

# Arquitectura e Ingeniería de Computadores



## TEMA 3

ILP, Panificación dinámica,  
Predicción de saltos, Especulación

**D**EPARTAMENTO DE  
**A**RQUITECTURA DE **C**OMPUTADORES  
Y **A**UTOMÁTICA

Curso 2009-2010

# Contenidos

---

- ❑ Introducción: ILP
- ❑ Técnicas SW: Compilador
- ❑ Planificación dinámica. Algoritmo de Tomasulo.
- ❑ Tratamiento de dependencias de control: Predicción de saltos
  - ❑ Tratamiento de saltos
  - ❑ Técnicas de predicción: Estáticas y dinámicas
- ❑ Especulación
- ❑ Bibliografía
  - o Cap 2 [HePa07]
  - o Capítulos 4 , 7 y 8 de [SiFK97]
  - o Simulador DLXview

# Introducción

❑ **OBJETIVO:** Ejecutar el mayor numero de instrucciones por ciclo

❑ **Obtener el máximo numero de instrucciones independientes**

$$CPI = CPI \text{ ideal} + \text{Penaliz. Media por Instr. (paradas "pipe")}$$

¿ Que técnicas conocemos?

Conflictos de recursos - Riesgos estructurales → Replicación/segmentación

Dependencias de datos → Cortocircuitos

Dependencias de control ( Un salto cada 4-7 instrucciones) → Saltos retardados

## Mecanismos para explotar ILP

Basados en HW en tiempo de ejecución ( dinámicos). Ej Pentium, AMD, IBM

Toda la información disponible en ejecución

Código independiente de la implementación

Basados en SW en tiempo de compilación (estáticos). Ej Itanium

Dependencias de memoria muy difíciles de determinar

# Introducción

---

## ❑ Paralelismo a nivel de instrucción ILP

- ❑ Es la técnica consistente en explotar paralelismo entre instrucciones próximas en la secuencia
- ❑ El bloque básico es muy pequeño
  - ❑ Un bloque básico (BB) es una secuencia de código sin saltos. Un solo punto de entrada y salida
  - ❑ Solo de 4 a 7 instrucciones
  - ❑ Fuertes dependencias entre ellas
- ❑ El camino es explotar ILP entre varios BB
- ❑ El caso más simple: paralelismo a nivel de bucle

```
for ( i =1; i<=1000; i++)  
    x (i) = x (i) + s ;
```

  - ❑ Todas las iteraciones son independientes (saltos)

# Introducción

## □ Técnicas para explotar ILP

	Técnica	Reduce
Dinámicas	Planificación Dinámica	Paradas por riesgos de datos
	Predicción dinámica de saltos	Paradas por riesgos de control
	Lanzamiento múltiple	CPI Ideal
	Varias instrucciones por ciclo	
	Especulación	Riesgos de datos y control
	Dynamic memory disambiguation	Paradas por riesgos de datos en memoria
Estáticas	Desenrollado de bucles	Paradas por riesgos de control
	Planificación por el compilador	Paradas por riesgos de datos
	Software pipelining	CPI Ideal y Paradas por riesgos de datos
	Predicción estática y Especulación por el Compilador	CPI Ideal, paradas por riesgos de datos y control

# Dependencias

---

- ❑ Determinar las dependencias es crítico para obtener el máximo paralelismo

¿ Cuáles hay ? , ¿ A qué recursos afectan?

Las dependencias son propias de los programas

- o La presencia de una dependencia indica la posibilidad de aparición de un riesgo, pero la aparición de éste y la posible parada depende de las características del "pipe"
- o Las dependencias
  - Indican la posibilidad de un riesgo
  - Determinan el orden de cálculo de los resultados
  - Imponen un límite al paralelismo que es posible obtener

# Dependencias

## □ Tipos de Dependencias

- Dependencias de datos
  - Dependencia verdadera (LDE)
  - Dependencias de nombre
    - Antidependencia (EDL)
    - Dependencia de salida (EDE)
- Dependencias de control

## □ Dependencia verdadera (LDE)

- o La instrucción i depende de j
  - i produce un resultado que usa j
  - j depende de k y k depende de i

LD      F0,0(R1)  
ADDD   F4,F0,F2

# Dependencias

## □ Dependencias de nombre ( Reutilización de los registros )

- o Dos instrucciones i y j donde i precede a j presentan dependencias de nombre en las siguientes situaciones:

- o Antidependencia WAR (EDL)

- La instrucción j escribe ( Reg o memoria) antes de que i lea.

```
ADDD    F4,F0,F2
LD       F0,-8(R1)
```

- o Dependencia de salida WAW (EDE)

- Las instrucciones i y j escriben el mismo reg. o memoria

```
ADDD    F4,F0,F2
SUBD    F4,F3,F2
```

## □ ILP y Dependencias de datos

- o Los mecanismos de ejecución deben preservar el orden del programa. Mismo resultado que en ejecución secuencial
- o Explotar todo el paralelismo posible sin afectar al resultado de la ejecución
- o Para las dependencias de nombre eliminar la dependencia usando otros "nombres"



# Dependencias

## □ Dependencias de control

o Cada instrucción depende de un conjunto de saltos y en general esta dependencia debe preservarse para preservar el orden del programa

```
if P1 (  
    S1;  
);  
if P2 (  
    S2;  
)
```

S1 depende de P1 ; S2 depende de P2

Las dependencias de control pueden violarse. Se pueden ejecutar instrucciones no debidas **si esto no afecta** al resultado correcto del programa

**LO IMPORTANTE: el comportamiento de las excepciones y el flujo de datos deben preservarse**

# Dependencias

## ❑ Dependencias de control y Excepciones

- o Comportamiento de excepciones se debe preservar. Cualquier cambio en el orden de ejecución no debe cambiar como las excepciones son atendidas en la ejecución.

```
DADDU R2,R3,R4
BEQZ   R2,L1
LW     R1,0(R2)
L1:    ---    ---
```

- o LW no se puede mover antes de BEQZ ( posible fallo de pagina )

## ❑ Dependencias de control y flujo de datos

- o Se debe mantener el flujo de datos entre instrucciones productoras y consumidoras de datos.

```
DADDU R1,R2,R3
BEQZ   R4,L1
DSUBU R1,R5,R6
L1:    ---    ---
OR     R7,R1,R8
```

- o OR usa el valor de DADDU o DSUBU dependiendo del comportamiento del salto. El flujo de datos se debe preservar.

# Dependencias

- ❑ El compilador se concentra en dependencias del programa, no en riesgos de un “pipe determinado”

- ❑ **Dependencias de datos**

Fáciles de determinar para registros

Difíciles para direcciones de memoria

¿Son el mismo dato 100( R4 ) y 20( R6 )?

En dos iteraciones diferentes 20(R6) y 20(R6) ¿son el mismo dato?

Debe conocer dependencias entre load y stores para permitir su reordenación

Más registros para evitar dependencias de nombre

- ❑ **Dependencias de control**

En general:

- Una instrucción dependiente de un salto no puede moverse antes del salto
- Una instrucción no dependiente de un salto no puede moverse después del salto

Efecto de las dependencias de control sobre el orden de las excepciones y el flujo de datos

**SOLUCIÓN : HW + SW (PROCESADOR + COMPILADOR)**

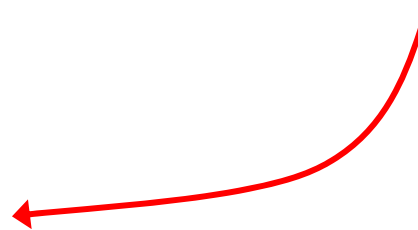
# Técnicas SW para explotar ILP

## □ Un programa: Bucle simple

```
for ( i =1; i <= 1000; i++)  
    x (i) = x (i) + s ;
```

### ✓ Código maquina DLX

```
Loop  LD      F0,O(R1)  
      ADDD   F4,F0,F2  
      SD     O(R1),F4  
      SUBI   R1,R1,#8  
      BNEZ   R1,Loop
```



*Datos de la etapa  
de ejecución*

Instrucción que produce resultado	Instrucción que usa el resultado	Latencia de uso
FP ALU	FP ALU	3
FP ALU	STORE FP	2
LOAD FP	FP ALU	1
LOAD FP	STORE FP	0
Entera	Entera	0

# Técnicas SW para explotar ILP

## □ Un programa: Bucle simple

```
for ( i =1; i <= 1000; i++)  
    x (i) = x (i) + s ;
```

### ✓ Ejecución en el procesador

Loop	LD	F0,0(R1)	Ciclo 1
	Espera		2
	ADDD	F4,F0,F2	3
	Espera		4
	Espera		5
	SD	0(R1),F4	6
	SUBI	R1,R1,#8	7
	Espera		8
	BNEZ	R1,Loop	9
	Espera		10

Una instrucción cada 2 ciclos

# Técnicas SW para explotar ILP

## □ Planificación de instrucciones

Loop	LD	F0,0(R1)	Ciclo1	Reordenamiento para ocultar latencias  6 ciclos    1 instrucción por ciclo  2 ciclos de overhead por el salto
	SUBI	R1,R1,#8	2	
	ADDD	F4,F0,F2	3	
	Espera		4	
	BNEZ	R1,Loop	5	
	SD	#8(R1),F4	6	

## □ Desenrollado 4 veces para más paralelismo (elimina saltos)

Loop	LD	F0,0(R1)	Expone más paralelismo y elimina saltos
	ADDD	F4,F0,F2	
	SD	0(R1),F4	
	LD	F6,-8(R1)	Se elimina 3 saltos y 3 decrementos
	ADDD	F8,F6,F2	
	SD	-8(R1),F8	
	LD	F10,-16(R1)	Permanecen dependencias y paradas
	ADDD	F12,F10,F2	
	SD	-16(R1),F12	
	LD	F14,-24(R1)	<u>MÁS REGITROS = Renombrado por el Compilador</u> (Imprescindible ??)
	ADDD	F16,F14,F2	
	SD	-24(R1),F16	
	SUBI	R1,R1,#32	
	BNEZ	R1,Loop	

# Técnicas SW para explotar ILP

## □ Desenrollado + Planificación

Loop	LD	F0,0(R1)
	LD	F6,-8(R1)
	LD	F10,-16(R1)
	LD	F14,-24(R1)
	ADDD	F4,F0,F2
	ADDD	F8,F6,F2
	ADDD	F12,F10,F2
	ADDD	F16,F14,F2
	SD	0(R1),F4
	SD	-8(R1),F8
	SD	-16(R1),F12
	SUBI	R1,R1,#32
	BNEZ	R1,Loop
	SD	8(R1),F16; 8-32 = -24

- ✓ 3.5 ciclos por iteración
- ✓ Más registros (Imprescindible !!)

Mover SD después de SUBI, valor de R1

El compilador planifica para minimizar los riesgos y eliminar las paradas del "pipe"

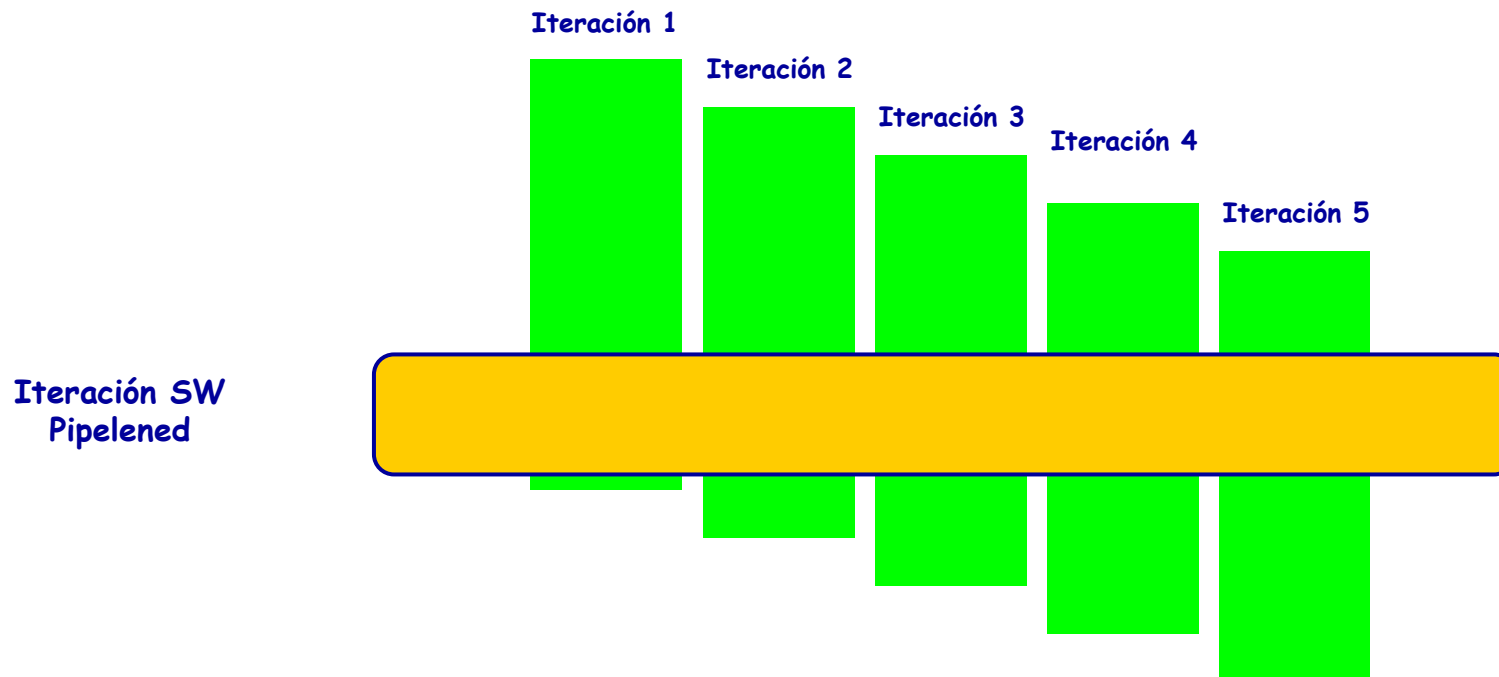
# Técnicas SW para explotar ILP

## ❑ Software “pipelining”

- Idea:

Si las diferentes iteraciones de un bucle son independientes, tomemos instrucciones de diferentes iteraciones para aumentar el ILP

Reorganiza los bucles de manera que cada instrucción pertenece a una iteración diferente





# Técnicas SW para explotar ILP

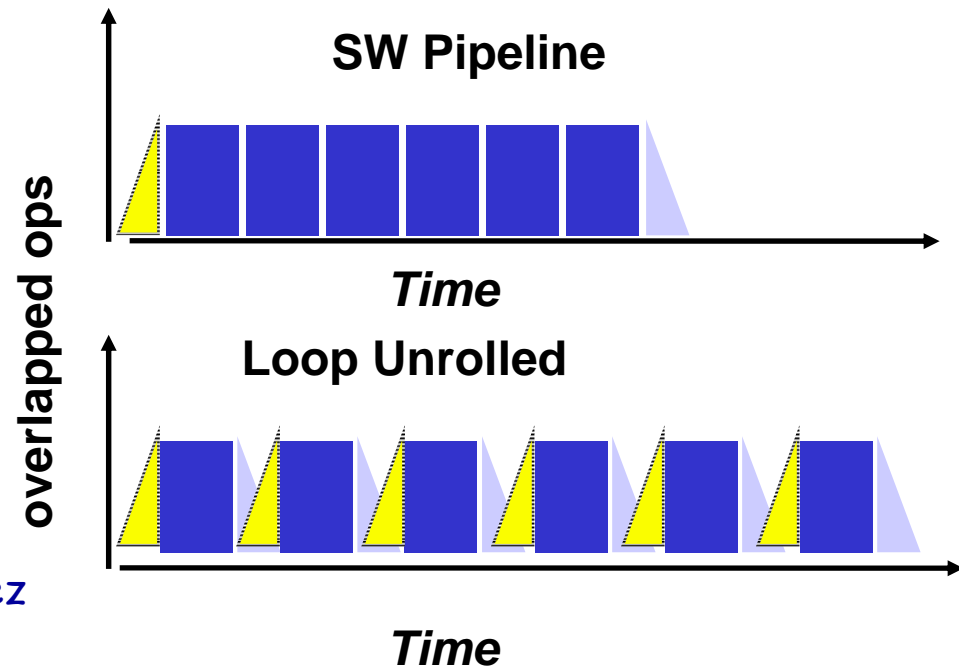
## ❑ Software “pipelining”

Antes: Unrolled 3 veces

1	LD	F0,0(R1)
2	ADDD	F4,F0,F2
3	SD	O(R1),F4
4	LD	F6,-8(R1)
5	ADDD	F8,F6,F2
6	SD	-8(R1),F8
7	LD	F10,-16(R1)
8	ADDD	F12,F10,F2
9	SD	-16(R1),F12
10	SUBI	R1,R1,#24
11	BNEZ	R1,LOOP

Después: Software Pipelined

1	SD	O(R1),F4 ; Stores M[i]
2	ADDD	F4,F0,F2 ; Adds to M[i-1]
3	LD	F0,-16(R1); Loads M[i-2]
4	SUBI	R1,R1,#8
5	BNEZ	R1,LOOP



- ✓ Loop unrolling simbólico
  - Maximiza la distancia resultado-uso
  - Menor tamaño del código
  - Llenado y vaciado del pipe solo una vez

# Técnicas SW para explotar ILP

## Ejecución SW pipelined (suposición R1=1000)

$F0 \leftarrow M(1000)$ $F4 \leftarrow F0 + F2 ; M(1000)+F2$ $F0 \leftarrow M(992)$	Cabecera	LD ADDD LD	$F0, 0(R1)$ $F4, F0, F2$ $F0, -8(R1)$
$M(1000) \leftarrow F4 ; M(1000)+F2$ $F4 \leftarrow F0 + F2 ; M(992)+F2$ $F0 \leftarrow M(984)$ $R1 \leftarrow 992$	Iteración 1	SD ADDD LD	$O(R1), F4 ; \text{Stores } M[i]$ $F4, F0, F2 ; \text{Adds to } M[i-1]$ $F0, -16(R1); \text{Loads } M[i-2]$
$M(992) \leftarrow F4 ; M(992)+F2$ $F4 \leftarrow F0 + F2 ; M(984)+F2$ $F0 \leftarrow M(976)$ $R1 \leftarrow 984$	Iteración 2	SD ADDD LD	$O(R1), F4$ $F4, F0, F2$ $F0, -16(R1)$
... $F0 \leftarrow M(0) ; \text{Ahora } R1=16$ $R1 \leftarrow 8$	Iteración n		• • •
$M(8) \leftarrow F4 ; M(8)+F2$ $F4 \leftarrow F0 + F2 ; M(0)+F2$ $M(0) \leftarrow F4 ; M(0)+F2$	Cola	SD ADDD SD	$O(R1), F4$ $F4, F0, F2$ $-8(R1), F4$

# Técnicas SW para explotar ILP

---

## □ Comparación

### ▪ Loop Unrolling

- Bloque grande para planificar
- Reduce el numero de saltos
- Incrementa el tamaño del código
- Tiene que incluir iteraciones extra
- Presión sobre el uso de registros

### ▪ Software Pipelining

- No hay dependencias en el cuerpo del bucle
- No reduce el numero de saltos
- Necesita inicio y finalización especial

## Tratamiento de dependencias de datos en ejecución

### ❑ Planificación dinámica : Procesador.

Modifica la secuencia de instrucciones resolviendo las dependencias en tiempo de ejecución. Disponibilidad de más unidades funcionales. Código valido para diferentes implementaciones

### ❑ Problema : Lanzamiento de instrucciones en orden.

DIVD	F0,F2,F4	<b>S1</b>	S2 depende de S1
ADDD	F10,F0,F8	<b>S2</b>	
SUBD	F12,F8,F14	<b>S3</b>	S3 es independiente de la demás

La etapa ID bloquea la ejecución en S2 hasta que se resuelve la dependencia ( F0 disponible) y SUBD no puede ejecutarse.

### ❑ Solución : Dividir la etapa ID en dos etapas diferenciadas.

Issue: Decodifica y chequea riesgos estructurales.

Lectura de operandos : Chequea disponibilidad de operandos. Debe implementarse para permitir el flujo de instrucciones.

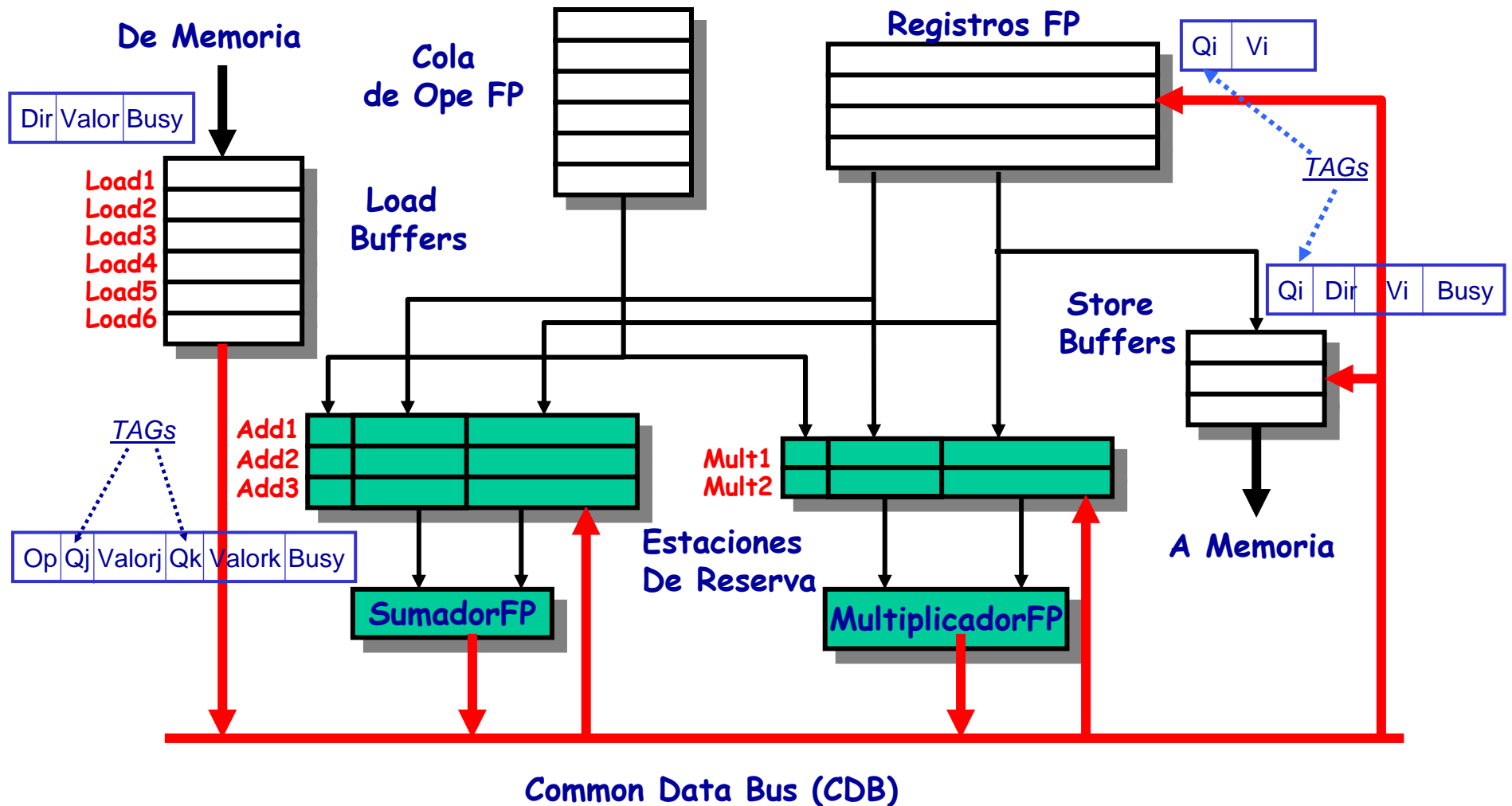
***Ejecución fuera de orden → ¿Finalización fuera de orden?***

# Planificación Dinámica : Tomasulo

---

- ❑ (IBM 360/91, año 1967)
- ❑ Elimina dinámicamente los riesgos EDE y EDL mediante el renombrado de registros
- ❑ Motivación:
  - Arquitectura 360, 4 registros FP( Pocos registros, muchas dependencias )
  - Arquitectura RM
  - UF segmentadas Add(3), Mul(2), Load (6), Stores (3)
- ❑ Solo consideraremos las operaciones de PF en la explicación
- ❑ Casi todos los procesadores actuales usan mecanismo basados en este algoritmo
  - Alpha 21264, HP 8000, MIPS 10000, Pentium III-4-Core, PowerPC 604, ...

# Planificación Dinámica : Tomasulo



Excepciones imprecisas

Issue en orden-- Ejecución fuera de orden—Finalización fuera de orden

# Planificación Dinámica : Tomasulo

## □ Tres estados para una instrucción en el algoritmo

### 1 Issue

Toma la instrucción de la cola de instrucciones. Envía la instrucción a la ER correspondiente si hay entradas disponibles. Envía los operandos si están disponibles o UF que los generara. En load/store si hay buffer libre.

(Copiar "Tag+Valor" de registros fuente sobre campos "Tag+Valor" de ER)

Marca registro destino (tag) con ID de la ER que ejecutará la instrucción

### 2 Ejecución

Monitoriza CDB para disponibilidad de operandos. Cuando lista manda ejecutar.

### 3 Escritura de resultados

Vía CDB en registros y estaciones de reserva (ER) y marca ER como libre.

Bus normal: dato + destino, " va a "

CDB: dato + fuente, " viene de "

No chequea riesgo EDE ni EDL (renombrado dinámico)

## □ Componentes de las ER

Op: Operación a realizar

Vj, Vk: Valores de los operandos fuente

Store buffers tienen un campo V para el resultado que debe ser almacenado.

Qj, Qk: ER produciendo los operandos fuente Notar: Qj, Qk=0 => ready

Store buffers tienen Qi para ER produciendo el resultado

Busy: Indica ER ocupada

Estado de registro resultado—Indica la FU que producira el resultado para el Registro. En blanco ninguna instrucción esta pendiente de escribir en el registro.

# Planificación Dinámica : Tomasulo

## □ Ejemplo: Ejecución de la secuencia:

(S1): COD1 F2, ---, ---

(S2): COD2 F4, ---, ---

(S3): **ADDD F0, F2, F4**

Suposiciones: S1 y S2 ya lanzadas a ejecución, S3 se va a lanzar

Estado inicial de registros:

	TAG	VALOR
F0	??	??
F2	Y	??
F4	Z	??



# Planificación Dinámica : Tomasulo

## □ Ejemplo: Ejecución de la secuencia:

(S1): COD1 F2, ---, ---

(S2): COD2 F4, ---, ---

(S3): **ADDD F0, F2, F4**

Paso 1: Lanzamiento de ADD a la ER X (X es una ER libre de la UF de Suma/Resta de Punto Flotante)

*Estación de reserva*

	Busy	OP	TAGj	Valorj	TAGk	Valork
X	yes	+	Y	??	Z	??

*Estado de registros*

	TAG	VALOR
F0	X	??
F2	Y	??
F4	Z	??

# Planificación Dinámica : Tomasulo

## □ Ejemplo: Ejecución de la secuencia:

(S1): COD1 F2, ---, ---

(S2): COD2 F4, ---, ---

(S3): **ADDD F0, F2, F4**

Paso 2: Esperar a que se generen los operandos

a) Escritura del resultado de S1 sobr el CDB:

### Estación de reserva

	Busy	OP	TAGj	Valorj	TAGk	Valork
X	yes	+	0	22.57	Z	??

### Estado de registros

	TAG	VALOR
F0	X	??
F2	0	22.57
F4	Z	??

b) Escritura del resultado de S2 sobre el CDB:

### Estación de reserva

	Busy	OP	TAGj	Valorj	TAGk	Valork
X	yes	+	0	22.57	0	3.2

### Estado de registros

	TAG	VALOR
F0	X	??
F2	0	22.57
F4	0	3.2

# Planificación Dinámica : Tomasulo

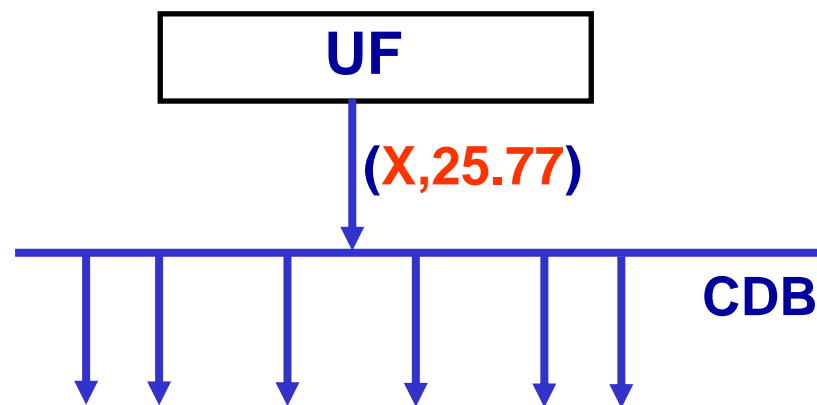
## □ Ejemplo: Ejecución de la secuencia:

(S1): COD1 F2, ---, ---

(S2): COD2 F4, ---, ---

(S3): ADDDF0, F2, F4

Paso 3: Ejecutar operación y escribir resultado sobre el CDB



A todas las ER, Store Buffers y Registros que tengan la marca X

Estado de registros		
	TAG	VALOR
F0	0	25.77
F2	0	22.57
F4	0	3.2

# Planificación Dinámica:TOMASULO

## □ Ejemplo

LD 2 ciclos, ADDD y SUBD 2 ciclos, MULT 10 ciclos, DIVD 40 ciclos

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2			
LD	F2	45+	R3			
MULT	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Ocupada	Dirección
Load1		
Load2		
Load3		

### Estado de ER

Operación

Qj y Qk ER produciendo operandos

Vj y Vk valores de los operandos

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1						
	Mul2						

### Estado de REG

Qué FU escribirá en el Reg

	F0	F2	F4	F6	F8	F10	F12	
FU								

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 1

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULT	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Ocupada	Dirección
Load1	SI	34+R2
Load2	NO	
Load3	NO	

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1	NO					
	Add2	NO					
	Add3	NO					
	Mul1	NO					
	Mul2	NO					

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU				Load1				

# Planificación *Dinámica*:TOMASULO

## ❑ Ciclo 2

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1		
LD	F2	45+	R3	2		
MULT	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Ocupada	Dirección
Load1	SI	34+R2
Load2	SI	45+R3
Load3	NO	

No hay bloqueo

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1	NO					
	Add2	NO					
	Add3	NO					
	Mul1	NO					
	Mul2	NO					

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU		Load2		Load1				

# Planificación Dinámica:TOMASULO

## □ Ciclo 3

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1	2-3	
LD	F2	45+	R3	2		
MULT	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Ocupada	Dirección
Load1	SI	34+R2
Load2	SI	45+R3
Load3	NO	

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1	NO					
	Add2	NO					
	Add3	NO					
	Mul1	SI	Mult		R(F4)	Load2	
	Mul2	NO					

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU	Mult1	Load2		Load1				

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 4

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1	2-3	4
LD	F2	45+	R3	2	3-4	
MULT	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Ocupada	Dirección
Load1	NO	
Load2	SI	45+R3
Load3	NO	

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1	SI	Subd	M(34+R2)			Load2
	Add2	NO					
	Add3	NO					
	Mul1	SI	Mult		R(F4)	Load2	
	Mul2	NO					

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU	Mult1	Load2		M(34+R2)	Add1			



# Planificación Dinámica:TOMASULO

## ❑ Ciclo 6

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1	2-3	4
LD	F2	45+	R3	2	3-4	5
MULT	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Ocupada	Dirección
Load1	NO	
Load2	NO	
Load3	NO	

Ningún bloqueo

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
2	Add1	SI	Subd	M(34+R2)	M(45+R3)		
	Add2	SI	Addd		M(45+R3)	Add1	
	Add3	NO					
10	Mul1	SI	Mult	M(45+R3)	R(F4)		
	Mul2	SI	Divd		M(34+R2)	Mult1	

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU	Mul1	M(45+R3)		Add2	Add1	Mul2		

F6 reasignado

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 8

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1	2-3	4
LD	F2	45+	R3	2	3-4	5
MULT	F0	F2	F4	3		
SUBD	F8	F6	F2	4	6-7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Ocupada	Dirección
Load1	NO	
Load2	NO	
Load3	NO	

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1	NO					
2	Add2	SI	Add	M()-M()	M(45+R3)		
	Add3	NO					
7	Mul1	SI	Mult	M(45+R3)	R(F4)		
	Mul2	SI	Divd		M(34+R2)	Mult1	

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU	Mul1	M(45+R3)		Add2	M()-M()	Mul2		

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 13

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1	2-3	4
LD	F2	45+	R3	2	3-4	5
MULT	F0	F2	F4	3		
SUBD	F8	F6	F2	4	6-7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	9-10	11

	Ocupada	Dirección
Load1	NO	
Load2	NO	
Load3	NO	

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1	NO					
	Add2	NO					
	Add3	NO					
3	Mul1	SI	Mult	M(45+R3)	R(F4)		
	Mul2	SI	Divd		M(34+R2)	Mult1	

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU	Mul1	M(45+R3)		F8+M()	M()-M()	Mul2		

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 16

Instuc		J	K	Issue	Ejecución	escritura
LD	F6	34+	R2	1	2-3	4
LD	F2	45+	R3	2	3-4	5
MULT	F0	F2	F4	3	6-15	16
SUBD	F8	F6	F2	4	6-7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	9-10	11

	Ocupada	Dirección
Load1	NO	
Load2	NO	
Load3	NO	

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P.J Qj	ER.P:K Qk
	Add1	NO					
	Add2	NO					
	Add3	NO					
	Mul1	NO					
40	Mul2	SI	Divd	M*F4	M(34+R2)		

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU	M*F4	M(45+R3)		F8+M()	M()-M()	Mul2		

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 57

Finalización en desorden ☒ Excepciones

Instuc		J	K	Issue	Ejecución.	escritura
LD	F6	34+	R2	1	2-3	4
LD	F2	45+	R3	2	3-4	5
MULT	F0	F2	F4	3	6-15	16
SUBD	F8	F6	F2	4	6-7	8
DIVD	F10	F0	F6	5	17-56	57
ADDD	F6	F8	F2	6	9-10	11

	Ocupada	Dirección
Load1	NO	
Load2	NO	
Load3	NO	

Estado de ER

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1	NO					
	Add2	NO					
	Add3	NO					
	Mul1	NO					
	Mul2	SI	Divd	M*F4	M(34+R2)		

Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU	M*F4	M(45+R3)		F8+M()	M()-M()	Mul2		

## □ Renombrado dinámico en un unrolling

```
Loop LD      F0,0(R1)
      MULTD  F4,F0,F2
      SD      0(R1),F4
      SUBI    R1,R1,#8
      BNEZ    R1,Loop
```

Operación: vector F0 \* escalar F2

### Suposiciones:

MULT 4 ciclos

En 1ª iter. Load 8 ciclos ( fallo ), en 2ª 1 ciclo (acierto)

Mostraremos dos iteraciones

# Planificación Dinámica:TOMASULO

## □ Bucle

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1			
MULT	F4	F0	F2			
SD	F4	0	R1			
LD	F0	0	R1			
MULT	F4	F0	F2			
SD	F4	0	R1			

	Ocupada	Dirección	
Load1	NO		
load2	NO		
load3	NO		Qi
store1	NO		
store2	NO		
store3	NO		

## Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1						
	Mul2						

## Estado de REG

	F0	F2	F4	F6	F8	F10	F12	
FU								

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 1

Instuc		J	K	Issue	ejecución	escritura
LD	F0	0	R1	1		
MULT	F4	F0	F2			
SD	F4	0	R1			
LD	F0	0	R1			
MULT	F4	F0	F2			
SD	F4	0	R1			

	Ocupada	Dirección	
Load1	SI	80	
load2	NO		
load3	NO		<u>Qi</u>
store1	NO		
store2	NO		
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1						
	Mul2						

Estado de REG  
 R1=80

	F0	F2	F4	F6	F8	F10	F12	
FU	Load1							

*Ojo latencia del primer load*



# Planificación Dinámica:TOMASULO

## ❑ Ciclo 2

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1		
MULT	F4	F0	F2	2		
SD	F4	0	R1			
LD	F0	0	R1			
MULT	F4	F0	F2			
SD	F4	0	R1			

	Ocupada	Dirección	
Load1	SI	80	
load2	NO		
load3	NO		Qi
store1	NO		
store2	NO		
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P.J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1	SI	MULT		R(F2)	Load1	
	Mul2						

Estado de REG  
 R1=80

	F0	F2	F4	F6	F8	F10	F12	
FU	Load1		Mul1					

Ojo latencia del 1 load

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 3

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1		
MULT	F4	F0	F2	2		
SD	F4	0	R1	3		
LD	F0	0	R1			
MULT	F4	F0	F2			
SD	F4	0	R1			

	Ocupada	Dirección	
Load1	SI	80	
load2	NO		
load3	NO		Qi
store1	SI	80	Mult1
store2	NO		
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P.J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1	SI	MULT		R(F2)	Load1	
	Mul2						

Estado de REG  
 R1=80

	F0	F2	F4	F6	F8	F10	F12	
FU	Load1		Mul1					

Ojo latencia del 1er load

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 6

Cierre del bucle

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1		
MULT	F4	F0	F2	2		
SD	F4	0	R1	3		
LD	F0	0	R1	6		
MULT	F4	F0	F2			
SD	F4	0	R1			

	Ocupada	Dirección	
Load1	SI	80	
load2	SI	72	
load3	NO		Qi
store1	SI	80	Mult1
store2	NO		
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1	SI	MULT		R(F2)	Load1	
	Mul2						

Estado de REG  
 R1=72

Ojo latencia del 1 load

	F0	F2	F4	F6	F8	F10	F12	
FU	Load2		Mul1					

Renombrado

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 7

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1		
MULT	F4	F0	F2	2		
SD	F4	0	R1	3		
LD	F0	0	R1	6		
MULT	F4	F0	F2	7		
SD	F4	0	R1			

	Ocupada	Dirección	
Load1	SI	80	
load2	SI	72	
load3	NO		Qi
store1	SI	80	Mult1
store2	NO		
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1	SI	MULT		R(F2)	Load1	
	Mul2	SI	MULT		R(F2)	Load2	

Estado de REG  
 R1=72

Ojo latencia del 1 load

	F0	F2	F4	F6	F8	F10	F12	
FU	Load2		Mul2					

Renombrado

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 10

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1	2- 9	10
MULT	F4	F0	F2	2		
SD	F4	0	R1	3		
LD	F0	0	R1	6	7-10	
MULT	F4	F0	F2	7		
SD	F4	0	R1	8		

	Ocupada	Dirección	
Load1	NO		
load2	SI	72	
load3	NO		Qi
store1	SI	80	Mult1
store2	SI	72	Mult2
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
4	Mul1	SI	MULT	M(80)	R(F2)		
	Mul2	SI	MULT		R(F2)	Load2	

Estado de REG  
 R1=64

	F0	F2	F4	F6	F8	F10	F12	
FU	Load2		Mul2					

Ojo latencia del 1 load

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 11

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1	2-9	10
MULT	F4	F0	F2	2		
SD	F4	0	R1	3		
LD	F0	0	R1	6	7-10	11
MULT	F4	F0	F2	7		
SD	F4	0	R1	8		

	Ocupada	Dirección	
Load1	NO		
load2	NO		
load3	SI	64	Qi
store1	SI	80	Mult1
store2	SI	72	Mult2
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
3	Mul1	SI	MULT	M(80)	R(F2)		
4	Mul2	SI	MULT	M(72)	R(F2)		

### Estado de REG R1=64

	F0	F2	F4	F6	F8	F10	F12	
FU			Mul2					

# Planificación Dinámica:TOMASULO

## ❑ Ciclo 15

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1	2-9	10
MULT	F4	F0	F2	2	11-14	15
SD	F4	0	R1	3		
LD	F0	0	R1	6	7-10	11
MULT	F4	F0	F2	7	12-15	
SD	F4	0	R1	8		

	Ocupada	Dirección	
Load1	NO		
load2	NO		
load3	SI	64	Qi
store1	SI	80	M()*F2
store2	SI	72	Mult2
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1						
	Mul2	SI	MULT	M(72)	R(F2)		

### Estado de REG R1=64

	F0	F2	F4	F6	F8	F10	F12	
FU			Mul2					

# Planificación Dinámica:TOMASULO

## □ Ciclo 16

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1	2- 9	10
MULT	F4	F0	F2	2	11-14	15
SD	F4	0	R1	3	16	--
LD	F0	0	R1	6	7-10	11
MULT	F4	F0	F2	7	12-15	16
SD	F4	0	R1	8		

	Ocupada	Dirección	
Load1	NO		
load2	NO		
load3	SI	64	Qi
store1	NO		
store2	SI	72	M()*F2
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1						
	Mul2						

### Estado de REG R1=64

	F0	F2	F4	F6	F8	F10	F12	
FU								



# Planificación Dinámica:TOMASULO

## ❑ Ciclo 17

Instuc		J	K	Issue	Ejecución	escritura
LD	F0	0	R1	1	2-9	10
MULT	F4	F0	F2	2	11-14	15
SD	F4	0	R1	3	16	--
LD	F0	0	R1	6	7-10	11
MULT	F4	F0	F2	7	12-15	16
SD	F4	0	R1	8	17	--

	Ocupada	Dirección	
Load1	NO		
load2	NO		
load3	SI	64	Qi
store1	NO		
store2	NO		
store3	NO		

### Estado de ER

Loop LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,Loop

Tiempo	FU	Ocupada	Operación	S1 Vj	S2 Vk	ER.P:J Qj	ER.P:K Qk
	Add1						
	Add2						
	Add3						
	Mul1						
	Mul2						

### Estado de REG R1=64

	F0	F2	F4	F6	F8	F10	F12	
FU								

# Planificación Dinámica:TOMASULO (detalle de las fases)

## □ Nomenclatura:

o Instrucción aritmética:  $D \leftarrow OP(S1, S2)$

- Instrucción aritmética que realiza la operación OP sobre el contenido de los registros S1 y S2, y deposita el resultado en el registro D

o Instrucción Load:  $D \leftarrow Mem[Dirección]$

- Carga en el registro D el contenido de la posición de memoria "Dirección"

o Instrucción Store:  $Mem[Dirección] \leftarrow S$

- Almacena el contenido del registro S en la posición de memoria "Dirección"

o Estación de reserva x:  $ER(x)(Busy, OP, Q_j, V_j, Q_k, V_k)$

o Registro x:  $Reg(x)(Q_i, V_i)$

o Load Buffer x:  $LB(x)(Busy, Dir)$

o Store Buffer x:  $SB(x)(Busy, Dir, Q_i, V_i)$

## Planificación Dinámica: TOMASULO (detalle de las fases)

### □ Fase Issue

Tipo de instrucción	Esperar hasta que ...	Hacer ...
Aritmética: $D \leftarrow OP(S1, S2)$	La estación de reserva $ER(x)$ está libre y es capaz de ejecutar $OP$	$ER(x).Busy \leftarrow \text{Yes}$ $ER(x).OP \leftarrow OP$ $ER(x).Q_j \leftarrow Reg(S1).Q_i$ $ER(x).V_j \leftarrow Reg(S1).V_i$ $ER(x).Q_k \leftarrow Reg(S2).Q_i$ $ER(x).V_k \leftarrow Reg(S2).V_i$  $Reg(D).Q_i \leftarrow x$
Load: $D \leftarrow Mem[\text{Dirección}]$	El Load Buffer $LB(x)$ está libre	$LB(x).Busy \leftarrow \text{Yes}$ $LB(x).Dir \leftarrow \text{Dirección}$  $Reg(D).Q_i \leftarrow x$
Store: $Mem[\text{Dirección}] \leftarrow S$	El Store Buffer $SB(x)$ está libre	$SB(x).Busy \leftarrow \text{Yes}$ $SB(x).Dir \leftarrow \text{Dirección}$ $SB(x).Q_i \leftarrow Reg(S).Q_i$ $SB(x).V_i \leftarrow Reg(S).V_i$

## Planificación Dinámica: TOMASULO (detalle de las fases)

### □ Fase Ejecución

Tipo de instrucción	Esperar hasta que ...	Hacer ...
Aritmética: $D \leftarrow OP(S1, S2)$	$(ER(x).Q_j = 0) \text{ Y } (ER(x).Q_k = 0)$	Ejecutar cálculo OP sobre la UF usando operandos $V_j$ y $V_k$ Generar RESULTADO
Load: $D \leftarrow Mem[Dirección]$	(La dirección efectiva está disponible) Y (LB(x).Dir no tiene dependencias respecto de Stores lanzados antes)  (Uso de cola de Load / Store: explicación posterior)	$RESULTADO \leftarrow Mem[LB(x).Dir]$
Store: $Mem[Dirección] \leftarrow S$	(La dirección efectiva está disponible) Y (SB(x).Dir no tiene dependencias con Load ni Stores previos) Y $(SB(x).Q_i = 0)$	$Mem[SB(x).Dir] \leftarrow SB(x).V_i$ $SB(x).Busy \leftarrow No$

## Planificación Dinámica: TOMASULO (detalle de las fases)

### □ Fase Write

Tipo de instrucción	Esperar hasta que ...	Hacer ...
Aritmética: $D \leftarrow OP(S1, S2)$	(Ejecución completa en $ER(x)$ ) Y (CDB disponible)	<u>Escribir sobre CDB: <math>(x, RESULT)</math></u> $\forall z (Si\ Reg(z).Q_i = x) \Rightarrow (Reg(z).Q_i = 0) \ Y \ (Reg(z).V_i = RESULT)$ $\forall z (Si\ ER(z).Q_j = x) \Rightarrow (ER(z).Q_j = 0) \ Y \ (ER(z).V_j = RESULT)$ $\forall z (Si\ ER(z).Q_k = x) \Rightarrow (ER(z).Q_k = 0) \ Y \ (ER(z).V_k = RESULT)$ $\forall z (Si\ SB(z).Q_i = x) \Rightarrow (SB(z).Q_i = 0) \ Y \ (SB(z).V_i = RESULT)$  $ER(x).Busy = No$
Load: $D \leftarrow Mem[Dirección]$	(Acceso a memoria completo en $LB(x)$ ) Y (CDB disponible)	<u>Escribir sobre CDB: <math>(x, RESULT)</math></u> $\forall z (Si\ Reg(z).Q_i = x) \Rightarrow (Reg(z).Q_i = 0) \ Y \ (Reg(z).V_i = RESULT)$ $\forall z (Si\ ER(z).Q_j = x) \Rightarrow (ER(z).Q_j = 0) \ Y \ (ER(z).V_j = RESULT)$ $\forall z (Si\ ER(z).Q_k = x) \Rightarrow (ER(z).Q_k = 0) \ Y \ (ER(z).V_k = RESULT)$ $\forall z (Si\ SB(z).Q_i = x) \Rightarrow (SB(z).Q_i = 0) \ Y \ (SB(z).V_i = RESULT)$  $LB(x).Busy = No$
Store: $Mem[Dirección] \leftarrow S$	Nada	Nada

# Planificación Dinámica:TOMASULO

---

## *Resumen de ventajas e inconvenientes*

- Elimina el cuello de botella de los registros
- Evita EDL y EDE
- Permite el unrolling en HW
- No esta limitado a bloques básicos si existe predicción de saltos
- Complejidad
- Muchos cargas de registros asociativas por ciclo
- CDB limita el rendimiento
- Excepciones imprecisas

# Reduciendo la penalización de los saltos

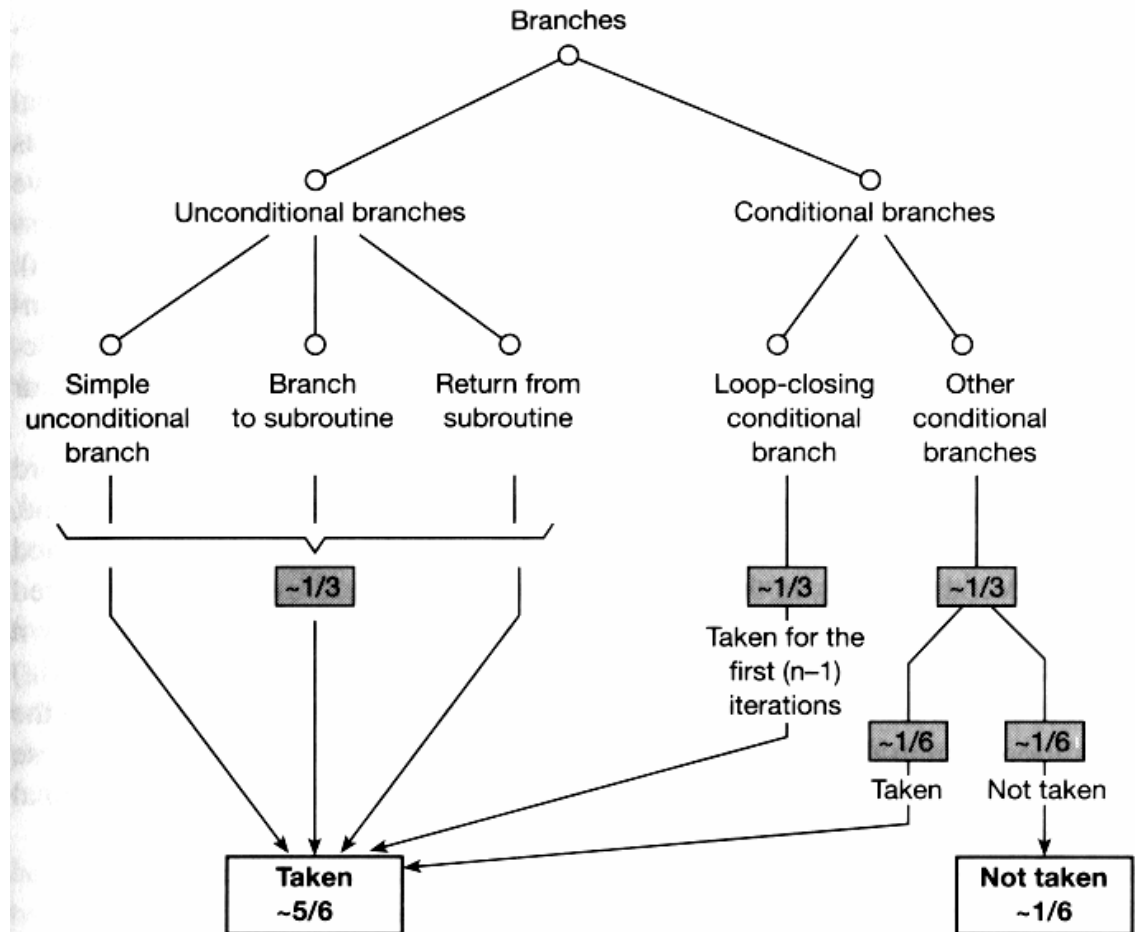
## □ Tipos de saltos: Estadísticas

### En promedio

- Instrucciones de salto  
1 de cada 5 instrucc.
- Saltos condicionales  
2 de cada 3 saltos
- Saltos incondicionales  
1 de cada 3 saltos
- Saltos tomados  
5 de cada 6 saltos
- Saltos condicionales tomados  
3 de cada 4 saltos condic.
- Saltos incondicionales tomados  
Todos

### Conclusión (en promedio)

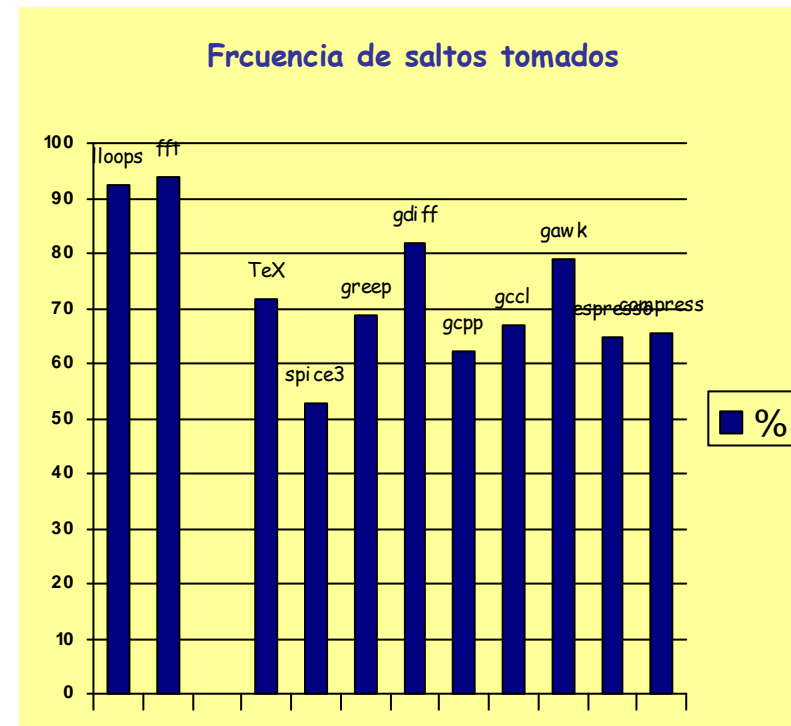
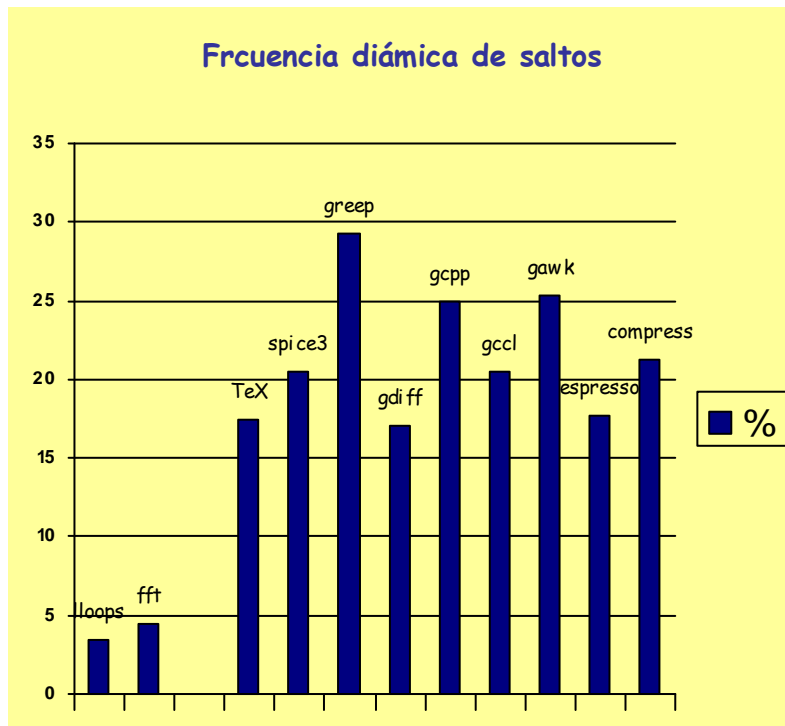
- 1 de cada 6 instrucciones  
es un salto tomado
- 1 de cada 8 instrucciones  
es un salto condicional
- 1 de cada 10 instrucciones  
es un salto condicional y tomado



Programas enteros una de cada 4-5 instrucciones. Flotantes 1 de cada 10-20 instrucciones

# Reduciendo la penalización de los saltos

## □ Tipos de saltos: Estadísticas



### Conclusión

- Frecuencia de los saltos depende del tipo de programa
- El comportamiento depende del tipo de programa



# Reduciendo la penalización de los saltos

## □ Predicción

### Idea Básica

Cuando se detecta una instrucción de salto condicional sin resolver

- Se supone o predice el camino del salto: *tomado o no tomado (Taken - Untaken)*
- Si el salto se predice como tomado se predice la dirección destino del salto
- La ejecución continúa de forma *especulativa* a lo largo del camino supuesto

Cuando se resuelve la condición

- Si la predicción fue correcta
  - ⇒ La ejecución se confirma y continúa normalmente
- Si la predicción fue incorrecta (fallo de predicción o "*misprediction*")
  - ⇒ Se descartan todas las instrucciones ejecutadas especulativamente
  - ⇒ Se reanuda la ejecución a lo largo del camino correcto

### Problemas a resolver en instrucciones de salto

1) Predecir el camino que tomará el salto

- TAKEN (Tomado)
- UNTAKEN (No Tomado)

2) Predecir la dirección de la instrucción destino del salto con un retardo mínimo (para saltos tomados)

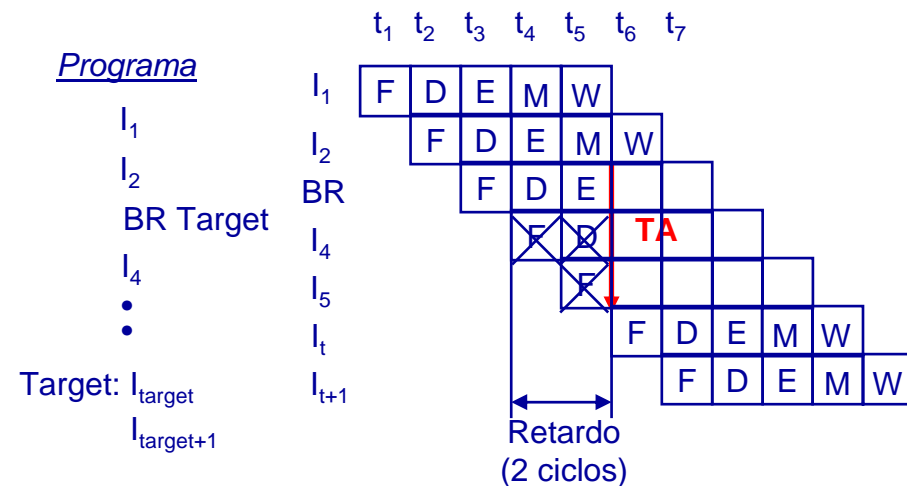
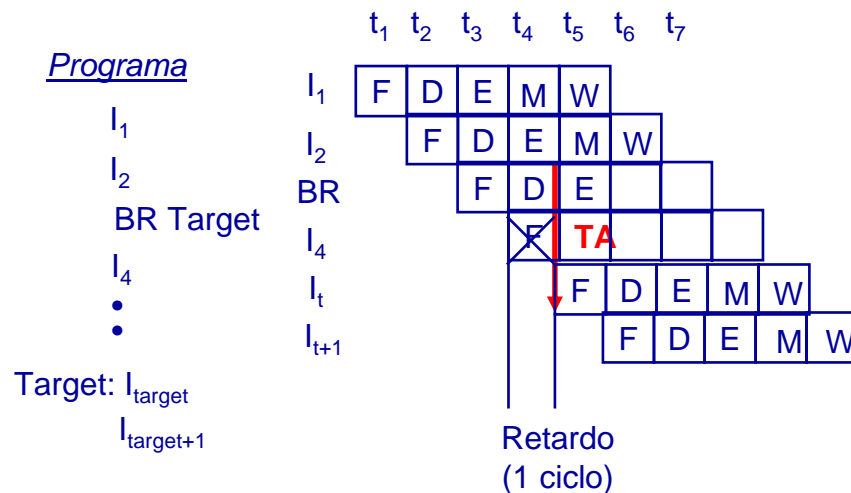
# Tratamiento de Saltos: Predicción

## ❑ Consideraciones generales

### 1) Predecir el camino que tomará el salto

TAKEN (Tomado) PREDICCIÓN CORRECTA

TAKEN (Tomado) PREDICCIÓN INCORRECTA

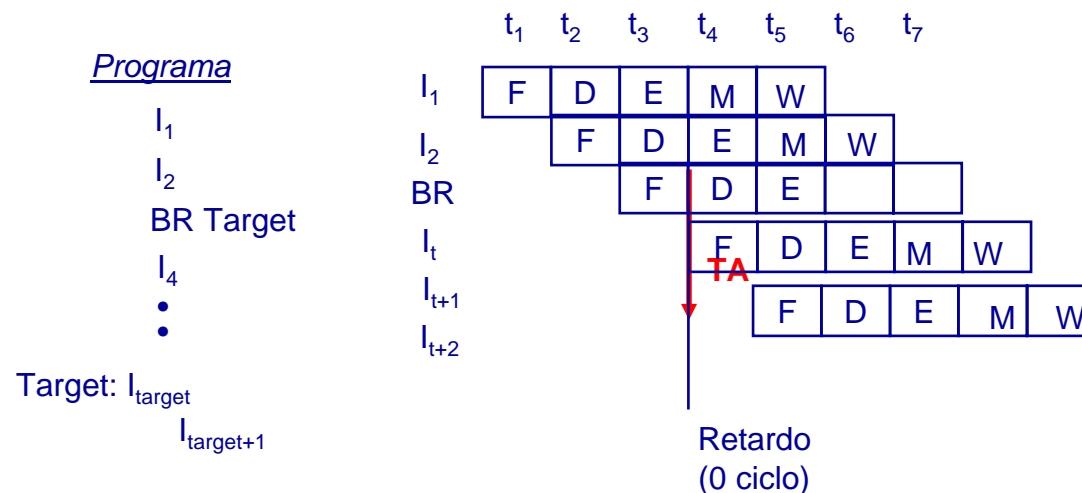


Ojo retardo mínimo

# Tratamiento de Saltos: Predicción

## ❑ Consideraciones generales

2) Predecir la dirección de la instrucción destino del salto con un retardo mínimo (para saltos tomados)



- Detectar lo más pronto posible el salto: En buffer de instrucciones, al buscarla en la cache.... Más importante para pipes de más etapas (frecuencia)

# Tratamiento de Saltos: Predicción

## ❑ Acceso a la instrucción destino del salto I

### Branch Target Address Cache (BTAC)

- Cache que almacena la dirección destino de los últimos saltos tomados
- Cuando se accede a una instrucción de salto
  - Se accede simultáneamente a la BTAC utilizando la dirección de la instrucción de salto
  - Si la instrucción está en la BTAC y el salto se predice como tomado
    - ⇒ la dirección destino del salto se lee de la BTAC

### Actualización de la BTAC

*La BTAC se actualiza cuando se ejecuta la instrucción de salto y se conoce:*

- Si el salto fue tomado o no
- La dirección destino del salto

Si el salto fue tomado

*Si no estaba en la BTAC*



Se introduce en la BTAC

*Si ya estaba en la BTAC*



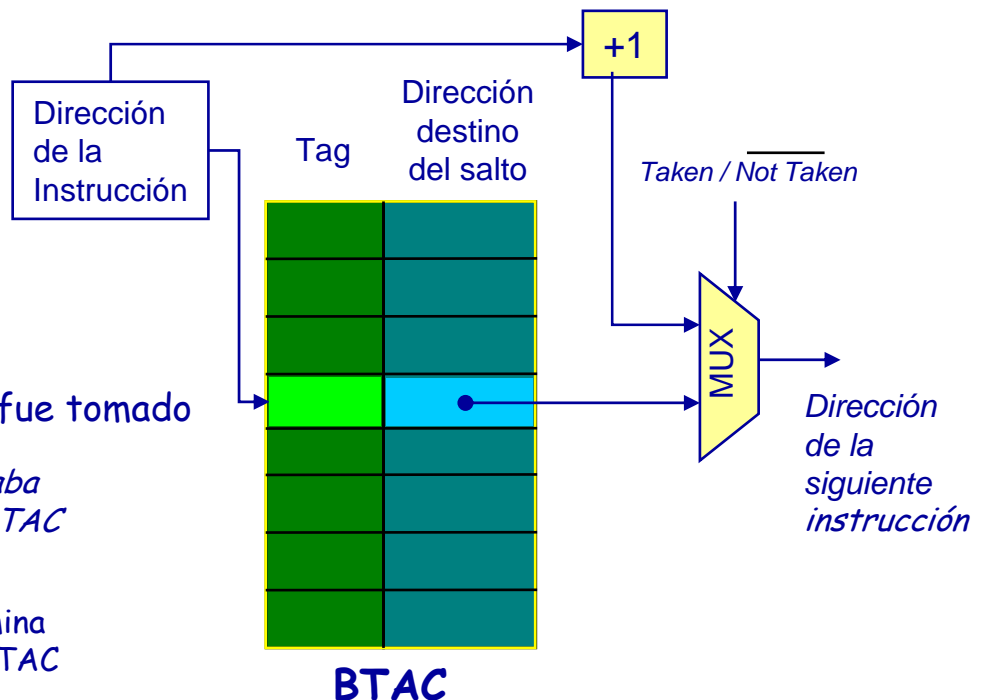
Se actualiza su dirección destino en la BTAC

Si el salto no fue tomado

*Si estaba en la BTAC*



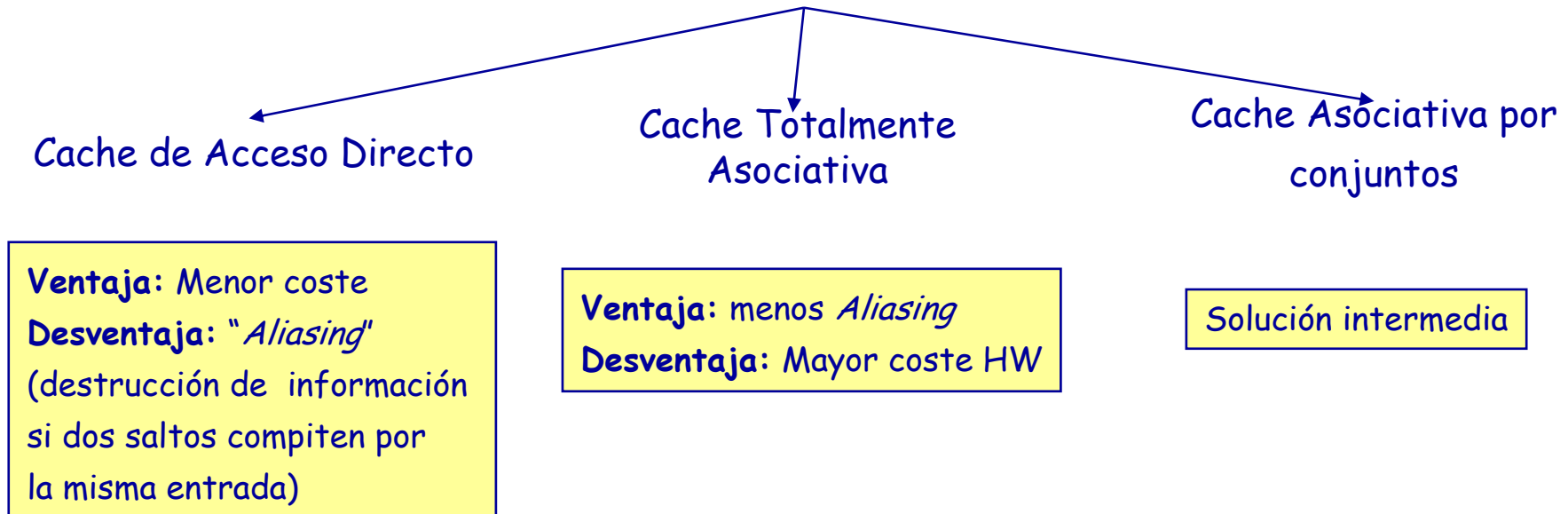
Se elimina de la BTAC



# Tratamiento de Saltos: Predicción

## ❑ Acceso a la instrucción destino del salto II

### Alternativas de diseño de la BTAC



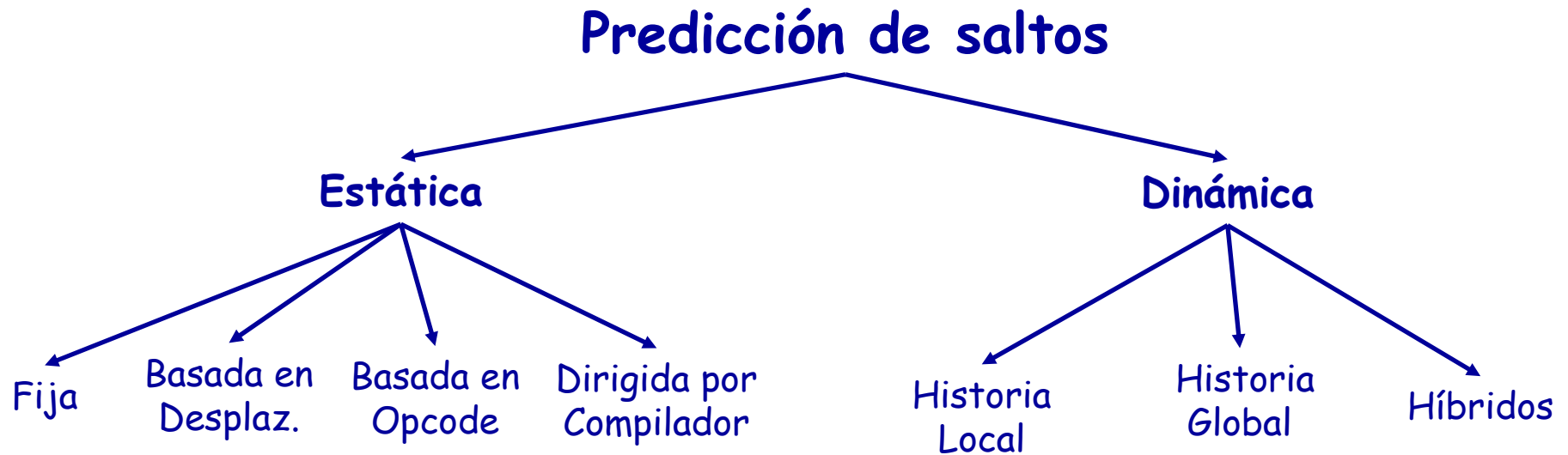
EJEMPLOS: Pentium (256) , Pentium II (512), Pentium 4 (4K) , AMD 64 (2K)

### *Variación Branch Target Instruction Cache*

- Almacenar la instrucción "más ventajas" si tiempo de acceso a las instrucciones es alto
- Ejemplos: AMD K6,K7, NexGen Nx586

# Tratamiento de Saltos: Predicción

## □ Clasificación de técnicas de predicción de saltos



# Tratamiento de Saltos: Predicción

## ❑ Predicción estática

### Predicción Fija

***ALWAYS TAKEN***

- Predecir todos los saltos como *tomados*
- Mayor número de aciertos de predicción (3 de cada 4 saltos cond. son tomados)
- Mayor coste hardware (necesita almacenar la dirección destino del salto)

***ALWAYS NOT TAKEN***

- Predecir todos los saltos como *no tomados*
- Menor número de aciertos de predicción (sólo 1 de cada 4 saltos cond. es no tomado)
- Menor coste hardware

### Predicción basada en la DIRECCIÓN del salto

***Saltos hacia atrás : TOMADOS***

La mayoría de saltos hacia atrás corresponden a bucles

***Saltos hacia delante: NO TOMADOS***

La mayoría de saltos hacia delante corresponden a IF-THEN-ELSE

*Mal comportamiento en programas con pocos bucles y muchos IF-THEN-ELSE*

# Tratamiento de Saltos: Predicción

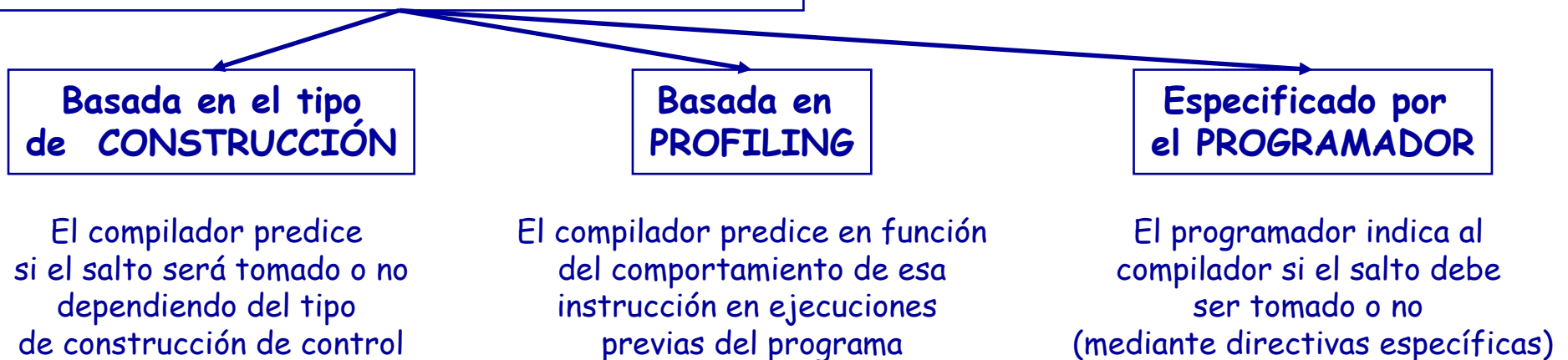
## ❑ Predicción estática

### Predicción basada en el OPCODE de la instrucción de salto

**Fundamento:** La probabilidad de que un salto sea tomado depende del tipo de salto

*El salto es tomado para ciertos códigos de operación y no tomado para otros*

### Predicción dirigida por el COMPILADOR



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



# Tratamiento de Saltos: Predicción

## □ Predictores Dinámicos

### Idea básica

La predicción se realiza observando el comportamiento de las instrucciones de salto en las últimas ejecuciones ( Historia )

*Necesario almacenar la historia de las ultimas ejecución del salto*

*Predictores de  
1 bit de historia*

#### EJEMPLOS

- Gmicro 100 (1991)
- Alpha 21064 (1992)
- R8000 (1994)

*Predictores de  
2 bits de historia (bimodal)*

#### EJEMPLOS

- MC68060 (1993)
- Pentium (1994)
- Alpha 21064A (1994)
- Alpha 21164 (1995)

- PA 8500 (1999)
- UltraSparc (1995)
- PowerPC 604 (1995)
- PowerPC 620 (1996)
- R10000 (1996)

*Predictores de  
3 bits de historia*

#### EJEMPLOS

- PA 8000 (1996)

### Evolución

Predictores correlacionados  
Predictores híbridos

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos

### Predictor de un BIT

- Utilizan un bit de predicción por cada instrucción de salto
  - El bit de predicción refleja el comportamiento de la última ejecución de la instrucción de salto
- ⇒ Indica si en la anterior ejecución el salto fue tomado o no

#### Predicción

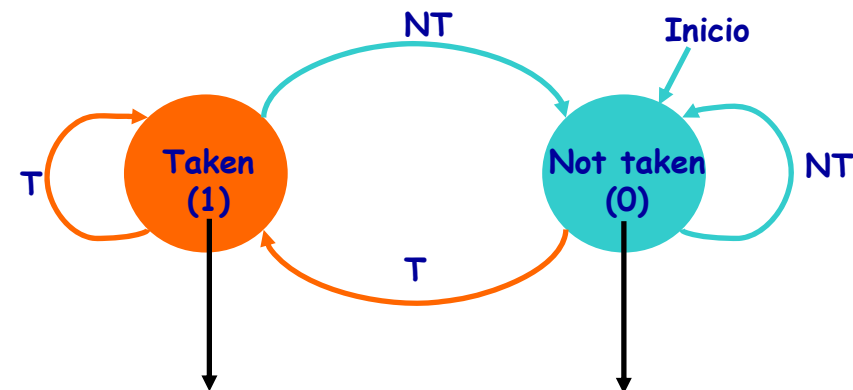
- El salto se predice como *Taken* si en la última ejecución fue tomado
- El salto se predice como *Not Taken* si en la última ejecución no fue tomado

#### FUNCIONAMIENTO

- Máquina de dos estados:
  - Not taken (0)
  - Taken (1)
- Registro de historia
  - Contador saturado de 1 bit
- Predicción
  - Valor del registro de historia

#### LIMITACIÓN

- Sólo se registra el comportamiento de la última ejecución del salto
- Dos malas predicciones en los cambios



Predicción: *Taken*

Predicción: *Not taken*

Más Bits

#### Cambios de estado:

- T: el salto ha sido tomado
- NT: el salto no ha sido tomado

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos

### Predictor de dos bits (BIMODAL)

- Utilizan dos bits de predicción por cada instrucción de salto
- Estos bits reflejan el comportamiento de las últimas ejecuciones de ese salto

#### Predicción

- Un salto que se toma repetidamente se predice como *Taken*
- Un salto que no se toma repetidamente se predice como *Not taken*
- Si un salto toma una dirección inusual una sola vez, el predictor mantiene la predicción usual

#### Funcionamiento

Máquina de cuatro estados:

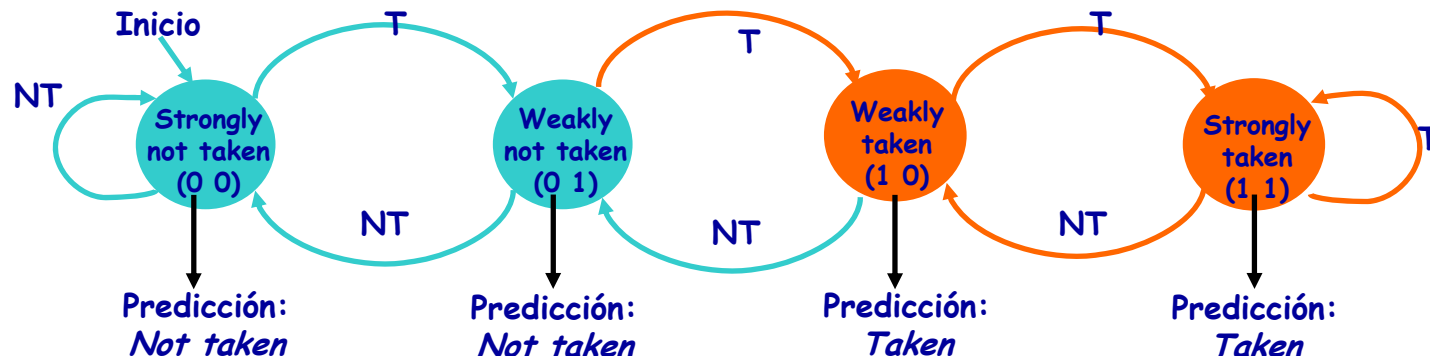
- Strongly not taken (00)
- Weakly not taken (01)
- Weakly taken (10)
- Strongly taken (11)

• Registro de historia

- Contador saturado de 2 bits

• Predicción

- bit más significativo del registro de historia



Cambios de estado:

T: el salto ha sido tomado

NT: el salto no ha sido tomado

# Tratamiento de Saltos: Predicción

## □ Predictores Dinámicos

### *Implementación de los bits de predicción*

#### 1) Branch Target Buffer (BTB)

Añade los bits de predicción a las entradas de la BTAC. La BTAC con bits de predicción se denomina BTB

##### EJEMPLOS

• MC 68060	256 x 2 bit
• Pentium	256 x 2 bit
• R8000	1K x 1 bit
• PM1	1K x 2 bit
• Pentium II	512x2 bit
• Pentium 4	4Kx2bits

#### 2) Tabla de historia de saltos (BHT)

Utiliza una tabla especial, distinta de la BTAC para almacenar los bits de predicción

##### EJEMPLOS

• Gmicro 100	256 x 1 bit
• PowerPC 604	512 x 2 bit
• R10000	512 x 2 bit
• PowerPC 620	2K x 2 bit
• PA 8000	256 x 3 bit
• Alpha 21164A	2K x 2 bit
• AMD64	16Kx2bits

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos: Implementación

### 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*

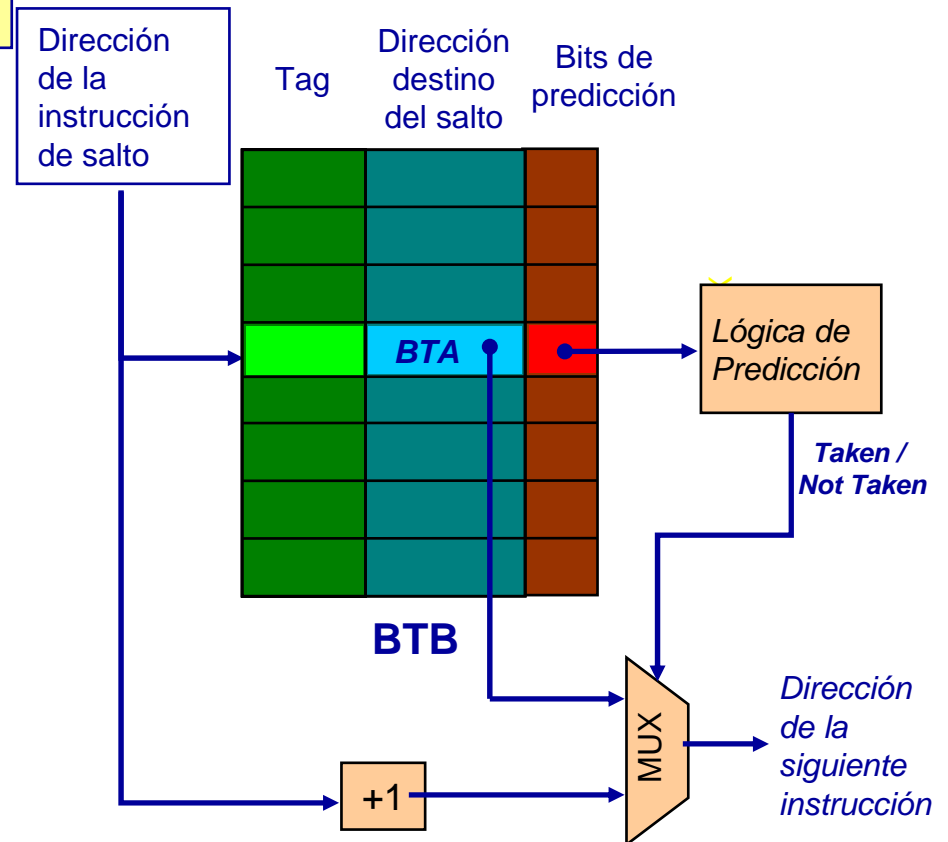
*Los campos de la BTB se actualizan después de ejecutar el salto y se conoce:*

- 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

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos: Implementación

### 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 todas las instrucciones de salto condicional

#### **Ventaja**

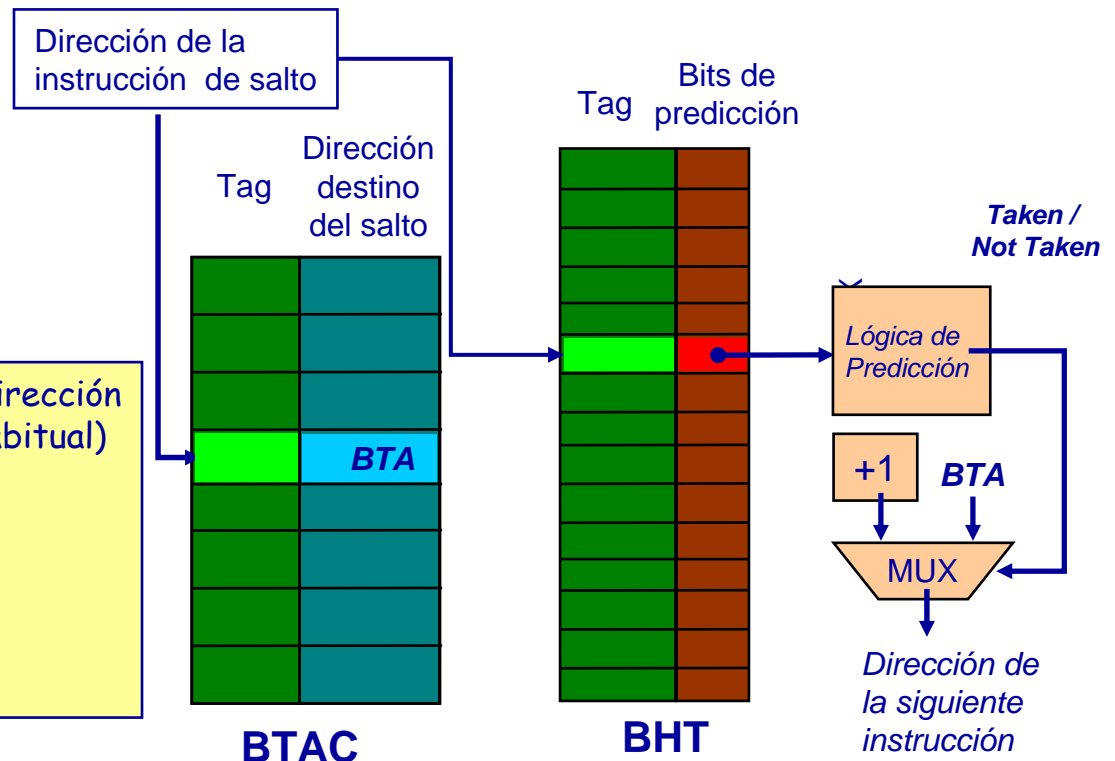
Puede predecir instruc. que no están en la BTAC (más entradas en BHT que en BTAC)

#### **Desventaja**

Aumenta el hardware necesario  
⇒ 2 tablas asociativas

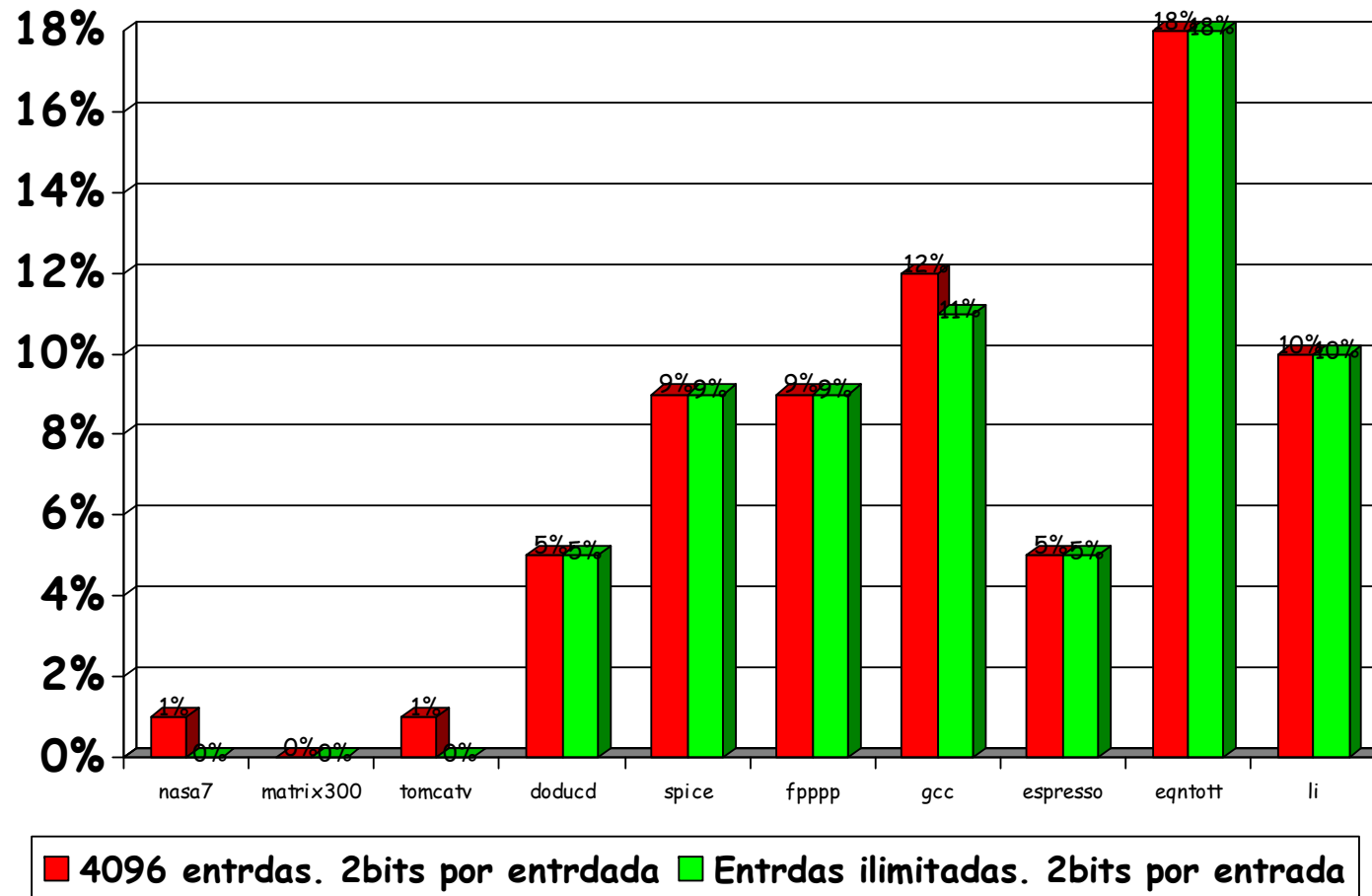
#### **Acceso a la BHT**

- 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



# Tratamiento de Saltos: Predicción

## □ Predictores Dinámicos Comportamiento



Muchos fallos en algunos programas (enteros) ¿ Por que?

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos

Otras formas de gestionar la historia

### 1. Muchas instrucciones de salto ejecutan patrones repetitivos

**Idea** : Los saltos están relacionados. El comportamiento de los últimos saltos afecta a la predicción actual. Almacenar el comportamiento de los últimos  $n$  saltos y usarlo en la selección de la predicción.



*Si conocemos el comportamiento del salto en las 3 últimas ejecuciones podemos predecir como se comportará en la siguiente ejecución*

⇒ Predicción basada en historia **LOCAL**

Historia	Predicción
111	0 (NT)
011	1 (T)
101	1 (T)
110	1 (T)



# Tratamiento de Saltos: Predicción

## □ Predictores Dinámicos

2. Muchas instrucciones de salto dependen del comportamiento de otros saltos recientes ( historia global)

Ejemplo:

b1: if (d = 0) then  
    d = 1  
b2: if (d = 1) then

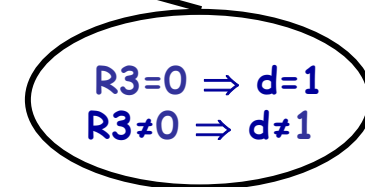


L1:

L2:

BNEZ R1, L1 ; salto b1 (Salto si d ≠ 0)  
ADDI R1, R0, #1 ; Como d=0, hacer d=1  
SUBI R3, R1, #1 ; R3=d(R1)-1  
BNEZ R3, L2 ; salto b2 (Salto si d≠ 1)

.....



*Si conocemos el comportamiento de la última ejecución de b1 podemos predecir el comportamiento de b2 en la siguiente ejecución*



Predicción basada en historia **GLOBAL**

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos

### Ejemplo ( continua )

Relación entre los dos saltos

L1:       BNEZ   R1, L1 ; salto b1 (Salto si  $d \neq 0$ )  
          ADDI   R1, R0, #1 ; Como  $d=0$ , hacer  $d=1$   
          SUBI   R3, R1, #1 ;  $R3=d(R1)-1$   
          BNEZ   R3, L2 ; salto b2 (Salto si  $d \neq 1$ )  
L2:       .....

Caso 1:  $d=0,1,2,\dots$

Valor de d	$d \neq 0?$	b1	d antes de b2	$d \neq 1?$	b2
0	no	NT	1	no	NT
1	si	T	1	no	NT
2	si	T	2	si	T

Si b1 no se toma, entonces b2 tampoco: correlación entre saltos

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos

### Ejemplo ( continua )

Comportamiento del predictor de un bit. ( estado inicial "not taken" NT)

L1:       BNEZ   R1, L1 ; salto b1 (Salto si  $d \neq 0$ )  
          ADDI   R1, R0, #1 ; Como  $d=0$ , hacer  $d=1$   
          SUBI   R3, R1, #1 ;  $R3=d(R1)-1$   
          BNEZ   R3, L2 ; salto b2 (Salto si  $d \neq 1$ )  
L2:       .....

Caso 2:  $d=2,0,2,0,\dots$

Valor de d	Predicción de b1	b1	Nueva predicción de b1	Predicción de b2	b2	Nueva predicción de b2
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

Muchos fallos de predicción

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos

### ▪ Solución

#### • Predictor de dos niveles (1,1):

Para cada salto existen  $2^1$  predictores de 1 bit. El comportamiento último salto (1) determina el predictor que se usa.

#### • Predictor de dos niveles (m,n)

Para cada salto existen  $2^m$  predictores de n bits. El comportamiento de los últimos m saltos determinan el predictor que se usa

Significado de los bit de predicción en un predictor (1,1)

	Bits de predicción	Predicción si el ultimo salto no tomado	Predicción si el ultimo salto tomado
	NT/NT	NT	NT
Dos predictores de un bit	NT/T	NT	T
	T/NT	T	NT
	T/T	T	T

Ejemplo: Tamaño de un predictor (2,2) de 4k entradas  $4 \times 2 \times 4K = 32Kb$

# Tratamiento de Saltos: Predicción

## □ Predictores Dinámicos

### Ejemplo ( continua )

Comportamiento del predictor de dos niveles (1,1). ( estado inicial "not taken" NT)

	BNEZ	R1, L1 ; salto b1 (Salto si $d \neq 0$ )
	ADDI	R1, R0, #1 ; Como $d=0$ , hacer $d=1$
L1:	SUBI	R3, R1, #1 ; $R3=d(R1)-1$
	BNEZ	R3, L2 ; salto b2 (Salto si $d \neq 1$ )
L2:	.....	

Caso 2:  $d=2,0,2,0,\dots$

Sólo se predice mal la 1ª iteración ( $d=2$ )

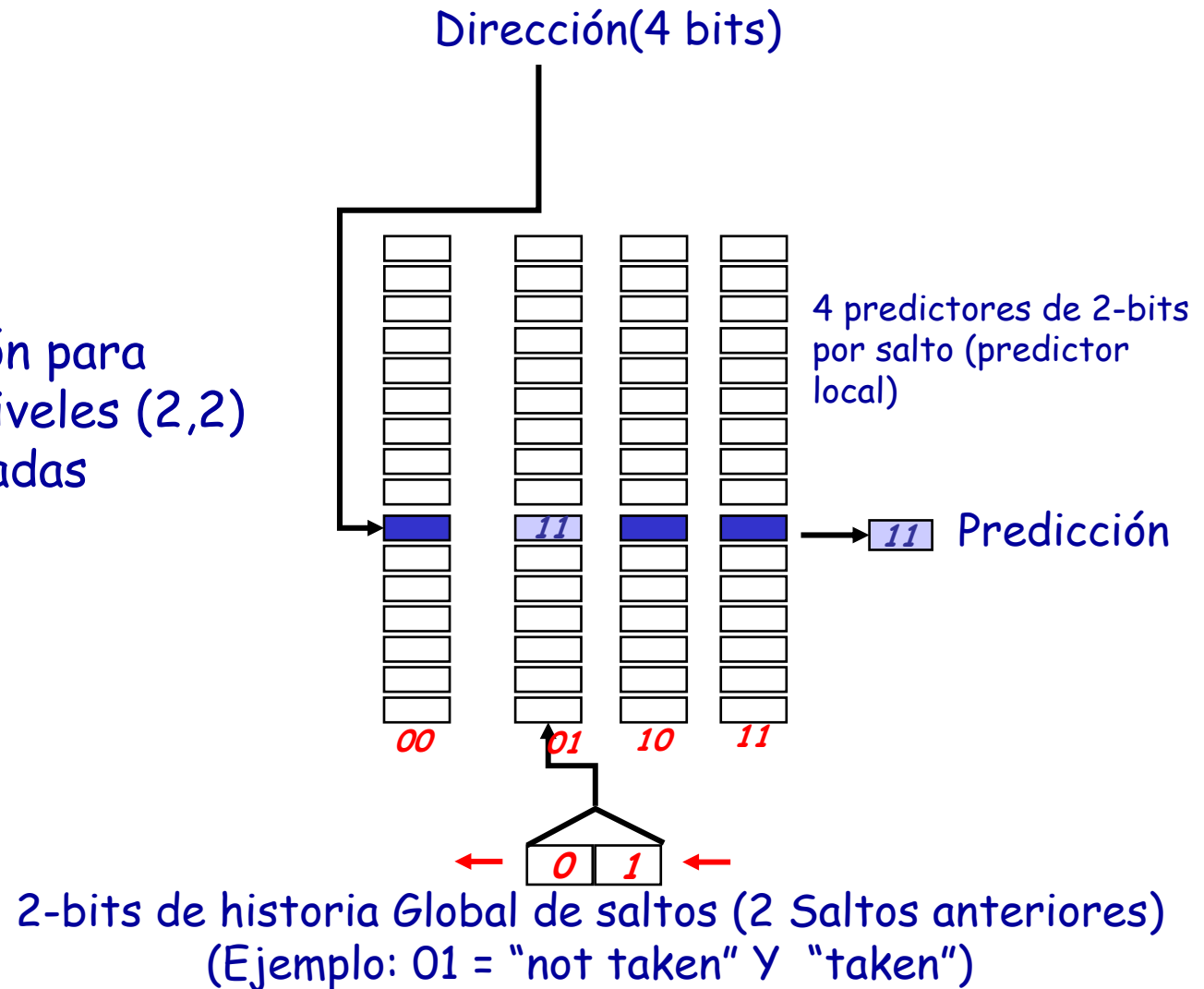
d = ?	Predicción de b1	b1	Nueva predicción de b1	Predicción de b2	b2	Nueva predicción de b2
2	NT/NT	<u>T</u>	T/NT	NT/ <u>NT</u>	<u>T</u>	NT/T
0	<u>T</u> /NT	NT	T/NT	<u>NT</u> /T	NT	NT/T
2	<u>T</u> /NT	T	T/NT	NT/ <u>T</u>	T	NT/T
0	<u>T</u> /NT	NT	T/NT	<u>NT</u> /T	NT	NT/T

(Subrayado en rojo: Bit de predicción seleccionado en cada caso, en función del comportamiento del salto anterior)

# Tratamiento de Saltos: Predicción

## ❑ Predictores Dinámicos

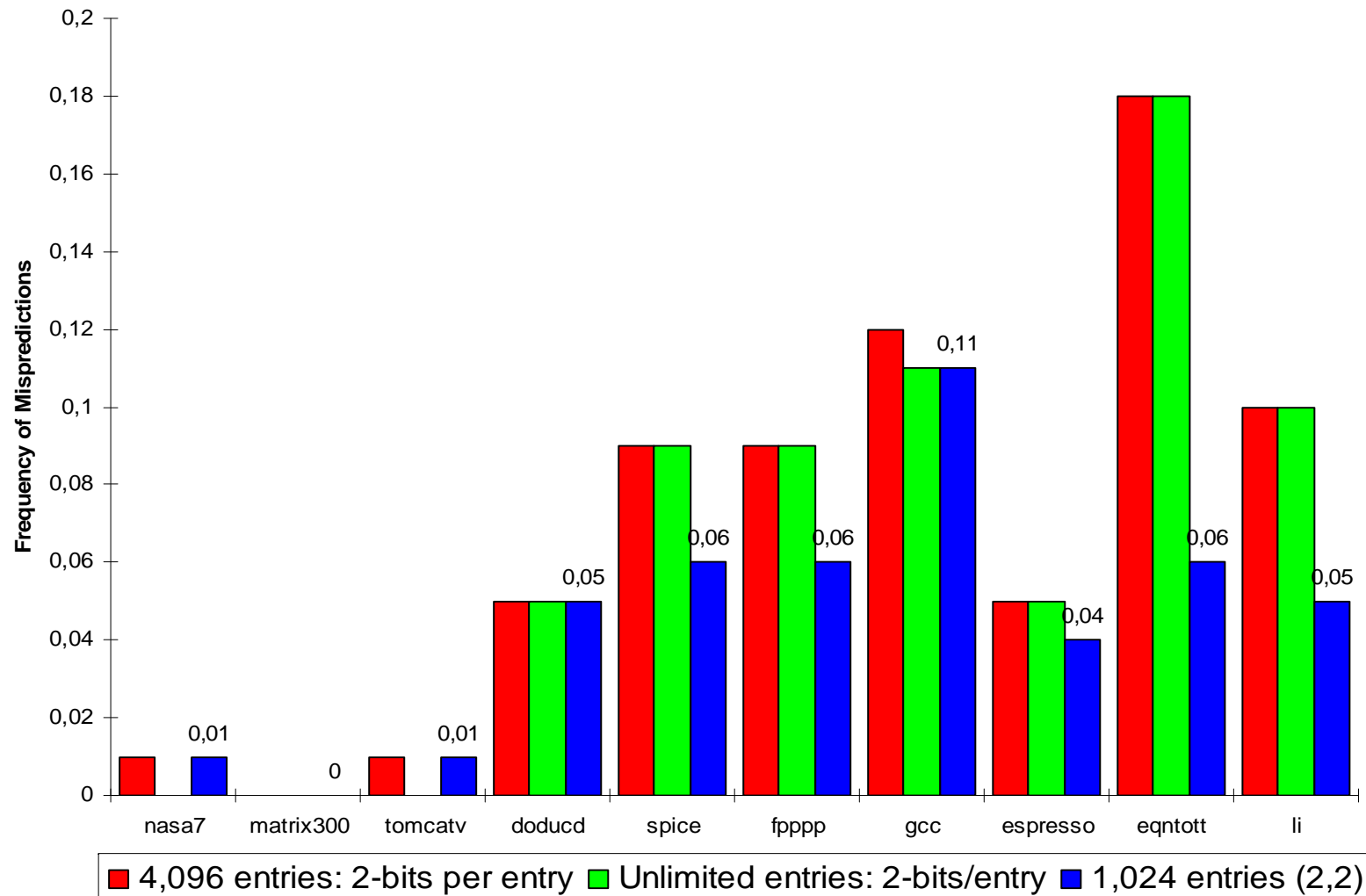
Implementación para  
Predictor de dos niveles (2,2)  
con 16 entradas



# Tratamiento de Saltos: Predicción

## □ Predictores Dinámicos

### Comportamiento



# Tratamiento de Saltos: Predicción

## ❑ Predictores híbridos

### Idea básica

- Cada uno de los predictores estudiados tiene sus ventajas y sus inconvenientes
- Combinando el uso de distintos predictores y aplicando uno o otro según convenga, se pueden obtener predicciones mucho más correctas

### Predictor híbrido

Mezcla varios predictores y añade un mecanismo de selección del predictor

### Mecanismo de selección

Elige, en cada caso, el predictor que haya dado mejores resultados hasta el momento

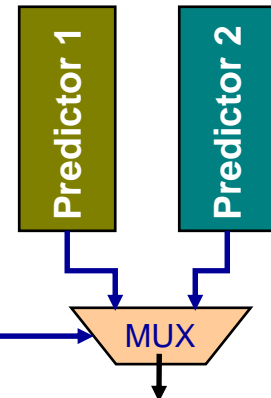
### Implementación del mecanismo de selección

Para combinar dos predictores, P1 y P2, se utiliza una tabla de **contadores saturados de dos bits** indexada por la dirección de la instrucción de salto

Instrucción captada

Dirección

Tabla de Selección



Predicción  
Taken/Not taken)

P1	P2	Actualiz. del contador
Fallo	Fallo	Cont no varía
Fallo	Acierto	Cont = Cont +1
Acierto	Fallo	Cont = Cont -1
Acierto	Acierto	Cont no varía

- Si P2 acierta más que P1  
⇒ *Cont* aumenta
- Si P1 acierta más que P2  
⇒ *Cont* disminuye

Bit más signif. del contador	Predictor seleccionado
0	P1
1	P2



# Ejemplo: Alpha 21264

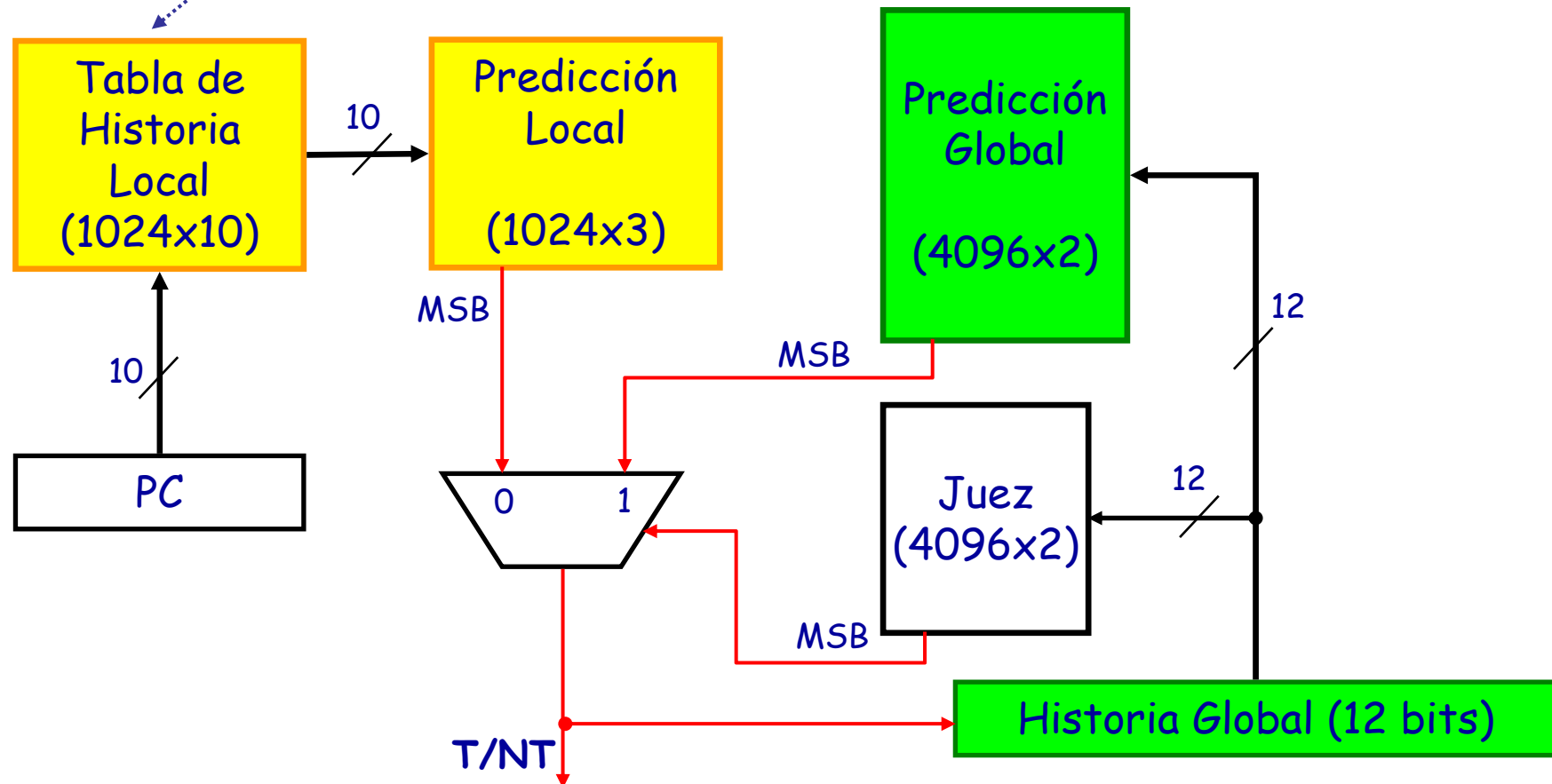
---

- ❑ Predictor competitivo (Tournament Predictor)
- ❑ Predictor Local: Predicción de un salto en función del comportamiento previo de ese mismo salto
  - o Considera las 10 últimas ejecuciones del salto
- ❑ Predictor global: Predicción de un salto en función del comportamiento de los últimos 12 saltos ejecutados
- ❑ Juez: Decide cuál de las dos predicciones se aplica
  - o Selecciona el predictor que esté manifestando el mejor comportamiento
- ❑ Actualización: al resolver cada salto
  - o Se actualizan los predictores en función de su acierto o fallo
  - o Si los dos predictores hicieron una predicción distinta, se actualiza el juez para que favorezca al que acertó
- ❑ Gran importancia para la ejecución especulativa en 21264 (hasta 80 instrucciones en la ventana)
- ❑ Tasas de predicción correcta (benchmarks): 90-100%

# Tournament predictor del Alpha 21264

Comportamiento de las 10 últimas  
ejecuciones de 1024 saltos

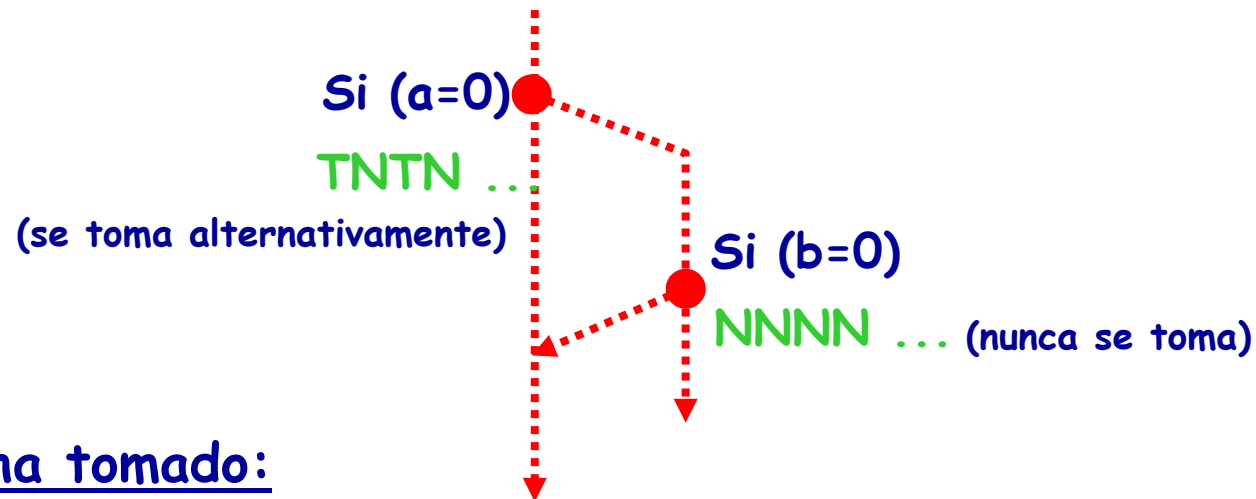
(IEEE Micro, Marzo 1999)



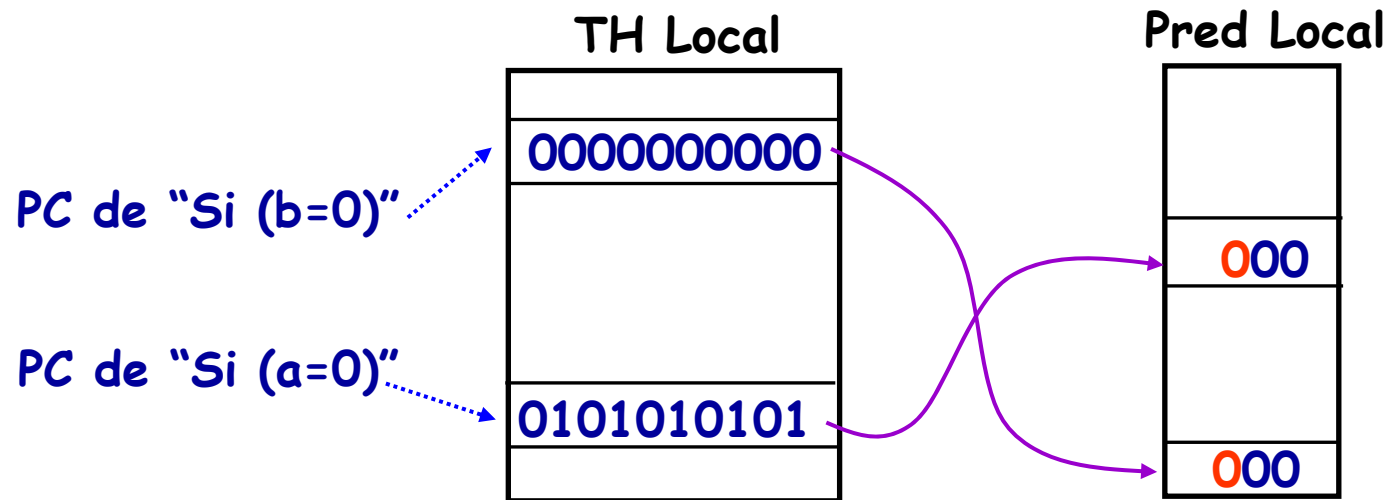
Juez: Acierto global y fallo local => incrementa  
Fallo global y acierto local => decrementa

# Ejemplos de funcionamiento (1)

Programa con dos saltos que tiene el comportamiento descrito

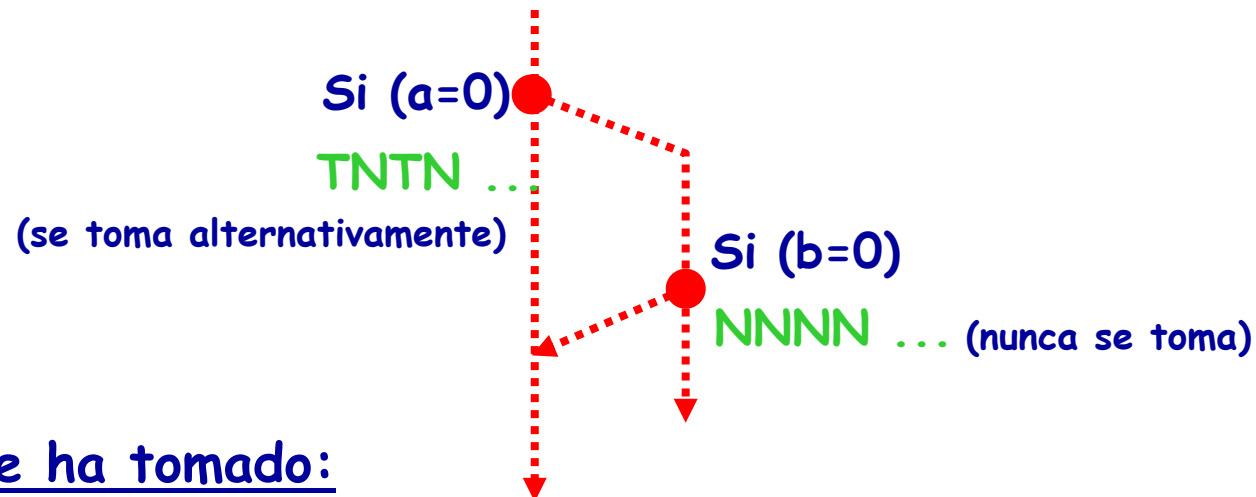


"Si(a=0)" se ha tomado:

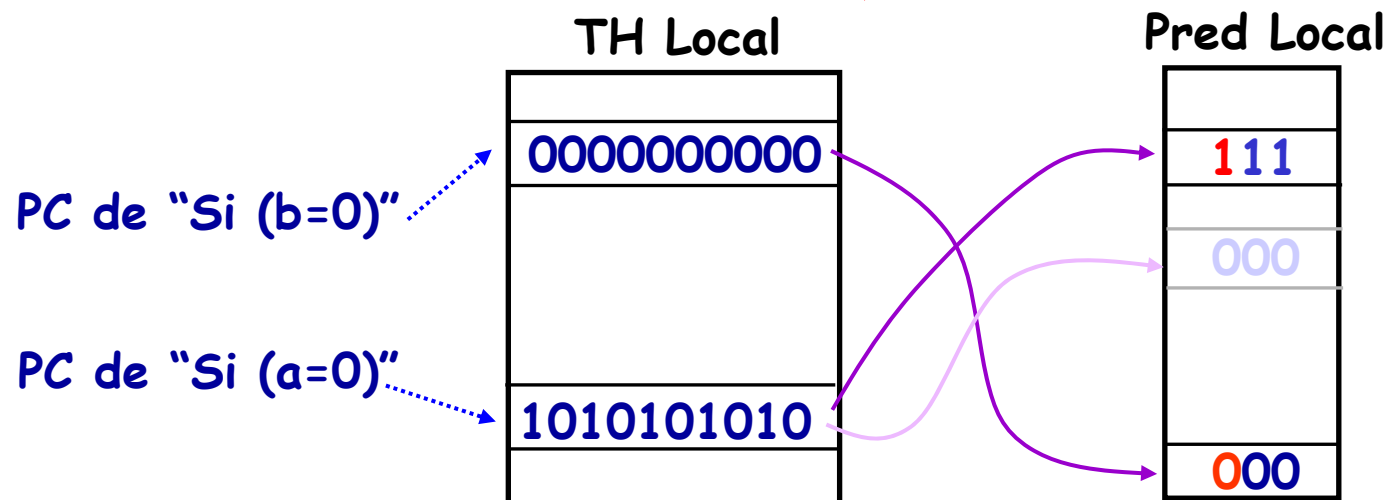


## Ejemplos de funcionamiento (2)

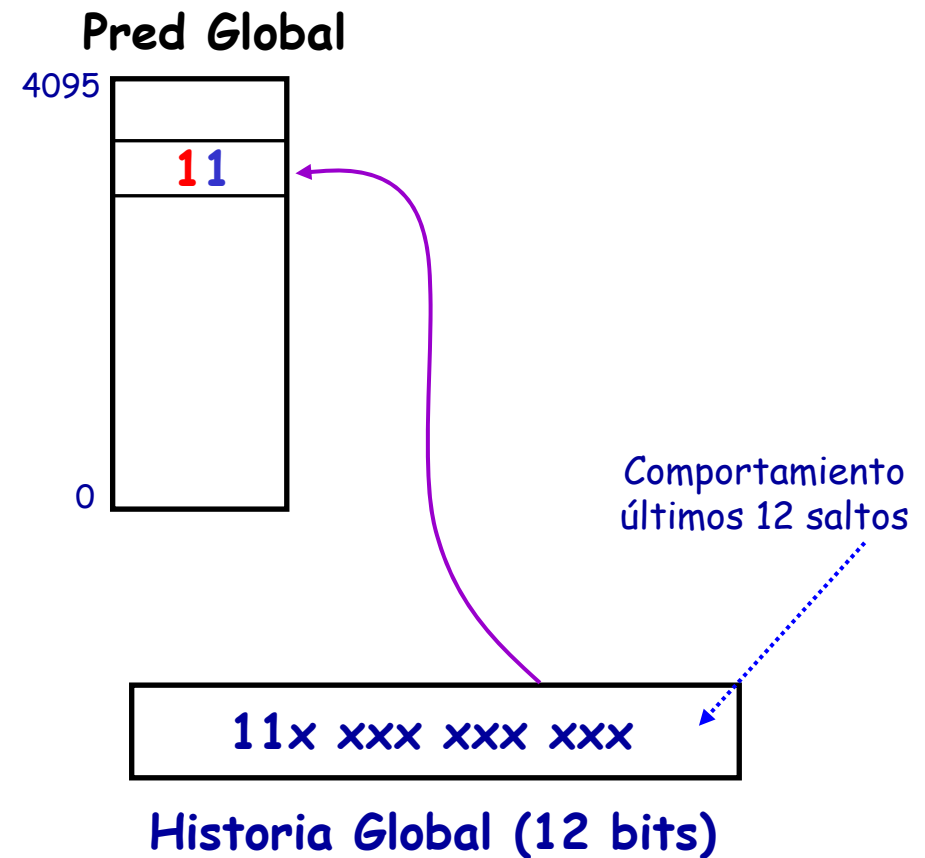
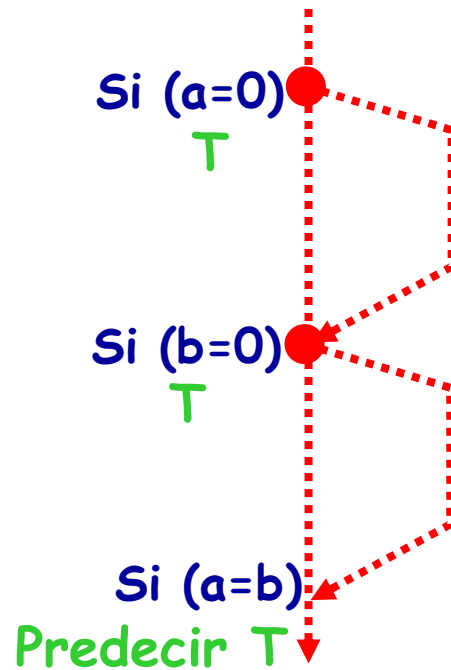
Programa con dos saltos que tiene el comportamiento descrito



"Si(a=0)" no se ha tomado:

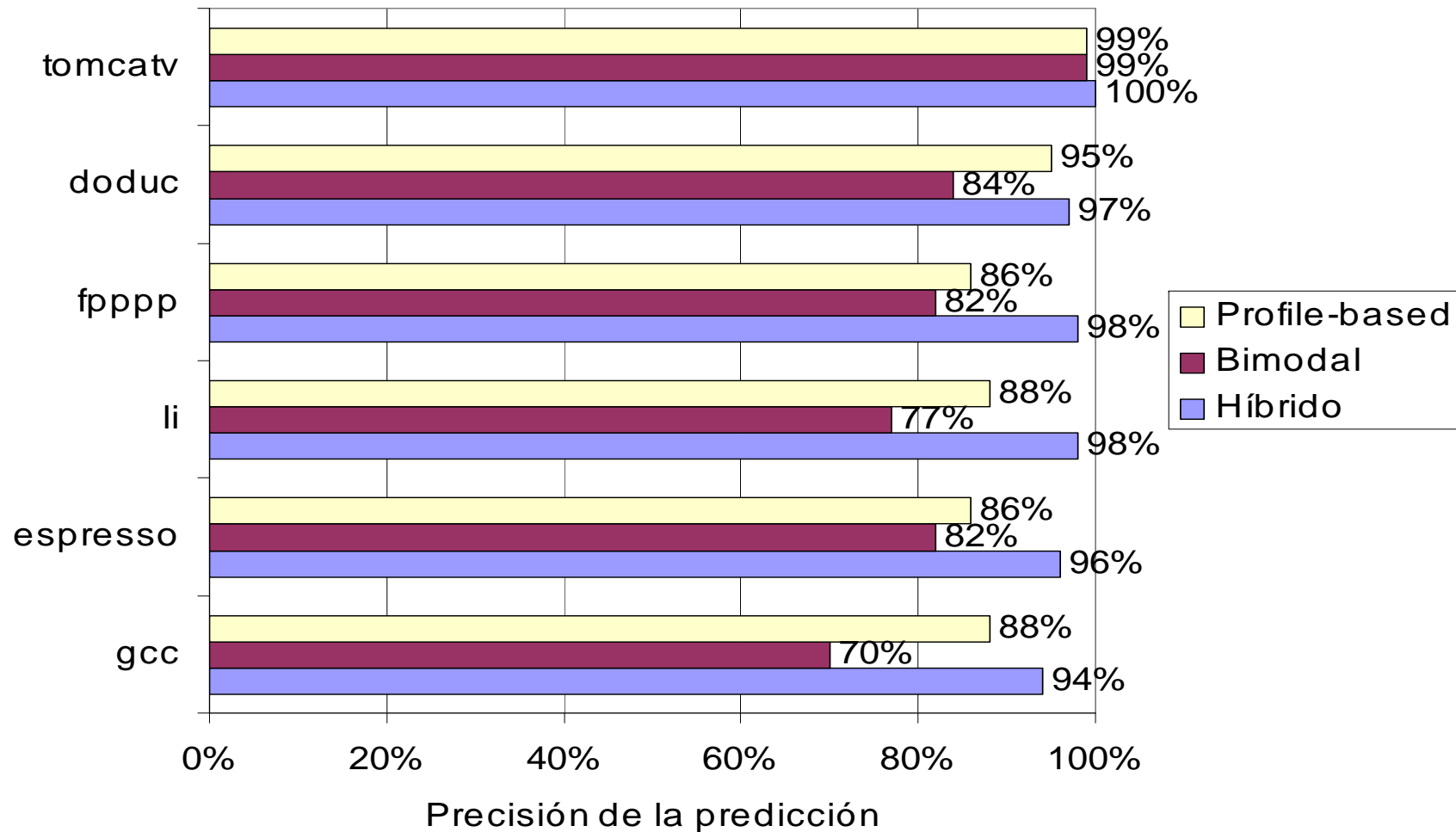


\_\_\_\_\_



# Tratamiento de Saltos: Predicción

## □ Predictores: Comportamiento

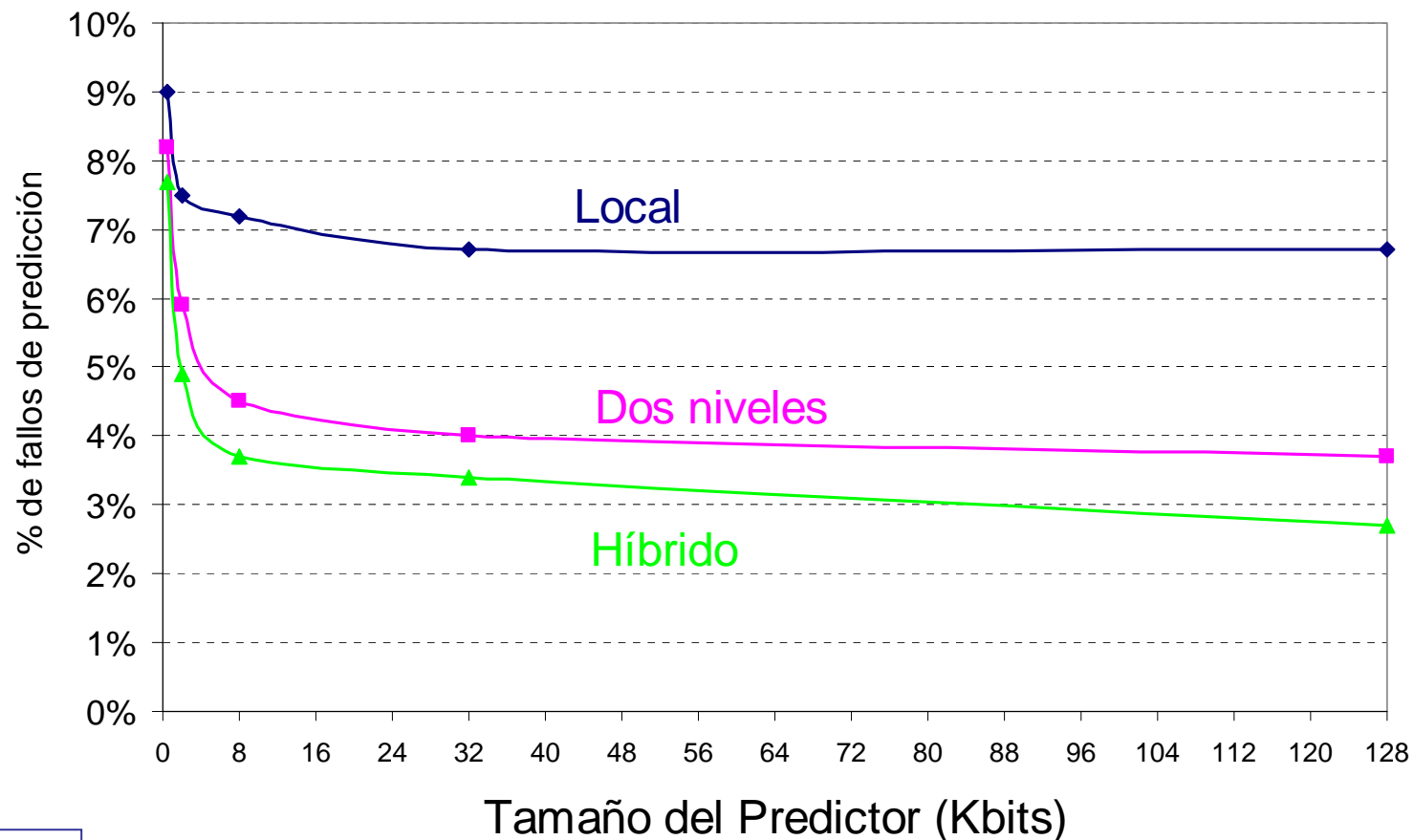


Profile\_based- Predictor estático

# Tratamiento de Saltos: Predicción

## □ Predictores: Comportamiento

- La ventaja del predictor híbrido es su capacidad de seleccionar el predictor correcto para un determinado salto
- Muy importante para programas enteros
  - Un predictor híbrido selecciona el global casi 40% de las veces para SPEC integer y menos del 15% de las veces para SPEC FP

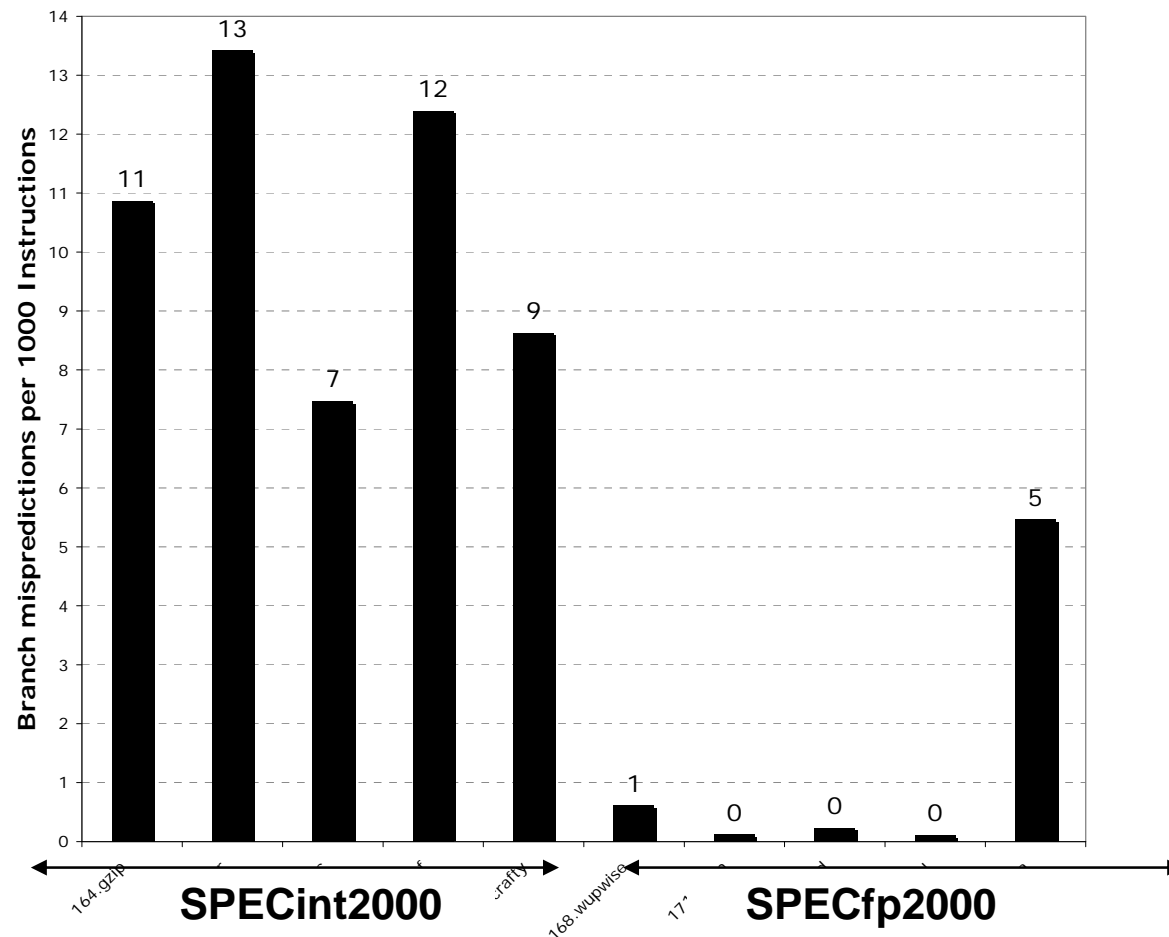


## Tratamiento de Saltos: Predicción

- ❑ Pentium 4 : tasa de fallos de predicción ( por 1000 instrucciones, no por salto)

≈6% de tasa de fallos SPECint (19% instrucciones INT son saltos, 186 de 1000 )

≈2% de tasa de fallos SPECfp ( 5% instrucciones FP son saltos, 48 de 1000)





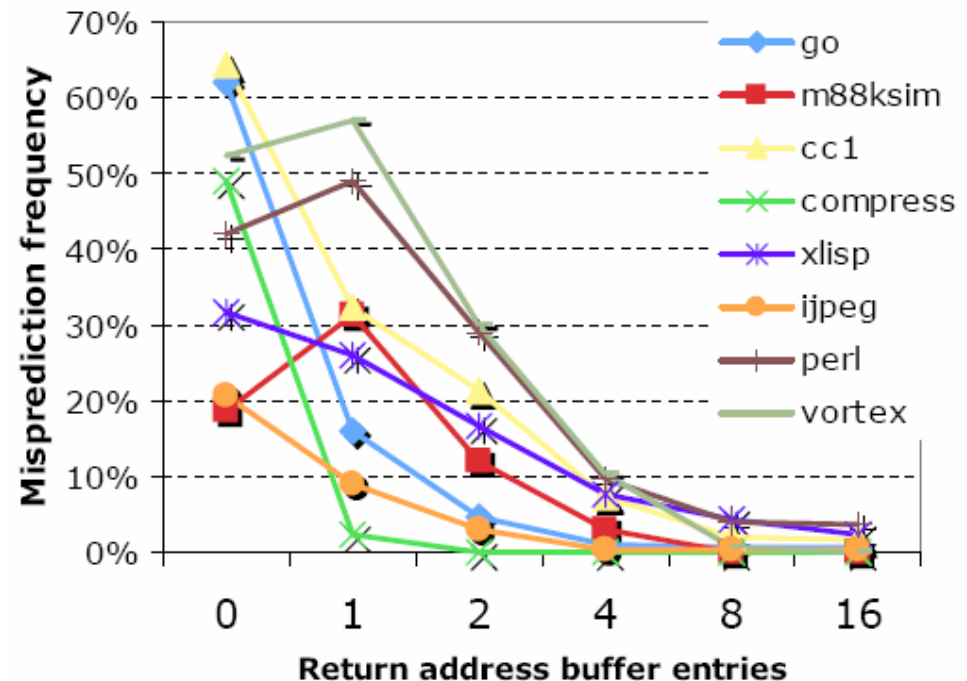
# Tratamiento de Saltos: Predicción

## □ Predicción de los retornos

- La precisión de los predictores con los retornos es muy baja: La dirección de retorno es diferente en función de la llamada
- Solución : Pila de direcciones de retorno( 8 a 16 entradas )

### EJEMPLOS

- UltraSparc I, II      4 entradas
- Pentium Pro          16 entradas
- R10000                1 entrada



# Tratamiento de Saltos: Predicción

## ❑ Detección de la instrucción de salto

### Retardos en la detección del salto

- Para poder aplicar el mecanismo de predicción es necesario saber que se trata de una instrucción de salto
- Problema: el tipo de instrucción no se conoce hasta que no finaliza la etapa de DECODIFICACIÓN  
⇒ Varios ciclos de retardo en cada salto



**Tipo de instrucción**  
Si es salto ⇒ Aplicar predicción

### Alternativas al esquema básico

#### Detección en paralelo con la Decodificación

Utiliza un decodificador de saltos dedicado capaz de detectar las instrucciones de salto antes del final de la etapa decodificación



**Detección de saltos**

#### EJEMPLOS

- Alpha 21064
- Alpha 21164
- Alpha 21064A
- PowerPC601

#### Detección después de la etapa de *Fetch*

Detecta las instrucciones de salto en el buffer de instrucciones, antes de que sean decodificadas



**Detección de saltos**

#### EJEMPLOS

- Power1
- Power2
- PowerPC 603

#### Detección Durante la etapa de *Fetch*

Detecta las instrucciones de salto al tiempo que se leen de la cache de instrucciones



**Detección de saltos**

#### EJEMPLOS

- PowerPC 604
- PowerPC 620
- R8000
- PA8000

# Tratamiento de Saltos: Predicción

## ❑ Recuperación de fallos de predicción (misprediction)

### Tareas básicas

- 1) Descartar los resultados de las instrucciones ejecutadas especulativamente
- 2) Reanudar la ejecución por el camino correcto con un retardo mínimo

### 1) Descarte de los resultados

- Los resultados de estas instrucciones especulativas se almacenan en registros temporales (registros de renombramiento o *Buffer* de reordenamiento)
- Estas instrucciones no modifican los contenidos de los registros de la arquitectura ni de la memoria

Si la ejecución fue correcta

Se actualizan los registros de la arquitectura y/o la memoria

Si la ejecución fue incorrecta

Se descartan los resultados de los registros temporales

# Tratamiento de Saltos: Predicción

## ❑ Recuperación de fallos de predicción (misprediction)

### 2) Reanudación de la ejecución por el camino correcto

El procesador debe guardar, al menos, la dirección de comienzo del camino alternativo

Si la predicción fue "*Not taken*"

El procesador debe calcular y almacenar la dirección destino del salto

Si la predicción fue "*Taken*"

El procesador debe almacenar la dirección de la instrucción siguiente al salto

Ejemplos: PowerPC 601 - 603 - 605

### Reducción de los retardos en la recuperación de fallos

El procesador puede guardar, no solo la dirección del camino alternativo, sino prebuscar y almacenar algunas instrucciones de este camino

Si la predicción fue "*Taken*"

- El procesador almacena la dirección del camino secuencial
- El procesador prebusca y almacena las primeras instrucciones secuenciales

Si la predicción fue "*Not taken*"

- El procesador calcula y almacena la dirección destino del salto
- El procesador prebusca y almacena las primeras instrucciones del destino del salto

Ejemplos: 2 buffer Power1, Power2, Pentium, UltraSparc( 16 ), R10000 (256 bits)  
3 buffer Nx586 ( 2 pendientes )

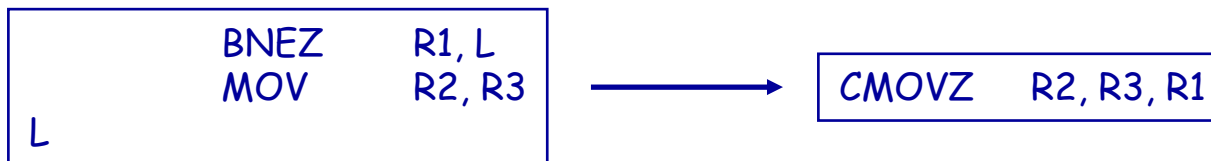
# Tratamiento de Saltos: Otras alternativas

## ❑ Ejecución condicional de instrucciones

### Idea básica

- Eliminar, parcialmente, los saltos condicionales mediante *instrucciones de ejecución condicional*
- Una instrucción de ejecución condicional está formada por:
  - Una condición
  - Una operación
- Ejecución condicional
  - Si la condición es **cierta**  $\Rightarrow$  La instrucción se ejecuta
  - Si la condición es **falsa**  $\Rightarrow$  La instrucción se comporta como NOP

### Ejemplo



**Ventaja:** Buena solución para implementar alternativas simples de control

**Desventaja:** Consumen tiempo en todos los casos. Más lentas que las incondicionales

**Ejemplos:** Alpha, Hp-Pa, MIPS, Sparc

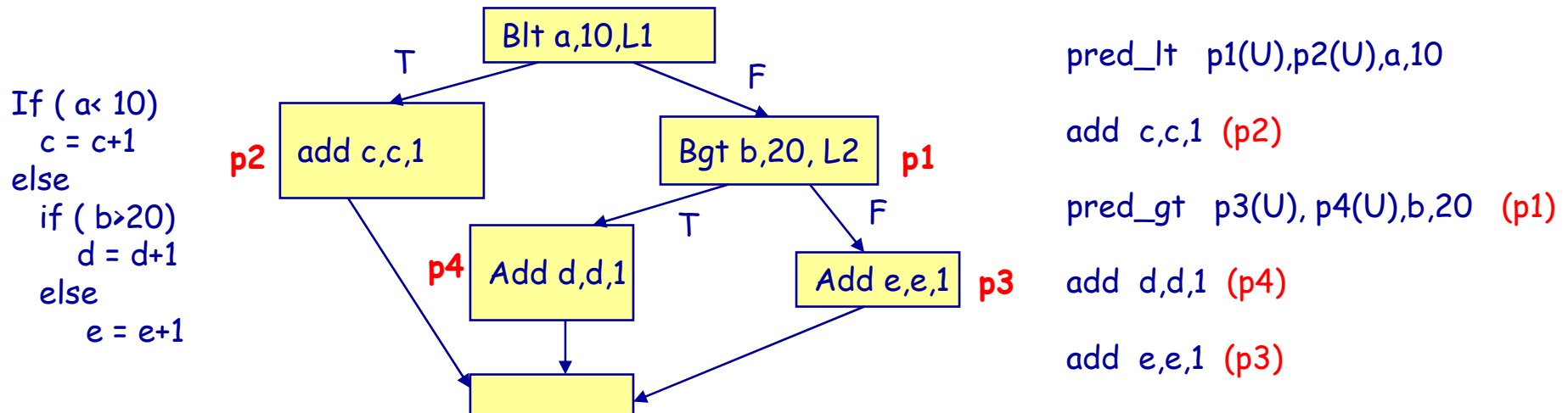
# Tratamiento de Saltos: Otras alternativas

## □ Ejecución con predicados

### Idea básica

- Transformar todas las *instrucciones en condicionales*
- Una instrucción de ejecución condicional está formada por:
  - Una parte de condición, denominada *predicado* o *guarda*
  - Una parte de operación
- Ejecución predicada:
  - Si la condición es *cierta*  $\Rightarrow$  La instrucción se ejecuta
  - Si la condición es *falsa*  $\Rightarrow$  La instrucción se comporta como NOP

### Ejemplo



## Resumen

- ✓ Predictor bimodal bueno para Loop (programas FP )
- ✓ Predictores de dos niveles buenos para IF then else
- ✓ Predicción de la dirección destino importante
- ✓ Ejecución condicional y predicada reduce el numero de saltos

# Especulación

---

## ❑ La predicción de saltos introduce ESPECULACION

- o Dos tipos de instrucciones en el procesador
  - Las independientes
  - Las que dependen de una predicción de salto. Su finalización depende del acierto o fallo en la predicción.

¿ Como podemos implementar esta distinción con un modelo de ejecución con finalización Fuera de orden?

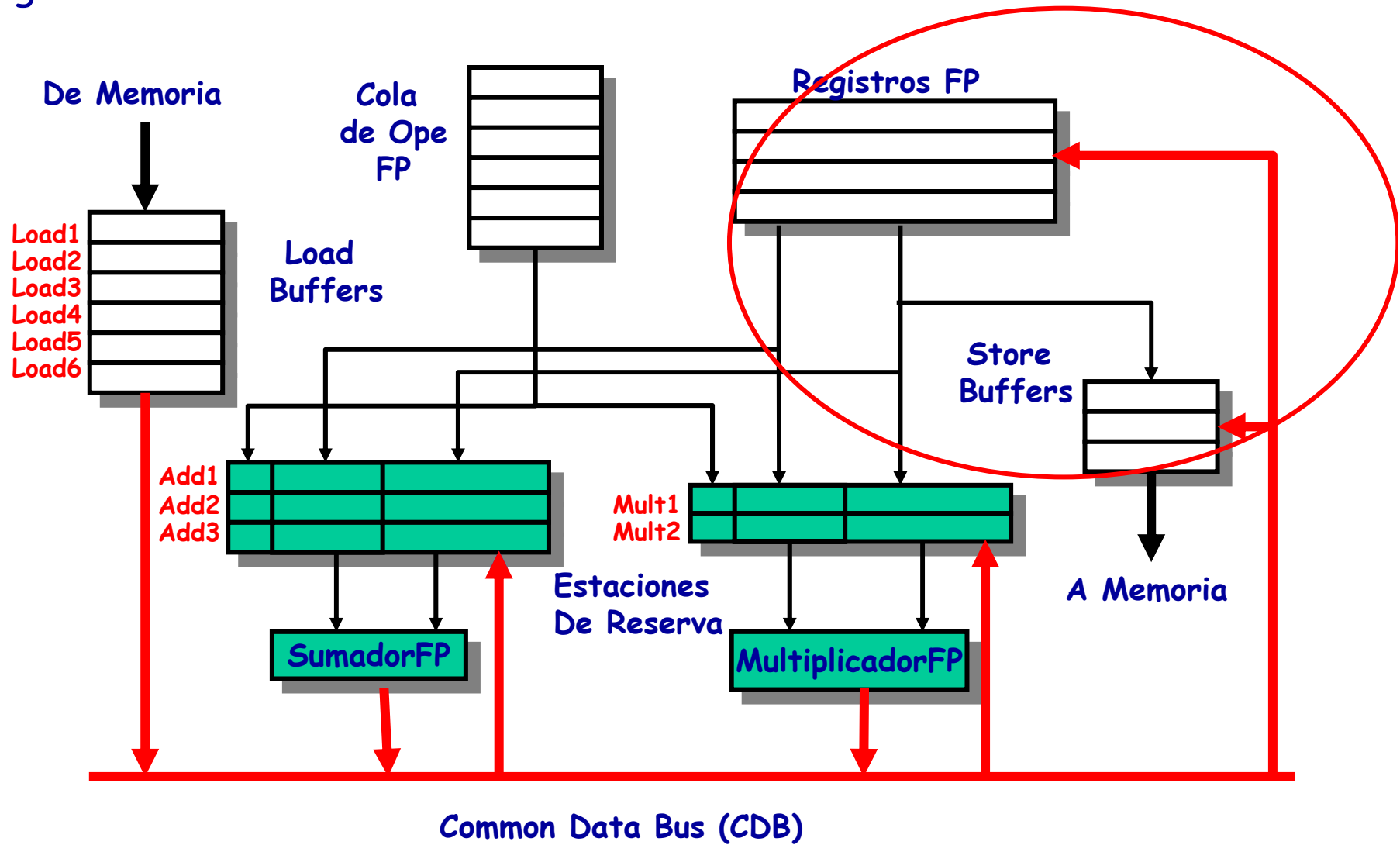
Modificando el Algoritmo de Tomasulo para **forzar** finalización en orden

ALGORITMO DE TOMASULO CON ESPECULACION



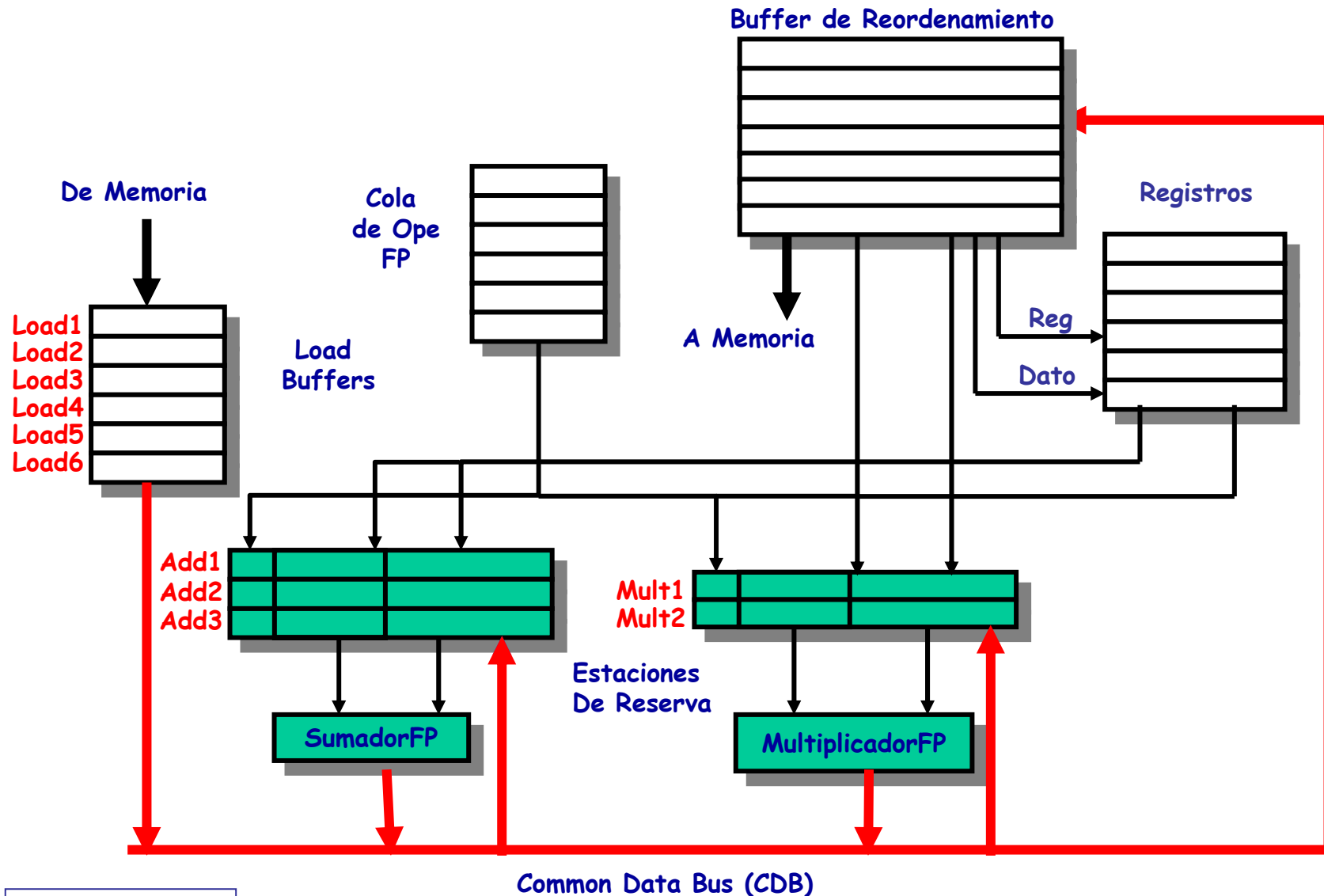
# Especulación

## ❑ Algoritmo de TOMASULO



# Especulación

## ❑ Algoritmo de TOMASULO con especulación



# Especulación

---

## ❑ El Buffer de Reordenamiento (ROB)

- o Almacena resultados de instrucciones cuya ejecución ha finalizado, pero...
  - están a la espera de actualizar registros o memoria (finalización en orden)
  - son dependientes de un salto (ejecución especulativa)
- o Permite el paso de operandos entre instrucciones especuladas con dependencia LDE.

## ❑ Los operandos de una instrucción pueden llegar hasta la ER desde:

- o Registros (la instrucción que genera el operando ha finalizado completamente)
- o ROB (la instrucción que genera el operando se ha ejecutado, pero no ha actualizado el banco de registros)
- o CBD (la instrucción que genera el operando todavía no ha realizado la fase de escritura)

# Especulación

---

- ❑ Estructura del ROB: cada entrada contiene 4 campos
  - o Tipo de instrucción
    - Salto (sin reg destino), Store (destino en memoria), Aritmética/Load (con destino en registro)
  - o Destino
    - Número de registro (Aritmética/Load)
    - Dirección de memoria (Store)
  - o Valor
    - Resultado de la ejecución de la instrucción. Guarda el valor hasta que se actualiza registro destino o memoria.
  - o Listo
    - La instrucción ha completado la fase de ejecución y el resultado está disponible en el campo "Valor"

# Especulación: fases

## ❑ Algoritmo de TOMASULO con especulación

### • Los 4 estados del algoritmo de Tomasulo especulativo

#### ✓ **Issue:** Toma la instrucción de la cola

Es necesario: ER con entrada libre y Buffer de Reordenamiento (ROB) con entrada libre.

Toma operandos de registros o de resultados almacenados en ROB por instrucciones previas.

Marca R. Destino con n° de ROB asignado

Marca ER asignada con n° de ROB asignado

#### ✓ **Ejecución:** Opera sobre los operandos

Espera hasta que los operandos estén disponibles. Chequea CDB.

#### ✓ **Escribe resultados:** Finaliza ejecución

Escribe a través de CDB en todas las ER de Fus y entradas del ROB que estén a la espera del resultado. Libera ER. No escribe en registros, ni memoria.

Envía por CDB resultado + n° de ROB al que se dirige

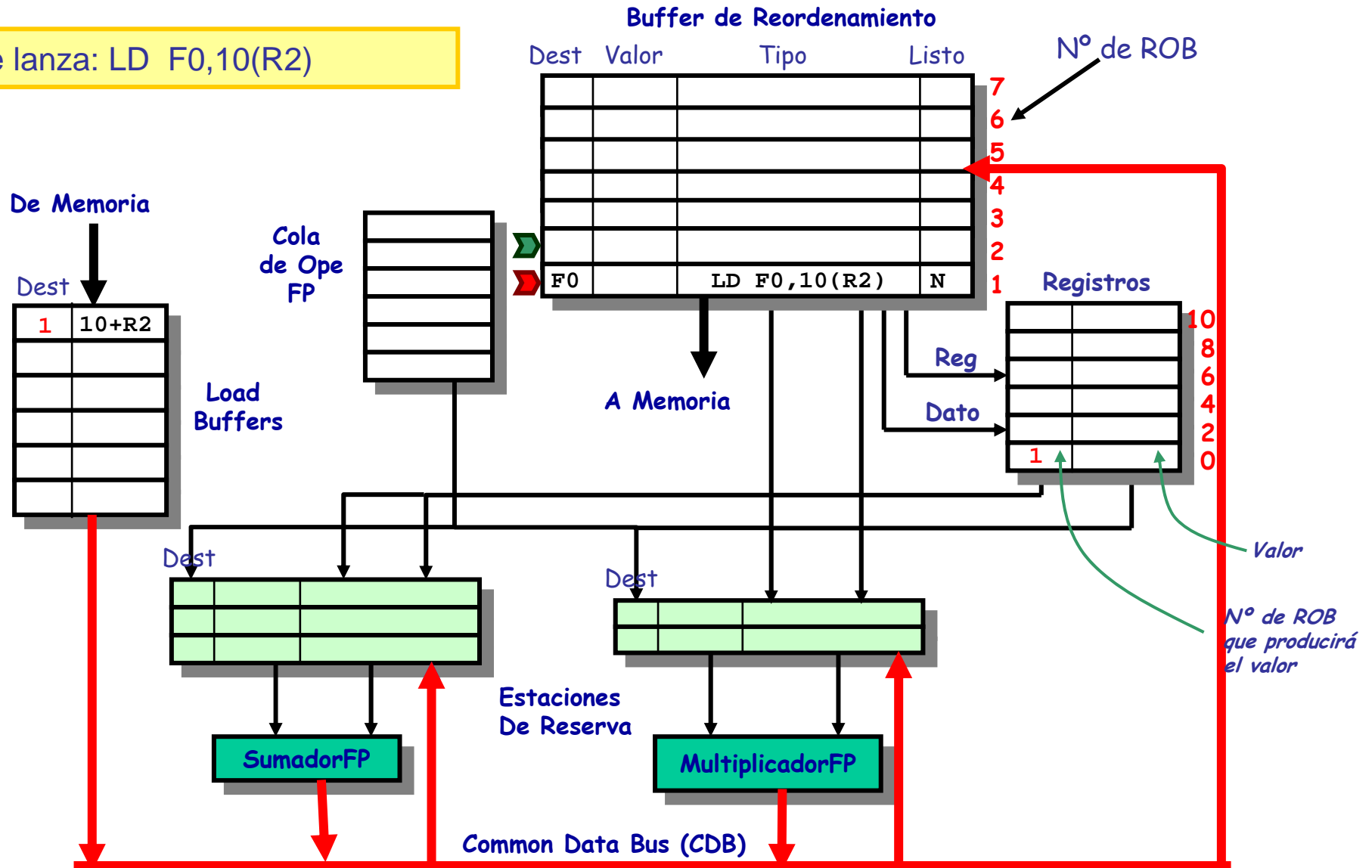
#### ✓ **Commit:** Actualiza registros desde el ROB

Cuando la Instrucción esta en la cabecera del ROB y resultado presente:  
Actualiza Registro (o escribe en memoria) y elimina la instrucción del ROB.

Tratamiento de saltos y Excepciones

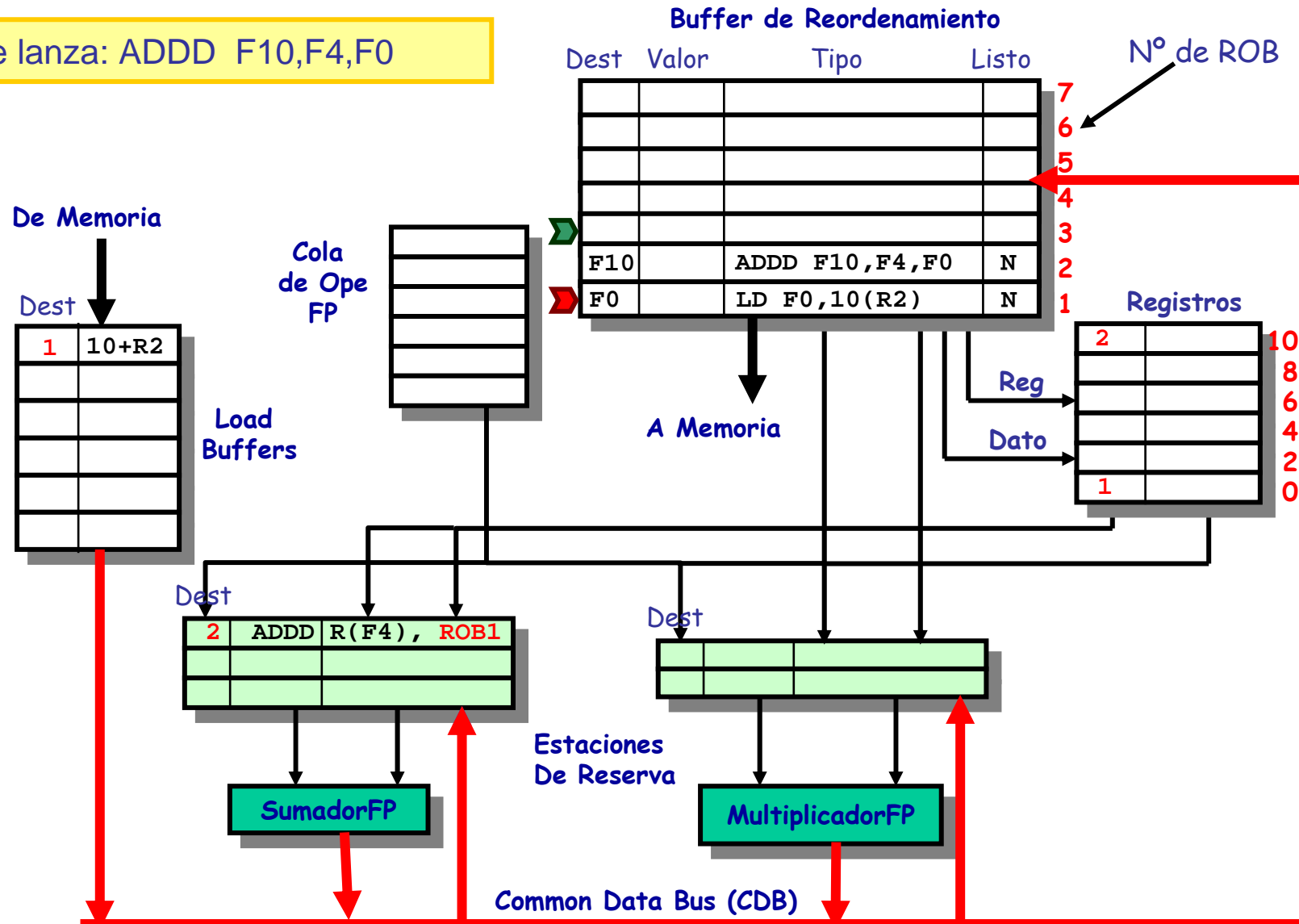
# Especulación: Ejemplo

Se lanza: LD F0,10(R2)



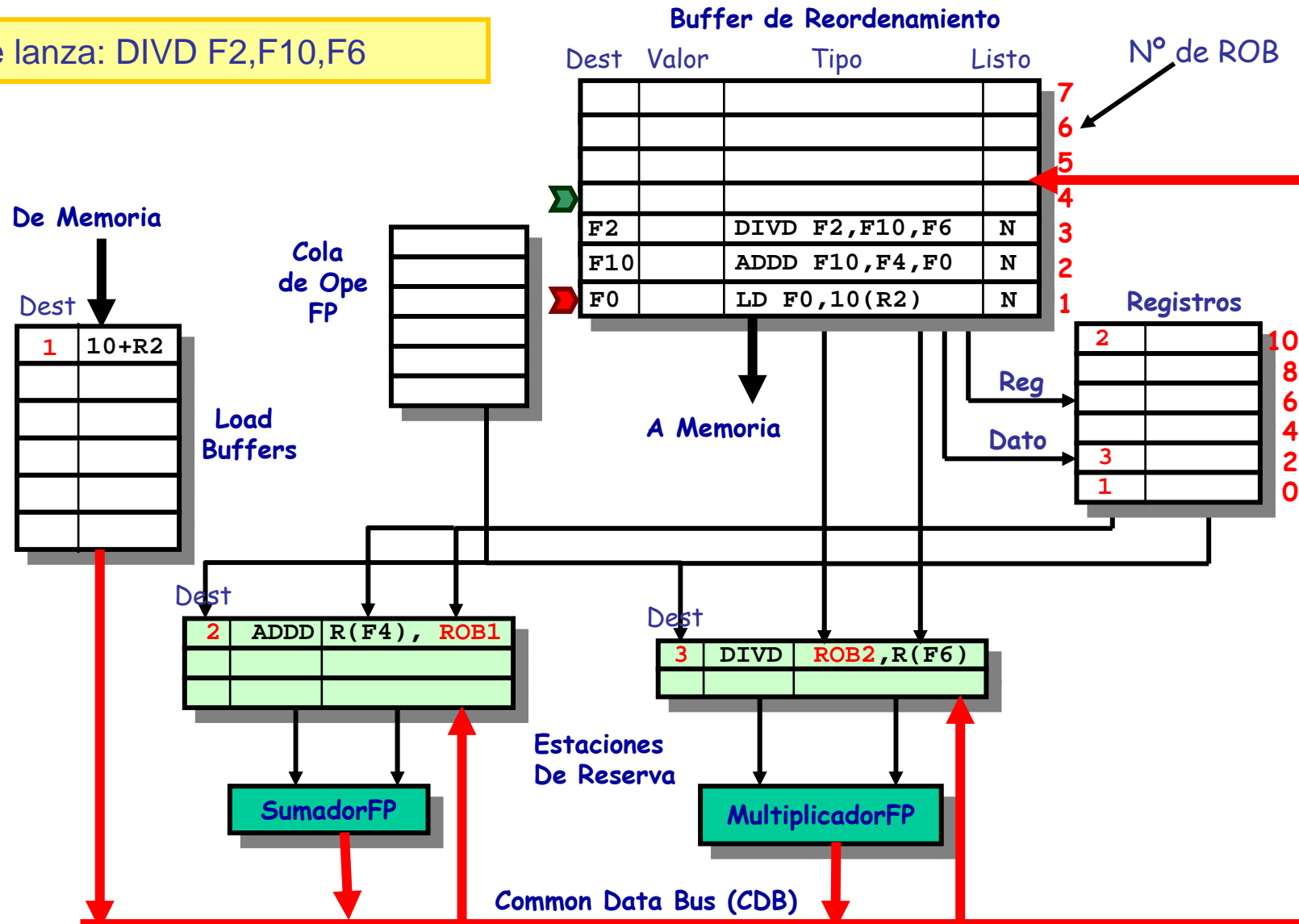
# Especulación: Ejemplo

Se lanza: ADDD F10,F4,F0



## Especulación: Ejemplo

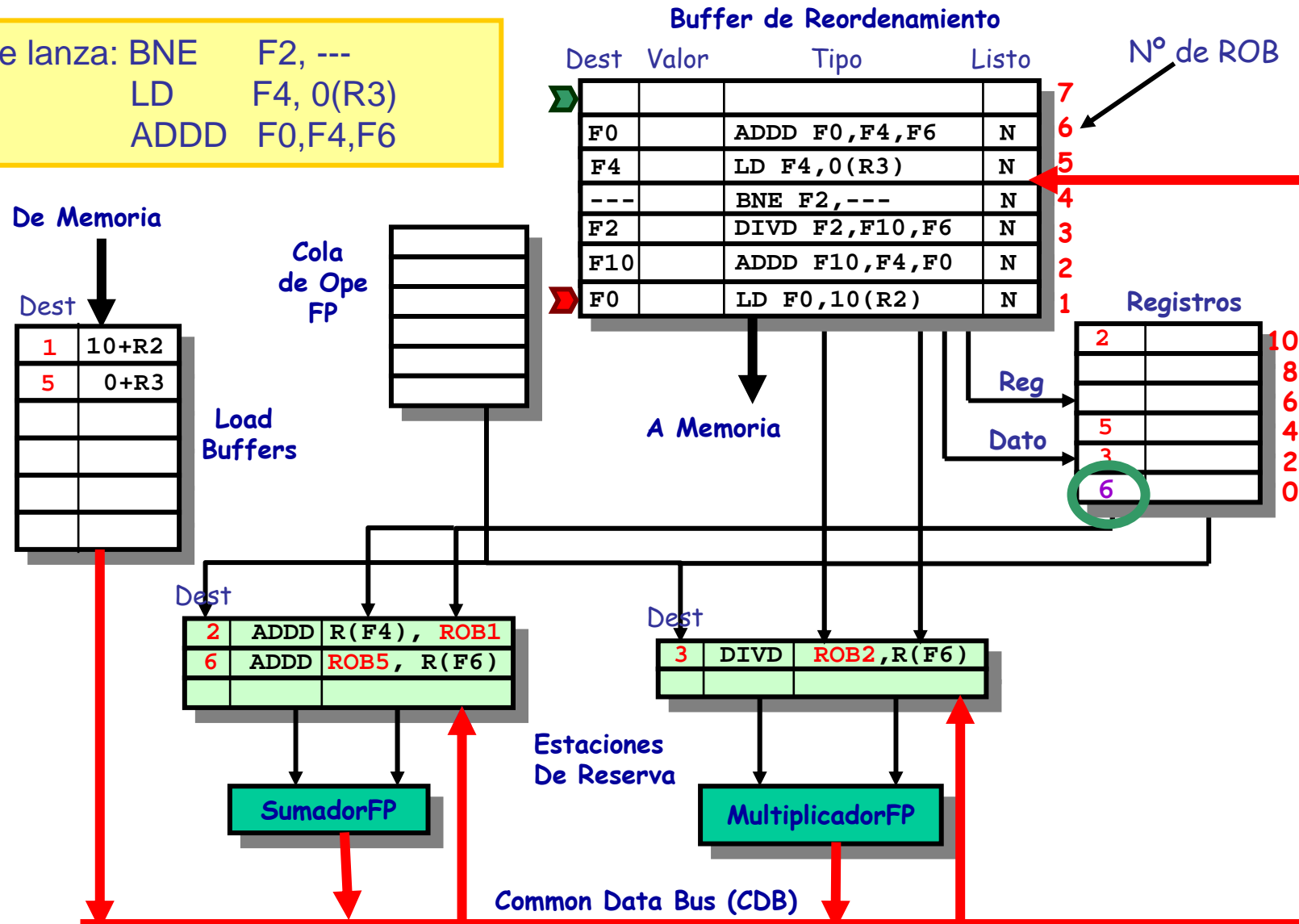
Se lanza: DIVD F2,F10,F6





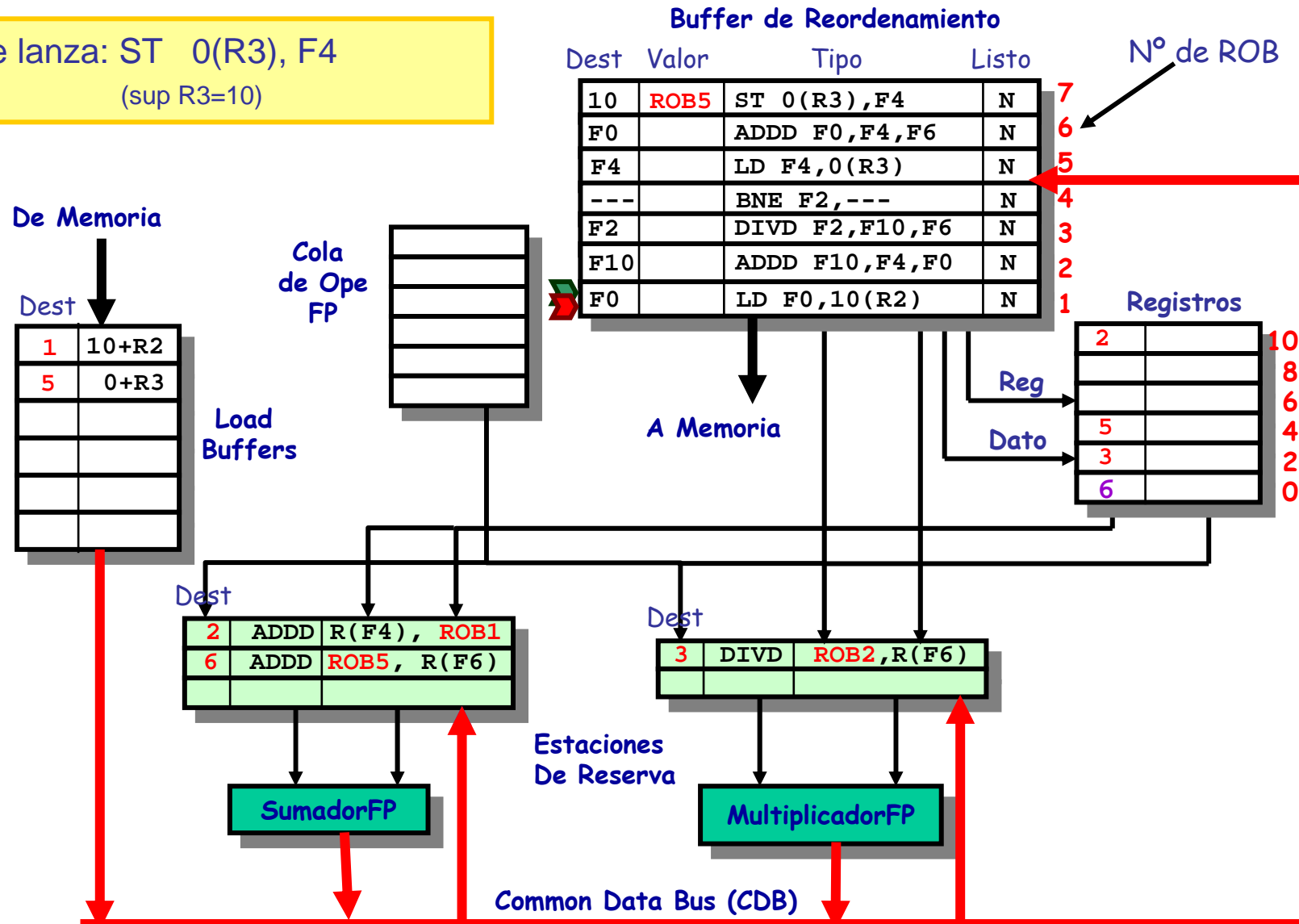
# Especulación: Ejemplo

Se lanza: BNE F2, ---  
LD F4, 0(R3)  
ADDD F0,F4,F6



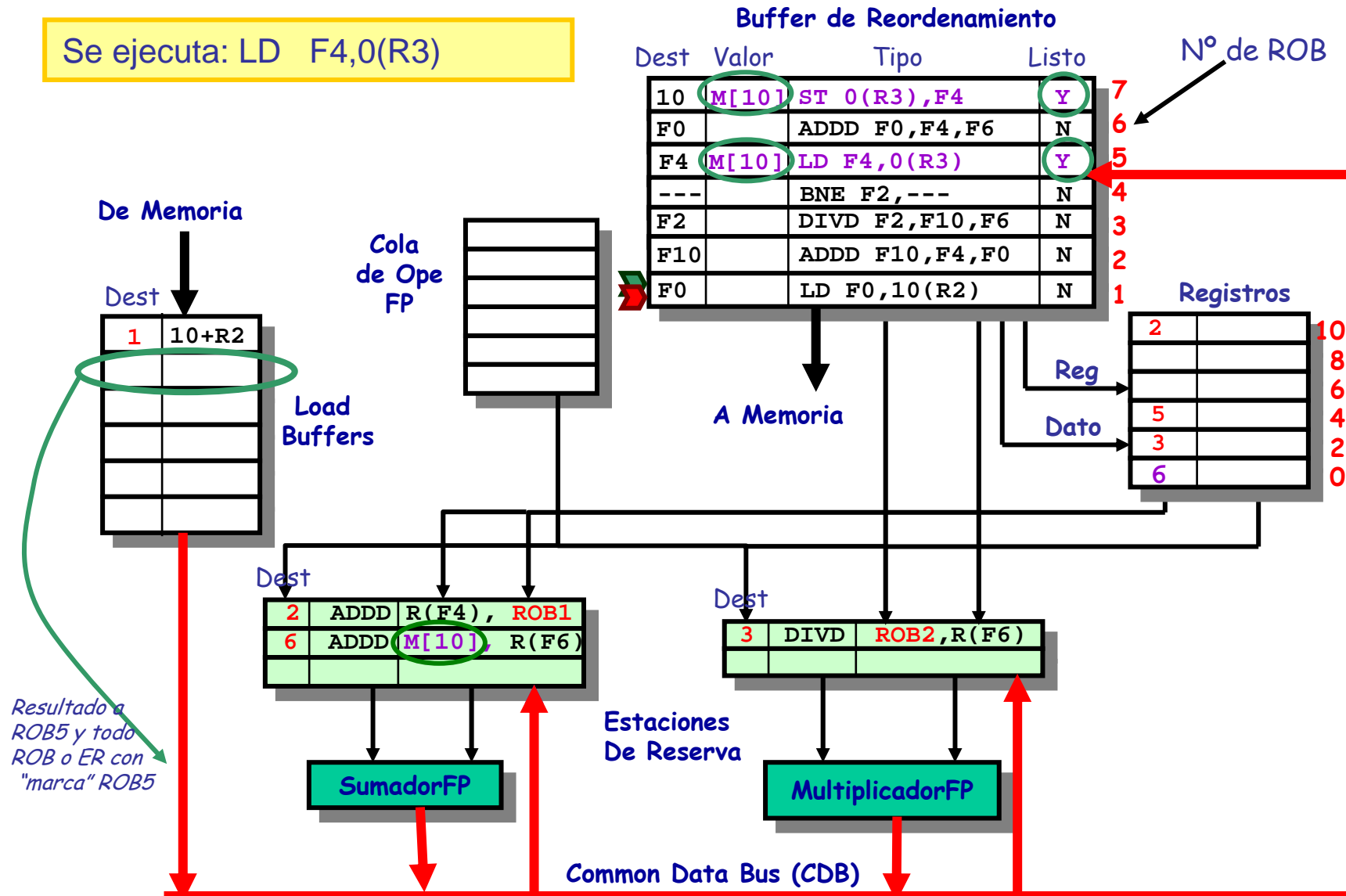
# Especulación: Ejemplo

Se lanza: ST 0(R3), F4  
(sup R3=10)



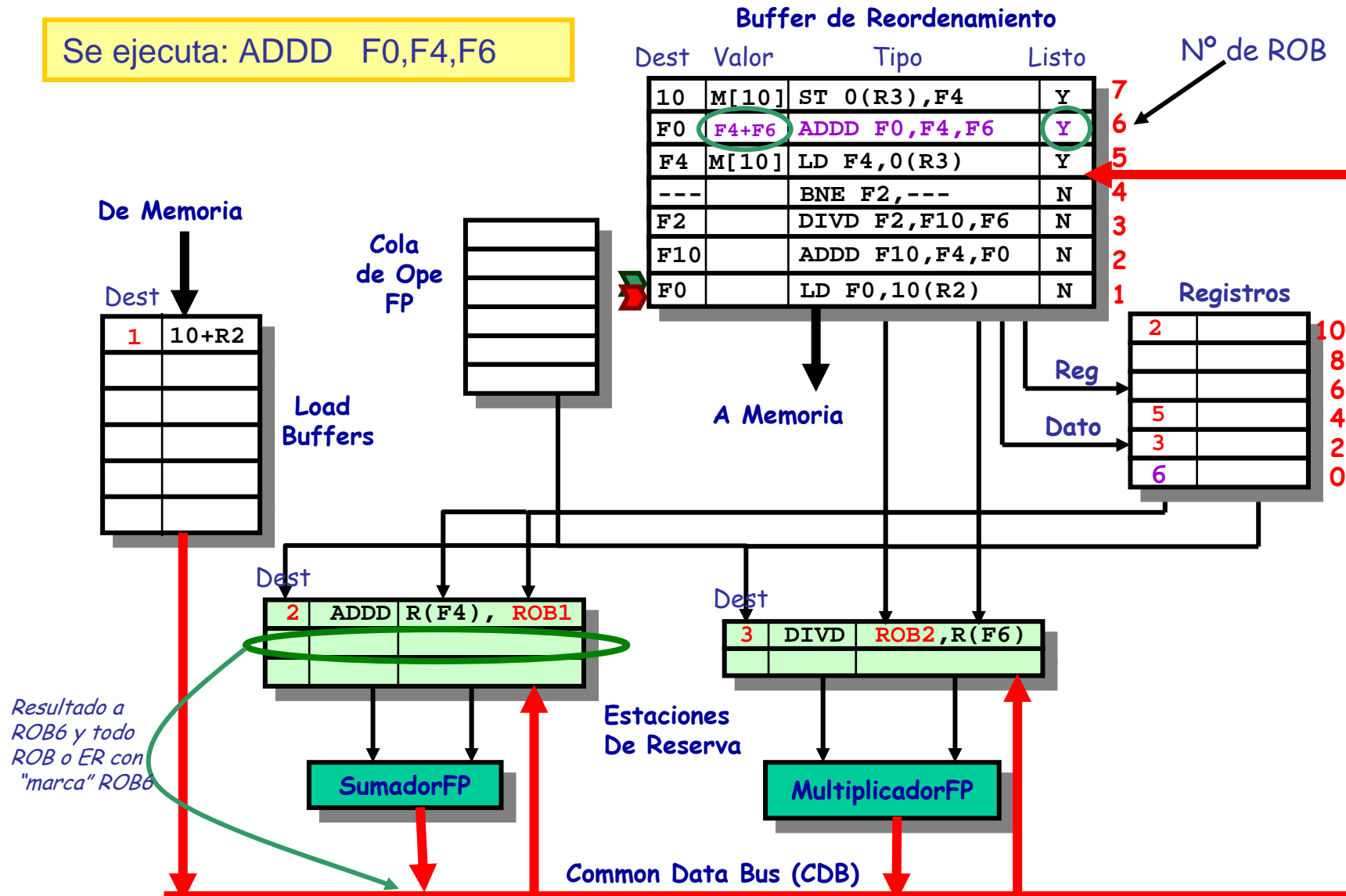
# Especulación: Ejemplo

Se ejecuta: LD F4,0(R3)



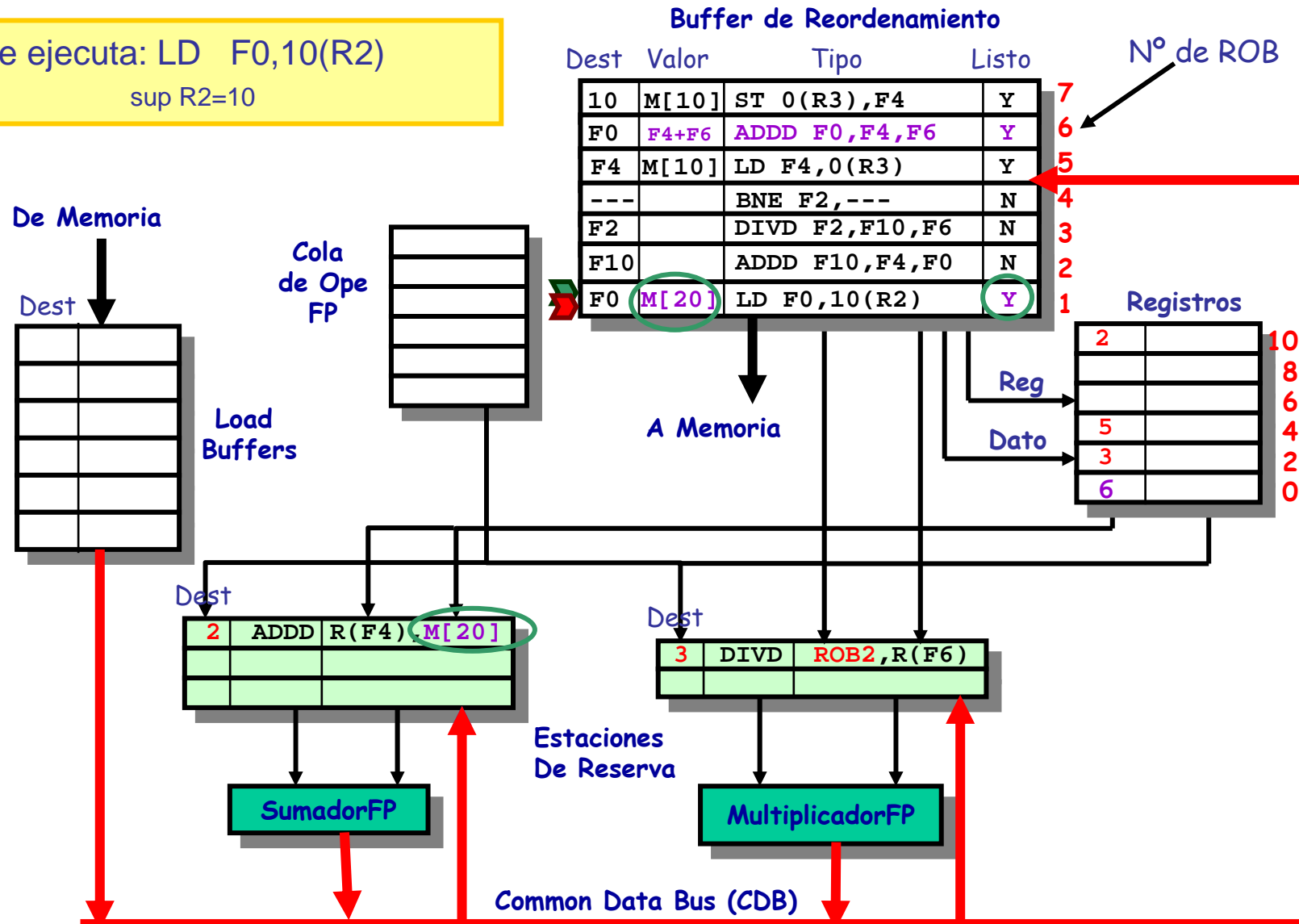
# Especulación: Ejemplo

Se ejecuta: ADDD F0,F4,F6



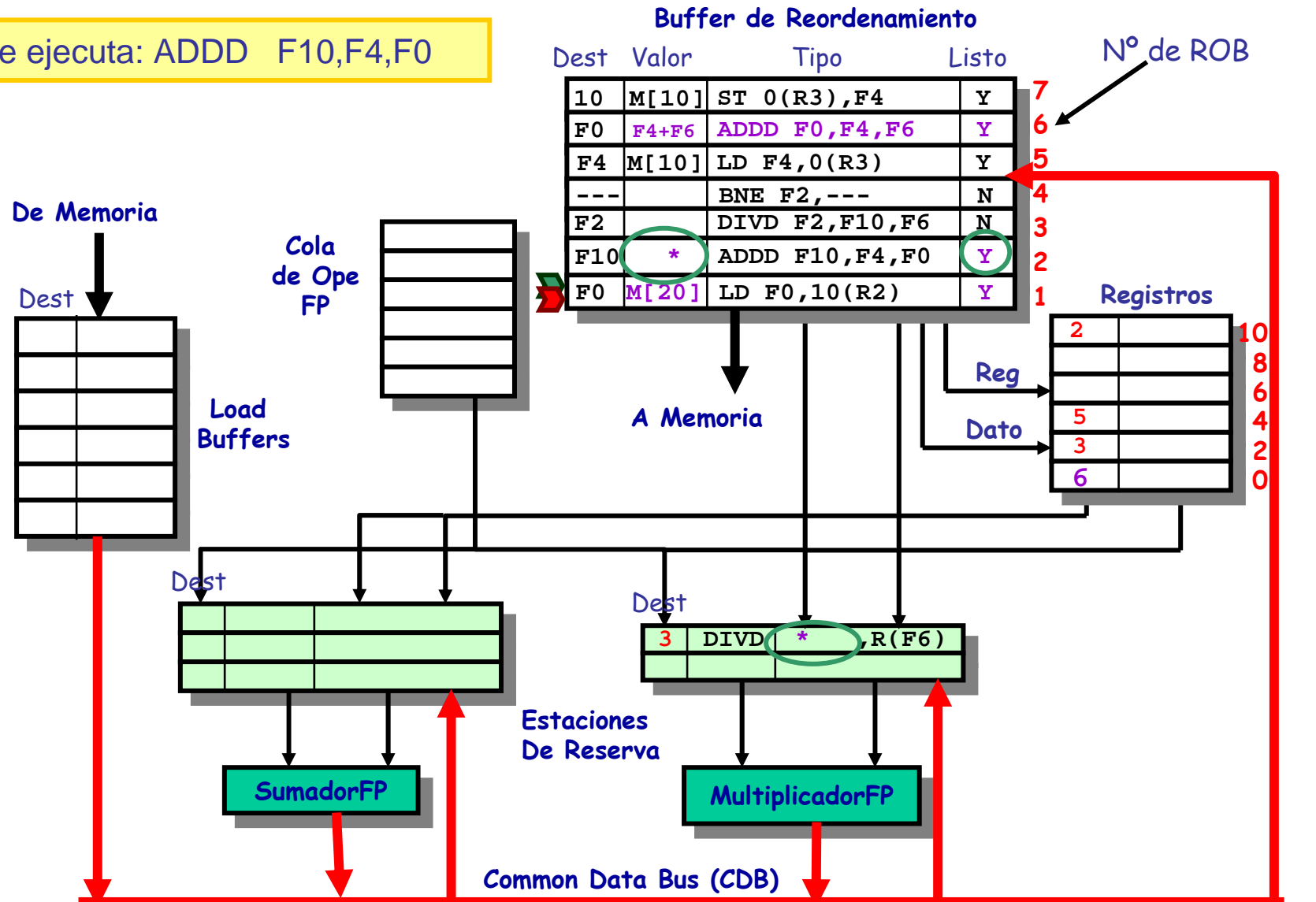
# Especulación: Ejemplo

Se ejecuta: LD F0,10(R2)  
sup R2=10



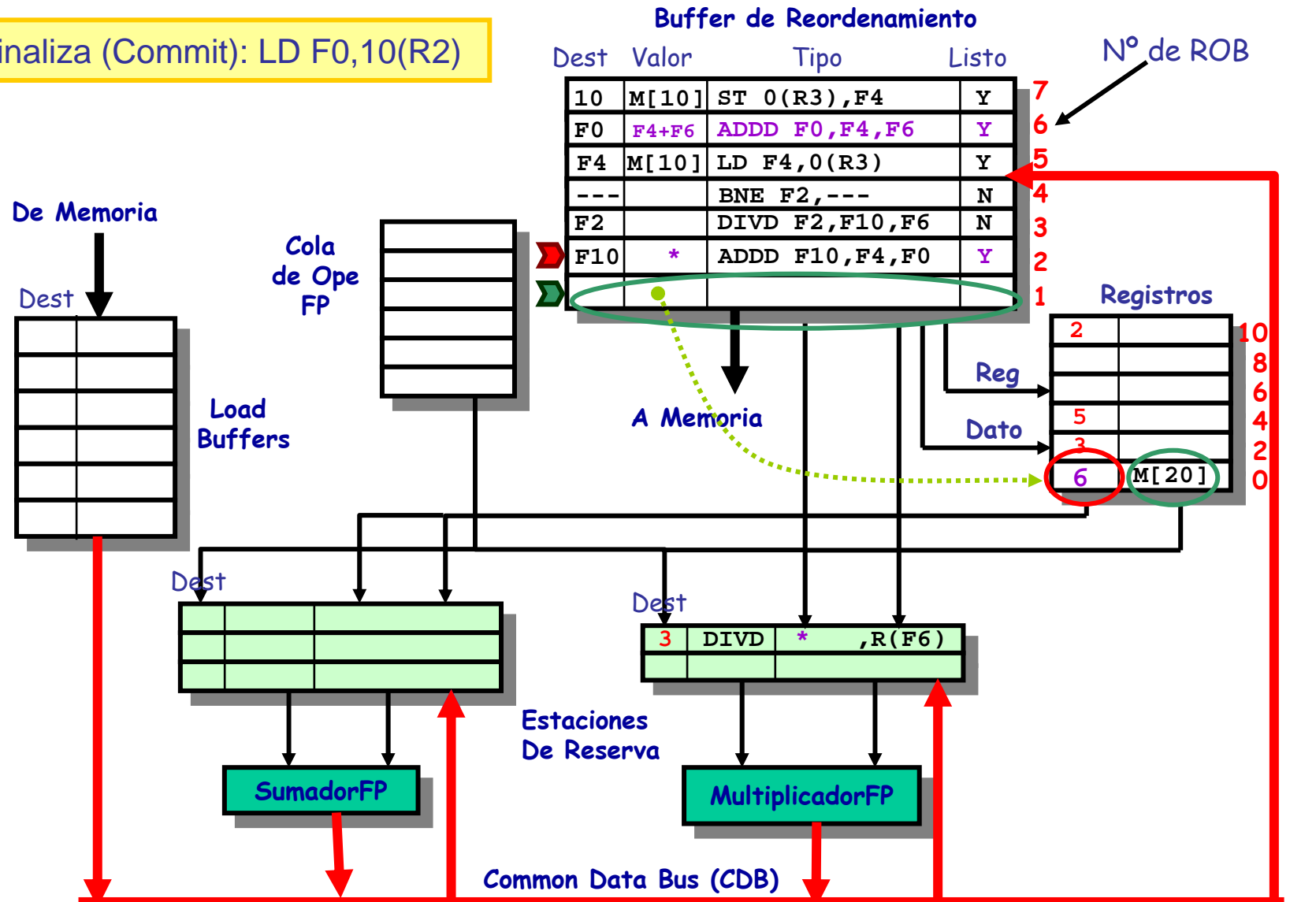
# Especulación: Ejemplo

Se ejecuta: ADDD F10,F4,F0



# Especulación: Ejemplo

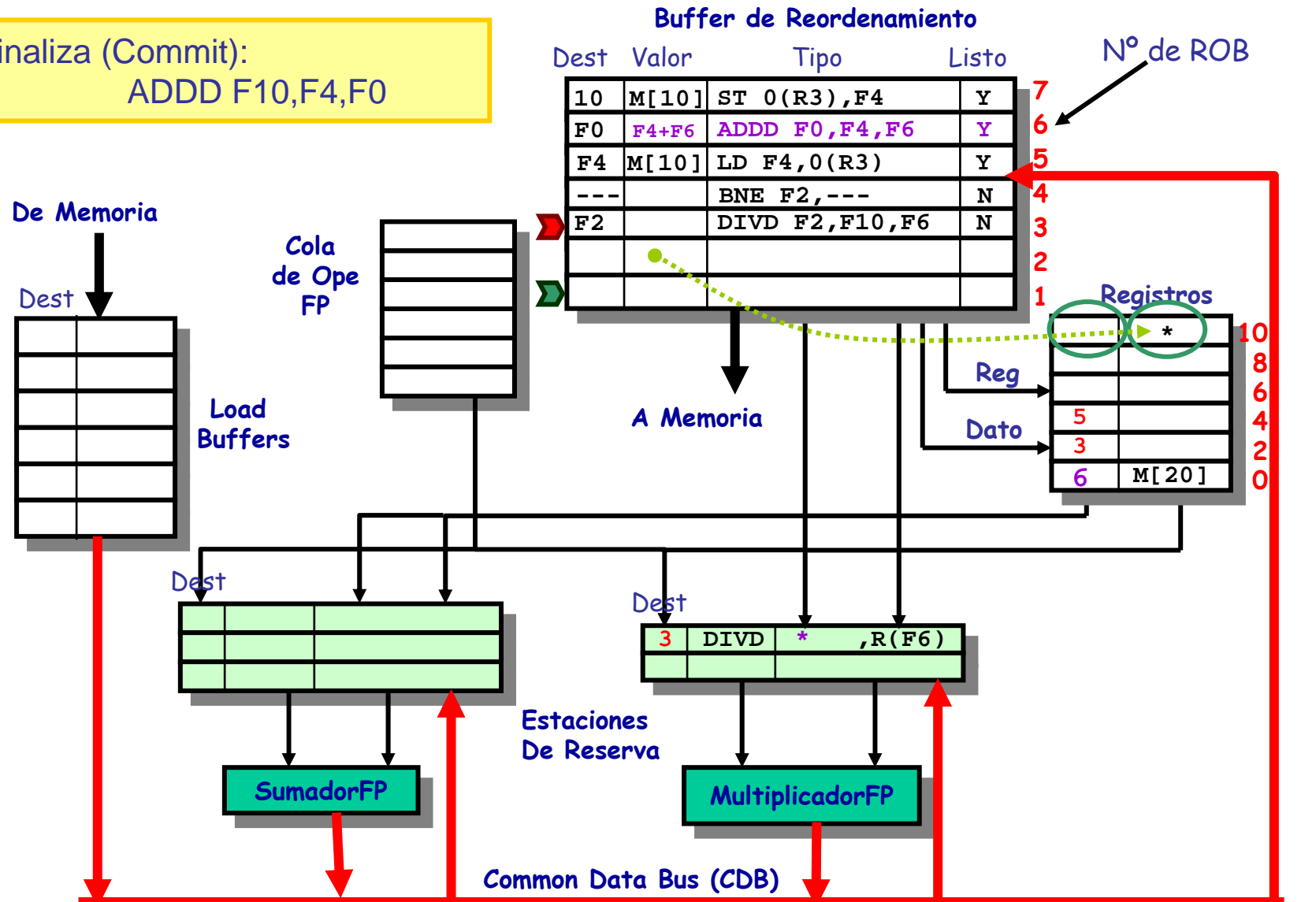
Finaliza (Commit): LD F0,10(R2)



\* = R(F4)+M[20]

# Especulación: Ejemplo

Finaliza (Commit):  
ADDD F10,F4,F0



\* = R(F4)+M[20]



## Especulación: Riesgos a través de memoria

---

- ❑ Riesgos EDE y EDL: no pueden aparecer dado que la actualización de memoria se hace en orden.
  - o Esperar hasta que la instrucción ST se halle en la cabecera de ROB => Todos los LD y ST anteriores se han completado.
  
- ❑ Riesgos LDE: Podrían producirse si un LD accede a la posición de memoria A, habiendo en el ROB un ST previo que almacena el resultado en A. Se evitan mediante el siguiente mecanismo:
  - o Un LD no ejecuta el acceso a memoria si hay un ST previo en el ROB con la misma dirección de memoria.
  - o Tampoco se ejecuta el LD si está pendiente el cálculo de la dirección efectiva de algún ST del ROB

## Especulación: Saltos e interrupciones

---

- ❑ El ROB permite recuperarse de saltos mal predichos e implementar un modelo de excepciones precisas
- ❑ Si una instrucción de salto bien predicha llega a cabecera de ROB =>
  - o Eliminarla de ROB
- ❑ Si una instrucción de salto mal predicha llega a cabecera de ROB =>
  - o Borrar contenido del ROB
  - o Borrar marcas (campo "Nº de ROB)" de todos los registros.
  - o Buscar instrucción correcta.
- ❑ Si una instrucción genera una interrupción =>
  - o Registrar la petición en el ROB
  - o Si la instrucción llega a la cabecera del ROB (no especulada), entonces reconocer la interrupción.
  - o Cualquier instrucción anterior habrá finalizado. Por tanto ninguna instrucción anterior puede provocar una excepción.

# Especulación: más ejemplos

## ❑ Algoritmo de TOMASULO con especulación

✓ Un ejemplo

LD F6,34(R2)  
 LD F2,45(R3)  
 MULTD F0,F2,F4  
 SUBD F8,F6,F2  
 DIVD F10,F0,F6  
 ADDD F6,F8,F2

### Estaciones de Reserva

	Estado	Opera	Vj	Vk	Qj	Qk	Nº ROB Destino
ADD1	Libre						
ADD2	Libre						
ADD3	Libre						
MUL1	Libre	MULT	M(45+(R3))	F4			#3
MUL2	Ocup.	DIV		M(34+(R2))	#3		#5

### Buffer de Reordenamiento

	Estado	Instrucción	Estado	Destino	Valor
1	libre	LD F6,34(R2)	Commit	F6	M(34+(R2))
2	libre	LD F2,45(R3)	Commit	F2	M(45+R3))
3	Ocupada	MULTD F0,F2,F4	Escri. Res	F0	#2x(F4)
4	Ocupada	SUB F8,F6,F2	Escri. Res.	F8	#1 - #2
5	Ocupada	DIVD F10,F0,F6	Ejecuta	F10	
6	Ocupada	ADDD F6,F8,F2	Escri. Res.	F6	#4 + #2

### Estado de los registros FP

	F0	F2	F4	F6	F8	F10	F12	...
Buffer Nº	3			6	4	5		
Ocupado	si	no	no	si	si	si	no	

# Especulación: más ejemplos

## ❑ Algoritmo de TOMASULO con especulación

### ✓ Un ejemplo

	Estado	Opera	Vj	Vk	Qj	Qk	Nº ROB Destino
MUL1	Libre	MULT	M(0+(R1))	F2			#2
MUL2	Libre	MULT	M(0+(R1))	F2			#7

LOOP LD F0,0(R1)  
 MULTD F4,F0,F2  
 SD 0(R1),F4  
 SUBI R1,R1,#8  
 BNEZ R1,LOOP

	Estado	Instrucción	Estado	Destino	Valor
1	libre	LD F0,0(R1)	Commit	F0	M(0+(R1))
2	libre	MULTD F4,F0,F2	Commit	F4	F0 x F2
3	Ocupada	SD 0(R1),F4	Escri. Res	0+R1	#2
4	Ocupada	SUBI R1,R1,#8	Escri. Res.	R1	R1-8
5	Ocupada	BNEZ R1, loop	Escri. Res		

ESPECULADAS

6	Ocupada	LD F0,0(R1)	Escri. Res.	F0	M(#4)
7	Ocupada	MULTD F4,F0,F2	Escri. Res	F4	#6xF2
8	Ocupada	SD 0(R1),F4	Escri. Res	0+R1	#7
9	Ocupada	SUBI R1,R1,#8	Escri. Res	R1	#4 - #8
10	Ocupada	BNEZ R1,Loop	Escri. Res.		

Estado de los registros FP

	F0	F2	F4	F6	F8	F10	F12	...
Buffer Nº	6		7					
Ocupado	si	no	si	no	no	no	no	