

De la Lógica Digital al diseño de un procesador

Javier Resano, Jose Luis Briz

GAZ: Grupo de Arquitectura de Computadores,

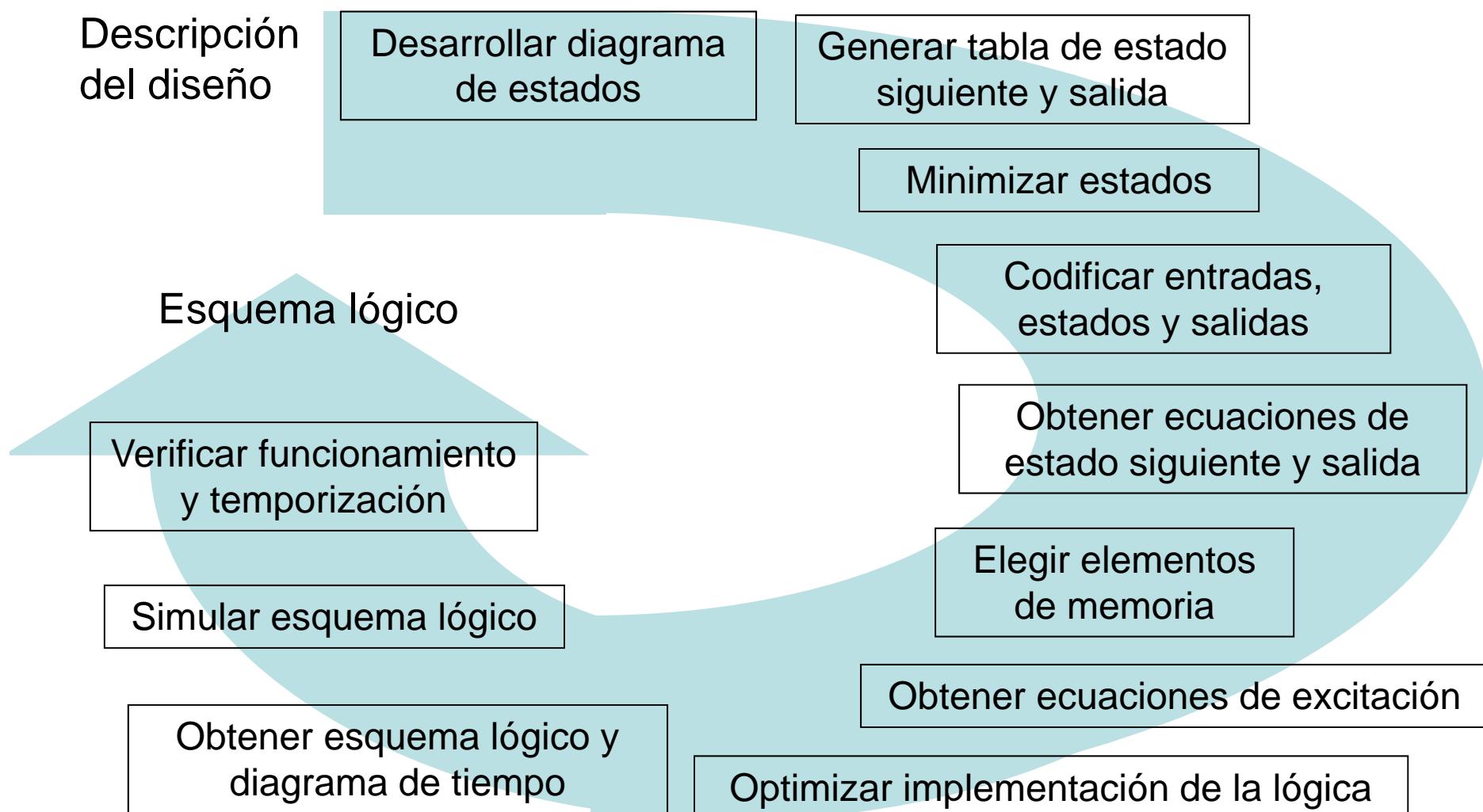


Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

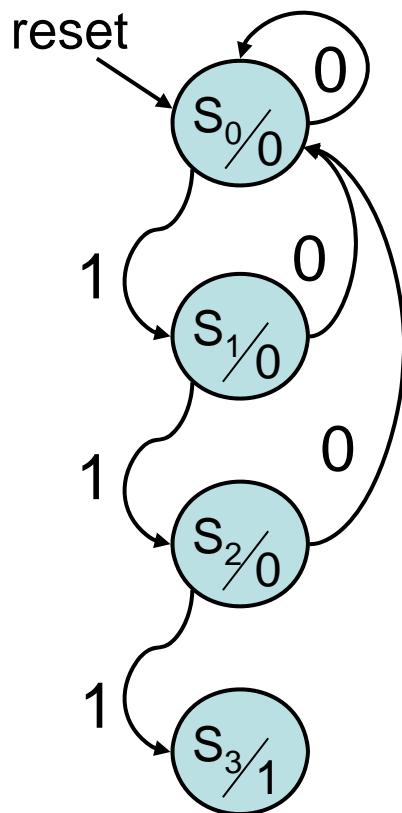
Diseño de un procesador específico

- 1. Cómo hacer un diseño digital complejo**
 1. Diseño jerárquico y reutilización
 2. Diseño algorítmico
2. Análisis temporal de un diseño

Motivación: la implementación canónica no es siempre es adecuada



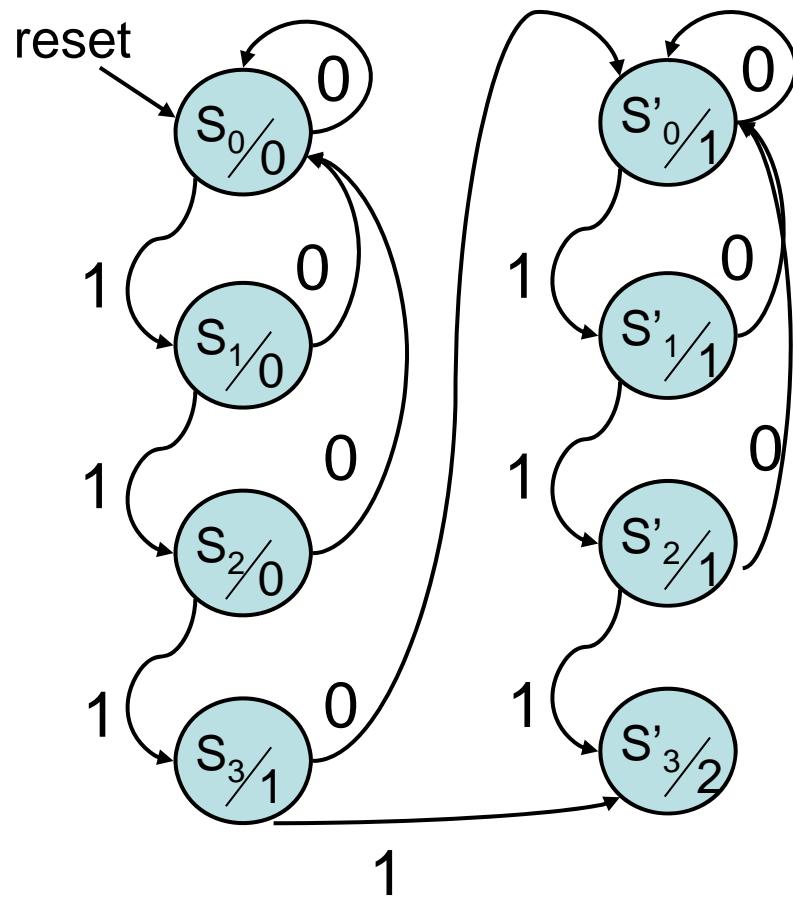
Motivación: contador de la secuencia "111" con solapamiento



Debe ser capaz de contar hasta 255 secuencias

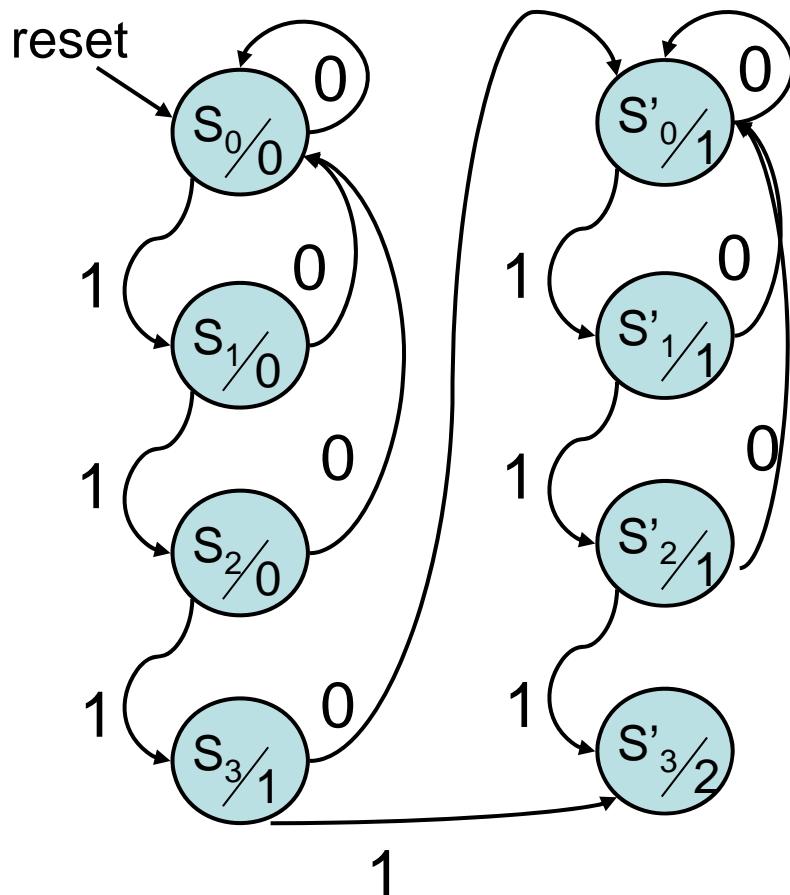
estados que identifican la primera secuencia

Motivación: contador de la secuencia "111" con solapamiento



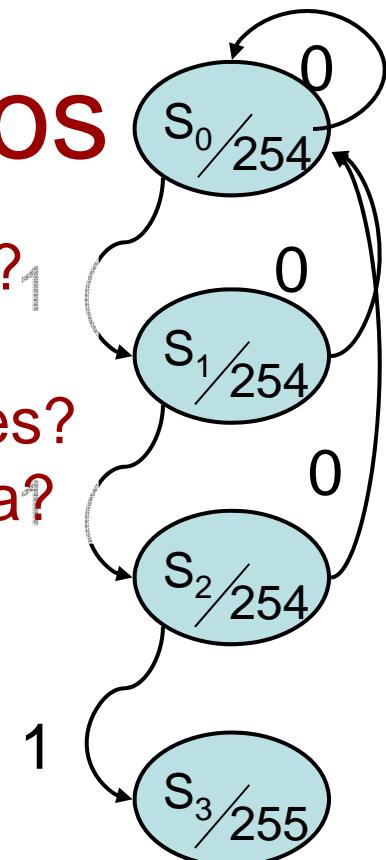
primera y segunda secuencia

Motivación: contador de la secuencia "111" con solapamiento



1020 estados

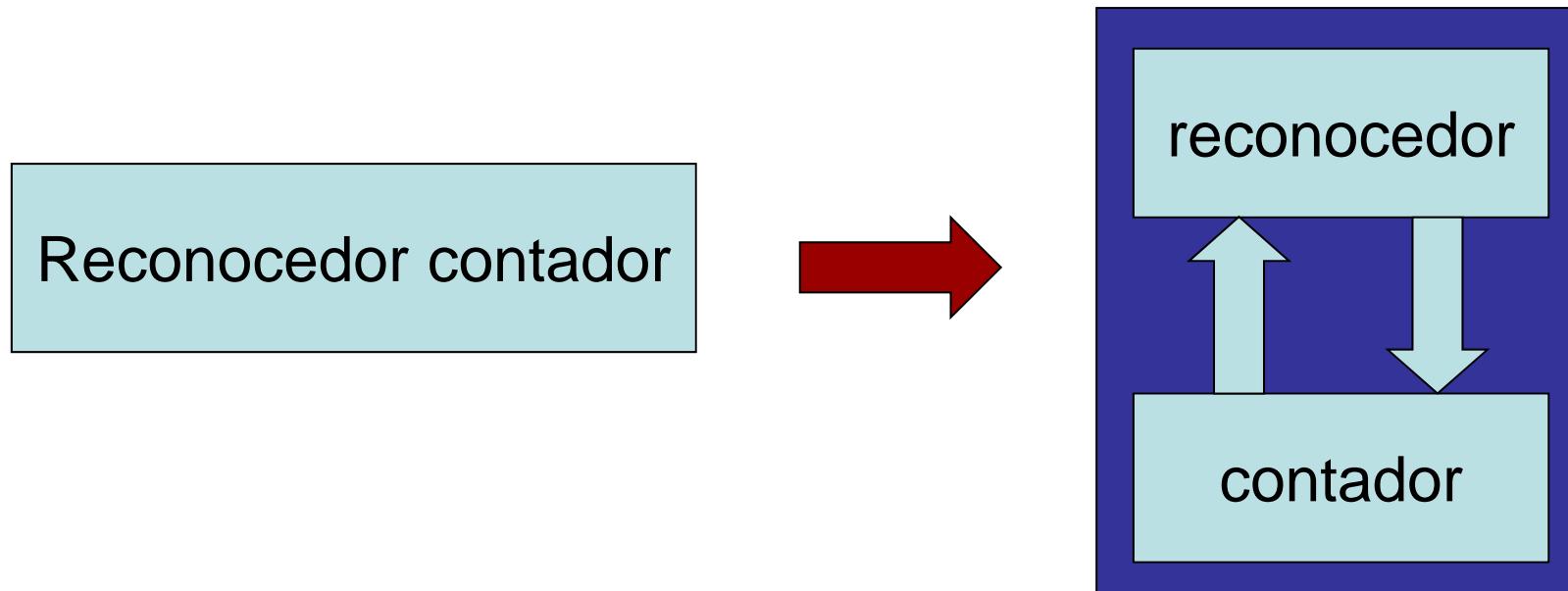
¿Minimizar estados?
¿codificar?
¿Elegir los biestables?
¿Simplificar la lógica?



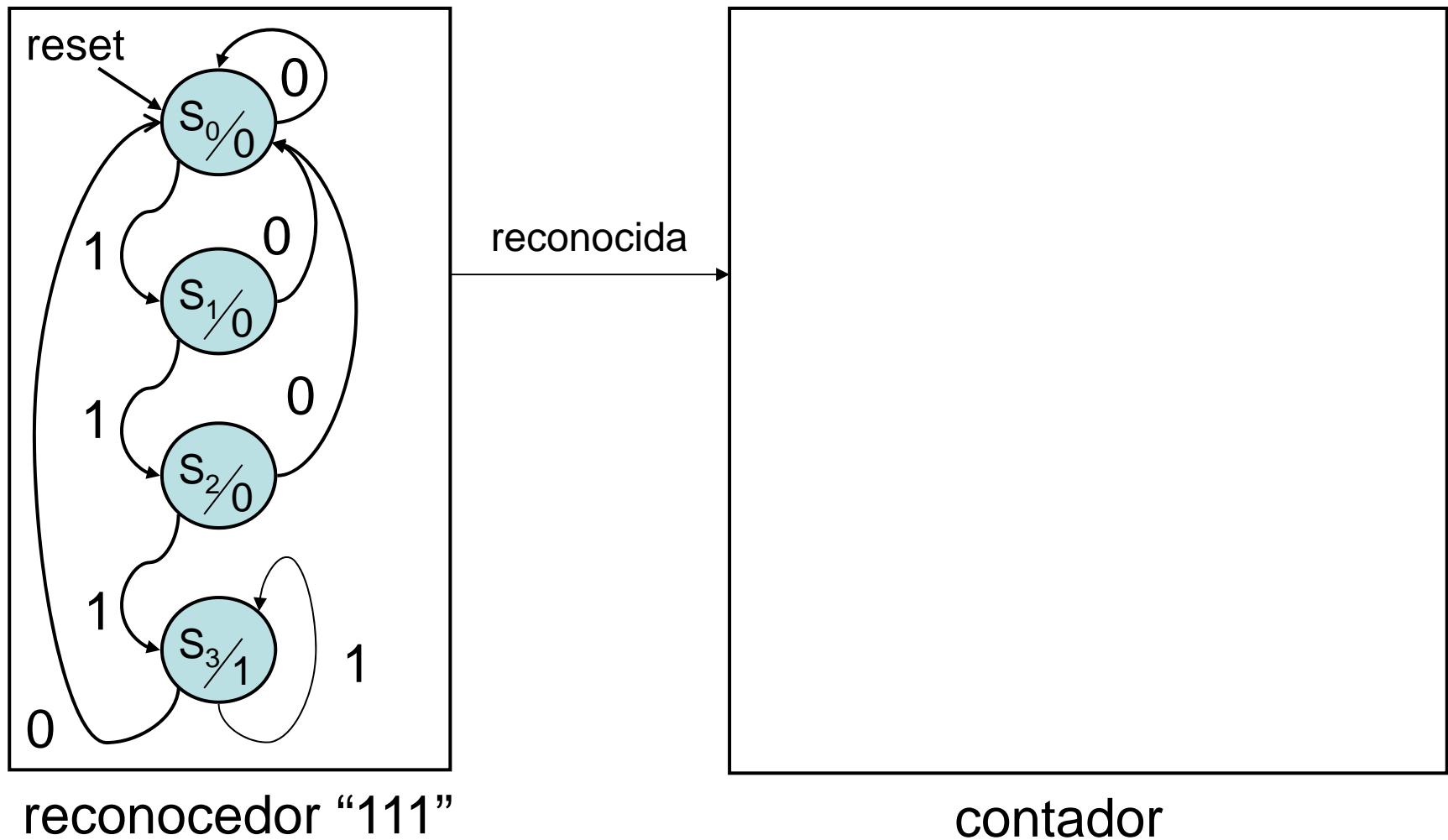
secuencia 255

Motivación: contador de la secuencia "111" con solapamiento

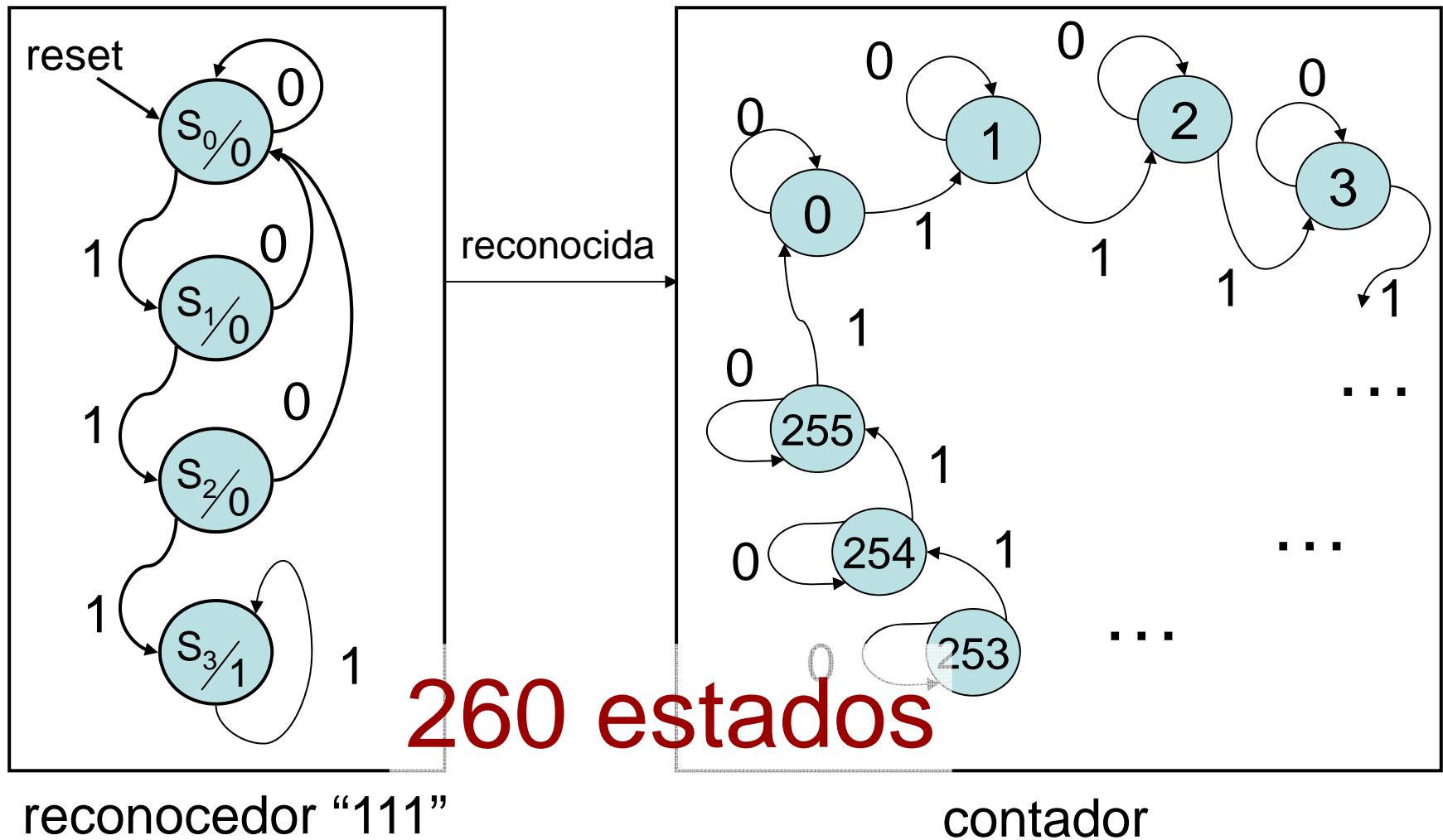
¿Y si dividimos el problema en dos sub-problemas más sencillos?



Motivación: contador de la secuencia "111" con solapamiento

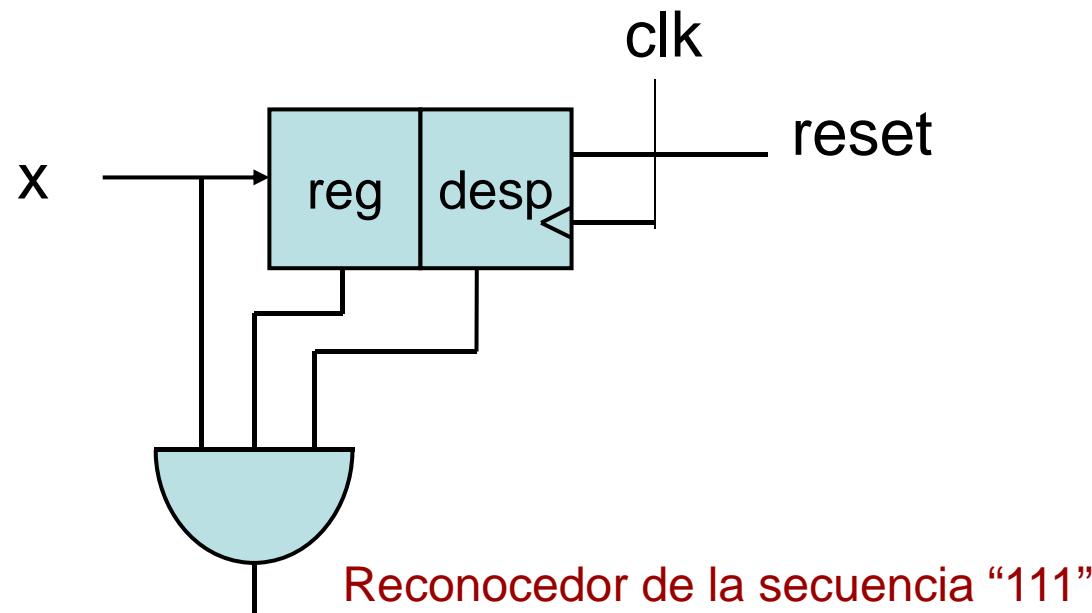


Motivación: contador de la secuencia "111" con solapamiento



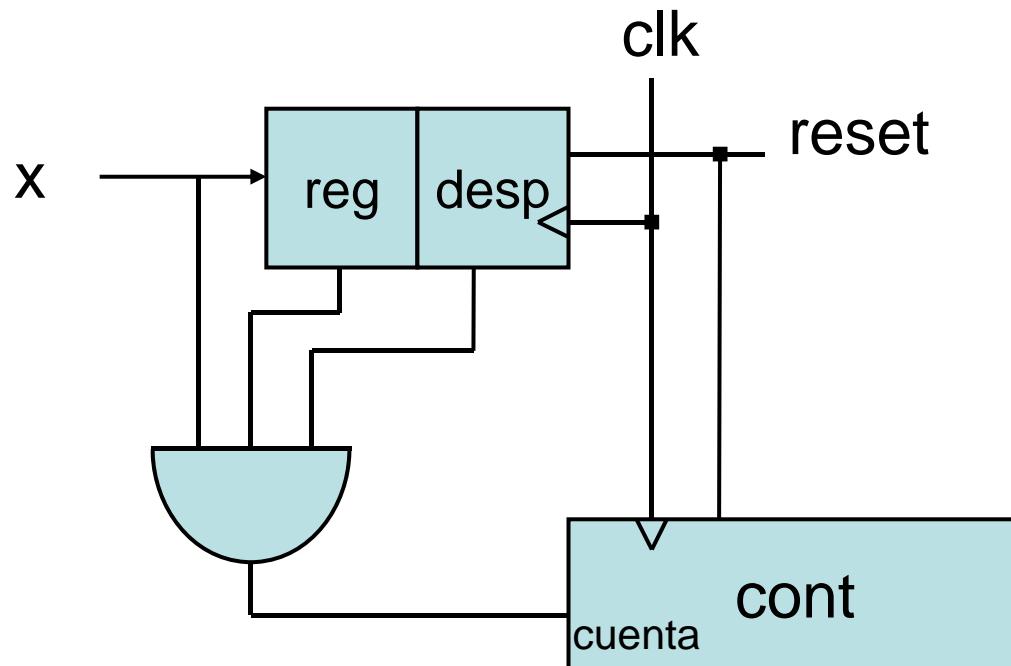
Motivación: contador de la secuencia "111" con solapamiento

¿Y si utilizamos módulos prediseñados para simplificar el proceso de diseño?



Motivación: contador de la secuencia "111" con solapamiento

¿y si utilizamos módulos prediseñados para simplificar el proceso de diseño?



Diseño jerárquico y reutilización

- Realizar diseño complejo
 - dividirlo en sub-problemas más sencillos que seamos capaces de resolver
 - **particionamiento**
- Para simplificar todavía más el diseño en lugar de comenzar el diseño desde cero podemos **reutilizar componentes estándar** :
 - registros
 - contadores
 - memorias RAM
 - ALUs
 - Muxes, Decodificadores, Codificadores...

¿Podemos enfrentarnos al diseño directamente?

- Sí, somos capaces de realizar el diseño
 - Comenzamos a trabajar
- No, el diseño es demasiado complejo
 - Fase de división o particionamiento:
 - Dividimos en subproblemas más sencillos
 - Aplicamos nuestras técnicas a cada subproblema por separado
 - Realizamos el diseño final utilizando los subproblemas
 - Esta fase **simplifica** el problema pero también **limita las posibilidades de optimización**:
 - Las decisiones tomadas por separado para cada subproblema pueden comprometer el agregado final

Reutilización: clave para un diseño rápido y fiable

- Hay módulos que se utilizan muy frecuentemente y cuya implementación:
 - es conocida
 - está optimizada y verificada
 - se incluye en las herramientas de ayuda al diseñador
- Comenzar desde cero no es una estrategia óptima
 - ¿por qué hacer lo que ya está hecho?
- Reutilizar componentes de una librería
 - permite acelerar el proceso de diseño
 - muchos componentes son comunes a todas las librerías
- El diseñador debe conocer estos módulos y saber utilizarlos

Diseño algorítmico: motivación

- Frecuentemente no es práctico representar un sistema utilizando tan sólo tablas y diagramas de estado
- Ejemplo:
 - Entrada: un número de 8 bits.
 - Salida: número de veces que se ha introducido esa misma entrada (módulo 16).
 - Necesitamos un vector de estado de $2^8 * 2^4$ (12 bits)
 - Tabla de transiciones de estado: 32 bits por entrada, 2^{20} filas
- Para describir diseños complejos necesitamos aumentar el nivel de abstracción de la especificación: **utilizamos algoritmos**

Motivación

- ¿Y si representamos el sistema usando un algoritmo?
- Ejemplo:
 - Entrada: un número de 8 bits.
 - Salida: número de veces que se ha introducido esa misma entrada (módulo 16).

Algoritmo:

- leer la entrada
- comprobar cuántas veces se ha introducido previamente la misma entrada
- sumar 1
- actualizar el estado y generar la salida

¿y cómo hacemos el diseño a partir del algoritmo?

Algoritmo:

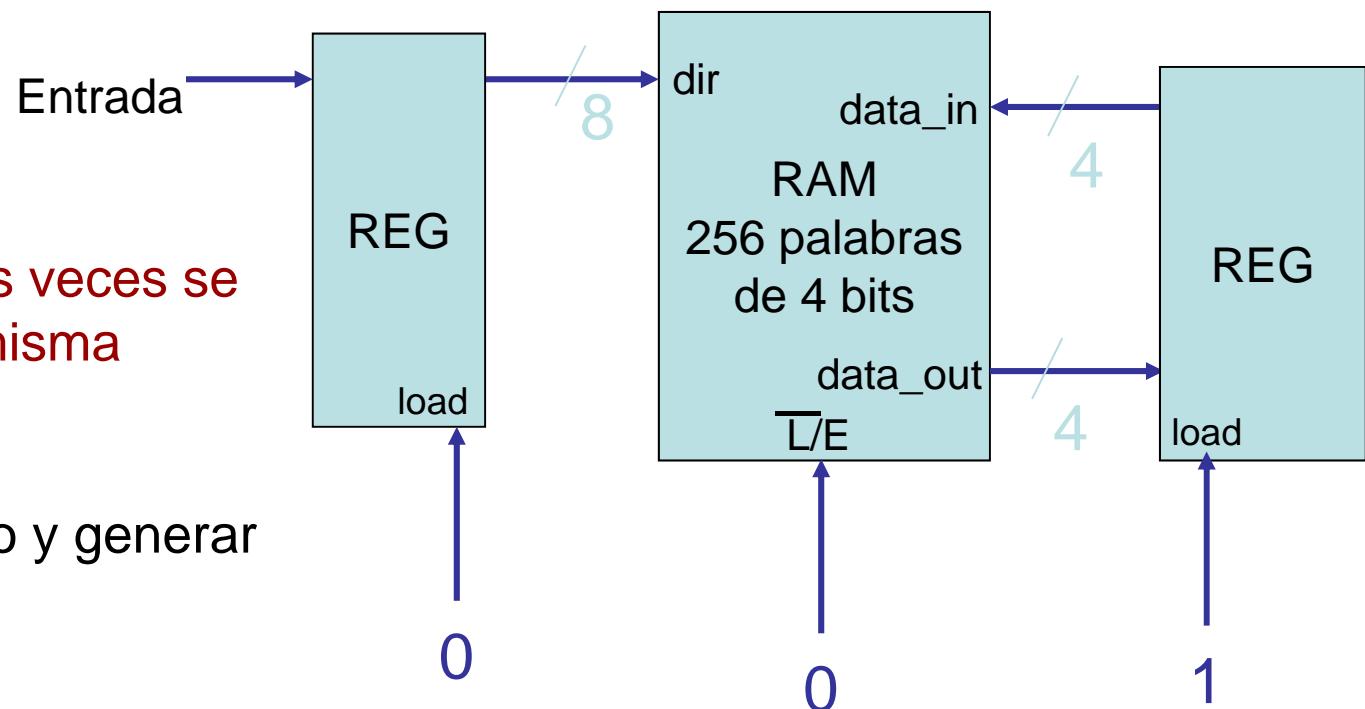
- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida



¿y cómo hacemos el diseño a partir del algoritmo?

Algoritmo:

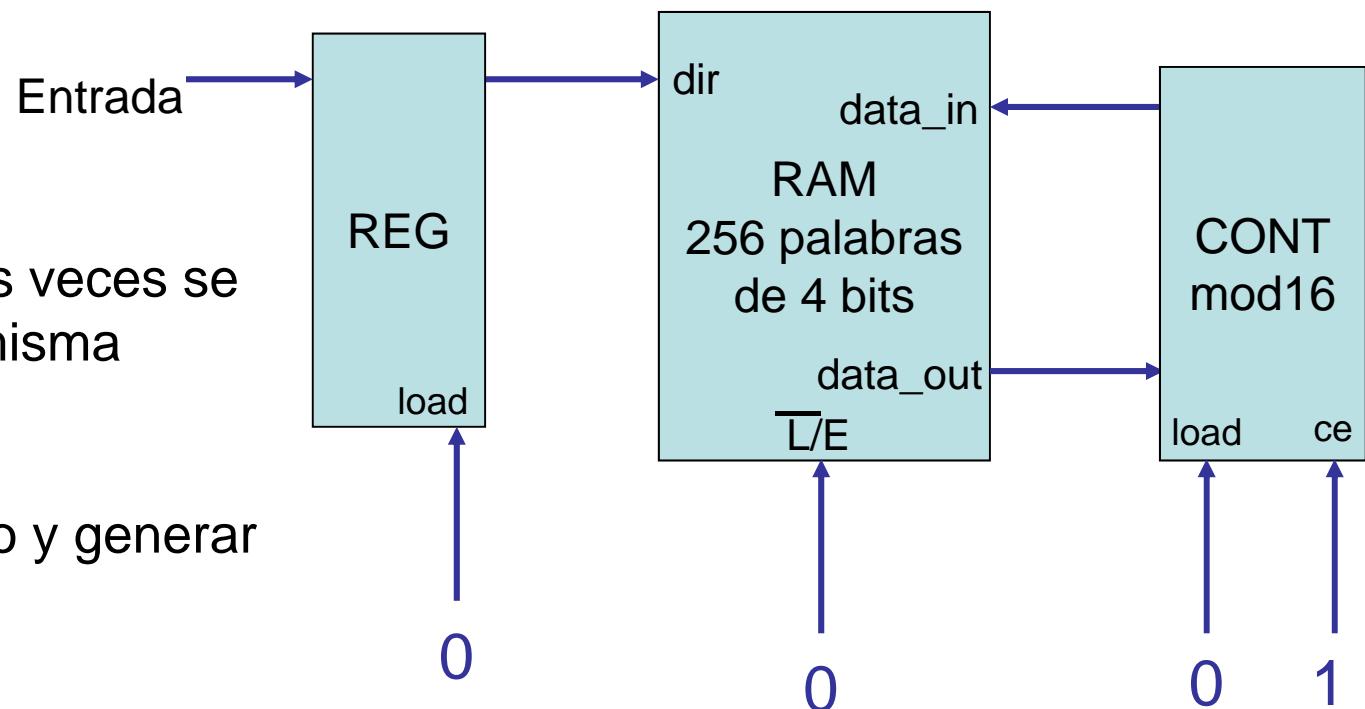
- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida



¿y cómo hacemos el diseño a partir del algoritmo?

Algoritmo:

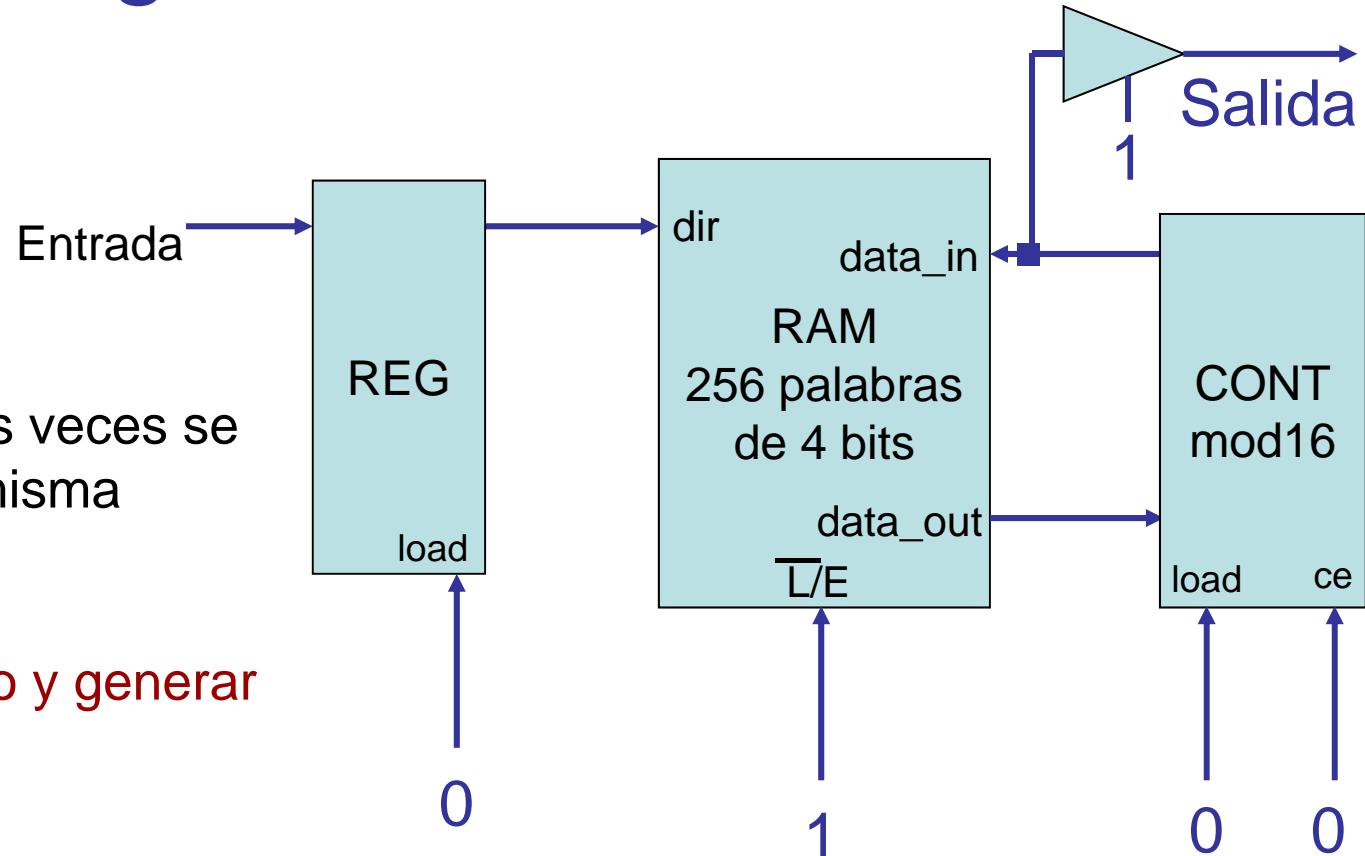
- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- **sumar 1**
- actualizar el estado y generar la salida



¿y cómo hacemos el diseño a partir del algoritmo?

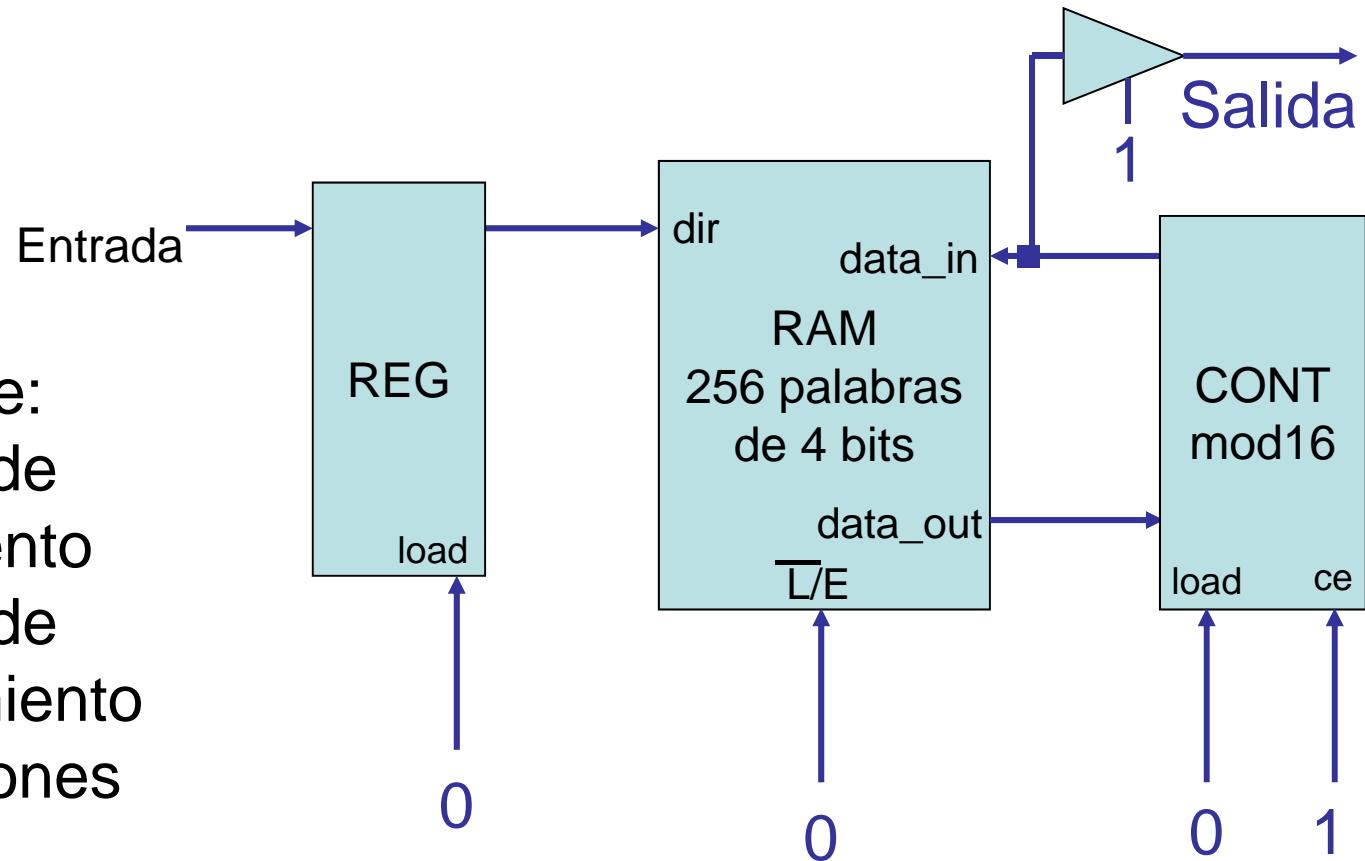
Algoritmo:

- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida



El camino de datos es el módulo que procesa los datos de entrada

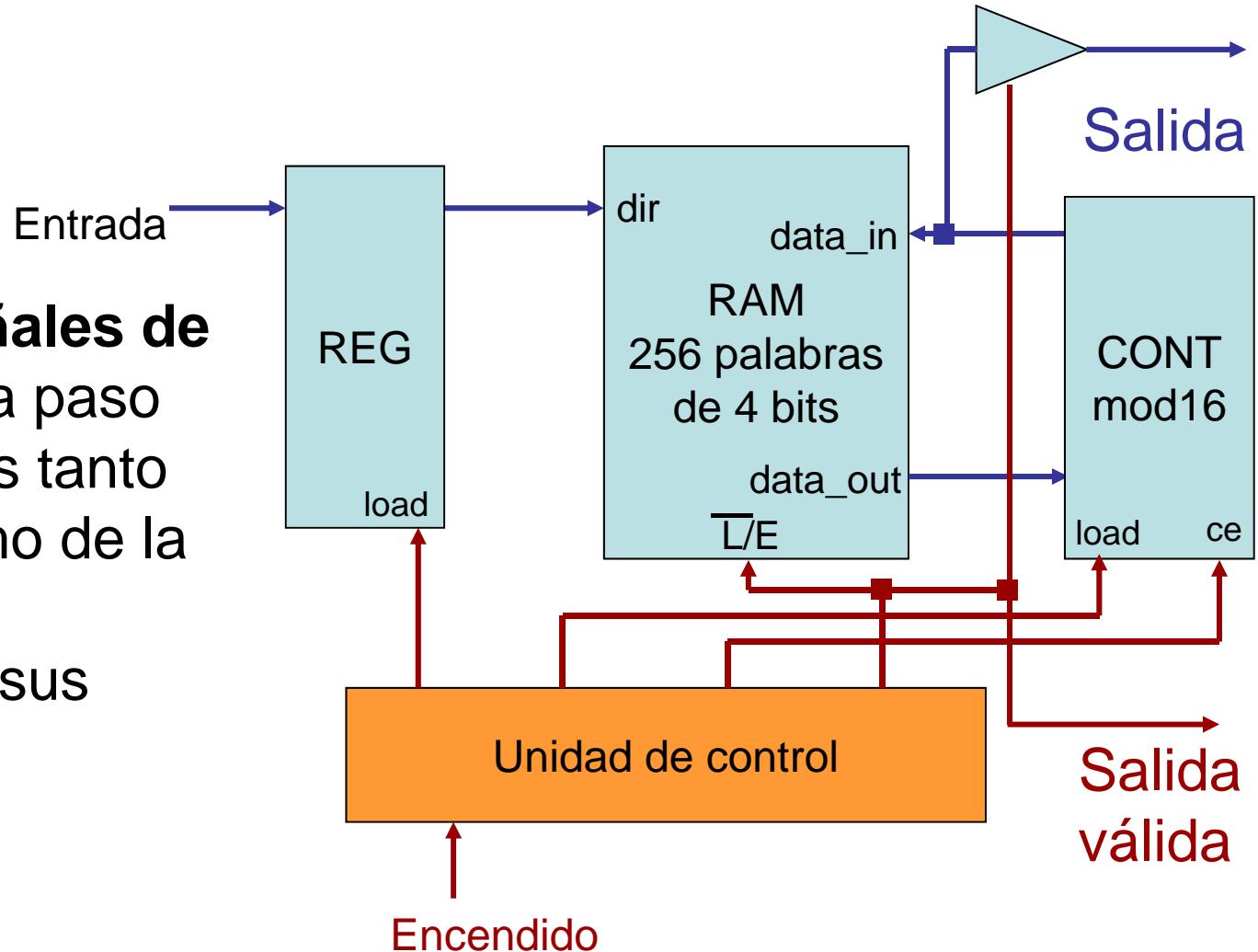
- Se compone de:
 - elementos de procesamiento
 - elementos de almacenamiento
 - interconexiones



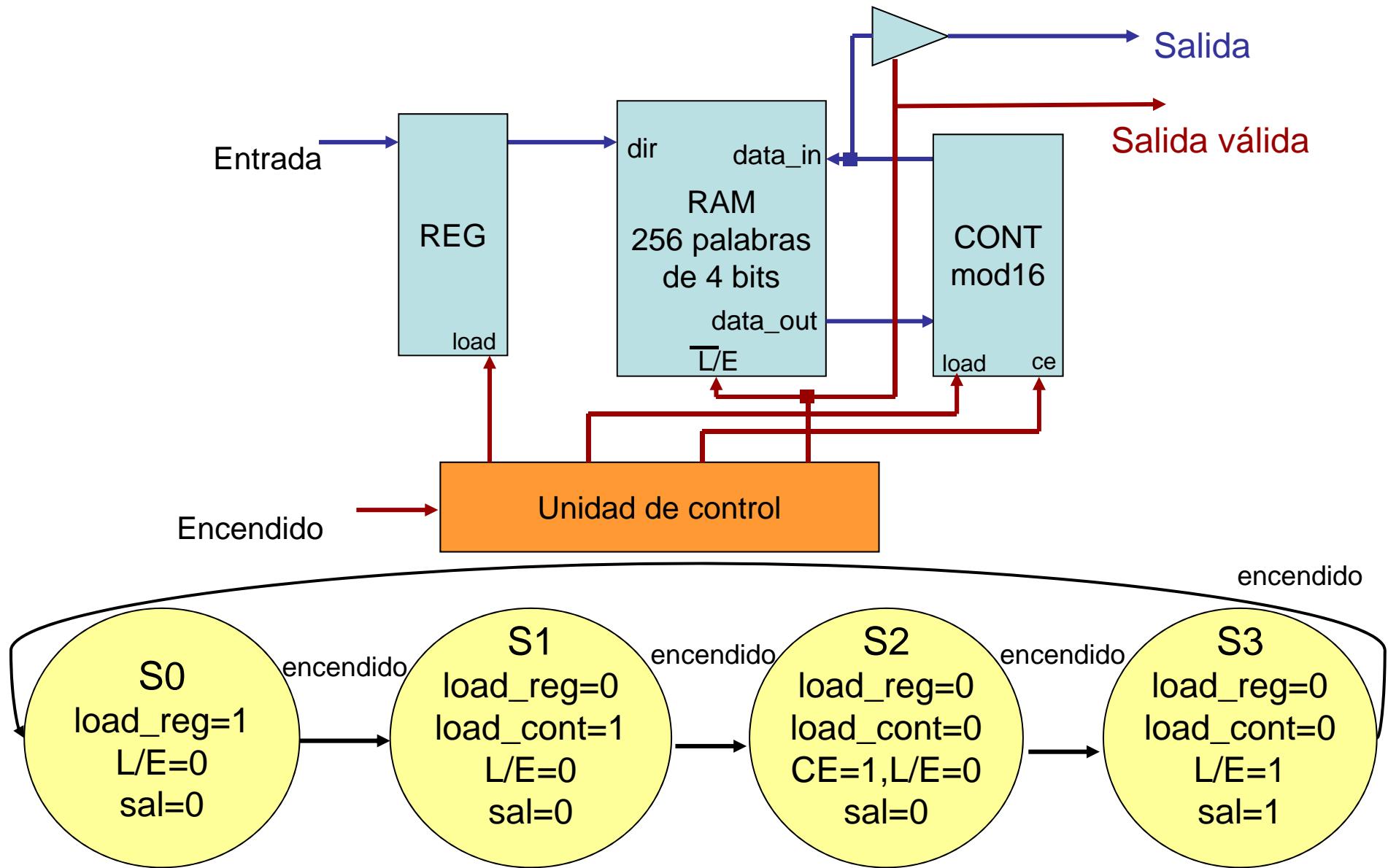
Pero ¿quién asigna el valor correcto a las señales que controlan el camino de datos?

La unidad de control dirige a la ruta de datos

- **Genera las señales de control** de cada paso
- Recibe entradas tanto del exterior como de la ruta de datos
- Puede generar sus propias salidas



Diseño de la unidad de control



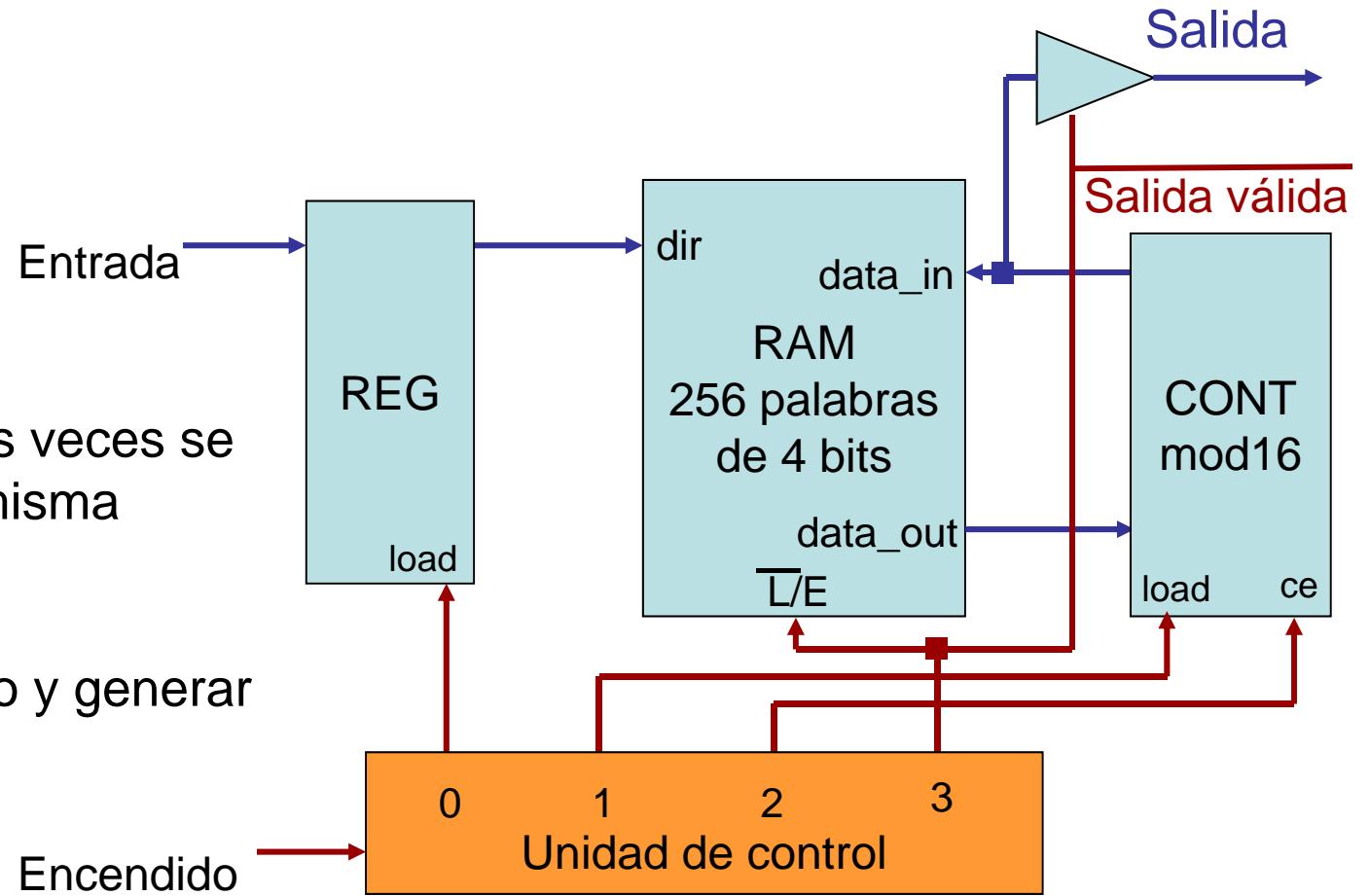
Diseño final

Si encendido =1

- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida

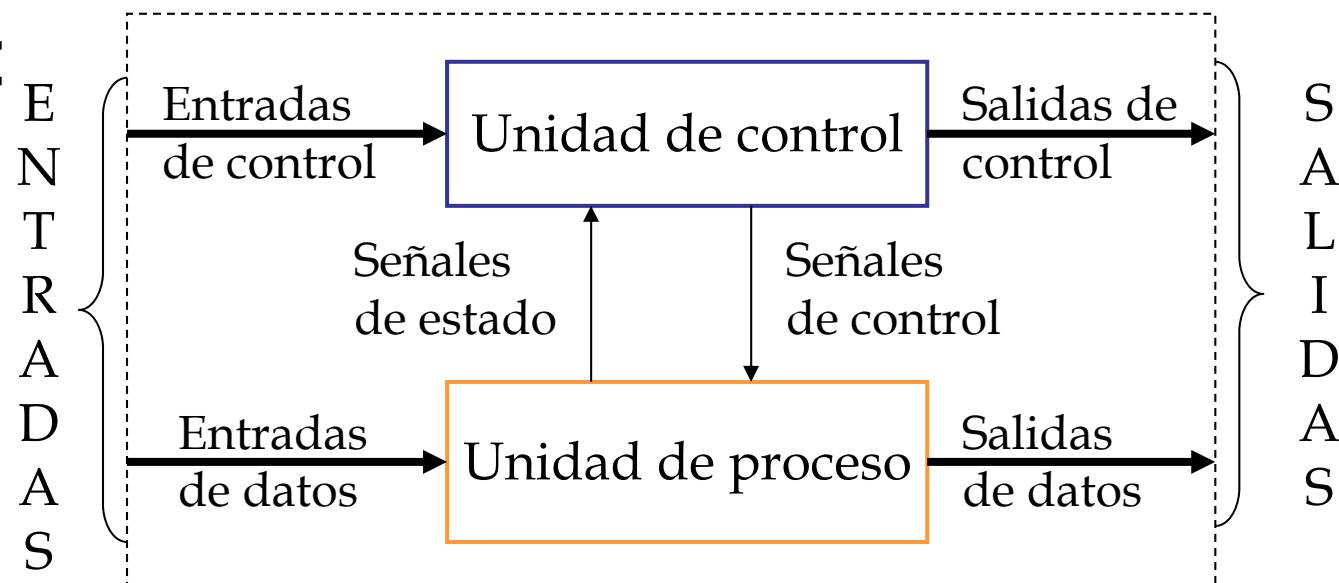
Si encendido =0

- esperar



¿Qué son los sistemas algorítmicos?

- Sistemas secuenciales síncronos
- Comportamiento definido IMPLÍCITAMENTE
 - No se especifica el valor de Z sino el modo de calcularlo: el algoritmo.
- **Modelo:**



Proceso de diseño

1. Definición del sistema

- Especificar: entradas y salidas, función, bloques disponibles

2. Diseño del algoritmo

- Obtener conjunto ordenado y finito de operaciones.
- Modelo de máquina de estados generalizada.

3. Definición de la arquitectura del sistema

- Determinar entradas, salidas y señales internas

4. Diseño de la unidad de proceso o ruta de datos

5. Diseño de la unidad de control

6. Ensamblado (de U.P y U.C.) y verificación

Conclusiones

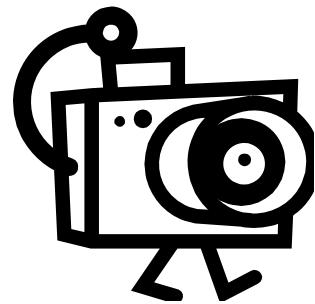
- El flujo de diseño lógico canónico no es adecuado para diseños complejos
- El diseño algorítmico nos permite enfrentarnos a estos problemas **subiendo el nivel de abstracción**
- **dividiremos** los diseños en:
 - el **camino de datos** que se ocupa de procesar los datos
 - la **unidad de control** que genera las señales necesarias para que cada etapa del algoritmo se ejecute correctamente
- Para terminar **ensamblamos** ambas partes y **verificamos** el funcionamiento

Diseño de un procesador específico

1. Alternativas a la implementación canónica
- 2. Análisis temporal de un diseño**
 1. Retrasos de propagación de puertas lógicas
 2. Estudio temporal de Sistemas Combinatorios
 3. Estudio Temporal de los Sistemas Secuenciales.
Tiempo de Ciclo
 4. Modelo temporal de las memorias

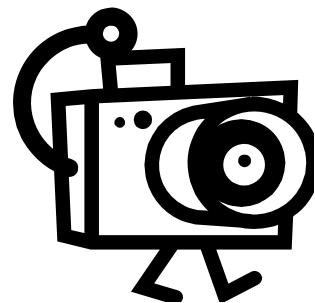
¿Por qué es importante el análisis temporal?

Suponemos que queremos hacer una foto a un ciclista saltando, queremos que salga en el aire



¿Por qué es importante el análisis temporal?

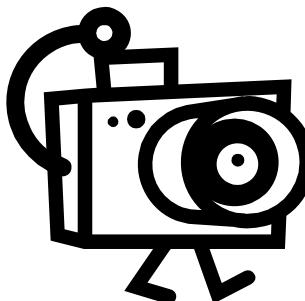
Si hacemos la foto muy pronto....



¿Por qué es importante el análisis temporal?

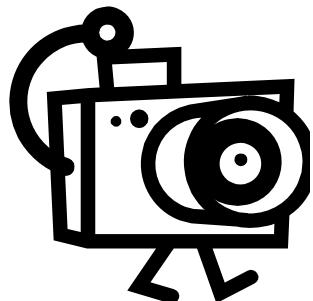
Si hacemos la foto muy tarde...

El ciclista estaba en el aire al disparar



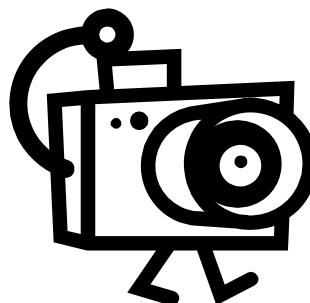
¿Por qué es importante el análisis temporal?

Si la hacemos “muy rápido”



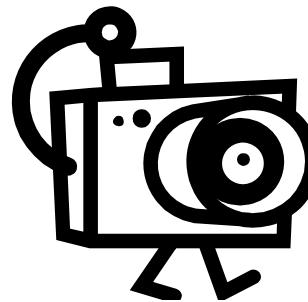
¿Por qué es importante el análisis temporal?

O muy lento...



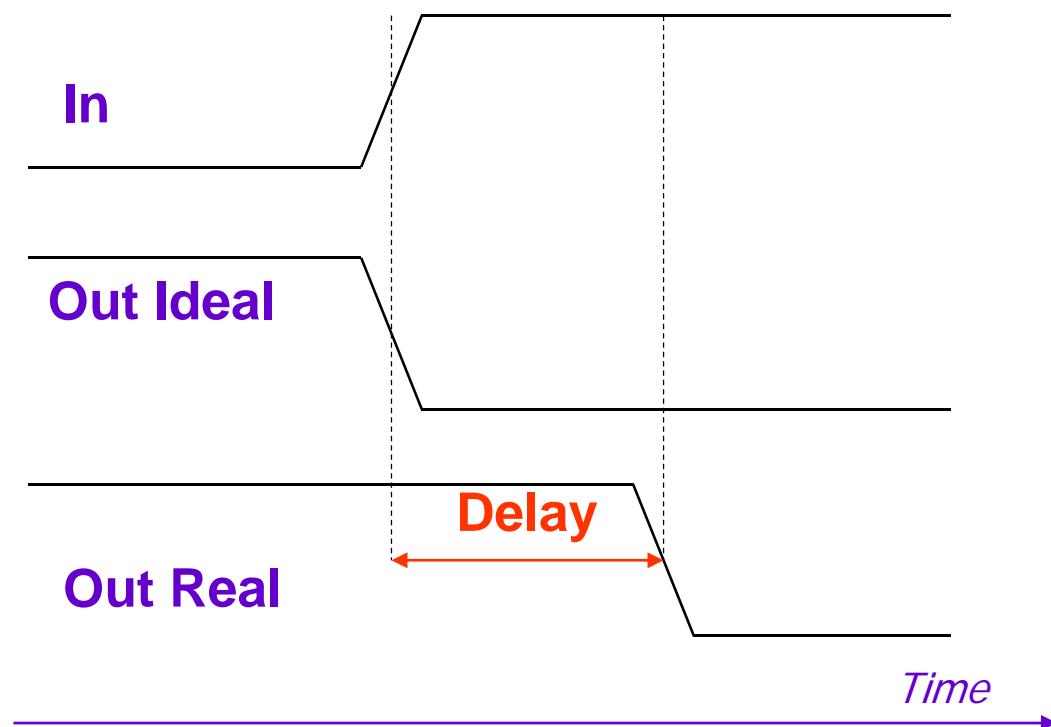
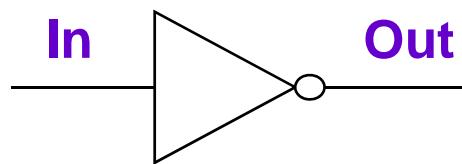
Hay que hacer las fotos en el momento preciso

- Hay que esperar a que se inicie el salto
- Se tiene que disparar cuando todavía esté un cierto tiempo en el aire
- La “velocidad” del disparo no puede ser muy pequeña



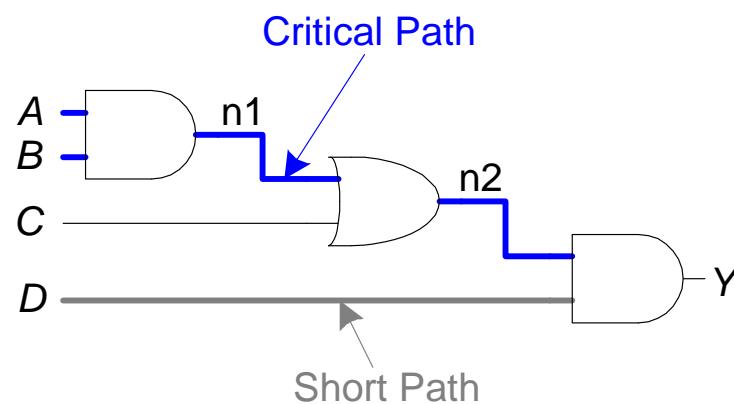
Con los circuitos digitales ocurre algo parecido, **Una de las claves en el diseño de circuitos es el “timing”.**

Los cambios en las entradas tardan un tiempo en propagarse a las salidas

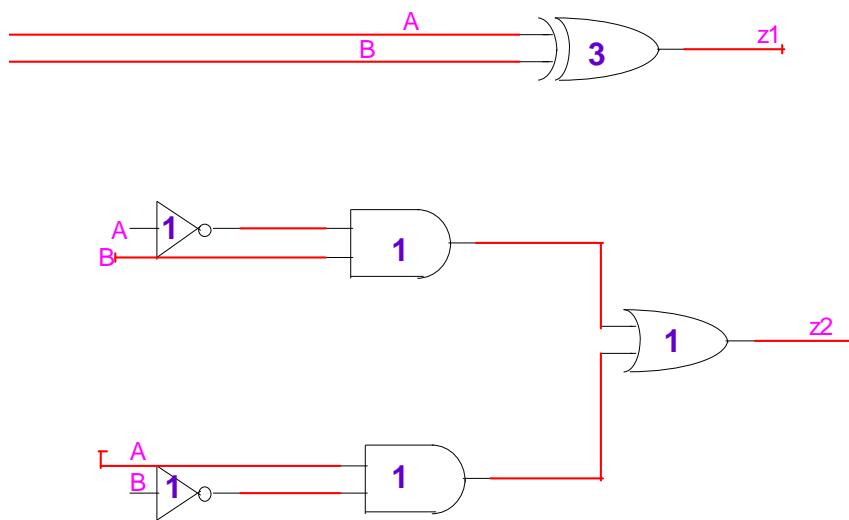


Cada camino puede tener un retardo distinto

- Nos interesa conocer el retardo máximo (o camino crítico) :
 - ¿Cuánto debemos esperar para estar seguros de que leeremos la salida correcta?
- El retardo mínimo puede ser importante
 - nos importa si no queremos que una señal cambie demasiado rápido

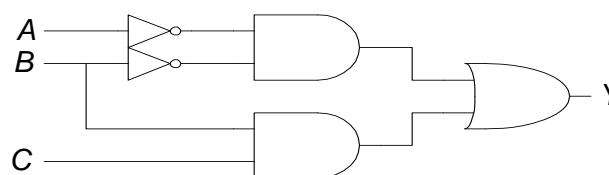


Otro ejemplo



Glitches: salidas temporalmente incorrectas

- Cada camino tiene retardos distintos
- Una entrada se puede actualizar antes en un camino que en otro generando brevemente salidas incorrectas
- Por eso debemos conocer el camino crítico



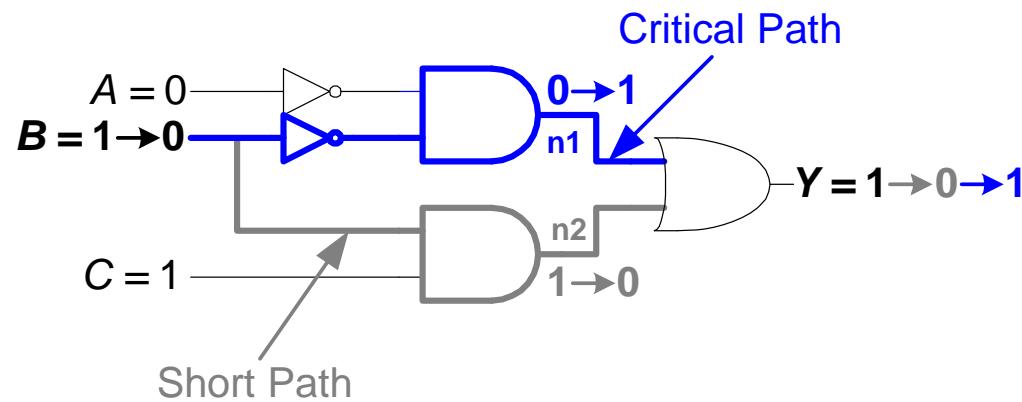
A Karnaugh map for the function Y. The columns are labeled AB (00, 01, 11, 10) and the rows are labeled C (0, 1). The map shows the following values:

	AB\C	00	01	11	10
0	1	0	0	0	
1	1	1	1	0	

Two blue-outlined rectangles highlight the minterms 100 and 111, which are the primary terms of the minimized Boolean expression.

$$Y = \overline{A}\overline{B} + BC$$

Glitch Example (cont.)



Recordatorio: Biestables

- Los elementos de memoria más sencillos son los biestables.
- Se usan de forma independiente o agrupados en registros
- Vistos en Intro. a los Computadores
- Flip-flops, latches,....
- En este tema nos centramos fundamentalmente en su comportamiento temporal.

¿Cómo puede un circuito recordar?

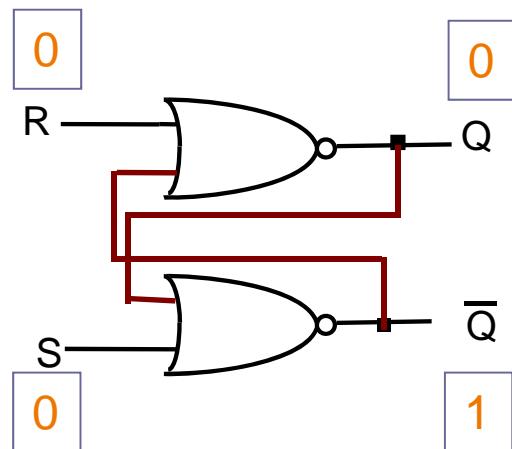
- Los circuitos combinacionales carecen de memoria. Siempre dan la misma salida para las mismas entradas
- Un circuito secuencial es aquél capaz de recordar eventos pasados y actuar en consecuencia:
 - **Las salidas** de los circuitos secuenciales dependen de las entradas y **también de su estado**
 - **El estado** de un circuito **depende de** los valores que tomaron las entradas en **el pasado**
 - El estado de un circuito se almacena utilizando **componentes de memoria** dentro del circuito
 - Estos elementos utilizan **lazos de realimentación** para ser capaces de almacenar información

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



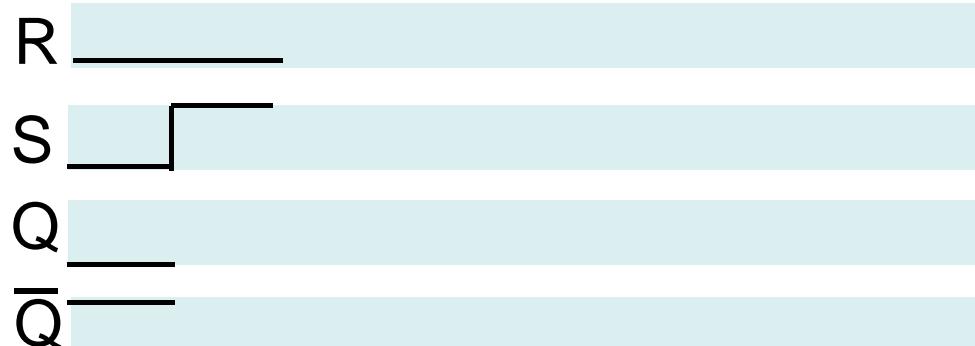
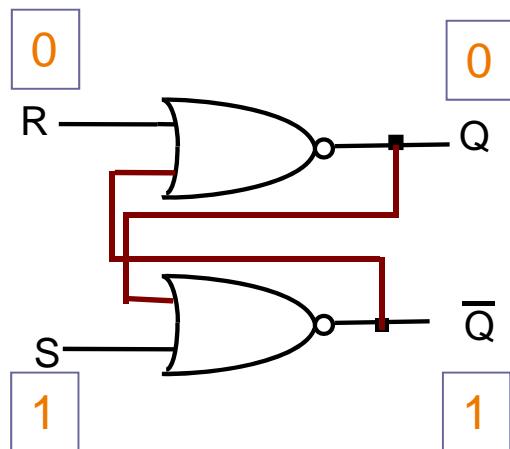
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



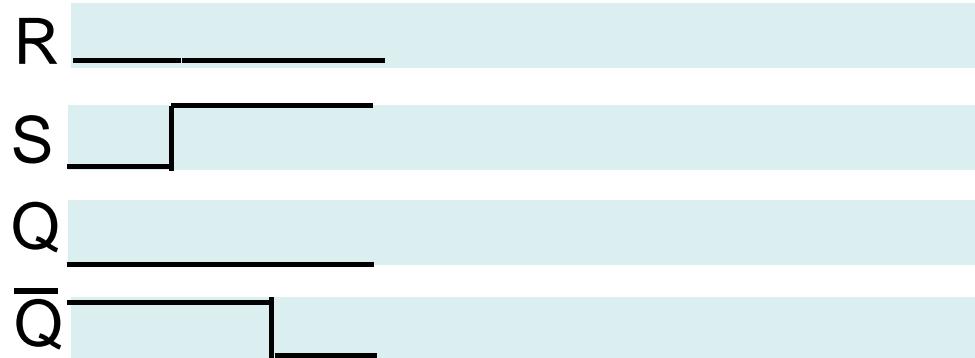
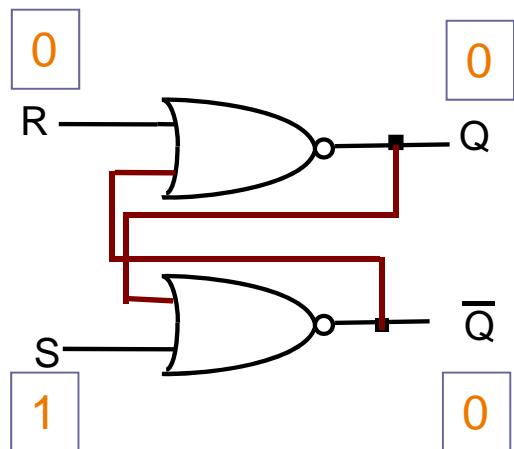
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



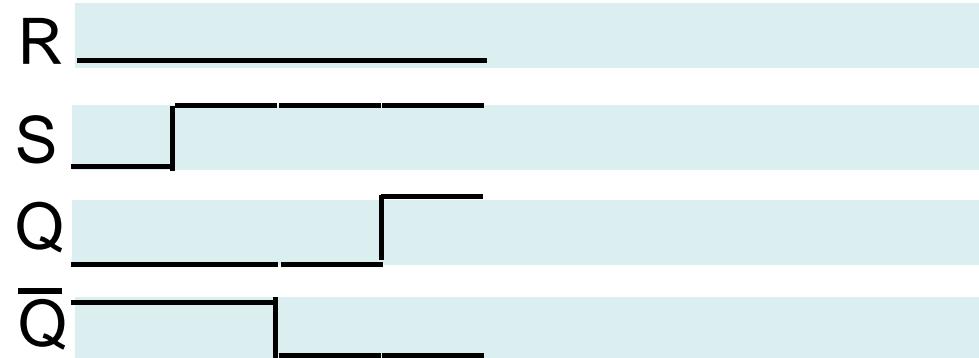
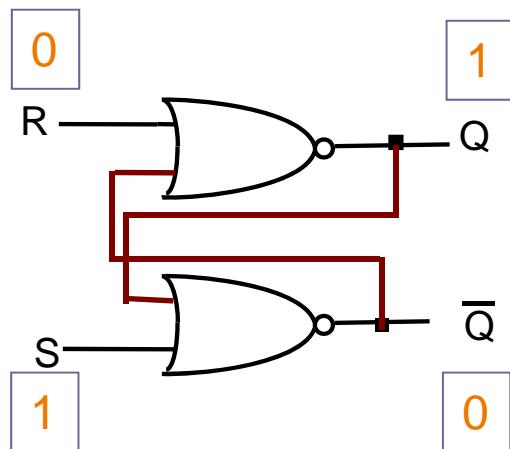
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



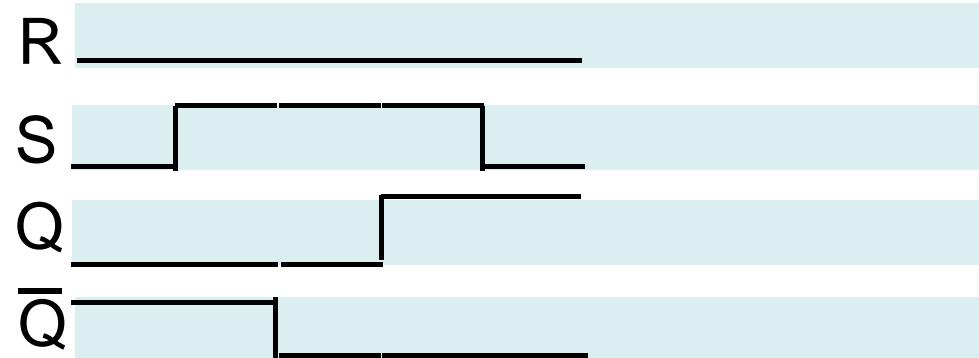
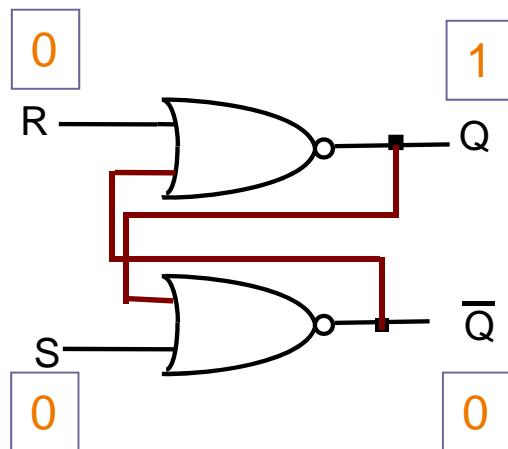
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



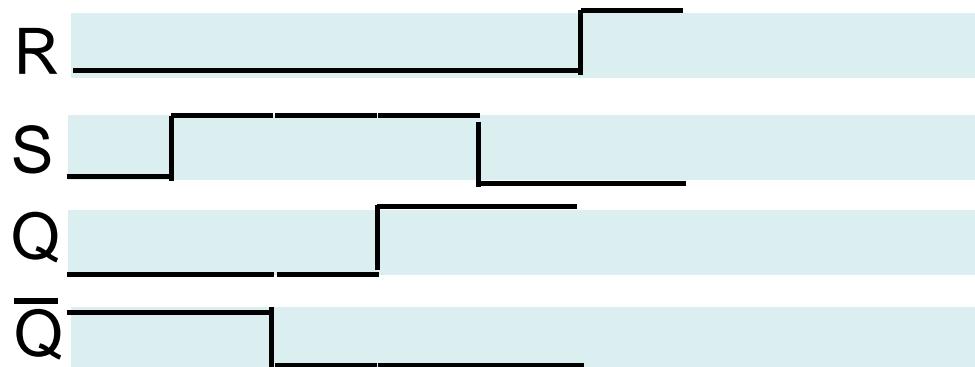
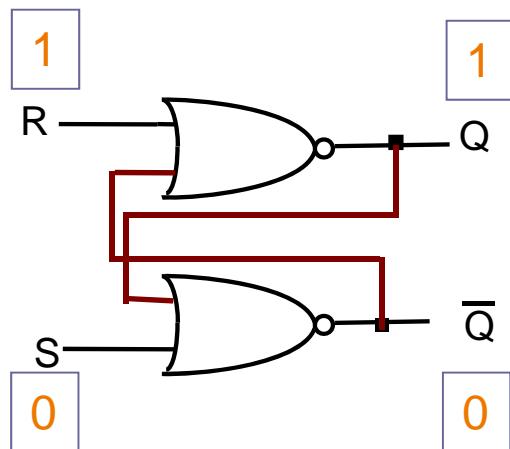
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



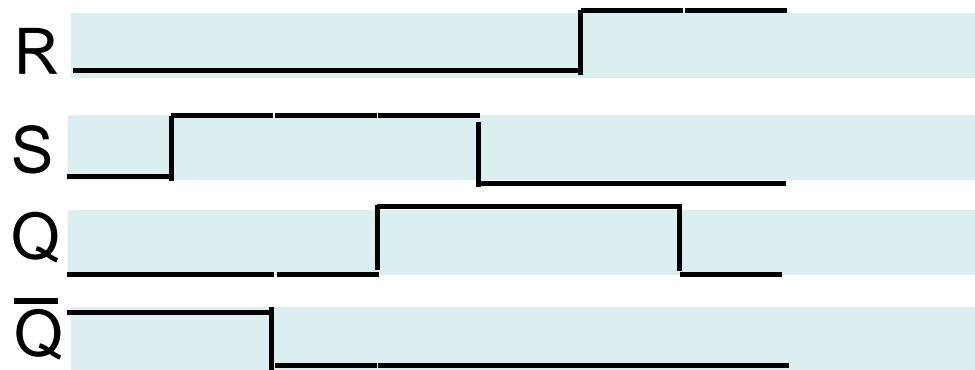
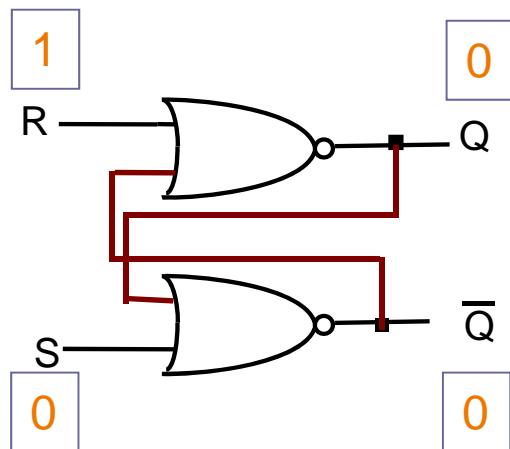
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



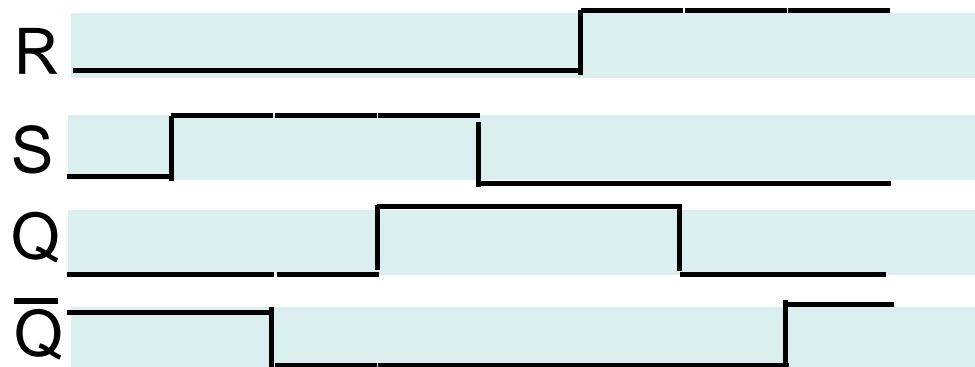
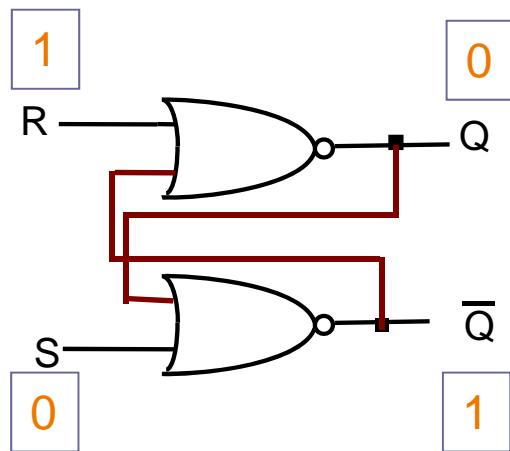
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



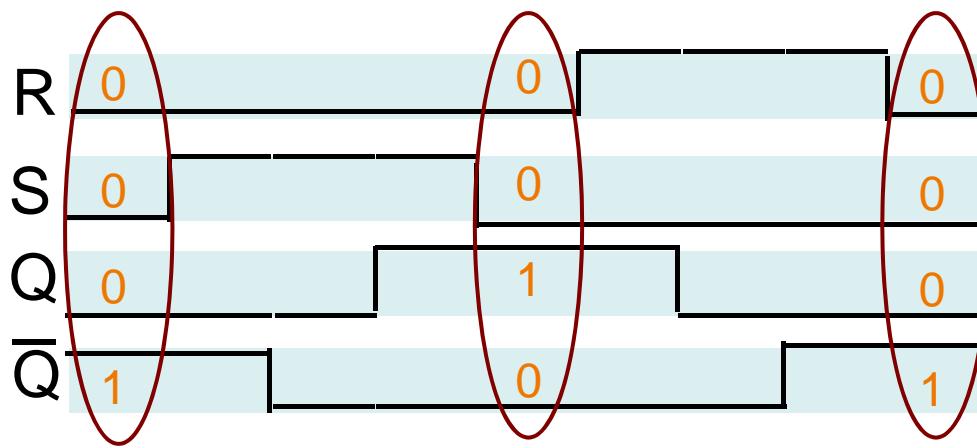
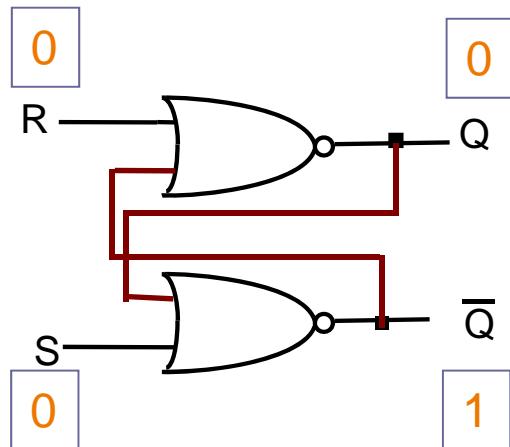
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?



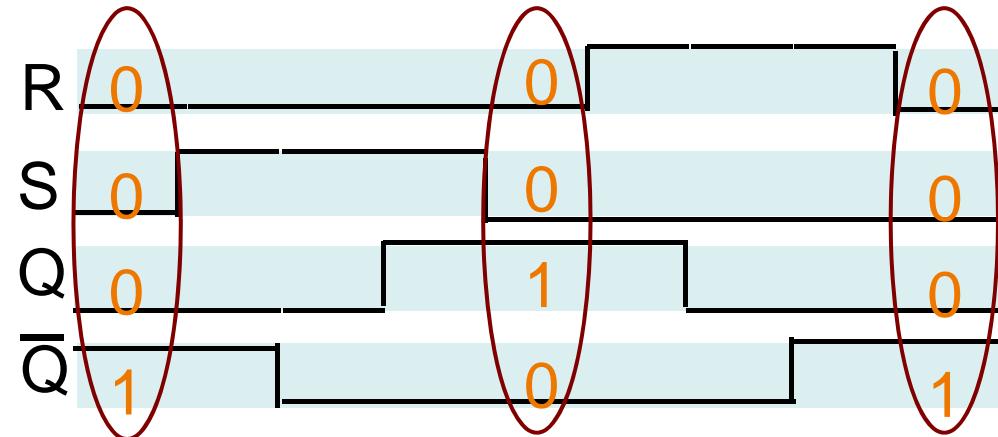
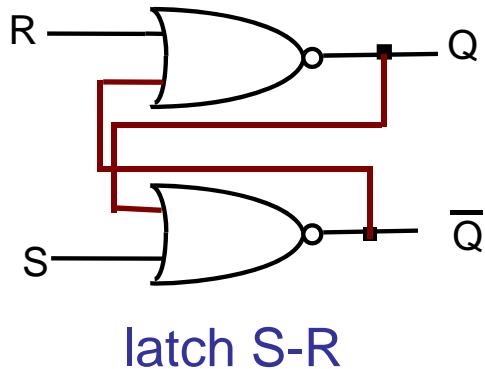
R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



Añadimos lazos de realimentación

Un circuito secuencial pueda tener salidas distintas para las mismas entradas



El comportamiento de un circuito se describe en una tabla con:

- el **estado actual**
- las **entradas**

y a partir de estos datos calcular el:

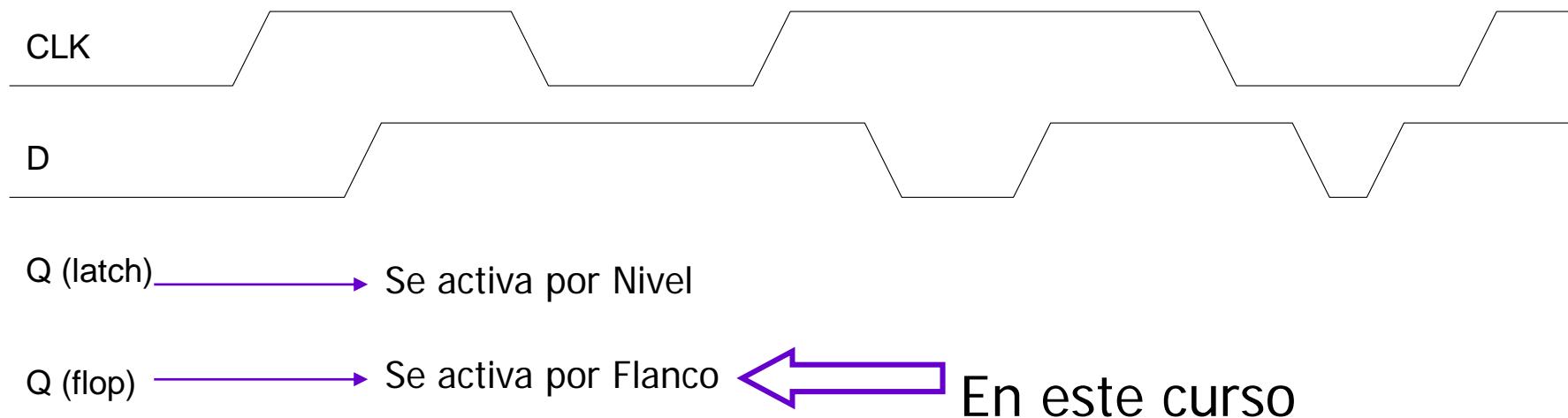
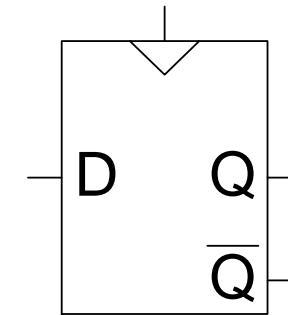
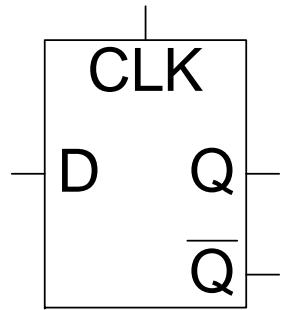
- **estado siguiente**
- **las salidas** (para este ejemplo coinciden)

Q	R	S	Qsig
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	Proh
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	Proh

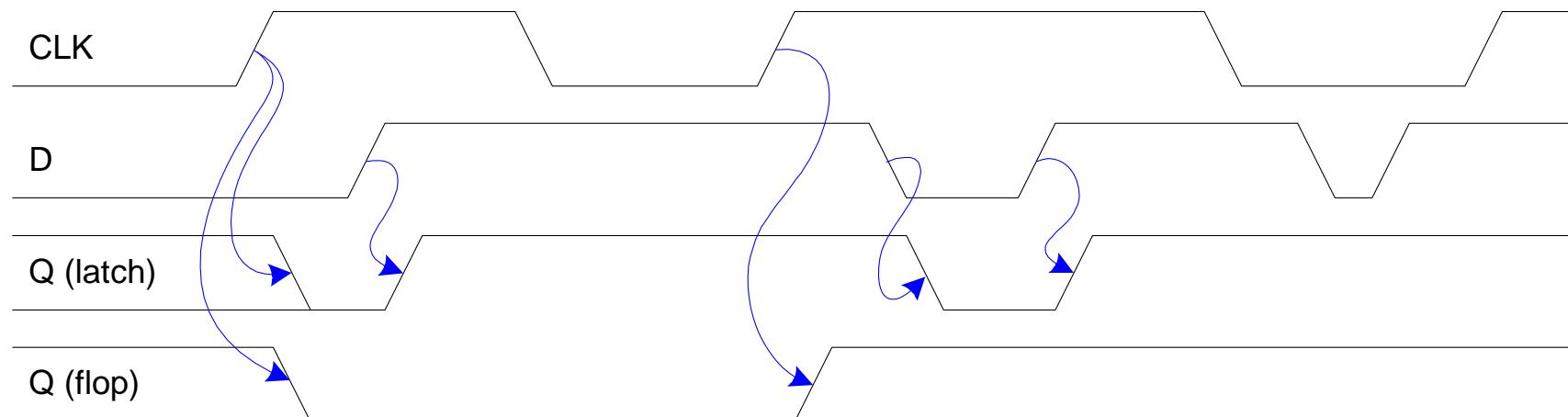
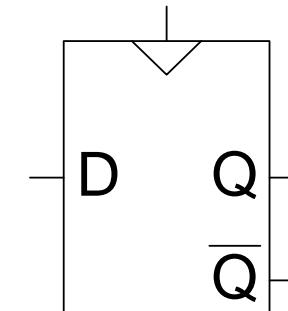
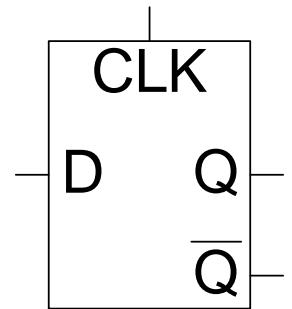
Tipos de biestables: según su comportamiento temporal

- asíncrono (latch) :
 - siempre responde a las entradas
- sensible a nivel (latch síncrono) :
 - responde a las entradas si la señal de capacitación vale 1
- disparado por flanco (flip-flop):
 - sólo lee las entradas cuando detecta un flanco (de subida o bajada) en la señal de capacitación
- maestro-esclavo (flip-flop):
 - Como el anterior pero implementado con un biestable sensible a alta y otro a baja

D Flip-Flop vs. D Latch



D Flip-Flop vs. D Latch



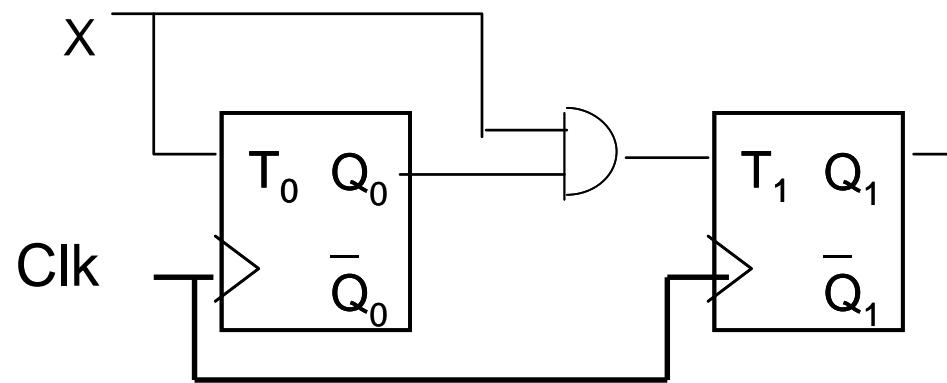
Características temporales de los biestables (y de los registros)

- **Retardo de propagación**
 - Desde el cambio en la entrada hasta el cambio en la salida
 - Para un biestable hay varios retardos (tantos como distintos cambios en las diferentes entradas)
- **Tiempo de set-up (establecimiento)**
 - Tiempo mínimo que la entrada debe permanecer estable ANTES del suceso del reloj
- **Tiempo de hold (mantenimiento)**
 - Tiempo mínimo que la entrada debe permanecer estable DESPUÉS del suceso del reloj

Cálculo del tiempo de ciclo

- Calcular el retardo de todos los caminos posibles y quedarnos con el mayor (el camino crítico)
- Caminos:
 - conexiones entre dos biestables,
 - o entre un biestable y una salida
- Considerar el retardo de los biestables y de la lógica combinacional.
- Objetivo:
 - Cuando llegue un nuevo flanco de reloj, todos los cambios generados por el flanco anterior deben ser estables
- Si un camino termina en un biestable ¡hay que incluir el tiempo de set-up!

Cálculo del tiempo de ciclo

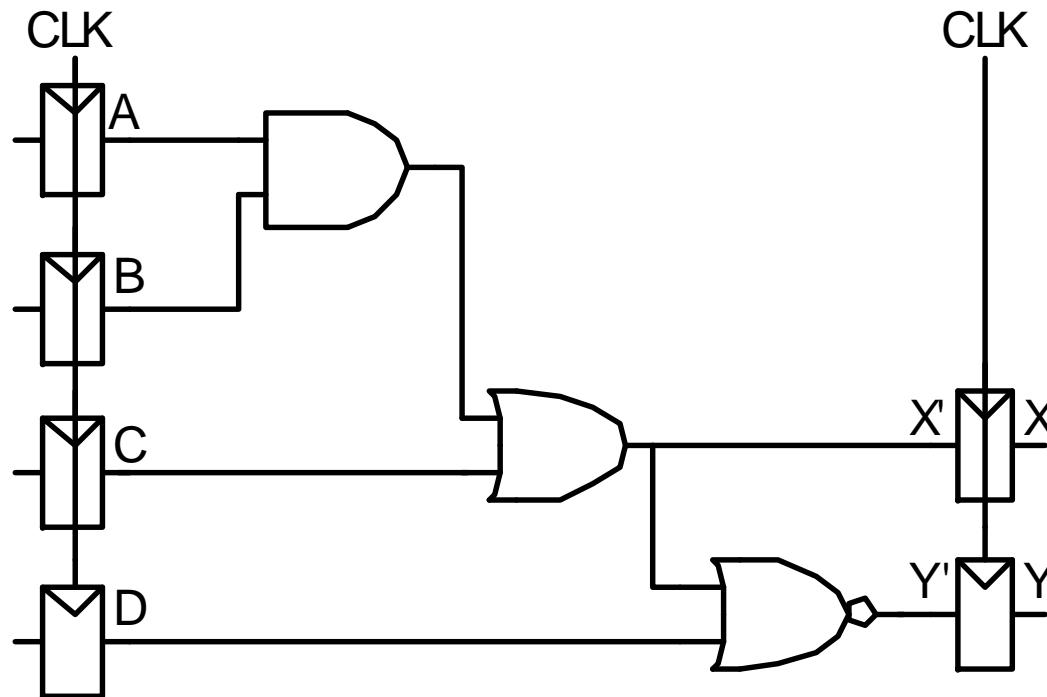


Retardo de los biestables (Tdelay): 6ns

Retardo puerta AND: 2 ns

Tset_up: 3ns

Cálculo del tiempo de ciclo



Retardo de los registros(Tdelay): 6ns

Retardo puertas: 2 ns

Tset_up: 3ns

Resumen de los biestables

- Los biestables permiten a los circuitos recordar
- El **valor de los biestables** en cada instante define el **estado del circuito**
- En un **circuito síncrono** los biestables deben actualizarse **todos a la vez, y tan sólo una vez por ciclo** de reloj
- Por ello es recomendable utilizar biestables disparados por flanco
- Al diseñar un circuito se debe asignar el tiempo de ciclo adecuado para que todos los biestables puedan actualizarse correctamente

Comportamiento temporal de las memorias

Las memorias tienen muchos parámetros temporales y su análisis en detalle es complejo, veamos un modelo muy simplificado

- Modo Lectura
 - Se comporta como un sistema combinacional con tiempo de acceso de lectura
- Modo Escritura
 - Se comporta como un registro: hay que dar tiempo a que se escriban las cosas (~setup)
 - Preparo todas las entradas
 - Espero tiempo de acceso en escritura

Práctica 1

**Diseño de un banco de registros
(2 sesiones de laboratorio)**

Objetivos

- Aprender a realizar diseños modulares.
 - Vamos a realizar un diseño complejo a partir de módulos estándar: muxes, dec y registros.
- Realizar análisis temporal de un módulo síncrono complejo:
 - tiempo de *setup*
 - retardo de propagación
 - Tiempo de ciclo/frecuencia máxima

Primera sesión: diseño de un banco de registros

- Utilizaréis módulos estándar
- **Trabajo previo:**
 - Aprender cómo funciona Logisim:
 - Primeros pasos con Logisim
 - Logisim Beginner's tutorial
 - Realizad el diseño en papel.
- Resultados:
 - Comprobar que el diseño funciona
 - Entender los retardos de cada operación

Segunda sesión: Estudio del tiempo de ciclo

- Integraréis vuestro diseño previo en un procesador sencillo
- Dos versiones:
 - un procesador que realiza todas las operaciones en un ciclo de reloj
 - El mismo procesador pero con algunos registros intermedios
- Debéis calcular el tiempo de ciclo mínimo para que el sistema funcione

DISEÑO DE LA RUTA DE DATOS Y LA UNIDAD DE CONTROL

Introducción

Contenidos

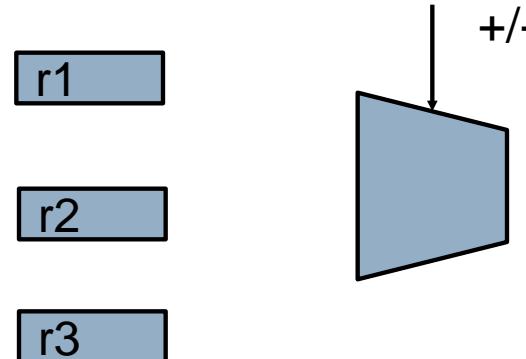
- Introducción
 - Importancia del diseño del procesador. Metodología de diseño de un procesador. Arquitectura MIPS: formato de la instrucción máquina y repertorio de instrucciones.
- Diseño de la ruta de datos monociclo
 - Componentes de la ruta de datos. Ensamblaje de la ruta de datos. Ruta de datos monociclo: puntos de control.
- Diseño del controlador monociclo
 - Determinación de los valores de los puntos de control. Control global vs. Control local. Ruta datos monociclo + controlador. Temporización monociclo.
- Diseño de la ruta de datos (multiciclo)
 - Ruta de datos multiciclo: con y sin buses.
- Diseño del controlador (multiciclo)
 - Diagrama de estados del controlador. El controlador como una FSM. Alternativas de implementación del controlador

Introducción

Metodología para el diseño de un procesador

- **Paso 1 Analizar el repertorio de instrucciones** para obtener los requisitos de la ruta de datos
 - **Ruta de datos**
 - **elementos de almacenamiento**
 - **visibles / transparentes** en el nivel de ISA (programador / compilador)
 - **Unidades funcionales**
 - **Elementos de direccionamiento e interconexión**
 - **Instrucción**
 - Conjunto de transferencias entre registros.
 - La ruta de datos debe ser capaz de soportar dichas transferencias.

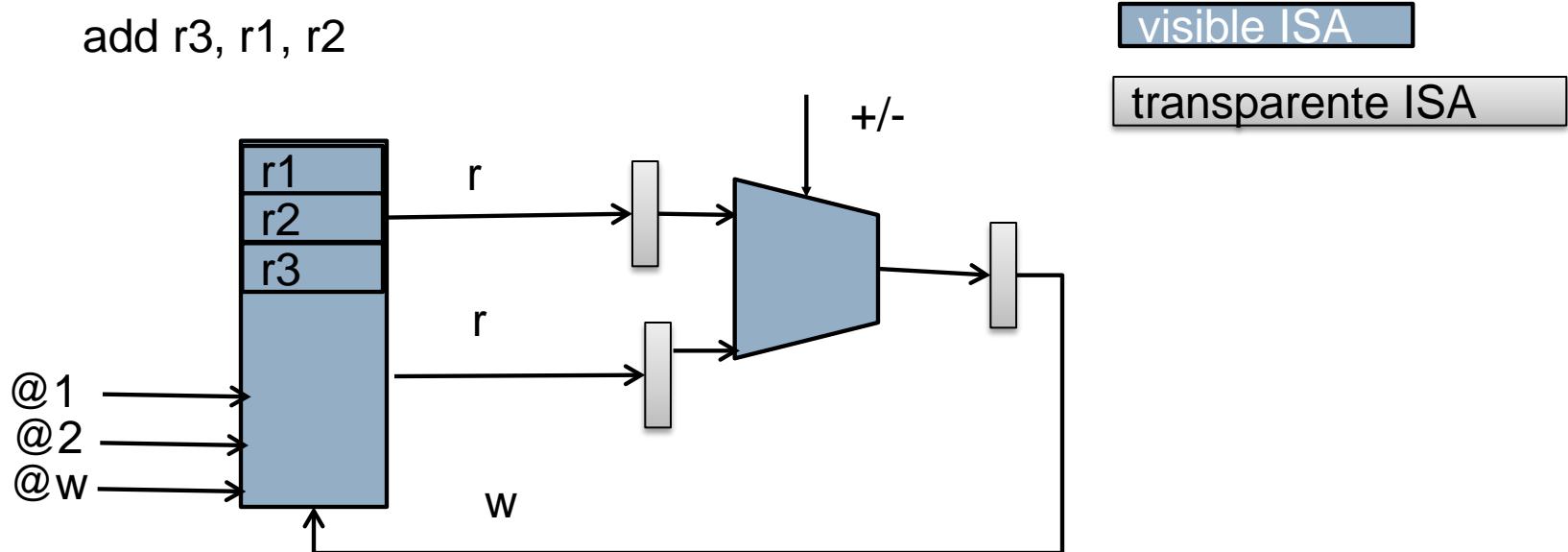
add r3, r1, r2



Introducción

Metodología para el diseño de un procesador

- **Paso 2 Establecer la metodología de temporización**
 - **Monociclo (CPI = 1)**: La instrucción se ejecuta en un ciclo
 - **Multiciclo (CPI > 1)**: La instrucción se ejecuta en varios ciclos
- **Paso 3 Seleccionar el conjunto de módulos** (de almacenamiento, operativos e interconexión) que forman la ruta de datos

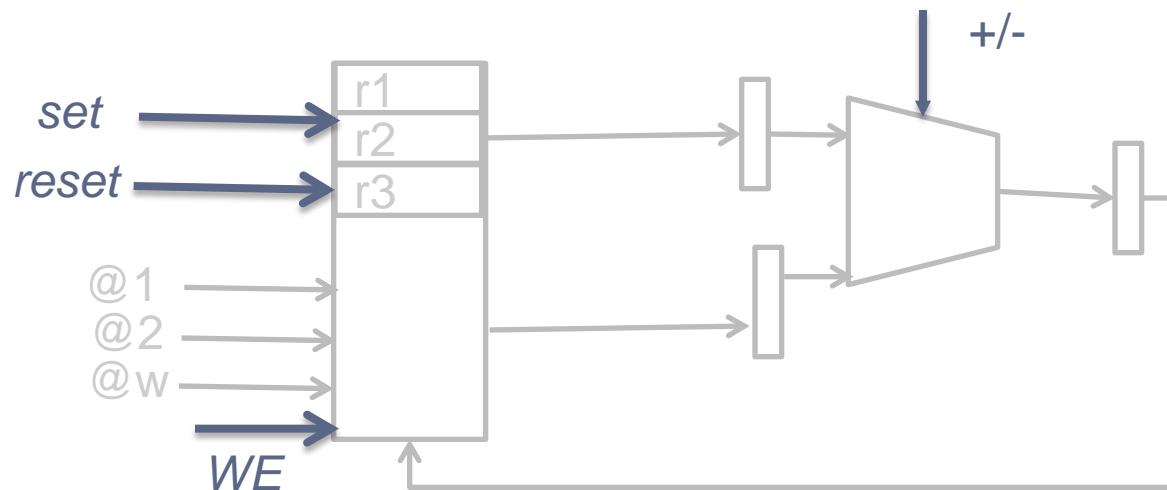


Introducción

Metodología para el diseño de un procesador

- **Paso 4 Ensamblar la ruta de datos** de modo que se cumplan los requisitos impuestos por el repertorio, localizando los puntos de control
- **Paso 5 Determinar los valores de los puntos de control** analizando las transferencias entre registros incluidas en cada instrucción.
- **Paso 6 Diseñar la lógica de control.**

add r3, r1, 32



Introducción

Arquitectura MIPS

- **Procesadores RISC de 32 y 64 bits**
- **Repertorio ortogonal**
 - Aprox. 100 instrucc. máquina + 30 instrucc. de aritmética en punto flotante
 - ⊕ Un tamaño de operandos y un modo de direccionamiento por instrucción
 - Instrucciones: cuatro grupos
 - ⊕ Movimiento de datos
 - ⊕ Aritmética entera, lógicas y desplazamiento
 - ⊕ Control de flujo
 - ⊕ Aritmética en punto flotante
 - 4 modos de direccionamiento
 - ⊕ Inmediato
 - ⊕ Directo de registros
 - ⊕ Indirecto con desplazamiento
 - ⊕ Indirecto con desplazamiento relativo al PC
 - 3 formatos de instrucción distintos con longitud única de 32 bits

Introducción

Arquitectura MIPS

- Arquitectura registro-registro
 - Sólo LOAD y STORE para referencia a memoria
 - Resto de instrucciones: operan sobre registros
 - Instrucciones con tres operandos
 - 2 op. fuente y 1 op. Destino
 - Notación ensamblador:
 - **op x, y, z ; x <- (y) op (z)**

Introducción

Arquitectura MIPS

- Banco de 64 registros (32 bits cada uno)
 - 32 de propósito general (R0-R31)
 - 32 para inst. en punto flotante (F0-F31).
 - Pueden usarse como:
 - 32 registros para ops en simple precisión (32 bits)
 - 16 registros para ops en doble precisión (64 bit)

Introducción

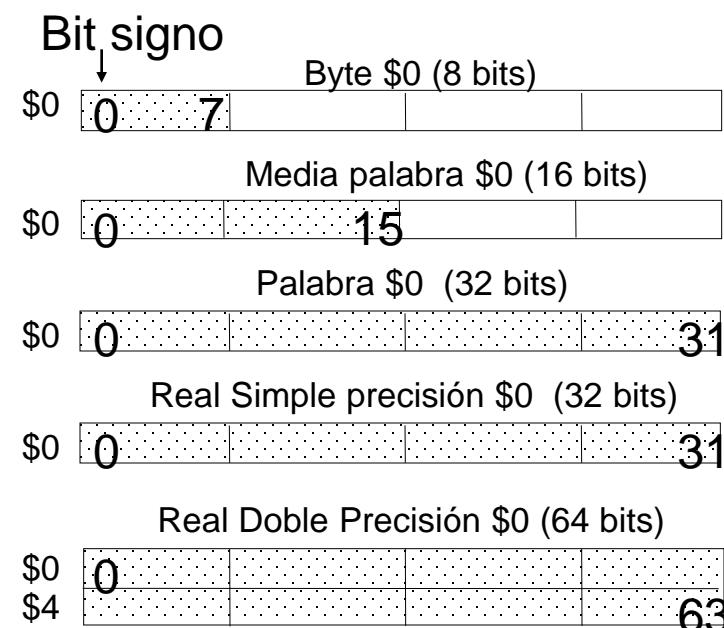
Tipos de datos arquitectura MIPS

- **Enteros**
 - **Tamaño Byte (8 bits)**
 - **Tamaño Media palabra (16 bits)**
 - **Tamaño Palabra (32 bits)**
 - **Reales en punto flotante**
 - **Simple precisión (32 bits)**
 - **Doble precisión (64 bits)**

Memoria (Ordenación Little-Endian)

<u>Palabra</u>	Media palabra \$0		Media palabra \$2	
\$00000000	Byte \$0	Byte \$1	Byte \$2	Byte \$3
\$00000004	Byte \$4	Byte \$5	Byte \$6	Byte \$7
	Media palabra \$4		Media palabra \$6	
	.		.	
	.		.	
	.		.	
\$FFFFFFFC				

Nota: Soporta tanto big-endian como little-endian



Introducción

Modos de direccionamiento MIPS

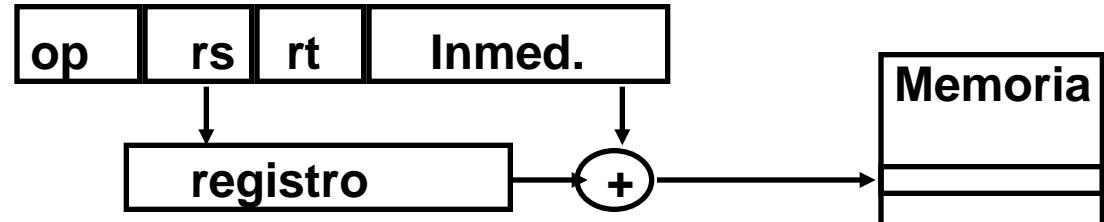
- ☒ Directo de registro



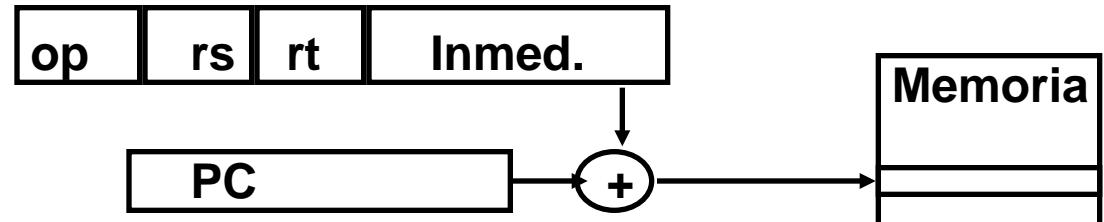
- ☒ Inmediato



- ☒ Indirecto con desplazamiento



- ☒ Indirecto con desplazamiento relativo a PC



Introducción

Arquitectura MIPS: Instrucciones que vamos a implementar

- **Instrucciones con referencia a memoria** (formato tipo I):
 - $lw \ rt, \text{inmed}(rs)$ $rt \leftarrow \text{Memoria}(rs + \text{SignExt}(\text{inmed}))$, $\text{PC} \leftarrow \text{PC} + 4$
 - $sw \ rt, \text{inmed}(rs)$ $\text{Memoria}(rs + \text{SignExt}(\text{inmed})) \leftarrow rt$, $\text{PC} \leftarrow \text{PC} + 4$
- **Instrucciones aritmético-lógicas con operandos en registros** (formato tipo R)
 - $add \ rd, \text{rs}, \text{rt}$ $rd \leftarrow \text{rs} + \text{rt}$, $\text{PC} \leftarrow \text{PC} + 4$
 - $sub \ rd, \text{rs}, \text{rt}$ $rd \leftarrow \text{rs} - \text{rt}$, $\text{PC} \leftarrow \text{PC} + 4$
 - $and \ rd, \text{rs}, \text{rt}$ $rd \leftarrow \text{rs and rt}$, $\text{PC} \leftarrow \text{PC} + 4$
 - $or \ rd, \text{rs}, \text{rt}$ $rd \leftarrow \text{rs or rt}$, $\text{PC} \leftarrow \text{PC} + 4$
- **Instrucciones de salto condicional** (formato tipo I)
 - $beq \ rs, \text{rt}, \text{inmed}$ si ($rs = rt$) entonces ($\text{PC} \leftarrow \text{PC} + 4 + 4 \cdot \text{SignExp}(\text{inmed})$)
en otro caso $\text{PC} \leftarrow \text{PC} + 4$

1. El ciclo de instrucción comienza buscando la instrucción en memoria (*fetch*)
 - instrucción $\leftarrow \text{Memoria}(\text{PC})$
2. En función del tipo de instrucción se realiza una de las anteriores operaciones
3. Se vuelve a comenzar

Introducción

Arquitectura MIPS: formato de la instrucción máquina

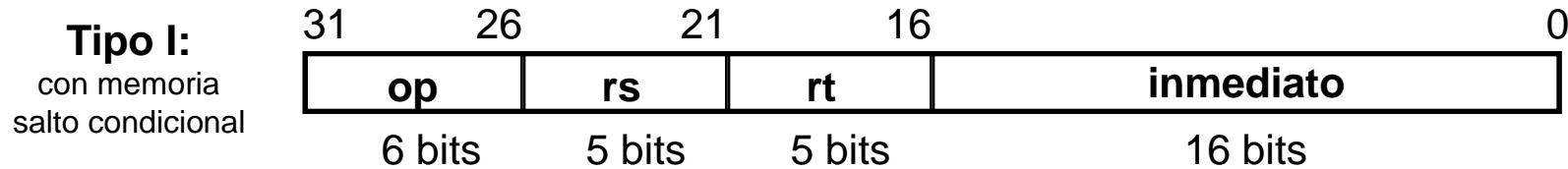
- Todas las instrucciones del repertorio del MIPS tienen 32 bits de anchura, repartidas en 3 formatos de instrucción diferentes:

	31	26	21	16	11	6	0
Tipo R: aritmético-lógicas	op	rs	rt	rd	shamt	funct	
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
Tipo I: con memoria salto condicional	31	26	21	16			0
	op	rs	rt		inmediato		
	6 bits	5 bits	5 bits		16 bits		
Tipo J: salto incondicional	31	26					0
	op				dirección		
	6 bits				26 bits		

- El significado de los campos es:
 - op**: identificador de instrucción
 - rs, rt, rd**: identificadores de los registros fuentes y destino
 - shamt**: cantidad a desplazar (en operaciones de desplazamiento)
 - funct**: selecciona la operación aritmética a realizar
 - inmediato**: operando inmediato o desplazamiento en direccionamiento a registro-base
 - dirección**: dirección destino del salto

Introducción

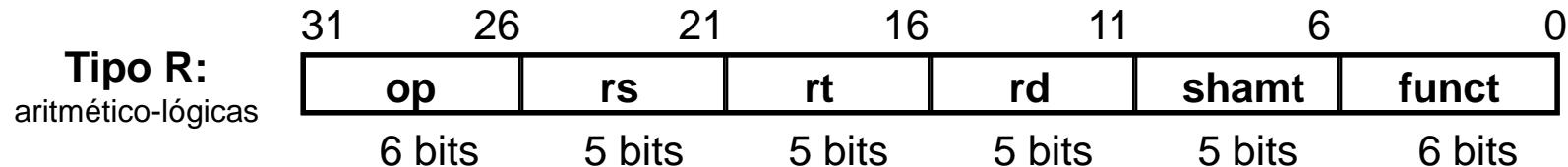
Arquitectura MIPS: formato de la instrucción máquina



- **Instrucciones con referencia a memoria** (formato tipo I):
 - **Iw rt, inmed(rs)** $rt \leftarrow \text{Memoria}(rs + \text{SignExt(inmed)})$, $PC \leftarrow PC + 4$
 - **sw rt, inmed(rs)** $\text{Memoria}(rs + \text{SignExt(inmed)}) \leftarrow rt$, $PC \leftarrow PC + 4$
- **Operaciones a realizar:**
 1. Leer la instrucción de la memoria de instrucciones
 2. Decodificarla para identificar el tipo de instrucción y sus operandos
 3. Leer los operandos de los registros
 4. Extender el signo del operando inmediato
 5. Realizar la suma de **rs** con el operando inmediato
 6. Acceder a memoria para leer o escribir
 7. En caso de la instrucción **Iw** se escribe el dato leído de memoria en el registro **rt**.
 8. Sumar 4 al contador de programa

Introducción

Arquitectura MIPS: formato de la instrucción máquina



- **Instrucciones aritmético-lógicas con operandos en registros** (formato tipo R)

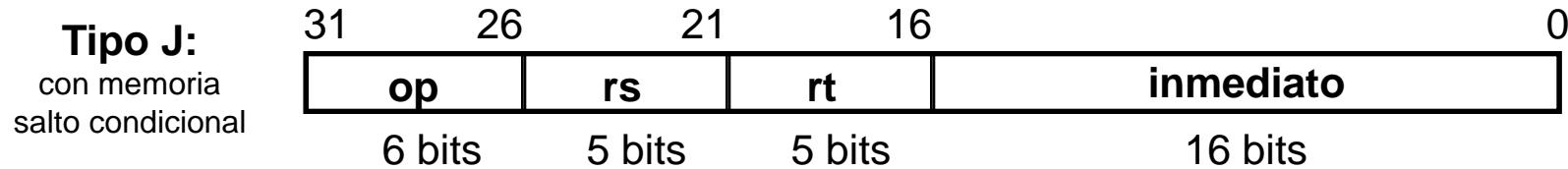
- add rd, rs, rt $rd \leftarrow rs + rt, PC \leftarrow PC + 4$
- sub rd, rs, rt $rd \leftarrow rs - rt, PC \leftarrow PC + 4$
- and rd, rs, rt $rd \leftarrow rs \text{ and } rt, PC \leftarrow PC + 4$
- or rd, rs, rt $rd \leftarrow rs \text{ or } rt, PC \leftarrow PC + 4$

- **Operaciones a realizar:**

1. Leer la instrucción de la memoria de instrucciones
2. Decodificarla para identificar el tipo de instrucción y sus operandos
3. Leer los operandos de los registros (rs y rt)
4. Realizar la operación indicada en funct
5. Escribir el resultado en el registro rd.
6. Sumar 4 al contador de programa

Introducción

Arquitectura MIPS: formato de la instrucción máquina



- **Instrucciones de salto condicional (formato tipo I)**
 - `beq rs, rt, inmed` si ($rs = rt$) entonces ($PC \leftarrow PC + 4 + 4 \cdot \text{SignExp}(inmed)$)
en otro caso $PC \leftarrow PC + 4$
- **Operaciones a realizar:**
 1. Leer la instrucción de la memoria de instrucciones
 2. Decodificarla para identificar el tipo de instrucción y sus operandos
 3. Leer los operandos de los registros
 4. Comparar rs con rt (se realiza una resta y se comprueba si el resultado es cero)
 5. Extender el signo del operando inmediato
 6. Sumar 4 al contador de programa
 7. Si se cumple la condición sumar el desplazamiento indicado en el operando inmediato

DISEÑO DE LA RUTA DE DATOS Y LA UNIDAD DE CONTROL

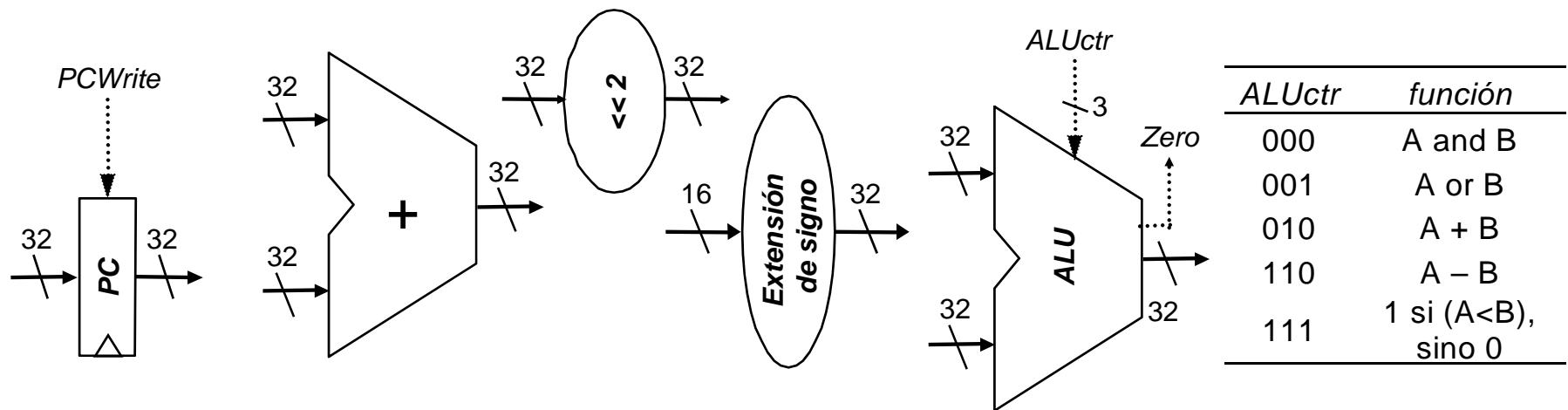
Ruta de datos monociclo

Contenidos

- Introducción
 - Importancia del diseño del procesador. Metodología de diseño de un procesador. Arquitectura MIPS: formato de la instrucción máquina y repertorio de instrucciones.
- Diseño de la ruta de datos monociclo
 - Componentes de la ruta de datos. Ensamblaje de la ruta de datos. Ruta de datos monociclo: puntos de control.
- Diseño del controlador monociclo
 - Determinación de los valores de los puntos de control. Control global vs. Control local. Ruta datos monociclo + controlador. Temporización monociclo.
- Diseño de la ruta de datos (multiciclo)
 - Ruta de datos multiciclo: con y sin buses.
- Diseño del controlador (multiciclo)
 - Diagrama de estados del controlador. El controlador como una FSM. Alternativas de implementación del controlador

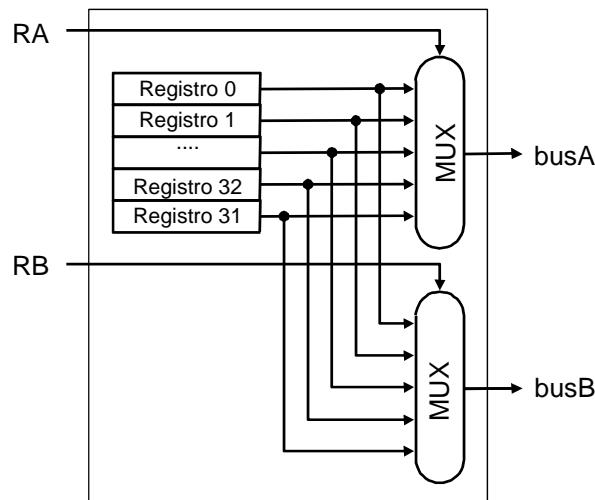
Diseño de la ruta de datos monociclo

- Para implementar el subconjunto del repertorio del MIPS en una implementación monociclo se requieren (el tamaño de palabra es de 32 bits):
 - Memoria de instrucciones**
 - Memoria de datos**
 - 32 registros de datos:** visibles por el programador.
 - Contador de programa**
 - 2 Sumadores:** para sumar 4 al PC, y para sumar al PC el valor inmediato de salto.
 - ALU:** capaz de realizar suma, resta, and, or, comparación de mayoría e indicación de que el resultado es cero (para realizar la comparación de igualdad mediante resta)
 - Extensor de signo:** para adaptar el operando inmediato de 16 bits al tamaño de palabra.
 - Desplazador a la izquierda:** para implementar la multiplicación por 4.

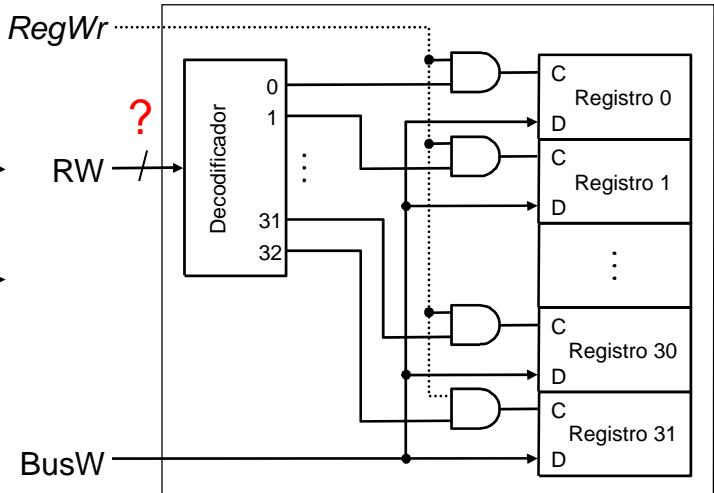
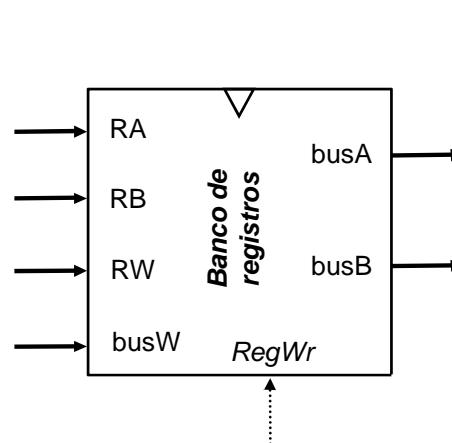


Diseño de la ruta de datos monociclo

- Los 32 registros se almacenan en un **banco de registros**. Dado que en las instrucciones de tipo R, se requiere un acceso simultáneo a 3 registros:
 - 2 salidas de datos de ____ bits
 - 1 entradas de datos de ____ bits
 - ____ entradas de ____ bits para la identificación de los registros
 - 1 entrada de control para habilitar la escritura sobre uno de los registros
 - 1 puerto de reloj (sólo determinante durante las operaciones de escritura, las de lectura son combinacionales)



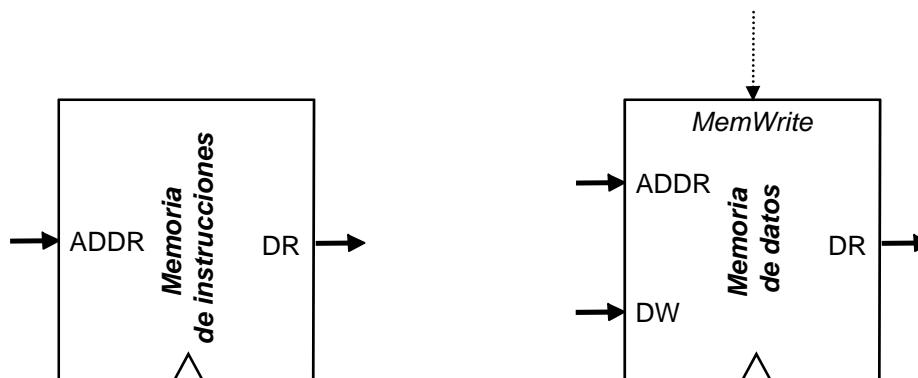
Mecanismo de lectura



Mecanismo de escritura

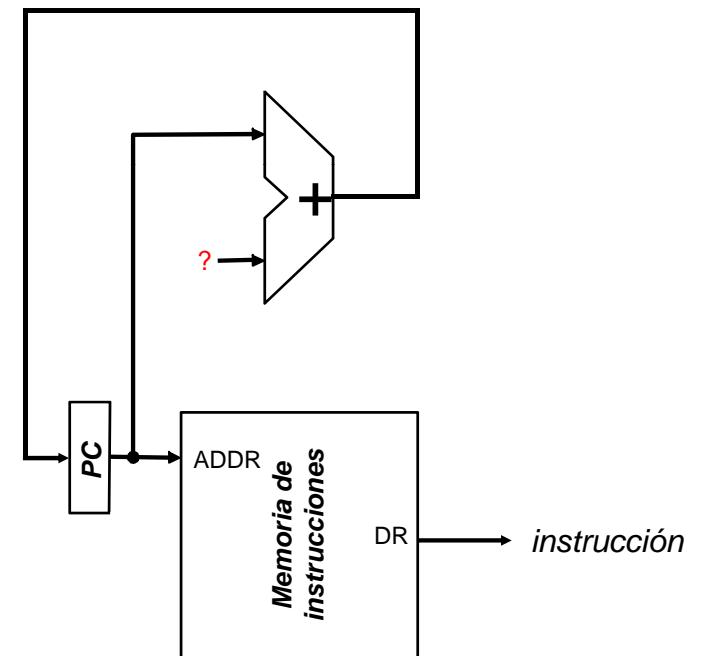
Diseño de la ruta de datos monociclo

- La memoria tendrá un comportamiento idealizado.
 - “Integrada” dentro de la CPU.
 - Direccionable por bytes, pero capaz de aceptar/ofrecer 4 bytes por acceso
 - 1 entrada de dirección
 - 1 salida de datos de 32 bits
 - 1 entrada de datos de 32 bits (sólo en la de datos)
 - 1 entrada de control para seleccionar el tipo de operación (lectura/escritura), sólo en la de datos
 - Se supondrá que se comporta temporalmente como el banco de registros (síncronamente)
 - tiempo de acceso menor que el tiempo de ciclo
 - Se supondrá dividida en dos para poder hacer dos accesos a memoria en el mismo ciclo:
 - Memoria de instrucciones
 - Memoria de datos



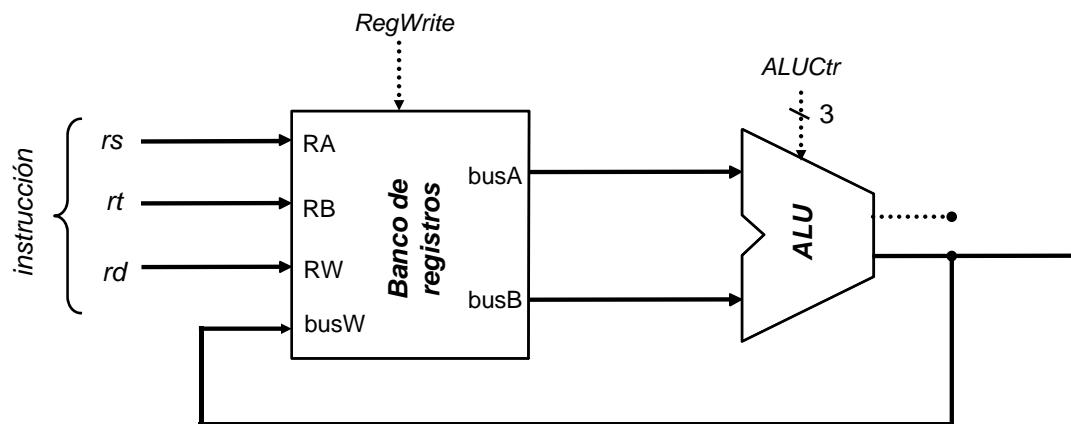
Diseño de la ruta de datos monociclo

- La **búsqueda de instrucciones** implica:
 - Leer la instrucción ubicada en la dirección de la **memoria de instrucciones** indicada por el **contador de programa**:
- La **ejecución secuencial** de programas implica el **secuenciamiento implícito**:
 - Actualizar el **contador de programa** para que apunte a la siguiente instrucción (sumando **?** por ser una memoria direccionable por bytes y una arquitectura con tamaño de palabra de **?** bits)

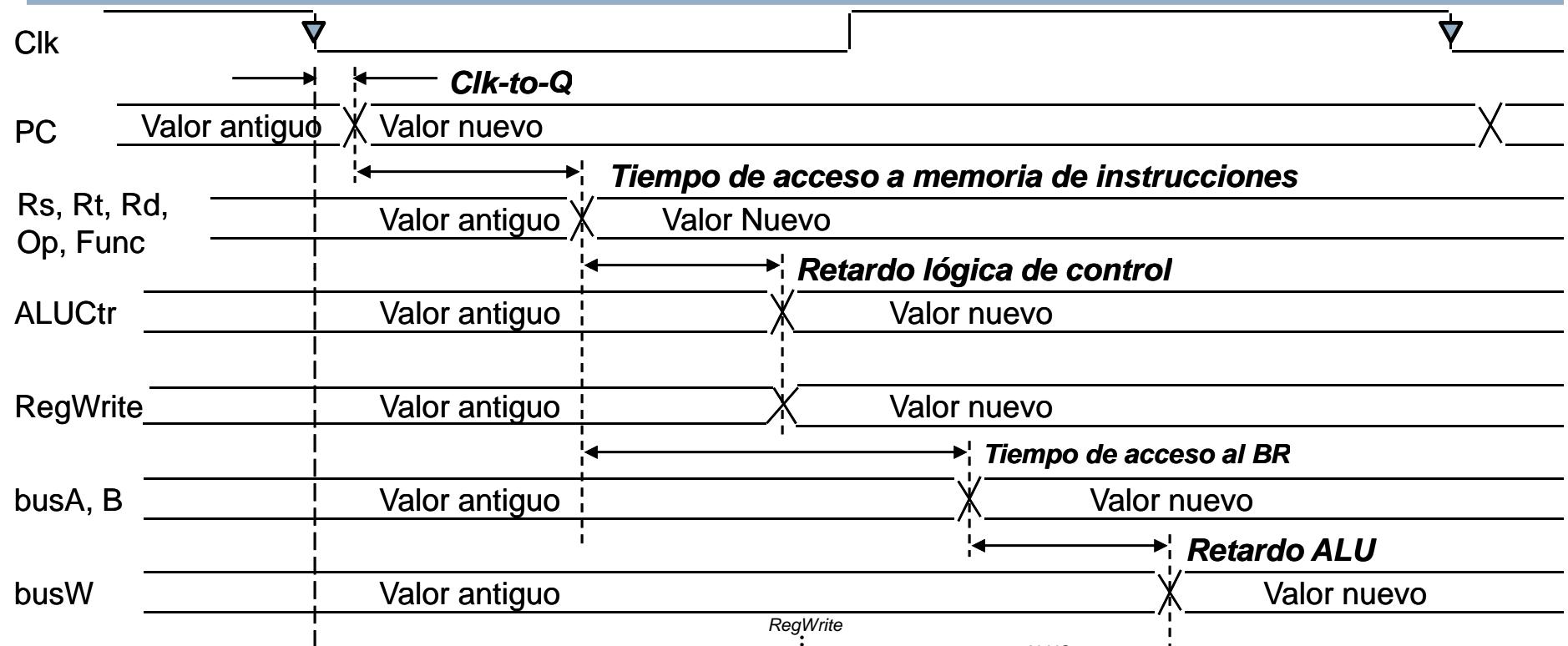


Diseño de la ruta de datos monociclo

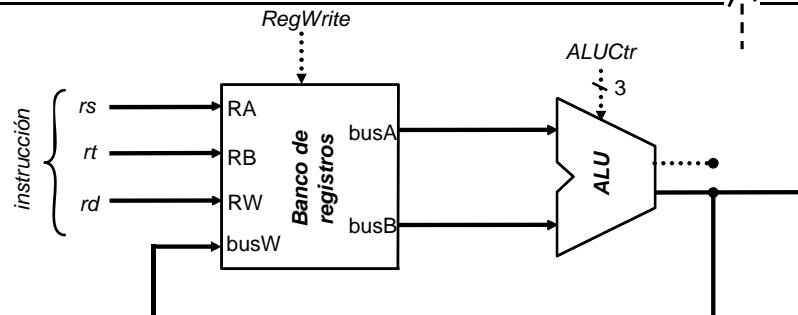
- Las instrucciones **aritmético lógicas** (tipo-R) implican:
 - $BR(rd) \leftarrow BR(rs) \text{ funct } BR(rt)$
 - Leer dos registros cuyos identificadores se ubican en los campos **rs** y **rt** de la instrucción:
 - Operar sobre ellos según el contenido del campo de código de operación aritmética (**funct**) de la instrucción
 - Almacenar el resultado en otro registro cuyo identificador se localiza en el campo **rd** de la instrucción



Diseño de la ruta de datos monociclo

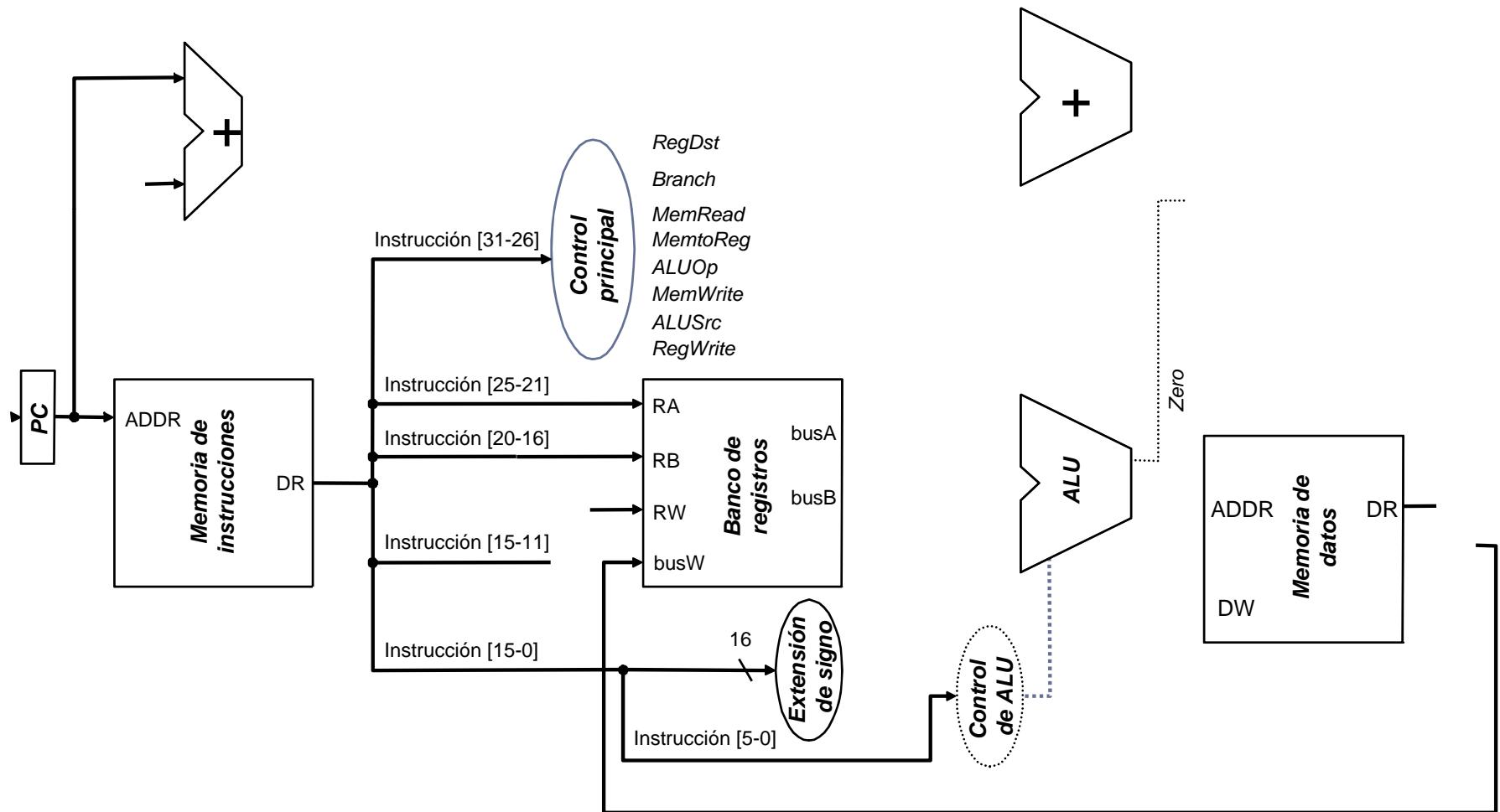


cronograma de una
operación aritmético-lógica



Señala la
escritura

Diseño de la ruta de datos monociclo - Completar



Diseño de la ruta de datos monociclo – Id/st

- La **instrucción de carga** (lw) implica:
 - $BR(rt) \leftarrow \text{Memoria}(BR(rs) + \text{SignExt}(inmed))$
 - Calcular la dirección efectiva de memoria:
 - Leyendo el *registro base* cuyo identificador se ubica en el campo **rs** de la instrucción
 - Obteniendo un *desplazamiento* de 32 bits a partir de la **extensión** del campo de operando inmediato (**inmed**) de la instrucción
 - **Sumando** base y desplazamiento.
 - Leer dato ubicado en la **memoria de datos** cuya dirección es la anteriormente calculada
 - Almacenar el dato leído de memoria en el registro cuyo identificador se especifica en el campo **rt** de la instrucción
- La **instrucción de almacenaje** (sw) implica:
 - $\text{Memoria}(BR(rs) + \text{SignExt}(inmed)) \leftarrow BR(rt)$
 - Leer el dato almacenado en el registro cuyo identificador se especifica en el campo **rt** de la instrucción
 - Calcular la dirección efectiva de memoria:
 - Leyendo el *registro base* cuyo identificador se ubica en el campo **rs** de la instrucción
 - Obteniendo un *desplazamiento* de 32 bits a partir de la **extensión** del campo de operando inmediato (**inmed**) de la instrucción
 - **Sumando** base y desplazamiento.
 - Almacenar el dato leído en la **memoria de datos** en la dirección anteriormente calculada

Diseño de la ruta de datos monociclo - beq

- La **instrucción de salto condicional** (beq) implica
 - si ($BR(rs) = BR(rt)$) entonces ($PC \leftarrow PC + 4 \cdot \text{SignExp}(inmed)$)
 - Leer dos registros cuyos identificadores se ubican en los campos **rs** y **rt** de la instrucción:
 - Comparar la igualdad de sus contenidos y en función del resultado:
 - No hacer nada o
 - **Sumar** al contador del programa un *desplazamiento* de 32 bits obtenido a partir de la **extensión** del campo de operando inmediato (**inmed**) de la instrucción

Señales de Control – arit / br

Aritméticas:

$rd \leftarrow rs \text{ and } rt, PC \leftarrow PC + 4$

Tipo R: aritmético-lógicas	31	26	21	16	11	6	0
	op	rs	rt	rd	shamt	funct	
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Salto:

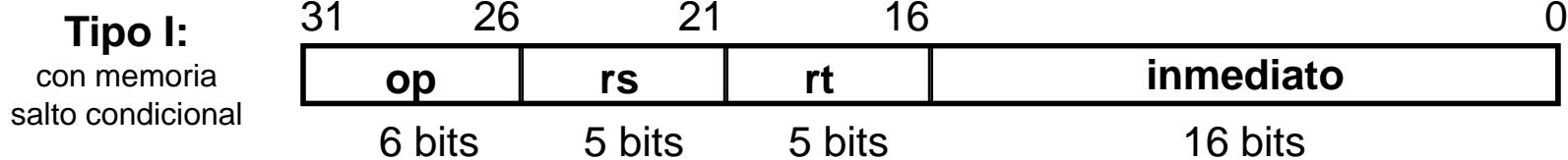
beq rs, rt, inmed si ($rs = rt$) entonces ($PC \leftarrow PC + 4 + 4 \cdot \text{SignExp}(\text{inmed})$)
 en otro caso $PC \leftarrow PC + 4$

Tipo I: con memoria salto condicional	31	26	21	16	0
	op	rs	rt	inmediato	
	6 bits	5 bits	5 bits	16 bits	

Señales de Control – Id/st

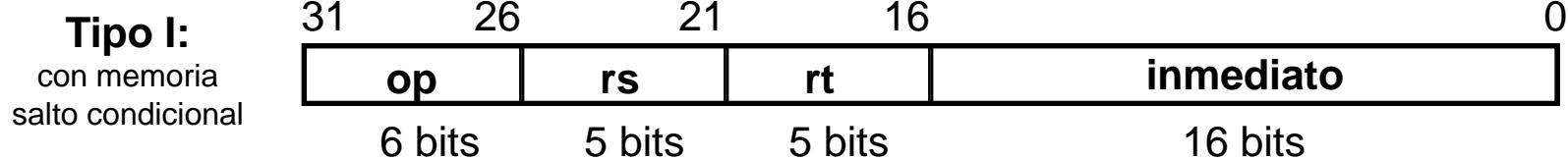
Load

$rt \leftarrow \text{Memoria}(rs + \text{SignExt}(inmed)) , PC \leftarrow PC + 4$



Store

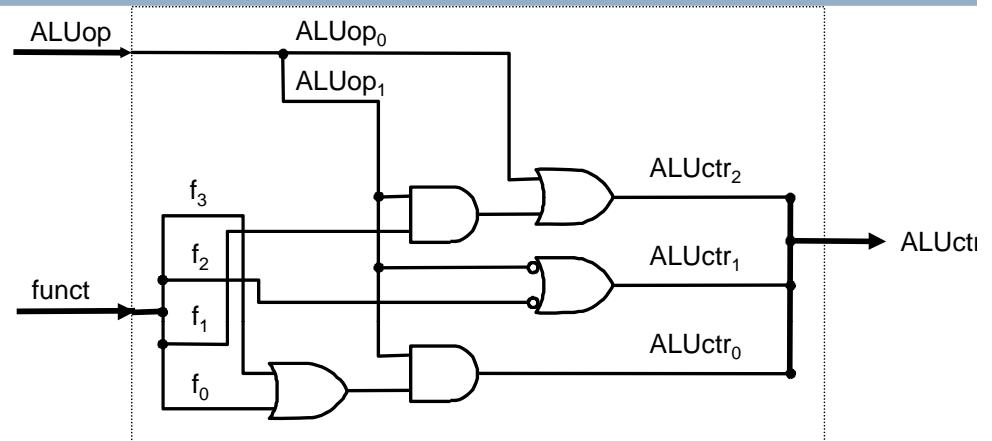
$\text{Memoria}(rs + \text{SignExt}(inmed)) \leftarrow rt, PC \leftarrow PC + 4$



diseño del controlador (monociclo)

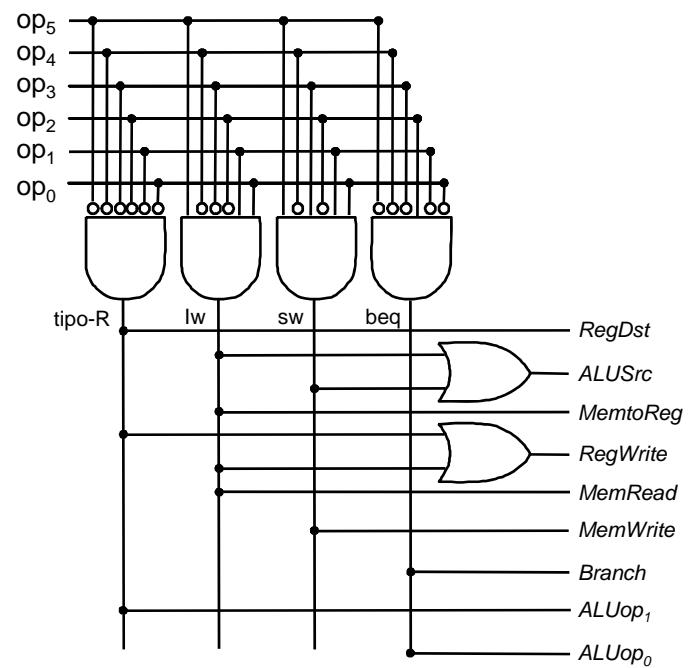
Control de la ALU

op	funct	ALUop	ALUctr
100011 (lw)		00	010
101011 (sw)	XXXXXX	00	010
000100 (beq)		01	110
	100000 (add)	11	010
	100010 (sub)	11	110
000000 (tipo-R)	100100 (and)	11	000
	100101 (or)	11	001
	101010 (slt)	11	111



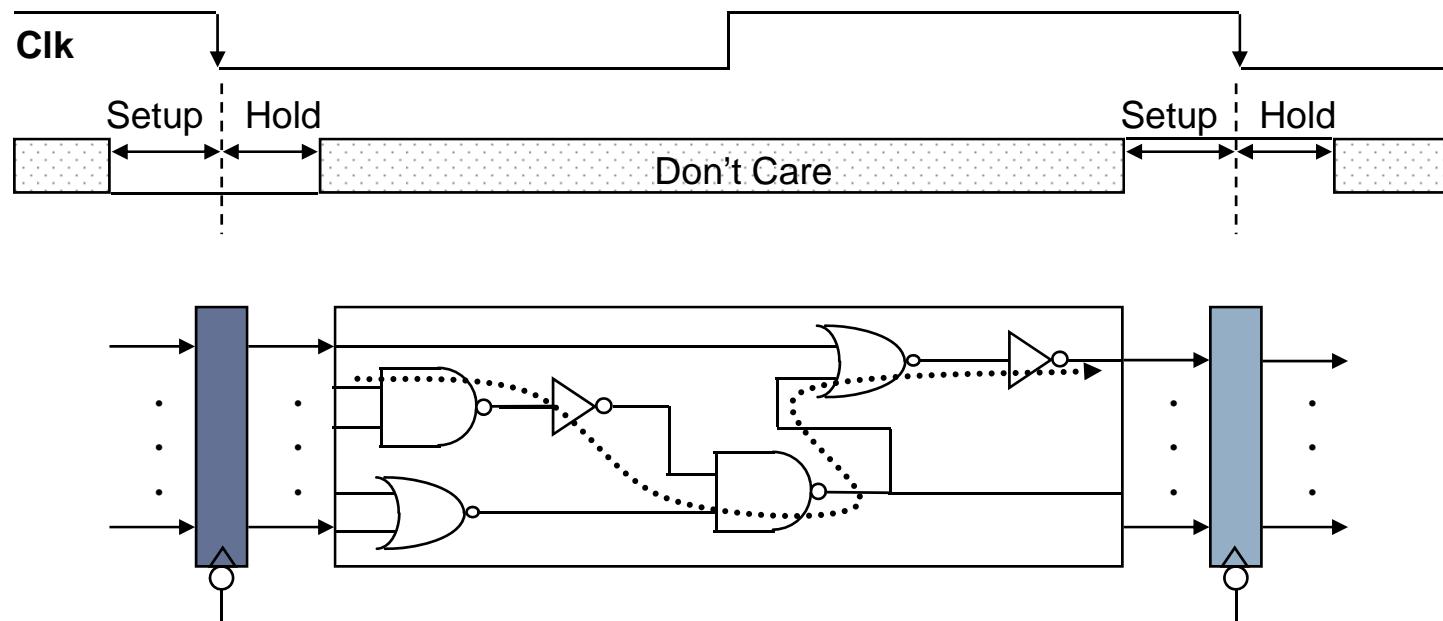
Control principal

op	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUop
100011 (lw)	0	1	1	1	1	0	0	00
101011 (sw)	X	1	X	0	0	1	0	00
000100 (beq)	X	0	X	0	0	0	1	01
000000 (tipo-R)	1	0	0	1	0	0	0	11



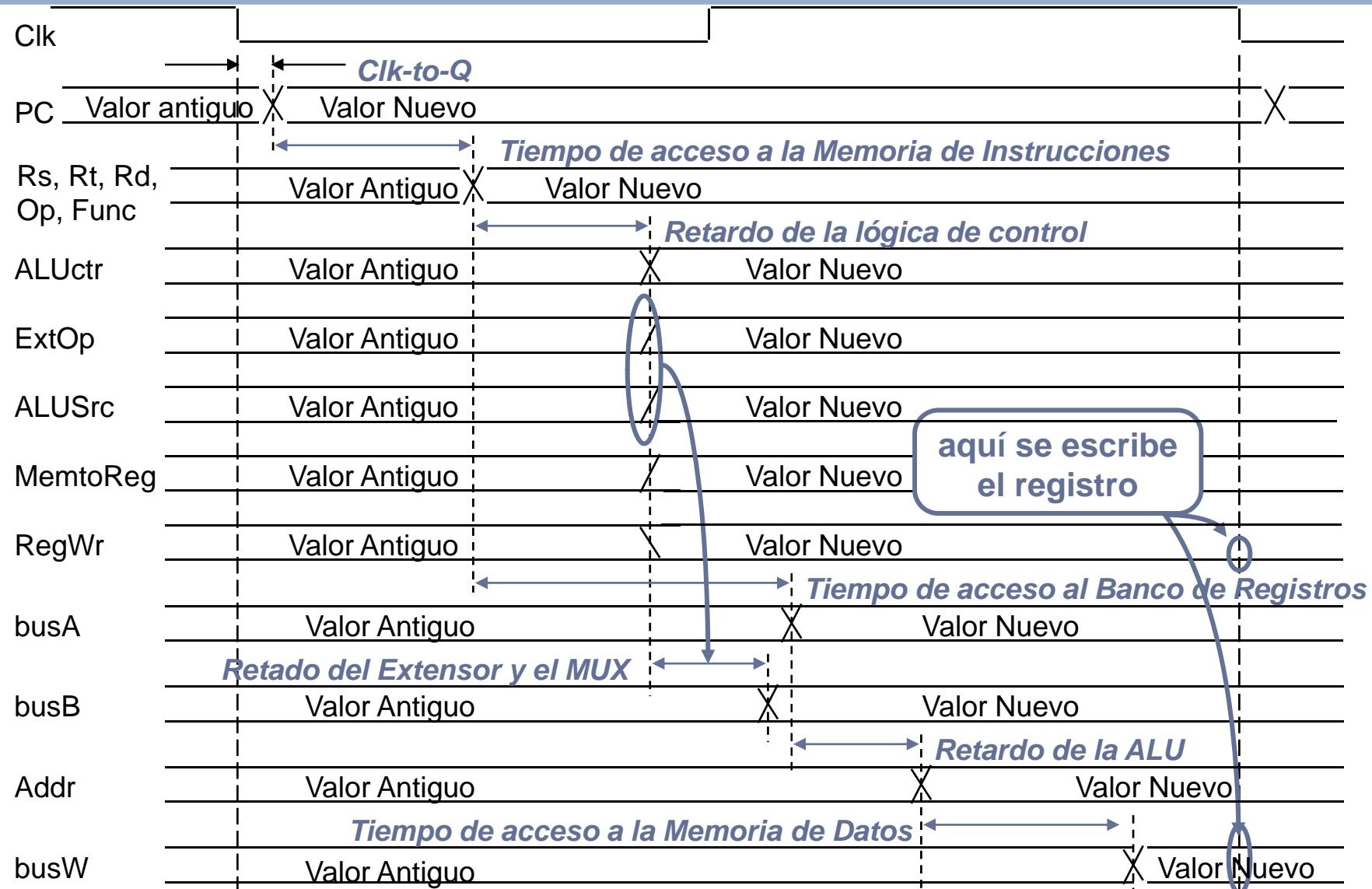
Temporización (monociclo)

- Ejecución típica (de una instrucción)
 - Todos los registros se cargan simultáneamente (de modo selectivo)
 - Todos los valores se propagan a través de las redes combinacionales hasta estabilizarse en las entradas de los registros
 - Se repite indefinidamente el proceso
- Todos los elementos de almacenamiento están sincronizados al mismo flanco de reloj:
 - Tiempo de ciclo = Hold + Camino con retardo máximo + Setup + Clock Skew



Cronograma de la ejecución de una instrucción lw

cronograma completo de la ejecución de la instrucción lw



Limitaciones del MIPS monociclo



- **Problema:** en un controlador monociclo:
 - El reloj debe tener igual periodo que la instrucción más lenta
 - No se puede reutilizar hardware
- **Solución:** dividir la ejecución de la instrucción en varios ciclos más pequeños:
 - Cada instrucción usará el número de ciclos que necesite
 - Un mismo elemento hardware se puede ser utilizado varias veces en la ejecución de una instrucción si se hace en ciclos diferentes
 - Se requieren elementos adicionales para almacenar valores desde el ciclo en que se calculan hasta el ciclo en que se usan.

Ejemplo de diseño de ruta de datos

■ Problema:

- Crear un procesador capaz de multiplicar y trasponer matrices (DSP: Digital Signal Processing)
- El procesador debe poseer siguiente juego de instrucciones:
 - Carga de un valor en un registro **LOAD #inmediato, Ri**
 - Lectura de una posición de memoria **READ Rf, Rd** (lee el valor de la memoria que se encuentra en Rf y lo escribe en Rd, además suma uno al valor almacenado en Rf).
 - Lectura especial de memoria **READE Rf, Rd** (lee el valor de la memoria que se encuentra en Rf y lo escribe en Rd, además suma 'n' al valor almacenado en Rf).
 - Escritura en una posición de memoria **WRITE Rf, Rd** (escribe e valor almacenado en Rf en la posición de memoria Rd, además suma uno al valor almacenado en Rd).
 - Escritura especial de memoria **WRITEE Rf, Rd** (escribe e valor almacenado en Rf en la posición de memoria Rd, además suma 'n' al valor almacenado en Rd).
 - Suma **ADD Rf1, Rf2, Rd => Rd = Rf1+Rf2**
 - Multiplicación **MUL Rf1, Rf2, Rd => Rd = Rf1*Rf2**
- Desarrollar el hardware necesario para que las instrucciones sean decodificadas y realicen su función.

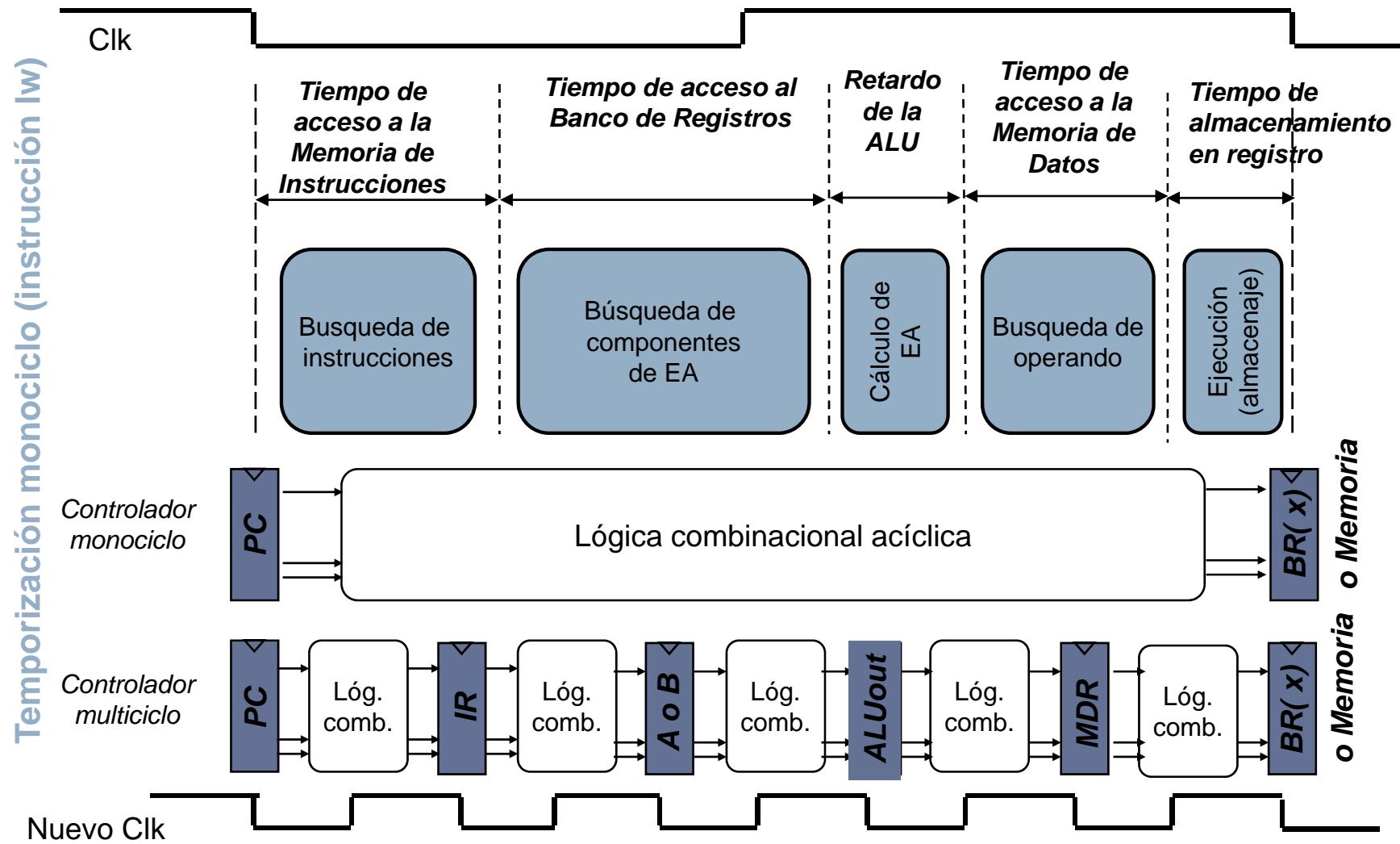
DISEÑO DE LA RUTA DE DATOS Y LA UNIDAD DE CONTROL

Ruta de datos multiciclo

Contenidos

- Introducción
 - Importancia del diseño del procesador. Metodología de diseño de un procesador. Arquitectura MIPS: formato de la instrucción máquina y repertorio de instrucciones.
- Diseño de la ruta de datos monociclo
 - Componentes de la ruta de datos. Ensamblaje de la ruta de datos. Ruta de datos monociclo: puntos de control.
- Diseño del controlador monociclo
 - Determinación de los valores de los puntos de control. Control global vs. Control local. Ruta datos monociclo + controlador. Temporización monociclo.
- Diseño de la ruta de datos (multiciclo)
 - Ruta de datos multiciclo: con y sin buses.
- Diseño del controlador (multiciclo)
 - Diagrama de estados del controlador. El controlador como una FSM. Alternativas de implementación del controlador

Temporización (multiciclo)



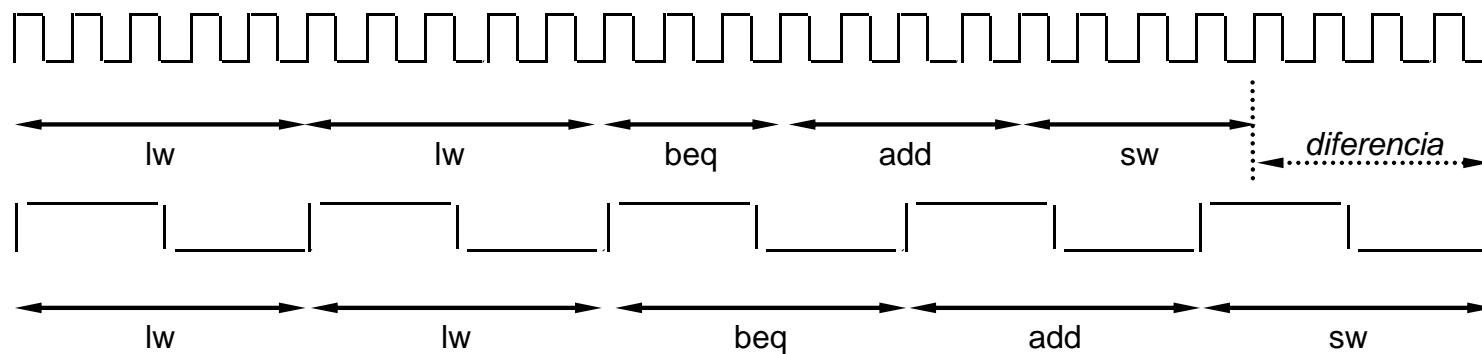
Comparación monociclo vs multiciclo

Asumiendo un comportamiento **ideal**:

- El procesador monociclo tarda 5ns en ejecutar una instrucción
- El procesador multiciclo divide la ejecución en 5 operaciones de 1 ns:
 - IF: Se lee la instrucción de la memoria de instrucciones
 - ID: se decodifica la instrucción y se leen los operandos del banco de registros. Se extiende el signo del operando inmediato
 - EX: Se utiliza la ALU para realizar los cálculos
 - MEM: Se lee o escribe en la memoria de datos
 - WB: Se escribe en banco de registros
- El procesador multiciclo necesita:
 - ciclos para instruccions Lw
 - ciclos para aritméticas
 - ciclos para Sw
 - para saltos tomados
 - para saltos no tomados

Comparación monociclo vs multiciclo

```
lw $t2, 0($t3)
lw $t3, 4($t3)
beq $t2, $t3, Label #asumir que no se salta
add $t5, $t2, $t3
sw $t5, 8($t3)
```



Comparación: monociclo vs. multiciclo

Operación	Frecuencia	Ciclos	CPI
tipo-R	50%	4	—
lw	20%	5	—
st	10%	4	—
beq (salta)	2.5%	4	—
beq (no salta)	17.5%	3	—

10⁶ instrucciones tardan en ejecutarse:
✓ $t_{multi} = 10^6 \cdot CPI_{multi} \cdot t_{multi} = 10^6 \cdot \underline{\hspace{2cm}} \cdot 1\text{ns}$
✓ $t_{mono} = 10^6 \cdot CPI_{mono} \cdot t_{mono} = 10^6 \cdot \underline{\hspace{2cm}} \cdot 5\text{ns}$

$$\boxed{=}$$

$$t_{multi} / t_{mono} = \underline{\hspace{2cm}} / \underline{\hspace{2cm}} = \underline{\hspace{2cm}}$$

Idealmente los programas tardan un
 $\underline{\hspace{2cm}} \% \text{ menos}$

en ejecutarse en el computador multiciclo

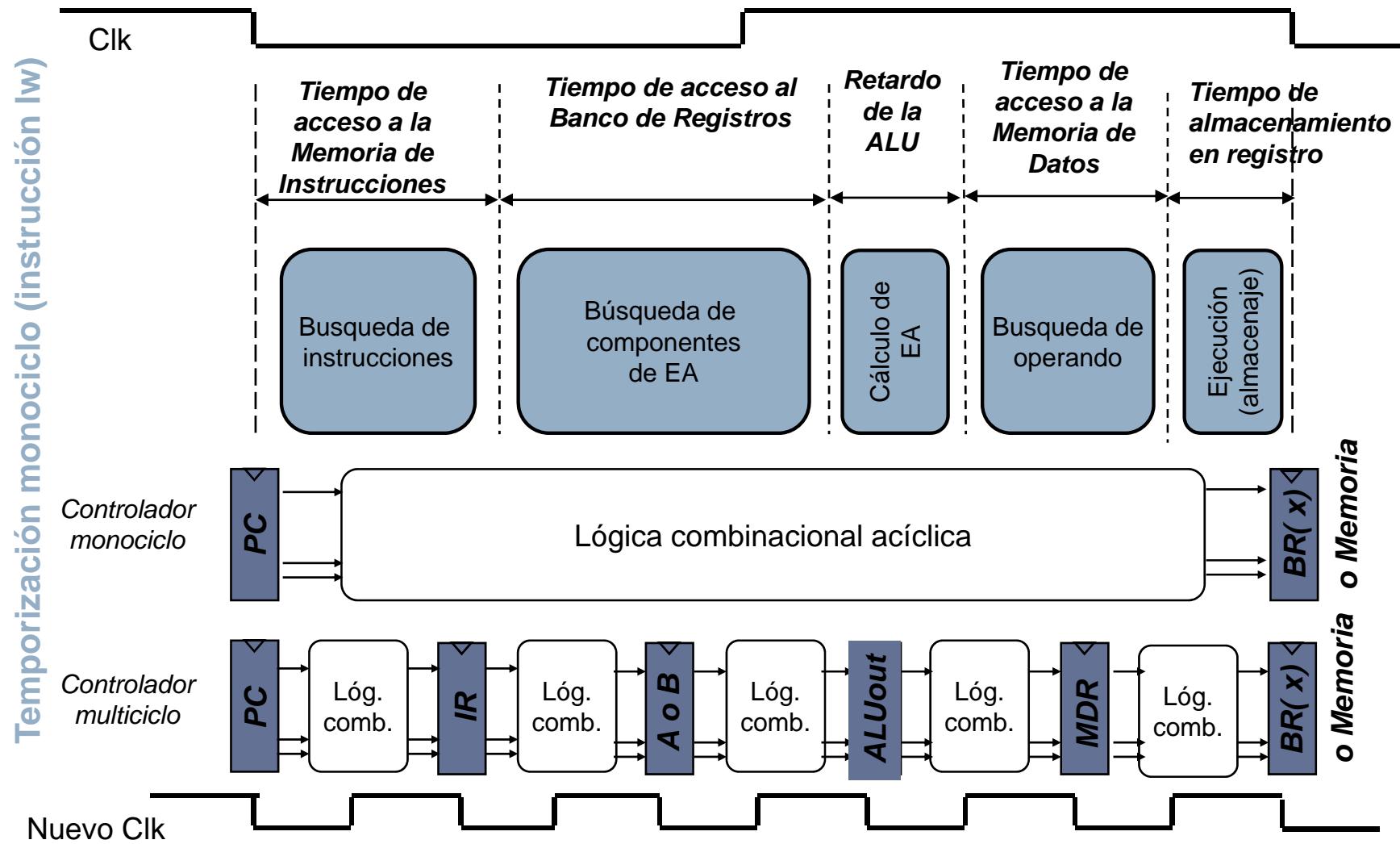
Pero en la práctica:

- No es posible dividir la ejecución en 5 etapas iguales
- Algunas etapas tendrán más retardo que otras
 - Reloj final marcado por la etapa más lenta
- Registros intermedios procesador multiciclo: retardos adicionales

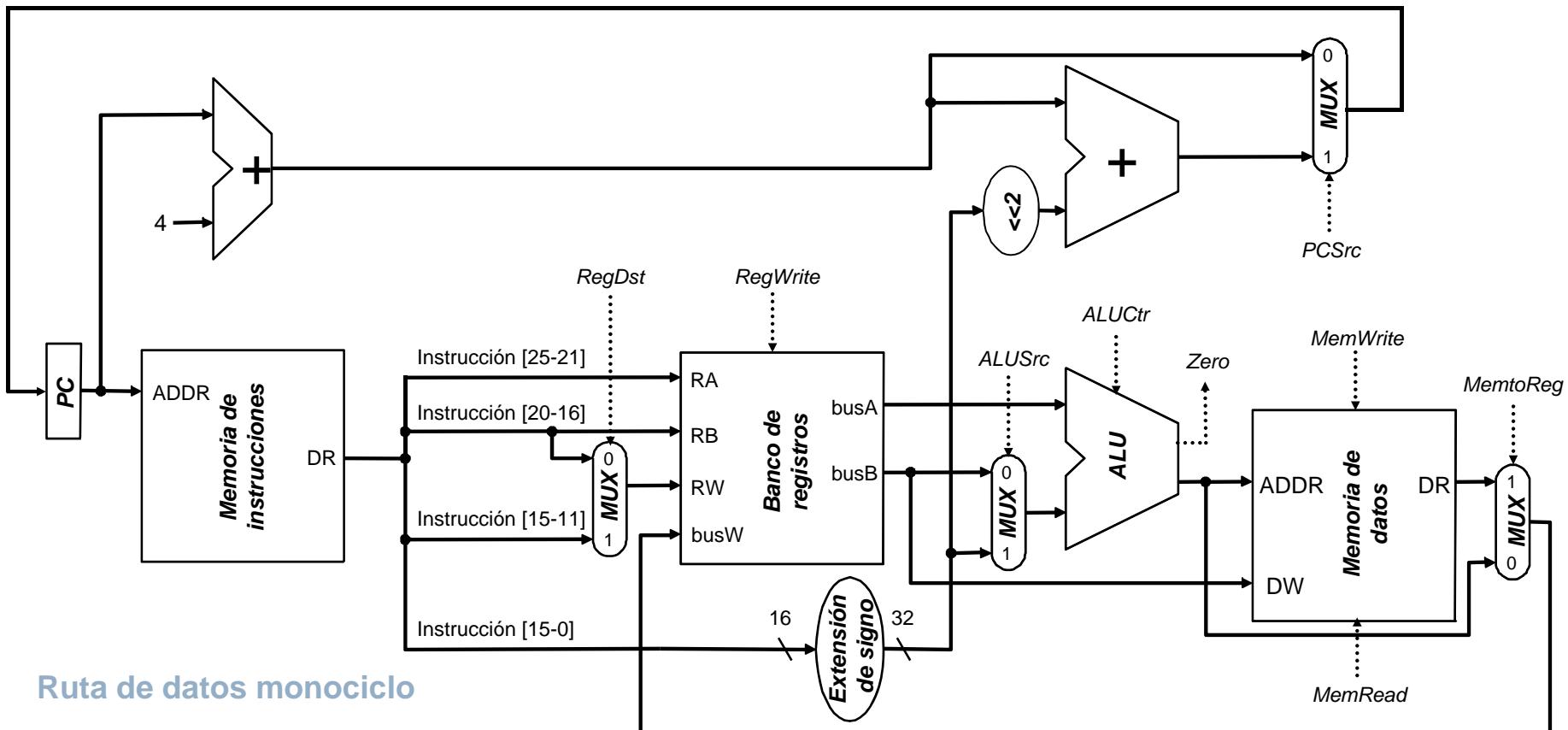
Comparación: monociclo vs. multiciclo

- **Ventajas monociclo:**
 - control más sencillo
 - No se necesitan almacenamientos intermedios
- **Desventajas monociclo:**
 - El reloj debe tener igual periodo que la instrucción más lenta
 - No se puede reutilizar hardware
- **Ventajas multiciclo:**
 - cada instrucción tarda el número de ciclos que sea necesario
 - podemos reutilizar un recurso HW en dos ciclos distintos
- **Desventajas multiciclo:**
 - El reloj debe tener igual periodo que la etapa más lenta
 - Se necesitan registros intermedios para que los datos generados en una etapa puedan usarse en las siguientes, esto conlleva un incremento del coste y del retardo.
 - Aumenta la complejidad del controlador

Temporización (multiciclo)



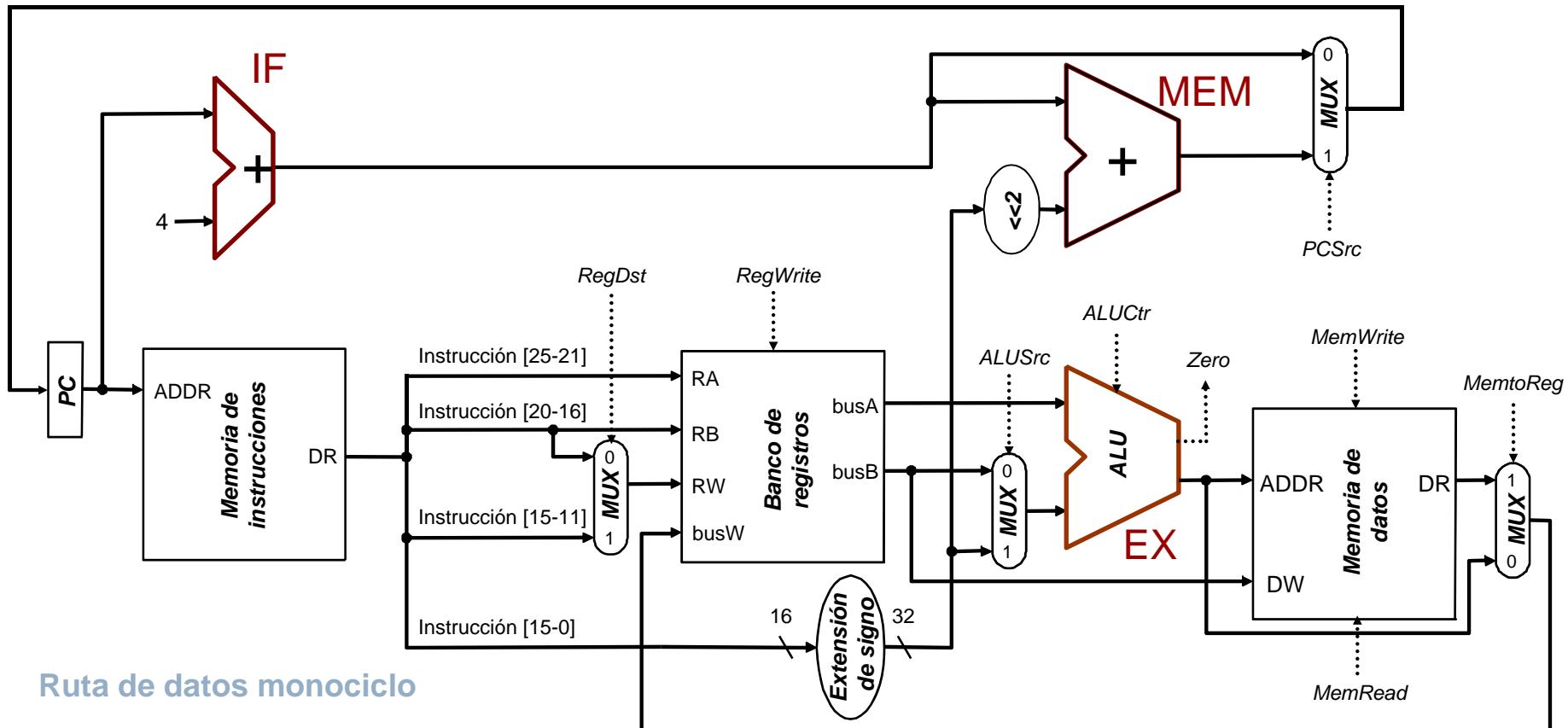
Diseño de la ruta de datos multiciclo



Podemos reutilizar hardware

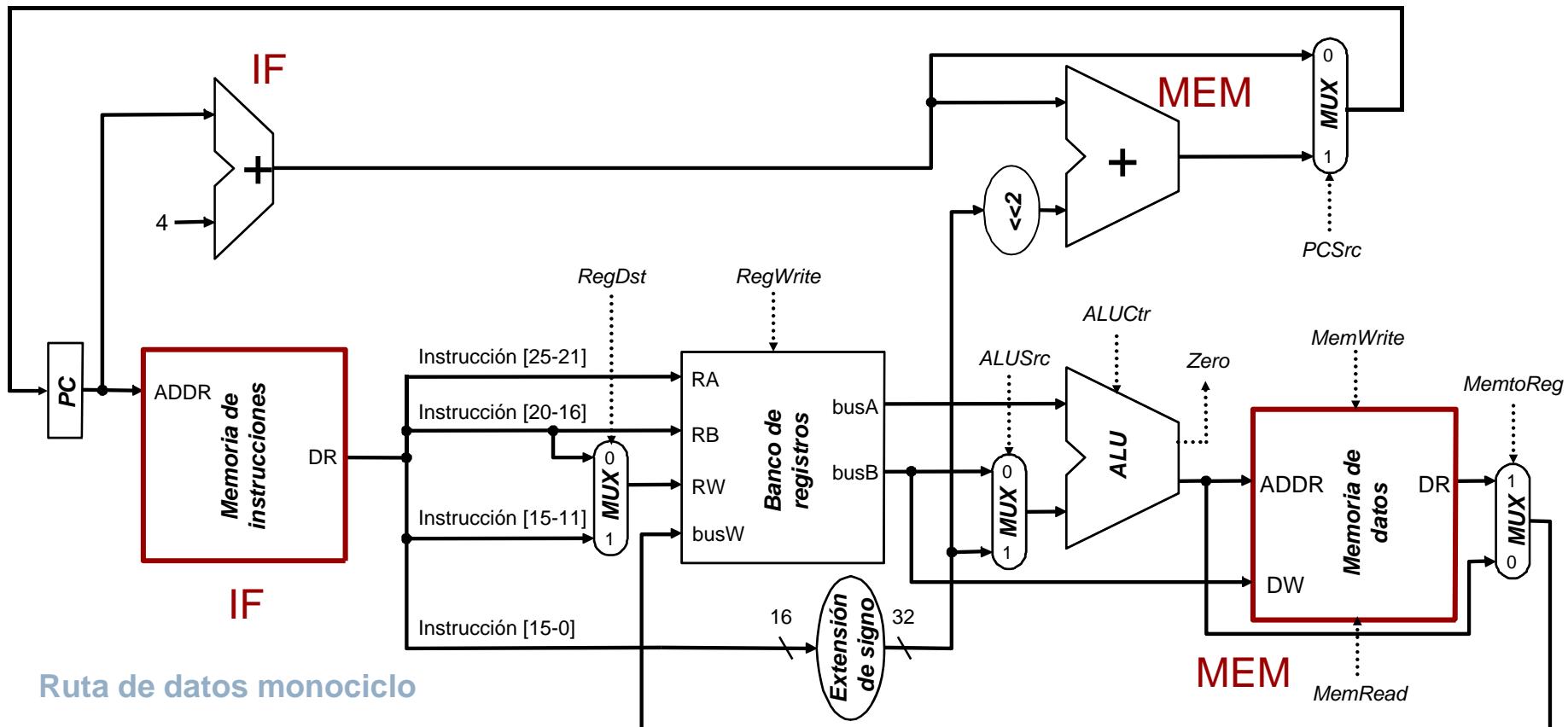
Hay que guardar todo aquello que se genere en un ciclo y se utilice más adelante

Diseño de la ruta de datos multiciclo: reutilización



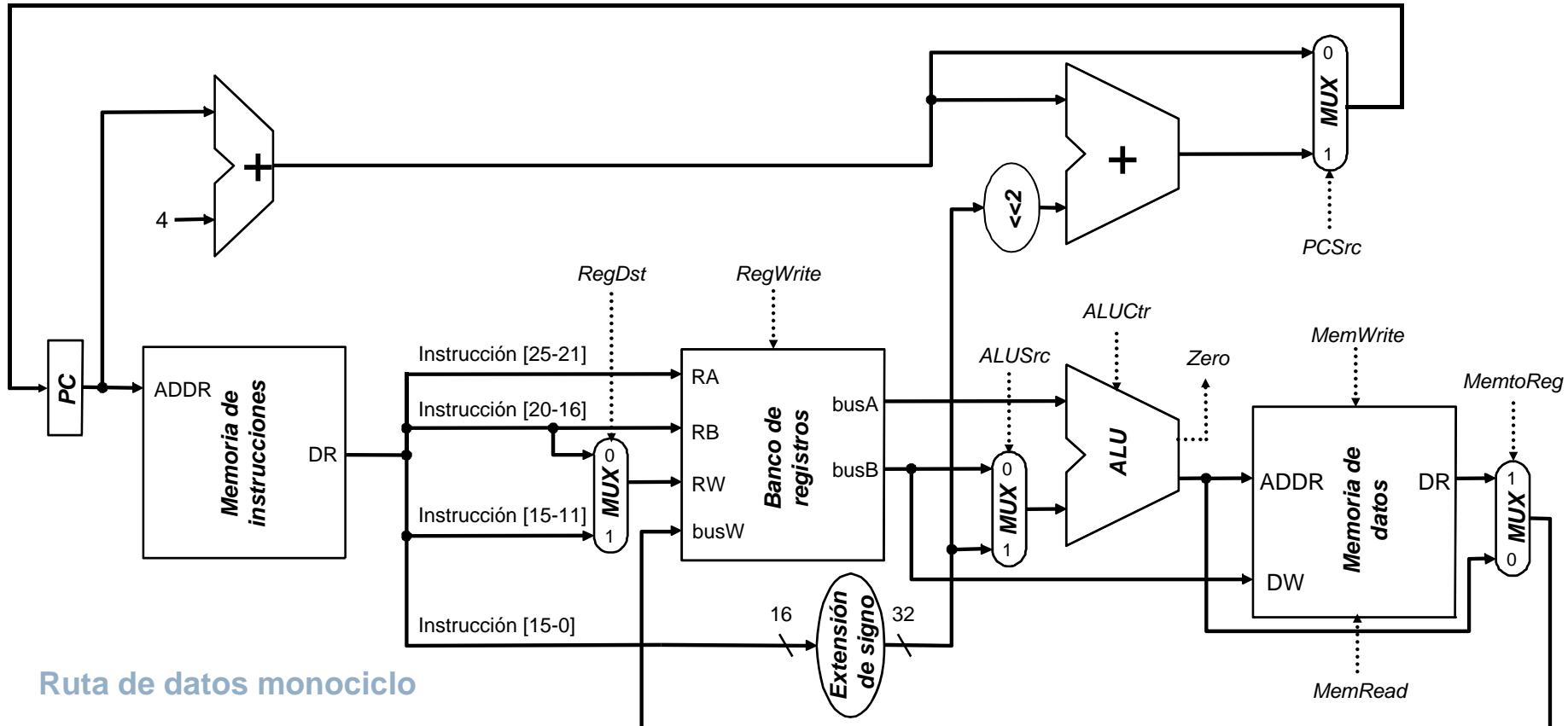
Las dos sumas las podría hacer la ALU siempre que se realicen en etapas distintas

Diseño de la ruta de datos multiciclo: reutilización



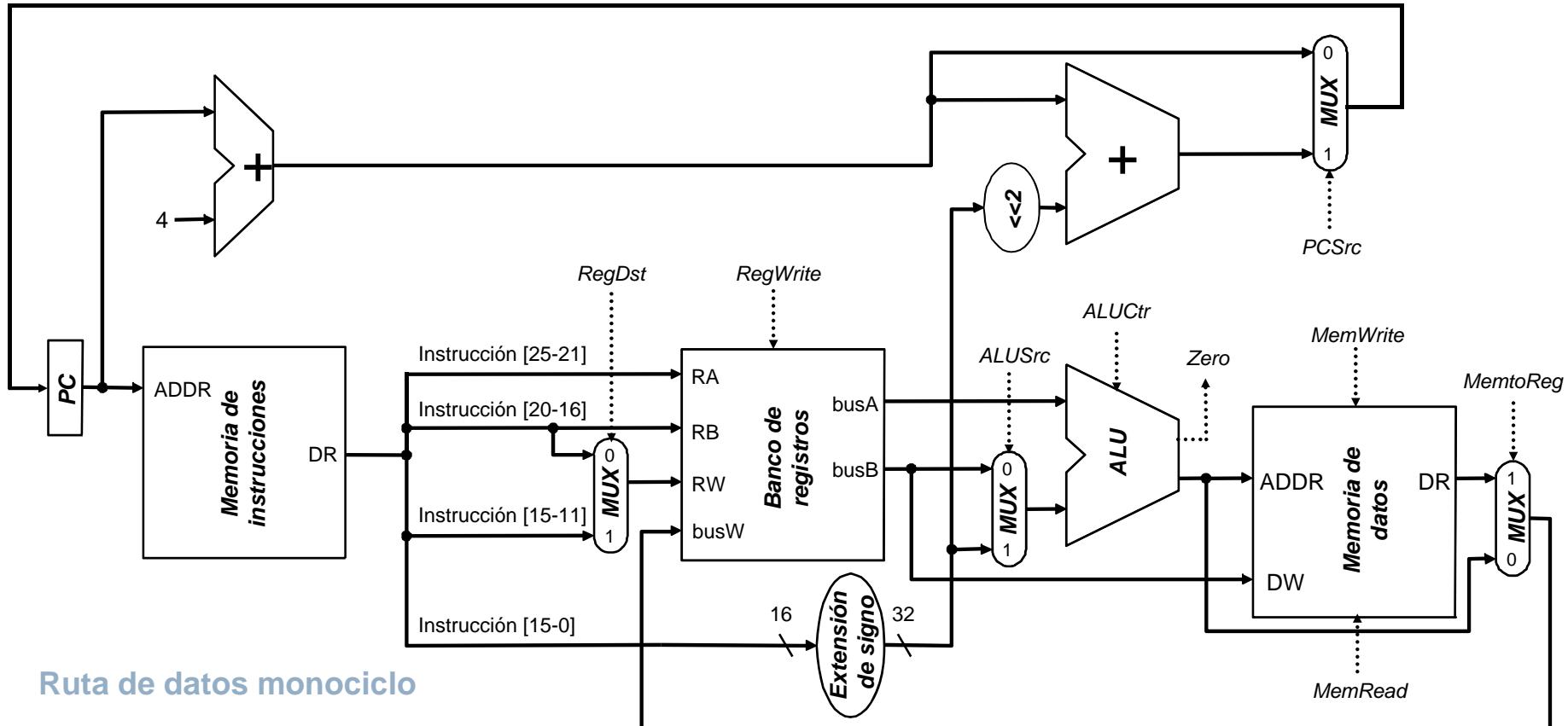
Ya no se accede en el mismo ciclo a la memoria de datos y a la de instrucciones:
Se puede utilizar una única memoria

Diseño de la ruta de datos multiciclo: registros intermedios



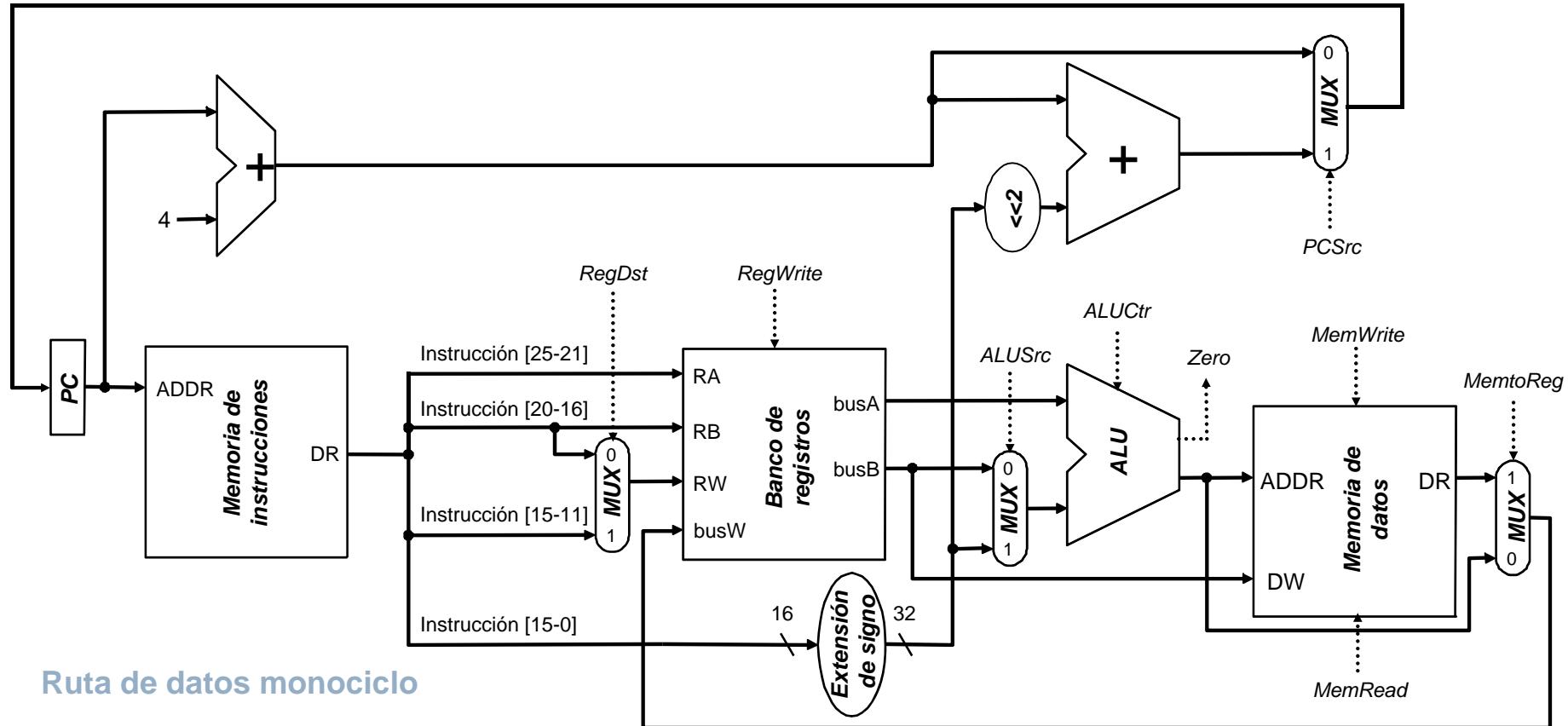
IF: leemos la instrucción de la memoria y hay que guardarla para utilizarla en las etapas posteriores: **registro IR (instruction register)**

Diseño de la ruta de datos multiciclo: registros intermedios



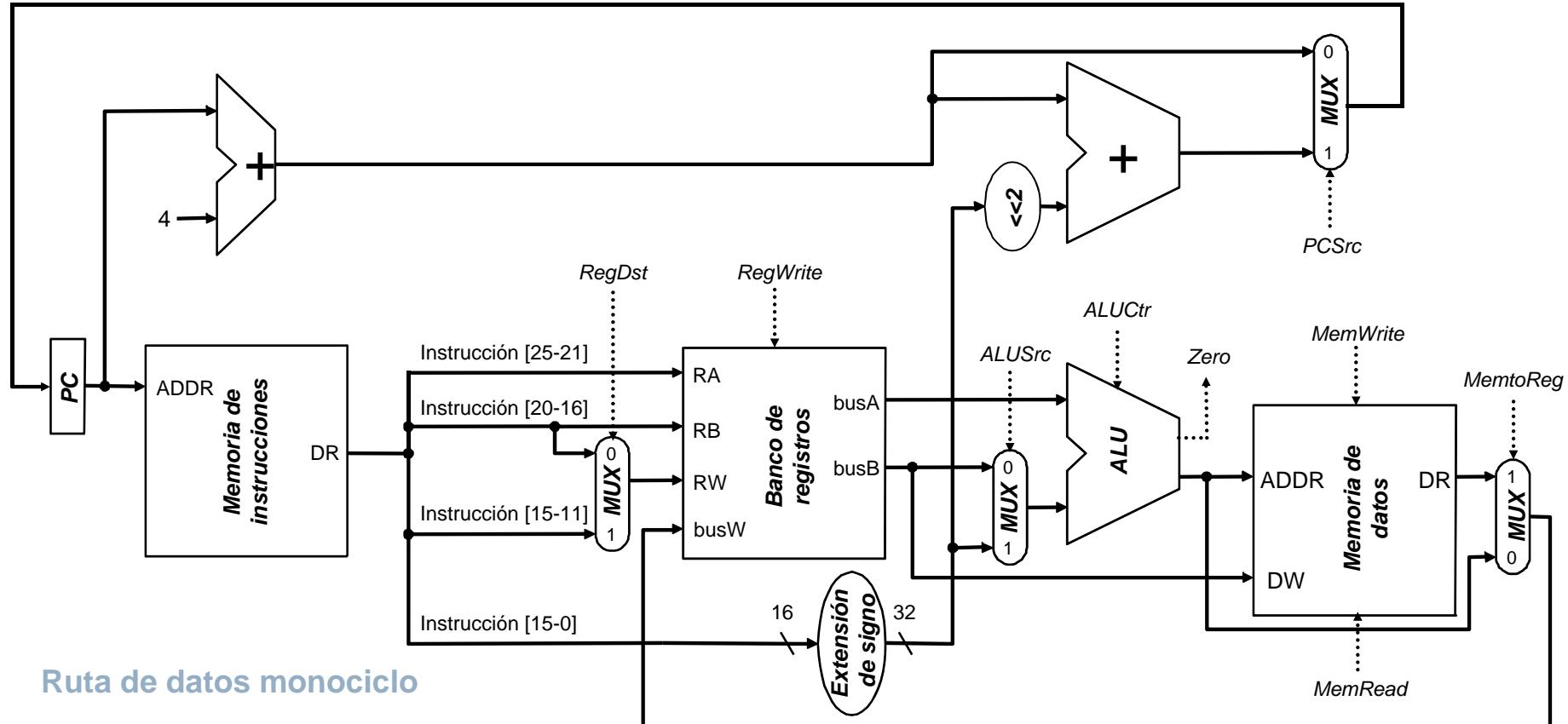
ID: leemos los operandos del banco de registros que se utilizarán en la etapa EX y
MEM: **registros A y B**

Diseño de la ruta de datos multiciclo: registros intermedios



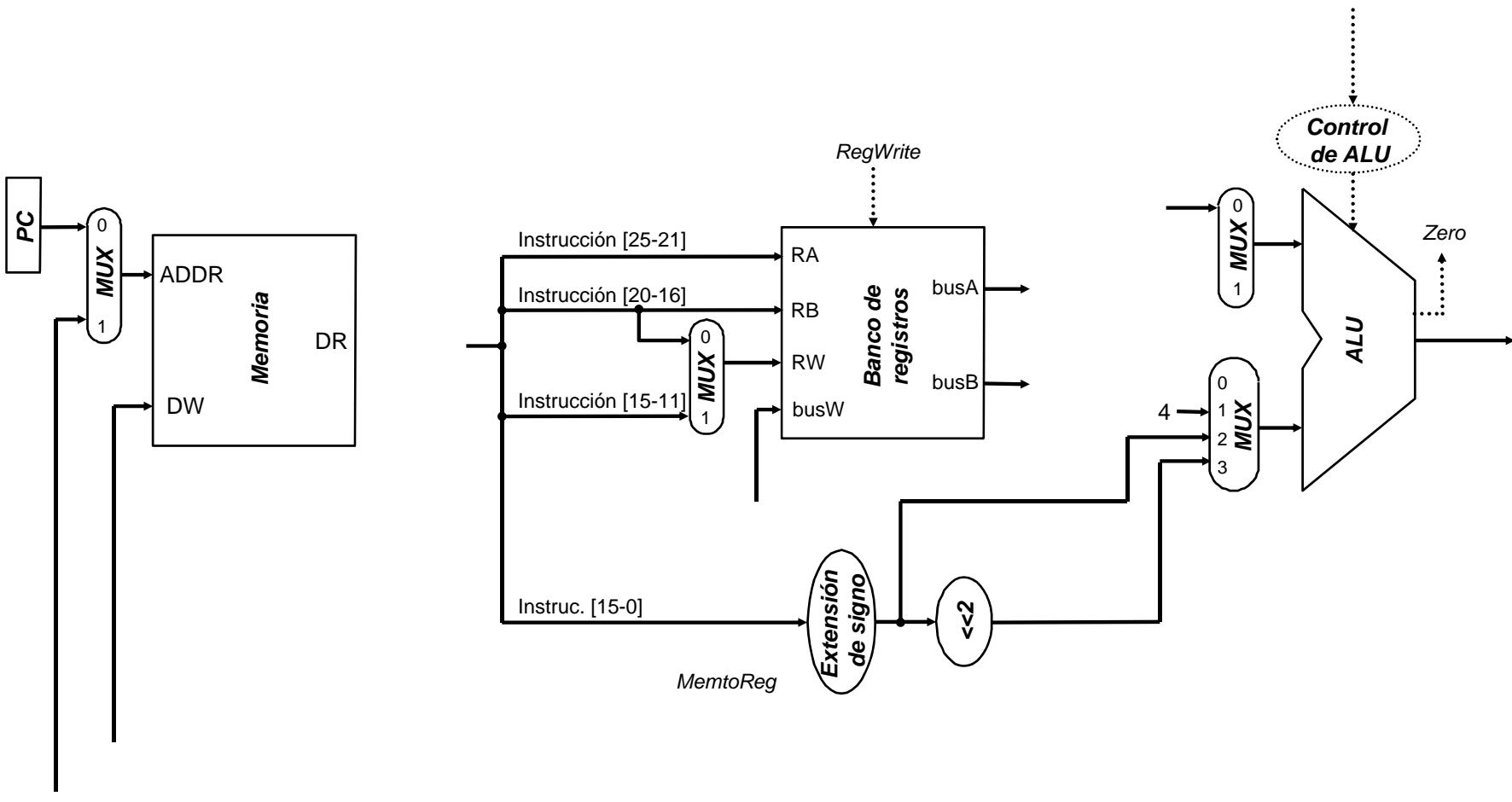
EX: Se realizan las operaciones con la ALU, el resultado se usa en los siguientes ciclos: **registro ALUout**

Diseño de la ruta de datos multiciclo: registros intermedios



MEM: en las instrucciones lw leemos un dato de memoria que se guardará en el ciclo siguiente en el banco de registros: **registro MDR (memory data register)**

Diseño de la ruta de datos (multiciclo)



Ruta de datos multiciclo

Diseño del controlador (multiciclo)

Transferencias entre registros “lógicas”

$BR(rt) \leftarrow \text{Memoria}(BR(rs) + \text{SignExt}(inmed))$, $PC \leftarrow PC + 4$

Transferencias entre registros “físicas”

1. _____ $\leftarrow \text{Memoria}(PC)$, $PC \leftarrow PC + 4$
2. _____ $\leftarrow BR(rs)$
3. _____ $\leftarrow A + \text{SignExt}(inmed)$
4. _____ $\leftarrow \text{Memoria}(ALUout)$
5. $BR(rt) \leftarrow _____$

Instrucción de carga (lw)

Transferencias entre registros “lógicas”

$\text{Memoria}(BR(rs) + \text{SignExt}(inmed)) \leftarrow BR(rt)$, $PC \leftarrow PC + 4$

Transferencias entre registros “físicas”

1. _____ $\leftarrow \text{Memoria}(PC)$, $PC \leftarrow PC + 4$
2. _____ $\leftarrow BR(rs)$, _____ $\leftarrow BR(rt)$
3. _____ $\leftarrow A + \text{SignExt}(inmed)$
4. $\text{Memoria}(ALUout) \leftarrow B$

Instrucción de almacenaje (sw)

Diseño del controlador (multiciclo)

Transferencias entre registros “lógicas”

$BR(rd) \leftarrow BR(rs) \text{ funct } BR(rt), PC \leftarrow PC + 4$

Transferencias entre registros “físicas”

1. _____ \leftarrow Memoria(PC), $PC \leftarrow PC + 4$
2. _____ \leftarrow $BR(rs)$, _____ \leftarrow $BR(rt)$
3. _____ \leftarrow _____ funct _____
4. $BR(rd) \leftarrow$ _____

Instrucción aritmética (tipo-R)

Transferencias entre registros “lógicas”

si ($BR(rs) = BR(rt)$) entonces $PC \leftarrow PC + 4 + 4 \cdot \text{SignExt}(inmed)$

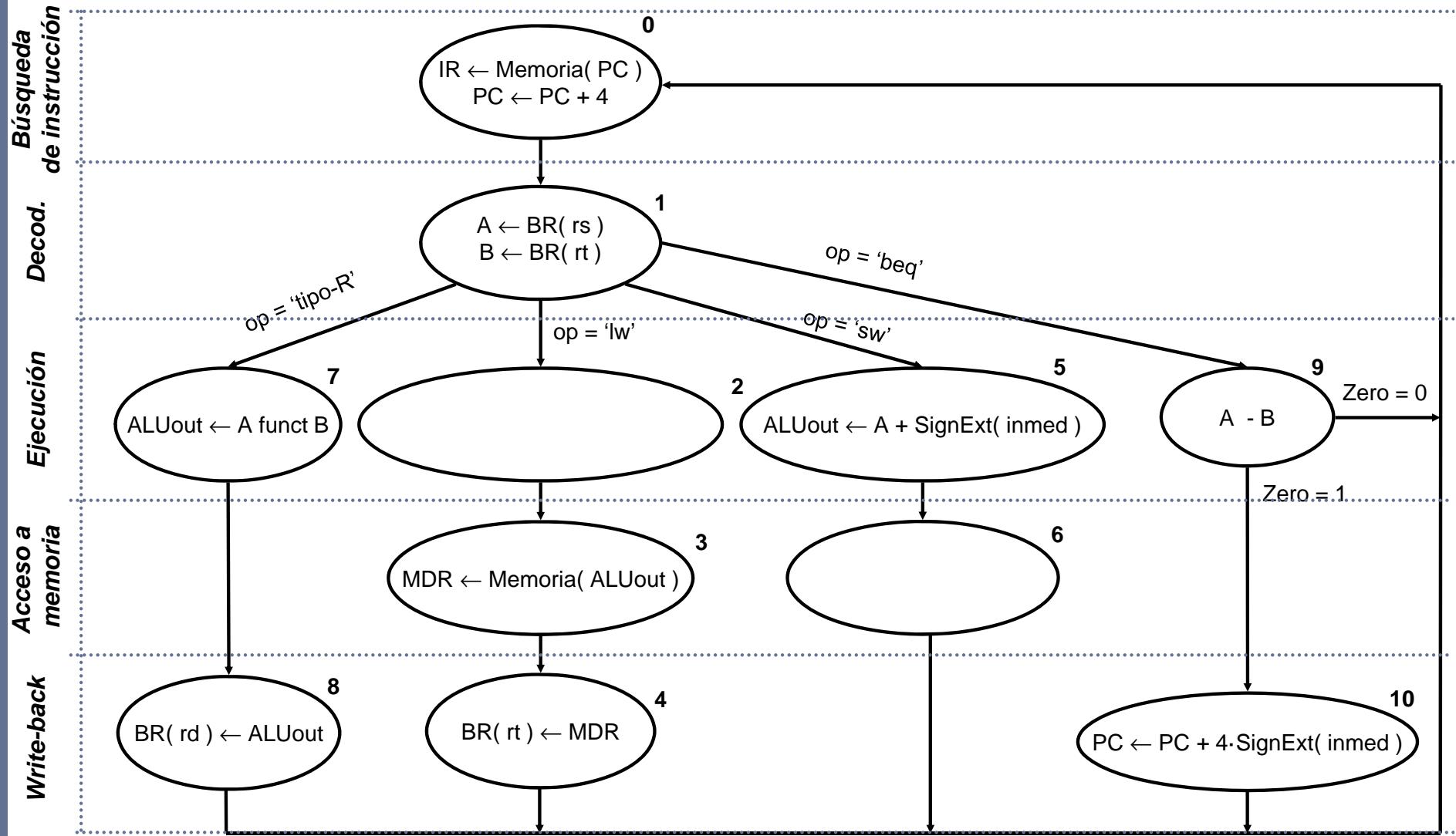
sino $PC \leftarrow PC + 4$

Transferencias entre registros “físicas”

1. _____ \leftarrow Memoria(PC), $PC \leftarrow PC + 4$
2. _____ \leftarrow $BR(rs)$, _____ \leftarrow $BR(rt)$,
3. _____ - _____
4. si Zero entonces $PC \leftarrow PC + 4 \cdot \text{SignExt}(inmed)$

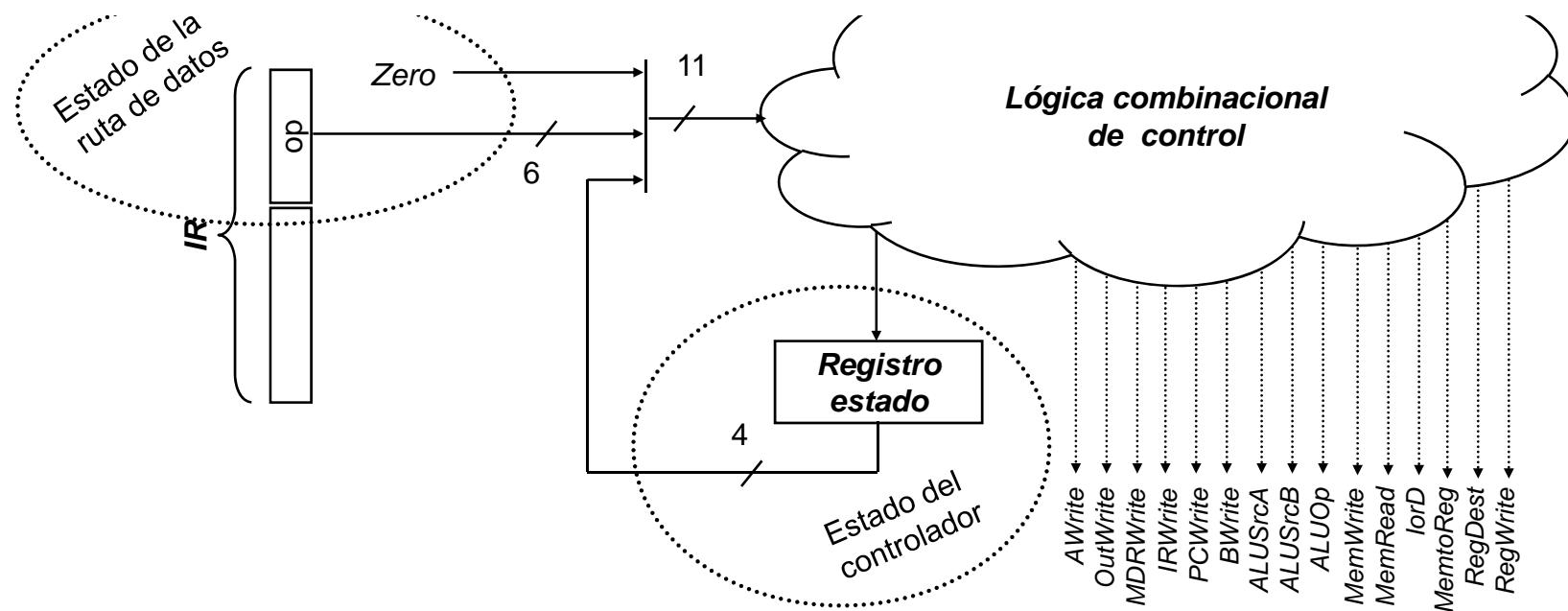
Instrucción de salto condicional (beq)

Diseño del controlador (multiciclo)



Diseño del controlador (multiciclo)

1. Se **traducen** las transferencias entre registros como **conjuntos de activaciones** de los puntos de control de la ruta de datos
2. Se **codifican** los estados
3. Mediante **tablas de verdad** se describen:
 - ✓ las **transiciones de estado** en función del código de operación y del estado de la ruta de datos
 - ✓ el **valor de las señales** de control en función del estado (controlador tipo Moore) y adicionalmente en función del estado de la ruta de datos (controlador tipo Mealy).
4. La **estructura del controlador** estará formada por:
 - ✓ **Registro de estado**
 - ✓ Conjunto de **lógica combinacional de control** que implementa las anteriores tablas



Conclusiones

- El procesador monociclo no es adecuado para repertorios de instrucciones muy heterogéneos
- El procesador multiciclo puede mejorar sensiblemente el rendimiento con respecto al monociclo porque:
 - permite utilizar un reloj más rápido
 - y adaptar el tiempo que de ejecución de cada instrucción a sus necesidades:
 - Instrucciones sencillas se ejecutan en pocos ciclos
 - Instrucciones complejas requieren un número de ciclos elevado
- Pero las ganancias obtenidas suelen estar lejos de las máximas teóricas porqué:
 - El reloj debe tener igual periodo que la etapa más lenta
 - Se necesitan registros intermedios que incrementan el retardo.
- En cuanto al área:
 - Permite reutilizar elementos hardware (como la ALU)
 - Para ello normalmente hay que añadir algunos multiplexores
 - Por otro lado es necesario añadir almacenamiento intermedio para todos los datos que se obtienen en un ciclo y se usan más adelante
 - La unidad de control es más compleja



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza



Tema 3

Rendimiento y modelos de ejecución

Arquitectura y Organización de Computadores 2
2º curso, Grado en Ingeniería Informática

Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

Guión del tema

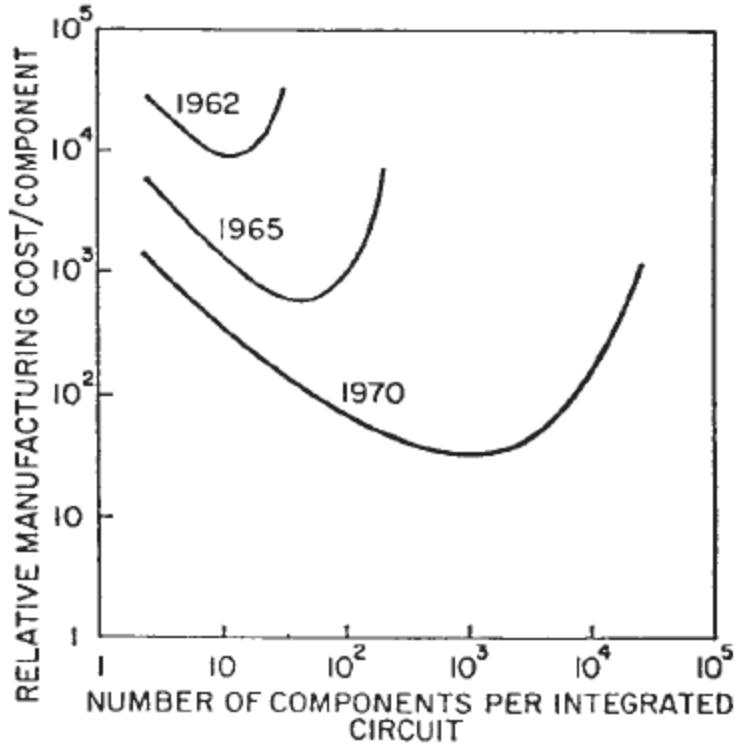
- Tendencias
- Medida del rendimiento
- Incremento del rendimiento por diseño: evolución de los modelos de ejecución

Tendencias en uso y tecnologías del computador

demanda (software) ↔ oferta (tecnología)

- Demanda
 - Tamaño de los programas en ejecución
 - Número de instrucciones a ejecutar
- Oferta: ley de Gordon E. Moore (co-fundador de Intel)
 - Procesador y memoria cache
 - Memoria DRAM
 - Disco Magnético

Ley de Moore



Pasa el tiempo, los programas crecen en tamaño

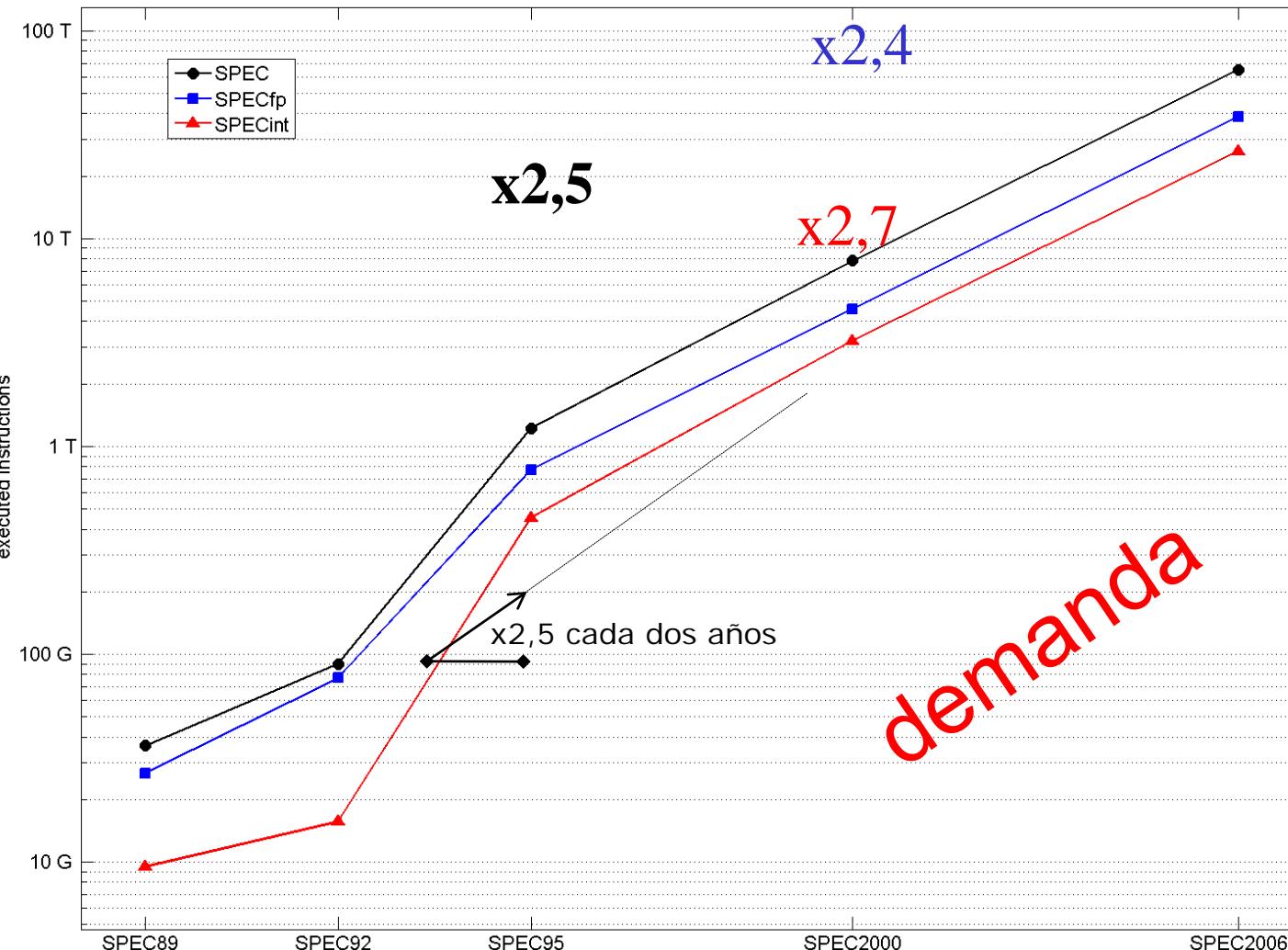
- Tamaño procesos (inst. + datos)
 - x1,5 - 2 cada año
 - 0,5 - 1 bit más de direccionamiento cada año
- Por qué?
 - Incremento de funcionalidad y/o precisión
- Primera ley del Software
 - Nathan Myhrvold (top-executive de Microsoft)

demanda

El software es un gas.
Se expande hasta ocupar todo el
recipiente que lo contiene.

veamos ...

Pasa el tiempo, los programas ejecutan más instrucciones



Actualización de: J. Alastruey, J.L. Briz, P. Ibáñez y V. Viñals

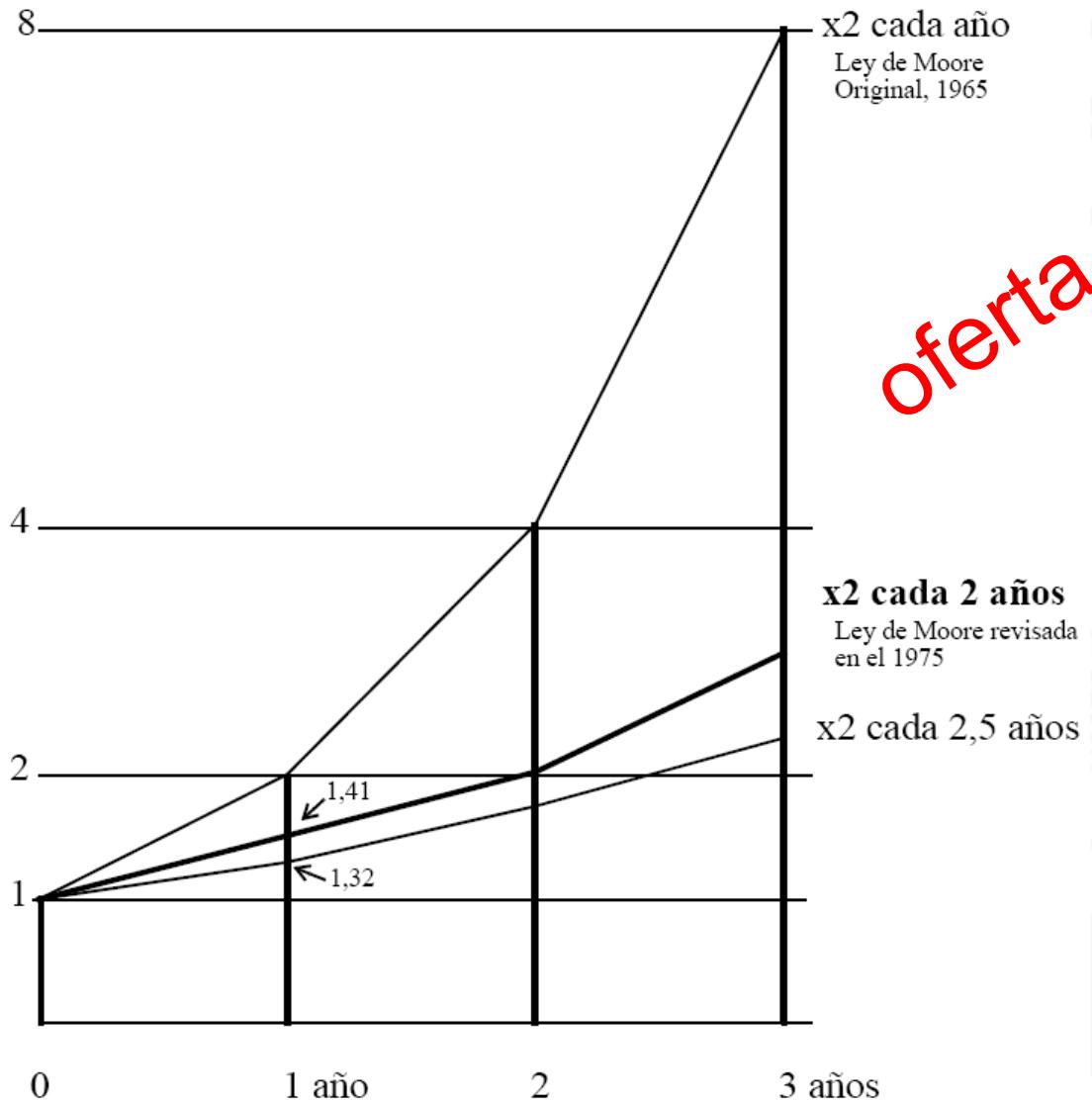
"Software Demand vs. Hardware Supply: SPEC CPU and processor resources"

IEEE Micro, IEEE Computer Society 2006

AOC2 - Tema 3

- Se diseña la arquitectura pensando en el compilador o en el intérprete
 - Vectorización y paralelización automática de programas secuenciales (C, C++, Fortran, ...)
- para ejecutar en los micros actuales (CMPs)

Ley de Moore: nº de transistores/chip vs. tiempo (1/4)



Ejemplo: familia Intel

procesador	introd.	#Trans.
4004	Abr71	2,3K
8008	Abr72	3,5K
8080	Abr74	6K
8086	Jun78	29K
8088	Jun79	29K
80286	Feb82	134K
80386	Oct85	275K
80486	Abr89	1.2M
Pentium	Mar93	3.1M
Pentium Pro	Mar 95	5.5M
Pentium II	May 97	7,5M
¿ Ley de Moore ?		

¹ L. Geppert, "The Media Event: Moore's Law Mania", *IEEE Spectrum*, Enero 1998, pp. 20:21.

Ley de Moore con datos y predicciones de consenso (2/4)

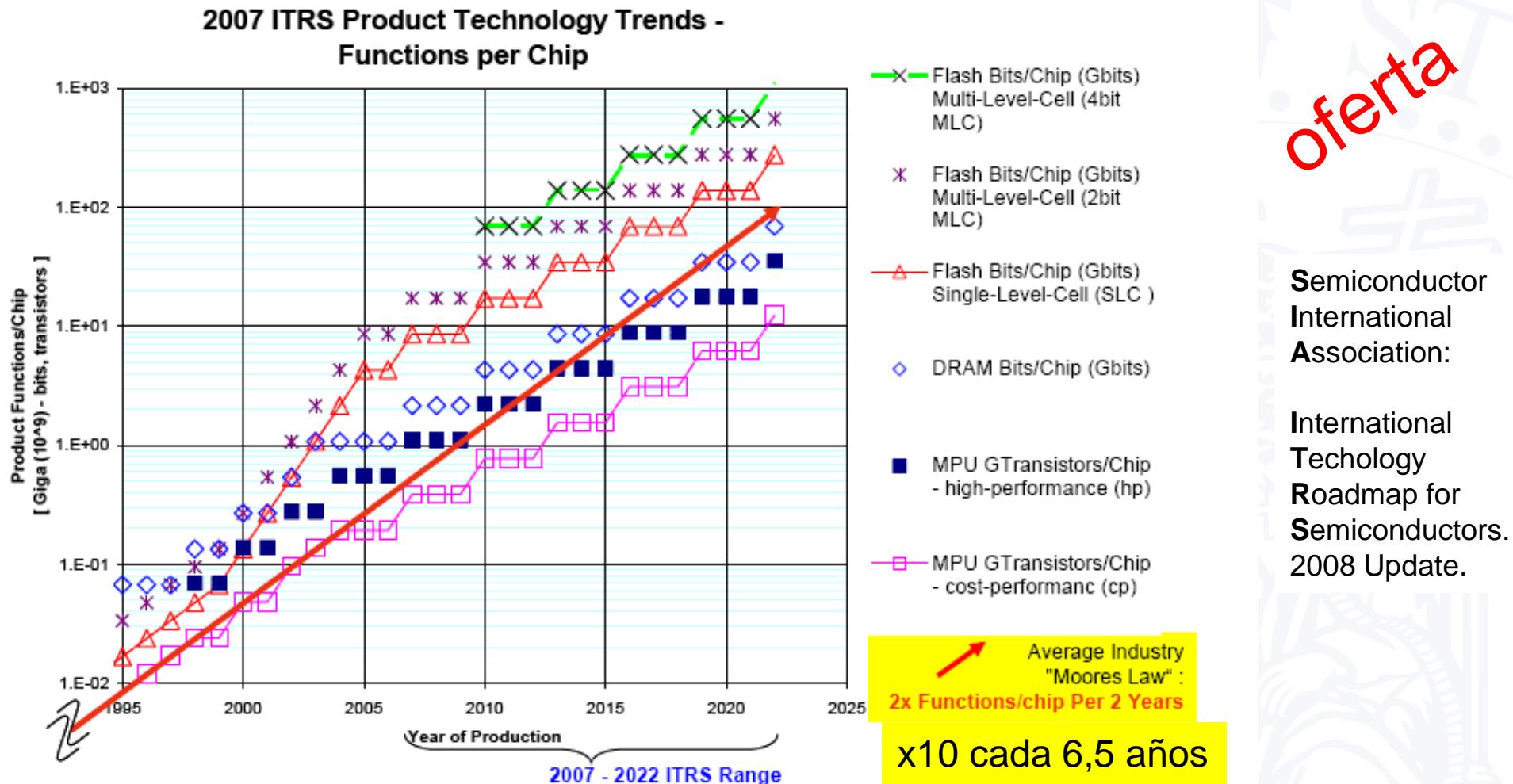
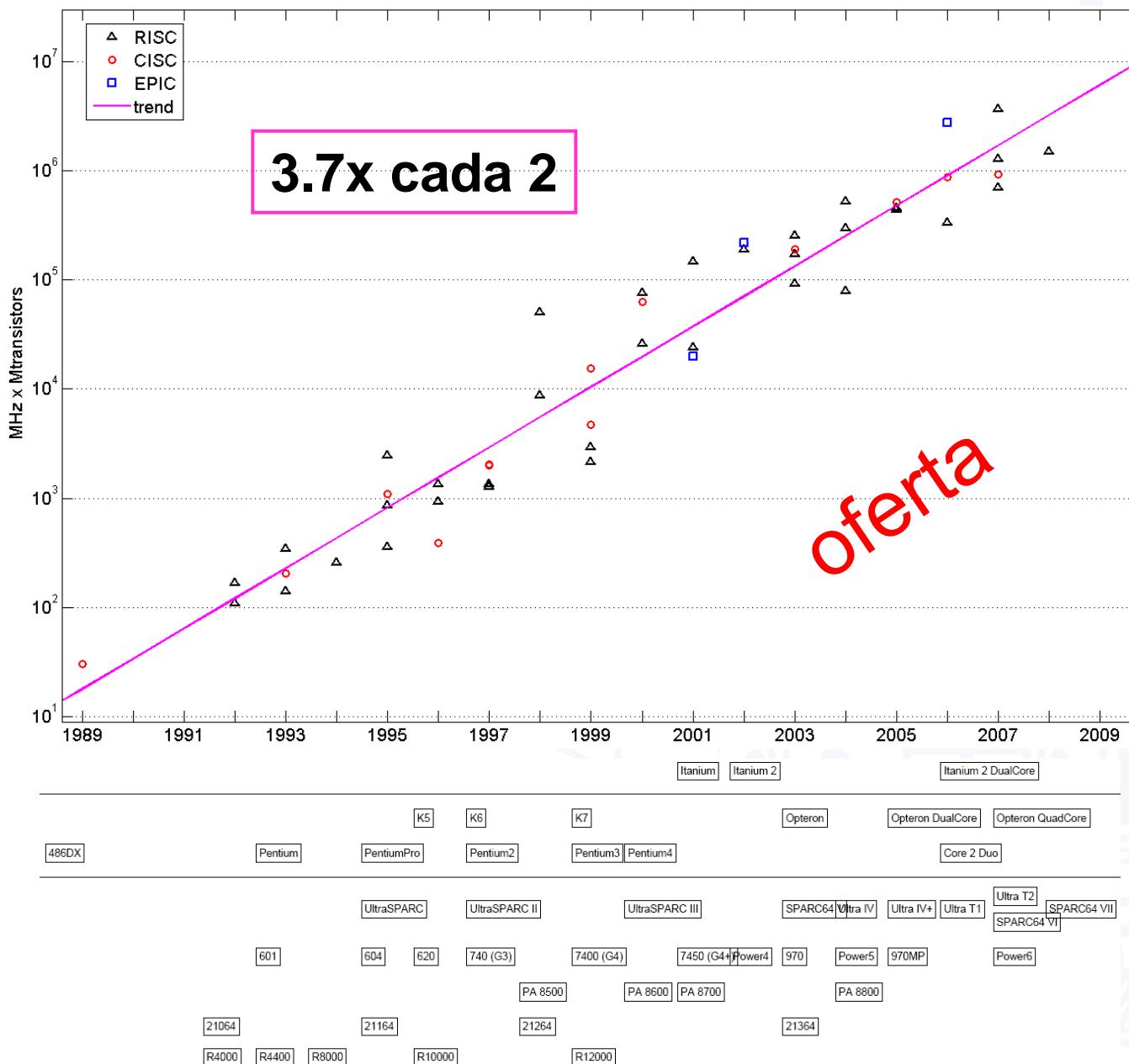


Figure ORTC2 ITRS Product Function Size Trends:

MPU Logic Gate Size (4-transistor); Memory Cell Size [SRAM (6-transistor); Flash (SLC and MLC), and DRAM (transistor + capacitor)] --Updated

Ley de More y aumento de frecuencia: Transfreq (3/4)

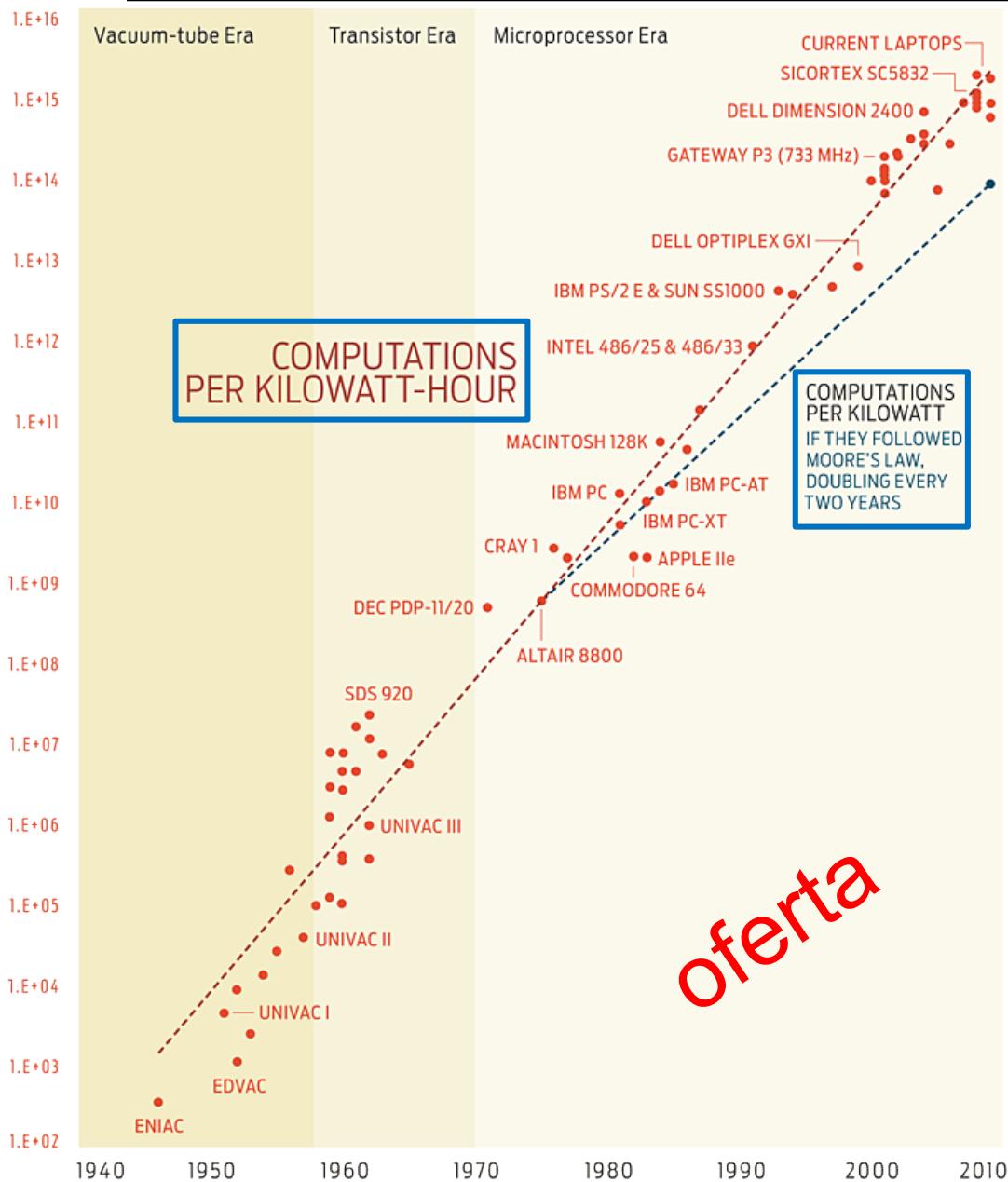


Transfreq =
transistor count
x
clock frequency

Tiene sentido este
producto ?
Mtransistores x Mhz

Actualización de:
J. Alastrauey, J.L. Briz, P. Ibáñez
y V. Viñals. "Software Demand
vs. Hardware Supply: SPEC
CPU and processor resources".
IEEE Micro, IEEE Computer
Society 2006.

Ley de Moore y eficiencia energética (4/4)



Outperforming Moore's Law

The number of transistors on a chip has doubled about every two years for decades, a trend that is popularly (but often imprecisely) encapsulated as Moore's Law. What is less well known is that the electrical efficiency in delivering computing performance began following a similar trend long before the microprocessor was invented.

The performance of desktop computers has doubled every 1.5 years, on average, since 1975, and in that time the number of computations per kilowatt-hour has grown about as quickly. But that measure of efficiency had also grown about that fast during the previous three decades, and even more rapidly during the vacuum-tube computing era and during the transition from tubes to transistors.

These are just a few of the conclusions drawn from a new study, "Assessing Trends in the Electrical Efficiency of Computation Over Time," prepared by me and my colleagues, researchers affiliated with Intel Corp., Lawrence Berkeley National Laboratory, Microsoft Corp., and Stanford University.

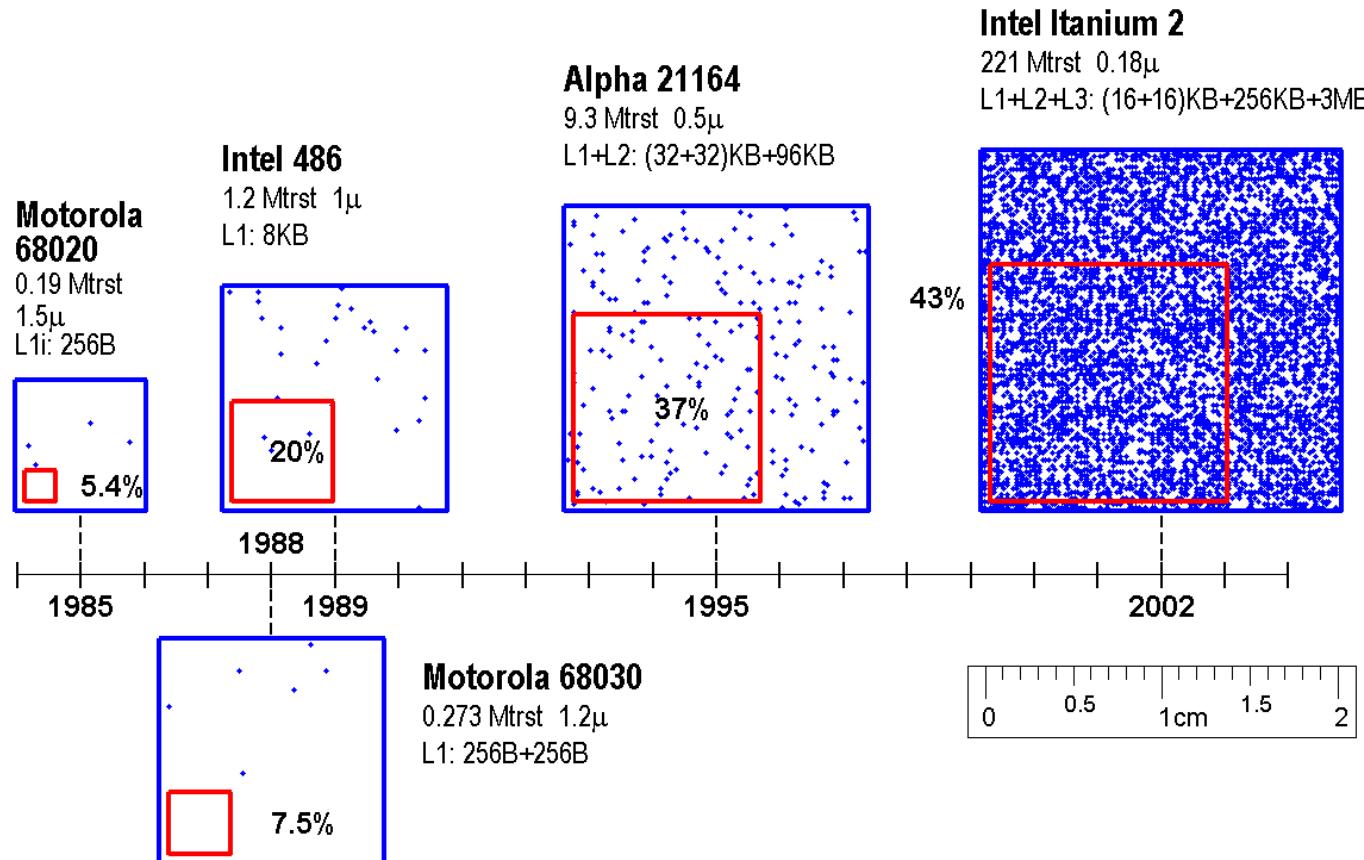
The main technology trends that have improved performance and reduced costs—at first better tubes, and then smaller transistors—also reduce power use, hence the similar improvements in computational performance and electrical efficiency, at similar rates, for such a long time. If these trends continue—and we have every reason to believe they will for at least the next 5 to 10 years—we can expect continued rapid reductions in the size and power requirements of computer-based devices. That should be especially welcome news for users of netbooks, smartphones, cameras, and other mobile devices.

Jonathan G. Koomey, IEEE Spectrum March, 2010, p. 56

Procesador y Memorias Cache

■ Integración de lógica y memoria rápida (registros, SRAM para Mc)

- nº transistores/chip: x2 cada 2-3 años (muy al principio, x8 cada 3 !)
 - ◆ densidad transistores: a este ritmo
 - ◆ superficie chip: estabilizada
 - ◆ velocidad conmutación: ≈ a este ritmo
 - ◆ retardo “RC” en metal: ≈ fijo



oferta

J. Alastruey, J.L. Briz, P. Ibáñez y V. Viñals
“Software Demand vs. Hardware Supply: SPEC CPU and processor resources”. *IEEE Micro*, IEEE Computer Society 2006

Memoria DRAM: capacidad y coste

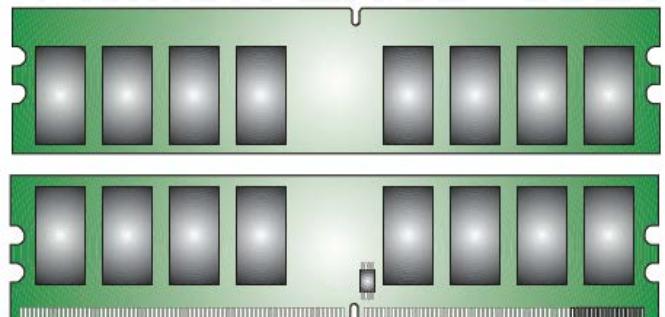
Oferta

■ nº bits/chip

- ..-2004: x2 cada 1.5 años
- 2000-04: x2 cada 2 años
- 2004-22: x2 cada 2-3 años
- Último hito: 4 gigabits en 2009 (Samsung, 1Q09)
 - ◆ Jun-2011: sample shipments of 8Gb DDR3 Chip by Elpida.
Based on 3D stacking (TSV, Through Silicon Via Technology)

■ Coste/chip

- No cambia demasiado
- Feb-2009
 - ◆ Módulo de 2 GB DDR2-800
 - 16 chips x 1 gigabit: ~ 20 €
 - ◆ Módulo de 4 GB DDR2-800
 - 16 chips x 2 gigabit: ~ 200 €

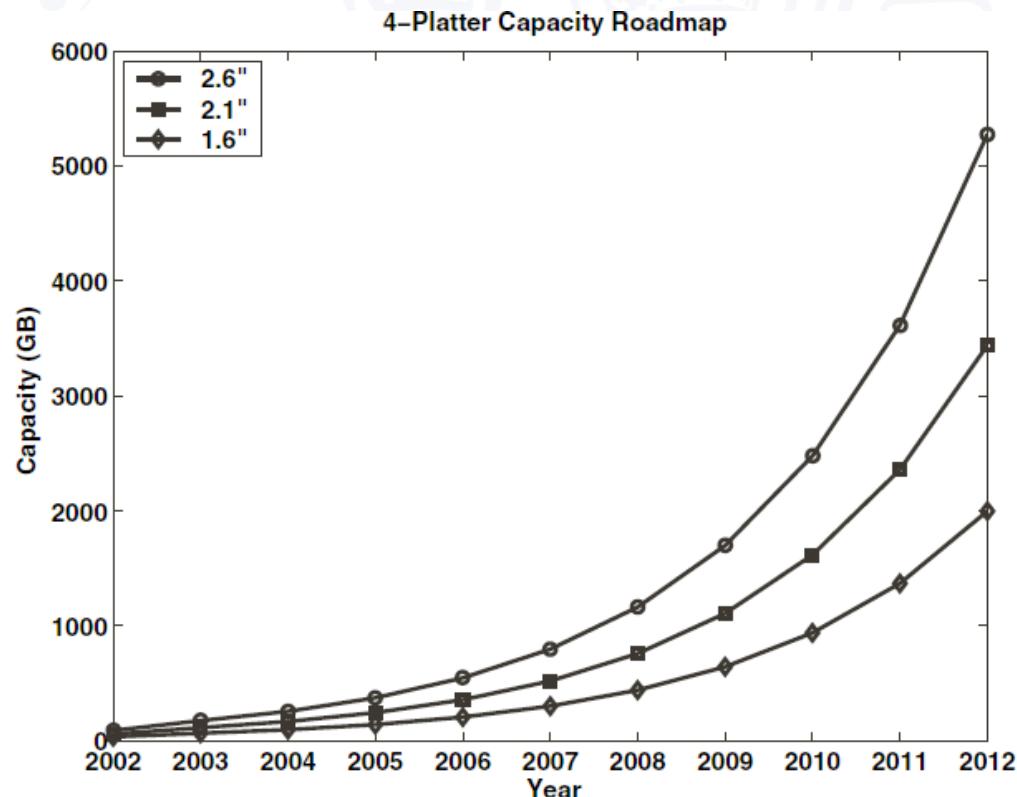
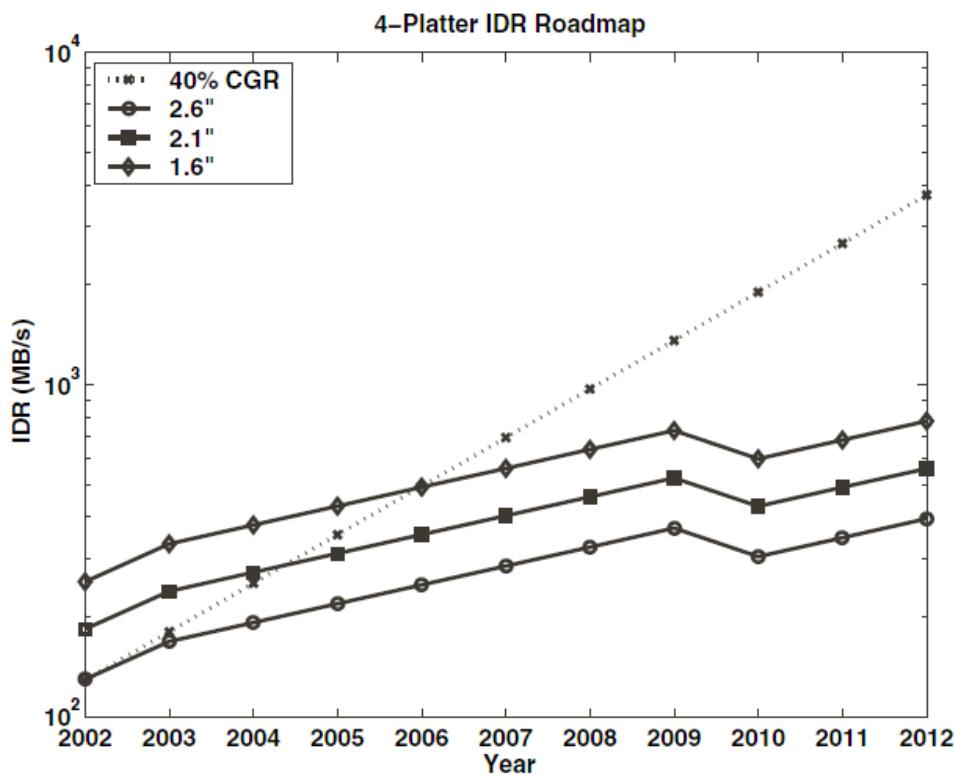


Disco magnético

- nº bits/mm²
 - x1,25 hasta el 90
 - x1,5 ahora
- tiempo de acceso
 - x0,94 cada año

oferta

Gurumurthi et al. "Disk Drive Roadmap from the Thermal Perspective: A Case for Dynamic Thermal Management". *Int. Symp. on Computer Architecture* (ISCA), 2005

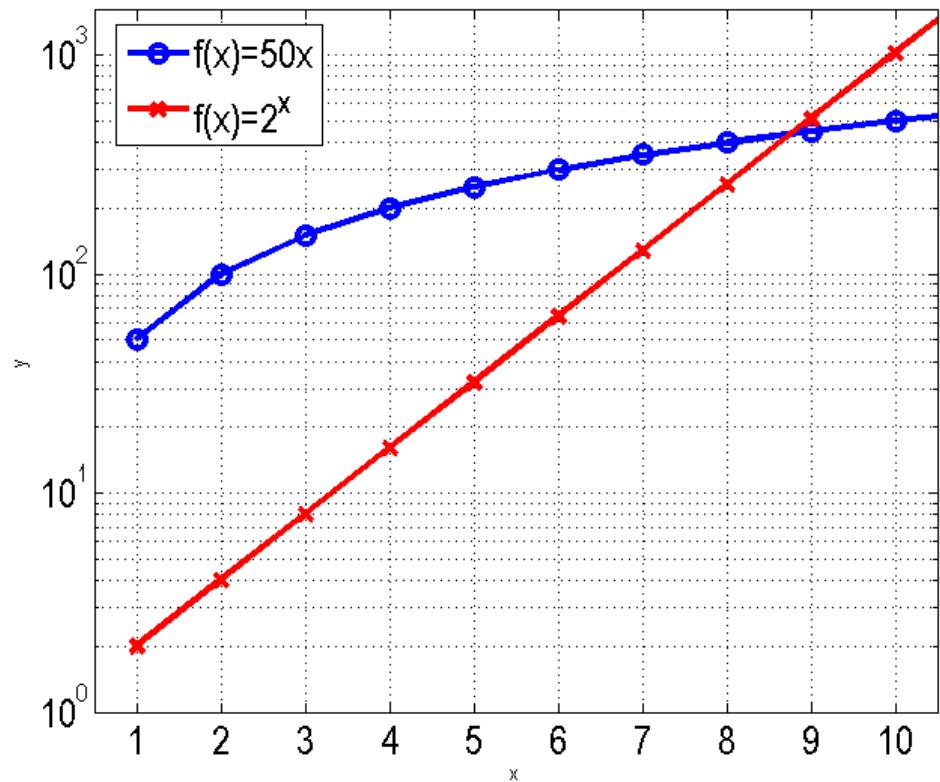
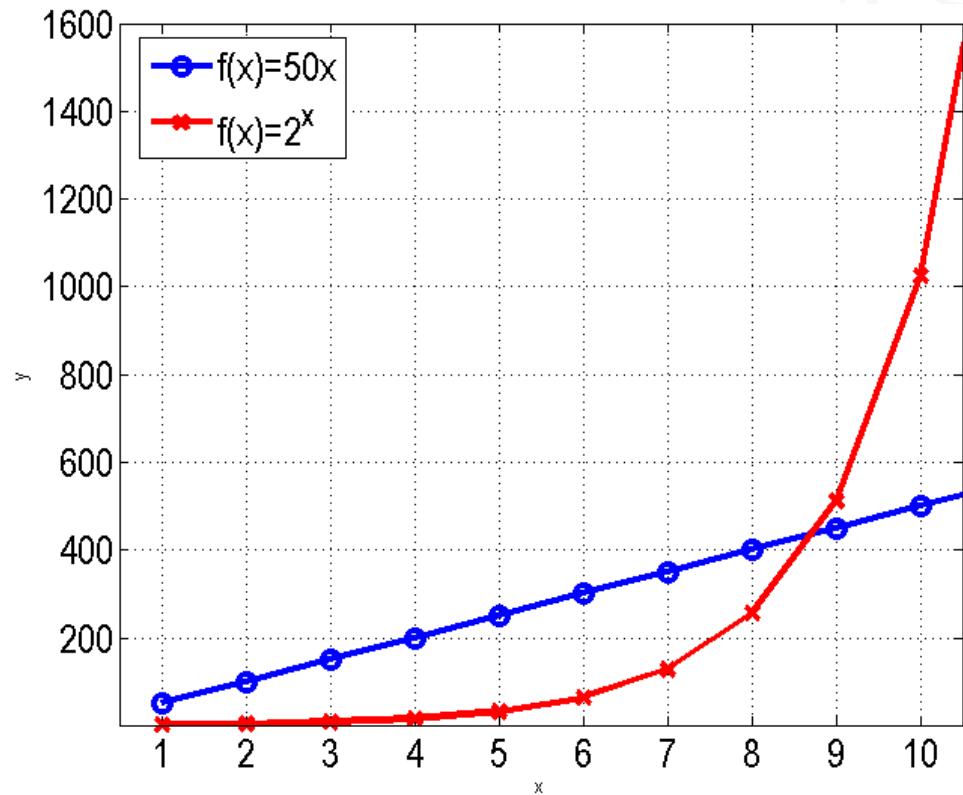


Crecimientos y medida del Rendimiento

- Crecimiento exponencial
 - formulación y ejemplo
 - casos de estudio de evolución temporal (1988-2001)
 - ◆ tiempo de ciclo del procesador
 - ◆ tiempo de acceso a fila de la memoria DRAM
- Rendimiento: tiempo, velocidad y consumo
- Ley de Amdhal

Crecimientos

- Lineal: el sistema no guarda memoria
 - Kilómetros de autopistas construidos por año
- Exponencial: cada valor se calcula a partir del anterior
 - Dinero en un banco, cuando los intereses se añaden al capital

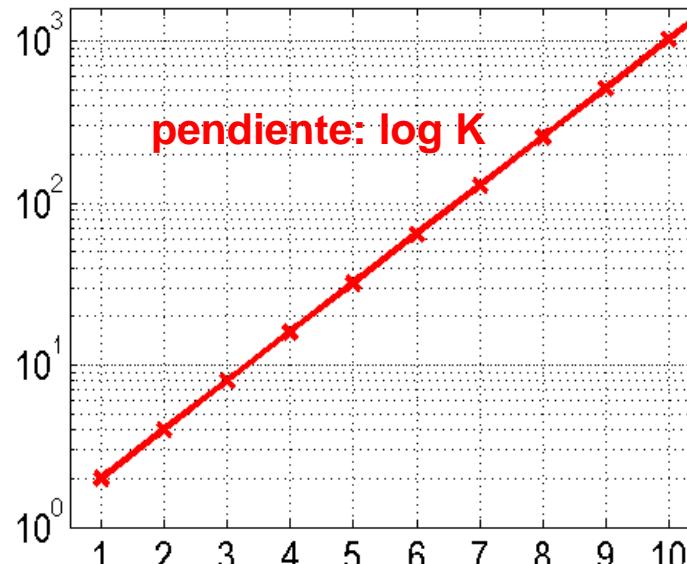
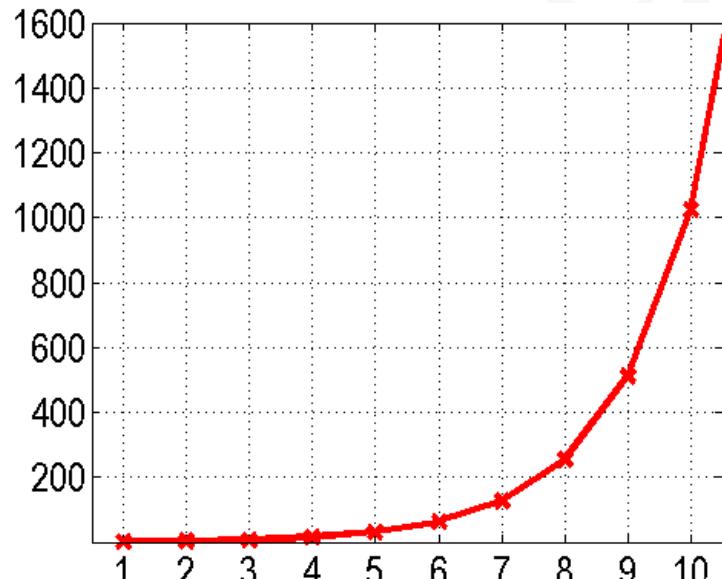


Crecimiento exponencial

- En caso de dominios discretos con intervalos iguales
 - Crecimiento geométrico
- Fórmula 1: **factor** de multiplicación anual (K)

$$I(t) = I_0 * K^{\Delta t}$$

I_0 = índice en el año de referencia (t_0)
 K = factor de crecimiento anual
 Δt = diferencia en años ($t-t_0$)



Crecimiento exponencial

- Fórmula 2: **fracción** de crecimiento anual (ΔI), o tanto por uno ($\Delta I = \Delta I\% / 100$)

$$I(t) = I_0 * K^{\Delta t}$$

$$K = 1 + \Delta I$$

$$I(t) = I_0 * (1 + \Delta I)^{\Delta t}$$

- Fórmula 3: **factor** de multiplicación (K_d) cada D años

$$I(t) = I_0 * K^{\Delta t}$$

$$K_d = K^D$$

$$K = K_d^{1/D}$$

$$I(t) = I_0 * (K_d^{1/D})^{\Delta t} = I_0 * K_d^{\Delta t/D}$$

Ejemplos de crecimiento

Dos formas de expresar
el mismo incremento exponencial:

- Crecimiento anual en %
 - “Las prestaciones de los sistemas informáticos en los años 60 y 70 crecían un **26% cada año**”
- Factor de multiplicación K_d cada D años
 - “Las prestaciones de los sistemas informáticos en los años 60 y 70 se **duplicaban cada 3 años**”
- Misma realidad con dos fórmulas:

$$1,26^3 = 2$$

Tiempo de ciclo del procesador

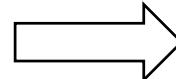
■ Frecuencias y tiempo de ciclo de procesadores, 1988 -2001

año	procesador	frecuencia (MHz)	ciclo (nsg.)
1988	motorola 88000	25	40,0
1990	intel i860	40	25,0
1991	HP 730	66	15,0
1991	mips R4000	100	10,0
1992	Alpha 21064	150	6,7
1993	Alpha 21064	200	5,0
1993	HP PA7100	100	10,0
1994	Alpha 21064	275	3,6
1994	HyperSparc	100	10,0
1995	PowerPC 604	133	7,5
1995	Alpha 21164	333	3,0
1996	Alpha 21164	500	2,0

año	procesador	frecuencia (MHz)	ciclo (nsg.)
1996	HP PA8000	180	5,6
1997	pentium II	300	3,3
1997	Alpha 21164	600	1,7
1998	Alpha 21264	800	1,3
1998	mips R12000	300	3,3
1999	pentium III	733	1,4
1999	Sparc Ultra II	300	3,3
2000	pentium III	1130	0,9
2000	HP PA8600	550	1,8
2001	pentium 4	2000	0,5
2001	PowerPC 74XX	733	1,4

Motorola
Intel
PA-Risc
Mips

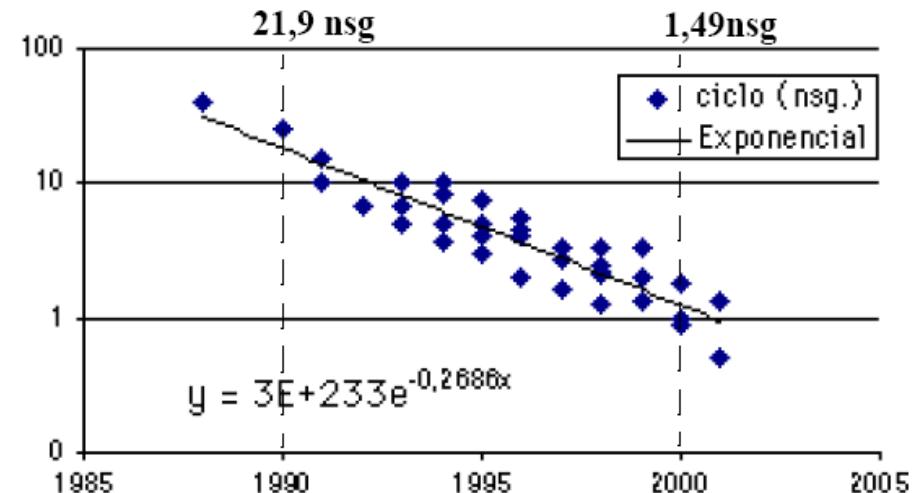
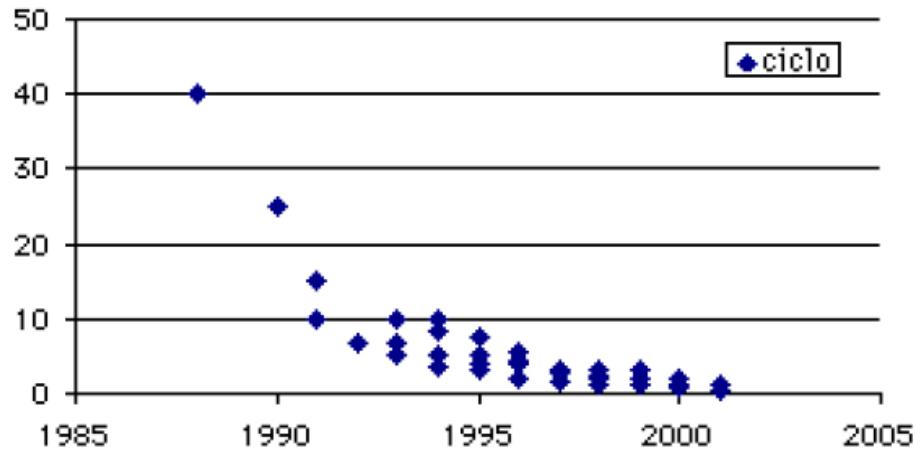
Alpha
Sparc
PowerPC



7 lenguajes máquina

Tiempo de ciclo del procesador

■ Tiempo de ciclo de procesadores, 1988-2001



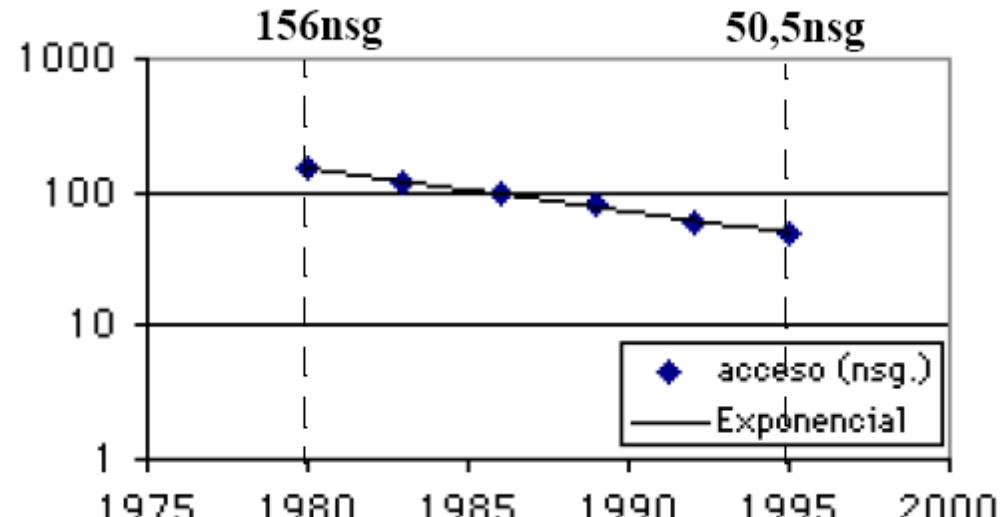
■ Expresión para Tiempo de ciclo (Tc) en función del año:

?

Tiempo de acceso a memoria DRAM

■ Tiempo de acceso a fila, 1980-1995

DRAM: tiempo de acceso RAS	
año	tiempo de acceso
1980	150
1983	120
1986	100
1989	80
1992	60
1995	50



■ Expresión para Tiempo de acceso (Ta) en función del año:

?

Rendimiento: tiempo, velocidad y consumo (1)

- Tiempo de ejecución de un trabajo (t)
 - Tiempo de ejecución, tiempo de respuesta, latencia
- Trabajos (T) por unidad de tiempo: día, segundo, ns ...(p)
 - Productividad (*throughput*), ancho de banda (*bandwidth*), *velocidad*
 - IPS, MIPS, MFLOPS

$$e = v * t$$

$$v = \frac{e}{t}$$

$$p = \frac{T}{t}$$

- Consumo
 - Energía consumida (Julios)
 - Energía consumida por unidad de tiempo: potencia (Watios)

Rendimiento: formas de comparar (2)

- Tiempo: segundos
- Productividad = velocidad =
 - cálculos / tiempo
 - bytes / tiempo
- **x** es **n%** más rápido que **y**

$$p(x) = p(y) + \frac{n}{100} p(y)$$

$$\frac{t(y)}{t(x)} = \frac{p(x)}{p(y)} = 1 + \frac{n}{100}$$

$$n = \frac{p(x) - p(y)}{p(y)} * 100$$

- **x** es **r veces** más rápido que **y**

$$\frac{t(y)}{t(x)} = \frac{p(x)}{p(y)} = r$$

Ejemplo: IPS, MIPS, MFLOPS, EPI, EPFLOP

```
Real*8 A(1000), B(1000), C(1000)  
  
DO i=1, 1000  
    A(i) = B(i) * C(i) + 3.0  
ENDDO
```

9 instrucciones de LM
2 operaciones PF
1 μ s todo el bucle
50 W

- IPS = instrucciones por segundo =
 - GIPS = IPS / 10^9 =
 - GFLOPS
= operaciones coma flotante por segundo / 10^9 =
FLOP S
 - EPI = energía por instrucción (nJ/inst) =
 - EPFLOP = energía por FLOP (nJ/FLOP) =

Ley del Hierro: Tiempo de ejecución

$$T_{ex} = I \times CPI \times T_c$$

- ¿Bajo qué condiciones puedo usar MIPS como métrica de comparación entre dos computadores?
- ¿Bajo qué condiciones puedo usar xFLOPS?
- ¿Bajo qué condiciones Tex?

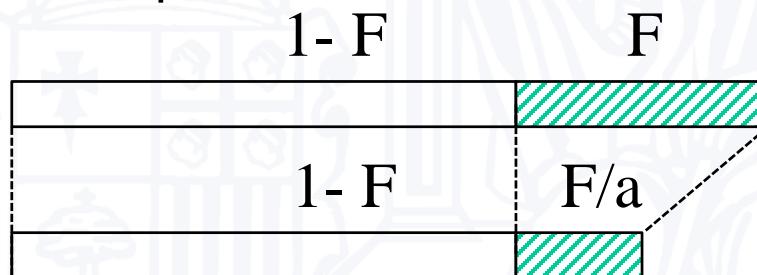
Ley de Gene Amdahl

- *Speedup*: incremento de prestaciones al introducir una mejora **E** en el sistema

$$\text{speedup}(E) = \frac{\text{prestaciones del nuevo sistema (con E)}}{\text{prestaciones del sistema viejo (sin E)}} = \frac{\text{Tiempo (sin E)}}{\text{Tiempo (con E)}}$$

- La ley de Amdahl formula el *speedup* en función de:
 - la fracción de tiempo (**F**) donde opera la mejora **E**
 - ganancia (factor **a**) durante ese tiempo

S_{up} = ?



Ley de Amdahl: ejercicio

■ Cambio de servidor web ?

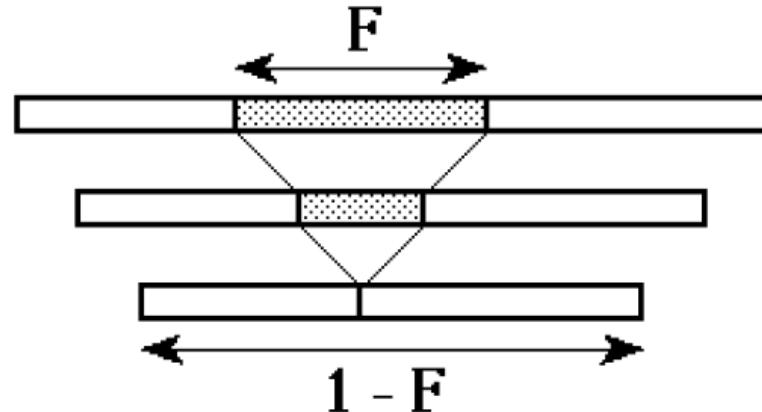
- estadísticas: 40% del tiempo haciendo cálculos, 60% E/S
- nuevo procesador 10 veces más rápido haciendo cálculos
- mantenemos el subsistema de E/S (discos + red)

Speed-up =

Ley de Amdahl: principios cuantitativos

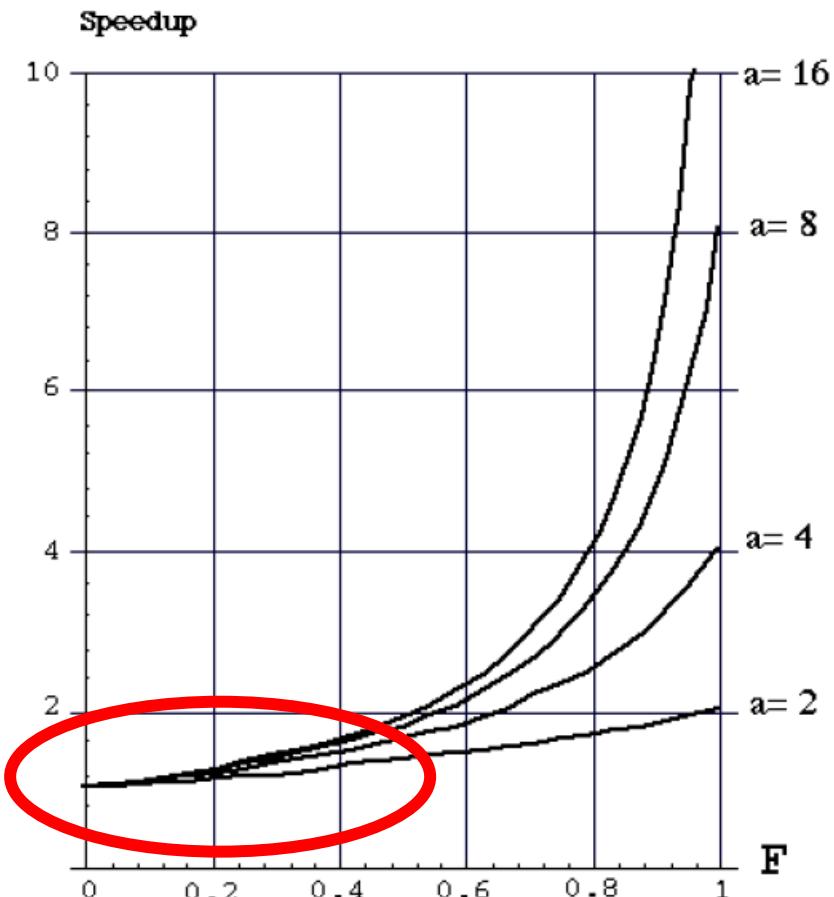
- Disminución del ratio Speedup / Coste
 - Si sólo se actúa sobre una parte del sistema, cada nueva mejora añadida trabaja sobre una fracción de tiempo F menor
- Límite de speedup
 - inverso de fracción de tiempo no mejorada

$$\text{máximo speedup}(E) = \frac{1}{1 - F}$$

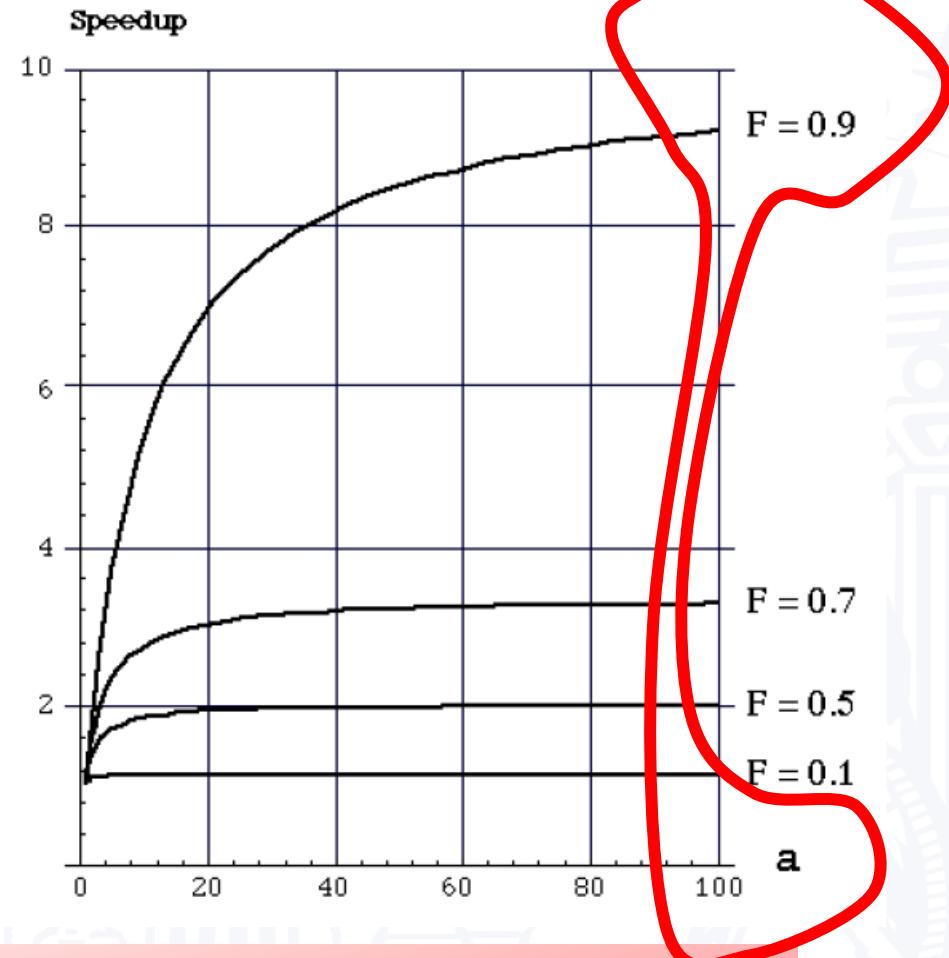


- Se debe actuar siempre sobre el **caso más común**, el que mayor F consume
- El caso más **simple** suele ser el **más común**

Ley de Amdahl: ejemplo



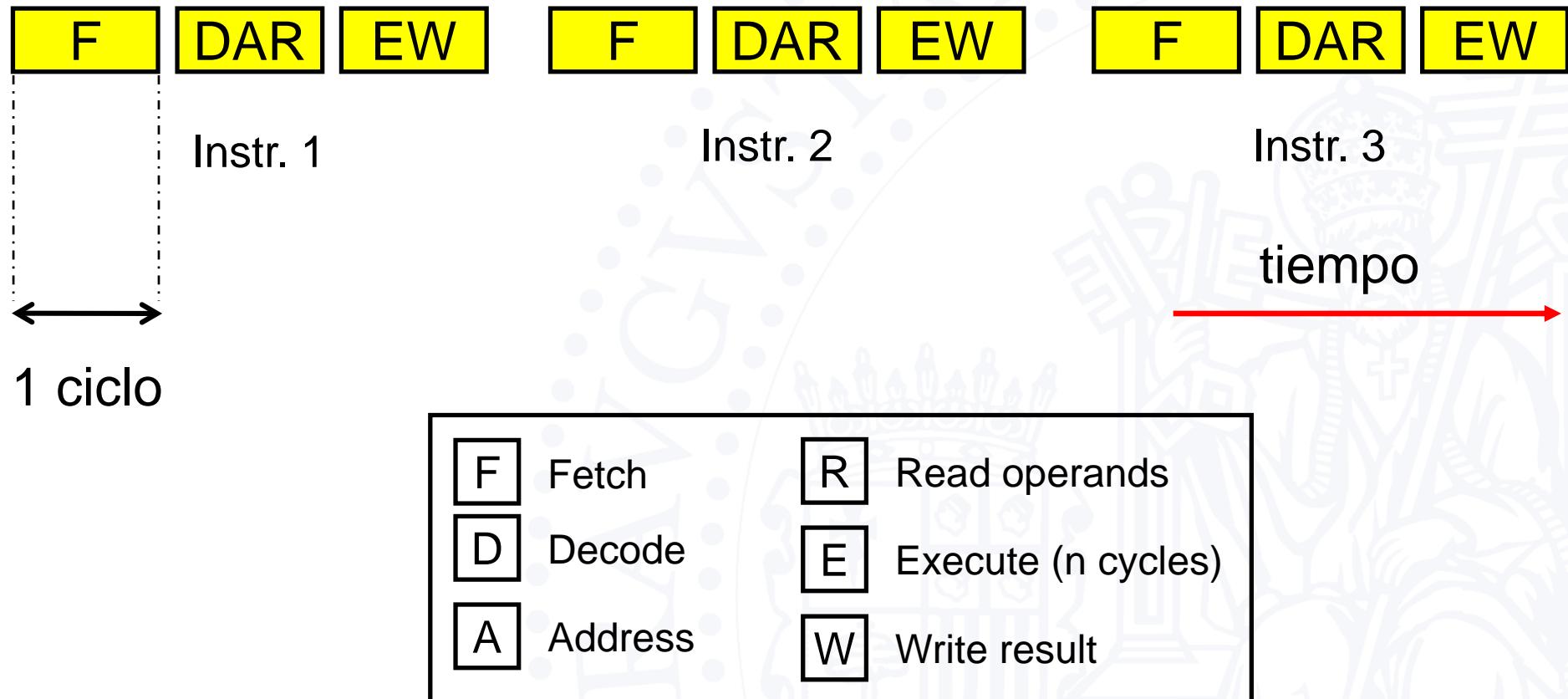
No importa el valor de a
si F es pequeña



Con valores grandes a ,
el límite de Speedup es $1/(1-F)$

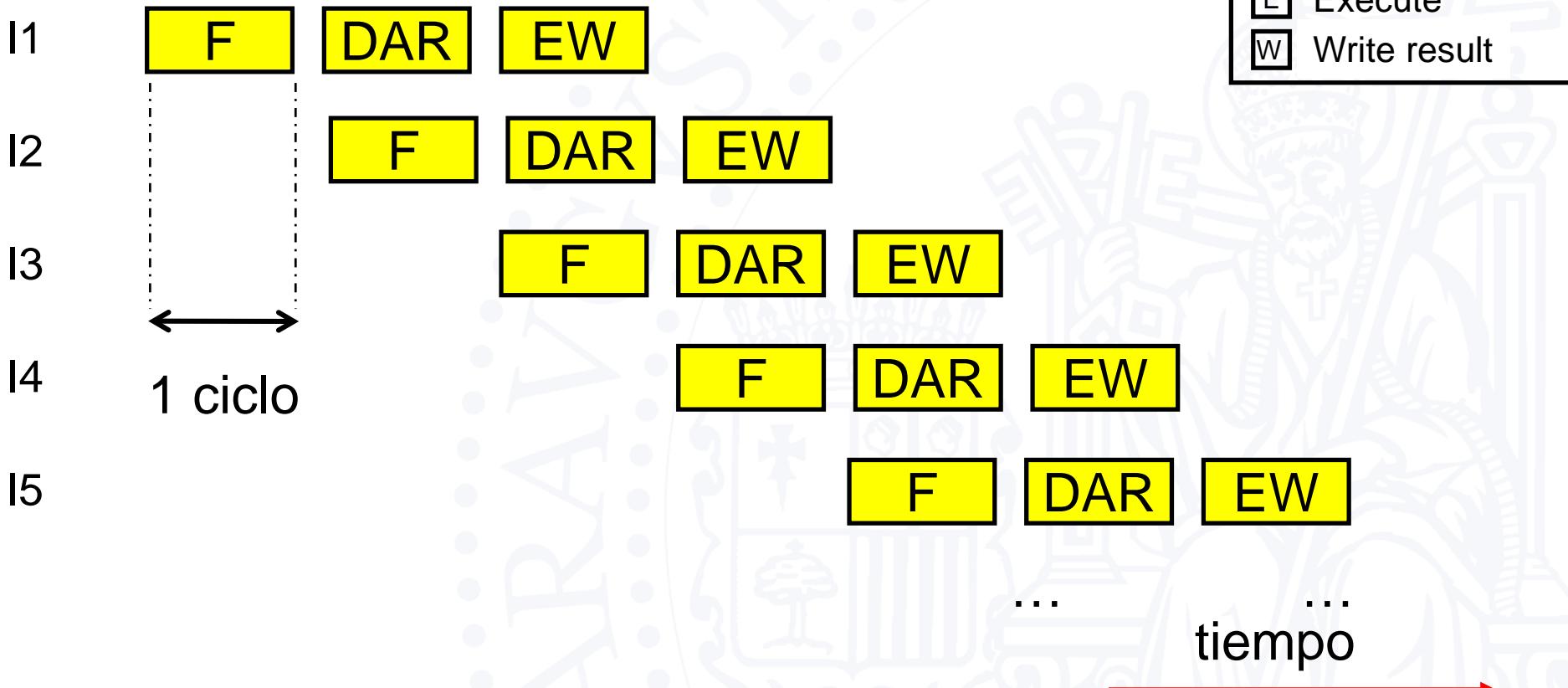
Modelo de ejecución y rendimiento

- Generación 1 (> 3 ciclos por instrucción)



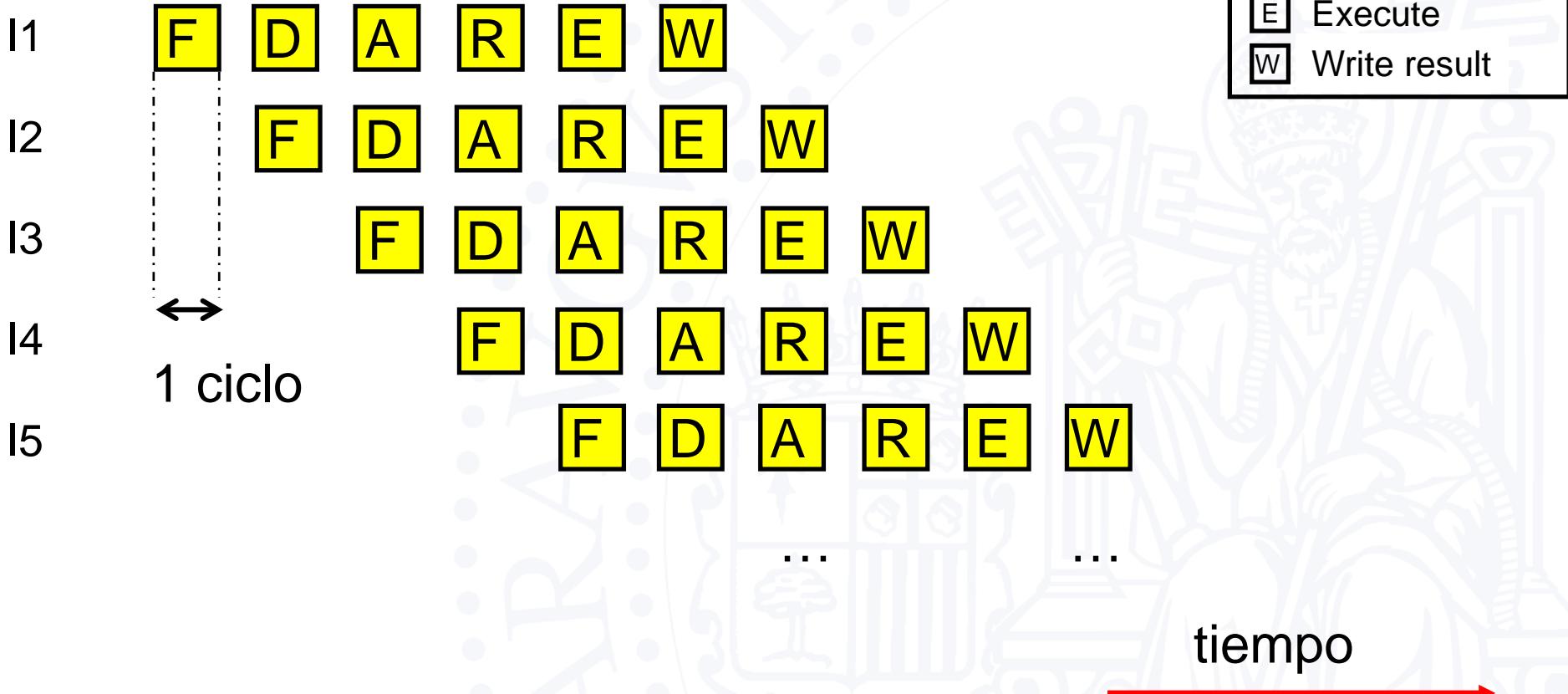
Evolución en el modelo de ejecución

- Generación 2 ($> 1 \text{ cpi}$)
- Segmentación



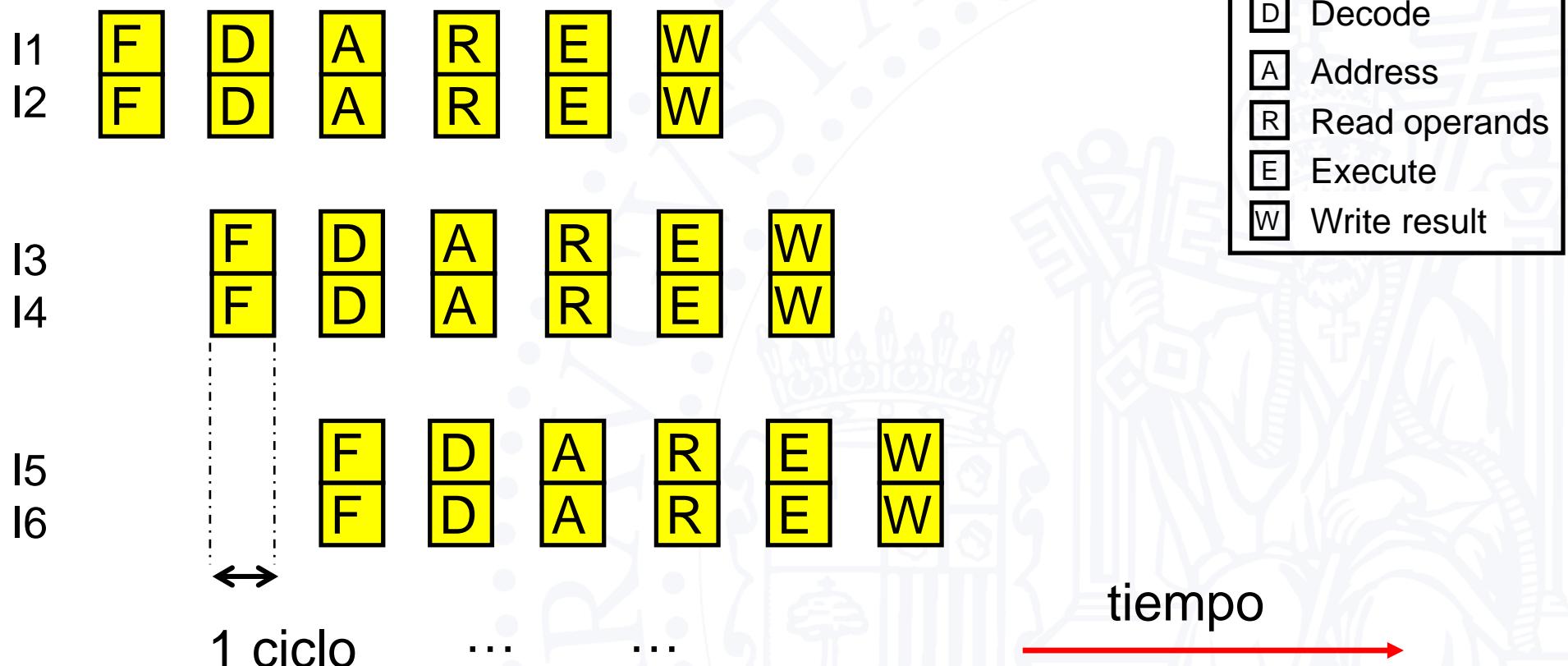
Evolución en el modelo de ejecución

- Generación 3 ($> 1 \text{ cpi}$)
- Aumenta la frecuencia



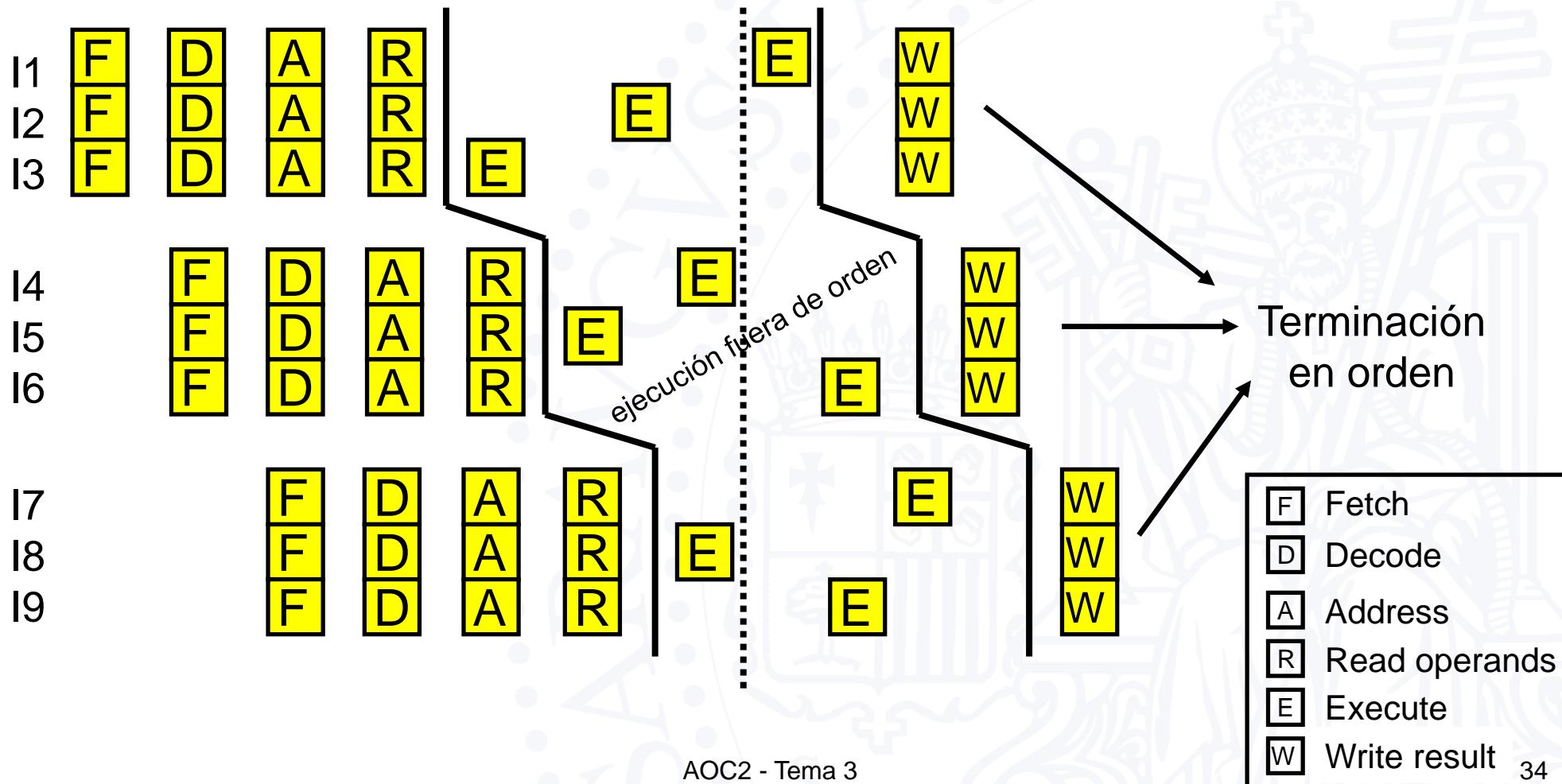
Evolución en el modelo de ejecución

- Generación 4 (> 0.5 cpi) – en μ a finales de los 80 –
- Procesador superescalar



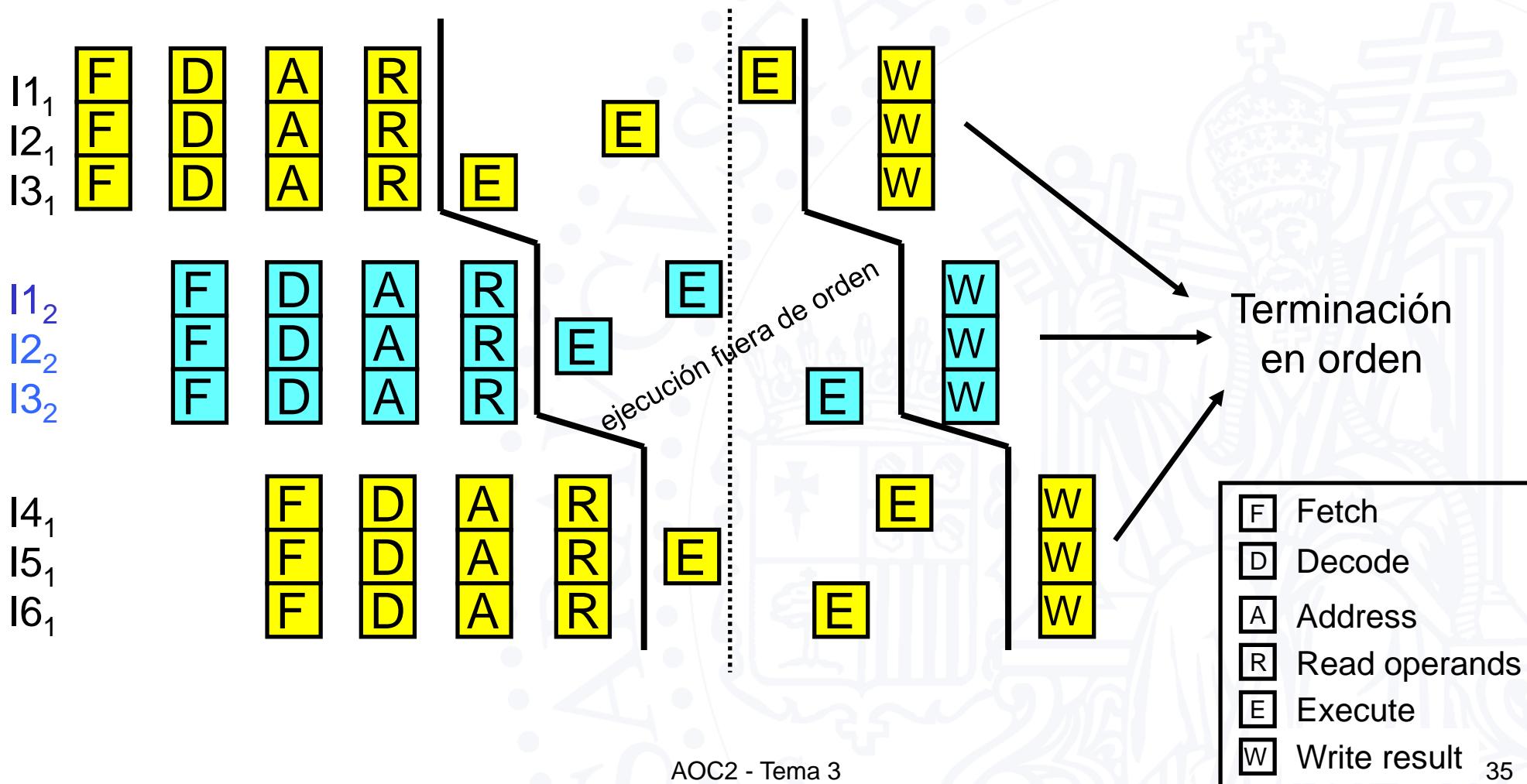
Evolución en el modelo de ejecución

- Generación 5 (≈ 0.3 cpi) – en μ a principios de los 90 –
- Ejecución en desorden



Evolución en el modelo de ejecución

- Generación 6 (≈ 0.25 cpi) – en μ en los años 2000 –
- Multithreading

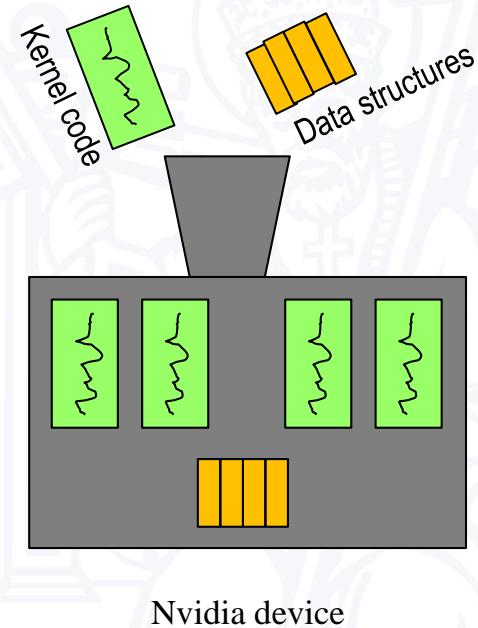


Otros modelos de ejecución

- SIMD / SIMP (CUDA/OpenCL)
 - Ejemplo: $C = A + B$

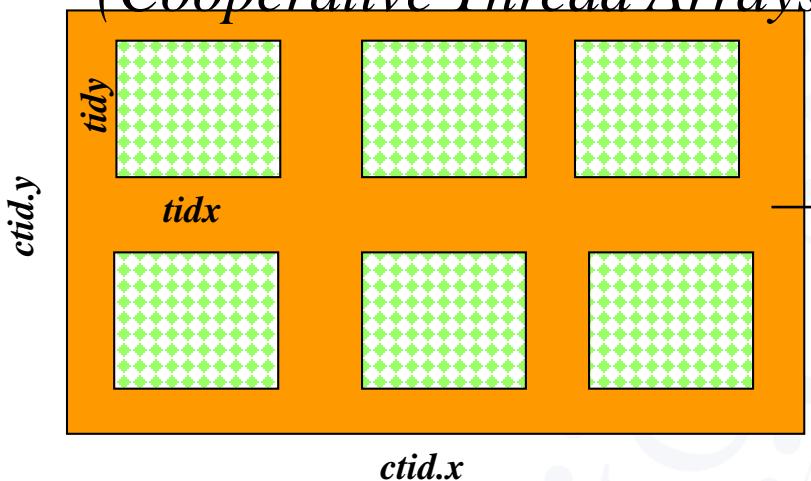
```
__global__ void matAdd(float A[N][N], float B[N][N],float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
// Allocate structures in device
// Transfer data to device
...
// Set grid parameters and number of threads
    dim3 dimBlock(N, N);
// Kernel invocation
    matAdd<<<1, dimBlock>>>(A, B, C);
// Transfer results from device
...
}
```

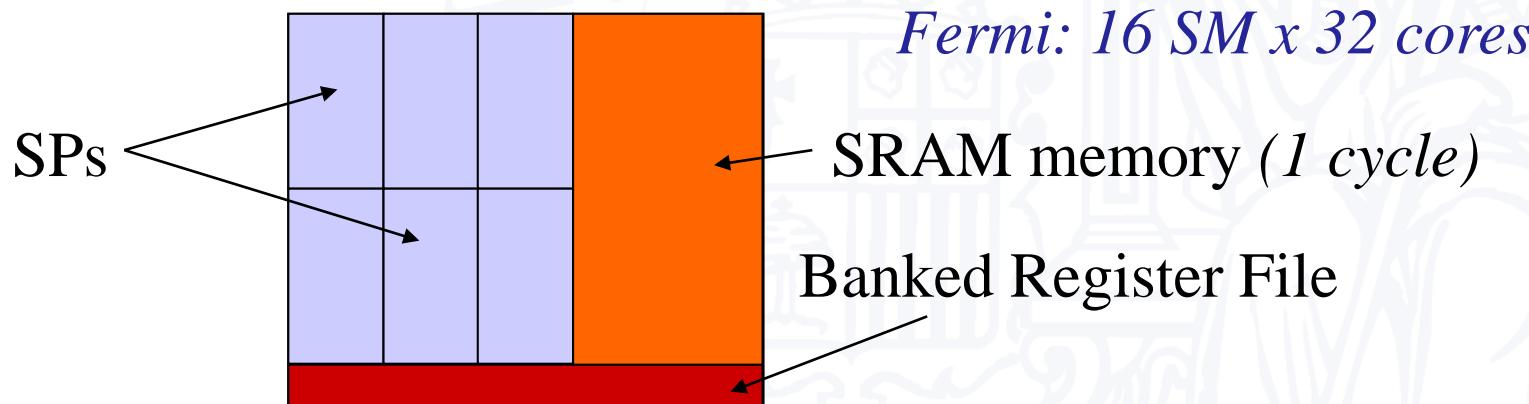
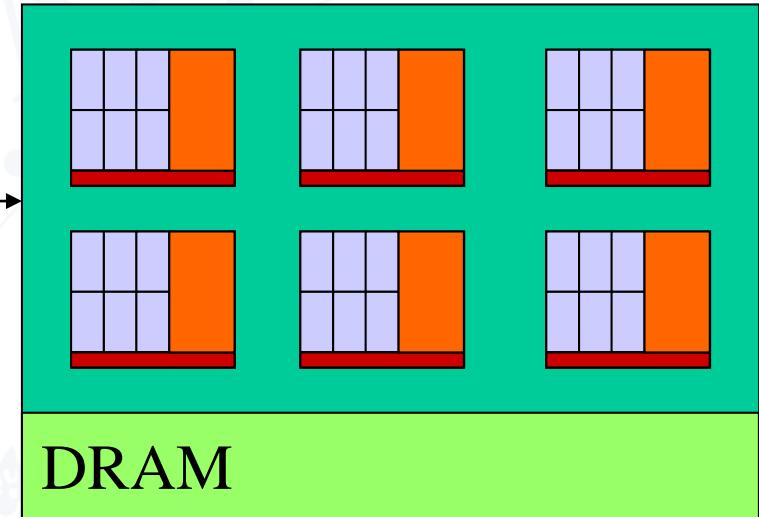


Programming model and hw

Logical Grid CTAs
(Cooperative Thread Arrays)



Hdw: Array of cores (SMs)



Fermi: 16 SM x 32 cores

Hacia dónde vamos ahora en AOC2

- Incremento del rendimiento por segmentación
 - Diseño de la ruta de datos segmentada
 - Diseño del control
 - Problemas para conseguir el CPI = 1: riesgos y soluciones
 - ◆ Estructurales
 - ◆ De datos
 - ◆ De control

Segmentación

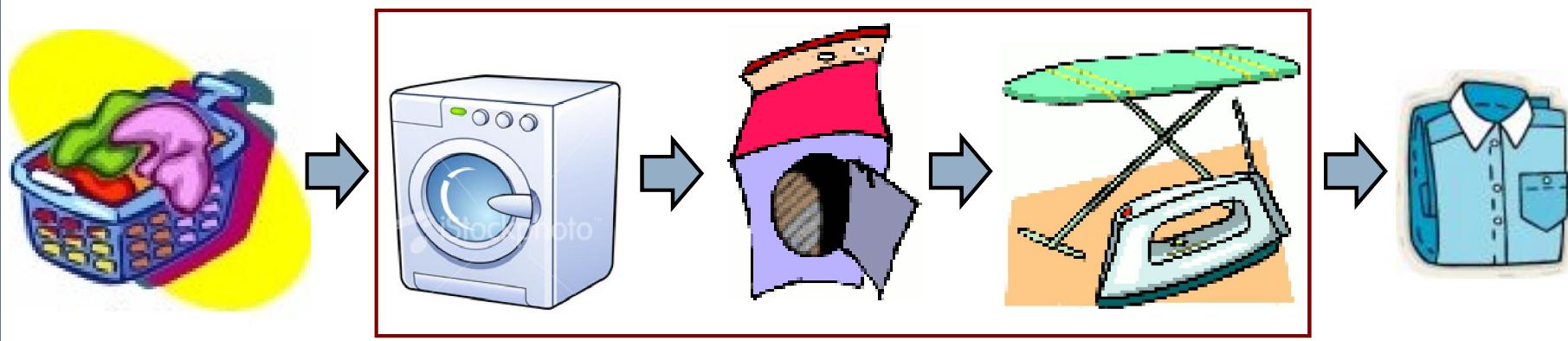
Segmentación

- 1.- Introducción
- 2.- Ruta de datos segmentada
- 3.- Control de la ruta de datos segmentada
- 4.- Riesgos
 - 4.1.- Riesgos de datos
 - Solución software a los riesgos LDE
 - Anticipación de operandos
 - Detención del pipeline
 - Reordenamiento de código
 - 4.2.- Riesgos de control
 - Soluciones a los riesgos de control
 - Predicción dinámica de saltos

Bibliografía:

- “Estructura y diseño de computadores” David A. Patterson & John L. Hennessy, Editorial Reverté, 2000
- “Computer architecture. A quantitative approach”, J.L. Hennessy & D.A. Patterson, Morgan Kaufmann, 2^a edic.

1.- Introducción



Lavadora

1 hora

Secadora

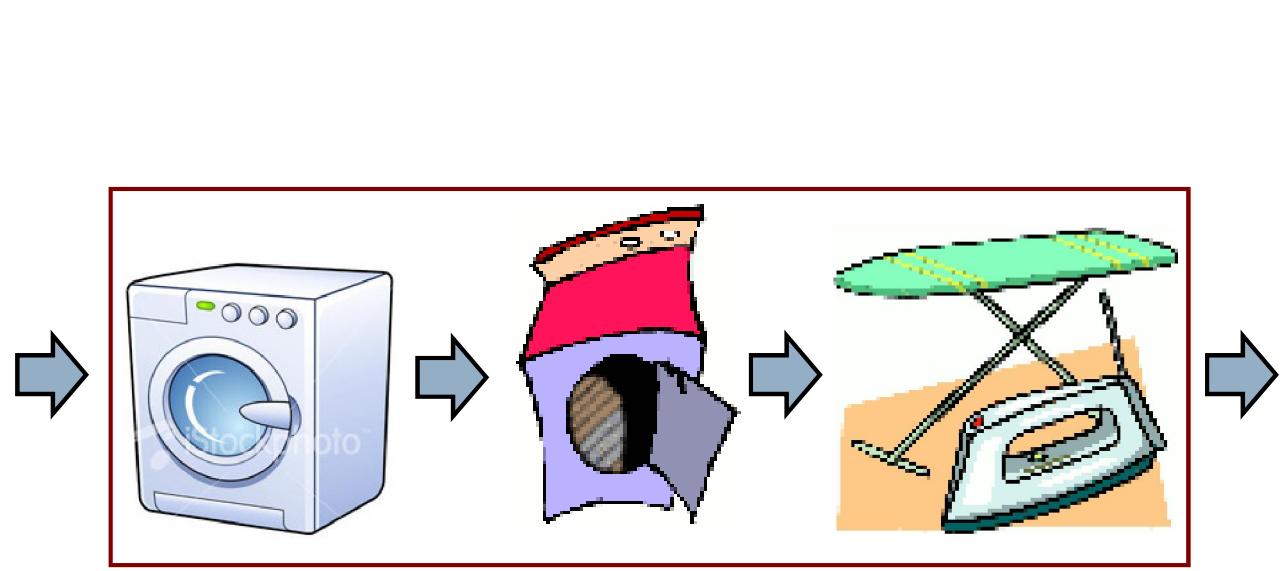
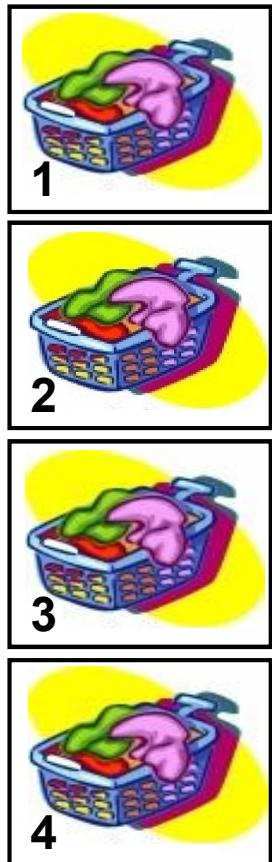
45 minutos

Planchado

1 hora

Total 2 horas 45 minutos

1.-¿Cómo se puede hacer más rápido?

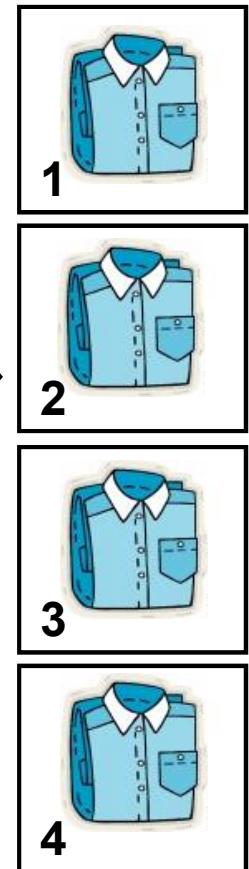


Lavadora
1 hora

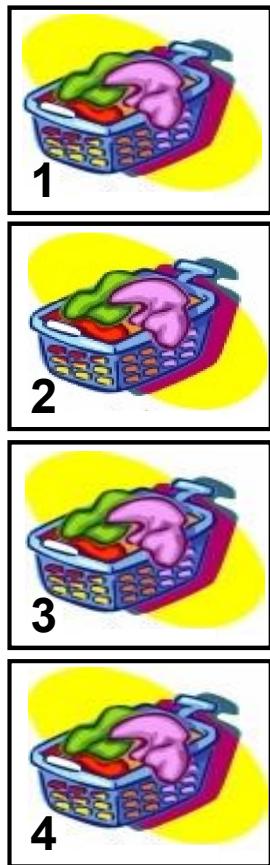
Secadora
45 minutos

Planchado
1 hora

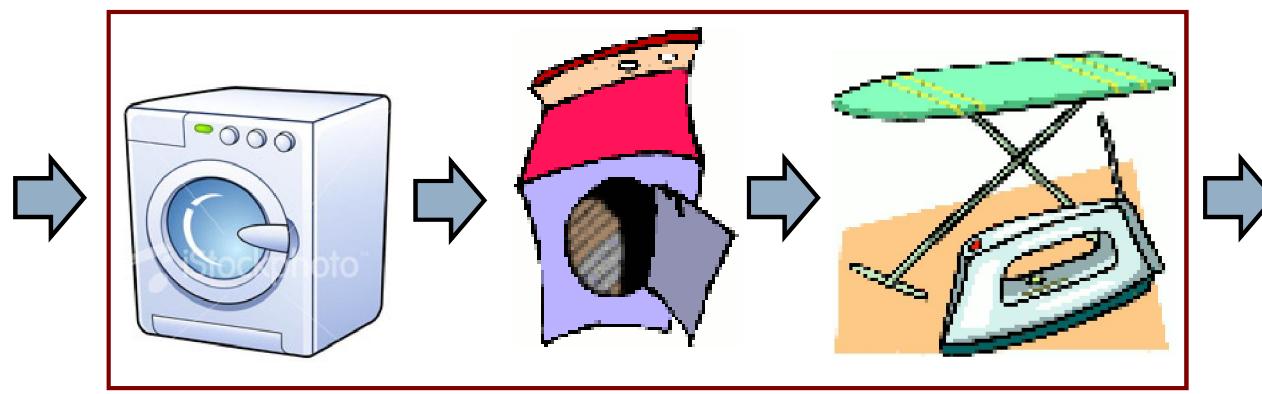
Total 11 horas



1.-¿Cómo se puede hacer más rápido?



Solución 1: ejecución segmentada



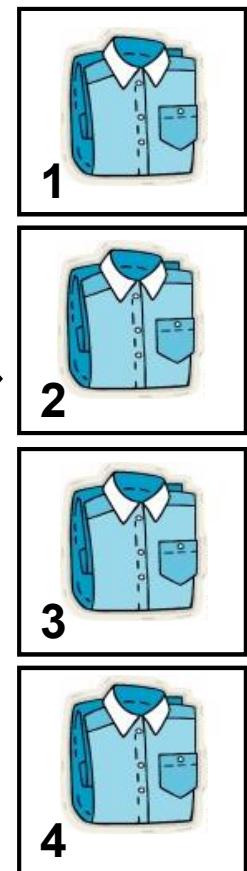
Lavadora
1 hora

Secadora
45 minutos

Planchado
1 hora

Tenemos tres recursos pero en cada instante sólo usamos uno

¿y si tratamos de usar los tres a la vez?



1.-¿Cómo se puede hacer más rápido?



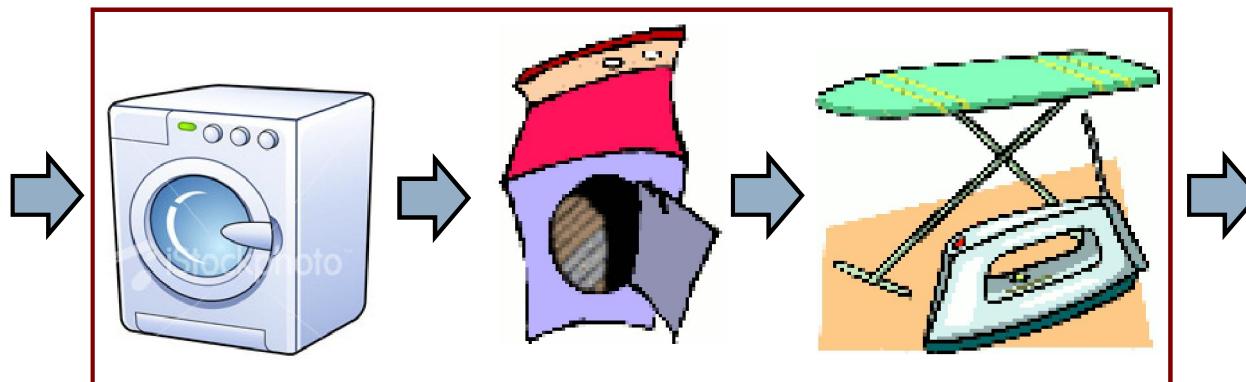
1.-¿Cómo se puede hacer más rápido?



Tiempo consumido: 1 hora

Usamos dos recursos en paralelo

1.-¿Cómo se puede hacer más rápido?



Lavadora
1 hora

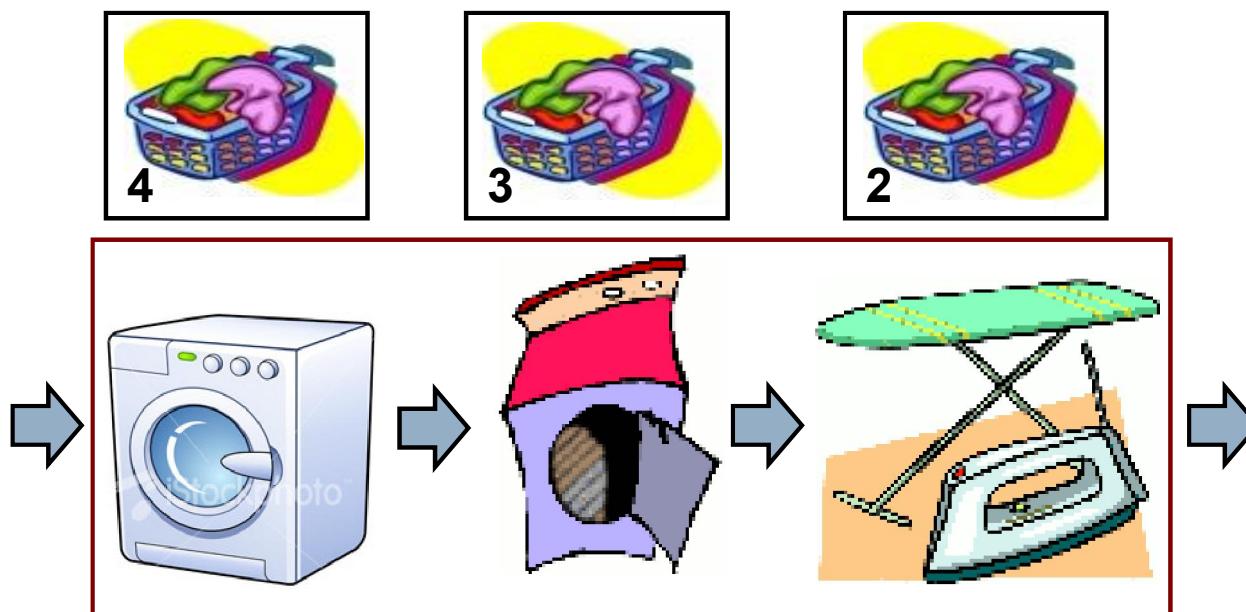
Secadora
45 minutos

Planchado
1 hora

Tiempo consumido: 2 horas

Usamos tres recursos en paralelo

1.-¿Cómo se puede hacer más rápido?



Lavadora
1 hora

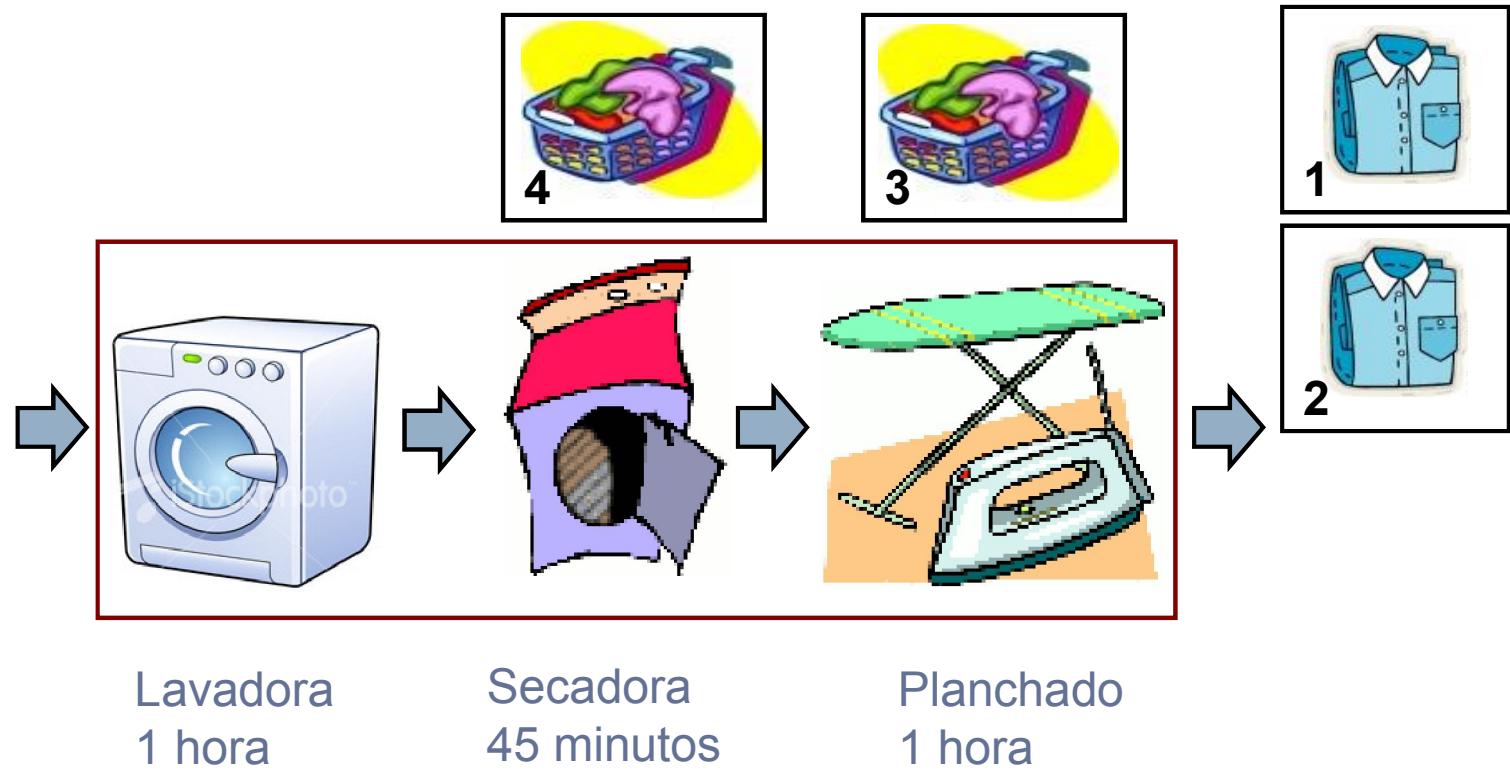
Secadora
45 minutos

Planchado
1 hora

Tiempo consumido: 3 horas

Usamos tres recursos en paralelo

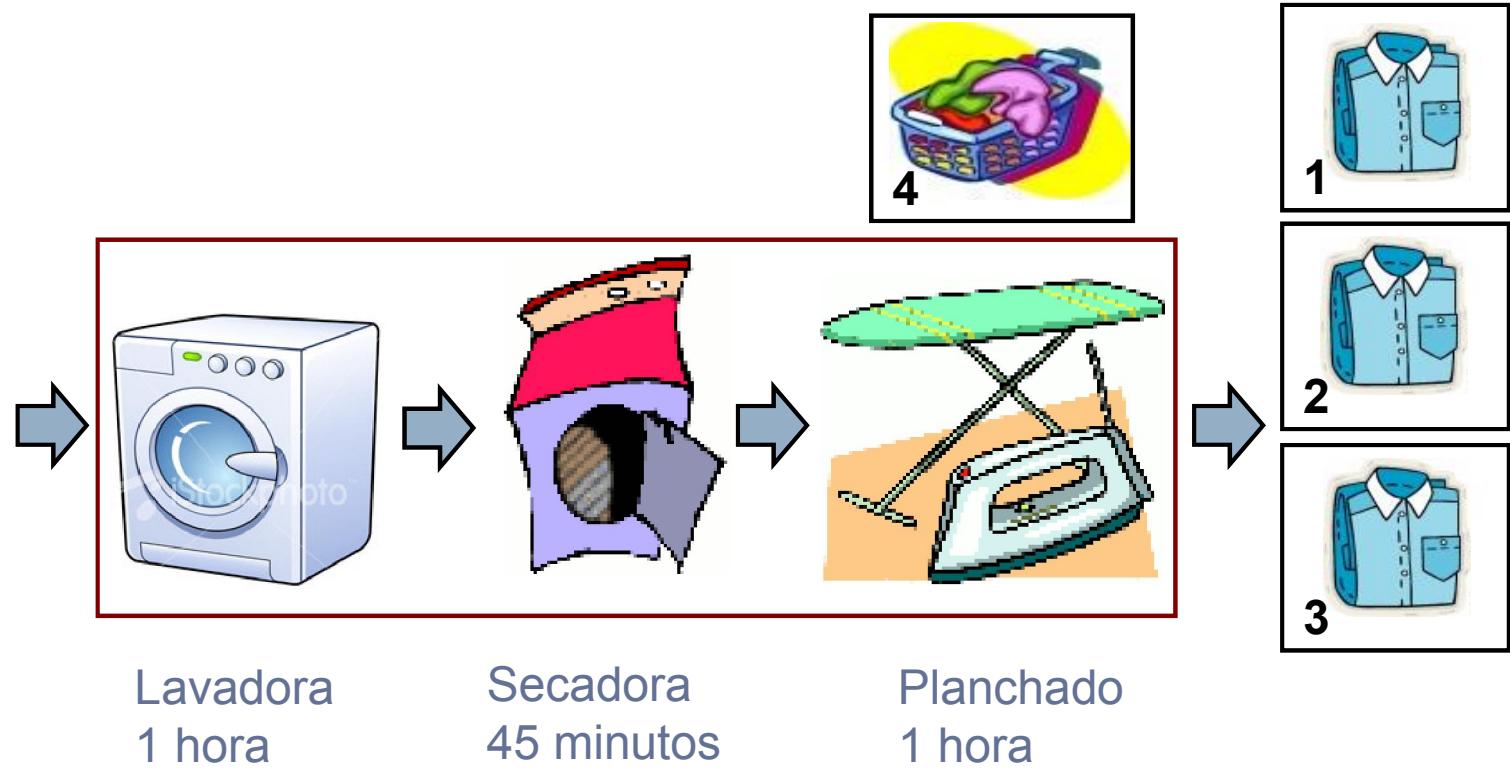
1.-¿Cómo se puede hacer más rápido?



Tiempo consumido: 4 horas

Usamos dos recursos en paralelo

1.-¿Cómo se puede hacer más rápido?



Tiempo consumido: 5 horas

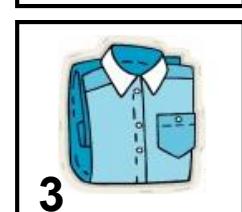
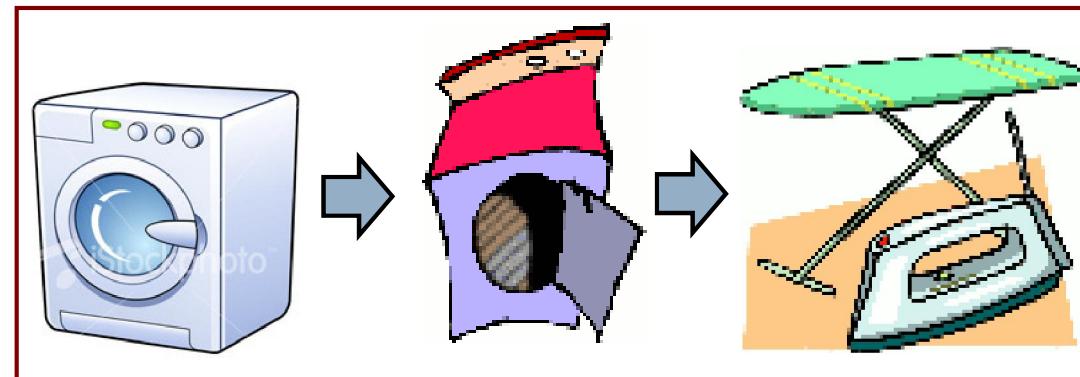
1.-¿Cómo se puede hacer más rápido?



Tiempo consumido: 6 horas

¡Antes eran 11 horas!

1.-¿Y si tuviésemos 1000 cestas de ropa?



• • •

Lavadora
1 hora

Secadora
45 minutos

Planchado
1 hora



Primera cesta: 3 horas
Segunda cesta: 4 horas

...
Enésima cesta $2 + n$ horas

...
Milésima cesta: 1002 horas

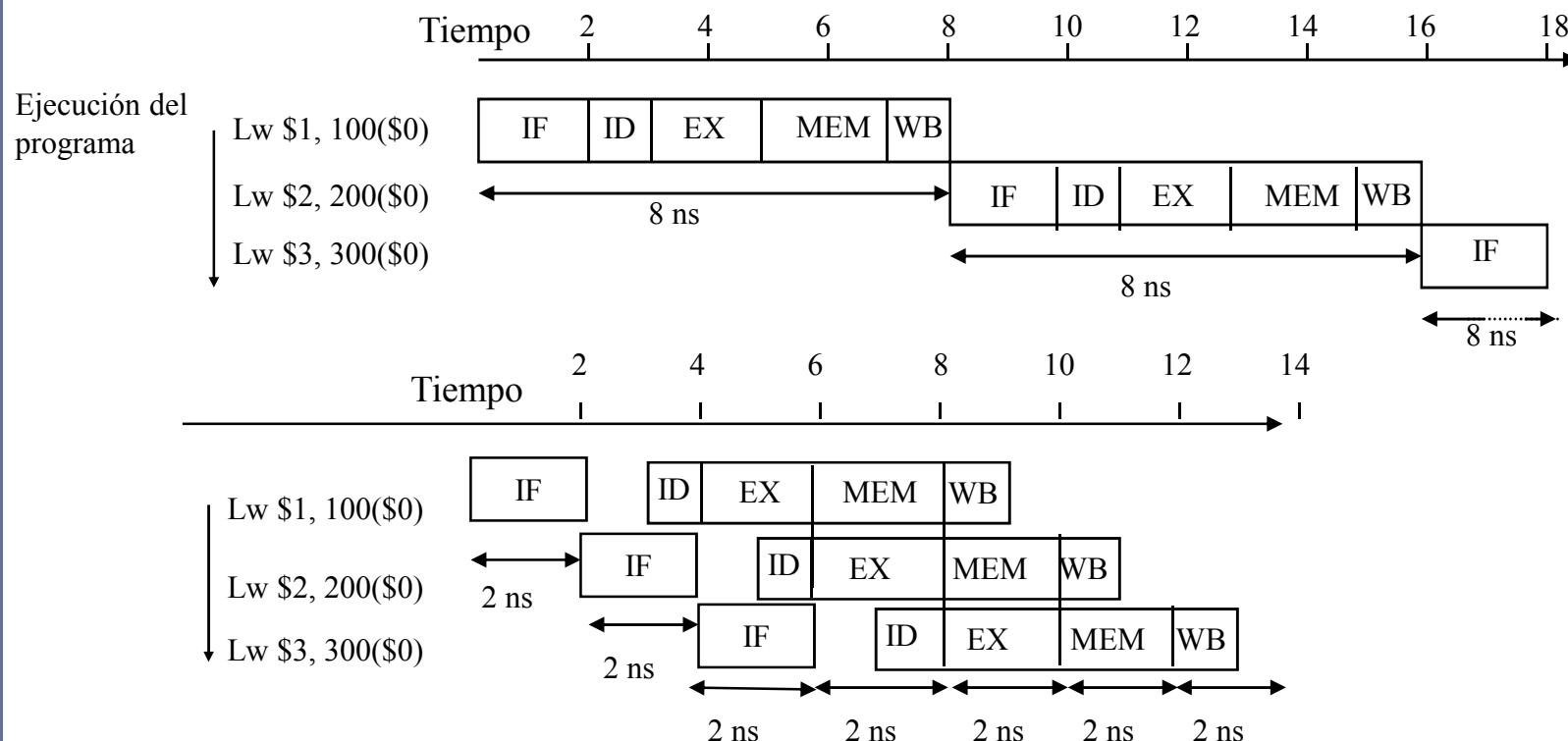
• • •



¿Cómo se puede aplicar a un procesador?

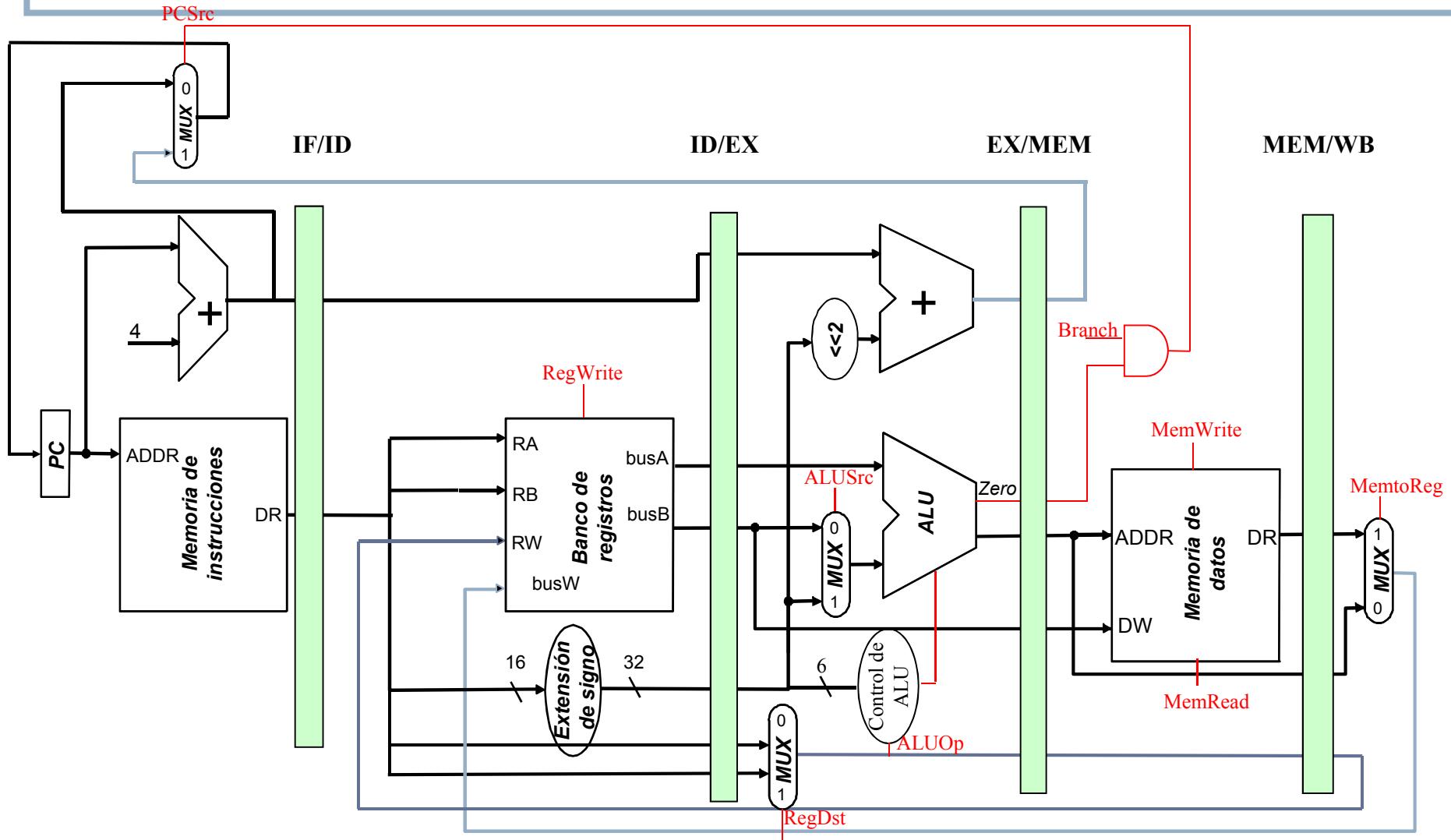
- Dividimos la ejecución de la instrucción en cinco etapas fundamentales:
 - IF: Se lee la instrucción de la memoria de instrucciones
 - ID: se decodifica la instrucción y se leen los operandos del banco de registros.
Se extiende el signo del operando inmediato
 - EX: Se utiliza la ALU para realizar los cálculos
 - MEM: Se lee o escribe en la memoria de datos
 - WB: Se escribe en banco de registros
- Cada etapa puede ejecutar una instrucción distinta
- Podemos ejecutar hasta cinco instrucciones en paralelo

1.- ¿Cómo se puede aplicar a un procesador?



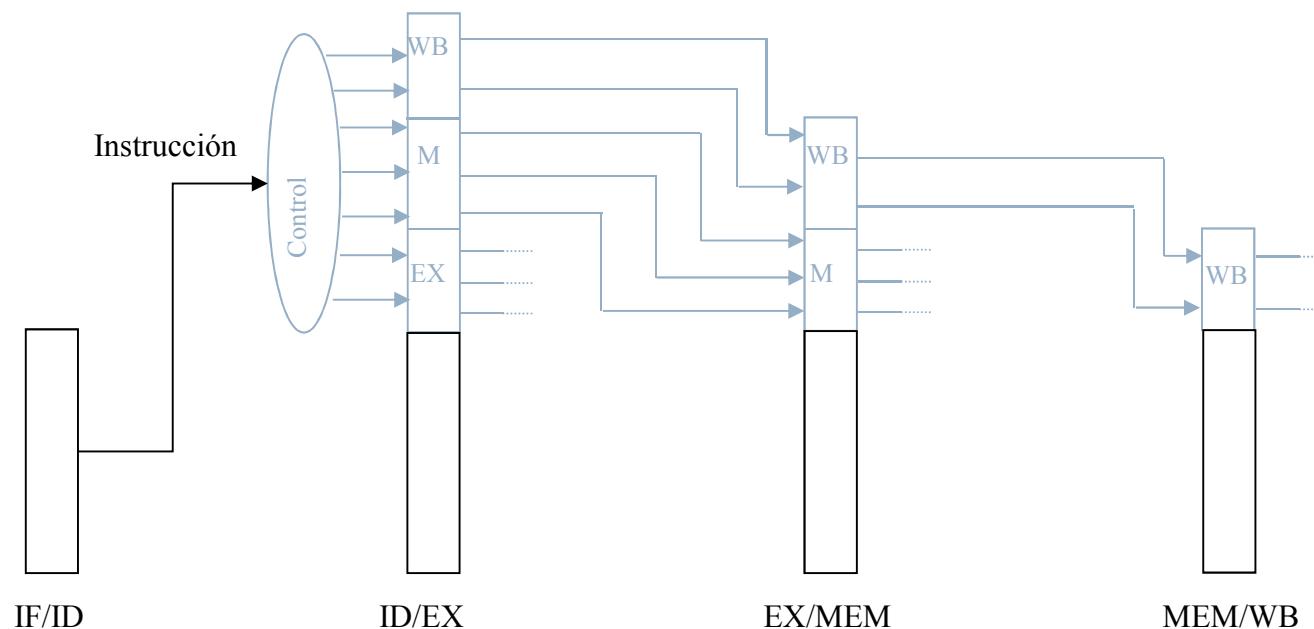
El speedup ideal = número de etapas
¿Es posible?

3.- Ruta de datos segmentada

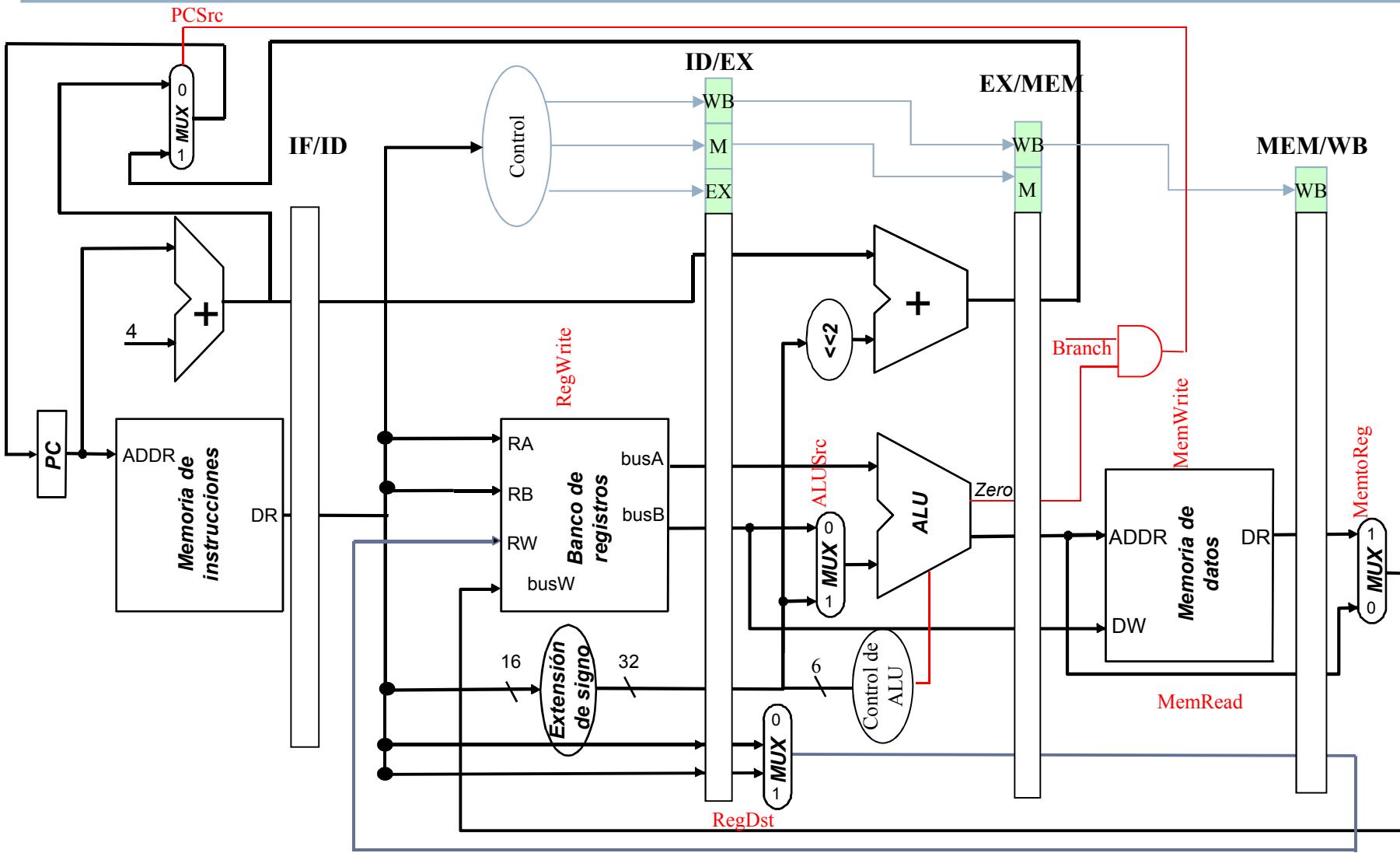


3.- Control de la ruta de datos segmentada

Las señales de control se generan en la U.C. y se van pasando de una etapa a otra como si fuesen datos.



3.- Control de la ruta de datos segmentada



4.- Riesgos

- Estructurales
 - No disponibilidad de un módulo cuando se necesita
 - No existen en nuestro MIPS porque tenemos:
 - Memoria de datos e instrucciones separadas
 - Arquitectura registro-registro
 - Sumadores adicionales para el cálculo del PC
 - En general se solucionan duplicando módulos y segmentando las ALUs
- De datos
 - No disponibilidad de un dato cuando se necesita
- De control
 - Desconocimiento de cuál es la próxima instrucción que debe ejecutarse

Dependencias vs. Riesgos

■ Dependencia

- Propiedad del código

subcc **r1**, r2, r3

addcc r5, **r1**, r4

■ Riesgo:

- Limitación hardware
orden de ejecución
 \neq *orden de programa*

■ Si hay dependencia puede existir o no riesgo

	Dependencia	Riesgo
	Dependencia s.s. <i>True dependency</i>	RAW <i>Read After Write</i>
Dependencias de almacenamiento / nombre (pueden eliminarse)	Antidependencia <i>Antidependency</i>	WAR <i>Write After Read</i>
	Dep. de salida <i>Output Dependency</i>	WAW <i>Write After Write</i>

4.1.- Dependencias y riesgos

- 1.- Antidependencia

sub \$2, **\$1**, \$3

add **\$1**, \$4, \$5

- Posible riesgo: *escritura después de lectura*
- No aparece en MIPS



- 2.- Dependencia de salida

sub **\$2**, \$1, \$3

add **\$2**, \$4, \$5

- Posible riesgo: *escritura después de escritura*
- No causan riesgo en MIPS



- 3.- Dependencia (*productor – consumidor*)

sub **\$2**, \$1, \$3 ; *productor*

add \$4, **\$2**, \$5 ; *consumidor a distancia 1*

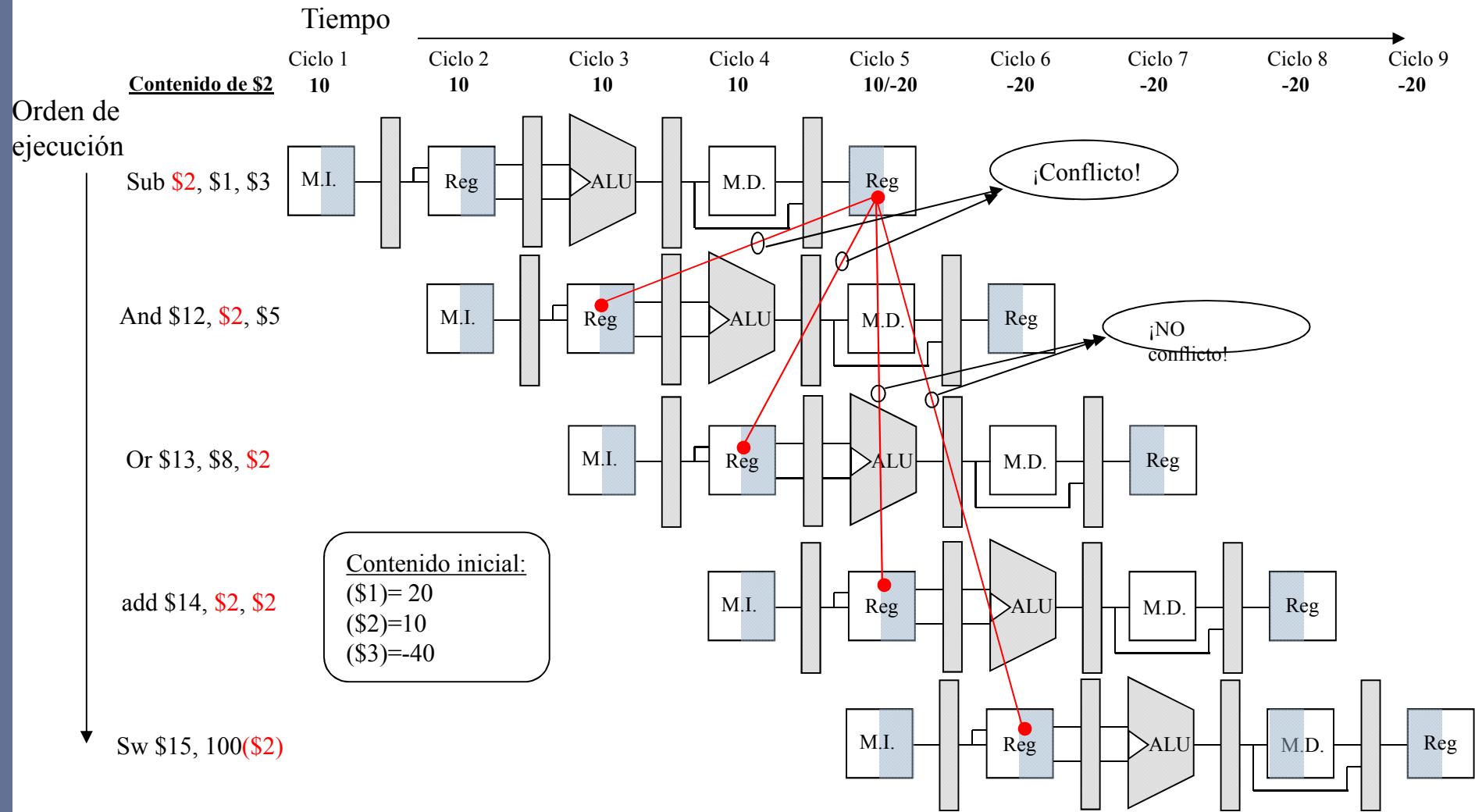
or \$6, \$7, **\$2** ; *consumidor a distancia 2*

- Causan *riesgo de lectura después de escritura (LDE)* en MIPS



4.1.- Riesgos de datos LDE

La ejecución de una instrucción comienza antes de finalizar las anteriores



Solución software a los riesgos LDE

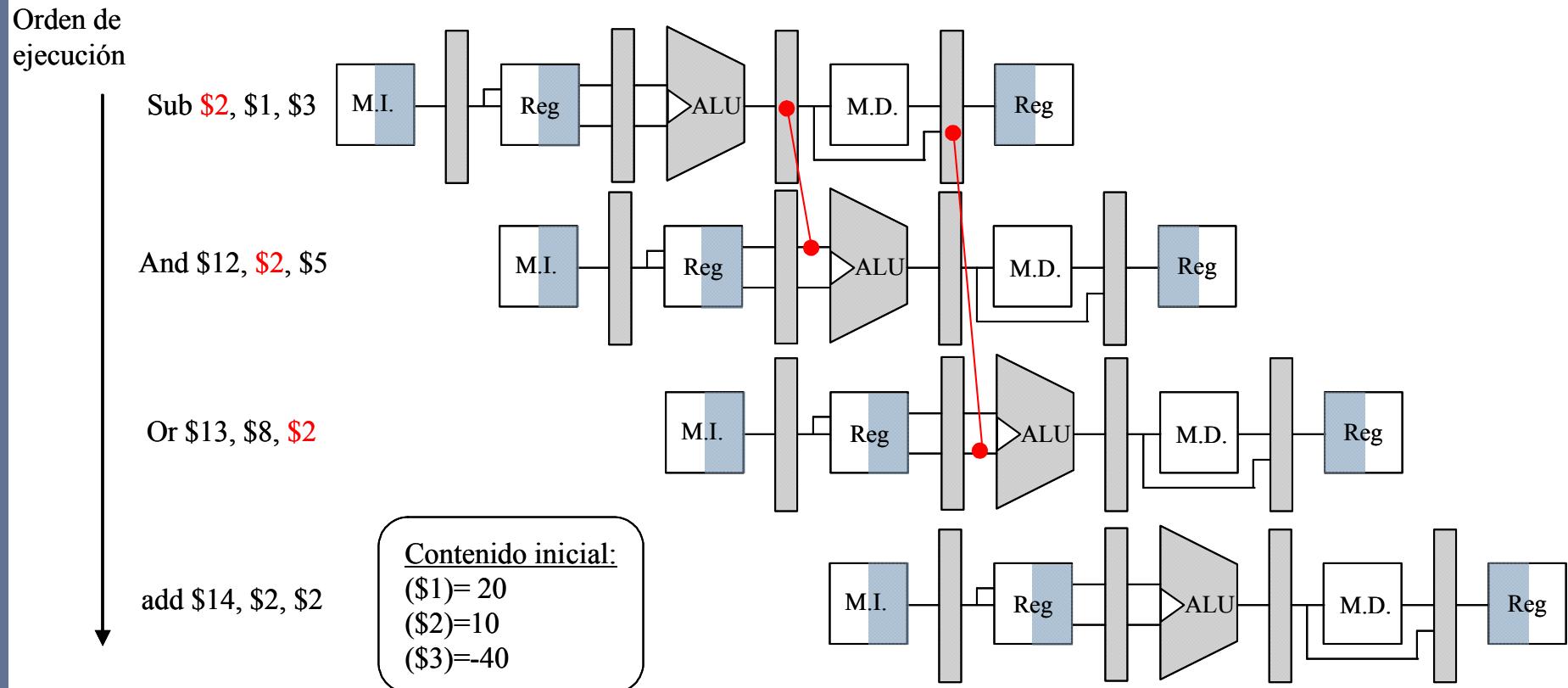
- El compilador se encarga de garantizar que no se produzcan riesgos
- Se *insertan instrucciones* NOP entre aquellas instrucciones que tengan dependencias de datos
- La ejecución se hace **más lenta**
 - $T_{ex} = I \times CPI \times T_c$ ¿qué término aumenta?

Sub	\$2, \$1, \$3
and	\$12, \$2, \$5
or	\$13, \$6, \$2
add	\$14, \$2, \$2
sw	\$15, 100(\$2)



4.1.- Anticipación de operandos (forwarding)

- Solución hardware a riesgos LDE
- Usa resultados temporales sin esperar a que se escriban en el banco de registros



4.1.- Anticipación de operandos

¿Cómo detectar si hay que activar un cortocircuito?



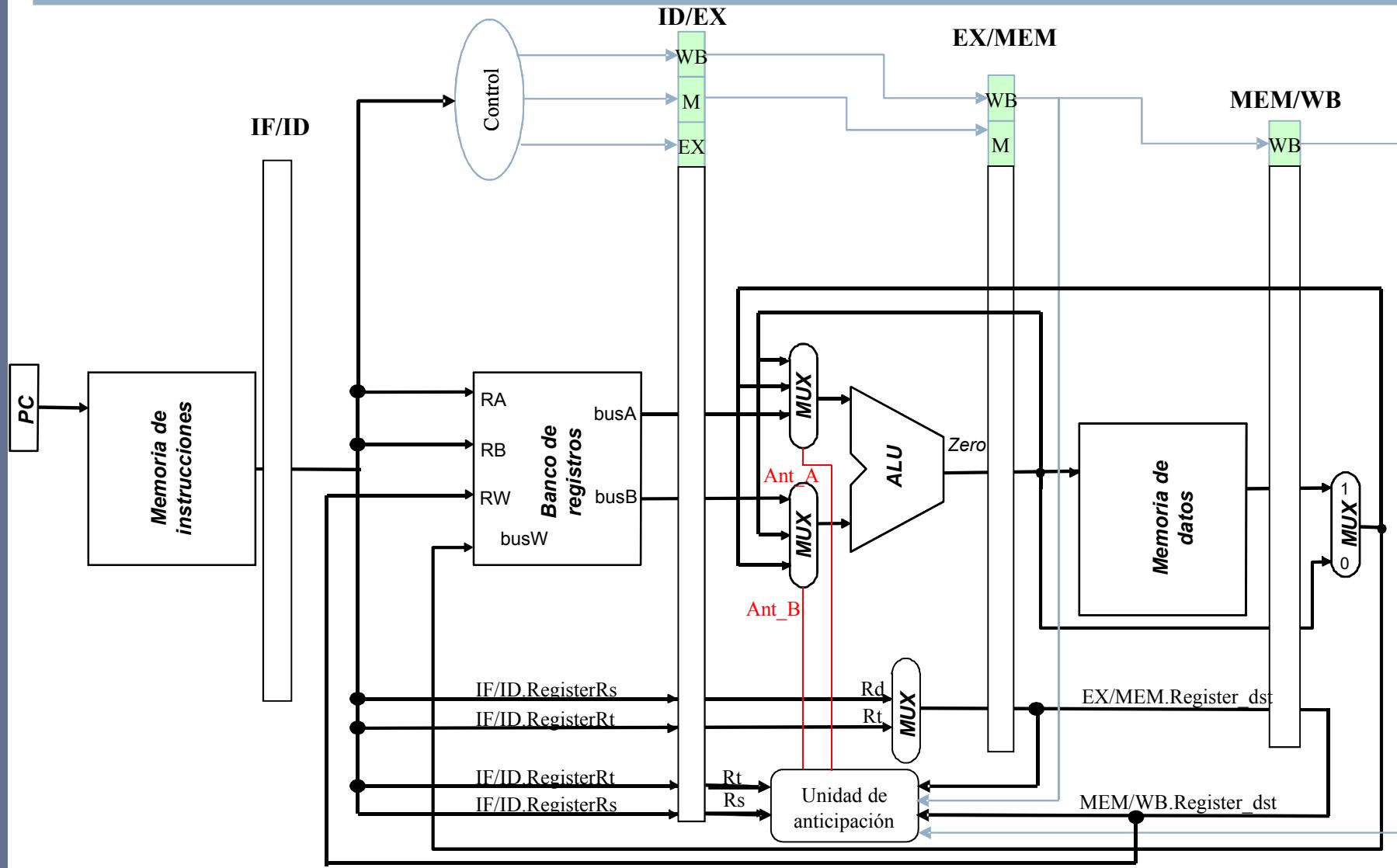
Si (**EX/MEM.RegWrite**
and **EX/MEM.Register_dst = ID/EX.RegisterRs**) **Ant_A** = 10

Si (**EX/MEM.RegWrite**
and **EX/MEM.Register_dst = ID/EX.RegisterRt**) **Ant_B** = 10

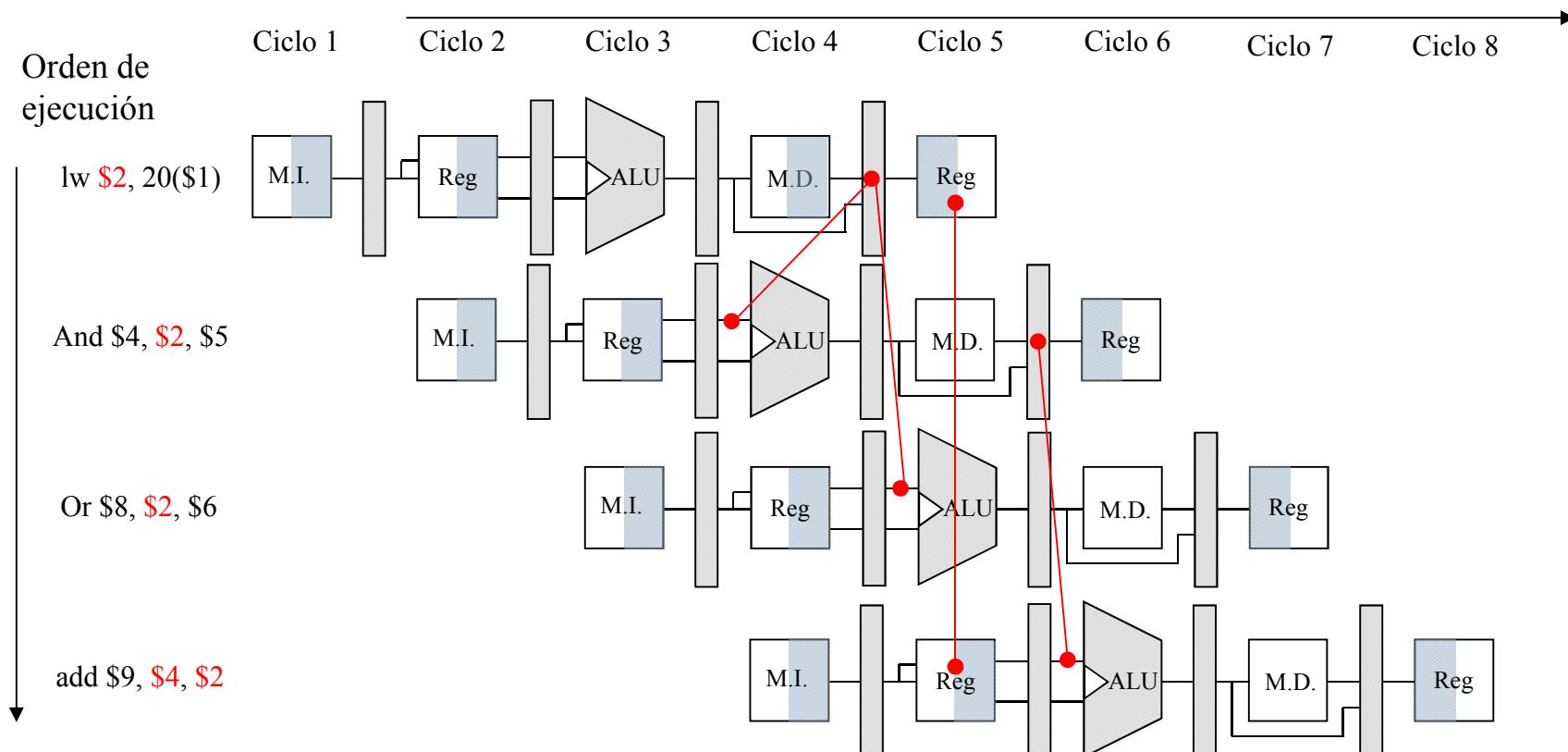
Si (**MEM/WB.RegWrite**
and **MEM/WB.Register_dst = ID/EX.RegisterRs**) **Ant_A** = 01

Si (**MEM/WB.RegWrite**
and **MEM/WB.Register_dst = ID/EX.RegisterRt**) **Ant_B** = 01

4.1.-Unidad de anticipación de operandos



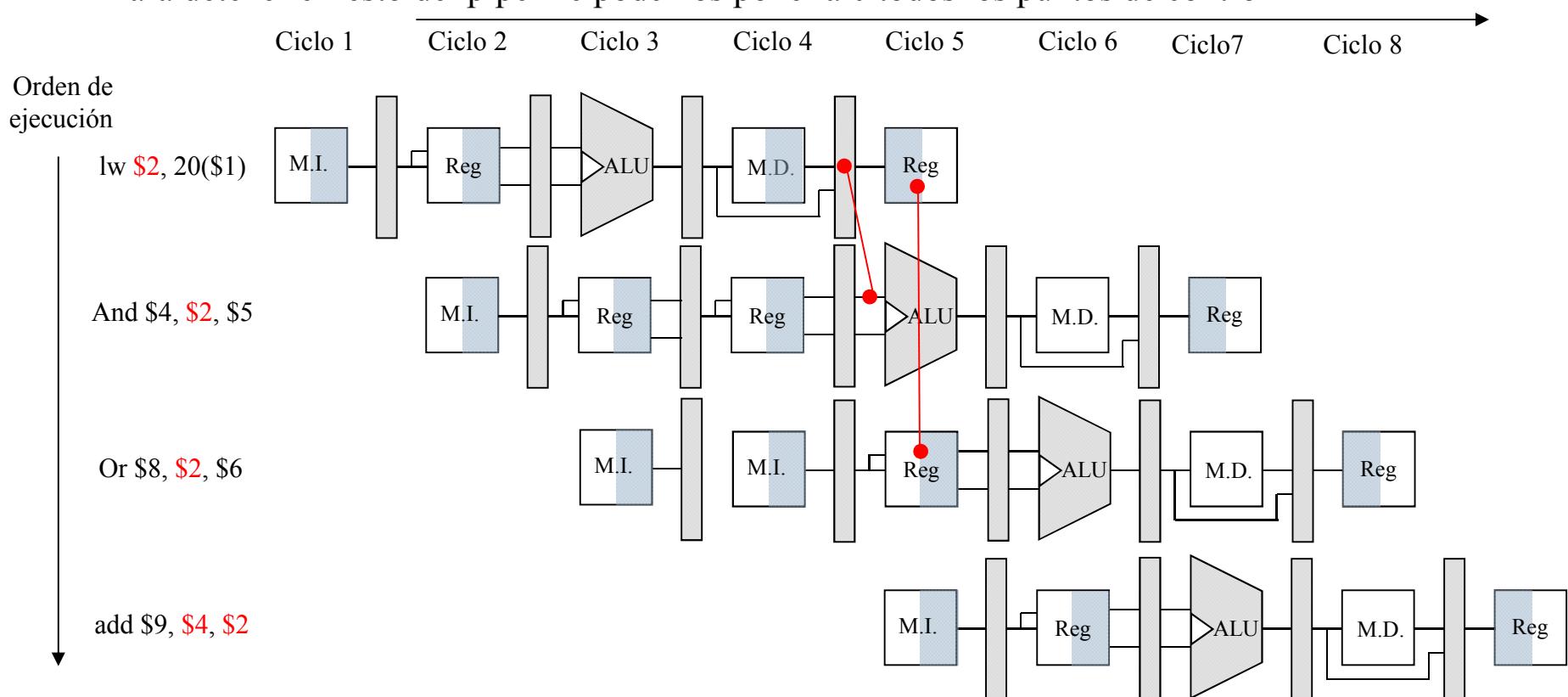
4.1.- La anticipación de operandos no siempre soluciona los riesgos



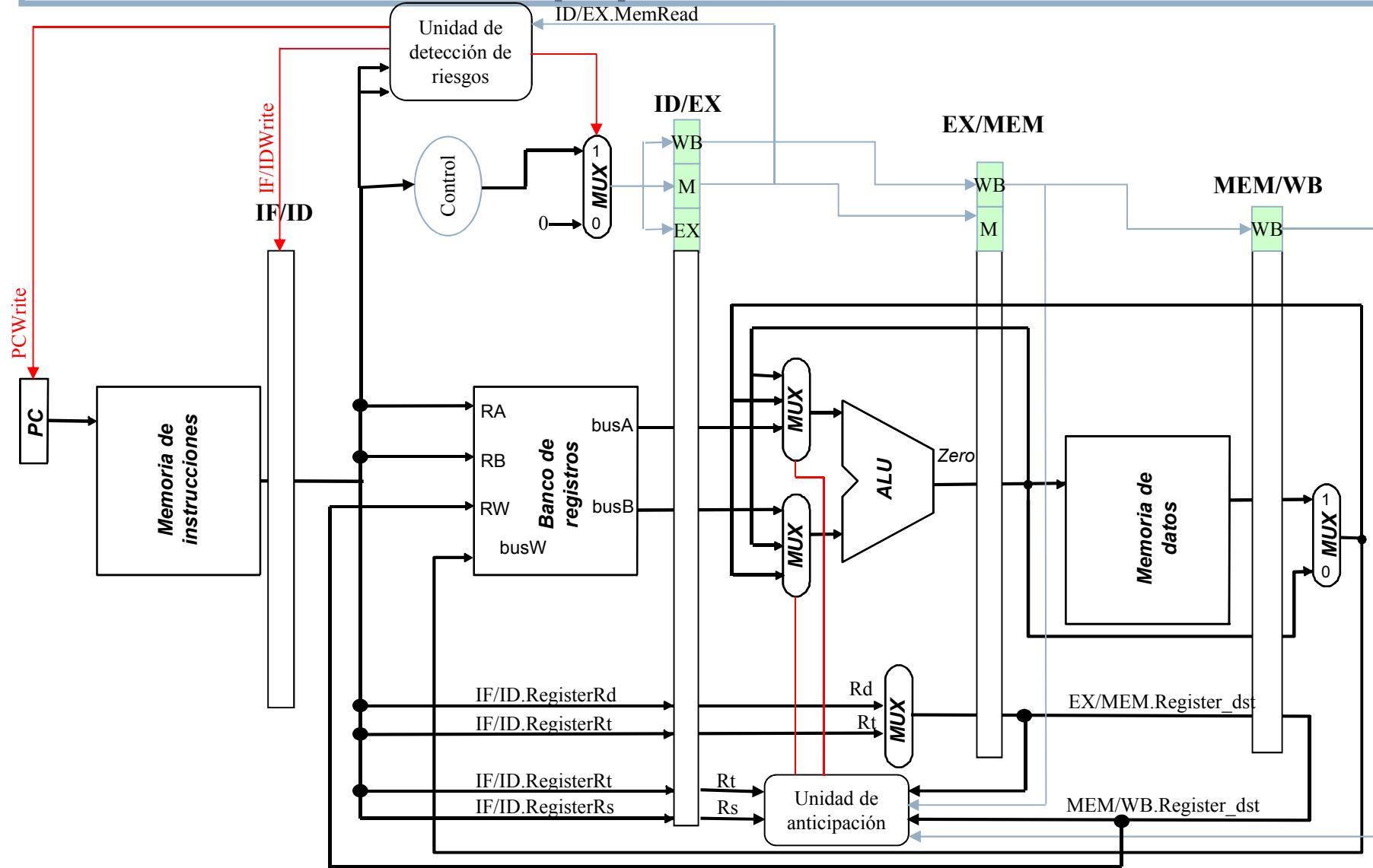
Necesitamos una unidad de detección de riesgos para **detener** a la instrucción load

4.1.- Detención del pipeline

- El conflicto LDE que se produce después de una instrucción **lw** no puede solucionarse.
- Hay que detener el pipeline manteniendo las instrucciones implicadas en la misma etapa, es decir evitando que el PC y el registro IF/ID cambien.
- Para detener el resto del pipeline podemos poner a 0 todos los puntos de control



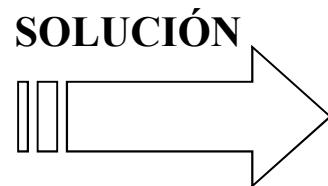
4.1.- Unidad de detención del pipeline



4.1.- Reordenamiento el código

- Estrategia en tiempo de compilación:
 - Reordenar el código para minimizar el número de detenciones

lw	\$2, 20(\$1)
and	\$12, \$2, \$5
or	\$13, \$6, \$2
add	\$14, \$5, \$13
sub	\$4, \$5, \$6



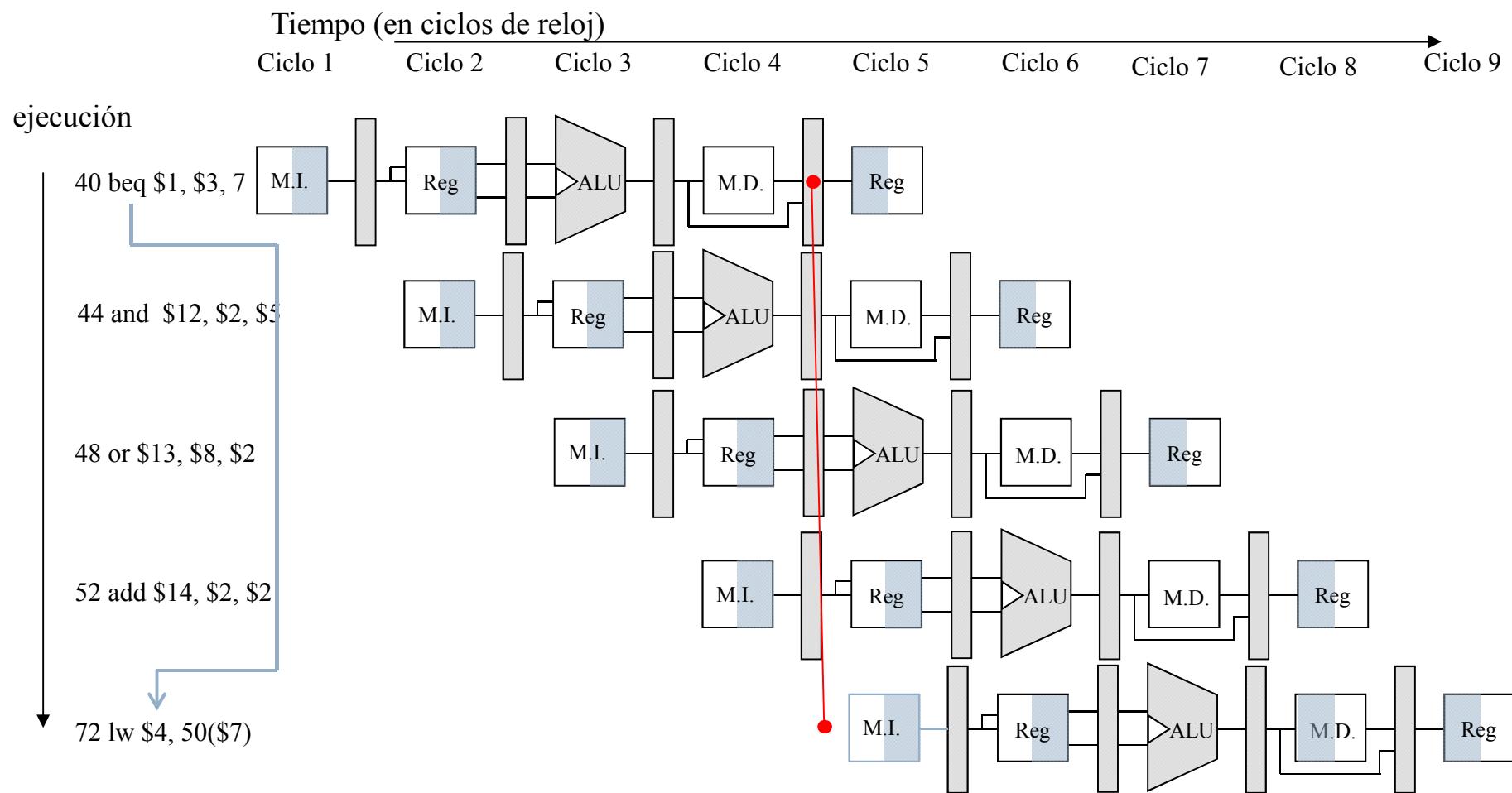
lw	\$2, 20(\$1)
sub	\$4, \$5, \$6
and	\$12, \$2, \$5
or	\$13, \$6, \$2
add	\$14, \$5, \$13

Estrategias comunes en buenos compiladores:

- Loop unrolling >> *lo veremos en problemas*
- Software pipelining

4.2.- Riesgos de control

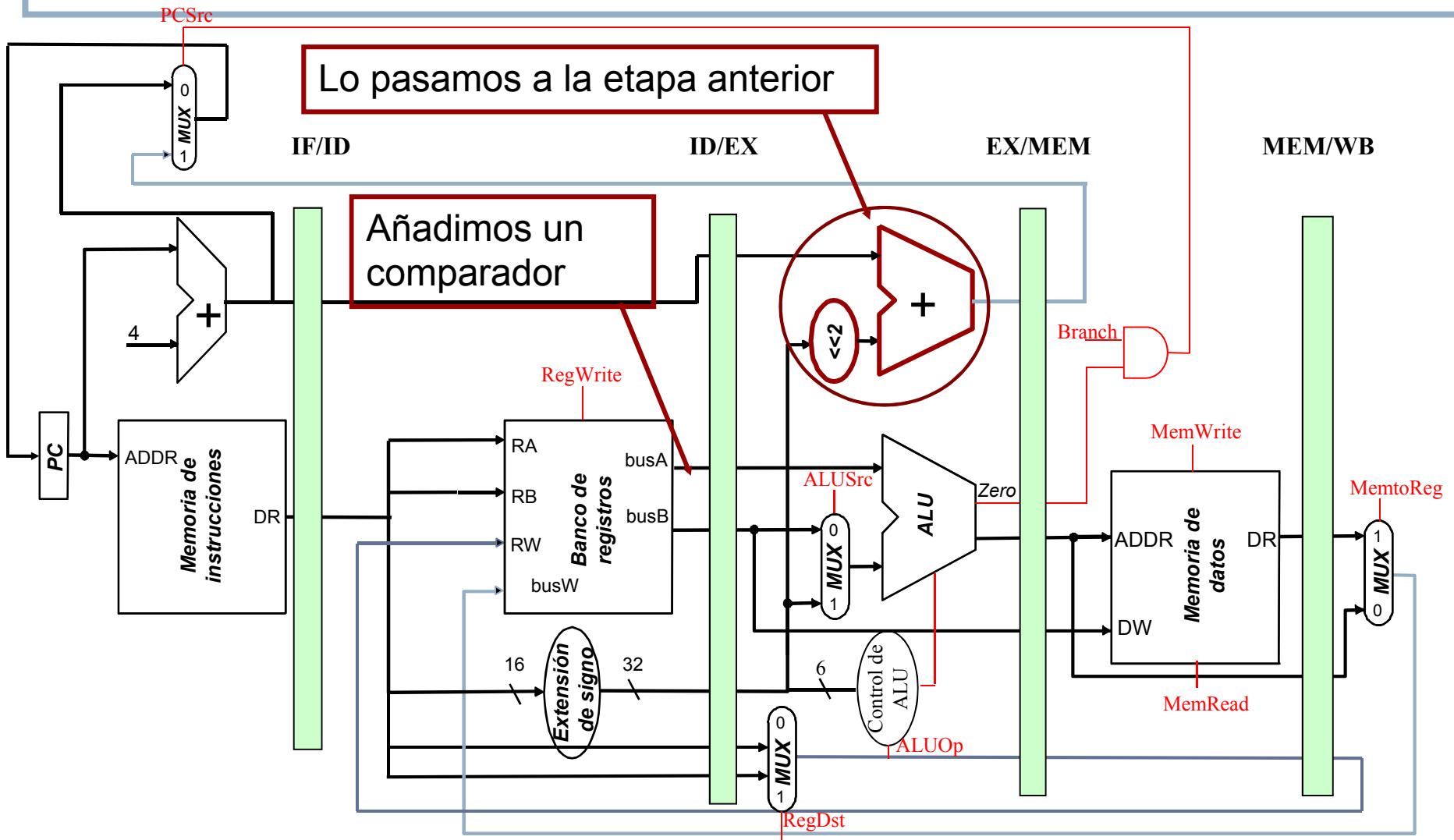
Salto condicional: no se conoce cuál es la siguiente instrucción hasta la etapa MEM



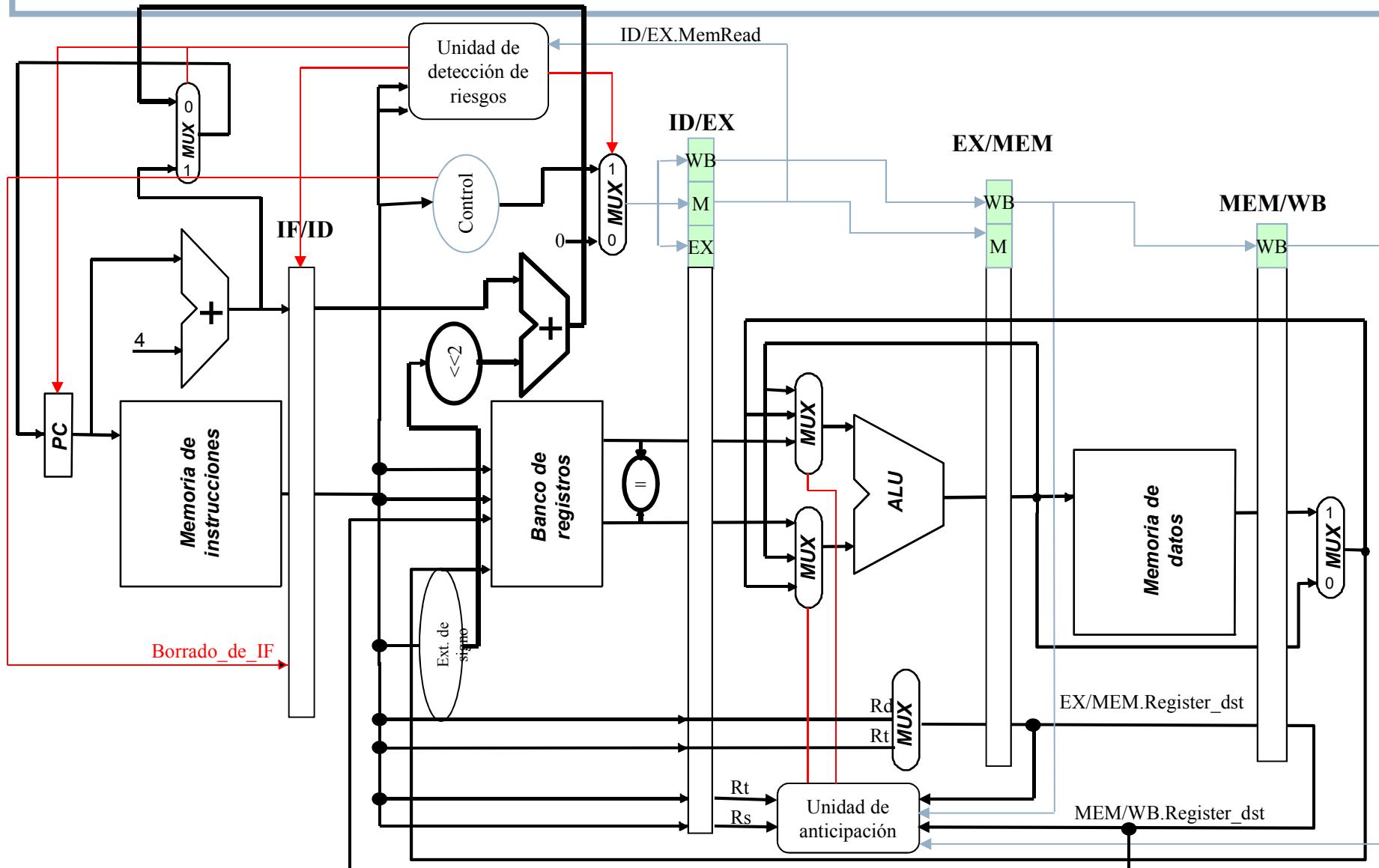
4.2.-Soluciones a los riesgos de control

- Reducir la penalización en los saltos
- Objetivo:
 - Calcular evaluación y destino lo antes posible
- Destino de salto
 - Etapa EX, pero tanto el sumador como la unidad << se podrían pasar a ID
- Evaluación del salto en BEQ (T / NT):
 - Comprobar si todos los bits de los dos operandos son iguales
 - Podemos realizarlo en ID con una XOR y una AND
 - La penalización sólo sería de 1 ciclo si salto NT

Cambios en la ruta de datos



4.2.-Soluciones a los riesgos de control



4.2.-Soluciones a los riesgos de control

- Detener el pipeline hasta que se conozca si el salto se produce.
 - Demasiado lenta.
- Asumir salto NT:
 - Buscar la siguiente instrucción secuencial
 - Si fallo de predicción (salto T):
 - Desechar todas las instrucciones que estén en ejecución equivocadamente.
 - Para ello se introducen ceros en sus valores de control.
 - Funciona bien si hay mucho saltos NT

4.2.- Predicción dinámica de saltos

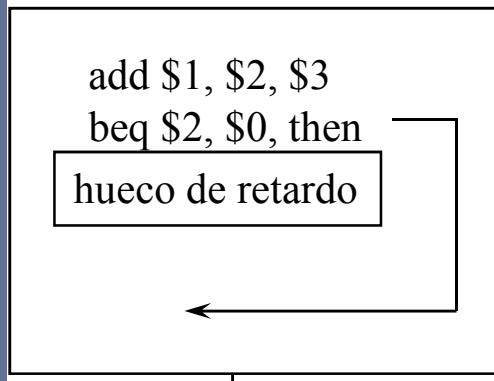
- Predicción dinámica de saltos:
 - Basada en los eventos pasados.
 - Predecimos que ocurrirá lo mismo que la vez anterior.
- El predictor almacena
 - la parte menos significativa de la dirección de la instrucción de salto
 - + un bit indicando si T o NT la última vez.
 - Por supuesto, esta información podría haberla guardado una instrucción diferente con los mismos bits finales (aliasing)
- Funciona bien en bucles

Saltos retardados

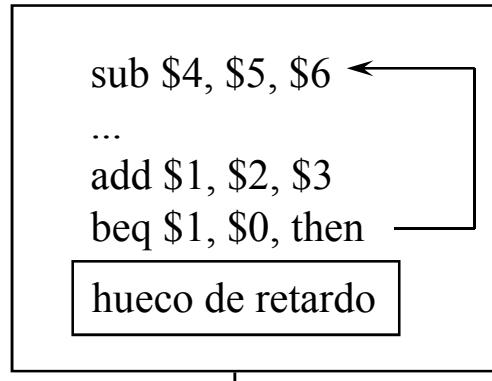
La siguiente instrucción a la de salto **siempre** se ejecuta.

El compilador debe seleccionar la instrucción adecuada:

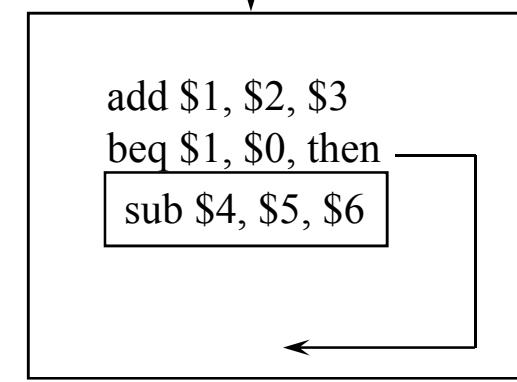
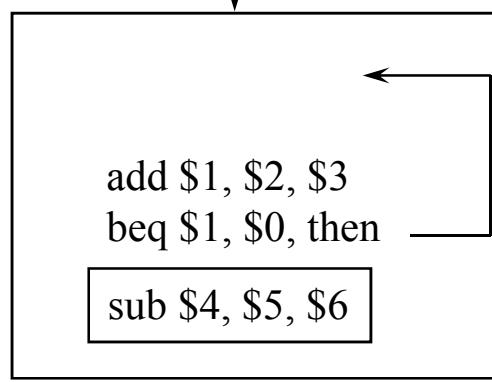
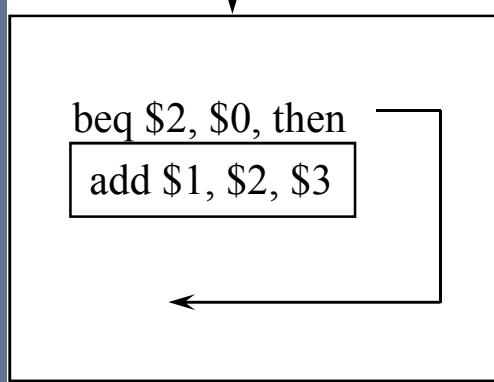
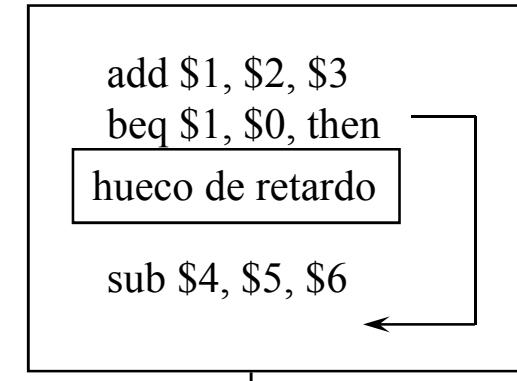
a) Anteriores al salto



b) Destino del salto



c) Destino del no salto



Conclusiones

- La segmentación es una herramienta poderosa para mejorar el rendimiento sin aumentar demasiado el coste
- Para aplicarla se requiere:
 - Añadir registros extra
 - No reutilizar HW
 - Hacer que la información de control fluya con los datos
- Para conseguir resultados óptimos las etapas deben tener una duración parecidas
- Surgen dos nuevos problemas:
 - Los riesgos de datos:
 - Se pueden reducir añadiendo HW que busque los operandos allá donde se encuentren y separando las instrucciones con dependencias
 - Los riesgos de control:
 - Se pueden reducir anticipando la ejecución de las instrucciones beq, con estrategias de compilación y tratando de predecir si el salto se toma o no
 - Aun así en ocasiones habrá que detener el pipe



AOC2 Segunda Parte – Jerarquía de Memoria

1. Principios de funcionamiento de la jerarquía de memoria
2. Organización de una cache: Correspondencia
3. Tamaño de bloque y reemplazo
4. Escrituras
5. Control - comportamiento
6. Caches multinivel
7. Memoria Principal



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza



Tema 5 – Principios de Funcionamiento de la Jerarquía de Memoria

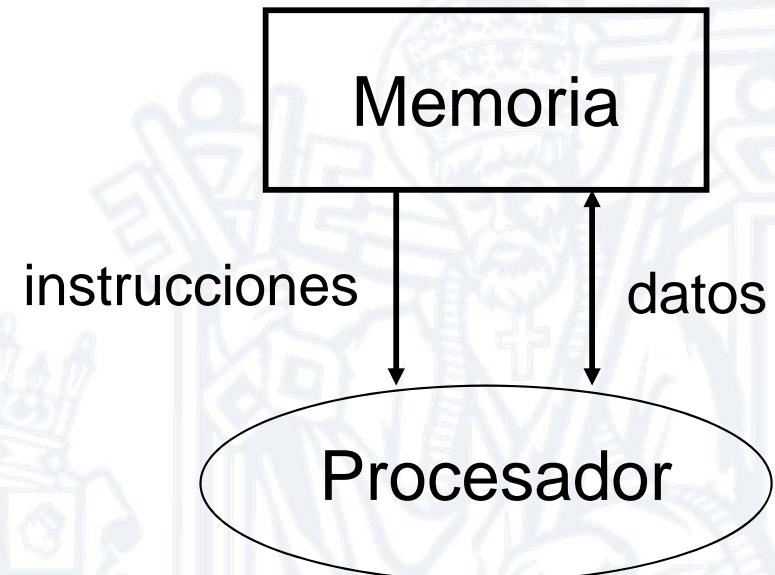
P. Ibáñez, J.L. Briz, V. Viñals, J. Alastruey, J. Resano
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

Objetivos del tema

- Motivar la existencia de la jerarquía de memoria
- Conocer los principios en que se basa la organización jerárquica
- Describir la estructura y el funcionamiento básico de una memoria cache

Guión del tema

- **Motivación: velocidad procesador vs. memoria**
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- Jerarquía de memoria
 - Memoria cache
- Ejemplos de memoria cache
- Descripción de una cache



Tiempo de ciclo del procesador

- Tabla de frecuencias y tiempo de ciclo de varios procesadores entre 1986 y 2001

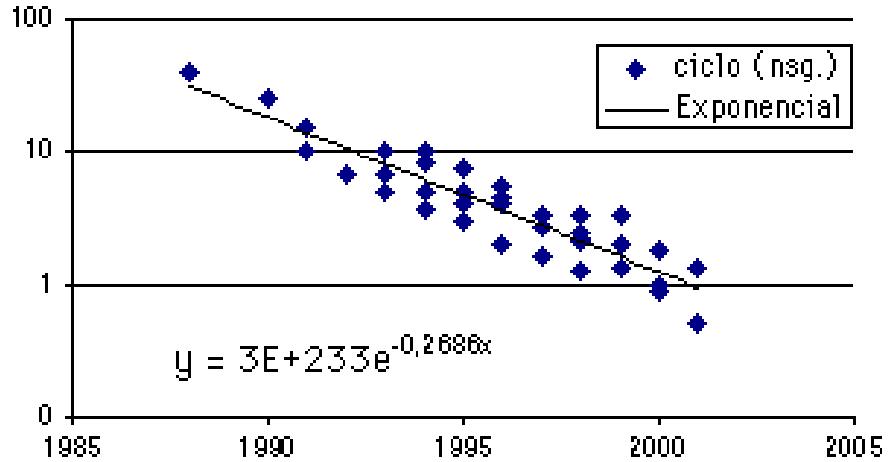
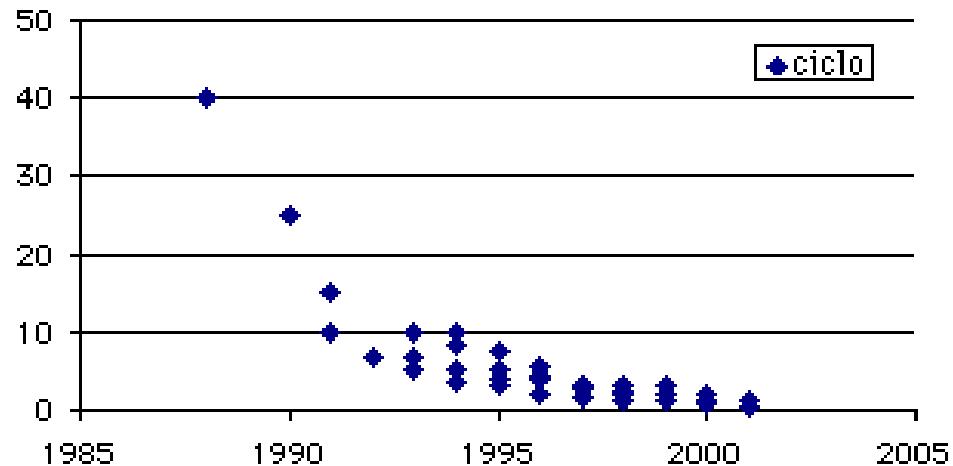
año	procesador	frecuencia (MHz)	ciclo (nsg.)
1988	motorola 88000	25	40,0
1990	intel i860	40	25,0
1991	HP 730	66	15,0
1991	mips R4000	100	10,0
1992	Alpha 21064	150	6,7
1993	Alpha 21064	200	5,0
1993	HP PA7100	100	10,0
1994	Alpha 21064	275	3,6
1994	HyperSparc	100	10,0
1995	PowerPC 604	133	7,5
1995	Alpha 21164	333	3,0
1996	Alpha 21164	500	2,0

año	procesador	frecuencia (MHz)	ciclo (nsg.)
1996	HP PA8000	180	5,6
1997	pentium II	300	3,3
1997	Alpha 21164	600	1,7
1998	Alpha 21264	800	1,3
1998	mips R12000	300	3,3
1999	pentium III	733	1,4
1999	Sparc Ultra II	300	3,3
2000	pentium III	1130	0,9
2000	HP PA8600	550	1,8
2001	pentium 4	2000	0,5
2001	PowerPC 74XX	733	1,4

Motorola
Intel
PA-Risc
Mips

Alpha
Sparc
PowerPC

Tiempo de ciclo del procesador



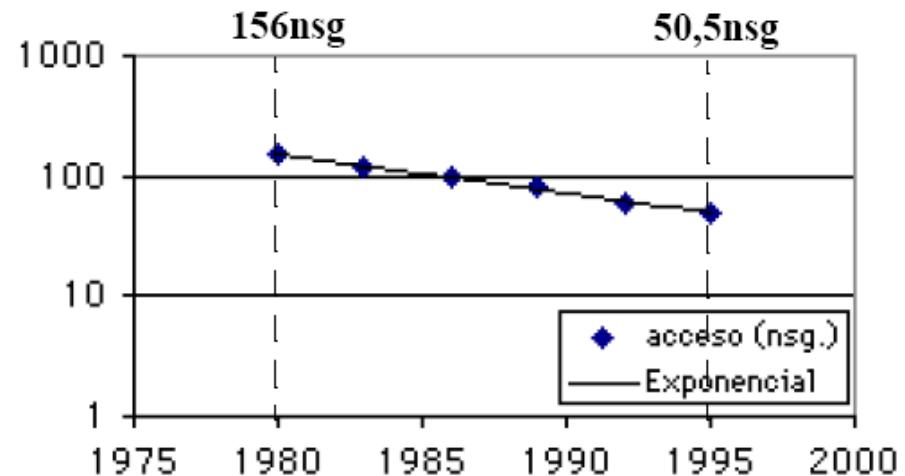
-23.6% cada año

- Expresión para Tiempo de ciclo (Tc) en función del año:
 - $T_c(\text{año}) = 21.9 * 0.764^{(\text{año}-1990)} \text{ ns}$
 - $T_c(2001) = 1.138 \text{ ns} \quad (880 \text{ MHz})$

Tiempo de acceso a memoria DRAM

- Tabla y gráfica de tiempos de acceso a memorias DRAM

DRAM: tiempo de acceso RAS	
año	tiempo de acceso
1980	150
1983	120
1986	100
1989	80
1992	60
1995	50

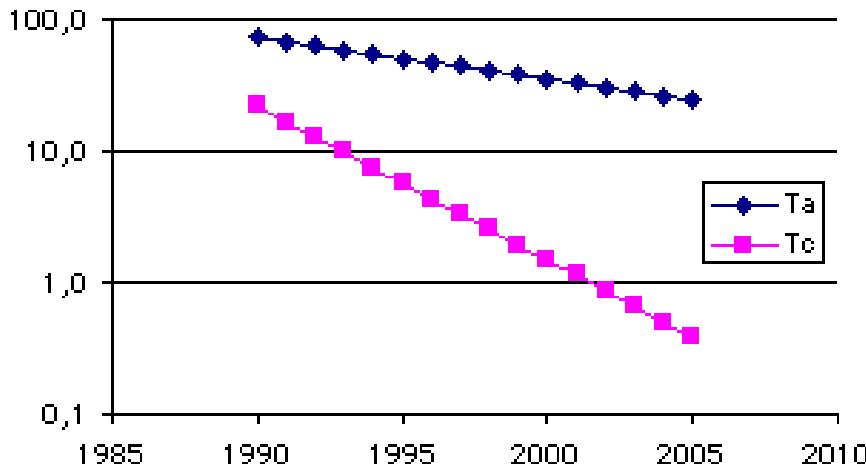


-7.2% cada año

- Expresión para Tiempo de acceso (Ta) en función del año:
 - $Ta(\text{año}) = 156 * 0,928^{(\text{año}-1980)} \text{ ns}$
 - $Ta(2001) = 32,5 \text{ ns} (307,7 \text{ MHz})$

Discrepancia entre procesador y memoria DRAM

- Speed gap,
memory wall

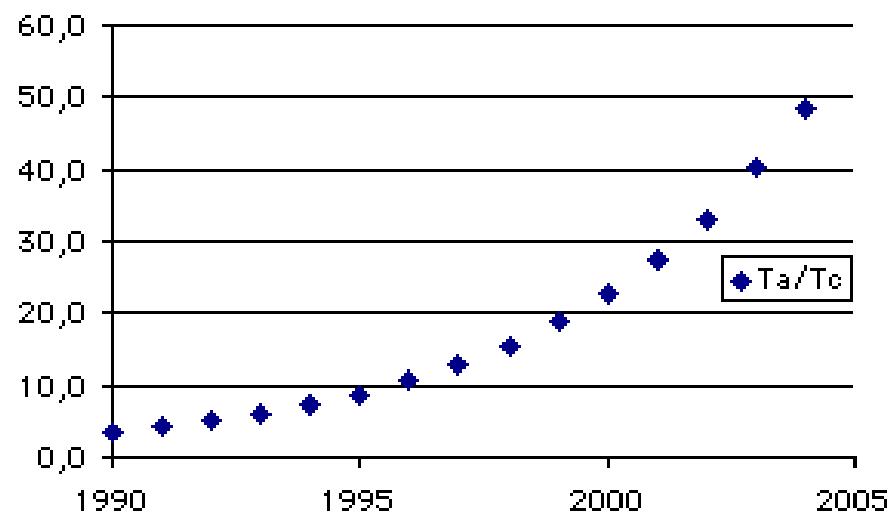


- Tiempo de acceso a memoria medido en ciclos de procesador:

$$\frac{\text{Tiempo de acceso a memoria}}{\text{Tiempo de ciclo de procesador}}$$

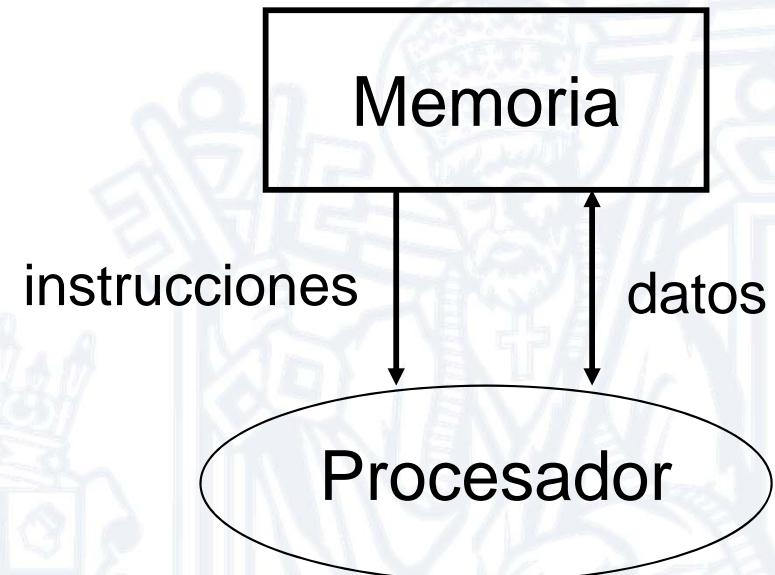
+21% cada año

- Ta (2001) = 27.4 ciclos por acceso



Guión del tema

- Motivación: velocidad procesador vs. memoria
- **Problema: cómo construir un almacén**
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- Jerarquía de memoria
 - Memoria cache
- Ejemplos de memoria cache
- Descripción de una cache



Problema: cómo construir un almacén

- *Grande*: datos e instrucciones del programa completo
- *Rápido*: velocidad del procesador
- *Barato*: poco coste por bit (= muchos bits / mm²)

Memoria dinámica
DRAM

Grande y barata

Memoria estática
SRAM

Rápida

- ... y que además *consume poca energía* ? ...

Memoria SRAM vs. DRAM¹

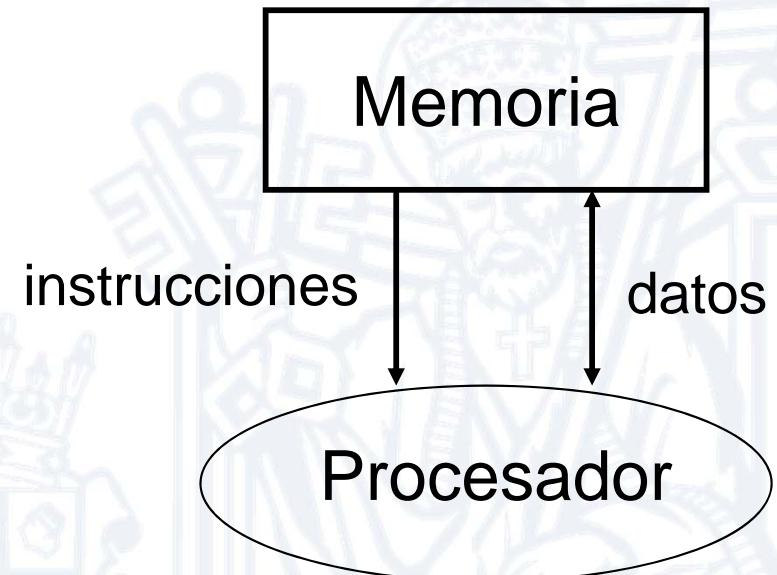
	Celda de almacenamiento	Capacidad (chip 2008)	Precio/MByte	Velocidad
DRAM	1 Condensador + 1 Transistor alta densidad	grande 2048 Mbits	barato 0,18\$	lenta 12.5 ns
SRAM	4Trans. + 2Trans. baja densidad (16-25x DRAM)	pequeña 32 Mbits	caro ~ 18\$	rápida 0.4 ns

- Ejemplo: 2048 Mbits de memoria principal para un PC
 - DRAM: 16 chips, 20 €
 - SRAM: 64 chips, ~2.000 € (¡ sólo los chips !)
- Otros problemas de SRAM:
 - Encaminamiento aumentaría mucho el tiempo de acceso de SRAM
 - Consumo mucho mayor en SRAM

1. The Stacked Capacitor DRAM Cell and Three-Dimensional Memory. Mitsumasa Koyanagi, January 2008 IEEE Solid State Circuits Society Journal.
http://www.ieee.org/portal/site/sscs/menuitem.f07ee9e3b2a01d06bb9305765bac26c8/index.jsp?&pName=sscs_level1_article&TheCat=2171&path=sscs/08Winter&file=Koyanagi.xml

Guión del tema

- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- **Propiedad de los programas**
 - **Localidad espacial y temporal**
- Jerarquía de memoria
 - Memoria cache
- Ejemplos de memoria cache
- Descripción de una cache



Localidad espacial y temporal

- Propiedad de los programas:

- Si un procesador referencia una posición de memoria

- Es probable que vuelva a referenciar pronto

- la misma** posición de memoria

- Localidad temporal**

- Es probable que refuerce pronto

- posiciones de memoria **cercanas**

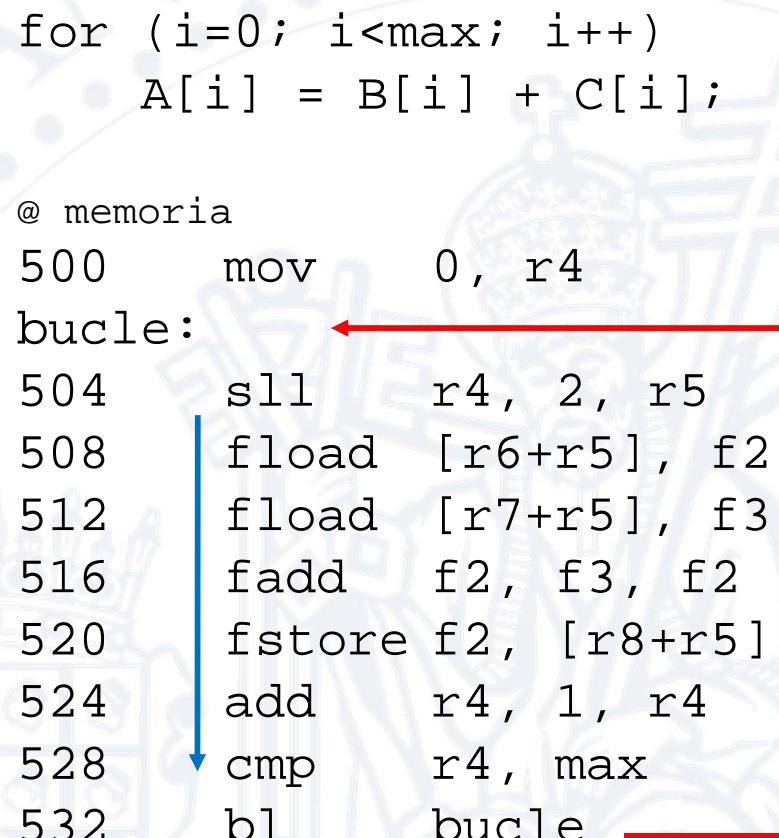
- Localidad espacial**

Localidad en código

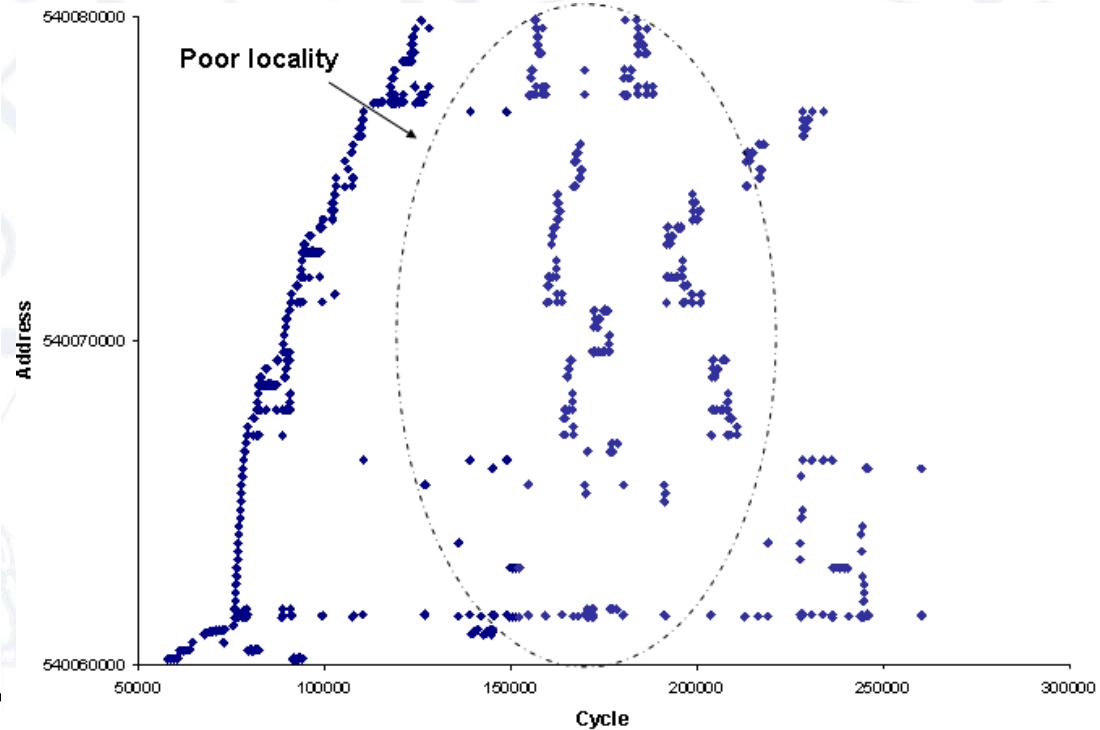
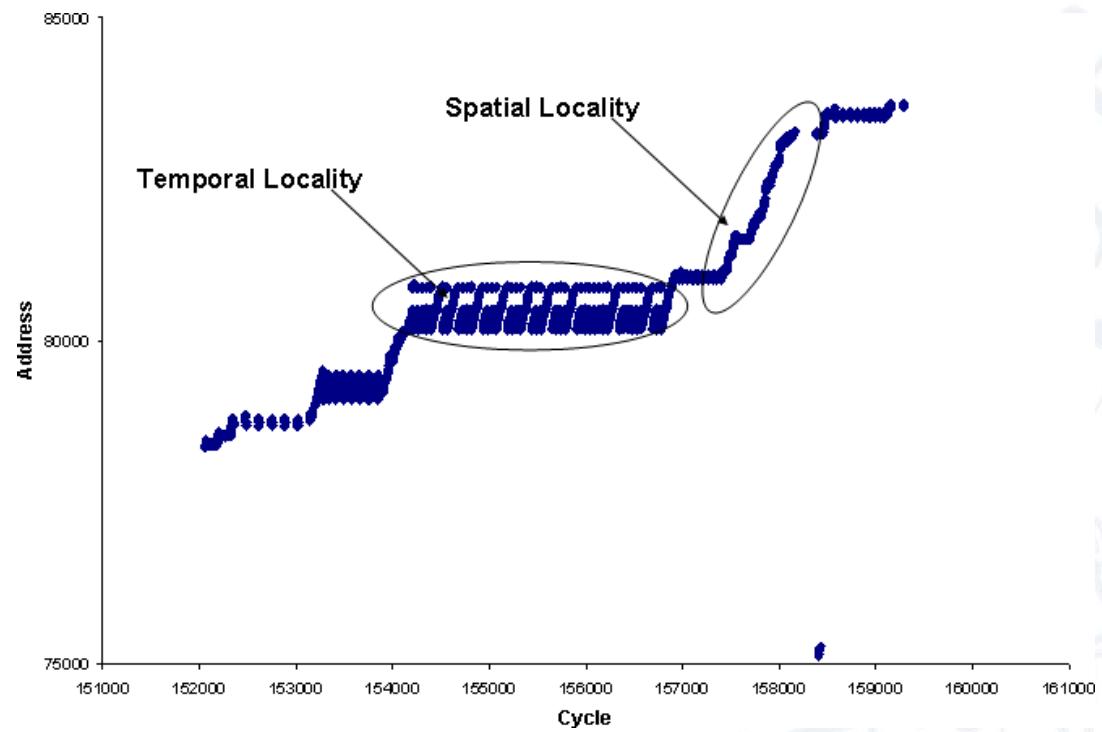
- Secuenciamiento implícito
 - Después de la instrucción i se ejecuta la instrucción $i+1$
=> Localidad espacial
- Ejecución de bucles
 - Cada N instrucciones se ejecuta la instrucción i
=> Localidad temporal

```
for (i=0; i<max; i++)
    A[i] = B[i] + C[i];

@ memoria
500    mov    0, r4
bucle:
504    sll    r4, 2, r5
508    fload [r6+r5], f2
512    fload [r7+r5], f3
516    fadd   f2, f3, f2
520    fstore f2, [r8+r5]
524    add    r4, 1, r4
528    cmp    r4, max
532    bl     bucle
```



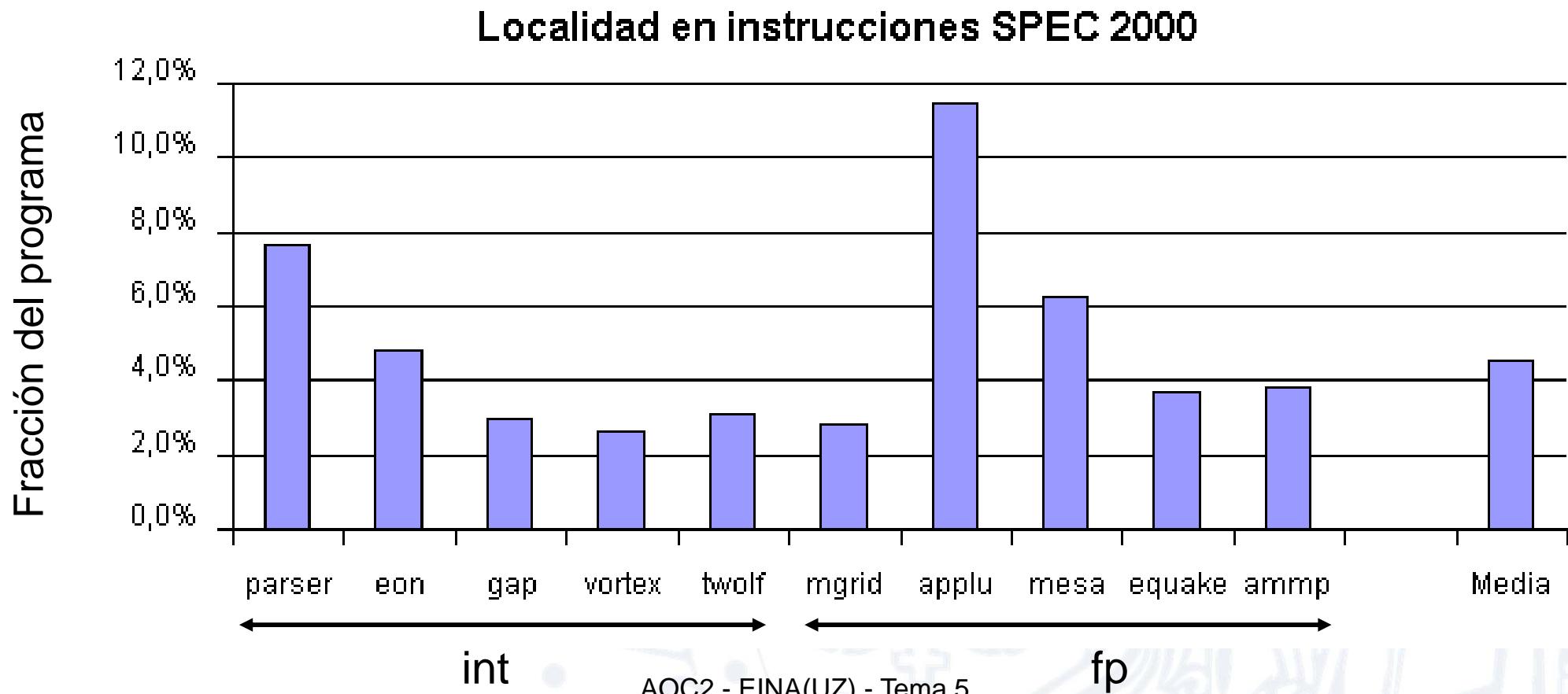
Localidad de instrucciones en acción



Localidad en código

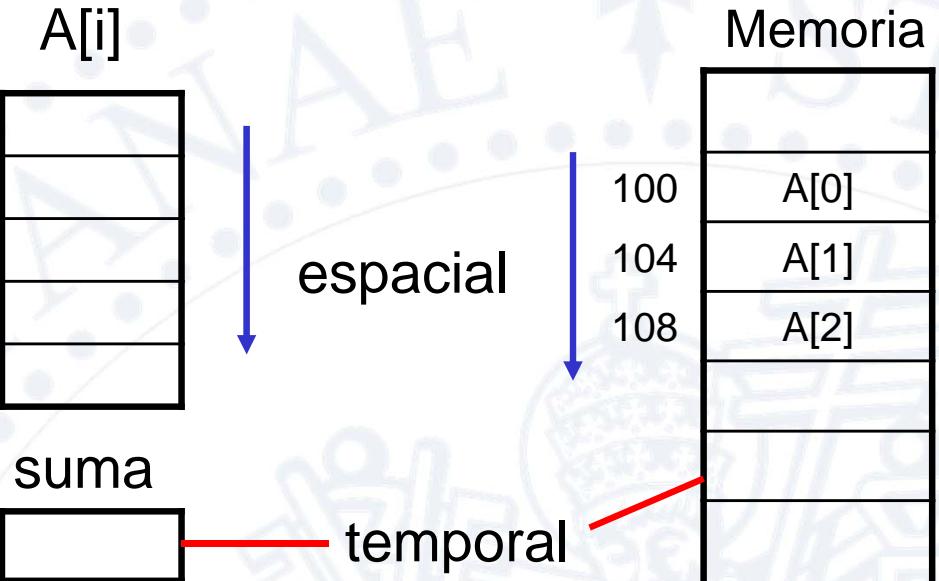
■ Regla empírica

El 90% de las instrucciones ejecutadas (dinámico) es atribuible a un 10% de las instrucciones (estático)



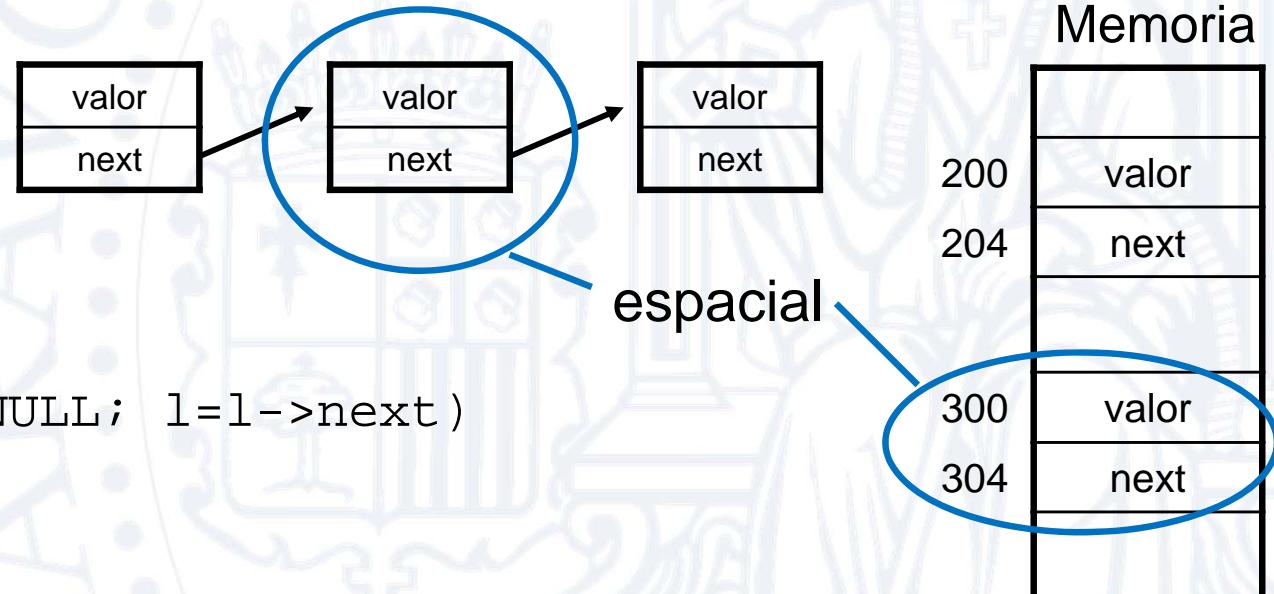
Localidad en datos

```
for (i=0; i<max; i++)  
    suma += A[i];
```



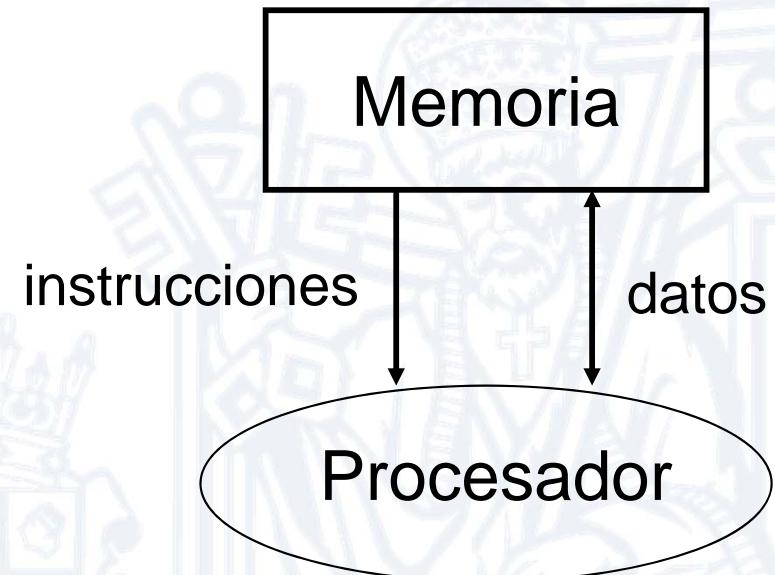
```
struct nodo {  
    int valor;  
    struct nodo *next;  
}
```

```
for (l=list; l->next!=NULL; l=l->next)  
    suma += l->valor;
```



Guión del tema

- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- **Jerarquía de memoria**
 - **Memoria cache**
- Ejemplos de memoria cache
- Descripción de una cache



Jerarquía de memoria: Cache

Características de las memorias

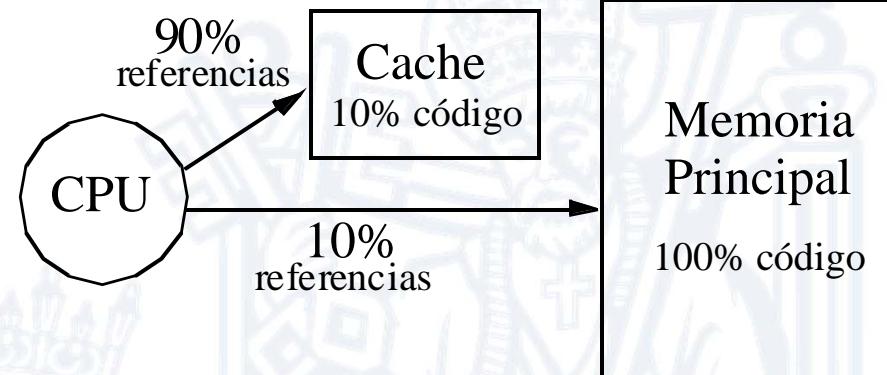
- DRAM barata y lenta
- SRAM cara y rápida

Propiedad de los programas

- Localidad espacial y temporal
(+ regla 90/10)

■ Solución: dos almacenes

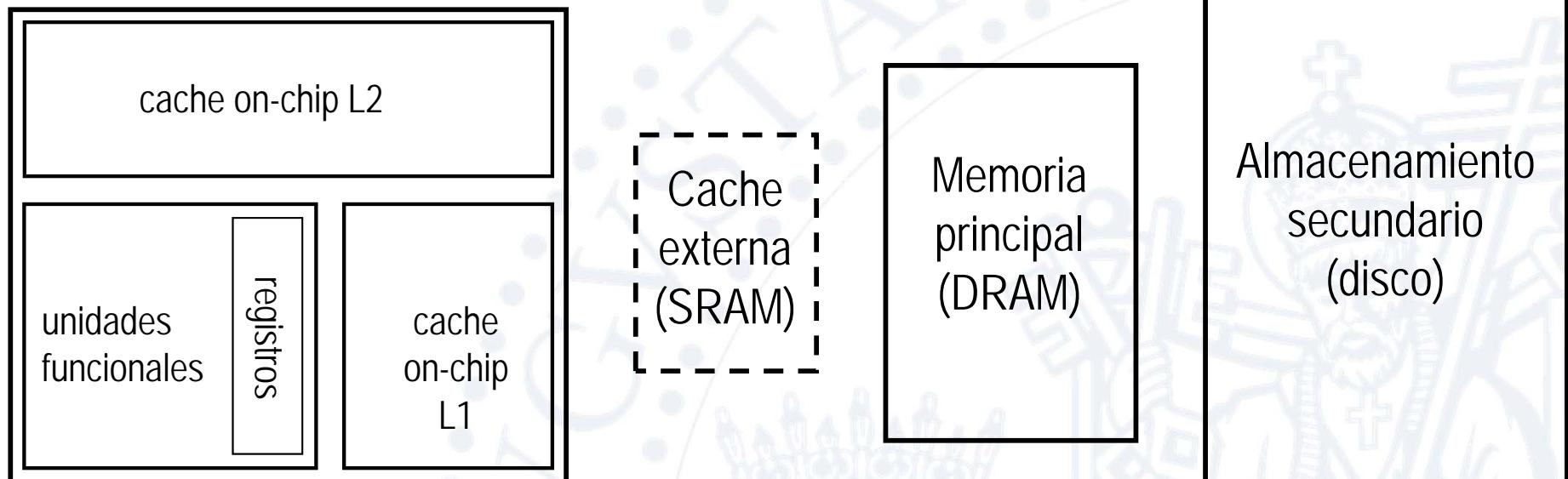
- Mem. principal: DRAM, grande y lenta
almacén global
- Mem. cache: SRAM, pequeña y rápida
subconjunto más usado



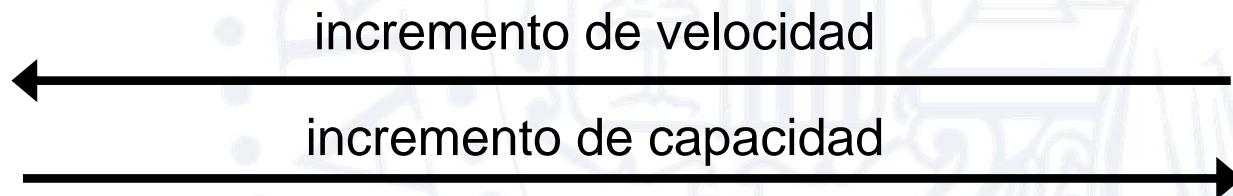
Memoria Cache: memoria pequeña y rápida que contiene el subconjunto más usado de memoria principal

Jerarquía de memoria

- Varios niveles de almacenamiento
procesador



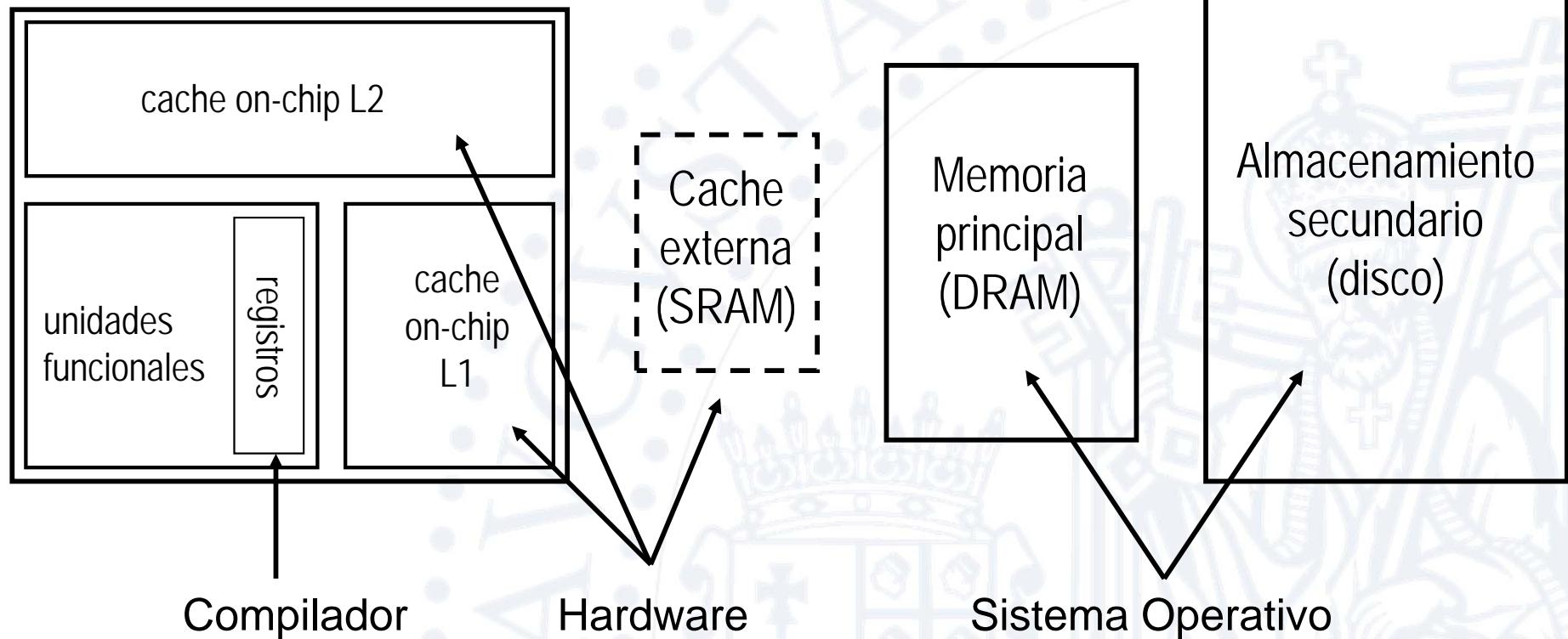
Tiempo (ciclos)	1	1-2	5-10	40	~ 80	10 M
Capacidad (bytes)	160 * 8	8-64 KB	~ 1 MB	~ 32 MB	Gigabytes	Terabytes



Jerarquía de memoria

■ Responsable de cada nivel

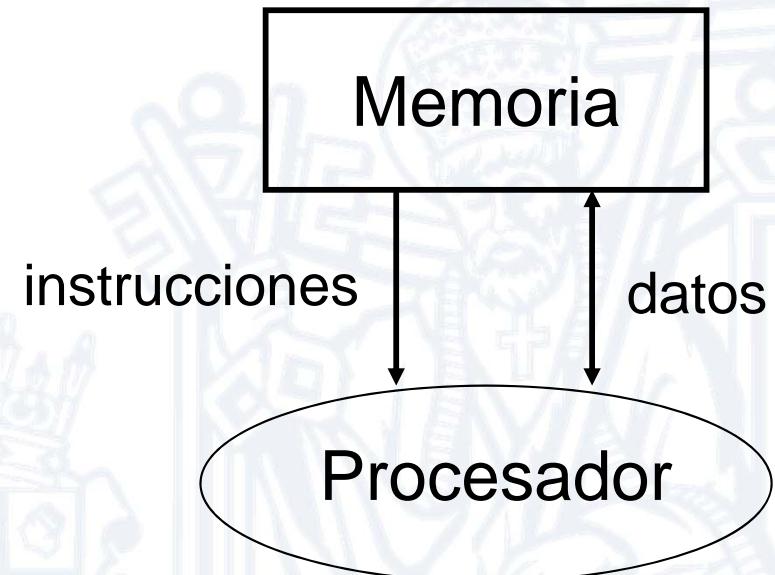
procesador



La memoria cache es transparente al nivel de lenguaje máquina

Guión del tema

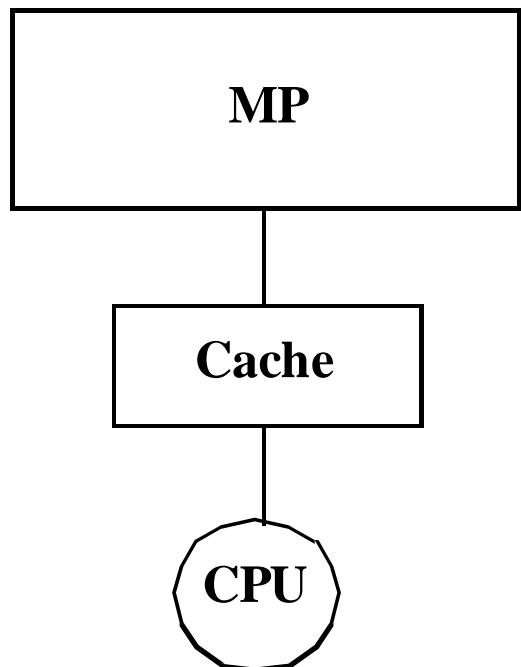
- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- Jerarquía de memoria
 - Memoria cache
- **Ejemplos de memoria cache**
- Descripción de una cache



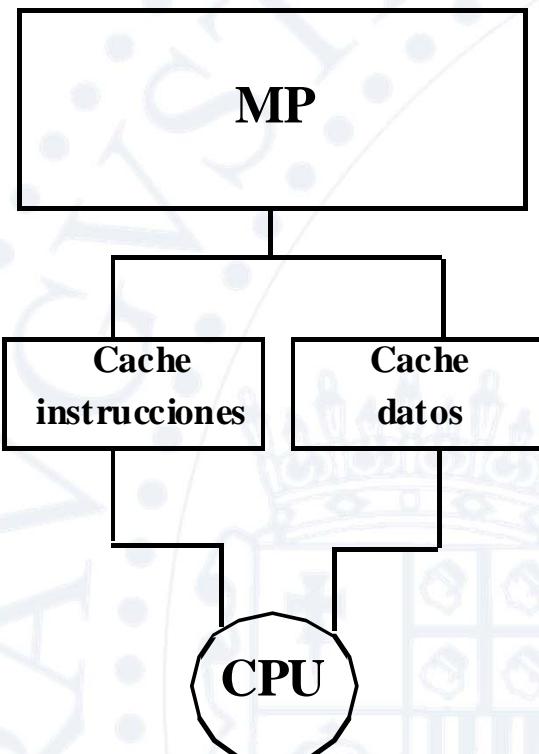
Ejemplos de memoria cache

■ Organización de la memoria cache

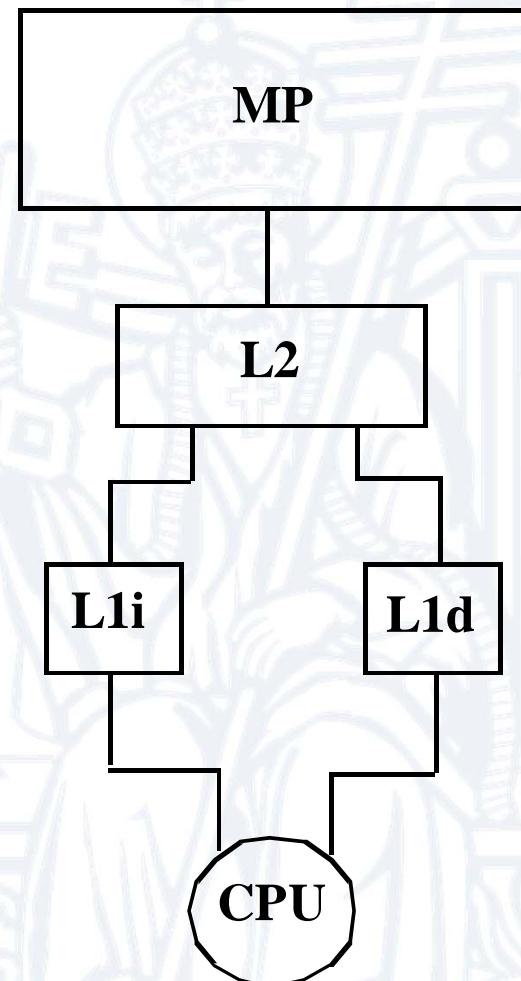
Cache unificada



Cache separada

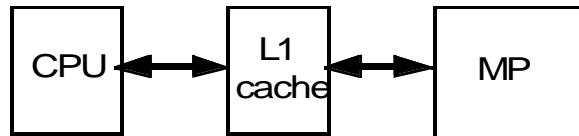


Cache multinivel

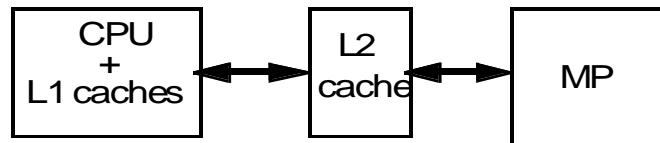


Evolución de la jerarquía en la familia Intel

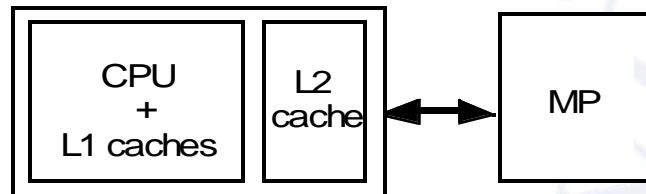
486 (1989)



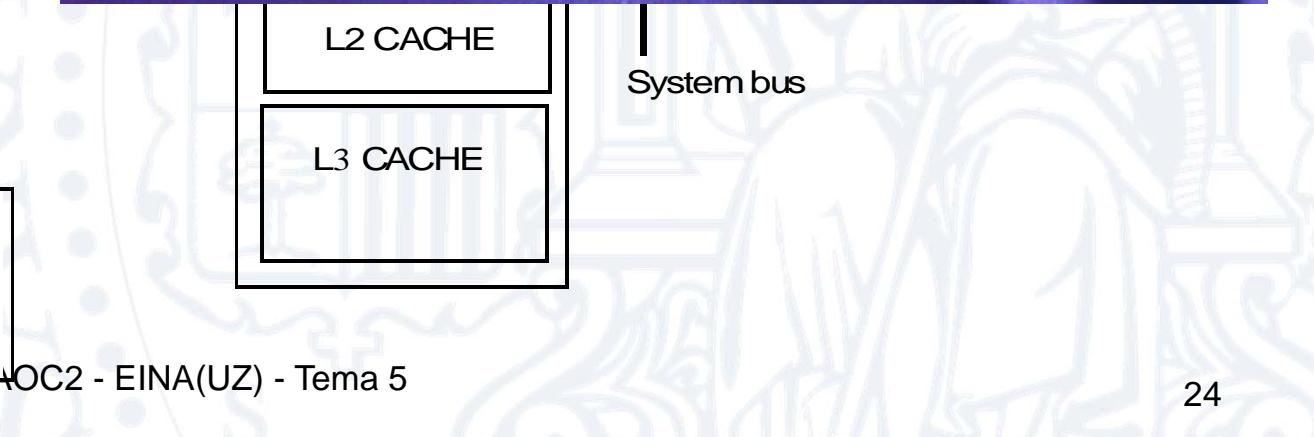
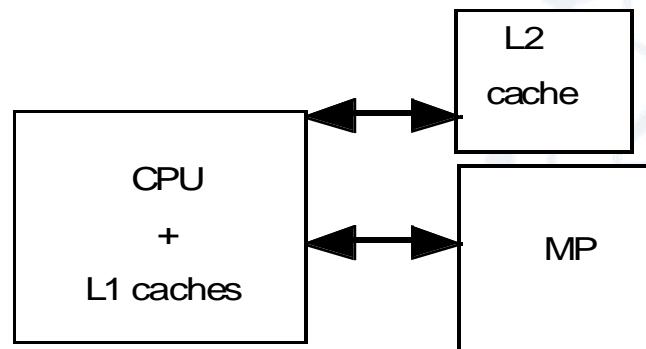
Pentium (1993)



Pentium Pro (1995)



Pentium II-III (1997-1999)



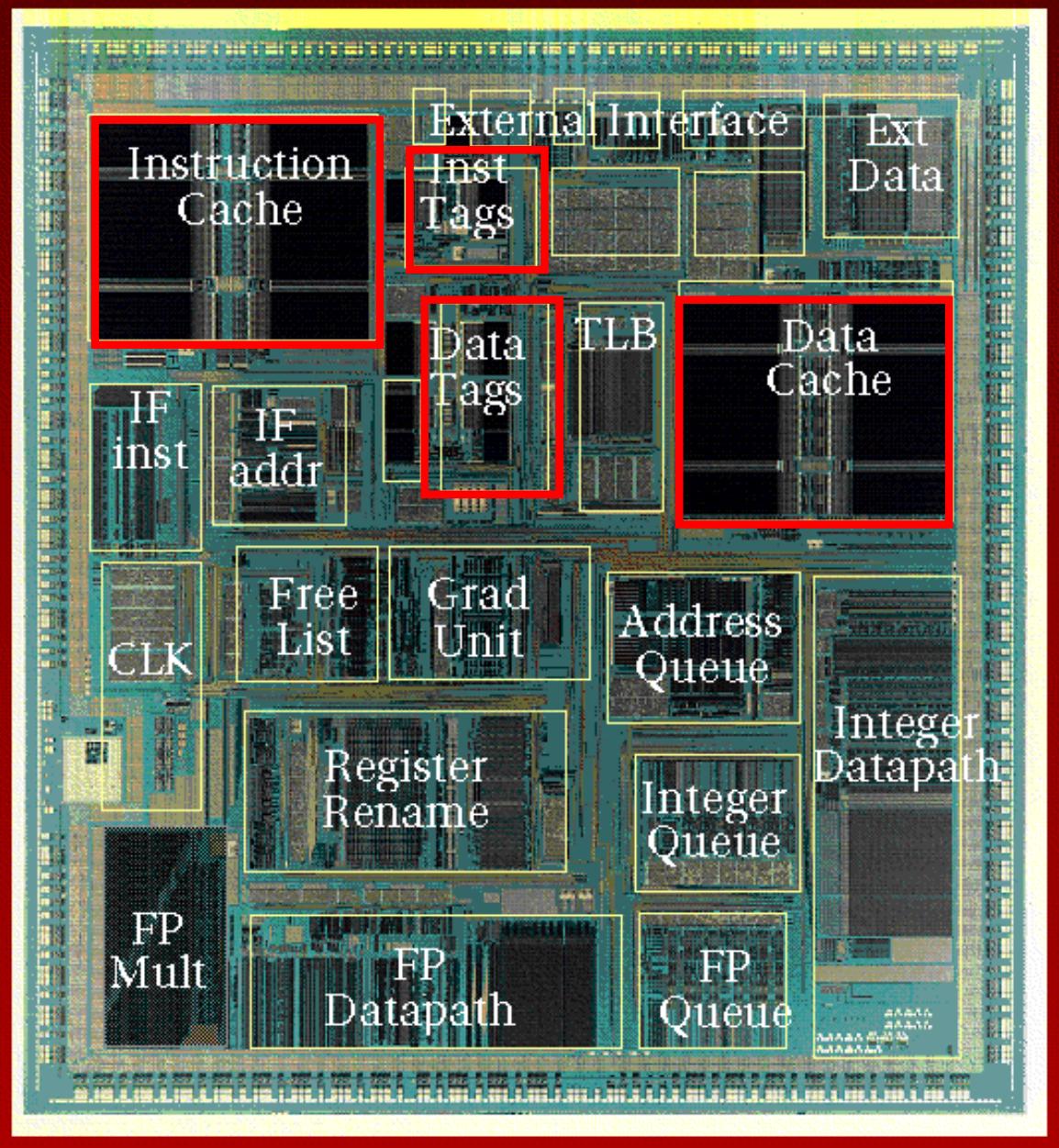
Ejemplos de memoria cache: capacidad

■ Chart Watch (nov-2009)

Procesador	L1i + L1d	L2	L3
Intel 2-core Xeon X5270	2 x (32 + 32) KB	6 MB	-
AMD 2-core Opteron X5270	2 x (64 + 64) KB	2 x 1 MB	-
Intel 4-core Xeon X7350	4 x (32 + 32) KB	2 x 4 MB	-
AMD 4-core Opteron 8360E	4 x (64 + 64) KB	4 x 512 KB	2 MB
Intel 6-core Xeon X7460	6 x (32 + 32) KB	3 x 3 MB	16 MB
Intel Itanium 2 9150M	2 x (16 + 16) KB	2 x (1MB + 256 KB)	12 MB
IBM Power6 (2-core)	2 x (64 + 64) KB	2 x 4 MB	32 M (off)
Fujitsu SPARC64 VII	4 x (64 + 64) KB	6 MB	-
Sun UltraSPARC T2+ (8-core)	8 x (8 + 16) KB	4 MB	-

MIPS R10K (1996-1998)

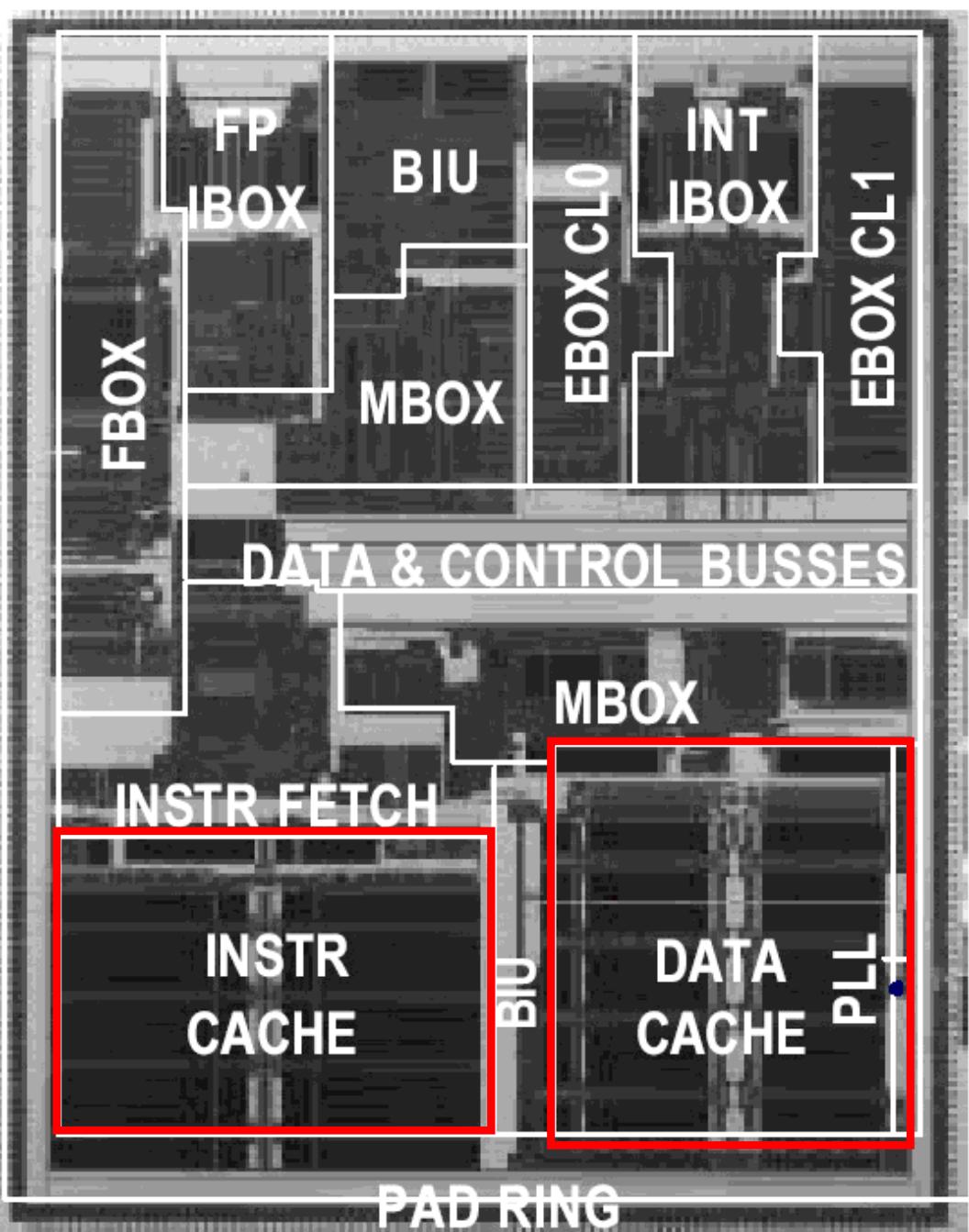
R10K die size 16.6mm x 17.9mm



	i	d
Capacidad	32 KB	32 KB
Total		64 KB
% transistores		65%
% area		~19%

6,8 Mtransistores
297 mm²

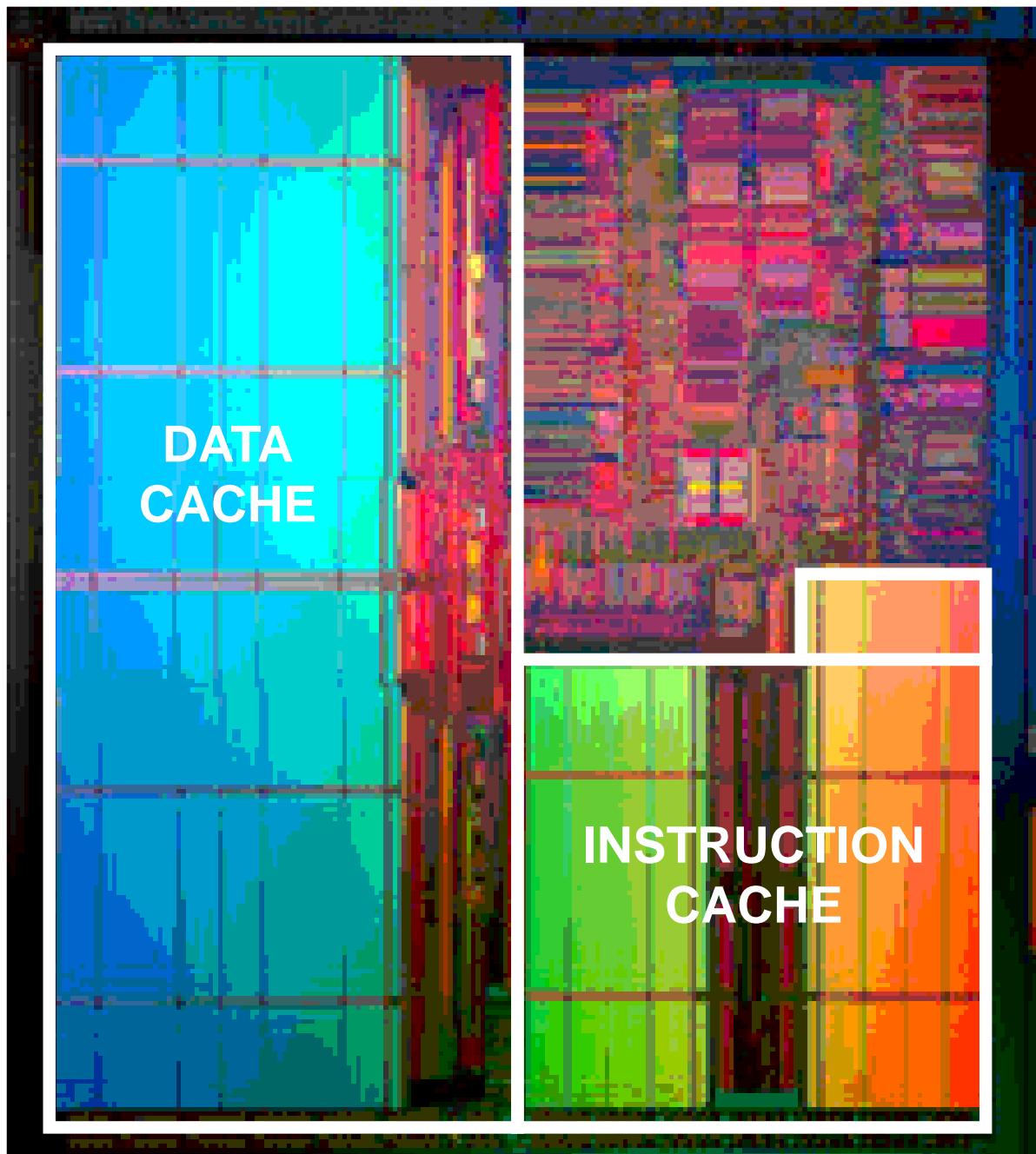
Alpha 21264 (1998-2000)



	i	d
Capacidad	64 KB	64 KB
Total		128 KB
% transistores		nd
% area		~27%

15 Mtransistores
15.8 x 19 mm = 300 mm²

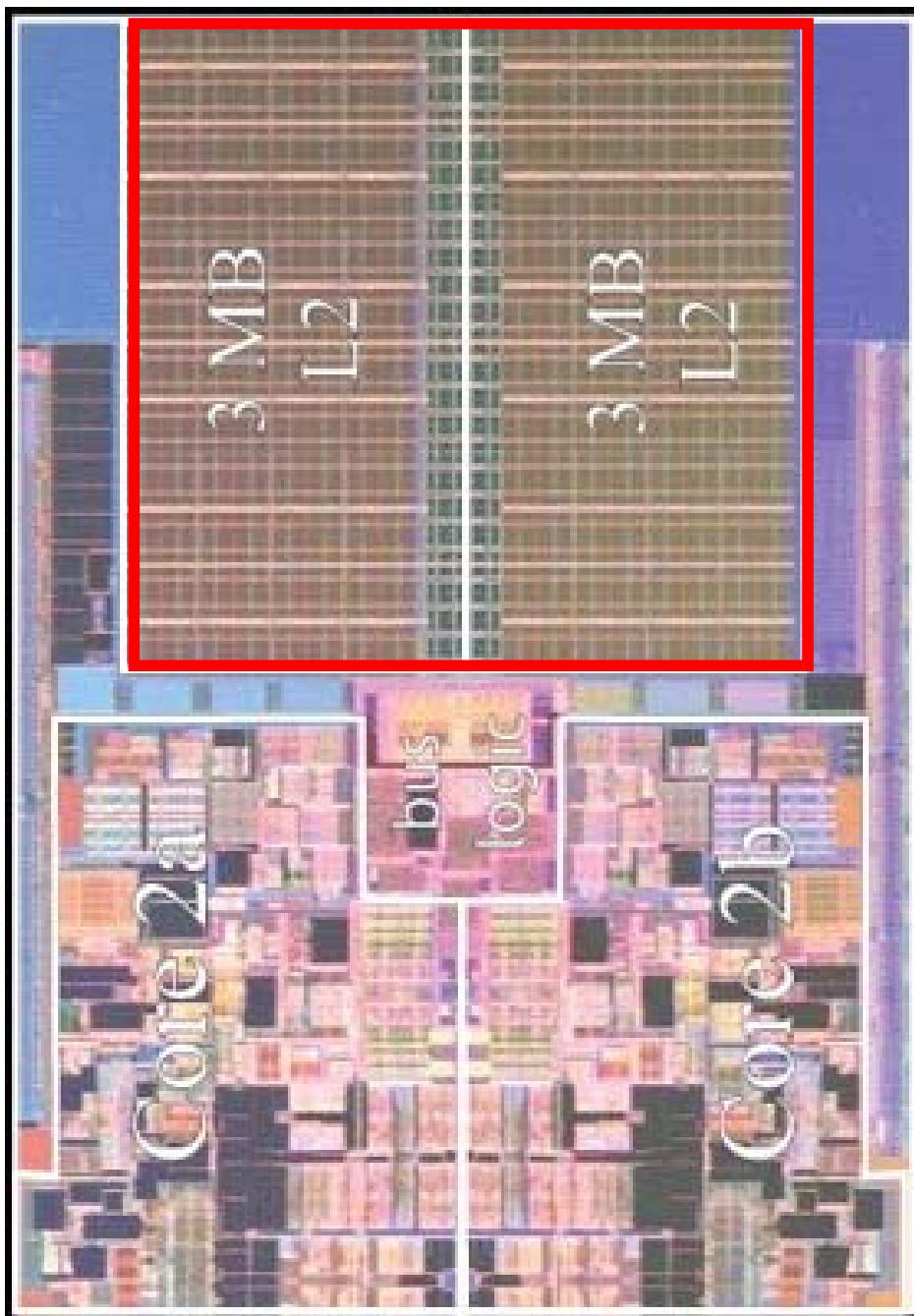
HP PA-8700 (2001-2002)



	i	d
Capacidad	768 KB	1536 KB
Total	2.25 MB	
% transistores	~90%	
% area	~75%	

186 Mtransistores
16 x 19 mm = 304 mm²

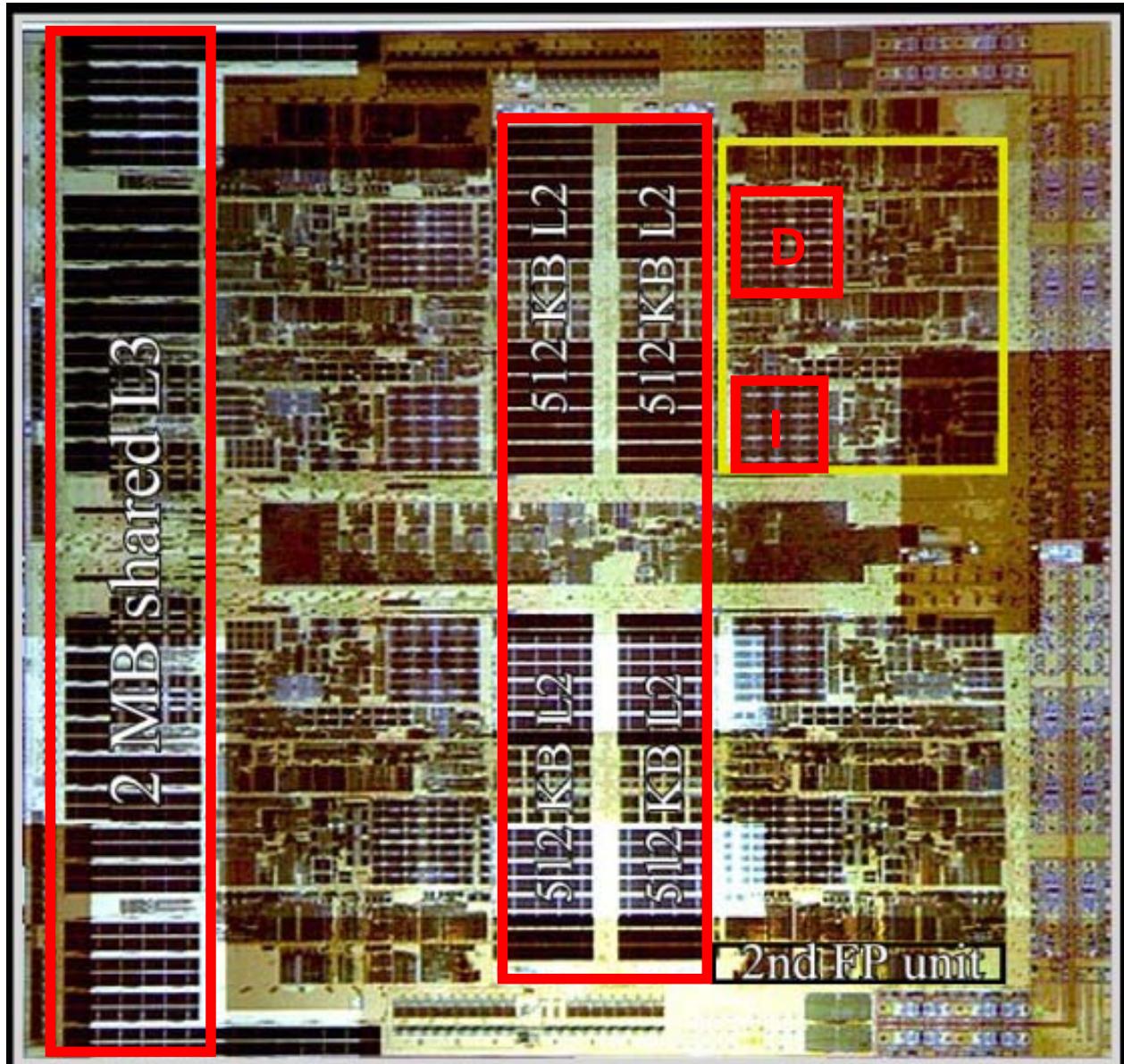
Intel Penryn Dual-Core (2008)



	i	d
L1	2x32 KB	2x32 KB
L2	2 x 3 MB	
L3	-	
Total	6.125 MB	
% transist.	~80%	
% area	~60% (a ojo)	

410 Mtransistores
 $12.3 \times 8.6 \text{ mm} = 107 \text{ mm}^2$ @ 65 nm

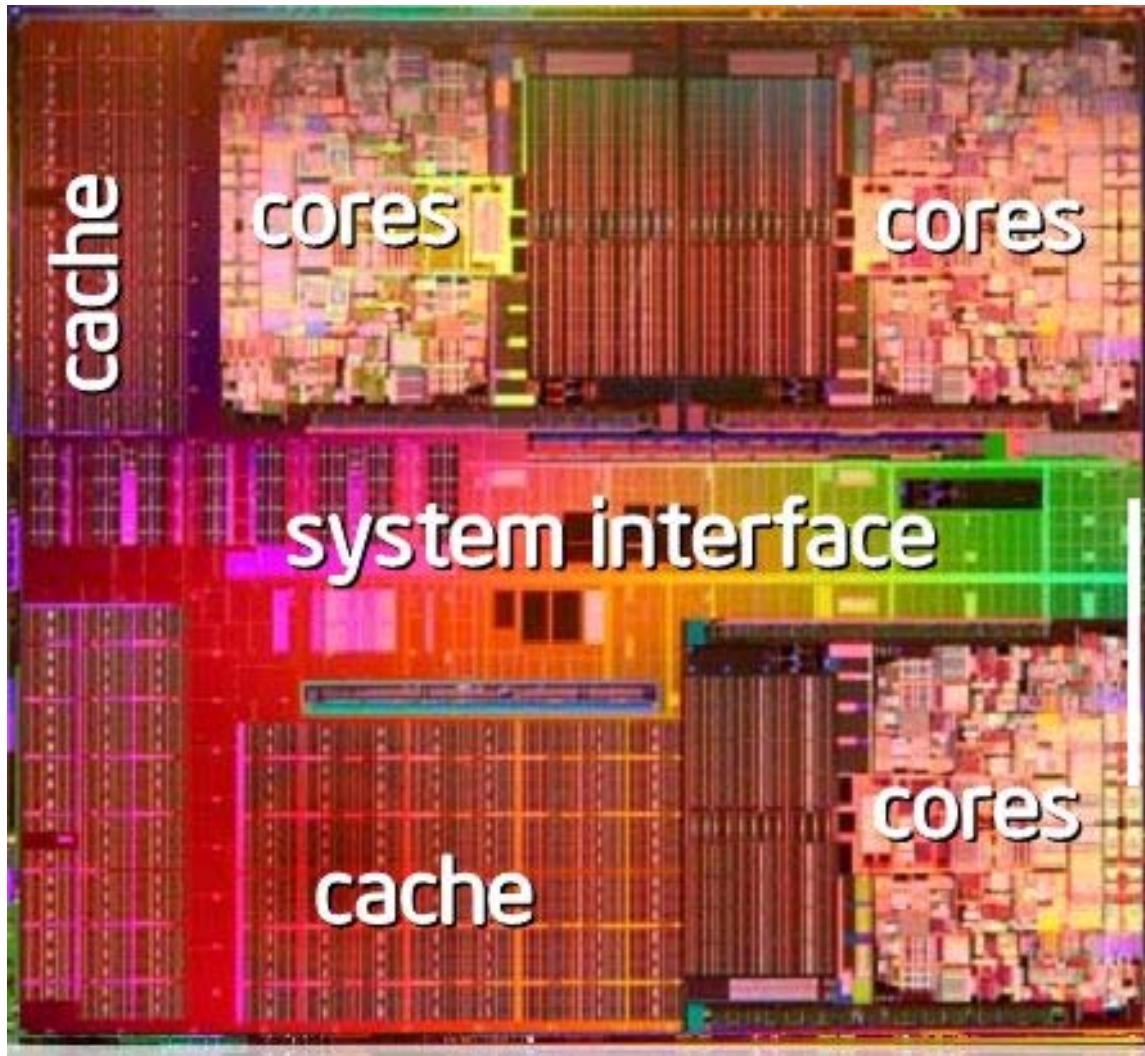
AMD Quad Core, K10 Barcelona (2008)



	i	d
L1	4x64 KB	4x64 KB
L2	4 x 512 KB	
L3		2 MB
Total		4.5 MB
% transist.		~50%
% area		nd

463 Mtransistores
283 mm² @ 65 nm

Intel Dunnington: 6-core Xeon X7460 (3Q 2008)

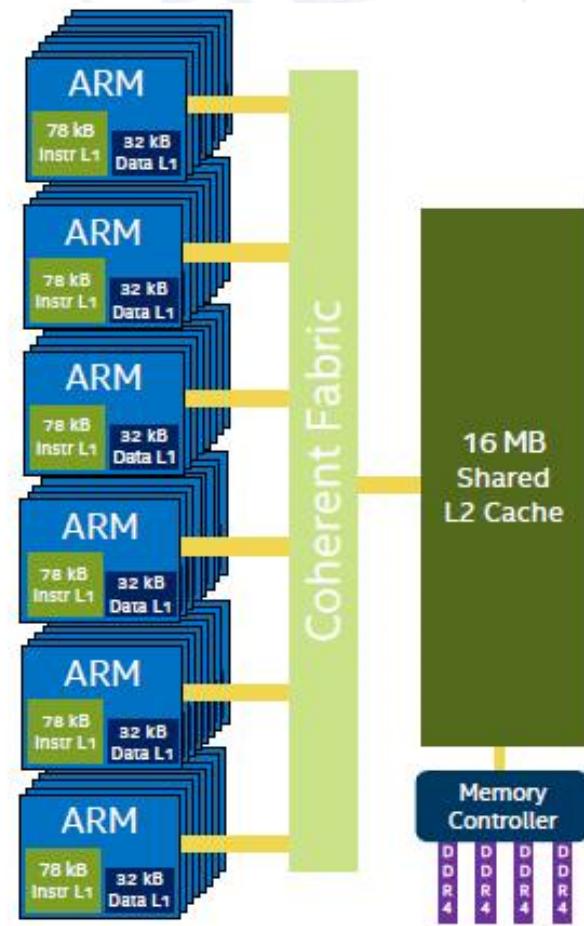
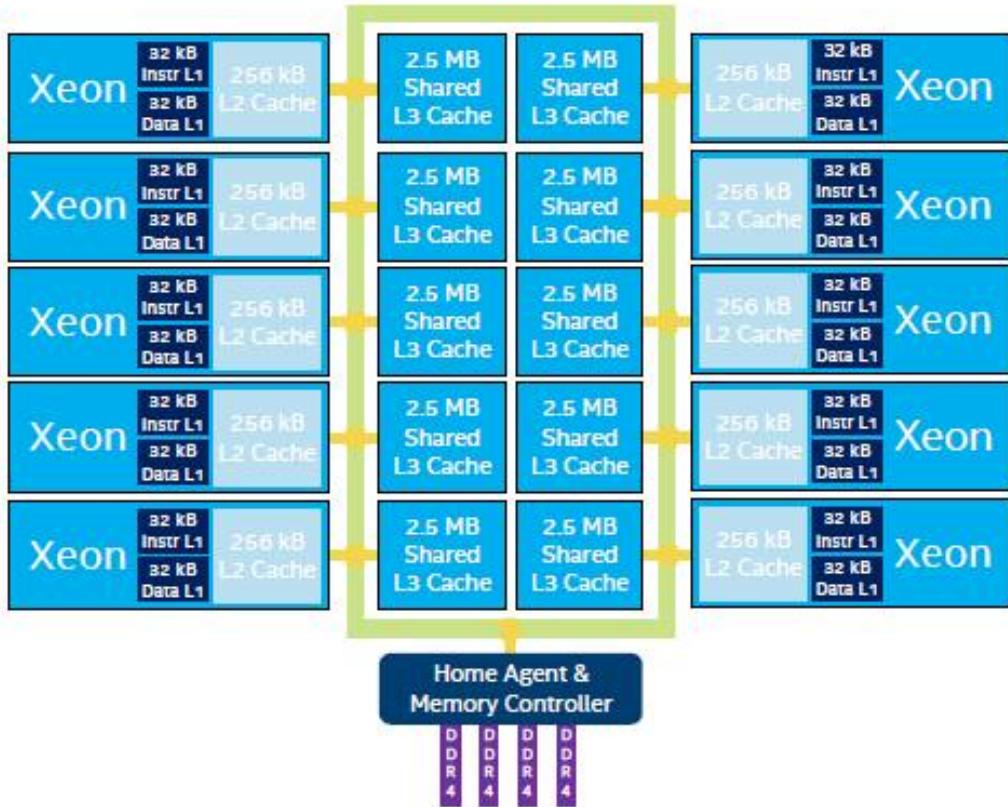


	i	d
L1	6x32 KB	6x32 KB
L2		3 x 3 MB
L3		16 MB (≈100 ciclos)
Total		25.3 MB
% transist.		nd
% area		nd

1.9 Gtransistores
503 mm² @ 45 nm high-K
2.66 GHz, 130W
\$2729 1K unidades Feb 2010
2 threads/core

SoCs para servidores: ARM vs Intel 2016

Cache and Memory Hierarchy



48 cores ARMv8 a 2 GHz

SoCs para servidores: ARM vs Intel 2016

Cache and Memory Latency Comparisons

Specifications		Measurements (nsec)		
		Cavium ThunderX_CP (48C, 2GHz)	Intel® Xeon® D-1581 (16C, 1.8GHz)	Intel® Xeon® E5-2640 v4 (10C, 2.6GHz)
	Cavium ThunderX <ul style="list-style-type: none">Dedicated L1 cache (78kB instr/32kB data)Shared L2 cache (16MB behind crossbar)No L3 cacheDDR4: 4 channels/CPU, up to 2400	Latency Measurement	1.51	1.67 Up to 10% slower
	Intel® Xeon® D-1581 <ul style="list-style-type: none">Dedicated L1 cache (32kB instr/32kB data)Dedicated L2 cache (256kB/core)Shared L3 cache (2.5MB per core)DDR4: 2 channels/CPU, up to 2133¹	L1 Cache read	20.4	5.01 Up to 75% faster
	Intel® Xeon® E5-2640 v4 <ul style="list-style-type: none">Dedicated L1 cache (32kB instr/32kB data)Dedicated L2 cache (256kB/core)Shared L3 cache (2.5MB per core)DDR4: 4 channels/CPU, up to 2133¹	L3 Cache read	No L3 cache	19.3
		DDR4 read	142-218**	63.7 Up to 55% - 70% faster
				61.1 Up to 56% - 71% faster

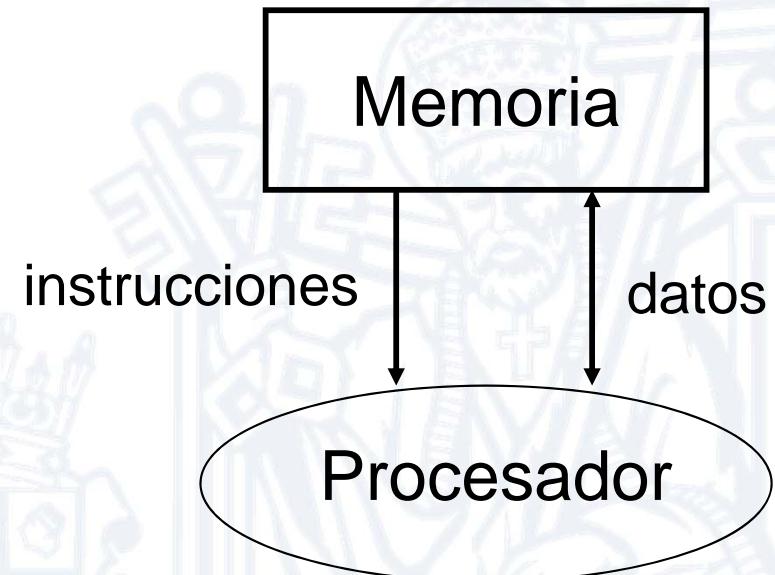
**Memory latencies vary from 142 to 218 nsec for unknown reasons

"Workloads that contain large sequential portions or are highly sensitive to latency are a poor match [for ThunderX]." David Kanter, Microprocessor Report, 2/1/2016

Intel va a añadir pronto una L4 en los Skylake Xeons!

Guión del tema

- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- Jerarquía de memoria
 - Memoria cache
- Ejemplos de memoria cache
- **Descripción de una cache**

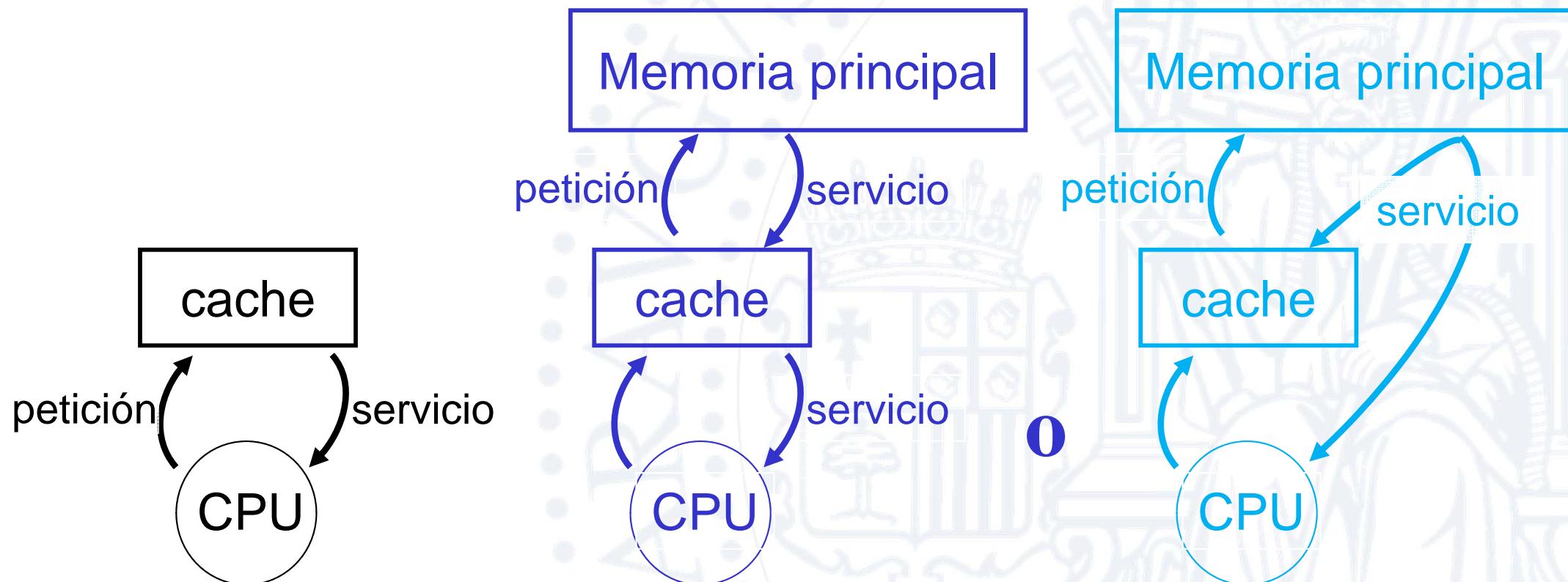


Descripción funcional

■ Acceso del procesador

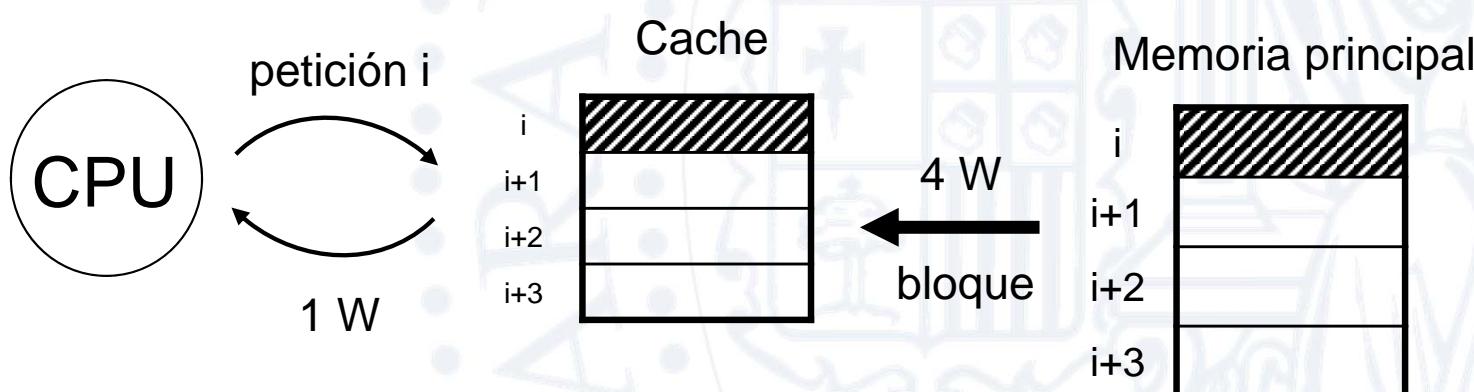
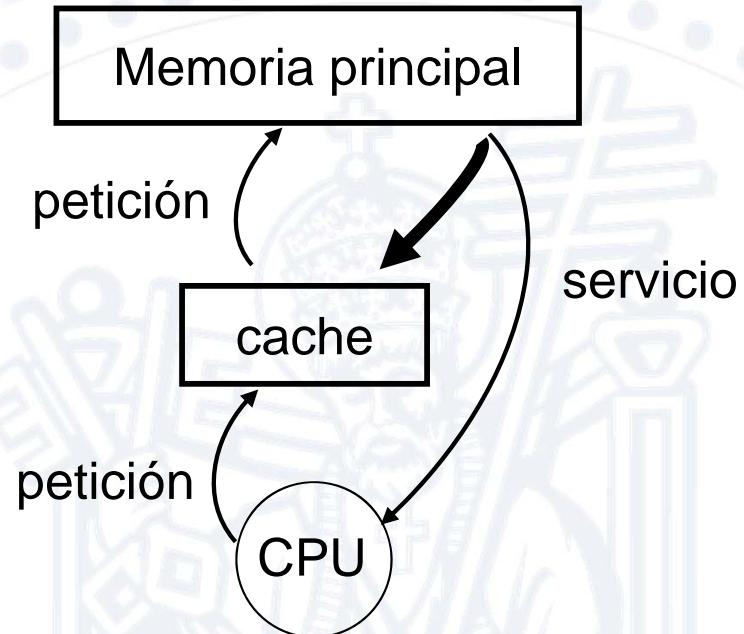
- Primero se mira en cache
 - ◆ Acierto (*hit*) : se encuentra la palabra buscada
 - ◆ Fallo (*miss*): no se encuentra la palabra buscada

- Sólo en caso de fallo se accede a memoria principal



Descripción funcional: gestión de contenidos

- Capacidad de Cache << capacidad de Memoria
 - ¿qué datos se guardan en cache?
- Aprovechar localidad temporal
 - Si CPU pide una palabra, probablemente la volverá a pedir
 - ◆ Guardar palabra en cache
- Aprovechar localidad espacial
 - Si CPU pide una palabra, probablemente pedirá las cercanas
 - ◆ Guardar varias palabras contiguas: bloque



Prestaciones

■ Tasas

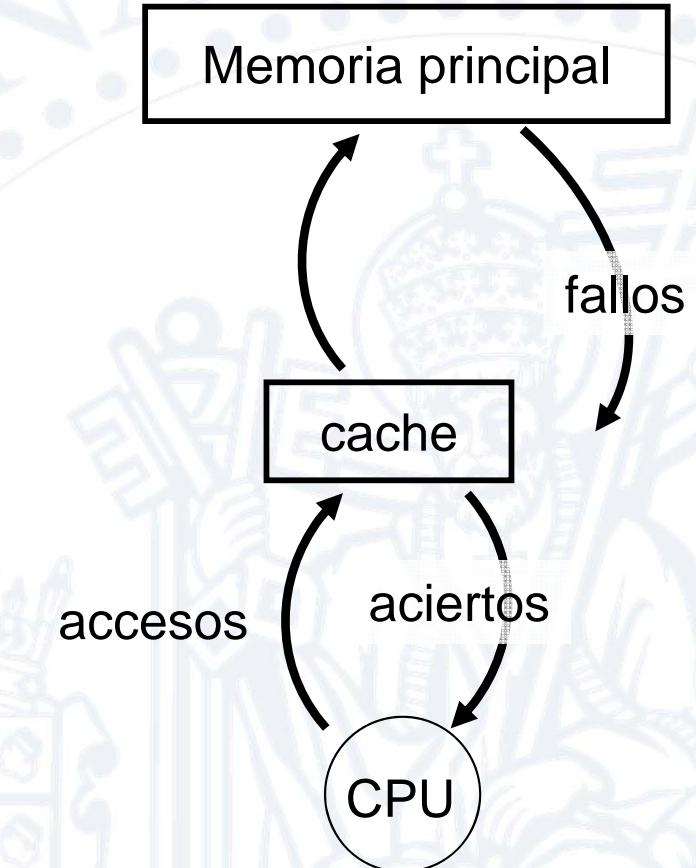
- Tasa de fallos (m)

$$m = \text{fallos} / \text{accesos}$$

- Tasa de aciertos (h)

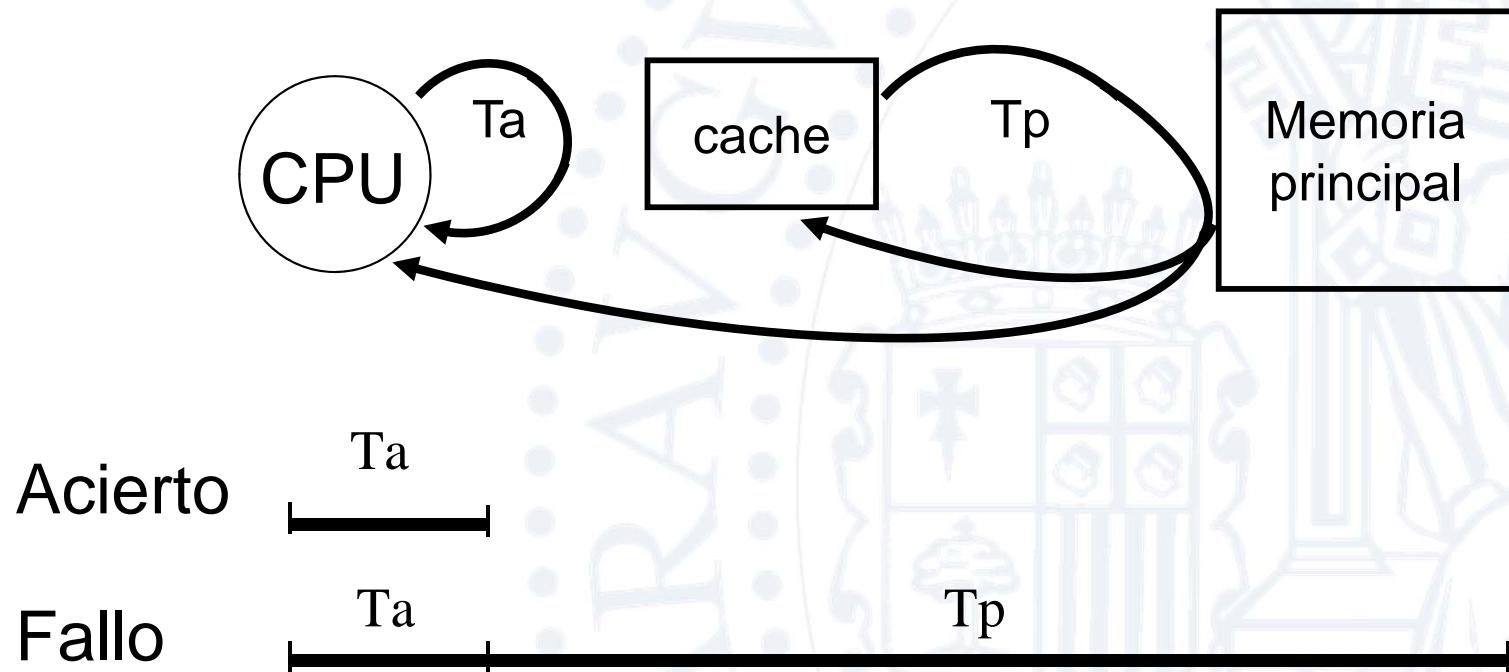
$$h = \text{aciertos} / \text{accesos}$$

$$h + m = 1$$



Costes (en tiempo o en nº de ciclos)

- Coste de acierto: T_a (ns) o C_a (ciclos)
 - Mirar en cache y entregar palabra al procesador
- Penalización de fallo: T_p (ns) o C_p (ciclos)
 - Desde que se detecta un fallo hasta que se sirve la palabra al procesador (leer bloque de MP, escribir bloque en cache ...)



Coste efectivo de acceso a memoria (Tef, Cef)

■ Tiempo medio de acceso a memoria



$$Tef = h * Ta + m(Ta + Tp)$$

$$Tef = Ta + \frac{m * Tp}{\text{penalización media}}$$

■ Ejemplo

$h = 0.9$ $m = 0.1$ $Ca = 1 \text{ ciclo}$ $Cp = 50 \text{ ciclos}$	Sin cache	$\text{Cef} = 50 \text{ ciclos}$
	Con cache	$\text{Cef} = 1 + 0.1 * 50 = 6 \text{ ciclos}$

Ejercicio

- Hay que seleccionar entre dos alternativas de cache:
 - a) 4 KB
 - b) 32 KB
- Tasas de fallos
 - a) 10%
 - b) 7.5%
- Calcula el CPI en cada caso sabiendo que:
 - El número medio de accesos a memoria por instrucción es 1,3
 - El CPI *ideal* es 3 (*ideal* = todos los accesos a memoria tardan un ciclo)
 - El coste de acierto es un ciclo, y la penalización de fallo son 10 ciclos
- La opción a) permite una frecuencia (procesador y cache) de 150MHz, mientras que la opción b) permite sólo 100MHz.
¿Qué opción da mejor rendimiento?

Parámetros de diseño

- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe

cache
8 KBytes

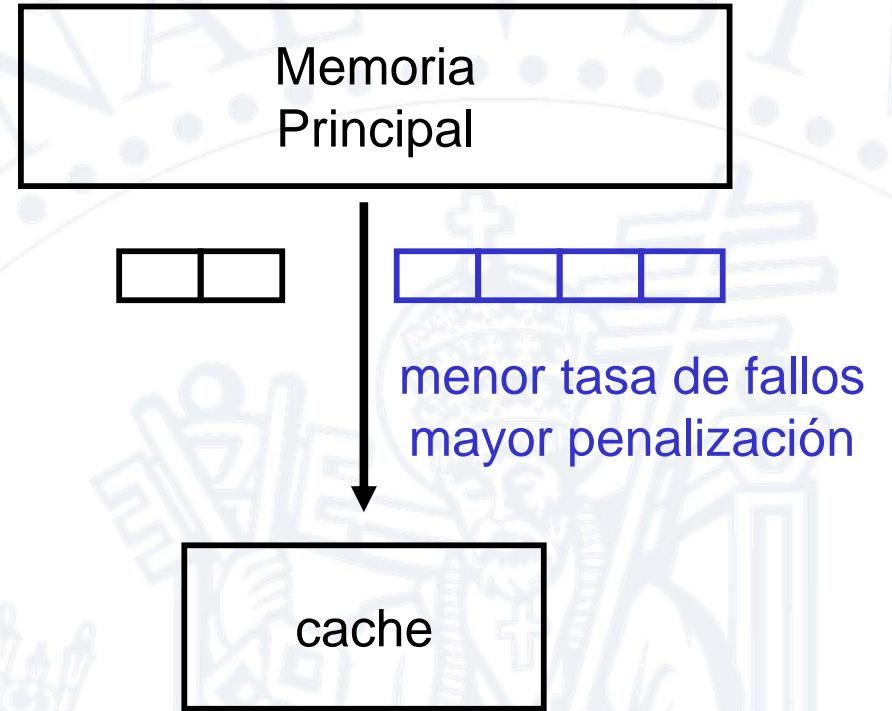
ó

cache
64 KBytes

menor tasa de fallos
mayor tiempo de acierto

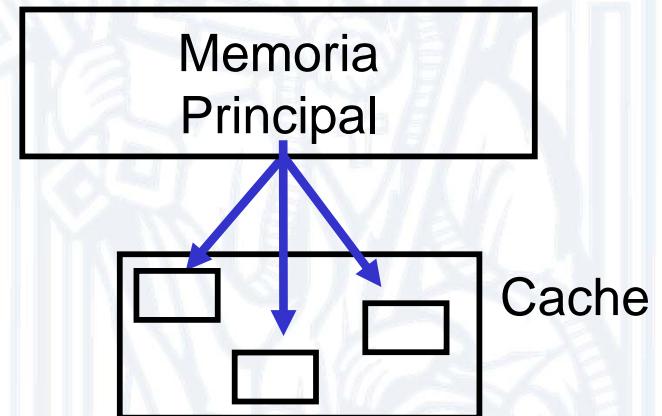
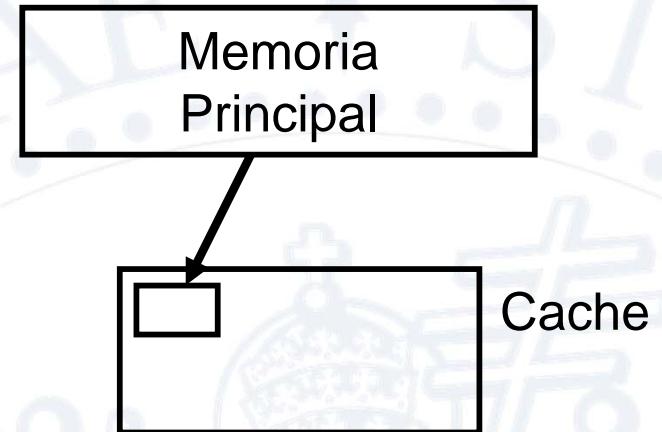
Parámetros de diseño

- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



Parámetros de diseño

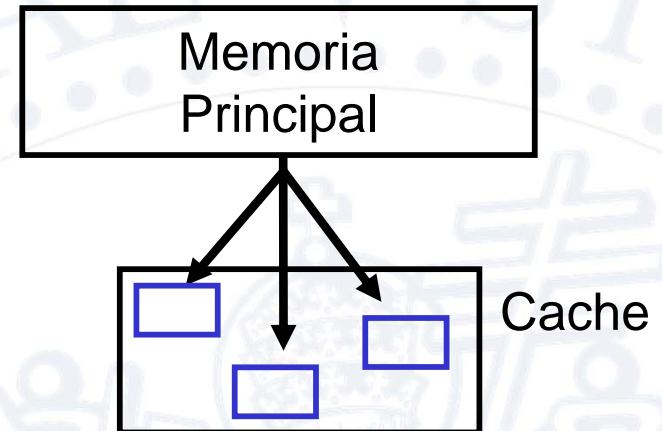
- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



menor tasa de fallos
mayor tiempo de acierto

Parámetros de diseño

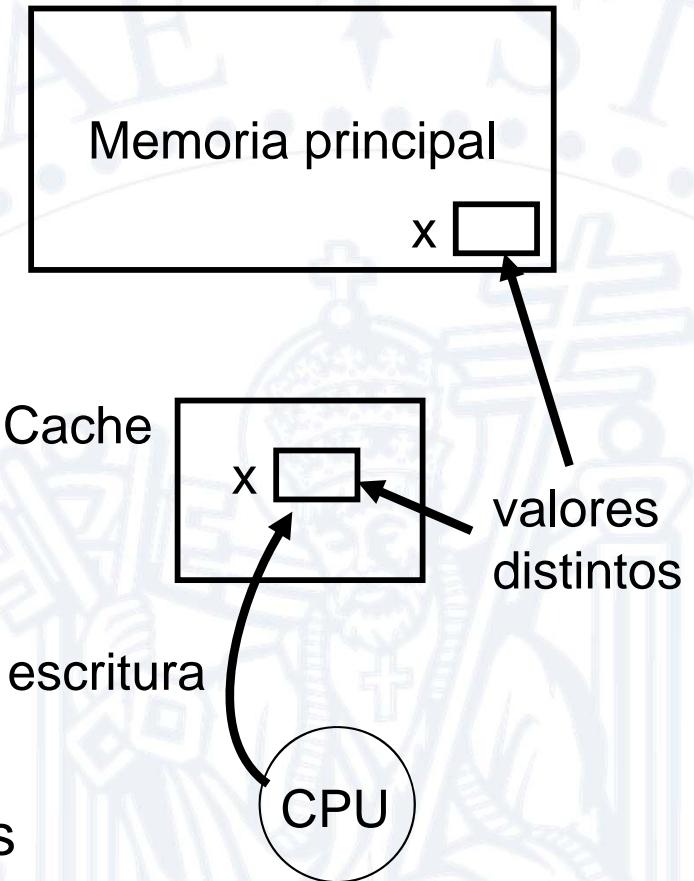
- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



¿qué bloque se expulsa?

Parámetros de diseño

- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



Resumen

- Memorias: las DRAM son muy lentas, las SRAM muy caras
- Localidad: los programas acceden a memoria con alta localidad espacial y temporal
- Jerarquía de memoria: almacenes cercanos al procesador rápidos y pequeños, almacenes lejanos lento y grandes
- Memoria cache: memoria pequeña y rápida que contiene el subconjunto más usado de memoria principal
- Modelo de prestaciones

$$T_{ef} = T_a + m^* T_p$$

Ejercicio de clase

- Calcular la tasa de fallos de una cache de datos.

Suponer que:

- El tamaño de bloque es de K elementos
- El número de bloques de la cache de datos (capacidad) es menor que el número de filas de la matriz A

Alg. 1

```
for (i = 0; i < max; i++)
    for (j = 0; j < max; j++)
        A[i][j] = 0;
```

Alg. 2

```
for (i = 0; i < max; i++)
    for (j = 0; j < max; j++)
        A[j][i] = 0;
```



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza



Tema 6 – Organización de una cache: Correspondencia

P. Ibáñez, J.L. Briz, V. Viñals, J. Alastruey, J. Resano
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

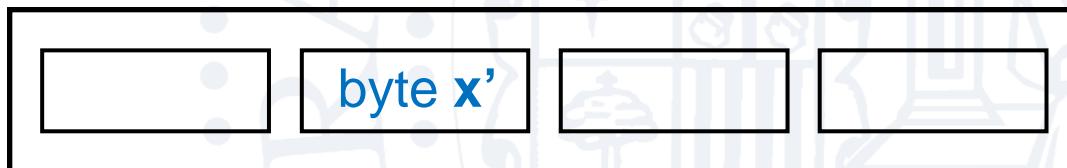
Guión del tema

- Organización de la memoria en bloques
- Descripción de alternativas de correspondencia
- Implementación
- Modelo de las 3 Cs

Organización de la memoria en bloques

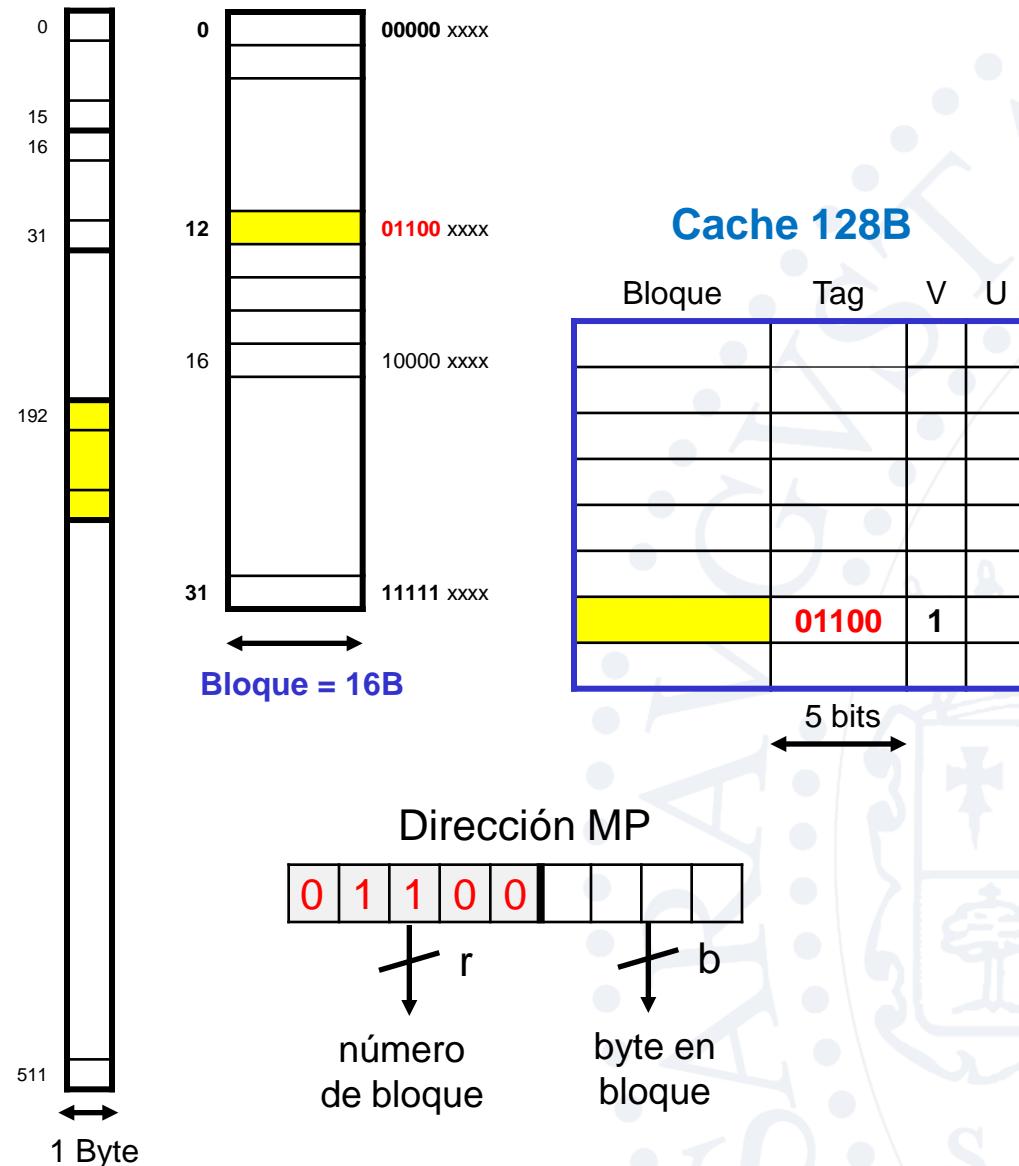
- Bloques alineados
- Unidad de correspondencia entre Mp y Mc
 - nº de bloque: clave para buscar un bloque en Mc
- Unidad de transferencia entre Mp y Mc
 - Bloques enteros fluyen desde Mp hacia Mc en caso de fallo de cache

Bloque x



Correspondencia totalmente asociativa

Memoria Principal: 512 Bytes



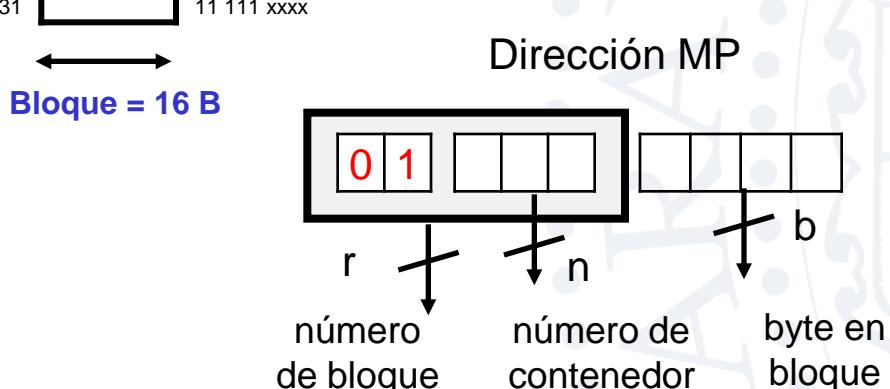
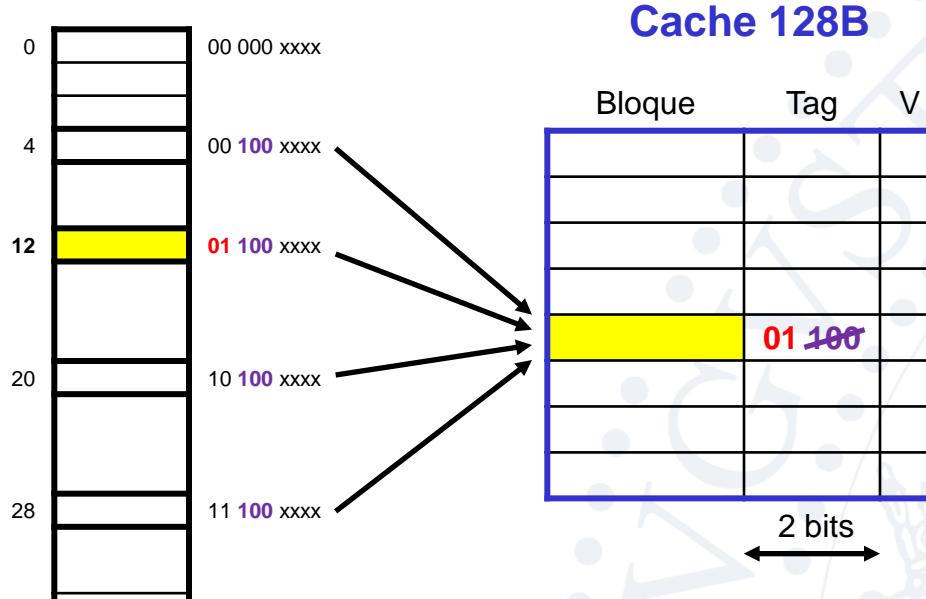
- Cualquier bloque puede estar en cualquier contenedor
- La búsqueda se realiza en paralelo
 - un comparador por contenedor
- Necesario un algoritmo de reemplazo

- V: bit de validez
- U: bits de uso.
→ bloque víctima (a reemplazar)

- ¿Implementación?
 - Transf. con el procesador: 4B
 - Transferencia con memoria: 4-16B
 - SRAM contenidos:
 - $8 \times 16\text{ B}$ o $32 \times 4\text{ B}$
 - ¿Reemplazo?

Correspondencia directa

Memoria Principal: 512 Bytes

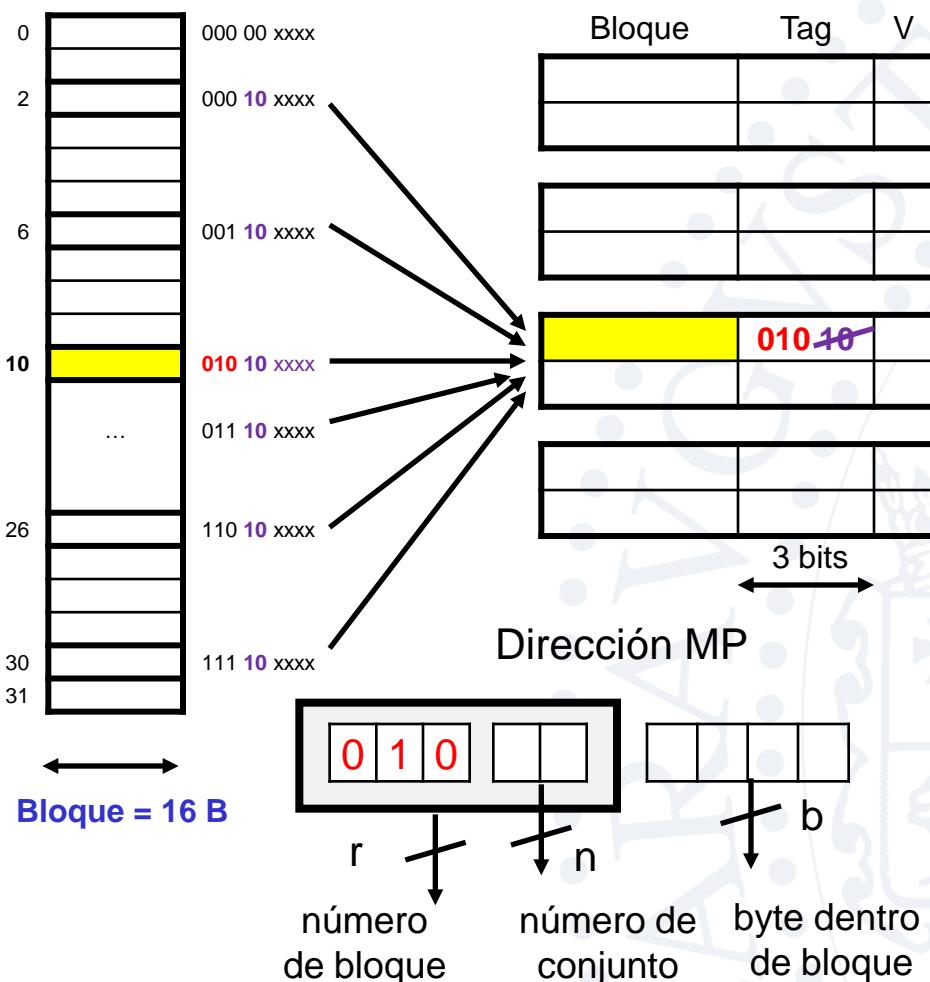


- Cada bloque de Mp sólo puede estar en **un contenedor o conjunto** de Mc
- Conflictos: aumentan la tasa de fallos
- Sin algoritmo de reemplazo
- Implementación
 - Se leen en paralelo tag y bloque
 - Un sólo comparador para toda la cache
 - A igualdad de tamaño, correspondencia con Tag menores

¿Implementación?

Correspondencia asociativa por conjuntos

Memoria Principal: 512 Bytes



- N conjuntos de tamaño As contenedores (o vías)
- Cada bloque de Mp sólo puede estar en un conjunto de Mc
- En un conjunto: colocación libre
- Solución de compromiso

$$128B = N \times As \times L$$

4 conjuntos
x 2 bloques/conjunto
x 16 Bytes/bloque

¿Implementación?

Correspondencias

- Ejercicio: calcular la capacidad del tercer nivel de cache del Intel 6-core Xeon 7400 (Dunnington) sabiendo que se compone de 4 bancos, y que cada uno tiene las siguientes características:
 - 4096 conjuntos
 - Asociatividad 16
 - Tamaño de bloque: 64 bytes
- Repite el cálculo para asociatividad 12 y 8 (correspondiente a procesadores de menor coste)

Correspondencias

- Ejercicio: suponiendo para el direccionamiento de bytes en memoria se utilizan 32 bits (4 GBytes direccionables), realizar la descomposición de una dirección en los campos necesarios para direccionar estas caches (r, n , b):
 - C = 64 KB, asoc. 8, bloque 32 bytes
 - C = 64 KB, asoc. 1, bloque 32 bytes

Correspondencias

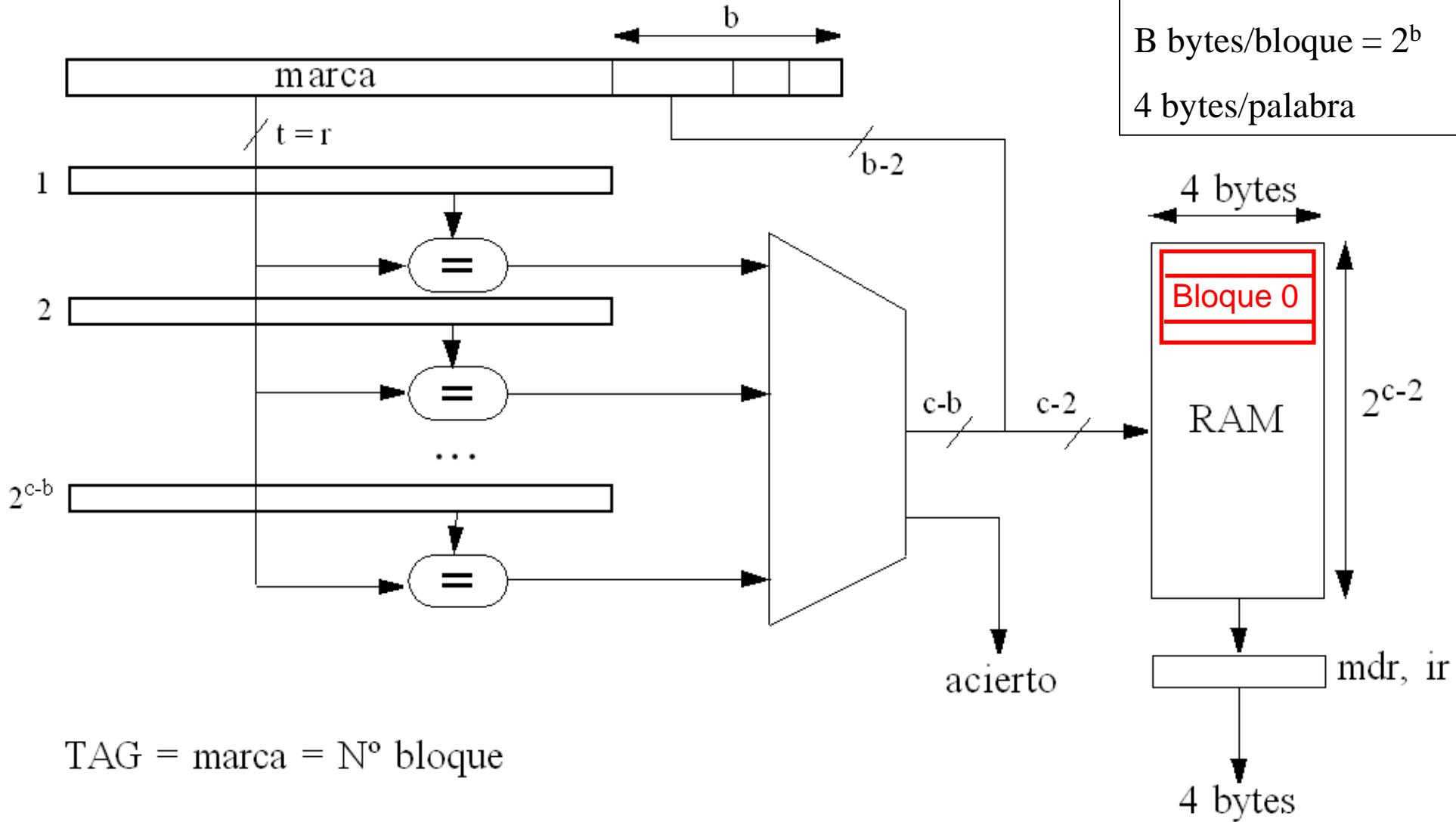
- Ejercicio: un procesador ejecuta un bucle que repite la secuencia de direcciones siguiente dos veces:
 - 0xA0, 0x34, 0x74, 0xA8, 0x30

Suponiendo que la cache está inicialmente vacía (todos los bloques inválidos), obtener la tasa de fallos para:

- $M_p = 256$ bytes
- Cache = 64 bytes
- Bloques de 16 bytes
- Asociatividad = {1, 2, 4}

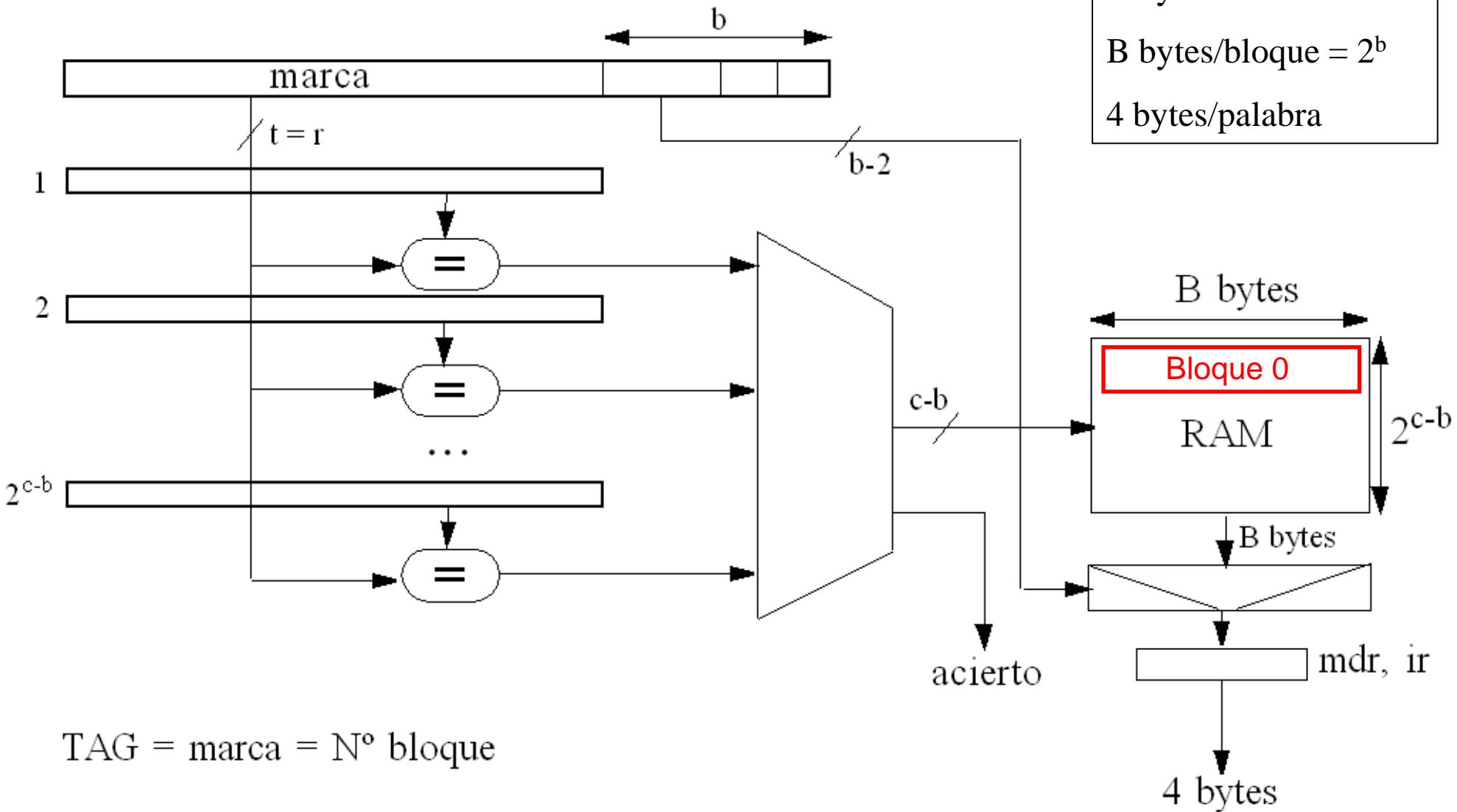
Implementación

■ Completamente asociativa: TTL (1)



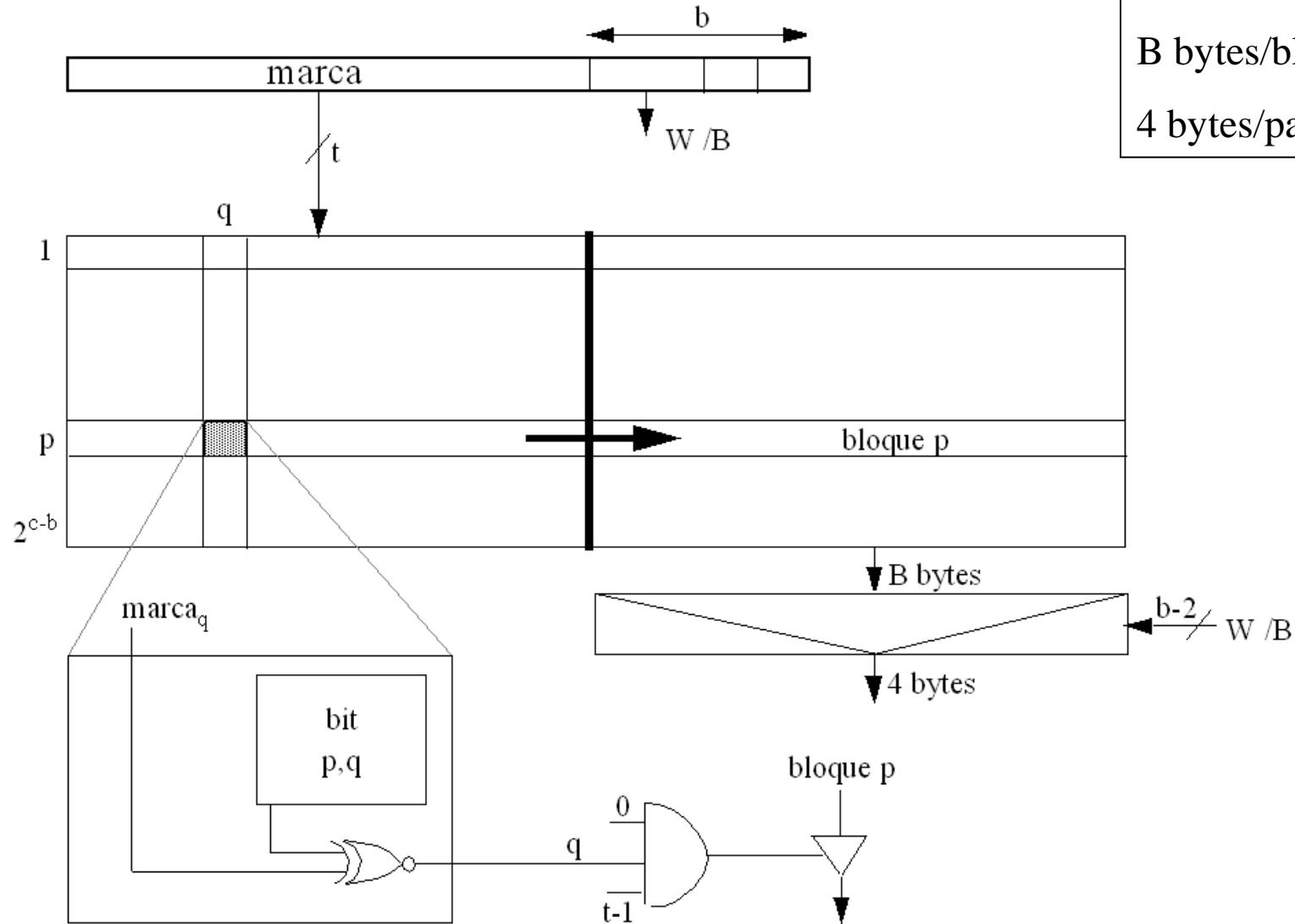
Implementación

■ Completamente asociativa: TTL (2)



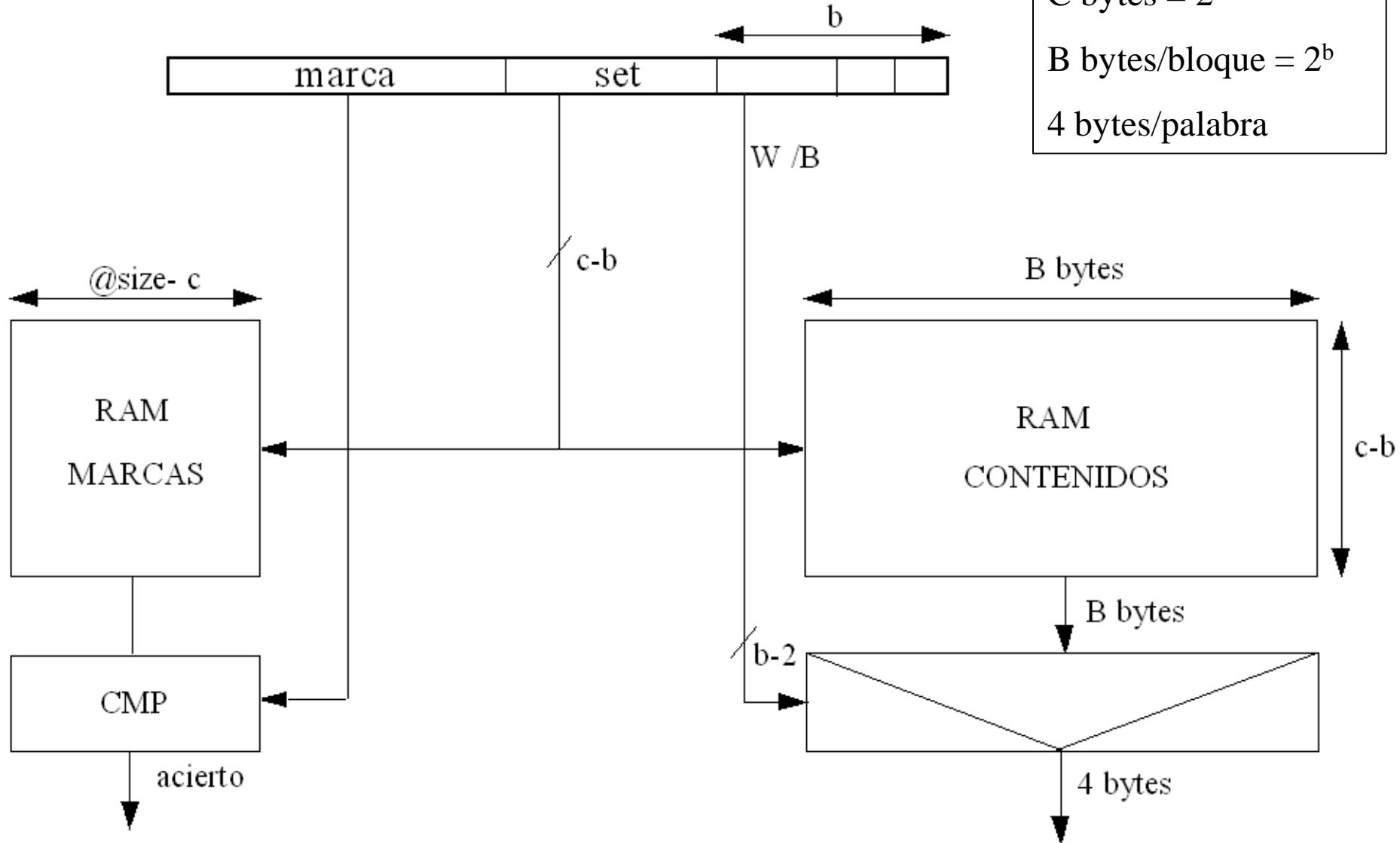
Implementación

■ Completamente asociativa: CAM (VLSI)



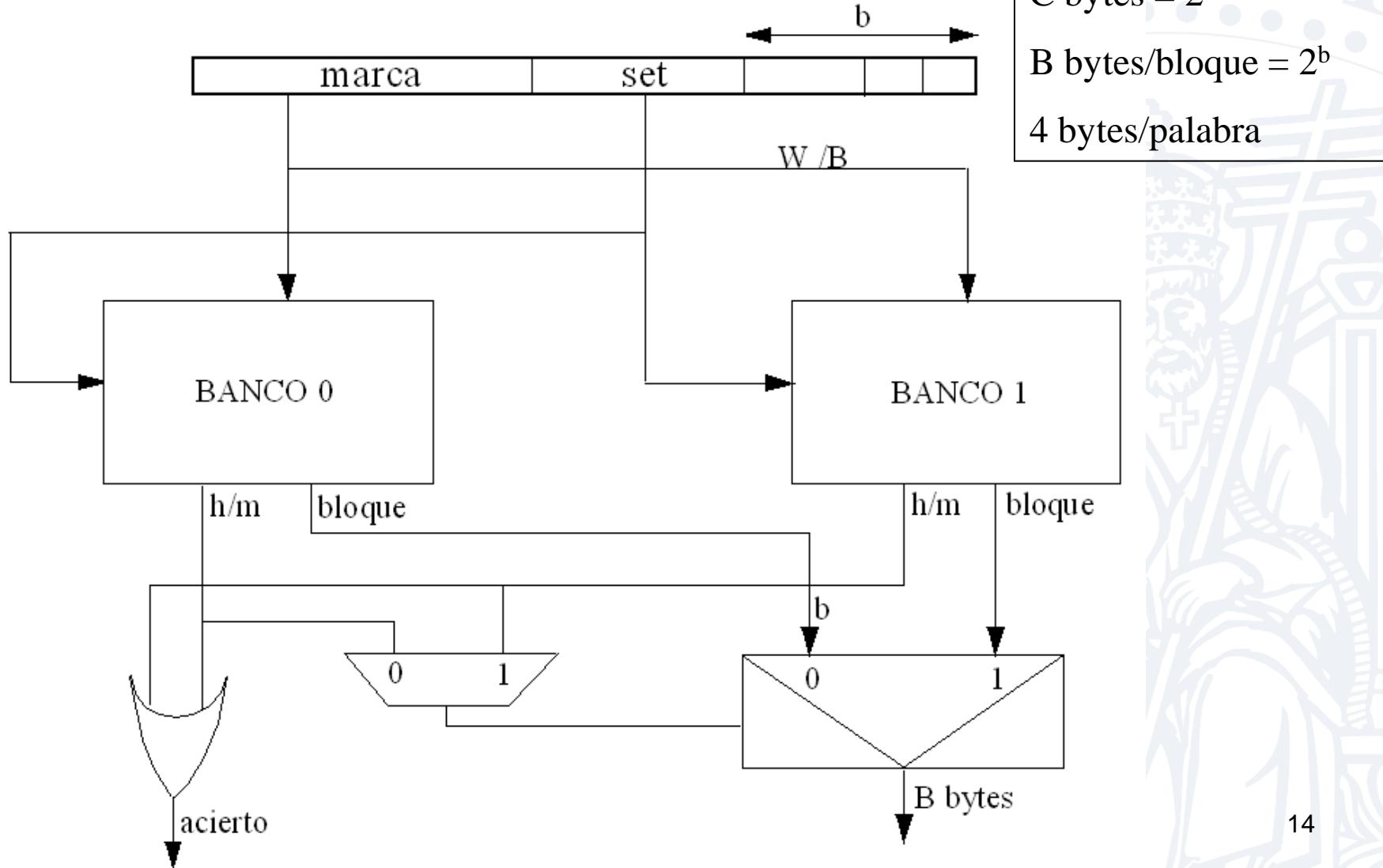
Implementación

■ Correspondencia directa (acceso paralelo)



Implementación

■ Asociativa por conjuntos (acceso paralelo)



Implementación: problema

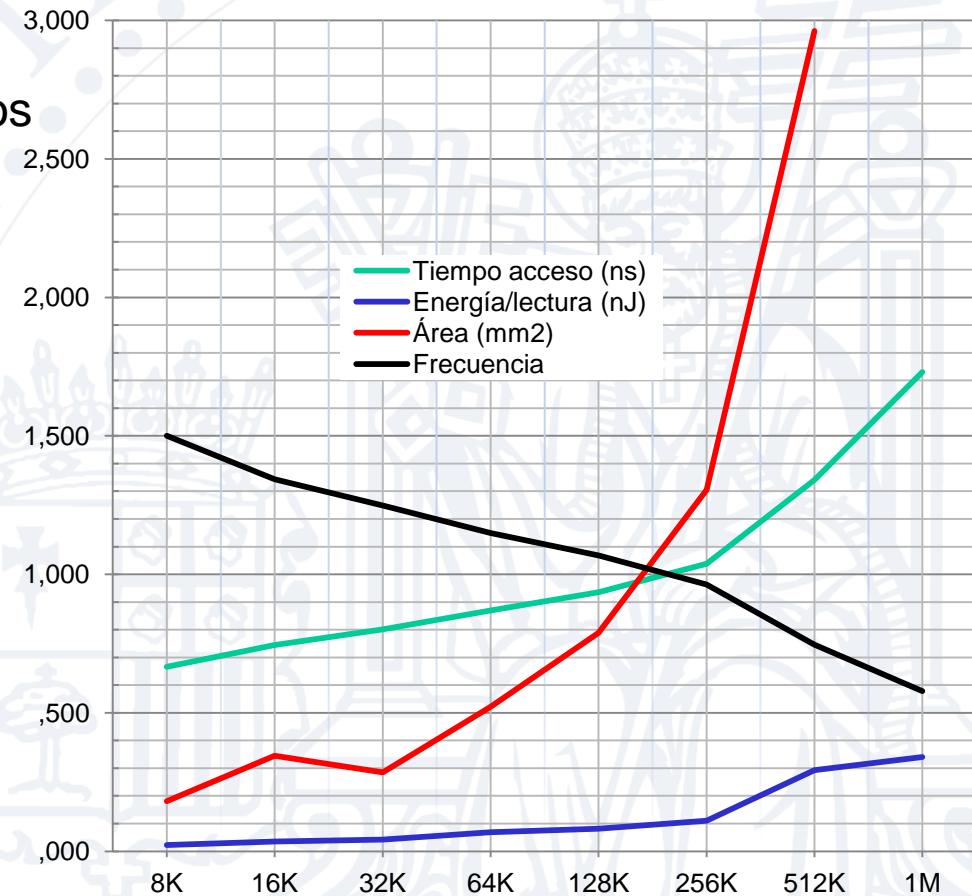
- En la cache asociativa por conjuntos de **acceso paralelo** se accede en paralelo a Marcas y Contenidos en todas las vías a la vez, lo cual consume mucha energía, pero a cambio es la opción más rápida
- Para asociatividades elevadas y caches de nivel 2 ó 3, la latencia es menos crítica, pero el consumo es más
- Proponed una implementación alternativa, llamada de **acceso secuencial**, que sea más lenta, pero que consuma menos

Implementación

■ Análisis cache: retardo, área y consumo

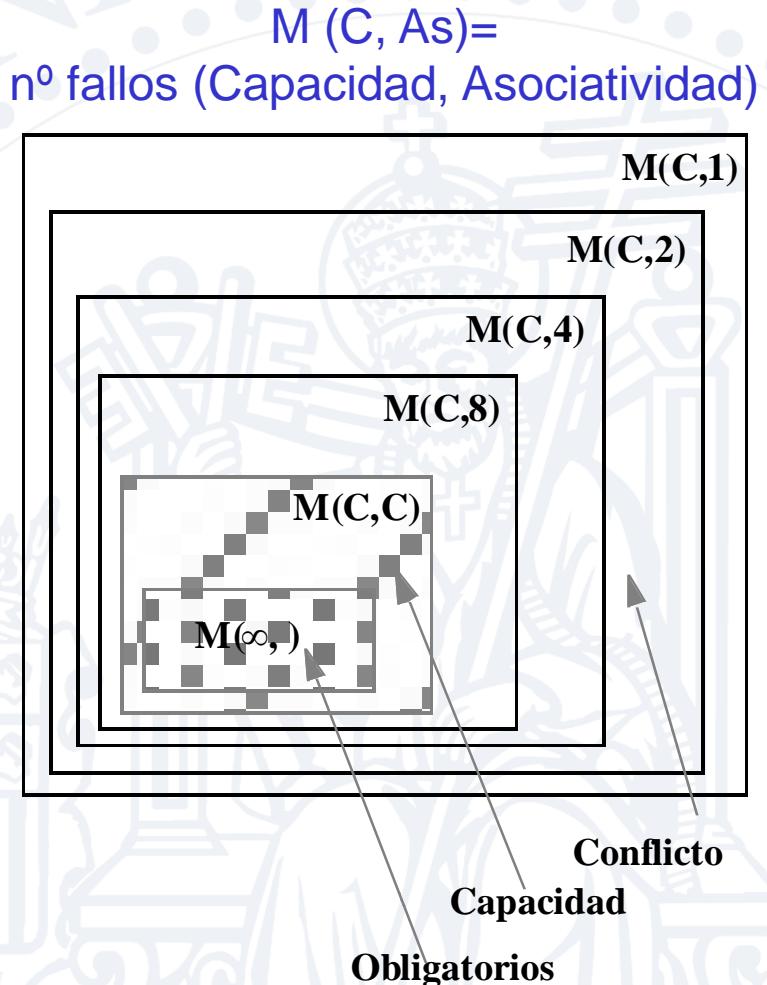
- V1.0 http://www.ece.ubc.ca/~stevew/cacti/run_frame.html
- V5.2 <http://quid.hpl.hp.com:9082/cacti/index.y>

escala bien transistores y cables
redes en H para interconectar bancos
soporta varios escenarios:
- low power, high performance, ...
modela DRAMs y SRAMs



Modelo de las 3 Cs: tipos de fallos

- Fallos **obligatorios** (*Compulsory*): $M(\infty,)$
 - Primer acceso a un bloque
 - Fallos de una cache infinita
- Fallos de **capacidad**
 - Tamaño programa > cache
 - ◆ código y/o datos
 - Aparecen al limitar el tamaño, pero en una cache completamente asociativa
- Fallos de **conflicto**
 - Los bloques compiten por conjuntos de asociatividad limitada

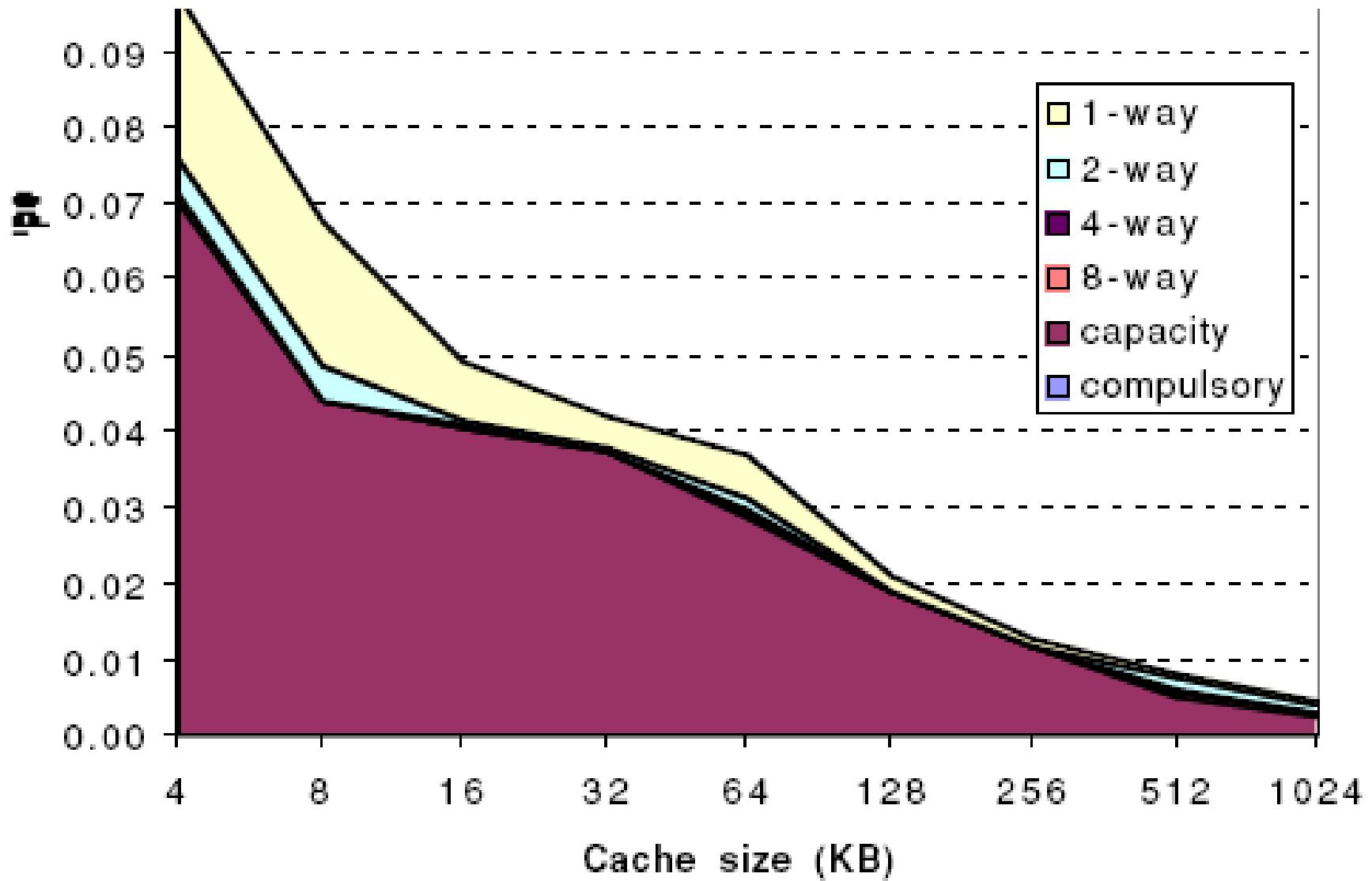


Cache size (KB)	Degree associative	Total miss rate	Miss rate components (relative percent) (sum = 100% of total miss rate)				
			Compulsory	Capacity	Conflict		
4	1-way	0.098	0.0001	0.1%	0.070	72%	0.027 28%
4	2-way	0.076	0.0001	0.1%	0.070	93%	0.005 7%
4	4-way	0.071	0.0001	0.1%	0.070	99%	0.001 1%
4	8-way	0.071	0.0001	0.1%	0.070	100%	0.000 0%
8	1-way	0.068	0.0001	0.1%	0.044	65%	0.024 35%
8	2-way	0.049	0.0001	0.1%	0.044	90%	0.005 10%
8	4-way	0.044	0.0001	0.1%	0.044	99%	0.000 1%
8	8-way	0.044	0.0001	0.1%	0.044	100%	0.000 0%
16	1-way	0.049	0.0001	0.1%	0.040	82%	0.009 17%
16	2-way	0.041	0.0001	0.2%	0.040	98%	0.001 2%
16	4-way	0.041	0.0001	0.2%	0.040	99%	0.000 0%
16	8-way	0.041	0.0001	0.2%	0.040	100%	0.000 0%
32	1-way	0.042	0.0001	0.2%	0.037	89%	0.005 11%
32	2-way	0.038	0.0001	0.2%	0.037	99%	0.000 0%
32	4-way	0.037	0.0001	0.2%	0.037	100%	0.000 0%
32	8-way	0.037	0.0001	0.2%	0.037	100%	0.000 0%
64	1-way	0.037	0.0001	0.2%	0.028	77%	0.008 23%
64	2-way	0.031	0.0001	0.2%	0.028	91%	0.003 9%
64	4-way	0.030	0.0001	0.2%	0.028	95%	0.001 4%
64	8-way	0.029	0.0001	0.2%	0.028	97%	0.001 2%
128	1-way	0.021	0.0001	0.3%	0.019	91%	0.002 8%
128	2-way	0.019	0.0001	0.3%	0.019	100%	0.000 0%
128	4-way	0.019	0.0001	0.3%	0.019	100%	0.000 1%
128	8-way	0.019	0.0001	0.3%	0.019	100%	0.000 0%
256	1-way	0.013	0.0001	0.5%	0.012	94%	0.001 6%
256	2-way	0.012	0.0001	0.5%	0.012	99%	0.000 0%
256	4-way	0.012	0.0001	0.5%	0.012	99%	0.000 0%
256	8-way	0.012	0.0001	0.5%	0.012	99%	0.000 0%
512	1-way	0.008	0.0001	0.8%	0.005	66%	0.003 33%
512	2-way	0.007	0.0001	0.9%	0.005	71%	0.002 28%
512	4-way	0.006	0.0001	1.1%	0.005	91%	0.000 8%
512	8-way	0.006	0.0001	1.1%	0.005	95%	0.000 4%

HePaz2003 pág.424.- Tasa total de fallos para cada tamaño de Mc y porcentaje de cada uno según el modelo de las tres C's. Los obligatorios son independientes del tamaño de Mc. Los de capacidad disminuyen cuando el tamaño de Mc aumenta. Los de conflicto bajan al aumentar la asociatividad. Observar que se cumple la regla del 2:1 hasta tamaños de 128 KB; una Mc de mapeo directo de tamaño N tiene aprox. la misma tasa de fallos que una 2-way de tamaño N/2. Datos de SPECint y SPECfp; reemplazo LRU.

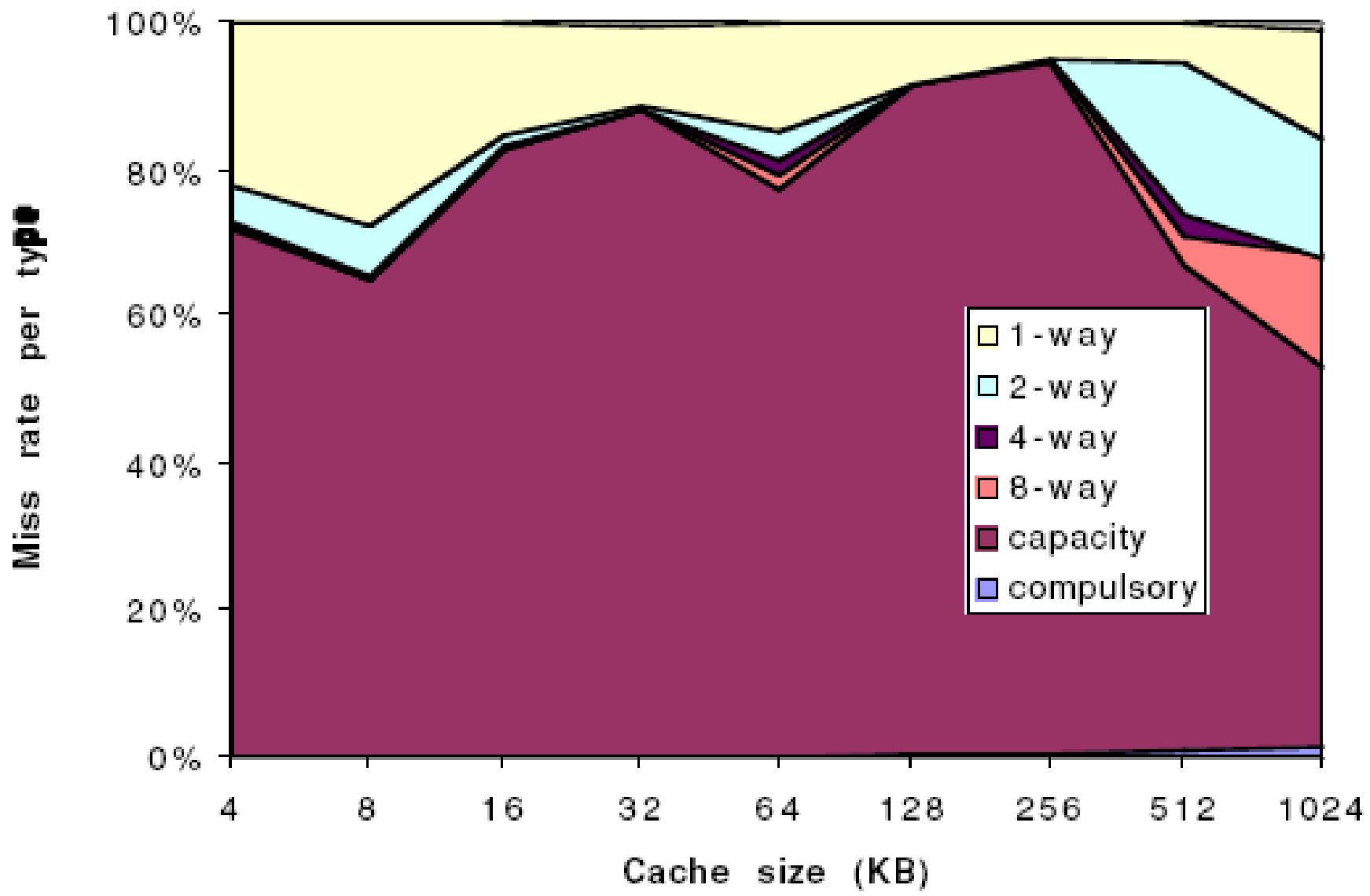
Modelo de las 3 Cs

■ Un ejemplo experimental



Modelo de las 3 Cs

■ Obtención





Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza



Tema 7 – Tamaño de bloque y Reemplazo

P. Ibáñez, J.L. Briz, V. Viñals, J. Alastruey, J. Resano
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

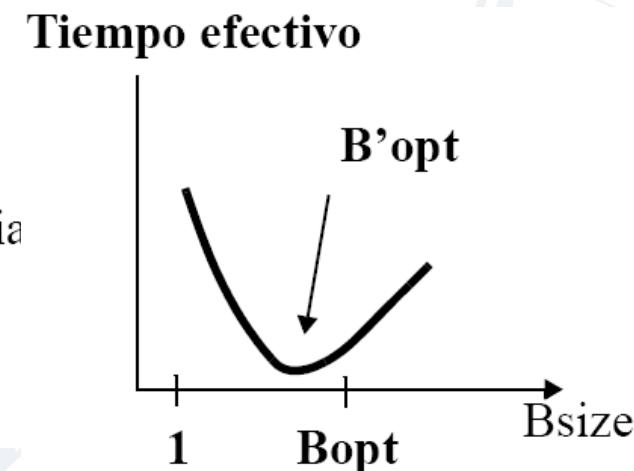
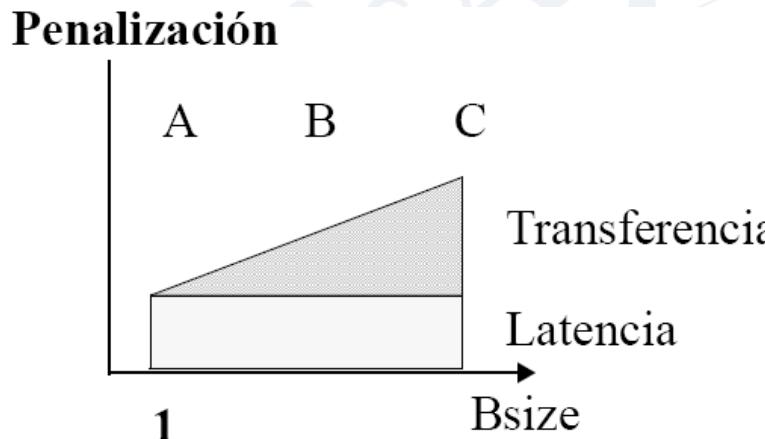
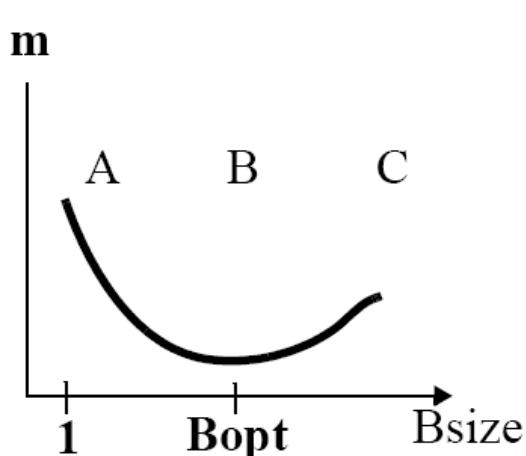
Guión del tema

- Influencia del tamaño de bloque
- Modelos de acceso a memoria principal
- Algoritmo de reemplazo

Influencia del tamaño de bloque

- Aumenta el tamaño de bloque
 - disminuye la tasa de fallos
 - aumenta la penalización de fallo

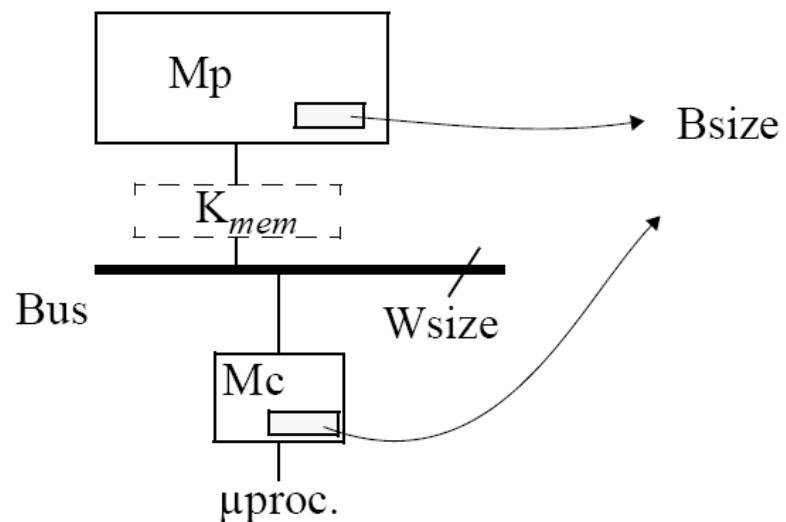
$$Tef = Ta + m * \text{penaliz}$$



- A: no explota localidad espacial
 - poca penalización, muchos fallos
- B: punto polución
 - Lo que se tira es más útil que lo que se trae
- C: explota bien la localidad espacial
 - pero no la temporal, muchos fallos

Aparece un nuevo óptimo (B_{opt}') **menor** que el anterior (B_{opt})

Acceso a memoria principal DRAM



$$CrB = L + R \times \left(\frac{Bsize}{Wsize} - 1 \right) = \\ = L + R \times (n - 1)$$

- **L:** latencia de la primera palabra
 - Arbitraje y concesión de bus + acceso al primer trozo
 - Depende del bus, del controlador (K_{mem}) y de la tecnología DRAM
 - Variable: el chip puede estar en fase de refresco, la “página” puede estar abierta o cerrada”, el chip concreto puede estar “cerca” o “lejos”, ...
- **R:** suministro del resto de palabras del bloque
 - Depende de la organización del chip DRAM
- Acceso aleatorio: $R \approx L$
- Modo secuencial: $R < L$

Reemplazo: selección del bloque víctima

- El bloque que queremos: **x**; el que expulsamos (víctima): **u**
- **u** en Correspondencia Directa está en el conjunto asignado a **x** ... no hay elección...
- **u** en el resto de correspondencias: bloque con menor utilidad en el futuro: LRU, pLRU, aleatorio, FIFO
- LRU (Least Recently Used)
 - Expulsa el bloque no referenciado desde hace más tiempo
 - En general, es un buen predictor del futuro
explota bien la localidad temporal
 - Desventajas
 - ◆ Elevado coste para asociatividades altas
 - $\approx As^2/2$ bits por conjunto en una buena implementación
 - ◆ Comportamiento pésimo en bucles (códigos vectoriales científicos)

Reemplazo: tasas de fallos

■ FIFO

- Sencillo.

Anomalía: puede ser que con mas capacidad mas fallos

■ Aleatorio

- Expulsa uno cualquiera
- Ventaja: muy barato, 1 contador de A_s bits que se incrementa en cada ciclo de CPU

Asociatividad

Tamaño	2-way			4-way			8-way		
	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

HePa2003: Figura 5-6.-Fallos por cada 1000 referencias a memoria. Simulación para un tamaño de bloque de 64 bytes, sobre una arquitectura Alpha, usando cinco aplicaciones de SPECint2000 (gap, gcc, gzip, mcf y perl) y cinco de SPECfp2000 (applu, art, equake, lucas, y swim)

Ejercicio

- Un procesador realiza la petición a memoria cache de las siguientes direcciones de memoria
 - 0xA0, 0xE4, 0x01, 0x21, 0xA2, 0x43, 0xA3, 0x01, 0x60, 0x02

Suponiendo que la cache está inicialmente vacía (todos los bloques son inválidos), obtener la tasa de fallos para:

- $M_p = 256$ bytes
- Cache = 64 bytes
- Bloques de 16 bytes
- Asociatividad = 4
- Reemplazo = {LRU, FIFO}

Ejercicio

- Un procesador ejecuta un bucle 2 veces. Dentro del bucle el patrón de accesos es:
 - 0x40, 0x60, 0x80, 0xA0, 0xB0

Suponiendo que la cache está inicialmente vacía (todos los bloques son inválidos), obtener la tasa de fallos para unos procesadores con las siguientes características

- $M_p = 256$ bytes
- Cache = 64 bytes
- Bloques de 16 bytes
- Asociatividad = 4
- Reemplazo = {LRU, Random}
- Imaginad que random reemplaza siempre un bloque inválido si lo hay y si no hay elige la siguiente secuencia de bloques: 2, 3, 0, 1



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza



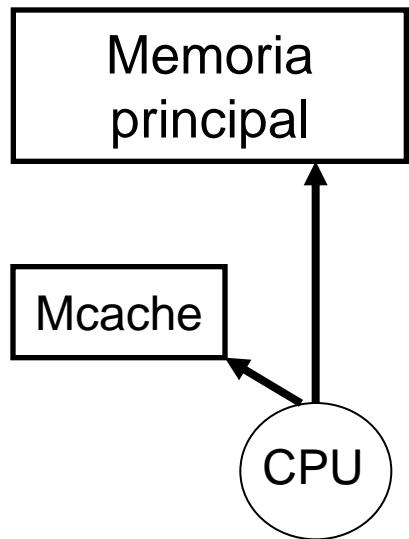
Tema 8 – Escrituras

P. Ibáñez, J.L. Briz, V. Viñals, J. Alastruey, J. Resano
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

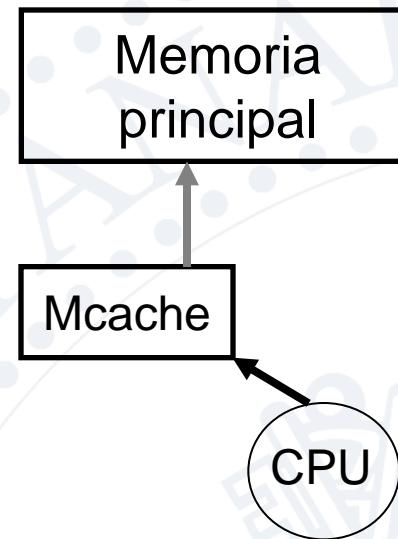
Guión del tema

- Escritura en acierto: inmediata y retardada
- Escritura en fallo:
reserva de contenedor y carga de bloque
- Ejemplo: inicialización de matriz

Políticas de Escritura en Acierto



Write-through
(escriutra inmediata)



Write-back
(escriutra retardada)

- Se escribe al mismo tiempo en Cache y Memoria Principal
- Frecuencia de escrituras en Mp muy alta = frecuencia de escrituras de CPU

¿tráfico con Mp?

- Sólo se escribe en Cache
 - Incoherencia con Mp
 - Actualizar Mp al expulsar bloque sucio
 - 1 bit por bloque: D = sucio/limpio
- Frecuencia de escrituras en Mp baja = frecuencia de reemplazo de bloques sucios

Tablas de tráfico CB vs. WT

■ CB

	CPU	Mc	Mp
rh			
rm			
wh			
wm		?	

■ WT

	CPU	Mc	Mp
rh			
rm			
wh			
wm		?	

- El tráfico está relacionado con el consumo de energía.
- Pero en cuanto a prestaciones, no existe una relación clarísima ...

Políticas de escritura en fallo

¿Qué hago con el bloque x ? victima u ?	¿Qué hago con el bloque x ?	
	Cargar x en Mc <i>Fetch on write-miss</i>	No cargar x en Mc <i>No Fetch on write-miss</i>
Reemplazo u <i>write Allocate</i>	[1º acción reemplazo] 2º leer bloque x de Mp 3º escribir palabra x' en Mc <i>AF = convencional</i>	[1º acción reemplazo] 2º escribir palabra x' en Mc bit validez/palabra <i>ANF = write validate</i>
No Reemplazo u <i>No write Allocate</i>	NO es posible	Escribir palabra x' en Mp sin traer nada a Mc <i>NANF = write around</i>

- Las 3 posibilidades pueden funcionar con *write-back* o *write-through*

Tablas de tráfico *Convencional* (i)

■ CB + AF

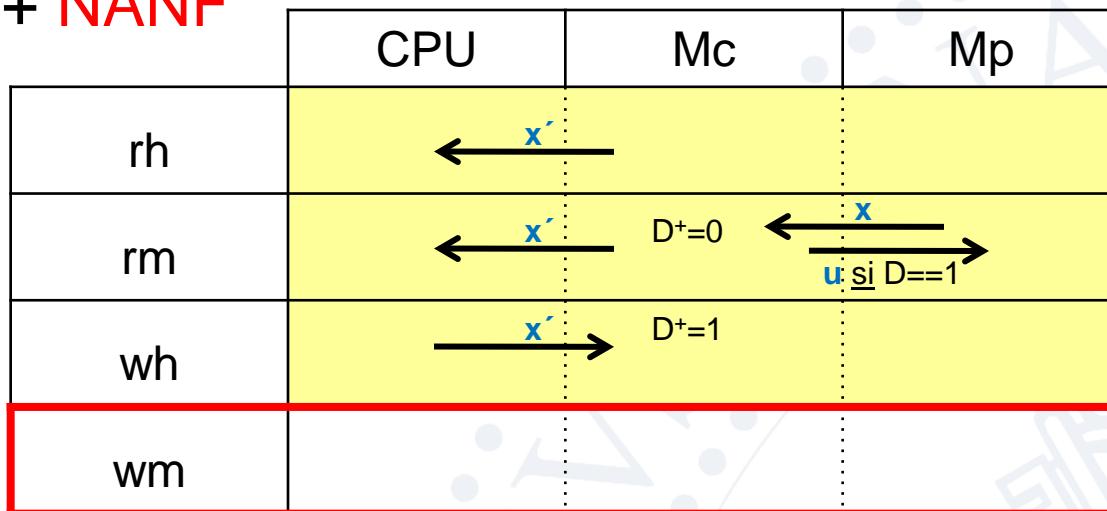
	CPU	Mc	Mp
rh			
rm		$D^+=0$	 <u>si</u> $D==1$
wh		$D^+=1$	
wm			

■ WT + AF

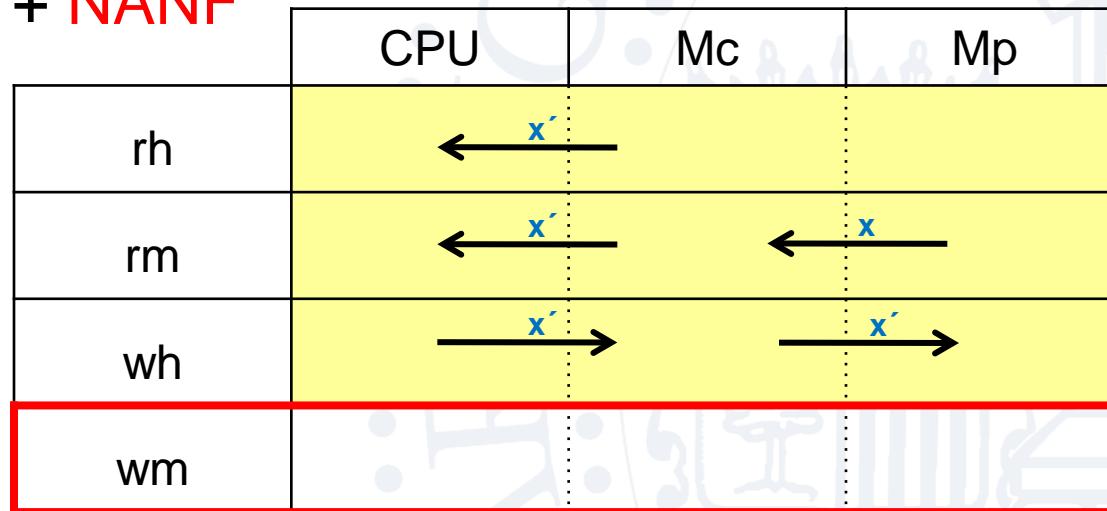
	CPU	Mc	Mp
rh			
rm			
wh			
wm			

Tablas de tráfico *Write Around* (ii)

■ CB + NANF



■ WT + NANF



Tablas de tráfico *Write Validate* (iii)

■ CB + ANF

	CPU	Mc	Mp
rh			
rm		$D^+=0$	 si $D==1$
wh		$D^+=1$	
wm			

■ WT + ANF

	CPU	Mc	Mp
rh			
rm			
wh			
wm			

Ejemplo: inicialización de matriz

- Medida de prestaciones: tráficos CPU-Mc-Mp
 - 4 elementos por bloque

```
for (i = 0; i < max; i++)  
    for (j = 0; j < max; j++)  
        A[i][j] = 0;
```

Calculad tráficos de

- CB+AF
- WT+NANF
- CB+ANF



**Escuela de
Ingeniería y Arquitectura**
Universidad Zaragoza



**Departamento de
Informática e Ingeniería
de Sistemas**
Universidad Zaragoza



Tema 9 – Descripción de Comportamiento

AOC2
Grado en Ing. Informática

P. Ibáñez, J.L. Briz, V. Viñals, J. Alastrauey, J. Resano
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

Guión del tema

- Notación
- Descripción mediante Diagrama de estados
- Ejercicio: diagrama para *write once*
- Descripción mediante Algoritmo
- Calculo de Tiempo Efectivo de Acceso, ejemplos

Notación (1 de 3)

1. Datos y direcciones

- Palabra: minúscula con prima
- Bloque: minúsculas
 - que referencia el procesador
 - expulsado de Mc
- Dirección de
- Dato y su dirección: mayúscula

x'



Bloque x

x

u

víctima

@

$X = <@x, x>$

bloque

$U = <@u, u>$

bloque

$X' = <@x', x'>$

palabra

Notación (2 de 3)

2. Enviar comando desde Mc a Mp

evento: comp_destino (**comando**, item)

rh: read hit
wh: write hit
rm: read miss
wm: write miss
rpl: replacement

Mp

rB: read block
wB: write block
rW: read word
WW: write word

parámetros del comando: @x, U, ...

Comandos entre otras parejas de componentes;
por ejemplo *desde Mc1 hacia Mc2*

evento: Mc1>Mc2 (**comando**, item)

Notación (3 de 3)

3. Cargar bloque o palabra en Mc

- Operador “+”

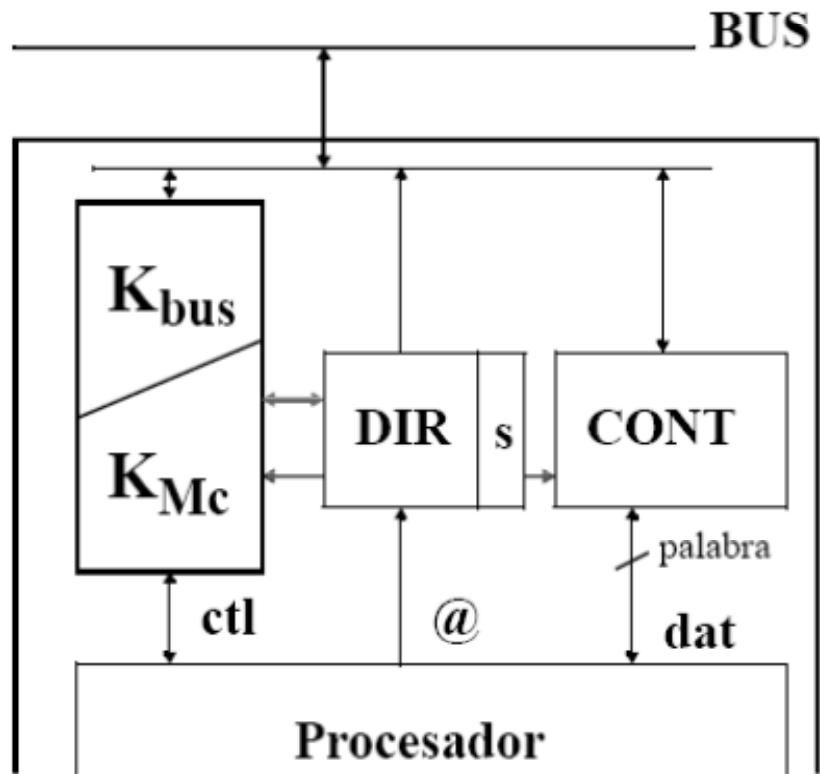
$Mc + x$; después de un fallo,
el bloque x se carga en Mc

4. Invalidar bloque en Mc

- Operador “-”

$Mc - u$; invalida el bloque u
el contenedor puede recibir un nuevo bloque

Papel del controlador de bus y Mc



- Estado S de cada bloque según las reglas de escritura y reemplazo
 - validez, sucio/limpio, ...
 - Estado de reemplazo
- Primer nivel en chip normalmente separado: datos e instrucciones
- Siguientes niveles en chip normalmente unificados

- **K_{Mc}**: implementa comunicación con el procesador y cambio de estado
- **K_{bus}**: implementa el autómata de comunicación con el bus

$K_{BUS} + K_{Mc}$: una máquina de estados

- Describimos un contenedor cualquiera de Mc
- Sólo detallamos relación Mc – Bus
 - P.e. no ponemos $CPU+x'$ o $Mc+x$

1. Localizar

- selección de contenedor
→ el que contiene el bloque x o el bloque víctima u
- comprobación de acierto/fallo

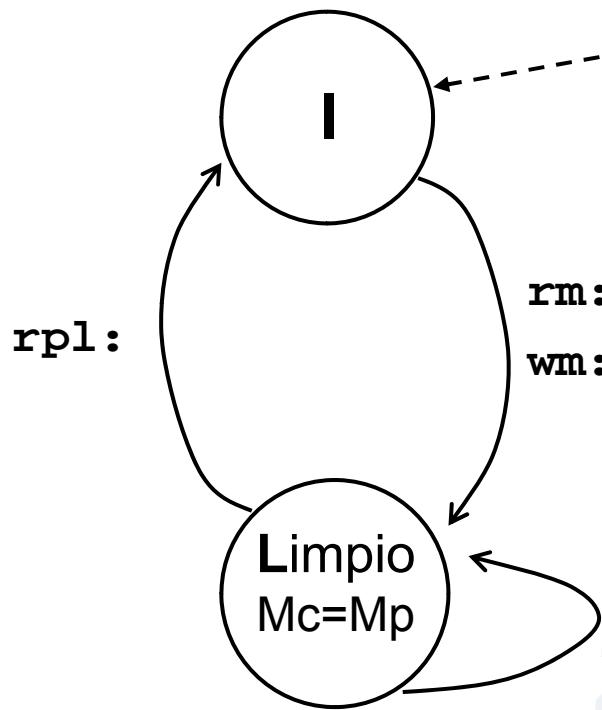
Esto no se ve en el diagrama de estados

2. Acierto: 1 transición ($rh \circ wh$)

Fallo: 1 transición (rpl) + 1 transición ($rm \circ wm$)

Escritura Inmediata – Write Through + AF

2 estados: Inválido y Limpio. Un solo bit por bloque (V bit)



Fallo en escritura:
Fetch on write-miss (AF)

COPIAS en Mc
COHERENTES con Mp

Comandos	¿cuándo?	comentario
Mp(rB, @x)	m (r ó w)	leemos bloque en fallo
Mp(ww, x')	w (h ó m)	escribimos palabra en Mp siempre

Escritura Retardada – Copy-Back + AF

3 estados: Inválido, Limpio y Sucio. Dos bits por bloque: V bit + D bit

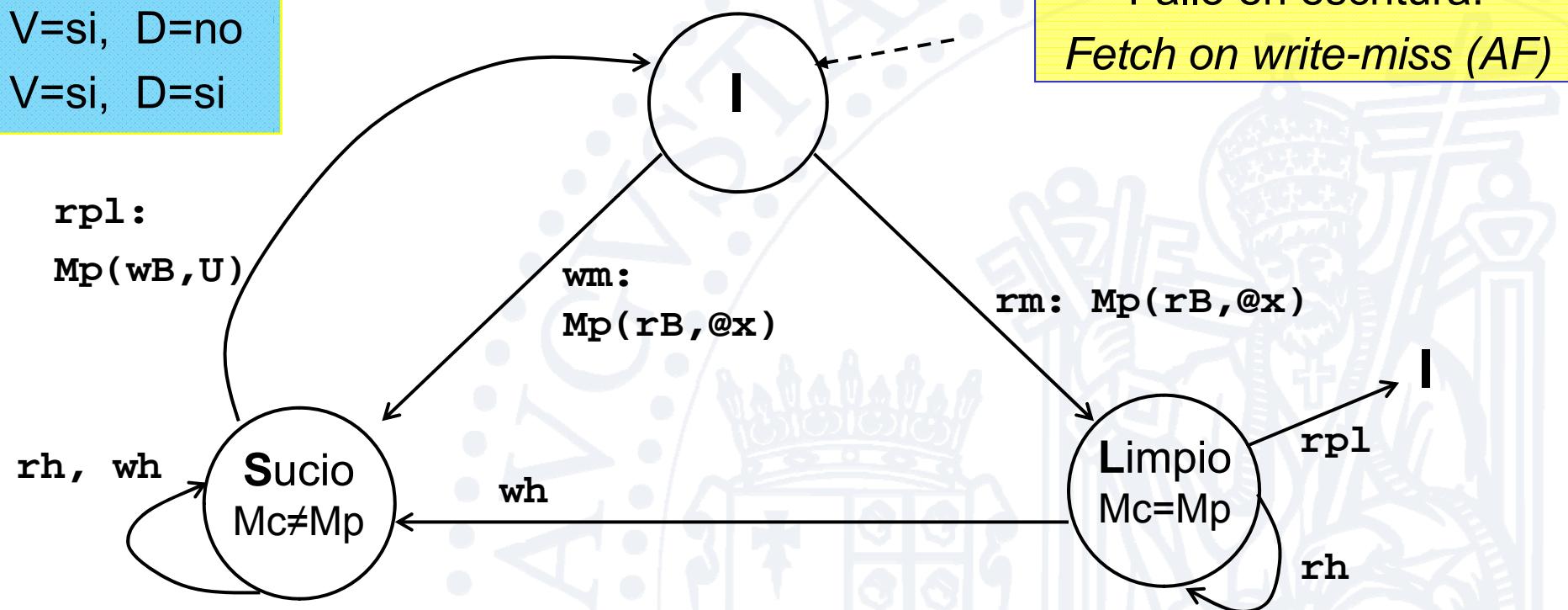
I: V=no, D=?

L: V=si, D=no

S: V=si, D=si

Fallo en escritura:

Fetch on write-miss (AF)



Comandos	¿cuándo?	comentario
$M_p(wW,$	nunca	No es posible
$M_p(wB,$	rpl ^ S	El algoritmo de reemplazo selecciona un bloque víctima u y lo envía a memoria principal
$M_p(rB,$	m	Se carga el bloque x . En caso de escritura se cambia el valor de x en el bloque recién traído

Ejercicio: Write Once

Aplica Write-Through en el primer acierto o fallo de escritura.

En el resto aplica Write-Back.

Política de carga de bloques: fetch on write-miss (AF)

Utilizado en el Motorola M88200

4 estados: Inválido, Limpio 0, Limpio 1 y Sucio.

Dos bits por bloque: V bit + D bit

M88200: chip con 16KB de SRAM cache + controlador pensado para funcionar con el micro Motorola 88000 (1990)

Escritura Inmediata Sin Buffering + AF

■ Algoritmo

Especificación	Tiempos
look-up(@x) ;	1
if miss(@x) {Mp(rB,@x) ; waitfor Mp; Mc+X; }	CrB + 1
MRU(@x) ;	0
switch (proc_r/w)	
case proc_r: ret x' ;	0
case proc_w: {Mc+x' ; Mp(wW,X') ; waitfor Mp; ret ; }	CwW

- CrB lectura de bloque desde Mp (latencia + transferencia)
- CwW escritura de palabra en Mp

Escritura Inmediata Sin Buffering + AF

- Duración media de un acceso a datos = ciclos efectivos de acceso (ciclos/ref)

$$C_{eff} = 1 + \frac{\sum wh \times C_{wW}}{\sum refs} + \frac{\sum rm \times (CrB + 1)}{\sum refs} + \frac{\sum wm \times (CrB + 1 + C_{wW})}{\sum refs}$$

- $CrB + 1$ penalización de fallo en lectura
- $CrB + 1 + C_{wW}$ penalización de fallo en escritura

Escritura Retardada sin Buffering + AF

■ Algoritmo

Especificación	Tiempos
look-up(@x);	1
if miss(@x) { u=LRU(); Mc-u; if sucio(u) {Mp(wB,U); waitfor Mp; } Mp(rB,@x); waitfor Mp; Mc+X;	0
MRU(@x);	CrB+1
switch (proc_r/w) case proc_r: ret x'; case proc_w: {Mc+x'; sucio(x)=true; ret ; }	0 1

- CwB escritura del bloque hacia Mp (transferencia + latencia)
 - CrB lectura de bloque desde Mp (latencia + transferencia)

Escritura Retardada Sin Buffering

- Cálculo del tiempo efectivo en ciclos (ciclos/ref)

$$C_{eff} = 1 + \frac{\sum wh \times 1}{\sum refs} + \frac{\sum rm \times (CrB + 1)}{\sum refs} + \frac{\sum wm \times (CrB + 2)}{\sum refs} + \frac{\sum (m \wedge sucio) \times CwB}{\sum refs}$$

- CrB + 1 penalización de fallo en lectura
- CrB + 2 penalización de fallo en escritura
- CwB penalización de fallo sucio



**Escuela de
Ingeniería y Arquitectura**
Universidad Zaragoza



**Departamento de
Informática e Ingeniería
de Sistemas**
Universidad Zaragoza



Tema 6 – Incremento de rendimiento

P. Ibáñez, J.L. Briz, V. Viñals, J. Alastrauey, J. Resano

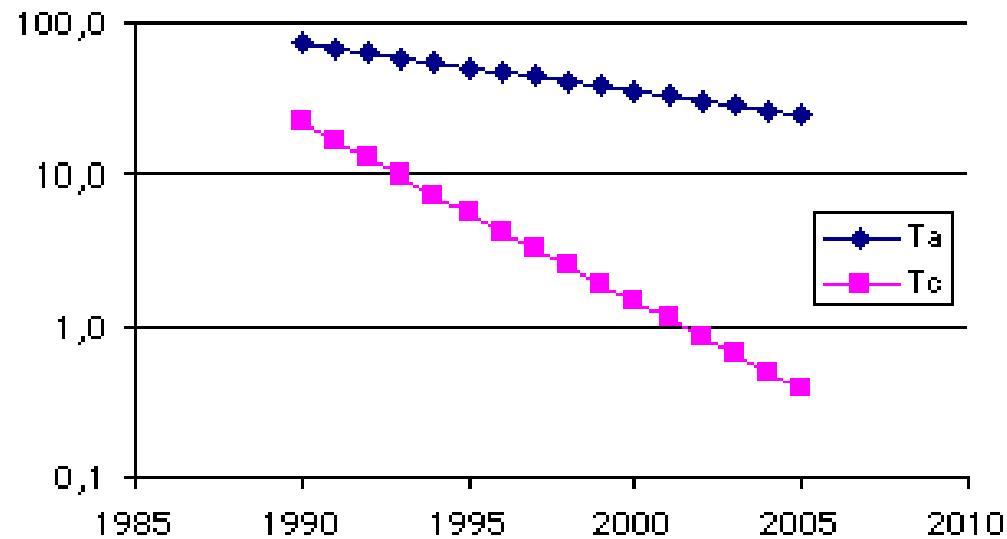
Arquitectura y Tecnología de Computadores

Departamento de Informática e Ingeniería de Sistemas

Guión del tema

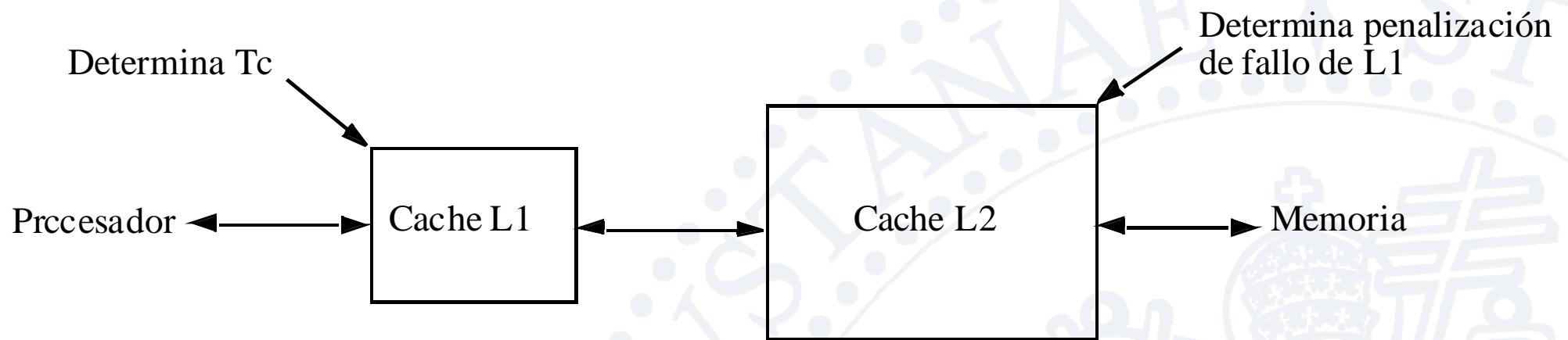
- El problema de la creciente penalización del fallo
- Organización multinivel
- Optimización de código por el compilador

Problema: creciente penalización del fallo



- Objetivos memoria cache
 - Servir al procesador a su ritmo
 - ◆ Diseñar cache pequeña y sencilla
 - Minimizar la tasa de fallos
 - ◆ Diseñar cache grande y compleja

Organización multinivel

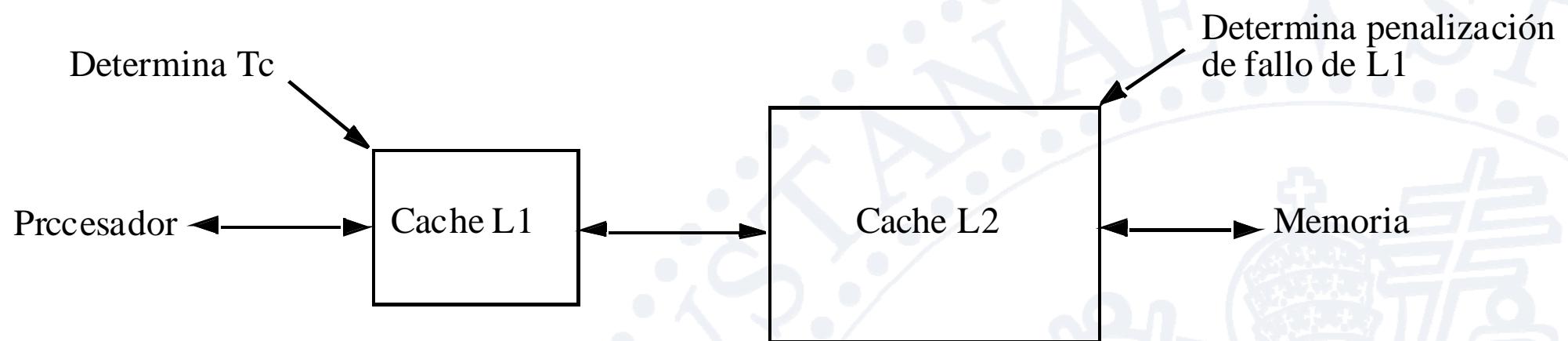


$$Tasa\ fallos\ L_1 = m_1 = \frac{fallos\ L_1}{accesos\ L_1}$$

$$Tasa\ fallos\ L_2 = m_2 = \frac{fallos\ L_2}{accesos\ L_2}$$

$$Tasa\ fallos\ global = m_g = \frac{fallos\ L_2}{accesos\ L_1} = \frac{fallos\ L_2}{accesos\ L_2} \times \frac{fallos\ L1}{accesos\ L_1} = m_2 \times m_1$$

Organización multinivel



$$Tef = Th_{L1} + m_1 * P_{L1} = Th_{L1} + m_1 \times [Th_{L2} + m_2 \times P_{L2}]$$

- P_{L1} : penalización por fallo de L1
- Th_{L2} : tiempo para transferir un bloque de L2 a L1
- P_{L2} : tiempo para transferir un bloque de Mp a L2 y L1

Parámetros de diseño en L2

- Objetivo diseño L2: minimizar tasa de fallos
 - Su tiempo de acceso no es el factor más determinante en Ta

$$Tef = Th_{L1} + m_1 \times P_{L1} = Th_{L1} + m_1 \times [Th_{L2} + m_2 \times P_{L2}]$$

- Tamaño mucho mayor que L1
- Asociatividad alta
- Tamaño de bloque mayor que en L1
- ¿Política de escritura?

- Inclusión de contenidos

Ejercicio

- Se tienen dos jerarquías, una con un nivel de cache (L1) y otra con dos niveles (L1+L2)
 - Tasa de fallos en L1: $m_1 = 0,1$ (10%)
 - Tasa local de fallos en L2: $m_2 = 0,5$ (50%)
 - Acierto en L1: 1 ciclo
 - Acierto en L2: 6 ciclos
 - Penalización por acceso a Mp: 100 ciclos
- Calcular para las 2 jerarquías
 - Tasa de fallos global
 - Ciclos efectivos de acceso, y T_{ef} si $F = 2$ GHz
- Calcular T_{ef} suponiendo m_1 , m_2 y F igual, pero:
 - $T_a(L1) = 0.5$ ns; $T_a(L2) = 2.8$ ns; $P_{Mp} = 49.55$ ns

Optimización de código por el compilador

■ Fusión de vectores

```
int val[SIZE], key[SIZE];
for (i=0; i<SIZE; i=i+20) {
    if (key[i] < x)
        acum = acum + val[i];
}
```

```
struct merge {
    int val;
    int key;
}
struct merge merged_array[SIZE];
for (i=0; i<SIZE; i=i+20) {
    if (merged_array[i].key < x)
        acum = acum + merged_array[i].val;
}
```

Optimización de código por el compilador

■ Intercambio de bucles (*loop interchange*)

```
for (j=0; j<100; j++) {  
    for (i=0; i<5000; i++)  
        x[i][j] = 2*x[i][j];  
}
```

```
for (i=0; i<5000; i++) {  
    for (j=0; j<100; j++)  
        x[i][j] = 2*x[i][j];  
}
```

Optimización de código por el compilador

■ Fusión de bucles (*loop fusion*)

```
for (i=0; i<N; i++) {  
    for (j=0; j<N; j++)  
        a[i][j]= 1/ b[i][j]* c[i][j];  
}  
for (i=0; i<N; i++) {  
    for (j=0; j<N; j++)  
        d[i][j]= a[i][j] + c[i][j];  
}
```

```
for (i=0; i<N; i++) {  
    for (j=0; j<N; j++) {  
        a[i][j]= 1/ b[i][j]* c[i][j];  
        d[i][j]= a[i][j] + c[i][j];  
    }  
}
```

Reduce sobrecarga bucles,
aumenta localidad ...

Optimización de código por el compilador

■ Fisión de bucles (*loop fission*)

```
for (i = 0; i < n; i++)
    y[i] = y[i] + x[i] + x[i+m];
```

Bajo rendimiento si
x[i] y x[i+m] coinciden en
el mismo bloque de cache
(ej: mapeo directo y
m potencia de 2)

```
for (i = 0; i < n; i++)
    y[i] = y[i] + x[i];

for (i = 0; i < n; i++)
    y[i] = y[i] + x[i+m];
```

Optimización de código por el compilador

■ Distribución de bucles (*loop distribution*)

```
for (i = 0; i < n; i++) {  
    x[i] = y[i] + z[i] + w[i];      /* S1 */  
    a[i+1] = (a[i-1] + a[i])/2.0; /* S2 */  
    y[i] = z[i] - x[i];          /* S3 */  
}
```

S1 y S3 pueden parallelizarse,
S2 no.

→ El bucle **no** puede
parallelizarse

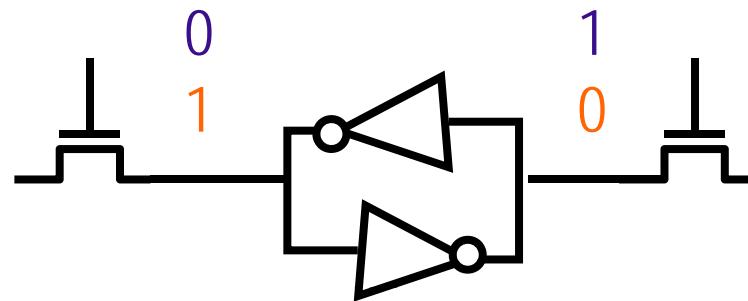
```
/* B1 */  
for (i = 0; i < n; i++) {  
    x[i] = y[i] + z[i] + w[i];      /* S1 */  
    y[i] = z[i] - x[i];          /* S3 */  
}  
/* B2 */  
for (i = 0; i < n; i++) {          /* L2 */  
    a[i+1] = (a[i-1] + a[i])/2.0; /* S2 */  
}
```

B1 puede parallelizarse,
B2 se ejecutará de forma
secuencial

Memorias

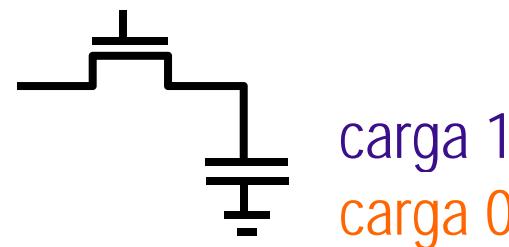
- ◆ Estáticas (SRAM)

- Información permanece mientras haya alimentación
- Alto coste, respuesta rápida
- Uso: caches, bancos de registros, controladores



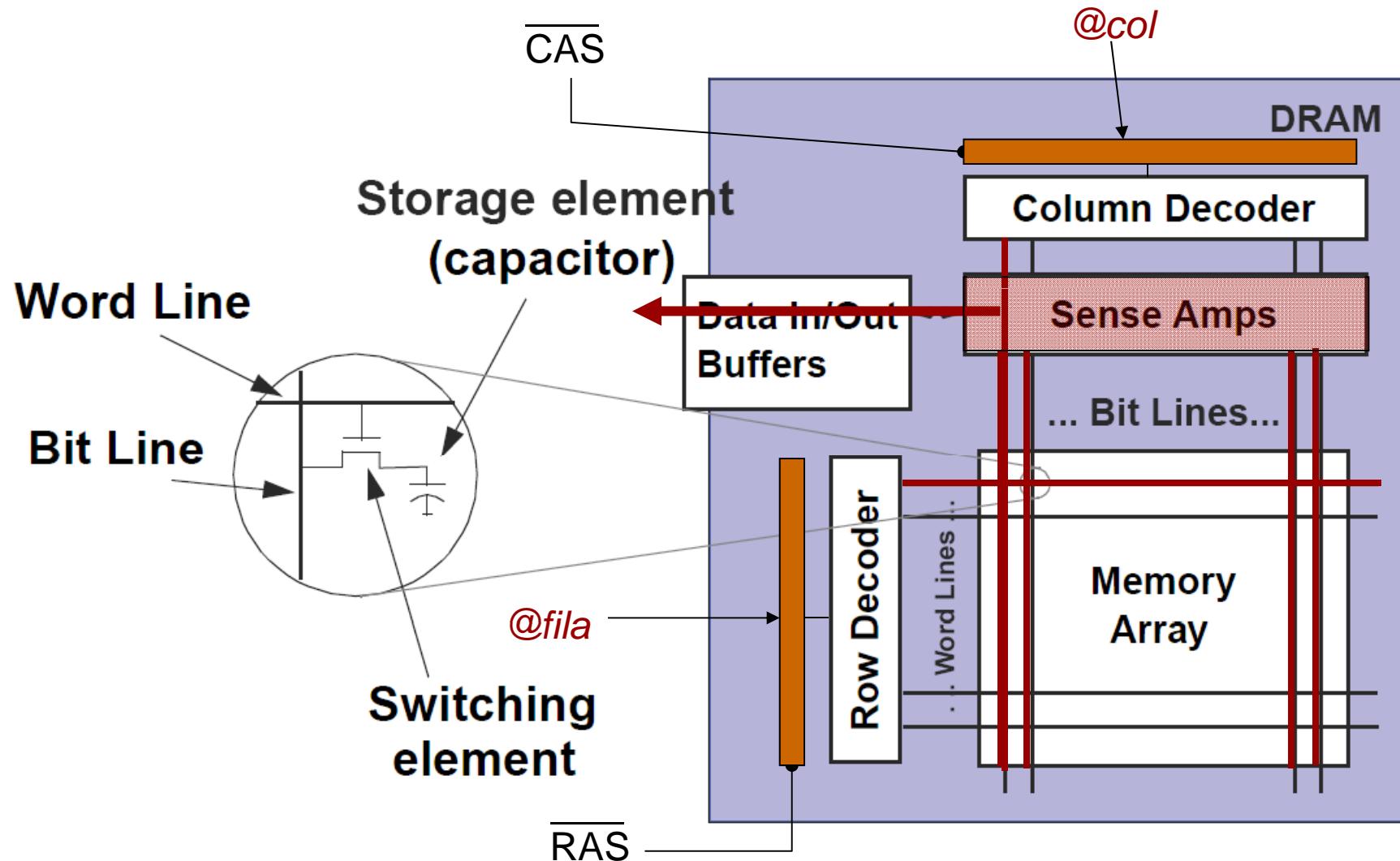
- ◆ Dinámicas (DRAM)

- Información se desvanece incluso con alimentación
- Precisan refresco periódico
- Alta capacidad a bajo coste, respuesta lenta
- Uso: Memorias principales



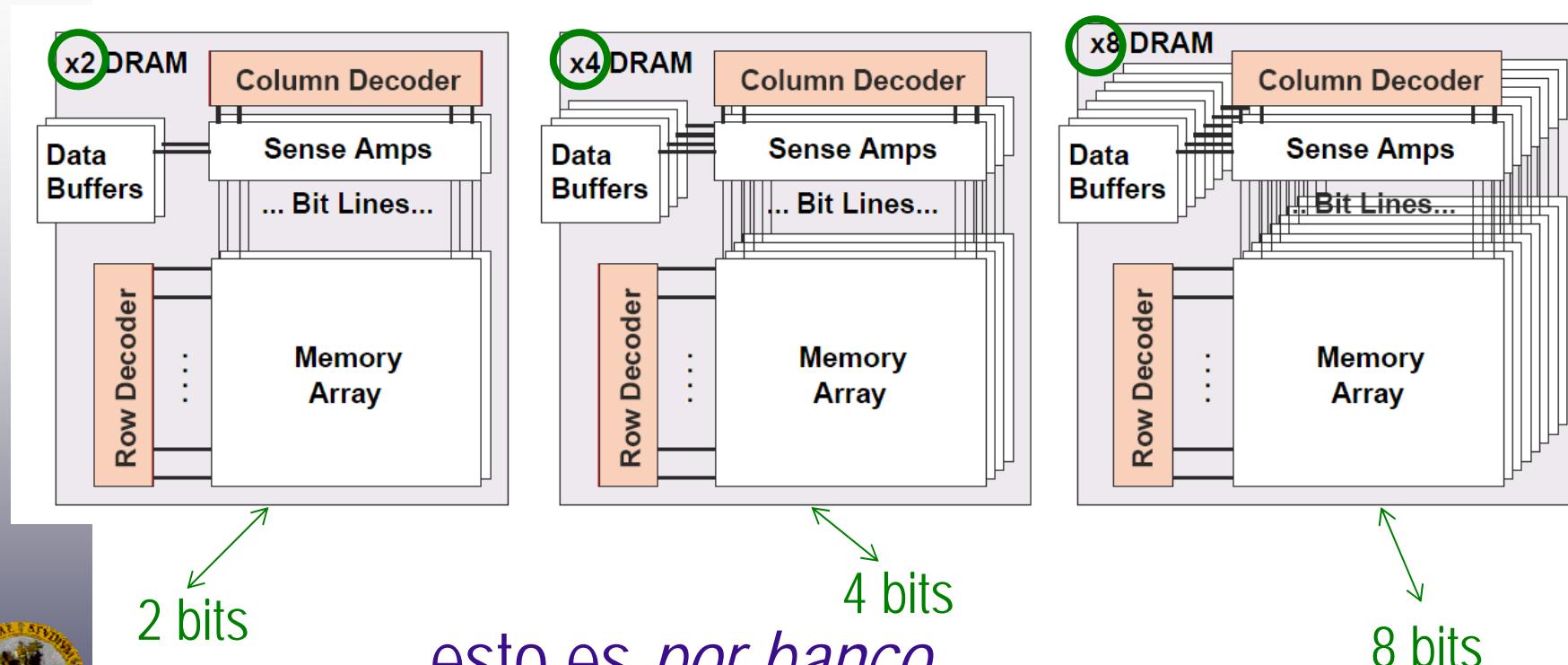
gaZ

DRAM- lectura de 1 bit de un núcleo DRAM



DRAM- bancos

- Normalmente necesitamos leer más de 1 bit de vez:

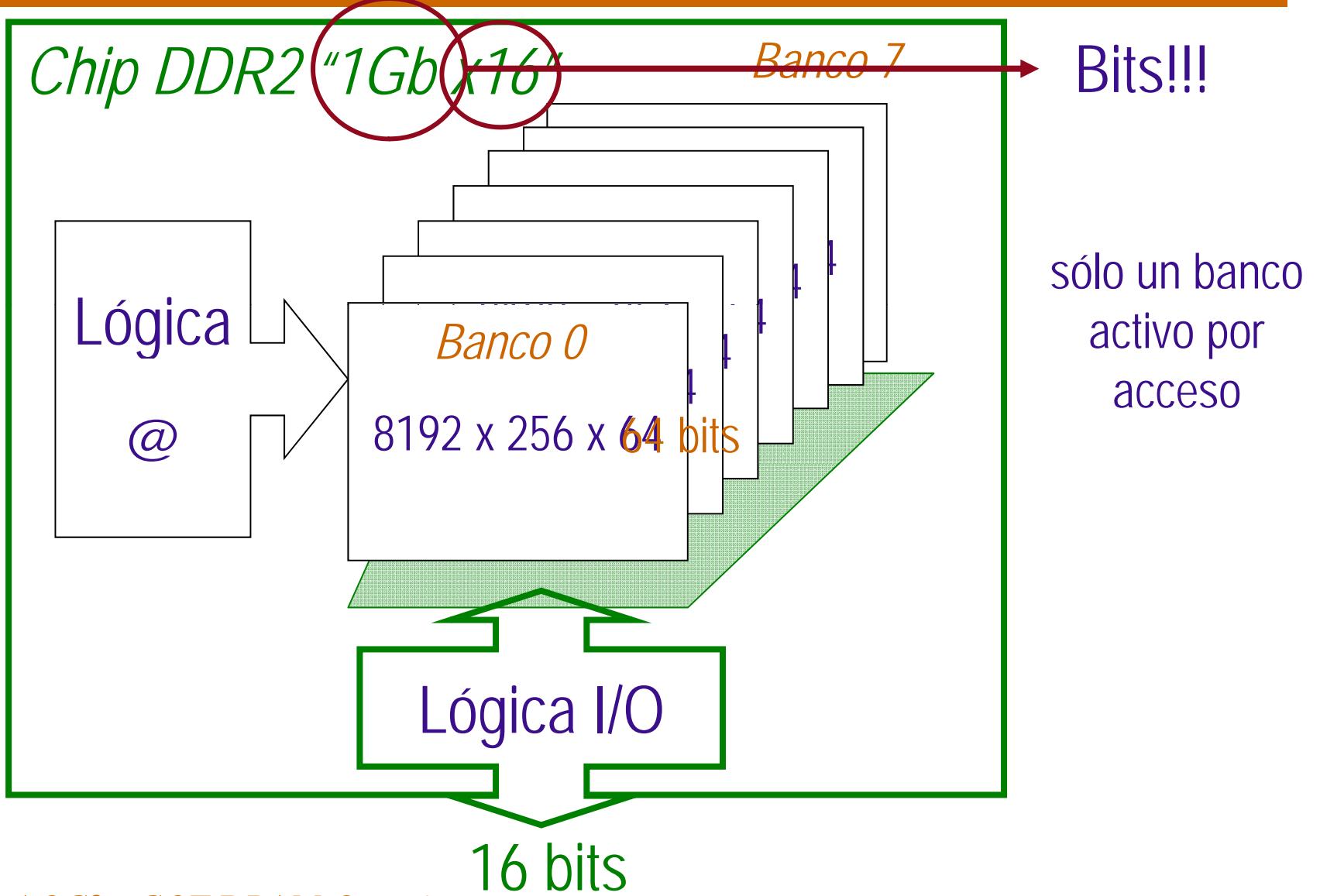


...esto es *por banco*
Chip DRAM: varios bancos



gaz

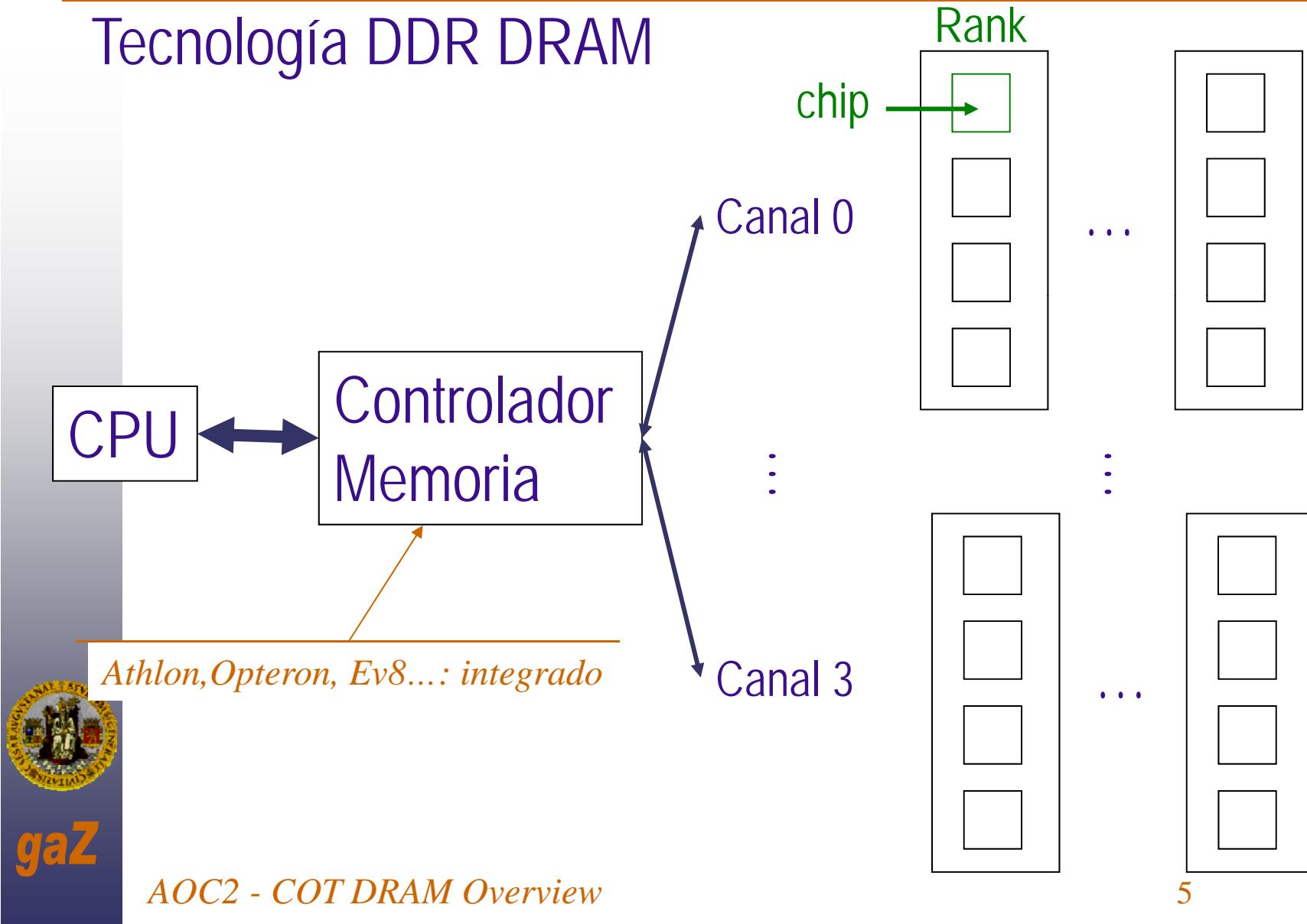
DRAM- Chips y denominación comercial



gaZ

DRAMs: Chips, Ranks, Channels

Tecnología DDR DRAM



gaZ

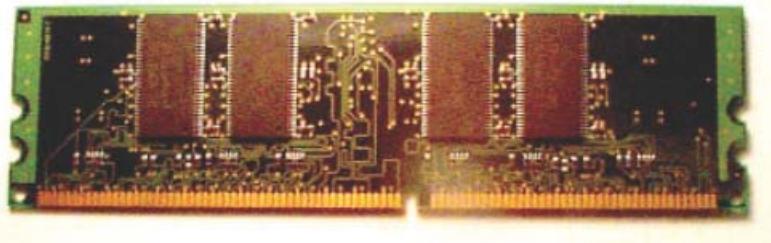
Notación comercial SDRAM DDR para PCs

Type	Component naming convention	Module naming convention	Bus speed	Peak bandwidth
DDR1	DDR200	PC1600	100 MHz	1.6 GB/s
	DDR266	PC2100	133 MHz	2.1 GB/s
	DDR333	PC2700	166 MHz	2.7 GB/s
	DDR400	PC3200	200 MHz	3.2 GB/s
DDR2	DDR2-400	PC2-3200R	200 MHz	3.2 GB/s
	DDR2-533	PC2-4300	266 MHz	4.3 GB/s
	DDR2-667	PC2-5300	333 MHz	5.3 GB/s
	DDR2-800	PC2-6400	400 MHz	6.4 GB/s
DDR3	DDR3-800	PC3-6400	400 MHz	6.4 GB/s
	DDR3-1066	PC3-8500	533 MHz	8.5 GB/s
	DDR3-1333	PC3-10600	667 MHz	10.6 GB/s
	DDR3-1600	PC3-12800	800 MHz	12.8 GB/s



DRAMs: DIMMs, SIMMs

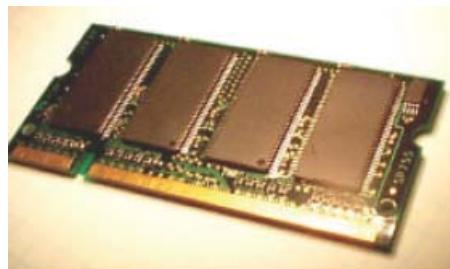
Tecnología DDR DRAM



Sobremesa, servidores
184 pines, 64 MB - 2 GB



DIMM



SIMM

Móvil, portátiles
200 pines, 32 - 512 MB



gaZ

DRAMs: Chips, Ranks, Channels

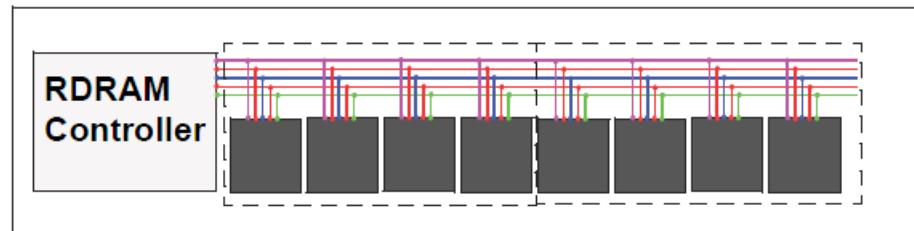
- ◆ Impresiónese a Vd. Mismo o comente con el compañero
 - Dispongo de chips x16
 - Los ranks (DDR) son de 4 chips
 - ¿cuántos bits puedo leer /escribir en una operación?



gaZ

DRAMs: Chips, Ranks, Channels

Tecnología Rambus (RDRAM, DRDRAM)



- ◆ Controlador integrado en SIMM
- ◆ Rank = chip
 - (32 chips típico)
- ◆ Chips especiales
Pago royalties...



gaZ

DRAMs: DDR vs. Rambus

	DDR	RDRAM
MHz	133*2	400*2
Pines bus datos	64	16
Pines controlador	101	33
MB/s teóricos	2128	1600
Eficiencia teórica (bits datos /ciclo/pin)	0.63	0.48
MB/s reales	986	1072
Latencia RAS + CAS (ns)	45~50	57~67
Latencia CAS (ns)	22~30	40~50



gaZ

Mercado DRAMs: DDR vs. Rambus

- ◆ DDR /DDR2
 - Menos escalable
 - Conexiones complejas controlador – ranks
 - Interfaz estándar, chips baratos, lógica DIMMs simple
- ◆ RDRAM, DRDRAM
 - Escalable
 - Conexiones controlador – ranks simples: bus
 - Interfaz y chips propietarios, lógica SIMMs compleja
 - Pago derechos



gaZ

Tema 12: Características y protocolos de los buses

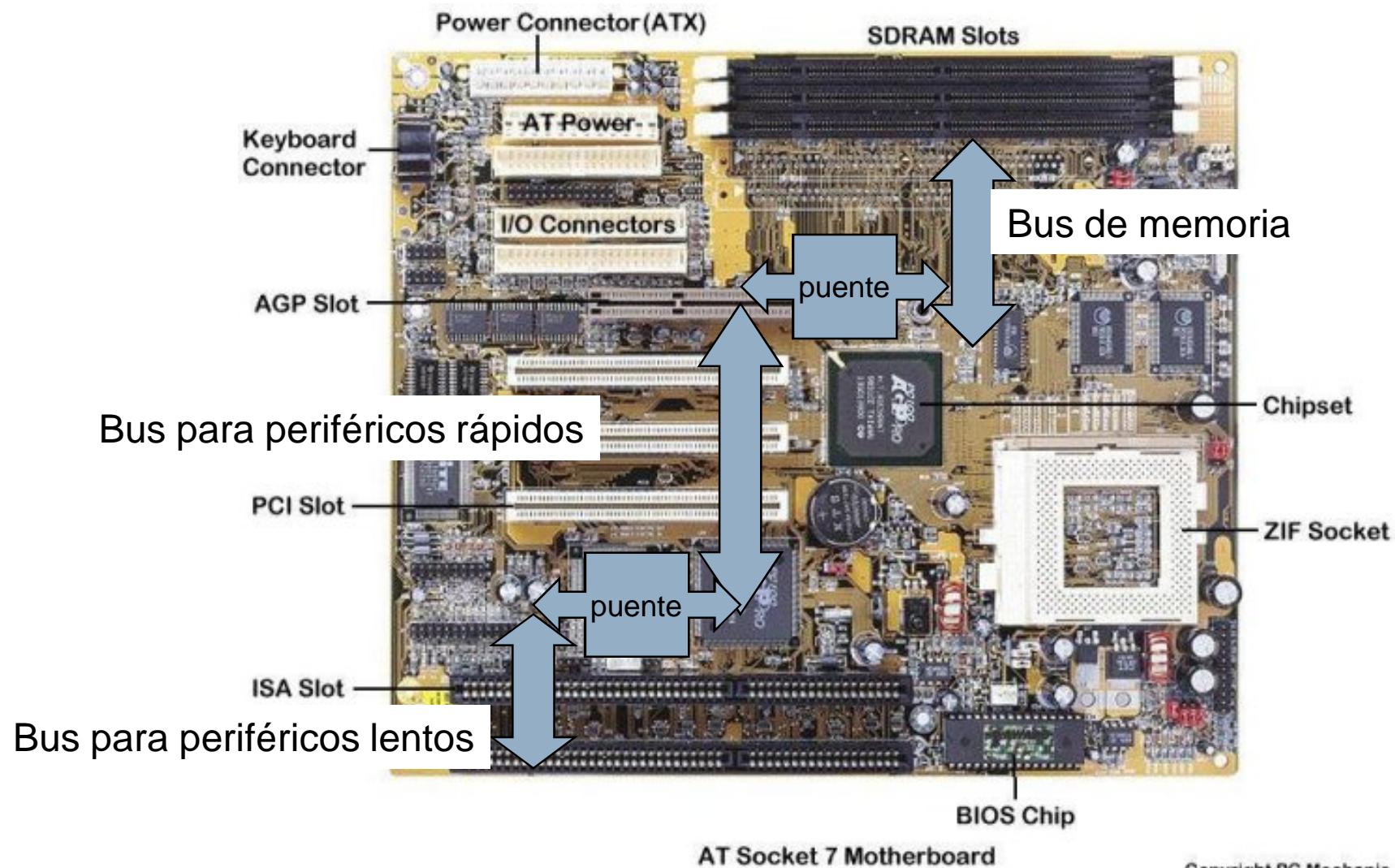


**Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza**

Características y protocolos de los buses

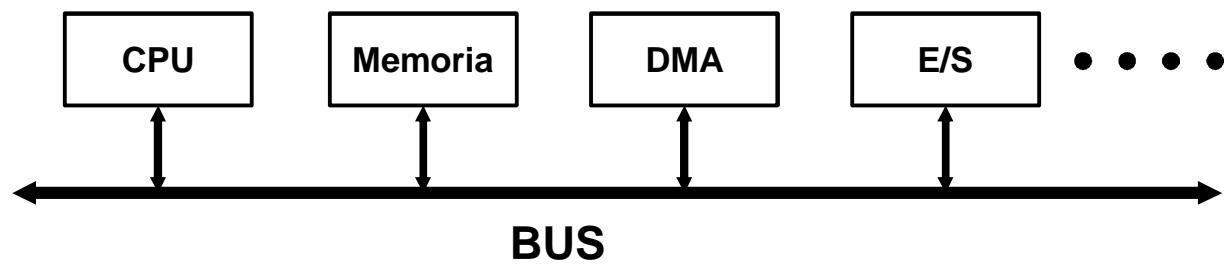
- 1. Importancia de la comunicación entre dispositivos**
- 2. Características de un bus**
- 3. Protocolos de transferencia**
- 4. Protocolos de arbitraje**

Motivación: ¿cómo se comunican todos estos elementos entre si?



Motivación

- Un computador está formado por múltiples elementos que colaboran entre sí
- Estos elementos deben intercambiar datos y sincronizarse para funcionar correctamente
- Los buses de una computadora proporcionan el soporte necesario para estas comunicaciones
- Los buses son elementos caros así que se comparten entre varios dispositivos



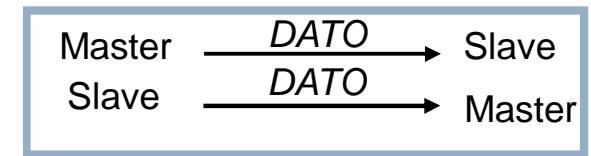
Definiciones

Un bus es un medio de transmisión compartido que interconecta dos o más dispositivos de un sistema digital

- **Función:** permitir la correcta comunicación entre los dispositivos interconectados
- **Estructura:**
 - Conjunto de líneas (conductores eléctricos) compartidas por todos los dispositivos
 - Cualquier señal transmitida por un dispositivo está disponible para los demás
 - Si dos dispositivos transmiten simultáneamente utilizando las mismas líneas las señales se solapan y la información resultante es inválida
 - Conexiones en paralelo: varias señales viajan juntas sobre distintas líneas

Definiciones

- **Elementos** implicados en una transferencia:
 - **Maestro (master)**: Inicia y dirige las transferencias (CPU, DMA, ...)
 - **Esclavo (slave)**: Obedece y accede a las peticiones del master (memoria, interfaz E/S, ...)
- **Tipos básicos** de transferencia:
 - **Escritura**: el maestro envía datos al esclavo
 - **Lectura**: el maestro pide datos al esclavo
- **Ciclo de bus**: pasos a seguir para realizar una transferencia (puede realizarse en uno o varios ciclos de reloj)
 - Operaciones básicas:
 1. **Direccionamiento** del slave
 2. Especificación del **tipo de operación** (lectura o escritura)
 3. **Transferencia** del dato
 4. **Finalización** del ciclo de bus



Definiciones

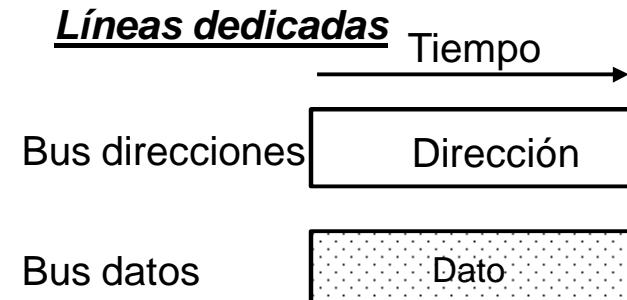
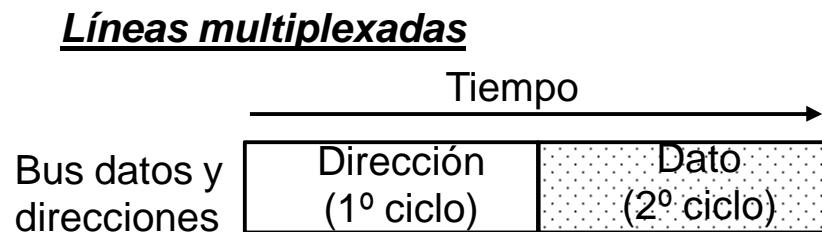
- **Sincronización:** método para determinar el inicio y el final de cada transferencia
- **Arbitraje:** método para controlar el acceso al bus en caso de varios masters
- **Capacidad de conexión:** número máximo de dispositivos conectables
- **Longitud de bus:** máxima distancia que puede separar a dos dispositivos conectados
- **Ancho de bus:** número total de líneas
- **Ancho de datos:** número total de líneas para datos
- **Ancho de banda:** caudal máximo de información que puede transmitirse

Clasificación de las líneas del bus según su función

- **Líneas de datos (*bus de datos*):** transmiten datos
- **Líneas de direcciones (*bus de direcciones*):** designan la fuente o el destino de un dato
- **Líneas de control:** gobiernan el acceso y el uso de las líneas de datos y direcciones
 - *Líneas de operación:* determinan el tipo de operación que debe realizar el slave
 - *Líneas de sincronización:* determinan el comienzo y final de cada transferencia
 - *Líneas de arbitraje:* determinan cuál de los elementos conectados puede usar el bus

En algunos buses una misma línea puede realizar más de una función

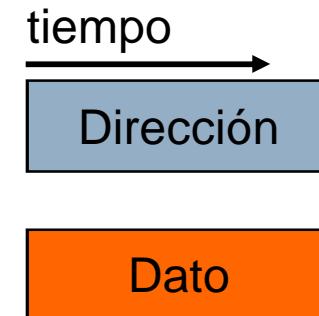
- **Líneas dedicadas:** tienen asignada una única función
- **Líneas multiplexadas:** realizan distintas funciones a lo largo del tiempo
 - *Ventajas:* menor número de líneas ⇒ ahorra espacio ⇒ menor coste
 - *Desventajas:* Circuitería más compleja y menor rendimiento (las funciones que realizan las líneas multiplexadas no pueden realizarse en paralelo)



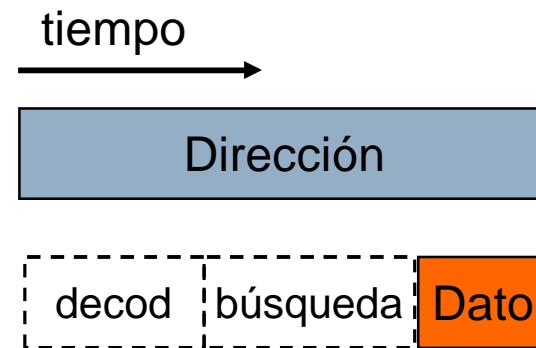
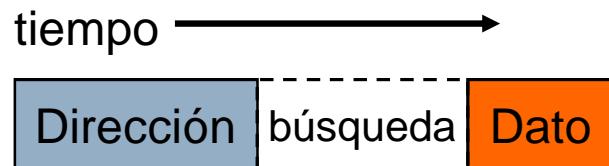
protocolos de transferencia

- **Función:**
 - Sincronizar los elementos implicados en una transferencia (master y slave)
 - Determinar el comienzo y el final de cada transferencia
- **Tipos de transferencia:**
 - Lectura
 - Escritura
 - Lectura de bloque
 - Escritura de bloque
 - Lectura-modificación-escritura
 - Lectura después de escritura
- **Tipos de protocolos de transferencia:**
 - Síncrono
 - Asíncrono
 - Semisíncrono
 - Ciclo partido

Tipos de transferencia



Operación de escritura



Operación de lectura

Tipos de transferencia

tiempo —————→



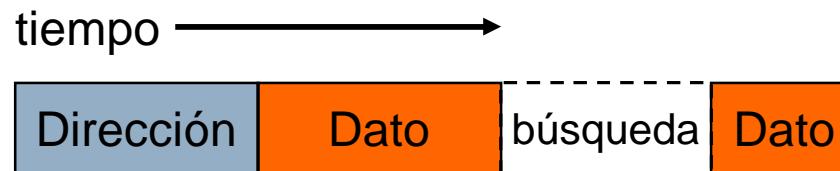
Operación de escritura de bloque

tiempo —————→



Operación de lectura de bloque

Tipos de transferencia



Operación de lectura tras escritura

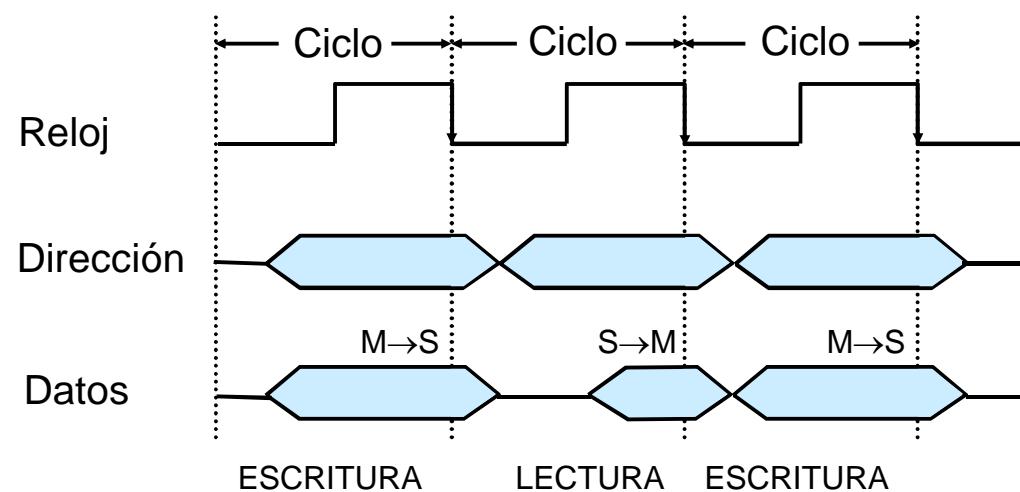
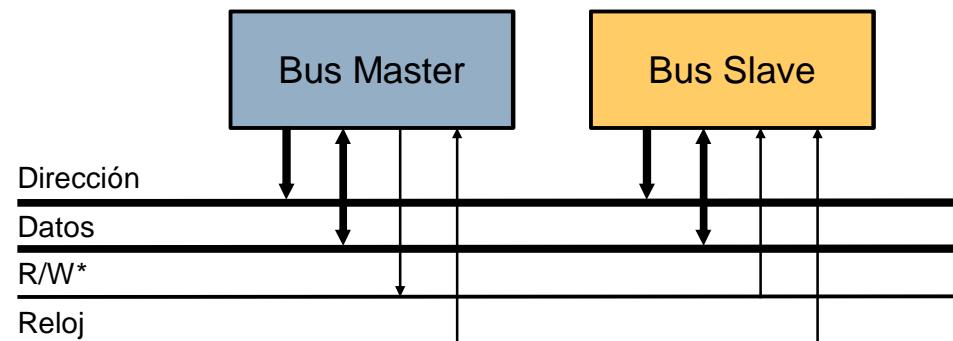


Operación de lectura, modificación y escritura

Protocolo de transferencia síncrono

- Las transferencias están gobernadas por una **única señal de reloj** compartida por todos los dispositivos
- Cada transferencia se realiza en un número fijo de periodos de reloj (1 en el ejemplo)
- Los **flancos del reloj** (de bajada en el ejemplo) determinan el comienzo de un nuevo ciclo de bus y el final del ciclo anterior

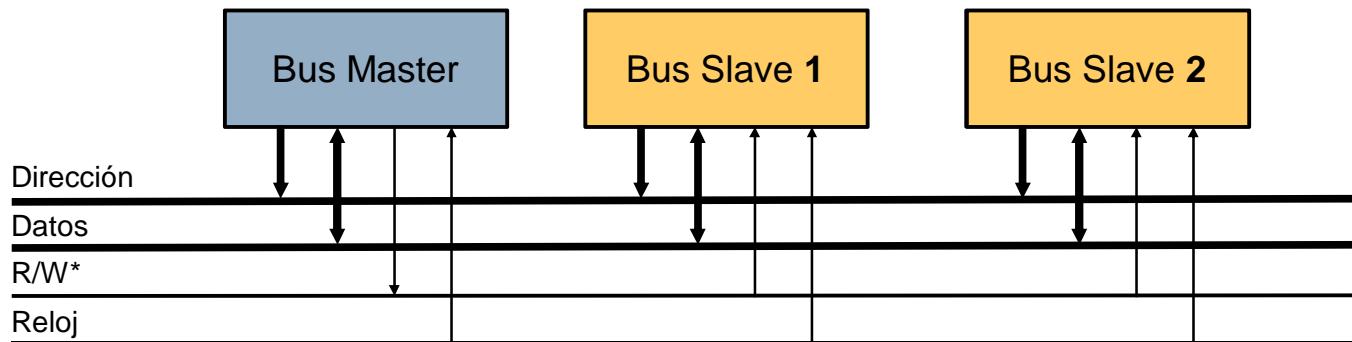
Protocolo de transferencia síncrono



Protocolo de transferencia síncrono

- *Ventajas:*
 - Simplicidad (de diseño y de uso)
 - Sólo se necesita una señal (reloj) para llevar a cabo la sincronización
 - Mayor velocidad (en relación a protocolo asíncrono)
- *Desventajas:*
 - El periodo de reloj **se tiene que adaptar a la velocidad del dispositivo más lento** (por lo que suele usarse para conectar dispositivos homogéneos)
 - **No existe confirmación de la recepción de los datos**
 - La necesidad de distribuir la señal de reloj limita la longitud del bus:
 - **Tiempo de desplazamiento relativo (skew time):** diferencia de tiempo entre la llegadas al receptor de dos señales que partieron del emisor simultáneamente, o la llegada de una señal a dos receptores distintos
 - **Cuanto mayor es el bus, mayor es el skew**

Protocolo de transferencia síncrono



Tiempo de decodificación: 30 ns

Tiempo de setup: 2 ns

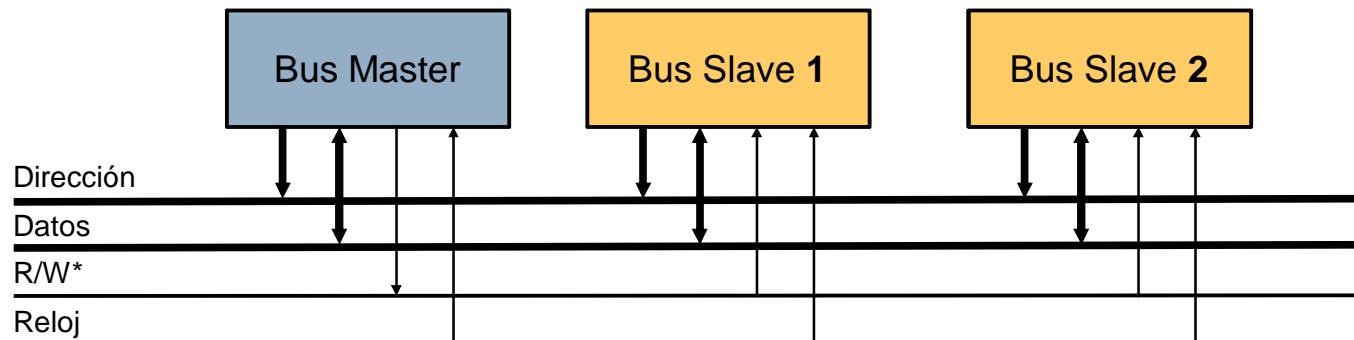
Skew: 2 ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Si queremos que se haga una transferencia por ciclo:

Protocolo de transferencia síncrono



Tiempo de decodificación: 30 ns

Tiempo de setup: 2 ns

Skew: 2 ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Si queremos que se haga una transferencia por ciclo:

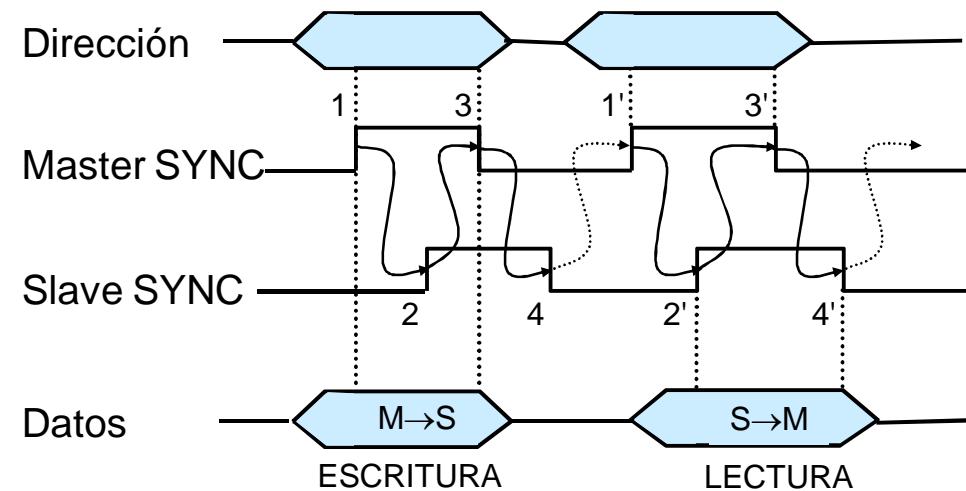
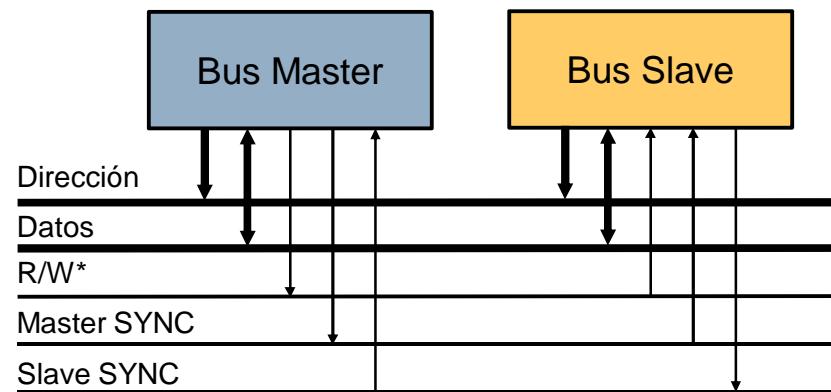
Tiempo de ciclo: $30+2+2+\max(100,1000)= 1034\text{ns}$

Óptimo para el slave 2 pero muy ineficiente para el 1

Protocolo de transferencia asíncrono

- **La señal de reloj se sustituye por dos señales de sincronización:**
 - Master SYNC (procedente del master)
 - Slave SYNC (procedente del slave)
 - A cada flanco del master le sigue uno del slave
- **Ciclo de escritura:**
 - 1: (M a S) Hay un dato en el bus
 - 2: (S a M) He tomado el dato
 - 3: (M a S) Veo que lo has tomado
 - 4: (S a M) Veo que lo has visto (Bus libre)
- **Ciclo de lectura:**
 - 1': (M a S) Quiero un dato
 - 2': (S a M) El dato está en el bus
 - 3': (M a S) He tomado el dato
 - 4': (S a M) Veo que lo has tomado (Bus libre)

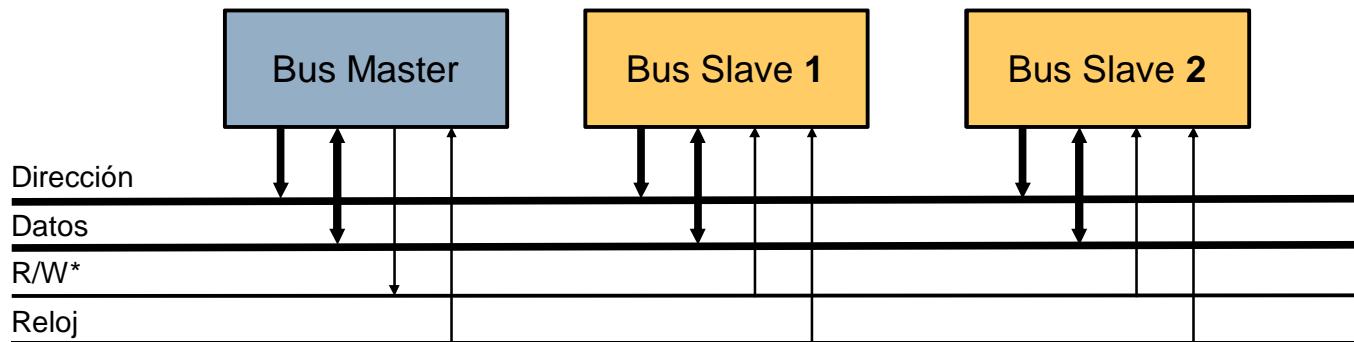
Protocolo de transferencia asíncrono



Protocolo de transferencia asíncrono

- *Ventajas:*
 - Facilidad para conectar elementos de diferentes velocidades
 - Fiabilidad: la recepción siempre se confirma
- *Desventajas:*
 - El intercambio de señales de control introduce retardos adicionales
 - A igualdad de velocidades de los dispositivos, menos eficiente que el síncrono
- *Ejemplos:* Unibus (PDP-11), MC680XX(10,20,30), Bus VME, FutureBus+

Protocolo de transferencia asíncrono



Tiempo de decodificación: 30 ns

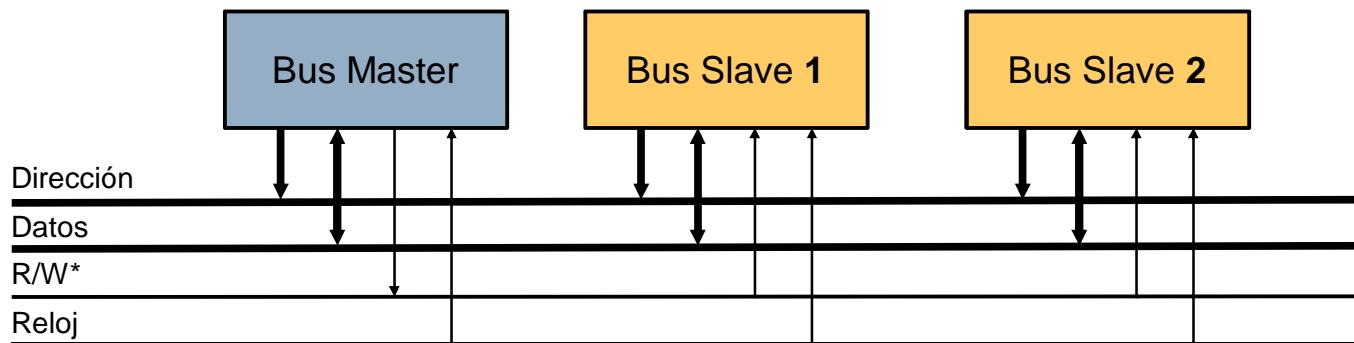
Tiempo de intercambio de señales de sincronización: 10ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Si queremos que se haga una transferencia por ciclo:

Protocolo de transferencia asíncrono



Tiempo de decodificación: 30 ns

Tiempo de intercambio de señales de sincronización: 10ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Si queremos que se haga una transferencia por ciclo:

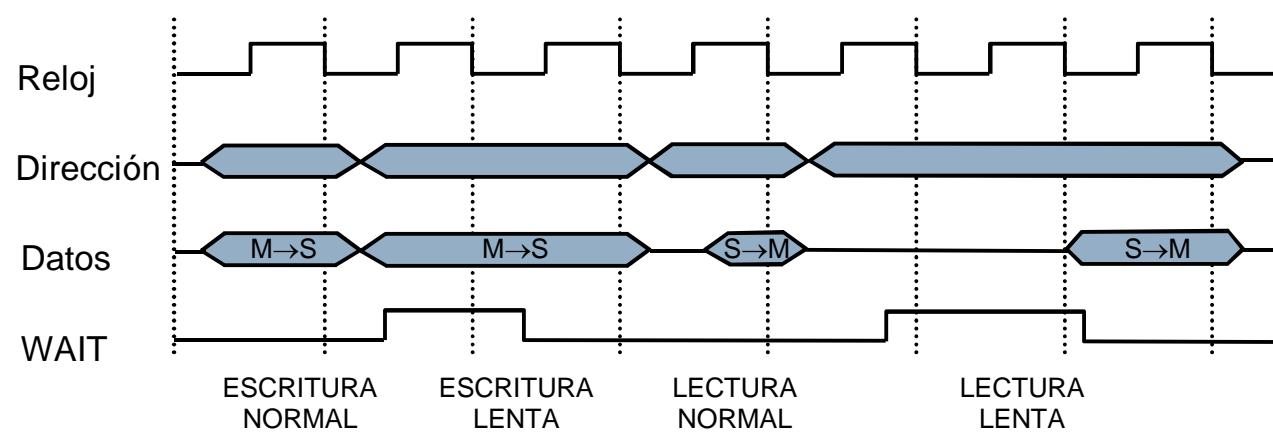
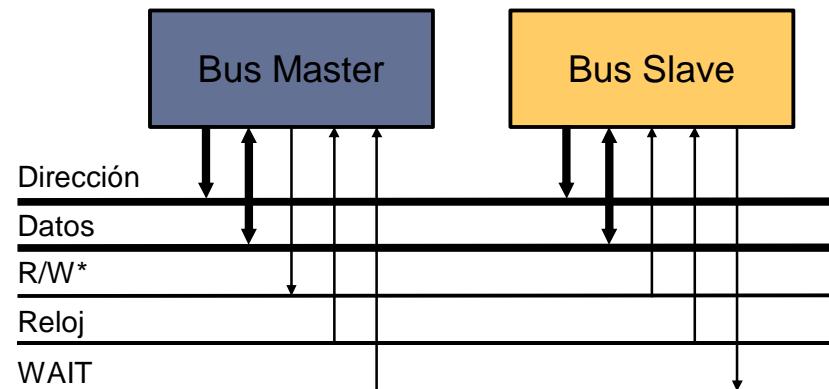
Tiempo de transferencia de lectura:

- slave1: $30+10+10+10+100 = 170\text{ns}$
- slave2: $30+10+10+10+10+1000 = 1070\text{ns}$
- tiempo medio: **620ns**

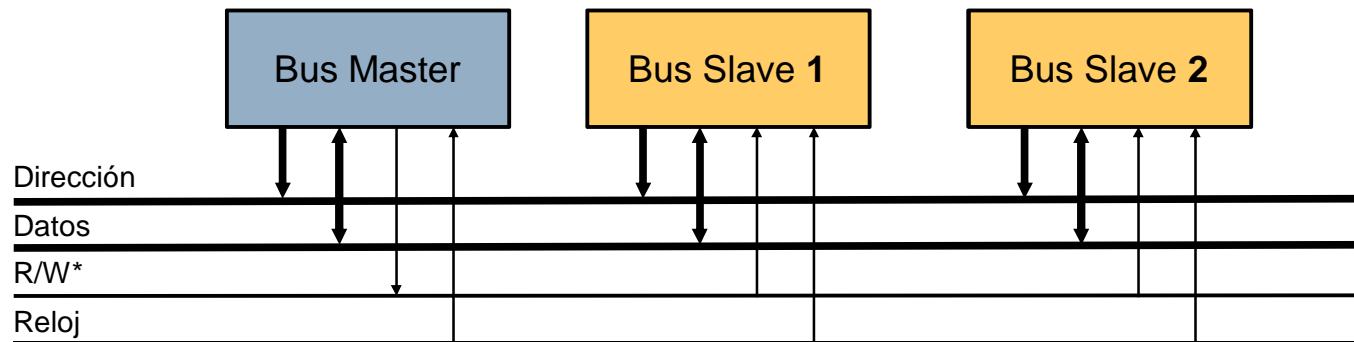
Protocolo de transferencia semi-síncrono

- Las transferencias se rigen por una **única señal de reloj**
- Cada transferencia puede ocupar uno o varios periodos de reloj
- Señal de WAIT: la activa el Slave si no es capaz de realizar la transferencia en el tiempo estipulado
 - **Dispositivos rápidos:** operan como en un bus síncrono
 - **Dispositivos lentos:** activan la señal de WAIT y congelan la actuación del Master
- *Ejemplos:* i80x86, MC68040, Bus PCI

Protocolo de transferencia semi-síncrono



Protocolo de transferencia semi-síncrono



Tiempo de decodificación: 30 ns

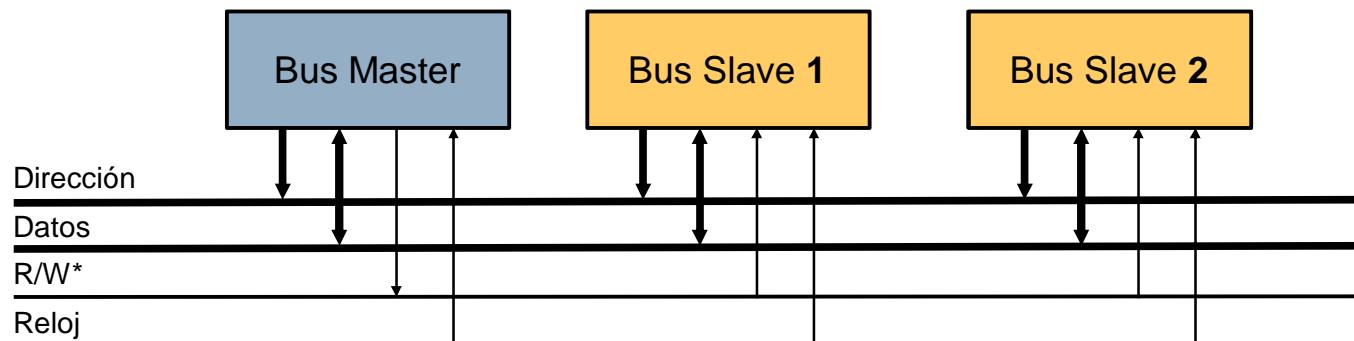
Tiempo de setup: 2 ns ; Skew: 2 ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Queremos que los dispositivos rápidos hagan una transferencia por ciclo:

Protocolo de transferencia semi-síncrono



Tiempo de decodificación: 30 ns

Tiempo de setup: 2 ns ; Skew: 2 ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Queremos que los dispositivos rápidos hagan una transferencia por ciclo:

Tiempo de ciclo: $30+2+2+100= 134$ ns

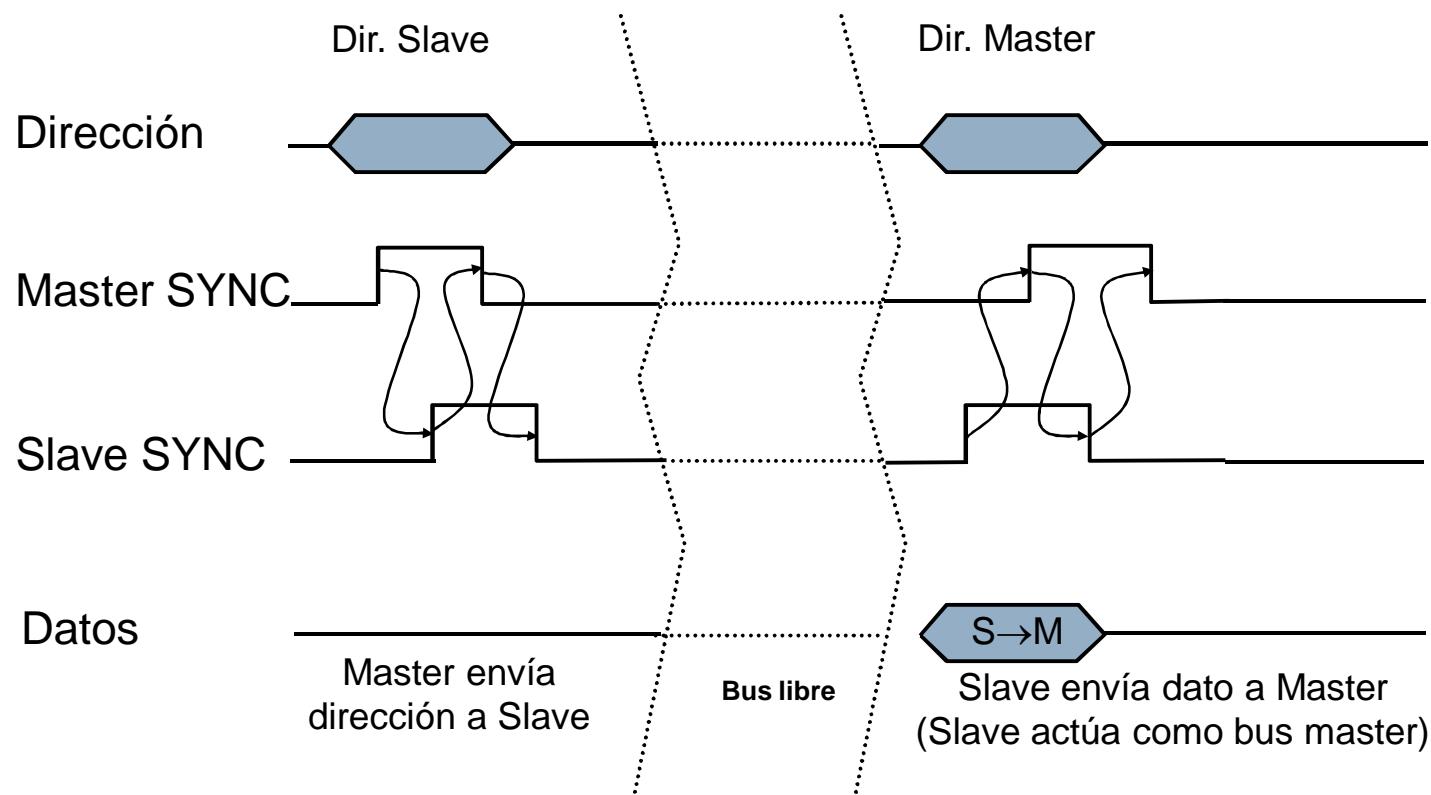
Tiempo de transferencia:

- slave1: 134 ns (1 ciclo)
- slave2: 134×8 ciclos = 1072 ns
- tiempo medio: **603 ns**

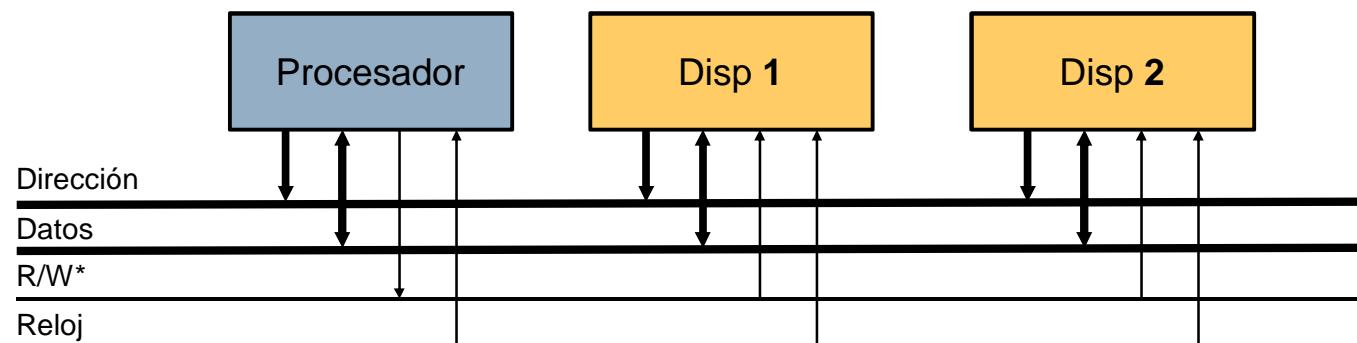
Transferencias de ciclo partido

- Mejora el rendimiento del bus en las operaciones de lectura
- El ciclo de lectura se divide en dos transacciones separadas
 - El Master envía al Slave la petición de lectura y deja el bus libre
 - Cuando el Slave dispone del dato solicitado, inicia un ciclo de bus y envía el dato al Master
(Slave actúa como master del bus)
- *Desventajas:*
 - Lógica más compleja: ambos dispositivos deben ser capaces de actuar como Master y como Slave
 - Necesidad de incluir un protocolo de arbitraje
- *Ejemplos:* VAX-11/780, iAPX-432

Transferencias de ciclo partido



Transferencia ciclo partido asíncrona



Tiempo de decodificación: 30 ns

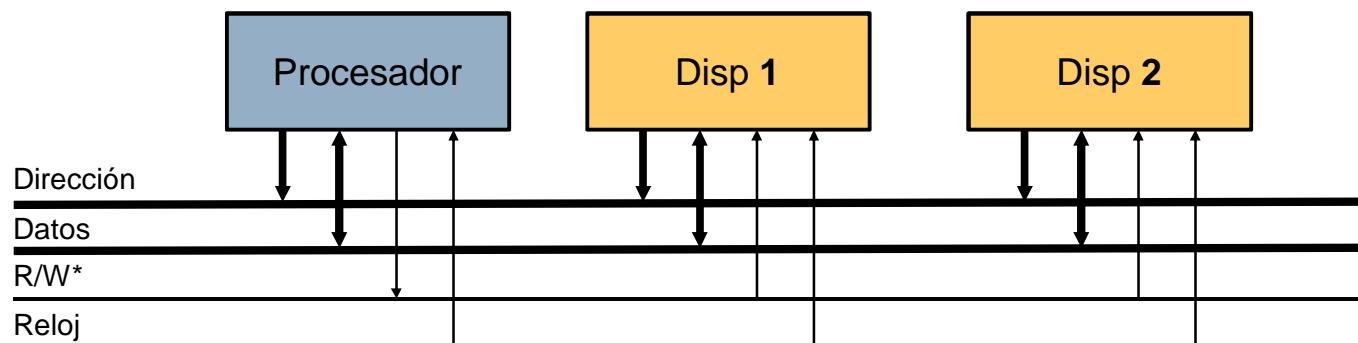
Tiempo de intercambio de señales de sincronización: 10ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Queremos que un dispositivo rápido pueda hacer una transferencia por ciclo:

Transferencia ciclo partido asíncrona



Transferencias de lectura:

- envío dirección:
 $30+10+10+10+10 = 70\text{ns}$
- envío dato:
 $30+10+10+10+10 = 70\text{ns}$
- ocupación del bus: **140ns**
- tiempo mínimo para la lectura:
 - disp1: 240 ns
 - disp2: 1140 ns
 - tiempo medio: **690 ns**

Tiempo de decodificación: 30 ns

Tiempo de intercambio de señales de sincronización: 10ns

Tiempo de búsqueda 1: 100 ns

Tiempo de búsqueda 2: 1000 ns

Queremos que un dispositivo rápido pueda hacer una transferencia por ciclo:

Protocolos de arbitraje

- **Función:**

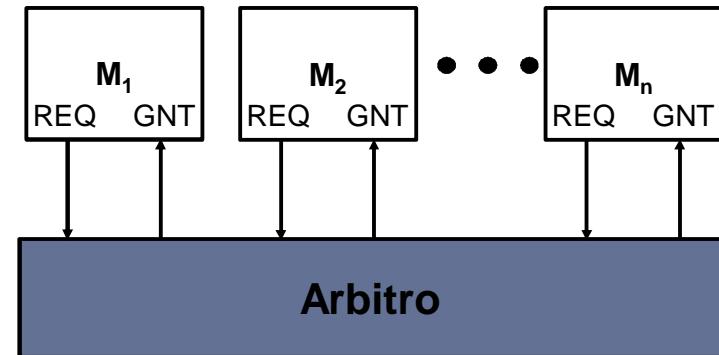
- Garantizar el acceso al bus libre de conflictos cuando existen varios masters alternativos
 - Procesador + Controladores DMA
 - Procesador + Procesador de E/S + Coprocesador matemático + ...
 - Sistema multiprocesador
 - Buses con protocolo de ciclo partido

- **Tipos** de protocolos de arbitraje

- *Centralizados*: existe un **árbitro** del bus o **master principal** que controla el acceso al bus
 - Protocolo en estrella
 - Protocolo daisy-chain
- *Distribuidos*: el control de acceso al bus se lleva a cabo entre todos los posibles masters de una forma cooperante
 - Protocolo de líneas de identificación
 - Protocolo de códigos de identificación

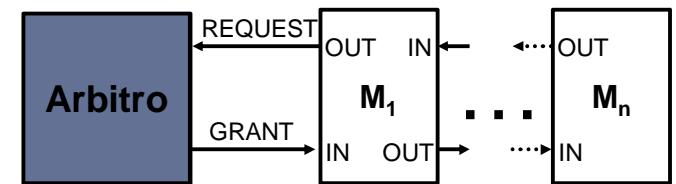
Protocolo de arbitraje en estrella

- Cada master se conecta al árbitro mediante dos líneas individuales:
 - BUS REQUEST (REQ): línea de petición del bus
 - BUS GRANT (GNT): línea de concesión del bus
- **Varias peticiones** de bus pendientes: el árbitro puede aplicar distintos algoritmos de decisión
 - FIFO
 - Prioridad fija o variable
- **Ventajas:**
 - Algoritmos de arbitraje simples
 - Pocos retardos de propagación de las señales (comparado con daisy-chain)
- **Desventajas:**
 - Número elevado de líneas de arbitraje en el bus (dos por cada posible master)
 - Número de masters alternativos limitado por el número de líneas de arbitraje
- **Ejemplo:** PCI

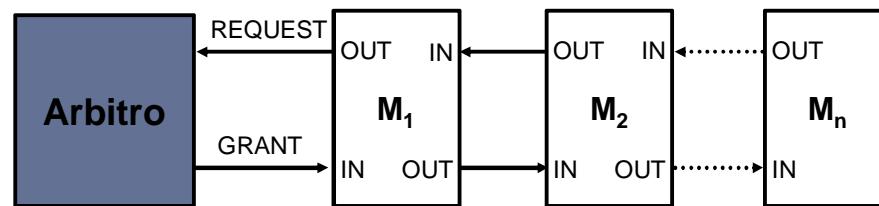


Protocolo de arbitraje daisy-chain

- Dos líneas de arbitraje comunes
 - BUS REQUEST: Línea de petición del bus
 - BUS GRANT: Línea de concesión del bus
- *Funcionamiento:*
 - El master que quiere tomar el control del bus activa REQUEST
 - Los restantes masters propagan REQUEST hasta el árbitro
 - El árbitro genera un pulso en la línea GRANT
 - Si un master detecta un flanco de subida GRANT y no pidió el bus, lo propaga al siguiente
 - Si un master detecta un **flanco de subida** en GRANT y tiene una petición pendiente, toma el control del bus
- *Prioridades:*
 - El orden en que se conectan los Masters a las líneas de arbitraje determina la prioridad de los mismos
- *Ejemplo:* bus IDE

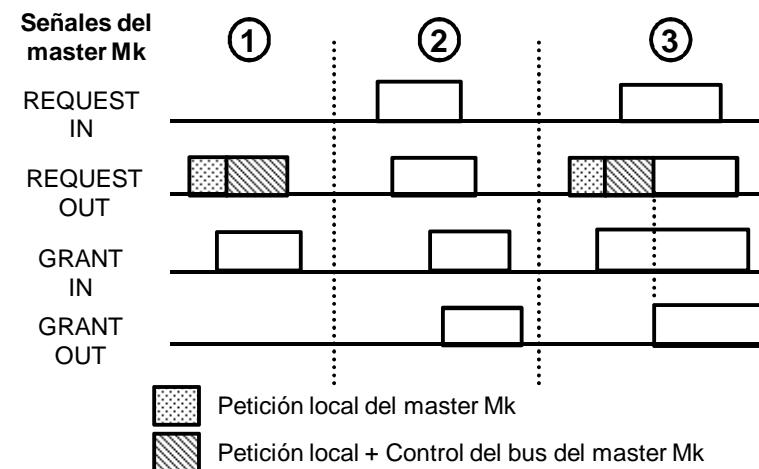


Protocolo de arbitraje daisy-chain



Situaciones típicas:

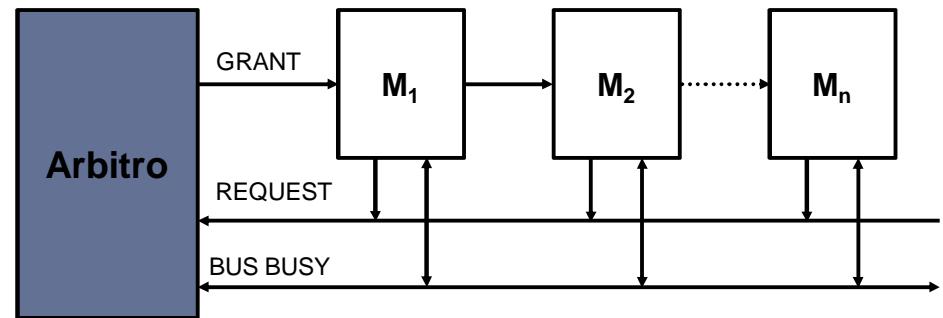
- 1) M_k pide el Bus
Cuando M_k recibe GRANT toma el control del bus
- 2) M_k recibe una petición de M_{k+1}
Cuando M_k recibe GRANT la propaga a M_{k+1}
- 3) M_k pide el bus
Cuando M_k recibe GRANT toma el control del bus
Mientras M_k controla el bus, recibe una petición de M_{k+1}
Cuando M_k termina de usar el bus, propaga GRANT a M_{k+1}



Protocolo daisy-chain con 3 hilos

- *Líneas de arbitraje:*

- **BUS REQUEST:** línea de petición de bus
- **BUS GRANT:** línea de concesión del bus
- **BUS BUSY:** línea de bus ocupado



- *Funcionamiento:*

- Cuando un master toma el control del bus activa BUS BUSY
- Un master solicita el bus activando REQUEST
- El árbitro activa GRANT cuando detecta REQUEST activado y BUS BUSY desactivado
- Si un master recibe GRANT y no ha pedido el bus, entonces propaga GRANT al siguiente
- Un master puede tomar el control del bus si se cumplen las tres condiciones siguientes:
 - a) El master tiene una petición local pendiente
 - b) La línea BUS BUSY está inactiva
 - c) El master recibe el **flanco de subida** de la señal GRANT

- *Ejemplo:* VME

protocolos de arbitraje

Situaciones típicas

1) M₁ solicita el bus. Bus libre

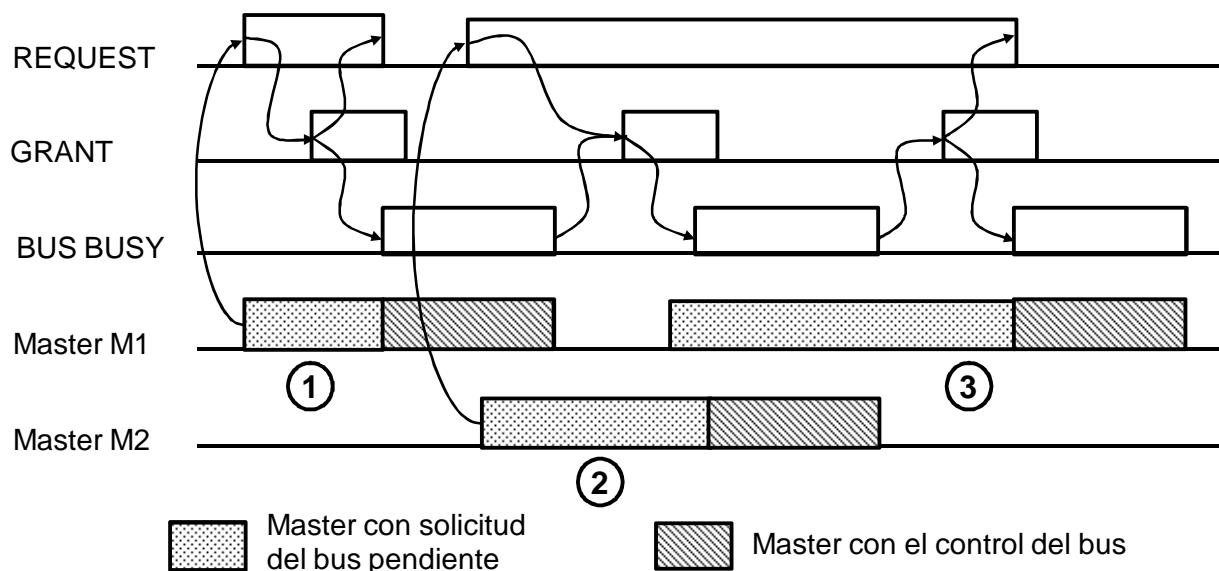
Cuando M₁ detecta el flanco de GRANT toma el control del bus

2) M₂ solicita el bus. Bus ocupado

M₂ debe esperar bus libre y detectar el flanco de GRANT

3) M₁ solicita el bus. Bus libre. GRANT activado

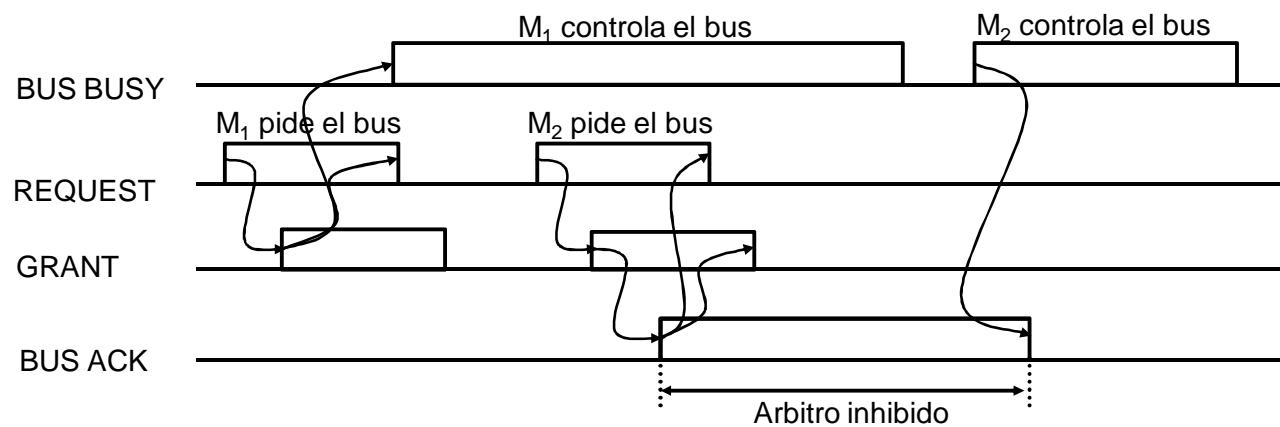
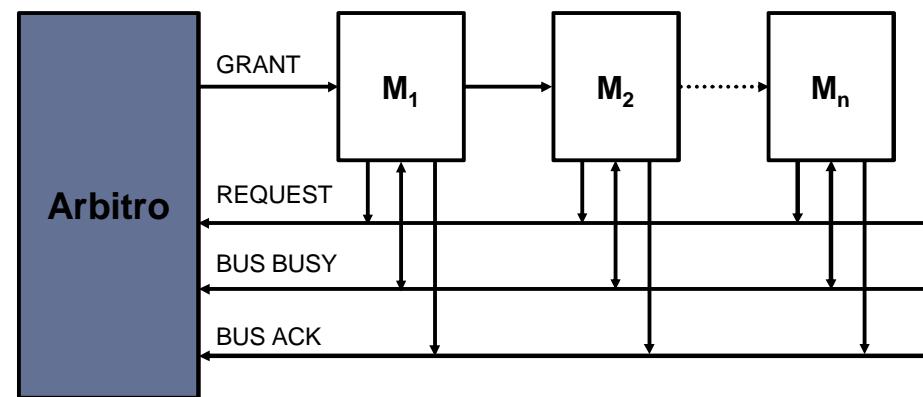
M₁ debe esperar que GRANT baje y vuelva a subir



Protocolo daisy-chain con 4 hilos

- Permite solapar la transferencia del ciclo actual con el arbitraje del ciclo siguiente
- *Líneas de arbitraje:*
 - **BUS REQUEST:** línea de petición de bus
 - **BUS GRANT:** línea de concesión del bus
 - **BUS BUSY:** línea de bus ocupado
 - **BUS ACK:** línea de confirmación
 - La activa el master que solicitó el bus en respuesta BUS GRANT, cuando el bus está ocupado
 - Cuando está activada el árbitro queda inhibido
- *Ejemplo:* Unibus (PDP-11), MC68000

Protocolo daisy-chain con 4 hilos



Protocolo de arbitraje distribuido: líneas de identificación

- *Funcionamiento:*
 - Cuando un master quiere tomar el control del bus activa su línea de identificación
 - Cada línea de identificación tiene asignada una prioridad:
 $\text{Prioridad}(\text{ID}_0) < \text{Prioridad}(\text{ID}_1) < \dots < \text{Prioridad}(\text{ID}_{n-1})$
 - Si varios masters activan simultáneamente sus líneas de identificación, gana el de mayor prioridad
 - Funcionamiento alternativo: las prioridades pueden ser variables
- *Desventajas:*
 - Número de dispositivos limitado por el número de líneas de arbitraje
- *Ejemplos:*
 - **Prioridad fija:** VAX SBI, SCSI
 - **Prioridad variable:** DEC 70000/10000 AXP, AlphaServer 8000

Protocolo de arbitraje distribuido: códigos de identificación

- *Funcionamiento:*
 - Cada master tiene un código de identificación de n bits (máximo 2^n masters)
 - Existen n líneas de arbitraje: $\text{ARB}_0, \text{ARB}_1, \dots, \text{ARB}_{n-1}$
 - Cuando un master quiere tomar el control del bus pone su código en las n líneas de arbitraje
 - Si varios masters compiten por el bus, gana el de mayor identificador
- Requiere **menos líneas de control** que con líneas de identificación pero **necesita lógica de decodificación** mayor coste y retardo
- *Ejemplos:* Multibus II, Futurebus+

Conclusiones

- Los buses permiten la comunicación entre los dispositivos
- El rendimiento de un bus depende de el ancho de banda de datos, el protocolo de transferencia, la longitud y el número de dispositivos conectados
- Los protocolos síncronos (o semi-síncronos) son más adecuados para buses pequeños que deban proporcionar alto rendimiento
- Los asíncronos son más adecuados para buses grandes con elementos heterogéneos conectados
- Si hay varios “masters” en un mismo bus, se debe incluir un mecanismo de arbitraje que evite las colisiones

Tema 13: Jerarquía de buses y buses estándares



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

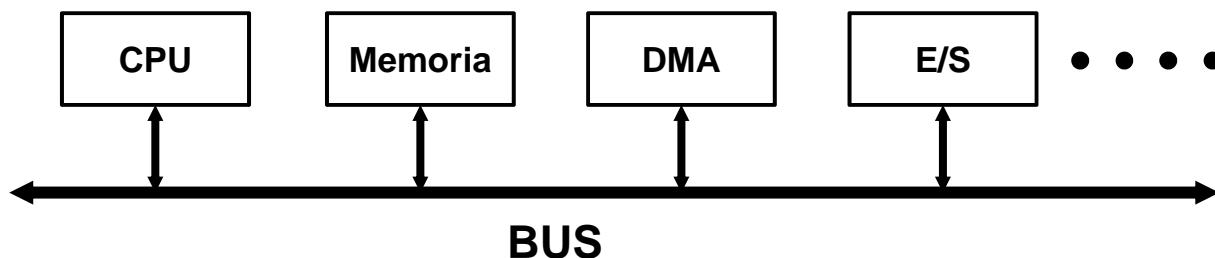
Jerarquía de buses y buses estándares

Jerarquía de buses y buses estándares

- 1. Limitaciones de los sistemas basados en un único bus**
- 2. Jerarquía de buses**
- 3. Jerarquía de buses en los PC**
- 4. Ejemplos de buses estándar**

Problemas de los sistemas basados en un único bus

- Conectar un gran número de dispositivos heterogéneos a un mismo bus provoca una *disminución del rendimiento global del sistema*
- Además, si los dispositivos han sido diseñados por compañías distintas a la que diseñó el bus pueden existir *incompatibilidades del bus con los dispositivos*

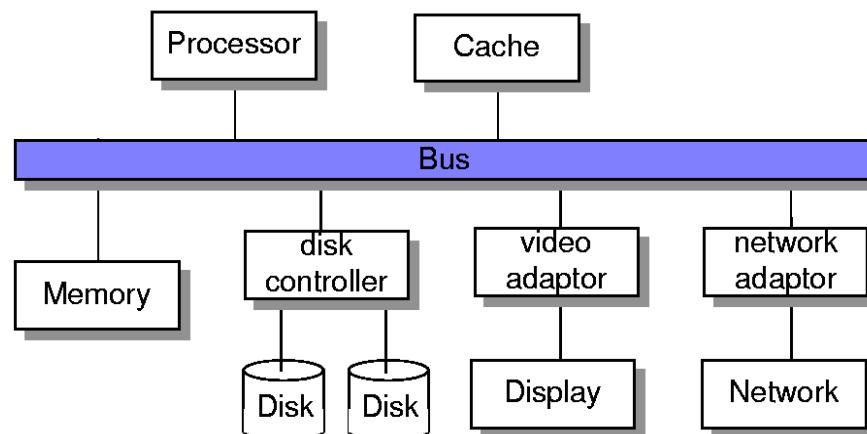


Problema 1: disminución del rendimiento

- Al conectar muchos dispositivos heterogéneos
 - Aumenta el ***retardo de propagación*** de las señales
 - El bus debe tener ***mayor longitud*** para soportar mayor número de dispositivos
 - Las ***señales de arbitraje*** (GRANT), si son encadenadas, debe propagarse a través de un mayor número de posibles masters
 - El bus puede actuar como un “***cuello de botella***”
 - Si la demanda de la transferencia es mayor que la capacidad del bus los dispositivos deberán esperar mucho tiempo para poder transmitir
 - La ***diferencia de velocidad de los dispositivos*** afecta negativamente al rendimiento global
 - En el mismo tiempo que un dispositivo lento realiza una transferencia, uno rápido podría haber realizado miles de transferencias

Problema 1: disminución del rendimiento

- Procesador a 200 MHz (tiempo ciclo = 5 ns.)
- Ciclo medio por instrucción: CPI = 2 ciclos
- El procesador se conecta a la cache y al resto de dispositivos a través de un único bus del sistema
- El disco tiene un tiempo de acceso de 10 ms y una velocidad de transferencia de 10 MB/seg:
 - ¿Cuánto se tarda en realizar una transferencia de 512 KB de disco a memoria?
 - ¿Cuántas instrucciones podrían haberse ejecutado?



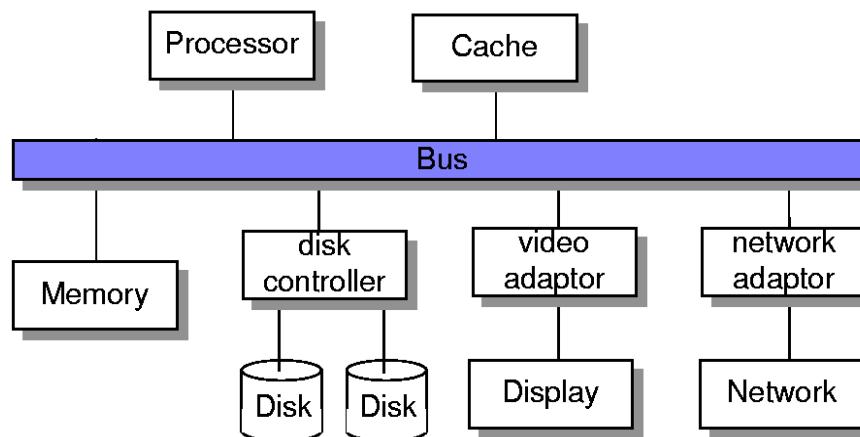
Problema 1: disminución del rendimiento

Queremos realizar una transferencia de 512 KB de disco a memoria

$$\text{Tiempo} = 10 \text{ ms} + \frac{512 \text{ KB}}{10.000 \text{ KB/s}} = 61,2 \text{ ms}$$

En ese tiempo, la CPU podría haber ejecutado:

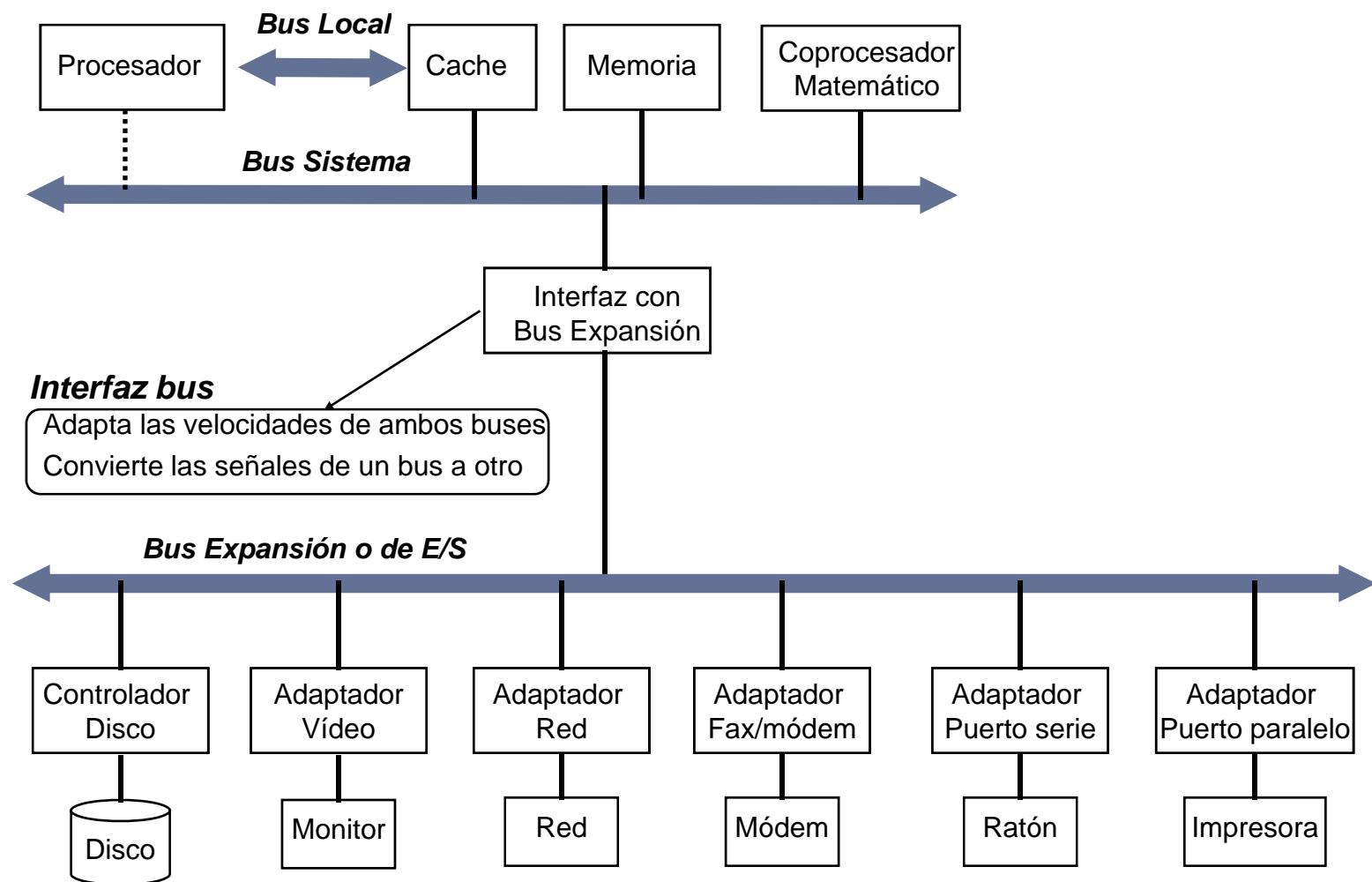
$$(0,0612 \text{ s}) \times (100 \times 10^6 \text{ instruc /s}) = 6,12 \text{ millones de instrucciones}$$



Problema 2: incompatibilidades

- Un computador está compuesto por muchos elementos que, frecuentemente, han sido desarrollados por empresas distintas
- El bus del sistema es clave para el rendimiento y en general está diseñado para un procesador concreto
- Los fabricantes de dispositivos de E/S no quieren diseñar un dispositivo distinto para cada procesador
- ***Solución ideal:*** que todos los computadores utilizasen un estándar de bus uniforme
- ***Problema:*** un cada fabricante diseña sus propios buses optimizados para sus arquitecturas, por lo que es muy difícil que todos se pongan de acuerdo

Jerarquía de buses: buses local, del sistema y de expansión



Jerarquía de buses

- Bus Local y Bus del Sistema
 - Buses rápidos, cortos
 - Buses Propietarios (no estándares)
 - Optimizados para la arquitectura
 - Nº fijo de dispositivos de prestaciones conocidas
- Bus de expansión
 - Buses más largos y lentos
 - Bus abierto (estándar)
 - Accesible por el usuario
 - Nº indeterminado de dispositivos de distintas prestaciones

Ventajas de la jerarquía de buses

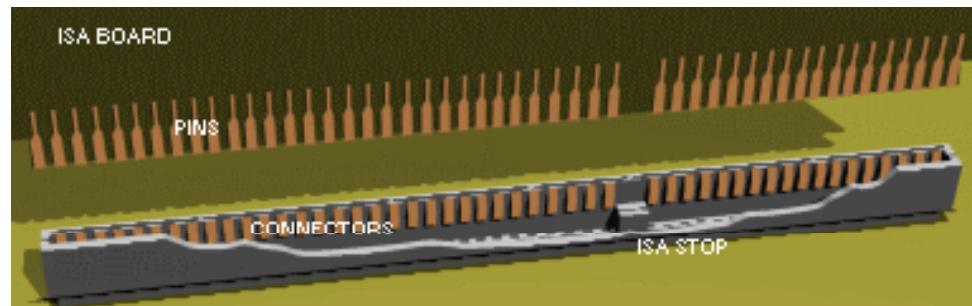
- El bus local entre el procesador y la cache aísla el tráfico de E/S del procesador
 - Se puede transferir información entre la memoria y la E/S sin interrumpir la actividad del procesador
- El bus de expansión reduce el tráfico en el bus del sistema
 - La transferencia entre cache y memoria principal se pueden realizar de forma más eficiente
- Se elimina el problema de la incompatibilidad
 - El bus local y del sistema suelen ser propietarios (no estándar) y están optimizados para cada arquitectura particular
 - Los buses de expansión son buses estándares o abiertos (ISA, PCI, PCI-Express, etc.)

Interfaces, o puentes entre buses

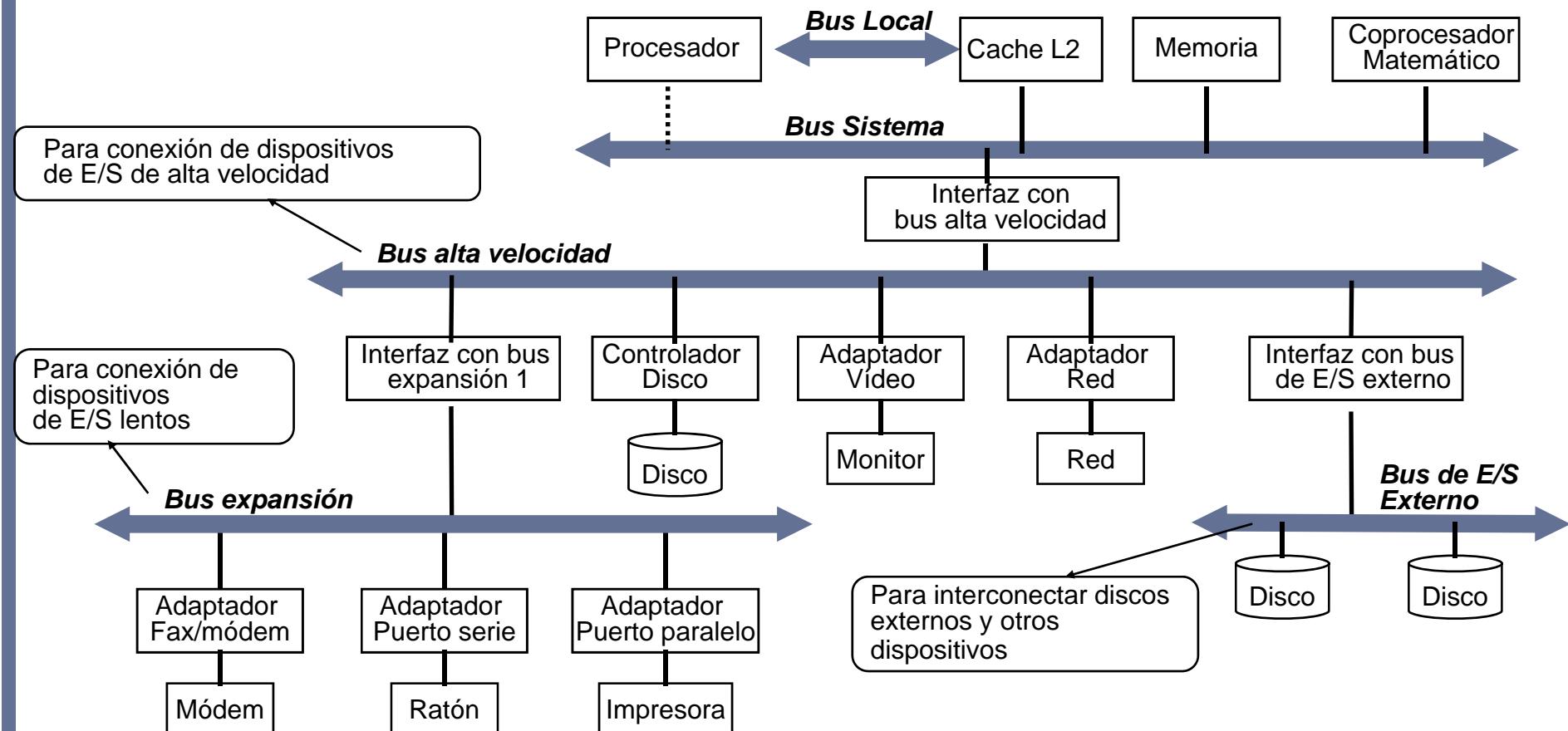
- Adaptar las velocidades de ambos buses
 - El bus del sistema es, en general, más rápido que el bus de expansión
 - El adaptador debe actuar como buffer de almacenamiento intermedio para evitar la pérdida de datos
 - Conversión de líneas del bus
 - Los buses pueden tener utilizar señales distintas para realizar funciones similares
 - **Ejemplos:**
 - 1) **Líneas de operación distintas**
Bus sistema: Una única línea RD/WR*
Bus expansión: Dos líneas READ - WRITE separadas
 - 2) **Líneas multiplexadas y dedicadas**
Bus sistema: líneas de dirección/datos multiplexadas (AD0, AD15, A16-A19)
Bus expansión: líneas de dirección y datos dedicadas (A0-A19, D0-D15)
 - 3) **Distinto número de líneas de datos**
Bus sistema: D0-D31
Bus expansión: D0-D15
 - ⇒ El adaptador debe dividir cada transferencia de 32 bits en dos transferencias de 16 bits
 - 4) **Distinto mecanismo de sincronización**
Bus sistema: síncrono
Bus expansión: asíncrono
 - ⇒ El adaptador deberá comunicarse de forma síncrona con el bus del sistema y de forma asíncrona con el bus de expansión
 - ⇒ El adaptador deberá ser capaz las señales de sincronización adecuadas dependiendo del bus con el que se comunique
- Etc.

Especificaciones de un bus estándar

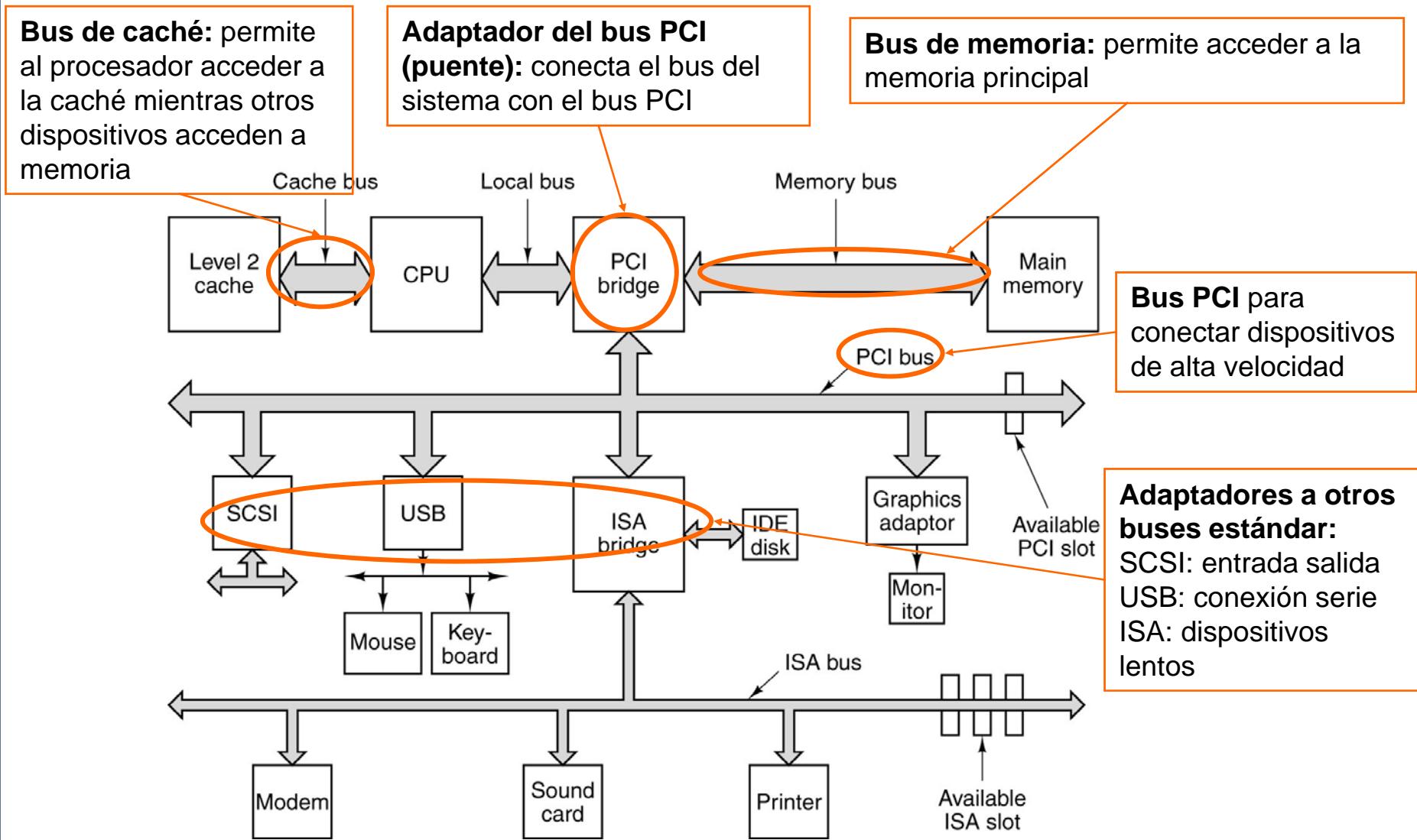
- Las especificaciones de un bus estándar deben estar perfectamente definidas y recogidas en un documento de estandarización
- En las especificaciones se distinguen varios niveles:
 - **Nivel eléctrico**
 - Valores de las tensiones de alimentación
 - Límites de valores eléctricos de las señales lógicas
 - P. ej. 1 lógico → de 0,2 V a 0,5 V;
 - 0 lógico → de -0,2 V a -0,5 V
 - **Nivel mecánico**
 - Forma y tamaño de los conectores
 - Número de contactos del conector
 - Número de dispositivos que soporta
 - **Nivel lógico**
 - Funciones a cada señal (bus de datos, bus de direcciones, bus de control)
 - Asignación de señales a los contactos del conector
 - **Nivel de temporización básico**
 - Protocolos de sincronización empleados
 - **Nivel de arbitraje**
 - Protocolos de arbitraje empleados



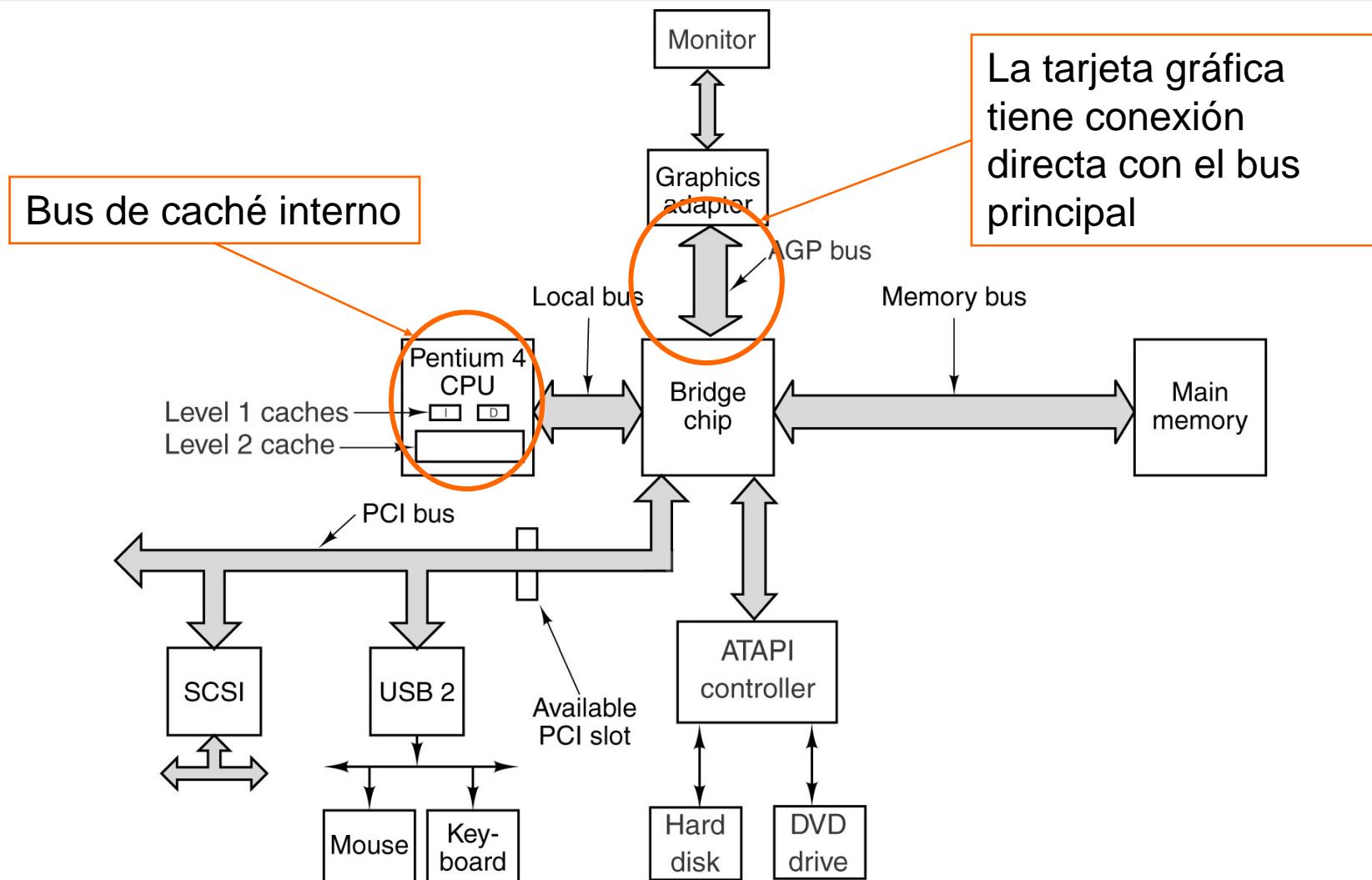
Jerarquía de buses de un PC



Ejemplo 1: jerarquía de buses en un PC pentium



Ejemplo 2: jerarquía de buses en un Pentium IV



Jerarquía de buses en un PC x86

Comparación de las prestaciones de los buses de un PC Pentium

Bus	Ancho datos	Frec. reloj	Velocidad transmisión	
Sistema	64	133 MHz	1.064 Gbytes/s	Buses de expansión de alta velocidad
PCI (V 2.0)	32	33 MHz	132 Mbytes/s	
PCI (V 2.1)	64	66 MHz	528 Mbytes/s	
EISA	32	8 MHz	32 Mbytes/s	Buses de expansión de baja velocidad
ISA	16	8 MHz	5-8 Mbytes/s	
SCSI-1	8	5 MHz	4 Mbytes/s	
SCSI-2	16/32	10 MHz	20/40 Mbytes/s	Buses de E/S Externos
IDE	16	1.6 MHz	3.18 Mbytes/s	
EIDE	16	5.5 MHz	10.6 Mbytes/s	
USB 2	Serie	--	60 Mbits/s	“Buses” para conexión de discos
				Bus serie

Se consigue mejorar el ancho de banda aumentando el número de líneas de datos y la frecuencia de reloj

Jerarquía de buses en un PC Pentium

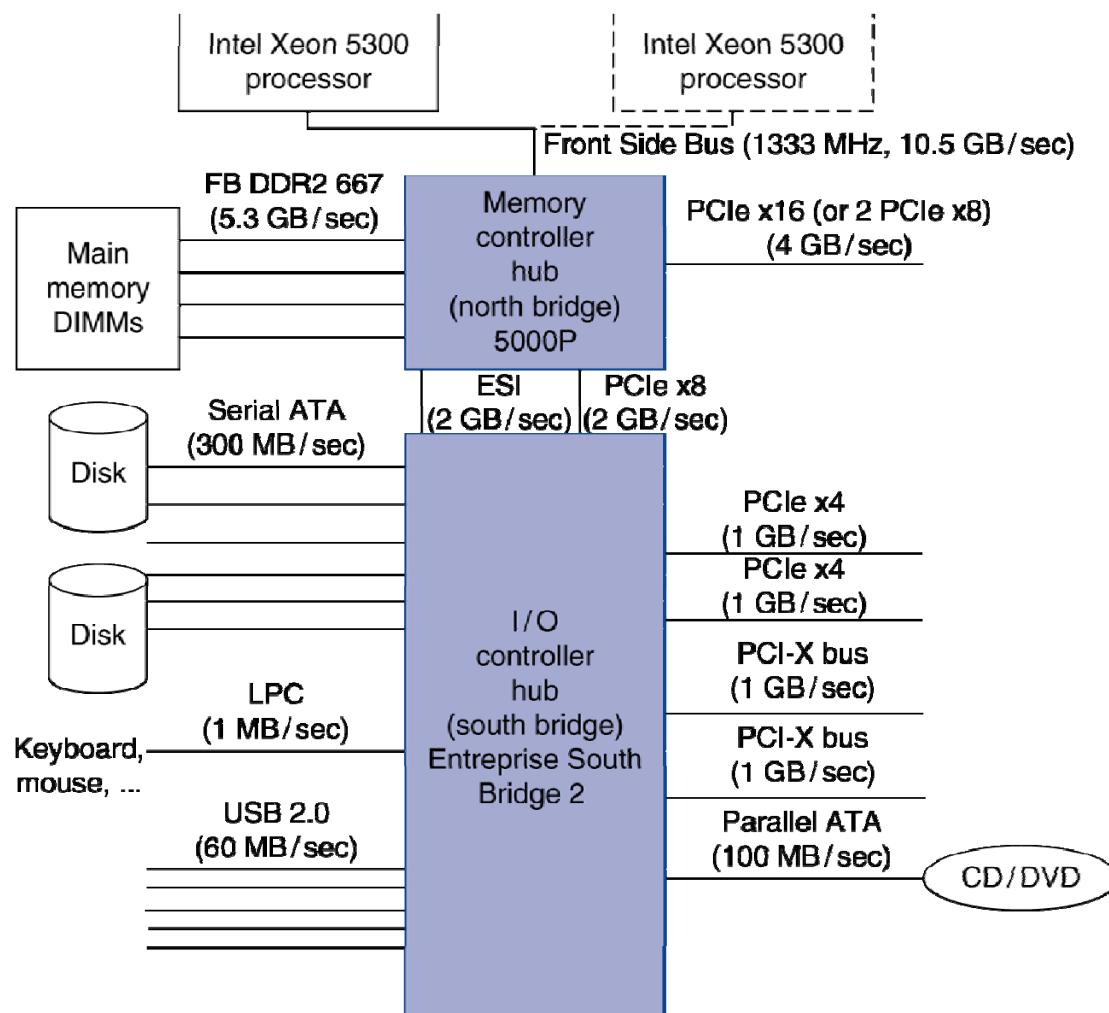
Buses más recientes

Bus	Año	Líneas de datos	Freq. reloj	Ancho de banda	Uso
USB 2.0	2000	Serie	---	480Mbit/s	periféricos
AGP	1996	32	66 MHz	264 MB/s	Tarjetas gráficas
AGP 8X	2002	32	533 MHz	2GB/s	Tarjetas gráficas
PCI-X	1998	64	133 MHz	1GB/s	Dispositivos rápidos
PCI-Express	2004	De 1 a 16 conexiones serie dobles	---	250MB/ conexión	Dispositivos rápidos, tarjetas gráficas

Tendencia:

- Buses on-chip síncronos
- Conexiones serie de alta velocidad para conectar dispositivos rápidos
 - Razón: *tiempo de desplazamiento relativo (skew time)*

Typical x86 PC I/O System



Ejemplo de bus estándar: bus PCI

Bus PCI (Peripheral Component Interconnect Bus, 1993)

- Bus de expansión diseñado para el i80486 y Pentium
 - **Bus de datos:**
 - Versión 2.0: 32 bits de datos
 - Versión 2.1: 64 bits de datos
 - **Bus de direcciones:** 32 bits (4 GB direccionables)
 - **Ciclo de reloj:**
 - Versión 2.0: 33 MHz
 - Versión 2.1: 66 MHz
 - **Velocidad de transferencia máxima:**
 - Versión 2.0: 132 Mbytes/s
 - Versión 2.1: 528 Mbytes/s
 - **Protocolo de bus:** semisíncrono
 - **Protocolo de arbitraje:** centralizado en estrella
 - **Otras características**
 - Hasta 16 slots de expansión
 - Soporte para gran variedad de controladores de dispositivos de E/S de alta velocidad
 - Vídeo, Sonido, Redes alta velocidad, Adaptadores SCSI, etc.
 - Soporte Plug-and-Play (conecta y listo)
 - Tarjetas controladoras autoconfigurables (línea de interrupción, dirección de E/S, etc.)

Ejemplo de bus estándar: bus PCI

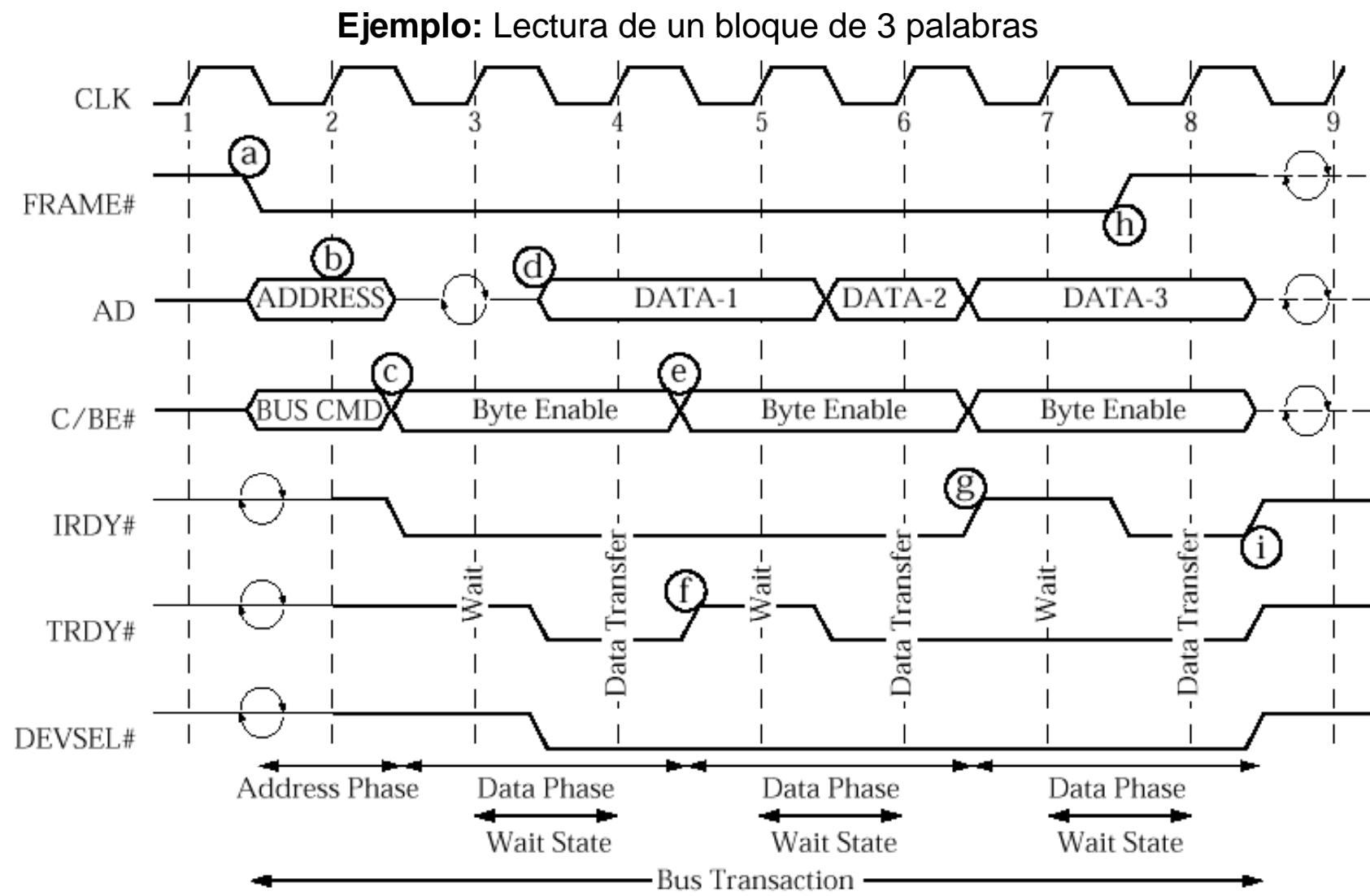
Protocolo de sincronización del bus PCI

- Protocolo semisíncrono
- Modos de transferencias
 - **Modo palabra**
 - Se transmite una única palabra a una dirección de memoria o E/S específica
 - Las palabras pueden ser de 1, 2, 3 ó 4 bytes
 - **Modo bloque**
 - Se transfiere un bloque de datos desde/hacia posiciones de memoria consecutivas, a partir de una posición inicial
- Lineas del bus
 - **CLK:** señal de reloj
 - **AD0-AD31:** Líneas multiplexadas de datos y direcciones
 - **C0*-C3*/BE0*-BE3*:** Líneas multiplexadas de *orden y byte activo (byte enabled)*
 - **Orden (C0*-C3*):** la activa el master durante el primer ciclo de la transferencia para especificar el tipo de transferencia a realizar:
 - Lectura de memoria, escritura de memoria, lectura de E/S, escritura de E/S, etc.
 - **Byte activo (BE0*-BE3*):** la activa el master durante la transferencia de datos para indicar qué líneas del bus transportan los datos
 - BE0* activada ⇒ AD0-AD7 transporta datos
 - BE1* activada ⇒ AD8-AD15 transporta datos
 - BE2* activada ⇒ AD16-AD23 transporta datos
 - BE3* activada ⇒ AD24-AD31 transporta datos

Ejemplo de bus estándar: bus PCI

- Líneas del bus (cont.)
 - **FRAME***: Señal para indicar el comienzo y la duración de una transferencia
 - La activa el master al poner la dirección en el bus para indicar el comienzo de la transferencia
 - Si la transferencia es *modo bloque* la señal se mantiene activa durante toda la transferencia del bloque y se desactiva al transferir la última palabra
 - **DEVSEL***: Señal de *dispositivo seleccionado* (device selected)
 - La activa el slave para indicar que ha reconocido su dirección
 - **TRDY***: Señal de *slave preparado* (*target ready*)
 - La activa el slave al inicio de la transferencia junto con DEVSEL*
 - El slave desactiva esta señal en caso de que no pueda completar la transferencia en un solo ciclo de reloj
 - **IRDY***: Señal de *master preparado* (*initiator ready*)
 - La activa el master al inicio de la transferencia
 - El master desactiva esta señal en caso de que no pueda completar la transferencia en un solo ciclo de reloj, por ejemplo en caso de que el master se quede temporalmente sin capacidad de almacenamiento

Ejemplo de bus estándar: bus PCI



Ejemplo de bus estándar: bus PCI

a El master realiza las siguientes acciones

- Pone la dirección en el bus (AD0-AD31)
- Indica el tipo de operación a realizar (C0*-C3*)
- Activa FRAME* para indicar el inicio de la transferencia

b El slave descodifica y reconoce su dirección en el bus

c El master deja libre el bus de datos e indica en BE0*-BE3* qué líneas transportarán los datos y activa IRDY* para indicar que está preparado para recibir el 1^{er} dato

d Cuando el slave tiene el 1^{er} dato válido realiza las siguientes acciones

- Activa DEVSEL* para indicar ha reconocido su dirección
- Pone el dato en el bus y activa TRDY* para indicar que el dato está en el bus

e El master lee el dato

- A partir de aquí, mientras esté la señal FRAME* activada, se leerá un dato en cada ciclo de reloj (siempre que el slave no desactive TRDY*)

f El slave necesita más de 1 ciclo para poner el 2º dato en el bus

- Desactiva TRDY* hasta que tiene el nuevo dato preparado

g El master no está preparado para recibir el 3^{er} dato

- Desactiva IRDY* hasta que está preparado para poder recibir correctamente el siguiente dato

h Transferencia del último dato

- El master desactiva FRAME* para indicar el final de la transferencia del bloque

i El master desactiva IRDY* y el slave desactiva TRDY* y DEVSEL*

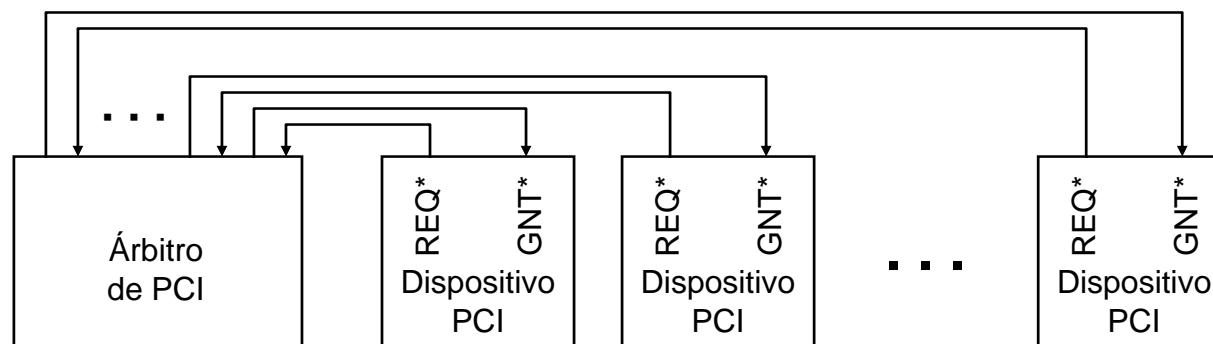
- El bus queda libre para la siguiente transferencia

Ejemplo de bus estándar: bus PCI

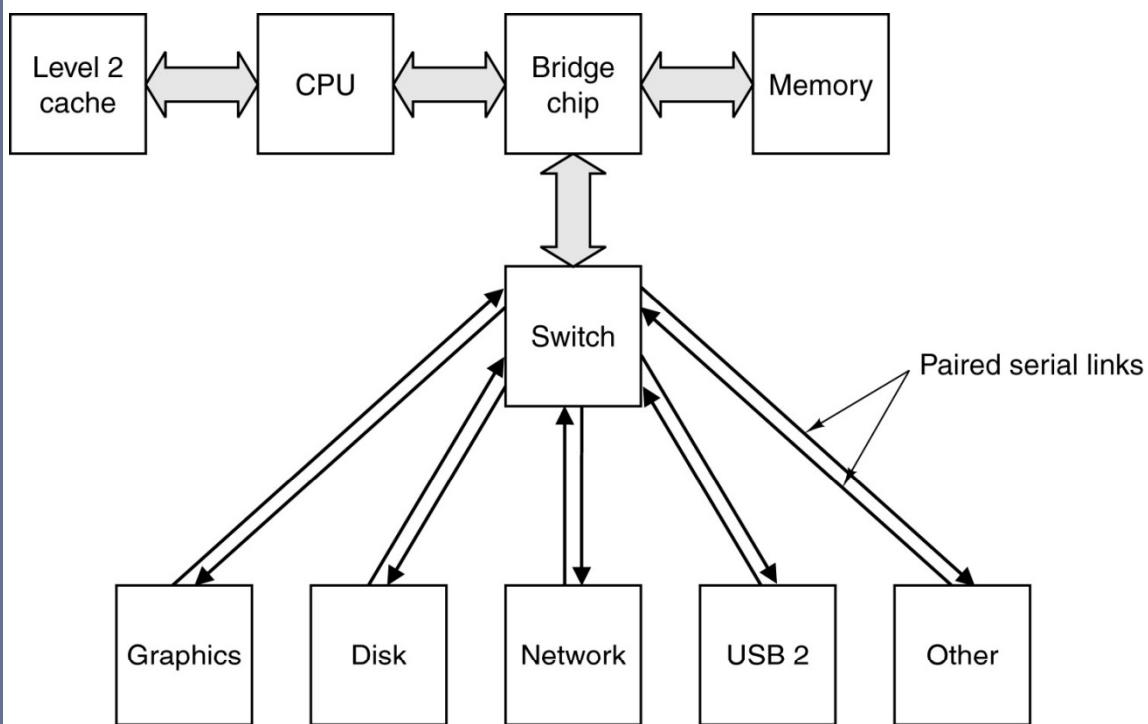
Protocolo de arbitraje del bus PCI

- Protocolo centralizado en estrella
 - Cada máster se conecta al árbitro mediante 2 líneas dedicadas
 - REQ: petición del bus
 - GNT: concesión del bus
 - La especificación de PCI no indica un algoritmo de arbitraje particular
 - Pueden utilizarse distintos tipos de algoritmos
 - FIFO
 - Prioridad fija
 - Prioridad variable
 - Rotativo
 - etc.

Líneas de arbitraje del bus PCI



PCI Express



- Conexiones serie dedicadas
- Cada conexión proporciona 250 Mbs para entrada de datos y otros tantos para salida
- Se pueden utilizar hasta 16 conexiones para un mismo dispositivo

Conclusiones

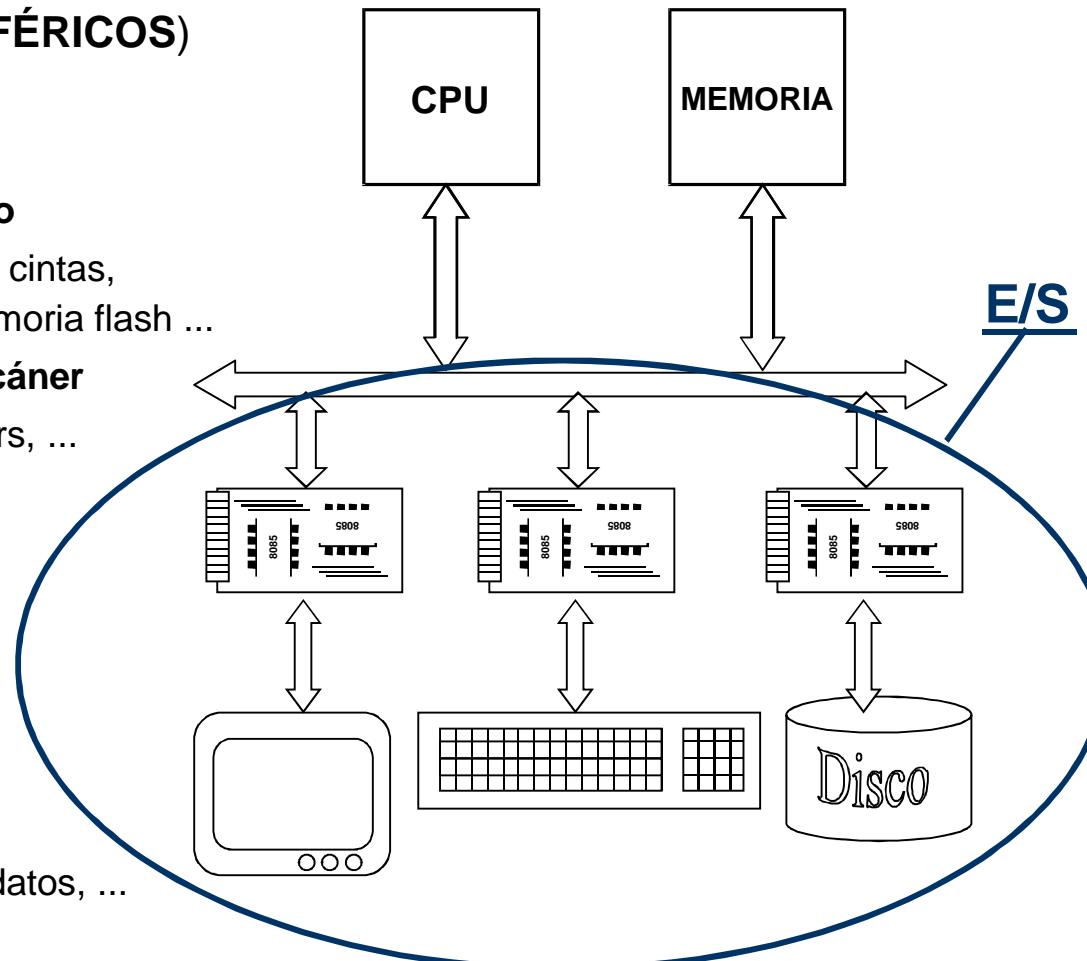
- Los sistemas basados en un único bus no funcionan bien cuando se deben conectar muchos dispositivos heterogéneos
- Incluyendo una jerarquía de buses se puede trabajar con muchos dispositivos de una forma eficiente
- Los dispositivos lentos se conectan a buses de expansión que a su vez pueden comunicarse con el procesador atravesando uno o más interfaces
- Con este esquema todos los dispositivos lentos tienen asignado el mismo ancho de banda que un dispositivo rápido
- Además los buses de expansión pueden ser estándar, con lo que cualquier compañía puede desarrollar componentes para estos buses
- Por otro lado los buses de caché y de sistema pueden ser propietario y estar optimizados para un determinado procesador

Organización del sistema de entrada/salida

1. Organización del subsistema de E/S
2. Mecanismos básicos de E/S
3. Gestión de interrupciones
4. DMA

La E/S permite al computador interactuar con el “mundo exterior”

- Dispositivos típicos de E/S (**PERIFÉRICOS**)
 - **Dispositivos de E/S básica**
 - teclado, ratón, pantalla
 - **Dispositivos de almacenamiento**
 - discos, disquetes, CD-ROM, cintas, discos magneto-ópticos, memoria flash ...
 - **Dispositivos de impresión y escáner**
 - impresoras, plotters, scanners, ...
 - **Dispositivos de comunicación**
 - redes, módems, ...
 - **Dispositivos multimedia**
 - audio, video, ...
 - **Dispositivos de automatización y control**
 - sensores, alarmas, sistemas de adquisición de datos, ...

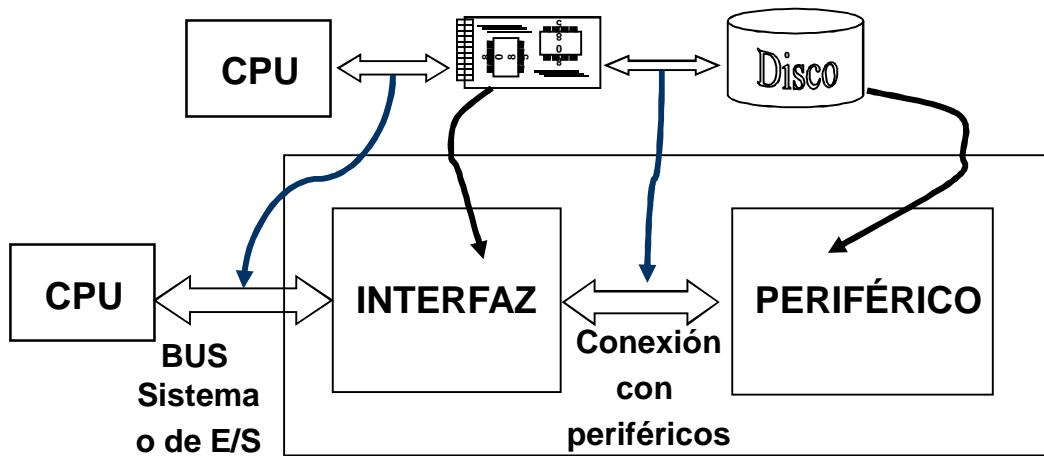


Funciones básicas del subsistema de E/S

- **Direccionamiento**
 - Necesario seleccionar el dispositivo de E/S con el que se realiza la transferencia
- **Transferencia de datos** entre el computador y el periférico
 - Tipos de transferencia
 - Lectura: computador ← periférico
 - Escritura: computador → periférico
 - Puede requerir ciertas conversiones de formato de los datos
 - Conversión de niveles eléctricos
 - TTL: 1 → V > 2,0 Volt; 0 → V < 0,8 Volt
 - RS-232-C: 1 → V < -3,0 Volt; 0 → V > +3,0 Volt
 - Conversión del tipo de codificación
 - Caracteres (ASCII, EBCDIC)
 - Enteros (magnitud y signo, C'1, C'2, ...)
 - Reales (punto fijo punto flotante, simple precisión, doble precisión, ...)
 - Conversión serie-paralelo / paralelo-serie
 - Conversión digital-analógico / analógico-digital
- **Sincronización y control de la transferencia**
 - Necesario un mecanismo de sincronización de la transferencia
 - El computador debe conocer
 - Si el periférico está preparado para enviar o recibir datos
 - Si el periférico ha terminado de realizar una transferencia y puede iniciar una nueva
 - No confundir con la sincronización elemental a nivel de transferencias de palabras a través del bus

El interfaz de E/S

- Conecta a los dispositivos con el computador
 - Interfaz = Controlador = Adaptador = Tarjeta de E/S



Ejemplo

Ordenes CPU → Interfaz

Leer N bytes a partir de
Superficie S
Cilindro C
Sector T

Ordenes Interfaz → periférico

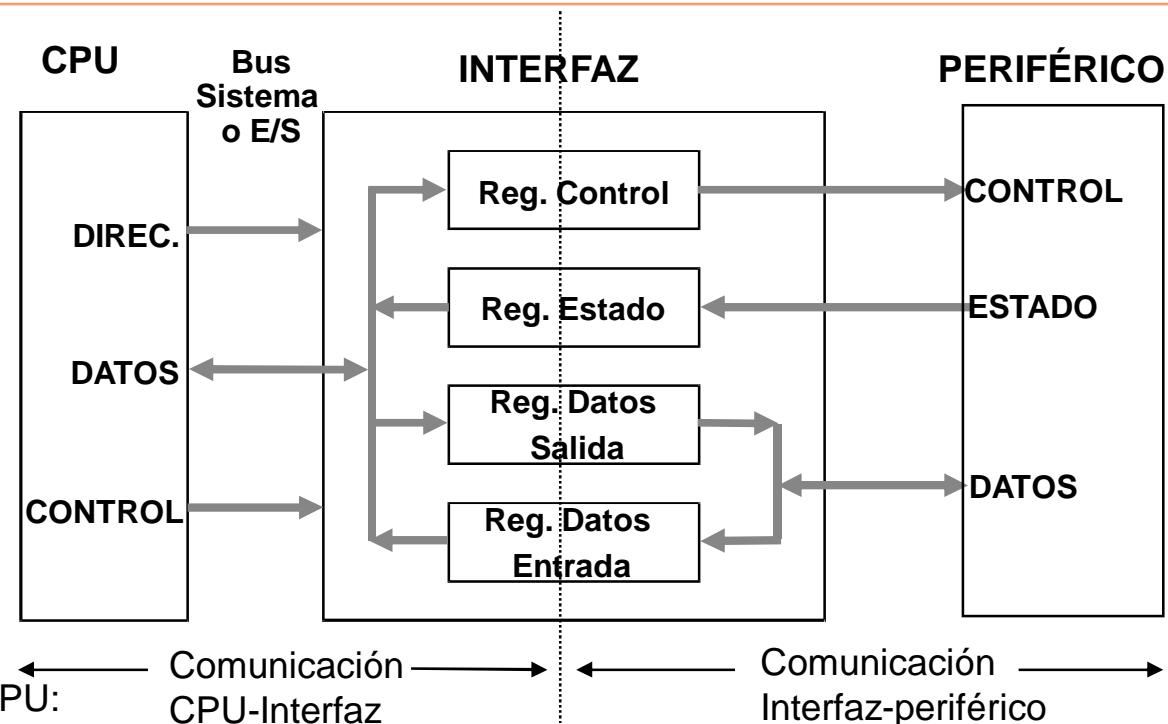
Posicionar cabezales en cilindro C
Posicionar cabezales en sector T
Seleccionar cabezal de superficie S
Leer N bytes
Retirar cabezales

Funciones del interfaz de E/S

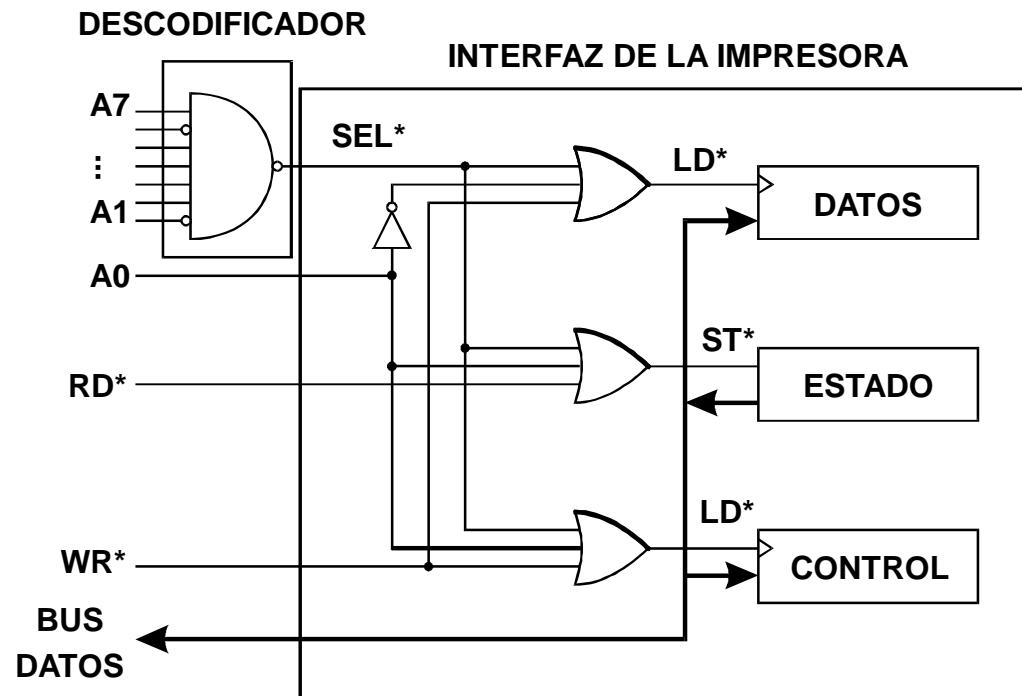
- Interpretar las órdenes que recibe de la CPU y transmitirlas al periférico
- Controlar la transferencia de datos entre la CPU y el periférico
 - Conversión de formatos
 - Adaptar la diferencia de velocidades entre CPU y periférico (mediante buffers de almacenamiento)
- Informar a la CPU del estado del periférico

Estructura del interfaz de E/S

- La comunicación se realiza a través de los registros del interfaz:
- Registro de datos de salida**
 - Almacena los datos enviados al periférico
- Registro de datos de entrada**
 - Almacena los datos enviados por el periférico
- Registro de estado**
 - Proporciona información a la CPU:
 - Dispositivo preparado/no preparado
 - Reg. datos lleno/vacío
 - Transferencia finalizada/no finalizada, etc.
- Registro de control**
 - Almacena las órdenes de la CPU
 - Leer/escribir N bytes en cilindro C, pista P, sector S (para discos)
 - Rebobinar / avanzar / leer N bytes (para cintas)
 - Imprimir carácter / saltar de línea / saltar de página (para impresoras), etc.



Ejemplo de conexión de un interfaz de E/S al bus



- Enviar un carácter almacenado en el registro R1 a la impresora
 - MOVE R1, \$BD
 - OUT R1, \$BD
- Enviar una orden almacenada en el registro R2 a la impresora
 - MOVE R2, \$BC
 - OUT R2,\$BC
- Leer estado de la impresora y almacenarlo en el registro R3
 - MOVE \$BC, R3
 - IN \$BC, R3

Dirección Registro Estado/Control: 10111100 (\$BC)

Dirección Registro Datos: 10111101 (\$BD)

Nota: los registros del interfaz de E/S también se llaman *puertos de E/S*

Ejemplo Placa ARM: temporizadores

- Son contadores descendentes con cierta lógica adicional
- Incluyen **5 registros de control** mapeados en memoria que permiten al procesador:
 - Configurar la velocidad a la que cuentan
 - Especificar dónde comienza la cuenta
 - Si se utilizan para generar una onda cuadrada definir cuando deben generar salida 0 y cuando 1
 - Indicar si deben contar una vez o repetir la cuenta indefinidamente
 - Indicar si deben avisar al procesador cada vez que lleguen a cero
 - Parar la cuenta
 - Reiniciar la cuenta
 - Indicar si deben avisar al DMA
- **Un registro de salida** que permite ver el valor actual de la cuenta

Alternativas de diseño: E/S aislada o localizada en memoria

E/S aislada

- **La E/S y la memoria utilizan un espacio de direcciones distinto**
 - El conjunto de direcciones de que utiliza la memoria y el que utiliza la E/S son independientes
- **Existen instrucciones específicas de E/S**
 - IN dir_E/S, Ri (CPU ← Periférico)
 - OUT Ri, dir_E/S (Periférico ← CPU)
- **El bus dispone de líneas de control específicas** (MEM/IO*) para indicar si se trata de una operación con memoria o una operación de E/S
 - Si MEM/IO* = 1
 - Operación con memoria (MOVE, LOAD, STORE) ⇒ La dirección del bus corresponde a una posición de memoria
 - Si MEM/IO* = 0
 - Operación de E/S (IN, OUT) ⇒ La dirección del bus corresponde a un puerto de E/S
- **Ejemplos**
 - i8086 y demás computadores de la familia intel x86

E/S localizada en memoria

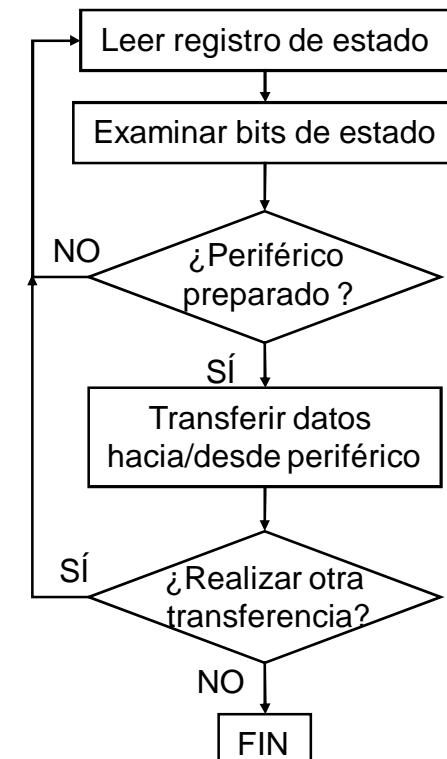
- ***La E/S y la memoria comparten el mismo espacio de direcciones***
- ***No se requieren instrucciones específicas de E/S***
 - Las mismas instrucciones que se utilizan para movimiento de datos con memoria (MOVE) pueden utilizarse para realizar operaciones de E/S
 - MOVE dir_E/S, Ri (CPU ← Periférico)
 - MOVE Ri, dir_E/S (Periférico ← CPU)
- ***En el bus no existe una línea especial*** para distinguir operaciones con memoria de operaciones de E/S
 - Un puerto de E/S no puede tener asignada la misma dirección que una posición de memoria válida
 - Normalmente se asigna a los dispositivos de E/S una **porción contigua del espacio de direcciones** que no se utiliza para la memoria
- ***Ventajas de la E/S localizada en memoria***
 - Es más flexible que la E/S aislada ya que permite realizar distintos tipos de operaciones sobre los puertos de E/S (aritméticas, lógicas, manipulación de bits, etc.) y no sólo de movimiento de datos
- ***Ejemplo***
 - ARM

Sincronización de la E/S

- Cuando la CPU quiere enviar/recibir datos a/desde un periférico **tiene asegurarse de que el dispositivo está preparado para realizar la transferencia**
- Existen dos mecanismos básicos de sincronización de la E/S
 - E/S programada con espera de respuesta
 - E/S por interrupciones

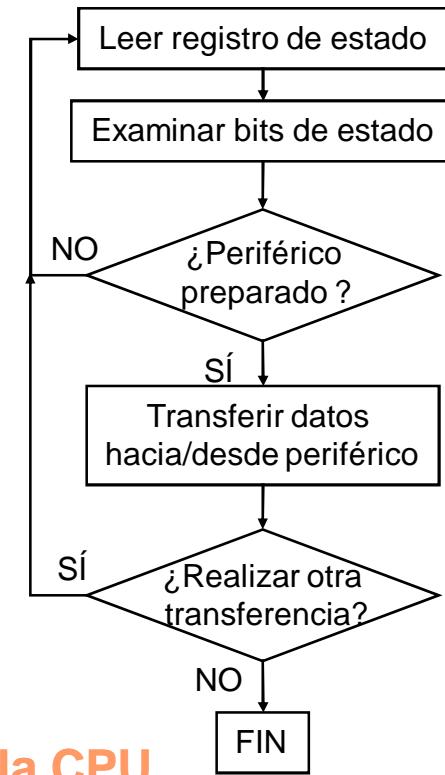
E/S programada con espera de respuesta

- Cada vez que la CPU quiere realizar una transferencia entra en un bucle hasta que éste está preparado para realizar la transferencia
- **Problema: ¡Es muy ineficiente!**



Problemas de la E/S con espera de respuesta

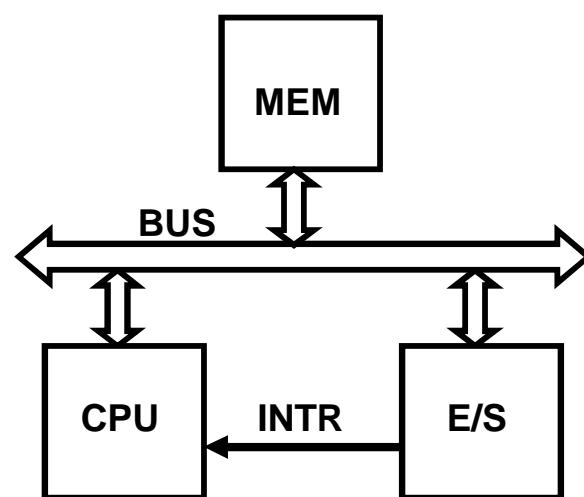
- La CPU no hace trabajo útil durante el bucle de espera
 - Con dispositivos lentos el bucle podría repetirse miles/millones de veces
- La dinámica del programa se detiene durante la operación de E/S
 - Ejemplo: en un vídeo-juego no se puede detener la dinámica del juego a espera que el usuario puse una tecla o mueva el *jostick*
- Dificultades para atender a varios periféricos
 - Mientras se espera a que un periférico esté listo para transmitir, no se puede atender a otro



Solución: Los periféricos deben poder comunicarse con la CPU

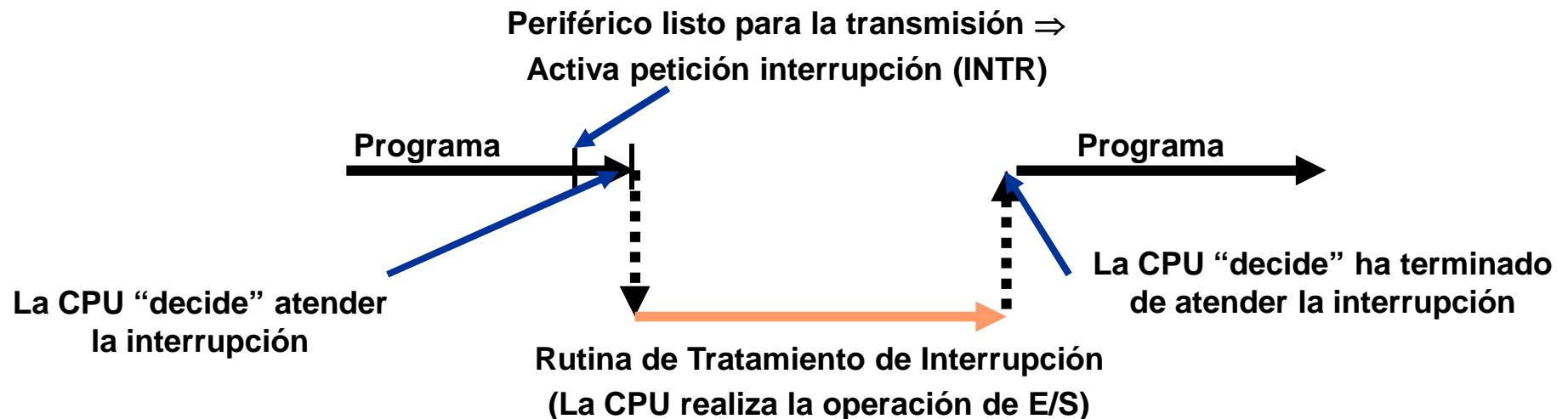
E/S por interrupciones

- No existe bucle de espera
- Cuándo la CPU da una orden puede realizar otras tareas mientras espera
- Cuando un periférico está listo para transmitir se lo indica a la CPU activando una línea especial del bus de control denominada **LÍNEA DE PETICIÓN INTERRUPCIÓN**
 - La CPU decide qué periféricos tienen capacidad para interrumpir



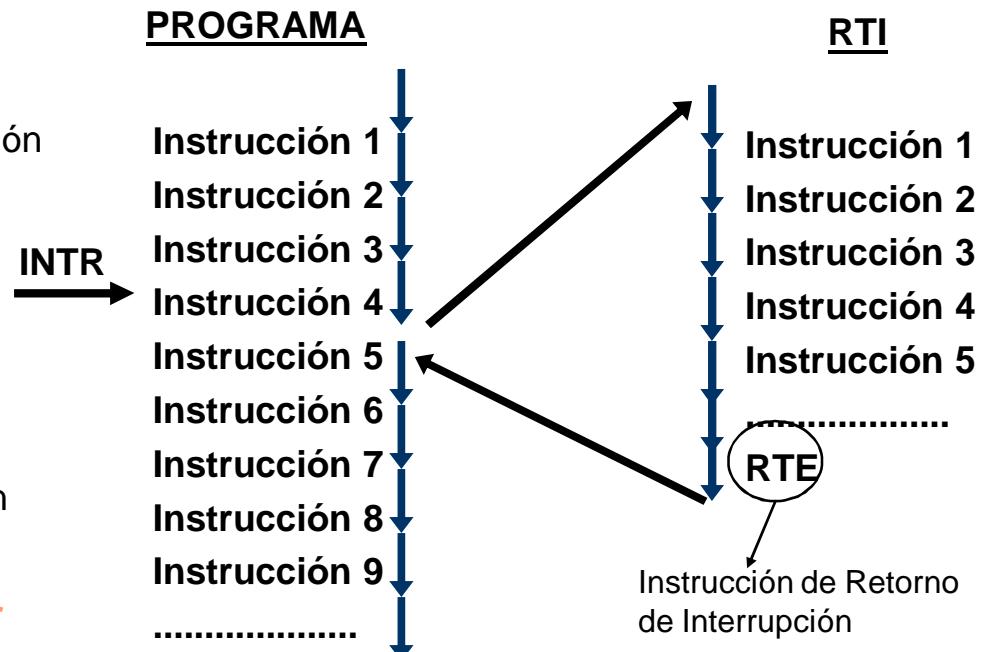
E/S por interrupciones

- Cuándo la CPU recibe una interrupción puede elegir si quiere atenderla o no en ese instante
- Para atender una interrupción la CPU ejecuta su **rutina de tratamiento de interrupciones** (RTI)
 - Cuando la CPU recibe una señal de petición de interrupción salta a ejecutar una **RTI**
 - La RTI se encarga de atender al periférico que interrumpió y realizar la operación de E/S

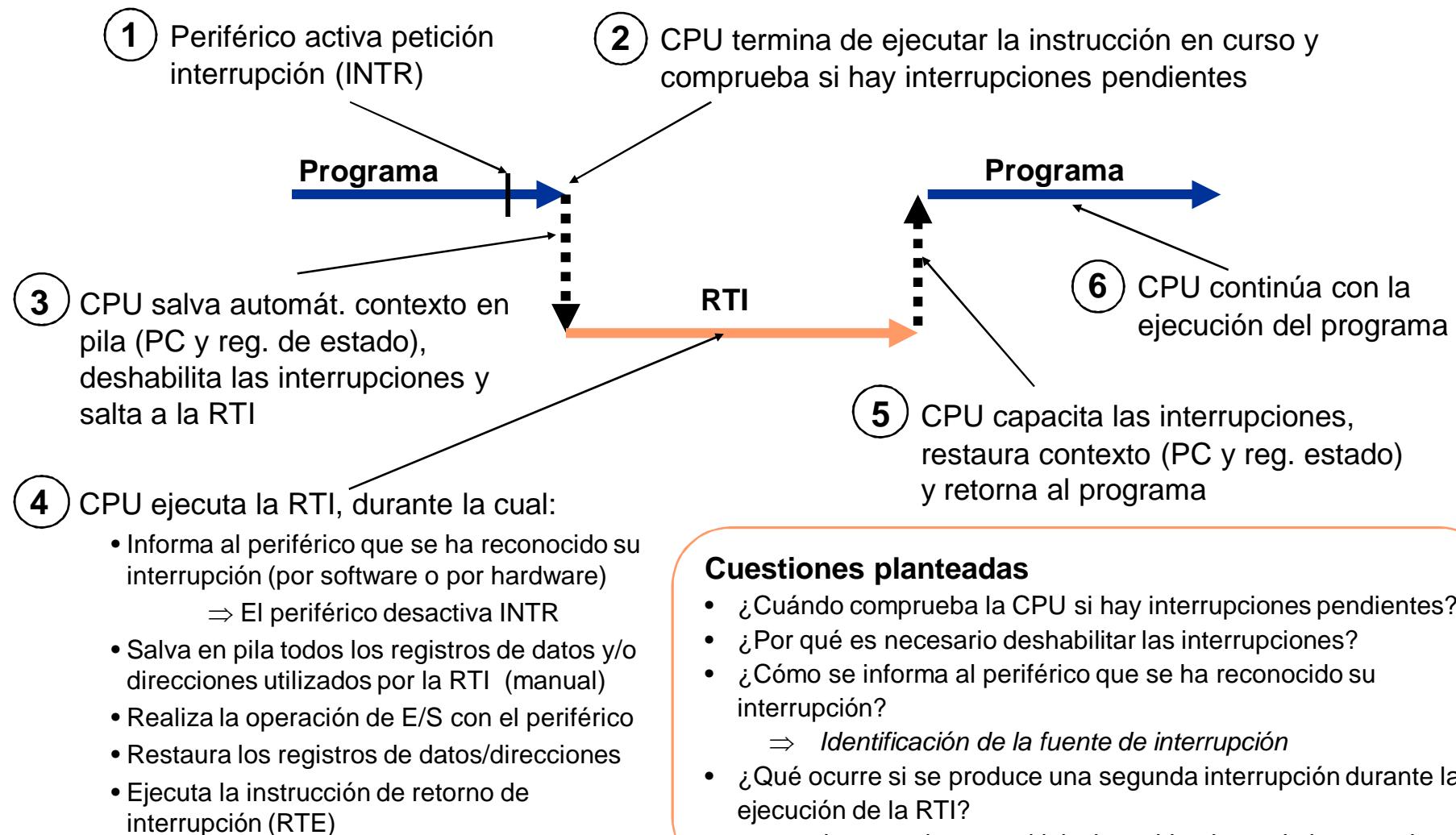


E/S por interrupciones

- **Analogías entre una subrutina y una RTI**
 - Se rompe la secuencia normal de ejecución
 - Cuando terminan de ejecutarse se debe retornar al punto de ruptura
 - Debemos **guardar el PC** en ambos casos
- **Diferencias entre una subrutina y una RTI**
 - En una subrutina el programador sabe en qué punto exacto se rompe la secuencia
 - **Una RTI puede ejecutarse en cualquier momento, sin control del programador**
 - Necesario **guardar el registro de estado** y restaurarlo al retornar de la RTI
 - Normalmente se realiza automáticamente
 - Necesario **guardar los registros** que utiliza la RTI en la pila y restaurarlos al retornar de la RTI
 - Normalmente hay que realizarlo de forma manual



Secuencia de eventos en el tratamiento de una interrupción



Cuestiones planteadas

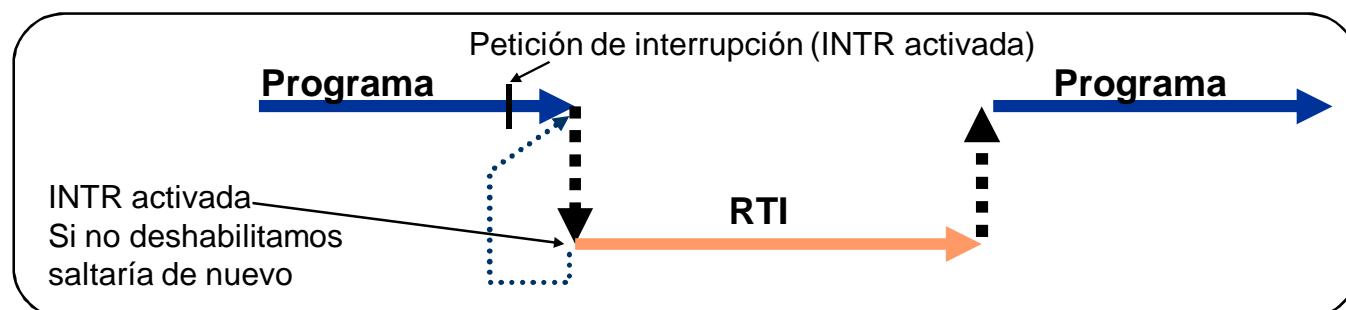
- ¿Cuándo comprueba la CPU si hay interrupciones pendientes?
- ¿Por qué es necesario deshabilitar las interrupciones?
- ¿Cómo se informa al periférico que se ha reconocido su interrupción?
⇒ *Identificación de la fuente de interrupción*
- ¿Qué ocurre si se produce una segunda interrupción durante la ejecución de la RTI?
⇒ *Interrupciones multinivel y anidamiento de interrupciones*

Comprobación de peticiones de interrupción pendientes

- La CPU comprueba si hay interrupciones pendientes (línea INTR activada) **al final de la ejecución de cada instrucción**
 - **Motivo:**
 - Sólo es necesario guardar el PC, el reg. de estado y los registros accesibles por programa (registros de datos y/o direcciones)
 - Si se interrumpiese una instrucción en mitad de la ejecución sería necesario guardar el valor de todos los registros internos de la CPU
 - Reg. de instrucción, registros de dirección de datos, registros de datos de memoria, etc.
 - **¿Qué ocurre con los procesadores segmentados?**

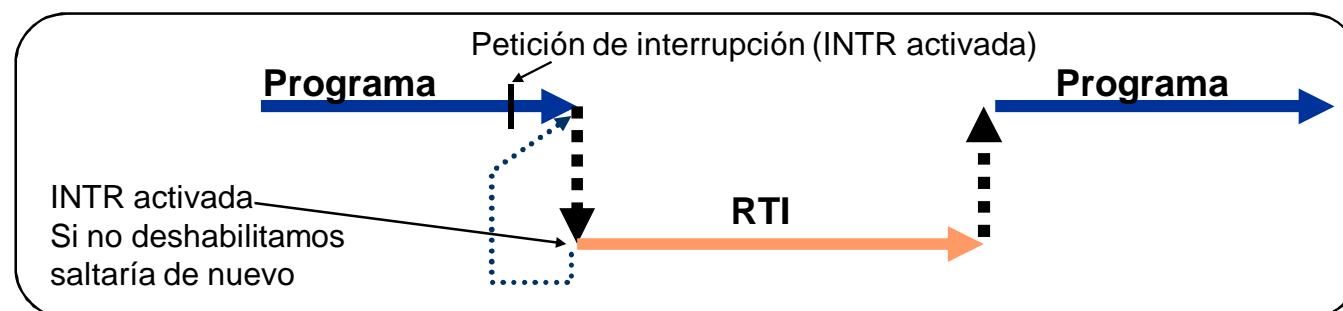
Inhibición o deshabilitación de las interrupciones

- Antes de saltar a la RTI es necesario inhibir o deshabilitar las interrupciones
 - **Motivo:** Si no se inhiben la CPU puede entrar en un bucle infinito
 - Cuando se entra en la RTI el periférico todavía no ha desactivado su petición
 - Si las interrupciones están capacitadas ⇒ la CPU detecta una interrupción pendiente y vuelve saltar a la RTI una y otra vez
 - Antes de finalizar la RTI hay que asegurarse de que el periférico ha desactivado la línea de petición INTR
 - **Por software:** accediendo al registro de estado o de datos del interfaz
 - **Por hardware:** activando una señal de reconocimiento de interrupción (INTA)



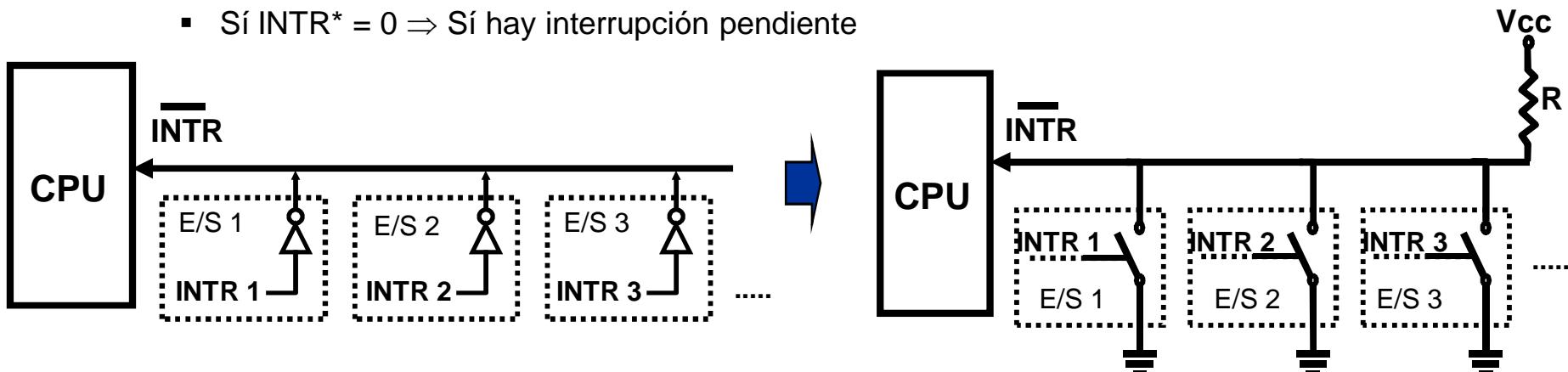
Inhibición o deshabilitación de las interrupciones

- Alternativas
 - **deshabilitación global**
 - Se inhiben todas las interrupciones ⇒ ningún otro periférico podrá interrumpir durante la ejecución de la RTI
 - **deshabilitación o enmascaramiento selectivo**
 - Cuando hay varios niveles de interrupción se pueden deshabilitar las interrupciones por el nivel que interrumpe, pero no necesariamente por el resto de niveles
 - Véase interrupciones multinivel y anidamiento de interrupciones



Identificación de la fuente de interrupción

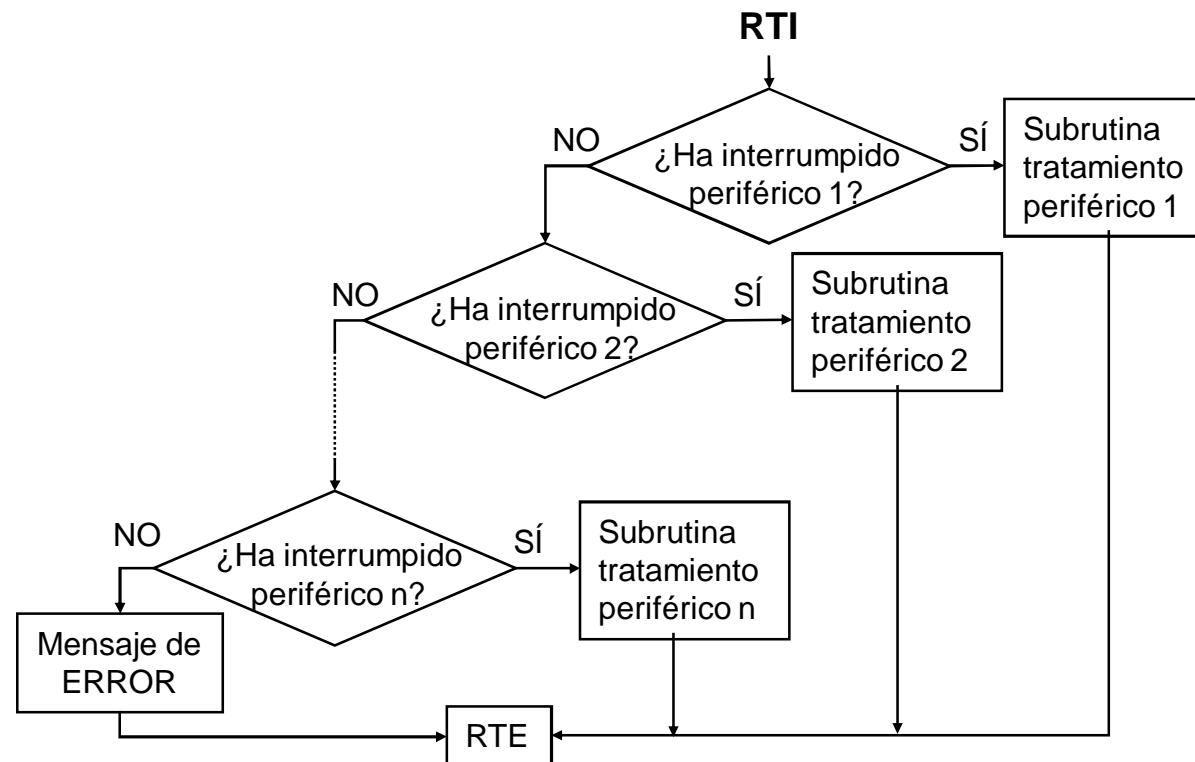
- Normalmente hay más periféricos que líneas de interrupción
- En una misma línea de interrupción es posible conectar varios periféricos
 - Normalmente se utiliza lógica negativa ($\overline{\text{INTR}}^*$) y cableada (en colector abierto, “open collector”)
 - Sí $\overline{\text{INTR}}^* = 1 \Rightarrow$ No hay interrupción pendiente
 - Sí $\overline{\text{INTR}}^* = 0 \Rightarrow$ Sí hay interrupción pendiente



- Cuando existen varias fuentes de interrupción es necesario un mecanismo para identificar al periférico que interrumpió y ejecutar la RTI adecuada para atender a ese periférico particular
 - **Identificación software:** por encuesta (polling)
 - **Identificación hardware:** por vectores

Identificación software por encuesta (polling)

- La RTI examina los registros de estado de cada periférico hasta hallar el que tiene activado su bit de petición de interrupción
 - Una vez detectado el periférico que interrumpió se ejecuta una subrutina particular del periférico en cuestión
 - Durante la ejecución de esa rutina se debe desactivar el bit de petición de interrupción del periférico



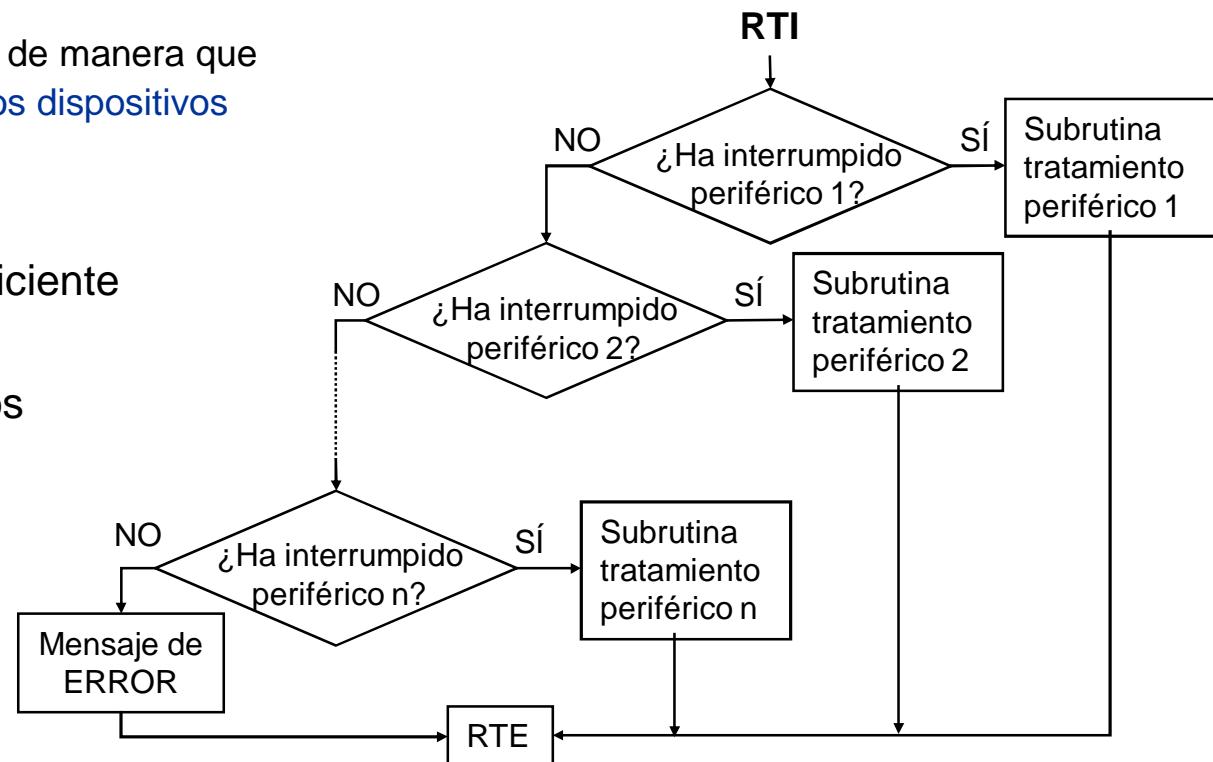
Identificación software por encuesta (polling)

▪ Asignación de prioridades

- El método de encuesta introduce un mecanismo de prioridades software
 - En caso de peticiones simultáneas se atiende por orden de encuesta
 - La RTI se suele diseñar de manera que se pregunta primero a los dispositivos más prioritarios

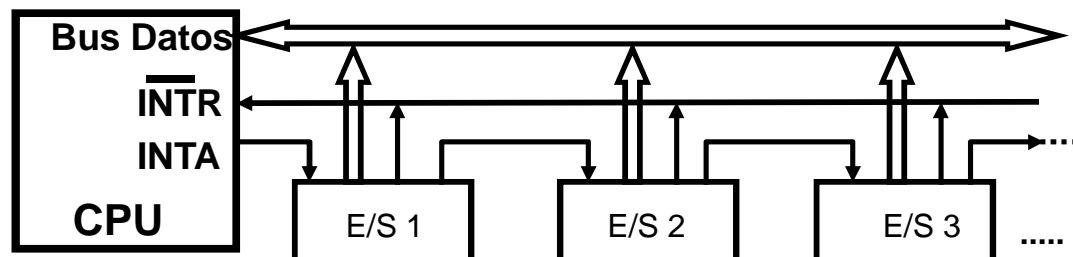
▪ Rendimiento

- No es un sistema muy eficiente
- Se desperdicia tiempo consultando a dispositivos que no han solicitado servicio



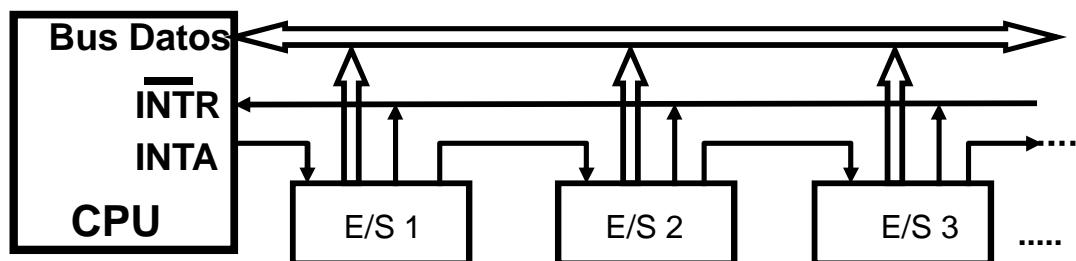
Identificación hardware: interrupciones vectorizadas

- Los periféricos son capaces de identificarse ellos mismos enviando un código (**nº de vector**) a la CPU.
 - Cuando el periférico recibe una señal de reconocimiento de interrupción **INTA** (“*Interruption Ack.*”) envía el **nº de vector** a través del bus de datos
 - A partir del **nº de vector** la CPU calcula la dirección de comienzo de la RTI



Identificación hardware: interrupciones vectorizadas

- Secuencia de eventos en el tratamiento de una interrupción vectorizada
 1. El periférico activa la señal de interrupción (**INTR*=0**)
 2. La CPU activa la señal de confirmación de interrupción (**INTA=1**) que **se conecta a los dispositivos de forma encadenada** (daisy-chain)
 3. Un periférico que no ha interrumpido, cuando recibe la señal INTA, la propaga al siguiente
 4. Cuando el periférico que interrumpió recibe la señal INTA vuelca su número de vector sobre el bus de datos y desactiva la señal de petición de interrupción. Este periférico no propaga INTA
 5. La CPU calcula la dirección de comienzo de la RTI a partir del nº de vector
 6. La CPU salva el contexto en pila (CPU y reg. de estado) y salta a la RTI
 7. Se guardan los registros accesibles por programa, se ejecuta la operación de E/S y se retorna de la interrupción al programa principal restaurando previamente todo el contexto



Identificación hardware: interrupciones vectorizadas

- **Ventajas**

- La transmisión de INTA es totalmente hardware ⇒ es mucho más rápido que el método de encuesta

- **Desventajas**

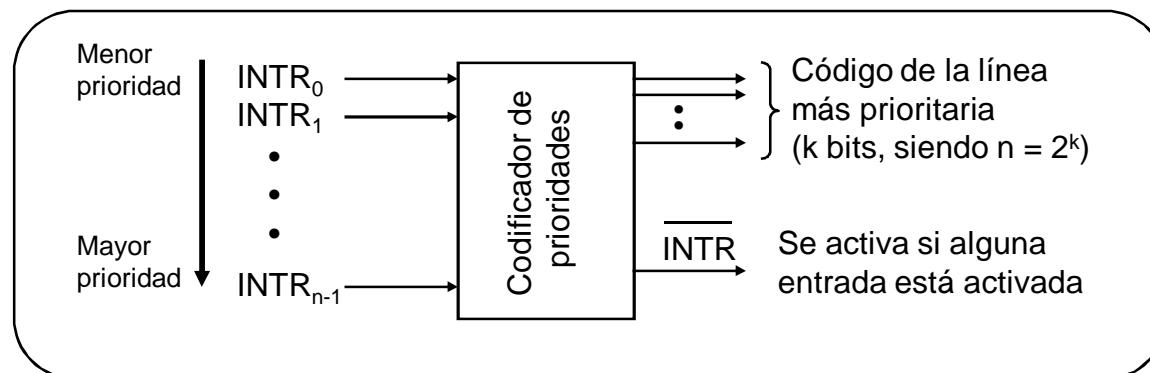
- El nº de dispositivos que se pueden identificar con este método depende del nº de bits que utilicemos para el nº vector
 - Ejemplo: con un nº de vector de 4 bits podemos identificar 16 dispositivos

- **Solución:** pueden utilizarse una estrategia mixta

- Un mismo nº de vector puede utilizarse para identificar a un grupo de varios dispositivos
 - Cuando la CPU recibe un nº de vector de grupo, la RTI debe identificar al dispositivo particular de ese grupo mediante encuesta

Sistemas con varios niveles de interrupciones

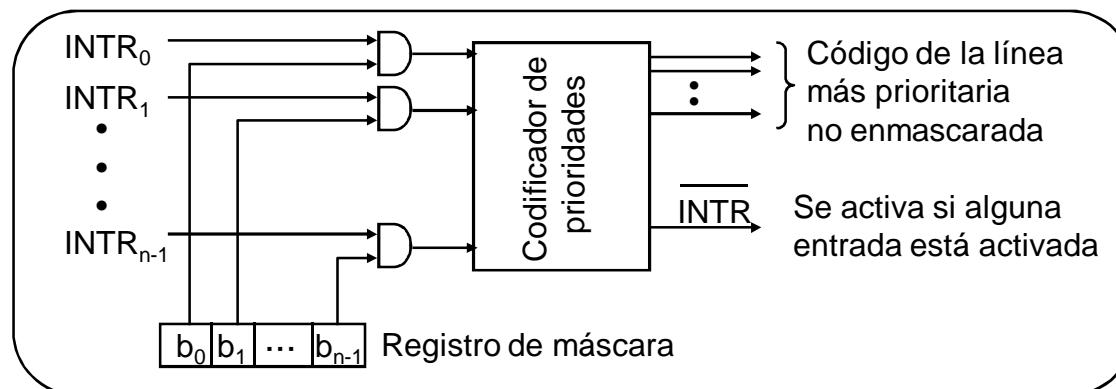
- **Interrupciones multinivel**
 - Existen varias líneas o niveles de petición de interrupción
 - Cada nivel tiene asignado una prioridad distinta
 - A cada línea de interrupción se pueden conectar uno o varios dispositivos
- **Resolución de conflictos:**
 - Peticiones simultáneas por la misma línea
 - Se resuelve con alguno de los mecanismos estudiados anteriormente
 - Mediante encuesta (software)
 - Mediante vectores (hardware)
 - Peticiones simultáneas por líneas distintas
 - Se suele resolver mediante un **codificador de prioridades** ⇒ Se atiende a la línea más prioritaria



Sistemas con varios niveles de interrupciones

▪ Enmascaramiento selectivo:

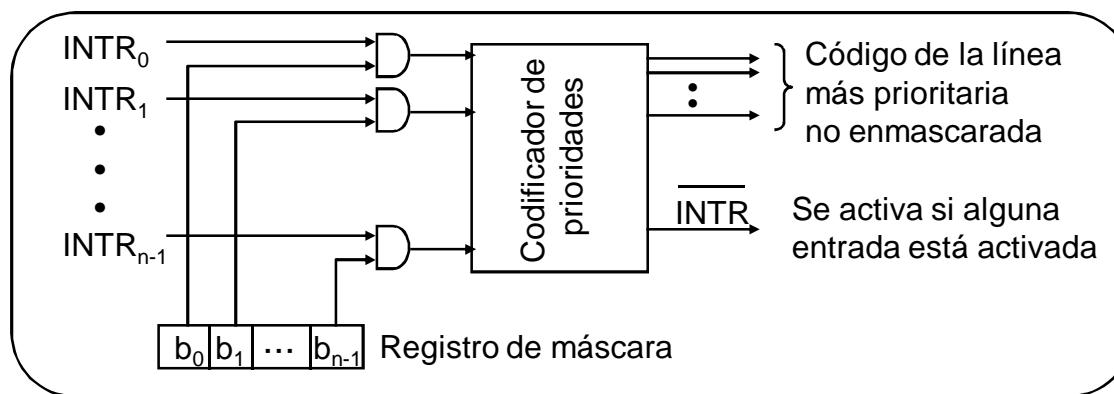
- Los sistemas de interrupciones multinivel permiten enmascarar o deshabilitar selectivamente las interrupciones por determinados niveles
- Para ello se utiliza un **registro de máscara**
 - 1 bit de máscara b_k por nivel
 - Si $b_k = 1 \rightarrow$ Nivel INTR_k habilitado
 - Si $b_k = 0 \rightarrow$ Nivel INTR_k deshabilitado o enmascarado



Sistemas con varios niveles de interrupciones

Anidamiento de interrupciones

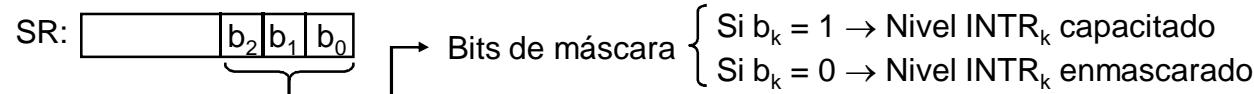
- En general, en los sistemas de interrupciones multinivel se permite el anidamiento de interrupciones
 - Mientras se ejecuta la RTI de un determinado nivel se inhiben las interrupciones por el mismo nivel o inferiores, pero se pueden atender petición de interrupción de mayor nivel
- El anidamiento se controla mediante el registro de máscara
 - Cuando se produce una interrupción de prioridad P_k se enmascaran todas las interrupciones de prioridad $P \leq P_k$



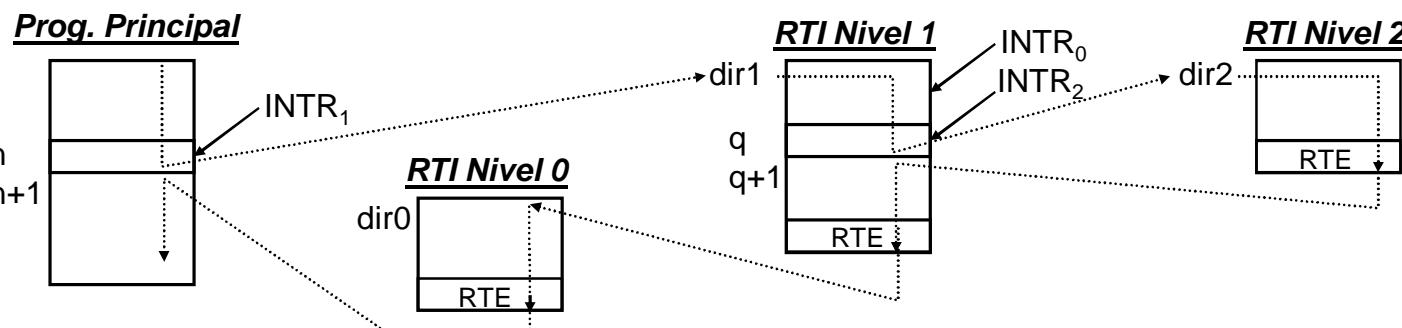
3. Ejemplo de anidamiento de interrupciones

Sistema con 3 niveles de interrupción: $\begin{cases} \text{INTR}_0 \\ \text{INTR}_1 \\ \text{INTR}_2 \end{cases}$ ($\text{INTR}_0 < \text{INTR}_1 < \text{INTR}_2$)

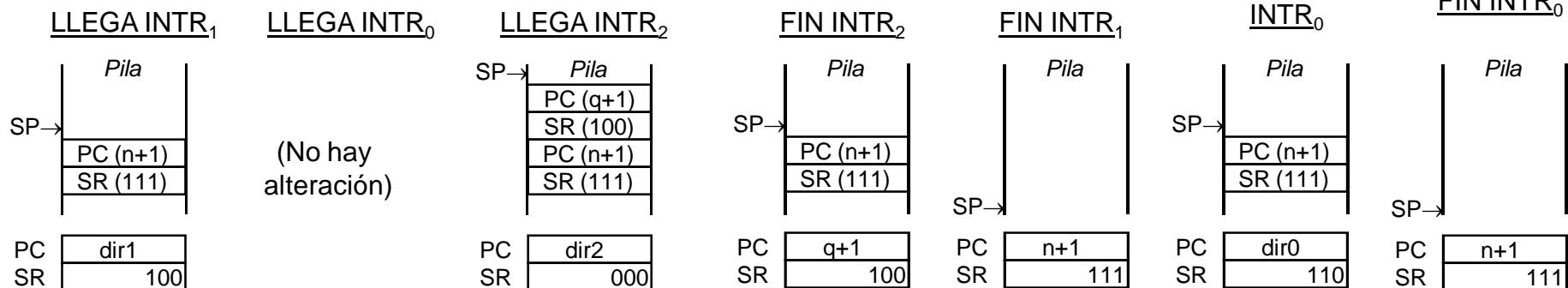
Registro de estado



Supongamos que se producen 3 peticiones de interrupción en el orden $\text{INTR}_1 - \text{INTR}_0 - \text{INTR}_2$



Evolución de PC, SR y Pila



Interrupciones autovectorizadas

- La mayoría de sistemas de interrupciones multinivel asignan un vector de interrupción por defecto a cada nivel de interrupción denominada **autovector**
 - El autovector almacena la **dirección de comienzo de la RTI asignada por defecto** a ese nivel
 - Los autovectores se utilizan para periféricos que no son capaces de generar su propio nº de vector
 - El autovector activara una RTI por defecto
 - La RTI por defecto utilizará un **mechanismo de encuesta** para identificar al periférico que interrumpió

Ejemplo

Computador con 4 niveles de interrupción:
(direcciones de 32 bits)

$\left\{ \begin{array}{l} \text{INTR}_0 \rightarrow \text{autovector } \$00000010 \\ \text{INTR}_1 \rightarrow \text{autovector } \$00000014 \\ \text{INTR}_2 \rightarrow \text{autovector } \$00000018 \\ \text{INTR}_3 \rightarrow \text{autovector } \$0000001C \end{array} \right.$

MEMORIA

\$00000010	:	
		dir. Inicio RTI Nivel 0
\$00000014		dir. Inicio RTI Nivel 1
\$00000018		dir. Inicio RTI Nivel 2
\$0000001C		dir. Inicio RTI Nivel 3
	:	

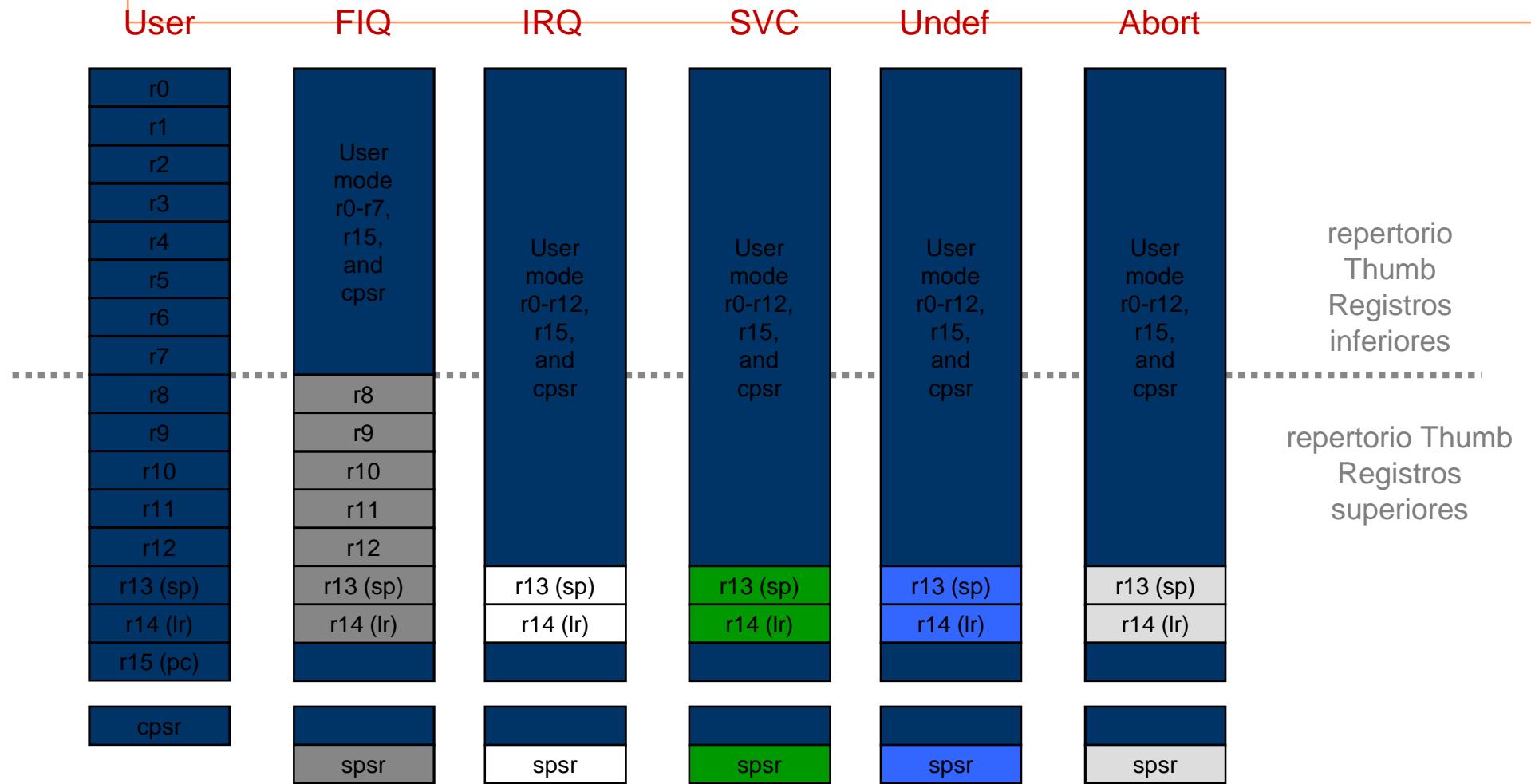
Recordatorio: excepciones en ARM

- Los procesadores ARM tienen 7 modos de operación:
 - **User (usr)**: estado normal de ejecución
 - **FIQ** : manejo de interrupciones rápidas para transferencias de datos
 - **IRQ** : manejo de interrupciones de propósito general o lentas
 - **Supervisor (svc)**: modo protegido para el sistema operativo
 - **Abort (abt)**: usado para gestionar las fallos de acceso a memoria (prefetching o accesos convencionales)
 - **Undef (und)**: manejo de excepciones por códigos de operación no definidas
 - **System** : modo privilegiado para el sistema operativo, usando los mismos registros que en el modo usuario

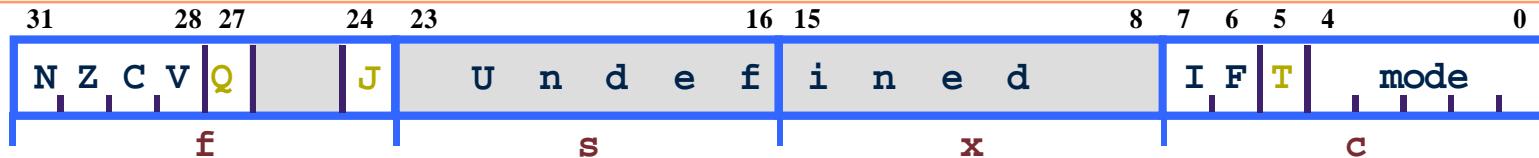
Registros y modos

- El modo en el que se encuentra el procesador determina los registros accesibles
- En cada modo se puede acceder a:
 - Un conjunto particular de registros generales (r0-r12)
 - Registros de puntero de pila (r13) y enlace (r14) particulares
 - El contador de programa (r15)
 - El registro de estado actual del programa (cpsr)
- En los modos privilegiados (excepto system) se puede acceder también a:
 - Un registro especial que almacena el estado del programa (spsr): **permite guardar el estado sin tener que apilarlo**

Registros y modos



Registro de estado



- Códigos de condición
 - N = resultado de la ALU negativo (C2)
 - Z = resultado de la ALU cero
 - C = operación de ALU produce acarreo
 - V = operación de ALU desborda (C2)
 - Sólo para la arquitectura 5TE/J
- Bits de habilitación de interrupciones
 - I = 1: Deshabilita IRQ.
 - F = 1: Deshabilita FIQ.
 - Sólo para arquitecturas -T
 - T = 0: repertorio ARM
 - T = 1: repertorio Thumb
- Bits de modo
 - Especifican el modo del procesador

Gestión de excepciones

- Cuando se produce una excepción **de forma automática**:
 - Se copia el CPSR en SPSR_<modo>
 - Se modifican los bits adecuados del CPSR
 - Cambio a repertorio ARM
 - Actualizar el registro de estado con la excepción correspondiente
 - Deshabilitar interrupciones (si procede)
 - Se guarda la dirección de retorno en LR_<modo>
 - Se actualiza el PC con el vector correspondiente

0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Tabla de Vectores

Gestión de excepciones

- Además se debe actualizar la dirección de retorno:
 - Restar 4 a LR en el caso de las interrupciones
- Guardar en pila los registros compartidos que se vayan a modificar.
 - ¿hay que guardar LR en pila?
- Para volver, la rutina de tratamiento de excepción tiene que:
 - Restaurar los registros
 - Restaurar el CPSR a partir del SPSR_<modo>
 - Restaurar el PC a partir del LR_<modo>
 - Se puede hacer con una única instrucción de store múltiple

Marco de pila de una rutina

- Estructura de una rutina:

Código de entrada (prólogo)
Cuerpo de la rutina
Código de salida (epílogo)

Construye el marco

**Destruye el marco y
hace el retorno**

Marco de pila de una rutina

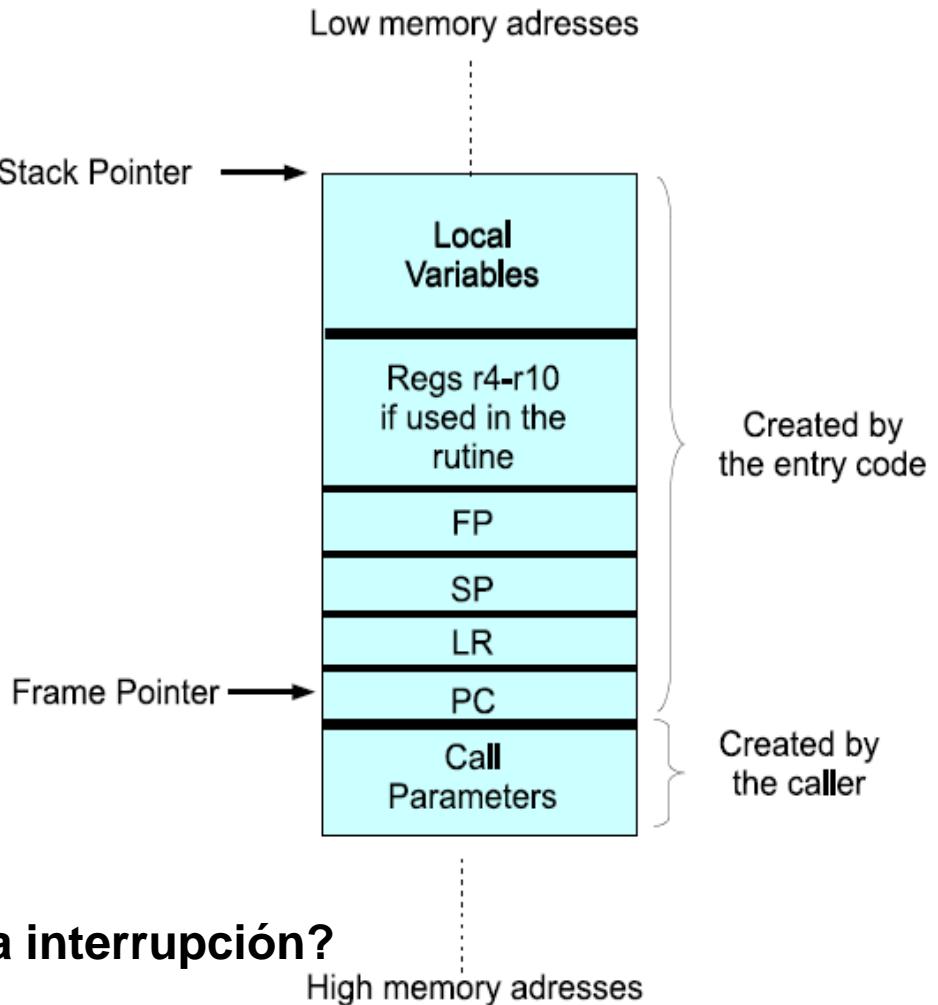
■ Estructura de una rutina:

- Prólogo

```
MOV    IP, SP  
STMDB  SP!, {r4-r10,FP,IP,LR,PC}  
SUB   FP, IP, #4  
SUB   SP, #SpaceForLocalVariables
```

- Epílogo:

```
LDMDB  FP, {r4-r10,FP,SP,PC}
```

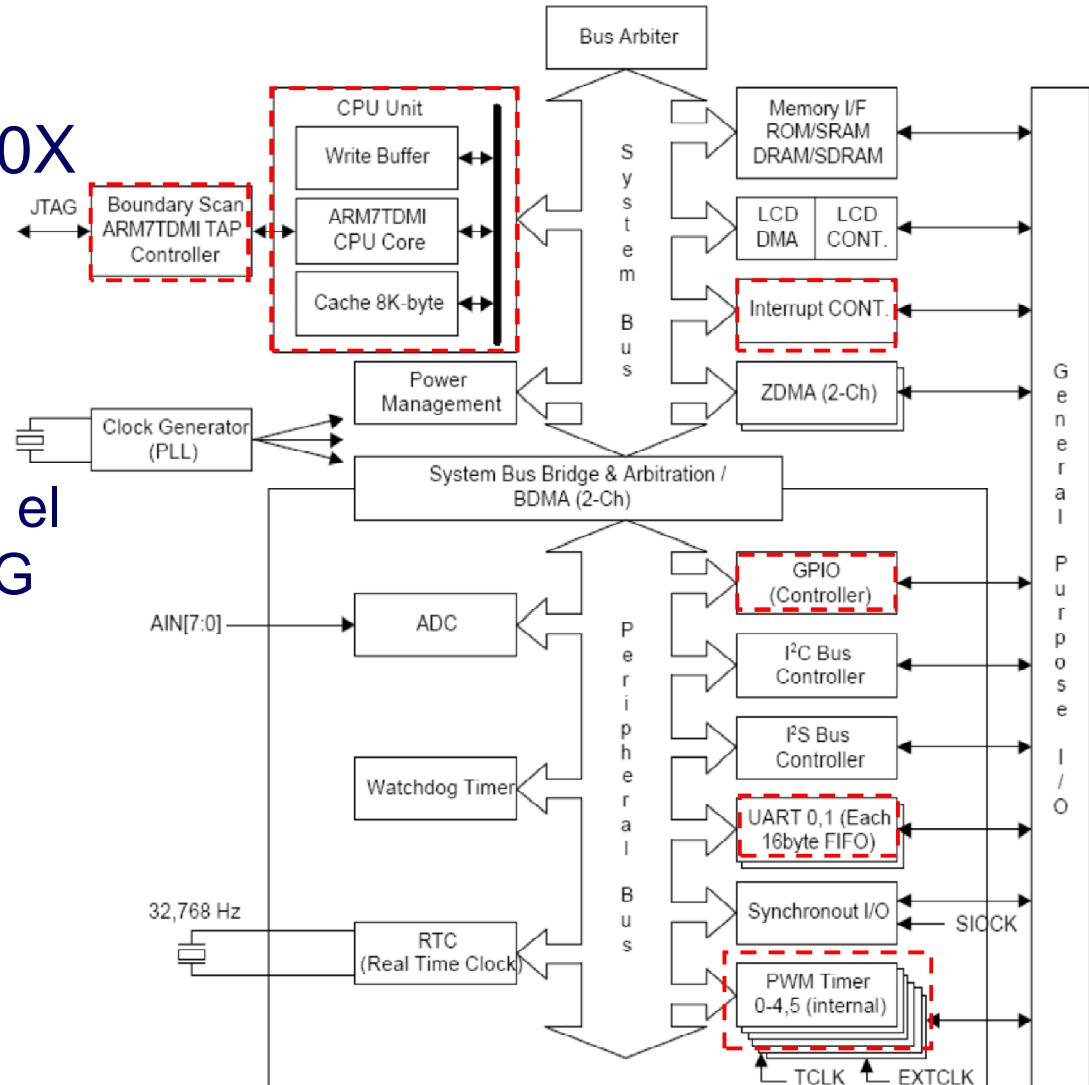


¿Qué hay que añadir en el caso de una interrupción?

Gestión de las interrupciones en nuestra placa de laboratorio:

System-on-Chip S3C44B0X

- Procesador: ARM7TDMI
- Controladores E/S
- Buses
- Comunicación directa con el PC a través del puerto JTAG
- Temporizadores
- **Controlador de interrupciones**
- y muchas otras cosas...

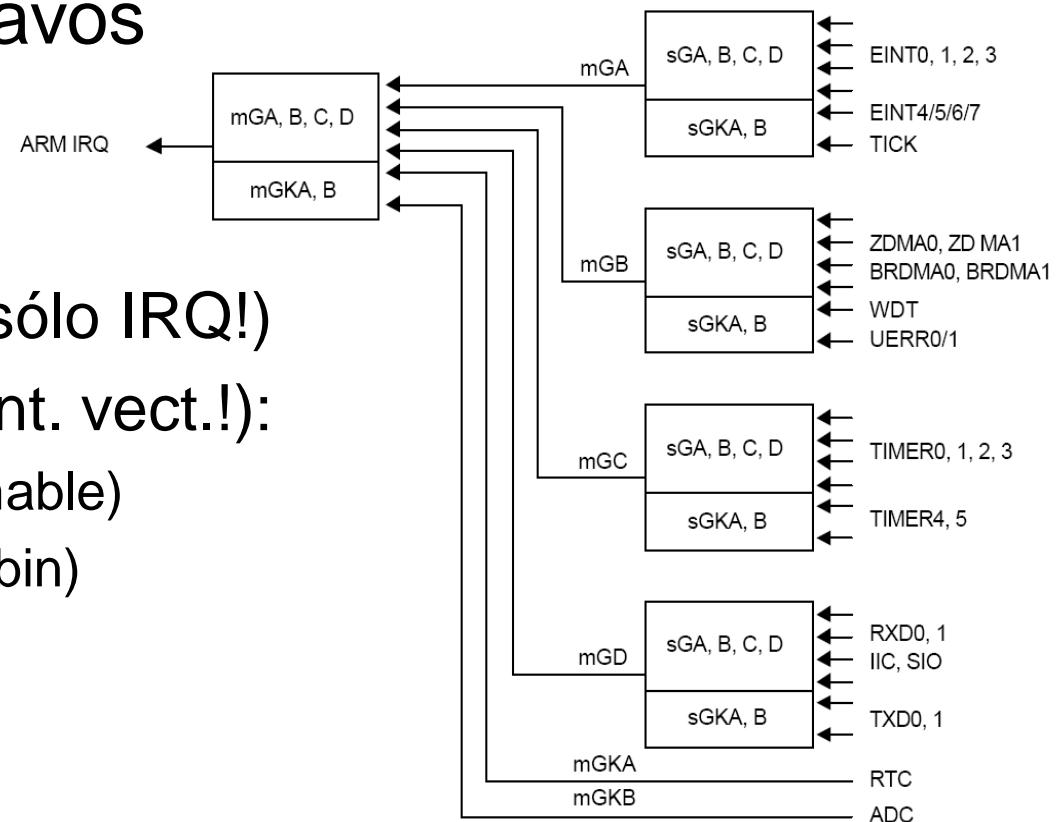


Controlador de interrupciones del S3C44B0X

- Permite ampliar el nº de líneas de petición de interrupción del ARM7TDMI
- 30 posibles fuentes de interrupción usando 26 líneas (algunas fuentes comparten línea)
- Añade soporte de interrupciones vectorizadas (¡Solo para aquellas fuentes que usan IRQ!)
- Implementado mediante 5 controladores encadenados (1 master + 4 slaves)

Controlador de interrupciones

- 26 líneas de interrupción
- 1 maestro + 4 esclavos
- Configurable:
 - Modo IRQ/FIQ
 - Int. vectorizadas (¡sólo IRQ!)
 - Prioridades (¡sólo int. vect.!):
 - Orden (fijo/programable)
 - Modo (fijo/round-robin)



Funcionamiento interrupciones vectorizadas:

- Cuando el ARM intenta leer la instrucción en la dir. 0x18 (vector IRQ)
- El controlador actúa sobre el bus de datos e inserta una instrucción de salto al vector correspondiente a la línea más prioritaria activa
 - EINT0 (0x20)
 - EINT1 (0x24)
 - ...
 - EINT4/5/6/7 (0x30)
 - ...
 - INT_ADC (0xc0)

Controlador de interrupciones

- Código de ejemplo para int. vectorizadas:

...

b HandlerIRQ ; 0x18

b HandlerFIQ ; 0x1c

ldr pc,=HandlerEINT0 ; 0x20

ldr pc,=HandlerEINT1 ; 0x24

ldr pc,=HandlerEINT2 ; 0x28

ldr pc,=HandlerEINT3 ; 0x2c

ldr pc,=HandlerEINT4567 ; 0x30

...

Controlador de interrupciones

- Registros de configuración
 - INTCON (Interrupt Control Register), 3 bits
 - V (bit [2]) = 0, habilita las interrupciones vectorizadas
 - I (bit [1]) = 0, habilita la línea IRQ
 - F (bit [1]) = 0, habilita la línea FIQ
 - INTMOD (Interrupt Mode Register), 1 bit por línea
 - 0 = modo IRQ; 1 = modo FIQ

Controlador de interrupciones

- Registros de gestión
 - INTPND (Interrupt Pending Register), 1 bit por línea
 - 0 = no hay solicitud; 1 = hay una solicitud
 - INTMSK (Interrupt Mask Register), 1 bit por línea
 - 0 = int. disponible; 1 = int. enmascarada
 - I_ISPC (IRQ Int. Service Pending Clear register), 1 bit por línea
 - 1 = borra el bit correspondiente del INTPND e indica al controlador el final de la rutina de servicio (**iFIN RUTINA SERVICIO!**)
 - F_ISPC (FIQ Int. Service pending Clear register), 1 bit por línea
 - Idem

Controlador de interrupciones

- Registros de gestión para int. vectorizadas:
 - I_ISPR (IRQ Int. Service Pending Register), 1 bit por línea
 - Indica la interrupción que se está sirviendo actualmente
 - Sólo el bit de la línea más prioritaria puede estar a 1 (#INTPND)

4. DMA: Necesidad de acceso directo a memoria

- ☒ La E/S con espera de respuesta o por interrupciones resulta inadecuada para periféricos de alta velocidad, sobre todo si hay que transferir una gran cantidad de datos

☒ Ejemplo periférico lento

- Procesador a 200 MHz (tiempo ciclo = 5 ns.; Ciclo medio por instrucción: CPI = 2 ciclos
 - ⇒ Una instrucción tarda en promedio $2 \times 5 \text{ ns} = 10 \text{ ns}$ ⇒ el computador puede ejecutar ~100 MIPS
- Queremos imprimir un fichero de 10 Kbytes en una impresora láser de 20 páginas por minuto
- 1 página $\cong 3.000$ caracteres (1 carácter = 1 byte)
 - ⇒ La impresora imprime 60.000 caracteres por minuto = 1 Kbyte/s

a) E/S con espera de respuesta

- La CPU entra en un bucle y envía un nuevo byte cada vez que la impresora está preparado para recibirlo
 - ⇒ La impresora tarda 10 s en imprimir 10 Kbyte
 - ⇒ **La CPU está ocupada con la operación de E/S durante 10 s**
(en ese tiempo la CPU podría haber ejecutado 1000 millones de instrucciones)

b) E/S por interrupciones

- La impresora genera una interrupción cada vez que está preparada para recibir un nuevo byte
 - ⇒ Suponemos que la RTI tiene 10 instrucciones (salvar contexto, comprobar estado, transferir byte, restaurar contexto, RTE)
 - ⇒ Para transferir 10 Kbyte tenemos que ejecutar 10.000 veces la RTI
 - ⇒ hay ejecutar 100.000 instrucciones para atender al periférico ⇒ la CPU tarda 0,001 s
 - ⇒ **La CPU está ocupada con la operación de E/S durante 0,001 s**

CONCLUSIÓN

- La E/S por interrupciones reduce en 10.000 veces el tiempo que la CPU está ocupada gestionando la impresora

Necesidad de DMA

☒ Ejemplo periférico rápido

- Procesador a 200 MHz (tiempo ciclo = 5 ns.; Ciclo medio por instrucción: CPI = 2 ciclos
 - ⇒ Una instrucción tarda en promedio $2 \times 5 \text{ ns} = 10 \text{ ns}$ ⇒ el computador puede ejecutar ~100 MIPS
- Disco con velocidad de transferencia de 10 Mbytes/s (1 byte cada $2 \times 10^{-7} \text{ seg}$)
- Queremos transferir un fichero de memoria a disco de 10 Mbytes

a) E/S con espera de respuesta

- La CPU entra en un bucle y envía un nuevo byte cada vez que el disco está preparado para recibirlo
 - ⇒ El disco tarda 1 seg en recibir un fichero de 10 Mbyte
 - ⇒ **La CPU está ocupada con la operación de E/S durante 1 s**
(en ese tiempo la CPU podría haber ejecutado 200 millones de instrucciones)

b) E/S por interrupciones

- El disco genera una interrupción cada vez que está preparado para recibir un nuevo byte
 - ⇒ Suponemos que la RTI tiene 10 instrucciones (salvar contexto, comprobar estado, transferir byte, restaurar contexto, RTE)
 - ⇒ Para transferir 10 Mbytes tenemos que ejecutar 10^7 veces la RTI
 - ⇒ hay ejecutar 100 millones de instrucciones para atender al periférico ⇒ la CPU tarda 1 s
 - ⇒ **La CPU está ocupada con la operación de E/S durante 1 s**

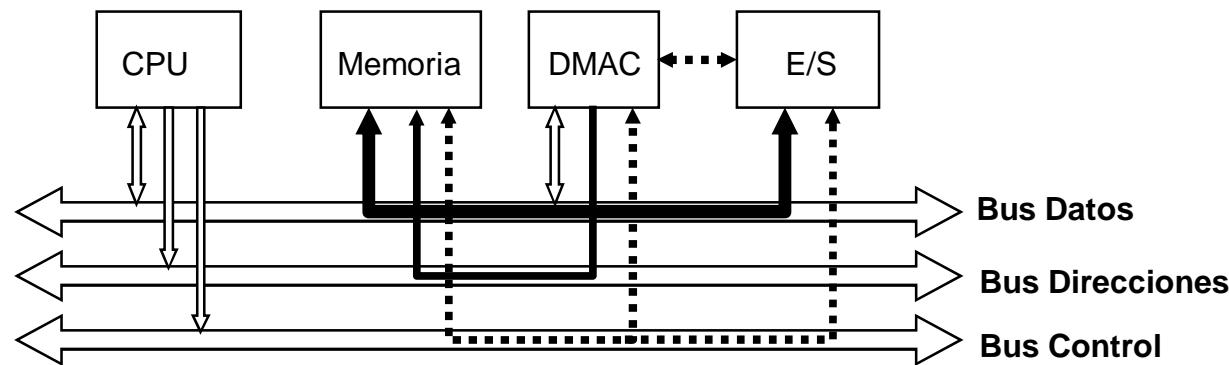
CONCLUSIÓN

- La E/S por interrupciones no mejora el tiempo que la CPU está ocupada en atender al periférico

La técnica de DMA permite la transferencia de datos entre un periférico y la memoria **sin intervención de la CPU** (salvo en la fase de inicialización de los parámetros de la transferencia)

El controlador de DMA (DMAC)

- El controlador de DMA es un dispositivo capaz de controlar una transferencia de datos entre un periférico y memoria sin intervención de la CPU



- El DMAC debe actuar como máster del bus durante la transferencia DMA y debe ser capaz de
 - Solicitar el uso del bus mediante las señales y la lógica de arbitraje necesarias
 - Especificar la dirección de memoria sobre la que se realiza la transferencia
 - Generar las señales de control del bus
 - Tipo de operación (lectura/escritura)
 - Señales de sincronización de la transferencia

Etapas de una transferencia DMA

⊗ Inicialización de la transferencia

- La CPU debe enviar al interfaz del periférico y al DMAC los parámetros de la transferencia

Inicialización del interfaz (Bus master: CPU - Bus slave: Interfaz)

- ⇒ Nº de bytes a transferir
- ⇒ Tipo de transferencia (lectura/escritura)
- ⇒ Otra información de control (pista, sector, etc.)

Inicialización del controlador DMA (Bus master: CPU - Bus slave: DMAC)

- ⇒ Nº de bytes o palabras a transferir
- ⇒ Tipo de transferencia (lectura/escritura)
- ⇒ Dirección de memoria inicial para la transferencia
- ⇒ Nº de canal (para DMAs con varios canales)

- Después de la inicialización la CPU retorna a sus tareas y ya no se preocupa más de la evolución de la transferencia

⊗ Realización de la transferencia

- Cuando el periférico está listo para realizar la transferencia se lo indica al DMAC
- El DMAC pide el control del bus y se realiza la transferencia entre el periférico y la memoria
 - ⇒ **Bus máster:** DMAC + Periférico - **Bus slave:** Memoria
 - ⇒ Despues de la transferencia de cada palabra se actualizan los registros del DMA
 - ✓ Nº de bytes o palabras a transferir
 - ✓ Dirección de memoria

⊗ Finalización de la transferencia

- El DMAC libera el bus y devuelve el control a la CPU
- El DMAC suele activar una señal de interrupción para indicar a la CPU la finalización de la operación de E/S solicitada

Transferencias DMA

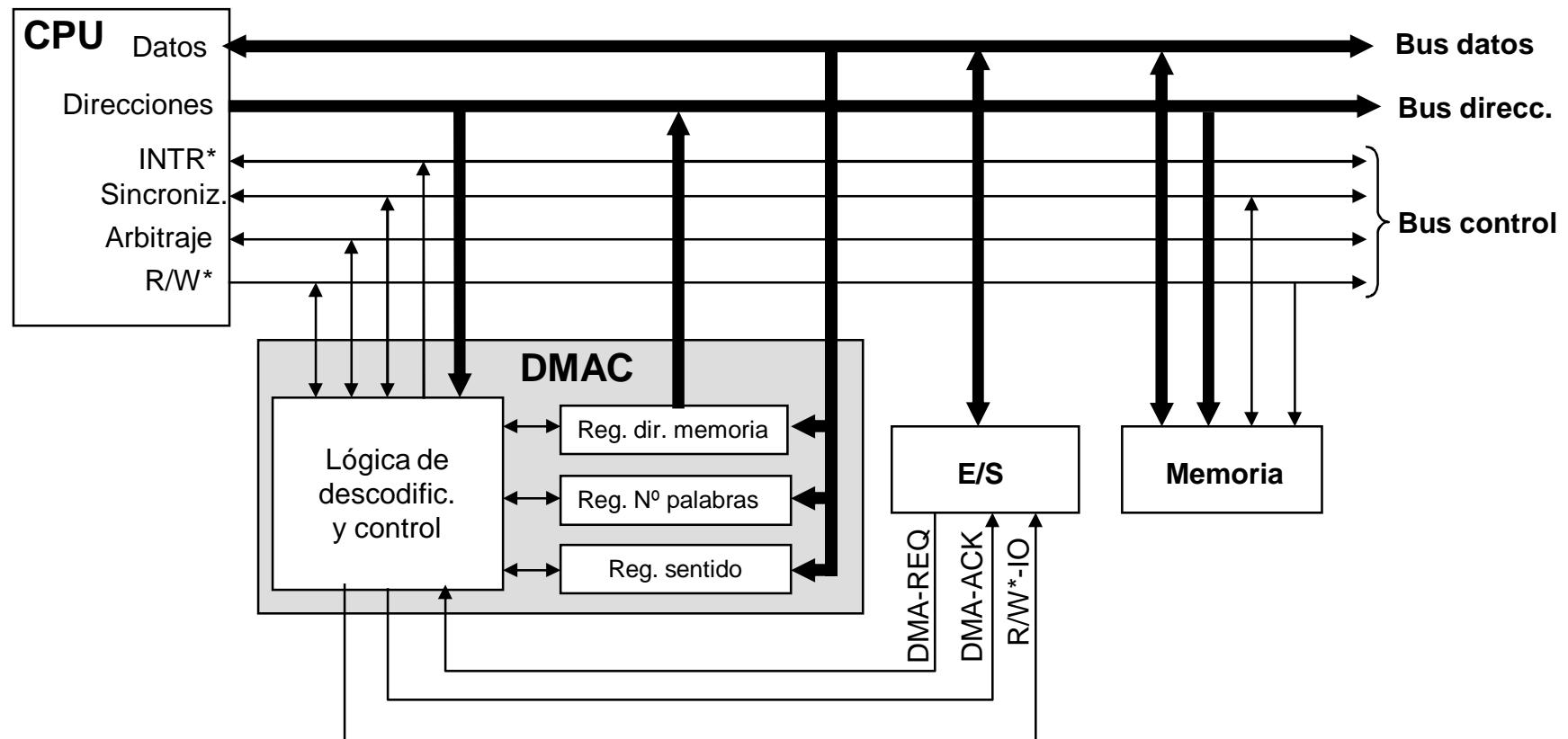
⊗ Transferencia DMA modo ráfaga

- El DMAC solicita el control del bus a la CPU
- Cuando la CPU concede el bus el DMAC no lo libera hasta haber finalizado la transferencia de todo el bloque de datos completo
- **VENTAJAS:**
 - ⇒ La transferencia se realiza de forma rápida
- **DESVENTAJAS:**
 - ⇒ Durante el tiempo que dura la transferencia la CPU no puede utilizar el bus con memoria, lo que puede degradar el rendimiento del sistema

⊗ Transferencia DMA modo robo de ciclo

- El DMAC solicita el control del bus a la CPU
- Cuando la CPU concede el bus al DMAC, se realiza la transferencia de una única palabra y después el DMAC libera el bus
- El DMAC vuelve a solicitar el control del bus tantas veces como sea necesario hasta haber finalizado la transferencia del bloque completo
- La CPU cede el control del bus durante los ciclos que hace uso del mismo
- **VENTAJAS:**
 - ⇒ No se degrada el rendimiento del sistema
- **DESVENTAJAS:**
 - ⇒ La transferencia tarda más tiempo en llevarse a cabo

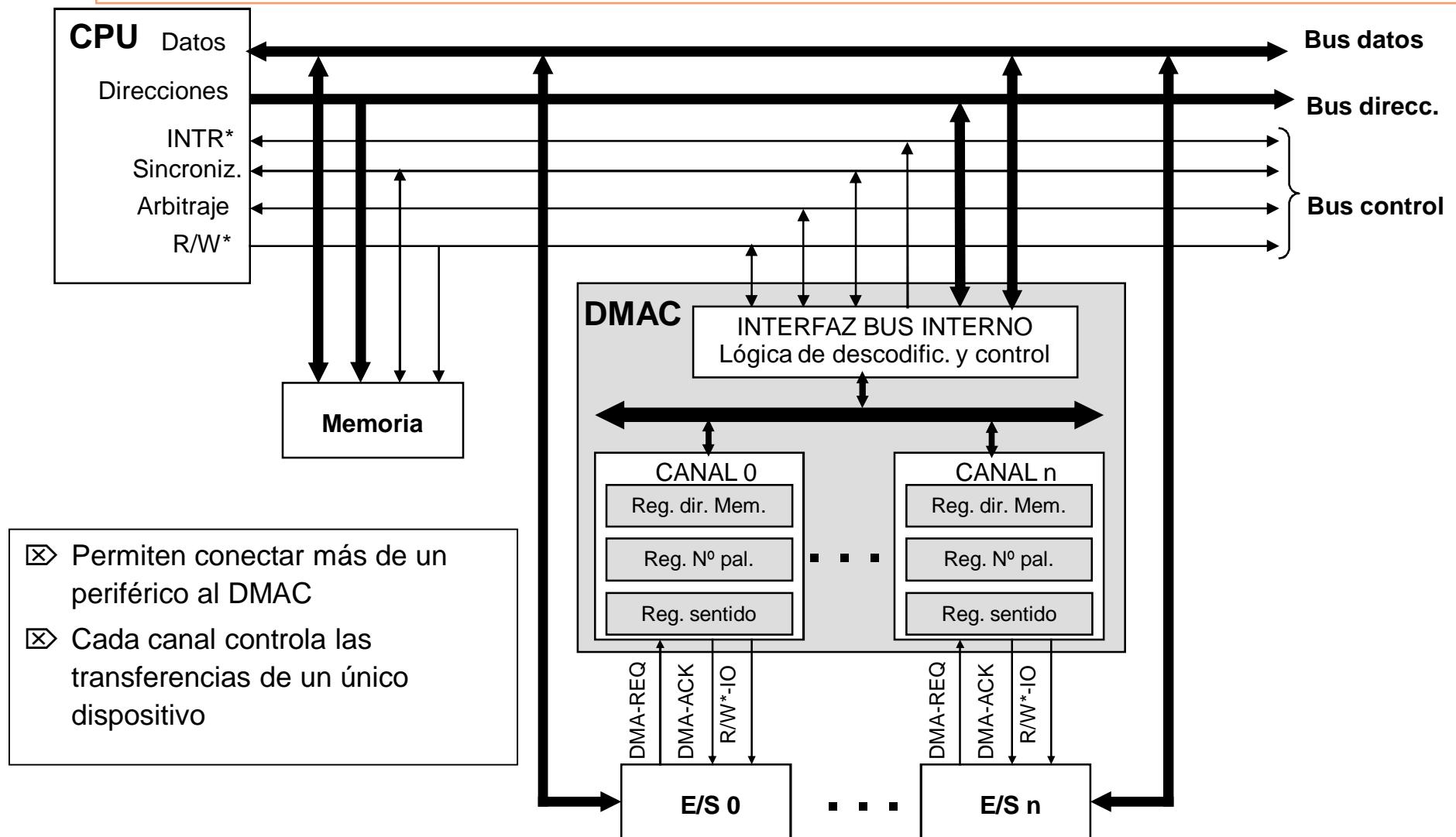
Estructura hardware de un DMA



Registros del DMA

- ☒ **Reg. dir. memoria:** almacena la dir. inicial de memoria y se incrementa/decrementa después de transferir cada palabra
- ☒ **Reg. Nº palabras:** almacena el número de palabras a transferir y se decrementa después de transferir cada palabra
- ☒ **Reg. sentido:** almacena el sentido de la transferencia (lectura o escritura)

Controlador DMA de varios canales



Conclusiones

- Los procesadores deben interactuar con todo tipo de dispositivos
- Estos dispositivos suelen incluir un interfaz para facilitar las comunicaciones
- Las interrupciones son la clave para una E/S eficiente
- Debido al gran número de fuentes de interrupción muchos sistemas incluyen un controlador de interrupciones
- Las transferencias de datos son claves para el rendimiento de un sistema:
 - Los DMAs permiten descargar al procesador