UT 2. Computadores segmentados
Tema 2.3 Predicción dinámica de saltos

J. Flich, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València

UNIVERSIDAD POLITECNICA DE VALENCIA

DISCA

Índice

Bibliografía

John L. Hennessy and David A. Patterson.
Computer Architecture, Fifth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5 edition, 2012.

Índice

Motivación

- Dependencias de control $\rightarrow$ Riesgos de control $\rightarrow$ penalización por ciclos de parada, que reduce las prestaciones:

Ejemplo:

15 % de saltos, 75 % efectivos, 3 ciclos de penalización
Con *predict-not-taken* $\rightarrow$ $CPI = 1 + 0{,}15 \cdot 0{,}75 \cdot 3 = 1{,}33$

Problema: En este caso, la tasa de acierto de *predict-not-taken* es sólo de un 25 %.

Objetivo:

Realizar una predicción más precisa del comportamiento de los saltos para tener las instrucciones correctas ya buscadas (e incluso iniciada su ejecución) en el momento de la resolución del salto.

- El comportamiento de distintas instrucciones de salto de un mismo programa no tiene por que ser el mismo.
- El comportamiento de una misma instrucción de salto puede variar a lo largo de la ejecución del programa

$\Rightarrow$ Técnicas de predicción dinámicas (mediante *hardware*).

Clasificación

- ¿Qué se predice? Condición o Condición+Dirección
- ¿Cómo se predice? A partir del PC de la instrucción de salto. Dirección completa o sólo algunos bits.
- ¿Cómo se almacena la predicción? Correspondencia directa o totalmente asociativa.
- ¿Cuándo se realiza la predicción? Antes (IF) o después de decodificar el salto ($>=$ID).

Índice

También llamados Branch History Table (BHT).

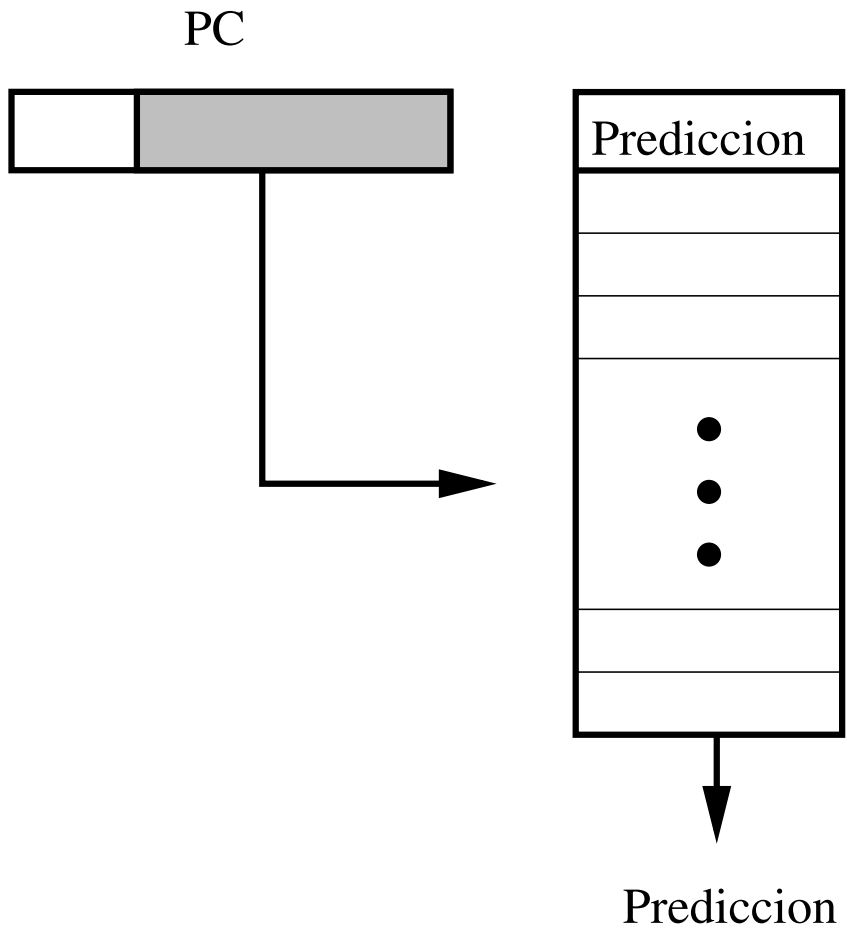
Objetivo:

Predecir la condición de salto.

- Datos:
  - Dirección de la instrucción (de salto).
- Resultado:
  $\Rightarrow$ Salto “salta” (*taken*) o ‘no salta” (*not-taken*)

Esquema

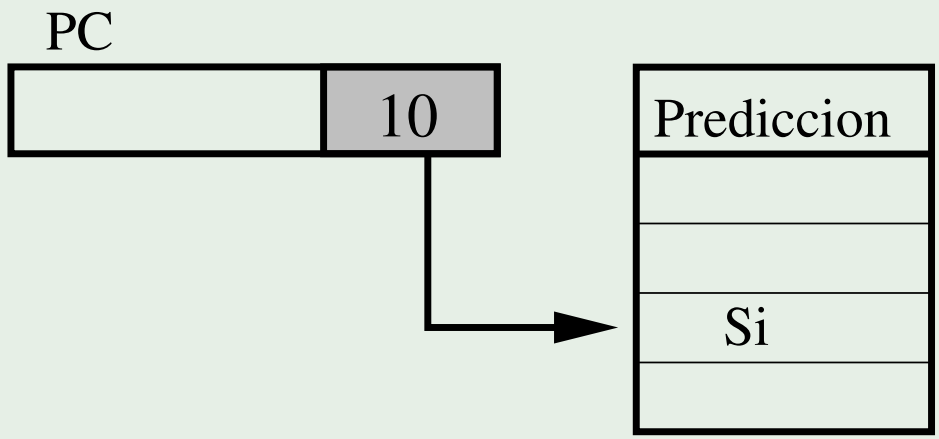
- Tabla indexada con los bits de menor peso de la dirección de la instrucción de salto.
- La entrada de la tabla contiene la predicción para esa instrucción de salto.



Ejemplo:

```
...010   BEQZ D1
......
    D1: DADD R1,R2,R3
......
...100   BNEZ D2
```

Tras `BEQZ D1`:



Implementación

Ubicación de la tabla Varias posibilidades:

- Memoria que se accede en paralelo con la memoria cache de instrucciones durante la fase IF.
- Bits de predicción añadidos a cada bloque de la memoria cache de instrucciones.

Predictor Algoritmo que predice

- Autómata de estados finitos.
- Se cambia de estado según el comportamiento real de la instrucción de salto.
- La predicción ("salta"/"no salta") depende del estado.

$\Rightarrow$ Caso más sencillo: predictor de un bit

- Estado del autómata = Condición obtenida en la última ejecución del salto.
- Predicción = Estado del autómata.

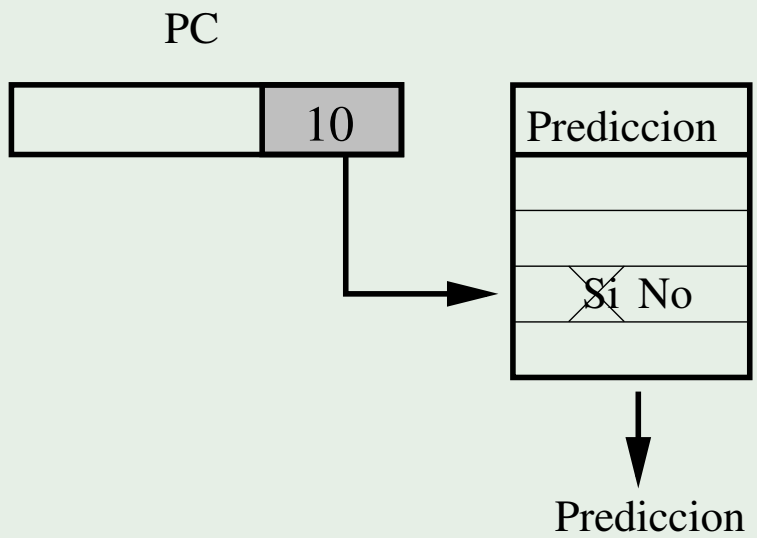
Problema: dos o más saltos en la misma entrada de la tabla

- Dos instrucciones de salto pueden tener la misma parte baja → la predicción puede ser incorrecta si se comportan de forma distinta.

Ejemplo:

```
...010   BEQZ D1
......
    D1: DADD R1,R2,R3
......
...110   BNEZ D2
```

Tras `BNEZ D2`:



- La solución es considerar más bits del PC en el índice de la tabla → tamaño mayor de tabla.

Predicción del salto en los bucles

Las instrucciones de salto que implementan un bucle siguen un patrón predecible.

- Si el bucle tiene n iteraciones, el salto es efectivo $n-1$ veces y una vez (la última iteración) no salta.
- Sin embargo, un predictor de un bit fallará en la primera (el predictor no está entrenado) y en la última iteración.

Ejemplo: Bucle anidado

```
for j := 1 to m do          bucle_j: ...
  for i := 1 to n do        bucle_i: ...
    ...                              ...
  end;                               bnez Ri, bucle_i
end;                                 bnez Rj, bucle_j
```

Comportamiento del salto `bnez Ri, bucle_i`:

j	1				2				3
i	1	2	3	n	1	2	3	n	...
¿Salta?	Si	Si	Si	No	Si	Si	Si	No	
Predicción	-	Si	Si	Si	No	Si	Si	Si	
¿Es correcta?	-	OK	OK	NO	**NO**	OK	OK	**NO**	

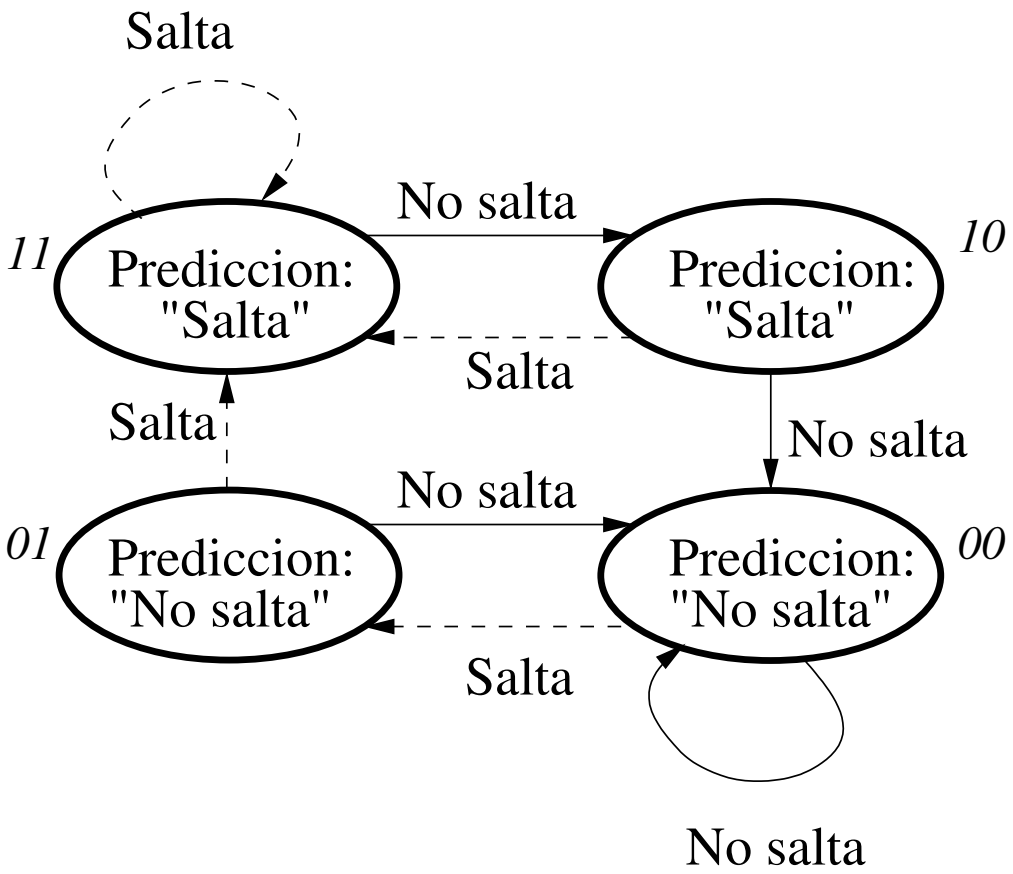
- La predicción del bucle interno falla dos veces por cada iteración del bucle externo.
- Porcentaje de veces que salta = $\frac{n-1}{n} * 100\,\%$
- Porcentaje de acierto en la predicción= $\frac{n-2}{n} * 100\,\%$
- Solución: predictor de dos bits

Predictores de dos bits

- Con dos bits, el autómata tiene cuatro estados:

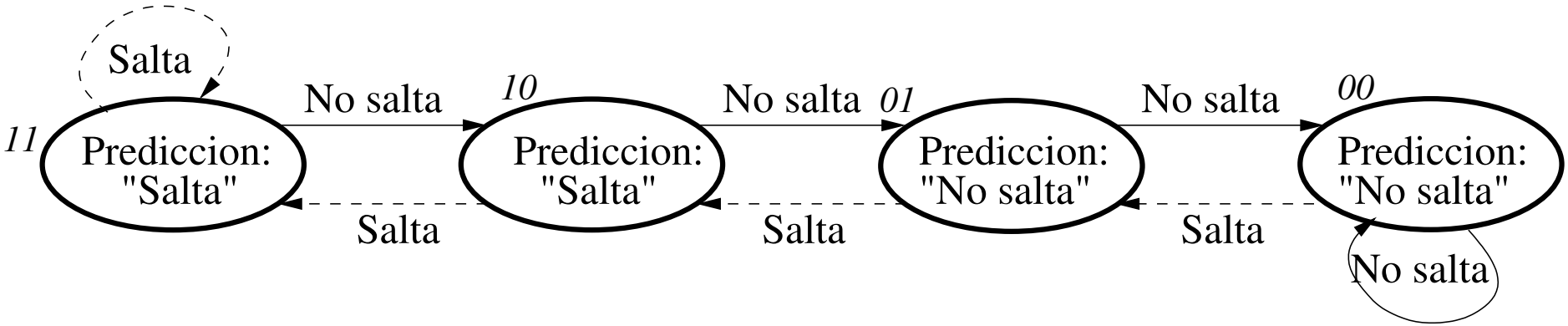
  Strongly not taken (estado 00): clara tendencia a no saltar
  Weakly not taken (estado 01): alguna tendencia a no saltar
  Weakly taken (estado 10): cierta tendencia a saltar
  Strongly taken (estado 11): clara tendencia a saltar
- Idea básica: la predicción debe fallar dos veces antes de ser modificada.
- Los dos diseños más habituales:
  - Predictor de dos bits con histéresis
  - Predictor de dos bits con saturación

Predictor de dos bits con histéresis:



Predictor de *n* bits con saturación:

Cada entrada tiene un valor de *n* bits (0 a 2^n -1):



- Si el valor es $\geq 2^{n-1}$ se predice “salta”
- Si el valor es $< 2^{n-1}$ se predice “no salta”

$\rightarrow$ la predicción debe fallar 2^{n-1} veces antes de modificarla.

Predictores de dos niveles o correlacionados

Predicen en función del comportamiento de la instrucción de salto actual (historia local), y también del de otras instrucciones de salto (historia global).

Ejemplo (eqntott):

Código fuente:

```
if (aa==2)
   aa=0;
if (bb==2)
   bb=0;
if (aa!=bb)
```

Código ensamblador MIPS:

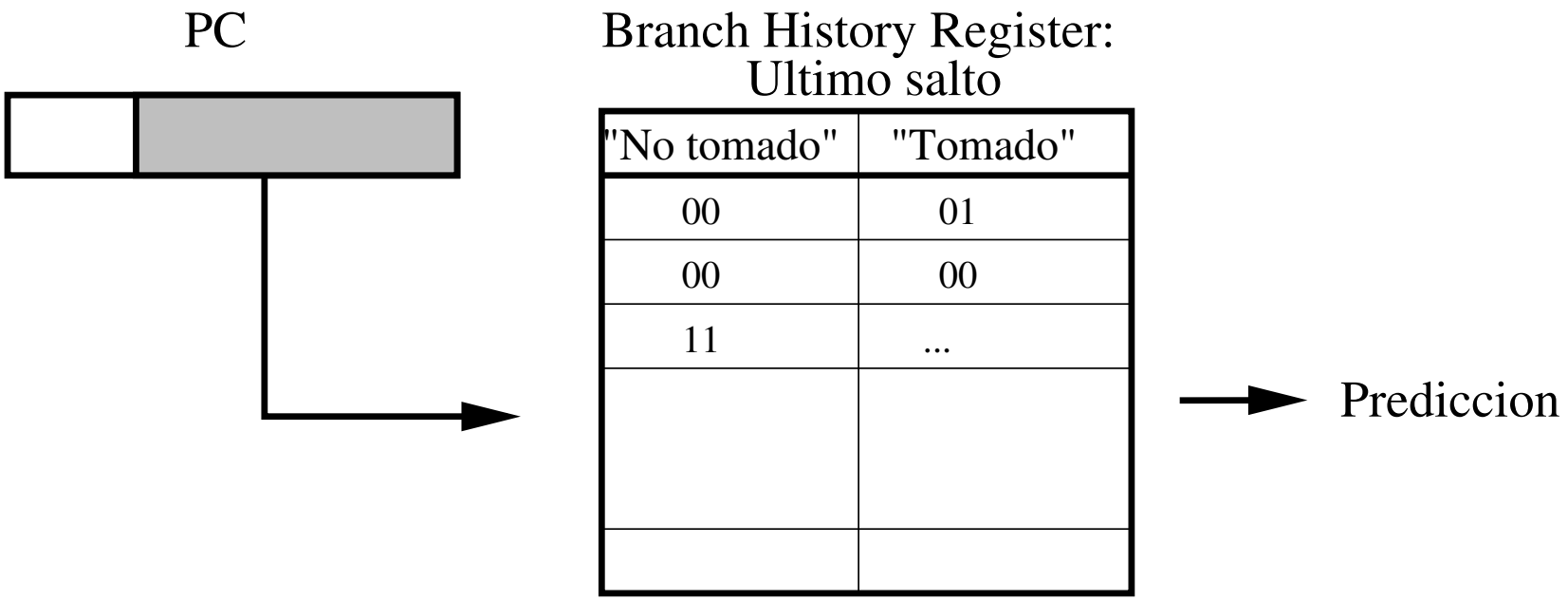
```
    DSUB R3,R1,#2
    BNEZ R3,L1      ; aa!=2 (b1)
    DADD R1,R0,R0   ; aa=0
L1: DSUB R3,R2,#2
    BNEZ R3,L2      ; bb!=2 (b2)
    DADD R2,R0,R0   ; bb=0
L2: DSUB R3,R1,R2   ; aa-bb
    BEQZ R3,L3      ; aa==bb (b3)
```

$\rightarrow$ Si los saltos *b*1 y *b*2 no saltan (entonces `aa` y `bb` serán 0), el salto *b*3 será efectivo.

Predictores de dos niveles o correlacionados (cont.)

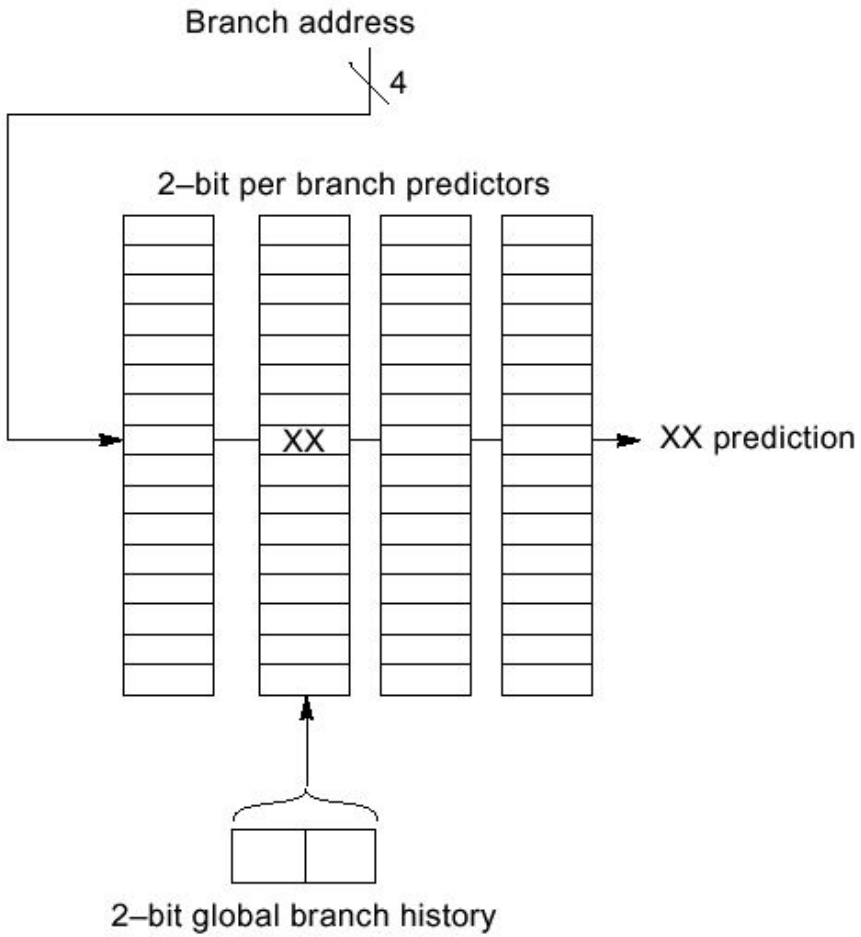
Un predictor (g : *global*, l : *local*) utiliza el comportamiento de los últimos g saltos para elegir entre 2^g predictores de l bits.

- $(0, 2)$: Predictor simple de dos bits.
- $(1, 2)$: Predictor que utiliza el comportamiento del último salto:



Predictores de dos niveles o correlacionados (cont.)

- $(2, 2)$: Predictor que utiliza el comportamiento de los dos últimos saltos:



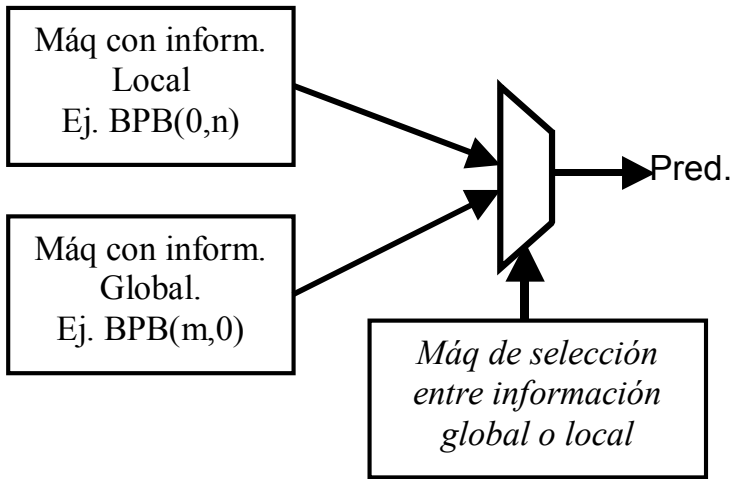
Predictores híbridos

Idea básica

- Cada predictor es apropiado para diferentes patrones.
- Combinando el uso de distintos predictores y aplicando uno o otro según convenga, se puede aumentar la precisión de la predicción.

Predictor híbrido. Varios predictores más un mecanismo de selección.

Ejemplo: *Tournament Predictor:*



Predictores híbridos (cont.)

Mecanismo de selección. Elige el predictor que haya dado mejores resultados hasta el momento.

Implementación del mecanismo de selección: tabla de contadores saturados indexada por la dirección de la instrucción de salto.

- Para dos predictores:

P1	P2	Acciones
Fallo	Fallo	Contador =
Fallo	Acierto	Cont++
Acierto	Fallo	Cont– –
Acierto	Acierto	Contador =

  - Si P2 acierta más que P1 $\rightarrow$ Cont++
  - Si P1 acierta más que P2 $\rightarrow$ Cont– –

- El bit más significativo del contador indica el predictor:

MSB(Contador)	Predictor seleccionado
0	P1
1	P2

Índice

Objetivo:

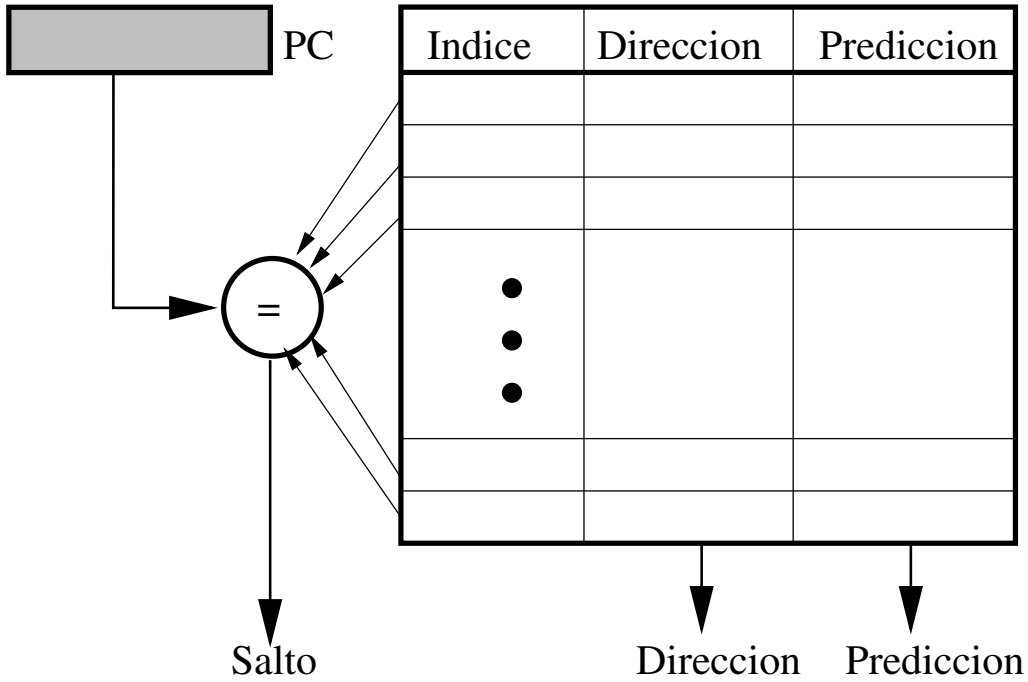
Predecir la condición y la dirección destino del salto

- Datos:
  - Dirección de la instrucción (de salto)
- Resultado:
  - Salto “salta” (*taken*) o ‘no salta” (*not-taken*)
  - Dirección destino del salto (si salta)
  - Si la instrucción es un salto

Esquema

Tabla completamente asociativa, con tres campos:

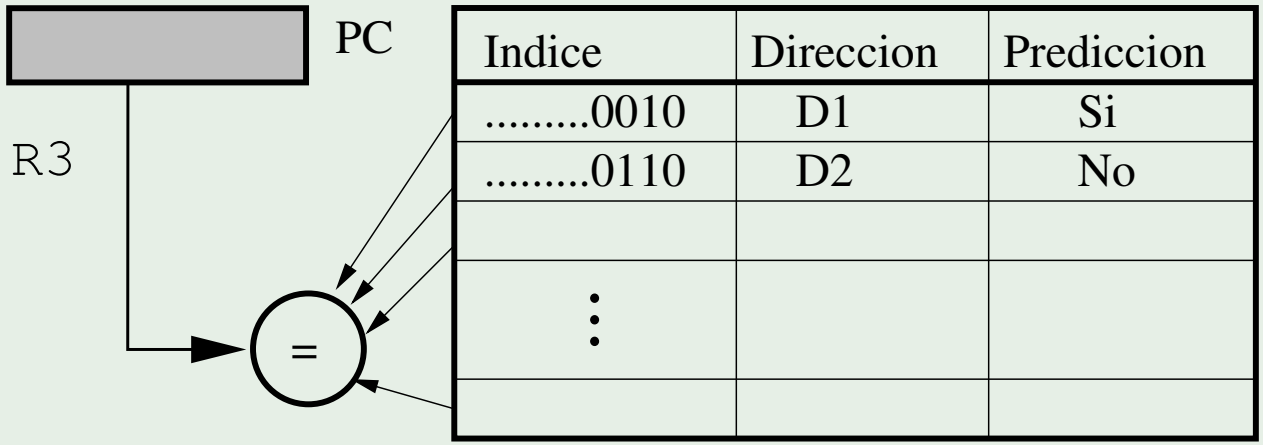
- Índice: dirección de la instrucción de salto
- Predicción: “Salta” o “No salta”
- Dirección: destino del salto



Ejemplo:

Tras `BNEZ D2`:

```
...010   BEQZ D1
......
    D1: DADD R1,R2,R3
......
...110   BNEZ D2
```



Indice	Direccion	Prediccion
.....0010	D1	Si
.....0110	D2	No
⋮		

Eventos en las fases del ciclo de instrucción:

Ejemplo:

Unidad de instrucción segmentada que calcula:

- dirección destino, condición de salto y escritura del PC en ID
- predicción en IF

Comportamiento de *predict-not-taken*:

- Saltos no efectivos:

Instr. salto	IF	**ID**	EX	ME	WB	
PC+1		IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 0 ciclos.

- Saltos efectivos:

Instr. salto	IF	**ID**	EX	ME	WB		
PC+1		IF	ID	EX	ME	WB	
Destino			IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 1 ciclo.

Eventos en las fases del ciclo de instrucción:

IF Búsqueda de la instrucción
// Consulta a la BTB, cero ciclos de penalización

- Si ($\exists$ instrucción en la BTB)
  // Es un salto
  - Si (predicción == “salta”)
    $\Rightarrow$ buscar instrucciones siguientes en la dirección destino del salto proporcionada por la BTB
- En cualquier otro caso, predicción = “no salta”
  // O bien la instrucción $\nexists$ en la BTB
  // O bien la predicción de la BTB es “no salta”
  $\Rightarrow$ buscar instrucciones siguientes en direcciones consecutivas al PC actual

Eventos en las fases del ciclo de instrucción:

ID Decodificación de la instrucción
Si (instrucción == salto)

- Calcular la condición y dirección destino del salto
- Actualización de la BTB
  - Si ($\exists$ instrucción en la BTB)
    $\Rightarrow$ actualizar bit(s) de predicción
  - Sino
    $\Rightarrow$ añadir instrucción a la BTB (inicializar índice, dirección destino y predicción)
- Comprobación de la predicción de IF
  - Si (predicción != condición)
    $\Rightarrow$ cancelar instrucciones lanzadas, buscar instrucciones en la dirección correcta

Funcionamiento:

- El salto no tiene entrada en la tabla y finalmente **no** salta:

Instr. salto	**IF**	ID	EX	ME	WB	
PC+1		IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 0 ciclos.

- El salto no tiene entrada en la tabla y finalmente **sí** salta:

Instr. salto	**IF**	ID	EX	ME	WB		
PC+1		IF	ID	EX	ME	WB	
Destino			IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 1 ciclo.

- Se predice que **no** salta y finalmente **no** salta:

Instr. salto	**IF**	ID	EX	ME	WB	
PC+1		IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 0 ciclos.

- Se predice que **no** salta y finalmente **sí** salta:

Instr. salto	**IF**	ID	EX	ME	WB		
PC+1		IF	ID	EX	ME	WB	
Destino			IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 1 ciclo.

- Se predice que **sí** salta y finalmente **no** salta:

Instr. salto	**IF**	ID	EX	ME	WB		
Destino		IF	ID	EX	ME	WB	
PC+1			IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 1 ciclo.

- Se predice que **sí** salta y finalmente **sí** salta:

Instr. salto	**IF**	ID	EX	ME	WB	
Destino		IF	ID	EX	ME	WB

$\Rightarrow$ Penalización: 0 ciclos.

Comparación con predict-not-taken:

Predicción	Condición	BTB	*pnt*	BTB es ... que *pnt*
no	no	0	0	igual
no	sí	1	1	igual
sí	no	1	0	peor
sí	sí	0	1	mejor

$\Rightarrow$ Importancia de la precisión de la predicción: tasa de acierto del predictor alta.

Atención:

En la práctica, la condición de salto puede depender de una instrucción previa de alta latencia y, por tanto, tardar mucho tiempo en conocerse. En ese caso, la mejora proporcionada por BTB sería muy superior.

Algunos aspectos de implementación:

- La tabla almacena la dirección completa de la instrucción de salto, ya que se accede en la etapa IF, *antes de* decodificar la instrucción. Sólo debe haber un acierto si la instrucción es *seguro* de salto.
- La tabla no tiene por qué ser totalmente asociativa. Podría ser asociativa por conjuntos (los bits de menor peso del PC indexan el conjunto y no se almacenan en la tabla).
- La BTB y el predictor están desacoplados:
  - BHT que almacena los bits de predicción y suele ser grande.
  - BTB que almacena la dirección destino de los últimos saltos efectivos. El tamaño de la BTB suele ser más pequeño.

  Si la predicción de la BHT indica “salta” y hay acierto en la BTB, se comienza la búsqueda de instrucciones en la dirección destino.

Predictores de salto en continua evolución:

- Tienen gran importancia en las prestaciones de los procesador actuales que ejecutan múltiples inst./ciclo.
- La importancia está relacionada con la penalización de salto y el número de instrucciones que busca por ciclo el procesador.
  - Por ej., si la penalización es de 8 ciclos y se buscan 4 inst./ciclo, se cancelan 32 inst. en cada fallo del predictor.
  - Necesidad imperiosa de predictores con alta precisión.
- ¿Cómo mejorar la precisión?
  - ... sabiendo que un predictor de salto *aprende el patrón* a partir del PC del salto y de la *historia de salto* (patrones T/NT) ?
  - 1. Elegir entre múltiples predictores. Puede utilizarse un predictor híbrido o historias de salto de distinta longitud.
  - 2. Mejorar la captura de patrones.
    - Los patrones de salto varían con la evolución de las aplicaciones.
    - Ajustar la longitud del registro de historia no es fácil. Historias más largas capturan patrones difíciles de predecir, pero tarda más en aprender y reduce la precisión de los saltos fáciles de predecir.