2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores
Tema 4

Lección 12. Consistencia del procesador y Procesamiento de Saltos

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

ugr | Universidad de Granada

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

ATC
Departamento de Arquitectura y Tecnología de Computadores
UNIVERSIDAD DE GRANADA

Lecciones

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

Bibliografía

- Fundamental
  - Capítulo 3, Secc. 3.3.4, 3.4, 3.5. 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

Contenido de la Lección 12

- Consistencia. Reordenamiento
- Procesamiento especulativo de saltos

Consistencia I

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

- El Final de la Ejecución de la Operación 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** 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 II

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)

Tendencia

Tendencia / Prestaciones

Reordenamiento Load/Store I

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

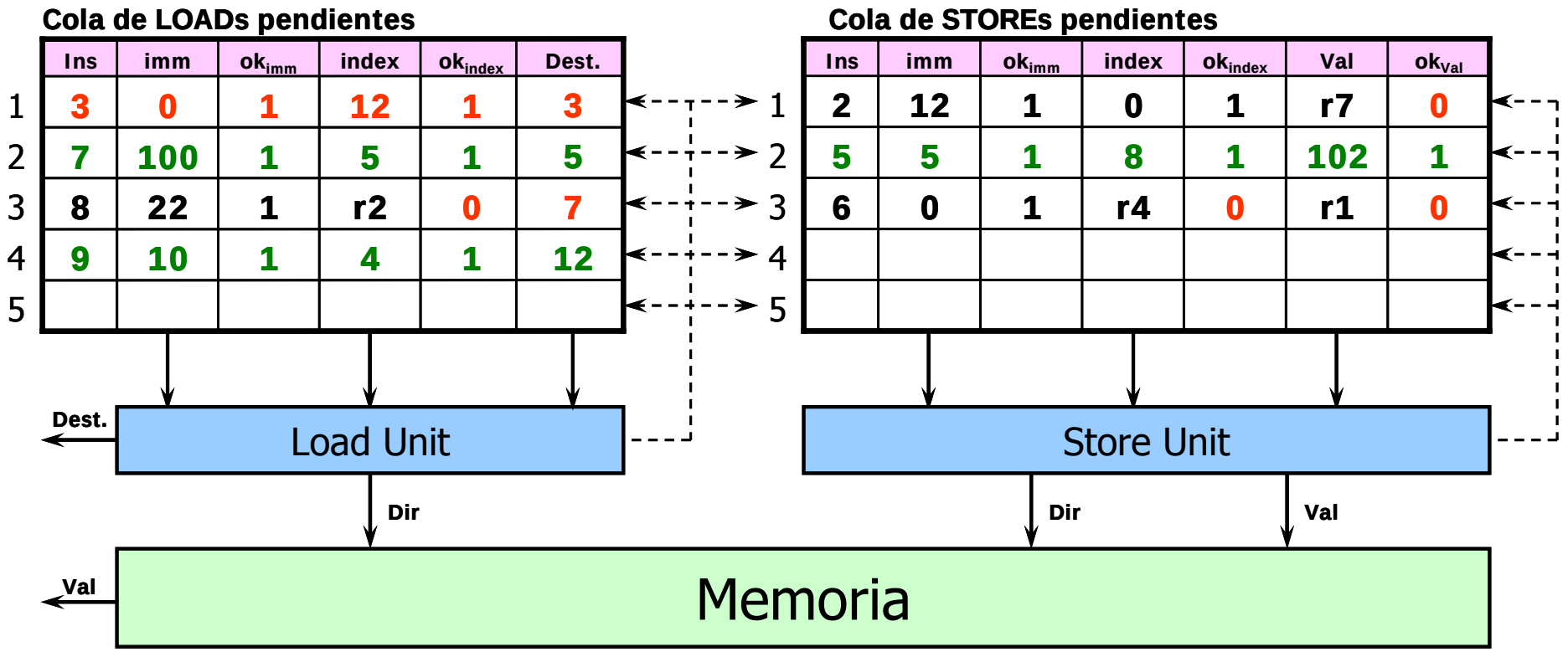
LOAD: - Cálculo de Dirección en ALU o Unidad de Direcciones
- Acceso a Cache
- Escritura del Dato en Registro

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)
- **Permite los 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.

Reordenamiento Load/Store II



- (3) no puede adelantar a (2) porque acceden a la misma posición de memoria
- (5) puede adelantar a (2) porque no acceden a la misma posición de memoria
- (7) y (9) pueden adelantar a (2), (3) y (5), y *si se permiten **LOADs especulativos**, podrán adelantar también a (6)*

Reordenamiento Load/Store III


```
loop: ld   r1, 0x1C(r2)        ld   r1, 0x1C(r2)
      mul  r1, r1, r6          mul  r1, r1, r6        ld   r3, 0x2D(r2)
      st   r1, 0x1C(r2)        st   r1, 0x1C(r2)      mul  r3, r3, r6      addi r2, r2, #1
      ld   r3, 0x2D(r2)                               st   r3, 0x2D(r2)    subi r4, r4, #1
      mul  r3, r3, r6                                                      bnz  r4, loop
      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)

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

ld r3,0x2D(r7)

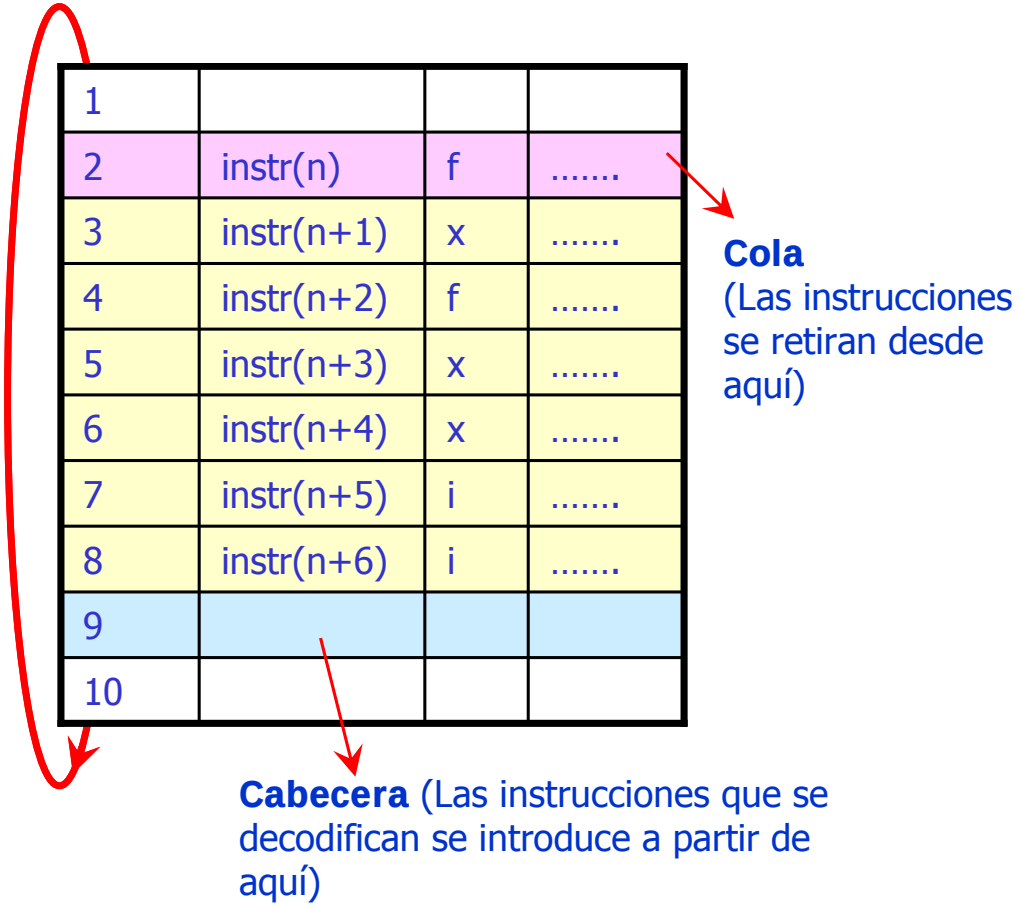
Las direcciones 0x1C(r2) y 0x2D(r7) podrían coincidir.

Se tendría un **load especulativo**

Reordenamiento Load/Store IV

<table>
<tr><td rowspan="3">Reordena-miento debido a adelantos LOADs/STOREs</td><td colspan="2">STOREs adelantan LOADs (R->W)</td><td></td></tr>
<tr><td rowspan="2">LOADs adelantan STOREs (W->R)</td><td>No especulativos</td><td>IBM 360/91 (67)
MC88110 (93)</td></tr>
<tr><td>Especulativos</td><td>PowerPC 602 (95)
PowerPC 620 (96)
UltraSparc (95)
R10000 (96)</td></tr>
<tr><td rowspan="2">Reordena-miento en caso de Faltas de Cache</td><td colspan="2">LOADs adelantan a LOADs (R->R)</td><td>UltraSparc (95)
PowerPC 620 (96)</td></tr>
<tr><td colspan="2">STOREs adelantan a STOREs (W->W)</td><td></td></tr>
</table>

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)**
- 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.

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 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

Buffer de Reordenamiento (ROB) III

Ciclo 7

#	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	-	0	x
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
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 Termina **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

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

Ciclo 13

#	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 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 implementaciones del ROB en procesadores

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

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*

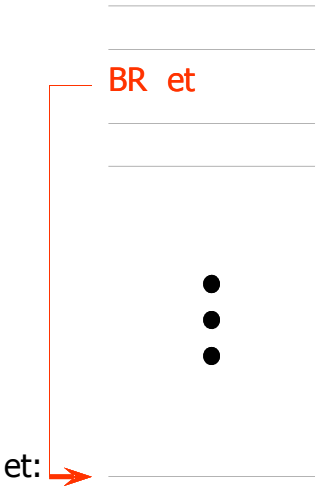
Contenido de la Lección 12

- Consistencia. Reordenamiento
- Procesamiento especulativo de saltos

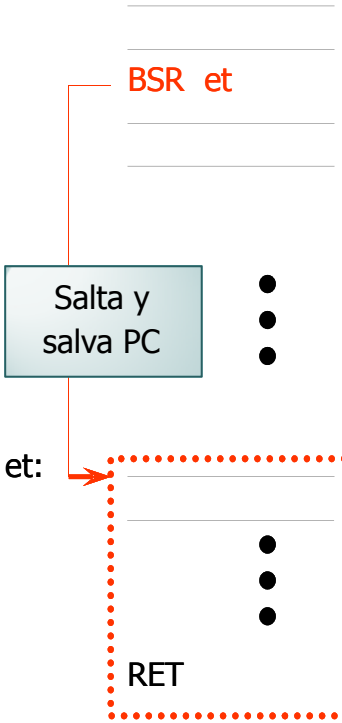
Clasificación de los Saltos

Saltos Incondicionales

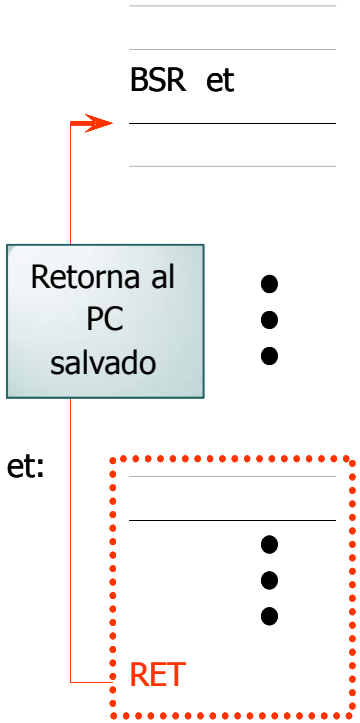
Salto Incondicional



Llamada a Subrutina

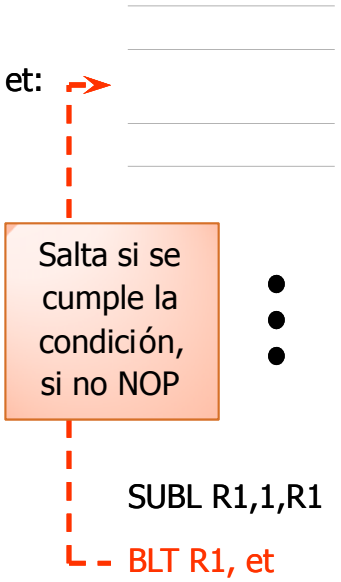


Retorno de Subrutina



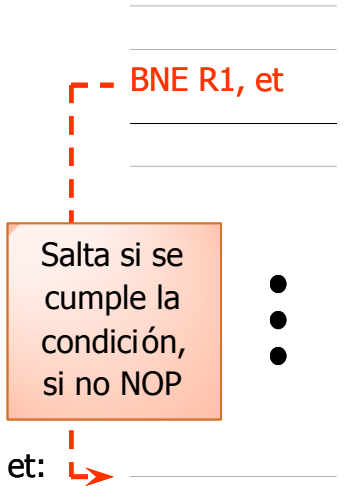
Saltos Condicionales

Condición de Bucle



Salto hacia atrás

Otro Salto Condicional



Salto hacia atrás o hacia adelante

Alternativas para la condición de salto

Estado del Resultado

Existen bits de Estado que se modifican al realizar operaciones o mediante operaciones que comprueban específicamente el valor de los registros

```
add r1,r2,r3
beq cero
div r5,r4,r1
.....
cero:
```

Dependencia que limita las Prestaciones en VLIW y superescalares

Ejemplos:

IBM/360, PDP-11, VAX, X-86, Pentium, PowerPC, Sparc

Comprobación Directa

Los resultados de las operaciones se comprueban directamente respecto a las condiciones específicas mediante instrucciones específicas

Dos Instrucciones

```
add r1,r2,r3
cmpeq r7, r1, 0
bt r7, cero
div r5,r4,r1
.....
cero:
```

Ejemplos:

Am 29000

Una Instrucción

```
add r1,r2,r3
bz r1, cero
div r5,r4,r1
.....
cero:
```

Ejemplos:

Cray, MIPS, MC881X0, HP PA, DEC Alpha

Aspectos del Procesamiento de Saltos en un procesador Superescalar

- 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 e incluso en la captación (si hay predecodificación) .
- 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

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.

Gestión de Saltos Condicionales no resueltos

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 , URPR2)
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 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)
Predicción basada en el Desplazamiento del Salto Si el desplazamiento es positivo (salto hacia delante) se predice que no se toma el salto y si el desplazamiento es negativo (salto hacia atrás) se predice que se toma.	Alpha 21064 (92) PowerPC 603 (93)
Predicción dirigida por el Compilador El compilador es el que establece la predicción fijando, para cada instrucción, el valor de un bit específico que existe en la instrucción de salto (bit de predicción)	AT&T 9210 (93) PowerPC 603 (93) MC88110 (93)

Ejemplo: Predicción Estática en el MC88110

Formato	Instrucción		Predicción
	Condición Especificada	Bit 21 de la Instr.	
bcnd (*Branch Conditional*)	$\neq 0$	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 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

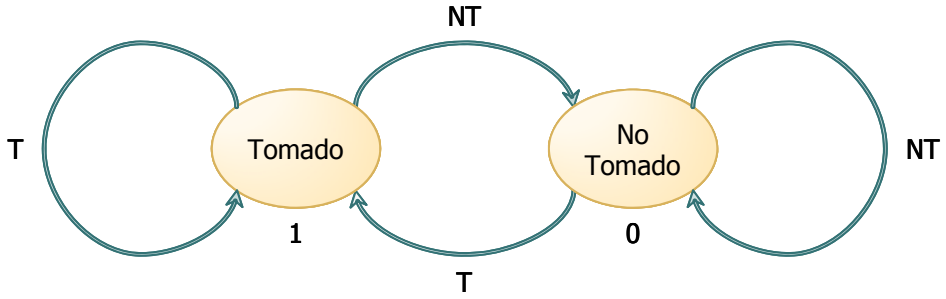
- **Predicción Dinámica Explícita**

  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)

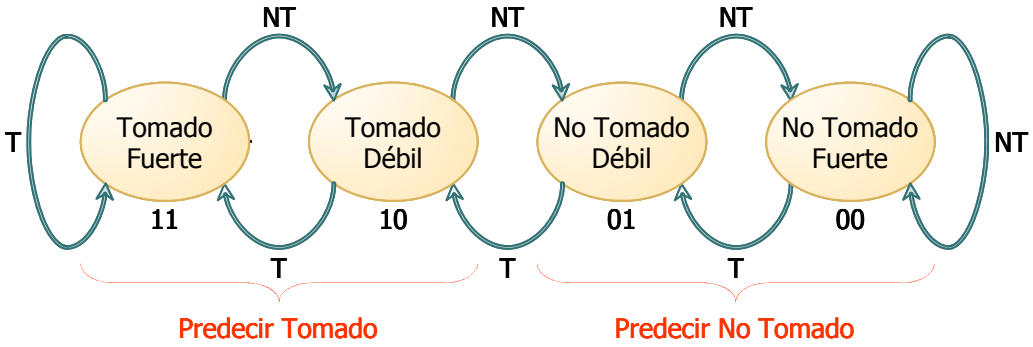


Predicción con 2 bits de historia

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 instrucción (T o NT)

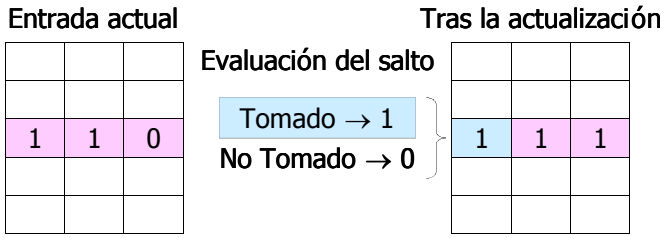


Predicción con 3 bits de historia

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



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)

Recuperación de Predicción Incorrecta

La recuperación de una predicción incorrecta comprende:

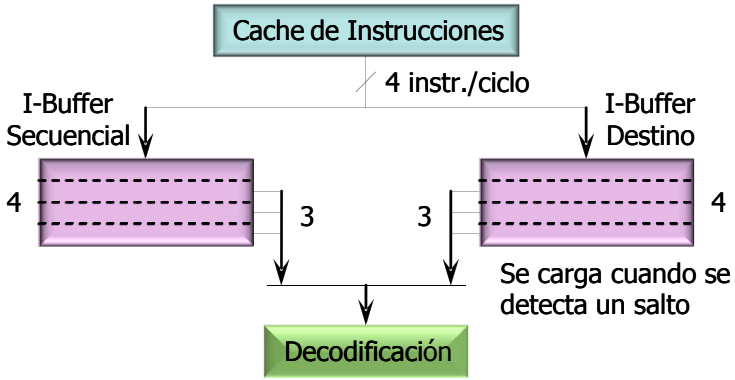
- **Descartar los resultados de la ejecución especulativa**
- **Continuar la ejecución de la secuencia de instrucciones alternativa (la correcta)**

Recuperación desde un salto efectuado:

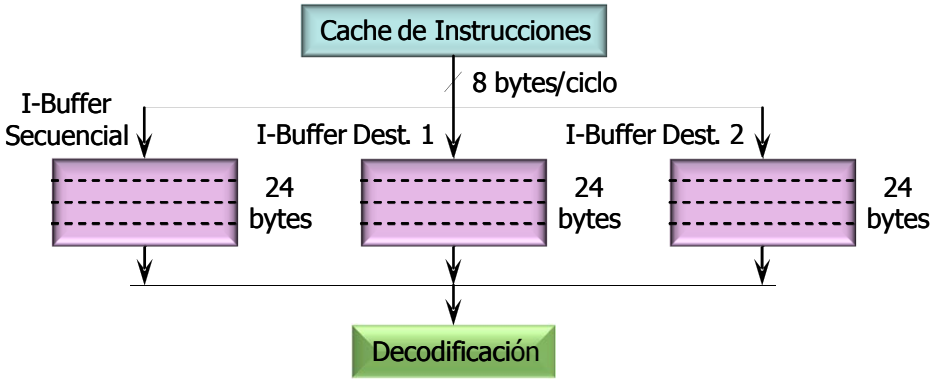
- El procesador debe guardar la dirección de la instrucción siguiente a la de salto para utilizarla si la predicción es incorrecta.
- La recuperación es más rápida si no se descartan las instrucciones que se habían precaptado junto con la de salto

Recuperación cuando no se ha saltado:

- Pre-calcular la dirección de salto y almacenarse para permitir la recuperación.
- La recuperación es más rápida si se precaptan instrucciones de la secuencia que empieza a partir de la dirección a la que se salta.



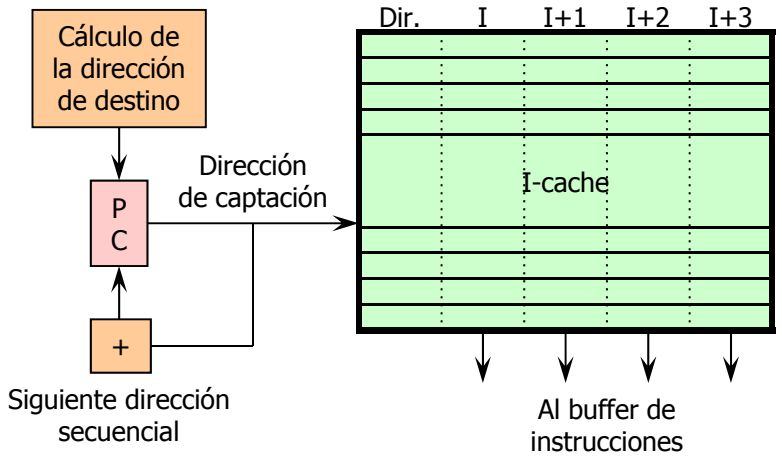
Sistema de 2 buffers para recuperación
(SuperSparc, Power2, Pentium)



Sistema de 3 buffers para recuperación
(Nx586)

Acceso a la Secuencia de Salto I

Si se detecta una instrucción de salto, se calcula su dirección de destino para acceder a la posición de memoria correspondiente si se produce el salto



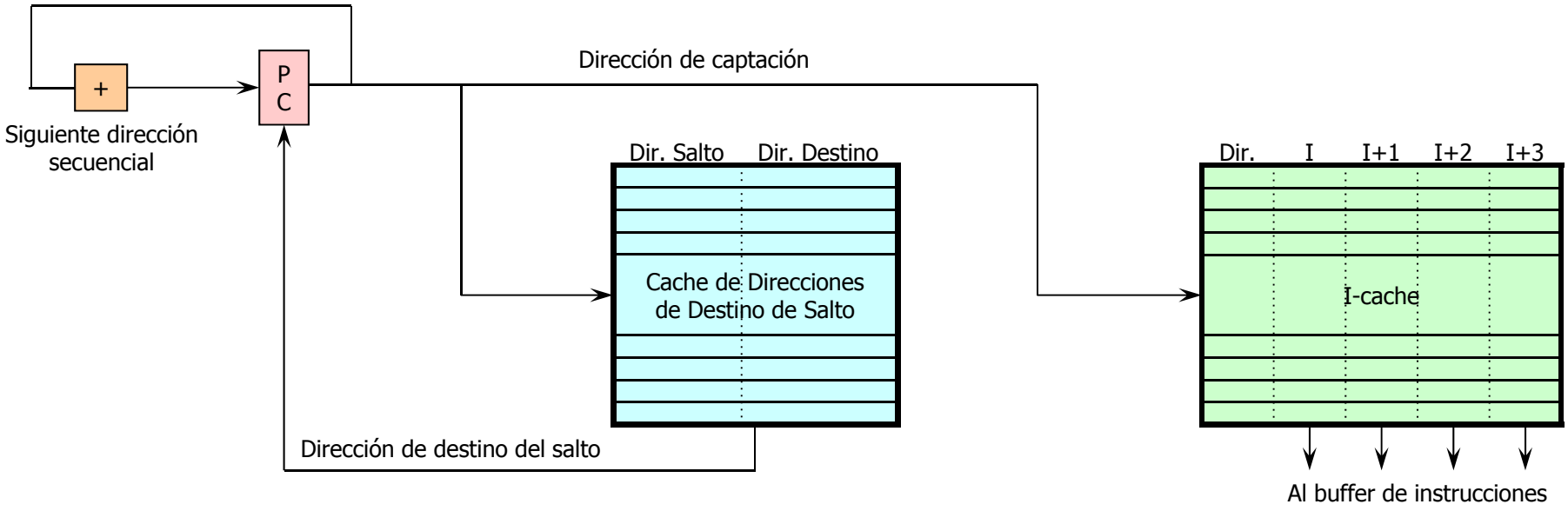
Los saltos condicionales efectuados ('taken') son más frecuentes que los no efectuados ('not taken'). Por ello, sería interesante **reducir al máximo el tiempo de acceso a la secuencia de instrucciones a partir de la dirección de salto y reducir la penalización para las predicciones incorrectas de los saltos efectuados.**

La rapidez de acceso a la secuencia de instrucciones que empieza en la dirección a donde se salta es fundamental para mejorar las prestaciones del esquema de gestión de los saltos condicionales.

Acceso a la Secuencia de Salto II

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.



Instrucciones de Ejecución Condicional (*Guarded Execution*)

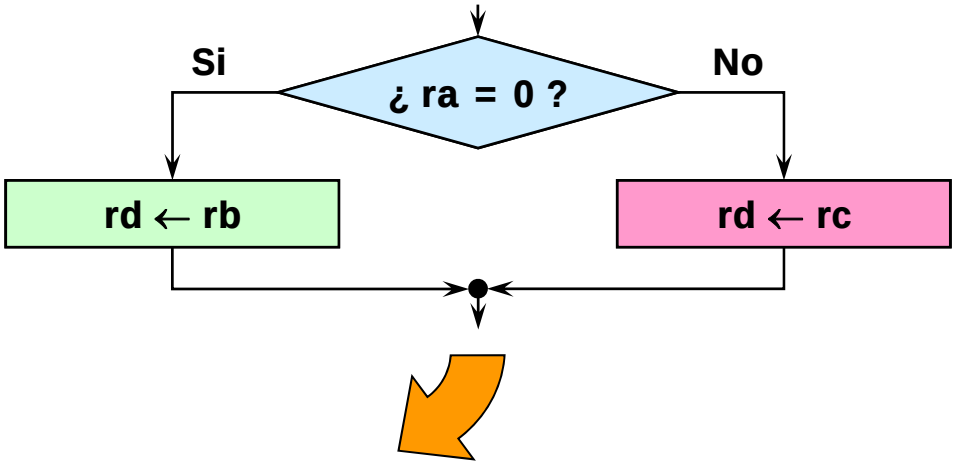
- Se pretende **reducir el número de instrucciones de salto** incluyendo en el **repertorio máquina instrucciones con operaciones condicionales** (*'conditional operate instructions'* o *'guarded instructions'*)
- Estas instrucciones tienen dos partes: la **condición** (denominada *guardia*) y la parte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

Para ampliar …

- Páginas Web:
  - http://www.delphion.com/research
    (Se pueden consultar las patentes sobre elementos de los procesadores. En particular las relativas a ROBs y elementos para mantener la consistencia secuencial en el procesador)
  - http://www.aceshardware.com/Spades/read.php?article_id=51
    (Artículo: J. De Gelas: "The Secrets of High Performance CPUs, Part2")
- Artículos de Revistas:
  - Smith, J.E.; Pleszkun, A.R.: "Implementing precise interrupts in pipelined processors". IEEE Trans. Computers, Vol.C37, pp.562-573. Mayo, 1988.
    (Se propone por primera vez el uso del ROB)

Para ampliar .....

- Artículos de Revistas:
  - Smith, J.E.: “A study of branch prediction strategies”. Proc. Eighth Symp. on Computer Architecture, pp.135-148, 1981.
  - Smith, J.E.; Lee, J.: “Branch prediction strategies and branch target buffer design”. IEEE Computer, pp.6-22. Enero, 1984.
  - Yeh, T.; Patt, Y.N.: “A comparison of dynamic branch prediction that use two levels of branch history”. Proc. 20th Symp. On Computer Architecture, pp.257-266, 1993.
  - Mahlke, S.A.: “A comparison of full and partial predicated execution support for ILP processors”. Proc. 22nd Symp. On Computer Architecture, pp.138-150. Junio, 1995.
  - Smith, J.E.; Plezskun, A.R.: “Implementing Precise Interrupts in Pipelined Processors”. IEEE Trans. On Computers, Vol.37, No.5, pp.562-573. Mayo, 1988.
  - Torng, H.C.; Day, M.: “Interrupt Handling of Out-of-Order Execution Processors”. IEEE Trans. On Computers, Vol.42, No.1, pp.122-127. Enero, 1993.
  - Walker, W.; Cragon, H.G.: “Interrupt Processing in Concurrent Processors”. IEEE Computer, pp.36-46. Junio, 1995.
  - Moudgill, M.; Vassiliadis, S.: “Precise Interrupts”. IEEE Micro, pp.58-67. Febrero, 1996.
  - Samadzadeh, M.; Garalnabi, L.E.: “Hardware/Software Cost Analysis of Interrupt Processing Strategies”. IEEE Micro, pp.69-76. Mayo-Junio, 2001.