

2º curso / 2º cuatr.

Grados en  
Ing. Informática

# Arquitectura de Computadores

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

Material elaborado por Mancia Anguita

Profesores de teoría: Mancia Anguita, Maribel García y Christian Morillas



*ugr*

Universidad  
de Granada

# Bibliografía

## ➤ Fundamental

- Capítulo 4. Sección 2 y 3. M. Anguita, J. Ortega.  
*Fundamentos y problemas de Arquitectura de Computadores*, Editorial Técnica Avicam. ESIIT/C.1 ANG fun
- Capítulos 3 y 5. 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

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



Arq. con **DLP**  
(*Data Level Parallelism*)

Ejecutan las **operaciones** de una instrucción **concurr.** o en **paralelo**

Unidades **funcionales** vectoriales o SIMD (90)

Arq. con **ILP**  
(*Instruction Level Parallelism*)

Temas [1,4], BP4

Ejecutan múltiples **instrucciones** **concurr.** o en **paralelo**

Cores escalares segmentados (80), superescalares (90) o VLIW (90)

Arq. con **TLP** (*Thread Level Parallelism*) explícito y **una** instancia de SO

Temas [1-3], BP [0-3]

Ejecutan múltiples **flujos de instrucciones** **concurr.** o en **paralelo**

Cores que modifican la arquit. ILP para ejecutar threads **concurr.** o en **paralelo** (2000)

**Multi-procesadores (60):** ejecutan threads en *paralelo* en un computador con múltiples cores (incluye **multicores (2000)**)

Arq. con **TLP** explícito y **múltiples** instancias SO

Ejec. múltiples **flujos de instr.** en **paralelo**

**Multi-computadores (85):** ejecutan threads en *paralelo* en un sistema con múltiples computadores

# Apartados Tema 4

AC ATC

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

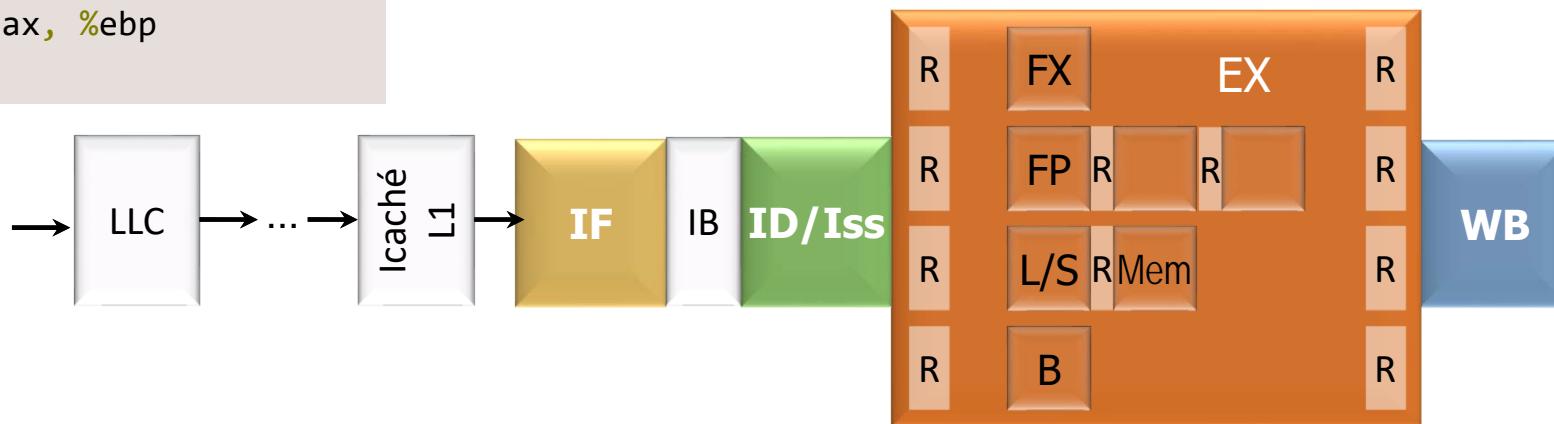


.L6:  
movsd (%r12,%rax,8), %xmm0  
mulsd %xmm1, %xmm0  
addsd (%r13,%rax,8), %xmm0  
movsd %xmm0, (%r13,%rax,8)  
addq \$1, %rax  
cmpl %eax, %ebp  
jg .L6

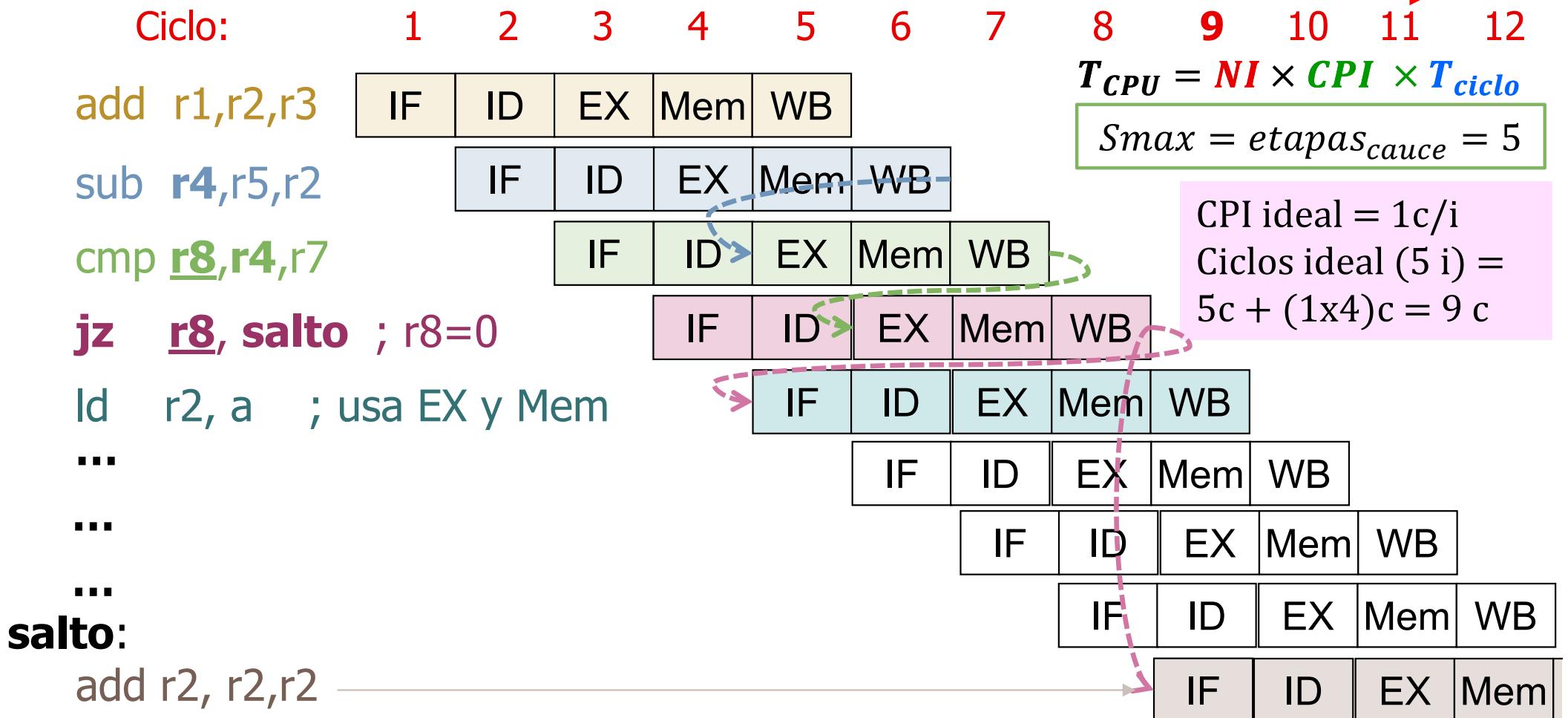
- Microarquitectura de núcleos ILP superescalar
- Microarquitectura de núcleos ILP VLIW Ej.: Google TPU, Elbrus (x86), Nvidia Denver (ARM)

Procesamiento

Ejecución



# Ejecución concurrente de instr. y riesgos (*hazards*): de datos, de control



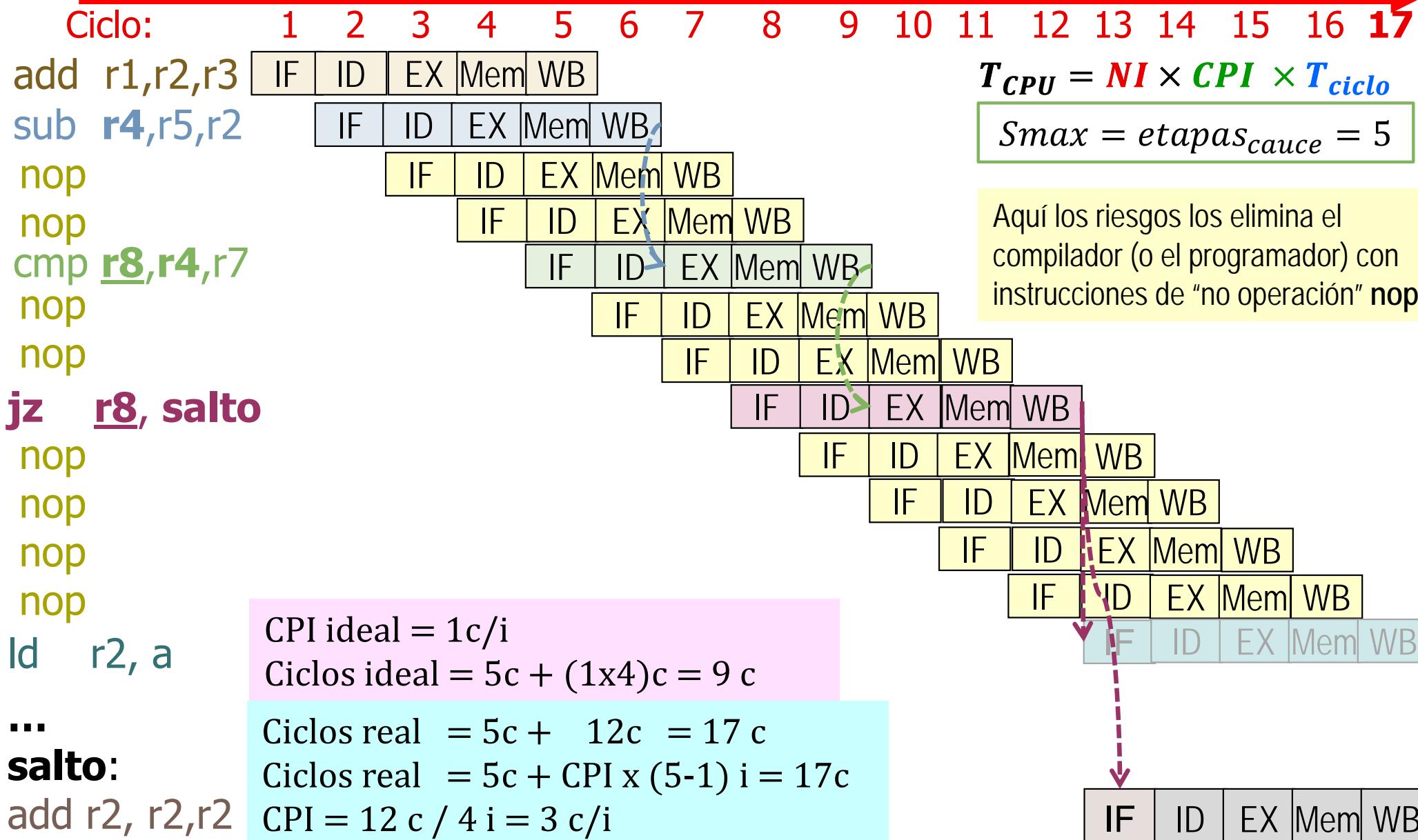
La dependencia es una propiedad del código.

3 dependencias: 2 RAW y 1 de control

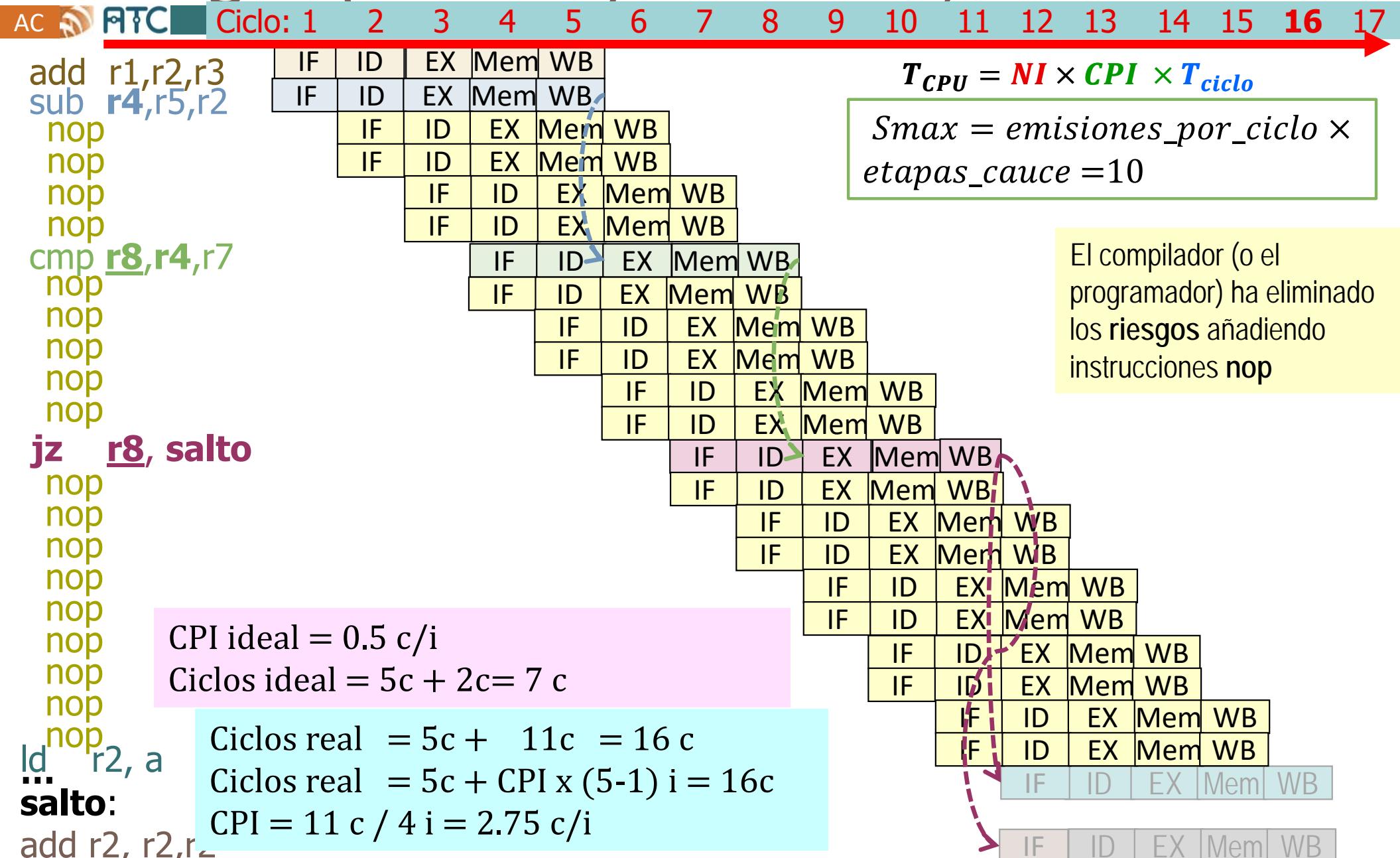
Las dependencias pueden provocar riesgos o conflictos (*hazards*) en un cauce segmentado que pueden llevar a resultados incorrectos (distintos a los que obtendría una ejecución secuencial de las instruc.).

3 riesgos: 2 RAW y 1 de control, ¿Qué se puede hacer para eliminarlos?

# Ejecución concurrente de instr. y riesgos (hazards): de datos, de control



# Ejecución concurrente y paralela de instr. y riesgos (*hazards*): de datos, de control



# Ej. concurrente y paralela de intr. y riesgos (hazards): de datos, de control, estructural

AC ATC Ciclo: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

add r1,r2,r3

nop

sub r4,r5,r2

nop

nop

Nop

nop

cmp r8,r4,r7

nop

nop

nop

nop

nop

jz r8, salto

nop

nop

nop

nop

nop

nop

nop

nop

ld r2, a

salto:

add r2, r2, r2

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

IF	ID	EX	Mem	WB
----	----	----	-----	----

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

$$Smax = \text{emisiones\_por\_ciclo} \times \text{etapas\_cauce} = 10$$

El compilador (o el programador) ha eliminado los riesgos añadiendo instrucciones nop

$$\text{CPI ideal} = 0.5 \text{ c/i}$$

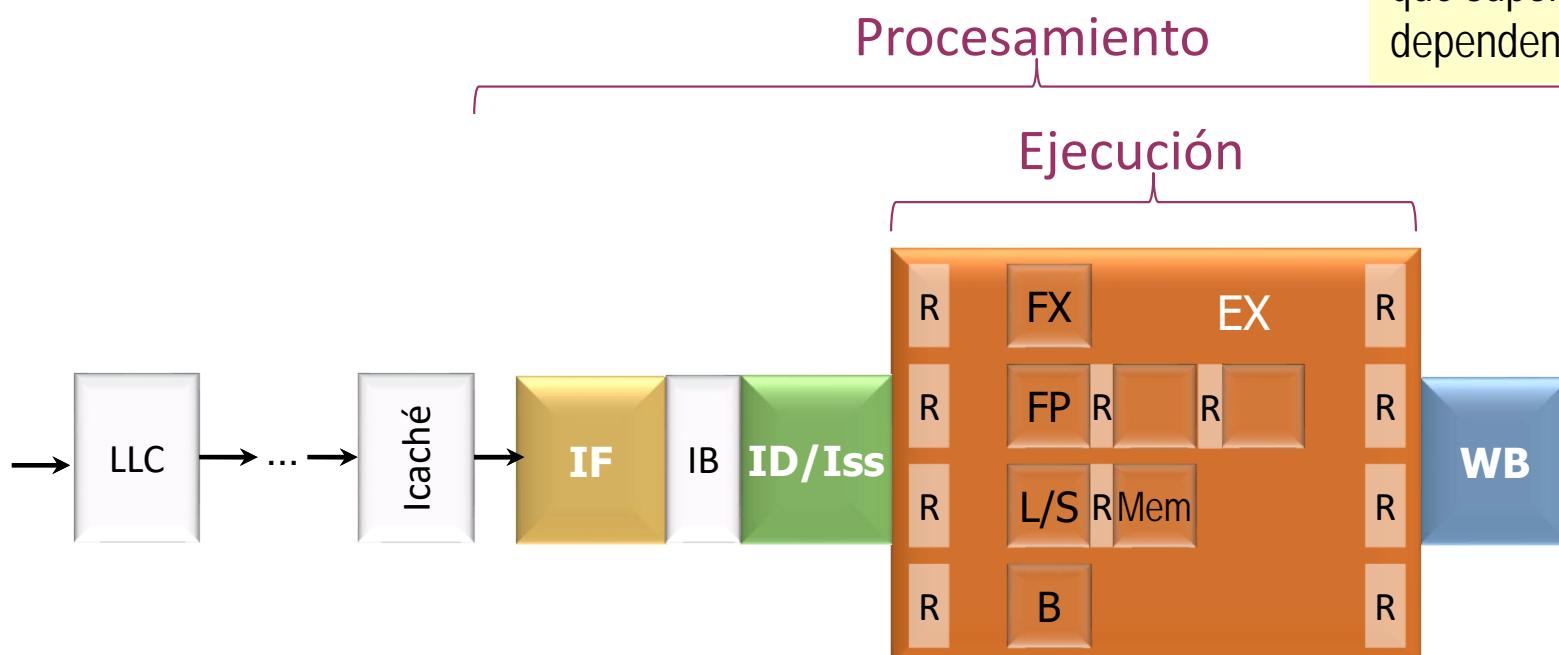
$$\text{Ciclos ideal} = 5c + 2c = 7 \text{ c}$$

$$\begin{aligned} \text{Ciclos real} &= 5c + 12c = 17 \text{ c} \\ \text{Ciclos real} &= 5c + \text{CPI} \times (5-1) i = 17c \\ \text{CPI} &= 12 \text{ c} / 4 i = 3 \text{ c/i} \end{aligned}$$

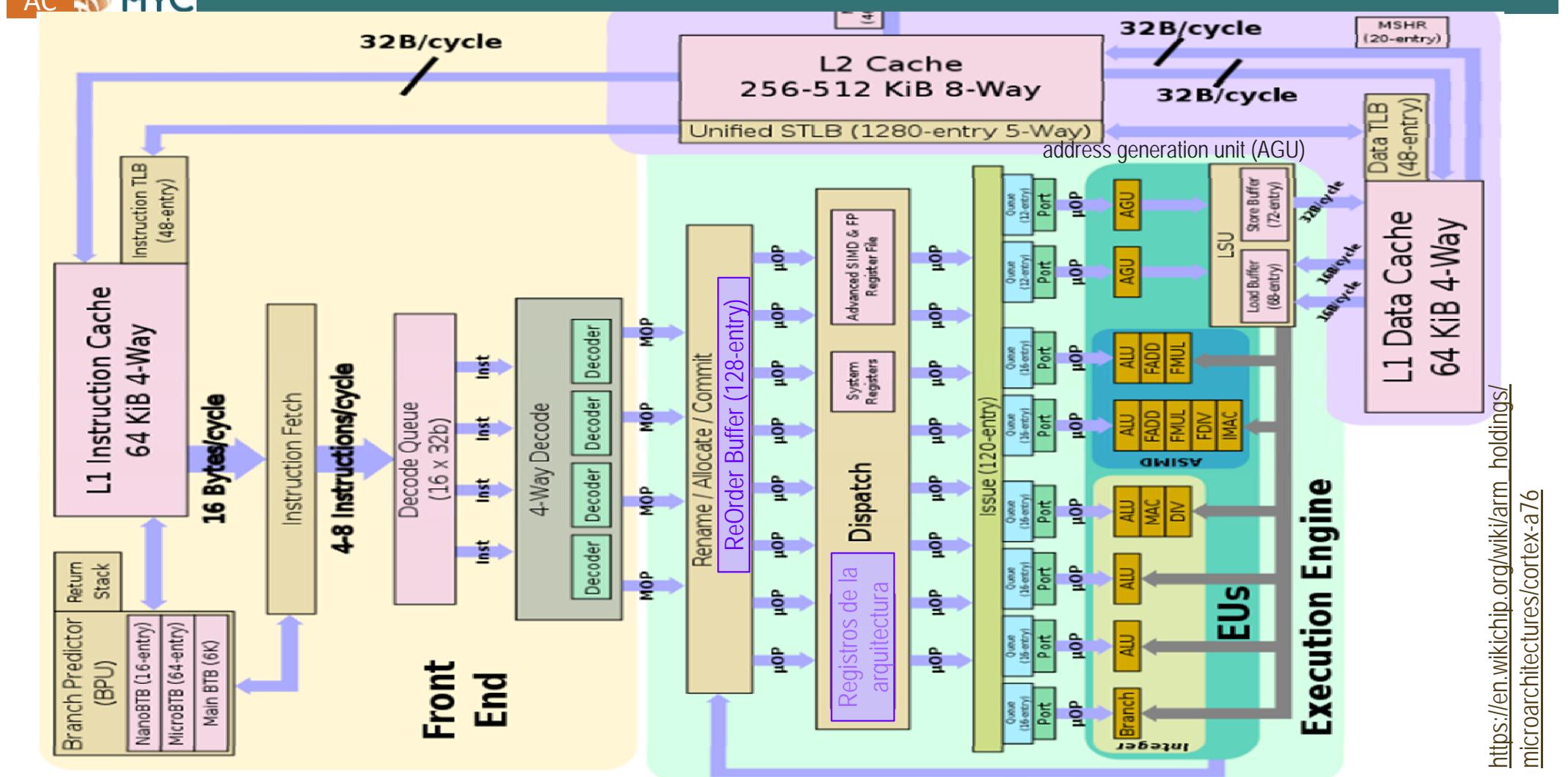
# Apartados

- Microarquitectura de núcleos ILP superescalar
  - Cauce superescalar
  - Emisión
  - Consistencia del procesador y buffer de reorden
  - Procesamiento de saltos
- Microarquitectura de núcleos ILP VLIW

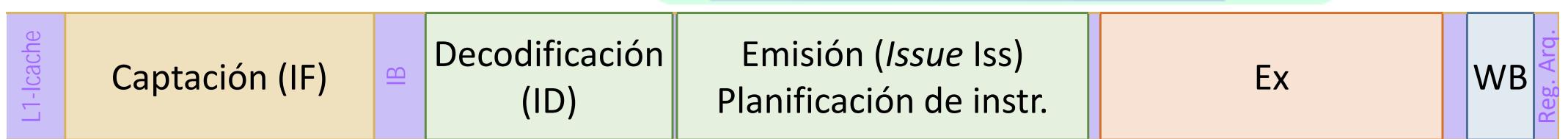
El hardware de un núcleo superescalar **planifica dinámicamente** la ejecución de las instrucciones, eliminando los riesgos provocados por **dependencias** y reduciendo o eliminando la penalización que suponen las dependencias.



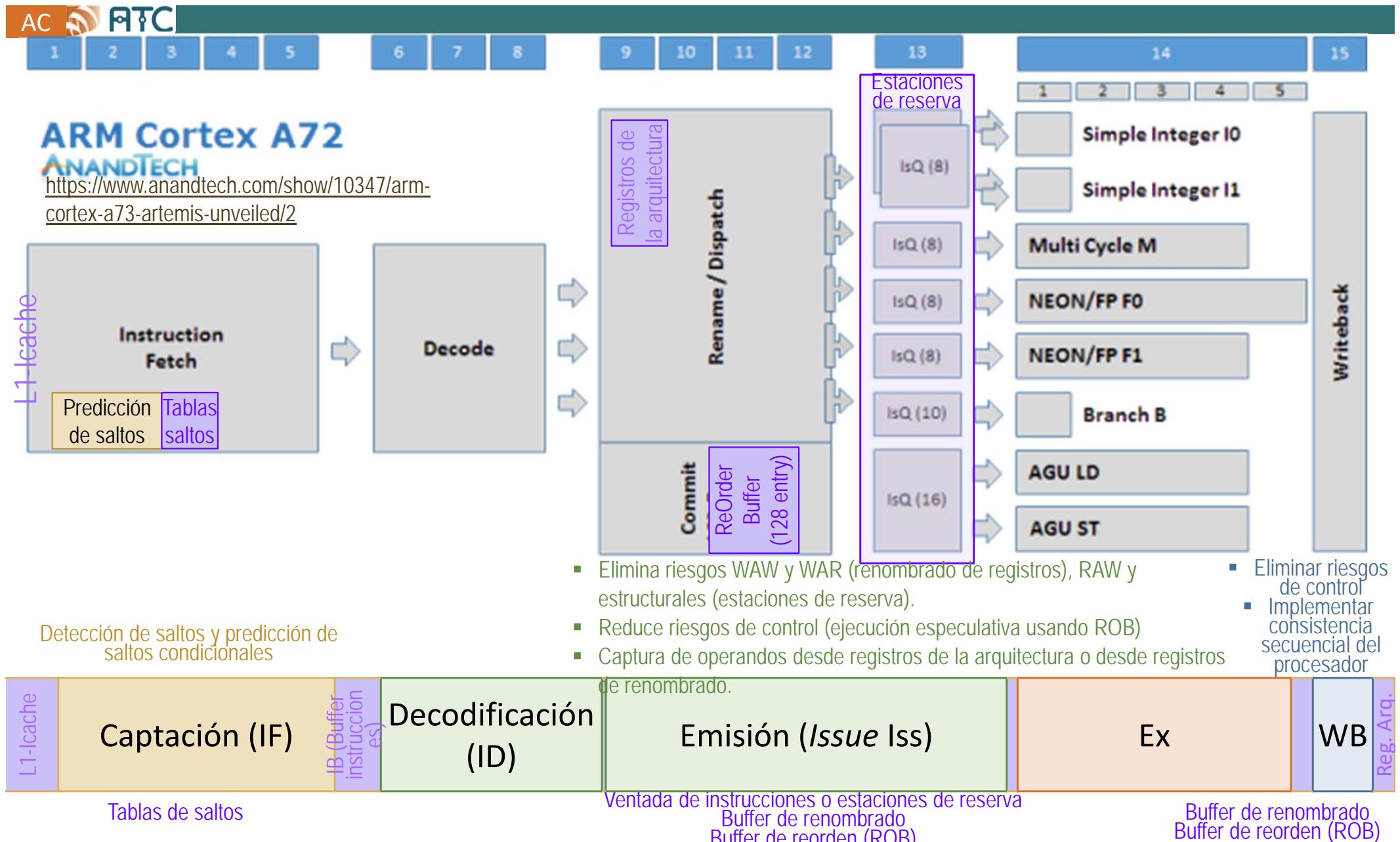
# Ejemplo de Cauce (ARM cortex A76, microarquitectura de ARMv8)



[https://en.wikichip.org/wiki/arm\\_holdings/microarchitectures/cortex-a76](https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a76)



# Ejemplo de Cauce (ARM cortex A72)



# Apartados

- Microarquitectura ILP superescalar
  - Cauce superescalar
  - Emisión (Emisión +Envío a UF) ( $\text{Issue} \Leftrightarrow \text{Dispatch}$ ) (Algoritmo de Tomasulo, 1967 IBM 360/91, se extendió en los 90). Planificación dinámica de instrucciones
  - Consistencia del procesador y buffer de reorden
  - Procesamiento de saltos

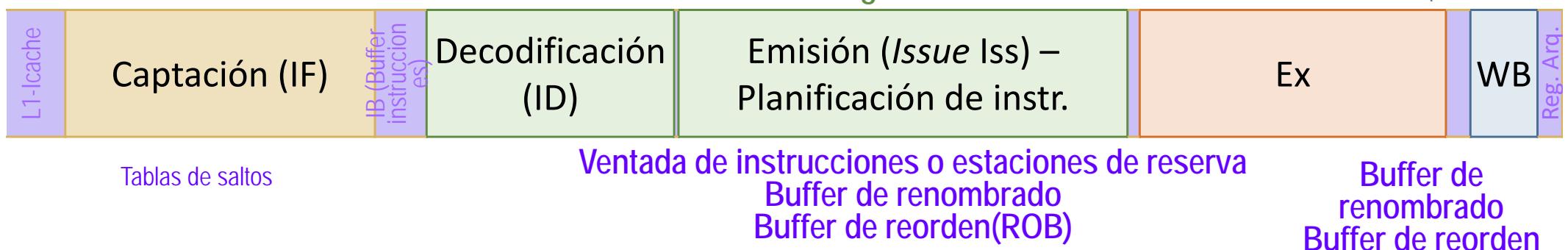
Procesamiento en el orden del programa

Procesamiento desordenado

- Detección de saltos y predicción de saltos condicionales

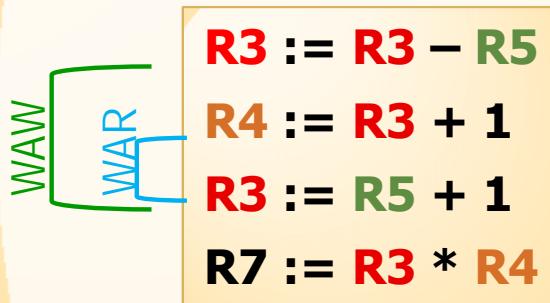
- Elimina riesgos RAW y estructurales (estaciones de reserva)
- Elimina riesgos WAW y WAR (renombrado de registros)
- Reduce riesgos de control (ejecución especulativa en ROB)
- Captura de operandos desde registros de la arquitectura o desde registros de renombrado.

- Ejecución especulativa
- Implementar consistencia secuencial del procesador



# 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).



Cada escritura se asigna a un registro físico distinto



R.M. Tomasulo (67)

**Implementación Estática:** Durante la Compilación

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

#	U	Dest.	Valor	Válido	Ult.
0	1	5		0	0
1	1	3	62	1	0
2	1	5	58	1	1
3	1	3	4	1	0
4	1	4	5	1	1
5	1	3		0	1
6	1	7		0	1
7	0				
8	0				

**Características del fichero de registros (*buffer*) de Renombrado**

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

**Velocidad del Renombrado**

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

# Tomasulo I

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arg



## Emisión de la multiplicación

RAW

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

Cola de Instrucciones  
Emisión ordenada

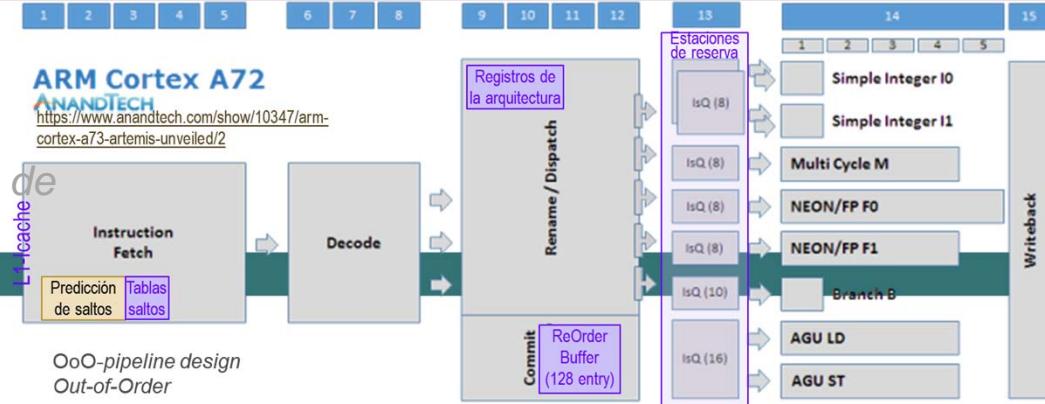
OC:Código de operación  
OS: operando fuente  
VS: bit válido fuente  
Estaciones de Reserva

2

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

Banco de Registros

1

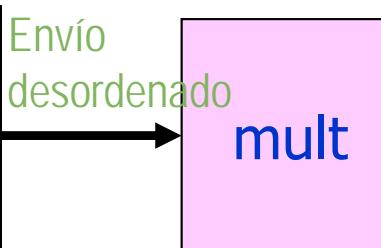


2

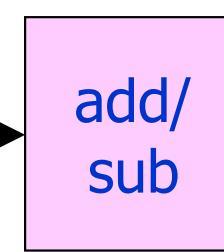
#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			1
2	0				
3	0				
4	0				

espiar (snoop)

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



OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



CDB (Common Data Bus)

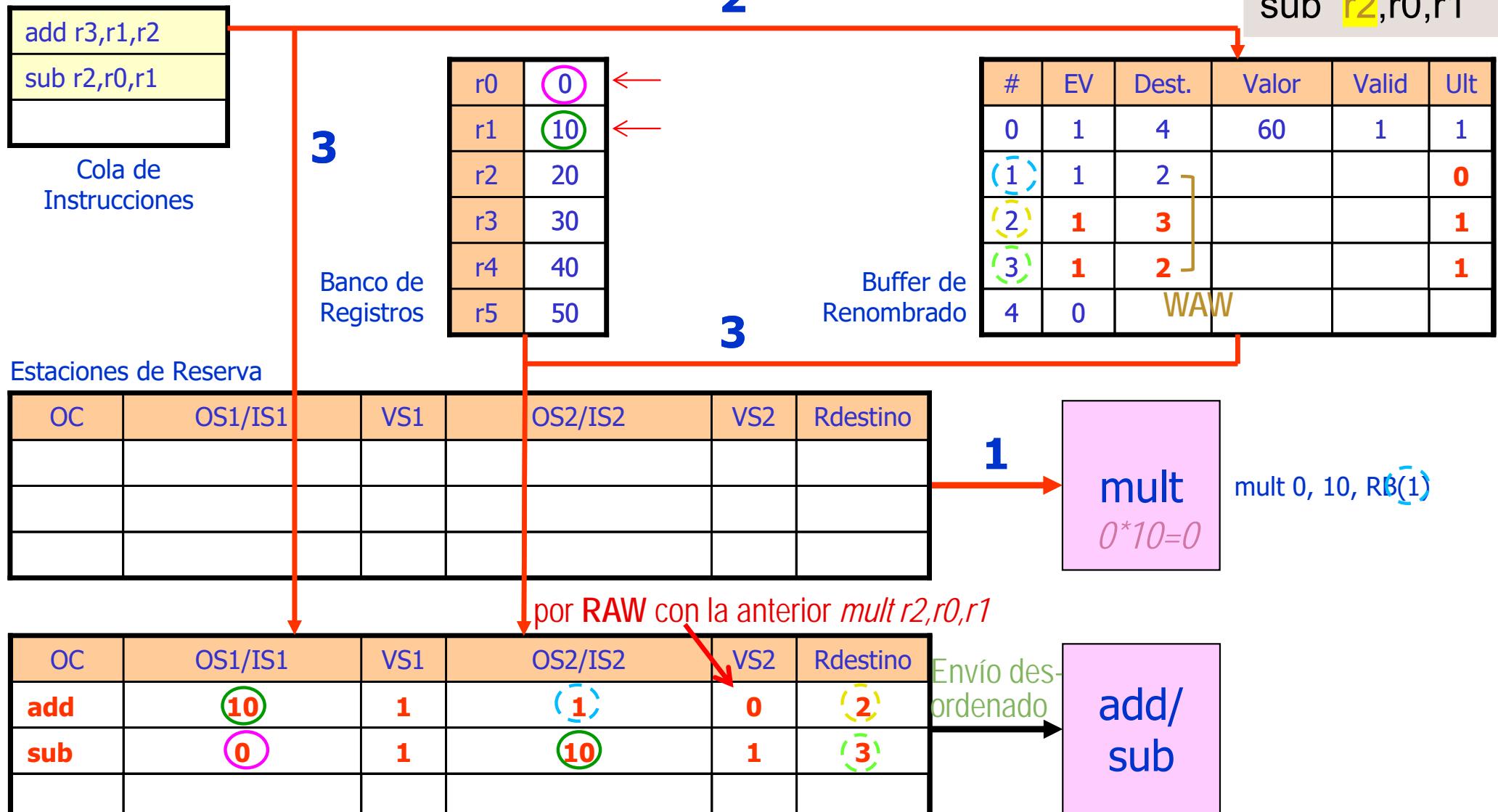
# Tomasulo II

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



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

2



# Tomasulo III

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



## Envío de la resta


Cola de Instrucciones

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

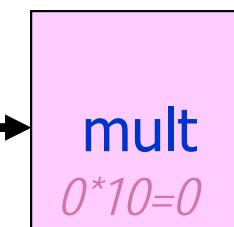
Banco de Registros

#	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				

Buffer de Renombrado

Estaciones de Reserva

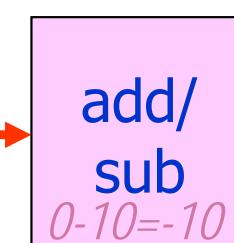
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



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)

# Tomasulo IV

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



## Termina la resta


Cola de Instrucciones

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

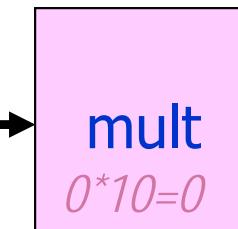
Banco de Registros

#	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				

Buffer de Renombrado

Estaciones de Reserva

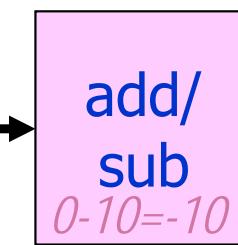
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

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



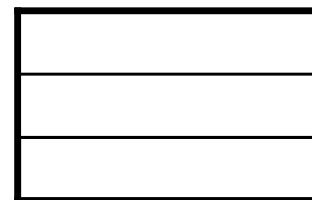
sub 0, 10, RB(3)

# Tomasulo V

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arg



## Termina la multiplicación



Cola de Instrucciones

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

Banco de Registros

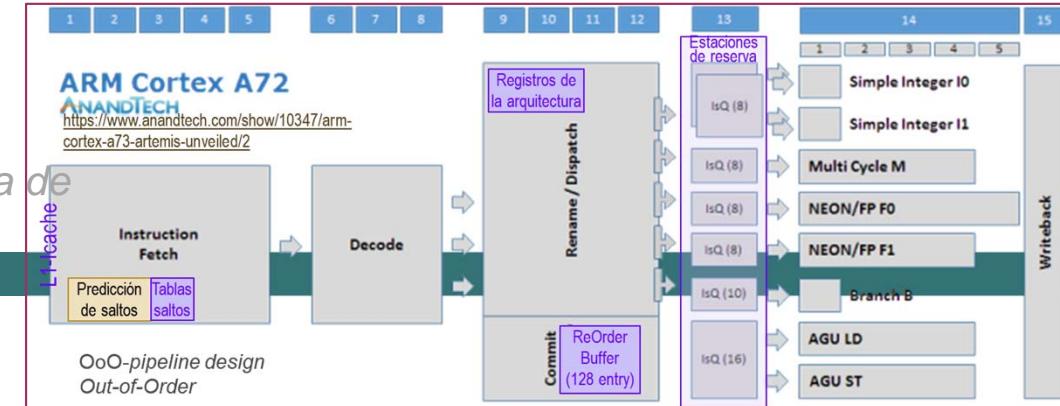
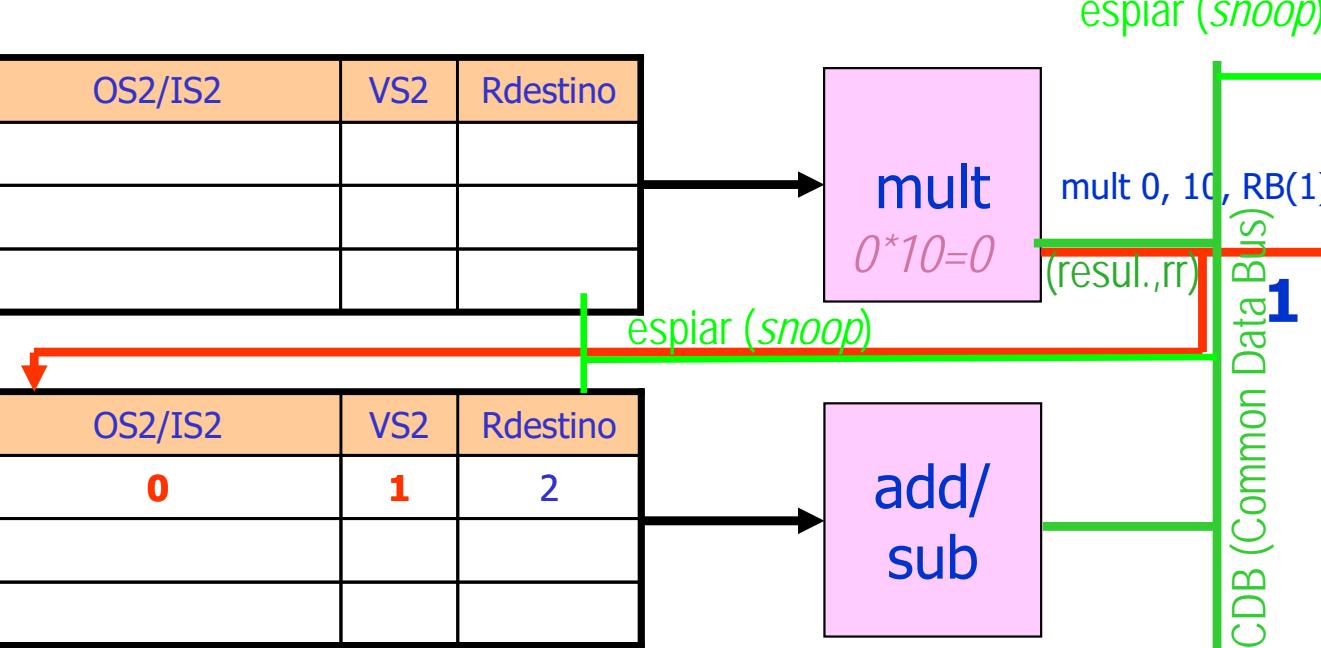
#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Buffer de Renombrado

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

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



# Tomasulo VI

Ejemplo de J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*. Thomson, 2005. ESIIT/C.1 ORT arq



## Envío de la suma


Cola de Instrucciones

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

Banco de Registros

#	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				

Buffer de Renombrado

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

1

add/  
sub  
 $10+0=10$

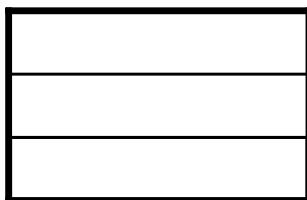
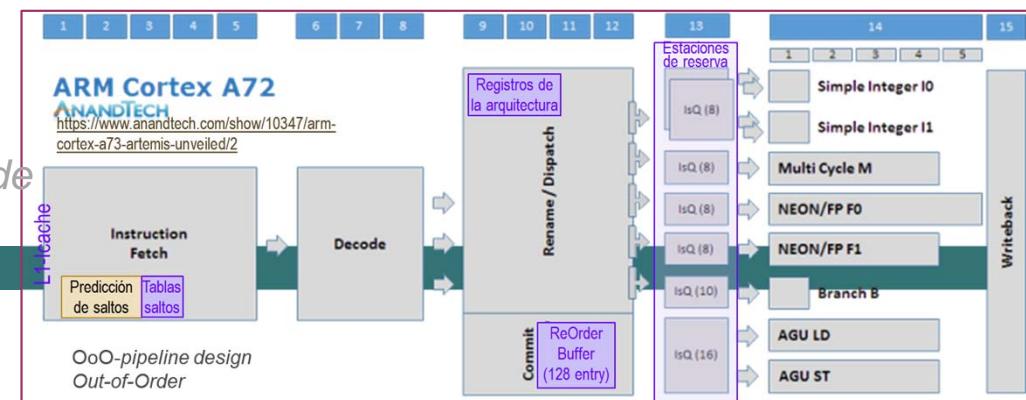
add 10, 0, RB(2)

# Tomasulo VII

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



## Termina la suma



Cola de Instrucciones

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

Banco de Registros

#	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				

Buffer de Renombrado

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

add/  
sub  
 $10+0=10$

add 10, 0, RB(2)

# Tomasulo VIII

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



**Se actualizan los registros (etapa WB - commit)**

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


Cola de Instrucciones

r0	0
r1	10
r2	-10
r3	10
r4	60
r5	50

Banco de Registros

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

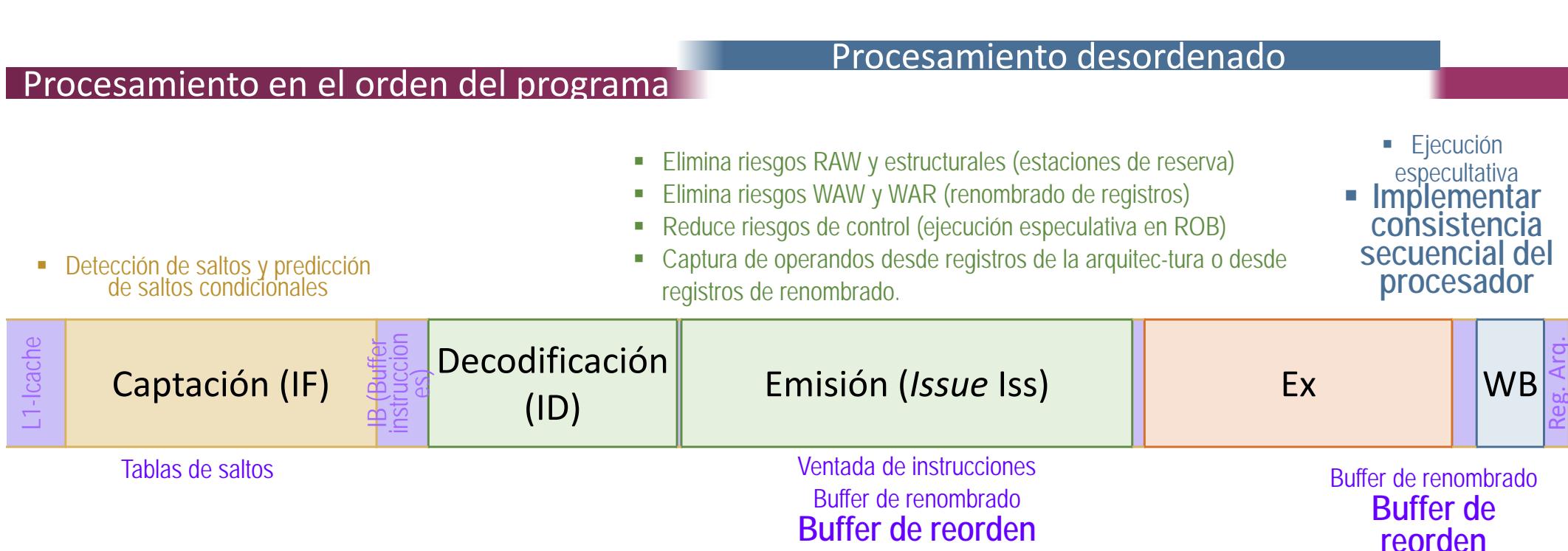
mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

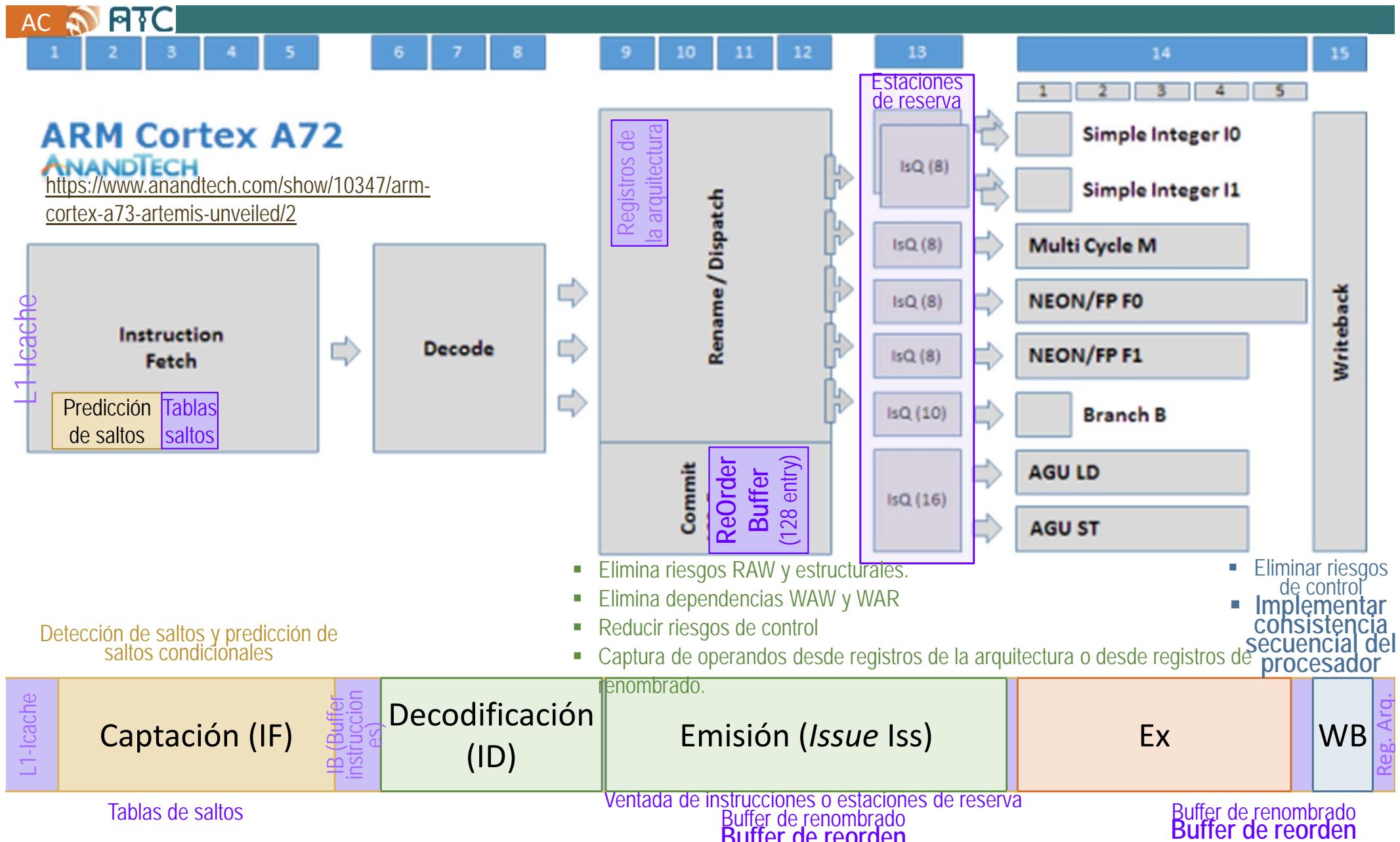
add/  
sub

# Apartados

- Microarquitectura ILP superescalar
  - Cauce superescalar
  - Emisión (Algoritmo de Tomasulo)
  - **Consistencia del procesador y buffer de reorden**
  - Procesamiento de saltos

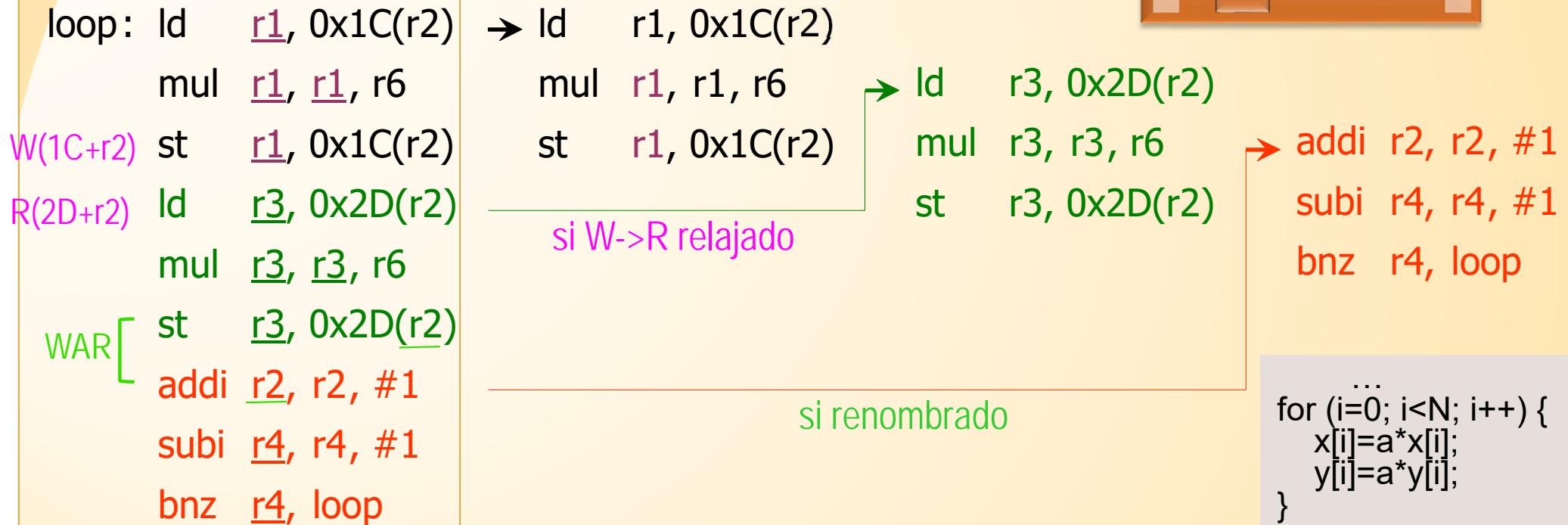
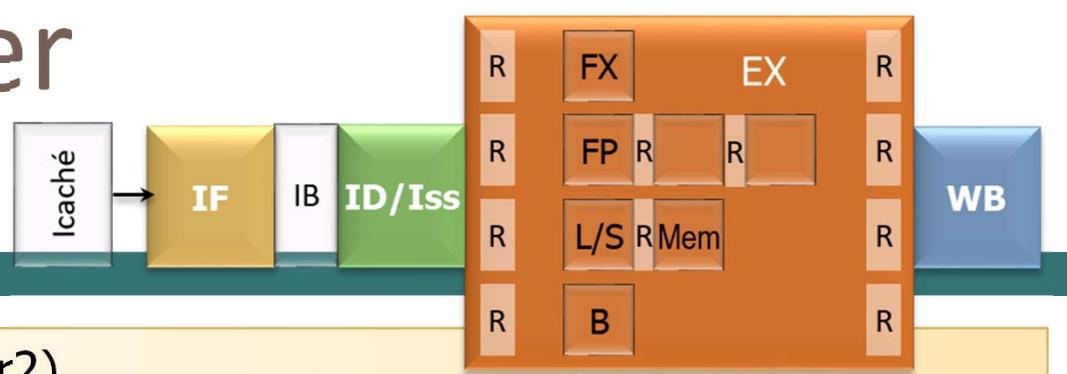


# Ejemplo de Cauce (ARM cortex A72)



# Reordenar y extraer paralelismo

AC ATC



Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq

Si se tuviera: st r1,0x1C(r2) W(1C+r2)

Id r3,0x2D(**r7**) R(2D+r7)

las direcciones 0x1C(r2) y 0x2D(r7) podría coincidir.

¿RAW?

¿ 0x1C(r2) = 0x2D(r7) ?

Se podría usar entonces un **load especulativo**

# Buffer Reordenamiento (ROB) y consistencia secuencial del procesador I

1			
2	instr( $n$ )	f	.....
3	instr( $n+1$ )	x	.....
4	instr( $n+2$ )	f	.....
5	instr( $n+3$ )	x	.....
6	instr( $n+4$ )	x	.....
7	instr( $n+5$ )	i	.....
8	instr( $n+6$ )	i	.....
9			
10			

## Cola

(Las instrucciones se retiran desde aquí)

*La gestión de interrupciones y la ejecución especulativa se pueden implementar fácilmente mediante el ROB*

**Cabecera** (Las instrucciones que se decodifican se introducen a partir de aquí)

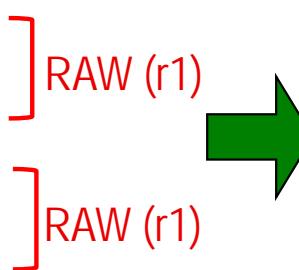
# Buffer de Reordenamiento (ROB) II

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



## Ejemplo de uso del ROB

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 emite 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 emite a la estación de reserva de la *ALU* para esperar a **r1**

Estación de Reserva (Unidad de Almacenamiento)

codop	dirección	op1	ok1
I2 st	0x1ca	3	0

Estación de Reserva (ALU)

codop	dest	op1	ok1	op2	ok2
I4 xor	6	5	0	[r3]	1

3, 5 y 6 renombran todos a r1

Líneas del ROB

# Buffer de Reordenamiento (ROB) III

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



## Ciclo 7 Situación en este ciclo

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca
3	I1 mult	7	r1	int_mult	-	0	x ] RAW
4	I2 st	8	-	store	-	0	i ] RAW
5	I3 add	9	r1	int_add	-	0	x ] RAW
6	I4 xor	10	r1	int_alu	-	0	i ] RAW

## Ciclo 9 Terminó add, pero todavía no se puede retirar

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

## Ciclo 10 Terminó xor, pero todavía no se puede retirar

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

# Buffer de Reordenamiento (ROB) IV

Ejemplo de J. Ortega, M. Anguita, A. Prieto. Arquitectura de Computadores. Thomson, 2005. ESIIT/C.1 ORT arq



**Ciclo 12** Terminaron las instr. de la cola, **mult** y **st**, y se retiran (completan el procesamiento)

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	Ok	marca
3	I1	mult	7	r1	int_mult	33	1
4	I2	st	8	-	store	-	1
5	I3	add	9	r1	int_add	17	1
6	I4	xor	10	r1	int_alu	21	1

**Ciclo 13** **add** y **xor** se pueden retirar ya al encontrarse en la cola (completan el procesamiento)

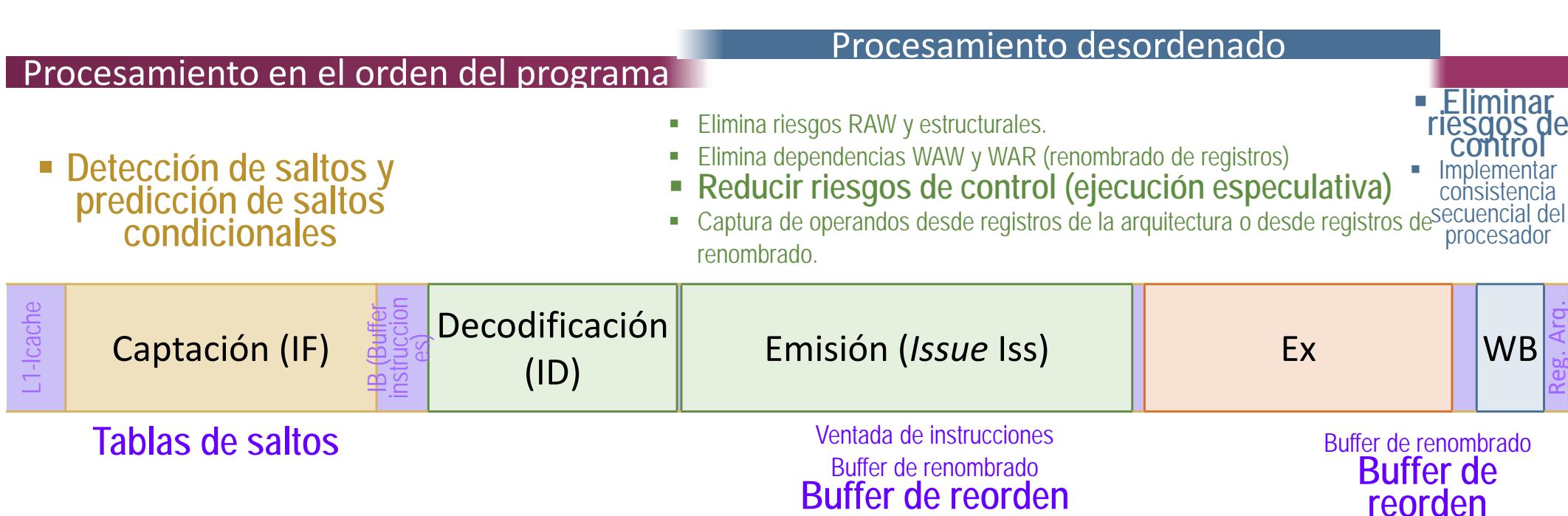
#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca
5	add	9	r1	int_add	17	1	f
6	xor	10	r1	int_alu	21	1	f

- Se ha supuesto que se pueden retirar (completar) ***dos instrucciones por ciclo.***
- Tras finalizar las instrucciones **mult** y **st**, se pueden retirar en el ciclo siguiente. **st** se considera finalizada cuando tiene todos los operandos, que es cuando puede pasar al buffer de escrituras, ya que no modifica los registros de la arquitectura.
- Despues, se retiraran las instrucciones **add** y **xor**.

cauce

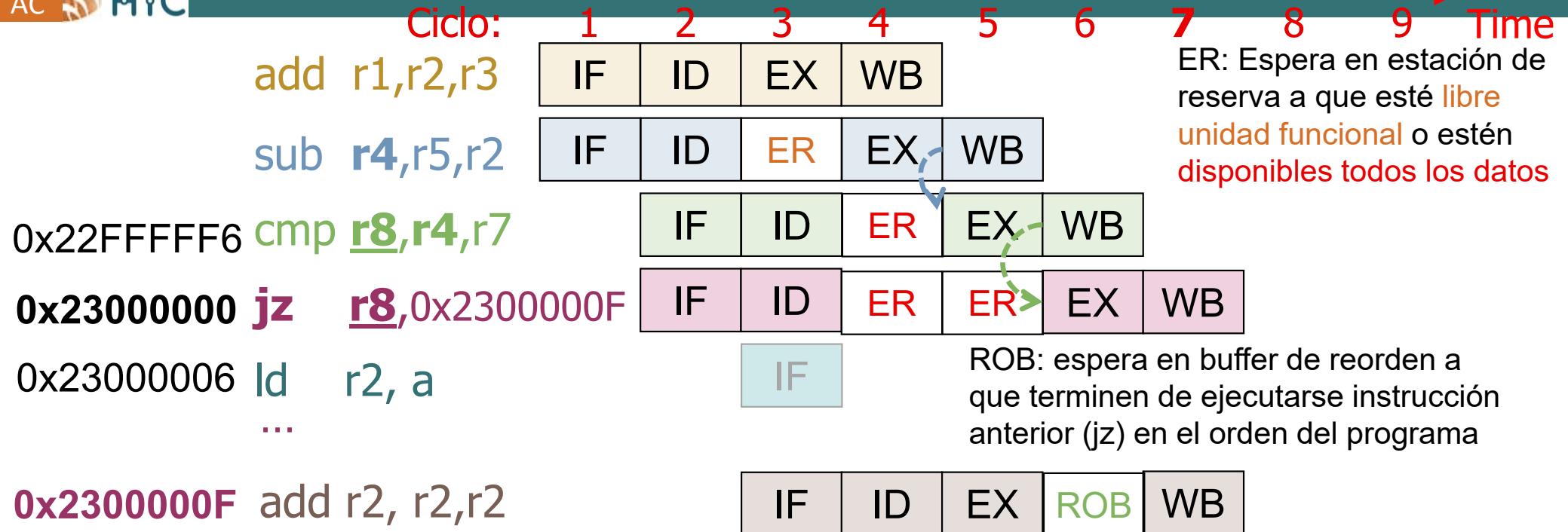
# Apartados

- Microarquitectura ILP superescalar
  - Cauce superescalar
  - Emisión (Algoritmo de Tomasulo)
  - Consistencia del procesador y buffer de reorden
  - **Procesamiento de saltos**



# Tabla de saltos (BTC: Branch Target Cache)

AC ATC



CPI ideal = 0.5 c/i

Ciclos real = 4 c + 3 c = 7 c

CPI = 3/4 = 0.75 c/s

Se elimina el riesgo de control (se ha supuesto que r8 va a contener un 0)

Dirección instr. salto	Dirección objetivo salto	Historial (3 bits)
0x23000000	0x2300000F	101
0x2300AB00	0x2300AB14	100
BTC/Branch Target Cache, BTAC/Branch Target Address Cache		

# Predicción de saltos

Predicción Fija: se toma siempre la misma decisión  
“Saltar” o “No Saltar”

Predicción Verdadera

- **Predicción Estática:** la predicción depende de atributos de la instrucción de salto: el código de operación (jL, jZ, jG, jGE, jLE) , el desplazamiento (salto hacia atrás o negativo, salto hacia adelante o positivo), la decisión del compilador)
- **Predicción Dinámica:** la predicción depende del resultado de ejecuciones pasadas de la instrucción (historia de la instrucción de salto)
  - **Implícita:** se almacena la dirección de la última instrucción ejecutada
  - **Explícita:** se almacena un estado que depende de ejecuciones anteriores

↓  
Prestaciones

# Predictión estática según desplazamiento

jg .L6 “Saltar”

jle .L7 “No saltar”

.L6:

```
movsd (%r12,%rax,8), %xmm0  
mulsd %xmm1, %xmm0  
addsd (%r13,%rax,8), %xmm0  
movsd %xmm0, (%r13,%rax,8)  
addq $1, %rax  
cmpl %eax, %ebp  
jg .L6
```

.L6:

```
addq $1, %rax  
cmpl %eax, %ebp  
jle .L7  
movsd (%r12,%rax,8), %xmm0  
mulsd %xmm1, %xmm0  
addsd (%r13,%rax,8), %xmm0  
movsd %xmm0, (%r13,%rax,8)  
jmp .L6
```

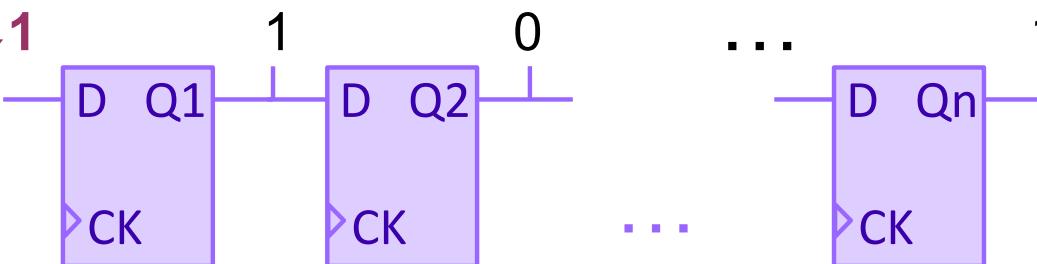
.L7:

# Predicción Dinámica de Saltos explícita.

## Nº impar de bits en historial



Si SÍ se salta  
introduce 1 en  
el registro de  
desplazamiento

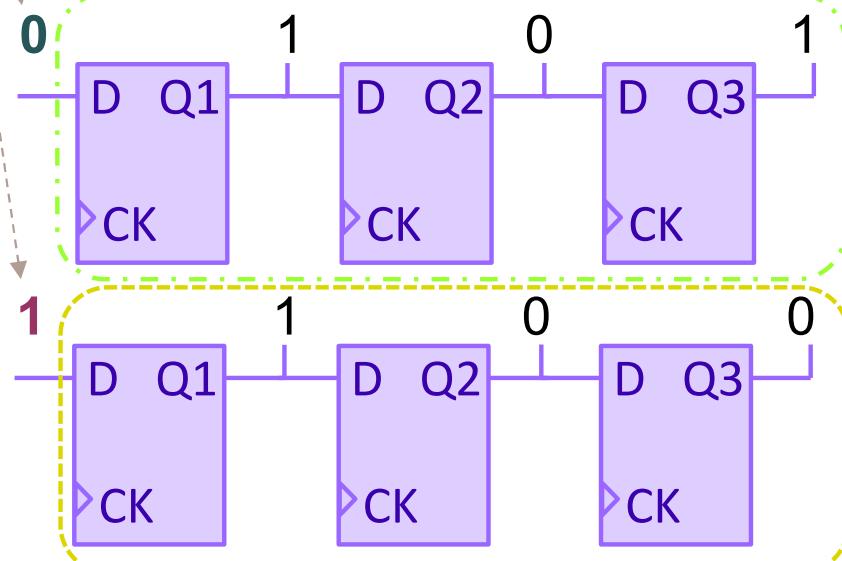


Un 1 en Qi indica que  
se saltó hace i  
ejecuciones del salto  
Un 0 indica que no se  
saltó

¿Cuál debe ser la predicción si hay mayoría de unos?  
¿"Saltar" o "No Saltar"?

Resultado de la ejecución del salto a  
introducir en el registro de desplazamiento

Si NO se  
salta,  
introduce  
0  
Si SÍ se  
salta,  
introduce  
1



Dirección instr. salto	Dirección objetivo salto	Historial (3 bits)
0x23000000	0x2300000F	101
0x2300AB00	0x2300AB14	100
BTC/Branch Target Cache, BTAC/Branch Target Address Cache		

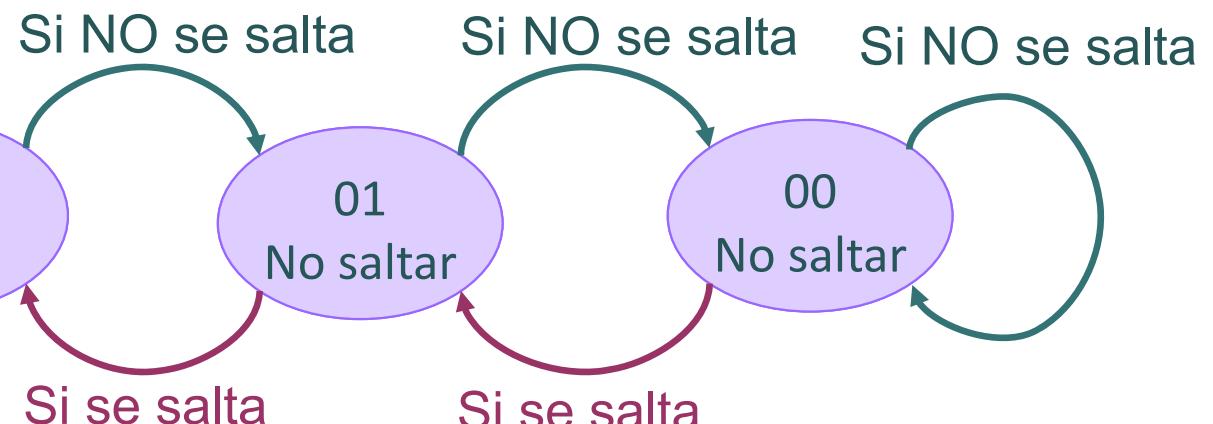
# Predictión Dinámica de Saltos explícita.

## 2 bits

Predicción “Saltar” para 11 o 10



Predicción “No saltar” para 01 o 00



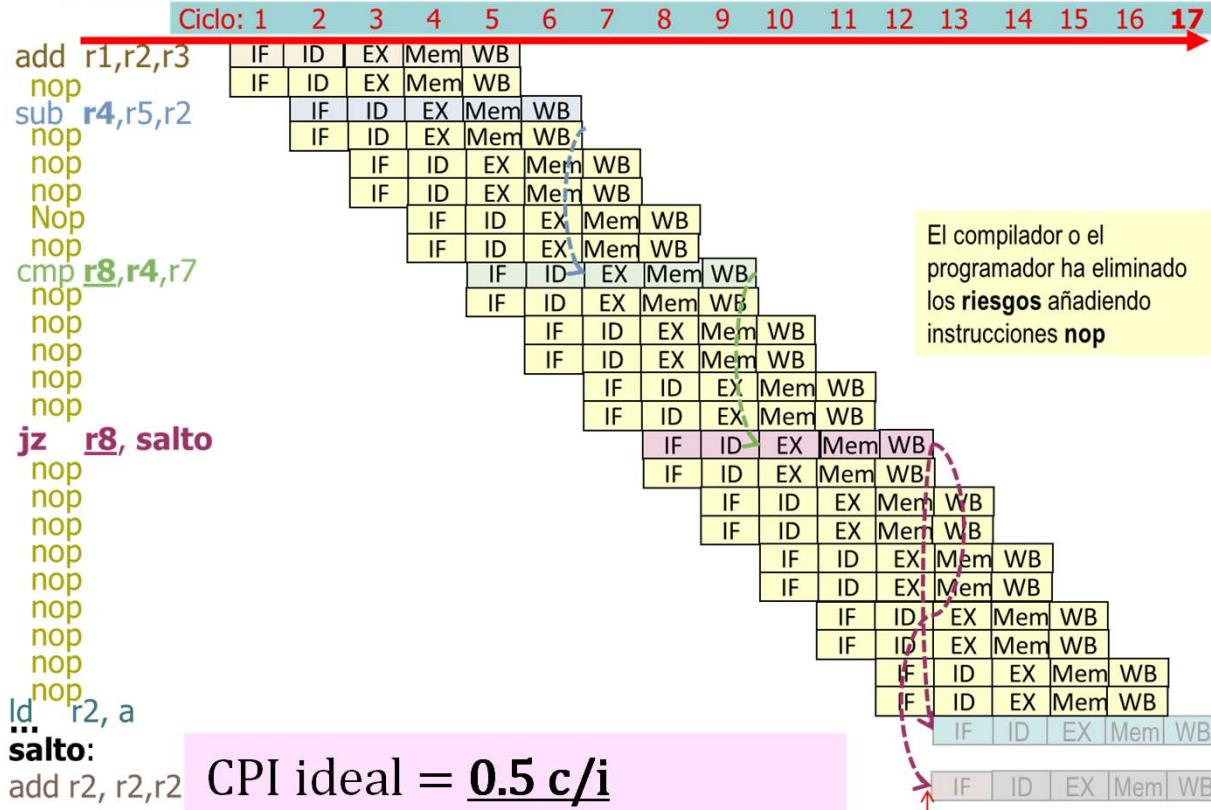
Dirección instr. salto	Dirección objetivo salto	Historial (2 bits)
0x23000000	0x2300000F	10
0x2300AB00	0x2300AB14	01

**BTC/Branch Target Cache,  
BTAC/Branch Target Address Cache**

# Ganancia con la extracción de paralelismo del hardware

AC ATC

Núcleo sin hardware para extraer paralelismo a nivel de instrucción

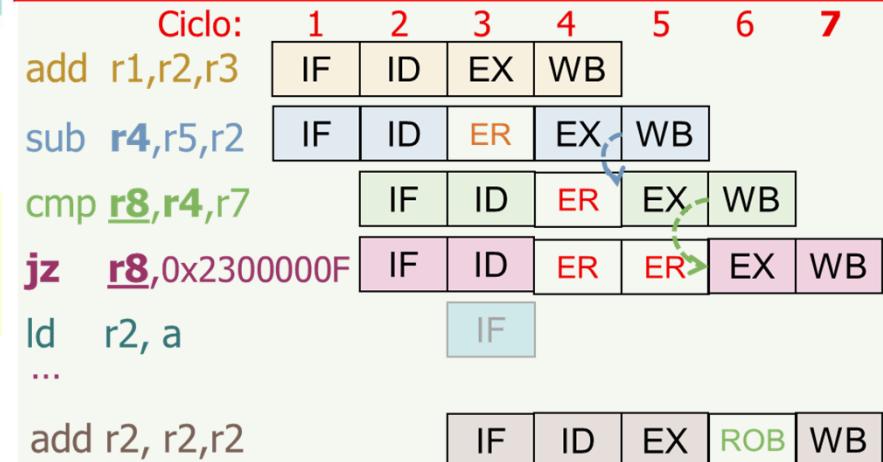


$$\text{Ciclos real} = 5c + 12c = 17c$$

$$\text{Ciclos real} = 5c + \text{CPI} \times (5-1) i = 17c$$

$$\text{CPI} = 12 c / 4 i = \underline{\underline{3 c/i}}$$

Núcleo con hardware para extraer paralelismo (núcleo superescalar)



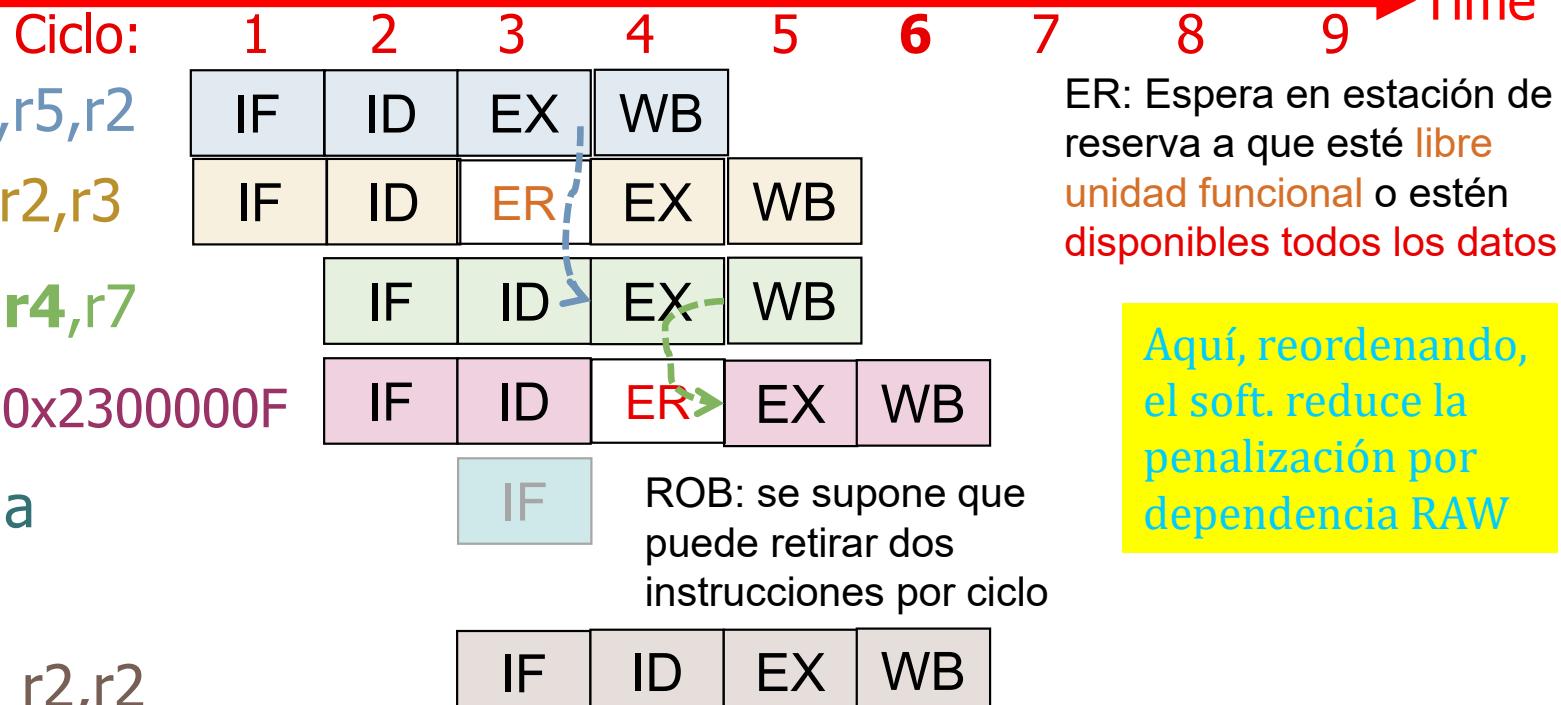
$$\text{Ciclos real} = 4 c + 3 c = 7 c$$

$$\text{CPI} = 3/4 = \underline{\underline{0.75 c/s}}$$

# ¿Y el software puede hacer algo?

## Reordenar código

Intercambio  
sub r4,r5,r2  
add r1,r2,r3  
cmp r8,r4,r7  
jz r8,0x2300000F  
ld r2, a  
...  
0x2300000F add r2, r2,r2



CPI ideal = 0.5 c/i

Ciclos ideal =  $5c + 2c = 7 c$

Ciclos real =  $4c + 2c = 6 c$

Ciclos real =  $4c + CPI \times 4 i = 6c$

CPI =  $2 c / 4 i = 0.5 c/i$

$$S=3/0.5=6$$

¿Y el software puede hacer algo?

- ✓ Puede extraer paralelismo, *eliminando dependencias* (de datos, control o estructurales), o *reduciendo o eliminando su penalización en tiempo* -> **beneficioso para Superescalares, VLIW**
- ✓ Puede evitar que las dependencias lleven a un resultado de ejecución incorrecto (eliminar riesgos) agrupando en palabras de instrucciones aquellas que se pueden emitir en paralelo por ser independientes y necesitar UF distintas, y añadiendo instrucciones de no operación, nop, (*planificación estática de instrucciones*) -> **necesario para VLIW**

# ¿Y el software puede hacer algo?

## Instrucciones de Ejecución Condicional

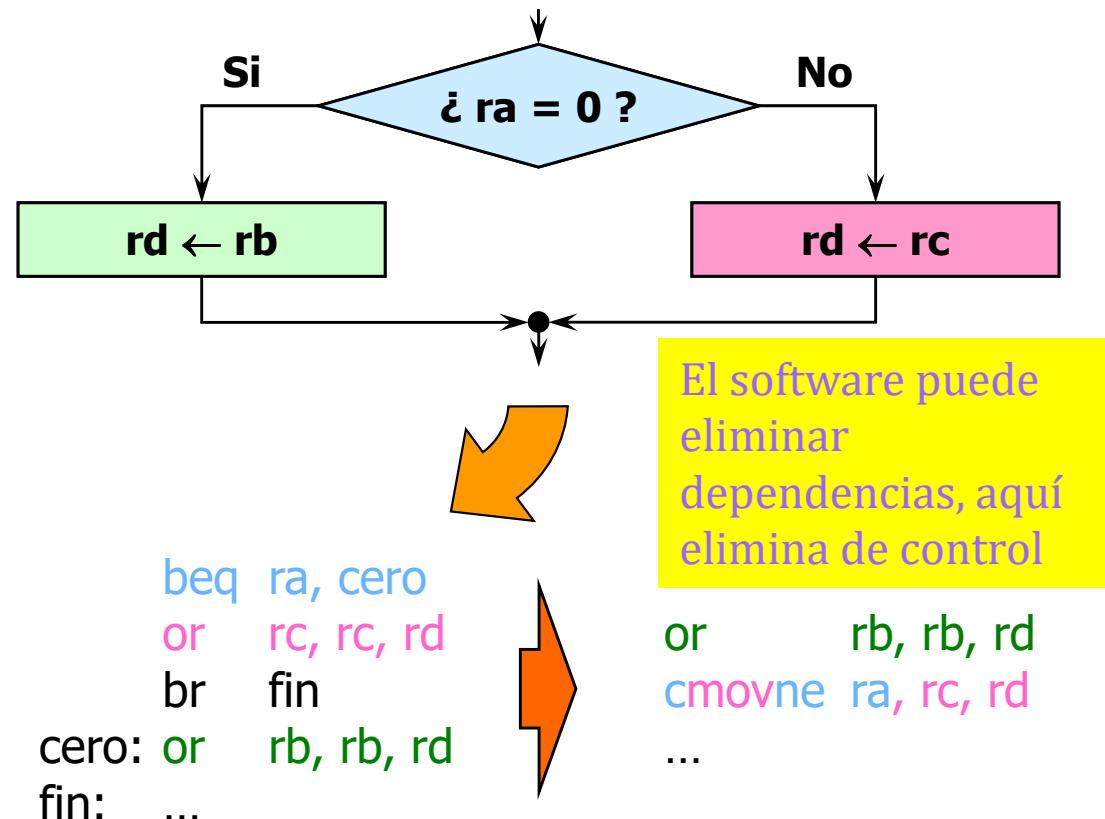
- Se pretende **reducir el número de instrucciones de salto** incluyendo en el **repertorio máquina instrucciones con operaciones condicionales** ('conditional operate instructions' o 'guarded instructions')
- Estas instrucciones tienen dos partes: la **condición** (denominada *guardia*) y la parte de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

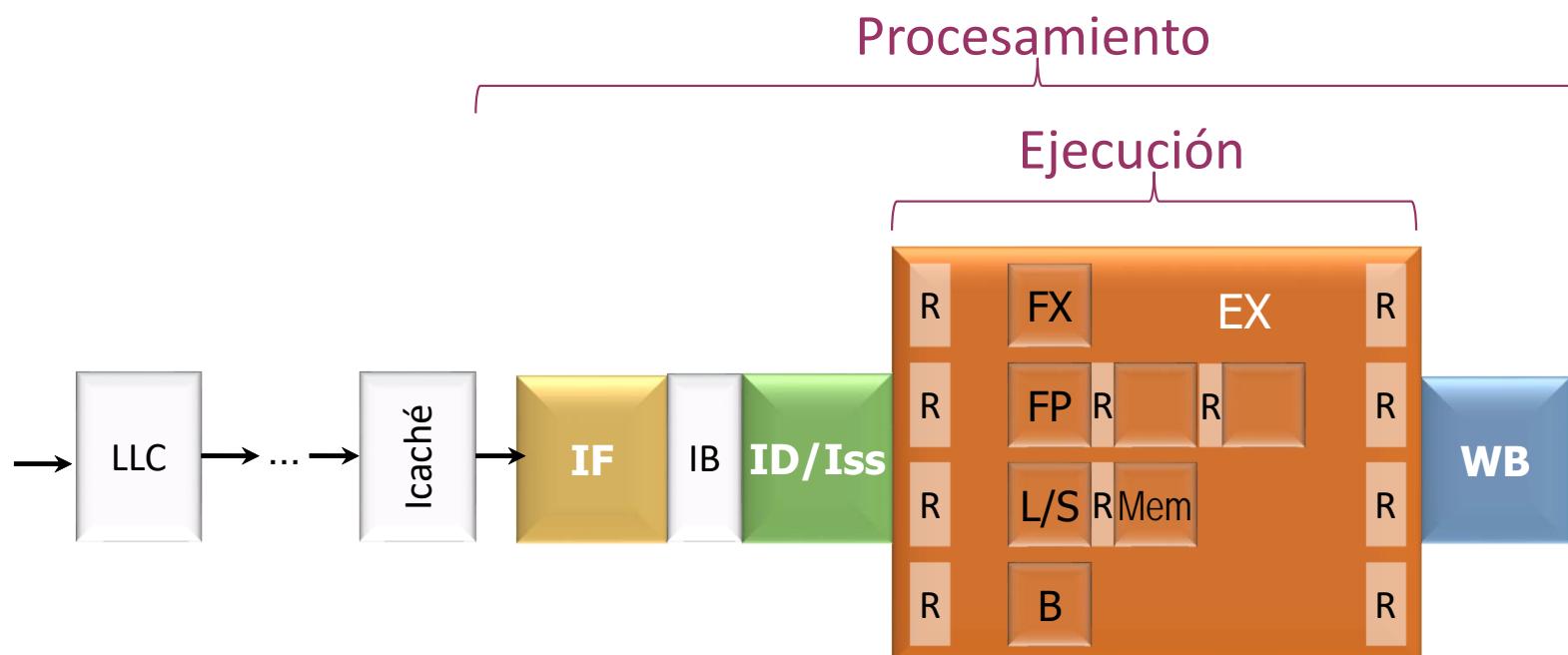


# Apartados

## ➤ Microarquitectura ILP superescalar

## ➤ Microarquitectura ILP VLIW

El software para un núcleo VLIW debe planificar la ejecución de las instrucciones, eliminando los **riesgos** provocados por **dependencias** y reduciendo la penalización que suponen las dependencias (**planificación estática**)



# Características generales de los procesadores VLIW (Very Large Inst. Word) II

