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:
Mancia Anguita, Julio Ortega

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

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)

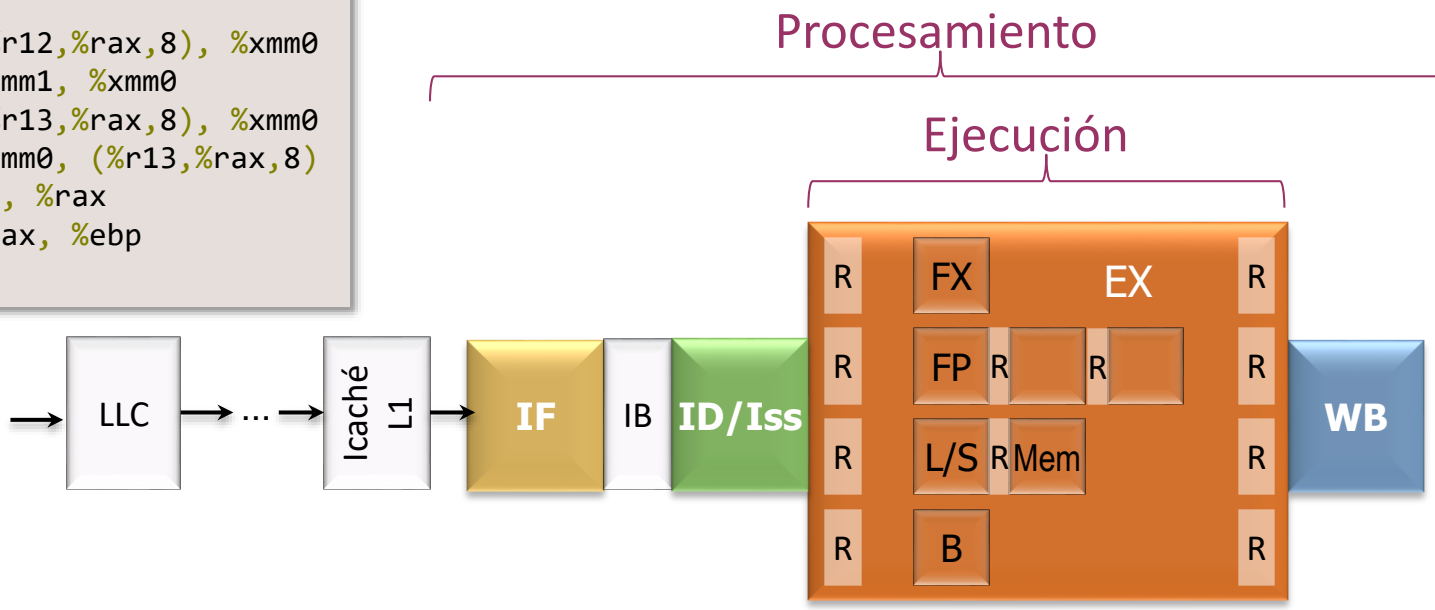
Arq. con **DLP** (*Data Level Parallelism*)	Arq. con **ILP** (*Instruction Level Parallelism*) Temas[1,4], BP4	Arq. con **TLP** (*Thread Level Parallelism*) explícito y **una** instancia de SO Temas[1-3], BP[0-3]		Arq. con **TLP** explícito y **múltiples** instancias SO
Ejecutan las ***operaciones*** de una instrucción **concurr.** o en **paralelo**	Ejecutan múltiples ***instrucciones*** **concurr.** o en **paralelo**	Ejecutan múltiples ***flujos de instruciones*** **concurr.** o en **paralelo**		Ejec. múltiples ***flujos de instr.*** en **paralelo**
Unidades funcionales vectoriales o SIMD (90)	**Cores** escalares segmentados (80), superescalares (90) o VLIW (90)	**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)**)	**Multi-computadores (85)**: ejecutan threads en *paralelo* en un sistema con múltiples computadores

Apartados Tema 4

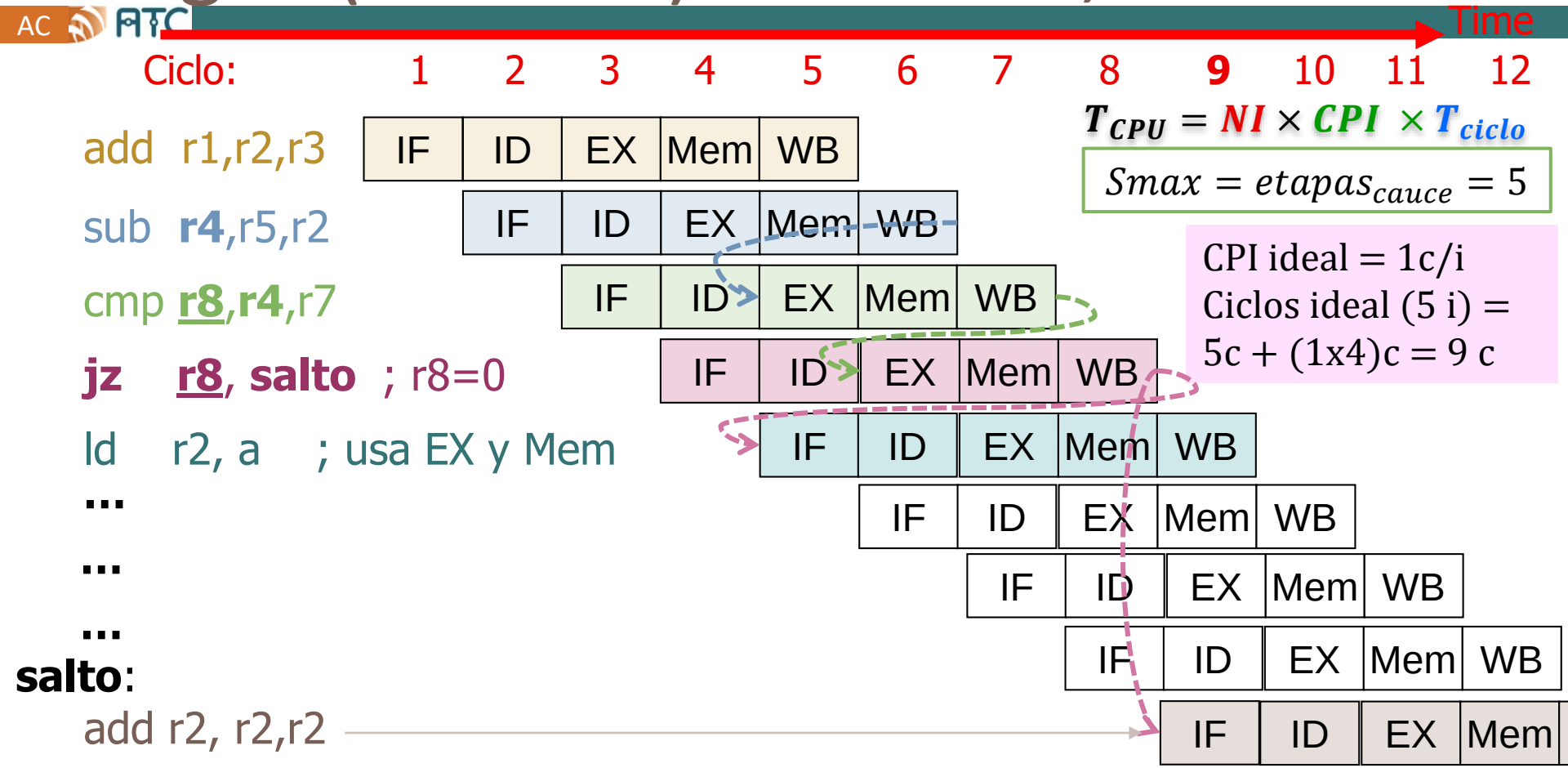
- Lección 11. Microarquitecturas ILP. Cauces Superescalares
- Lección 12. Consistencia del procesador y Procesamiento de Saltos
- Lección 13. Procesamiento VLIW

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

```
.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
```



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



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

$$Smax = etapas_{cauce} = 5$$

CPI ideal = 1c/i
Ciclos ideal (5 i) = 5c + (1x4)c = 9 c

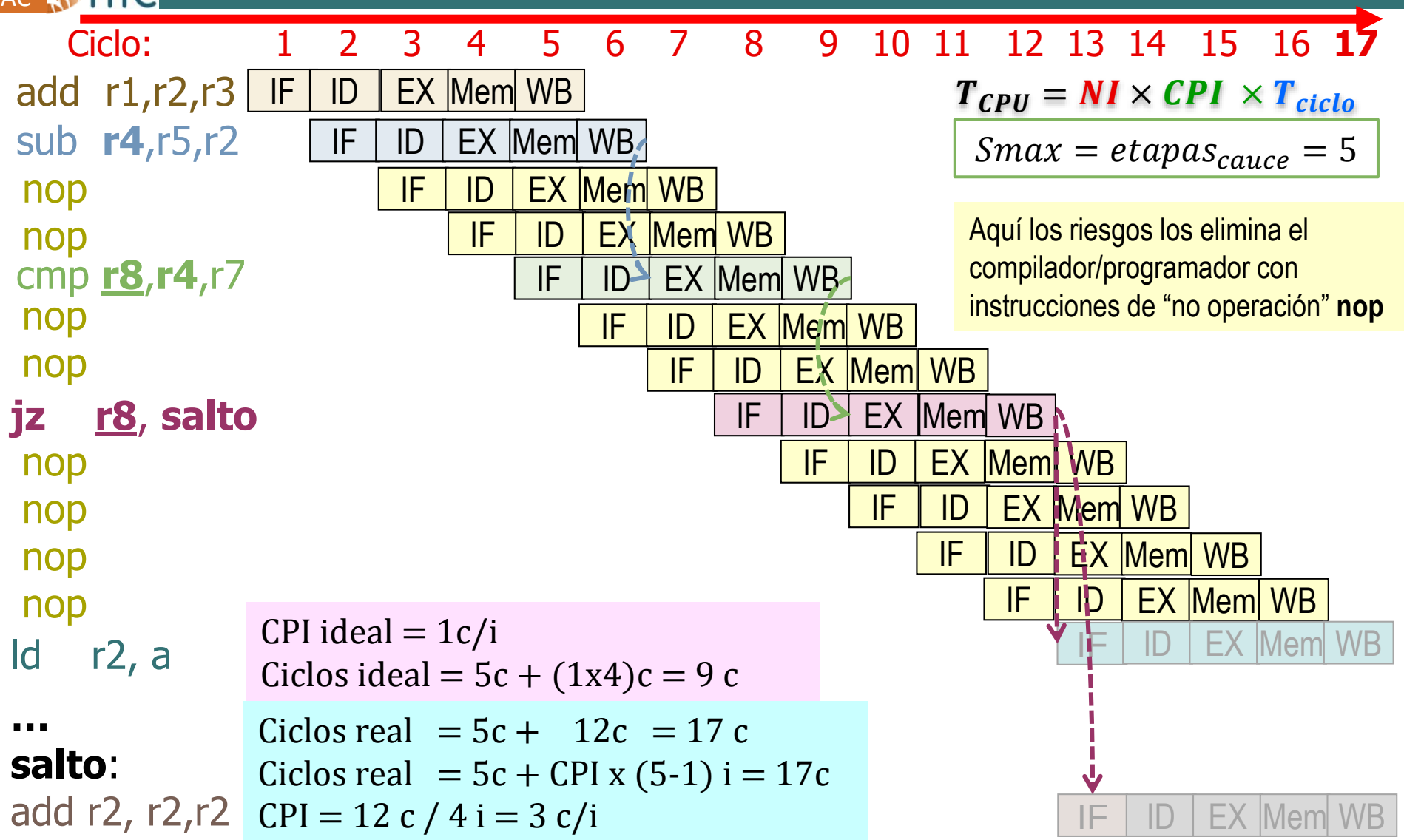
La **dependencia** es una propiedad del código.

3 dependencias: 2 RAW y 1 de control

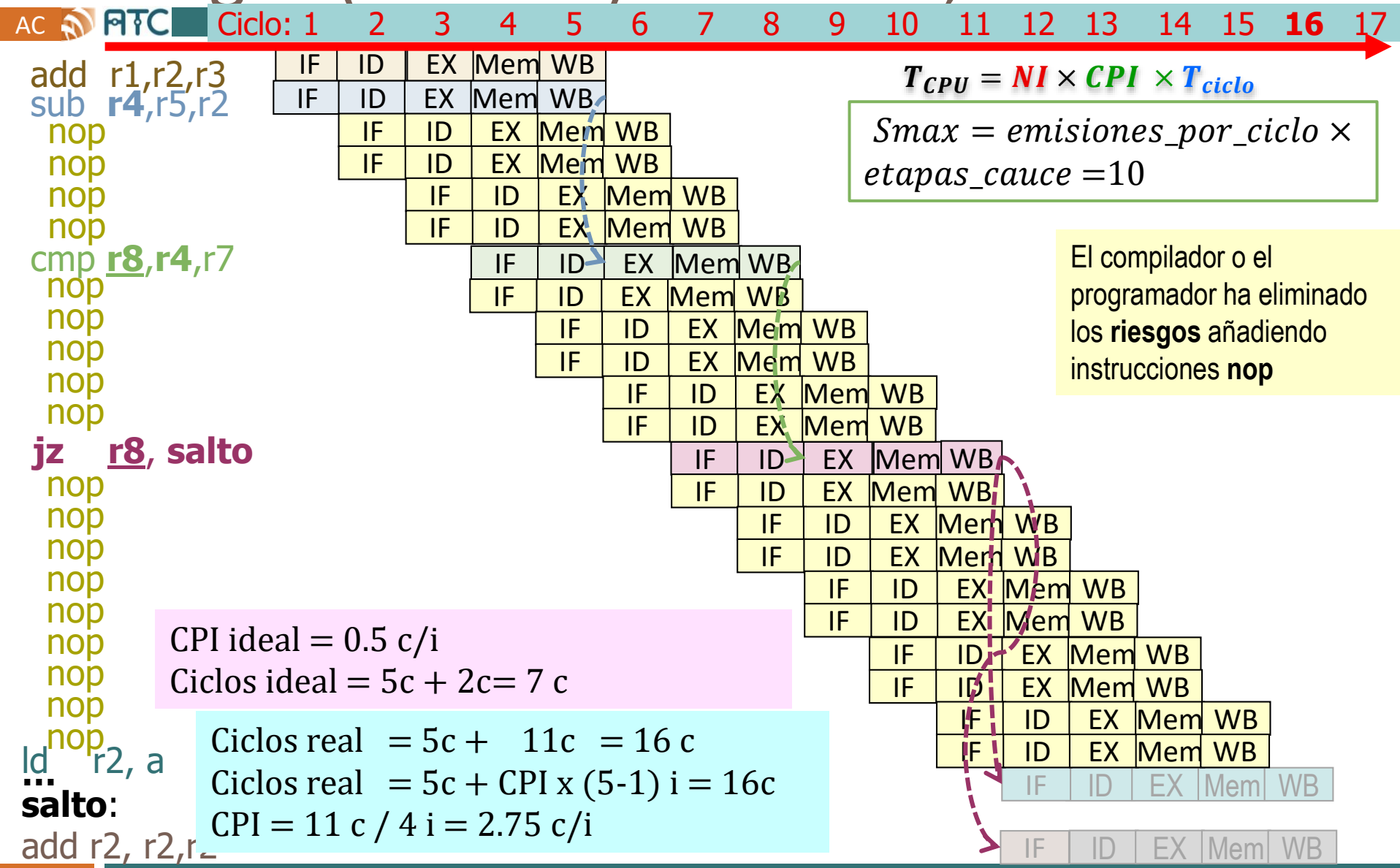
Las **dependencias** pueden provocar **riesgos** o conflictos (*hazards*) en un cauce segmentado que pueden llevar a **resultados incorrectos** (distintos a los que obtendría una ejecución secuencial de las instruc.

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

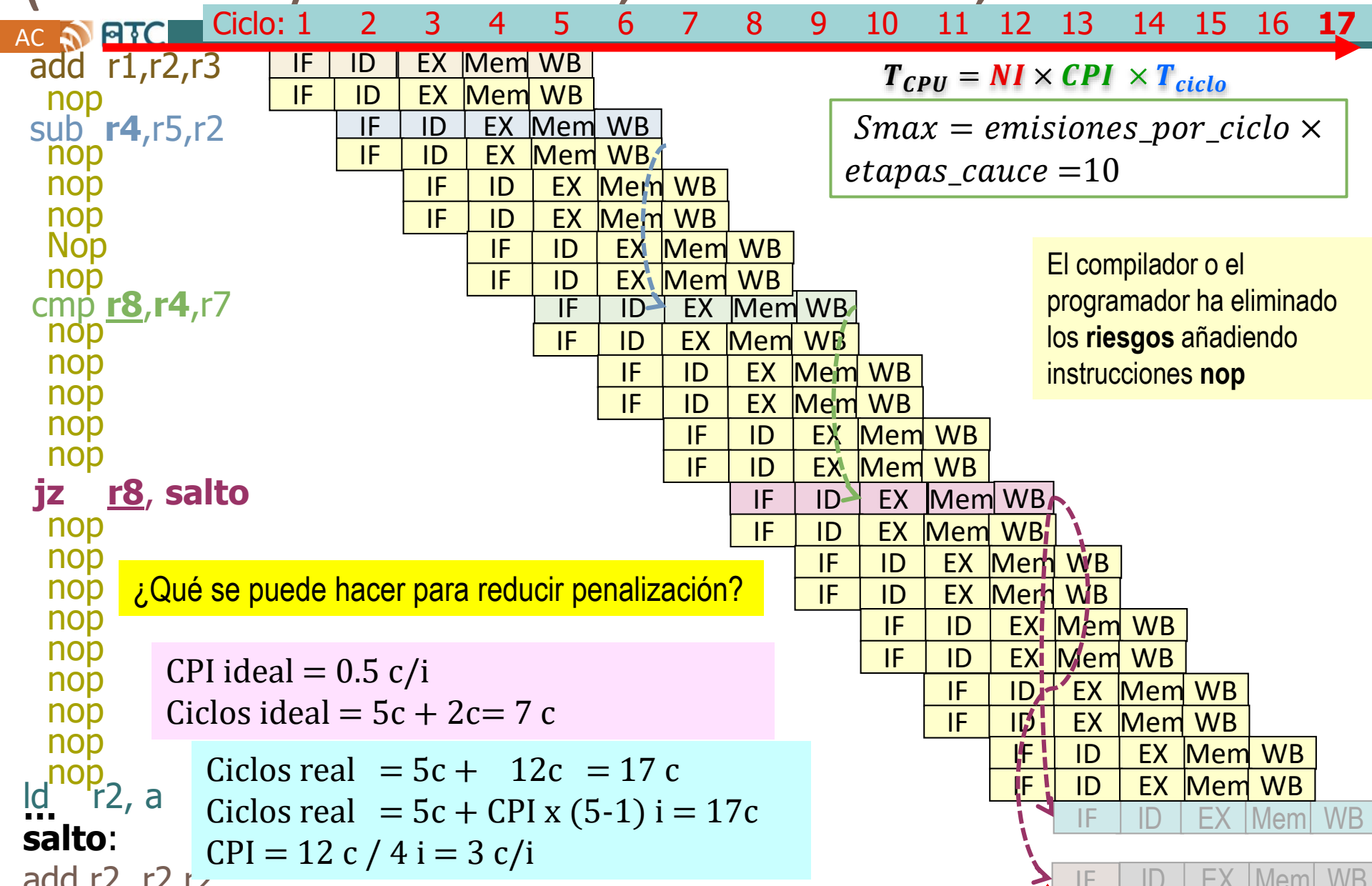
Ejecución concurrente de instr. y riesgos (*hazards*): de **datos**, de **control**



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



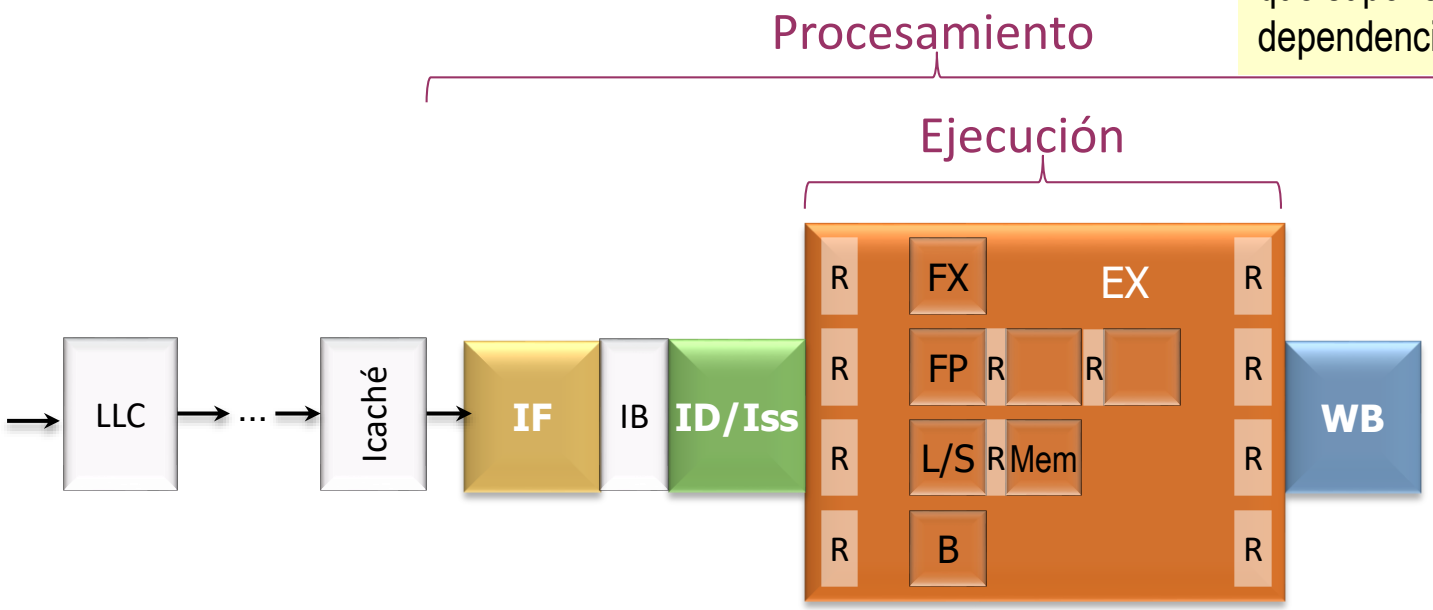
Ej. concurrente y paralela de intr. y riesgos (*hazards*): de **datos**, de **control**, **estructural**



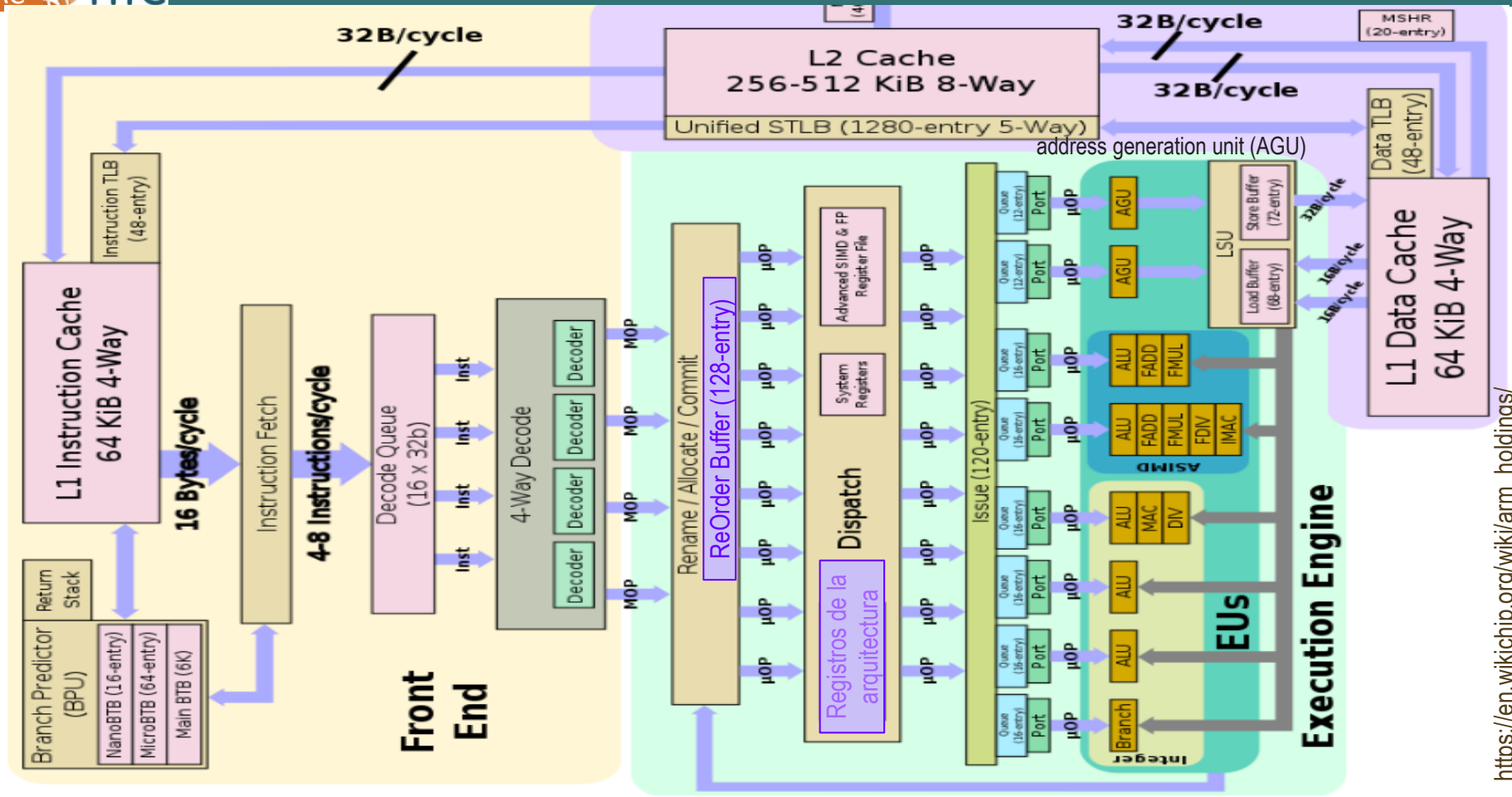
Apartados

- 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.



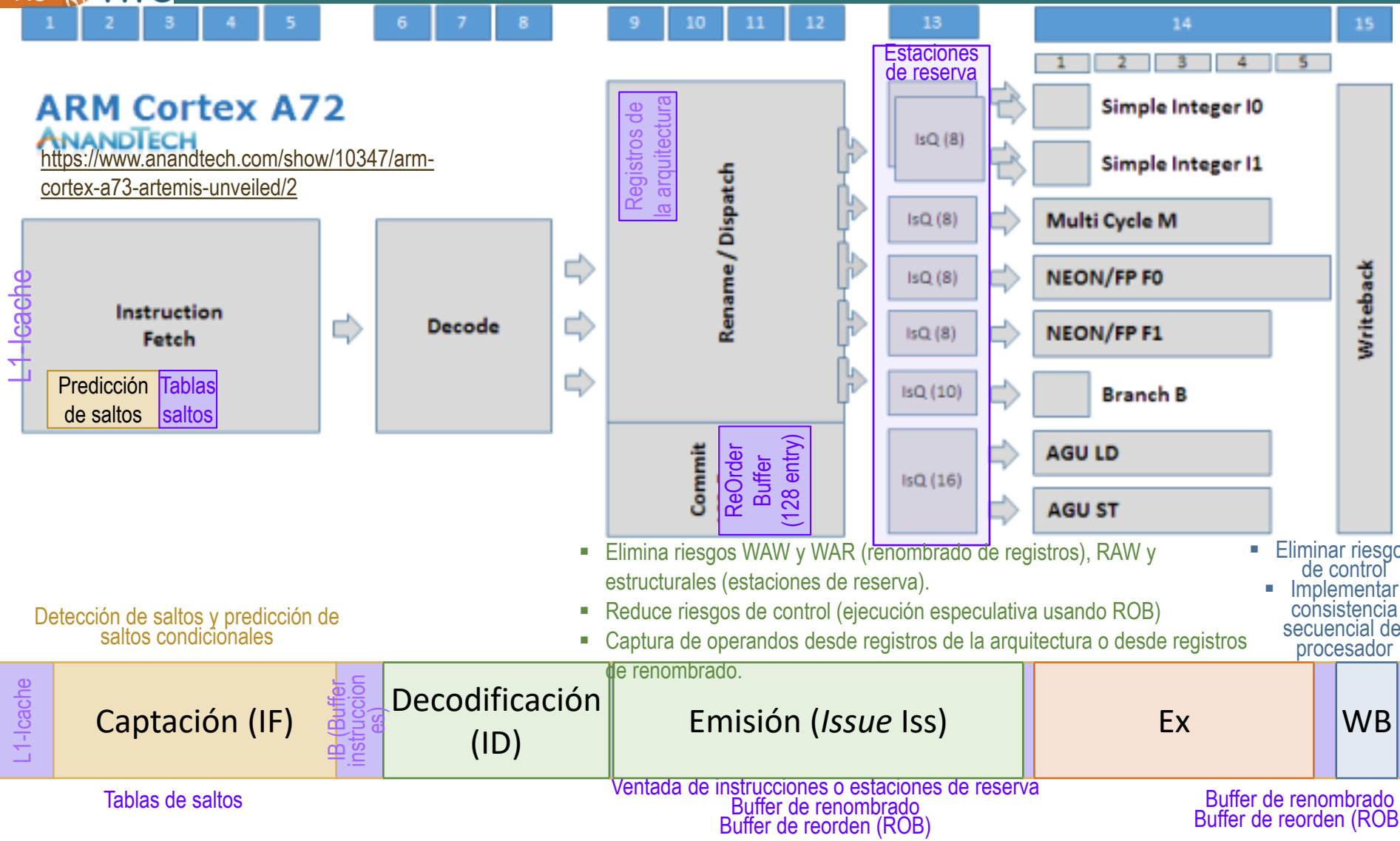
Ejemplo de Cauce (ARM cortex A76, microarquitectura de ARMv8)



https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a76

L1-Icache	Captación (IF)	IB	Decodificación (ID)	Emisión (*Issue* Iss) Planificación de instr.	Ex	WB	Reg. Arq.

Ejemplo de Cauce (ARM cortex A72)



Apartados

- Microarquitectura ILP superescalar
  - Cauce superescalar
  - **Emisión (Emisión +Envío a UF) (Issue⇔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

- **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

- Detección de saltos y predicción de saltos condicionales

Tablas de saltos

Ventada de instrucciones o estaciones de reserva
Buffer de renombrado
Buffer de reorden(ROB)

Buffer de renombrado
Buffer de reorden

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)**.**

Implementación Estática: Durante la Compilación

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

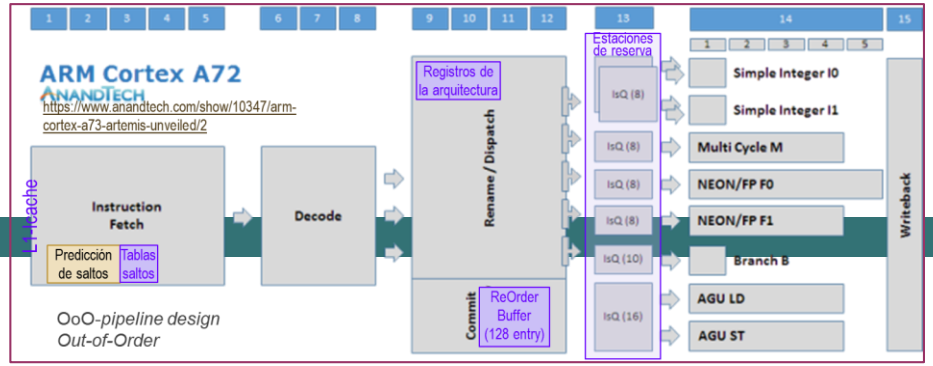
Características del fichero de registros (o *buffer*) de Renombrado

- **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

Algoritmo de Tomasulo I



Emisión de la multiplicación

RAW

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

WAW

Cola de Instrucciones

Emisión ordenada

1

2

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			1
2	0				
3	0				
4	0				

Buffer de Renombrado

2

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

Estaciones de Reserva

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

Envío desordenada

mult

(resul.,rr)

espiar (*snoop*)Z

espiar (*snoop*)

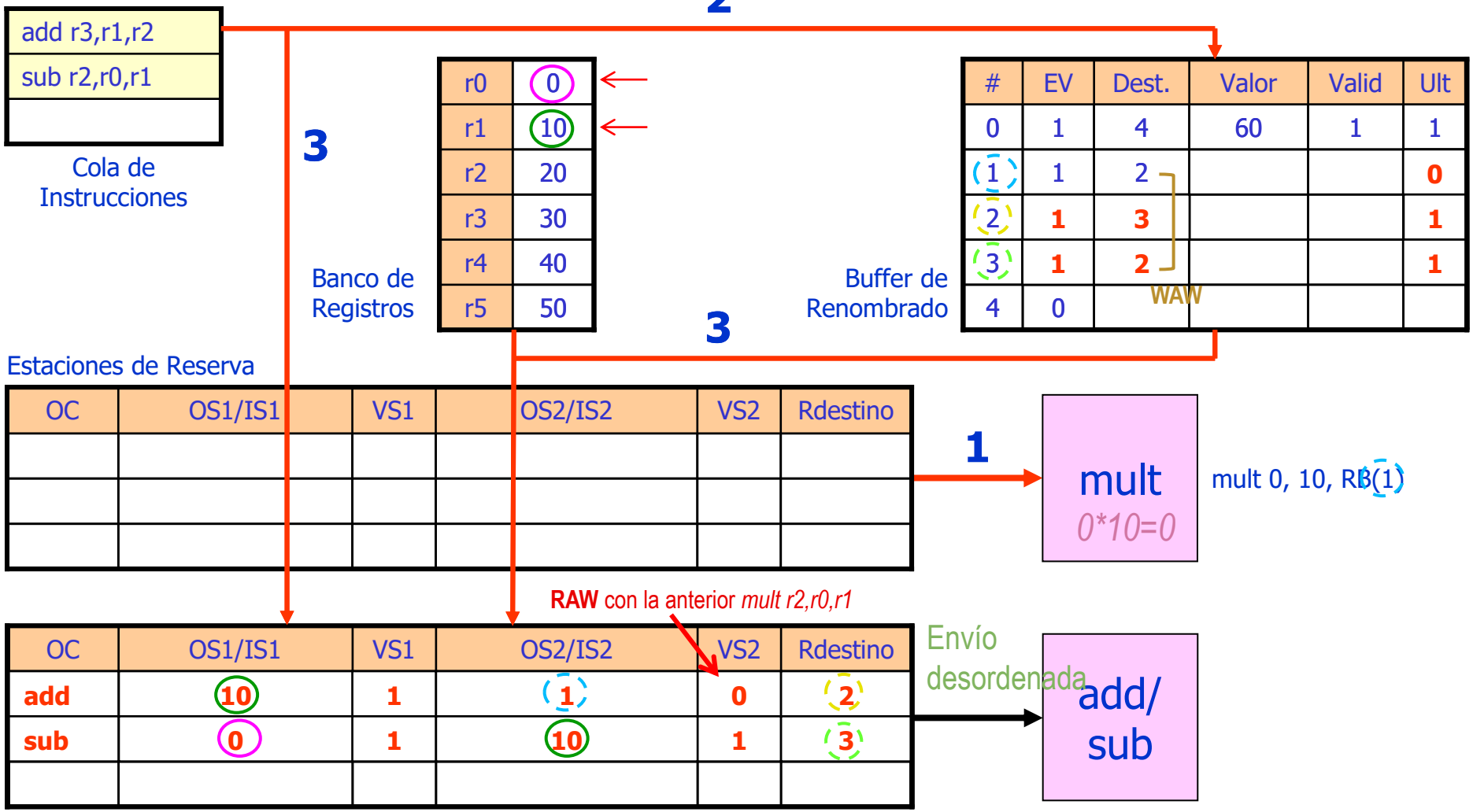
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

add/ sub

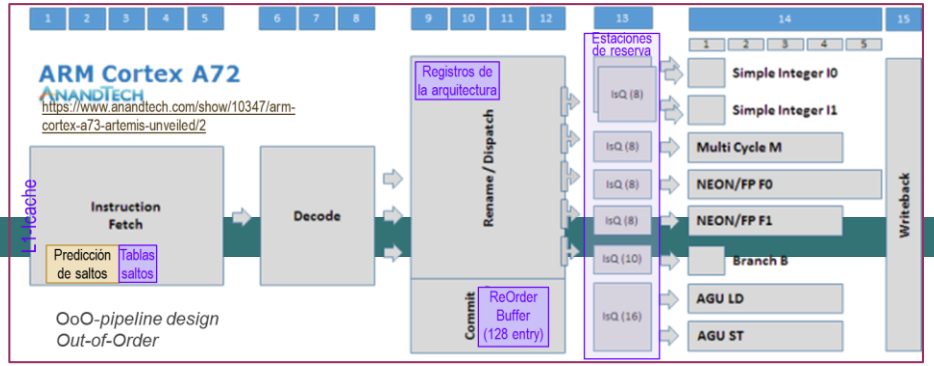
CDB (Common Data Bus)

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

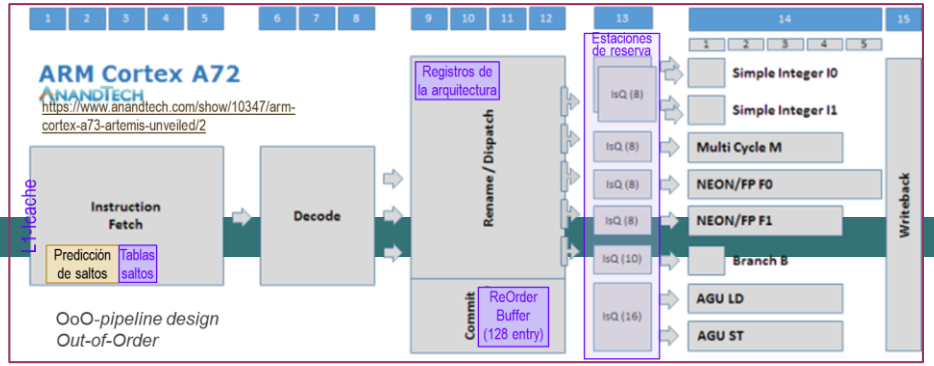
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

→ mult *0*10=0* — mult 0, 10, RB(1)

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

1 → add/ sub *0-10=-10* — 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=0*

mult 0, 10, RB(1)

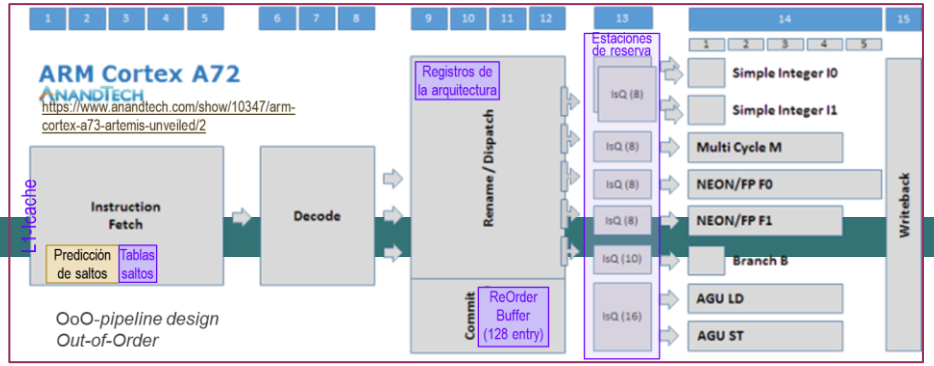
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2

add/ sub
0-10=-10

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
*0*10=0*

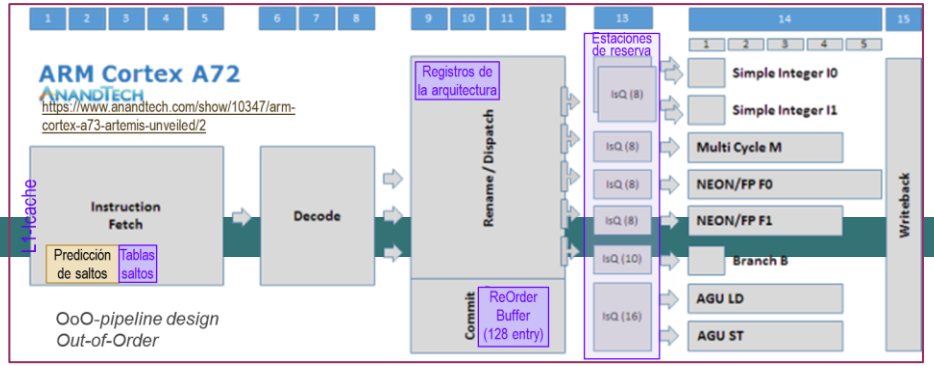
mult 0, 10, RB(1)

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

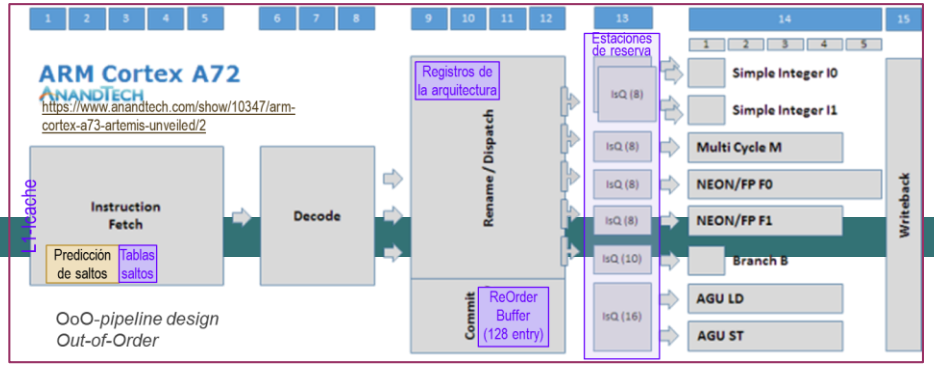
→ mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

1 → add/ sub *10+0=10*

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

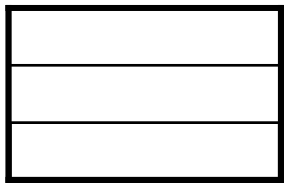
→ add/ sub *10+0=10*

add 10, 0, RB(2)

1

Algoritmo de Tomasulo VIII

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



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

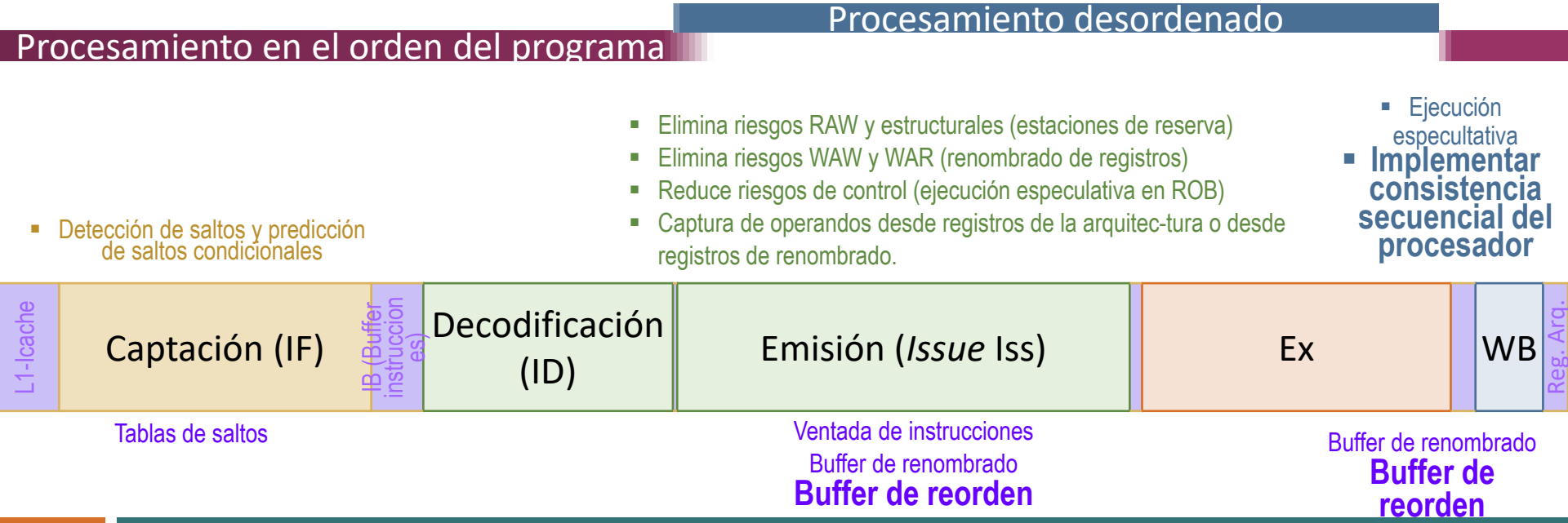
mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

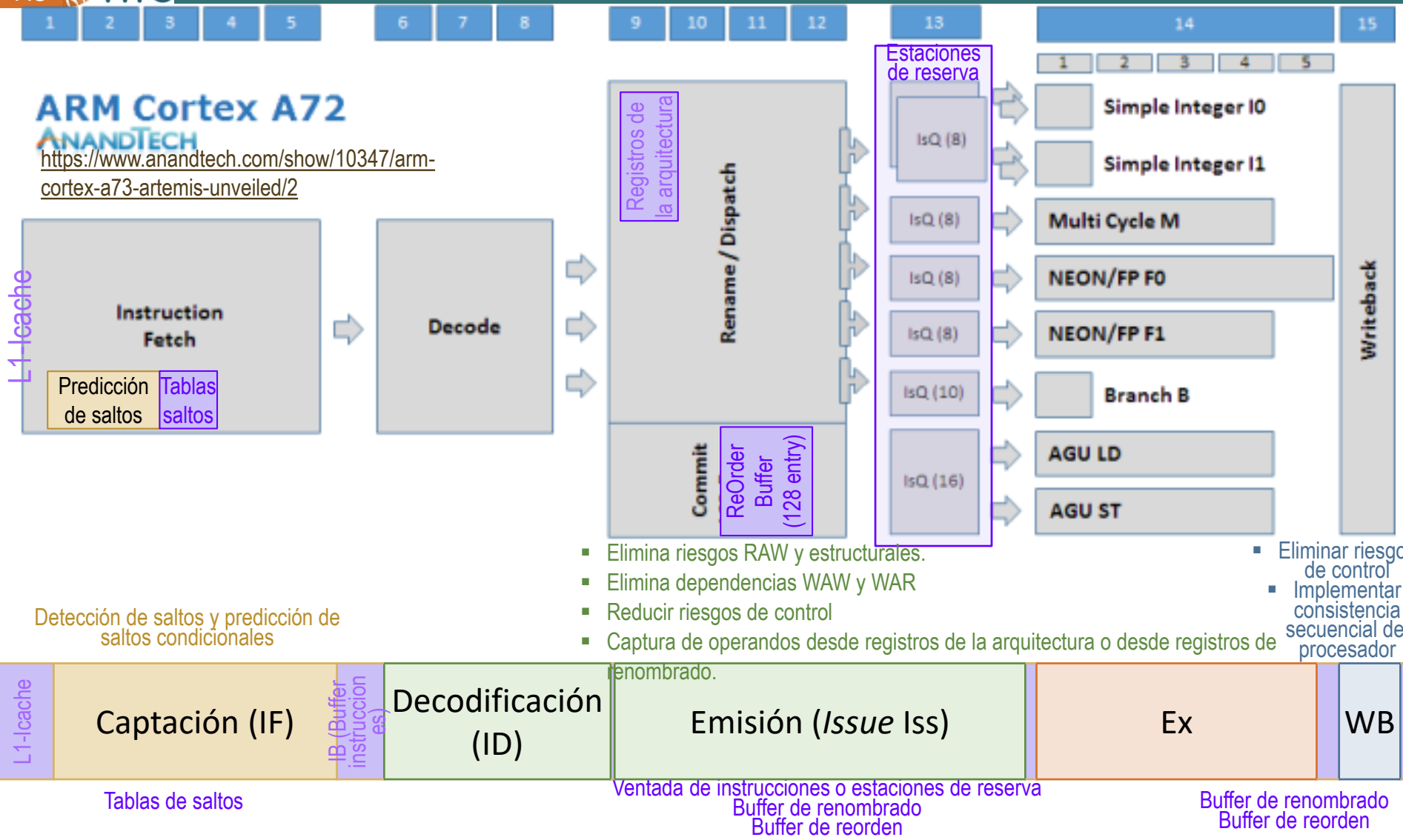
add/ sub

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)

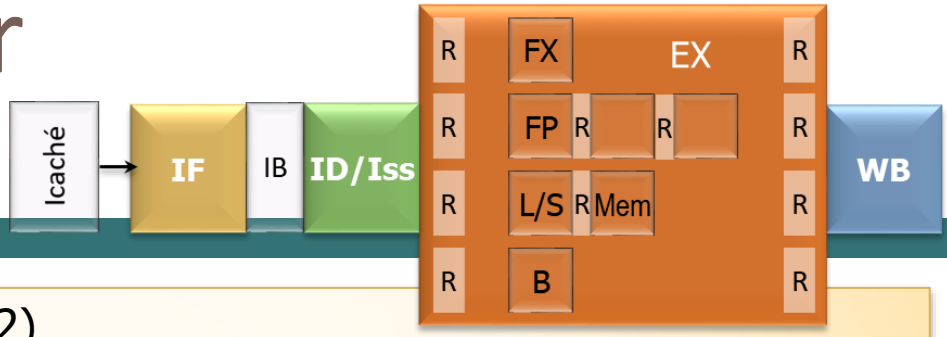


Consistencia

		cauce	Tendencia
Consistencia de Procesador Consistencia en el orden en que se completan las instrucciones	Débil/relajado: Las instrucciones se pueden completar desordenadamente siempre que no se vean afectadas las dependencias	Deben detectarse y resolverse las dependencias	Power1 (90) PowerPC 601 (93) Alpha R8000 (94) MC88110 (93)
	Fuerte: Las instrucciones deben completarse estrictamente en el orden en que están en el programa	Se consigue mediante el uso de ROB	PowerPC 620 PentiumPro (95) UltraSparc (95) K5 (95) R10000 (96)
Consistencia de Memoria Consistencia del orden de los accesos a memoria	Débil/relajado: Los accesos a memoria (Load/Stores) pueden realizarse desordenadamente siempre que no afecten a las dependencias	Deben **detectarse** y **resolverse** las dependencias de acceso a memoria	MC88110 (93) PowerPC 620 UltraSparc (95) R10000 (96)
	Fuerte: Los accesos a memoria deben realizarse estrictamente en el orden en que están en el programa	Se consigue mediante el uso del ROB	PowerPC 601 (93) E/S 9000 (92)

Tendencia / Prestaciones

Reordenar y extraer paralelismo




```
loop: ld   r1, 0x1C(r2)
      mul  r1, r1, r6
W(1C+r2) st   r1, 0x1C(r2)
R(2D+r2) ld   r3, 0x2D(r2)
      mul  r3, r3, r6
WAR   st   r3, 0x2D(r2)
      addi r2, r2, #1
      subi r4, r4, #1
      bnz  r4, loop
```

```
ld   r1, 0x1C(r2)
mul  r1, r1, r6
st   r1, 0x1C(r2)
```

```
ld   r3, 0x2D(r2)
mul  r3, r3, r6
st   r3, 0x2D(r2)
```

```
addi r2, r2, #1
subi r4, r4, #1
bnz  r4, loop
```

Se evita tener que esperar a las multiplicaciones y se pueden ir adelantando cálculos correspondientes al control del número de iteraciones (**suponiendo que hay renombrado en hardware**)

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

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

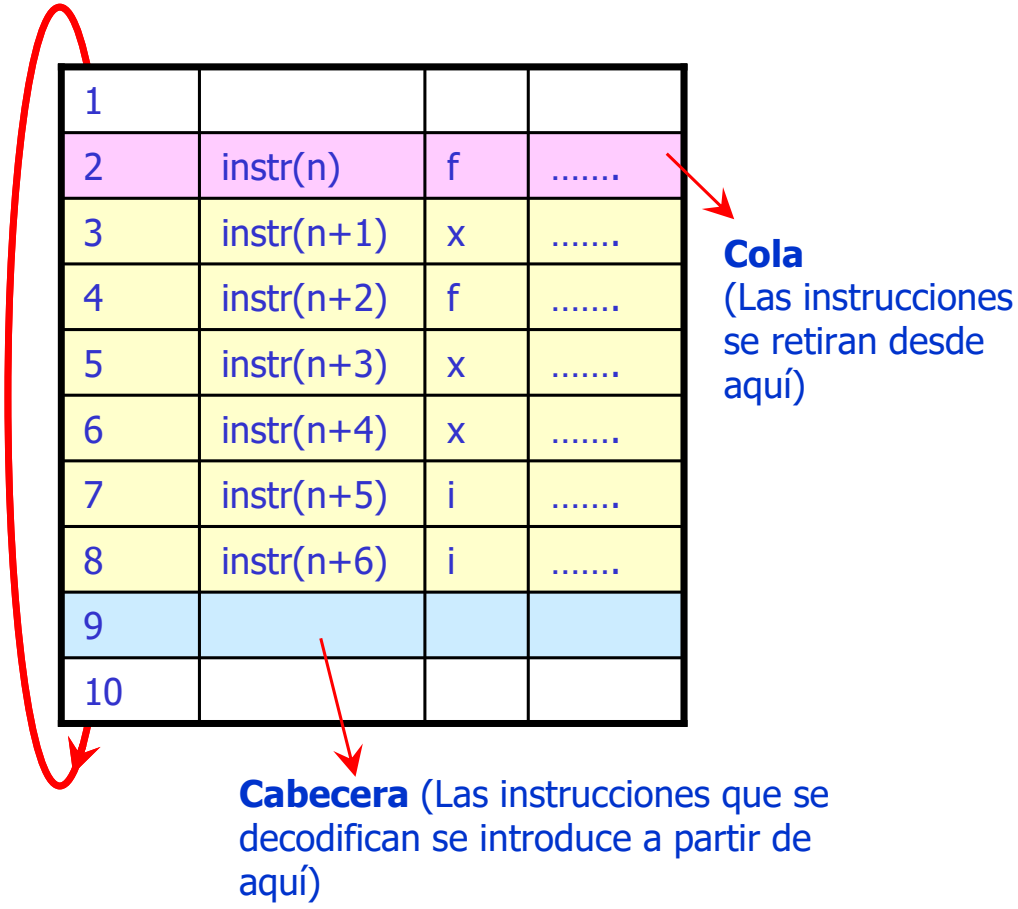
Las direcciones 0x1C(r2) y 0x2D(r7) podrían coincidir.

¿RAW?
¿ 0x1C(r2) = 0x2D(r7) ?

Se tendría un **load especulativo**

ARM

Buffer de Reordenamiento (ROB) I



La gestión de interrupciones y la ejecución especulativa se pueden implementar 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)**
- Una **instrucciones sólo se puede retirar** (y modificar los registros de la arquitectura) **si ha terminado su ejecución**, y todas las que les preceden también.
- La **consistencia secuencial 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.

Buffer de Reordenamiento (ROB) II

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

Ciclo 7 Situación en este ciclo

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

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

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**	**f**
4 I2	st	8	-	store	-	**1**	**f**
5 I3	add	9	r1	int_add	17	1	f
6 I4	xor	10	r1	int_alu	21	1	f

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.
- Después, se retirarán 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**

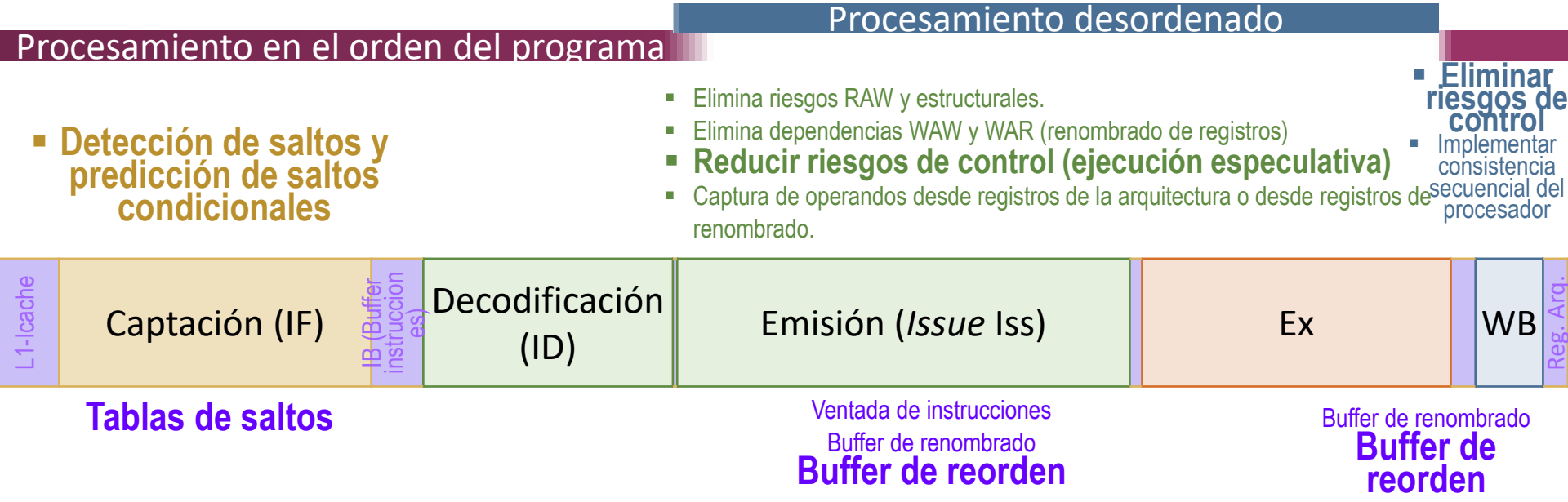
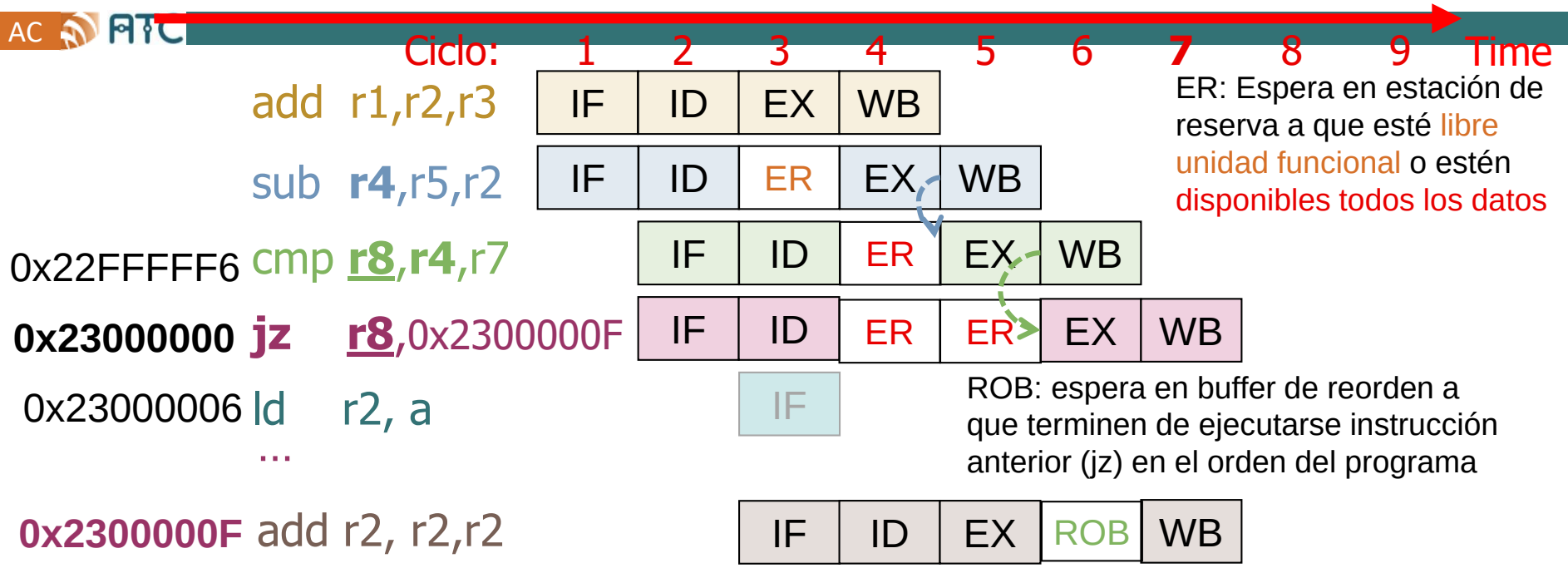


Tabla de saltos (BTC: *Branch Target Cache*)



ER: Espera en estación de reserva a que esté libre unidad funcional o estén disponibles todos los datos

ROB: espera en buffer de reorden a que terminen de ejecutarse instrucción anterior (jz) en el orden del programa

Dirección instr. salto	Dirección objetivo salto	Historial (3 bits)
0x23000000	**0x2300000F**	101
0x2300AB00	0x2300AB14	100

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

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 supone que r8 va a contener un 0)

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

Predicción estática

jg .L6 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
```

jle .L7 no saltar

```
.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

- 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*

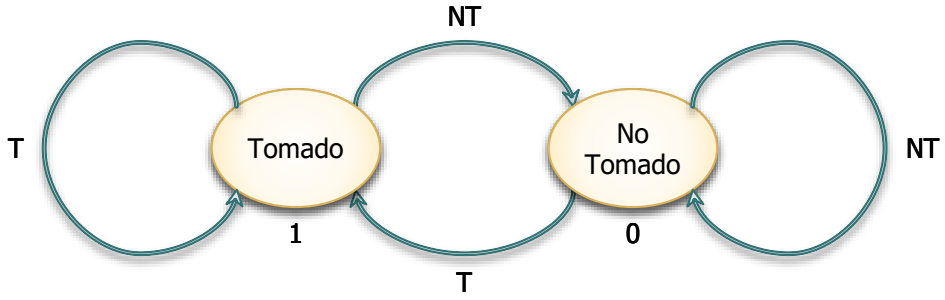
- **Predicción Dinámica Explícita**

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

Ejemplos de Procedimientos Explícitos de Predicción Dinámica de Saltos

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)

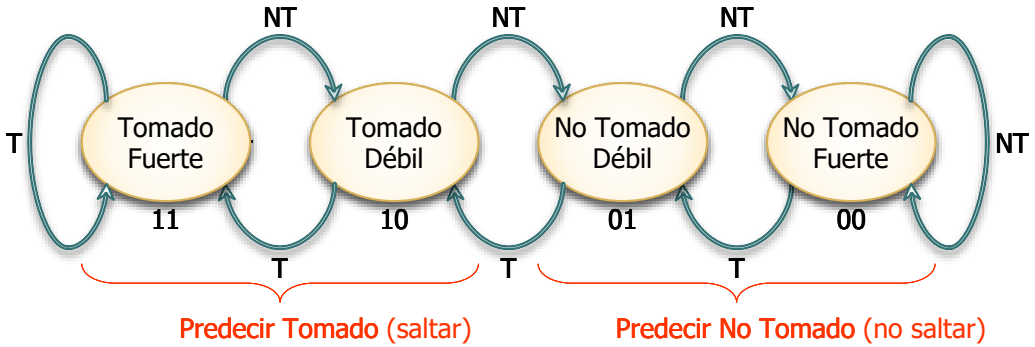


Predicción con 2 bits de historia

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 instrucci ón (T o NT)

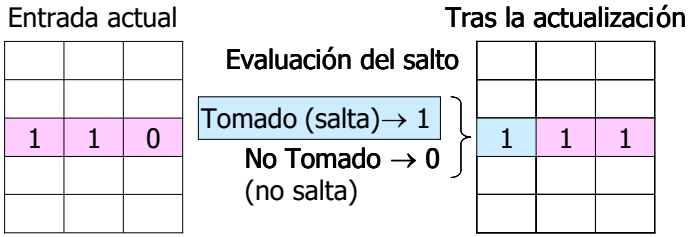


Predicción con 3 bits de historia

Cada entrada guarda las últimas ejecuciones del salto

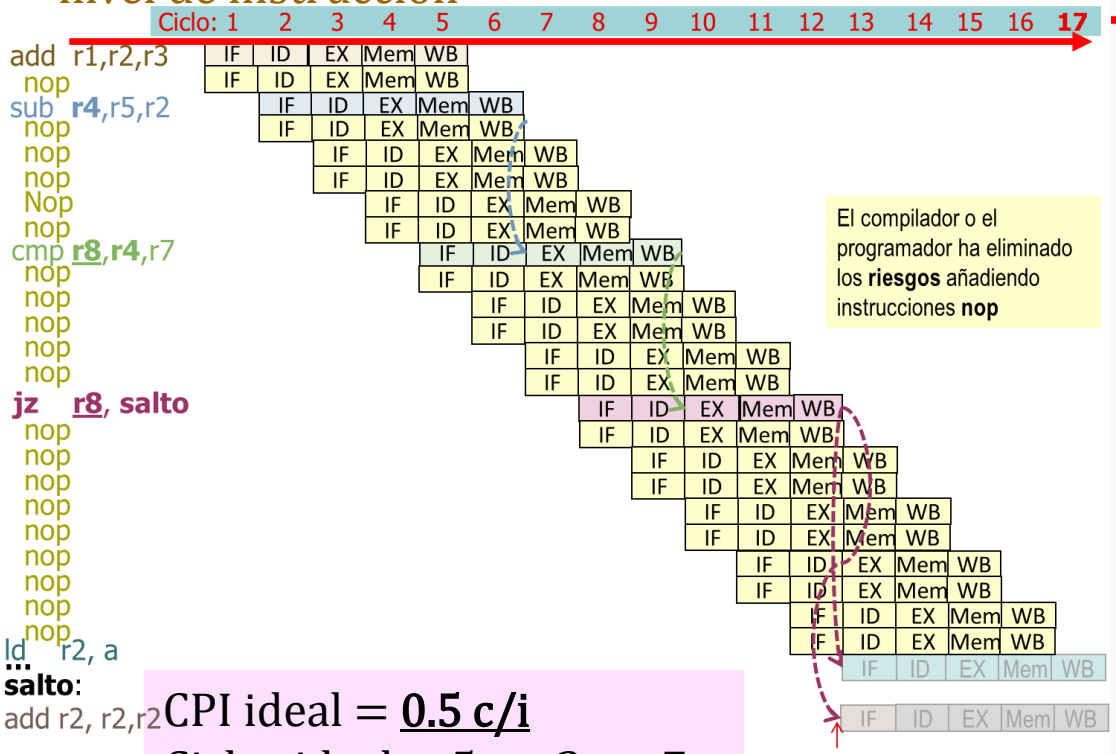
Se predice según el bit mayoritario (por ejemplo, si hay mayoría de unos en una entrada se predice salto)

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



Ganancia con la extracción de paralelismo del hardware

Núcleo sin hardware para extraer paralelismo a nivel de instrucción



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

CPI ideal = **0.5 c/i**

Ciclos ideal = 5c + 2c = 7 c

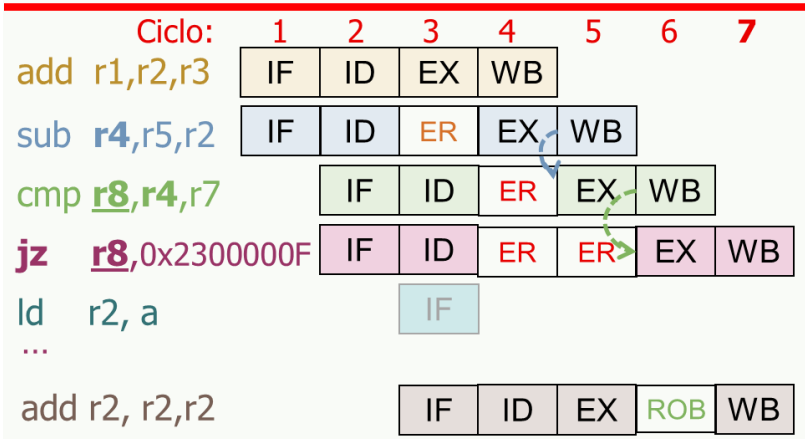
Ciclos real = 5c + 12c = 17 c

Ciclos real = 5c + CPI x (5-1) i = 17c

CPI = 12 c / 4 i = **3 c/i**

S=3/0.75=4

Núcleo **superescalar** o núcleo con hardware para extraer paralelismo

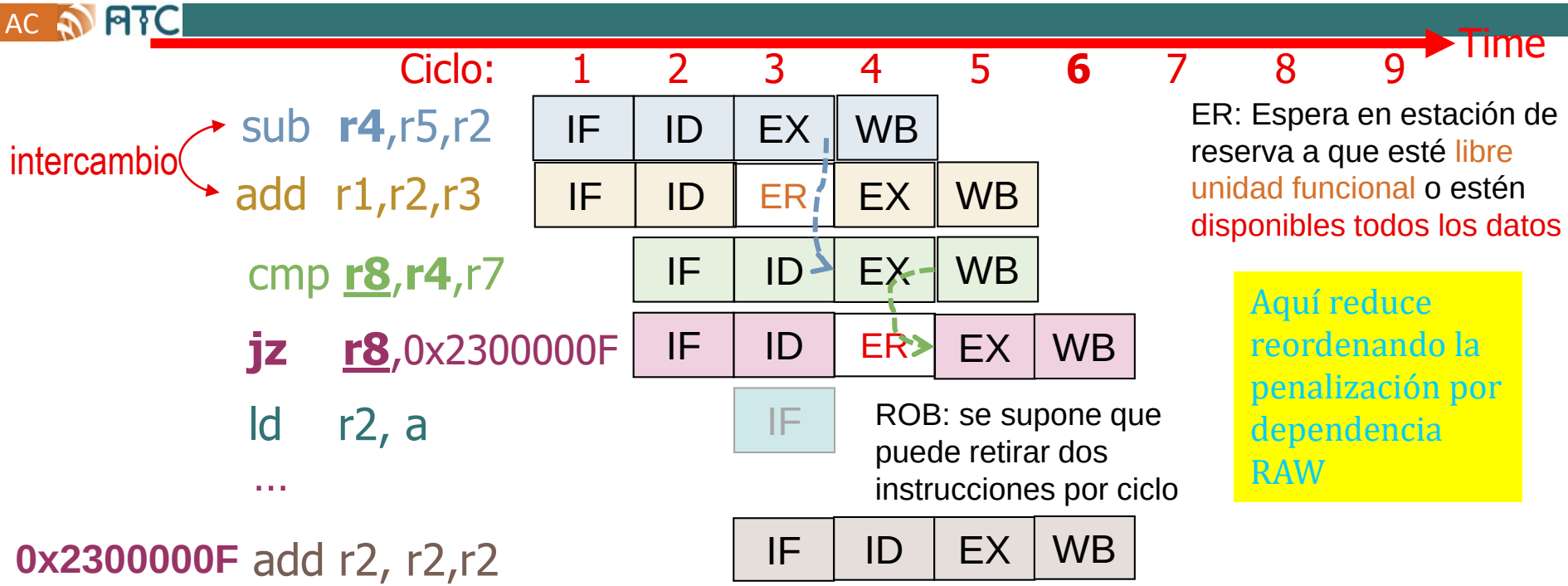


El hardware ha eliminado los **riesgos**, y las dependencias suponen una penalización en tiempo mucho menor.

Ciclos real = 4 c + 3 c = 7 c

CPI = 3/4= **0.75 c/s**

¿Y el software puede hacer algo?



CPI ideal = **0.5 c/i**
Ciclos ideal = 5c + 2c = 7 c

Ciclos real = 4c + 2c = 6 c
Ciclos real = 4c + CPI x 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 Superescalres, VLIW**

✓Puede evitar que los 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**

Instrucciones de Ejecución Condicional (*Guarded Execution*)

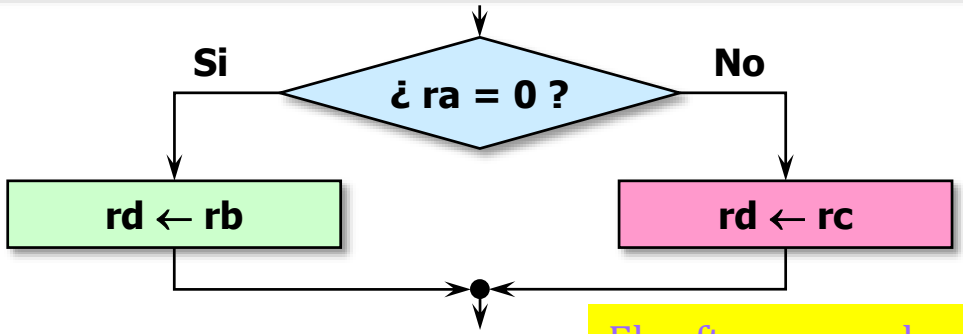
- Se pretende **reducir el número de instrucciones de salto** incluyendo en el **repertorio máquina instrucciones con operaciones condicionales** (*'conditional operate instructions'* o *'guarded instructions'*)
- Estas instrucciones tienen dos partes: la **condición** (denominada *guardia*) y la parte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




```
        beq   ra, cero
        or    rc, rc, rd
        br    fin
cero:   or    rb, rb, rd
fin:    ...
```

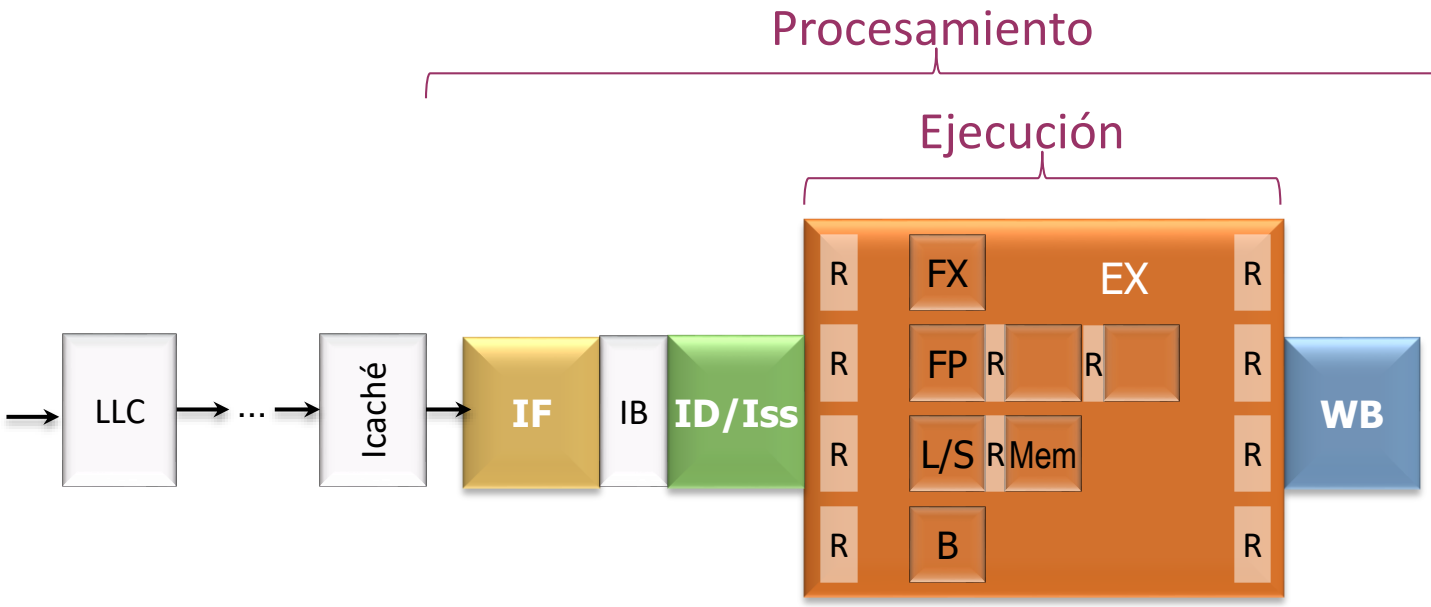
El software puede eliminar dependencias, aquí de control

```
or      rb, rb, rd
cmovne  ra, rc, rd
...
```

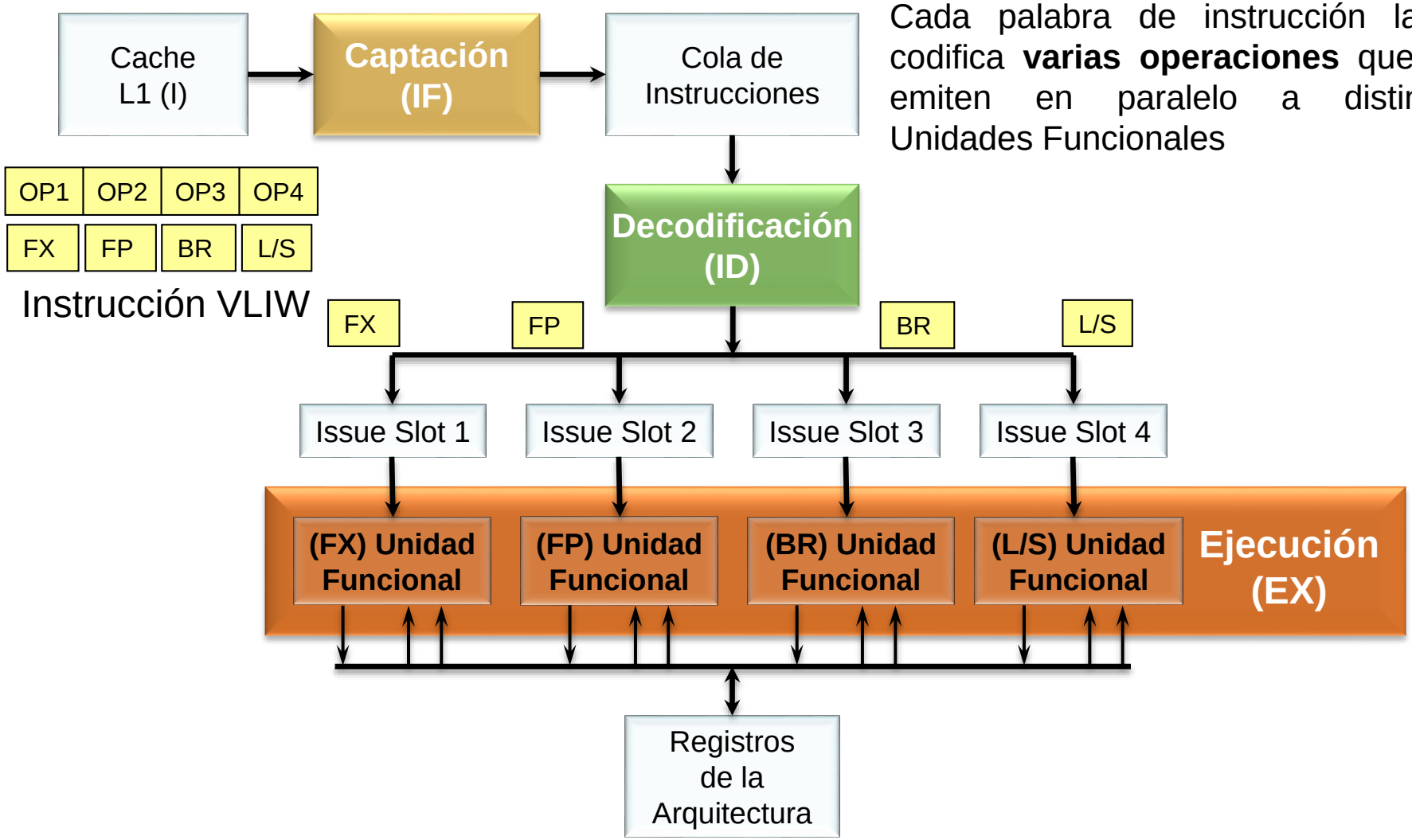
Apartados

- 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



Cada palabra de instrucción larga codifica **varias operaciones** que se emiten en paralelo a distintas Unidades Funcionales