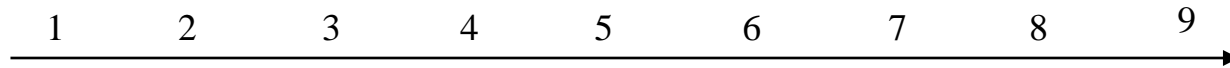


Arquitectura de Computadoras

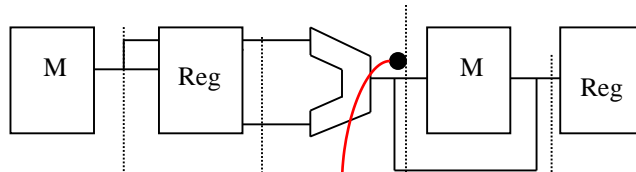
Predictores de Saltos

Breve repaso: CPI de un salto

ciclos



Beq \$1, \$3, 60



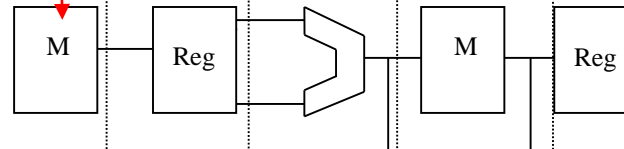
NOP



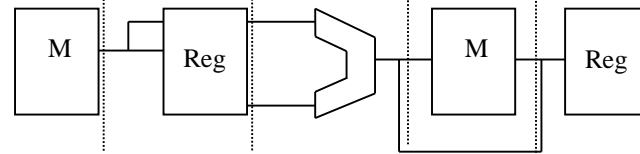
NOP



And \$12, \$2, \$5



Or \$13, \$6, \$2



El CPI de la
instrucción de
salto es 3

Latencia=2

Breve Repaso: Ejecución de un salto

IR<---MEM[PC]; PC<--PC+ 4;

BusA<--R[rs]; BusB<--R[rt];

dst<--- PC+4+ Signext(inm16)*4; Z<-- (BusA==BusB) ;

si (Z) entonces PC<---dst;

- Cálculo de la dirección efectiva: **EX**
- Evaluación de la condición : **EX**
- Ejecución del salto: cambio del PC: **MEM**

Breve repaso: CPI de un salto

Alternativa: suponer no realizado.

ciclos 1 2 3 4 5 6 7 8 →

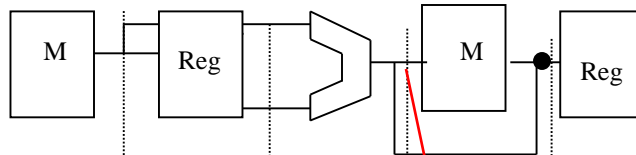
Suponemos que el salto no se realiza

Si es correcto, CPI del salto es 1.

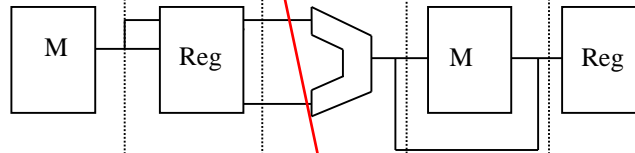
Si es incorrecto...

Eliminar !!

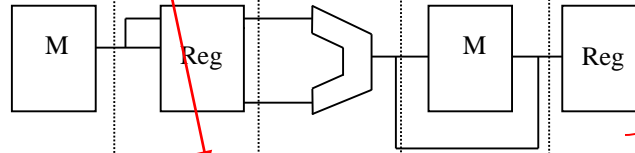
Beq \$1, \$3, 60



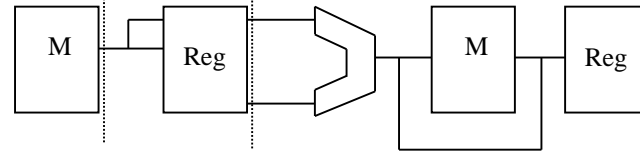
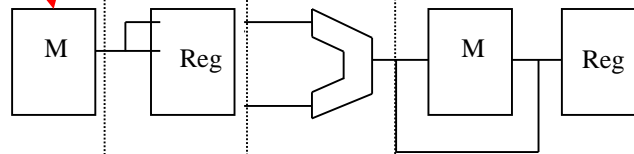
And \$12, \$2, \$5



Or \$13, \$6, \$2

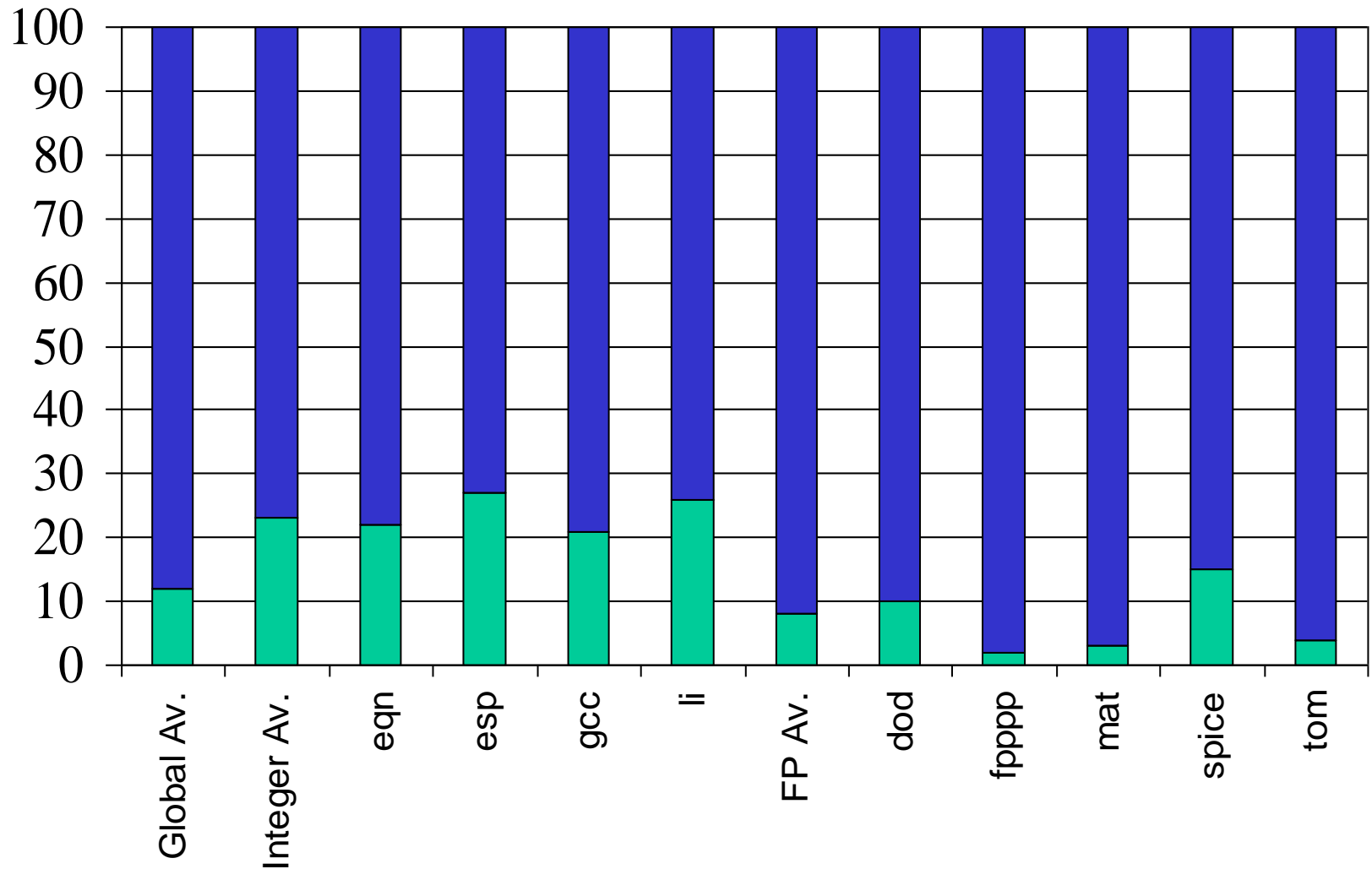


lw \$4,50(\$7)

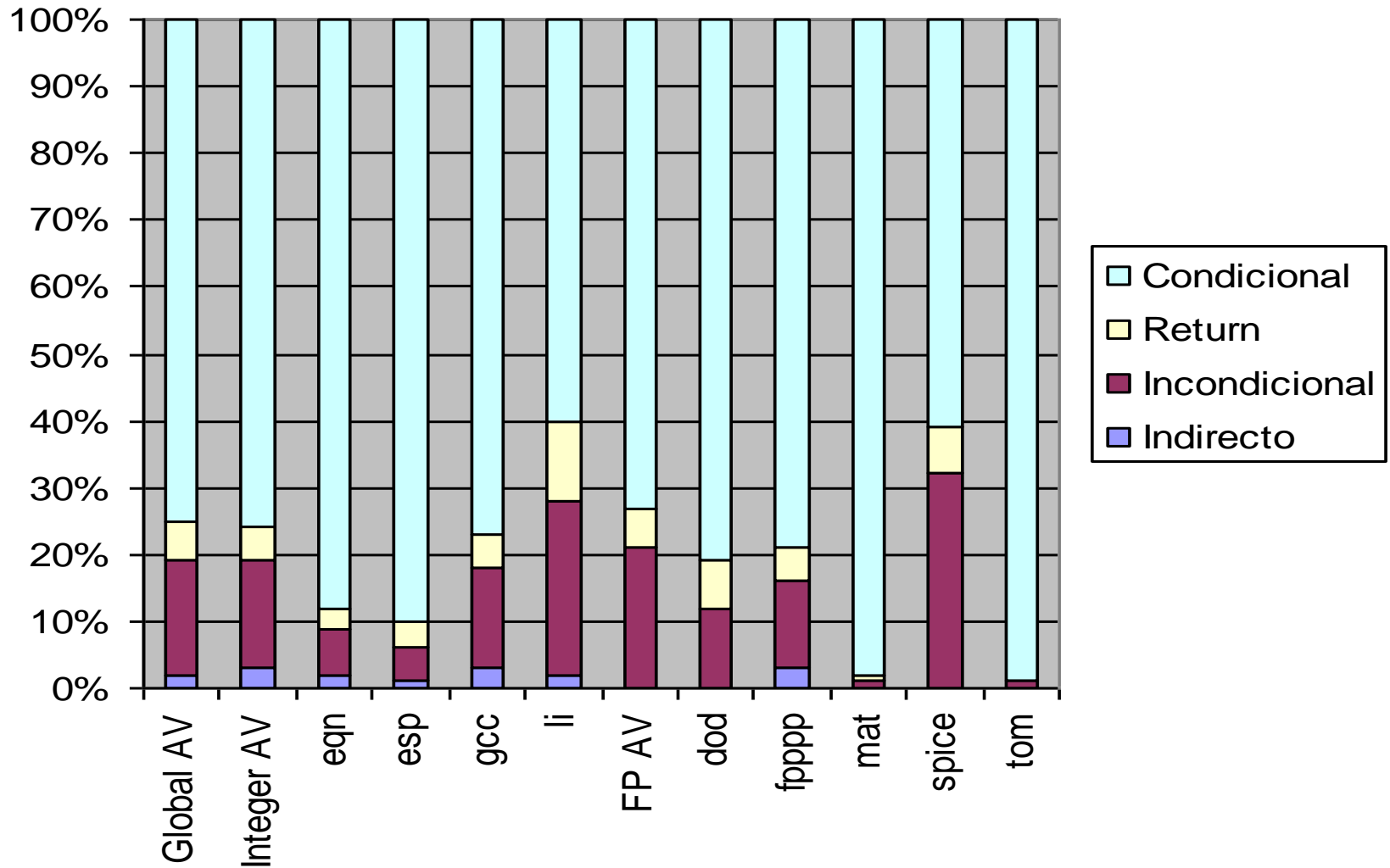


Estadísticas sobre saltos

¿Cuántos saltos podemos esperar en un programa?



Distribución de las instrucciones de salto



Performance

Solamente por los saltos, perdemos aproximadamente un 20% de rendimiento.

No se están considerando dependencias de datos => el rendimiento es aun peor.

Saltos: ¿dónde encontrarlos?

Condición de ciclos.

Adentro de un ciclo, determinando qué proceso hay que realizar sobre el dato.

Construcciones del tipo IF - THEN - ELSE.

Comportamiento esperado: condición de un ciclo

While <cond>

...

...

...

End while.



...

beq end-while

...

...

...

end-while:

...

Salto
fuertemente
no tomado

do

...

...

...

while <cond>



Do:

...

...

...

...

beq do

...

Salto
fuertemente
tomado

Comportamiento esperado: adentro de un ciclo

While <cond(i)>

...

switch(f(i))

case a:

...

case b:

...

End select;

...

End while.



Comportamiento con patrones
muy definidos.

Ejemplo: $(0 \ 1)^n$

Comportamiento esperado: IF - THEN -ELSE

If (vacío)

st1

else

st2

end if

Salto
fuertemente
tomado

If (!vacío)

st1

else

st2

end if

Salto
fuertemente
no tomado

If ((i mod k)==0)

st1

else

st2

end if

Salto con
patrón $(1\ 0^{k-1})^n$

Cuantitativamente...

- Es aproximadamente igual la cantidad de saltos hacia adelante y hacia atrás.
- Aprox. 50% de los saltos hacia adelante son tomados.
- Aprox. 90% de los saltos hacia atrás son tomados.

¿Qué está pasando?

En promedio, un 65-70% de los saltos condicionales son tomados.

La política “Suponer no tomado” no favorece el caso común.

¿Cómo mejorar la performance?

Dos alternativas:

- o cambiamos los programas para que haya mas saltos no tomados.

- o suponemos “Always taken”.

Otras soluciones...

Cambiando El Software

Ya que el procesador tiene predicción fija “Not taken”, hagamos que el código tenga mas saltos NO tomados.

Esto lo puede hacer:

- El **programador**, cambiando las condiciones de los IFs y cambiando los DO-LOOP por While-Wend.
- El **compilador** mismo, conociendo la política que usa el procesador o utilizando información de profiling.

Cambiando el software: papel del compilador

Conociendo la política que usa el procesador, puede generar código mucho mas eficiente.

Utilizando información del run-time del código, se puede recompilar para mejorar la performance.

Problemas:

- Sólo se asegura mejora de performance para los inputs que se usaron para el profiling.
- Hay que recompilar.

Alternativa hardware: suponer tomado

¿Costo de la mejora?

- BTB (Branch Target Buffer): la primera vez que nos encontramos con un determinado salto no se conoce la dirección efectiva de la rama “taken”. No se puede suponer tomado la primera vez. La BTB almacena la dirección efectiva del salto para la próxima vez que se lo tome.
- Recordar el PC secuencial hasta determinar la condición del salto.
- Poder anular las instrucciones especuladas.
- Problemas con saltos condicionales indirectos.

compilador+hardware

Usar un bit en el código de operación que indique:

- Especular usando la dirección secuencial.
- Especular usando la dirección destino del salto.

Para determinar el bit, usar información de profiling.

Performance:

Entre 88 y 94% de los saltos no tienen penalización.

Análisis de situación

Sin apoyarnos en el compilador la pérdida de performance es importante.

Cuando se usa una política fija crece la importancia del compilador y el programador.

Al usar profiling y especulación asistida por el compilador la situación mejora bastante.

¿Qué pasa en procesadores mas “reales”?

Procesadores Superescalares

Procesadores
Tradicionales

Procesadores
Escalares

Procesadores
SuperEscalares

Inicio Secuencial,
Ejecución Secuencial

Inicio Secuencial,
Ejecución Paralela

Inicio Paralelo
Ejecución Paralela

No
Segmentados

Segmentados o
múltiples UF

Múltiples UF
segmentadas

Procesador supersegmentado

Concepto básico: muchas mas etapas.

Segmentar las etapas del procesador segmentado básico.

Etapas mas cortas => tiempo de ciclo mas corto.

Mas etapas => mas instrucciones en paralelo.

Mas etapas => mas problemas de dependencias de datos

Mas etapas => MAYOR penalización en los saltos.

Penalización mas razonable para considerar: entre 4 y 8 ciclos.

Y si pensamos en un superescalar... Mejor ni hablar.

Definición: Bloque Básico

Secuencia de instrucciones, de las cuales sólo la última es una que puede modificar el secuenciamiento implícito.

Importancia: la ejecución de las instrucciones de un bloque básico sólo está limitada por las dependencias de datos entre ellas.

32 jmp 60

36 sub \$10, \$4, \$8

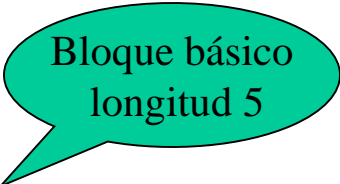
40 and \$12, \$2, \$5

44 or \$13, \$6, \$2

48 add \$14, \$2, \$2

52 beq \$12, \$13, 36

56 sw \$15, 100(\$2)



Bloque básico
longitud 5

Como mejorar

Tener en cuenta:

- La mayoría de los saltos tienen un comportamiento bastante marcado.
- Una política fija sólo favorece o bien a los tomados o bien no tomados.

Idea:

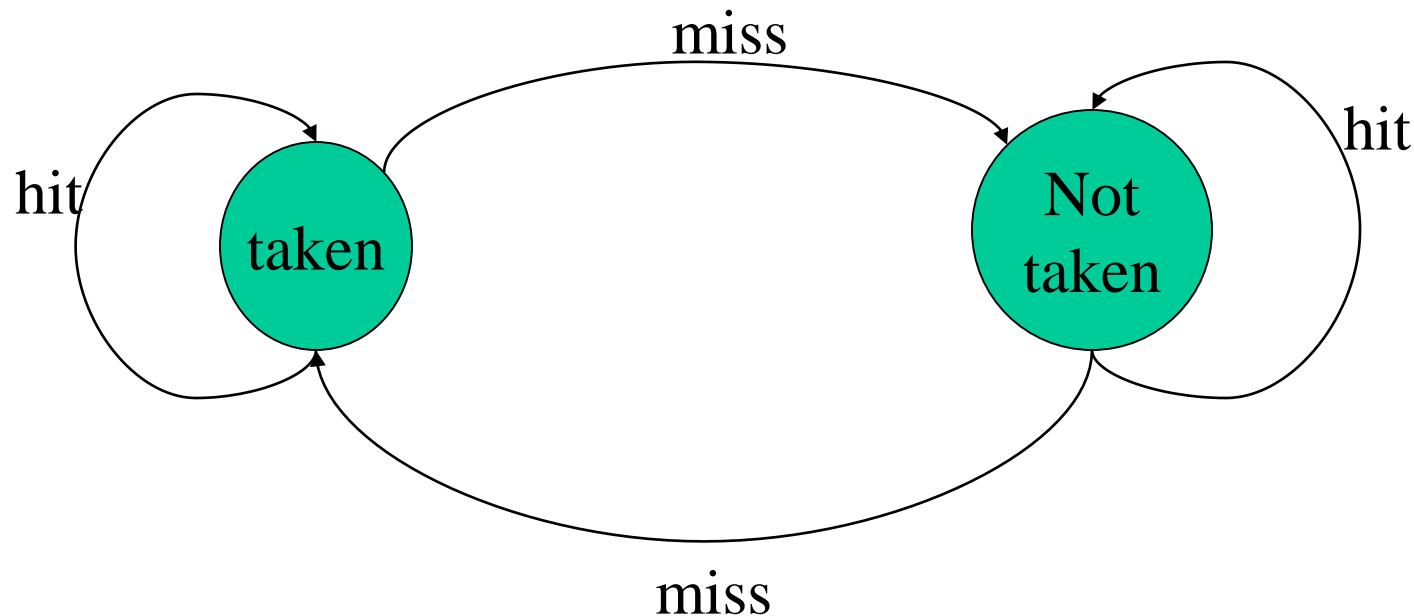
- Recolectar información dinámicamente para especular la próxima instrucción a ejecutar con mayor precisión.

Un esquema simple: 1 bit counter

Idea: una vez que se ejecuta, se va a repetir.

Guardar 1 bit de información para cada salto

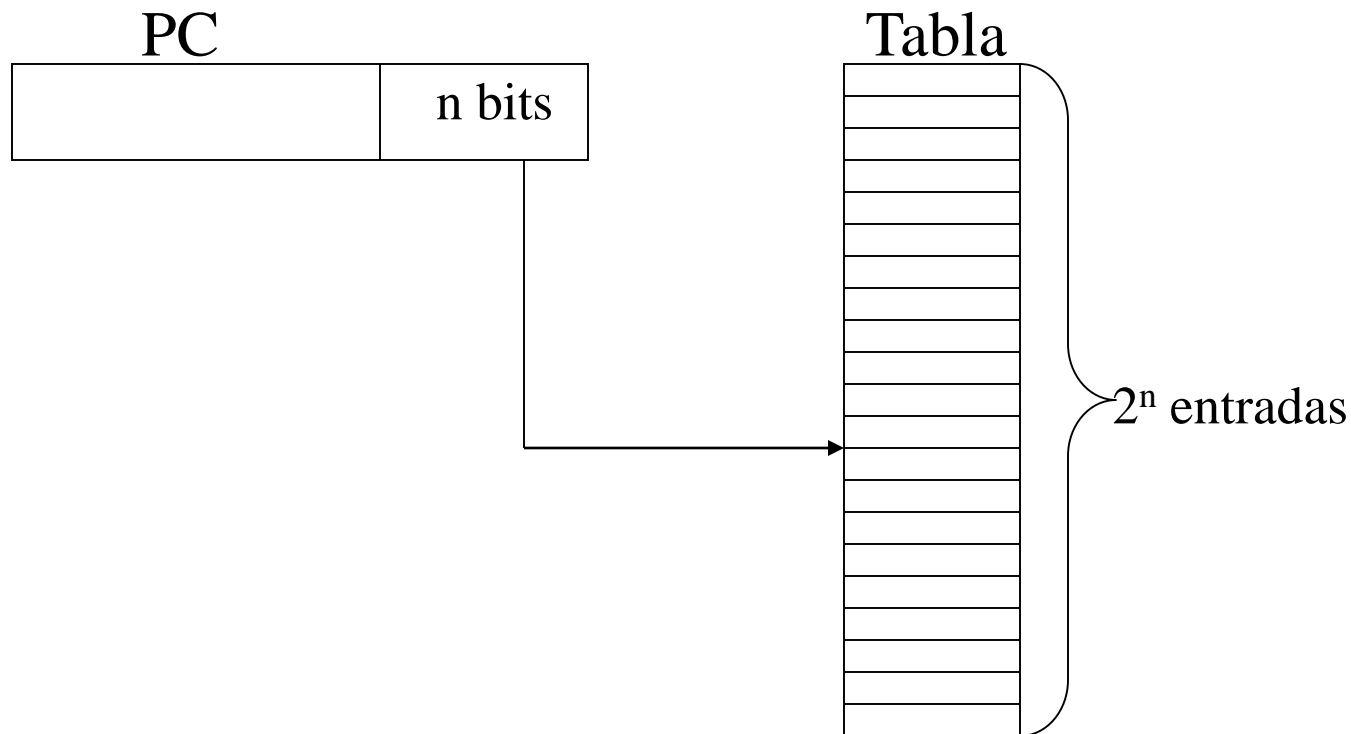
- 1 \Rightarrow Predecir tomado
- 0 \Rightarrow Predecir no tomado



Un esquema simple: 1 bit counter

Decisiones a tomar: cómo identificar cada salto!

Idea: bits mas bajos del PC. ¿Por qué no los mas altos? ¿Todo el PC?



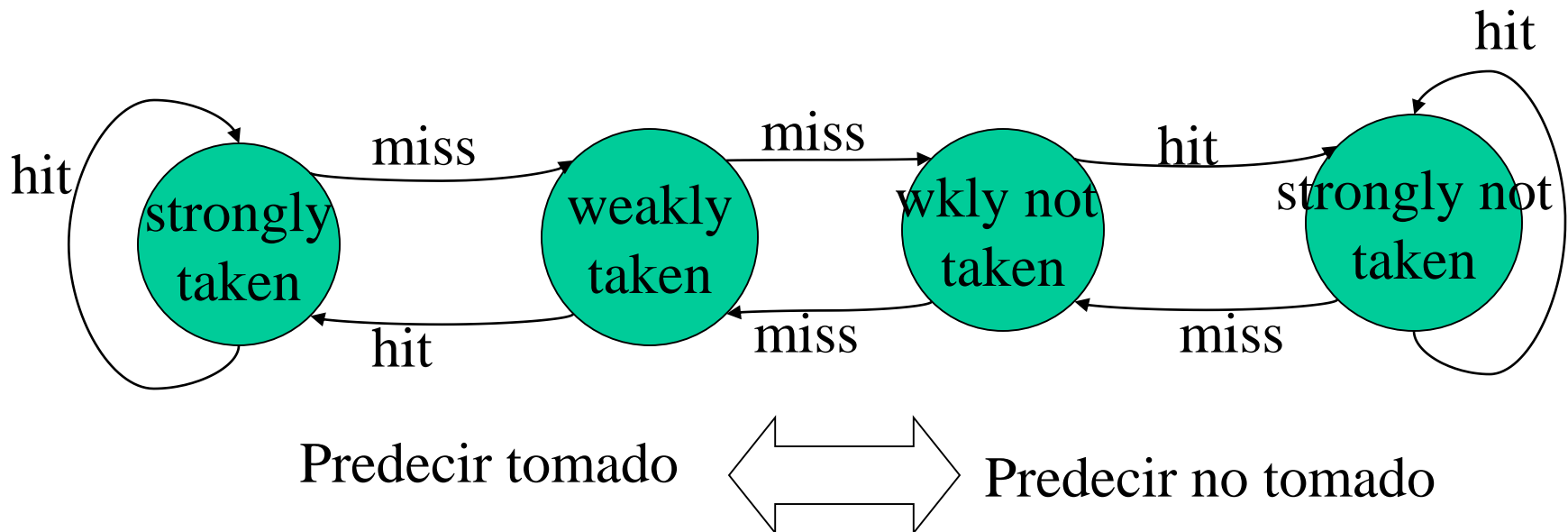
¿Qué problemas tiene el esquema?

Mejorando el esquema simple

Principio: generalizar el concepto utilizado en 1-bit counter.

Idea: en lugar de 1bit usar n bits.

Ejemplo clásico: 2-bit counter



Analizando el desempeño

Deben producirse dos predicciones incorrectas antes de modificar la predicción => tarda en reaccionar a los cambios.

Bueno para loops anidados!

```
FOR K=1 TO N
```

```
    FOR J=1 TO N
```

```
        C(J)=C(J)+A(K)*B(K,J)
```

```
Lk:
```

```
    Lj:
```

```
        BNEZ Rj, Lj
```

```
    BNEZ Rk, Lk
```

Analizar comportamiento de los dos predictores en este código.

Performance

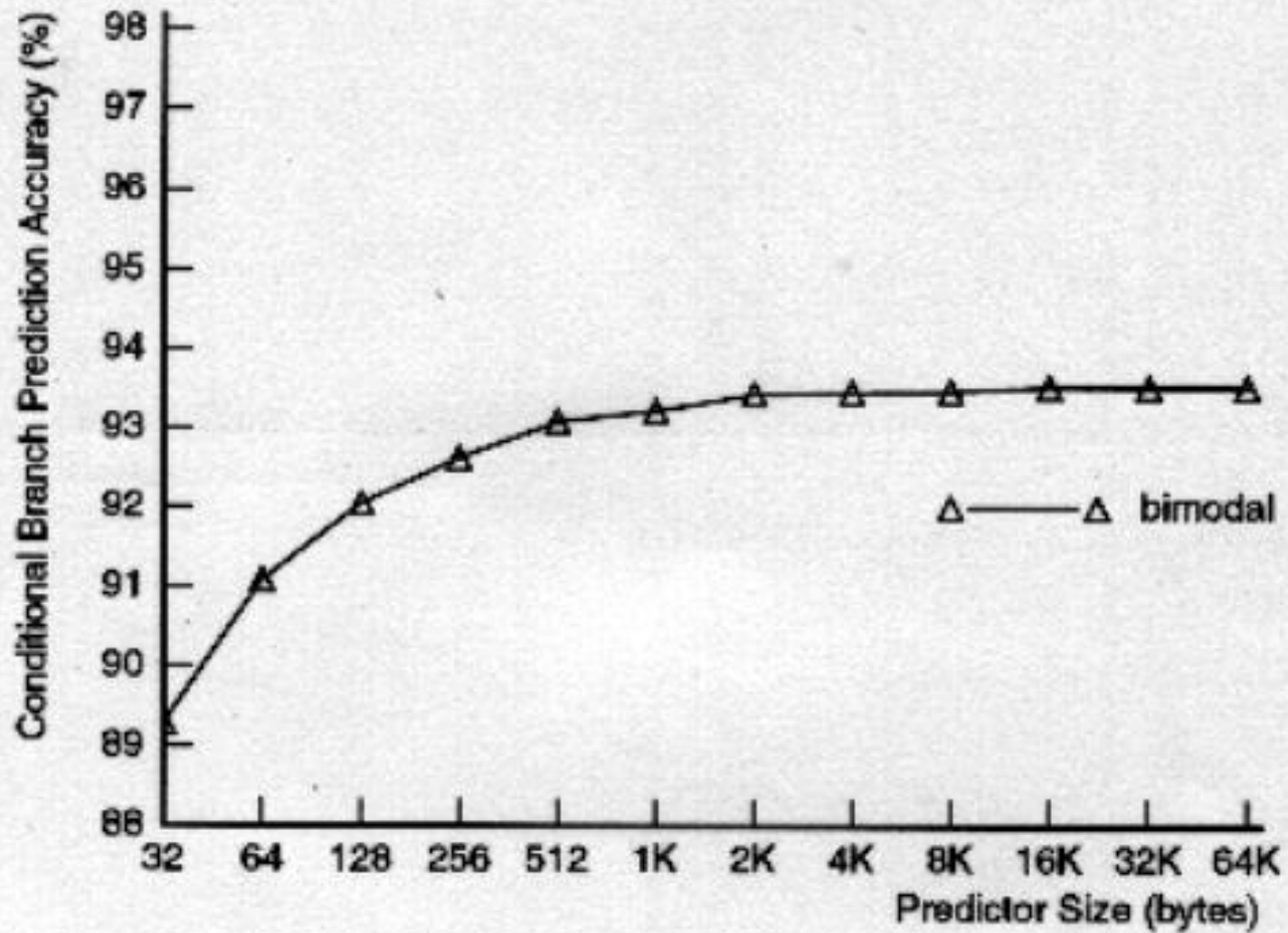


Figure 3: Bimodal Predictor Performance

Patrones de ejecución

El predictor no reconoce patrones, sólo aprovecha la tendencia de un salto a mantenerse en un sentido.

Supongamos el siguiente patrón: $(1101)^n$.

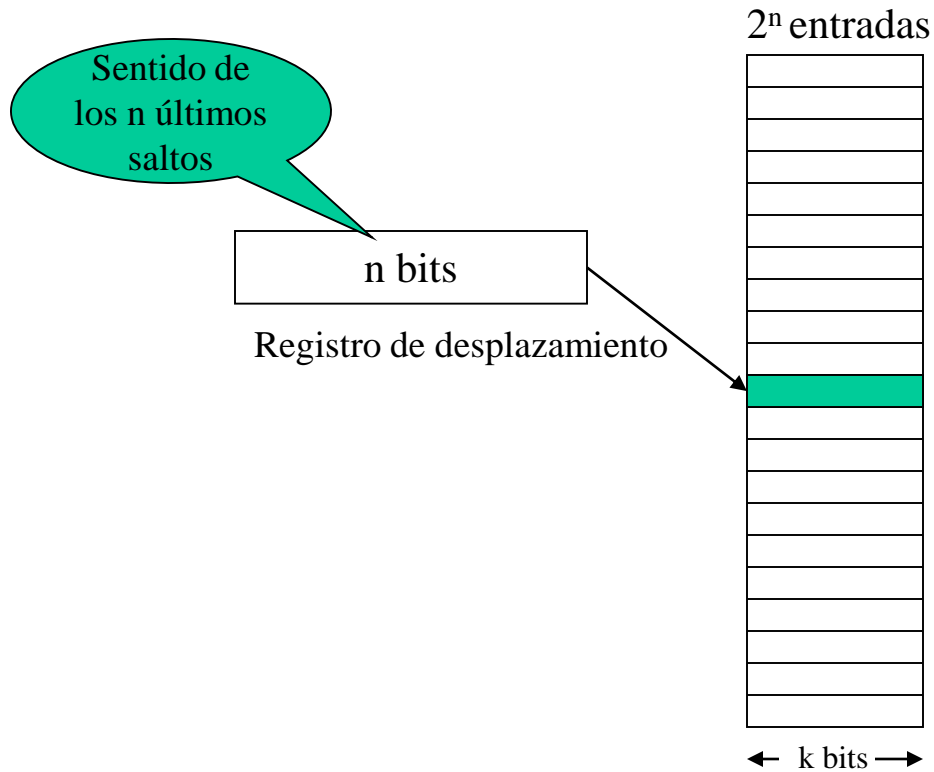
Patrón de longitud 4 \Rightarrow conociendo los últimos 3 podemos predecir:

Sentido	Predicción
110	1
101	1
011	1
111	0

Idea: guardar la historia de los últimos n saltos y deducir a partir de eso que es lo que va a suceder a continuación.

Predicción Global

Concepto: el camino tomado por un salto no solo depende de su historia sino que además depende de los saltos anteriores.



Análisis: Predictor Global

Ventaja:

- Aprende a seguir patrones.
- Implementación sencilla.
- Aprovecha la correlación entre saltos distintos.

Desventajas:

- Tarda en aprender.
- No guarda información sobre la ubicación de los saltos.

Uso en procesadores reales

Siempre no tomado: i486, R3000, MicroSparc, HyperSparc

Siempre Tomado: SuperSparc

Backward taken, Forward not taken: PA 7x00, MicroSparc2

Compiler: PowerPC 601, PowerPC 603, Power2, R4x00, i960

1-bit: AMD K5 (1K), R8000 (1 K), 21064 (2K), 21066 (2K)

2-bit: PA-8000 (256), Pentium (256), PPC604 (512), R10000 (512), UltraSparc (512), Cyrix M1 (256), PPC 620 (2K), Nx586 (2K), 21164 (2K), 21064A (4K)

Two-Levels : P6.

Resumiendo

La mayoría de los saltos tienen un comportamiento bastante marcado hacia un camino.

El camino a tomar por un salto depende también de los saltos anteriores.

Se puede considerar la historia global de los saltos (predictor global) o la historia del salto aislado (predictor local).

.