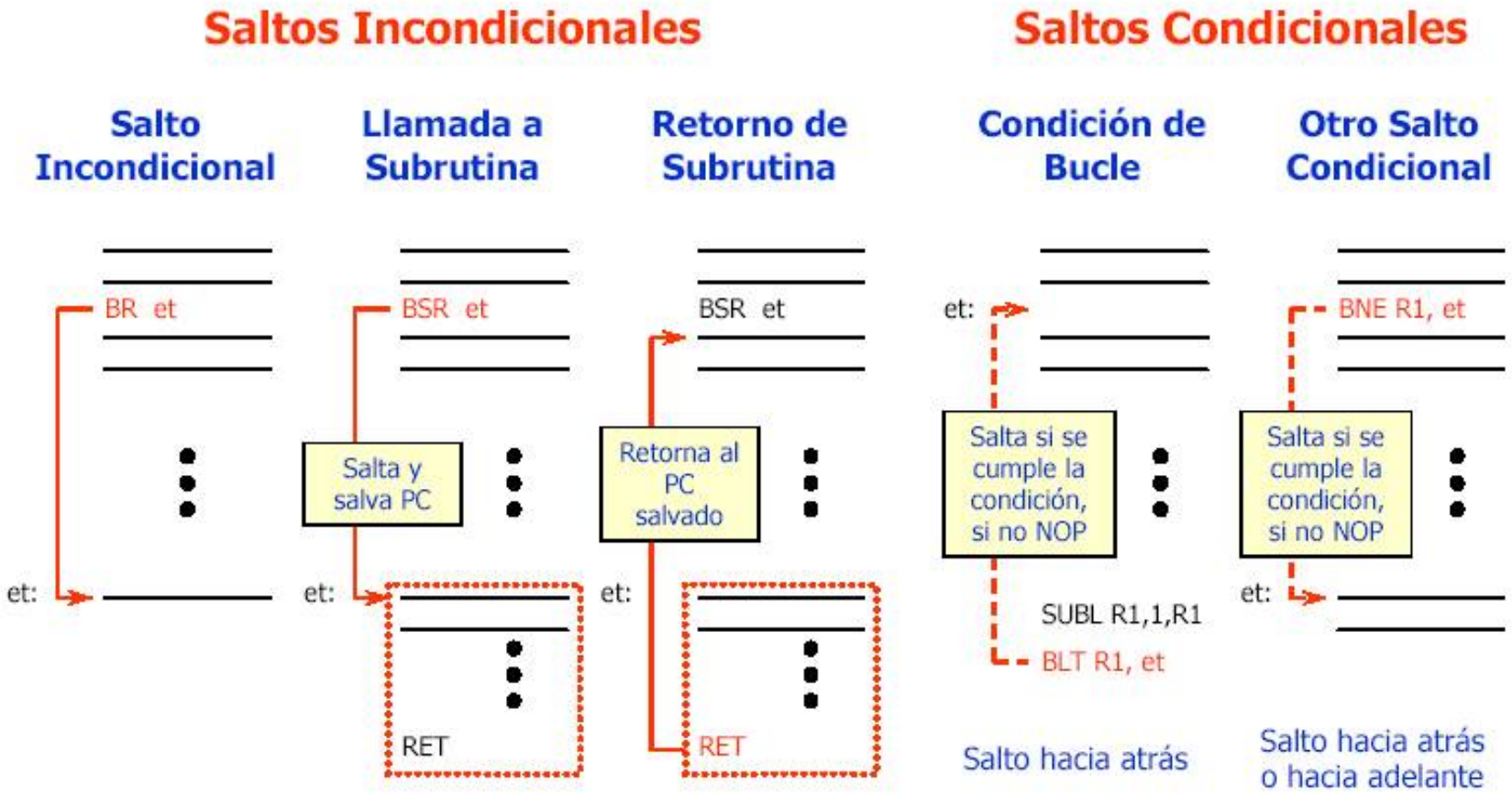
Ingeniería de los Computadores

Sesión 5. Riesgos de control

Clasificación de los Saltos



Riesgos

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

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

- **Detección de la Instrucción de Salto**

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

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

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

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

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

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

Riesgos

- Detección
  - En etapa de decodificación

  - Detección temprana (‘early branch detection’)
    - Detección paralela a decodificación

    - Detección anticipada (‘look-ahead branch detection’)

    - Detección integrada en captación

Riesgos **Gestión**

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

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 de saltos
Estática
Dinámica
Fija
Basada en Desplaz.
Basada en *Opcode*
Dirigida por Compilador
1 nivel de historia
2 niveles de historia
Híbridos

- Predicción fija

Aproximación 'Siempre No Tomado' - Toda condición de salto no resuelta se predice que no da lugar a un salto - Se continúa la ejecución por donde iba aunque se puede adelantar algo el procesamiento de la secuencia de salto (calculo de la dirección de salto, BTA) - Cuando se evalúa la condición se comprueba si la predicción era buena. - Si la predicción era buena el procesamiento continúa y se borra la BTA, y si era mala se abandona el procesamiento de la secuencia predicha (no se considera su efecto) y se captan instrucciones a partir de la BTA * Es más fácil de implementar que la aproximación de 'Siempre Tomado'	SuperSparc (1992) (TP: 1; NTP: 0) Alpha21064 Power I (1990) (TP: 3; NTP: 0) Power 2 (1993) (TP: 1; NTP: 0)
Aproximación 'Siempre Tomado' - Toda condición de salto no resuelta se predice que da lugar a un salto - En previsión de error de predicción se salva el estado de procesamiento actual (PC) y se empieza la ejecución a partir de la dirección de salto. - Cuando se evalúa la condición de salto se comprueba si la predicción era buena - Si la predicción es correcta se continúa, y si es errónea se recupera el estado almacenado y no se considera el procesamiento de la secuencia errónea * Necesita una implementación más compleja que la aproximación anterior aunque suele proporcionar mejores prestaciones	MC68040 (1999) (TP: 1; NTP: 2)

- Predicción estática

Predicción basada en el Código de Operación Para ciertos códigos de operación (ciertos saltos condicionales específicos) se predice que el salto se toma, y para otros que el salto no se toma	MC88110 (93) PowerPC 603(93)

Ejemplo: Predicción Estática en el MC88110

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

- Predicción estática

- Predicción estática

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

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

- 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

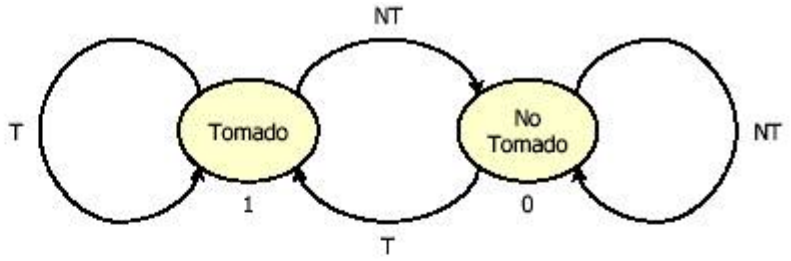
  - **Predicción Dinámica Explícita**

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

- Predicción dinámica explícita

Predicción con 1 bit de historia

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

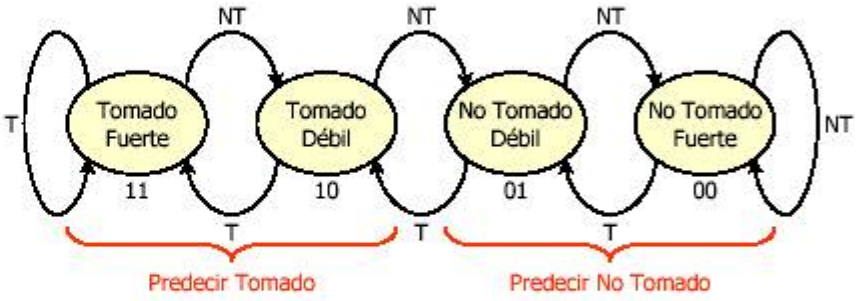


Predicción con 2 bits de historia

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

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

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

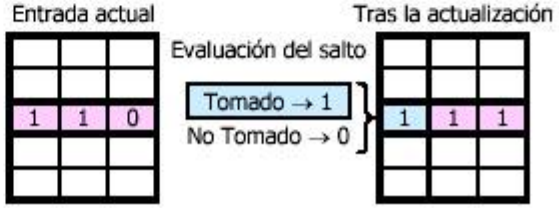


Predicción con 3 bits de historia

Cada entrada guarda las últimas ejecuciones del salto

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

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



- Predicción dinámica explícita

Implementación de los Bits de Historia

• **En la Cache de Instrucciones**	Alpha 21064 (92) (2k x 1 bit) Alpha 21064A (94) (4k x 2 bits) UltraSparc (92) (2k x 2 bits)
• **En una Tabla de Historia de Salto (BHT)**	PA8000(96) (253 x 3bits) PowerPC 620(95) (2K x 2bits) R10000 (96) (512 x 2bits)
• **En una Cache para las Instrucciones a las que se produce el Salto (BTIC) o,** • **En una Cache para las Direcciones a las que se produce el Salto (BTAC)**	Pentium (94) (256 x 2 bits) (BTAC) MC 68069 (93) (256 x 2bits) (BTAC)

- Predicción dinámica explícita

1) Branch Target Buffer (BTB): bits acoplados

La BTB almacena

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

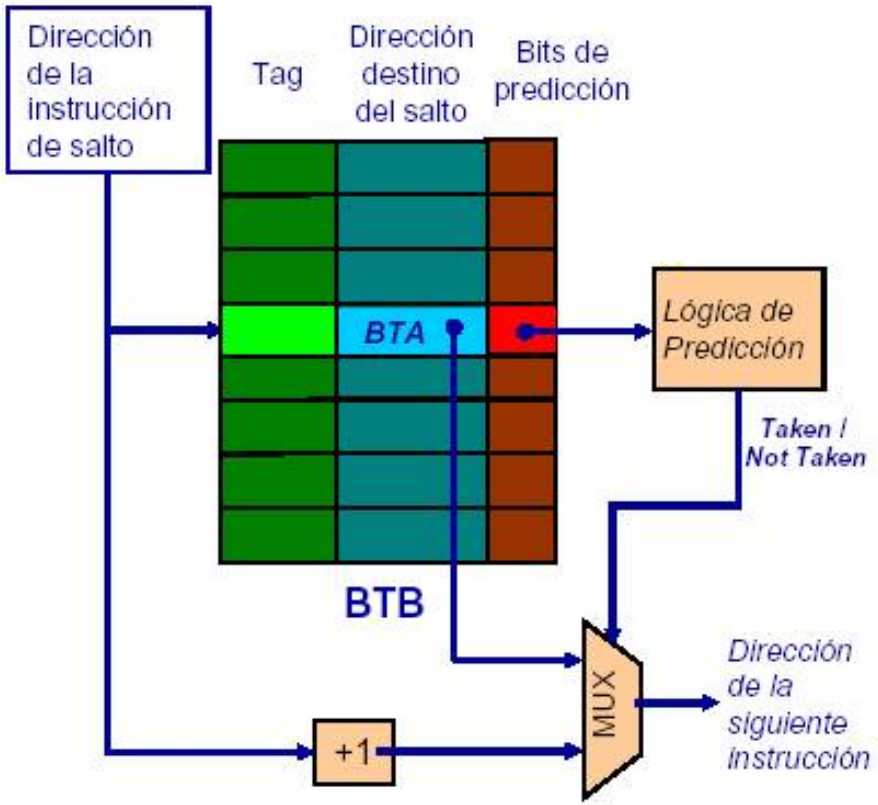
Actualización de la BTB

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

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

Predicción Implícita *(sin bits de predicción)*

Aplicable con un sólo bit de predicción
- Sí la instrucción de salto está en la BTB
  ⇒ El salto se predice como tomado
- Sí la instrucción de salto no está en la BTB
  ⇒ El salto se predice como no tomado



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

- Predicción dinámica explícita

2) Tabla de historia de saltos (BHT): bits desacoplados

Existen dos tablas distintas:

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

Ventaja

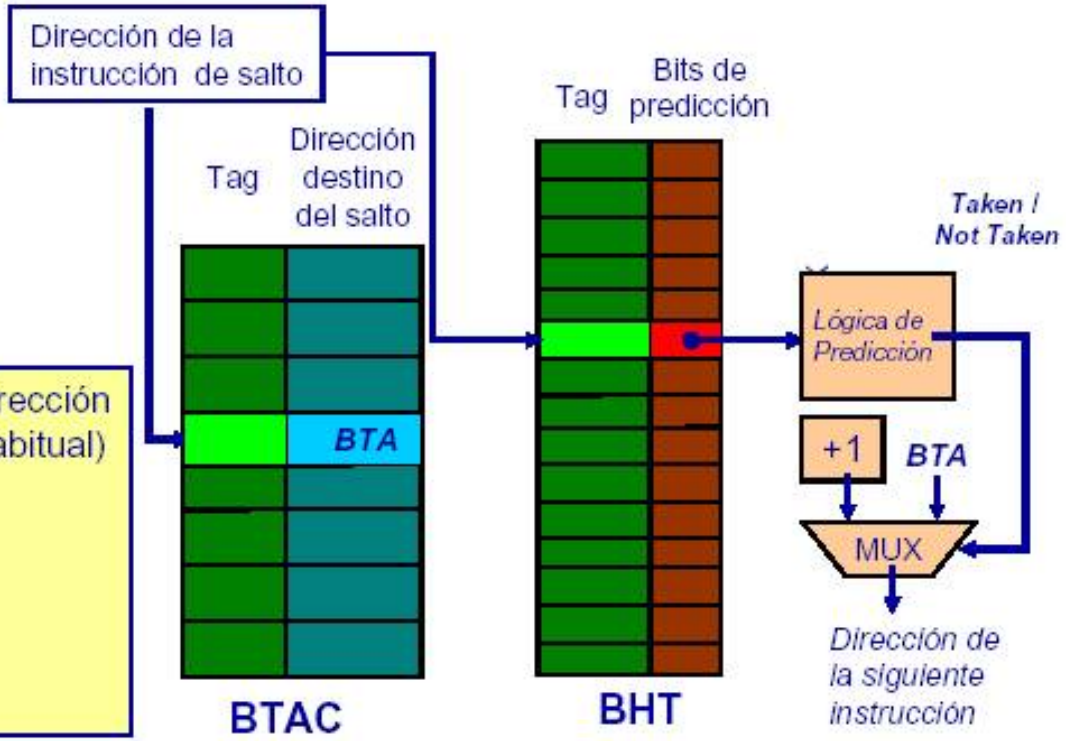
Puede predecir instruc.que no están en la BTAC (más entradas en BHT que en BTAC)

Desventaja

Aumenta el hardware necesario
⇒ 2 tablas asociativas

Acceso a la BHT

- Usando los bits menos significativos de la dirección
  - Sin TAGs ⇒ Menor coste (opción + habitual)
  - Compartición de entradas
    ⇒ Se degrada el rendimiento
- Asociativa por conjuntos
  - Mayor coste ⇒ Tablas pequeñas
  - Para un mismo coste hardware
    ⇒ Peor comportamiento



- Predicción dinámica explícita

3) Bits de predicción en la I-cache

Funcionamiento

Cuando se capta la instrucción de la cache
Si se trata de una instrucción de salto condicional
- Se accede en paralelo a los bits de predicción
- Si el salto se predice como tomado se accede a la instrucción destino del salto

Acceso a la instrucción destino del salto

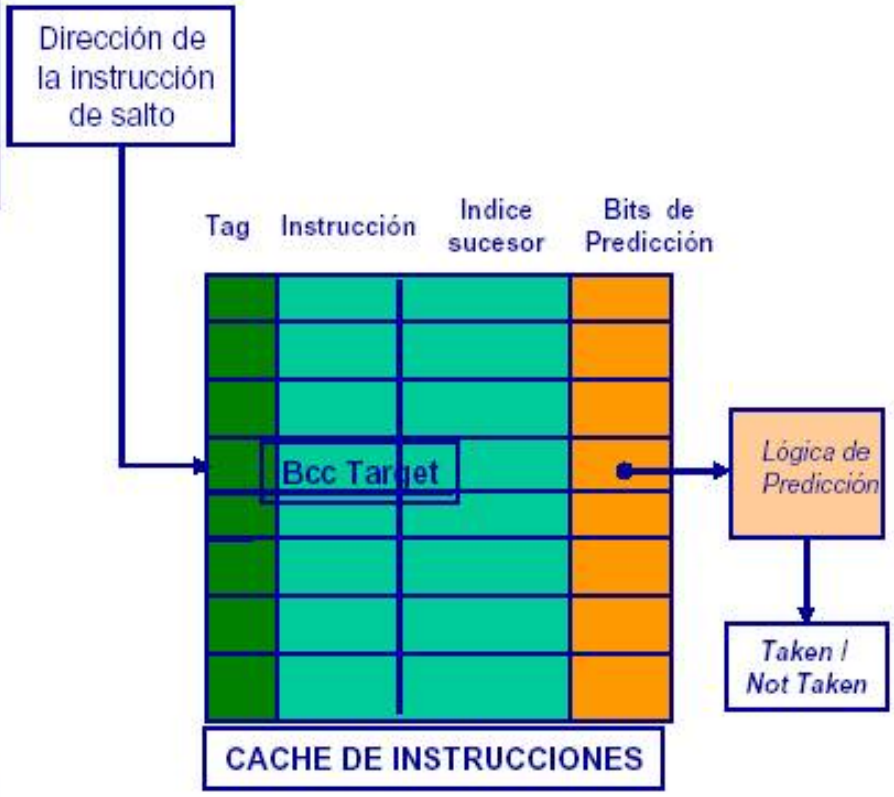
- BTB independiente
- Añadir *índice sucesor* a la I-cache

Alternativas de diseño

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

Ventajas

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



Extensión del Procesamiento Especulativo

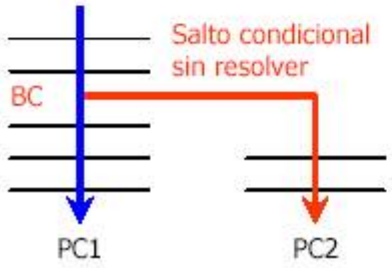
- Tras la predicción, **el procesador continúa ejecutando instrucciones especulativamente hasta que se resuelve la condición**.
- El intervalo de **tiempo entre el comienzo de la ejecución especulativa y la resolución** de la condición **puede variar considerablemente y ser bastante largo**.
- En los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

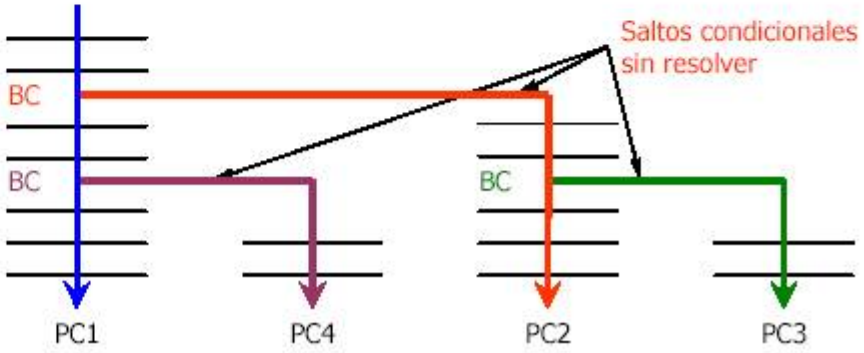
Así, **cuanto mejor es el esquema de predicción mayor puede ser el número de instrucciones ejecutadas especulativamente.**

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

- Ramificación multicamino

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





- Ejecución condicional

Instrucciones de Ejecución Condicional (*Guarded Execution*)

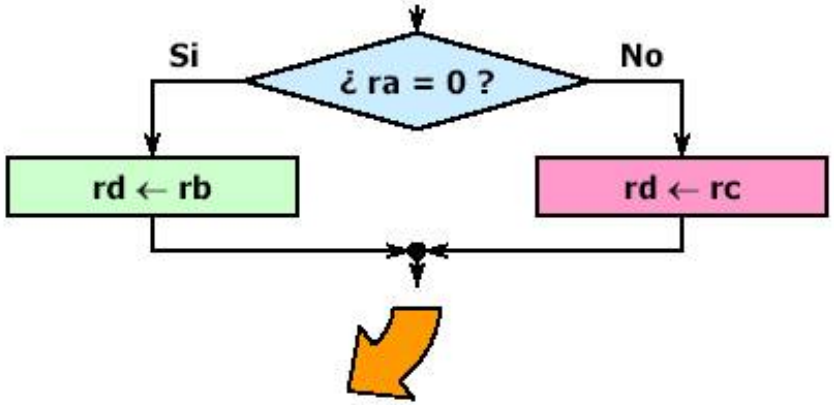
- Se pretende **reducir el número de instrucciones de salto** incluyendo en el **repertorio máquina instrucciones con operaciones condicionales** (*'conditional operate instructions'* o *'guarded instructions'*)
- Estas instrucciones tienen dos partes: la **condición** (denominada *guardia*) y la parte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

Ejemplo: cmovxx de Alpha

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

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

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

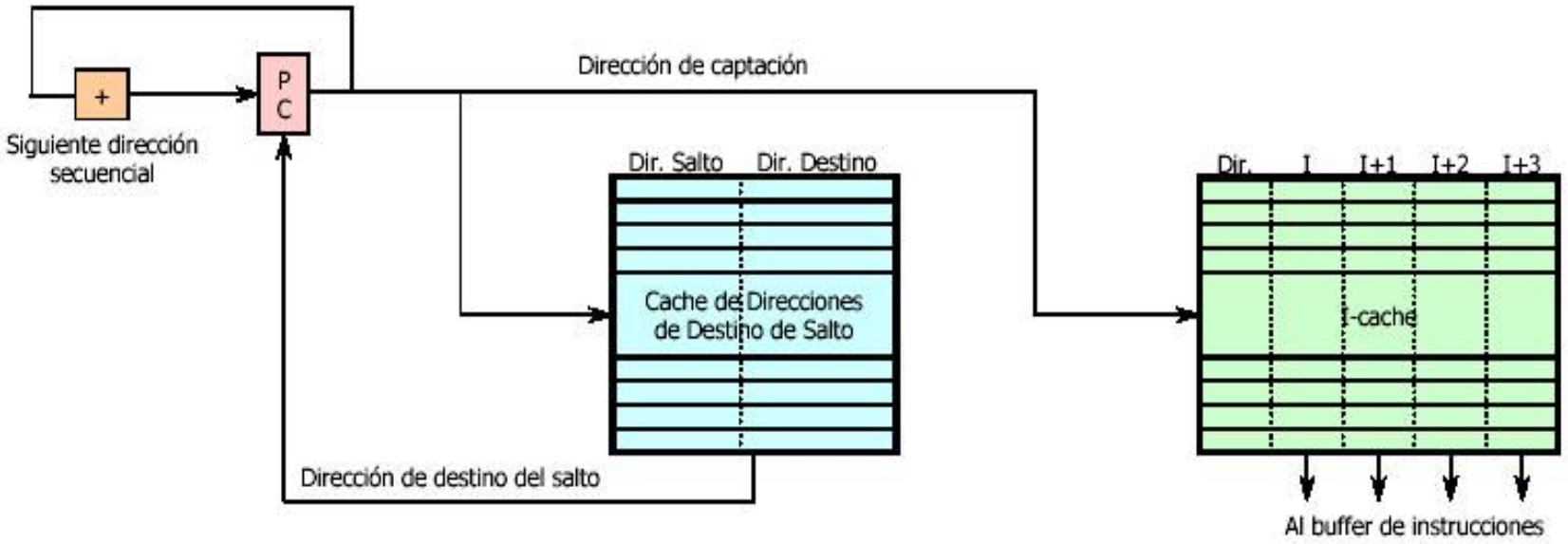



```
      beq  ra, cero
      or   rc, rc, rd
      br   fin
cero: or   rb, rb, rd
fin:  ...
```

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

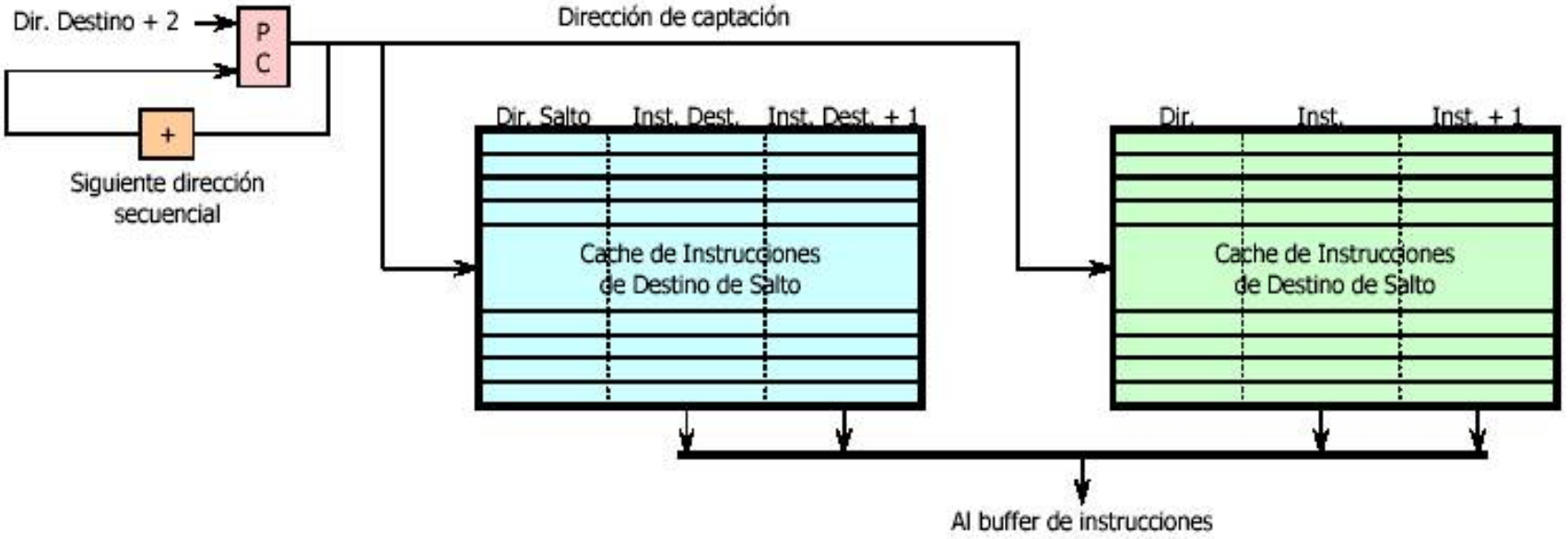
Esquema de cache de direcciones de destino de salto (BTAC)

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



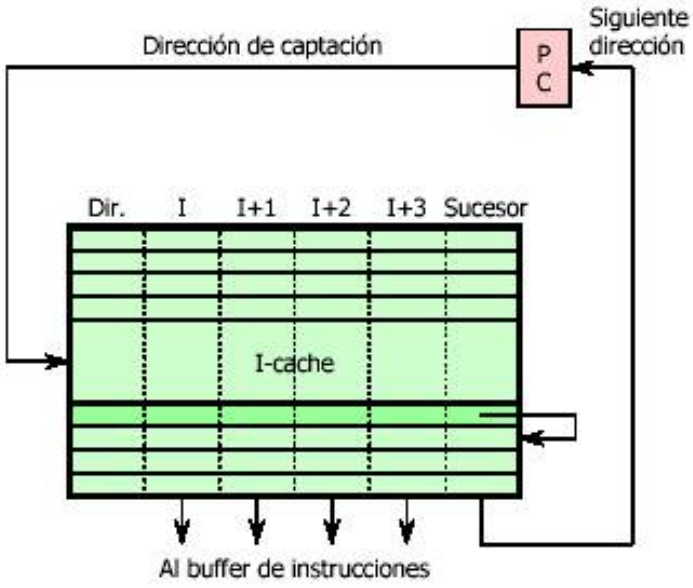
Esquema de cache de instrucciones de destino de salto (BTIC)

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



Esquema de índice sucesor en la cache de instrucciones

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



Ejemplos y evolución de los esquemas de Acceso

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

Renombrado

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

Implementación Estática: Durante la Compilación

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

Características de los Buffers de Renombrado

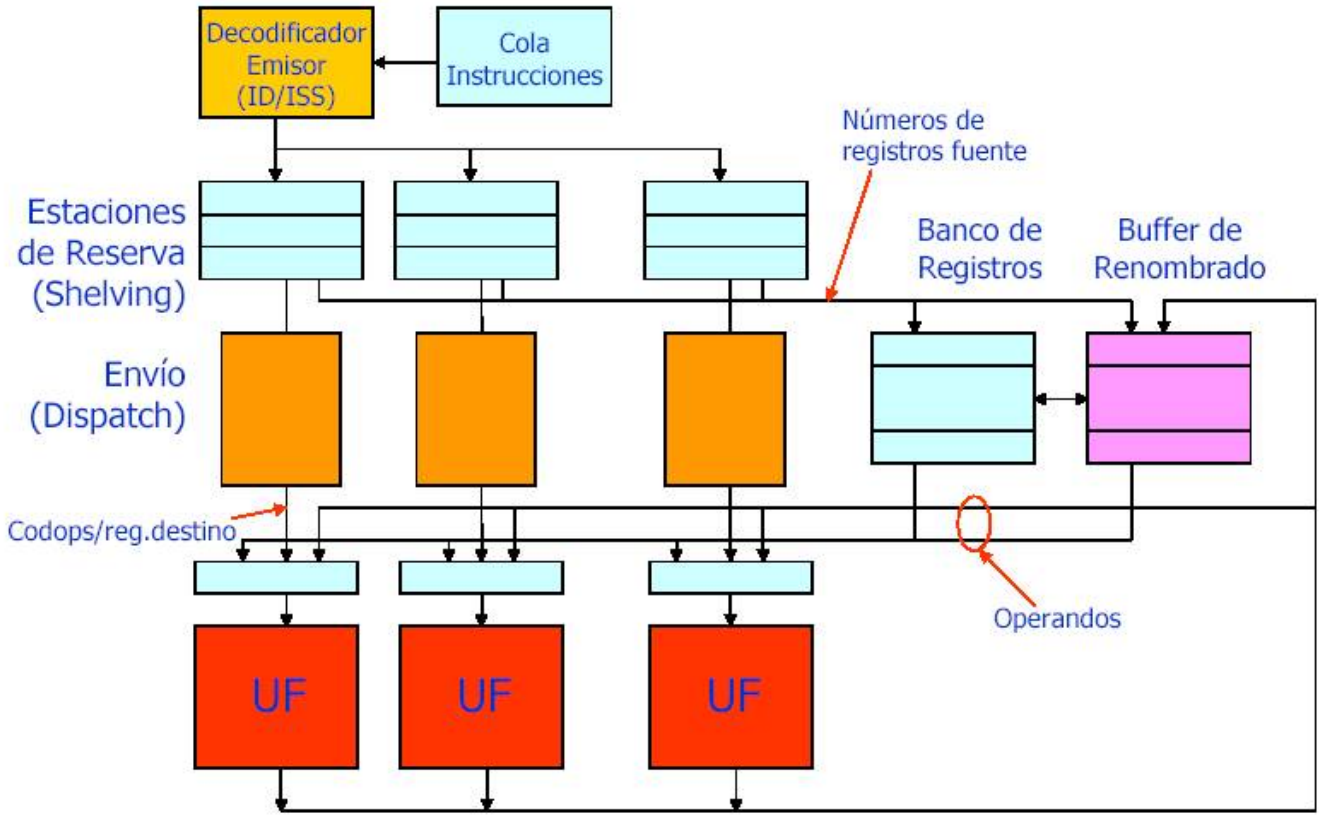
- **Tipos de Buffers** (separados o mezclados con los registros de la arquitectura)
- **Número de Buffers de Renombrado**
- **Mecanismos para acceder a los Buffers** (asociativos o Indexados)

Velocidad del Renombrado

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

Renombrado

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



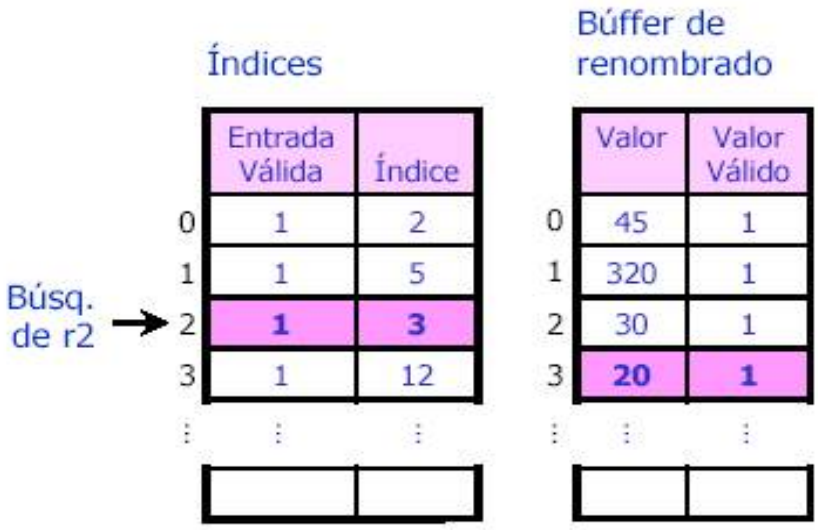
Renombrado

- Tipos de buffers de renombrado

Buffer de Renombrado con Acceso Asociativo

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

Buffer de Renombrado con Acceso Indexado



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

Ejemplo de Renombrado (I)

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

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

```
r2a = r0a * r1a
r3a = r1a + r2a
r2b = r0a – r1a
```

Situación Inicial:

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

Ejemplo de Renombrado (II)

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

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

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

Ciclo i:

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

Ejemplo de Renombrado (III)

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

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

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

Ciclo i + 1:

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

Ejemplo de Renombrado (IV)

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

r0: 0, "válido" **r1: 15, "válido"** **6**

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

Ciclo i + 2:

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

Ejemplo de Renombrado (V)

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

r1: 15, "válido" **r2: 0, "válido"**

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

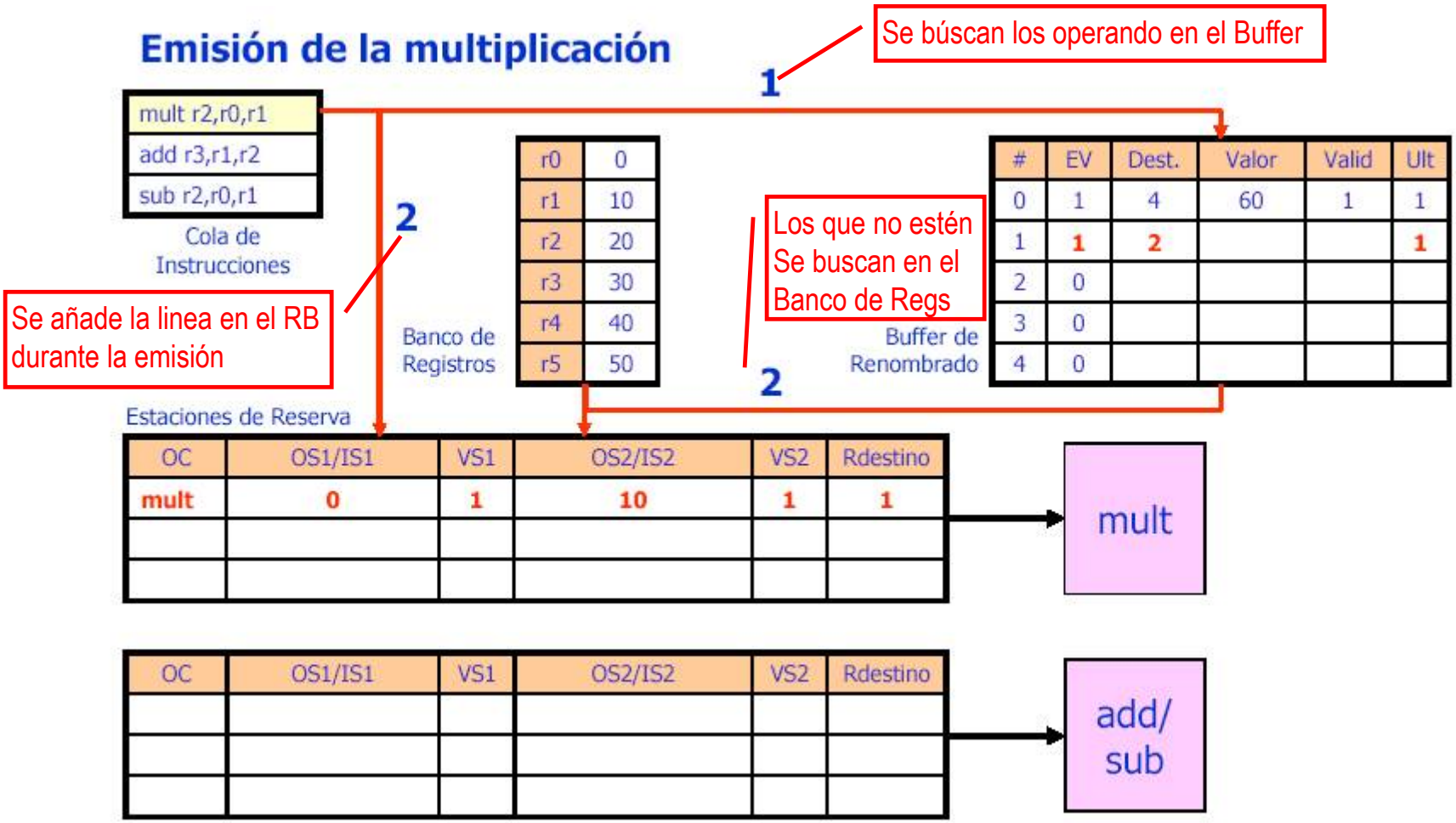
Ciclo i + 5:

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

Renombrado

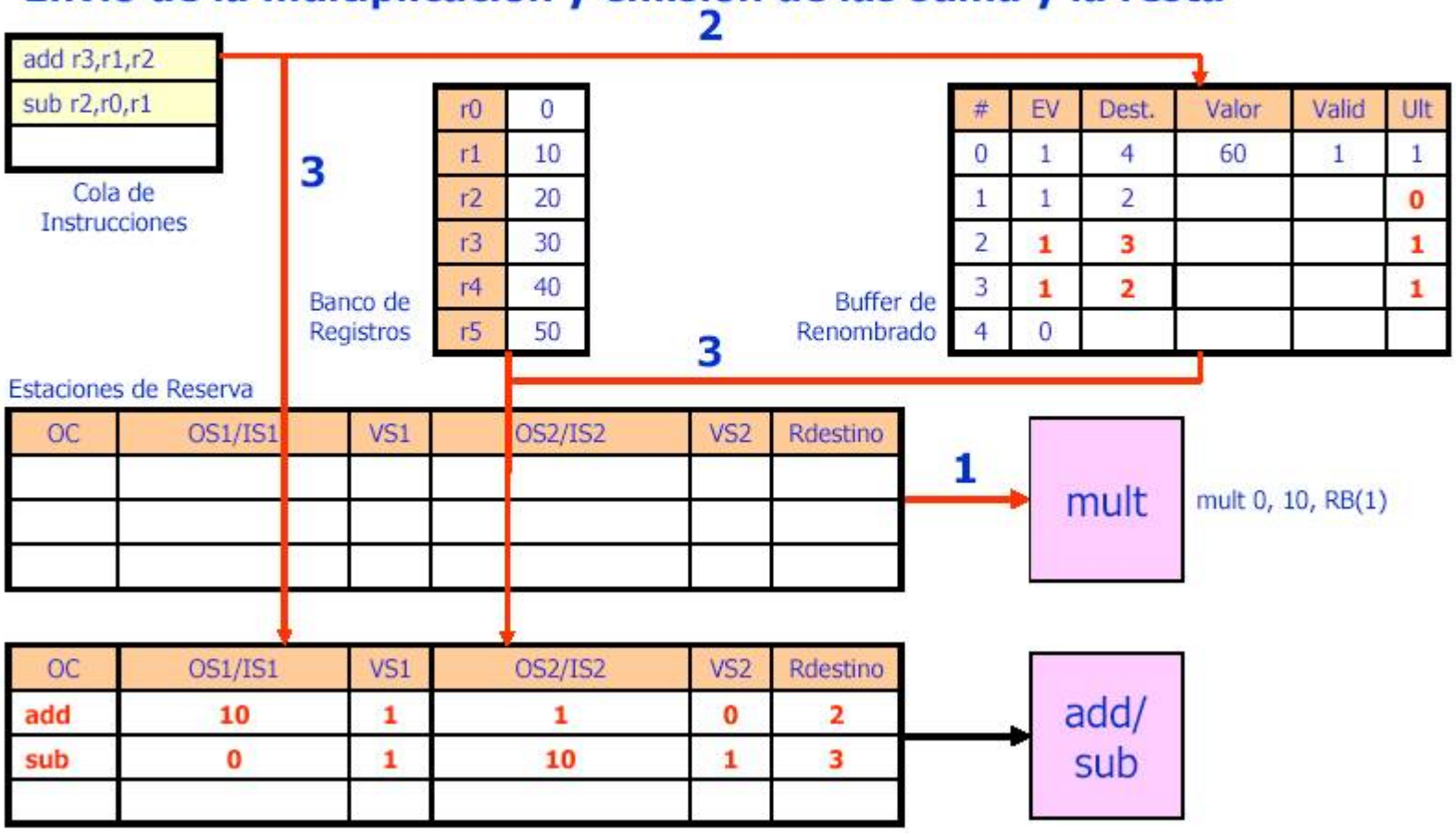
Ejemplo de Renombrado y Estaciones de Reserva

Emisión de la multiplicación



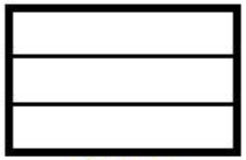
Ejemplo de Renombrado y Estaciones de Reserva (II)

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



Ejemplo de Renombrado y Estaciones de Reserva (III)

Envío de la resta



Cola de Instrucciones

Banco de Registros

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

Buffer de Renombrado

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

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

mult 0, 10, RB(1)

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

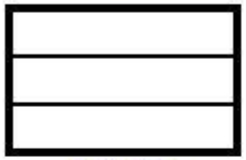
1

add/ sub

sub 0, 10, RB(3)

Ejemplo de Renombrado y Estaciones de Reserva (IV)

Termina la resta



Cola de Instrucciones

Banco de Registros

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

Buffer de Renombrado

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

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

mult 0, 10, RB(1)

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

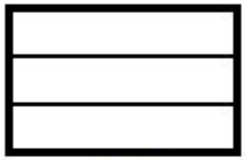
add/ sub

sub 0, 10, RB(3)

1

Ejemplo de Renombrado y Estaciones de Reserva (V)

Termina la multiplicación



Cola de Instrucciones

Banco de Registros

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

Buffer de Renombrado

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

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

mult 0, 10, RB(1)

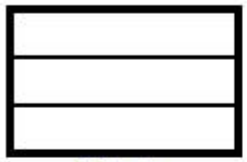
1

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

add/ sub

Ejemplo de Renombrado y Estaciones de Reserva (VI)

Envío de la suma



Cola de Instrucciones

Banco de Registros

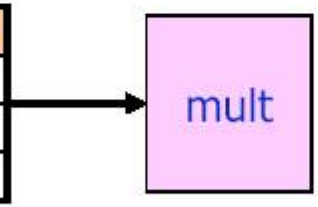
r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Buffer de Renombrado

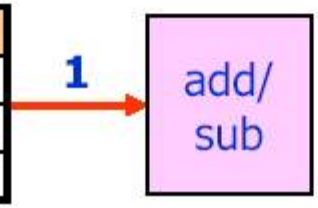
#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	−10	1	1
4	0				

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



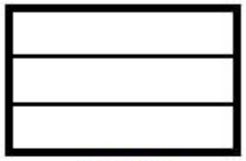
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



add 10, 0, RB(2)

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Termina la suma



Cola de Instrucciones

Banco de Registros

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

Buffer de Renombrado

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

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

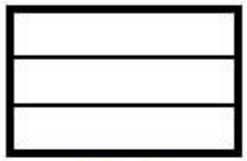
add/ sub

add 10, 0, RB(2)

1

Ejemplo de Renombrado y Estaciones de Reserva (VII)

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

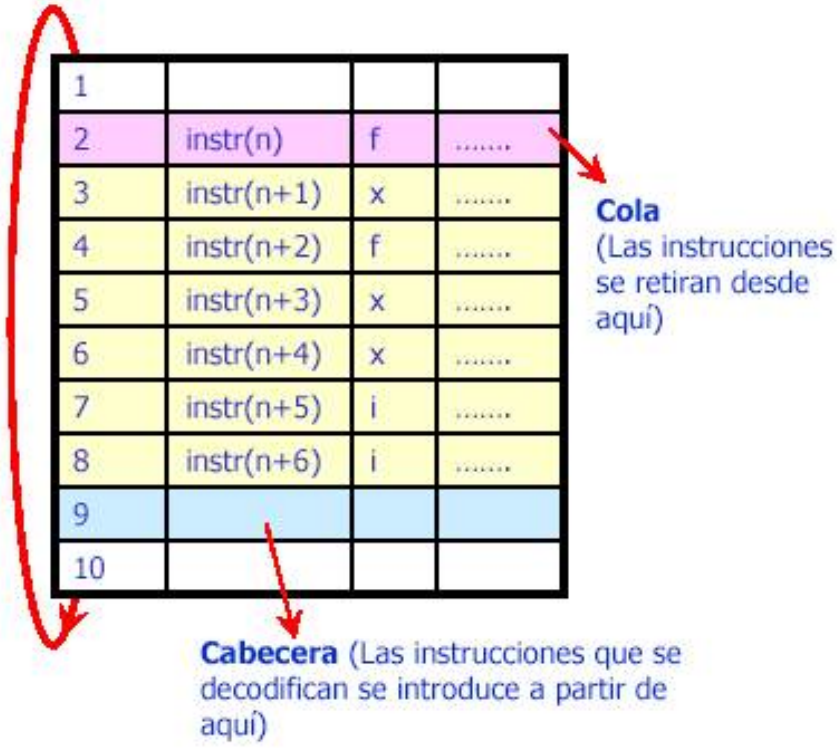
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

add/ sub

Renombrado **Reorden**

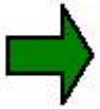


La gestión de interrupciones y la ejecución especulativa se realizan fácilmente mediante el ROB

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

Ejemplo de Uso del Buffer de Reorden (I)

I1: mult r1, r2, r3
I2: st r1, 0x1ca
I3: add r1, r4, r3
I4: xor r1, r1, r3



Dependencias:
RAW: (I1,I2), (I3,I4)
WAR: (I2,I3), (I2,I4)
WAW: (I1,I3), (I1,I4), (I3,I4)

I1: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r2** y **r3)**
I2: Se envía a la unidad de almacenamiento hasta que esté disponible **r1**
I3: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r4** y **r3)**
I4: Se envía a la estación de reserva de la ALU para esperar a **r1**

Estación de Reserva (Unidad de Almacenamiento)

codop	dirección	op1	ok1
st	0x1ca	3	0

Estación de Reserva (ALU)

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

Líneas del ROB

Ejemplo de Uso del Buffer de Reorden (II)

Ciclo 7

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

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

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

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

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

Ejemplo de Uso del Buffer de Reorden (III)

Ciclo 12

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

Ciclo 13

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

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