2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores

Tema 4. Arquitecturas con Paralelismo a nivel de Instrucción (ILP)

Material elaborado por los profesores responsables de la asignatura:
Julio Ortega – Mancia Anguita

ugr | Universidad de Granada

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

ATC
Departamento de Arquitectura y Tecnología de Computadores
UNIVERSIDAD DE GRANADA

Lecciones

- Lección 11. Microarquitecturas ILP. Cauces Superescalares
- Lección 12. Consistencia del procesador y Procesamiento de Saltos
- Lección 13. Procesamiento VLIW

2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores
Tema 4

Lección 11. Microarquitecturas ILP. Cauces Superescalares

Material elaborado por los profesores responsables de la asignatura:
Julio Ortega – Mancia Anguita

ugr | Universidad de Granada

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

ATC
Departamento de Arquitectura y Tecnología de Computadores
UNIVERSIDAD DE GRANADA

Lecciones

- Lección 11. Microarquitecturas ILP. Cauces Superescalares
  - Introducción: Motivación y Nota Histórica
  - Paralelismo entre instrucciones (ILP). Orden en Emisión y Finalización
  - Cauces superescalares
- Lección 12. Consistencia del procesador y Procesamiento de Saltos
- Lección 13. Procesamiento VLIW

Bibliografía

- Fundamental
  - Capítulo 3. Secc. 3.1, 3.3.1, 3.3.2, 3.3.3. J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*. Thomson, 2005. ESIIT/C.1 ORT arq
- Complementaria
  - Sima and T. Fountain, and P. Kacsuk. *Advanced Computer Architectures: A Design Space Approach*. Addison Wesley, 1997. ESIIT/C.1 SIM adv

Contenido de la Lección 11

- Introducción: Motivación y Nota Histórica
- Paralelismo entre instrucciones (ILP). Orden en Emisión y Finalización
- Cauces superescalares

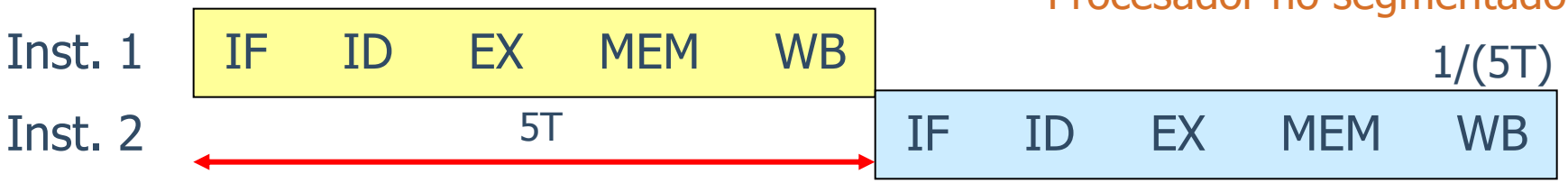
Arquitecturas con DLP, ILP y TLP (thread=flujo de control)

Arq. con **DLP** (*Data Level Parallelism*)	Arq. con **ILP** (*Instruction Level Parallelism*)	Arq. con **TLP** (*Thread Level Parallelism*) explícito y **una** instancia de SO		Arq. con **TLP** explícito y **múltiples** instancias SO
Tema 5	Tema 4	Temas 3, 5		IC.SCAP
Ejecutan las operaciones de una instrucción **concurr.** o en **paralelo**	Ejecutan múltiples instrucciones **concurr.** o en **paralelo**	Ejecutan múltiples flujos de control **concurr.** o en **paralelo**		Ejec. múltiples flujos de control en **paralelo**
Unidades funcionales vectoriales o SIMD	**Cores** escalares segmentados, superescalares o VLIW/EPIC	**Cores** que modifican la arquit. escalar segmentada, superescalar o VLIW/EPIC para ejecutar threads concurr. o en paralelo	**Multi-procesadores**: ejecutan threads en paralelo en un computador con múltiples cores (incluye **multicores**)	**Multi-computadores**: ejecutan threads en paralelo en un sistema con múltiples computadores

Paralelismo entre Instrucciones

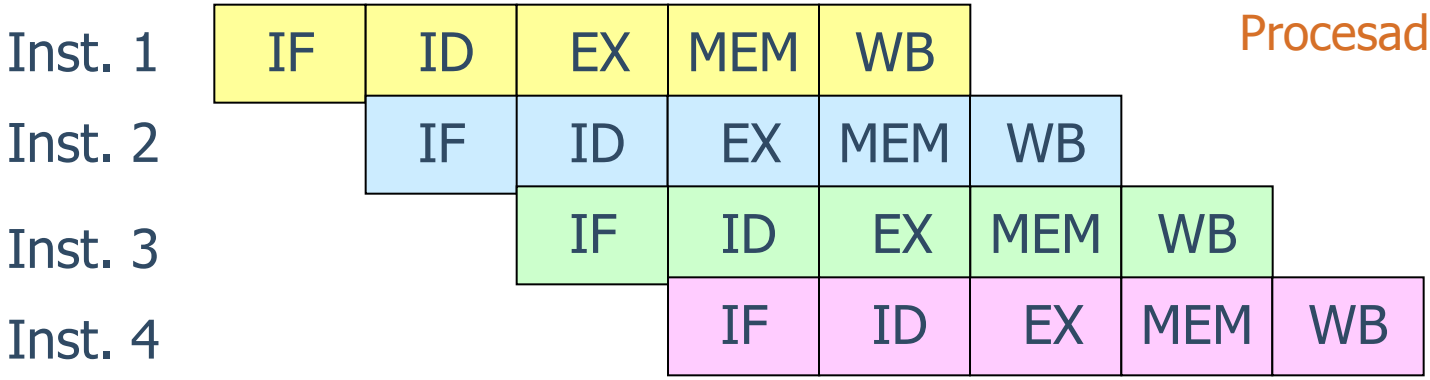
Procesador no segmentado

1/(5T)



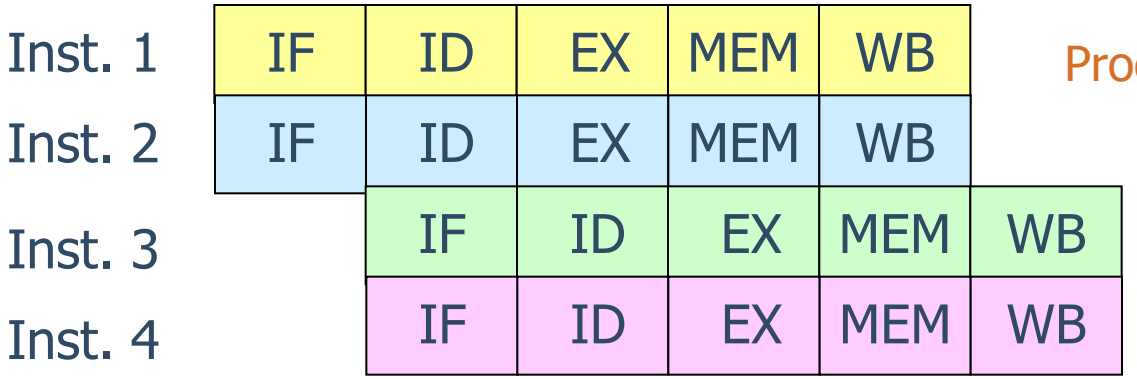
Procesador Segmentado

1/(T)



Procesador Superescalar o VLIW

2/(T)



Procesadores Superescalares y VLIW

- Los procesadores Superescalares y VLIW comparten las siguientes características (además del hecho de ser procesadores segmentados):
  - Disponen de varias unidades de ejecución
  - Pueden ejecutar varias instrucciones simultáneamente en esas unidades de ejecución
  - Pueden emitir múltiples instrucciones en paralelo a unidades de ejecución
- Pero:

En los procesadores superescalares es el hardware el que debe descubrir el paralelismo que se puede aprovechar a partir de las instrucciones que se van captando, **en los procesadores VLIW el paralelismo es explícito** (se captan juntas las instrucciones que se van a emitir juntas a unidades de ejecución)

La micro-arquitectura de los procesadores VLIW es más sencilla (sobre todo en lo que respecta a las etapas de emisión y finalización: buffers de renombrado, ROBs,...) ya que es el compilador el que debe detectar el paralelismo al seleccionar las instrucciones que se captarán juntas en la misma palabra de inst.

Mejora de las Prestaciones de los Procesadores

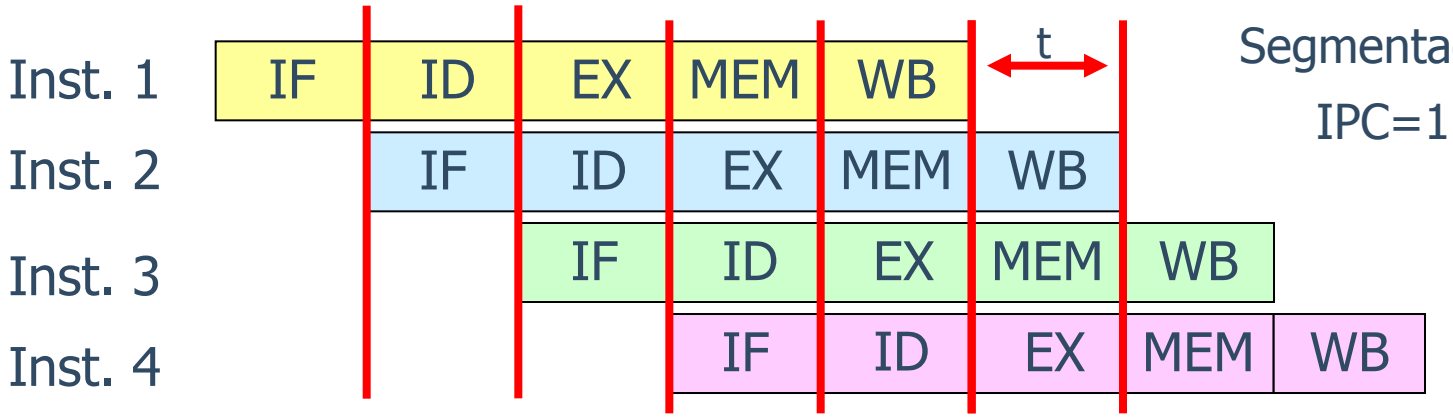
Más transistores por circuito integrado → **Microarquitecturas más complejas en un solo CI: Paralelismo entre Instrucciones (Procesadores Superescalares)**

Mejora de la Tecnología de Fabricación de CI basada en el Silicio → **Reducción del tamaño de los transistores + Aumento del tamaño del dado**

$$V_{CPU} = IPC \times F$$

Se reduce la longitud de puerta del transistor y con ello el tiempo de conmutación → **Mayores frecuencias de funcionamiento**

Evolución de Microarquitecturas I



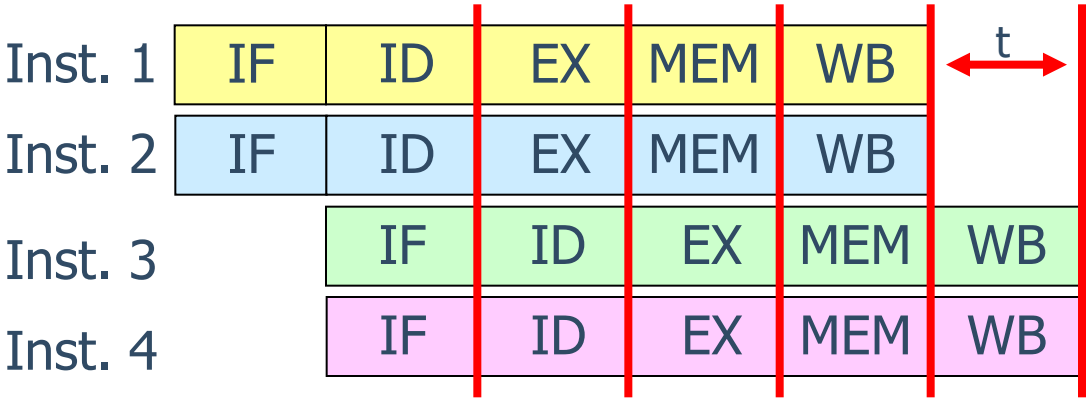
Segmentado

IPC=1 F=1/t

x 4

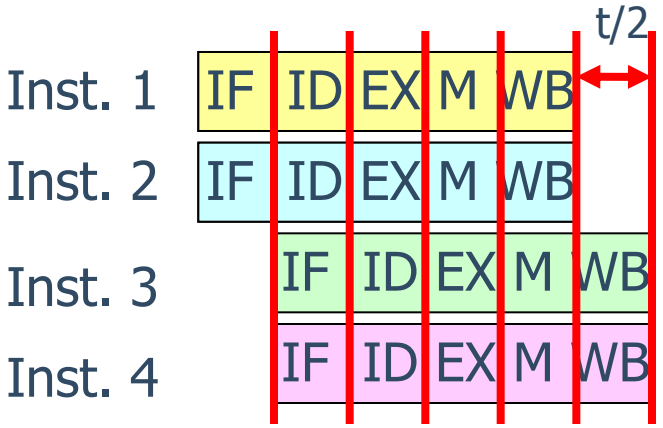
Aumento de IPC

Superescalar o VLIW IPC=2 F=1/t

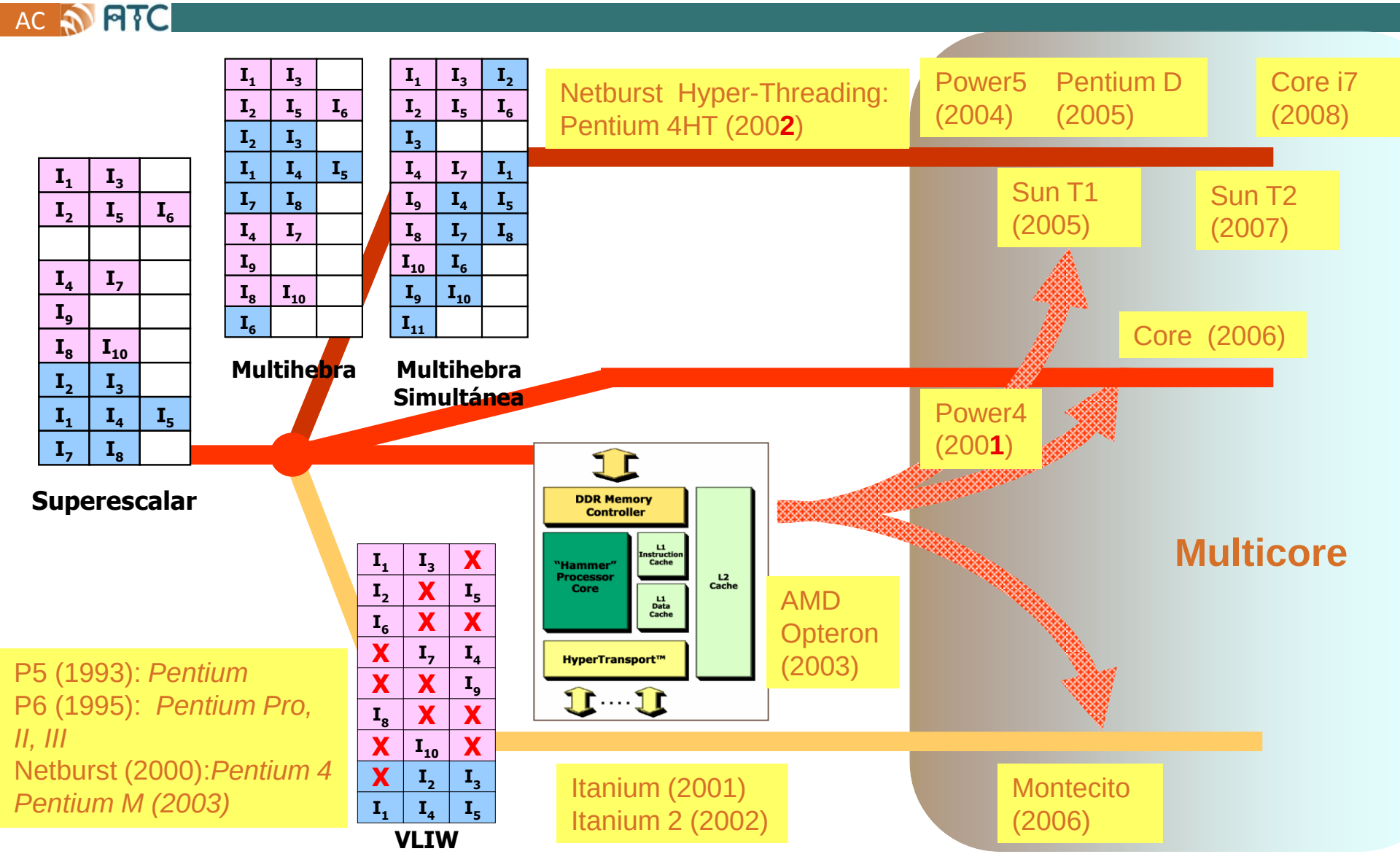


Aumento de IPC y F

IPC=2 F=2/t



Evolución de Microarquitecturas II



Nota histórica. DLP y ILP (*Instruction Level Parallelism*)

- DLP (Data Level Parallelism)
  - Unidades funcionales (o de ejecución) SIMD (o multimedia)
    - **1989** (Intel i860). **1991** (motorola M88110). **1993** (repertorio MAX en HP PA : PA7100LC). **1995** (repertorio VIS en Sun Sparc: Ultra I). **1997** (repertorio MMX en Intel x86: Pentium MMX). **1999** (repertorio SSE en Intel x86: Pentium III; repertorio Altivec en IBM Power: PowerPC 8000)
- ILP (*Instruction Level Parallelism*) :
  - Procesadores/cores segmentados
    - ***1961** (IBM 7030)*. **1982** (chip Intel i286, Motorola 68020). **1986** (chip MIPS R2000). **1987** (chip AMD Am29000). **1988** (chip Sun Sparc)…
  - Procesadores con múltiples unidades funcionales
    - **1967** (IBM 360/91) …
  - Procesadores/cores superescalares
    - **1989** (chip Intel 960CA (3)). **1990**: (chip IBM Power1 (4)). **1992**: (chips DEC α21064 (2/4), HP PA 7100 (2/2), Sun SuperSparc (3/5)) …
  - Procesadores/cores VLIW
    - **1990** (chip DSP Intel i860 (2)).**1997** (chip DSP TMS320C6x (8)). **2001** (chip Intel Itanium)…

TEMA 4

NOTA: Destacado en cursiva las primeras implementaciones y en color más claro los chip de propósito específico

Contenido de la Lección 11

- Introducción: Motivación y Nota Histórica
- Paralelismo entre instrucciones (ILP). Orden en Emisión y Finalización
- Cauces superescalares

Paralelismo entre instrucciones (ILP) y paralelismo de la máquina

Paralelismo entre Instrucciones

Depende de la frecuencia de las dependencias de datos y control, y del retardo de la operación (tiempo hasta que el resultado de una operación esté disponible).

```
load r1, r2(23)
add r3,#1          } (Paralelismo: 3)
add r4,r5
add r3, #1
add r4,r3          } (Paralelismo: 1)
store (r4),r0
```

Paralelismo de la Máquina

Determinado por el número de instrucciones que pueden captarse y ejecutarse al mismo tiempo (número de cauces paralelos) y por la velocidad y los mecanismos que usa el procesador para encontrar las dependencias entre instrucciones

Ordenaciones en una secuencia de instrucciones

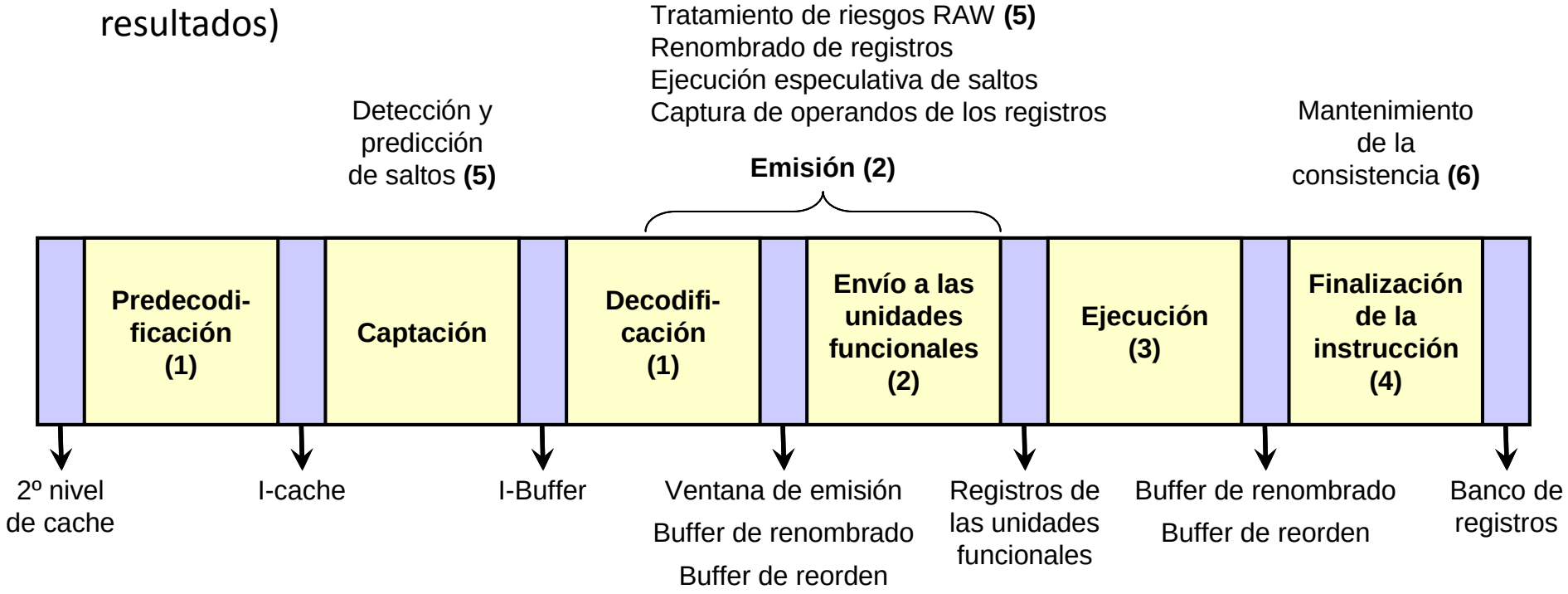
- En una secuencia de instrucciones se pueden distinguir tres tipos de ordenaciones:
  - El orden en que se captan las instrucciones (el orden de las instrucciones en el código)
  - El orden en que las instrucciones se ejecutan
  - El orden en que las instrucciones cambian los registros y la memoria.
- El procesador superescalar debe ser capaz de identificar el paralelismo entre instrucciones (ILP) que exista en el programa y organizar la captación, decodificación y ejecución de instrucciones en paralelo, utilizando eficazmente los recursos existentes (el paralelismo de la máquina).
- Cuanto más sofisticado sea un procesador superescalar, menos tiene que ajustarse a la ordenación de las instrucciones según se captan, para la ejecución y modificación de los registros, de cara a mejorar los tiempos de ejecución. La única restricción es que el resultado del programa sea correcto.

Contenido de la Lección 11

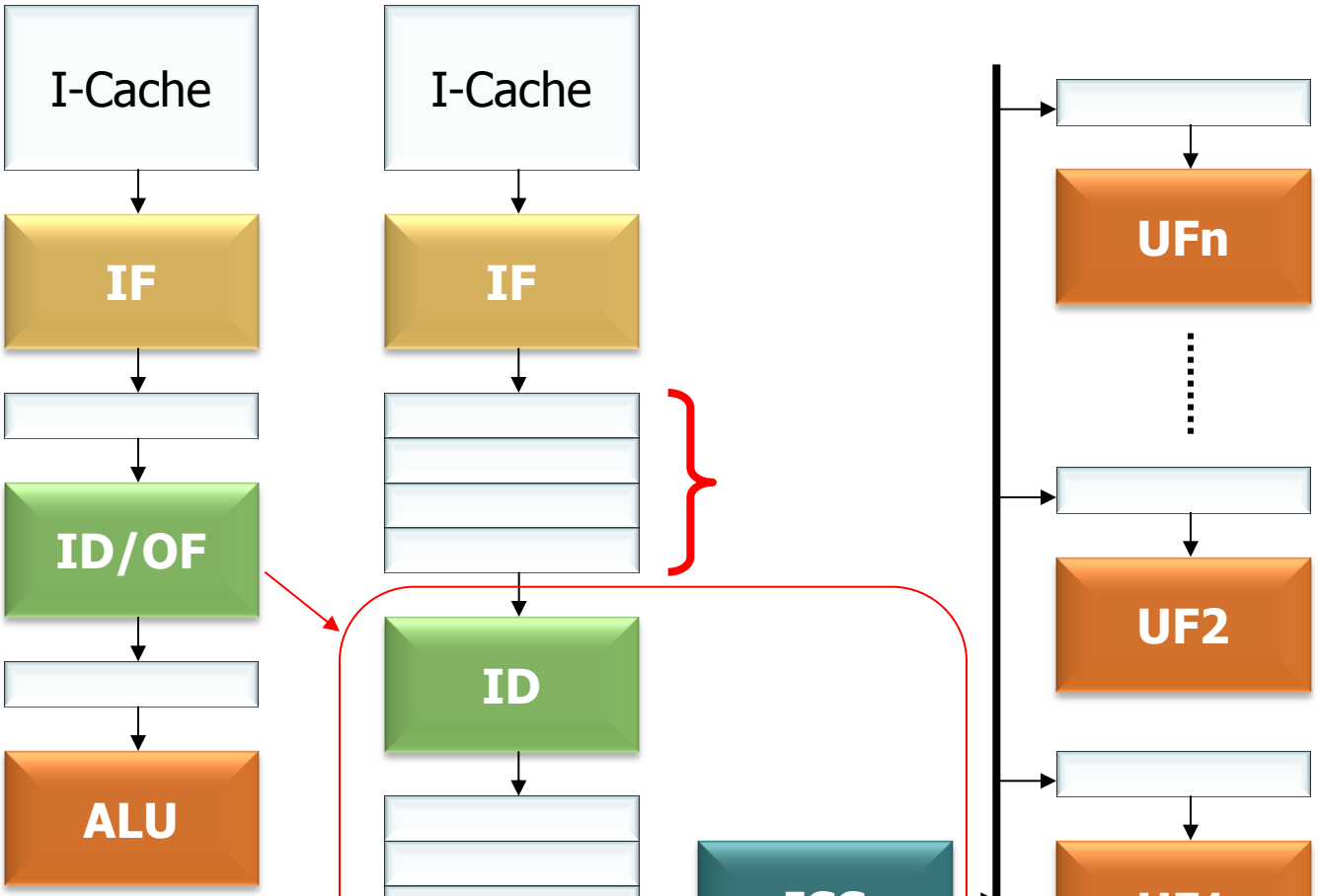
- Introducción: Motivación y Nota Histórica
- Paralelismo entre instrucciones (ILP). Orden en Emisión y Finalización
- Cauces superescalares
  - Decodificación Paralela y Predecodificación
  - Emisión Paralela de Instrucciones. Estaciones de Reserva
  - Renombramiento de registros

Aspectos del Procesamiento Superescalar y Etapas del Cauce

1. **Decodificación Paralela** (Decodificación a mayor velocidad → **Predecodificación**)
2. **Emisión Paralela** de Instrucciones a las Unidades Funcionales (**Dependencias**)
3. **Ejecución Paralela** en las distintas Unidades Funcionales, UF, (segmentadas).
4. **Finalización del Procesamiento** de la Instrucción
5. **Detección y predicción de saltos**
6. **Mantenimiento de la consistencia** secuencial (desacoplar la ejecución y la escritura de resultados)



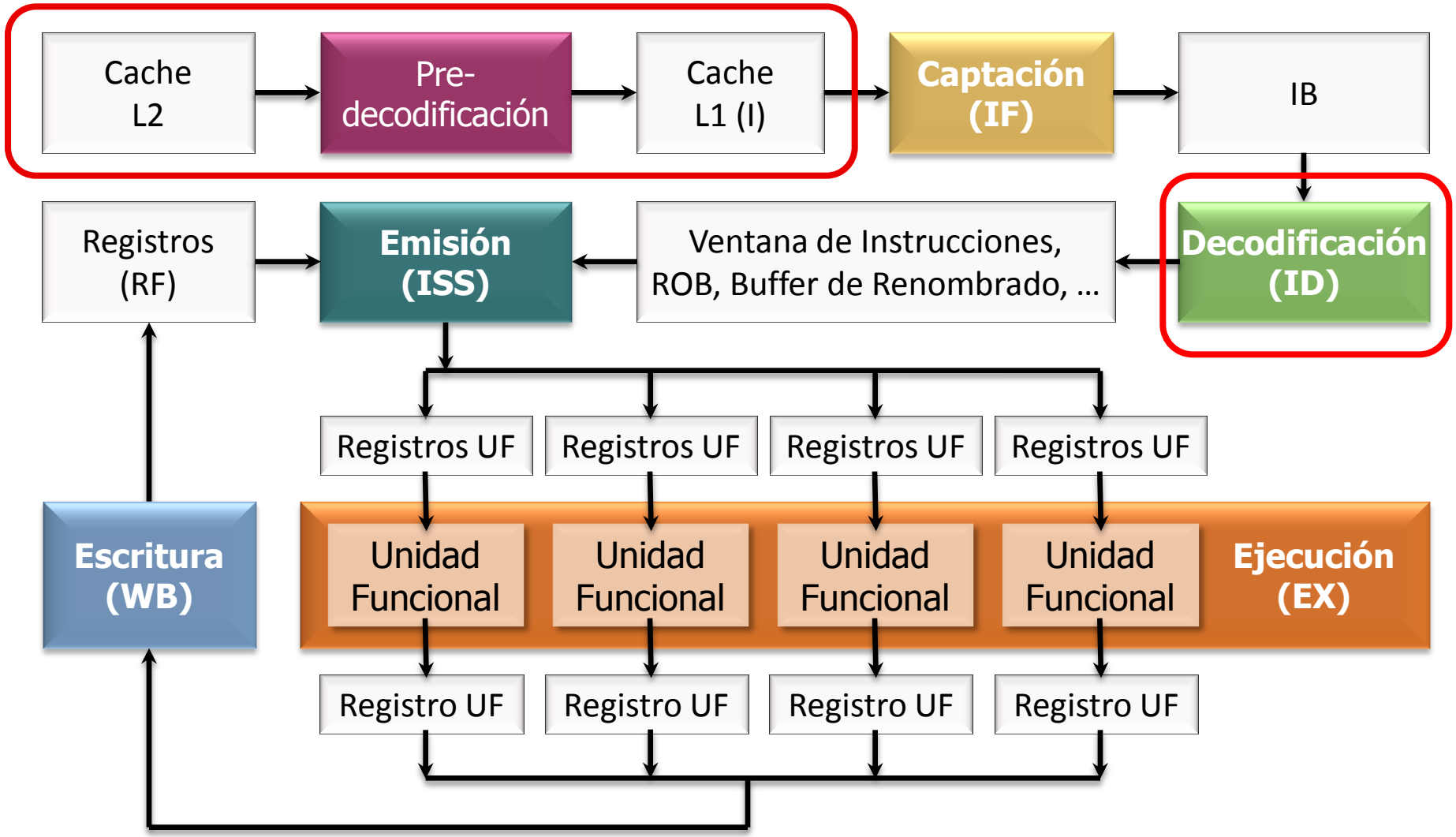
Etapas de un procesador Superescalar: Captación y Decodificación Paralela



Un procesador segmentado escalar sólo decodifica una instrucción por ciclo

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

Etapas de un Procesador Superescalar: Predecodificación I



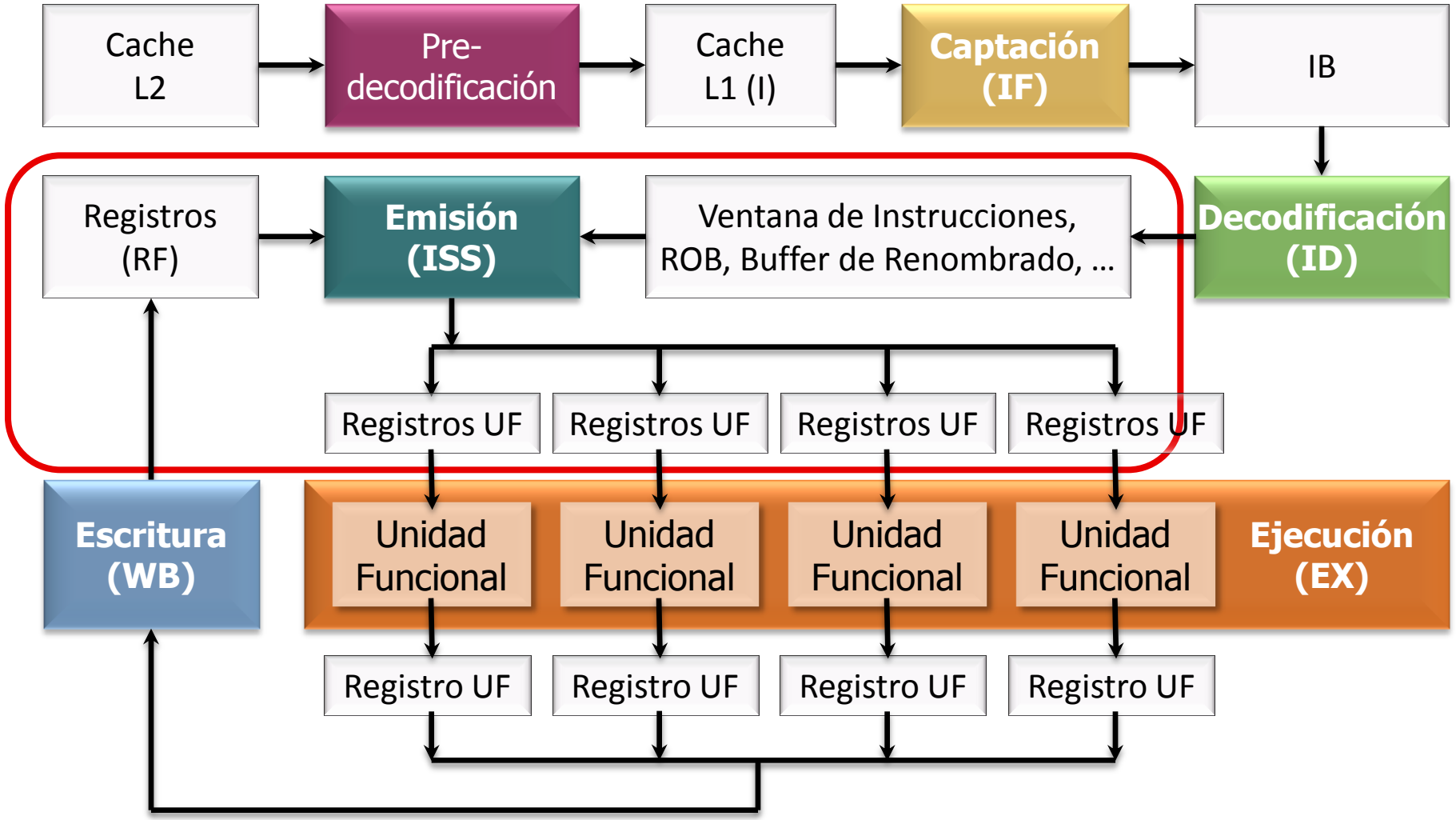
Predecodificación II

- Los bits que se añaden en la etapa de predecodificación (bits de predecodificación) suelen 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
- Ejemplos de uso de los bits de predecodificación:
  - HP-PA 7200: Se añaden bits por cada par de instrucciones para facilitar la comprobación de si las instrucciones se pueden ejecutar en paralelo en el cauce o hay algún tipo de dependencias (10% de SRAM).
  - AMD K6: La lógica de decodificación indica la longitud de la instrucción x86 en bytes permitiendo la localización del final de cada instrucción y se almacena en la cache L1 junto con las instrucciones. En una cache de 64 Kbytes se añaden 20 Kbytes para los bits de decodificación.

Contenido de la Lección 11

- Introducción: Motivación y Nota Histórica
- Paralelismo entre instrucciones (ILP). Orden en Emisión y Finalización
- Cauces superescalares
  - Decodificación Paralela y Predecodificación
  - Emisión Paralela de Instrucciones. Estaciones de Reserva
  - Renombramiento de registros

Etapas de un Procesador Superescalar: Emisión Paralela de Instrucciones

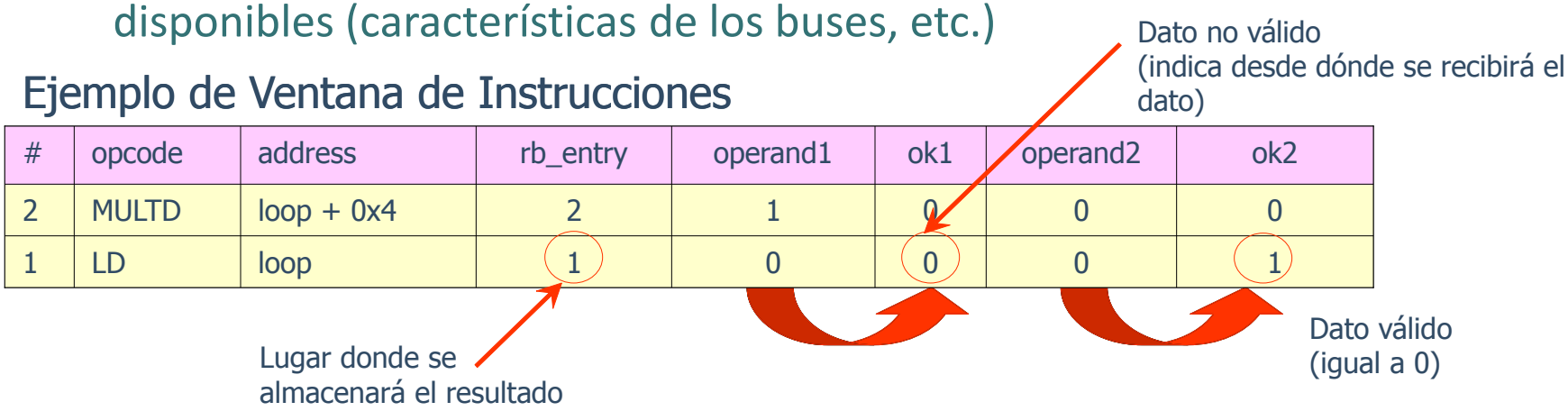


Ventana de Instrucciones

- 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
  - 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.)

Ejemplo de Ventana de Instrucciones

#	opcode	address	rb_entry	operand1	ok1	operand2	ok2
2	MULTD	loop + 0x4	2	1	0	0	0
1	LD	loop	1	0	0	0	1



Emisión Paralela de Instrucciones: Ordenada

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 Instrucciones

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)

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

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

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

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

Emisión Paralela de Instrucciones: Desordenada

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 Instrucciones

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	-	

Ejemplo de Emisión Paralela de Instrucciones Ordenada

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 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)

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

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	-

Ejemplo de Emisión Paralela de Instrucciones Desordenada

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 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)

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]

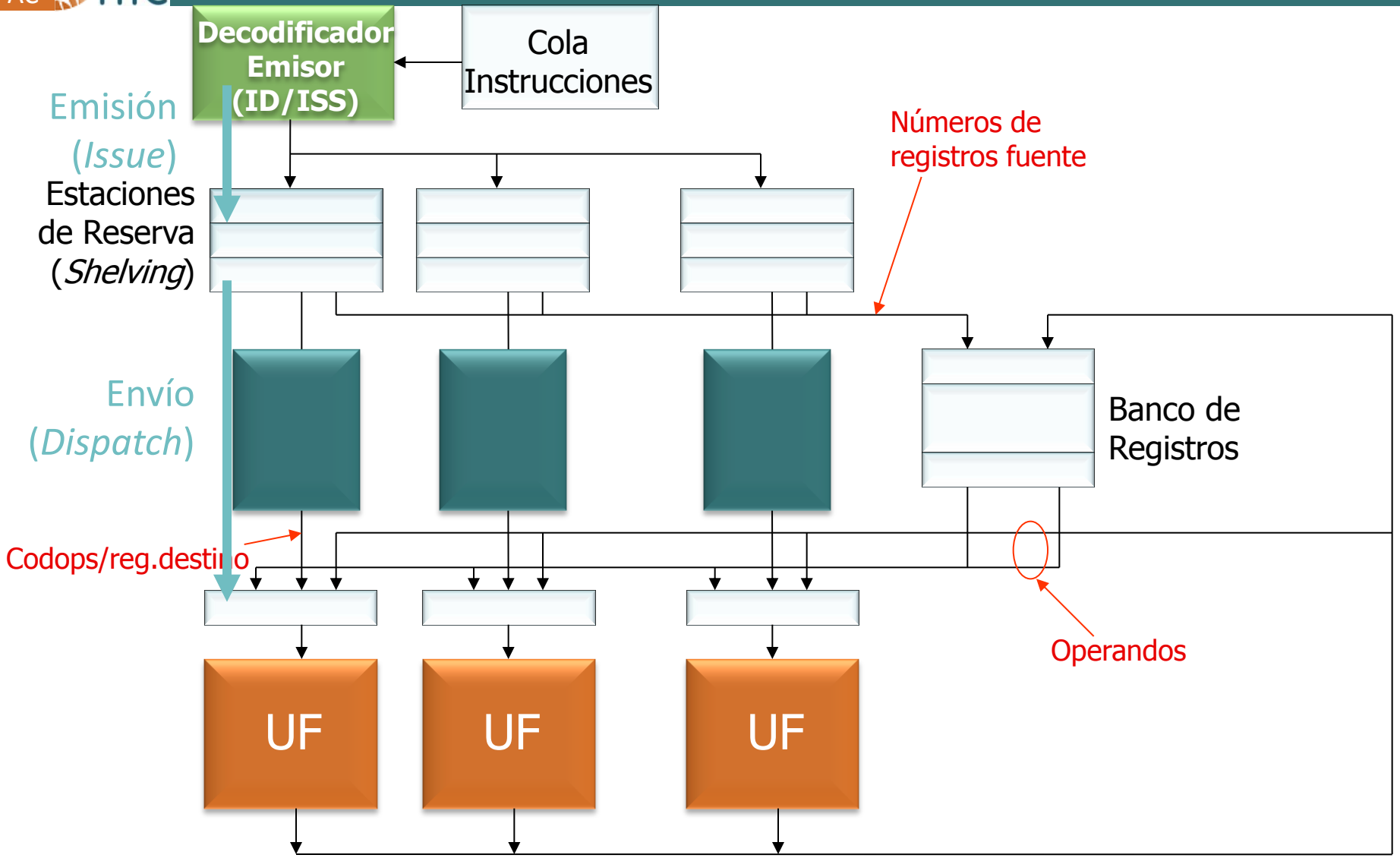
Se ha emitido [4] y ha terminado [2]: puede emitirse [3]

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

→

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

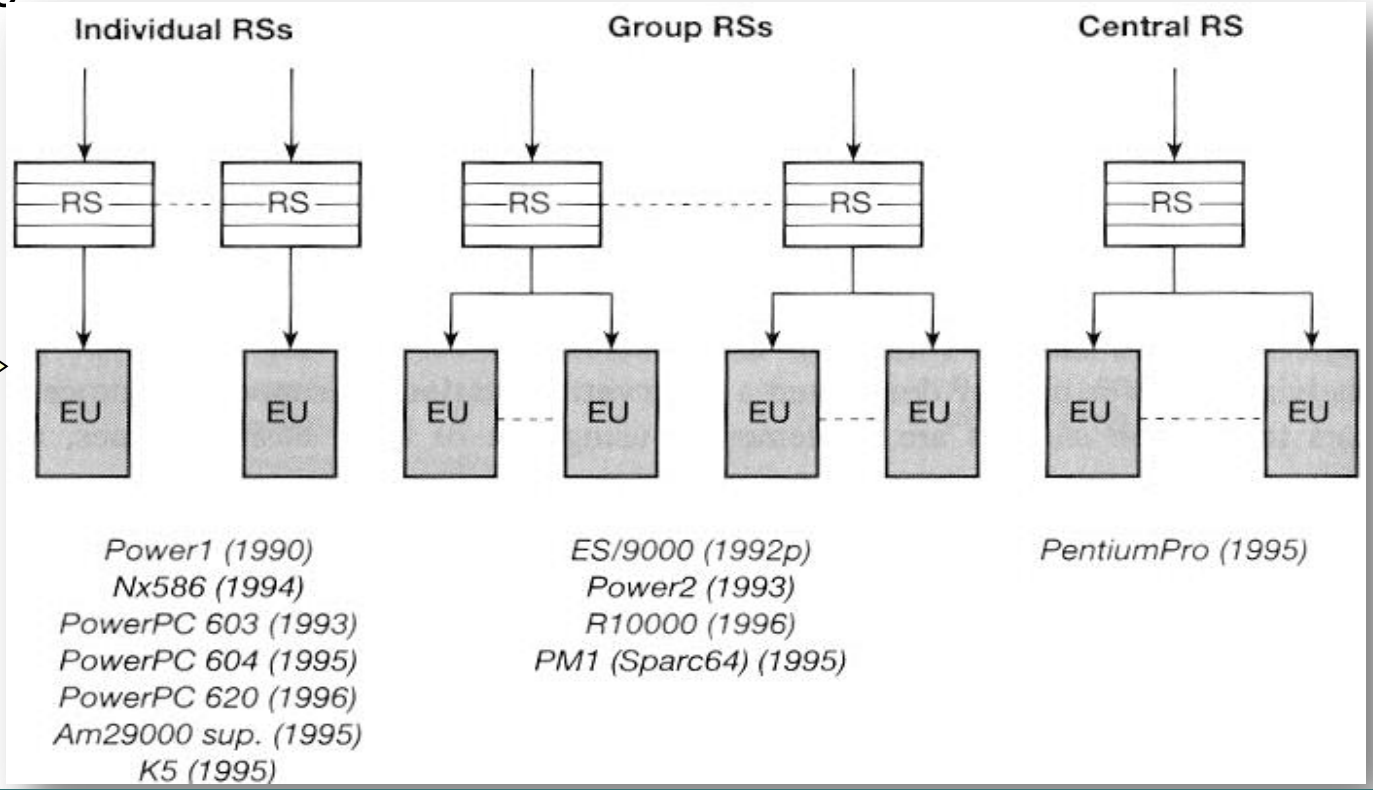
Estaciones de Reserva I (Ventana de Instrucciones Distribuida)



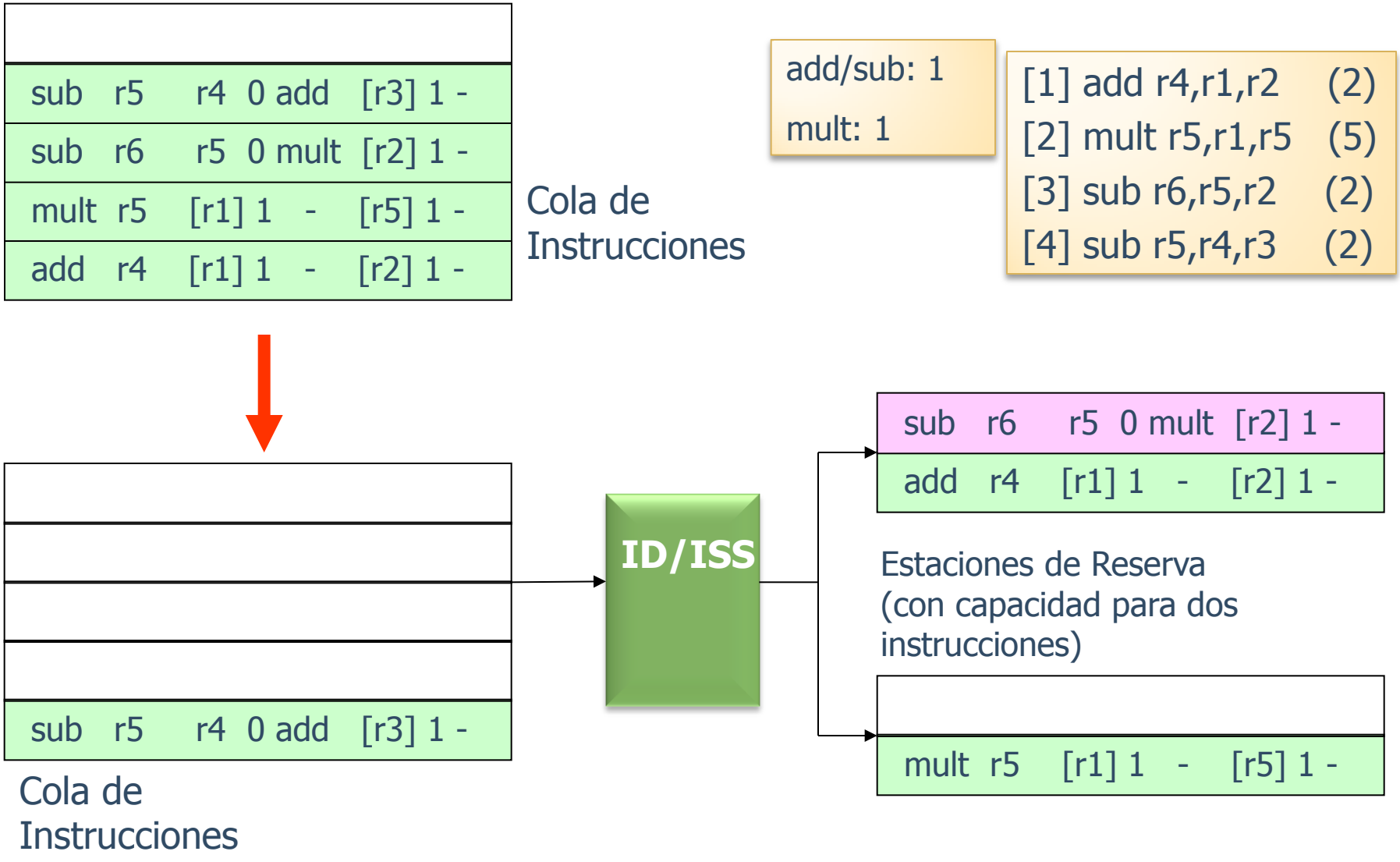
Estaciones de Reserva II

- Si no existen limitaciones en el hardware las instrucciones se emiten a las estaciones de reserva (buffers de *shelving* o consignas) independientemente de las dependencias
- Las instrucciones esperan hasta que se resuelvan las dependencias y se envían a las unidades funcionales una vez comprobada la disponibilidad de la unidad, cuando sea su turno

Tipos de Estaciones de Reserva



Ejemplo de Emisión con Estaciones de Reserva



Alternativas para el Envío a las Unidades Funcionales

Reglas de Selección: Se determina las instrucciones que pueden enviarse

Las instrucciones ejecutables

Reglas de Arbitraje: Instrucción que se envía si hay varias ejecutables

La más antigua entre las ejecutables

Orden de Envío: Ordenadas, Desordenadas, o Parcialmente ordenadas (ciertas instrucciones no ejecutables bloquean instrucciones de un tipo, pero no de otros)

Prest

Tend

Ordenada: Power 1 (90), PowerPC 603 (93), Am29000 (95)

Parcialmente ordenada: Power 2 (93), PowerPC 604 (95), PowerPC 620(96)

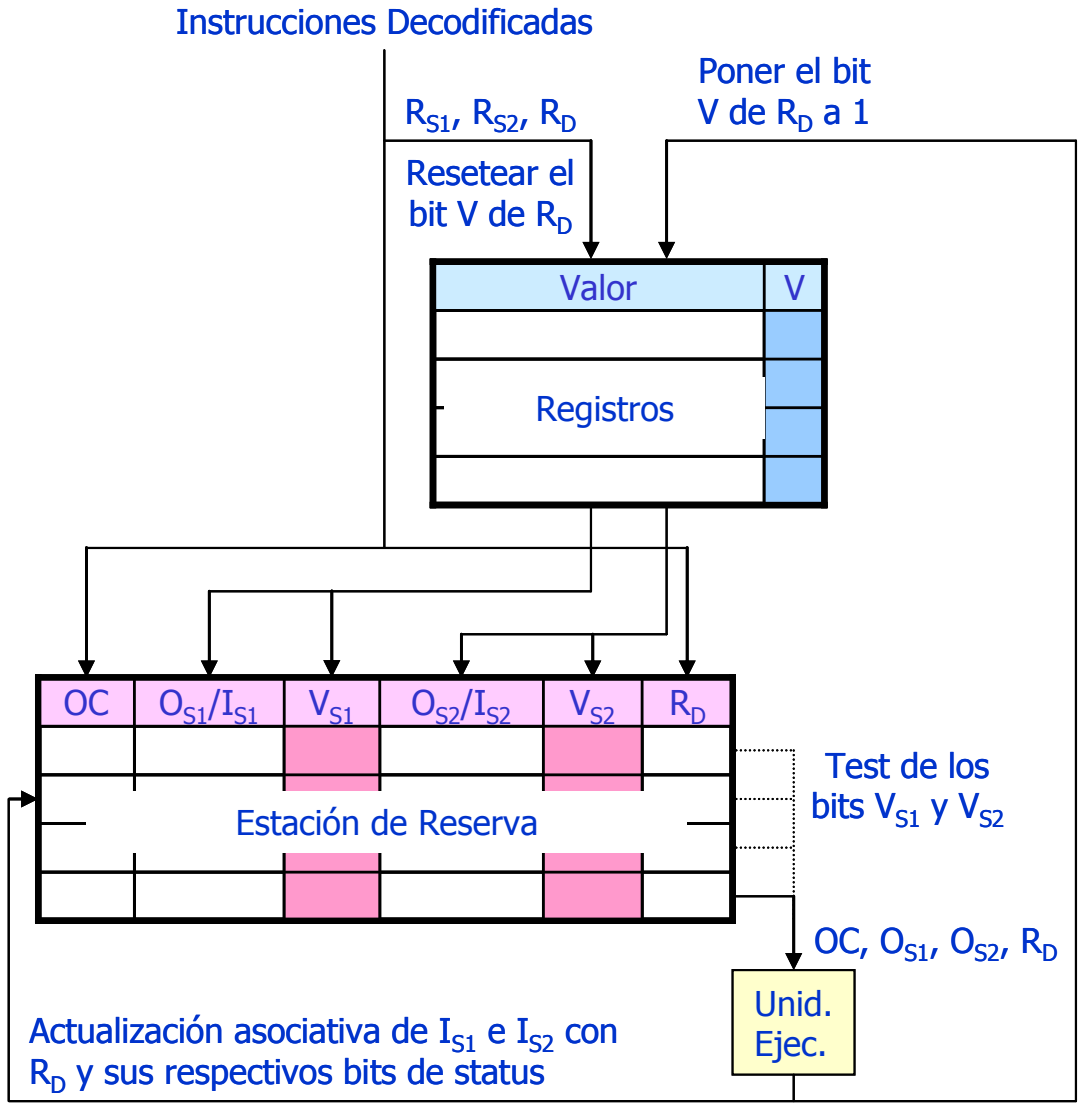
Desordenada: ES/9000(92), PentiumPro (95), R10000(96), PA8000 (96)

Velocidad de Envío: Número de instrucciones que se envían por ciclo

Una por ciclo: PowerPC 603 (93), PowerPC 604 (94), PowerPC 620 (95)

Varias por Ciclo: Sparc64(95), R10000 (96), PA8000 (96), PentiumPro (95)

Comprobación de los Operandos



Comprobación de los bits de validez en la ventana o estación de reserva

(Captación de los operandos en la emisión)

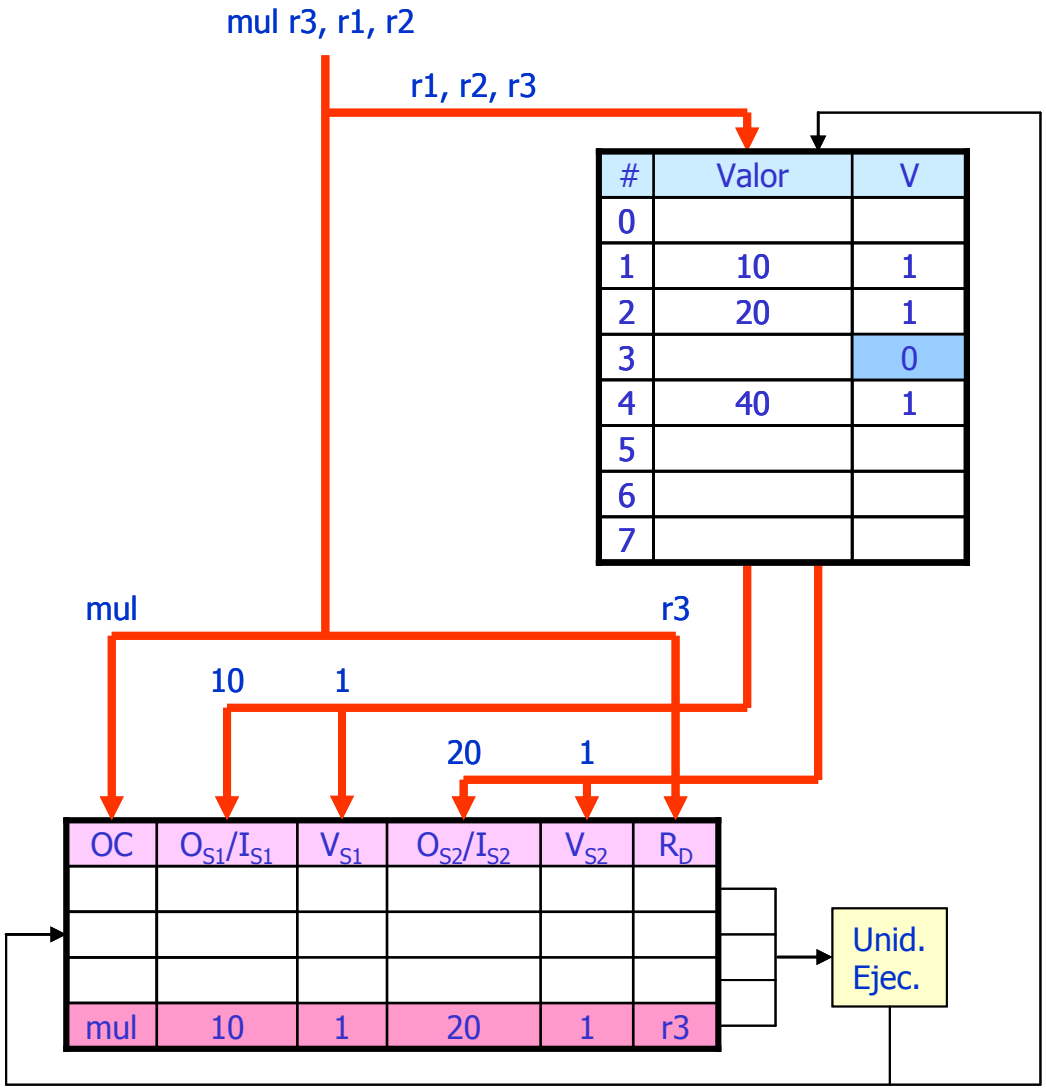
- OC: Código de operación
- R_{S1}, R_{S2}: Registros fuente
- R_D: Registro de destino
- O_{S1}, O_{S2}: Operandos fuente
- I_{S1}, I_{S2}: Identificadores de los operandos fuente
- V_{S1}, V_{S2}: Bits válidos de los operandos fuente

Ejemplo de uso de Estaciones de Reserva I

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

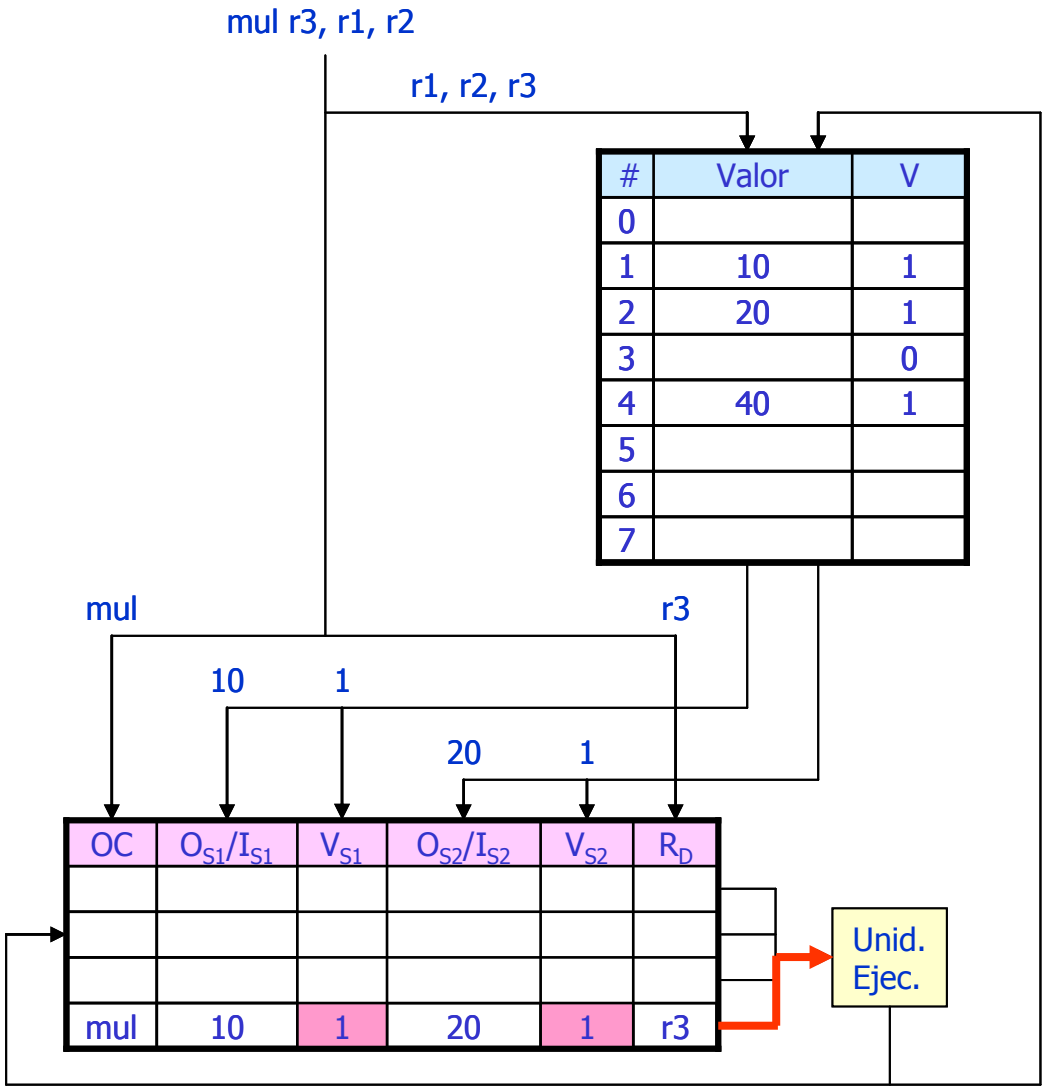


Ejemplo de uso de Estaciones de Reserva II

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



Ejemplo de uso de Estaciones de Reserva III

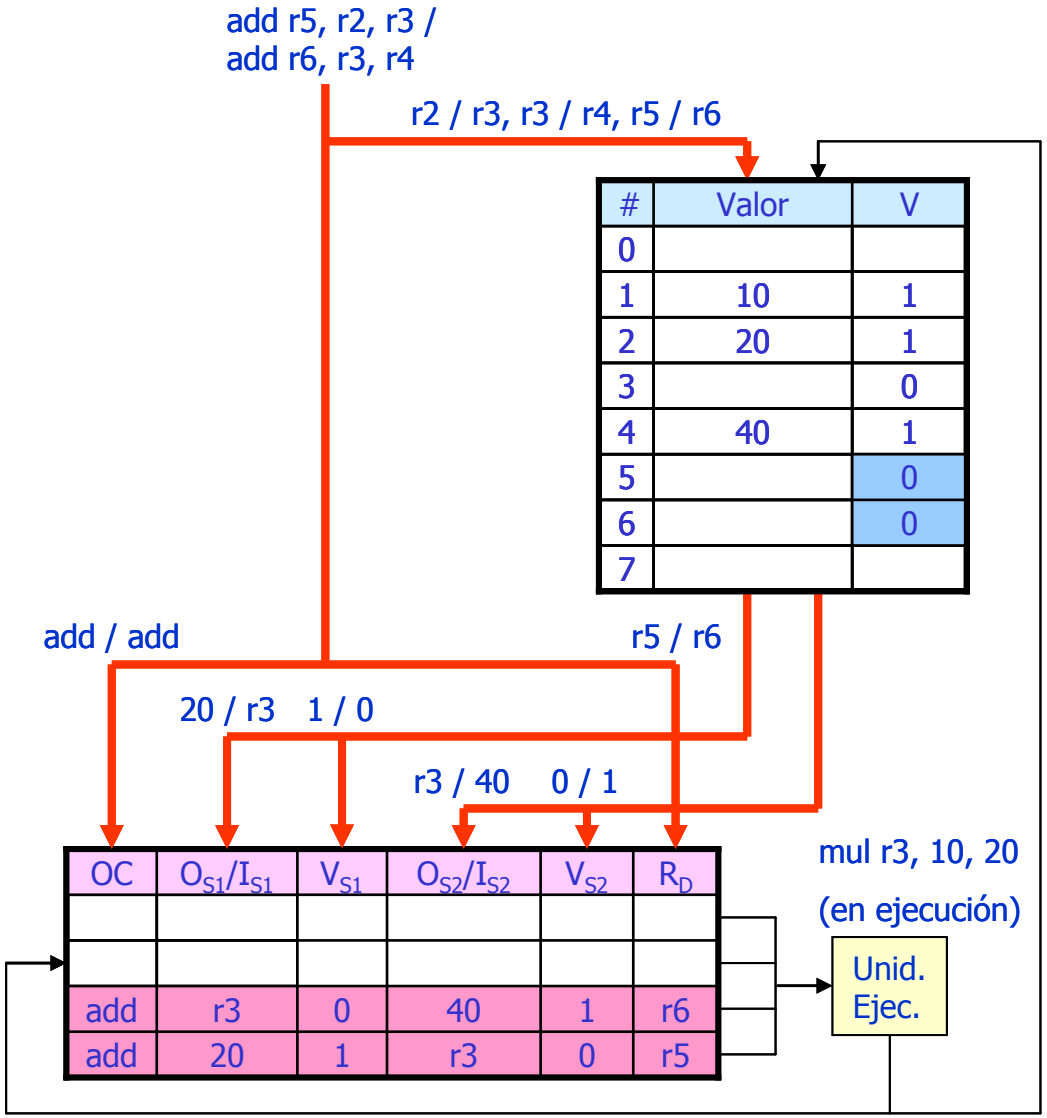
Ciclo i: mul r3, r1, r2

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

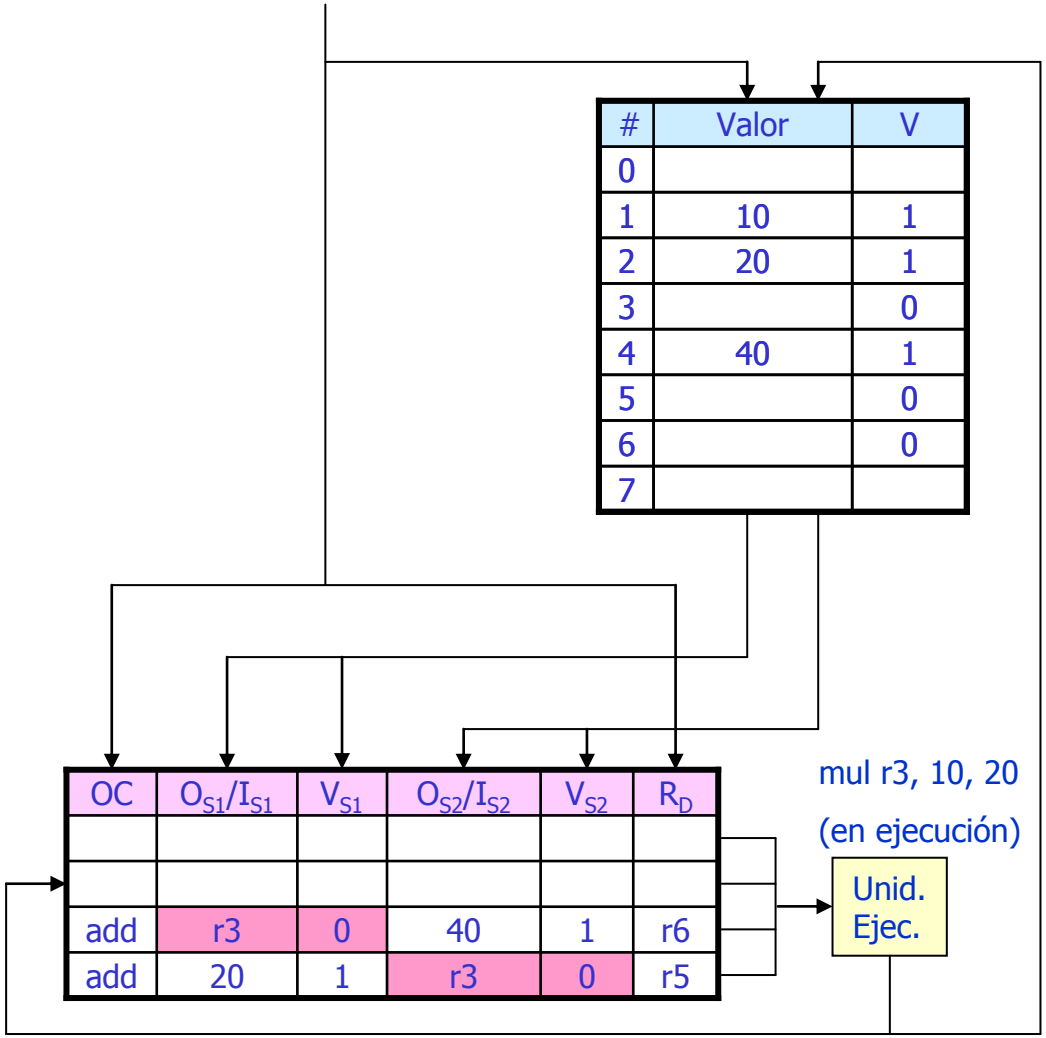


Ejemplo de uso de Estaciones de Reserva IV


```
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)

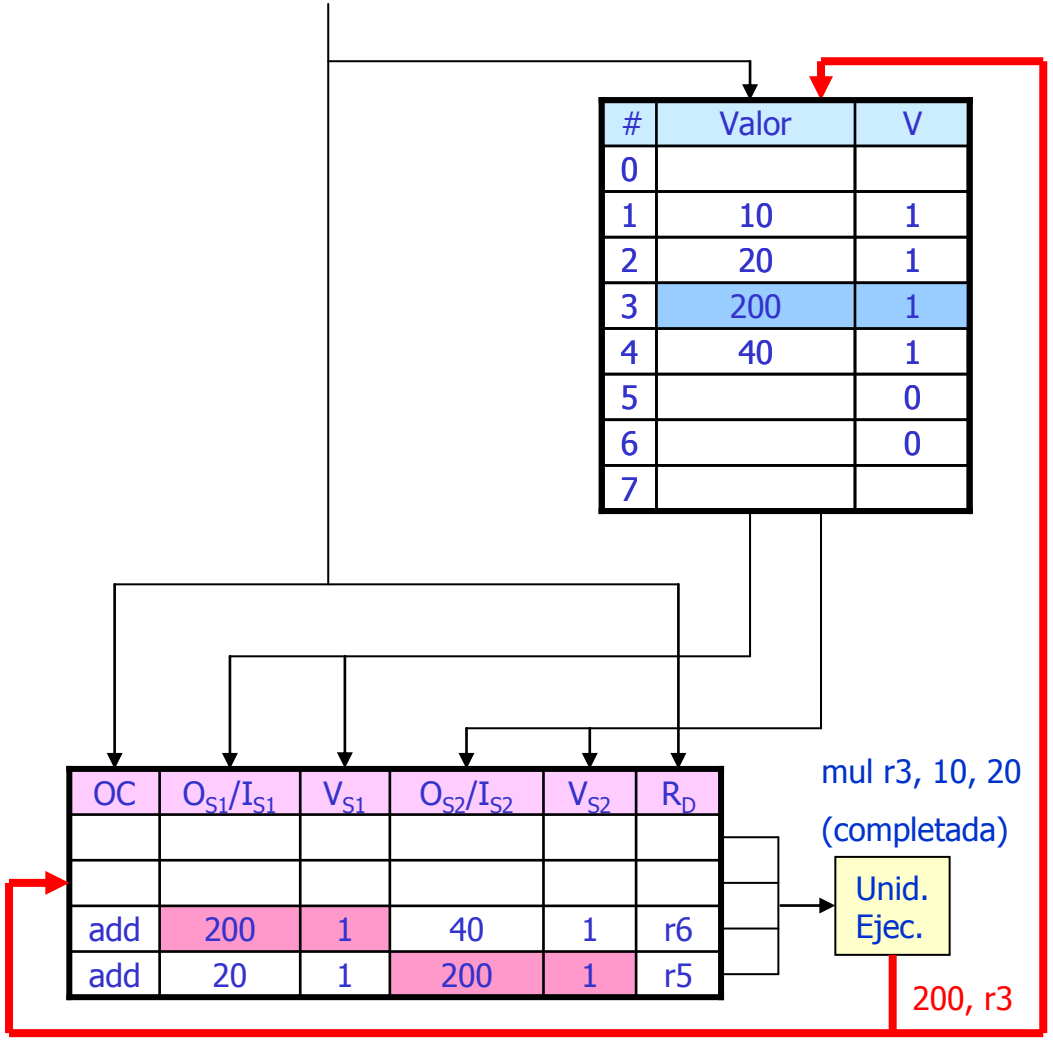


Ejemplo de uso de Estaciones de Reserva V


```
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

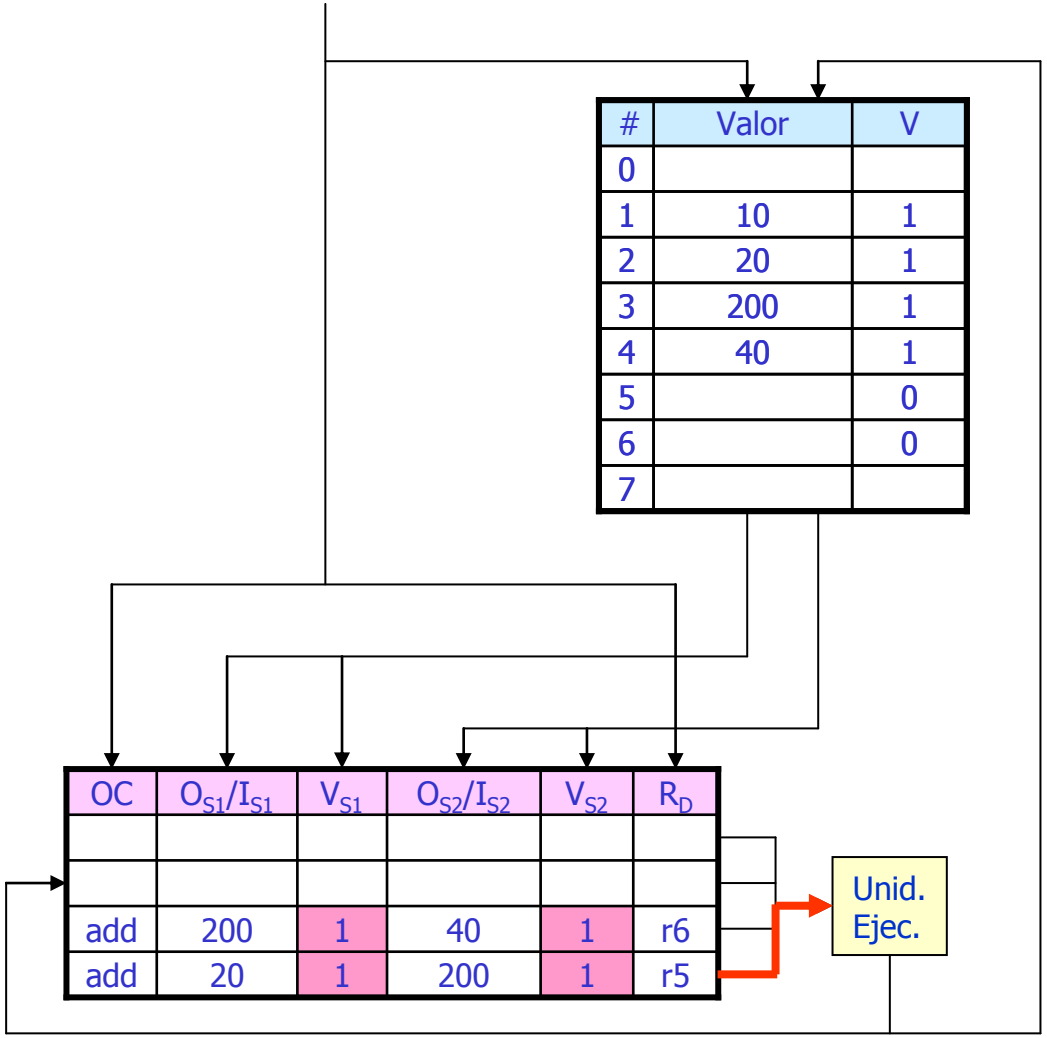


Ejemplo de uso de Estaciones de Reserva VI


```
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

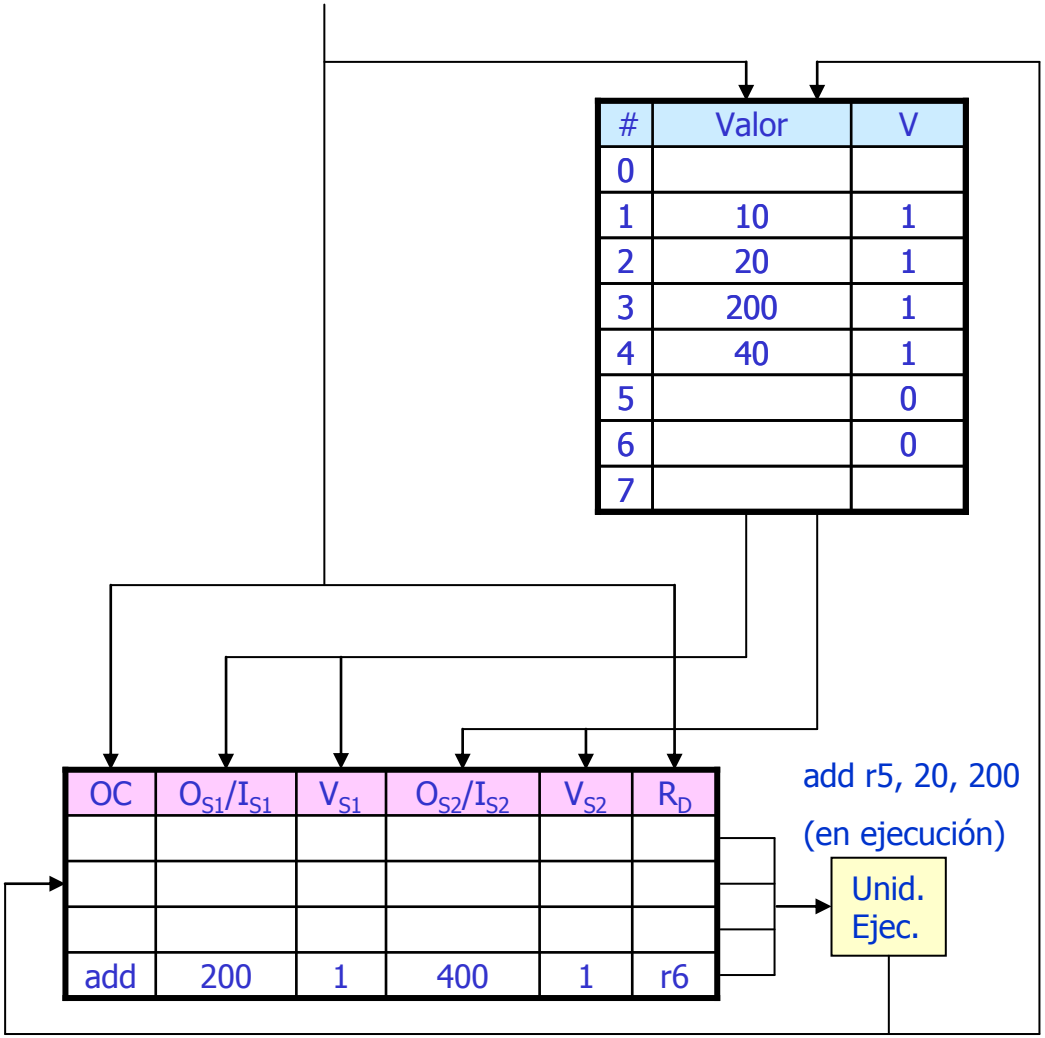


Ejemplo de uso de Estaciones de Reserva VII


```
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

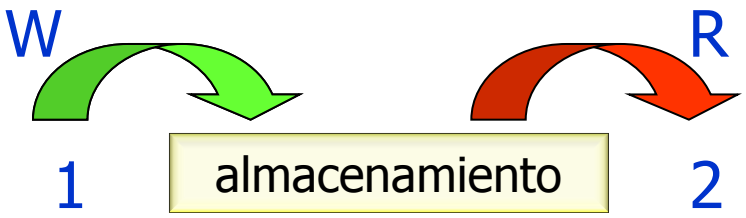


Contenido de la Lección 11

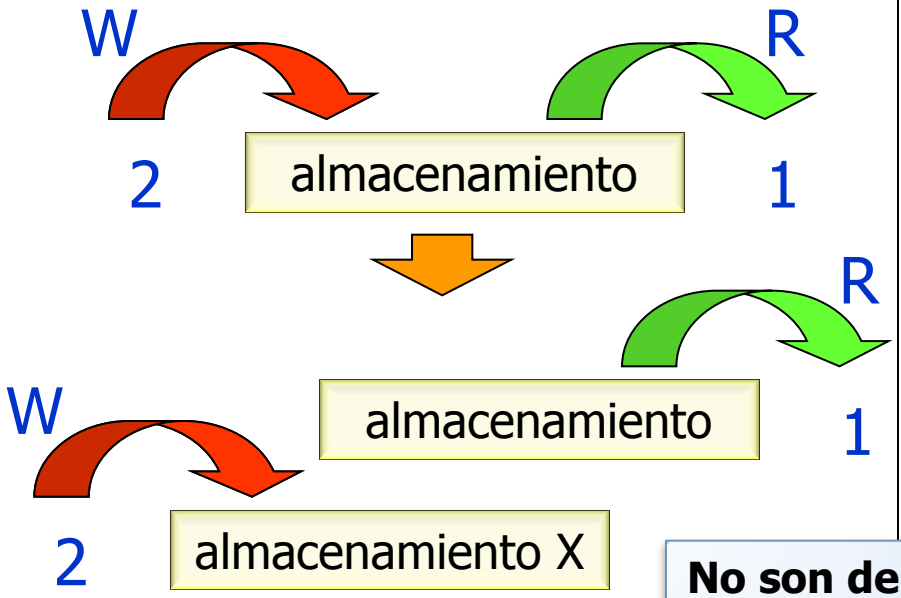
- Introducción: Motivación y Nota Histórica
- Paralelismo entre instrucciones (ILP). Orden en Emisión y Finalización
- Cauces superescalares
  - Decodificación Paralela y Predecodificación
  - Emisión Paralela de Instrucciones. Estaciones de Reserva
  - Renombramiento de registros

Riesgos de Datos

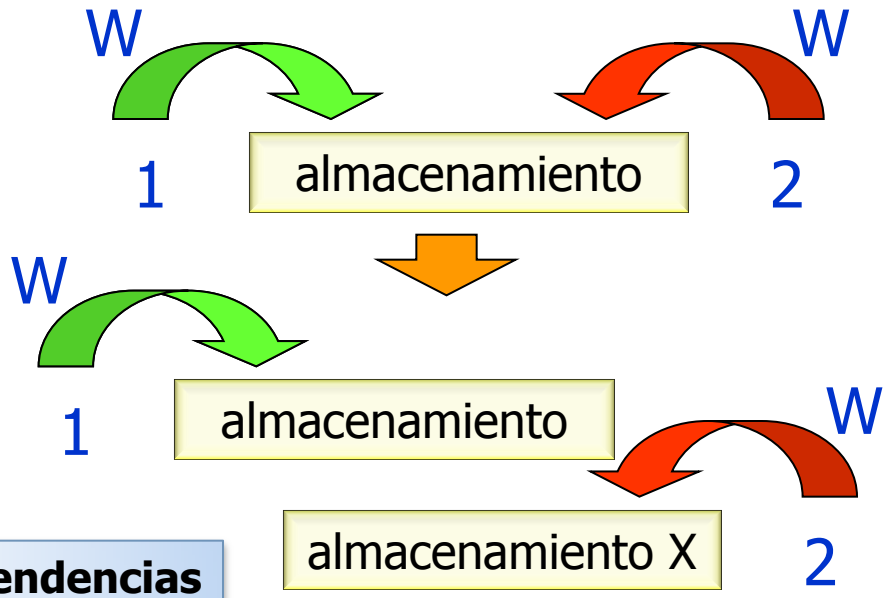
RAW (Read After Write)



WAR (Write After Read)



WAW (Write After Write)



No son dependencias verdaderas

Renombramiento de Registros

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 (circuitería adicional y registros extra)

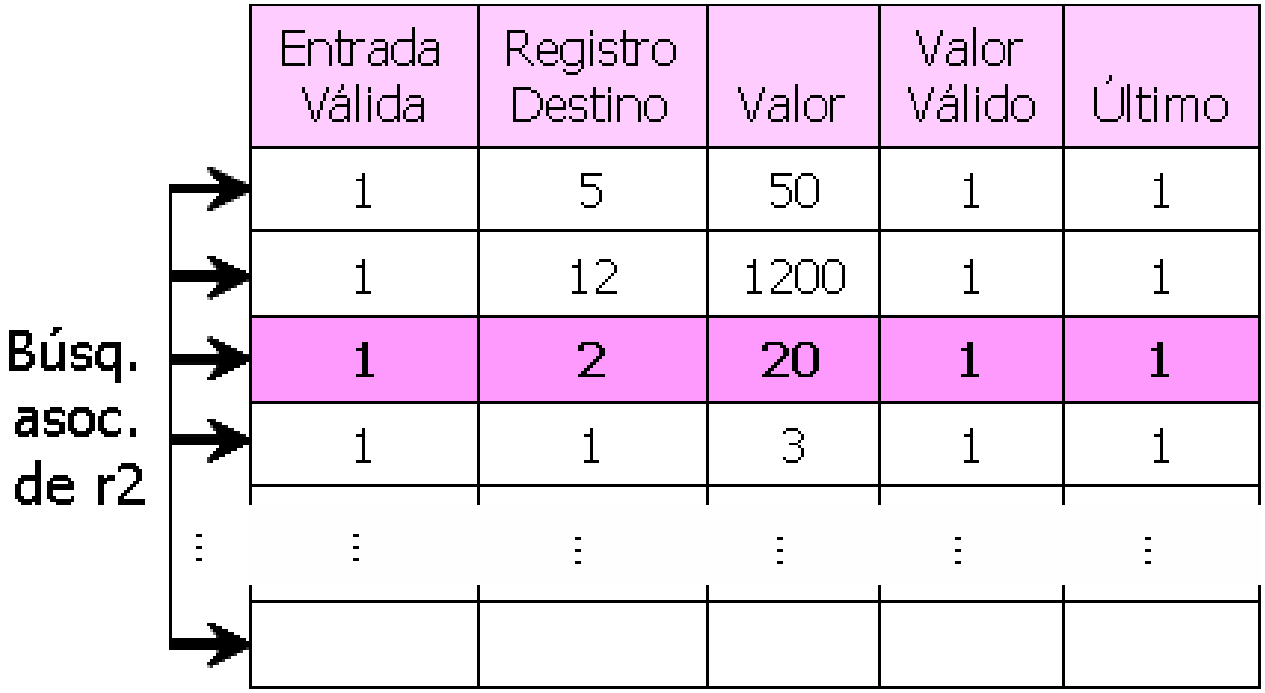
Características de los Buffers de Renombrado

- **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

Buffers de Renombramiento



Permite varias escrituras pendientes a un mismo registro

El bit de último se utiliza para marcar cual ha sido la más reciente

Ejemplo de Renombramiento 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
```

Situación Inicial:

- Las instrucciones se emiten de forma ordenada y de una en una, pero se pueden enviar de forma desordenada
- Existen dos renombrados de r1 en las entradas 2 y 3 del buffer
- La última está en la entrada 3
- Una estación de reserva para add/sub y otra para mul

Ejemplo de Renombramiento 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)

Ejemplo de Renombramiento 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: 4, "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)

Ejemplo de Renombramiento 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 envía la multiplicación y se emite la resta
- Se accede a sus operandos
- Se vuelve a renombrar r2 (destino de sub)

Ejemplo de Renombramiento 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	0
5	1	3		0	1
6	**1**	**2**	**–15**	**1**	**1**
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
```

Ciclo i + 3:

- Se envía la resta

Ciclo i + 4:

- Termina la resta
- Se actualiza el resultado en el buffer de renombrado

Ejemplo de Renombramiento VI

	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	–15	1	1
7	0				
8	0				
9	0				
10	0				
11	0				
12	0				
13	0				
14	0				

r1: 15, "válido" **r2: 0, "válido"**

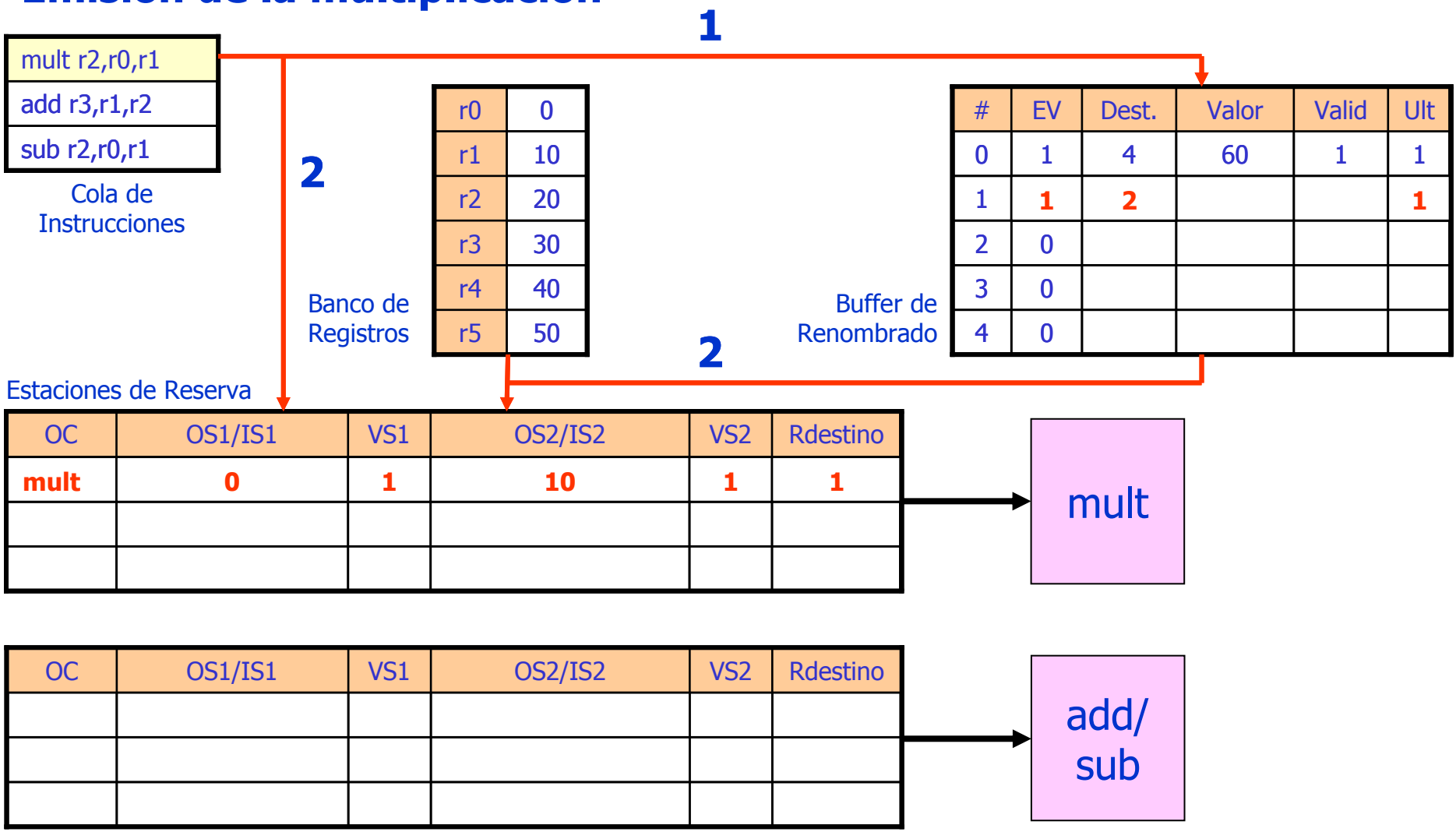
```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

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
- Sólo se escribirá en el banco de registros el último valor de r2 (el de la entrada 6)

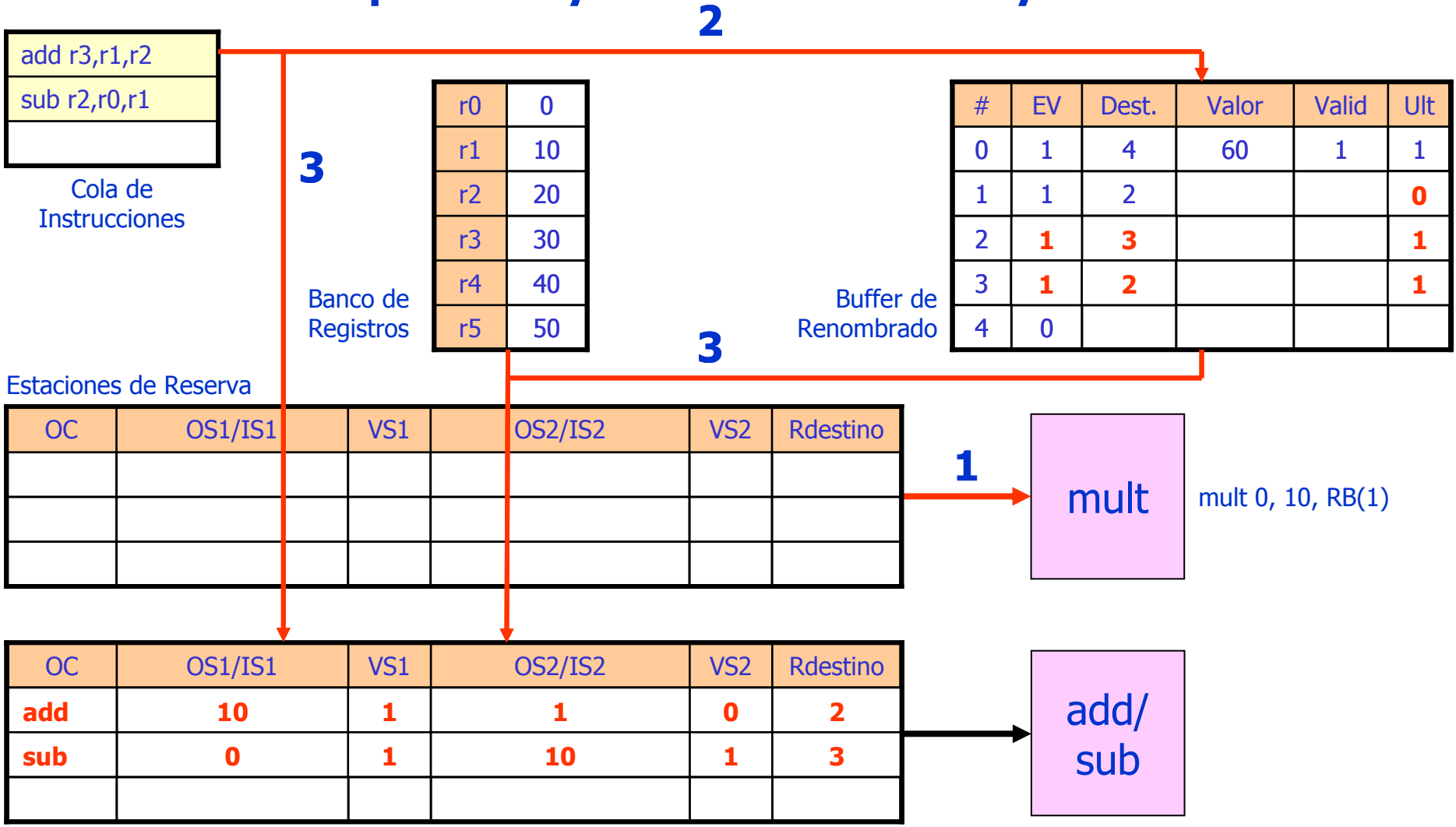
Algoritmo de Tomasulo I

Emisión de la multiplicación



Algoritmo de Tomasulo II

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



Algoritmo de Tomasulo III

Envío de la resta

Cola de Instrucciones

Banco de Registros

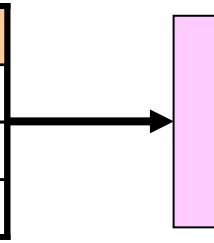
r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

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

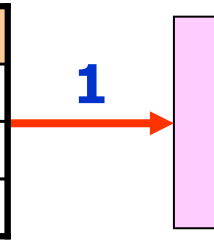
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

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

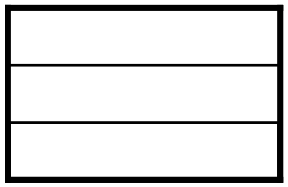
1



sub 0, 10, RB(3)

Algoritmo de Tomasulo IV

Termina la resta



Cola de Instrucciones

Banco de Registros

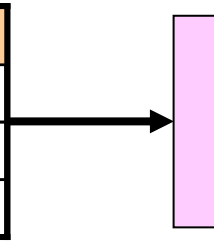
r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

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				

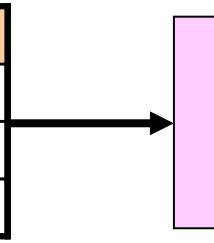
Estaciones de Reserva

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

Algoritmo de Tomasulo 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

mult 0, 10, RB(1)

1

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

add/ sub

Algoritmo de Tomasulo VI

Envío de la suma

Cola de Instrucciones

Banco de Registros

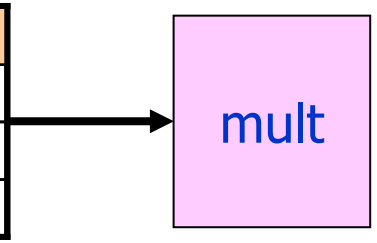
r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Buffer de Renombrado

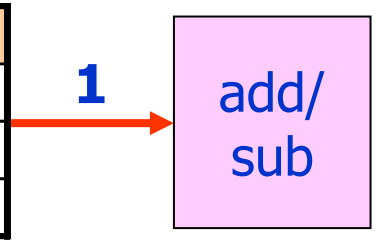
#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	–10	1	1
4	0				

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



add 10, 0, RB(2)

Algoritmo de Tomasulo 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

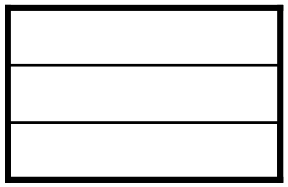
add/ sub

add 10, 0, RB(2)

1

Algoritmo de Tomasulo VIII

Se actualizan los registros



Cola de Instrucciones

Banco de Registros

r0	0
r1	10
r2	**–10**
r3	**10**
r4	**60**
r5	50

1

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

Para ampliar ……

- Artículos de Revistas:
  - Sussenguth, E.: “IBM’s ACS-1 Machine”. IEEE Computer, 22:11. Noviembre, 1999
  - Anderson, D.W.; Sparacio, F.J.; Tomasulo, R.M.: “The IBM 360 Model 91: Processor phylosophy and instruction handling”. IBM J. Research and Development, 11:1, pp.8-24. Enero, 1967.
  - Tomasulo, R.M.: “An efficient algorithm for exploiting multiple arithmetic units”. IBM Res. And. Dev., 11:1, pp.25-33. Enero, 1967.
  - Keller, R.M.: “Look-ahead processors”. Computing Surveys, 7, pp.177-196. Diciembre, 1975. (**Se introduce el término renombramiento de registros**).

Para ampliar ……

- Artículos de Revistas:
  - Agerwala, T.; Cocke, J.: “High Performance Reduced Instruction Set Processors”. IBM Tech. Report, Marzo, 1987. (Descripción del Proyecto América y de la generación del término Superescalar).
  - Wall, D.W.: “Limits of instruction-level parallelism”. Research Rep. 93/6, DEC. Noviembre, 1993.