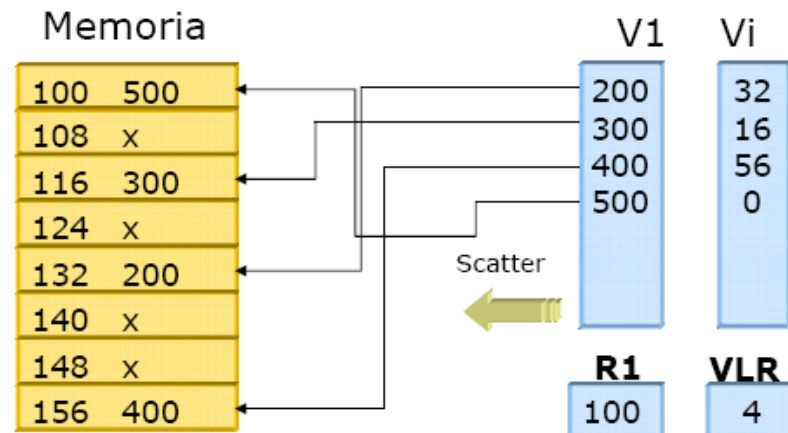
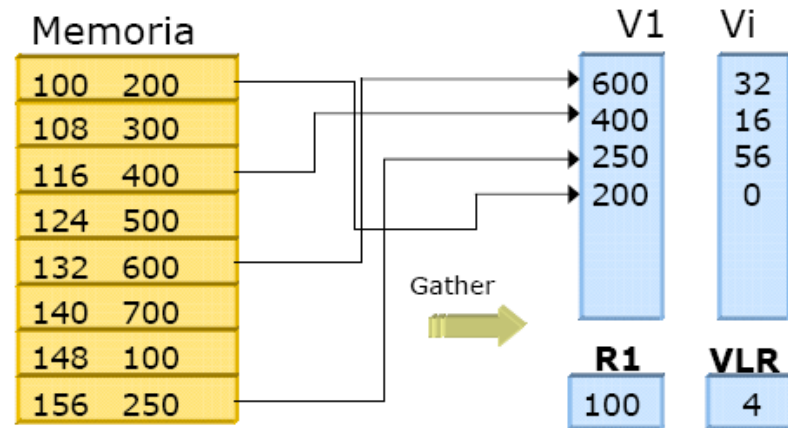
Permite acceder concurrentemente a M direcciones, con lo que soporta el uso de *strides* en los accesos a memoria

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Operaciones gather-scatter

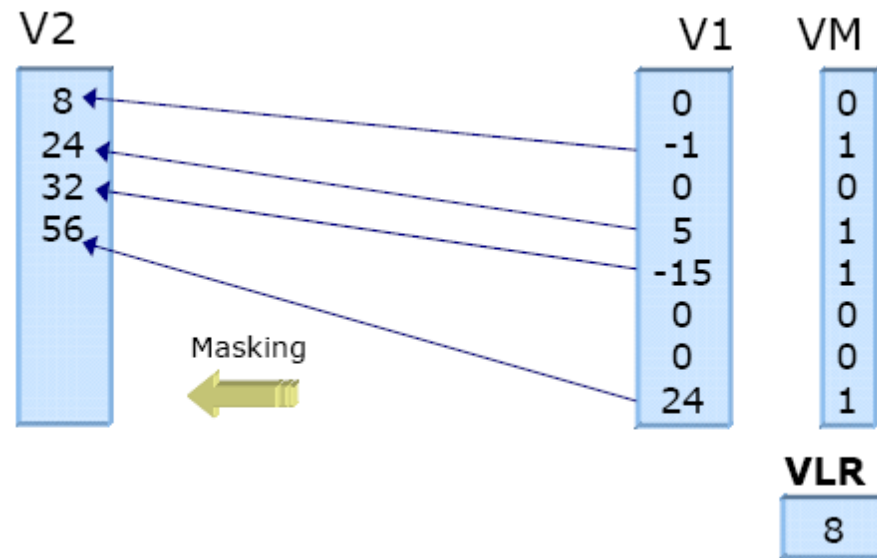


Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Enmascaramiento (gestión de matrices dispersas)

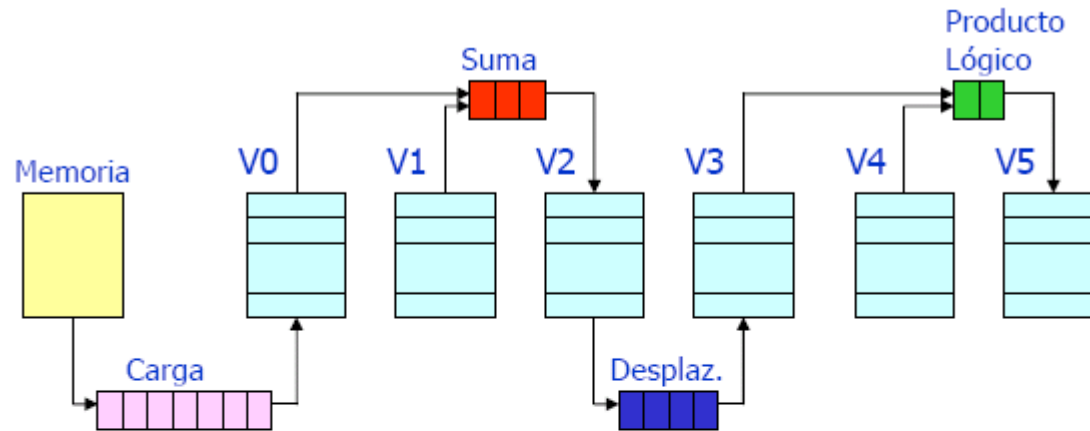


Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Rendimiento: encadenamiento de cauce



V0 = Load(Memoria)	(TLI=7)
V2 = V0 + V1	(TLI=3)
V3 = V1 < A3	(TLI=4)
V5 = V3 ^ V4	(TLI=2)

Unidad 1. Introducción al paralelismo

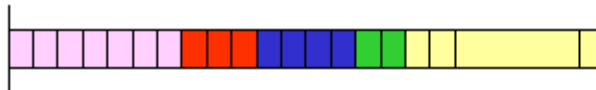
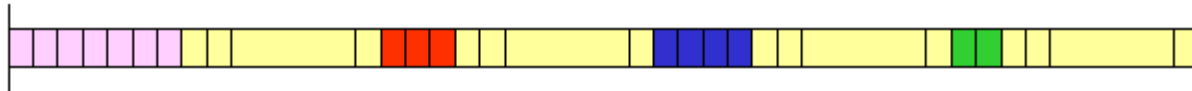
1.3 Arquitecturas vectoriales

Vectoriales

- Rendimiento: encadenamiento de cauce

Si se espera que termine una operación vectorial para que empiece otra

$$TCV = TLI(carga) + TLI(suma) + TLI(desplaz.) + TLI(Prod.Log) + 4 * K$$



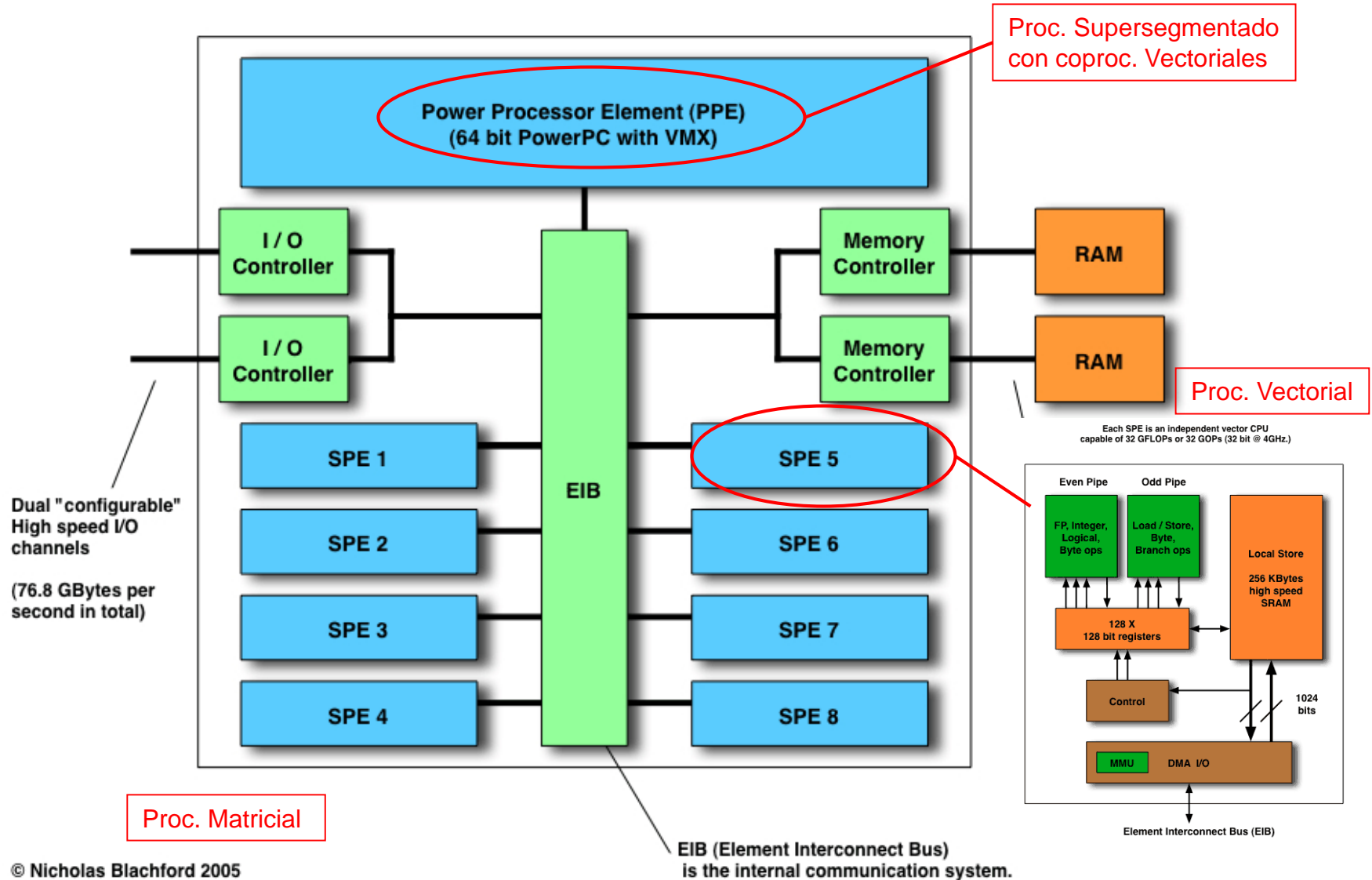
$$TCV = TLI(carga) + TLI(suma) + TLI(desplaz.) + TLI(Prod.Log) + K$$

Si se encadenan los cauces de las distintas operaciones

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales





Ingeniería de los Computadores

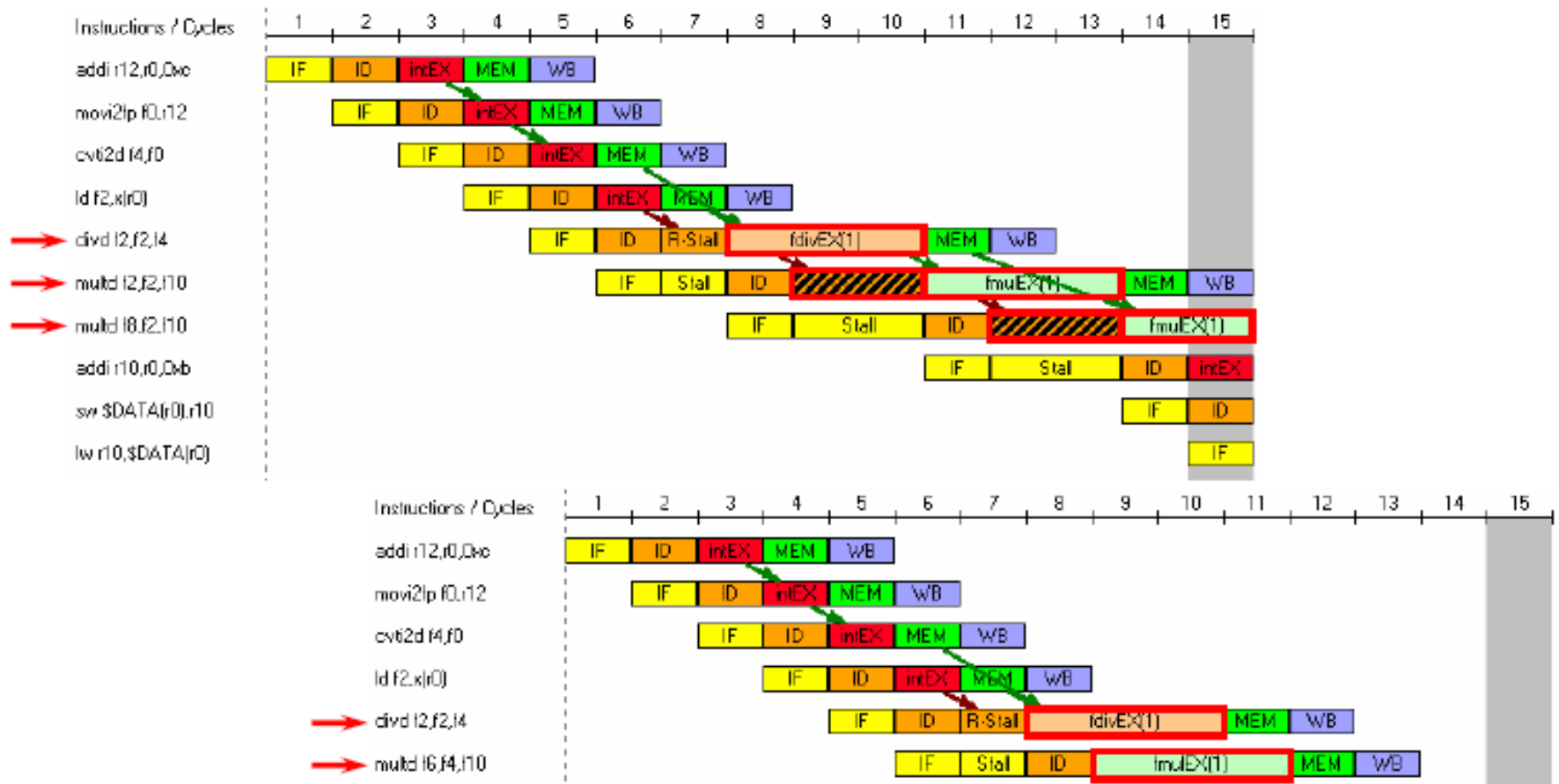
Unidad 2. Superescalares

Unidad 2. Superescalares

2.1. Introducción y motivación

Motivación

- Dependencias estructurales provocan pérdidas de ciclos
 - Ejemplo de una unidad FP vs. Varias unidades FP

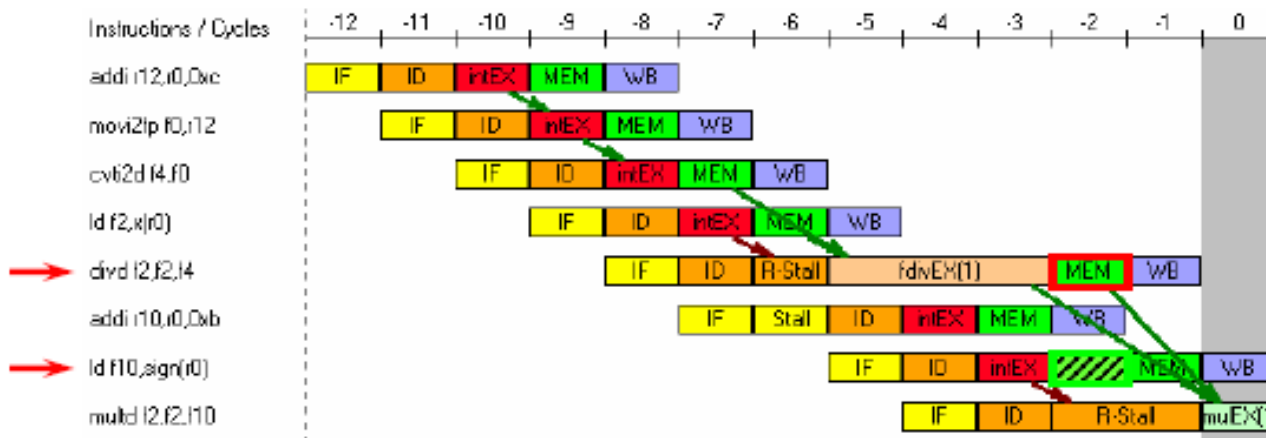


Unidad 2. Superescalares

2.1. Introducción y motivación

Motivación

- Varias unidades funcionales permiten la **ejecución fuera de orden**
 - Validar riesgos WAR y WAW



- Se obtienen mejores prestaciones si se pueden procesar varias instrucciones en la misma etapa → **procesamiento superescalar**

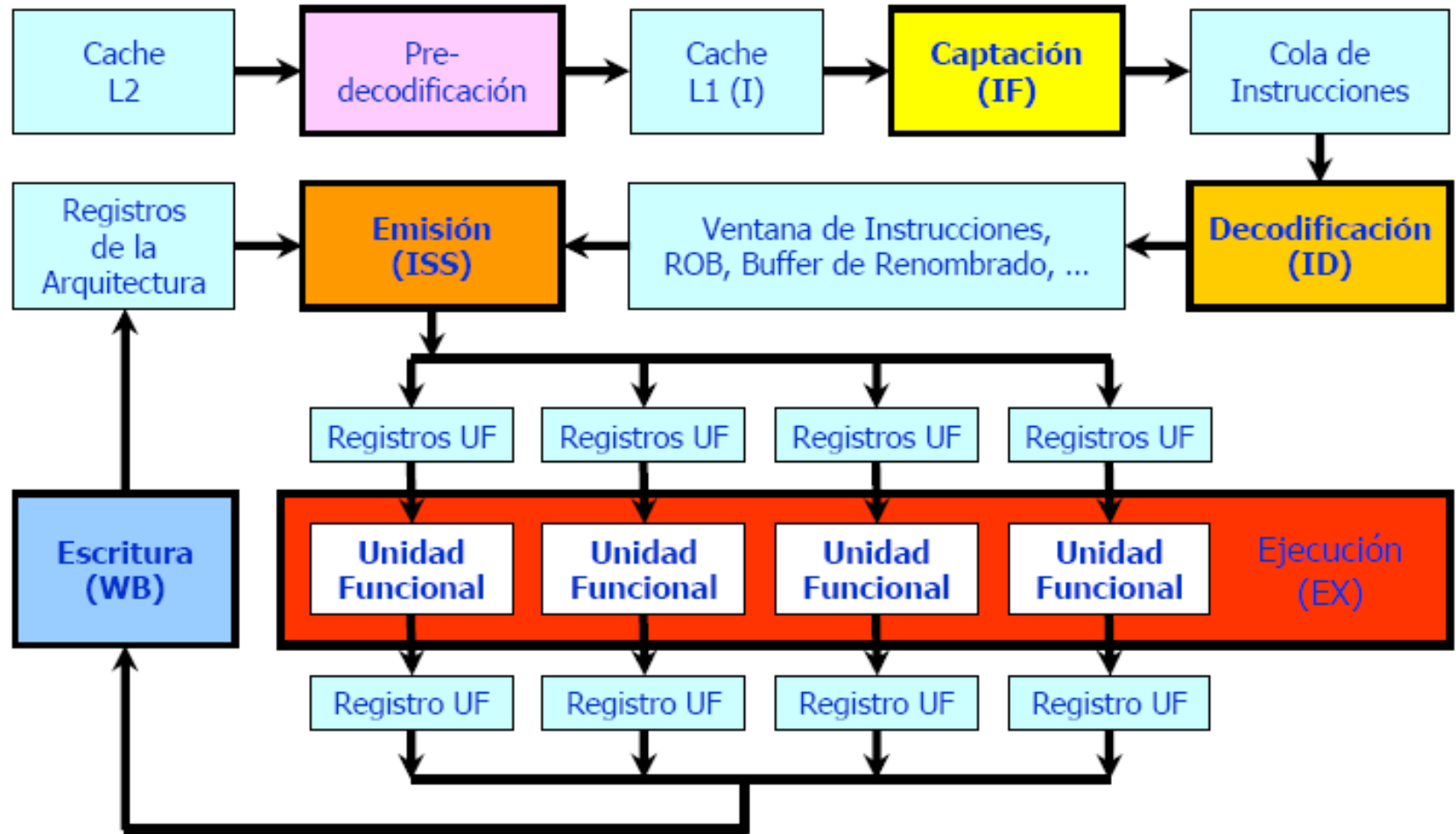
- Etapas
 - Captación de instrucciones (IF)
 - Decodificación de instrucciones (ID)
 - Emisión de instrucciones (ISS)
 - Ejecución de instrucciones (EX) – Instrucción finalizada o “finish”
 - Escritura (WB) – Instrucción completada o “complete”
- Características del procesamiento superescalar
 - Diferentes tipos de órdenes: orden de captación, orden de emisión, orden de finalización
 - Capacidad para identificar ILP existente y organizar el uso de las distintas etapas para optimizar recursos

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce



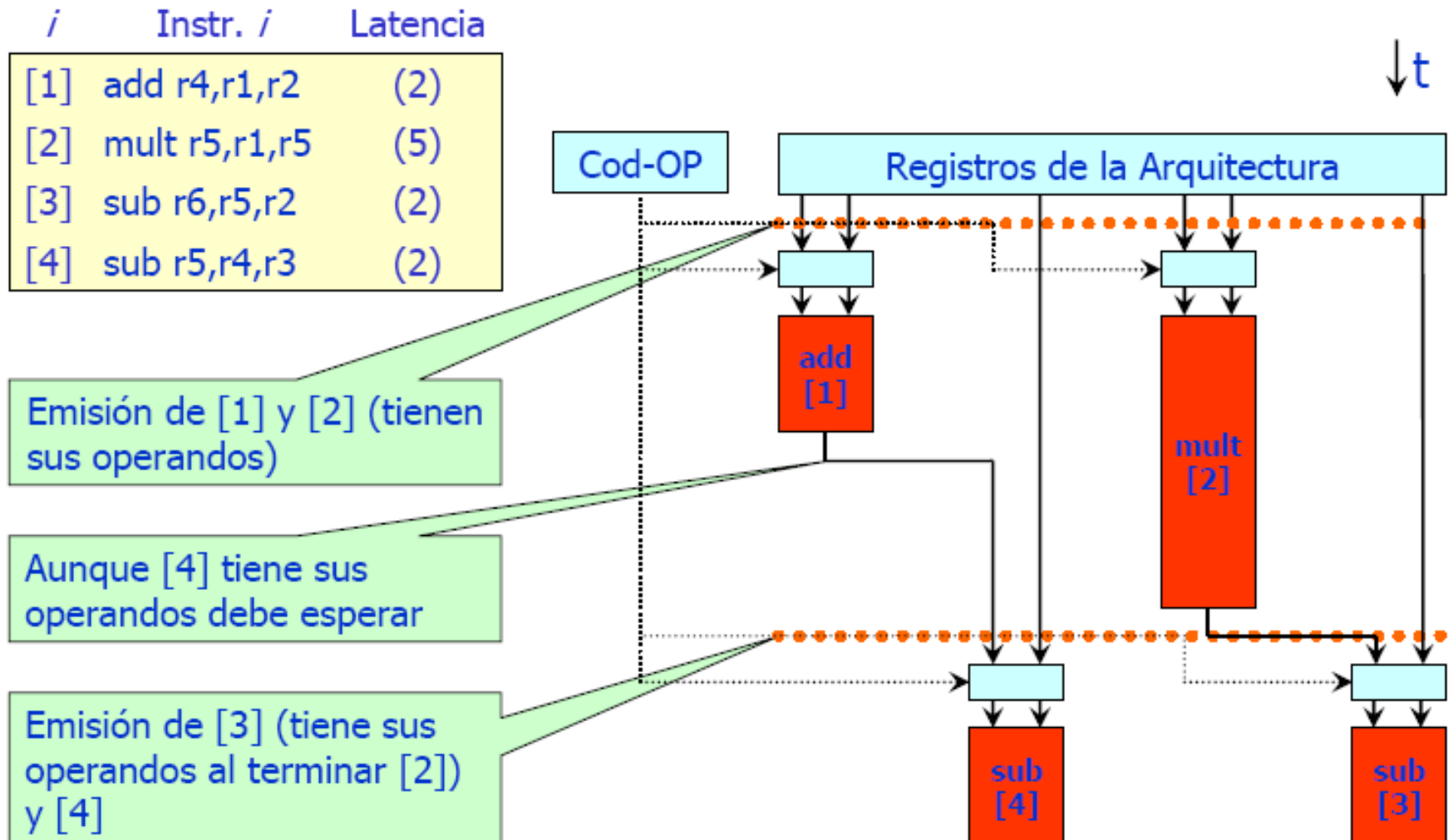
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Emisión ordenada



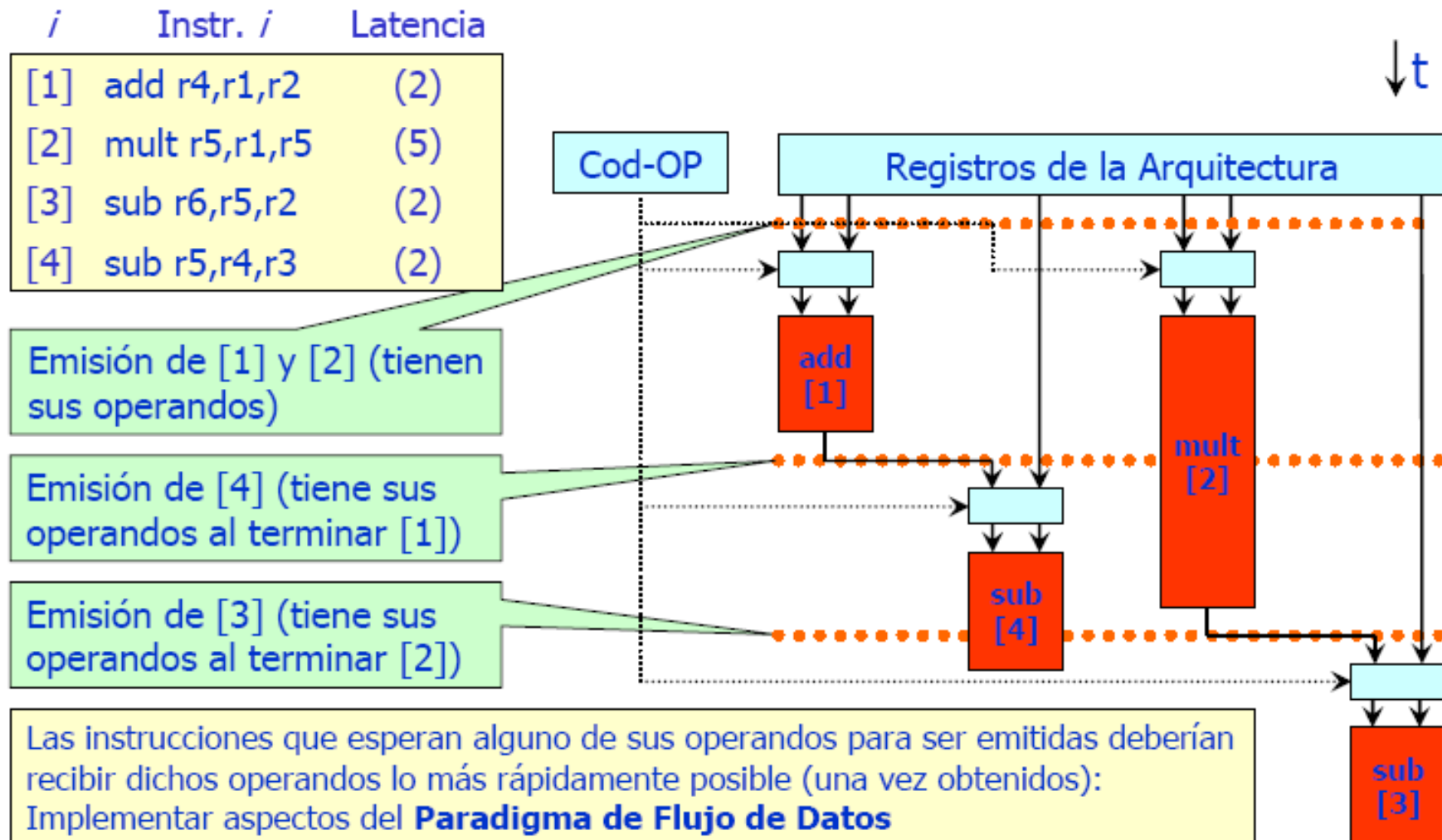
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Emisión desordenada



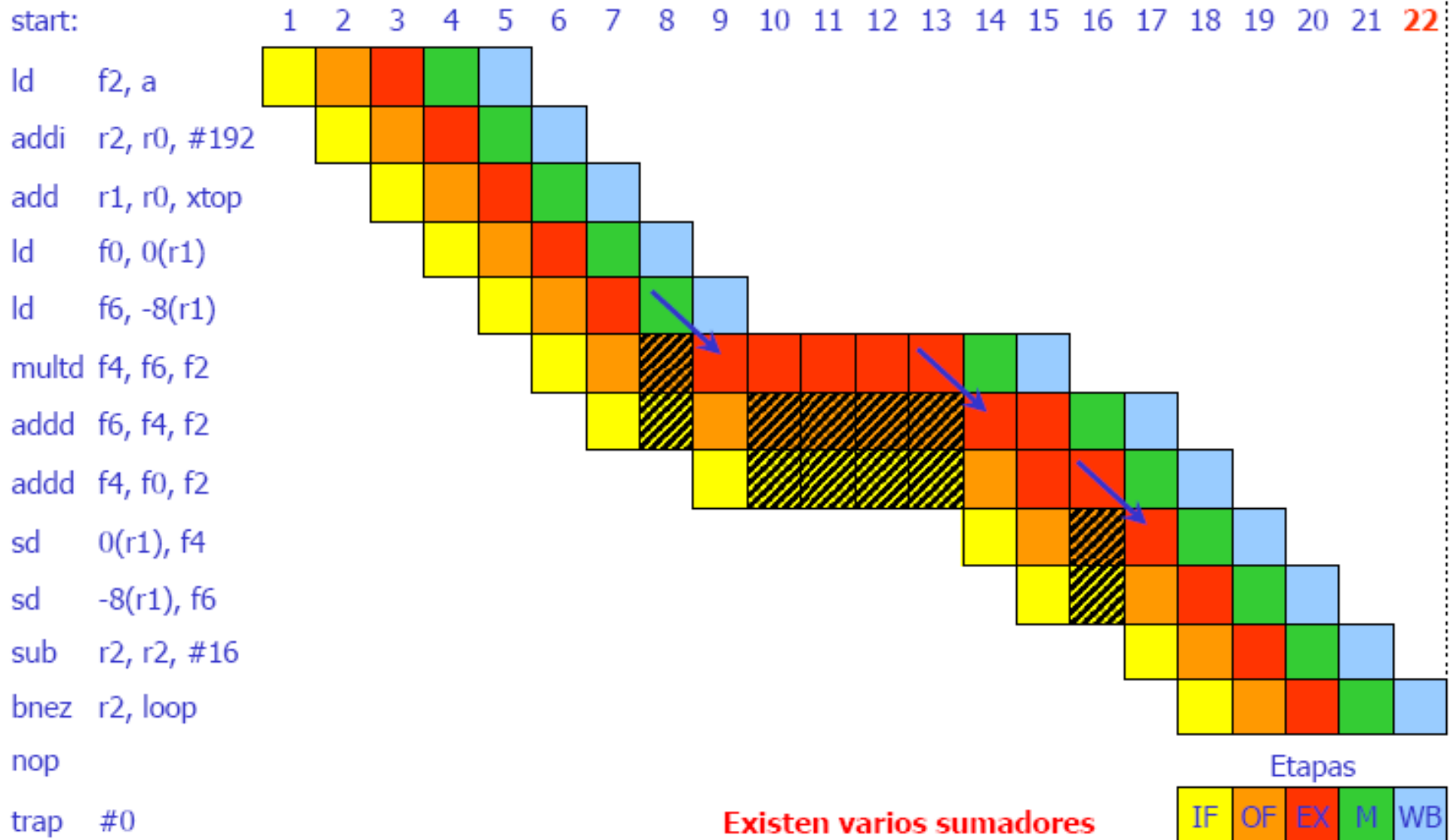
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Procesamiento segmentado



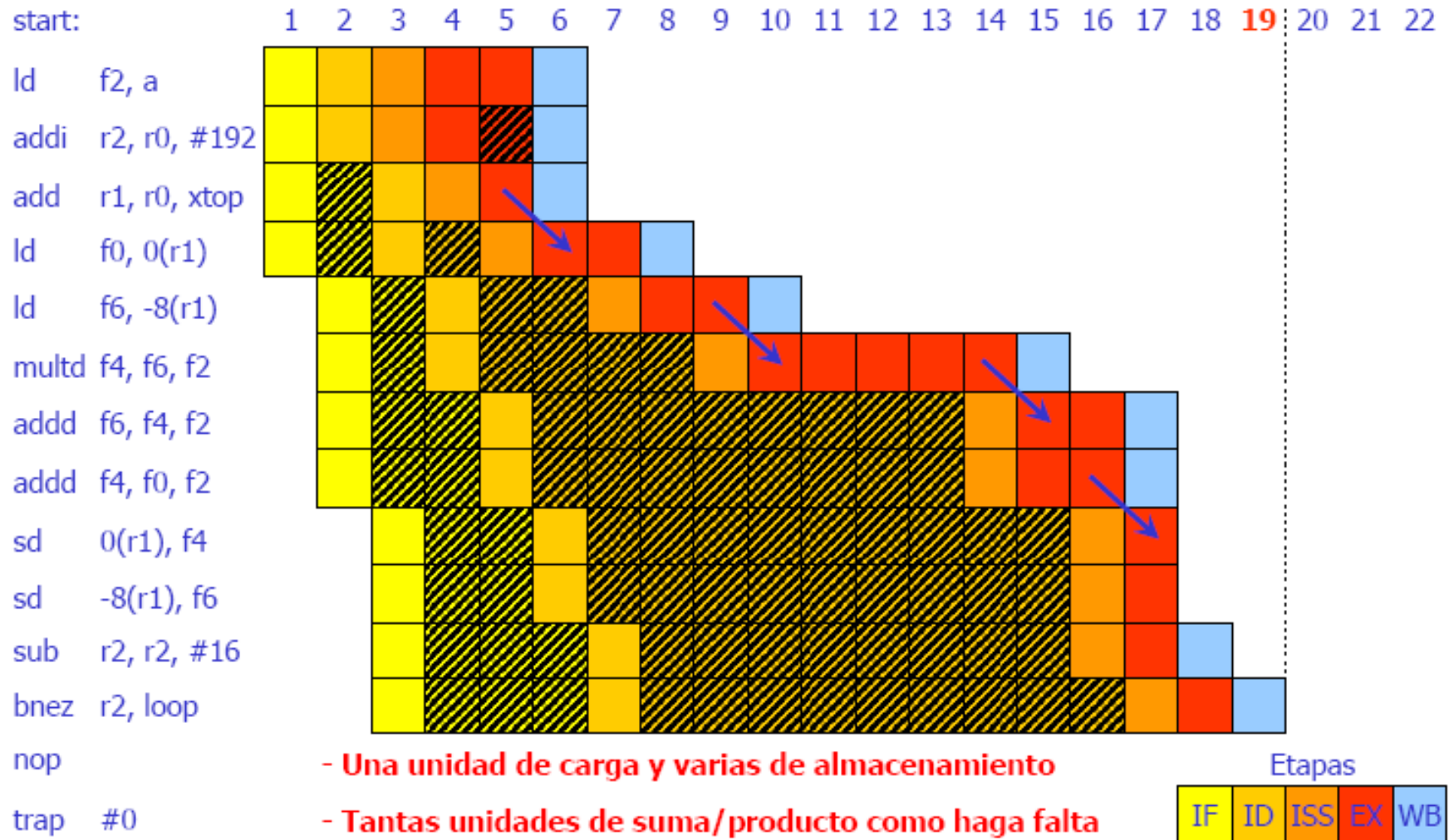
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión ordenada/finalización ordenada



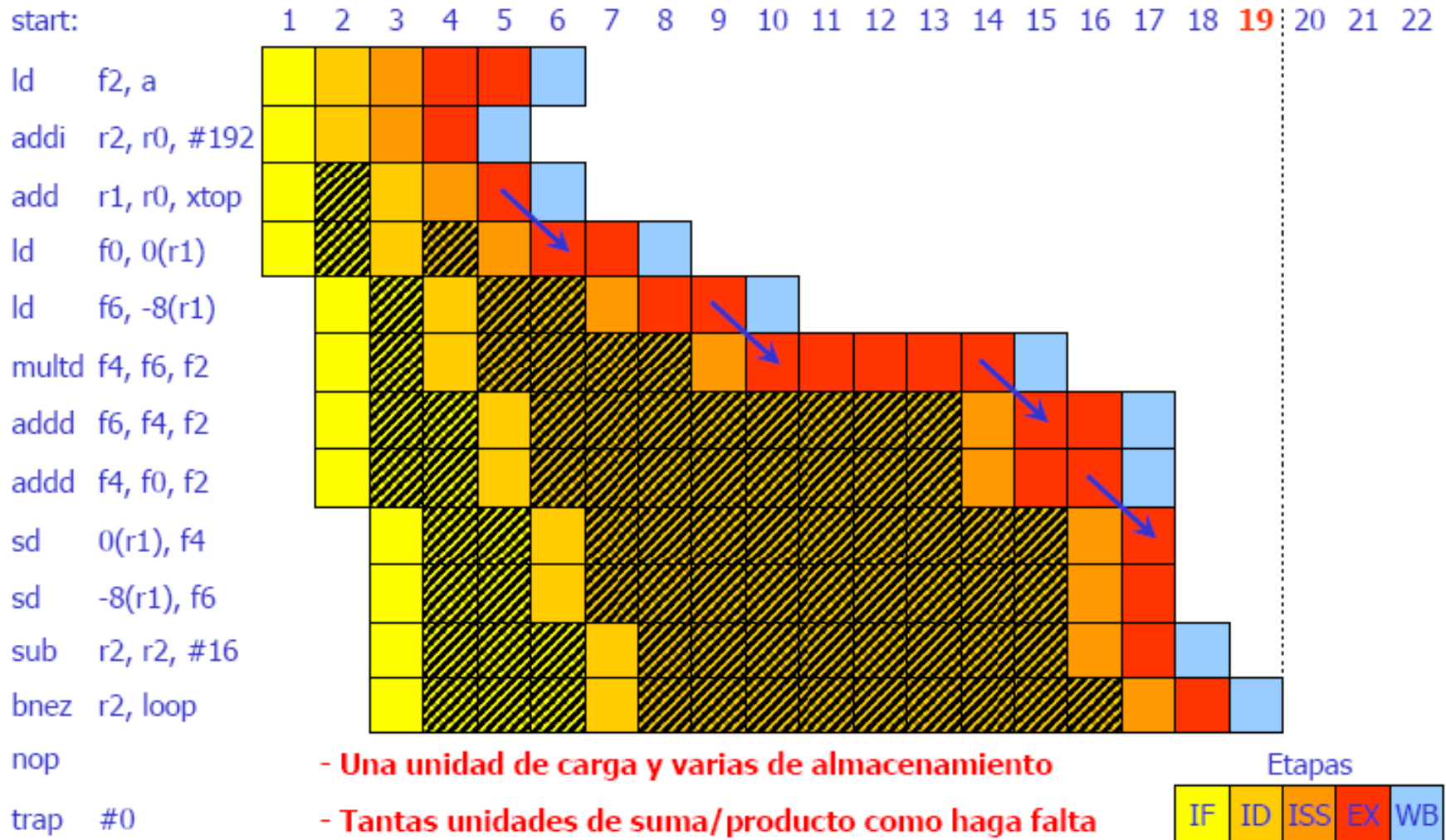
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión ordenada/finalización desordenada



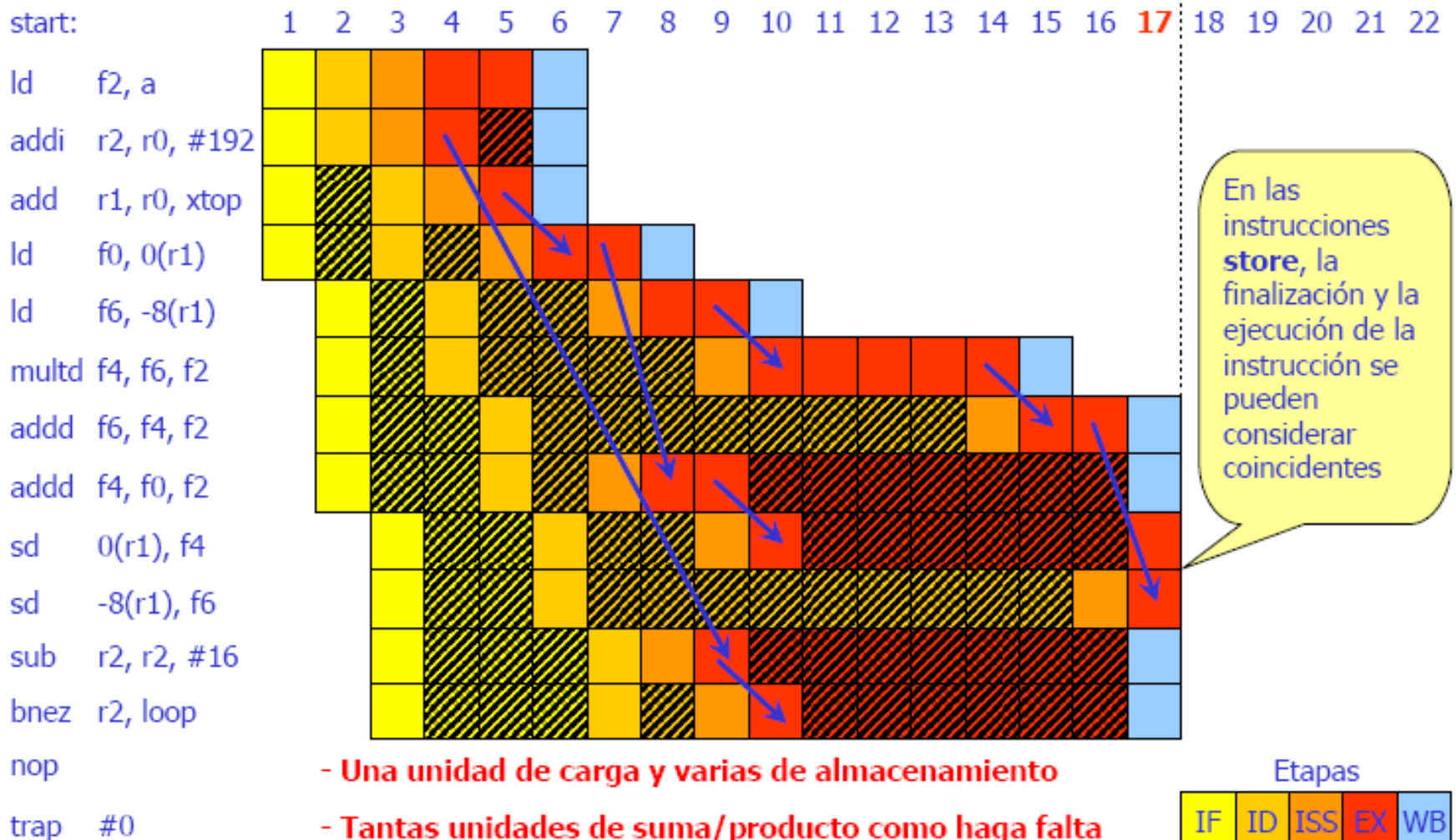
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión desordenada/finalización ordenada



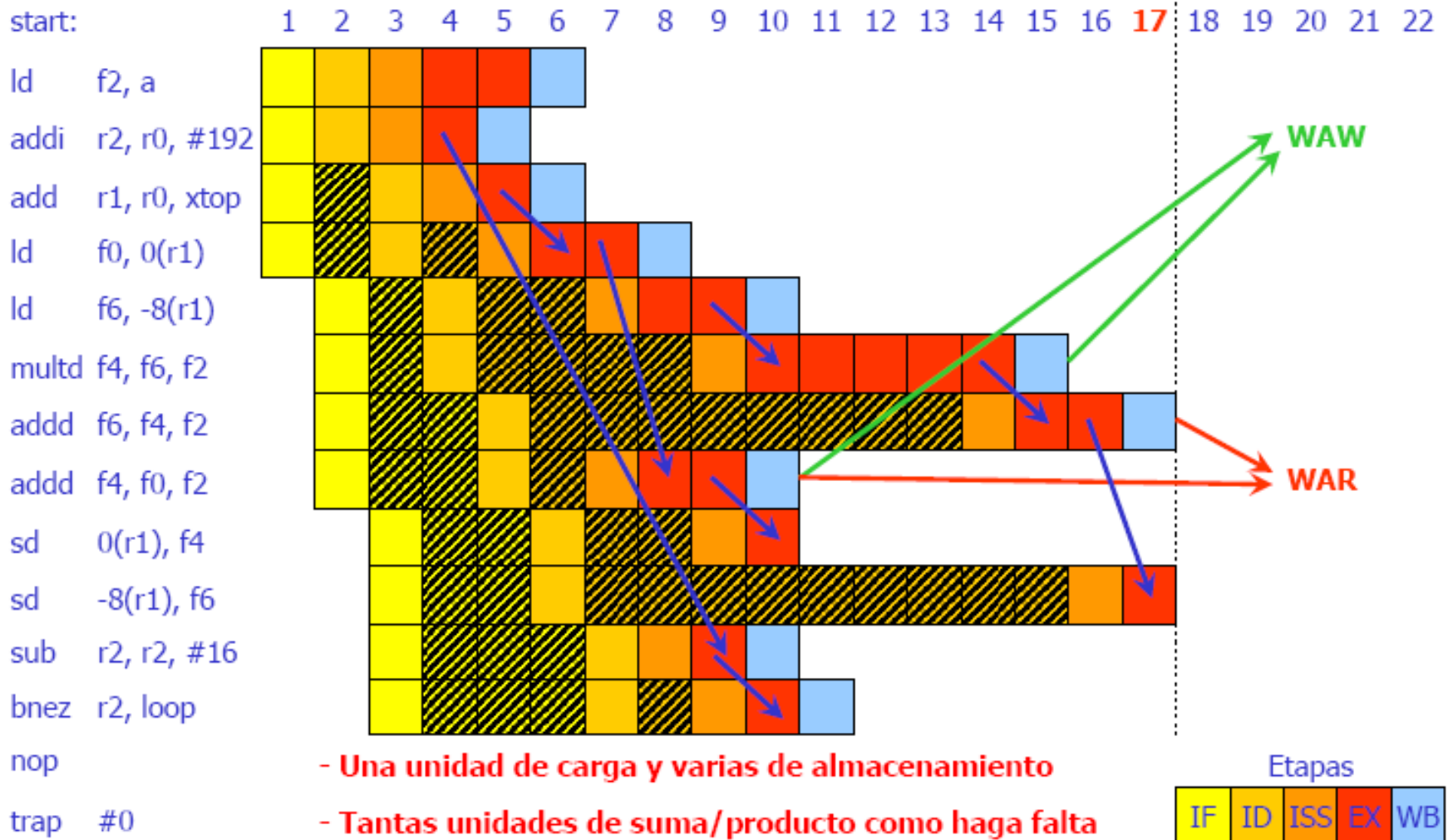
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión desordenada/finalización desordenada



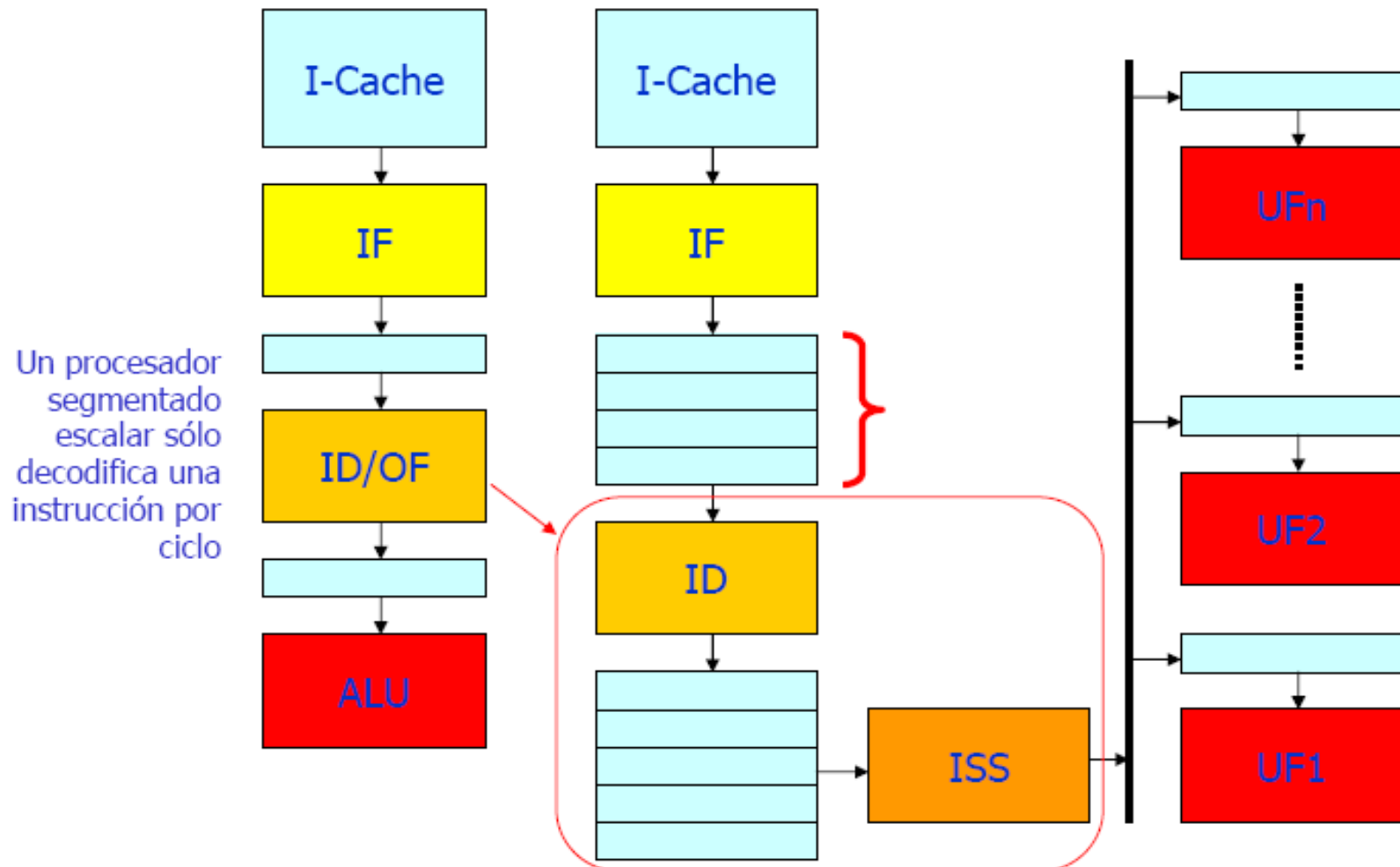
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación



En un procesador superescalar se han de decodificar varias instrucciones por ciclo (y comprobar las dependencias con las instrucciones que se están ejecutando)

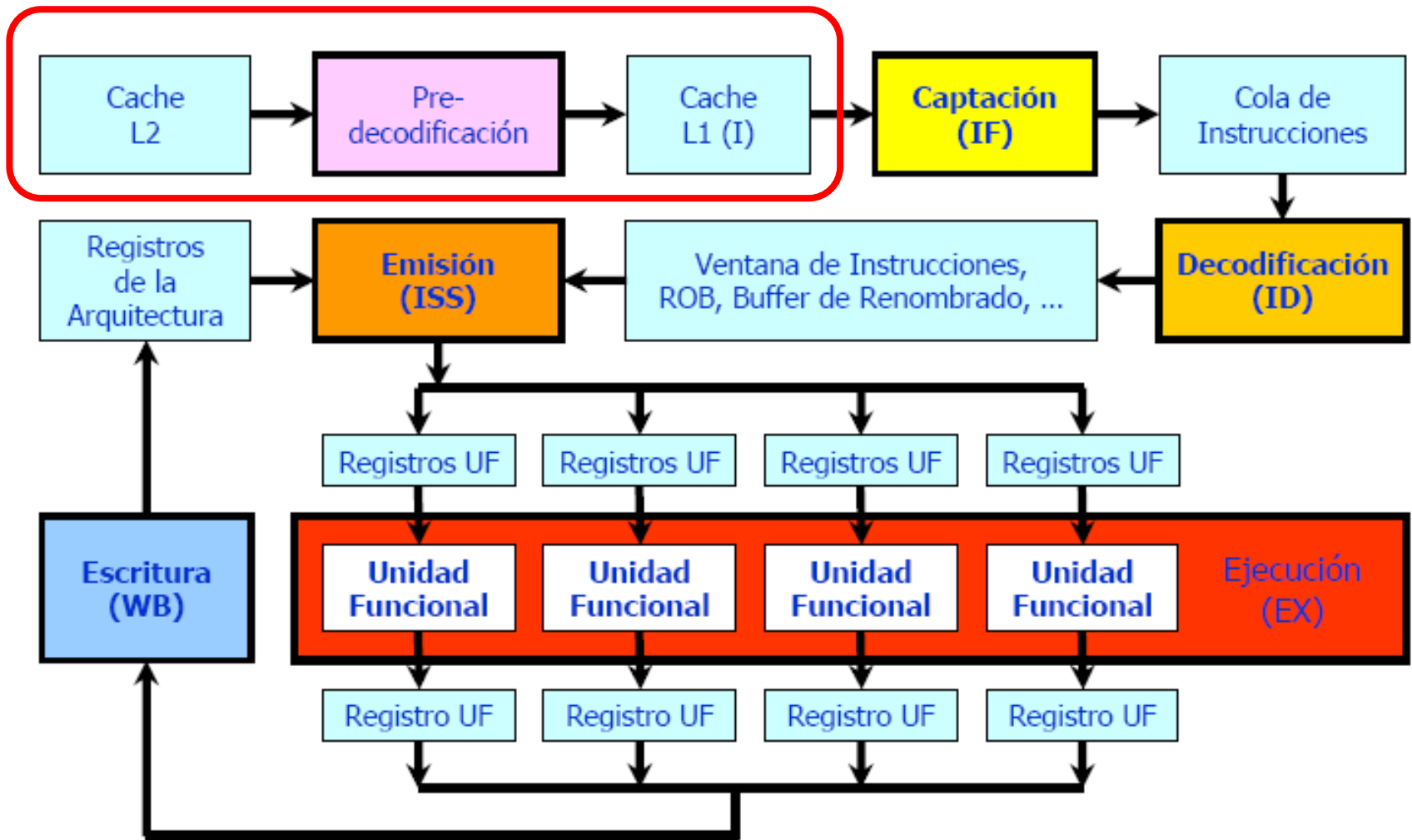
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

- Bits de predecodificación pueden indicar:
 - Si es una instrucción de salto o no (se puede empezar su procesamiento antes)
 - El tipo de unidad funcional que va a utilizar (se puede emitir más rápidamente si hay cauces para enteros o coma flotante...)
 - Si hace referencia a memoria o no

Unidad 2. Superescalares

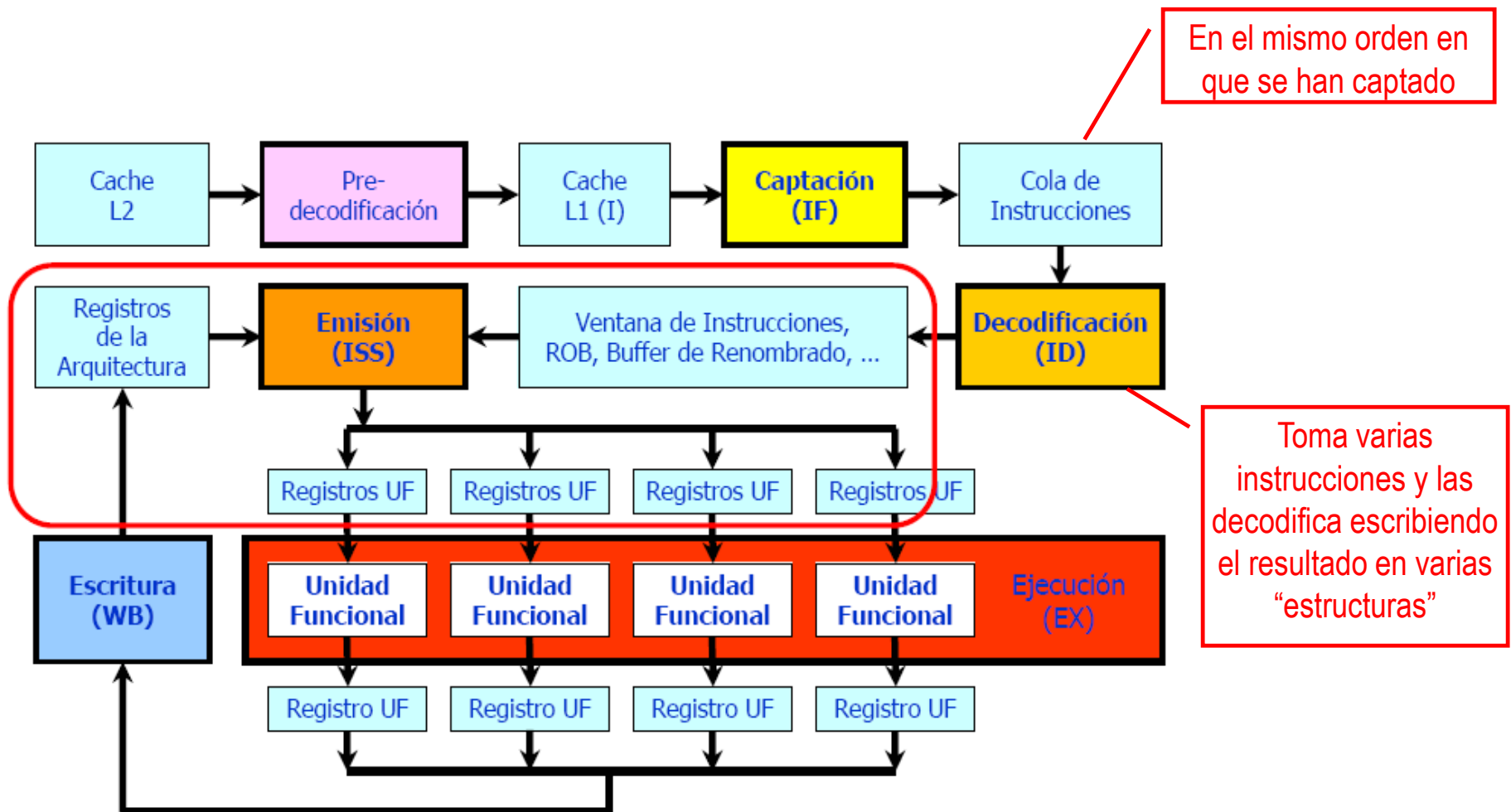
2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Ventana de instrucciones:
 - La ventana de instrucciones almacena las instrucciones pendientes (todas, si la ventana es centralizada o las de un tipo determinado, si es distribuida)
 - Las instrucciones se cargan en la ventana una vez decodificadas y se utiliza un bit para indicar si un operando está disponible (se almacena el valor o se indica el registro desde donde se lee) o no (se almacena la unidad funcional desde donde llegará el operando)
 - Una instrucción puede ser emitida cuando tiene todos sus operandos disponibles y la unidad funcional donde se procesará. Hay diversas posibilidades para el caso en el que varias instrucciones estén disponibles (características de los buses, etc.)

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

Ejemplo de Ventana de Instrucciones

#	opcode	address	rb_entry	operand1	ok1	typ1	operand2	ok2	typ2	pred
2	MULTD	loop + 0x4	2	1	0	FPD	0	0	FPD	—
1	LD	loop	1	0	0	INT	0	1	IMM	—

Dato no válido
(indica desde dónde se recibirá el dato)

Lugar donde se
almacenará el resultado

Dato válido
(igual a 0)

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

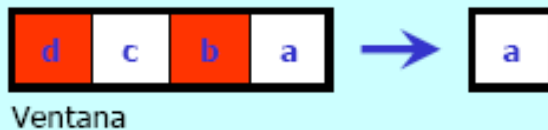
- **Orden:** Emisión Ordenada o Desordenada
- **Alineamiento:** Emisión Alineada o No alineada

Ejemplos:

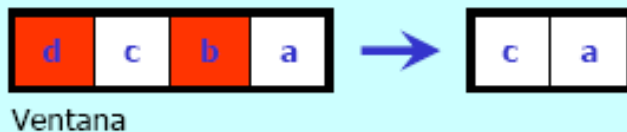
Alineada: SuperSparc (92), PowerPC (93, 95, 96), PA8000 (96), Alpha (92, 94, 95), R10000 (96)


No Alineada: MC88110 (93), PA7100LC (93), R8000 (94), UltraSparc (95)

Emisión Ordenada

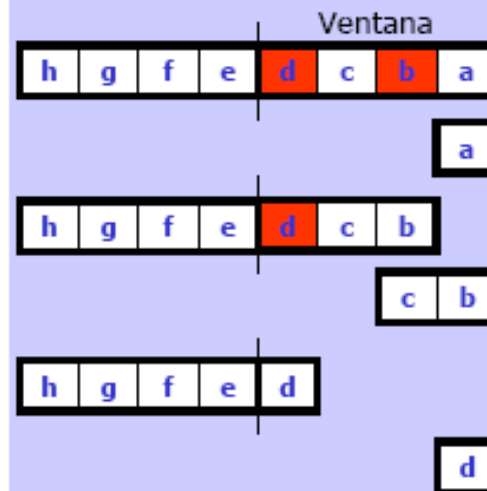


Emisión Desordenada



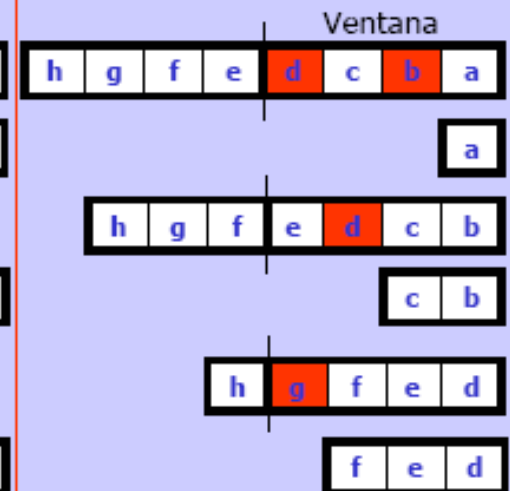
 Instrucción no preparada para la emisión

Emisión Alineada



Ordenada

Emisión No alineada



Ordenada

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Ventana de Registros

add/sub: 2
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

Ha terminado [2]: pueden emitirse [3] y [4]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



sub r5 [r4] 1 - [r3] 1 -
sub r6 [r5] 1 - [r2] 1 -

[4] puede emitirse pero debe esperar a la [3]

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(7)	(8)-(9)

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Ventana de Registros

add/sub: 2
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

Ha terminado [2]: pueden emitirse [3] y [4]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



sub r5 [r4] 1 - [r3] 1 -
sub r6 [r5] 1 - [r2] 1 -

[4] puede emitirse pero debe esperar a la [3]

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(7)	(8)-(9)

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Ventana de Registros

add/sub: 2
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

No afecta al tiempo

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(4)	(5)-(6)

↓ Ha terminado [1]

Se ha emitido [4] y ha terminado [2]: puede emitirse [3]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



sub r6 [r5] 1 - [r2] 1 -

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Ventana de Registros

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

sub r5 [r4] 1 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



Ha terminado [2] y se ha emitido [3].
Cuando la unidad quede libre se emitirá [4]

sub r5 [r4] 1 - [r3] 1 -

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(10)	(11)-(12)

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Ventana de Registros

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

Se ganan
3 ciclos

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(4)	(5)-(6)

↓ Ha terminado [1]

Se ha emitido [4] y ha terminado [2]: puede emitirse [3]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



sub r6 [r5] 1 - [r2] 1 -

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva

sub	r5	r4	0	add	[r3]	1	-
sub	r6	r5	0	mult	[r2]	1	-
mult	r5	[r1]	1	-	[r5]	1	-
add	r4	[r1]	1	-	[r2]	1	-

Cola de Instrucciones

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)



sub	r5	r4	0	add	[r3]	1	-

Cola de Instrucciones

ID/ISS

sub r6 r5 0 mult [r2] 1 -
add r4 [r1] 1 - [r2] 1 -

Estaciones de Reserva
(con capacidad para dos instrucciones)

mult r5 [r1] 1 - [r5] 1 -

Unidad 2. Superescalares

2.2. Cauce superescalar

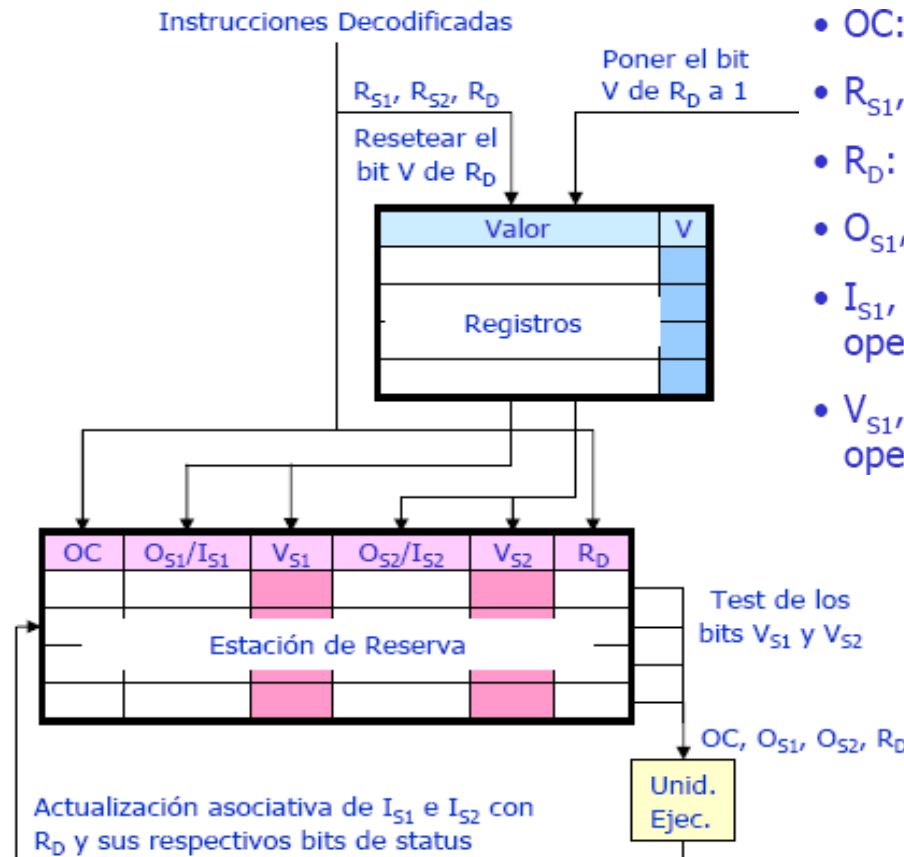
Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva



• OC: Código de operación

• R_{S1}, R_{S2} : Registros fuente

• R_D : Registro de destino

• O_{S1}, O_{S2} : Operandos fuente

• I_{S1}, I_{S2} : Identificadores de los operandos fuente

• V_{S1}, V_{S2} : Bits válidos de los operandos fuente

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (1)

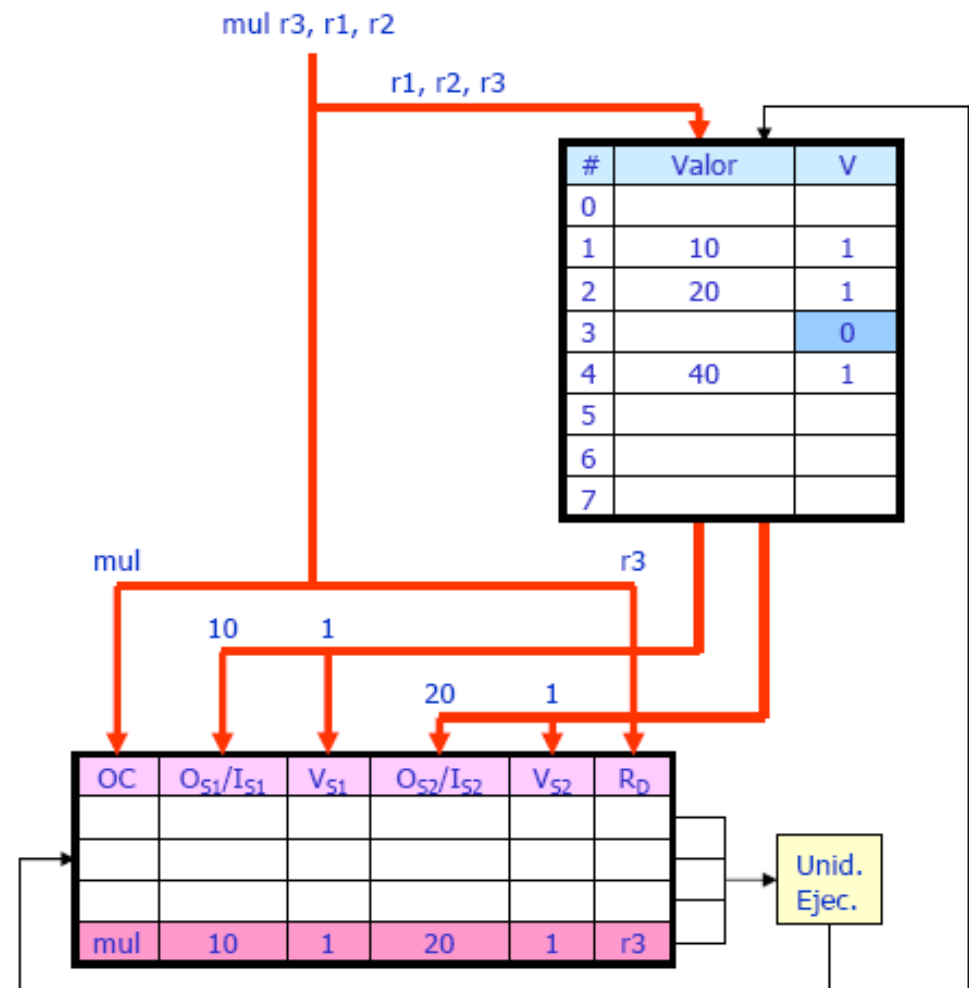
Ciclo i: mul r3, r1, r2

Ciclo i+1: add r5, r2, r3

add r6, r3, r4

Ciclo i:

- Se emite la instrucción de multiplicación, ya decodificada, a la estación de reserva
- Se anula el valor de r3 en el banco de registros
- Se copian los valores de r1 y r2 (disponibles) en la estación de reserva



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

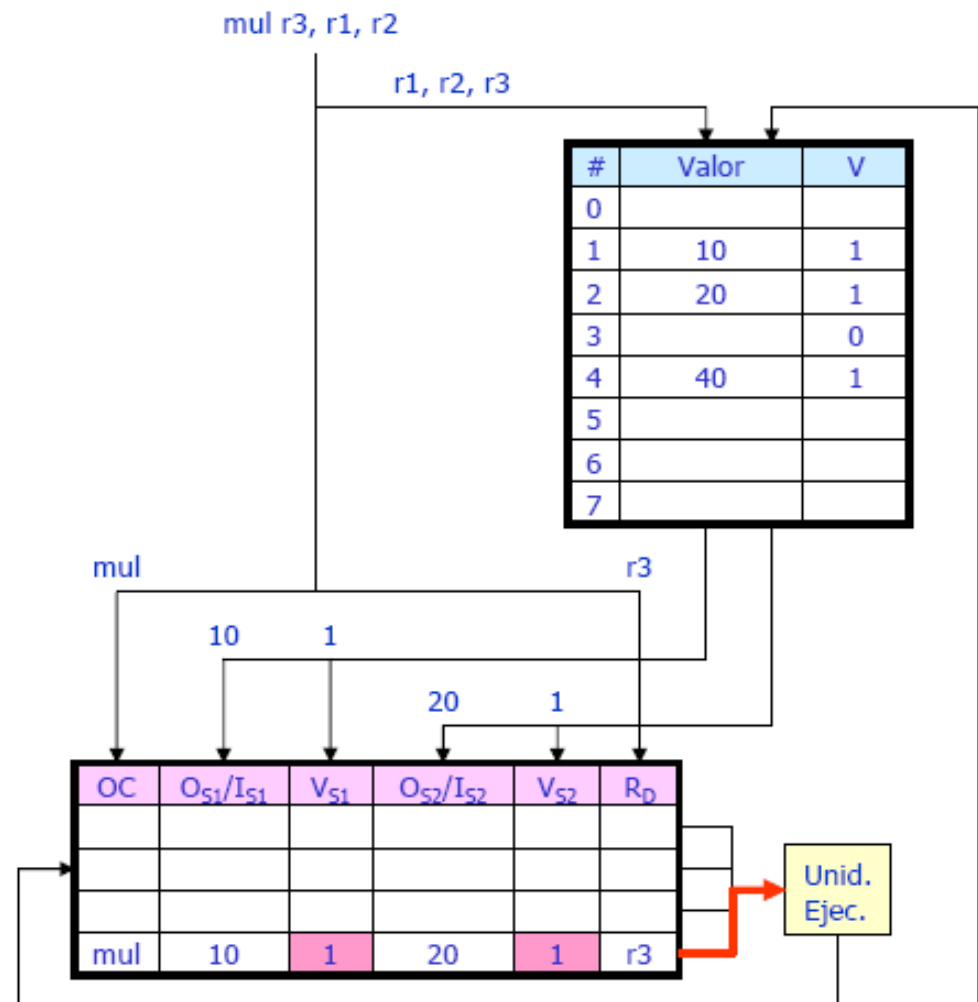
Emisión

- Estaciones de reserva. Ejemplo de uso (2)

Ciclo i: mul r3, r1, r2
Ciclo i+1: add r5, r2, r3
add r6, r3, r4

Ciclo i + 1:

- La operación de multiplicación tiene sus operadores preparados ($V_{S1} = 1$ y $V_{S2} = 1$)
- Así que puede enviarse a la unidad de ejecución



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (3)

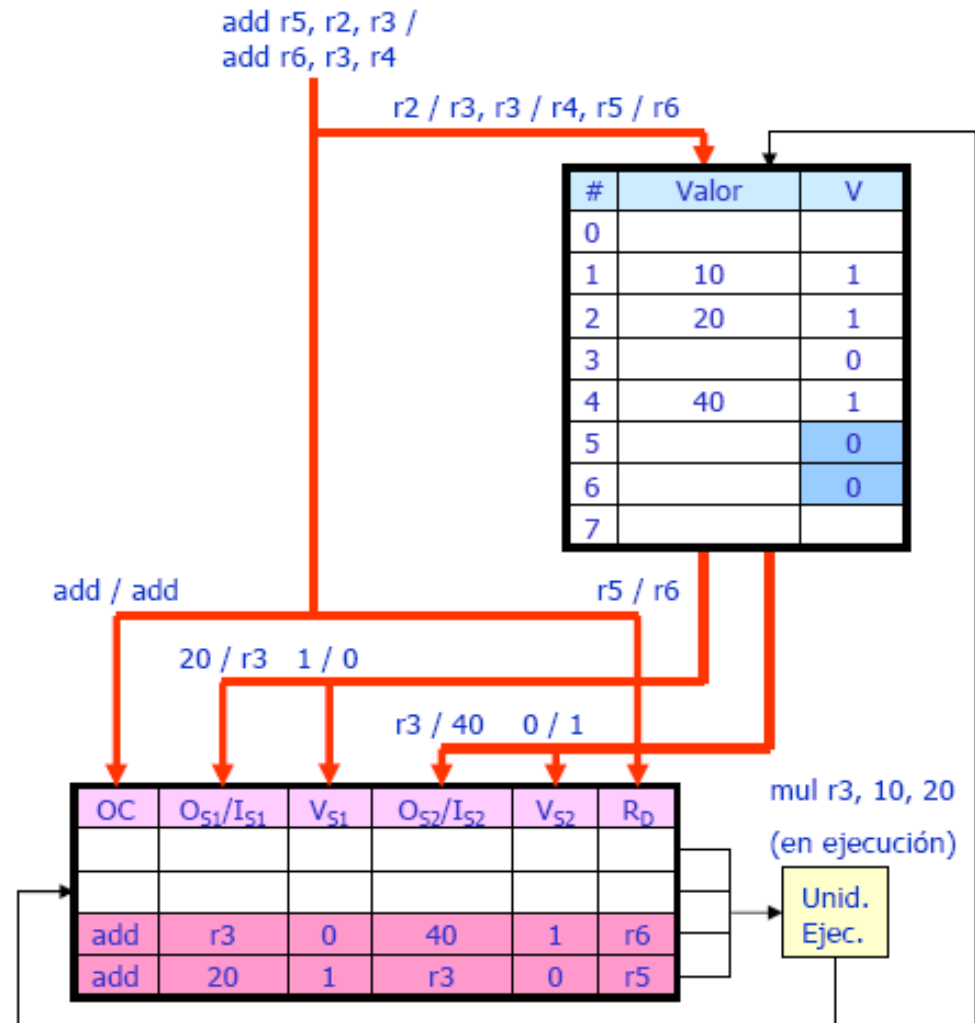
Ciclo i: mul r3, r1, r2

Ciclo i+1: add r5, r2, r3

add r6, r3, r4

Ciclo i + 1 (cont.):

- Se emiten las dos instrucciones de suma a la estación de reserva
- Se anulan los valores de r5 y r6 en el banco de registros
- Se copian los valores de los operandos disponibles y los identificadores de los operandos no preparados



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

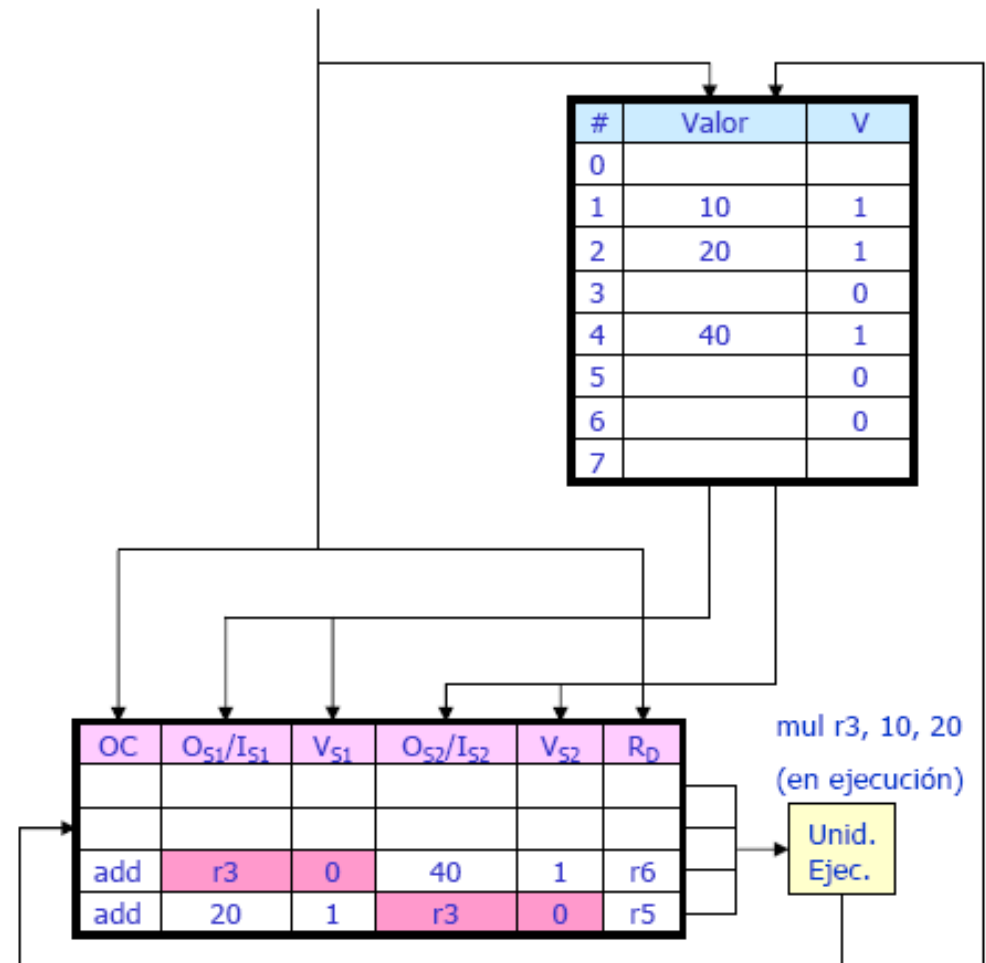
Emisión

- Estaciones de reserva. Ejemplo de uso (4)

Ciclo i: mul r3, r1, r2
Ciclo i+1: add r5, r2, r3
 add r6, r3, r4

Ciclos i + 2 .. i + 5:

- La multiplicación sigue ejecutándose
- No se puede ejecutar ninguna suma hasta que esté disponible el resultado de la multiplicación (r3)



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (5)

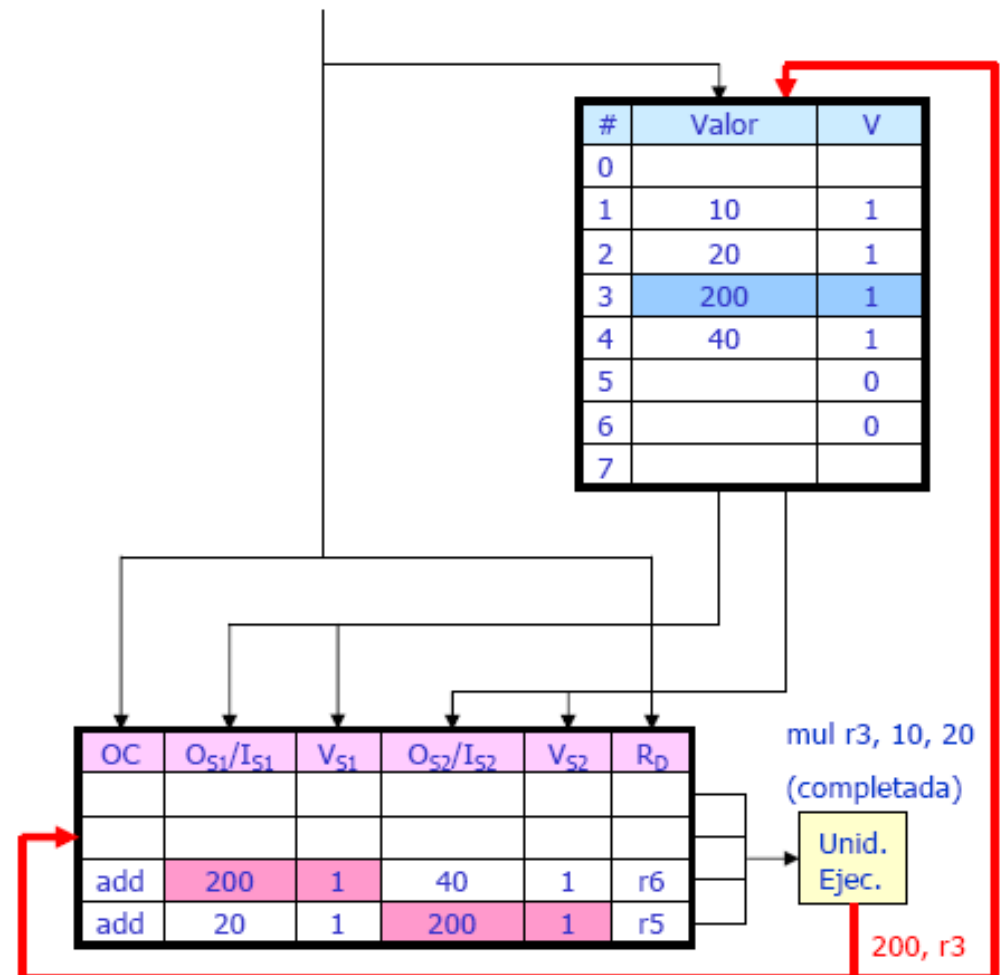
Ciclo i: mul r3, r1, r2

Ciclo i+1: add r5, r2, r3

add r6, r3, r4

Ciclo i + 6:

- Se escribe el resultado de la multiplicación en el banco de registros y en las entradas de la estación de reserva
- Se actualizan los bits de disponibilidad de r3 en el banco de registros y en la estación de reserva



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

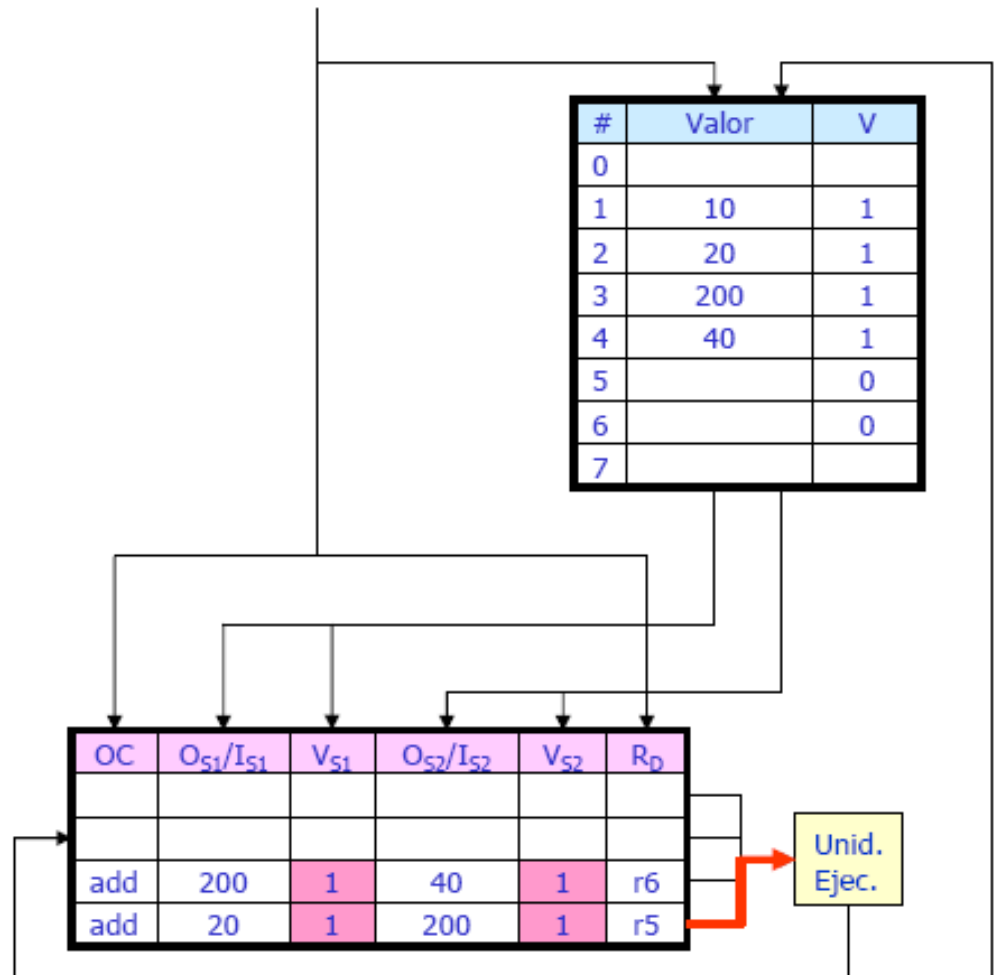
Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (6)

Ciclo i: mul r3, r1, r2
Ciclo i+1: add r5, r2, r3
 add r6, r3, r4



Ciclo i + 6 (*cont.*):

- Las sumas tienen sus operadores preparados (VS1 = 1 y VS2 = 1)
- Así que pueden enviarse a la unidad de ejecución

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (7)

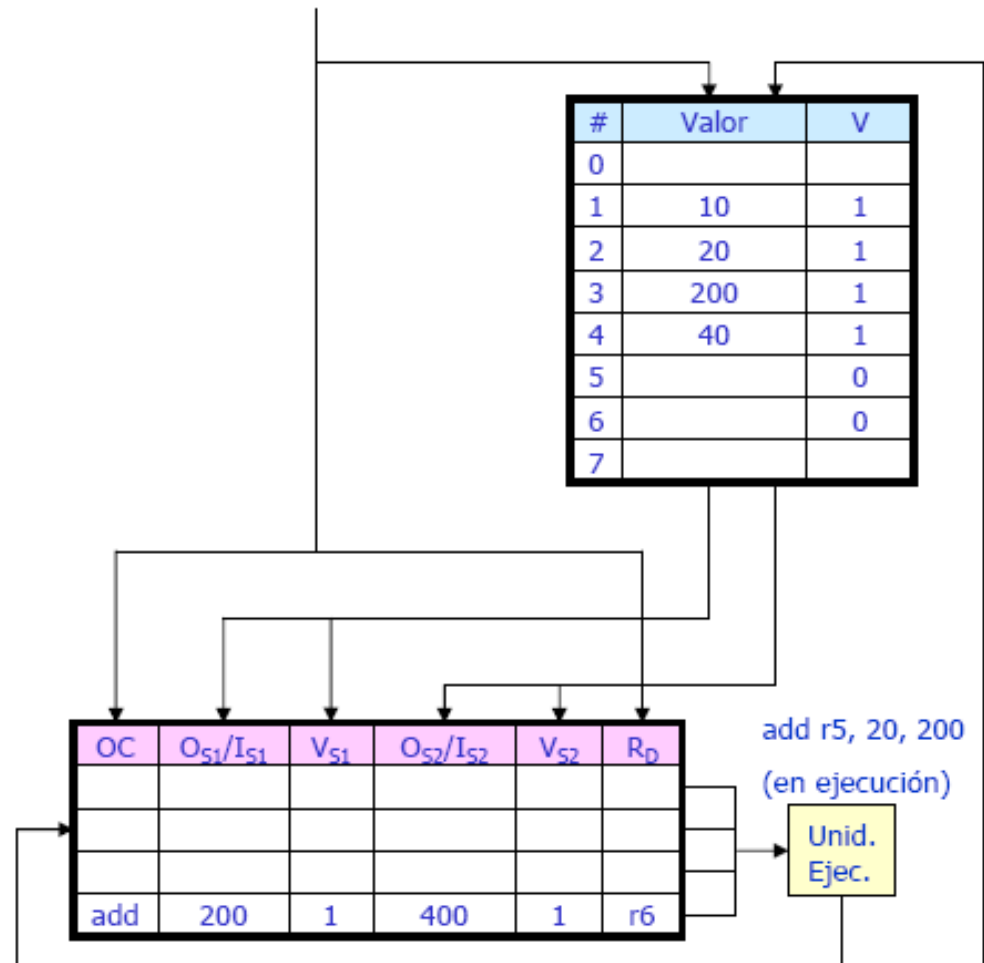
Ciclo i: mul r3, r1, r2

Ciclo i+1: add r5, r2, r3

add r6, r3, r4

Ciclo i + 6 (*cont.*):

- Como sólo hay una unidad de ejecución, se envía la instrucción más antigua de la estación de reserva, la primera suma



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Ejercicio
 - Emisión ordenada/desordenada
 - Recursos: 1 lw/sw, 1 add/sub, 1 mult/div
 - Recursos: 1 lw/sw, 2 add/sub, 2 mult/div
 - Latencia: lw/sw (5 ciclos), add/sub (2 ciclos), mult/div (5 ciclos)

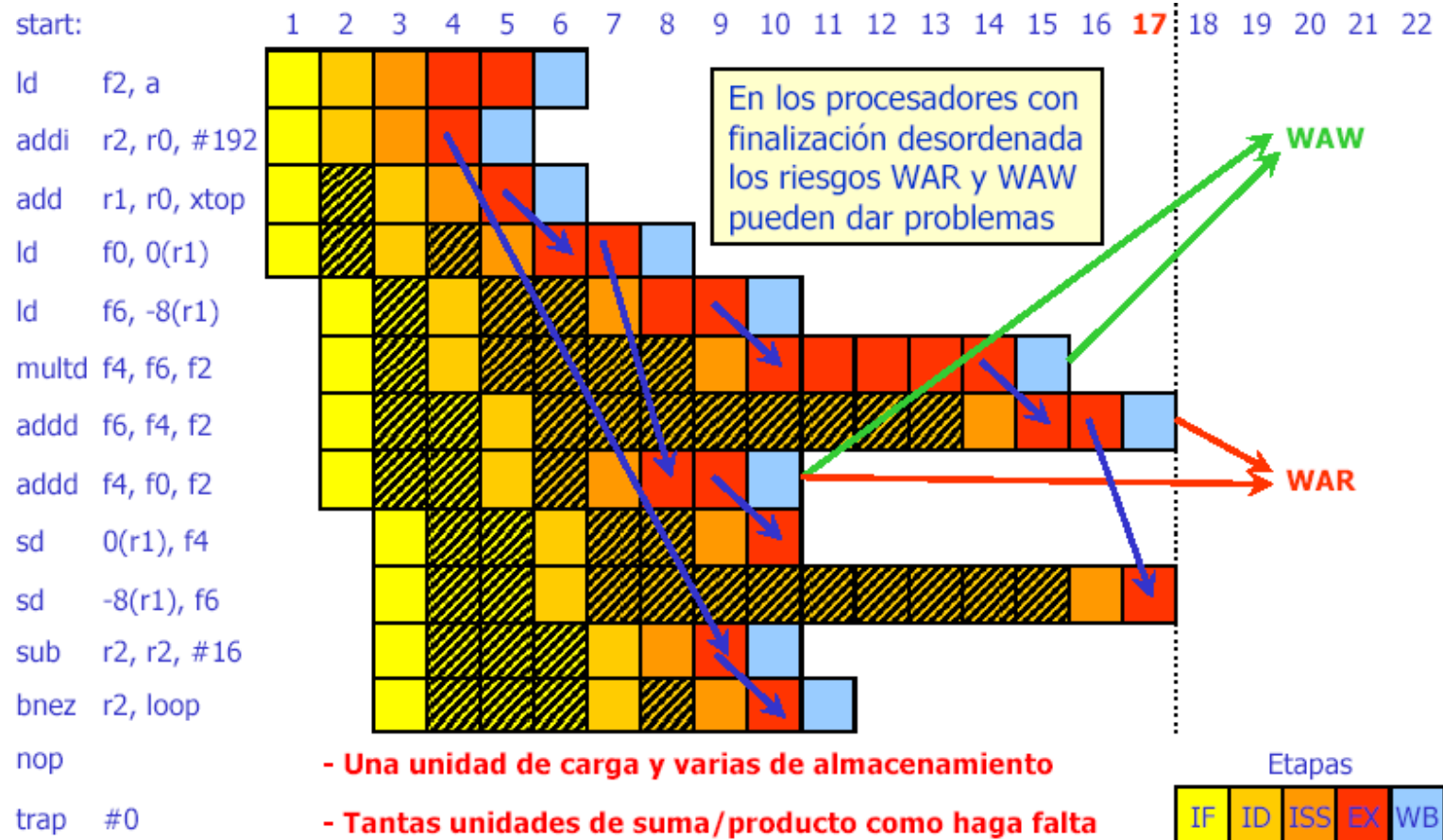
```
Lw r1,0(r2)
Add r2,r1,r3
Mult r3,r1,r2
Sub r4,r1,r2
Add r4,r1,r2
Div r4,r5,r6
```

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

- Renombrado de registros (motivación)



Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

- Técnica para **evitar el efecto de las dependencias WAR, o Antidependencias** (en la emisión desordenada) y **WAW, o Dependencias de Salida** (en la ejecución desordenada).

R3 := R3 - R5

R4 := R3 + 1

R3 := R5 + 1

R7 := R3 * R4



Cada escritura se asigna
a un registro físico distinto

R3b := R3a - R5a

R4a := R3b + 1

R3c := R5a + 1

R7a := R3c * R4a

Sólo RAW

R.M. Tomasulo (67)

Implementación Estática: Durante la Compilación

Implementación Dinámica: Durante la Ejecución (circuitaría adicional y registros extra)

Características de los Buffers de Renombrado

- **Tipos de Buffers** (separados o mezclados con los registros de la arquitectura)
- **Número de Buffers de Renombrado**
- **Mecanismos para acceder a los Buffers** (asociativos o Indexados)

Velocidad del Renombrado

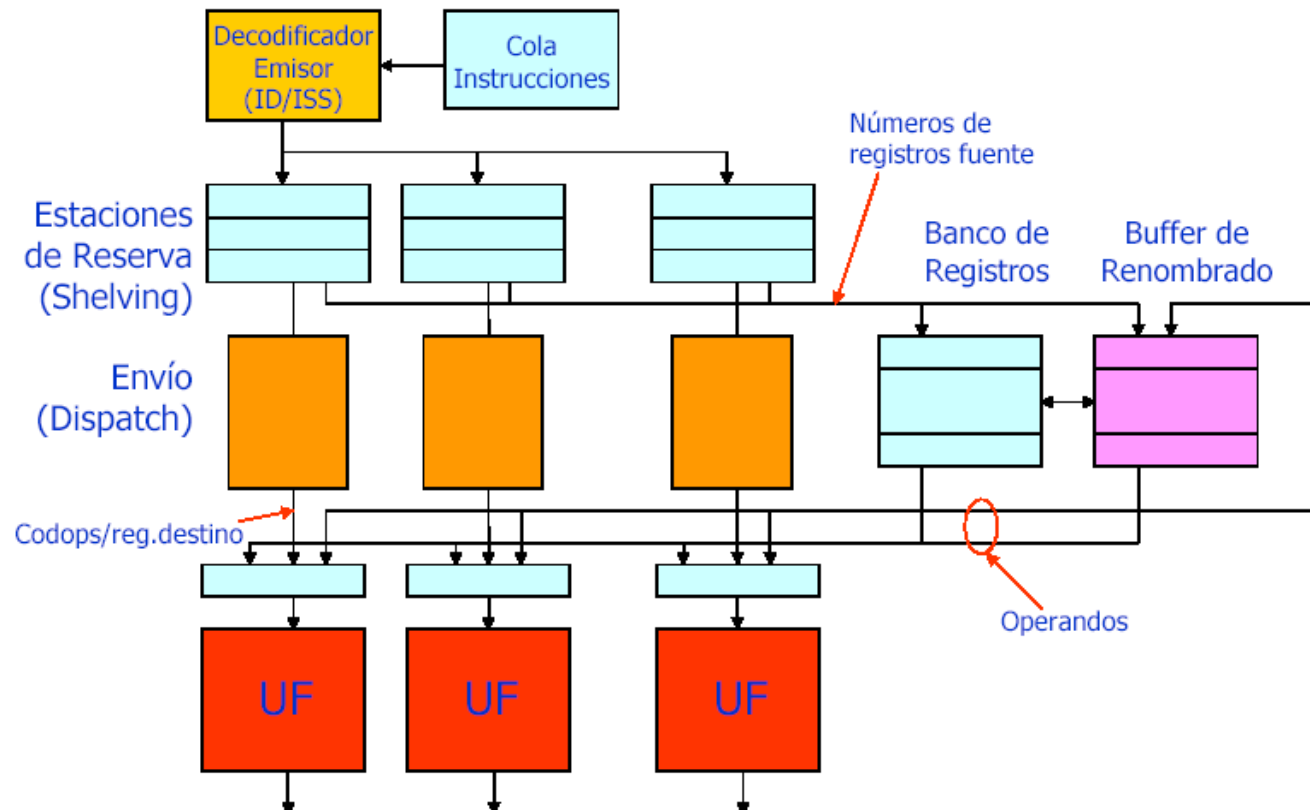
- **Máximo número de nombres asignados por ciclo** que admite el procesador

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

- Renombrado de registros
 - Estaciones de reserva + buffer de renombrado



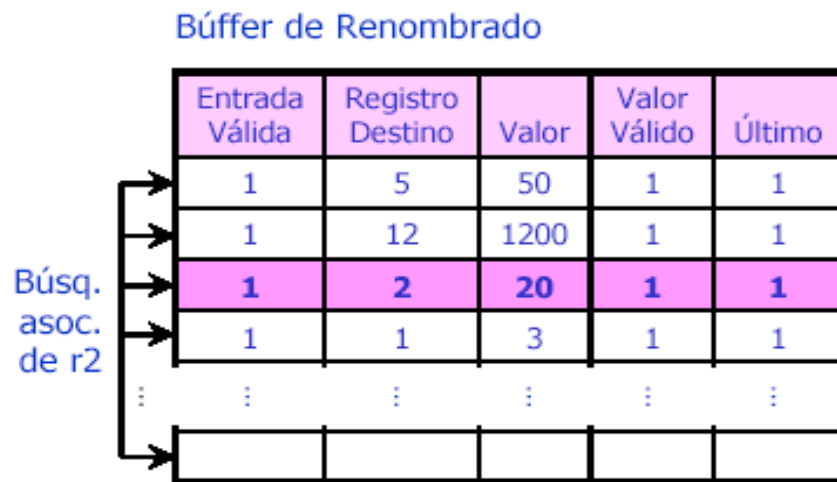
Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

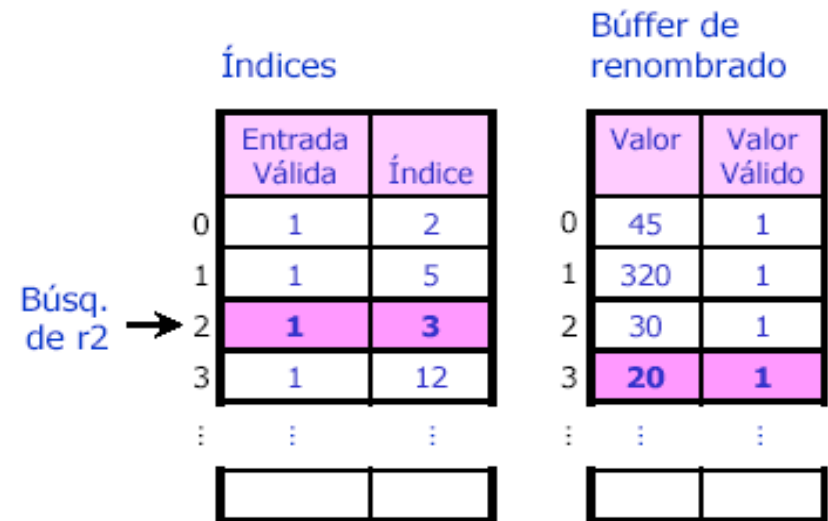
- Tipos de buffers de renombrado

Buffer de Renombrado con Acceso Asociativo



- Permite varias escrituras pendientes a un mismo registro
- Se utiliza el bit último para marcar cual ha sido la más reciente

Buffer de Renombrado con Acceso Indexado



- Sólo permite una escritura pendiente a un mismo registro
- Se mantiene la escritura más reciente

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (I)

	Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	0
3	1	1	15	1	1
4	0				
5	0				
6	0				
7	0				
8	0				
9	0				
10	0				
11	0				
12	0				
13	0				
14	0				

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

$r2a = r0a * r1a$
 $r3a = r1a + r2a$
 $r2b = r0a - r1a$

Situación Inicial:

- Existen dos renombrados de r1 en las entradas 2 y 3 del buffer
- La última está en la entrada 3




Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (II)

	Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	0
3	1	1	15	1	1
4	1	2		0	1
5	0				
6	0				
7	0				
8	0				
9	0				
10	0				
11	0				
12	0				
13	0				
14	0				

 **r0: 0, "válido"**  **r1: 15, "válido"**  **4**

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo i:

- Se emite la multiplicación
- Se accede a los operandos de la multiplicación, que tienen valores válidos en el buffer de renombrado
- Se renombra r2 (el destino de mul)

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (III)

	Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	0
3	1	1	15	1	1
4	1	2		0	1
5	1	3		0	1
6	0				
7	0				
8	0				
9	0				
10	0				
11	0				
12	0				
13	0				
14	0				

 **r1: 15, "válido"**  **r2: "no válido"**  **5**

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo $i + 1$:

- Se emite la suma
- Se accede a sus operandos, pero r2 no estará preparado hasta que termine la multiplicación
- Se renombra r3 (destino de add)

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (IV)

	Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	0
3	1	1	15	1	1
4	1	2		0	0
5	1	3		0	1
6	1	2		0	1
7	0				
8	0				
9	0				
10	0				
11	0				
12	0				
13	0				
14	0				


r0: 0, "válido"


r1: 15, "válido"


6

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo i + 2:

- Se emite la resta
- Se accede a sus operandos
- Se vuelve a renombrar r2 (destino de sub)

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (V)

	Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	0
3	1	1	15	1	1
4	1	2	0	1	0
5	1	3		0	1
6	1	2		0	1
7	0				
8	0				
9	0				
10	0				
11	0				
12	0				
13	0				
14	0				

 **r1: 15, "válido"**  **r2: 0, "válido"**

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo i + 5:

- Termina la multiplicación
- Se actualiza el resultado en el buffer de renombrado
- Ya se puede ejecutar la suma con el valor de r2 de la entrada 4
- Cuando termine la resta escribirá otro valor para r2 en la entrada 6
- Sólo se escribirá en el banco de registros el último valor de r2

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva

Emisión de la multiplicación

mult r2,r0,r1
add r3,r1,r2
sub r2,r0,r1

Cola de Instrucciones

Se añade la línea en el RB durante la emisión

2

Banco de Registros

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

1

Se buscan los operando en el Buffer

2

Los que no estén
Se buscan en el
Banco de Regs

Buffer de Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			1
2	0				
3	0				
4	0				

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
mult	0	1	10	1	1

mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

add/
sub

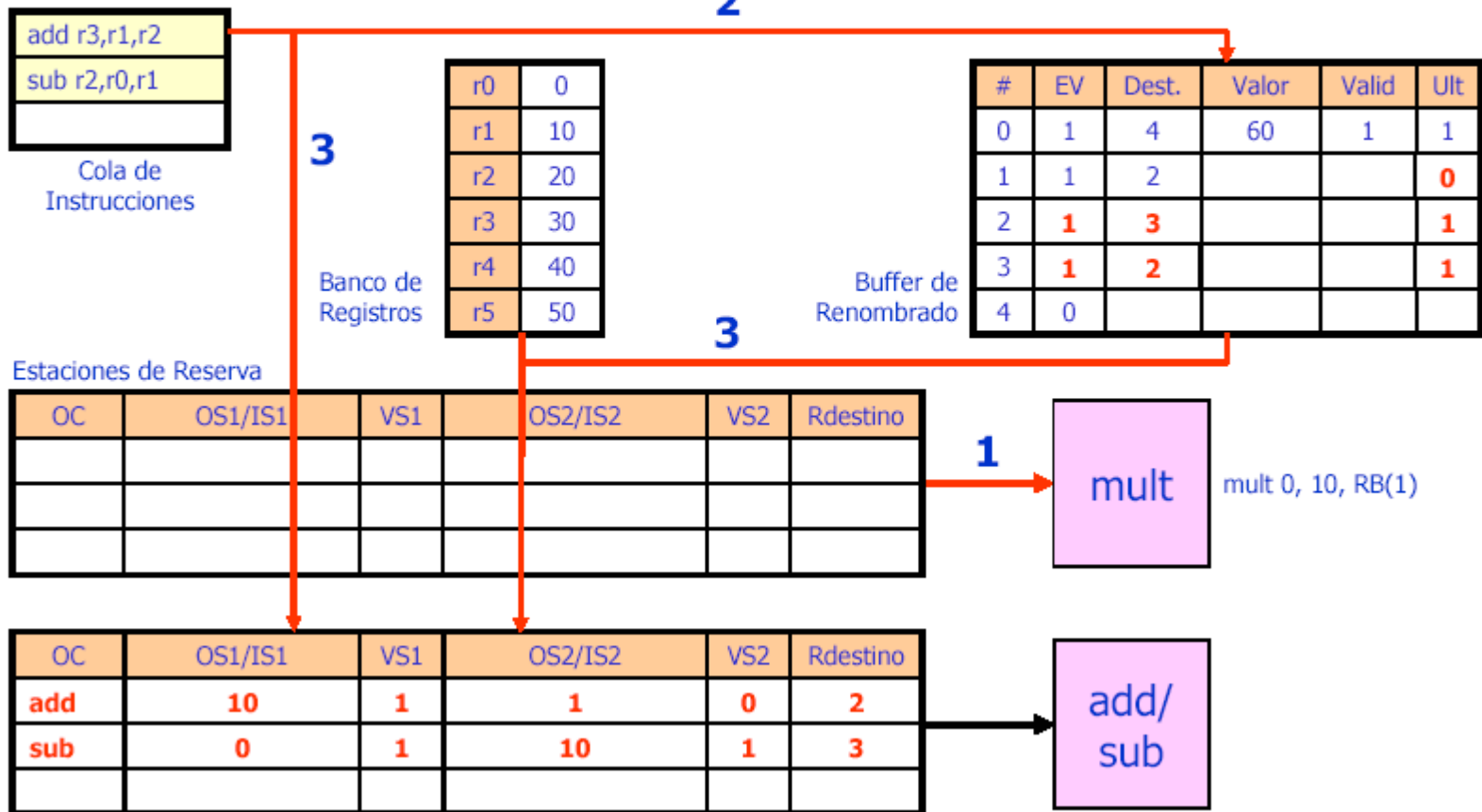
Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (II)

Envío de la multiplicación y emisión de las suma y la resta
2



Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (III)

Envío de la resta

Cola de Instrucciones

Banco de Registros

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Buffer de Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2			1
4	0				

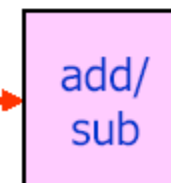
Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



sub 0, 10, RB(3)

1

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (IV)

Termina la resta

Cola de Instrucciones

Banco de Registros

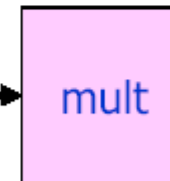
r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Buffer de Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2	-10	1	1
4	0				

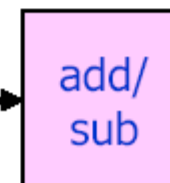
Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



sub 0, 10, RB(3)

1



Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (V)

Termina la multiplicación

Cola de Instrucciones

Banco de Registros

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Buffer de Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva

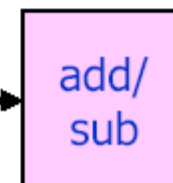
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	0	1	2



Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VI)

Envío de la suma

Cola de
Instrucciones

Banco de
Registros

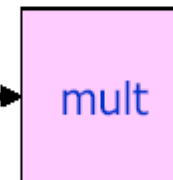
r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Buffer de
Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

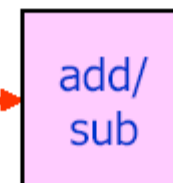
Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

1



add 10, 0, RB(2)

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Termina la suma

Cola de
Instrucciones

Banco de
Registros

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Buffer de
Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3	10	1	1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

add/
sub

add 10, 0, RB(2)

1

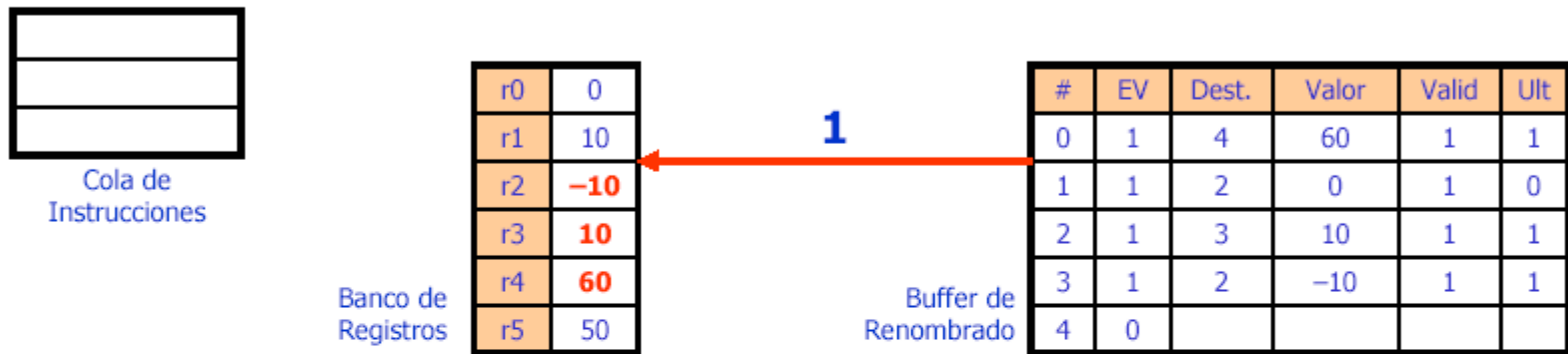
Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

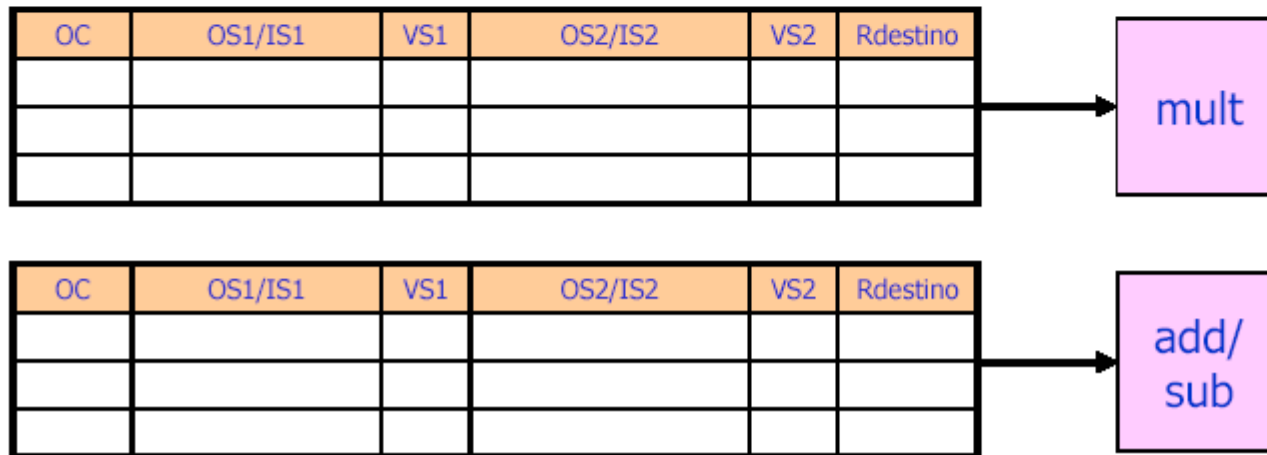
Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Se actualizan los registros



Estaciones de Reserva

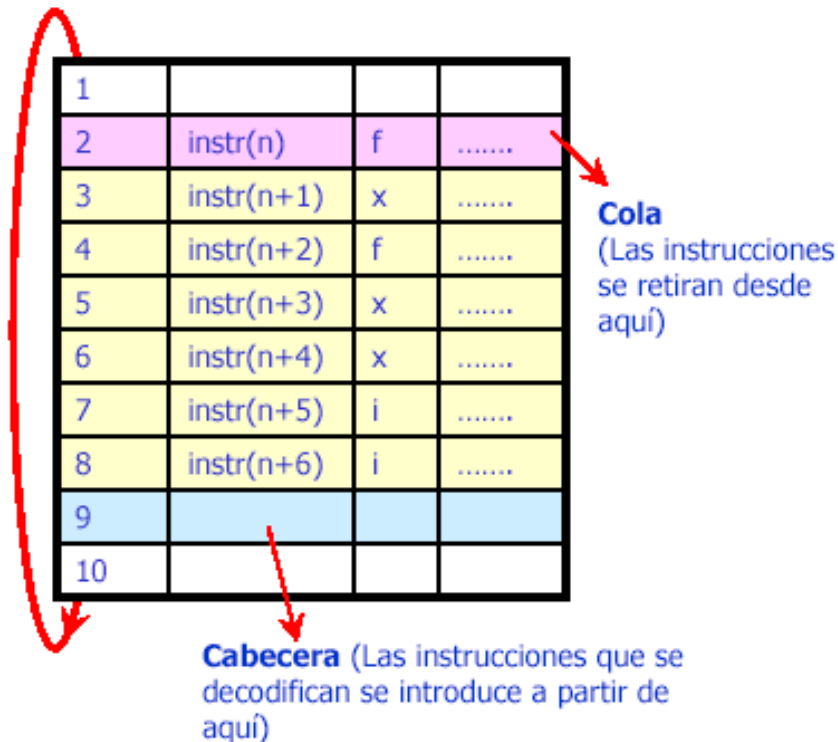


Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden



La gestión de interrupciones y la ejecución especulativa se realizan fácilmente mediante el ROB

- El puntero de cabecera apunta a la siguiente posición libre y el **puntero de cola a la siguiente instrucción a retirar**.
- Las instrucciones **se introducen en el ROB en orden de programa estricto** y pueden estar marcadas como **emitidas (issued, i)**, en **ejecución (x)**, o **finalizada su ejecución (f)**.
- Las **instrucciones sólo se pueden retirar** (se produce la finalización con la escritura en los registros de la arquitectura) **si han finalizado**, y todas las que les preceden también.
- La **consistencia se mantiene porque sólo las instrucciones que se retiran del ROB se completan** (escriben en los registros de la arquitectura) y se retiran en el orden estricto de programa.

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (I)

I1: mult r1, r2, r3

I2: st r1, 0x1ca

I3: add r1, r4, r3

I4: xor r1, r1, r3



Dependencias:

RAW: (I1,I2), (I3,I4)

WAR: (I2,I3), (I2,I4)

WAW: (I1,I3), (I1,I4), (I3,I4)

I1: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r2** y **r3**)

I2: Se envía a la unidad de almacenamiento hasta que esté disponible **r1**

I3: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r4** y **r3**)

I4: Se envía a la estación de reserva de la ALU para esperar a **r1**

Estación de Reserva (Unidad de Almacenamiento)

codop	dirección	op1	ok1
st	0x1ca	3	0

Estación de Reserva (ALU)

codop	dest	op1	ok1	op2	ok2
xor	6	5	0	[r3]	1

Líneas del ROB

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (II)

Ciclo 7

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	-	0	x	9
6	xor	10	r1	int_alu	-	0	i	-

Ciclo 9 No se puede retirar **add** aunque haya finalizado su ejecución

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	-	0	x	10

Ciclo 10 Termina **xor**, pero todavía no se puede retirar

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (III)

Ciclo 12

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	33	1	f	12
4	st	8	-	store	-	1	f	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Ciclo 13

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

- Se ha supuesto que se pueden retirar dos instrucciones por ciclo.
- Tras finalizar las instrucciones **mult** y **st** en el ciclo 12, se retirarán en el ciclo 13.
- Después, en el ciclo 14 se retirarán las instrucciones **add** y **xor**.

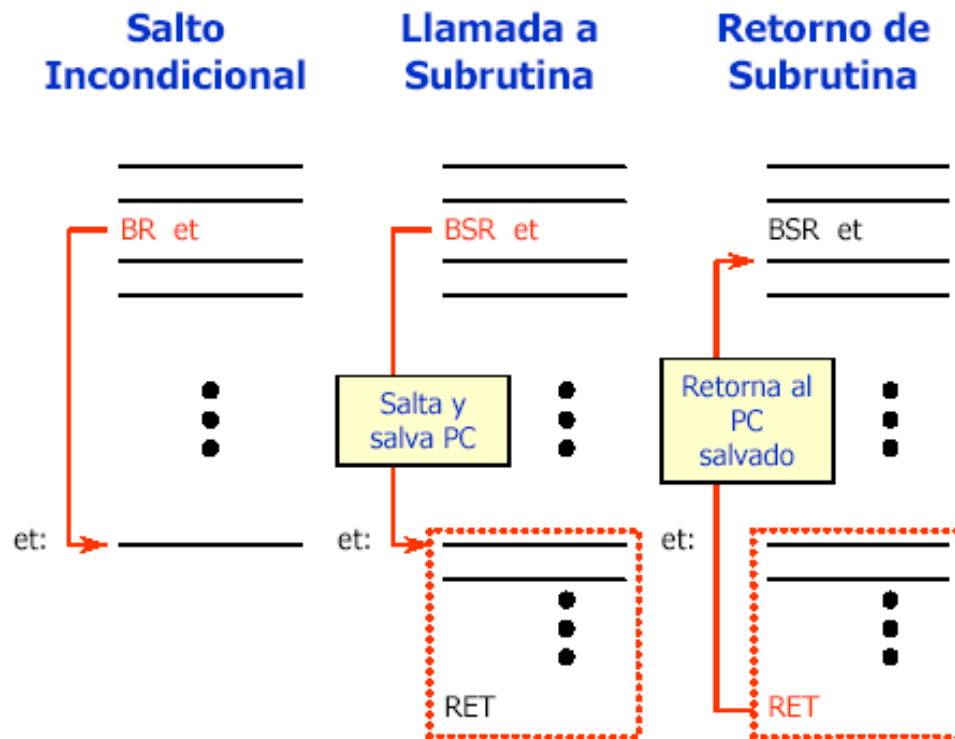
Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

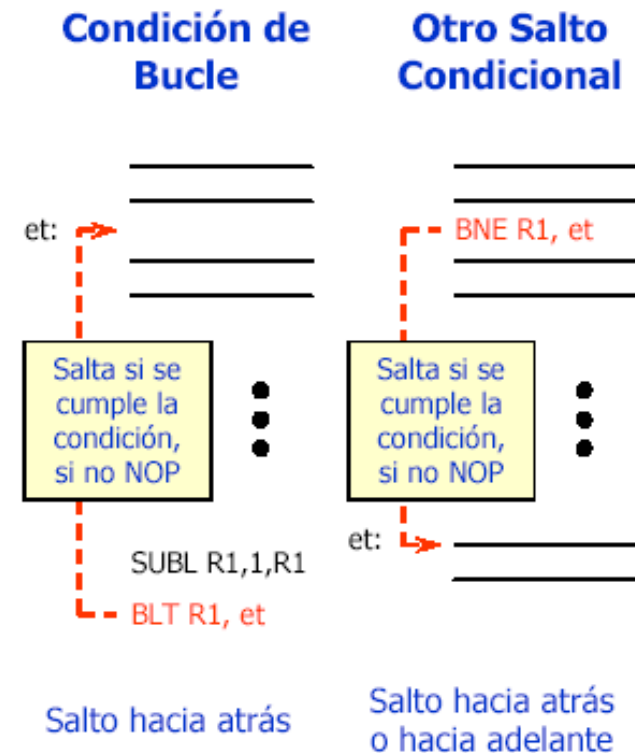
Riesgos

Clasificación de los Saltos

Saltos Incondicionales



Saltos Condicionales



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

El **efecto de los saltos en los procesadores superescalares es más pernicioso** ya que, al emitirse varias instrucciones por ciclo, *prácticamente en cada ciclo* puede haber una instrucción de salto.

El **salto retardado no tiene mucho interés** porque la unidad de emisión decide las instrucciones que pasan a ejecutarse teniendo en cuenta las dependencias.

- **Detección de la Instrucción de Salto**

Cuanto antes se detecte que una instrucción es de salto menor será la posible penalización. Los saltos se detectan usualmente en la fase de decodificación.

- **Gestión de los Saltos Condicionales no Resueltos**

Si en el momento en que la instrucción de salto evalúa la condición de salto ésta no se haya disponible se dice que *el salto o la condición no se ha resuelto*. Para resolver este problema se suele utilizar el **procesamiento especulativo del salto**.

- **Acceso a las Instrucciones destino del Salto**

Hay que determinar la forma de acceder a la secuencia a la que se produce el salto

- **La implementación física de las Unidades de Salto**

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

- Detección

- En etapa de decodificación



- Detección temprana ('early branch detection')

- Detección paralela a decodificación



- Detección anticipada ('look-ahead branch detection')



- Detección integrada en captación



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Uso de los ciclos que siguen a la inst. de salto condicional	Salto Retardado	Se utilizan los ciclos que siguen a la captación de una instrucción de salto para insertar instrucciones que deben ejecutarse independientemente del resultado del salto (Primeras arquitecturas RISC y posteriores)
Gestión de Saltos Condicionales no Resueltos (Una condición de salto no se puede comprobar si no se ha terminado de evaluar)	Bloqueo del Procesamiento del Salto	Se bloquea la instrucción de salto hasta que la condición esté disponible (68020, 68030, 80386)
	Procesamiento Especulativo de los Saltos	La ejecución prosigue por el camino más probable (se especula sobre las instrucciones que se ejecutarán). Si se ha errado en la predicción hay que recuperar el camino correcto. (Típica en los procesadores superescalares actuales)
	Múltiples Caminos	Se ejecutan los dos caminos posibles después de un salto hasta que la condición de salto se evalúa. En ese momento se cancela el camino incorrecto. (Máquinas VLIW experimentales: Trace/500 , URPR-2)
Evitar saltos condicionales	Ejecución Vigilada (<i>Guarded Exec.</i>)	Se evitan los saltos condicionales incluyendo en la arquitectura instrucciones con operaciones condicionales (IBM VLIW, Cydra-5, Pentium, HP PA, Dec Alpha)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Esquemas de Predicción de Salto

Predicción Fija

Se toma siempre la misma decisión: el salto siempre se realiza, *'taken'*, o no, *'not taken'*

Predicción Verdadera

La decisión de si se realiza o no se realiza el salto se toma mediante:

- **Predicción Estática:**
Según los atributos de la instrucción de salto (el código de operación, el desplazamiento, la decisión del compilador)
- **Predicción Dinámica:**
Según el resultado de ejecuciones pasadas de la instrucción (historia de la instrucción de salto)

Prestaciones

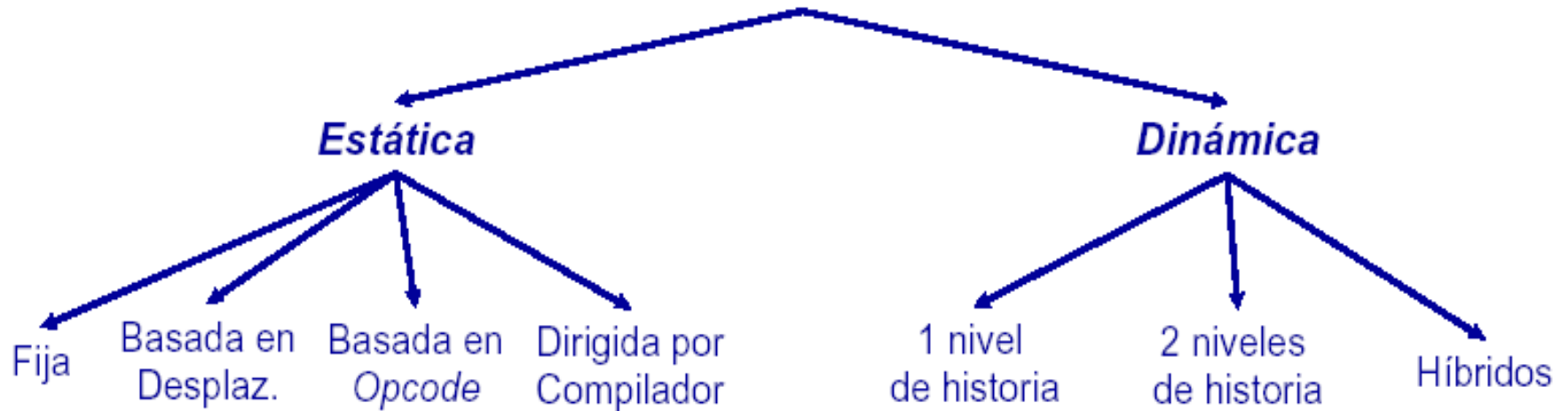
Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Predicción de saltos



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción fija

Aproximación 'Siempre No Tomado' <ul style="list-style-type: none">- Toda condición de salto no resuelta se predice que no da lugar a un salto- Se continúa la ejecución por donde iba aunque se puede adelantar algo el procesamiento de la secuencia de salto (cálculo de la dirección de salto, BTA)- Cuando se evalúa la condición se comprueba si la predicción era buena.- Si la predicción era buena el procesamiento continúa y se borra la BTA, y si era mala se abandona el procesamiento de la secuencia predicha (no se considera su efecto) y se captan instrucciones a partir de la BTA <p>* Es más fácil de implementar que la aproximación de 'Siempre Tomado'</p>	<p>SuperSparc (1992) (TP: 1; NTP: 0)</p> <p>Alpha21064 Power I (1990) (TP: 3; NTP: 0)</p> <p>Power 2 (1993) (TP: 1; NTP: 0)</p>
Aproximación 'Siempre Tomado' <ul style="list-style-type: none">- Toda condición de salto no resuelta se predice que da lugar a un salto- En previsión de error de predicción se salva el estado de procesamiento actual (PC) y se empieza la ejecución a partir de la dirección de salto.- Cuando se evalúa la condición de salto se comprueba si la predicción era buena- Si la predicción es correcta se continúa, y si es errónea se recupera el estado almacenado y no se considera el procesamiento de la secuencia errónea <p>* Necesita una implementación más compleja que la aproximación anterior aunque suele proporcionar mejores prestaciones</p>	<p>MC68040 (1999) (TP: 1; NTP: 2)</p>

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción estática

Predicción basada en el Código de Operación

Para ciertos códigos de operación (ciertos saltos condicionales específicos) se predice que el salto se toma, y para otros que el salto no se toma

MC88110 (93)

PowerPC 603(93)

Ejemplo: Predicción Estática en el MC88110

Formato	Instrucción		Predicción
	Condición Especificada	Bit 21 de la Instr.	
bcnd (<i>Branch Conditional</i>)	$\neq 0$	1	Tomado
	$= 0$	0	No Tomado
	> 0	1	Tomado
	< 0	0	No Tomado
	≥ 0	1	Tomado
	≤ 0	0	No Tomado
	bb1 (<i>Branch on Bit Set</i>)		Tomado
	bb0 (<i>Branch on Bit Clear</i>)		No Tomado

Unidad 2. Superescalares

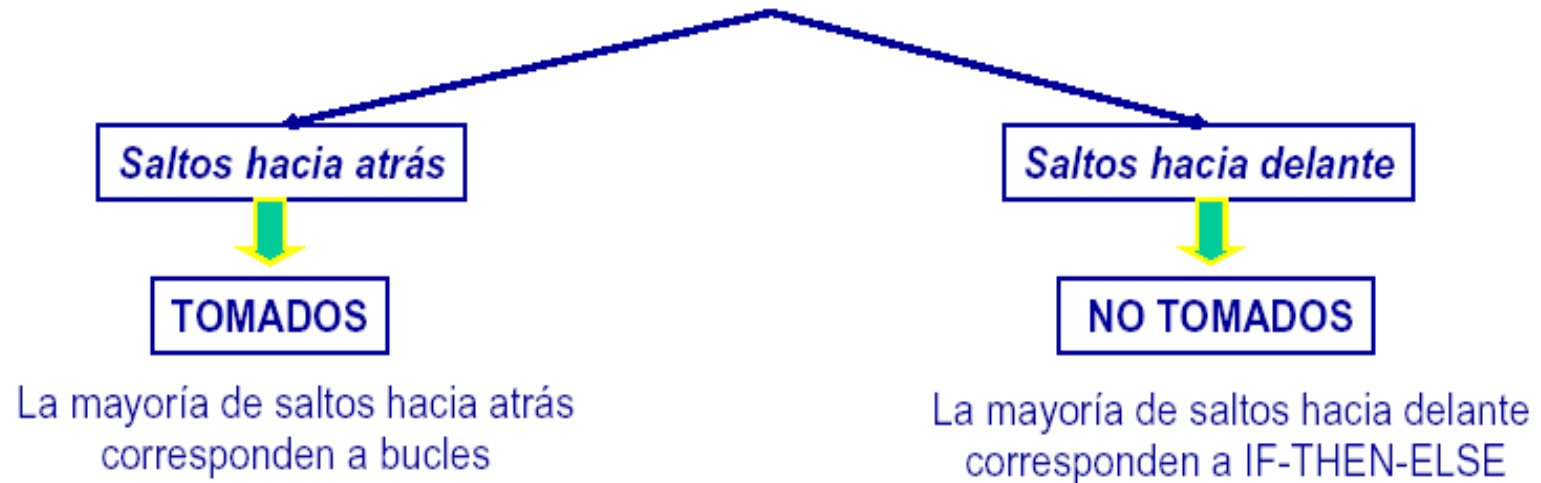
2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción estática

Predicción basada en la DIRECCIÓN del salto



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

EJEMPLOS

- Alpha 21064 (1992) (Opción seleccionable)
- PowerPC 601/603 (1993)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción estática



- Se añade un **Bit de Predicción** al opcode de la instrucción
- El compilador activa o desactiva este bit para indicar su predicción

EJEMPLOS

- AT&T 9210 Hobbit (1993)
- PowerPC 601/603 (1993)
- PA 8000 (1996)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica

- La predicción para cada instrucción de salto puede cambiar cada vez que se va a ejecutar ésta según la historia previa de saltos tomados/no-tomados para dicha instrucción.
- El presupuesto básico de la predicción dinámica es que es más probable que el resultado de una instrucción de salto sea similar al que se tuvo en la última (o en las n últimas ejecuciones)
- Presenta mejores prestaciones de predicción, aunque su implementación es más costosa

- **Predicción Dinámica Implícita**

No hay bits de historia propiamente dichos sino que se almacena la dirección de la instrucción que se ejecutó después de la instrucción de salto en cuestión

- **Predicción Dinámica Explícita**

Para cada instrucción de salto existen unos bits específicos que codifican la información de historia de dicha instrucción de salto

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

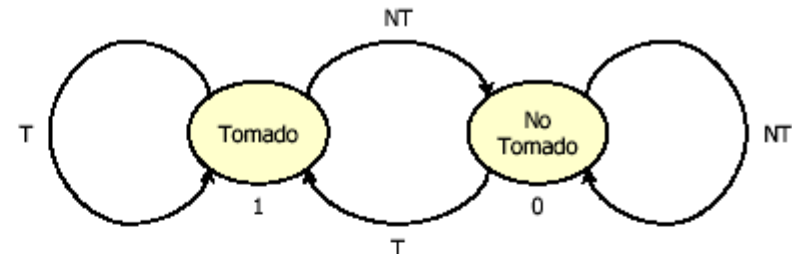
Riesgos

Gestión

- Predicción dinámica explícita

Predicción con 1 bit de historia

La designación del estado, Tomado (1) o No Tomado (0), indica lo que se predice, y las flechas indican las transiciones de estado según lo que se produce al ejecutarse la instrucción (T o NT)

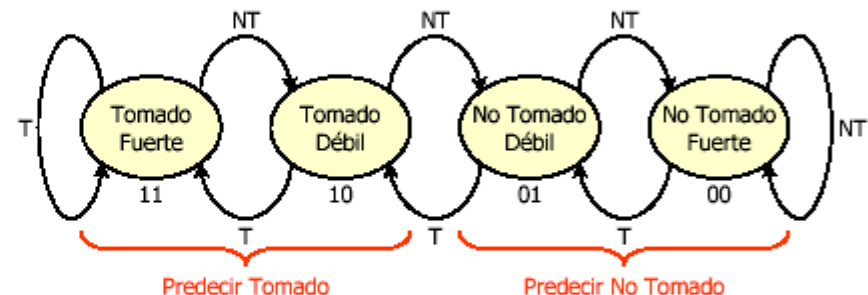


Predicción con 2 bits de historia

Existen cuatro posibles estados. Dos para predecir Tomado y otros dos para No Tomado

La primera vez que se ejecuta un salto se inicializa el estado con predicción estática, por ejemplo 11

Las flechas indican las transiciones de estado según lo que se produce al ejecutarse la instrucción (T o NT)

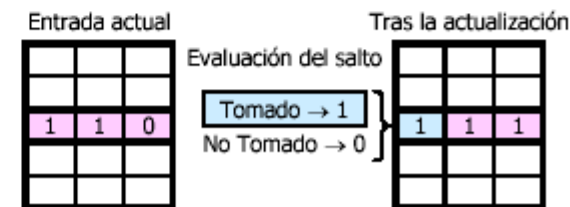


Predicción con 3 bits de historia

Cada entrada guarda las últimas ejecuciones del salto

Se predice según el bit mayoritario (por ejemplo, si hay mayoría de unos en una entrada se predice salto)

La actualización se realiza en modo FIFO, los bits se desplazan, introduciéndose un 0 o un 1 según el resultado final de la instrucción de salto



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita

Implementación de los Bits de Historia

<ul style="list-style-type: none">• En la Cache de Instrucciones	Alpha 21064 (92) (2k x 1 bit) Alpha 21064A (94) (4k x 2 bits) UltraSparc (92) (2k x 2 bits)
<ul style="list-style-type: none">• En una Tabla de Historia de Salto (BHT)	PA8000(96) (253 x 3bits) PowerPC 620(95) (2K x 2bits) R10000 (96) (512 x 2bits)
<ul style="list-style-type: none">• En una Cache para las Instrucciones a las que se produce el Salto (BTIC) o,• En una Cache para las Direcciones a las que se produce el Salto (BTAC)	Pentium (94) (256 x 2 bits) (BTAC) MC 68069 (93) (256 x 2bits) (BTAC)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita

1) Branch Target Buffer (BTB): bits acoplados

La BTB almacena

- La dirección destino de los últimos saltos tomados
- Los bits de predicción de ese salto

Actualización de la BTB

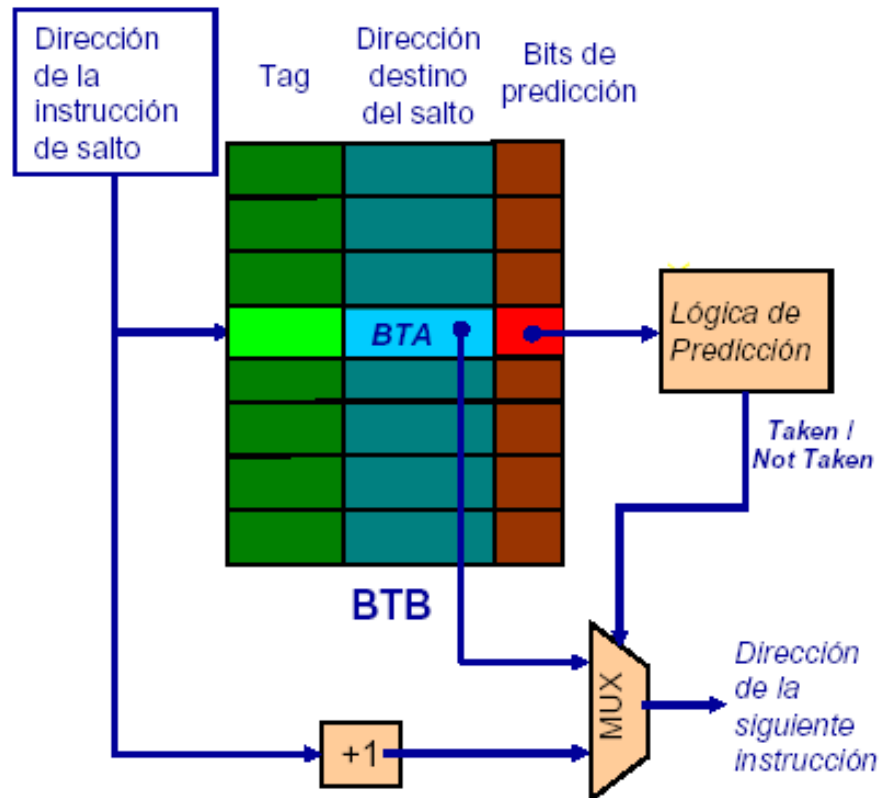
Los campos de la BTB se actualizan después de ejecutar el salto, cuando se conoce:

- Si el salto fue tomado o no \Rightarrow Actualizar bits de predicción
- La dirección destino del salto \Rightarrow Actualizar BTA

Predicción Implícita (sin bits de predicción)

Aplicable con un sólo bit de predicción

- Si la instrucción de salto está en la BTB
 \Rightarrow El salto se predice como tomado
- Si la instrucción de salto no está en la BTB
 \Rightarrow El salto se predice como no tomado



DESVENTAJA: Sólo se pueden predecir aquellas instrucciones de salto que están en la BTB

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita
 - 2) Tabla de historia de saltos (BHT): bits desacoplados

Existen dos tablas distintas:

- La BTAC, que almacena la dirección destino de los últimos saltos tomados
- La BHT, que almacena los bits de predicción de todas las instrucciones de salto condicional

Ventaja

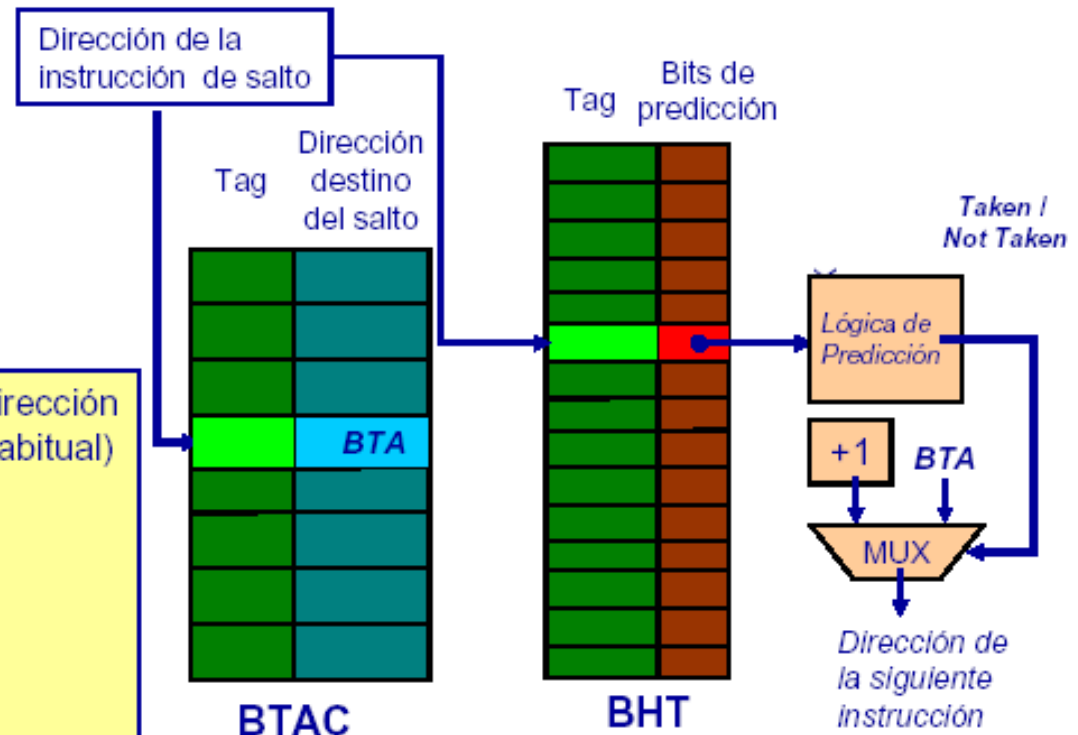
Puede predecir instruc. que no están en la BTAC (más entradas en BHT que en BTAC)

Desventaja

Aumenta el hardware necesario
⇒ 2 tablas asociativas

Acceso a la BHT

- 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



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita

3) Bits de predicción en la I-cache

Funcionamiento

Cuando se capta la instrucción de la cache
Si se trata de una instrucción de salto condicional

- Se accede en paralelo a los bits de predicción
- Si el salto se predice como tomado se accede a la instrucción destino del salto

Acceso a la instrucción destino del salto

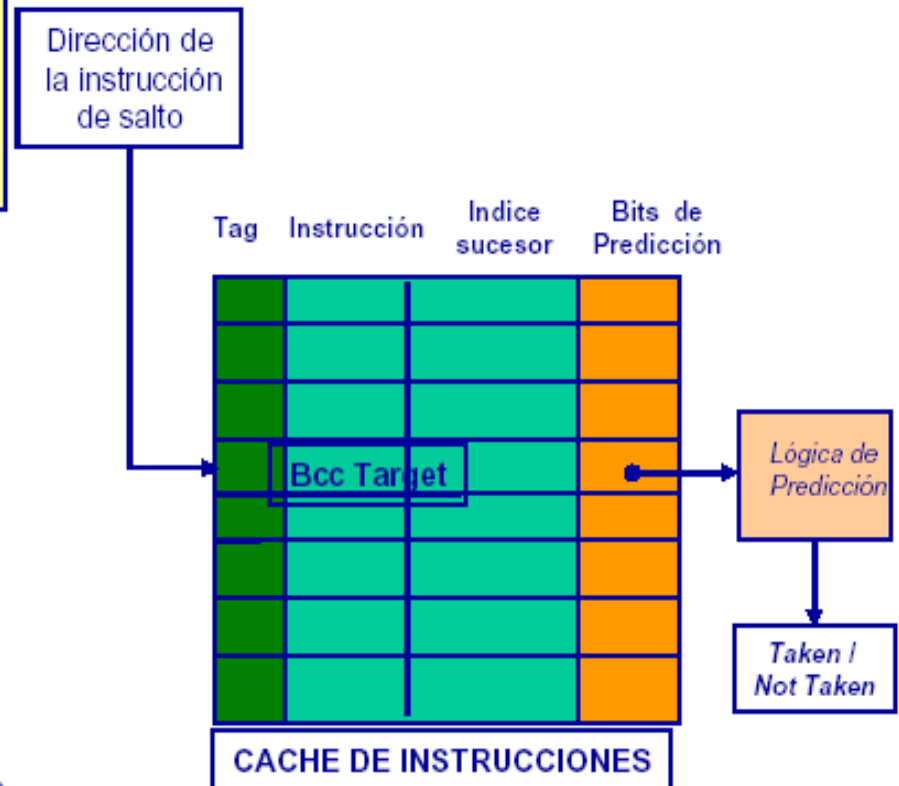
- BTB independiente
- Añadir *índice sucesor* a la I-cache

Alternativas de diseño

- Bits de predicción por cada instrucción de la cache
- Bits de predicción por cada línea de cache

Ventajas

- Puede predecir instrucciones que no están en la BTB
- No añade una cantidad extra de hardware excesiva



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Extensión del Procesamiento Especulativo

- Tras la predicción, **el procesador continúa ejecutando instrucciones especulativamente hasta que se resuelve la condición.**
- El intervalo de **tiempo entre el comienzo de la ejecución especulativa y la resolución** de la condición **puede variar considerablemente y ser bastante largo.**
- En los procesadores superescalares, que pueden emitir varias instrucciones por ciclo, **pueden aparecer más instrucciones de salto condicional no resueltas durante la ejecución especulativa.**
- Si el número de instrucciones que se ejecutan especulativamente es muy elevado y la predicción es incorrecta, la penalización es mayor.

Así, **cuanto mejor es el esquema de predicción mayor puede ser el número de instrucciones ejecutadas especulativamente.**



- **Nivel de Especulación:** Número de Instrucciones de Salto Condicional sucesivas que pueden ejecutarse especulativamente (si se permiten varias, hay que guardar varios estados de ejecución). **Ejemplos: Alpha21064, PowerPC 603 (1); Power 2 (2); PowerPC 620 (4); Alpha 21164 (6)**
- **Grado de Especulación:** Hasta qué etapa se ejecutan las instrucciones que siguen en un camino especulativo después de un salto. Ejemplos: Power 1 (Captación); PowerPC 601 (Captación, Decodificación, Envío); PowerPC 603 (Todas menos la finalización)

Unidad 2. Superescalares

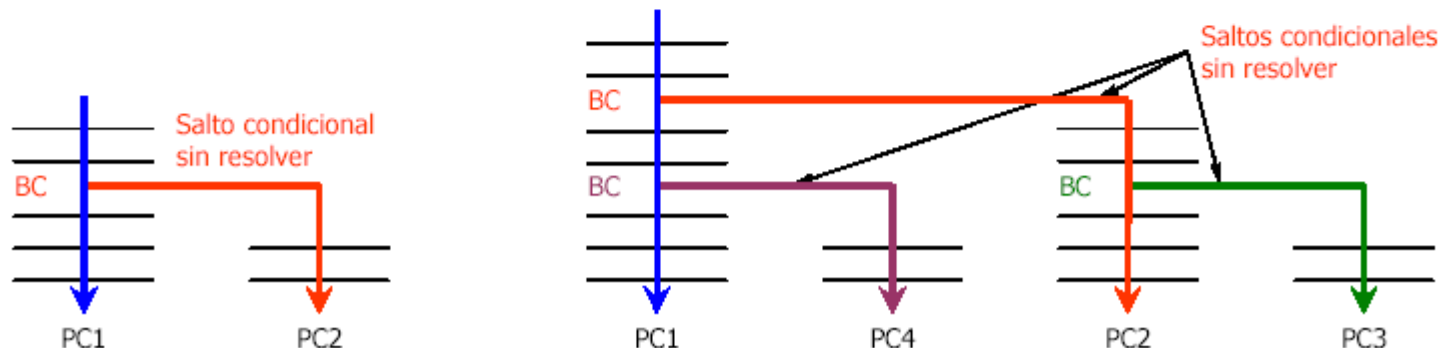
2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Ramificación multicamino

- Se **siguen las dos secuencias de instrucciones que aparecen a partir de una instrucción de salto** (la correspondiente al salto efectuado y al salto no efectuado). Una vez resuelto el salto la secuencia incorrecta se abandona
- Para implementar esta técnica **hacen falta varios contadores de programa**
- Se necesitan **gran cantidad de recursos hardware** y el proceso para descartar las instrucciones que se han ejecutado incorrectamente es complejo



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Ejecución condicional

Instrucciones de Ejecución Condicional (*Guarded Execution*)

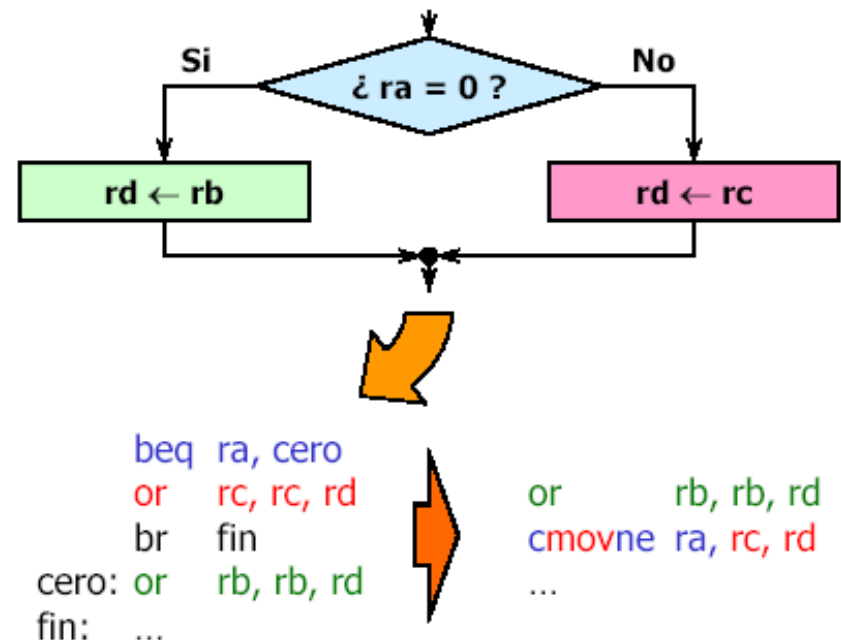
- Se pretende **reducir el número de instrucciones de salto** incluyendo en el **repertorio máquina instrucciones con operaciones condicionales** (*'conditional operate instructions'* o *'guarded instructions'*)
- Estas instrucciones tienen dos partes: la **condición** (denominada *guardia*) y la parte de **operación**

Ejemplo: **cmovxx** de Alpha

cmovxx ra.rq, rb.rq, rc.wq

- xx** es una condición
- ra.rq, rb.rq** enteros de 64 bits en registros ra y rb
- rc.wq** entero de 64 bits en rc para escritura
- El registro ra se comprueba en relación a la condición xx y si se verifica la condición rb se copia en rc.

Sparc V9, HP PA, y Pentium ofrecen también estas instrucciones.



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

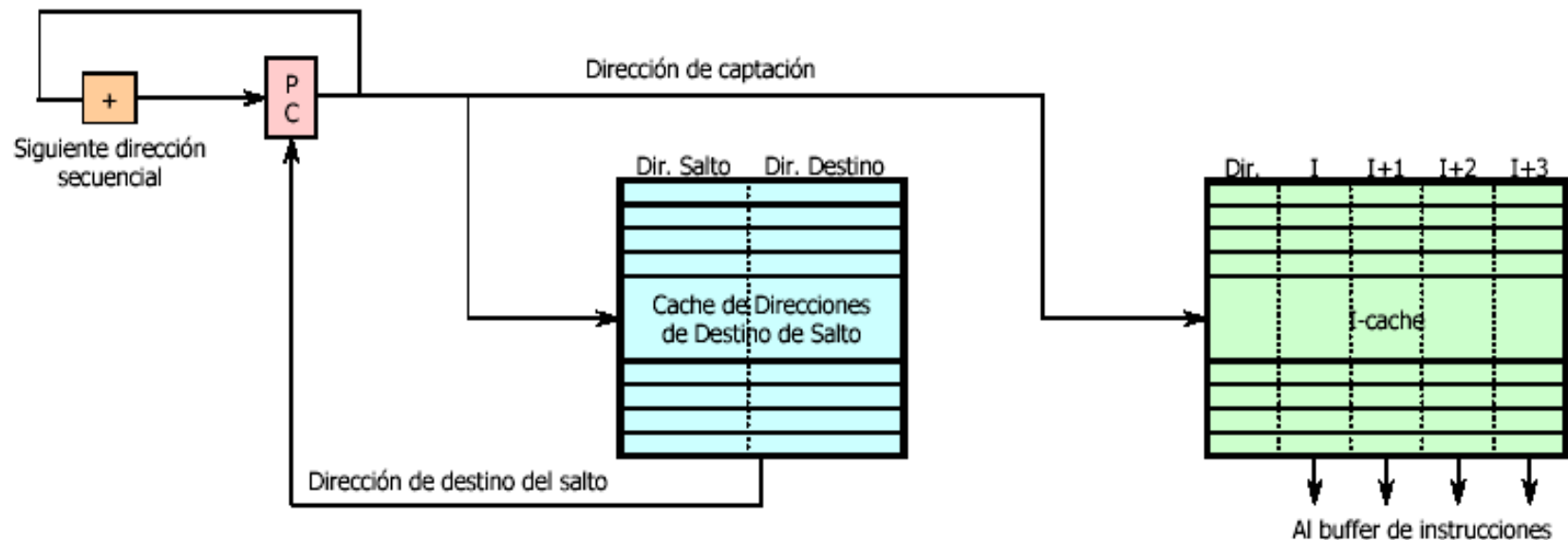
Riesgos

Gestión

Estructuras

Esquema de cache de direcciones de destino de salto (BTAC)

- Se añade una cache que contiene las direcciones de las instrucciones destino de los saltos, junto con las direcciones de las instrucciones de salto.
- Se leen las direcciones al mismo tiempo que se captan las instrucciones de salto.



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

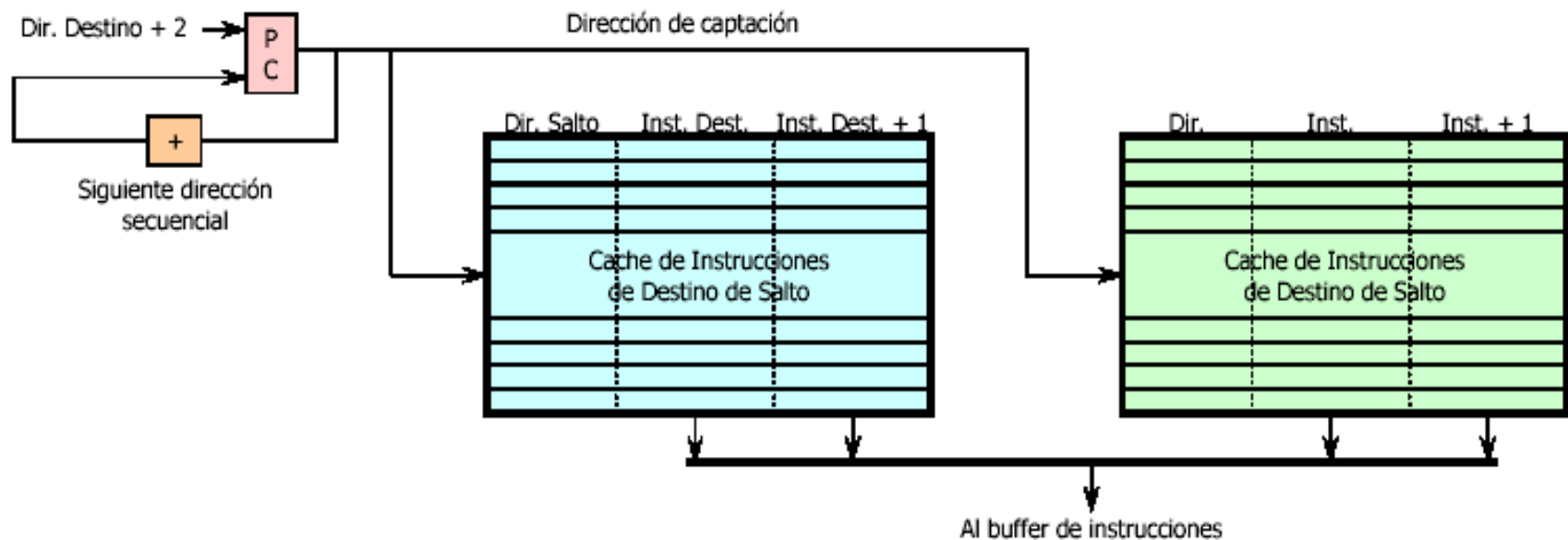
Riesgos

Gestión

Estructuras

Esquema de cache de instrucciones de destino de salto (BTIC)

- Se añade una cache que contiene las instrucciones siguientes a la dirección de destino de los saltos, junto con las direcciones de las instrucciones de salto.
- Sólo tiene sentido si la cache de instrucciones tiene una latencia muy alta.
- Mientras se procesan estas instrucciones se calcula la dirección de las siguientes.



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

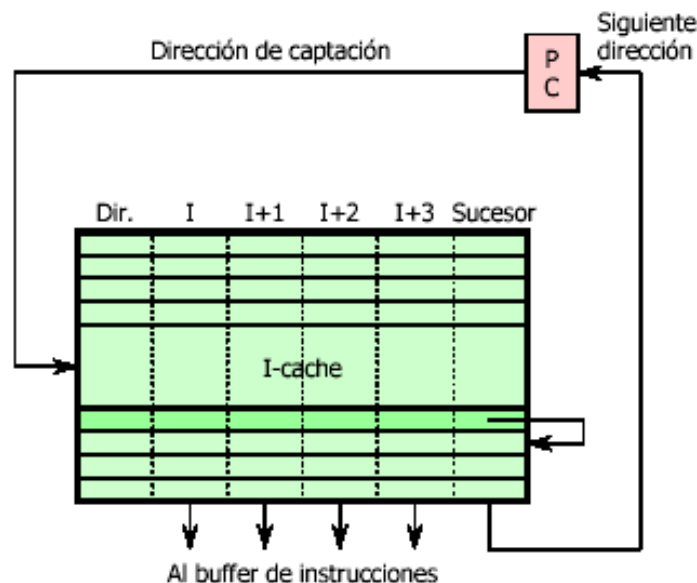
Riesgos

Gestión

Estructuras

Esquema de índice sucesor en la cache de instrucciones

- La cache de instrucciones contiene un índice sucesor que apunta a la siguiente línea de la cache de instrucciones que hay que captar (la siguiente, o la que se predice que se debe captar si hay una instrucción de salto condicional en esa línea).



Ejemplos y evolución de los esquemas de Acceso

Calcular/captar	BTIC	BTAC	Índice sucesor
i486 (1989)	→	Pentium (1993)	
MC68040 (1990)	→	MC 68060 (1993)	
	Am 29000 (1988)	→	Am 29000 superscalar (1995)
Sparc CYC 600 (1992) SuperSparc (1992)	→	→	UltraSparc (1995)
R4000 (1992) R10000 (19996)	→	→	R8000 (1994)
PowerPC 601 (1993) PowerPC 603 (1993)	→	PowerPC 604 (1995) PowerPC 620 (1996)	