

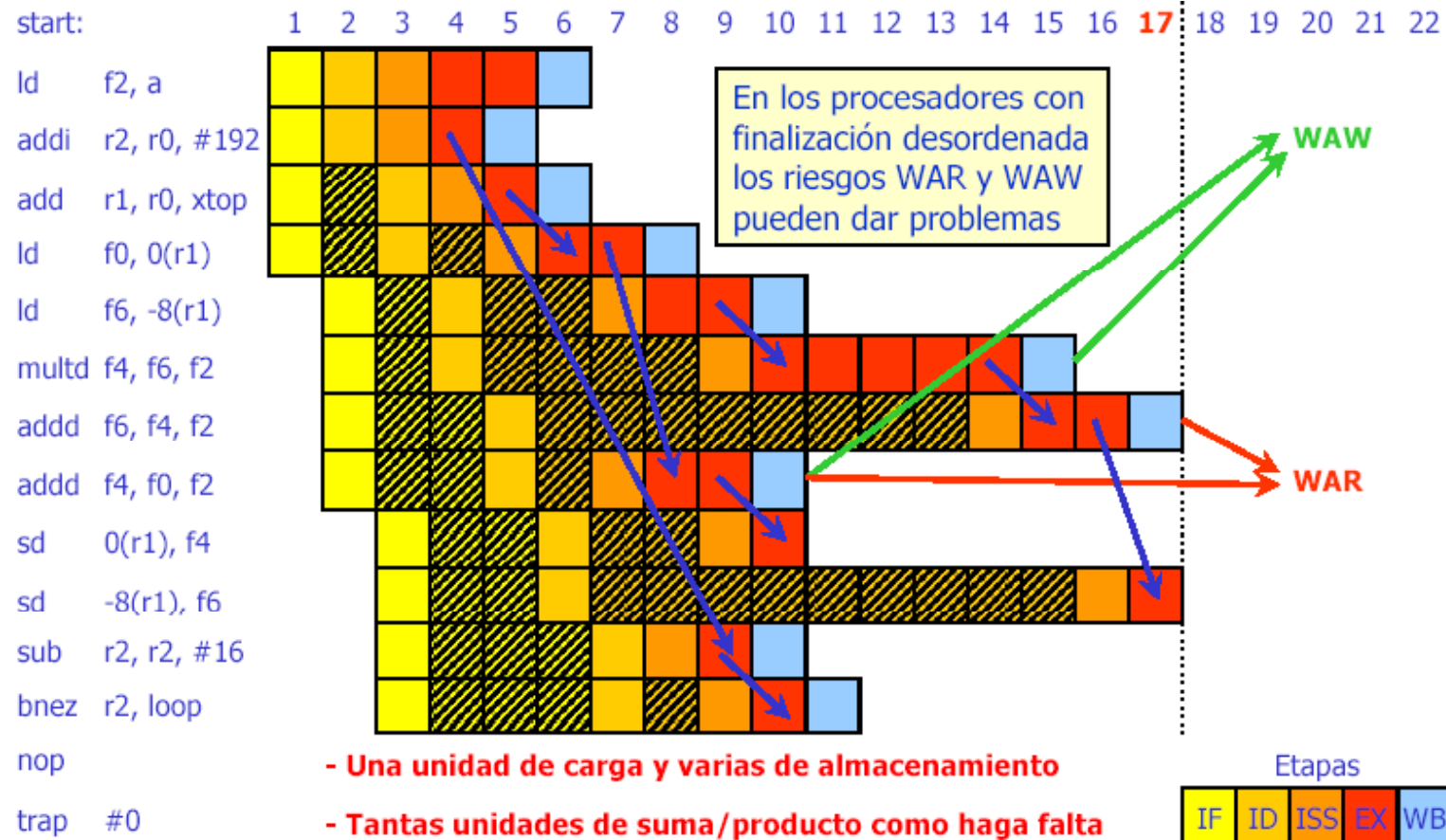
Arquitectura e Ingeniería de Computadores

Tema 2 – Segmentación y superescalares clase 5

Ingeniería en Informática

Departamento de Tecnología Informática y Computación

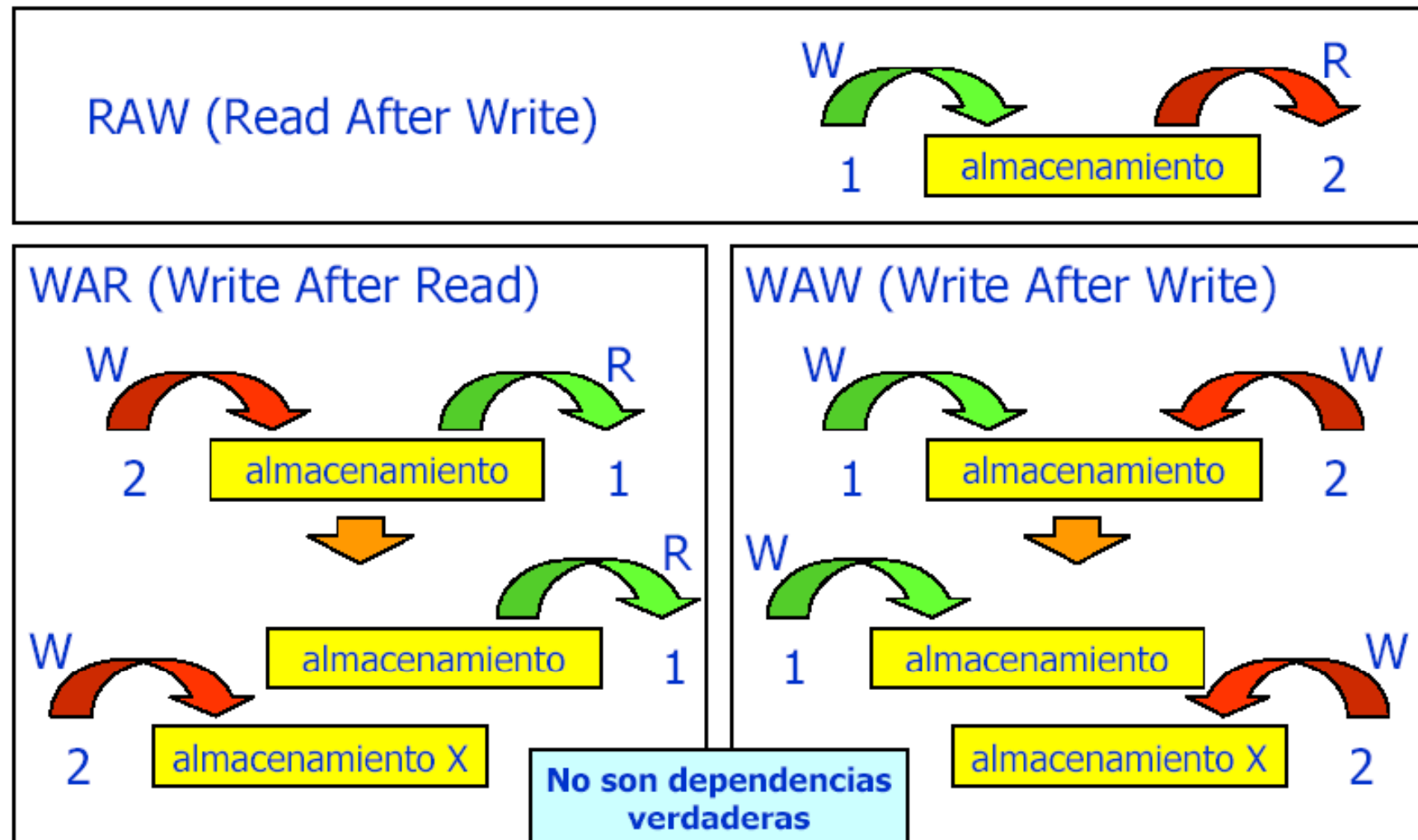
Renombramiento de registros: Motivación



Scoreboarding: evita los riesgos RAW pero en WAW y WAR produce detenciones del cauce (stalls)

Renombramiento: evita todos los tipos de riesgos por dependencias de datos

Revisión de los riesgos por dependencias



Renombrado 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 (circuitría 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

► Tipos de Buffer de Renombrado

Buffer de Renombrado con Acceso Asociativo

Indica si la línea se ha utilizado

Reg. que se ha renombrado

Último renombrado

Búffer de Renombrado

	Entrada Válida	Registro Destino	Valor	Valor Válido	Último
→	1	5	50	1	1
→	1	12	1200	1	1
→	1	2	20	1	1
→	1	1	3	1	1
⋮	⋮	⋮	⋮	⋮	⋮
→					

- 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

Índices

Búffer de renombrado

	Entrada Válida	Índice		Valor	Valor Válido
0	1	2	0	45	1
1	1	5	1	320	1
2	1	3	2	30	1
3	1	12	3	20	1
⋮	⋮	⋮	⋮	⋮	⋮

- 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

Organización de los Buffers de Renombrado

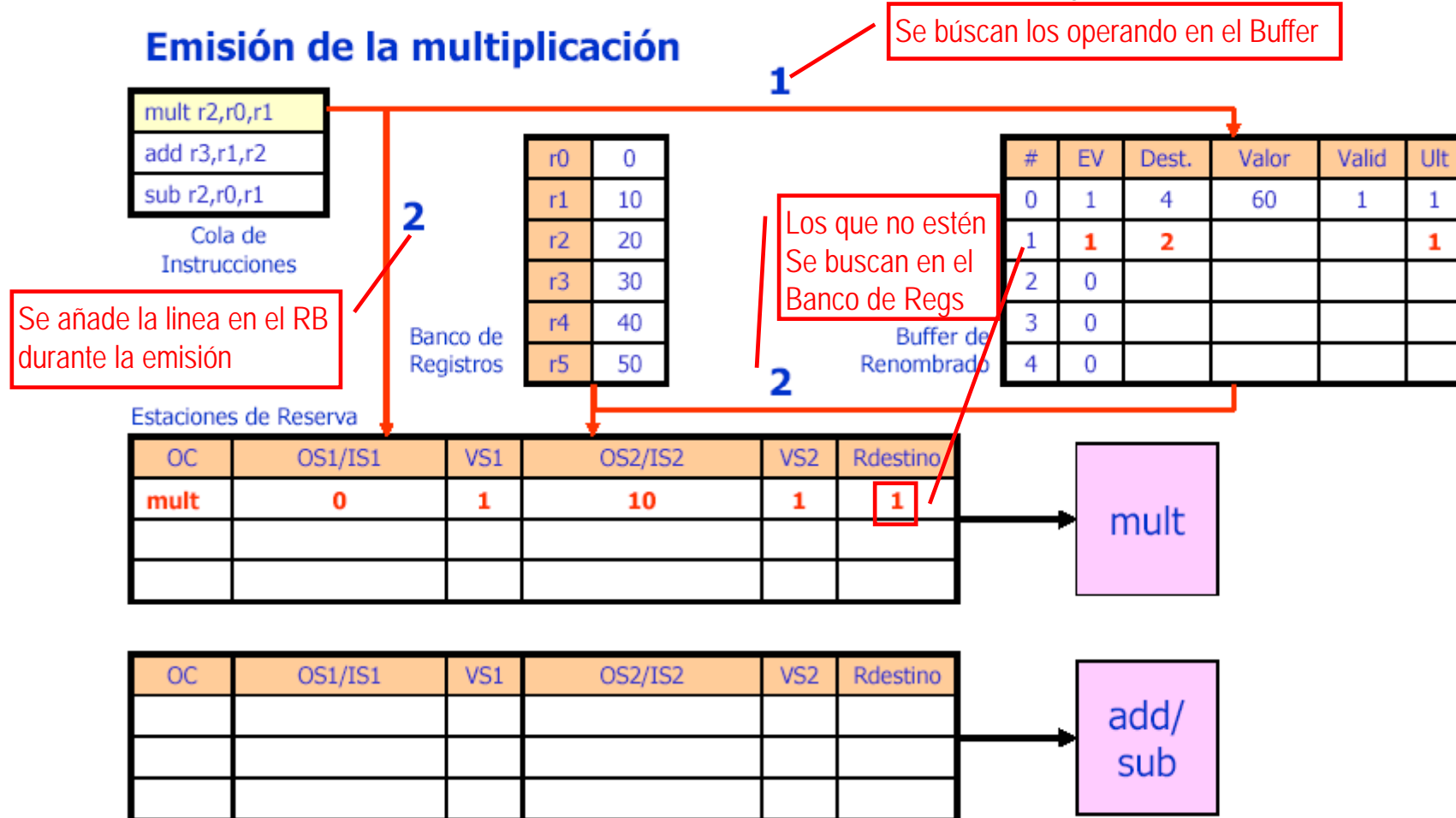
Procesador	FX	FP
Banco de registros de Arquitectura y de Renombrado		
Power 1 (90)	-	8 (32 arq + 8)
Power 2 (93)	-	22 (32 arq + 22)
ES/9000 (92)	16 (16 arq + 16)	12 (4 arq + 12)
Sparc64 (95)	38 (78 arq + 38)	24 (32 arq + 24)
R10000 (96)	32 (32 arq + 32)	32 (32 arq + 32)
Registros de la Arquitectura y de Renombrado separados		
PowerPC 603 (93)	-	4
PowerPC 604 (95)	12	8
PowerPC 620 (96)	8	8
Renombrado dentro del ROB		
AM29000 sup (95)	10	
K5 (95)	16	
Pentium Pro (95)	40	

Emisión: Políticas de Emisión y Renombrado

Emisión Escalar (Con Bloqueo / Sin renombrado)	Emisión Superescalar Directa (Con Bloqueo/ Sin renombrado)		Emisión Superescalar Avanzada (Sin Bloqueo/ Con Renombrado)
	Alineada	No alineada	
i86 (78) i286 (82) i386 (85) i486 (88)	Pentium (93)		Pentium II (98)
MC68000 (79) MC68020 (82) MC68030 (87) MC68040 (90)		MC68060 (93)	
R2000 (87) R3000 (89) R4000 (93)		R8000 (94)	R10000 (96)
	PA7100 (92)	PA7200 (95)	PA8000 (96)
Sparc (88) MicroSparc (91)	SuperSparc (92)	UltraSparc (95)	Sparc64 (95)
	PowerPC 601 (93)		PowerPC 603 (93) PowerPC 604 (94) PowerPC 620 (95)
	Alpha 21064 (93) Alpha 21164 (95)		K5 (95)

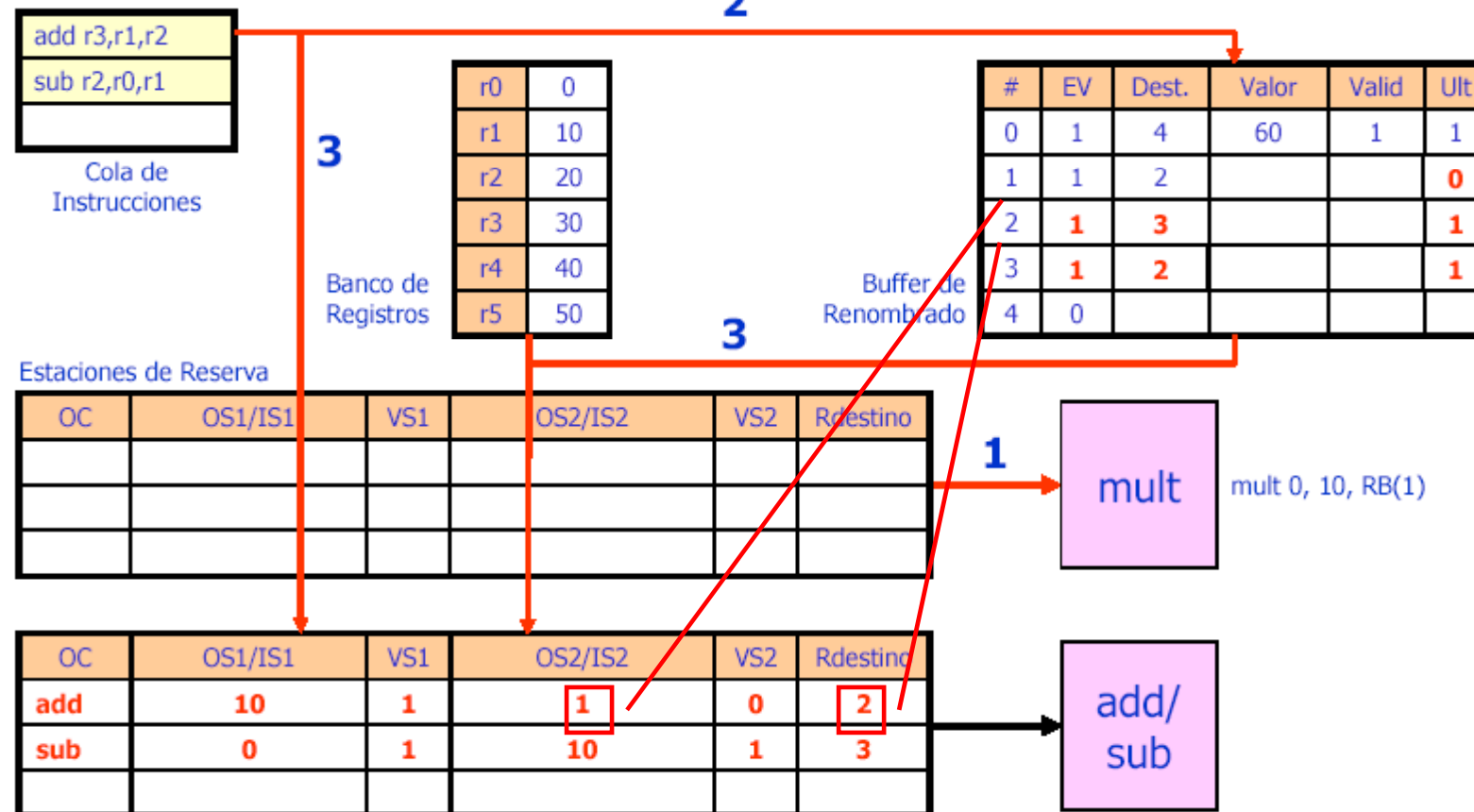
- Ejemplo de Estaciones de Reserva + Renombrado asociativo y captación en la emisión (Emisión desordenada/ejecución desordenada)

Emisión de la multiplicación



Ejemplo de Renombrado y Estaciones de Reserva (II)

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



Ejemplo de Renombrado y Estaciones de Reserva (III)

Envío de la resta/ la suma queda bloqueada

Cola de Instrucciones

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

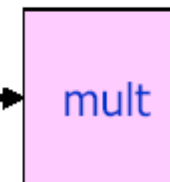
Banco de Registros

#	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

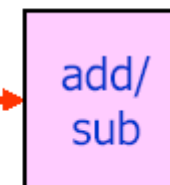
Estaciones de Reserva

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



sub 0, 10, RB(3)

1

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 / actualización de RB y Rs cuyo VS sea 0

Cola de Instrucciones

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

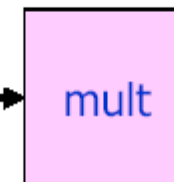
Banco de Registros

#	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

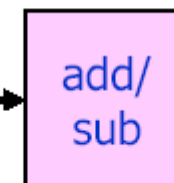
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

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



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

mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

1

add/
sub

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

Finalización ordenada

Cola de
Instrucciones

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

Banco de
Registros

1

#	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

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino

add/
sub

Características de Microprocesadores Actuales

Procesador	Alpha 21264B	AMD Athlon	HP PA8600	IBM Power3II	Intel Pentium III	Intel Pentium 4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
Reloj(MHz)	833	1200	522	450	1000	1500	400	480	900
Cache (I/D/L2)	64k/64k	64k/64k /256k	512k/1M	32k/64k	16k/16k /256k	12k/8k /256k	32k/32k	16k/16k	23k/64k
Emisión	4	3 (x86)	4	4	3 (x86)	3 (uop)	4	4	4
Etapas	7/9	9/11	7/9	7/8	12/14	22/24	6	6/9	14/15
Desorden.	80	72 uop	56	32	40 uop	126 uop	48	-	-
Reg. Ren. (i/fp)	48/41	36/36	56	16/24	40	128	32/32	-	-
Lín. BHT	4kx9bit	4kx2bit	2kx2bit	2kx2bit	>=512	4kx2bit	2kx2bit	512x2bit	16kx2bit
Lín. TLB	128/128	280/288	120	128/128	32I/64D	128I/65D	64	64I/64D	128I/512
Mem (Bw)	2.66GBs	2.1GBs	1.54GBs	1.6GBs	1.06GBs	3.2GBs	599MBs	1.9GBs	4.8GBs
Proceso IC	0.18u	0.18u	0.25u	0.22u	0.18u	0.18u	0.25u	0.29u	0.18
Dado(mm ²)	115	117	477	163	106	217	204	126	210
Transist.	15.4M	37M	130M	23M	24M	42M	7.2M	3.8M	29M
Power (W)	75	76	60	36	30	55	25	20	65
Disponible	1/01	4/00	3/00	4/00	2/00	4/00	2/00	3/00	4/00
Precio (\$)	160	62	330	110	39	110	125	70	145

◆ Mantenimiento de la consistencia

En el **Procesamiento de una Instrucción** se puede distinguir entre:

- El **Final de la Ejecución de la Operación** (*Finish*) codificada en las instrucciones
(Se dispone de los resultados generados por las UF pero no se han modificado los registros de la arquitectura).
- El **Final del Procesamiento de la Instrucción** o momento en que se **Completa la Instrucción** (*Complete o Commit*)
(Se escriben los resultados de la Operación en los Registros de la Arquitectura. Si se utiliza un Buffer de Reorden, ROB, se utiliza el término Retirar la Instrucción, Retire, en lugar de Completar)

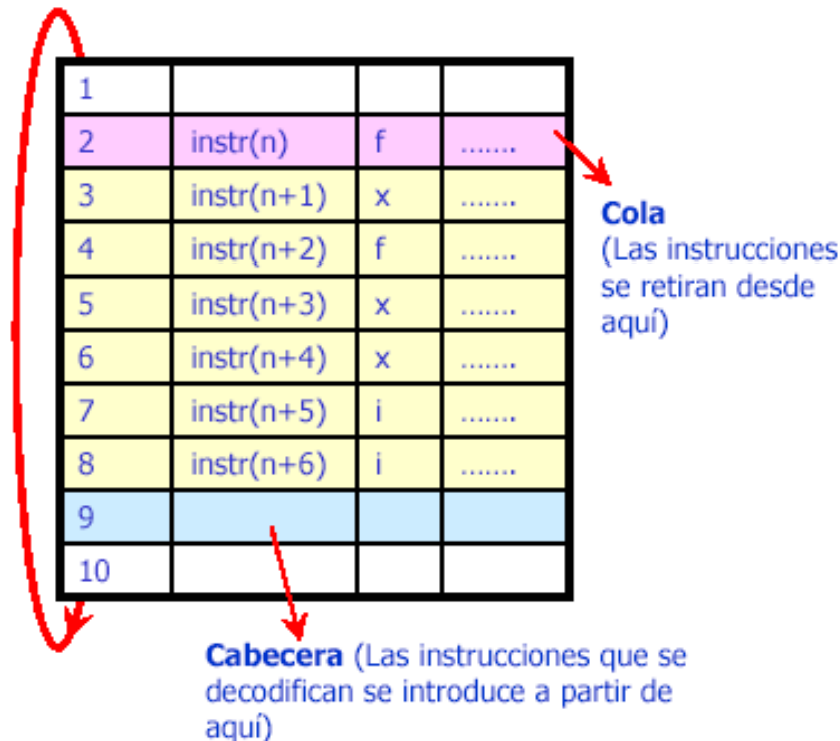
La **Consistencia Secuencial** de un Programa se refiere a:

- El **orden** en que las instrucciones se **completan**
- El **orden** en que se **accede a memoria** para leer (LOAD) o escribir (STORE)

Cuando se ejecutan instrucciones en paralelo, el orden en que termina (*finish*) esa ejecución puede variar según el orden que las correspondientes instrucciones tenían en el programa pero ***debe existir consistencia entre el orden en que se completan las instrucciones y el orden secuencial que tienen en el código de programa.***

Consistencia de Procesador Consistencia en el orden en que se completan las instrucciones	Débil: 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: 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)

◆ Consistencia del procesador



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

Implementa también el Renombrado

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

Xor puede empezar a ejecutarse

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 Termina mult y st se marca como finalizada

#	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 Se retiran mult y st

#	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**.

Ejemplos de Procesadores e Implementaciones del ROB

Procesador	Entradas ROB	Tasa de retirada	Almacena resultados intermedios	Denominación
ES/9000 (1992p)	32	2	No	<i>Completion Control Logic</i>
PowerPC 602 (1995)	4	1	No disponible	<i>Completion Unit</i>
PowerPC 603 (1993)	5	2	No	<i>Completion Buffer</i>
PowerPC 604 (1994)	16	4	No	<i>ROB</i>
PowerPC 620 (1995)	16	4	No	<i>ROB</i>
PentiumPro (1995)	40	3	Si	<i>ROB</i>
Am 29000 sup. (1995)	10	2	Si	<i>ROB</i>
K5 (1995)	16	4	Si	<i>ROB</i>
PM1 (Sparc64, 1995)	64	4	No	<i>Precise Stack Unit</i>
UltraSparc (1995)				<i>Instruction Reorder Buffer</i>
PA 8000 (1996)	56	4	Si	<i>Active List</i>
R 10000 (1996)	32	4	No	<i>Completion Unit</i>

Procesador	Entradas en las estaciones de reserva	Entradas en el ROB
PowerPC 603 (1993)	3	5
PowerPC 604 (1994)	12	16
PowerPC 620 (1995)	15	16
Nx586 (1994)	42	14
PentiumPro (1995)	20	40
K5 (1995)	14	16
PM1 (Sparc64, 1995)	36	64
R 10000 (1996)	48	32

◆ Consistencia de la memoria

Reorden Load/Store

Las instrucciones LOAD y STORE implican cambios en el Procesador y en Memoria

LOAD:

- Cálculo de Dirección en ALU o Unidad de Direcciones
- Acceso a Cache
- Escritura del Dato en Registro

STORE:

- Cálculo de Dirección en ALU o Unidad de Direcciones
- Esperar que esté disponible el dato a almacenar (en ese momento acaba)

La Consistencia de Memoria Débil (reordenación de los accesos a memoria):

- **'Bypass' de Loads/Stores:**

Los Loads pueden adelantarse a los Stores pendientes y viceversa (siempre que no se violen dependencias)

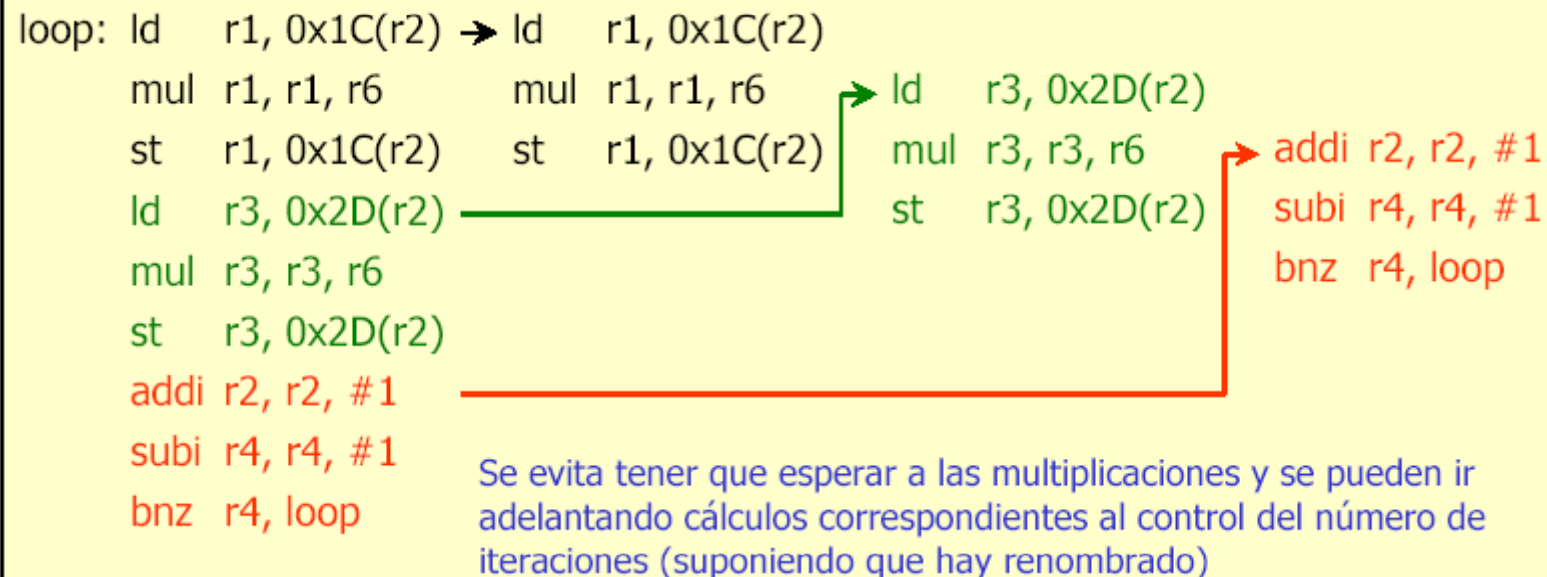
- **Hace posibles Loads y Stores Especulativos:**

Cuando un Load se adelanta a un Store que le precede antes de que se haya determinado la dirección se habla de Load especulativo. Igual para un Store que se adelanta a un Load o a un Store.

- **Permite ocultar las Faltas de Cache:**

Si se adelanta un acceso a memoria a otro que dio lugar a una falta de cache y accede a Memoria Principal.

Ejemplos de 'Bypass' de Loads/Stores



Si se tuviera: st r1,0x1C(r2)

 ld r3,0x2D(r7)

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

Se tendría un **load especulativo**

Reorden de Accesos a Memoria (Ejemplos)

Reorden debido a adelantos LOADs/STOREs	STOREs adelantan LOADs		
	LOADs adelantan STOREs	No especulativos	IBM 360/91 (67) MC88110 (93)
		Especulativos	PowerPC 602 (95) PowerPC 620 (96) UltraSparc (95) R10000 (96)
Reorden en caso de Faltas de Cache	LOADs adelantan a LOADs		UltraSparc (95) PowerPC 620 (96)
	STOREs adelantan a STOREs		