

# Tema 1. Introducción

**Arquitectura de los Computadores**

# Tema 1. Introducción

Arquitectura

Diseño

Introducción

- ➊ **1. Arquitectura de computadores**
- ➋ **2. El diseño de computadores**

# Tema1. Introducción

Arquitectura

Diseño

Introducción

## 1. Arquitectura de computadores

- ◆ 1.1 Niveles de descripción de un computador
- ◆ 1.2 Definición de arquitectura
- ◆ 1.3 Clasificación de arquitecturas

## 2. El diseño de computadores

- ◆ 2.1 El proceso de diseño
  - ◆ a) Establecer requerimientos funcionales
  - ◆ b) Decisiones de implementación
  - ◆ c) Consideración de las nuevas tendencias
- ◆ 2.2 Principios de diseño
  - ◆ a) Acelerar el caso común
  - ◆ b) Ley de rendimientos decrecientes
  - ◆ c) Localidad de referencia

# 1.1 Niveles de descripción de un computador

Arquitectura

Diseño

Introducción

## El concepto de arquitectura

- ➊ **Acuñado por IBM en 1964.** Parte del repertorio de instrucciones visible por el programador:

**[Amdahl, 64].** Presentación del IBM S/360: *La arquitectura de un computador es la estructura del computador que un programador en lenguaje máquina debe conocer para escribir un programa correcto*

- ➋ **Unificar** las diferentes **divisiones de IBM** en una **arquitectura común**
- ➋ **Referencia exclusivamente** al diseño del **repertorio de instrucciones**
- ➋ **Errores de diseño** por falta de consideración de aspectos de implementación

# 1.1 Niveles de descripción de un computador

## El concepto de arquitectura

Arquitectura

Diseño

Introducción

- ➊ Disciplina que trata el **diseño** de máquinas para ejecutar programas con criterios de optimización de rendimiento y coste
  
- ➋ **Aspectos relacionados**
  - ➌ **Diseño del repertorio de instrucciones**
  - ➌ **Diseño de la organización funcional**
  - ➌ **Diseño lógico**
  - ➌ **Implementación**

# 1.1 Niveles de descripción de un computador

Arquitectura

Diseño

Introducción

## Niveles estructurales de Bell y Newell [Bell, 71]:

- Descripción del computador mediante una aproximación por capas
- Cada capa utiliza los servicios que proporciona la del nivel inferior
- **Propone 5 niveles:**
  - De componente
  - Electrónico
  - Digital
  - Transferencia entre registros (RT)
  - Procesador-Memoria-Interconexión (PMS)

# 1.1 Niveles de descripción de un computador

Arquitectura

Diseño

Introducción

## Niveles de Interpretación de Levy [Bell 78, De Miguel 01]:

- Contemplan al computador desde un punto de vista funcional
- Constituido por una serie de máquinas virtuales superpuestas
- Cada máquina interpreta las instrucciones de su nivel, proporcionando servicios a la máquina de nivel superior y aprovechando los de la máquina de nivel inferior
- **Se distinguen 5 niveles:**
  - Aplicaciones
  - Lenguajes de alto nivel
  - Sistema Operativo
  - Instrucciones máquina
  - Microinstrucciones
- Estos niveles son similares a los niveles funcionales de [Tanenbaum 86, 99, 00].

# 1.1 Niveles de descripción de un computador

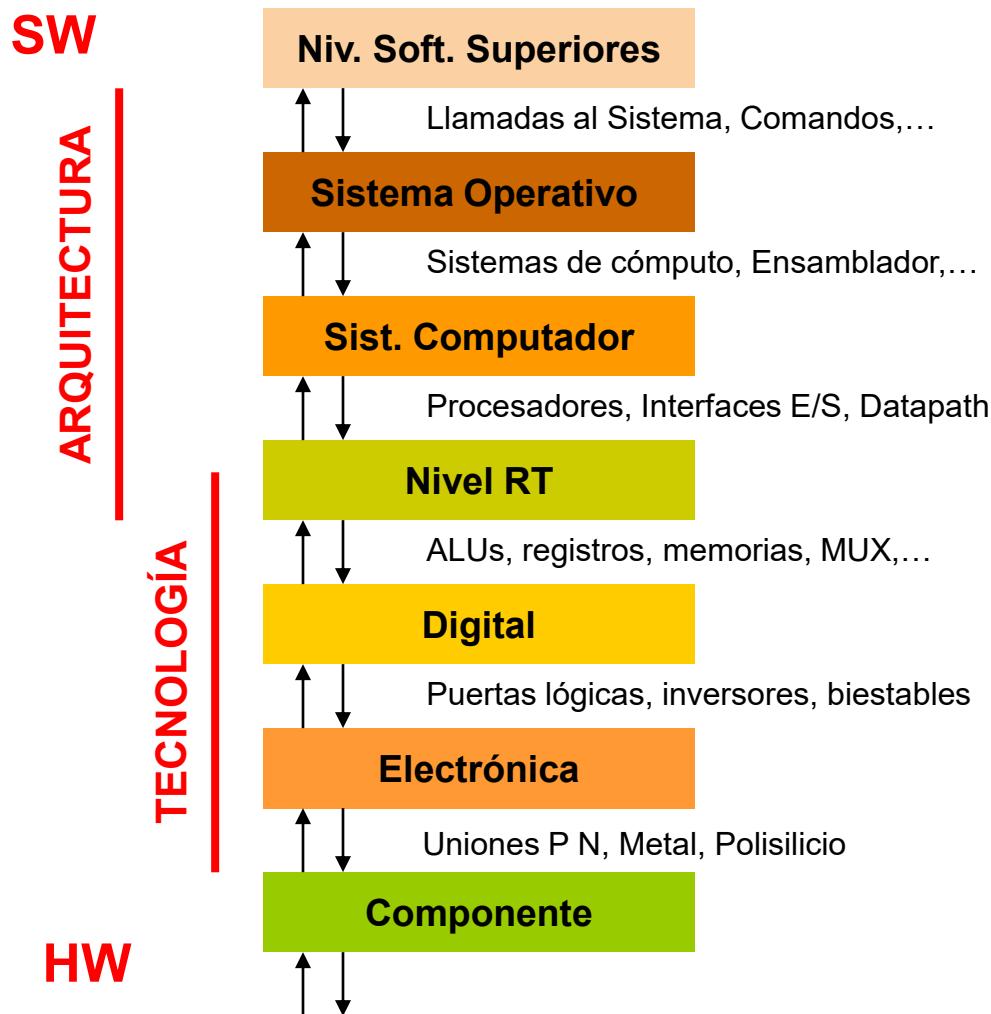
Arquitectura

Diseño

Introducción

## Niveles de abstracción para un computador:

Integra la orientación estructural de los niveles de Bell y Newell y el punto de vista funcional de los niveles de Levy y Tanenbaum.



# 1.1 Niveles de descripción de un computador

Arquitectura

Diseño

Introducción

## Niveles de abstracción para un computador:

### Círculo electrónico

- Puertas lógicas, biestables, etc. Utilizan componentes del nivel anterior.
- Las leyes que lo rigen son las de la electricidad, de naturaleza continua.
- El comportamiento del circuito se describe en términos de corrientes, tensiones y frecuencias.

### Componentes físicos:

- Semiconductores de tipo n y p, metales, polisilicio, etc...
- A partir de estos se construyen bloques: transistores, resistencias, etc.
- Las leyes que lo rigen son las de la electrónica física.

SW

ARQUITECTURA

TECNOLOGÍA

HW

### Niv. Soft. Superiores

Llamadas al Sistema, Comandos,...

### Sistema Operativo

Sistemas de cómputo, Ensamblador,...

### Sist. Computador

Procesadores, Interfaces E/S, Datapath

### Nivel RT

ALUs, registros, memorias, MUX,...

### Digital

Puertas lógicas, inversores, biestables

### Electrónica

Uniones P N, Metal, Polisilicio

### Componente

# 1.1 Niveles de descripción de un computador

Arquitectura

Diseño

Introducción

## Lógica digital

- Las leyes que lo rigen son las del Álgebra de Boole
- Se divide en 2 partes: circuitos combinacionales y secuenciales
- **Nivel combinacional:** se utilizan como componentes las puertas NAND, NOR, NOT, etc, para generar bloques como, multiplexores, decodificadores, conversores de códigos y circuitos aritméticos.
- **Nivel secuencial:** se utilizan como componentes elementos de memoria (biestables) y bloques del nivel anterior para obtener circuitos secuenciales, como registros, contadores, memorias, etc.

SW

ARQUITECTURA

TECNOLOGÍA

HW

## Niv. Soft. Superiores

Llamadas al Sistema, Comandos,...

## Sistema Operativo

Sistemas de cómputo, Ensamblador,...

## Sist. Computador

Procesadores, Interfaces E/S, Datapath

## Nivel RT

ALUs, registros, memorias, MUX,...

## Digital

Puertas lógicas, inversores, biestables

## Electrónica

Uniones P N, Metal, Polisilicio

## Componente

# 1.1 Niveles de descripción de un computador

Arquitectura

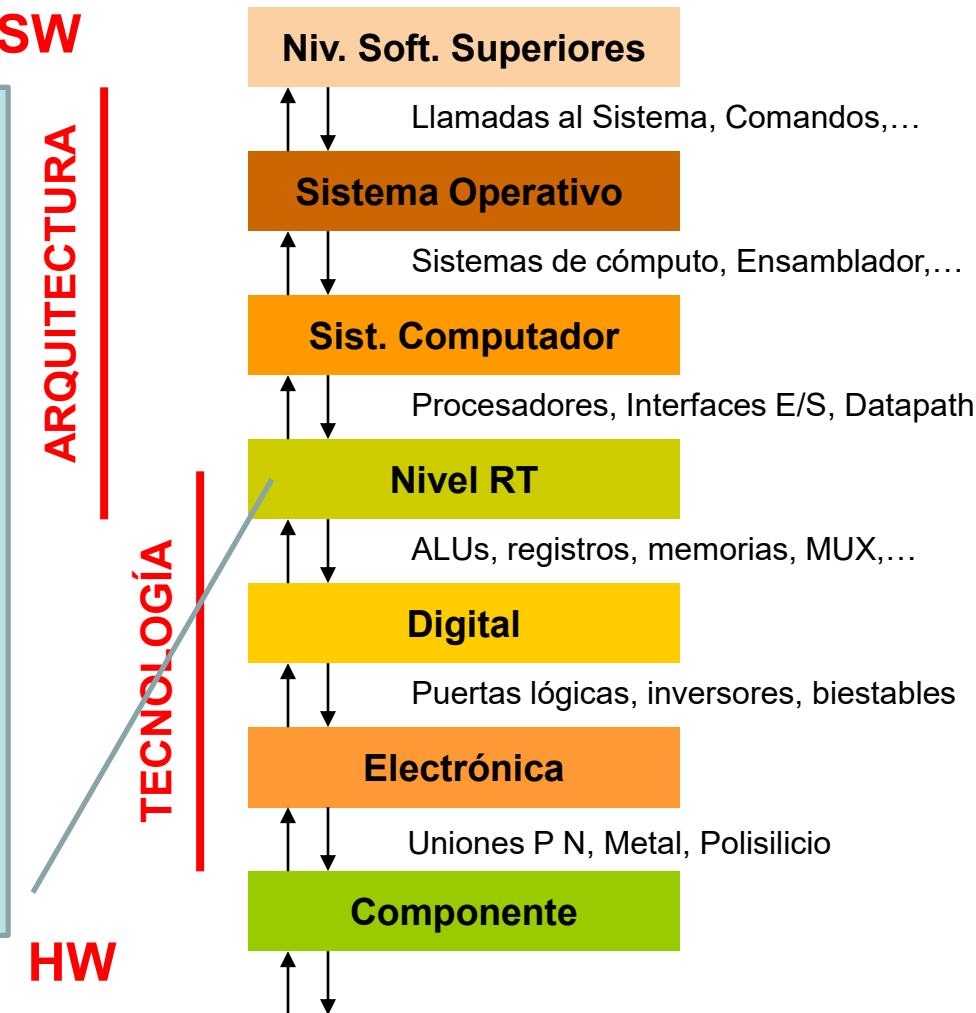
Diseño

Introducción

## Niveles de abstracción para un computador:

### Transferencia entre registros (RT)

- Estudio del comportamiento de las unidades de un computador en términos de transferencia de información entre registros.
- Utiliza los componentes del nivel anterior, registros, circuitos aritméticos, memorias, etc, para crear componentes del procesador o de otros elementos del computador (interfaces).
- En este nivel se incluye como un posible subnivel la microprogramación.



# 1.1 Niveles de descripción de un computador

Arquitectura

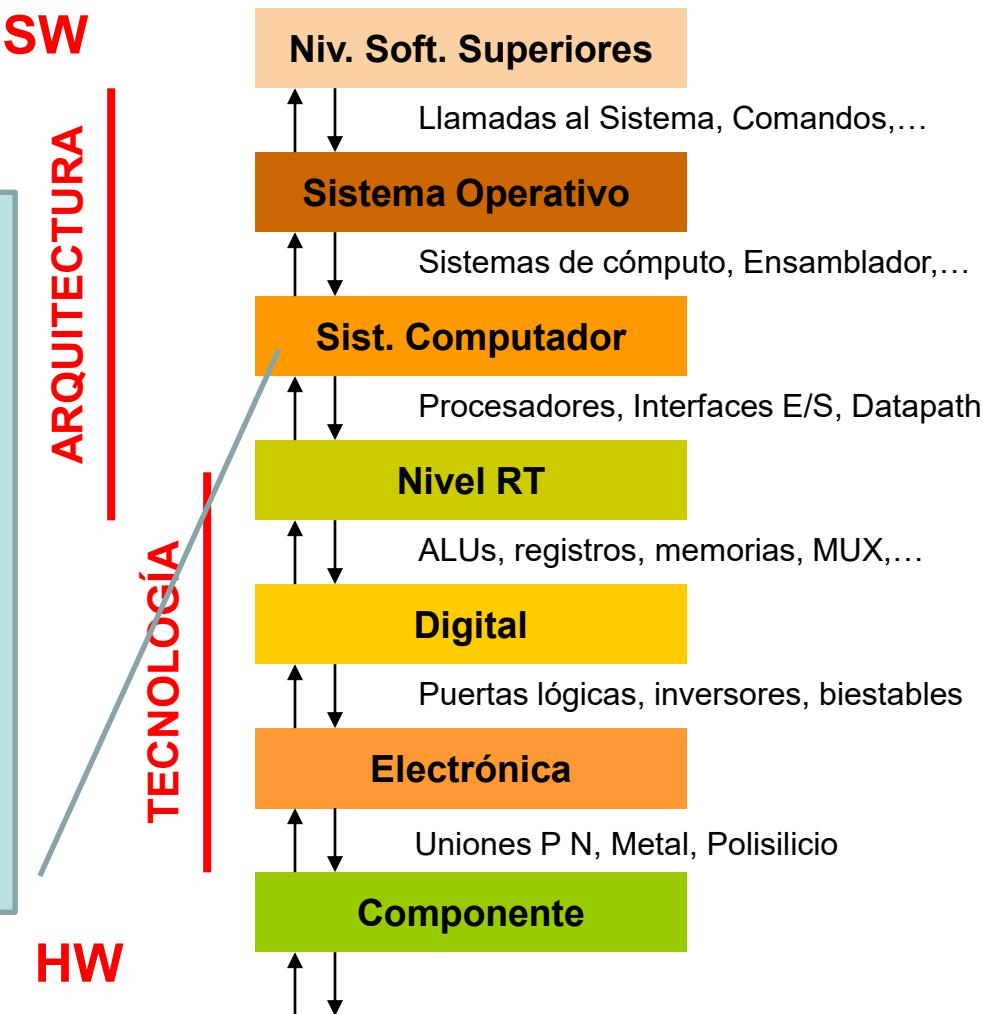
Diseño

Introducción

## Sistema computador

- Especificación de componentes (memorias, procesador, buses, redes de interconexión, periféricos, etc). interconexión entre ellos y operación del sistema completo.
- Programación a bajo nivel (lenguaje máquina y ensamblador).

## Niveles de abstracción para un computador:



# 1.1 Niveles de descripción de un computador

Arquitectura

Diseño

Introducción

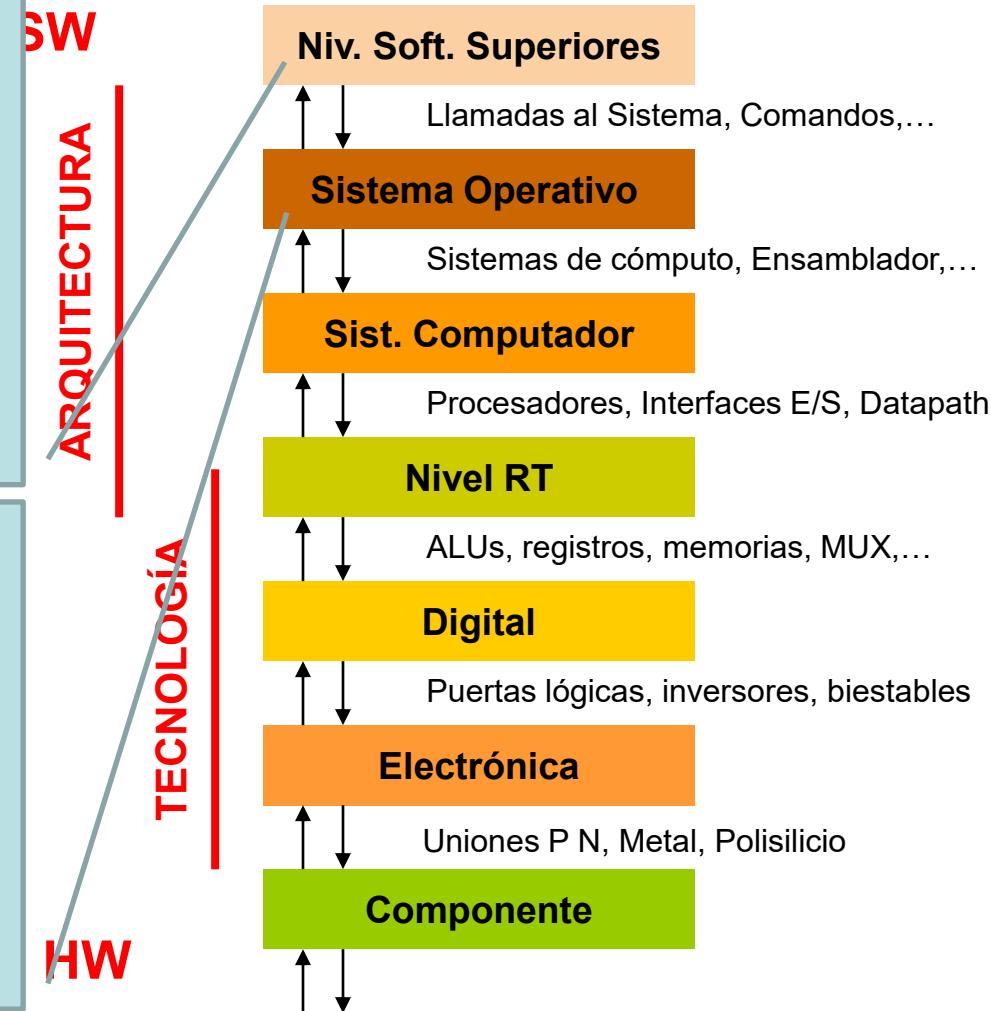
## Niveles de abstracción para un computador:

### Niveles superiores

- Niveles software: compiladores, programas escritos en lenguajes de alto nivel, etc.
- Realización de programas y compiladores eficientes requieren el conocimiento de la arquitectura del computador → incremento de prestaciones.

### Sistema operativo

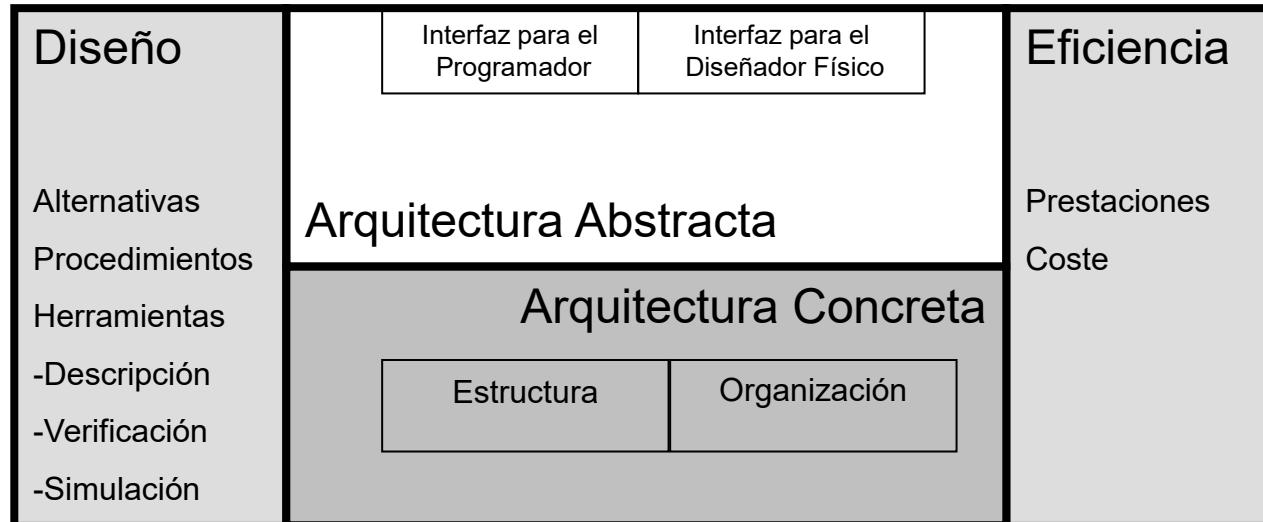
- Interfaz entre hardware y software.
- Encargado de facilitar el uso eficiente de los recursos hardware por parte de los usuarios y de los programas de aplicación.



# 1.1 Niveles de descripción de un computador

Arquitectura

Diseño



Introducción

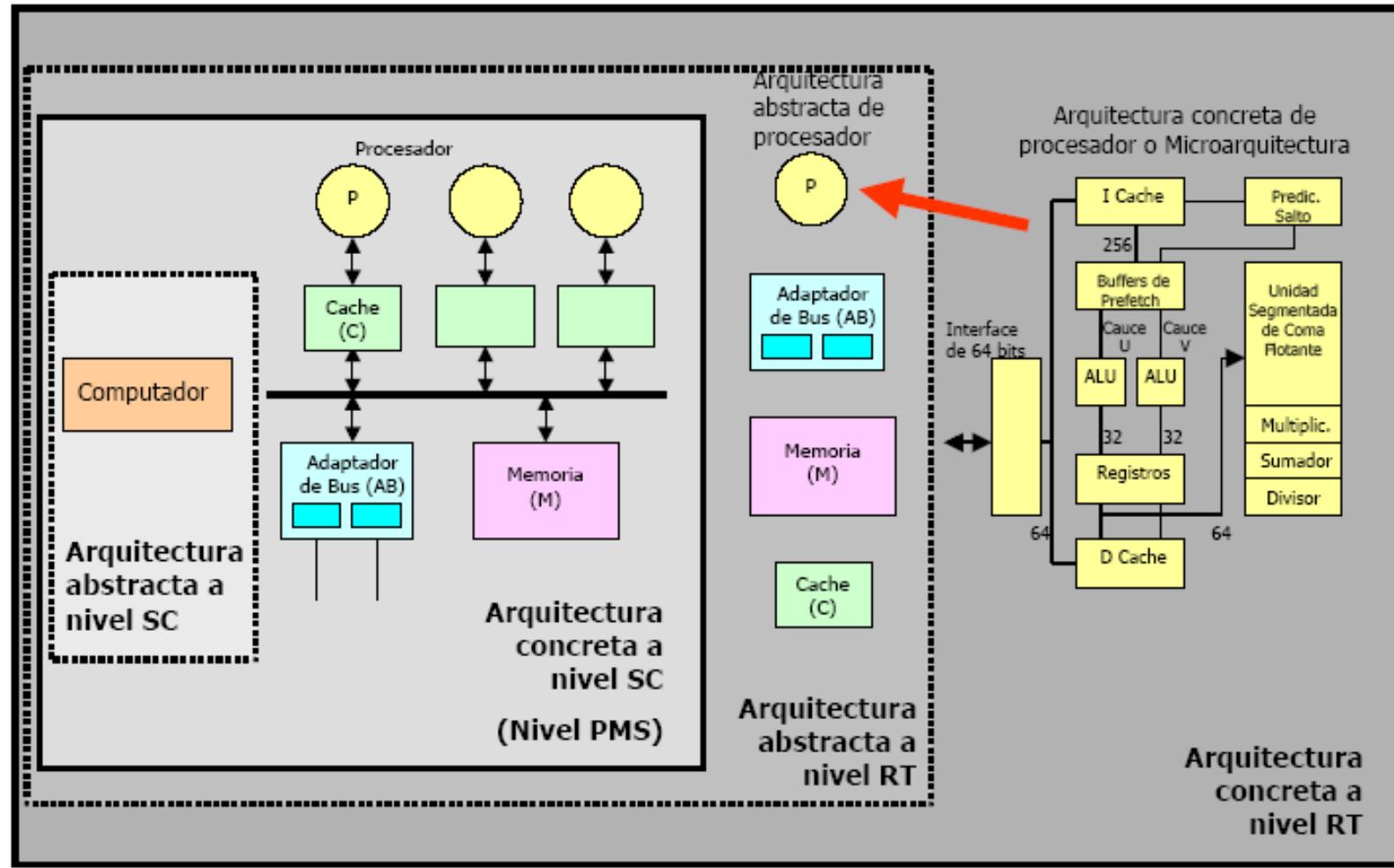
# 1.2 Definición de arquitectura

Arquitectura

Diseño

Introducción

## Niveles que abarca la Arquitectura



## 1.2 Definición de arquitectura

Arquitectura

Diseño

- ➊ **Definición de arquitectura** “Conjunto de instrucciones, recursos y características del procesador que son visibles al software que se ejecuta en el mismo. Por tanto, la arquitectura determina el software que el procesador puede ejecutar directamente, y esencialmente define las especificaciones a las que debe ajustarse la microarquitectura” [Ortega, 2005]

Introducción

## 1.2 Definición de arquitectura

Arquitectura

Diseño

- ➊ **Definición de microarquitectura:** “Conjunto de recursos y métodos utilizados para satisfacer las especificaciones que establece la arquitectura. El término incluye tanto la forma en que se organizan los recursos como las técnicas utilizadas para alcanzar los objetivos de costes y prestaciones planteados. La microarquitectura define las especificaciones para la implementación lógica”  
[Ortega, 2005]

Introducción

# 1.2 Definición de arquitectura

Arquitectura

Diseño

Introducción

## Ámbito de la arquitectura

### Arquitectura a nivel lenguaje máquina

#### Organización

alto nivel del diseño de un computador

Implementación

#### Hardware

componentes específicos de una máquina

Repertorio de instrucciones

Sistema de memoria, estructura del Bus, diseño interno de la CPU...

Diseño lógico detallado, la tecnología de encapsulamiento...

# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

Introducción

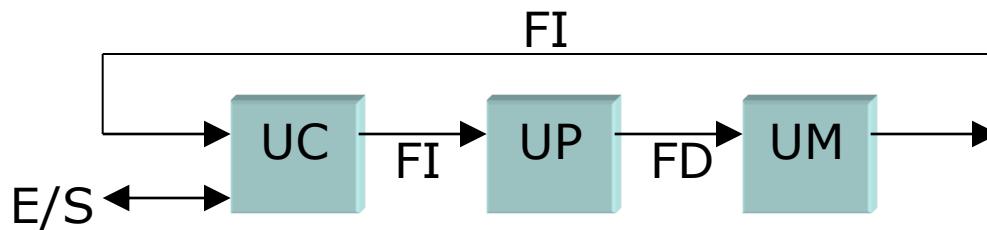
- Como toda clasificación, una clasificación (o taxonomía) de arquitecturas persigue dividir el conjunto de los computadores en una serie de clases de forma que, si se sabe la clase a la que pertenece un computador, automáticamente se conocen una serie de características interesantes del mismo
- La clasificación más extendida “**Taxonomía de Flynn**”:
  - SISD
  - SIMD
  - MISD
  - MIMD

# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

**Computadores SISD:** un único flujo de instrucciones procesa operandos y genera resultados, definiendo un único flujo de datos.



```
for i=1 to 4 do  
Begin  
    C[i] = A[i]+B[i];  
    F[i] = D[i]-E[i];  
    G[i] = K[i]*H[i];  
End;
```

Introducción

# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

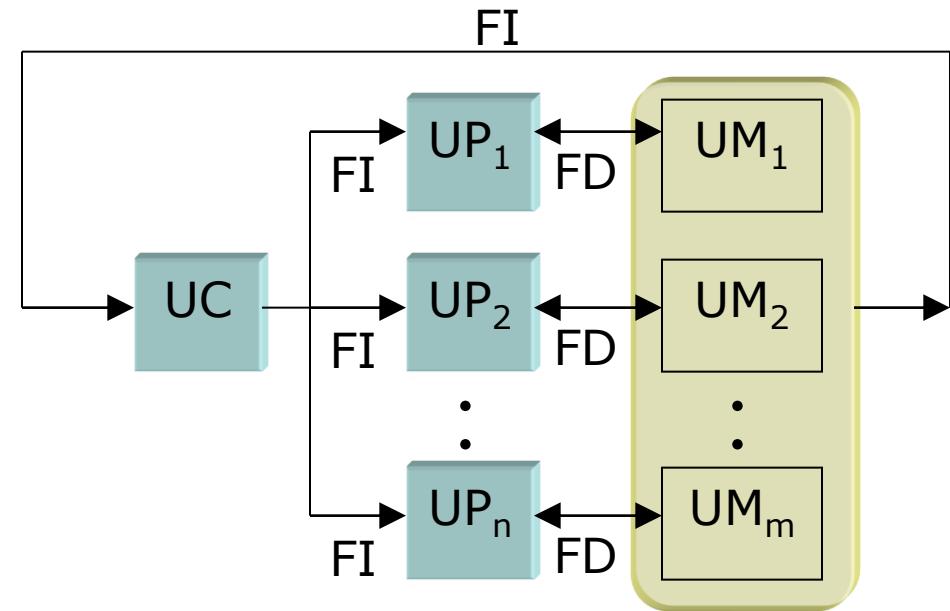
Introducción

**Computadores SIMD:** un único flujo de instrucciones procesa operandos y genera resultados, definiendo varios flujos de datos, dado que cada instrucción codifica realmente varias operaciones iguales, cada una actuando sobre operadores distintos.

```
for all Epi(i=1 to 4) do
Begin
    C[i] = A[i]+B[i];
    F[i] = D[i]-E[i];
    G[i] = K[i]*H[i];
End;
```

Procesadores matriciales

**ADDV C,A,B**  
**SUBV F,D,E**  
**MULV G,K,H**



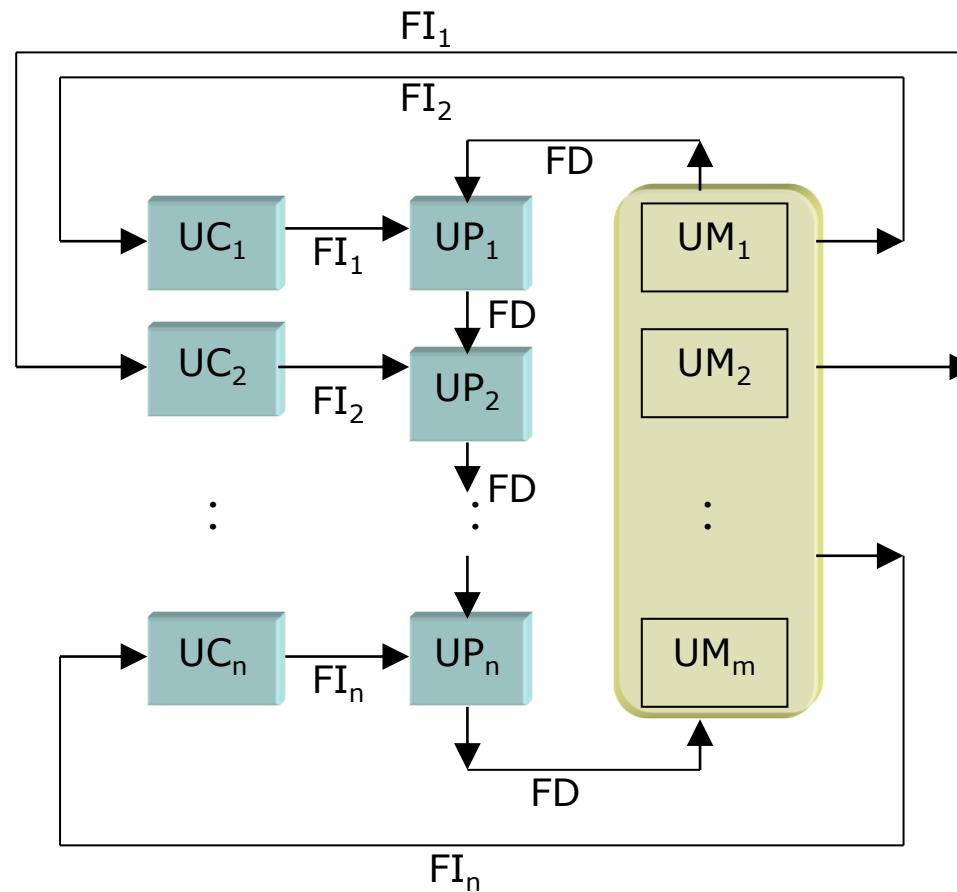
# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

Introducción

**Computadores MISD:** se ejecutan varios flujos distintos de instrucciones (MI) aunque todos actúan sobre el mismo flujo de datos. Actualmente no existen computadores que funcionen bajo este esquema



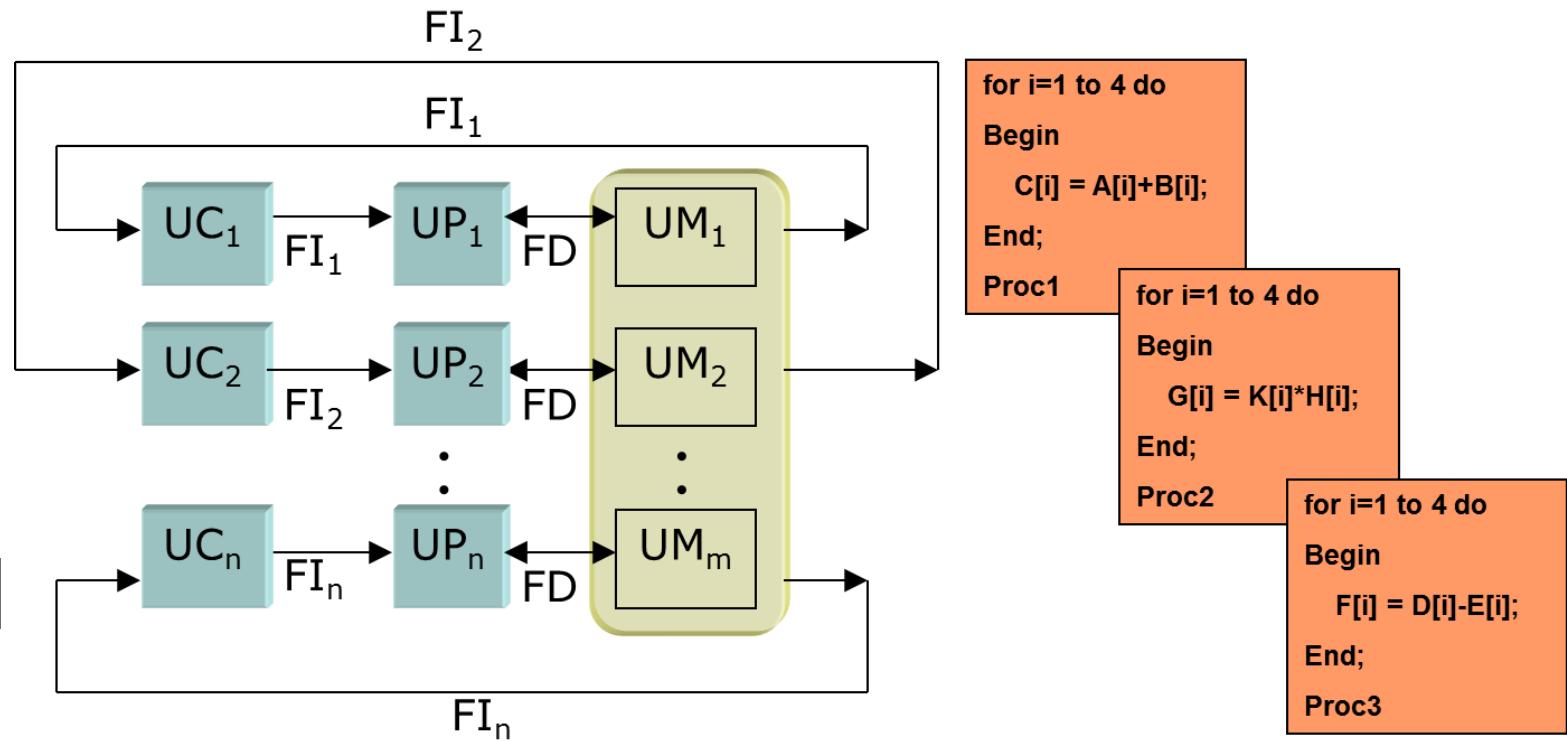
# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

Introducción

**Computadores MIMD:** el computador ejecuta varias secuencias o flujos distintos de instrucciones, y cada uno de ellos procesa operandos y genera resultados definiendo un único flujo de instrucciones, de forma que existen también varios flujos de datos uno por cada flujo de instrucciones.



# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

Introducción

**Paralelismo de datos:** La misma función, instrucción, etc. se ejecuta en paralelo pero en cada una de esas ejecuciones se aplica sobre un conjunto de datos distinto

# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

Introducción

**Parallelismo funcional:** Varias funciones, tareas, instrucciones, etc. (iguales o distintas) se ejecutan en paralelo. Se distinguen los siguientes niveles (según el tipo de entidades funcionales que se ejecutan en paralelo):

- **Nivel de instrucción (ILP)** – se ejecutan en paralelo las instrucciones de un programa. Granularidad fina.
- **Nivel de bucle o hebra (Thread)** – se ejecutan en paralelo distintas iteraciones de un bucle o secuencias de instrucciones de un programa. Granularidad fina/media.
- **Nivel de procedimiento (Proceso)** – los distintos procedimientos que constituyen un programa se ejecutan simultáneamente. Granularidad media.
- **Nivel de programa** – la plataforma ejecuta en paralelo programas diferentes que pueden corresponder, o no, a una misma aplicación. Granularidad gruesa.

# 1.3 Clasificación de arquitecturas

Arquitectura

Diseño

Introducción

## Tipos de computadores

- **Dispositivos móviles personales**
  - Teléfonos móviles, tablets, ... Coste y eficiencia energética
- **Ordenadores sobremesa**
  - Precio-rendimiento
- **Servidores**
  - Disponibilidad, escalabilidad, rendimiento
- **Clusters**
  - LANs de sobremesas y servidores actuando como un gran computador. SaaS: búsquedas, redes sociales, video compartido, juegos multiusuario
- **Embebidos**
  - Presentes en máquinas: microondas, lavadoras, impresoras, switches, coches ... Amplio espectro coste rendimiento

## 2. Diseño de computadores

Arquitectura

Diseño

Introducción

### ◆ **Tarea de diseño**

### ◆ **Propuesta conjunta ACM-IEEE**

- ◆ Informática basada en tres paradigmas: teoría, abstracción y diseño (no prima ninguno sobre el otro)

### ◆ **Teoría: Fuerte base matemática. Ciencias formales**

- ◆ Definición
- ◆ Teorema
- ◆ Demostración
- ◆ Interpretación

## 2. Diseño de computadores

Arquitectura

Diseño

### ◆ **Tarea de diseño**

### ◆ **Propuesta conjunta ACM-IEEE**

- ◆ Informática basada en tres paradigmas: teoría, abstracción y diseño (no prima ninguno sobre el otro)

### ◆ **Abstracción: Ciencias experimentales. (Física, química)**

- ◆ Hipótesis
- ◆ Construcción de un modelo y realización de predicciones
- ◆ Diseño de experimentos y recogida de resultados
- ◆ Análisis de resultados

Introducción

## 2. Diseño de computadores

Arquitectura

Diseño

Introducción

### ◆ Tarea de diseño

### ◆ Propuesta conjunta ACM-IEEE

- ◆ Informática basada en tres paradigmas: teoría, abstracción y diseño (no prima ninguno sobre el otro)

### ◆ Diseño: Ingenierías. Ciencias aplicadas

- ◆ Establecer requerimientos
- ◆ Especificar
- ◆ Realización del sistema
- ◆ Prueba del sistema

## 2.1 El proceso de diseño de computadores

Arquitectura

Diseño

Introducción

- ➊ **Establecer requerimientos funcionales**
- ➋ **Especificar el sistema**
- ➌ **Realización del sistema**
- ➍ **Prueba del sistema**

## 2.1 El proceso de diseño de computadores

Arquitectura

Diseño

### a. Establecer requerimientos funcionales y especificar

- ➊ **Funcionalidades inspiradas por el mercado y el software de aplicación que determinan características específicas del sistema**
- ➋ **Especificación en base a criterios de coste, rendimiento, consumo y disponibilidad para el mercado pensado**

Introducción

# 2.1 El proceso de diseño de computadores

Arquitectura

Diseño

Introducción

## a. Establecer requerimientos funcionales y especificar

Requerimientos funcionales	Características típicas requeridas o soportadas
<b>Área de aplicación</b> <ul style="list-style-type: none"><li>• Móviles</li><li>• PCs</li><li>• Servidores</li><li>• Clusters/Warehouse-Scale Computers</li><li>• Computación embebida</li></ul>	<b>Objetivo del computador</b> <ul style="list-style-type: none"><li>• Rendimiento en tiempo real para muchas tareas, incluyendo gráficos, video y audio; eficiencia energética.</li><li>• Rendimiento equilibrado para muchas tareas, incluyendo gráficos, video, y audio.</li><li>• Soporte para BB.DD. y transacciones; alta fiabilidad y disponibilidad; escalabilidad</li><li>• Alta productividad para tareas independientes; corrección de errores en memoria; proporcionalidad energética</li><li>• A menudo requiere soporte especial para video y audio (y otras extensiones específicas de aplicaciones); limitaciones en consumo y se puede requerir control de potencia; limitaciones de tiempo real.</li></ul>
<b>Nivel de compatibilidad software</b> En lenguaje de programación Código binario compatible	<b>Determina la cantidad de software existente para la máquina</b> Más flexible para el diseñador, necesita nuevo compilador La arquitectura está completamente definida (poca flexibilidad), pero no necesita invertir en software ni en portar programas
<b>Requerimientos del S.O.</b> Tamaño del espacio de direcciones Gestión de memoria Cambio de contexto Interrupciones Protección	<b>Características necesarias para soportar el S.O. requerido</b> Muy importante, puede limitar aplicaciones Para S.O. modernos; puede ser plana, paginada, segmentada. Requerido para interrumpir y recomenzar un programa Tipos de soporte impactan en el diseño hardware y S.O. Diferentes S.O. y necesidades de aplicación: protección de páginas frente a protección de segmentos.
<b>Estándares</b> Punto flotante Bus E/S Sistema operativo Redes Lenguajes de programación	<b>Ciertos estándares pueden ser requeridos por el mercado</b> Formato y aritmética: IEEE 754, aritmética especial para gráficos o procesamiento de señal Dispositivos E/S: Serial ATA, Serial Attach SCSI, PCI Express UNIX, Windows, Linux Distintas redes: Ethernet, Lenguajes (ANSI C, C++, Java, Fortran) afectan al repertorio de instrucciones.

## 2.1 El proceso de diseño de computadores

Arquitectura

Diseño

### b. Decisiones de implementación

**¿Como se implementa mejor una funcionalidad requerida?**

**La decisión de implementación software o hardware**

**◆ Ventajas implementación software**

- ◆ El bajo coste errores
- ◆ Facilidad de diseño
- ◆ Actualización simple

**◆ Ventajas implementación hardware**

- ◆ Rendimiento

Introducción

## 2.1 El proceso de diseño de computadores

Arquitectura

Diseño

### c. Consideración de las nuevas tendencias

#### ➊ **Diseñador consciente de tendencias:**

- ➌ Utilización del computador
- ➌ Tecnología de computadores
- ➌ Una arquitectura a nivel lenguaje máquina con éxito puede durar decenas de años (núcleo de la IBM 360 desde 1964, 50 años)

Introducción

# 2.1 El proceso de diseño de computadores

Arquitectura

Diseño

Introducción

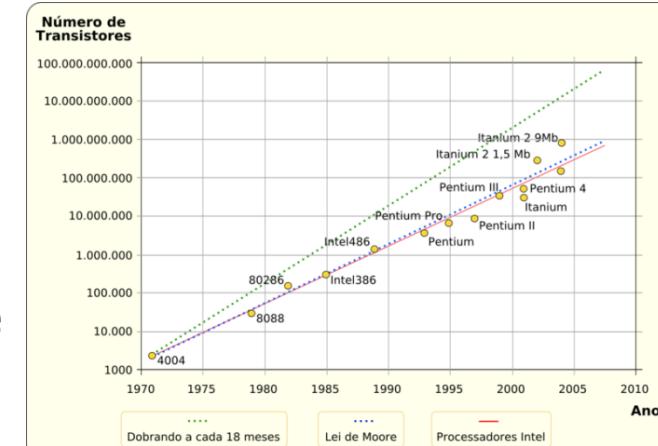
## c. Consideración de las nuevas tendencias

### ◆ Tendencias en tecnologías hardware [Agarwall, 00]

Tecnología	Tendencias de rendimiento y densidad
Tecnología de CI	El número de transistores en un chip aumenta aproximadamente el 35% por año, x4 en 4 años. La velocidad de los dispositivos aumenta casi a esa rapidez. [Agarwall, 00] disminución tasa crecimiento a 12% anual (consecuencia de procesos de 0,035 micras previstos para 2014)
DRAM semiconductor	La densidad aumenta en un 60% por año, cuadruplicándose en tres años. 2011 25-40% x2 cada 2-3 años [Kim, 2005] La duración del ciclo ha mejorado muy lentamente, decreciendo aproximadamente una tercera parte en diez años.
Tecnología de almacenamiento secundario	La densidad del disco magnético aumenta desde 2004 40% año x2 cada 3 años. El tiempo de acceso ha mejorado un tercio en diez años. Crecimiento de los discos SSD (NAND SLC, MLC y TLC)

### ◆ Ley de Moore

- ◆ Las velocidades de cómputo y las densidades de almacenamiento se duplican cada 18 meses



Arquitectura

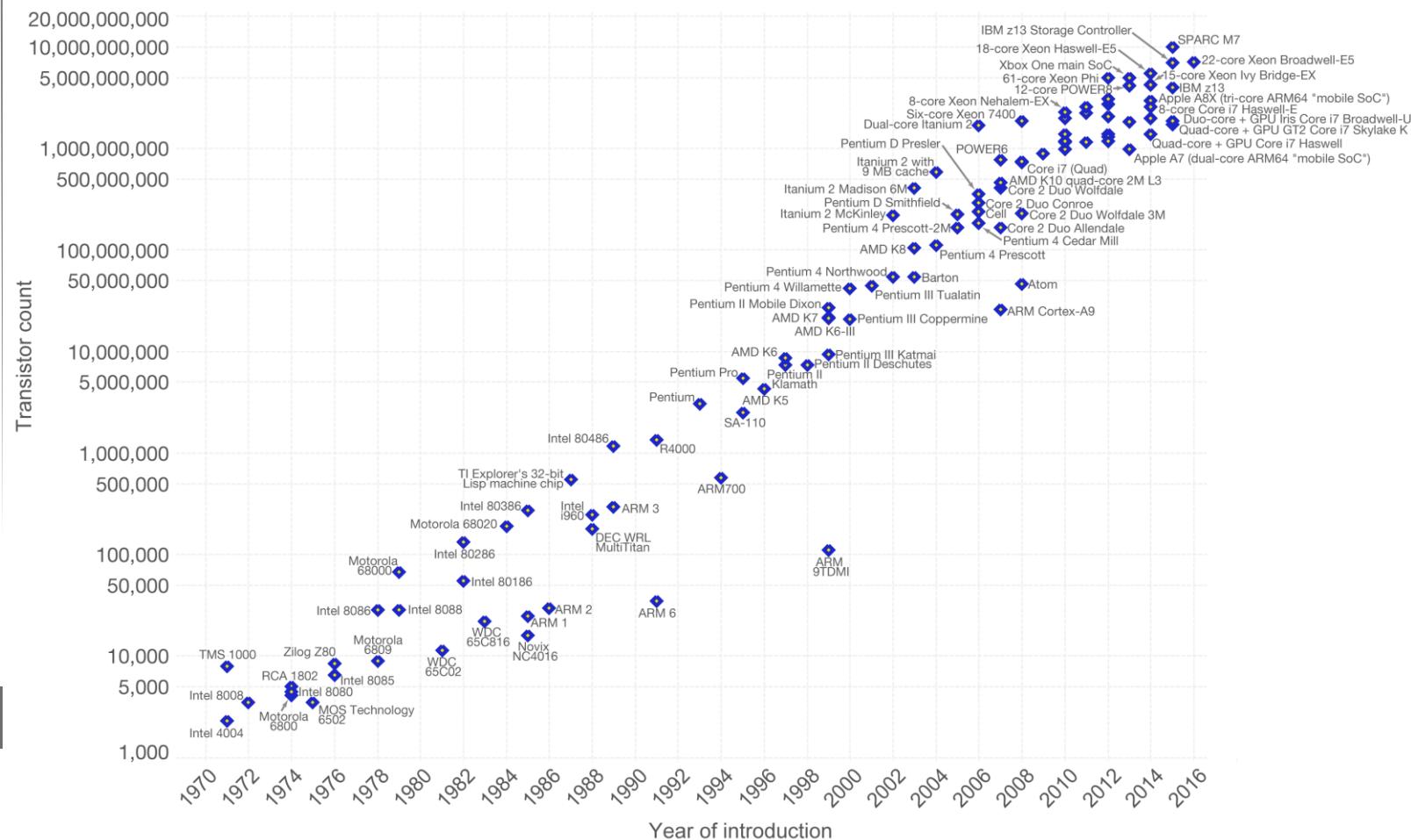
Diseño

Introducción

Our World  
in Data

## Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia ([https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count))

The data visualization is available at [OurWorldInData.org](http://OurWorldInData.org). There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

## 2.1 El proceso de diseño de computadores

Arquitectura

Diseño

### c. Consideración de las nuevas tendencias

#### ➊ Tendencias software

- ➊ Creciente cantidad de memoria utilizada por los programas y sus datos
- ➋ Sustitución del lenguaje ensamblador por los lenguajes de alto nivel.
- ➌ Reorientación de las arquitecturas hacia el soporte de los compiladores

Introducción

## 2.2 Principios de diseño de computadores

Arquitectura

Diseño

Introducción

### a. Acelerar el caso común

#### ◆ Favorecer el caso frecuente

- ◆ **Ejemplo:** El desbordamiento de la suma es poco frecuentemente.
- ◆ El principio indicaría la optimización del caso sin desbordamiento
- ◆ La cuantificación de este principio se conoce como la ley de Amdahl

#### ◆ Ley de Amdahl

- ◆ Define la ganancia de rendimiento o aceleración que puede obtenerse al mejorar alguna característica de un computador
- ◆ La mejora obtenida en el rendimiento al utilizar algún modo de ejecución más rápido está limitada por la fracción de tiempo en que se puede utilizar ese modo más rápido

$$\text{Aceleración Rendimiento} = \frac{\text{Rendimiento con mejora}}{\text{Rendimiento sin mejora}} = \frac{\text{Tiempo ejecución sin mejora}}{\text{Tiempo ejecución con mejora}}$$

## 2.2 Principios de diseño de computadores

Arquitectura

Diseño

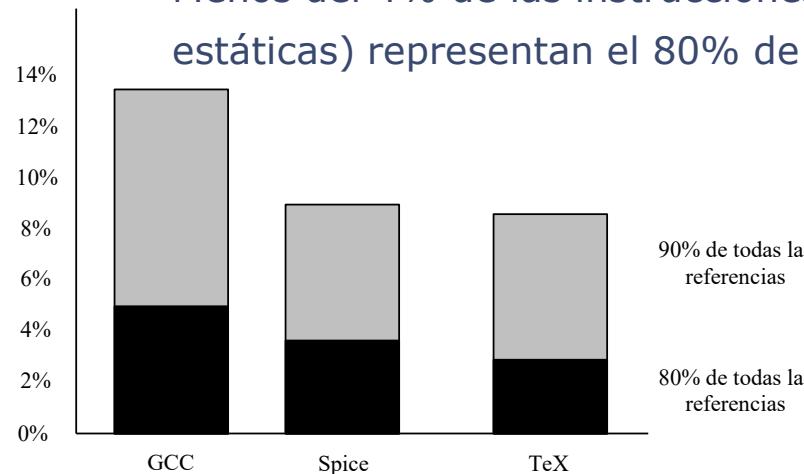
Introducción

### b. Ley de rendimientos decrecientes

- La mejora incremental en la aceleración conseguida por una mejora adicional en el rendimiento de una parte del cálculo disminuye tal y como se van añadiendo mejoras.

### c. Localidad de referencia

- Tendencia de los programas a reutilizar los datos e instrucciones usados recientemente. Los programas suelen emplear el 90% de su tiempo de ejecución en el 10% del código.
- Menos del 4% de las instrucciones del programa Spice (instrucciones estáticas) representan el 80% de las instrucciones dinámicas.



**Localidad temporal:** Los elementos accedidos recientemente probablemente serán accedidos en un futuro próximo.

**Localidad espacial:** Los elementos cuyas direcciones son próximas tienden a ser referenciados juntos en el tiempo.

# Tema 2

# Análisis del rendimiento

Arquitectura de los Computadores

# Tema 2. Análisis del rendimiento

## ◆ Objetivos

- ◆ Entender el concepto de rendimiento, la evolución del rendimiento en los computadores en los últimos años y su relación con el coste
- ◆ Saber cuantificar la ganancia de rendimiento o aceleración que puede obtenerse al mejorar alguna característica de un computador
- ◆ Mostrar al alumno distintas métricas para evaluar el rendimiento de una arquitectura, observando la relación que existe entre ellas.
- ◆ Adquirir conciencia de la necesidad de establecer métricas para llevar a cabo procesos de evaluación y comparación objetiva y contrastada de sistemas computacionales

# Tema 2. Análisis del rendimiento

## ◆ Contenido

### ◆ 2.1. Rendimiento. Concepto y definiciones

- ◆ Concepto de rendimiento
- ◆ Ley de Amdhal
- ◆ Relación entre rendimiento y coste

### ◆ 2.2. Evaluación del rendimiento

- ◆ Medidas del rendimiento
- ◆ Programas para evaluar el rendimiento
- ◆ Formulación de resultados

## **2.1. Rendimiento. Concepto y definiciones**

---

**Tema 2 Análisis del rendimiento**

**Arquitectura de los Computadores**

## 2.1 Rendimiento. Concepto y definiciones

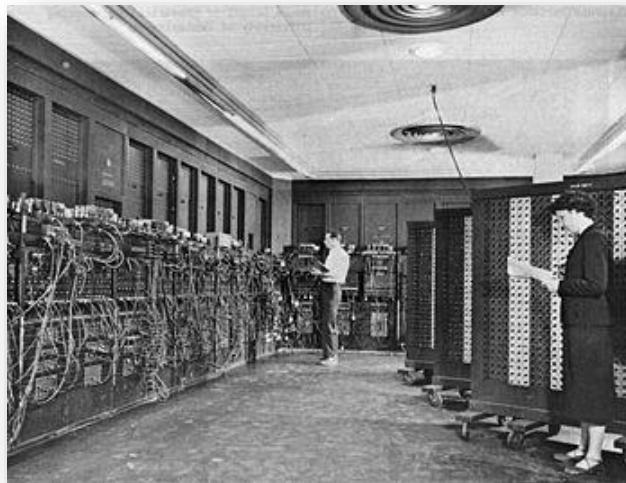
Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Evolución del rendimiento



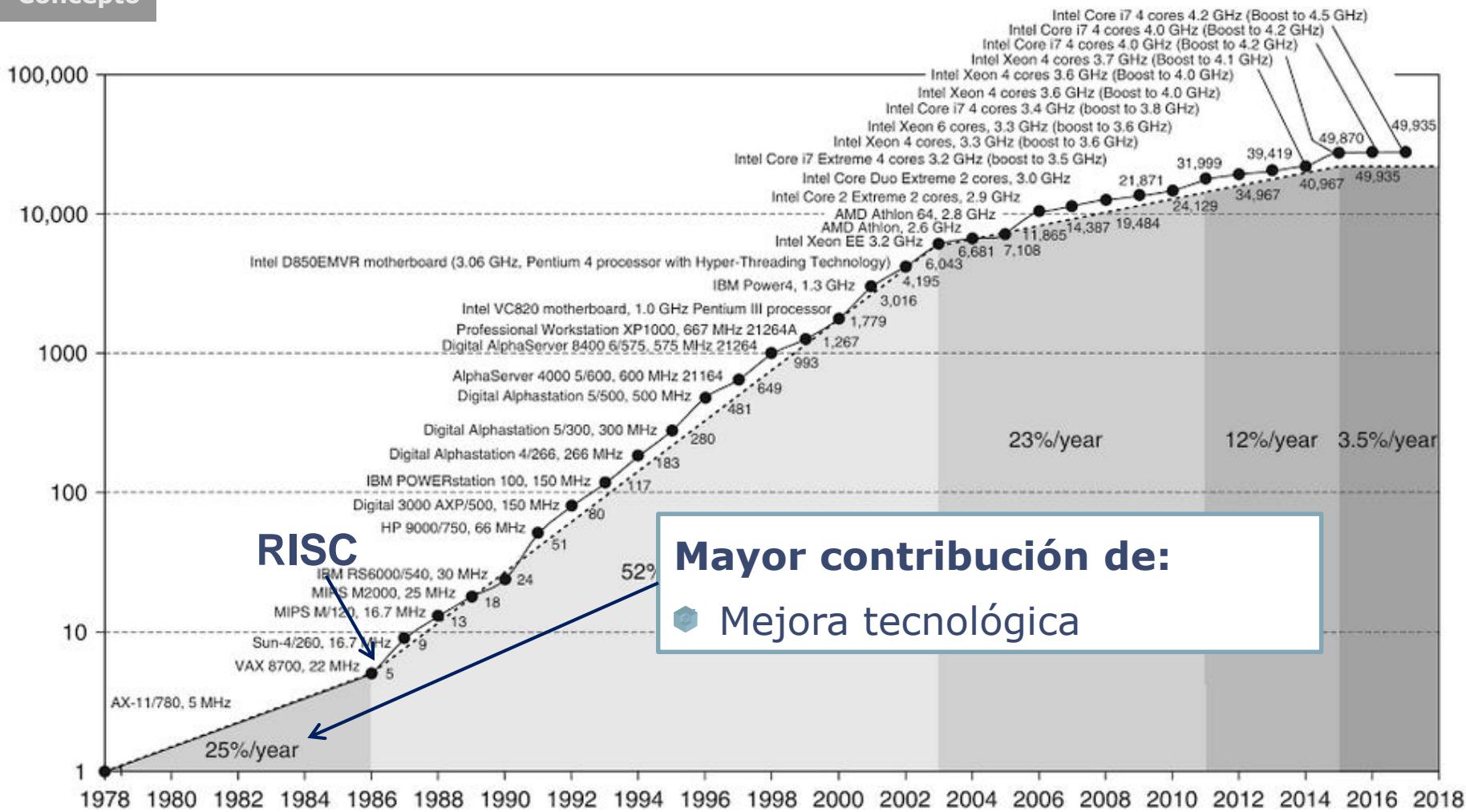
- ➊ La tecnología de computadores ha tenido un increíble progreso en los últimos 70 años
  - ➋ Por menos de 500€ es posible comprar un portátil que tiene más rendimiento, memoria y capacidad de disco que un computador que costaba 50 millones de € en 1993.
- 
- ➌ Esta rápida mejora se debe fundamentalmente a:
    - ➍ **Avances en la tecnología** (casi constante) usada para construir computadores
      - ➎ Tamaño de los elementos en el chip (feature size), velocidad del reloj
    - ➏ **Innovaciones en el diseño** (menos consistentes)
      - ➐ Compiladores de lenguajes de alto nivel, UNIX
      - ➑ Provocado por arquitecturas RISC

## 2.1 Rendimiento. Concepto y definiciones

### Concepto

#### Evolución del rendimiento

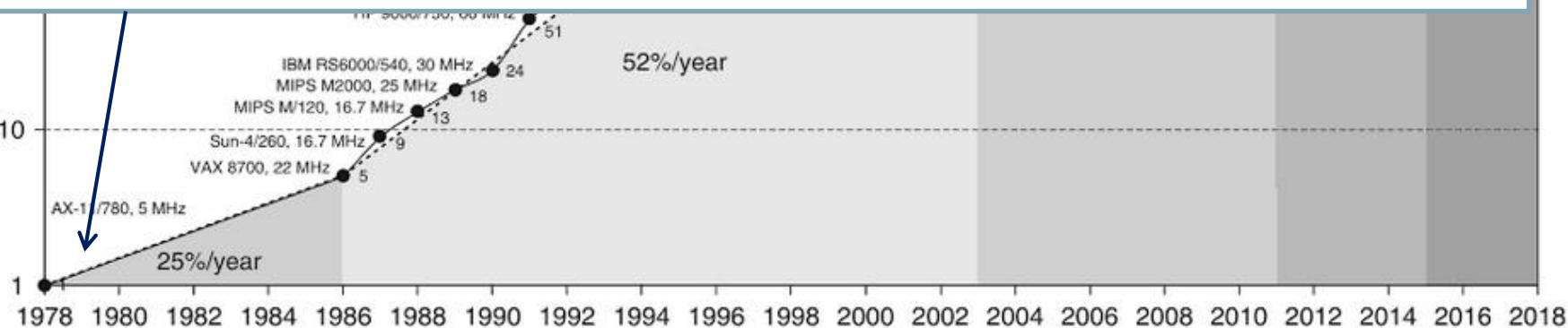
Performance (vs. VAX-11/780)



## Aparición del microprocesador ( $\mu$ P) (finales 1970)

- ◆ Capacidad de dirigir los avances en la tecnología de circuitos integrados
  - ◆ Tasa más alta de mejora del rendimiento (35% anual)
- ◆ Ventajas en el coste debido a la producción masiva de  $\mu$ Ps
  - ◆ Aumenta el número de computadores basados en  $\mu$ Ps.
- ◆ Además, dos cambios significantes:
  - ◆ Eliminación virtual de la programación en lenguaje ensamblador
    - ◆ Reduce la necesidad de la compatibilidad en el código objeto
  - ◆ Aparición de sistemas operativos estandarizados como UNIX
    - ◆ Reducen el coste y el riesgo en la aparición de una nueva arquitectura

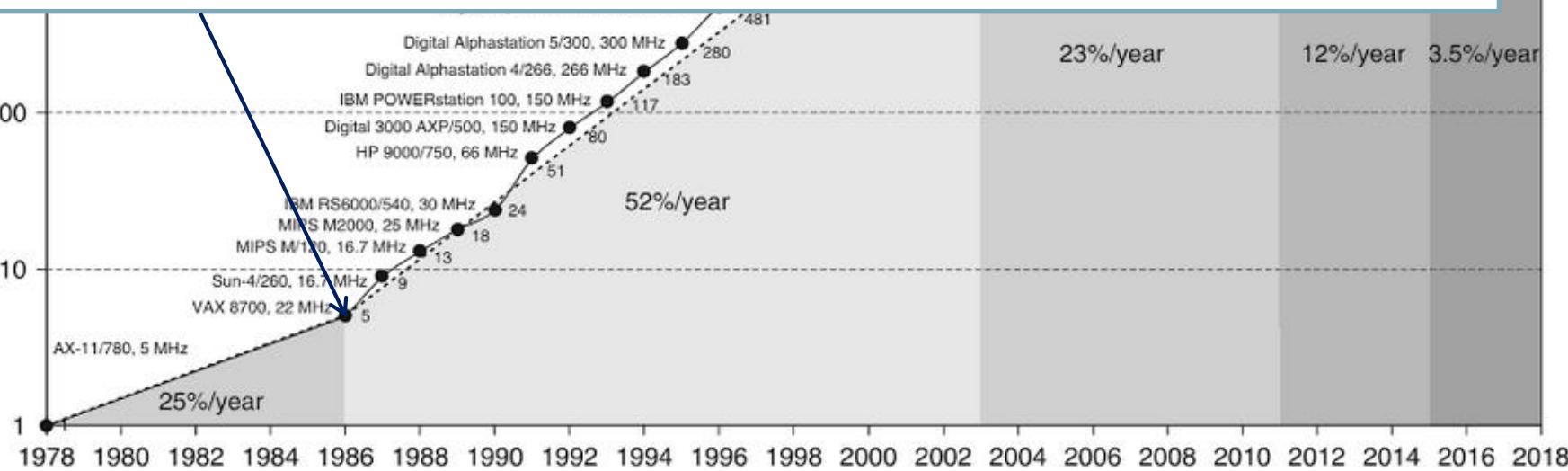
Performance (vs. VAX-11/780)



## RISC (Principios 1980)

- Cambios anteriores permiten el desarrollo de forma satisfactoria de un nuevo conjunto de arquitecturas con instrucciones más simples: arquitecturas RISC (Reduced Instruction Set Computer).
- Los diseñadores de máquinas RISC se centraron en dos técnicas clave para la mejora del rendimiento:
  - La explotación del paralelismo a nivel de instrucción
  - El uso de cachés
- El aumento del rendimiento forzó a las arquitecturas previas a mantener el ritmo o a desaparecer

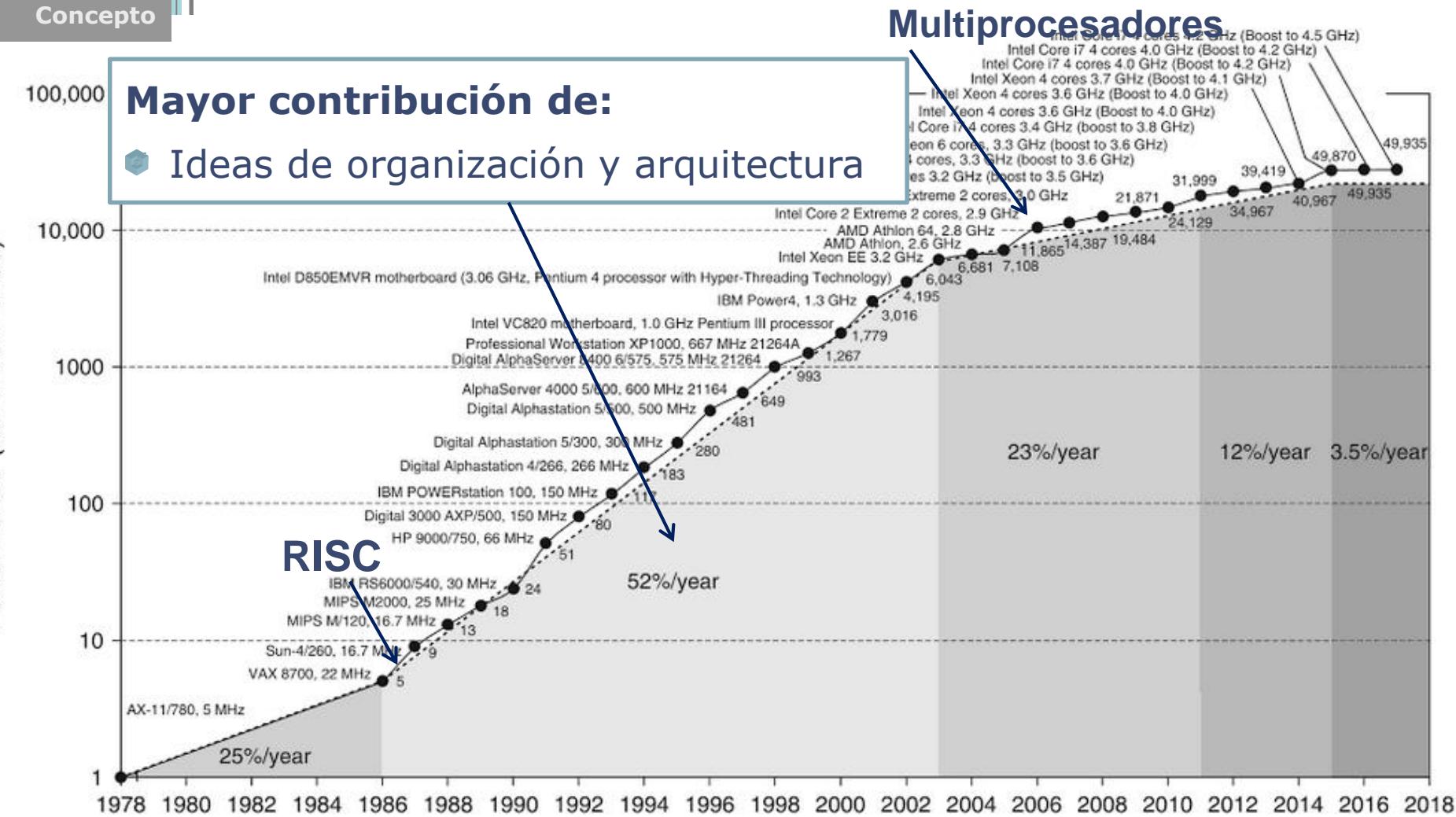
Performance (vs. VAX-11/780)



## 2.1 Rendimiento. Concepto y definiciones

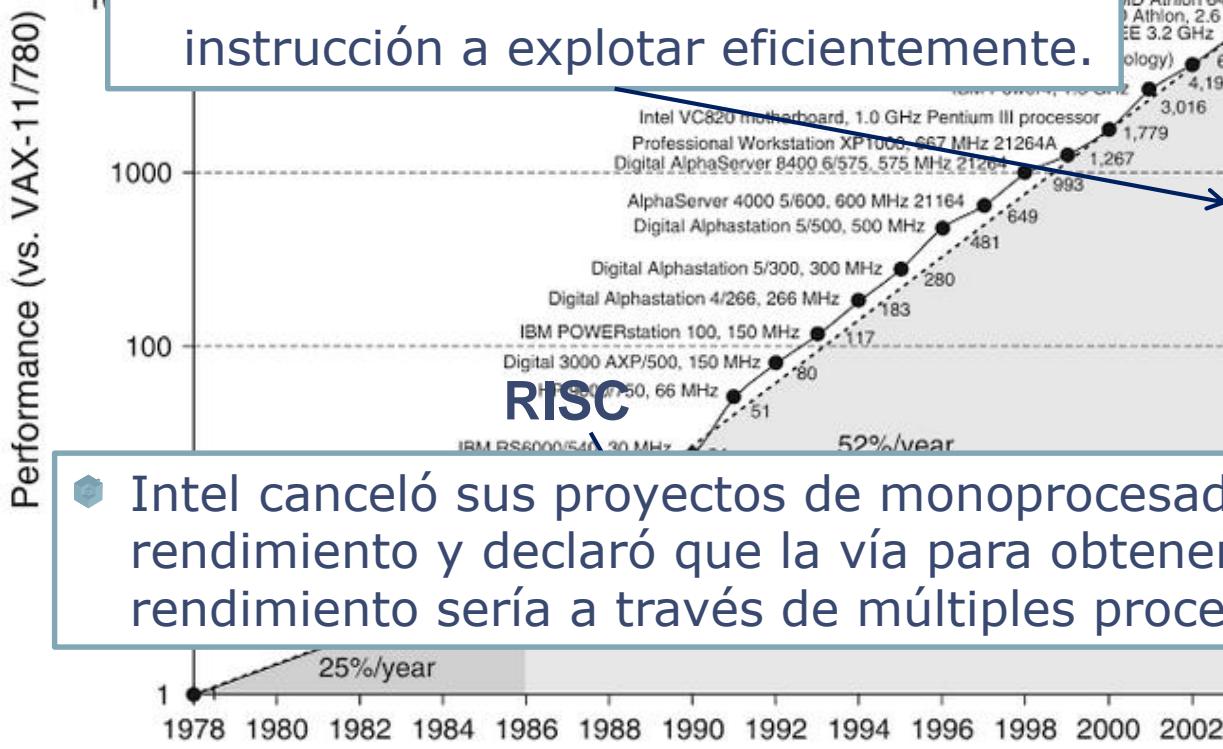
### Concepto

### Evolución del rendimiento

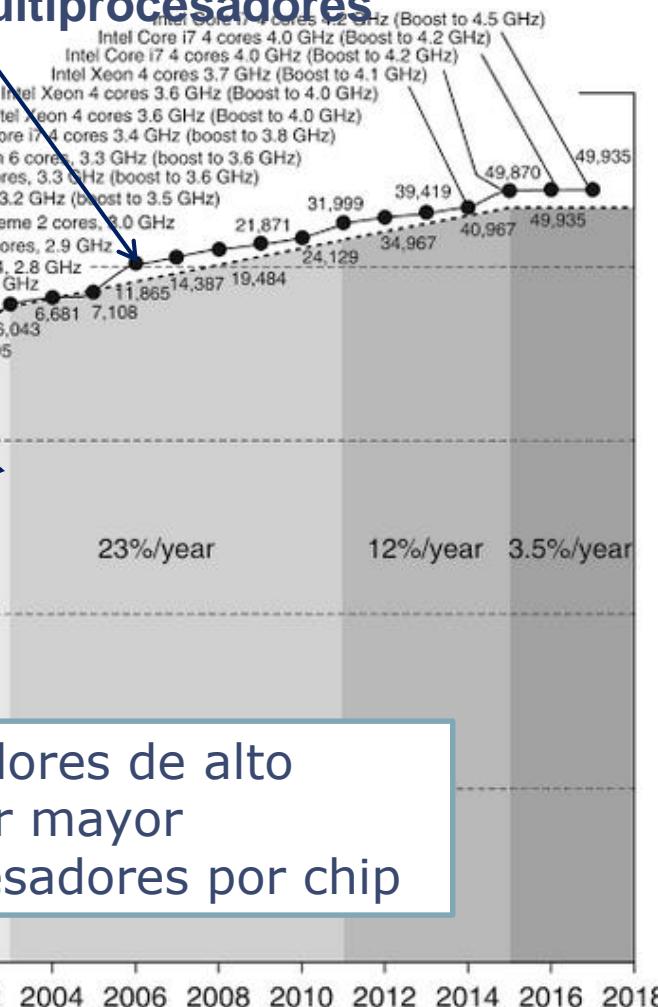


# Cae la mejora del rendimiento para un sólo procesador

- ➊ Máxima disipación de potencia de chips refrigerados por aire
  - ➋ Falta de más paralelismo a nivel de instrucción a explotar eficientemente.



## Multiprocesadores



- Intel canceló sus proyectos de monoprocesadores de alto rendimiento y declaró que la vía para obtener mayor rendimiento sería a través de múltiples procesadores por chip

## 2.1 Rendimiento. Concepto y definiciones

### El rendimiento se estanca para un solo procesador

- % de mejora ya no es significativo (Ley de Amdahl)

### ultiprocesadores

Intel Core i7 4 cores 4.2 GHz (Boost to 4.5 GHz)  
core i7 4 cores 4.0 GHz (Boost to 4.2 GHz)  
i7 4 cores 4.0 GHz (Boost to 4.2 GHz)  
cores 3.7 GHz (Boost to 4.1 GHz)  
3.6 GHz (Boost to 4.0 GHz)  
6 GHz (Boost to 4.8 GHz)  
GHz (boost to 3.8 GHz)  
(boost to 3.6 GHz)  
st to 3.6 GHz)  
3.5 GHz)

Intel Core Duo Extreme 2 cores, 3.0 GHz  
Intel Core 2 Extreme 2 cores, 2.9 GHz

AMD Athlon 64, 2.8 GHz  
AMD Athlon, 2.6 GHz

Intel Xeon EE 3.2 GHz  
6,681 7,108

IBM Power4, 1.3 GHz  
4,195

3,016 1,779

11,865 14,387 19,484

21,871 31,999 39,419

24,129 34,967 40,967

49,870 49,935 49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

49,935

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Concepto de rendimiento

- ◆ Qué significa cuando decimos que un computador es más rápido que otro?:

- ◆ **Usuario:**

- ◆ Un computador es más rápido cuando un programa se ejecuta en **menos tiempo**
    - ◆ Interesado en reducir el **tiempo de respuesta/tiempo de ejecución**
      - ◆ Tiempo transcurrido entre el inicio y el final de un evento

- ◆ **Administrador de un Cluster:**

- ◆ Un computador es más rápido cuando completa **más transacciones por hora** (Google, Amazon...)
    - ◆ Interesado en aumentar la **productividad/throughput**
      - ◆ Cantidad total de trabajo realizado en un tiempo determinado

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

### Ejemplo:

- ➊ ¿Las siguientes mejoras en el rendimiento afectan a la productividad, al tiempo de respuesta o a ambas cosas?
  - ➌ 1. Ciclo de reloj más rápido
  - ➌ 2. Múltiples procesadores para tareas separadas (sistema de reservas de una compañía aérea)
  - ➌ 3. Procesamiento paralelo de problemas científicos
  - ➌ **La disminución del tiempo de respuesta habitualmente mejora la productividad.**
    - ➌ 1 y 3 mejoran el tiempo de respuesta y en consecuencia la productividad. En el caso 2 no mejora el tiempo de respuesta pero si la productividad.
- ➋ La influencia de factores no determinísticos aconseja hablar de estas medidas de rendimiento con **distribuciones de probabilidad.**
  - ➌ Por ejemplo el tiempo de respuesta en un disco para una operación de entrada salida depende de:
    - ➌ Actividad del disco en el instante de petición.
    - ➌ Número de tareas intentando acceder al disco
  - ➌ Esta situación aconseja hablar de tiempo medio de respuesta de un acceso al disco

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Concepto de rendimiento

- ➊ El tiempo es la medida más fiable del rendimiento
- ➋ El tiempo de ejecución de un programa se mide en segundos
- ➌ La relación entre el tiempo y el rendimiento es inversa
- ➍ El rendimiento se mide como una frecuencia de eventos por segundo

$$\text{Rendimiento} = \frac{1}{\text{tiempo}}$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Relación de rendimientos entre máquinas

"X es más rápida que Y"

$$t_{ex} < t_{ey}$$

"X es n % más rápida que Y"

$$t_{ex} \text{ n\%} < t_{ey}$$

### Porcentaje incremental

$$\text{tiempo ejecución}_X + \frac{n}{100} \text{ tiempo ejecución}_X = \text{tiempo ejecución}_Y$$

### Aceleración

$$\frac{\text{tiempo ejecución}_Y}{\text{tiempo ejecución}_X} = 1 + \frac{n}{100}$$

# 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

## Relación de rendimientos entre máquinas

### ◆ En términos de rendimiento

$$1 + \frac{n}{100} = \frac{\text{tiempo ejecución}_Y}{\text{tiempo ejecución}_X} = \frac{1}{\frac{\text{Rendimiento}_Y}{\text{Rendimiento}_X}} = \frac{\text{Rendimiento}_X}{\text{Rendimiento}_Y}$$

### Despejando

$$\frac{n}{100} = \frac{\text{Rendimiento}_X}{\text{Rendimiento}_Y} - 1 = \frac{\text{Rendimiento}_X - \text{Rendimiento}_Y}{\text{Rendimiento}_Y}$$

$$n = 100 \frac{\text{Rendimiento}_X - \text{Rendimiento}_Y}{\text{Rendimiento}_Y}$$

### Expresado con tiempos de ejecución

$$n = 100 \frac{\text{tiempo ejecución}_Y - \text{tiempo ejecución}_X}{\text{tiempo ejecución}_X}$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Relación de rendimientos entre máquinas

- ➊ Ejemplo: Si la máquina A ejecuta un programa en 10 segundos y la máquina B ejecuta un programa en 30 segundos. ¿Cuál de las siguientes afirmaciones es correcta?
  - ➎ A es el 30% más rápida que B
  - ➎ A es el 200% más rápida que B

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Relación de rendimientos entre máquinas

- ◆ "La productividad de X es el 30 por 100 superior que la de Y"

$$P=nt/t; P_x=1,3*P_y$$

**Ejemplo:** Si la máquina A ejecuta un programa en 10 segundos y la máquina B ejecuta el mismo programa en 15 segundos. ¿En qué porcentaje es la máquina A más rápida que la B?

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Relación de rendimientos entre máquinas

- ◆ "La productividad de X es el 30 por 100 superior que la de Y"

$$P=nt/t; P_x=1,3*P_y$$

**Ejemplo:** Si la máquina A ejecuta un programa en 10 segundos y la máquina B ejecuta el mismo programa en 15 segundos. ¿En qué porcentaje es la máquina A más rápida que la B?

$$n = 100 \frac{\text{tiempo ejecución}_B - \text{tiempo ejecución}_A}{\text{tiempo ejecución}_A}$$

$$n = 100 \frac{15 - 10}{10} = 50$$

"A es el 50% más rápida que B"

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ◆ **Ejercicio 1:** Se dan los tiempos de ejecución en segundos del benchmark linpack y de 10.000 iteraciones del benchmark Dhystone en dos modelos VAX y un procesador actual:

Modelo	Año	Tiempo linpack	Tiempo Dhystone
VAX- 11/780	1978	4,9	5,69
VAX 8600	1985	1,43	1,35
Intel Xeon 4 cores	2010	$2,24 \times 10^{-4}$	$2,74 \times 10^{-4}$

- ◆ A) ¿Cuantas veces es más rápido el 8600 que el 780 utilizando Linpack? ¿Que ocurre cuando se utiliza Dhystone?
- ◆ B) ¿Cuantas veces es más rápido el 8550 que el 8600 utilizando Linpack? ¿Que ocurre cuando se utiliza Dhystone?
- ◆ C) ¿Cuantas veces es más rápido el Xeon que el 780 utilizando Linpack? ¿Que ocurre cuando se utiliza Dhystone?
- ◆ D) ¿Cual es el crecimiento medio del rendimiento por año entre el 780 y el 8600 utilizando linpack? ¿Que ocurre cuando se utiliza Dhystone?
- ◆ E) ¿Cual es el crecimiento medio del rendimiento por año entre el 8600 y el 8550 utilizando linpack? ¿Que ocurre cuando se utiliza Dhystone?
- ◆ F) ¿Cual es el crecimiento medio del rendimiento por año entre el 780 y el Xeon utilizando linpack? ¿Que ocurre cuando se utiliza Dhystone?

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### ◆ a y d) Comparación de rendimientos para el VAX 780 y el VAX 8600 (Según Linpack)

$$te_{VAX780} = 4,9 \text{ s}$$

$$rend_{VAX780} = \frac{1}{te_{VAX780}} = 0,204$$

$$te_{VAX8600} = 1,43 \text{ s}$$

$$rend_{VAX8600} = \frac{1}{te_{VAX8600}} = 0,699$$

### ◆ Porcentaje de incremento del rendimiento entre las dos arquitecturas:

$$\text{porcentaje} = n = 100 \frac{te_{VAX780} - te_{VAX8600}}{te_{VAX8600}} = 100 \frac{4,9 - 1,43}{1,43} = 243\%$$

$$\text{porcentaje} = n = 100 \frac{rend_{VAX8600} - rend_{VAX780}}{rend_{VAX780}} = 100 \frac{0,699 - 0,204}{0,204} = 243\%$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### ◆ Aceleración del rendimiento entre las dos arquitecturas

$$\text{aceleracion} = \left( 1 + \frac{n}{100} \right) = \frac{te_{VAX780}}{te_{VAX8600}} = \frac{4,9}{1,43} = 3,43 = \frac{\text{rend}_{VAX8600}}{\text{rend}_{VAX780}} = \frac{0,699}{0,204}$$

$$\text{rend}_{VAX8600} = 3,43 \cdot \text{rend}_{VAX780} \Rightarrow 0,699 = 3,43 \cdot 0,204$$

7 años

### ◆ Los incrementos anuales se aplican cada año sobre el anterior

$$\text{rend}_{an} = \Delta_{anual} \cdot \text{rend}_{an-1} \rightarrow \text{rend}_{a1} = \Delta_{anual} \cdot \text{rend}_{a0}$$

$$\text{rend}_{a2} = \Delta_{anual} \cdot \text{rend}_{a1} = (\Delta_{anual})^2 \text{rend}_{a0}$$

$$\text{rend}_{an} = \Delta_{anual} \cdot \text{rend}_{an-1} = (\Delta_{anual})^n \text{rend}_{a0}$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

◆ El incremento anual es:

$$\Delta_{anual} = \sqrt[n]{\frac{rend_{an}}{rend_{a0}}} = \sqrt[n]{\frac{te_{a0}}{te_{an}}}$$

◆ Si consideramos n=7

$$te_{a0} = te_{VAX780} \quad rend_{a0} = rend_{VAX780}$$

$$te_{a7} = te_{VAX8600} \quad rend_{a7} = rend_{VAX8600}$$

$$\Delta_{anual} = \sqrt[7]{\frac{rend_{a7}}{rend_{a0}}} = \sqrt[7]{\frac{te_{a0}}{te_{a7}}} = \sqrt[7]{\frac{rend_{VAX8600}}{rend_{VAX780}}} = \sqrt[7]{\frac{te_{VAX780}}{te_{VAX8600}}} = \sqrt[7]{3,43} = 1,193$$

$$aceleración = \left( 1 + \frac{n}{100} \right) = \Delta_{anual} \Leftrightarrow n = (\Delta_{anual} - 1) \cdot 100 = 19,3\%$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ **Ejercicio 2:** Hay dos equipos de diseño en dos compañías diferentes. La gestión de la compañía más pequeña y más agresiva pide un ciclo de diseño de dos años para sus productos. La gestión de la compañía más grande y menos agresiva apuesta por un ciclo de diseño de cuatro años. Supongamos que en el mercado de hoy día se demanda 1.5 veces el rendimiento de un Intel Xeon 4 cores según Linpack.
- ➋ ¿Cuales deberían ser los objetivos de rendimiento para cada producto, si las frecuencias de crecimiento necesarias son el 22% por año?
- ➌ Supongamos que las compañías acaban de empezar a utilizar chips de DRAM de 2048 Megabits. Dado que la capacidad por chip se ha incrementado entre un 25% y un 40% por año recientemente y casi doblándose de 2 a 3 años. Supongamos que las frecuencias de crecimiento serán 30% y se duplicará en 3 años ¿Que tamaños de DRAM hay que planificar para utilizar en estos proyectos?. Obsérvese que el crecimiento de las DRAM es discreto

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

- ➊ a) Actualmente 1.5 veces mas rendimiento que el Intel Xeon (según Linpack)

$$1.5 = \frac{te_{Xeon}}{te_{actual}} \Rightarrow te_{actual} = \frac{te_{Xeon}}{1.5} = \frac{2.24 \cdot 10^{-4}}{1.5} = 1.49 \cdot 10^{-4} s$$

- ➋ El rendimiento crece un 22% anual

$$rend_{a1} = rend_{act} + 0.22 \cdot rend_{act} = 1.22 \cdot rend_{act} \Rightarrow 1.22 = \frac{rend_{a1}}{rend_{act}} = \frac{te_{act}}{te_{a1}}$$

- ➌ El primer año  $te_{a1} = \frac{1}{1.22} \cdot te_{act} = 0.82 \cdot te_{act}$

- ➍ El segundo año  $te_{a2} = \frac{1}{1.22} \cdot te_{a1} = 0.82 \cdot te_{a1} = (0.82)^2 \cdot te_{act}$

- ➎ El año  $n$   $te_{an} = \frac{1}{1.22} \cdot te_{an-1} = 0.82 \cdot te_{an-1} = (0.82)^n \cdot te_{act}$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ a) Actualmente 1.5 veces mas rendimiento que el Intel Xeon (según Linpack)

$$1.5 = \frac{te_{Xeon}}{te_{actual}} \Rightarrow te_{actual} = \frac{te_{Xeon}}{1.5} = \frac{2.24 \cdot 10^{-4}}{1.5} = 1.49 \cdot 10^{-4} s$$

- ➋ El rendimiento crece un 22% anual

$$rend_{a1} = rend_{act} + 0.22 \cdot rend_{act} = 1.22 \cdot rend_{act} \Rightarrow 1.22 = \frac{rend_{a1}}{rend_{act}} = \frac{te_{act}}{te_{a1}}$$

- ➌ Empresa A (2 años)

$$te_{a2} = (0.82)^2 \cdot te_{act} = (0.82)^2 \cdot 1.49 \cdot 10^{-4} = 1 \cdot 10^{-4} s = 100 \mu s$$

- ➍ Empresa B (4 años)

$$te_{a4} = (0.82)^4 \cdot te_{act} = (0.82)^4 \cdot 1.49 \cdot 10^{-4} = 6 \cdot 74^{-5} s = 67.4 \mu s$$

## 2.1 Rendimiento. Concepto y definiciones

### Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ b) El tamaño de la memoria actual es 2048 Mb

- ➋ Tamaño de la memoria año 1

$$tm_{a1} = tm_{act} + 0.3 \cdot tm_{act} \Rightarrow tm_{a1} = 1.3 \cdot tm_{act}$$

- ➌ Tamaño de la memoria año 2

$$tm_{a2} = tm_{a1} + 0.3 \cdot tm_{a1} \Rightarrow tm_{a2} = 1.3 \cdot tm_1 = (1.3)^2 \cdot tm_{act}$$

- ➍ Tamaño de la memoria año n

$$tm_{an} = tm_{an-1} + 0.3 \cdot tm_{an-1} \Rightarrow tm_{an} = 1.3 \cdot tm_{an-1} = (1.3)^n \cdot tm_{act}$$

- ➎ Evolución continua

$$tm_{a2} = (1.3)^2 \cdot tm_{act} = (1.3)^2 \cdot 2048 = 3461,12Mb$$

$$tm_{a3} = (1.3)^3 \cdot tm_{act} = (1.3)^3 \cdot 2048 = 4449,45Mb$$

$$tm_{a4} = (1.3)^4 \cdot tm_{act} = (1.3)^4 \cdot 2048 = 5849,29Mb$$

$$tm_{a6} = (1.3)^6 \cdot tm_{act} = (1.3)^6 \cdot 2048 = 9885,30Mb$$

## 2.1 Rendimiento. Concepto y definiciones

### Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ b) El tamaño de la memoria actual es 2048 Mb

- ➋ Tamaño de la memoria año 1

$$tm_{a1} = tm_{act} + 0.3 \cdot tm_{act} \Rightarrow tm_{a1} = 1.3 \cdot tm_{act}$$

- ➌ Tamaño de la memoria año 2

$$tm_{a2} = tm_{a1} + 0.3 \cdot tm_{a1} \Rightarrow tm_{a2} = 1.3 \cdot tm_1 = (1.3)^2 \cdot tm_{act}$$

- ➍ Tamaño de la memoria año n

$$tm_{an} = tm_{an-1} + 0.3 \cdot tm_{an-1} \Rightarrow tm_{an} = 1.3 \cdot tm_{an-1} = (1.3)^n \cdot tm_{act}$$

- ➎ Evolución discreta (x2 cada 3 años)

- ➏ Empresa A (salto superior más cercano 3 años x 2)

$$tm_{EmpA} = 2048Mb_{act} \cdot 2 = 4096Mb$$

- ➐ Empresa B (salto superior más cercano 6 años x 2)

$$tm_{EmpB} = 2048Mb_{act} \cdot 2 \cdot 2 = 8192Mb$$

## 2.1 Rendimiento. Concepto y definiciones

### Ley de Amdhal

Concepto

Amdahl

Relación

- ◆ Relacionado con el principio de diseño de **favorecer el caso frecuente**
- ◆ **Define la ganancia** de rendimiento o aceleración que puede obtenerse al mejorar alguna característica de un computador
- ◆ **La mejora** obtenida en el rendimiento al utilizar algún modo de ejecución más rápido **está limitada por la fracción de tiempo en que se puede utilizar ese modo más rápido**

$$\text{Aceleración Rendimiento} = \frac{\text{Rendimiento con mejora}}{\text{Rendimiento sin mejora}} = \frac{\text{Tiempo ejecución sin mejora}}{\text{Tiempo ejecución con mejora}}$$

Análisis de  
rendimiento

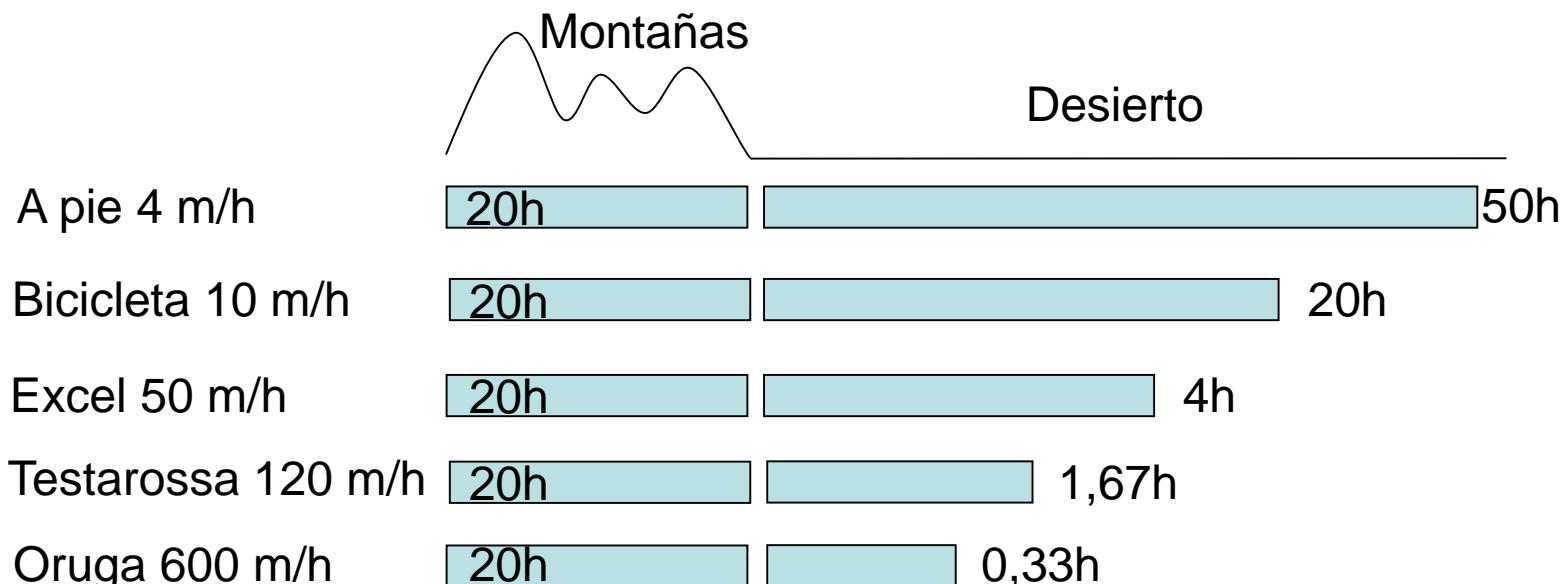
## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

- ◆ **Ejemplo:** Se pretende viajar entre dos puntos cuyo trayecto involucra a dos tipos de terreno. El primer tramo es a través de las montañas y el segundo por el desierto. Tenemos varios tipos de vehículos pero las montañas tienen que ser recorridas necesariamente a pie, empleando para ello 20 horas. El segundo tramo de 200 millas puede ser recorrido de cinco formas diferentes:



Análisis de  
rendimiento

## 2.1 Rendimiento. Concepto y definiciones

Concepto

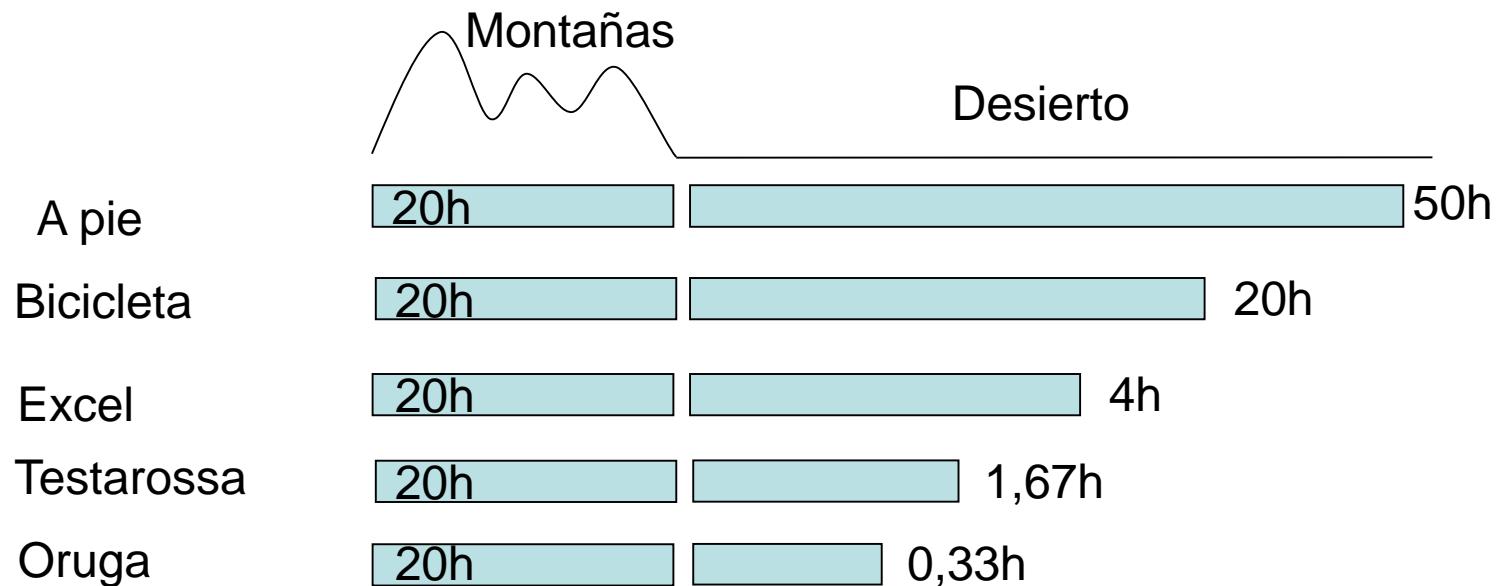
Amdahl

Relación

Análisis de rendimiento

- ◆ Tomando como referencia el recorrido a pie de la distancia completa

Vehículo segunda parte del viaje	Horas de la segunda parte del viaje	Aceleración en el desierto	Horas del viaje completo	Aceleración en el viaje completo
<b>A pie</b>	50	$1=4/4=50/50$	$70=50+20$	$1,0=70/70$
<b>Bicicleta</b>	20	$2,5=10/4=50/20$	$40=20+20$	$1,8=70/40$
<b>Excel</b>	4	$12,5=50/4=50/4$	$24=20+4$	$2,9=70/24$
<b>Testarossa</b>	1,67	$30=120/4=50/1,67$	$21,67=20+1,67$	$3,2=70/21,67$
<b>Vehículo oruga</b>	0,33	$150=600/4=50/0,33$	$20,33=20+0,33$	$3,4=70/20,33$



## 2.1 Rendimiento. Concepto y definiciones

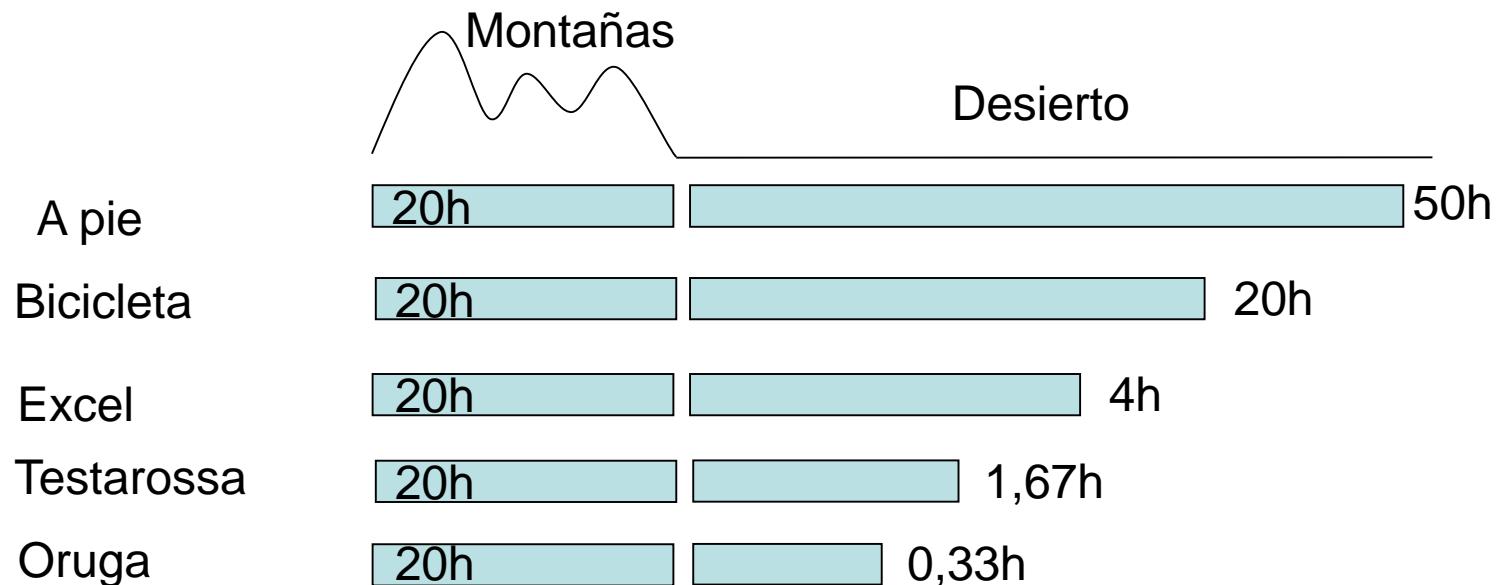
Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ **Ley de Amdahl:** aceleración dependiente de dos factores:
- ➋ **Fracción mejorada:** fracción de tiempo de la opción sin la mejora que puede utilizarse para aprovechar la mejora (siempre menor o igual que uno). En el ejemplo anterior 50/70.
- ➌ **Aceleración mejorada:** Aceleración del modo mejorado (mayor que uno). En el ejemplo anterior aceleración en el desierto.



## 2.1 Rendimiento. Concepto y definiciones

Concepto

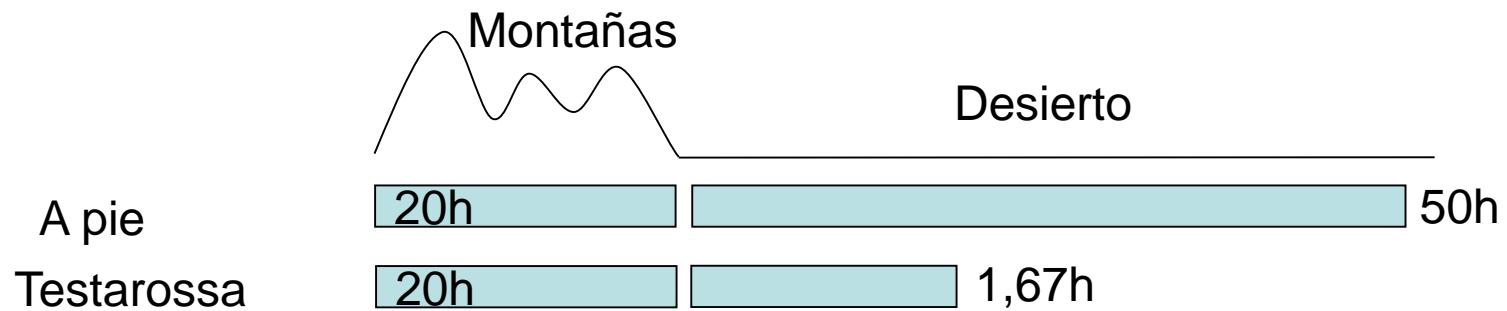
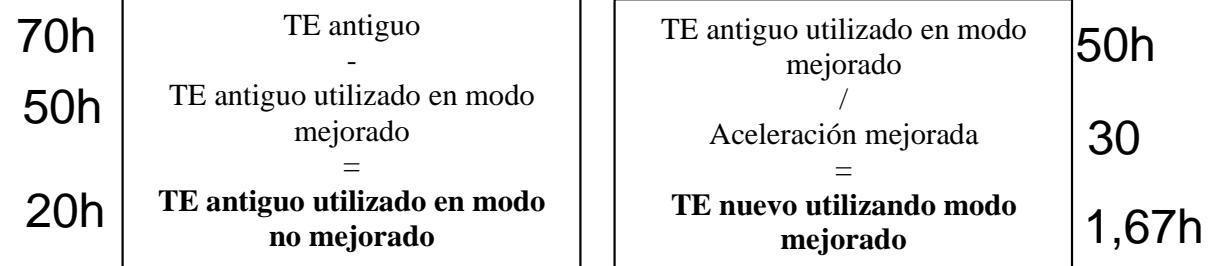
Amdahl

Relación

Análisis de rendimiento

- El **tiempo de ejecución nuevo** (máquina original con el modo mejorado) (21,67) es el tiempo empleado sin utilizar la parte mejorada (20) mas el tiempo empleado utilizando la parte mejorada (1,67) (tiempo a pie + tiempo en Testarossa)

$$TE_{nuevo} = TE_{antiguo} \left( \underbrace{(1 - Fracción_{mejorada})}_{\substack{\text{TE antiguo} \\ - \\ \text{TE antiguo utilizado en modo} \\ \text{mejorado}}} + \underbrace{\frac{Fracción_{mejorada}}{Aceleración_{mejorada}}}_{\substack{\text{TE antiguo utilizado en modo} \\ \text{mejorado} \\ / \\ \text{Aceleración mejorada}}} \right)$$



## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

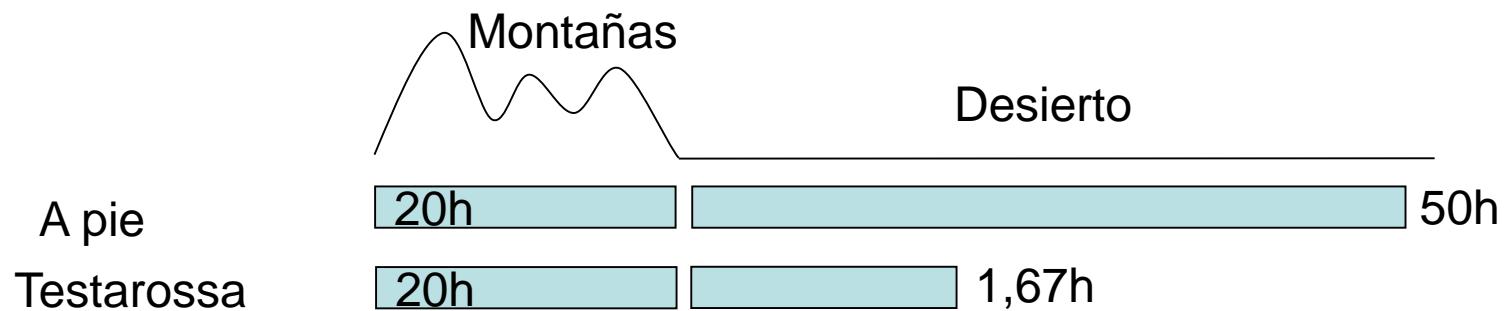
Análisis de rendimiento

- ◆ **El tiempo de ejecución nuevo** (máquina original con el modo mejorado) (21,67) es el tiempo empleado sin utilizar la parte mejorada (20) mas el tiempo empleado utilizando la parte mejorada (1,67) (tiempo a pie + tiempo en Testarossa)

$$Aceleración_{global} = \frac{TE_{antiguo}}{TE_{nuevo}} = \frac{1}{(1 - Fracción_{mejorada}) + \frac{Fracción_{mejorada}}{Aceleración_{mejorada}}}$$

$Fracción_{mejorada} = 1 \rightarrow Aceleración_{global} = Aceleración_{mejorada}$

$Fracción_{mejorada} = 0 \rightarrow Aceleración_{global} = 1$



## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ **Ejemplo:** Suponer que estamos considerando una mejora que corra diez veces más rápida que la máquina original, pero sólo es utilizable el 40% del tiempo. ¿Cuál es la aceleración global lograda al incorporar la mejora?

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

- ◆ **Ejemplo:** Suponer que estamos considerando una mejora que corra diez veces más rápida que la máquina original, pero sólo es utilizable el 40% del tiempo. ¿Cuál es la aceleración global lograda al incorporar la mejora?

$$\text{Fracción}_{\text{mejorada}} = 0,4$$

$$\text{Aceleración}_{\text{mejorada}} = 10$$

$$\text{Aceleración}_{\text{global}} = \frac{1}{0,6 + \frac{0,4}{10}} = \frac{1}{0,64} \approx 1,56$$

- ◆ **Ejemplo:** Suponer que una cache es 5 veces más rápida que la memoria principal y supongamos que la cache puede ser utilizada el 90% del tiempo. ¿Qué aumento de velocidad se logrará al utilizar la cache?

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

- ➊ **Ejemplo:** Suponer que estamos considerando una mejora que corra diez veces más rápida que la máquina original, pero sólo es utilizable el 40% del tiempo. ¿Cuál es la aceleración global lograda al incorporar la mejora?

$$\text{Fracción}_{\text{mejorada}} = 0,4$$

$$\text{Aceleración}_{\text{mejorada}} = 10$$

$$\text{Aceleración}_{\text{global}} = \frac{1}{0,6 + \frac{0,4}{10}} = \frac{1}{0,64} \approx 1,56$$

- ➋ **Ejemplo:** Suponer que una cache es 5 veces más rápida que la memoria principal y supongamos que la cache puede ser utilizada el 90% del tiempo. ¿Qué aumento de velocidad se logrará al utilizar la cache?

$$\text{Fracción}_{\text{mejorada}} = 0,9$$

$$\text{Aceleración}_{\text{mejorada}} = 5$$

$$\text{Aceleración}_{\text{global}} = \frac{1}{0,1 + \frac{0,9}{5}} = \frac{1}{0,25} = 3,6$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ **Ejercicio 3:** Para las cuatro siguientes preguntas, supongamos que se está considerando mejorar una máquina añadiéndole un modo vectorial. Cuando se ejecuta un cálculo en modo vectorial, es 20 veces más rápido que en el modo normal de ejecución. Llamamos al porcentaje de tiempo que puede emplearse el modo vectorial porcentaje de vectorización.
  - ➊ 1. Dibujar un gráfico donde se muestre la aceleración como porcentaje del cálculo realizado en modo vectorial. Rotular el eje y con "aceleración neta" y el eje x con "porcentaje de vectorización".
  - ➋ 2. ¿Qué porcentaje de vectorización se necesita para conseguir una aceleración de 2?
  - ➌ 3. ¿Qué porcentaje de vectorización se necesita para conseguir la mitad de la aceleración máxima alcanzable utilizando el modo vectorial?.
  - ➍ 4. Supongamos que hemos medido el porcentaje de vectorización de programas, obteniendo que es del 70%. El grupo de diseño hardware dice que puede duplicar la velocidad de la parte vectorizada con una inversión significativa de ingeniería adicional. Se desea saber si el equipo de compilación puede incrementar la utilización del modo vectorial como otra aproximación para incrementar el rendimiento. ¿Qué incremento en el porcentaje de vectorización (relativo a la utilización actual) se necesitará para obtener la misma ganancia de rendimiento? ¿Qué inversión es recomendable?.

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

### Ejercicio 3:

- 1. Dibujar un gráfico donde se muestre la aceleración como porcentaje del cálculo realizado en modo vectorial. Rotular el eje y con "aceleración neta" y el eje x con "porcentaje de vectorización".

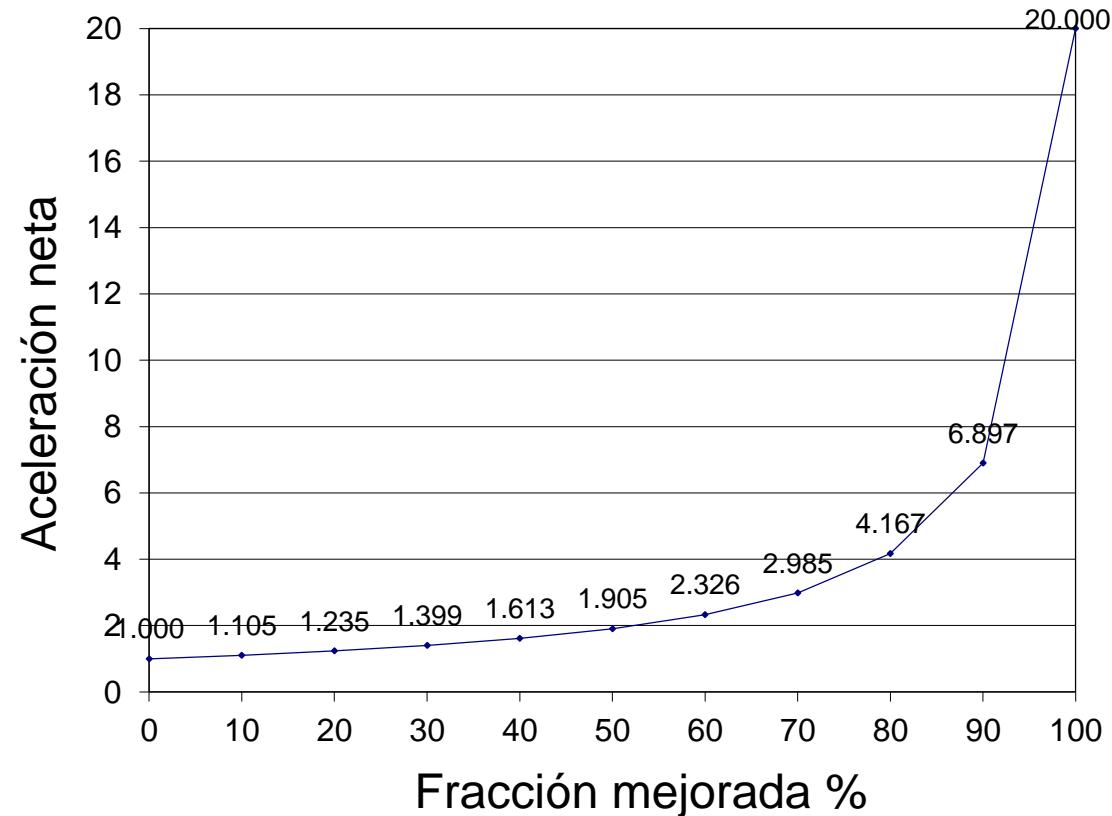
$$0\% \quad a_g = \frac{1}{1 + \frac{0}{20}} = 1$$

$$10\% \quad a_g = \frac{1}{0,9 + \frac{0,1}{20}} = 1,105$$

$$20\% \quad a_g = \frac{1}{0,8 + \frac{0,2}{20}} = 1,235$$

$$90\% \quad a_g = \frac{1}{0,1 + \frac{0,9}{20}} = 6,9$$

$$100\% \quad a_g = \frac{1}{0 + \frac{1}{20}} = 20$$



## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Ejercicio 3:

- 2. ¿Qué porcentaje de vectorización se necesita para conseguir una aceleración de 2?

$$2 = \frac{1}{(1 - f_m) + \frac{f_m}{20}} \Rightarrow f_m = 0,526 = 52,6\%$$

- 3. ¿Qué porcentaje de vectorización se necesita para conseguir la mitad de la aceleración máxima alcanzable utilizando el modo vectorial?.

$$10 = \frac{1}{(1 - f_m) + \frac{f_m}{20}} \Rightarrow f_m = 0,947 \simeq 95\%$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

- 4. Hemos medido el porcentaje de vectorización de programas, obteniendo que es del 70%. El grupo de diseño hardware dice que puede duplicar la velocidad de la parte vectorizada con una inversión significativa de ingeniería adicional. Se desea saber si el equipo de compilación puede incrementar la utilización del modo vectorial como otra aproximación para incrementar el rendimiento. ¿Que incremento en el porcentaje de vectorización (relativo a la utilización actual) se necesitará para obtener la misma ganancia de rendimiento? ¿Que inversión es recomendable?.

$$TE_{ant} = TE_{vec} * 20 = TE_{vec.r} * 40 \rightarrow TE_{vec} = TE_{vec.r} * 2$$

- Aceleración global conseguida por el grupo de hardware

$$a_g = \frac{1}{(1 - 0,7) + \frac{0,7}{40}} = 3,15$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- 4. Hemos medido el porcentaje de vectorización de programas, obteniendo que es del 70%. El grupo de diseño hardware dice que puede duplicar la velocidad de la parte vectorizada con una inversión significativa de ingeniería adicional. Se desea saber si el equipo de compilación puede incrementar la utilización del modo vectorial como otra aproximación para incrementar el rendimiento. ¿Que incremento en el porcentaje de vectorización (relativo a la utilización actual) se necesitará para obtener la misma ganancia de rendimiento? ¿Que inversión es recomendable?.

$$TE_{ant} = TE_{vec} * 20 = TE_{vec.r} * 40 \rightarrow TE_{vec} = TE_{vec.r} * 2$$

- Incremento del % de vectorización para alcanzar  $a_g=3,15$

$$a_g = 3,15 = \frac{1}{(1-f_m) + \frac{f_m}{20}} \Rightarrow f_m = 0,719$$

$$\Delta \% vector = |0,7 - 0,719| = 0,019 = 1,9\%$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ **Ejercicio 4:** El coprocesador de un computador mejora en un factor de 5 el procesamiento de números en coma flotante. El tiempo de ejecución de cierto programa es de 1 minuto con el coprocesador instalado, y de 2,5 minutos sin este.
  - ➋ a) Calcula el porcentaje del tiempo de ejecución, sin el coprocesador instalado, que el programa realiza operaciones en coma flotante.
  - ➋ b) Calcula el tiempo de ejecución del programa sin el coprocesador instalado, para realizar las operaciones en coma flotante.
  - ➋ c) Calcula el tiempo de ejecución del programa con el coprocesador instalado, para realizar las operaciones en coma flotante.
  - ➋ d) Calcula el tiempo de ejecución del programa para realizar las operaciones enteras.
  - ➋ e) Comprueba la coherencia del planteamiento sumando los resultados de los apartados c y d, obteniendo el tiempo de ejecución del programa con el coprocesador instalado.

## 2.1 Rendimiento. Concepto y definiciones

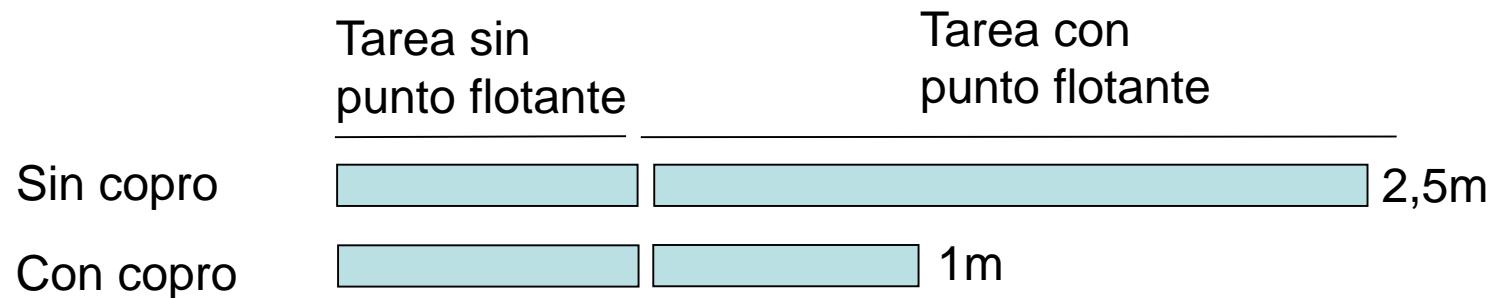
Concepto

Amdahl

Relación

Análisis de rendimiento

- ◆ **Ejercicio 4:** El coprocesador de un computador mejora en un factor de 5 el procesamiento de números en coma flotante. El tiempo de ejecución de cierto programa es de 1 minuto con el coprocesador instalado, y de 2,5 minutos sin este.
  - ◆ a) Calcula el porcentaje del tiempo de ejecución, sin el coprocesador instalado, que el programa realiza operaciones en coma flotante.



$$a_g = \frac{t_{e.ant}}{t_{e.nue}} = \frac{2,5}{1} = \frac{1}{(1 - f_m) + \frac{f_m}{5}} \Rightarrow f_m = 0,75$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

- ◆ **Ejercicio 4:** El coprocesador de un computador mejora en un factor de 5 el procesamiento de números en coma flotante. El tiempo de ejecución de cierto programa es de 1 minuto con el coprocesador instalado, y de 2,5 minutos sin este.
- ◆ b) Calcula el tiempo de ejecución del programa sin el coprocesador instalado, para realizar las operaciones en coma flotante.



$$t_{e.ant} = 2,5$$

$$t_{e.ant.flot} = 2,5 \cdot 0,75 = 1,875m$$

## 2.1 Rendimiento. Concepto y definiciones

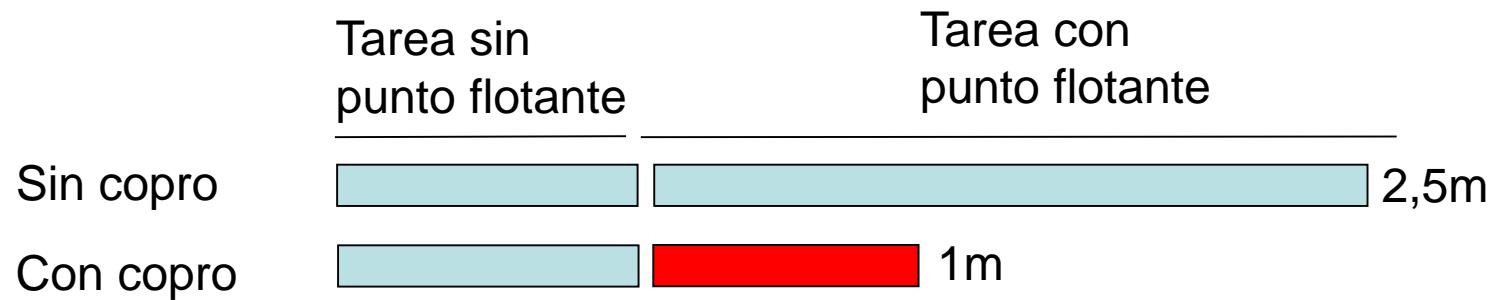
Concepto

Amdahl

Relación

Análisis de rendimiento

- ◆ **Ejercicio 4:** El coprocesador de un computador mejora en un factor de 5 el procesamiento de números en coma flotante. El tiempo de ejecución de cierto programa es de 1 minuto con el coprocesador instalado, y de 2,5 minutos sin este.
- ◆ c) Calcula el tiempo de ejecución del programa con el coprocesador instalado, para realizar las operaciones en coma flotante.



$$t_{e.ant} = 2,5$$

$$t_{e.nue.flot} = \frac{1,875}{5} = 0,375m$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

- ◆ **Ejercicio 4:** El coprocesador de un computador mejora en un factor de 5 el procesamiento de números en coma flotante. El tiempo de ejecución de cierto programa es de 1 minuto con el coprocesador instalado, y de 2,5 minutos sin este.
- ◆ d) Calcula el tiempo de ejecución del programa para realizar las operaciones enteras.



$$t_{e.int} = 2,5 - 1,875 = 0,625m$$

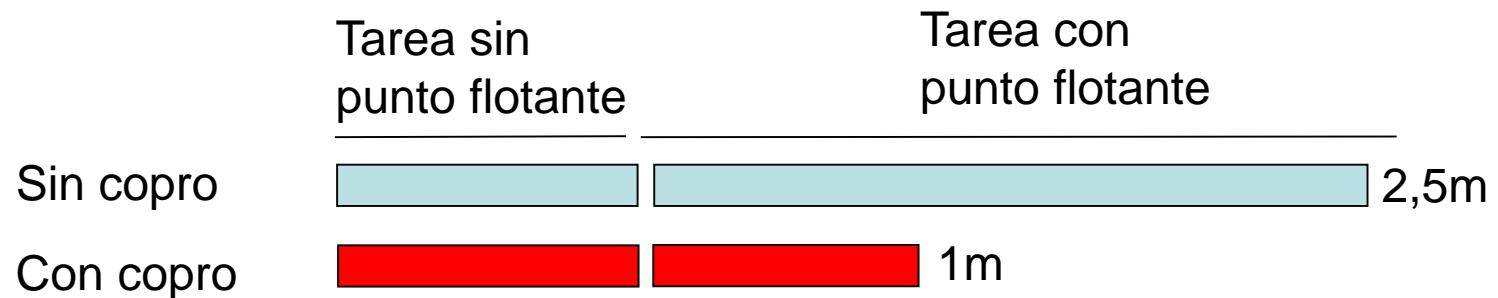
## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

- ◆ **Ejercicio 4:** El coprocesador de un computador mejora en un factor de 5 el procesamiento de números en coma flotante. El tiempo de ejecución de cierto programa es de 1 minuto con el coprocesador instalado, y de 2,5 minutos sin este.
- ◆ e) Comprueba la coherencia del planteamiento sumando los resultados de los apartados c y d, obteniendo el tiempo de ejecución del programa con el coprocesador instalado.



Análisis de rendimiento

$$t_{nue} = t_{\text{int}} - t_{\text{nue.flo}} = 0,375 + 0,625 = 1 \text{ m}$$

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Relación rendimiento y coste

¿Por qué el cliente se decide por un computador en lugar de otro?

- ➊ Las medidas estándares del rendimiento permiten comparar cuantitativamente a los diseñadores
- ➋ Las comparativas basan sus informes en aspectos relacionados con el rendimiento y coste de compra

## 2.1 Rendimiento. Concepto y definiciones

Concepto

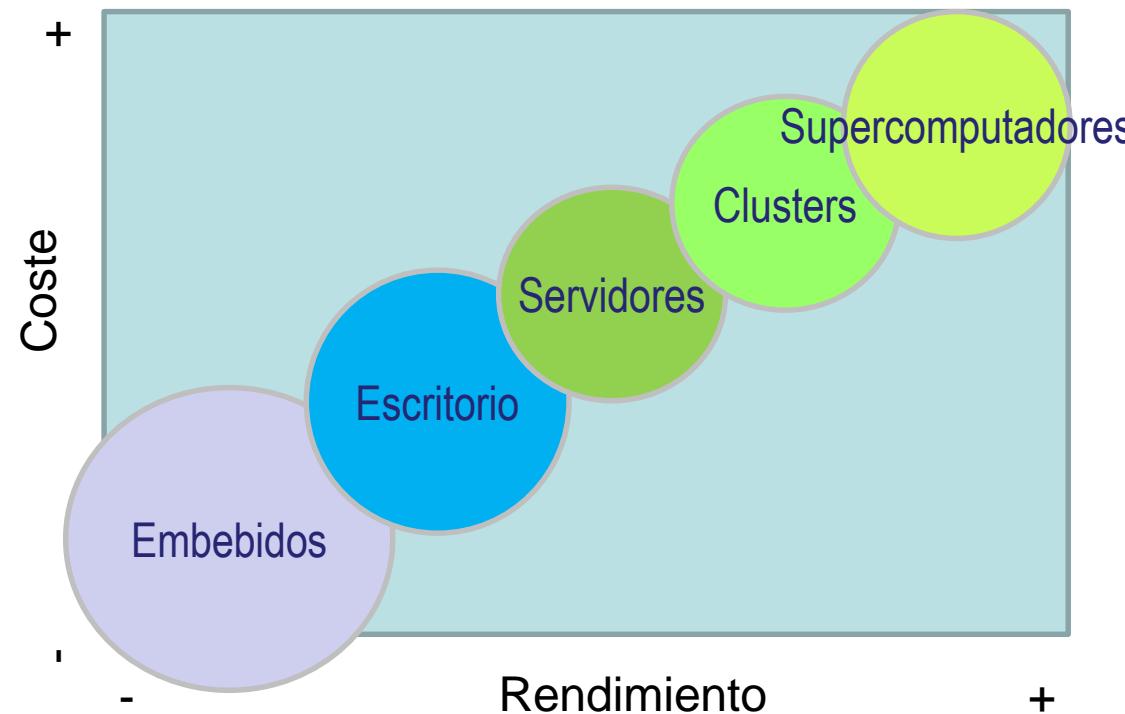
Amdahl

Relación

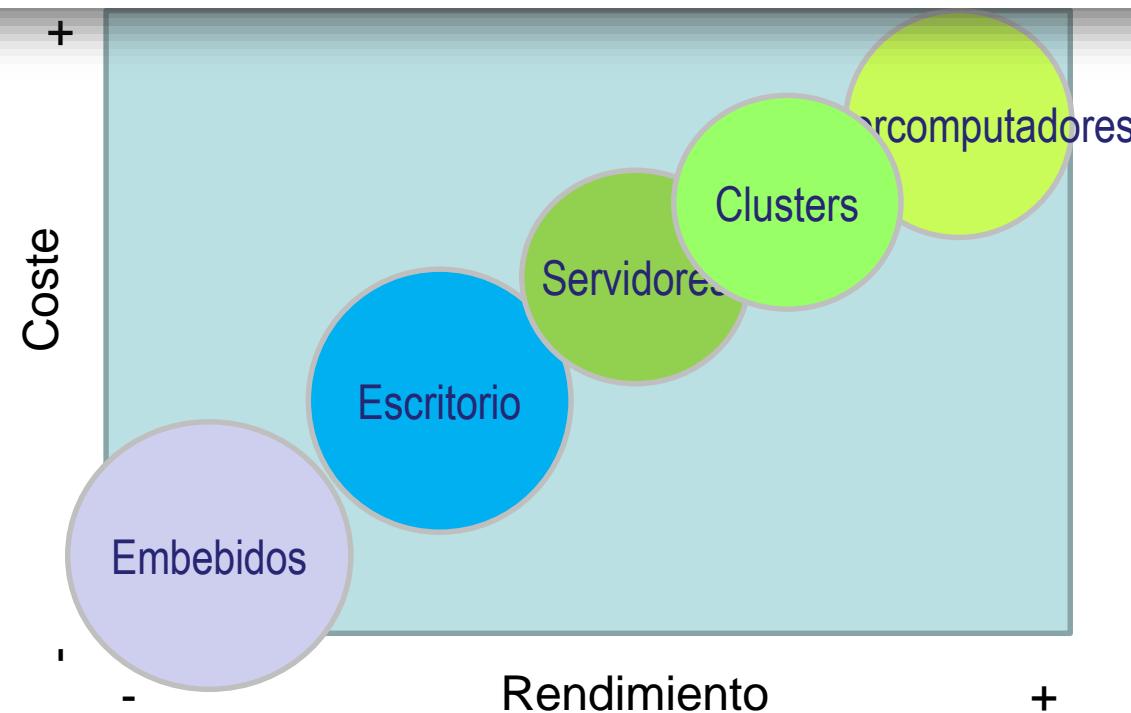
Análisis de rendimiento

### Relación rendimiento y coste

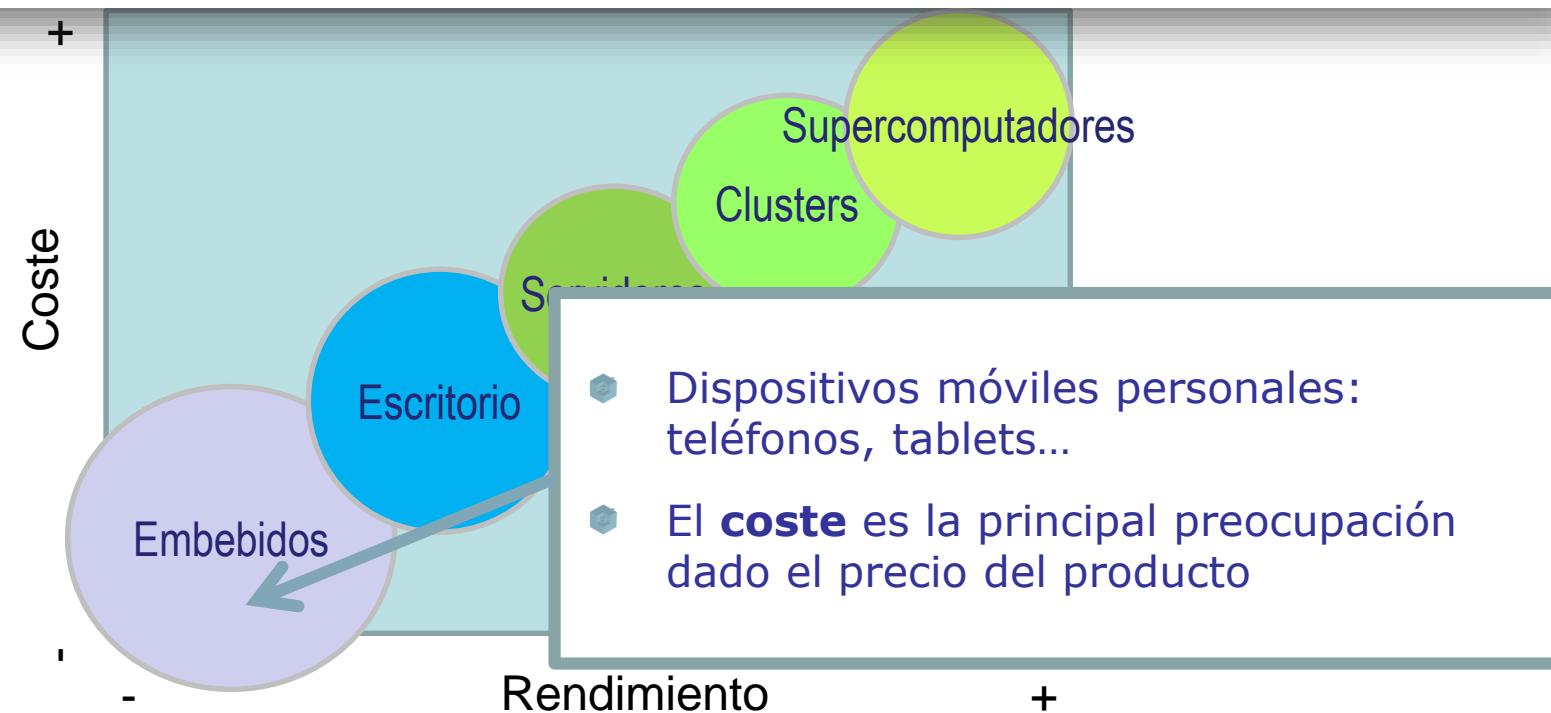
- El diseño de computadores abarca desde el alto coste alto rendimiento hasta el bajo coste bajo rendimiento



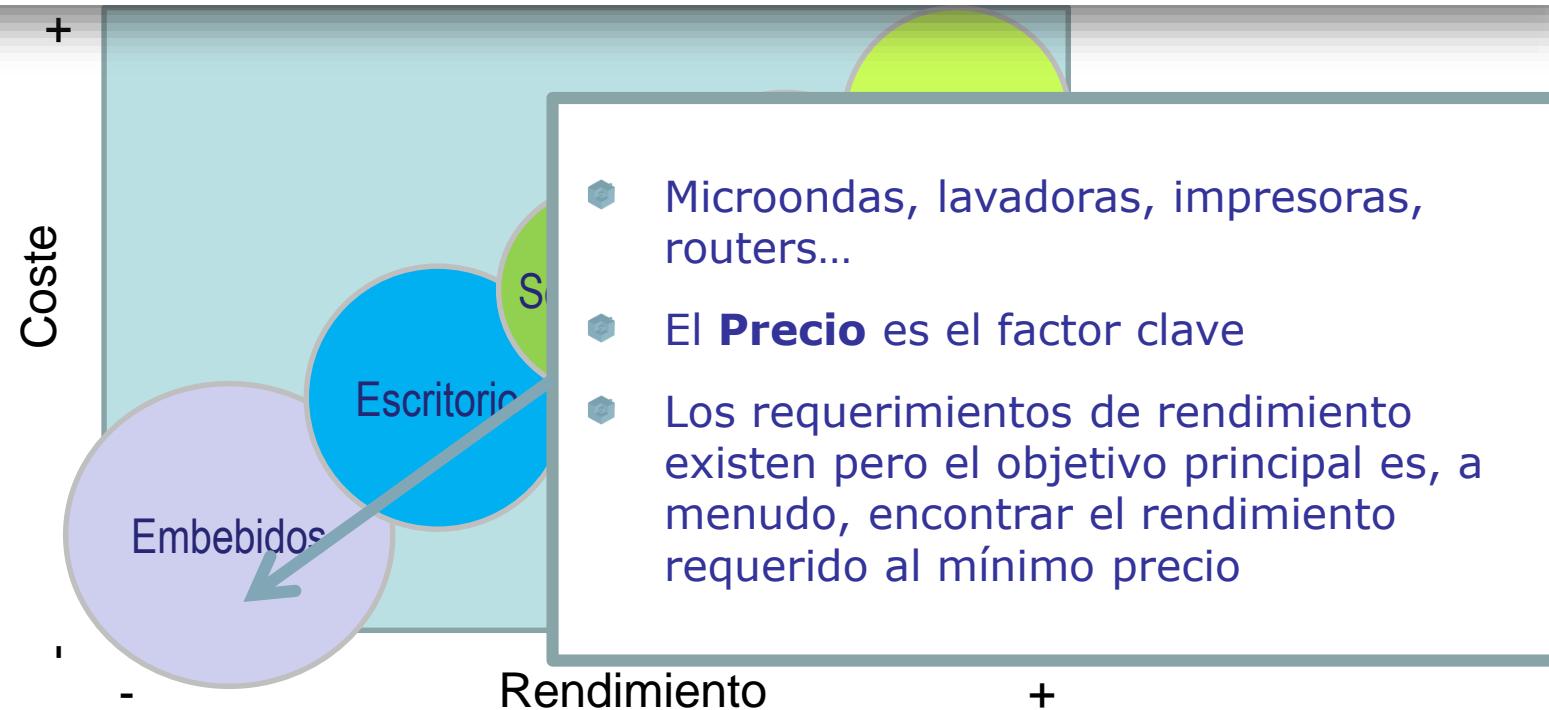
	Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Cost	Price of system	\$100–\$1000	\$300–\$2500	\$5000–\$10,000,000	\$100,000–\$200,000,000	\$10–\$100,000
Complexity	Price of micro-processor	\$10–\$100	\$50–\$500	\$200–\$2000	\$50–\$250	\$0.01–\$100
Architecture	Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance
Reliability						



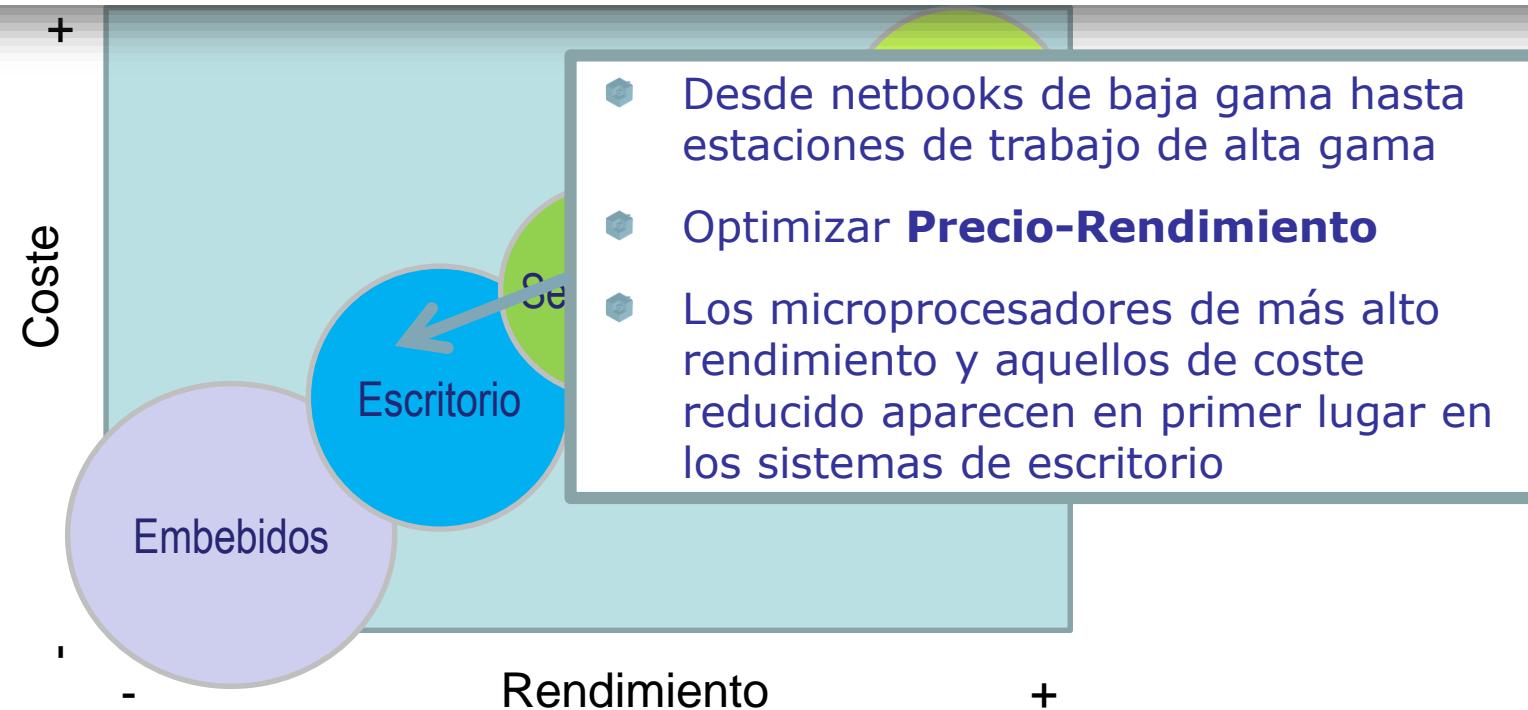
	Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Construcción de sistemas	Price of system	\$100–\$1000	\$300–\$2500	\$5000–\$10,000,000	\$100,000–\$200,000,000	\$10–\$100,000
	Price of micro-processor	\$10–\$100	\$50–\$500	\$200–\$2000	\$50–\$250	\$0.01–\$100
	Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance



	Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Cost	Price of system	\$100–\$1000	\$300–\$2500	\$5000–\$10,000,000	\$100,000–\$200,000,000	\$10–\$100,000
Architecture	Price of micro-processor	\$10–\$100	\$50–\$500	\$200–\$2000	\$50–\$250	\$0.01–\$100
Requirements	Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

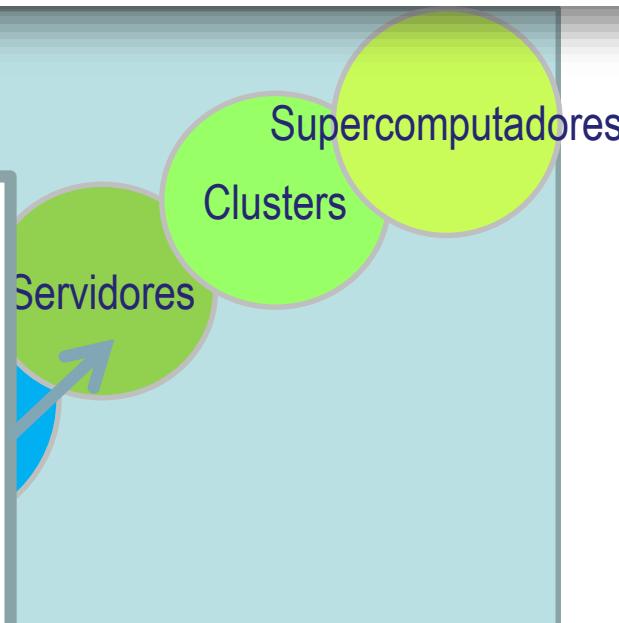


	Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Cost	Price of system	\$100–\$1000	\$300–\$2500	\$5000–\$10,000,000	\$100,000–\$200,000,000	\$10–\$100,000
Architecture	Price of micro-processor	\$10–\$100	\$50–\$500	\$200–\$2000	\$50–\$250	\$0.01–\$100
Reliability	Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance



	Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Cost	Price of system	\$100–\$1000	\$300–\$2500	\$5000–\$10,000,000	\$100,000–\$200,000,000	\$10–\$100,000
Architecture	Price of micro-processor	\$10–\$100	\$50–\$500	\$200–\$2000	\$50–\$250	\$0.01–\$100
Reliability	Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

+



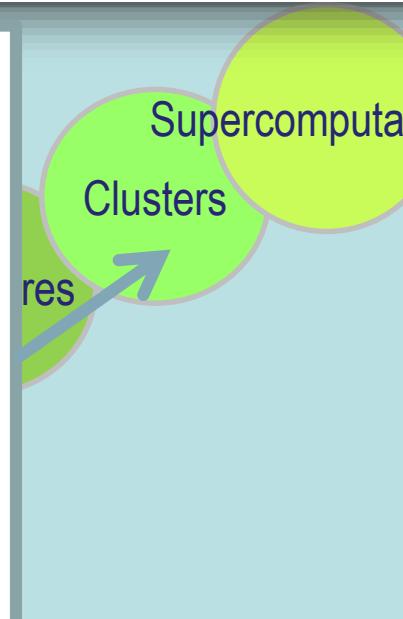
- ◆ Diseñados para una **productividad** eficiente
- ◆ Rendimiento global del servidor—en términos de transacciones por minuto o páginas web servidas por segundo—es crucial

Rendimiento

+

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Price of system	\$100–\$1000	\$300–\$2500	\$5000–\$10,000,000	\$100,000–\$200,000,000	\$10–\$100,000
Processor	Price of micro-processor	\$10–\$100	\$50–\$500	\$200–\$2000	\$0.01–\$100
Architecture	Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality
Reliability					Price, energy, application-specific performance

- Colecciones de computadores de escritorio o servidores conectados por redes de área local que actúan como un solo computador más grande
- Relacionados con aplicaciones de búsqueda, redes sociales, compartición de video, juegos multijugador...
- **Precio-rendimiento** es crítico

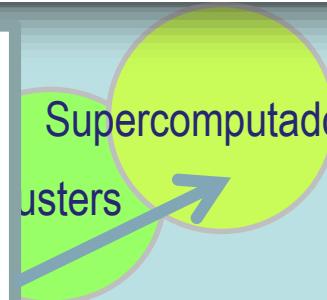


Rendimiento

+

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Price of system	\$100–\$1000	\$300–\$2500	\$5000–\$10,000,000	\$100,000–\$200,000,000	\$10–\$100,000
Processor	Price of micro-processor	\$10–\$100	\$50–\$500	\$200–\$2000	\$0.01–\$100
Architecture	Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality
Reliability					Price, energy, application-specific performance

- Énfasis en el rendimiento del punto flotante, capaces de ejecutar grandes programas por lotes que pueden correr por semanas
- Rendimiento** es crítico. El coste es menos importante



rendimiento

coste

Rendimiento

+

## 2.1 Rendimiento. Concepto y definiciones

### Coste

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ El coste es un parámetro a tener muy en cuenta al diseñar un nuevo procesador o al modificar uno existente
- ➋ Los factores principales que influyen en el coste de un computador son:
  - ➌ **Curva de aprendizaje:**
    - ➍ Costes de manufacturación decrecen a lo largo del tiempo incluso sin mejoras en la tecnología de implementación básica
    - ➎ El porcentaje de dispositivos manufacturados que pasan los procedimientos de prueba se incrementa a lo largo del tiempo

## 2.1 Rendimiento. Concepto y definiciones

### Coste

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ El coste es un parámetro a tener muy en cuenta al diseñar un nuevo procesador o al modificar uno existente
- ➋ Los factores principales que influyen en el coste de un computador son:
  - ➌ **Curva de aprendizaje:**
  - ➍ **Volumen:** El incremento del volumen afecta al coste en:
    - ➎ Reduciendo el tiempo necesario para bajar la curva de aprendizaje
    - ➏ Incrementando la eficiencia de compra y manufactura (10% menos por cada doble de volumen)
    - ➐ Reduciendo la cantidad del coste de diseño que debe ser amortizado para cada computador

## 2.1 Rendimiento. Concepto y definiciones

### Coste

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- ➊ El coste es un parámetro a tener muy en cuenta al diseñar un nuevo procesador o al modificar uno existente
- ➋ Los factores principales que influyen en el coste de un computador son:
  - ➌ **Curva de aprendizaje:**
  - ➌ **Volumen:**
  - ➌ **Mercado altamente competitivo:**
    - ➍ Las DRAMs, discos, monitores, teclados son vendidos por diferentes fabricantes y son esencialmente idénticos
    - ➍ La competencia reduce la distancia entre el precio de venta y el coste, pero también reduce el coste porque:
      - ➎ Los componentes tienen tanto un gran volumen como una clara definición

## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de  
rendimiento

### Coste de un circuito integrado (IC)

- ➊ Los costes de los ICs se están convirtiendo en una porción cada vez más grande del coste que varía entre computadores
- ➋ Los factores que influyen en el coste del silicio son:
  - ➌ **El número de puertas:** influye en el número de transistores necesarios. Un aumento de estos requiere un área de silicio mayor
  - ➌ **Conexiones** entre elementos: el número y la longitud
  - ➌ **Regularidad** del diseño: cuanto más regular sea el diseño, menos área ocupará
- ➋ Los procesos de IC están caracterizados por el “feature size”:
  - ➌ Tamaño mínimo de un transistor o conexión sea en la dimensión X o Y (10 micras en 1971 a 0.007 micras en 2018)

## 2.1 Rendimiento. Concepto y definiciones

Microprocessor	16-Bit address/ bus, microcoded	32-Bit address/ bus, microcoded	5-Stage pipeline, on-chip I & D caches, FPU	2-Way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip L2 cache	Multicore OOO 4-way on chip L3 cache, Turbo
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4	Intel Core i7
Year	1982	1985	1989	1993	1997	2001	2015
Die size (mm <sup>2</sup> )	47	43	81	90	308	217	122
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000	1,750,000,000
Processors/chip	1	1	1	1	1	1	4
Pins	68	132	168	273	387	423	1400
Latency (clocks)	6	5	5	5	10	22	14
Bus width (bits)	16	32	32	64	64	64	196
Clock rate (MHz)	12.5	16	25	66	200	1500	4000
Bandwidth (MIPS)	2	6	25	132	600	4500	64,000
Latency (ns)	320	313	200	76	50	15	4

- ◆ Los procesos de IC están caracterizados por el “feature size”:
  - ◆ Tamaño mínimo de un transistor o conexión sea en la dimensión X o Y (7 micras en 1971 a 0.007 micras en 2018)

## 2.1 Rendimiento. Concepto y definiciones

Concepto

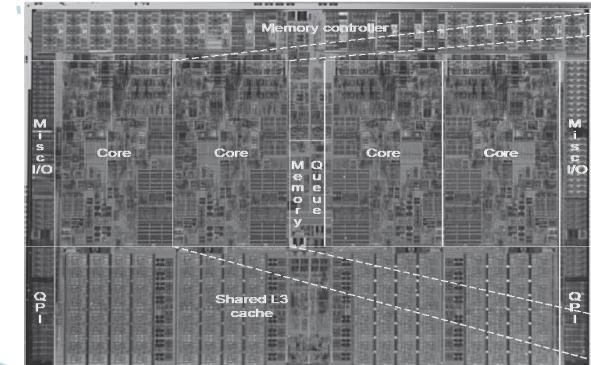
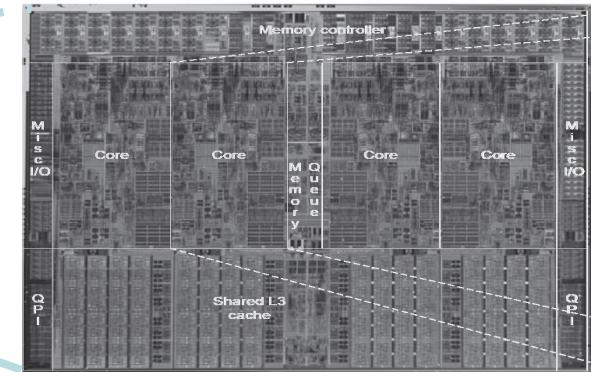
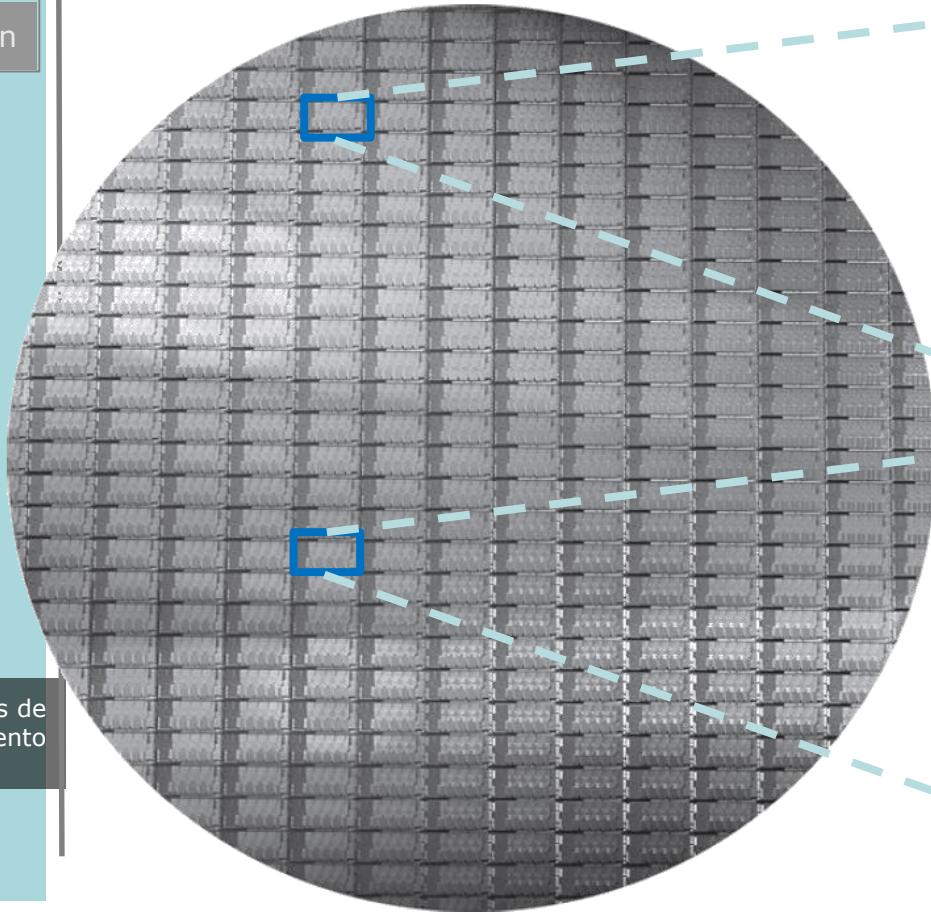
Amdahl

Relación

Análisis de rendimiento

### Coste de un circuito integrado (IC)

- El proceso básico de fabricación del silicio no ha cambiado: la oblea es testeada y cortada en dados que son empaquetados



## 2.1 Rendimiento. Concepto y definiciones

Concepto

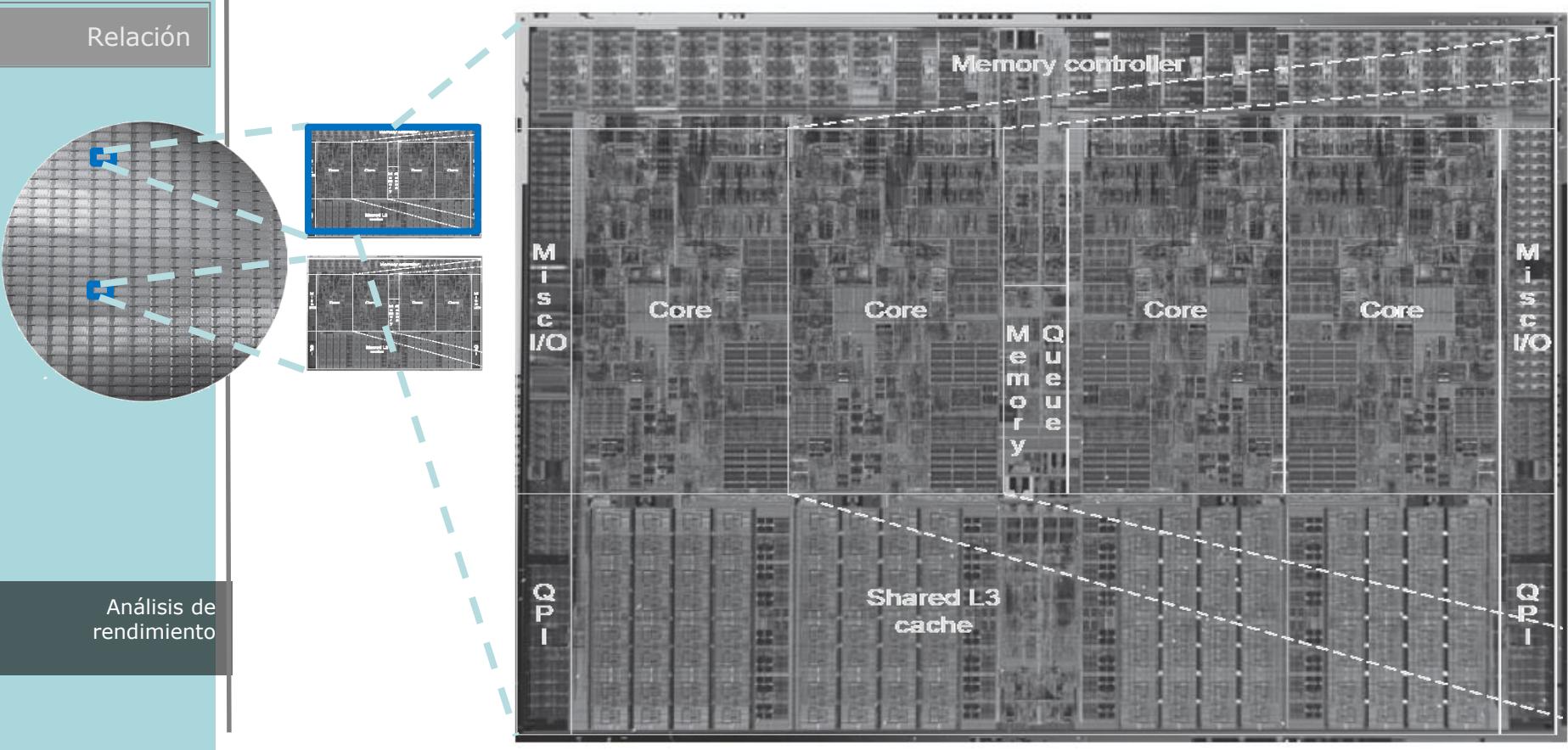
Amdahl

Relación

Análisis de rendimiento

### Coste de un circuito integrado (IC)

- El proceso básico de fabricación del silicio no ha cambiado: la oblea es testeada y cortada en dados que son empaquetados



## 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

### Coste de un circuito integrado (IC)

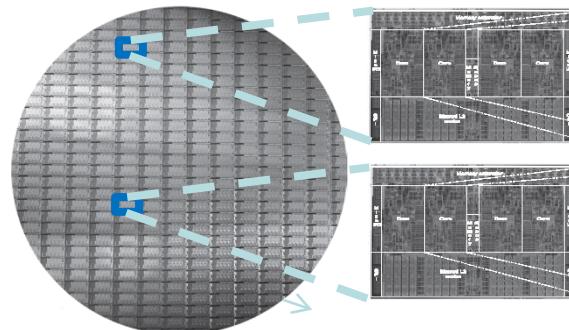
- El proceso básico de fabricación del silicio no ha cambiado: la óblea es testeada y cortada en dados que son empaquetados

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

- El número de dados por óblea es aproximadamente el área de la óblea dividida por el área del dado. De forma más precisa, puede estimarse por:

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \boxed{\frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}}$$



- Compensa los dados cerca de la periferia de la óblea. Aproximadamente el número de dados a lo largo del borde

# 2.1 Rendimiento. Concepto y definiciones

Concepto

Amdahl

Relación

Análisis de rendimiento

## Coste de un circuito integrado (IC)

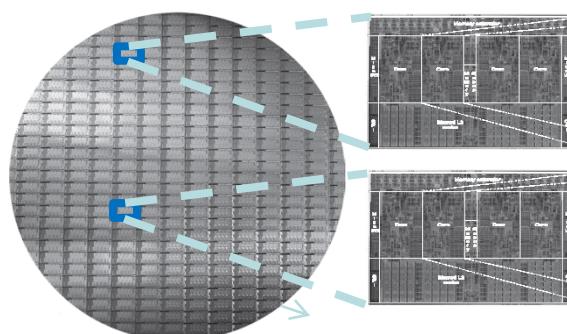
- Este es el numero máximo de dados por óblea:

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- ¿Cual es la fracción de dados no defectuosos en la óblea, o el rendimiento del dado (die yield)? Asumiendo que:
  - Los defectos están distribuidos aleatoriamente
  - el rendimiento es inversamente proporcional a la complejidad del proceso de fabricación

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

- Fórmula Bose-Einstein: modelo empírico teniendo en cuenta el rendimiento en muchas líneas de fabricación



- Defectos por cm<sup>2</sup> (2017)
  - = 0.012-0.016 (28 nm)
  - = 0.016-0.047 (16 nm)
- N, factor de complejidad del proceso
  - = 7.5-9.5 (28 nm)
  - = 10-14 (16 nm)

## 2.1 Rendimiento. Concepto y definiciones

### Ejercicio

Concepto

Amdahl

Relación

Análisis de  
rendimiento

- Intel utiliza obleas de 300mm para construir su Core i7. Estas obleas tienen un coste de 20000€. El dado del Core i7 tiene unas dimensiones 20.7 mm x 10.5 mm. Asumiendo que el proceso de fabricación de Intel tiene una densidad de defectos de 0.023 por cm<sup>2</sup> y que el factor de complejidad del proceso para un tamaño mínimo del transistor de 32 nm es 13.5, calcula el coste del dado. Por simplicidad, se asume que el rendimiento de la oblea es del 100% ¿Cuál sería el coste del dado dentro de 2 años, sabiendo que el factor de complejidad del proceso para la tecnología de 32nm decrece un 5% por año?

## **2.2. Evaluación del rendimiento**

---

**Tema 2 Análisis del rendimiento**

**Arquitectura de los Computadores**

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Tiempo de ejecución

- ➊ El tiempo es la medida más fiable del rendimiento
- ➋ El tiempo de ejecución de un programa se mide en segundos

### El rendimiento

- ➊ La relación entre el tiempo y el rendimiento es inversa
- ➋ El rendimiento se mide como una frecuencia de eventos por segundo

$$\text{Rendimiento} = \frac{1}{\text{tiempo}}$$

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Tiempo de programa / Tiempo de CPU

- ➊ **Tiempo de reloj, tiempo de respuesta, tiempo transcurrido:** Latencia para completar una tarea incluyéndolo todo: accesos a disco, accesos a memoria, actividades de entrada salida, gastos del sistema operativo, multiprogramación...
- ➋ **Rendimiento del sistema:** Este término se utiliza para referenciar el tiempo transcurrido en un sistema no cargado.
  
- ➌ **Tiempo de CPU**
  - ➍ CPU usuario + CPU sistema
  - ➎ Tiempo en que la CPU está calculando sin incluir tiempos de espera para E/S o para ejecución de otros programas
  - ➏ **Tiempo de programa:** Este término se refiere al tiempo de CPU del usuario (nos centraremos en rendimiento de la CPU)

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

### Tiempo de programa / Tiempo de CPU

- ◆ **Tiempo de reloj, tiempo de respuesta, tiempo transcurrido:** Latencia para completar una tarea incluyéndolo todo: accesos a disco, accesos a memoria, actividades de entrada salida, gastos del sistema operativo, multiprogramación...
- ◆ **Rendimiento del sistema:** Este término se utiliza para referenciar el tiempo transcurrido en un sistema no cargado.

La función *time* de Unix produce una salida de la forma: 90.7u 12.9s 2:39 65%, donde:

- Tiempo de CPU del usuario = 90.7 segundos
- Tiempo de CPU utilizado por el sistema = 12.9 segundos
- Tiempo de CPU= 90.7 seg.+ 12.9seg = 103.6
- Tiempo de respuesta = 2 minutos 39 segundos =159 segundos
- Tiempo de CPU = 65% del tiempo de respuesta = 159 segundos\*0.65 = 103.6
- Tiempo esperando operaciones de E/S y/o el tiempo ejecutando otras tareas 35% del tiempo de respuesta = 159 segundos\*0.35 = 55.6 segundos

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Tiempo de programa / Tiempo de CPU

- El tiempo de CPU de un programa puede expresarse en función del ciclo de reloj

$$\text{Tiempo de CPU} = \frac{\text{Ciclos de reloj de CPU para un programa}}{\text{Duración del ciclo de reloj}}$$

$$\text{Tiempo de CPU} = \frac{\text{Ciclos de reloj de CPU para un programa}}{\text{Frecuencia de reloj}}$$

- No tiene sentido mostrar el tiempo transcurrido en función del ciclo de reloj ya que la latencia de los dispositivos de entrada salida es independiente del ciclo de reloj de la CPU

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### CPI

- ◆ Número medio de ciclos de reloj por instrucción. Se expresa en función del número de ciclos de reloj y el número de instrucciones ejecutadas

$$\text{CPI} = \frac{\text{Ciclos de reloj de CPU para un programa}}{\text{Recuento de instrucciones}}$$

- ◆ Podemos expresar el tiempo de CPU en función del CPI:

$$\text{Tiempo de CPU} = \frac{\text{Ciclos de reloj de CPU para un programa}}{\text{Duración del ciclo de reloj}}$$

$$\text{Tiempo de CPU} = \frac{\text{Recuento de instrucciones}}{\text{Duración del ciclo de reloj}} * \text{CPI}$$

$$\text{Tiempo de CPU} = \text{RI} * \text{CPI} * \text{clk}$$

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

**En ocasiones se detalla el CPI por cada tipo de instrucción estática i**

$$\text{Ciclos de reloj de la CPU} = \sum_{i=1}^n (CPI_i \cdot I_i)$$

$I_i$  = instrucciones dinámicas para cada tipo de instrucción estática i.

$CPI_i$  = Número medio de ciclos de reloj para la instrucción tipo i.

- Esto nos permite expresar

$$\text{Tiempo de CPU} = \sum_{i=1}^n (CPI_i \cdot I_i) \cdot \text{Duración del ciclo de reloj}$$

$$CPI = \frac{\sum_{i=1}^n (CPI_i \cdot I_i)}{\text{recuento de instrucciones}} = \sum_{i=1}^n (CPI_i \cdot \frac{I_i}{\text{recuento de instrucciones}})$$

- CPI<sub>i</sub> medido, no obtenido de tabla de referencia. Deben considerarse fallos de cache y demás incidencias del sistema de memoria.

## 2.2 Evaluación del rendimiento

Métricas

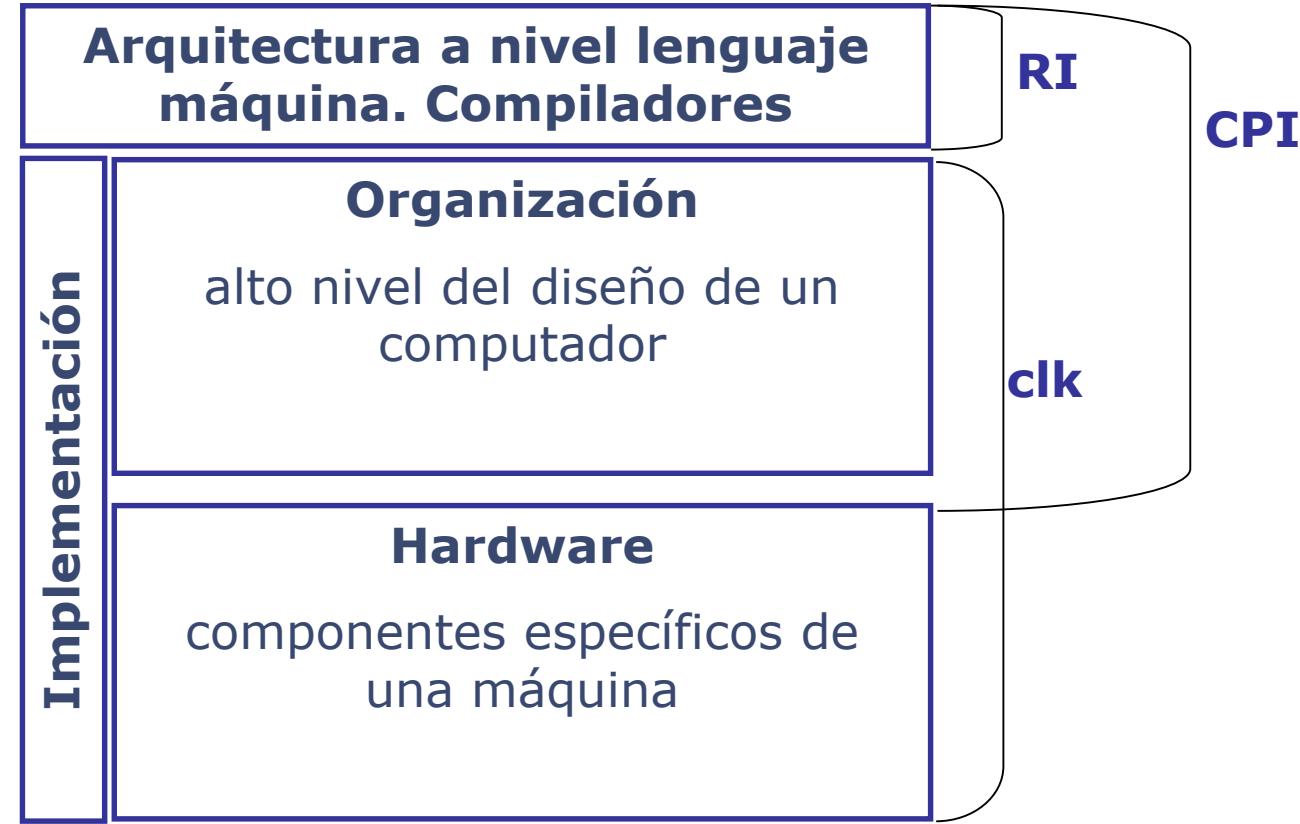
Benchmarks

Formulación

Análisis de rendimiento

### Tres parámetros interdependientes

$$\text{Tiempo de CPU} = \text{RI} * \text{CPI} * \text{clk}$$



## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

**Ejemplo:** Suponer que estamos considerando dos alternativas para una instrucción de salto condicional:

CPU A. Una instrucción de comparación inicializa un código de condición y es seguida por un salto que examina el código de condición.

CPU B. Se incluye una comparación en el salto.

En ambas CPU, la instrucción de salto condicional emplea 2 ciclos de reloj, y las demás instrucciones 1 (en este sencillo ejemplo se están despreciando las pérdidas del sistema de memoria). En la CPU A, el 20% de todas las instrucciones ejecutadas son saltos condicionales; como cada salto necesita una comparación, otro 20% de las instrucciones son comparaciones. Debido a que la CPU A no incluye la comparación en el salto, su ciclo de reloj es un 25% más rápido que el de la CPU B. ¿Qué CPU es más rápida?.

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

**Ejemplo:** Después de ver el análisis, un diseñador consideró que, volviendo a trabajar en la organización, la diferencia de las duraciones de los ciclos de reloj podía reducirse, fácilmente, a un 10%. ¿Qué CPU es más rápida ahora?.

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

**Ejemplo:** Supongamos que estamos considerando otro cambio en un repertorio de instrucciones. La máquina, inicialmente, sólo tiene instrucciones de carga y de almacenamiento en memoria, y, después, todas las operaciones se realizan en los registros. Tales máquinas se denominan máquinas de carga almacenamiento (load/store). A continuación observamos medidas de la máquina de carga almacenamiento que muestran la frecuencia de instrucciones, denominada mezcla de instrucciones (instruction mix) y número de ciclos de reloj por instrucción.

Operación	Frecuencia	Cuenta de ciclos de reloj
Ops ALU	43%	1
Load	21%	2
store	12%	2
Saltos	24%	2

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

Supongamos que el 25% de las operaciones de la unidad aritmético lógica (ALU) utilizan directamente un operando cargado que no se utiliza de nuevo.

Proponemos añadir instrucciones a la ALU que tengan un operando fuente en memoria. Estas nuevas instrucciones de registro memoria emplean dos ciclos de reloj. Supongamos que el repertorio extendido de instrucciones incrementa en 1 el número de ciclos de reloj para los saltos, pero sin afectar a la duración del ciclo de reloj. ¿Mejorará este cambio el rendimiento de la CPU?.

Operación	Frecuencia	Cuenta de ciclos de reloj
Ops ALU	43%	1
Load	21%	2
store	12%	2
Saltos	24%	2

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Alternativas para la medida del rendimiento

- ◆ La medida más fiable del rendimiento es el **tiempo de ejecución** de los **programas reales**
- ◆ Alternativas al tiempo como medida del rendimiento y a los programas reales como objetos de medida han conducido a errores en el diseño de computadores

**MIPS** Millones de instrucciones por segundo

$$MIPS = \frac{\text{recuento de instrucciones}}{\text{Tiempo de ejecución} \cdot 10^6}$$

- ◆ Los MIPS se muestran como un parámetro intuitivo para reflejar el rendimiento. Máquinas más rápidas tienen MIPS más altos.

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Alternativas para la medida del rendimiento

#### ◆ Considerando

$$\text{Recuento de instrucciones} = \frac{\text{tiempo de ejecución}}{\text{CPI} \cdot \text{ciclo de reloj}}$$

$$MIPS = \frac{\text{recuento de instrucciones}}{\text{Tiempo de ejecución} \cdot 10^6}$$

$$MIPS = \frac{\frac{\text{tiempo de ejecución}}{\text{CPI} \cdot \text{ciclo de reloj}}}{\text{Tiempo de ejecución} \cdot 10^6} = \frac{1}{\text{CPI} \cdot \text{ciclo de reloj} \cdot 10^6}$$

$$MIPS = \frac{\text{Frecuencia de reloj}}{\text{CPI} \cdot 10^6}$$

## 2.2 Evaluación del rendimiento

Métricas

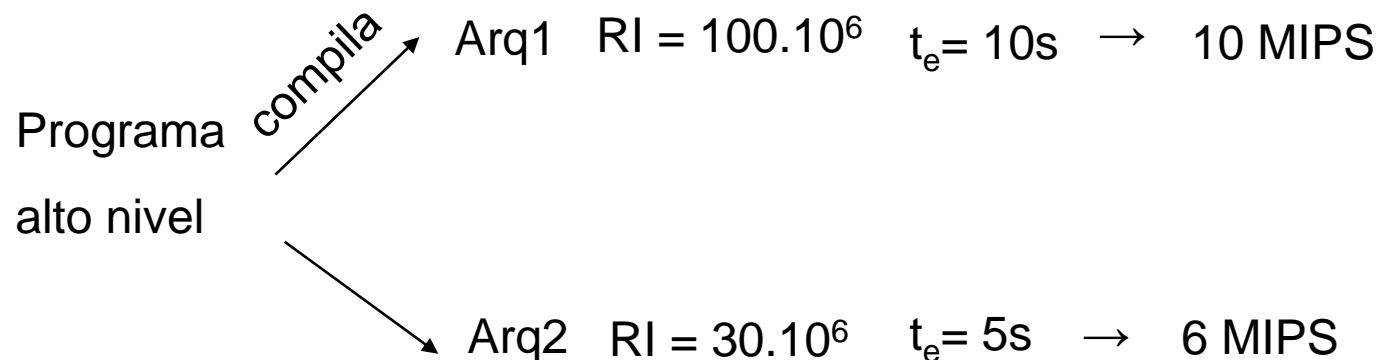
Benchmarks

Formulación

Análisis de  
rendimiento

### Problemas derivados de la utilización de los MIPS

- ❖ **Dependientes del repertorio de instrucciones.** No es aconsejable comparar los MIPS de computadores con repertorios de instrucciones diferentes.
  - ❖ Reflejan el ritmo de ejecución de instrucciones
  - ❖ No reflejan la efectividad del repertorio RI
  - ❖ Los MIPS pueden variar inversamente al rendimiento.
- ❖ **Dependencia del programa** en el mismo computador.
- ❖ **Ejemplo:**



## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

- ➊ **Ejemplo:** Supongamos que construimos un compilador optimizado para la máquina de carga/almacenamiento descrita en el ejemplo anterior. El compilador descarta el 50% de las instrucciones de la ALU aunque no pueda reducir cargas, almacenamientos ni saltos. Ignorando las prestaciones del sistema y suponiendo una duración del ciclo de reloj de 20 ns (frecuencia de reloj de 50Mhz). ¿Cuál es la frecuencia en MIPS para el código optimizado frente al código sin optimizar? ¿Está el criterio de los MIPS de acuerdo con el del tiempo de ejecución?

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### MIPS relativos y MIPS nativos

- ◆ **Tiempo referencia**= tiempo de ejecución de un programa en la máquina de referencia
- ◆ **Tiempo no estimado** = tiempo de ejecución del mismo programa en la máquina que se va a medir
- ◆ **MIPS** = estimación de los MIPS de la máquina de referencia

$$MIPS_{relativos} = \frac{Tiempo_{referencia}}{Tiempo_{no\ estimado}} \cdot MIPS_{referencia}$$

- ◆ Los MIPS relativos se apoyan en el tiempo de ejecución
- ◆ En los años 80 la máquina dominante como referencia era la VAX-11/780, denominada máquina de 1 MIPS.

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### FLOPS

**FLOPS** = Operaciones de punto flotante por segundo

$$MFLOPS = \frac{\text{Número de operaciones en punto flotante de un programa}}{\text{Tiempo de ejecución} \cdot 10^6}$$

$$GFLOPS = \frac{MFLOPS}{10^3}$$

- ➊ **Problemas** derivados de la utilización de los FLOPS
  - ➋ **Dependencia del repertorio**
  - ➋ Término basado en operaciones con objetivo de poder utilizarlo para comparar diferentes máquinas.
  - ➋ El conjunto de operaciones en punto flotante no es consistente en diferentes máquinas (CRAY-2 no tiene instrucciones de dividir mientras que el motorola 68882 si).

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### FLOPS

**FLOPS** = Operaciones de punto flotante por segundo

$$MFLOPS = \frac{\text{Número de operaciones en punto flotante de un programa}}{\text{Tiempo de ejecución} \cdot 10^6}$$

$$GFLOPS = \frac{MFLOPS}{10^3}$$

- ➊ **Problemas** derivados de la utilización de los FLOPS
  - ➋ **Dependencia del repertorio**
  - ➋ **Dependencia del programa**
  - ➋ La estimación de los MFLOPS cambia según la mezcla de operaciones rápidas y lentas en punto flotante del programa.
  - ➋ Si el 100% de las operaciones en punto flotante son sumas, la estimación será mayor que si el 100% son divisiones

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### FLOPS Normalizados

- ◆ Solución a los problemas de FLOPS
- ◆ Operaciones normalizadas (Livermore Loops)

Operaciones reales PF	Operaciones normalizadas PF
ADD, SUB, COMPARE, MULT	1
DIVIDE, SQRT,	4
EXP, SIN	8

#### ◆ Programa 1

$4 \cdot 10^6$  sumas

$3 \cdot 10^6$  div

$t_e = 10s$

#### MFLOPS Nativos

$$\frac{7 \cdot 10^6}{10 \cdot 10^6} = 0,7 \text{ MFLOPS}$$

#### MFLOPS Normalizados

$$\frac{16 \cdot 10^6}{10 \cdot 10^6} = 1,6 \text{ MFLOPS}$$

#### ◆ Programa 2

$7 \cdot 10^6$  div

$t_e = 17s$

#### MFLOPS Nativos

$$\frac{7 \cdot 10^6}{17 \cdot 10^6} = 0,4 \text{ MFLOPS}$$

#### MFLOPS Normalizados

$$\frac{28 \cdot 10^6}{17 \cdot 10^6} = 1,65 \text{ MFLOPS}$$

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

- ◆ **Ejemplo:** EL programa SPICE se ejecuta en la DECstation 3100 en 94 segundos. El número de operaciones en punto flotante ejecutadas en ese programa es el de la tabla.  
¿Cuántos son los MFLOPS nativos para ese programa? Usando las conversiones ¿Cuántos son los MFLOPS normalizados?.

ADDD	25.999.440
SUBD	18.266.439
MULD	33.880.810
DIVD	15.682.333
COMPARED	9.745.930
NEGD	2.617.846
ABSD	2.195.930
CONVERTD	1.581.450
<i>Total</i>	109.970.178

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Programas para evaluar el rendimiento

- ◆ Tiempo de ejecución de la carga de trabajo del usuario (workload) (mezcla de programas y órdenes del S.O.)
- ◆ **Programas reales.** compiladores de C, software de tratamiento de textos como TeX y herramientas CAD como Spice
- ◆ **Núcleos (Kernels).** pequeños fragmentos clave de programas. Livermore Loops y Linpack.
- ◆ **Benchmarks reducidos (toys).** 10 y 100 líneas de código. criba de Eratóstenes, Puzzle y clasificación rápida (quicksort).
- ◆ **Benchmarks Sintéticos.** se crean artificialmente intentando simular la frecuencia media de operaciones y operandos de un gran conjunto de programas. Whetstone y Dhrystone.
- ◆ Whetstone: instrucciones Algol principios de los años setenta.
- ◆ Dhrystone: Originalmente en ADA y más tarde en C y Pascal.

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Programas reales frente a otros benchmarks

- ➊ Importancia para las empresas de los benchmarks: empleo de recursos para optimizar el funcionamiento de estos pero no de programas reales.
- ➋ Ejemplo extremo: empleo de optimizadores de compiladores sensibles a los benchmarks que aplican optimizaciones.
- ➌ Si se utilizasen programas reales para evaluar el rendimiento, las mejoras repercutirían en el usuario final.

### Razones de la utilización de benchmarks pequeños

- ➊ **Portabilidad:** En el pasado lenguajes de programación inconsistentes entre máquinas dificultando el transporte
- ➋ **Fácil simulación:** Cuando se diseña una nueva máquina
- ➌ **Estandarización:** Los pequeños benchmarks más fácilmente
- ➍ En la actualidad la popularidad de los sistemas operativos estándares (UNIX, WINDOWS...) elimina la principal dificultad.

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Colecciones de benchmarks

- ➊ Medir el rendimiento de los procesadores con una variedad de aplicaciones
- ➋ Ventajas clave: la debilidad de algún benchmark es minimizada por la presencia de otros
- ➌ Las colecciones de benchmarks formadas por programas que pueden ser núcleos, pero fundamentalmente programas reales

## 2.2 Evaluación del rendimiento

### Colecciones de benchmarks

Métricas

Benchmarks

Formulación

Análisis de rendimiento

- ➊ **SPEC:** El grupo System Performance Evaluation Cooperative se formó en 1988 con representantes de diversas compañías (Hewlett-Packard, DEC, MIPS, Sun) que llegaron al acuerdo de ejecutar un conjunto de programas y entradas reales.
- ➋ SPEC89, SPEC92, SPEC95, SPEC CPU2000, SPEC CPU2006, SPEC CPU2017.
- ➌ Desktop Benchmarks
  - ➍ SPEC CPU 2006 (CINT, CFP)
  - ➍ SPECviewperf (OpenGL Graphics Library)
  - ➍ SPECCapc (ProEngineer;Solidworks;Unigraphics...)
- ➌ BenchMarks Para Servidores
  - ➍ SPECCrate (productividad)
  - ➍ SPECSFS (Rendimiento sistema ficheros de red)
  - ➍ SPECWeb (Rendimiento servidores web)
- ➌ BenchMarks sistemas embebidos

## 2.2 Evaluación del rendimiento

### Colecciones de benchmarks

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

#### ◆ Otras colecciones

- ◆ **Business Winstone:** Proceso por lotes Netscape, Corel, WordPerfect, Microsoft...
- ◆ **CC Winstone:** Aplicaciones creación contenido multimedia (Photoshop, Premiere, edición audio...)
- ◆ **Winbench:** Procesos por lotes miden rendimiento CPU, video, disco,... orientado a medida por subsistemas.

## 2.2 Evaluación del rendimiento

Métricas
Benchmarks
<b>Formulación</b>
Análisis de rendimiento

### Formulación de los resultados de la evaluación del rendimiento

- ➊ **Reproductibilidad:** enumerar todo lo necesario para repetir los experimentos.
- ➋ SPICE tarda 94 segundos en una DECstation 3100
- ➌ Este enunciado prescinde de aspectos como:
  - ➍ Entradas del programa
  - ➍ Versión del programa
  - ➍ Versión del compilador
  - ➍ Nivel de optimización y código compilado
  - ➍ Versión del sistema operativo
  - ➍ Cantidad de memoria principal
  - ➍ Número y tipos de disco
  - ➍ Versión de la CPU

## 2.2 Evaluación del rendimiento

### Formulación de los resultados de la evaluación del rendimiento

#### ● Informe SPEC:

- Descripción hardware
- Descripción software
- Descripción parámetros compilación
- Publicación resultados básicos y optimizados

Análisis de  
rendimiento

Métricas

Benchmarks

Formulación

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

### Formulación de los resultados de la evaluación del rendimiento

#### ◆ Informe SPEC:

##### ◆ Descripción hardware y software

#### Hardware

CPU Name: Intel Core i7-965 Extreme Edition  
CPU Characteristics: Intel Turbo Boost Technology up to 3.46 GHz  
CPU MHz: 3200  
FPU: Integrated  
CPU(s) enabled: 4 cores, 1 chip, 4 cores/chip, 2 threads/core  
CPU(s) orderable: 1 chip  
Primary Cache: 32 KB I + 32 KB D on chip per core  
Secondary Cache: 256 KB I+D on chip per core  
L3 Cache: 8 MB I+D on chip per chip  
Other Cache: None  
Memory: 12 GB (6 x 2GB Samsung M378B5673DZ1-CF8 DDR3-1066 CL7)  
Disk Subsystem: 80 GB Intel X-25M SATA Solid-State Drive  
Other Hardware: None

#### Software

Operating System: Windows Vista Ultimate w/ SP1 (64-bit)  
Compiler: Intel C++ Compiler Professional 11.0 for IA32 Build 20080930 Package ID: w\_cproc\_p\_11.0.054 Microsoft Visual Studio 2008 (for libraries)  
Auto Parallel: Yes  
File System: NTFS  
System State: Default  
Base Pointers: 32-bit  
Peak Pointers: 32-bit  
Other Software: None SmartHeap Library Version 8.1 from <http://www.microquill.com/>

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de rendimiento

### Formulación de los resultados de la evaluación del rendimiento

#### ● Informe SPEC:

- Descripción hardware y software
- Descripción parámetros compilación

```
Compiler Invocation

C benchmarks:
    icl -Qvc9 -Qc99

C++ benchmarks:
    icl -Qvc9

Base Optimization Flags

C benchmarks:
    -QxSSE4.2 -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qparallel
    -Qpar-runtime-control -Qvec-guard-write /F512000000

C++ benchmarks:
    -QxSSE4.2 -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qcxx-features
    /F512000000 shlw32m.lib           -link /FORCE:MULTIPLE

Peak Optimization Flags

C benchmarks:
    400.perlbench: -QxSSE4.2(pass 2) -Qprof_gen(pass 1) -Qprof_use(pass 2)
    -Qipo -O3 -Qprec-div- -Qansi-alias -Qopt-prefetch
    /F512000000 shlw32m.lib           -link /FORCE:MULTIPLE

    401.bzip2: -QxSSE4.2(pass 2) -Qprof_gen(pass 1) -Qprof_use(pass 2)
    -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qansi-alias
    /F512000000

    403.gcc: -QxSSE4.2(pass 2) -Qprof_gen(pass 1) -Qprof_use(pass 2)
    -Qipo -O3 -Qprec-div- /F512000000

    429.mcf: -QxSSE4.2 -Qipo -O3 -Qprec-div- -Qopt-prefetch
```

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

### Formulación de los resultados de la evaluación del rendimiento

#### ◆ Informe SPEC:

- ◆ Descripción hardware y software
- ◆ Descripción parámetros compilación
- ◆ Publicación resultados básicos y optimizados

Results Table

Benchmark	Base						Peak					
	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
400.perlbench	426	22.9	426	22.9	<u>426</u>	<u>22.9</u>	305	32.0	305	32.1	<u>305</u>	<u>32.0</u>
401.bzip2	526	18.3	<u>526</u>	<u>18.3</u>	526	18.3	517	18.7	518	18.6	<u>517</u>	<u>18.7</u>
403.gcc	<u>305</u>	<u>26.4</u>	305	26.4	302	26.7	264	30.5	267	30.1	<u>264</u>	<u>30.5</u>
429.mcf	189	48.3	<u>189</u>	<u>48.3</u>	190	48.0	189	48.2	192	47.5	<u>190</u>	<u>48.0</u>
445.gobmk	444	23.6	444	23.6	<u>444</u>	<u>23.6</u>	<u>396</u>	<u>26.5</u>	395	26.5	396	26.5
456.hmmer	496	18.8	495	18.9	<u>495</u>	<u>18.9</u>	392	23.8	<u>392</u>	<u>23.8</u>	392	23.8
458sjeng	494	24.5	494	24.5	<u>494</u>	<u>24.5</u>	472	25.6	472	25.6	<u>472</u>	<u>25.6</u>
462.libquantum	99.5	208	<u>99.6</u>	<u>208</u>	99.9	207	99.5	208	<u>99.6</u>	<u>208</u>	99.9	207
464.h264ref	599	37.0	599	36.9	<u>599</u>	<u>37.0</u>	538	41.1	<u>539</u>	<u>41.1</u>	539	41.1
471.omnetpp	255	24.5	<u>254</u>	<u>24.6</u>	254	24.6	218	28.7	<u>218</u>	<u>28.7</u>	217	28.7
473.astar	395	17.8	<u>395</u>	<u>17.8</u>	395	17.8	<u>355</u>	19.8	<u>356</u>	<u>19.7</u>	356	19.7
483.xalancbmk	232	29.8	231	29.9	<u>231</u>	<u>29.8</u>	232	29.8	231	29.9	<u>231</u>	<u>29.8</u>

Análisis de rendimiento

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

**Formulación**

Análisis de  
rendimiento

### Formulación de los resultados de la evaluación del rendimiento

	Computador A	Computador B	Computador C
Programa 1	1	10	20
Programa 2	1000	100	20
Tiempo total	1001	110	40

#### Afirmaciones:

- ➊ A es 900% más rápido que B para el programa 1.
- ➋ B es 900% más rápido que A para el programa 2.
- ➌ A es 1900% más rápido que C para el programa 1.
- ➍ C es 4900% más rápido que A para el programa 2.
- ➎ B es 100% más rápido que C para el programa 1.
- ➏ C es 400% más rápido que B para el programa 2.
- ➐ El contraste de las afirmaciones presenta un cuadro confuso.

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de rendimiento

### Resúmenes del rendimiento

#### ◆ Tiempo total de ejecución

	Computador A	Computador B	Computador C
Programa 1	1	10	20
Programa 2	1000	100	20
Tiempo total	1001	110	40

- ◆ B es 810% más rápido que A para los programas 1 y 2.
- ◆ C es 2400% más rápido que A para los programas 1 y 2.
- ◆ C es 175% más rápido que B para los programas 1 y 2.

#### ◆ Tiempo medio de ejecución

$$\frac{1}{n} \sum_{i=1}^n Tiempo_i$$

## 2.2 Evaluación del rendimiento

### Resúmenes del rendimiento

#### ◆ Tiempo de ejecución ponderado

- ◆ Asignar a cada programa un factor de peso  $w_i$  que indique la frecuencia relativa del programa en la carga de trabajo.

$$\sum_{i=1}^n w_i \cdot \text{Tiempo}_i$$

- ◆  $w_i$  = frecuencia del programa iésimo de la carga de trabajo
- ◆  $\text{Tiempo}_i$  = tiempo de ejecución del programa i-ésimo

#### ◆ Media Geométrica

$$\sqrt[n]{\prod_{i=1}^n \text{Tiempo}_i}$$

$$\frac{MG(x_i)}{MG(y_i)} = MG\left(\frac{x_i}{y_i}\right)$$

## 2.2 Evaluación del rendimiento

Métricas

Benchmarks

Formulación

Análisis de  
rendimiento

### Intel Core i7-965

Benchmark	Base Ref Time	Base Run Time	Base Selected
400.perlbench	9770	425.9	22.9396572
401.bzip2	9650	526.2	18.33903459
403.gcc	8050	305.2	26.37614679
429.mcf	9120	188.8	48.30508475
445.gobmk	10490	443.6	23.64743012
456.hammer	9330	494.7	18.8599151
458.sjeng	12100	494.2	24.48401457
462.libquantum	20720	99.6	208.0321285
464.h264ref	22130	598.9	36.95107697
471.omnetpp	6250	253.9	24.61599055
473.astar	7020	395.1	17.76765376
483.xalancbmk	6900	231.3	29.83138781

Media Aritmética	10960.83333	371.45	41.67912673	29.5082335
	MA(TR)	MA(TE)	MA(TR/TE)	MA(TR)/MA(TE)

Média Geométrica	10108.21274	334.1591446	30.2496966	30.2496966
	MG(TR)	MG(TE)	MG(TR/TE)	MG(TR)/MG(TE)

# Tema 3

# Diseño del repertorio de instrucciones

Arquitectura de los Computadores

# Tema 3. Diseño del repertorio de instrucciones

## Objetivos

- ➊ Analizar las arquitecturas desde el nivel de lenguaje máquina, aportando el punto de vista del diseñador de compiladores
- ➋ Comprender la influencia que ejercen los lenguajes y los compiladores sobre la arquitectura.
- ➌ Reflexionar sobre las ventajas e inconvenientes de los distintos enfoques para abordar el diseño de los repertorios de instrucciones, aportando una taxonomía de éstas.
- ➍ Conocer medidas que reflejen el distinto grado de utilización de los repertorios de instrucciones, dependiendo de la aplicación ejecutada.

# Tema 3. Diseño del repertorio de instrucciones

## Contenido

### ◆ **3.1 Introducción**

- ◆ 3.1.1 Introducción
- ◆ 3.1.2 Taxonomía
- ◆ 3.1.3 Arquitecturas GPR

### ◆ **3.2 Características del repertorio**

- ◆ 3.2.1 Direccionamiento de la memoria
- ◆ 3.2.2 Tipo y tamaño de los operandos
- ◆ 3.2.3 Operaciones

### ◆ **3.3 Evolución de la programación de los computadores**

- ◆ 3.3.1 Introducción
- ◆ 3.3.2 La arquitectura como objeto del compilador
- ◆ 3.3.3 Instrucciones de palabra muy larga (VLIW)

### ◆ **3.4 Ejemplos característicos**

- ◆ 3.4.1 DEC VAX
- ◆ 3.4.2 IBM 360/370
- ◆ 3.4.3 Intel x86
- ◆ 3.4.5 DLX

# Tema 3. Diseño del repertorio de instrucciones

## Debate inicial

- ➊ ¿Por qué no todas las máquinas tienen el mismo repertorio de instrucciones?
  - ➋ ¿Sería aconsejable esta situación?
- ➋ ¿Qué tipo de repertorio es mejor, un repertorio complejo o un repertorio simple?
- ➋ ¿En qué influye la forma de programar las máquinas?
  - ➋ ¿El compilador influye en el rendimiento?

# 3.1 Introducción

**Tema 3. Diseño del repertorio de instrucciones**

**Arquitectura de computadores**

### 3.1.1 Introducción

#### Definición

- ◆ **Arquitectura del repertorio de instrucciones (ISA / Instruction Set Architecture):**

- ◆ Se trata de la porción del computador visible por el programador o el diseñador de compiladores

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### 3.1.1 Introducción

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

## Áreas de aplicación

### • Escritorio

- ◆ **Énfasis** del rendimiento de los programas con tipos de datos **enteros** y de **punto flotante (FP)**,
- ◆ **Escasa preocupación** por el **tamaño** del programa o el **consumo** de energía

### • Servidores

- ◆ Bases de datos, servidor de archivos, aplicaciones web...
- ◆ El rendimiento del **FP** es mucho **menos importante** que el rendimiento para enteros o cadenas de caracteres

### • Aplicaciones embebidas

- ◆ **Valoran coste y potencia**
- ◆ Tamaño del código es importante -> menos memoria -> más barato y menos consumo
- ◆ Además, algunas clases de instrucciones (como FP) pueden ser opcionales para reducir costes del chip

## 3.1.2 Taxonomía de las arquitecturas a nivel ISA

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

- ◆ **Almacenamiento de operandos en la CPU**
  - ◆ GPR, pila, acumulador
- ◆ **Operandos explícitos**
  - ◆ 0,1,2,3
- ◆ **Posición del operando**
  - ◆ R-R, R-M, M-M
- ◆ **Operaciones**
  - ◆ CISC-RISC
- ◆ **Tipo y tamaño de los operandos**
  - ◆ Enteros, PF, decimales, caracteres, cadenas...

## 3.1.2 Taxonomía de las arquitecturas a nivel ISA

Introducción

Características

Programación

Ejemplos

### Tipo de almacenamiento interno de la CPU

- ◆ El tipo de almacenamiento interno es la **diferenciación más básica**
- ◆ **Arquitectura de pila:** Los operandos están implícitamente en la cima de la pila
- ◆ **Arquitectura de acumulador:** Un operando está implícitamente en el acumulador
- ◆ **Arquitectura de registros de propósito general (GPR):** Tienen sólo operandos explícitos en registros o en posiciones de memoria

## 3.1.2 Taxonomía de las arquitecturas a nivel ISA

Introducción

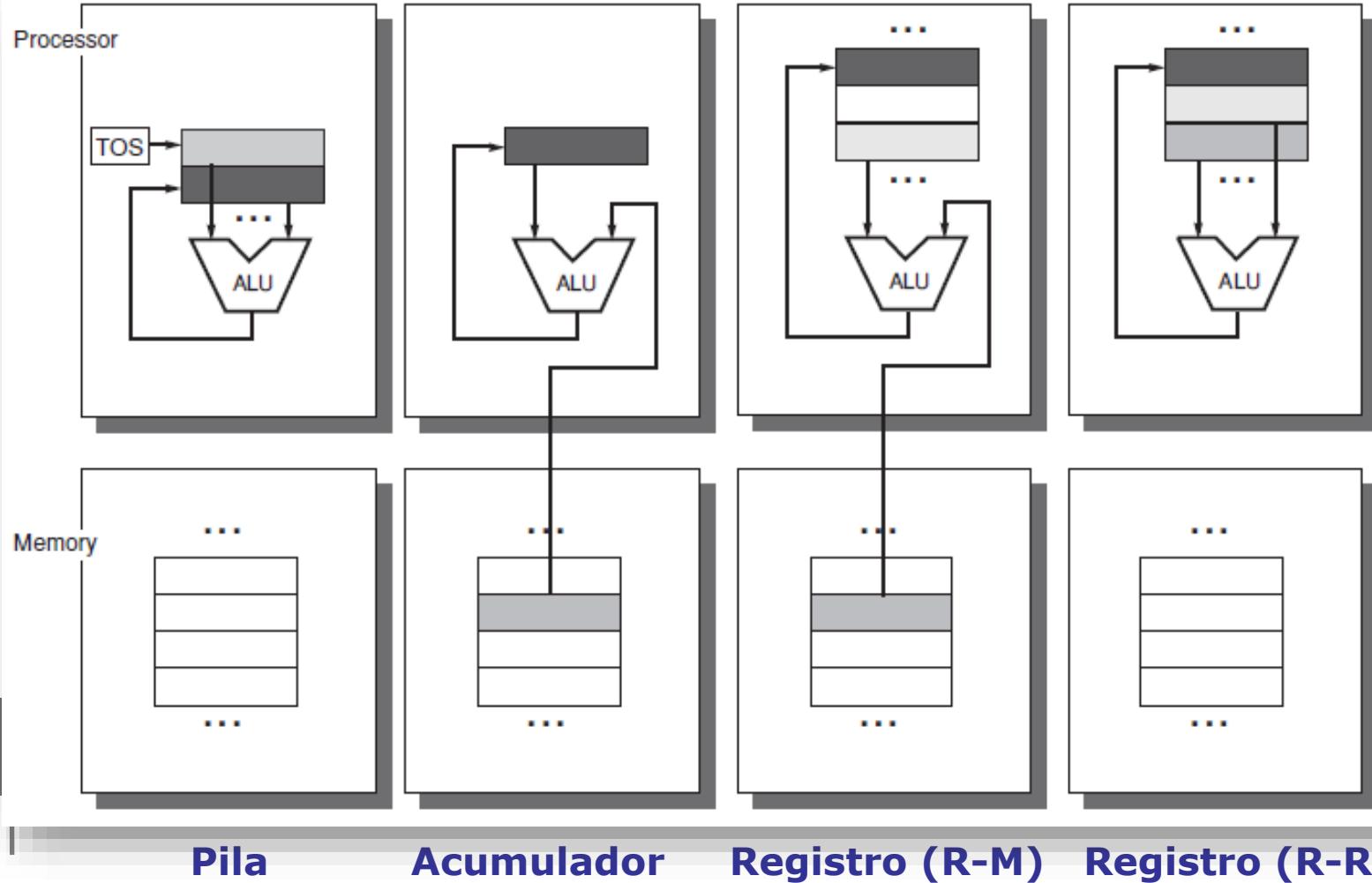
Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### Tipo de almacenamiento interno de la CPU



## 3.1.2 Taxonomía de las arquitecturas a nivel ISA

Introducción

Características

Programación

Ejemplos

### Tipo de almacenamiento interno de la CPU

- **Cuatro ejemplos:** Secuencia de código  $C=A+B$  en las cuatro clases de repertorios de instrucciones

Pila	Acumulador	Registro (R-M)	Registro (R-R)
Push A	Load A	Load R1, A	Load R1,A
Push B	Add B	Add R1, B	Load R2,B
Add	Store C	Store C, R1	Add R3,R1,R2
Pop C			Store R3,C

## 3.1.2 Taxonomía de las arquitecturas a nivel ISA

Introducción

Características

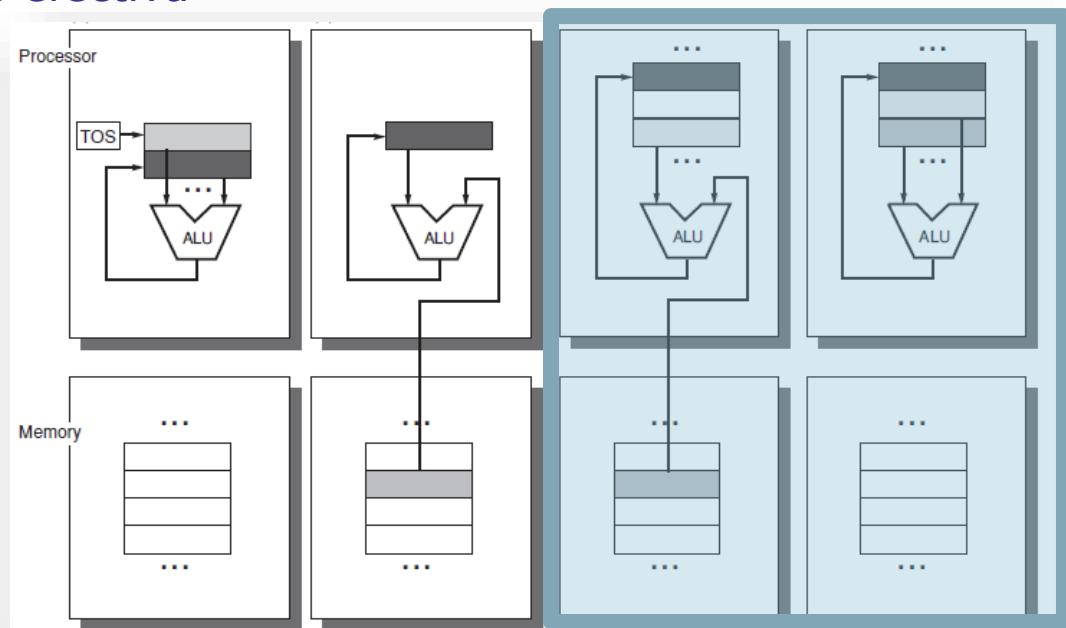
Programación

Ejemplos

Diseño del repertorio de instrucciones

### Tendencia actual GPR

- ◆ **Máquinas más antiguas** arquitecturas **pila y acumulador**
- ◆ **A partir de 1980** frecuentemente arquitecturas **GPR**
  - ◆ Los registros tienen acceso más rápido que la memoria
  - ◆ Los registros son más fáciles de utilizar por los compiladores y de manera más efectiva



### 3.1.3 Arquitecturas GPR

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

- ◆ **Ventaja: utilización efectiva de registros por el compilador**

- ◆ **Ubicación de variables:** reduce el tráfico de memoria y acelera el programa (los registros son más rápidos que la memoria) (Ej. Bucle del algoritmo de la burbuja)
- ◆ **Evaluar expresiones:** Los registros permiten una ordenación más flexible que las pilas o acumuladores (almacenamiento temporal subexpresiones)
- ◆ **Densidad de código:** Un registro se nombra con menos bits que una posición de memoria
- ◆ **Registros no reservados:** Los escritores de compiladores prefieren que los registros sean no reservados para ubicar las variables de forma más flexible (Ej. Registro EBX en x86)

### 3.1.3 Arquitecturas GPR

Introducción

Características

Programación

Ejemplos

◆ **Número de registros necesario:** depende del uso del compilador reservando registros para:

- ◆ Evaluar expresiones
- ◆ Paso de parámetros
- ◆ Ubicar variables. Según algoritmo de ubicación utilizado
  - Appendix A. Pg. 525-531 Hennessy 5th ed.

How many registers are sufficient? The answer, of course, depends on the effectiveness of the compiler. Most compilers reserve some registers for expression evaluation, use some for parameter passing, and allow the remainder to be allocated to hold variables. Modern compiler technology and its ability to effectively use larger numbers of registers has led to an increase in register counts in more recent architectures.

- MIPS 32 Int 32 FP
- <https://en.wikipedia.org/wiki/AVX-512>
- [AMD Ryzen](#) 160 RFP y 168 RInt

Diseño del repertorio de instrucciones

### 3.1.3 Arquitecturas GPR

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

#### Clasificación de las arquitecturas GPR

- ◆ **Número de operandos** de instrucciones ALU
- ◆ **Número de operandos** que se pueden direccionar **en memoria** en instrucciones ALU. (0..3)

### 3.1.3 Arquitecturas GPR

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

#### Clasificación de las arquitecturas GPR

- ◆ **Número de operandos** de instrucciones ALU
  - ◆ **Tres operandos:** Un resultado y dos fuentes
    - ◆ ADD R1,R2,R3
  - ◆ **Dos operandos:** Un operando es fuente y destino
    - ◆ ADD R1,R2

### 3.1.3 Arquitecturas GPR

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

#### Clasificación de las arquitecturas GPR

- ➊ **Número de operandos** de instrucciones ALU
- ➋ **Número de operandos** que se pueden direccionar **en memoria** en instrucciones ALU. (0..3)
  - ➌ **Registro-registro (carga almacenamiento):** Sin referencia a memoria para instrucciones ALU. **Solo** registros de la CPU
    - ADD R1,R2
  - ➍ **Registro-memoria:** Se permite un solo operando referenciando la memoria.
    - ADD R1,MEM
  - ➎ **Memoria-memoria:** Se permite más de un operando referenciando la memoria. (2 o 3)
    - ADD MEM1,MEM2

### 3.1.3 Arquitecturas GPR

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

#### Ventajas y desventajas de las arquitecturas GPR

Tipo	Ventajas	Desventajas
R-R	<ul style="list-style-type: none"><li>Codificación simple, instrucciones de longitud fija.</li></ul>	<ul style="list-style-type: none"><li>Mayor recuento de instrucciones que las arquitecturas con referencias a memoria.</li></ul>
M-M	<ul style="list-style-type: none"><li>No se emplean registros para temporales.</li><li>Código más compacto.</li></ul>	<ul style="list-style-type: none"><li>Gran variación en el tamaño de las instrucciones.</li><li>Gran variación en el trabajo por instrucción.</li><li>Los accesos a memoria crean cuellos de botella en memoria.</li></ul>

### 3.1.3 Arquitecturas GPR

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

#### Ventajas y desventajas de las arquitecturas GPR

Tipo	Ventajas	Desventajas
R-R	<ul style="list-style-type: none"><li>•Codificación simple, instrucciones de longitud fija.</li><li>•Las instrucciones emplean números de ciclos similares para ejecutarse.</li></ul>	<ul style="list-style-type: none"><li>•Mayor recuento de instrucciones que las arquitecturas con referencias a memoria.</li></ul>
R-M	<ul style="list-style-type: none"><li>•Los datos pueden ser accedidos sin cargarlos primero.</li></ul>	<ul style="list-style-type: none"><li>•Se destruye un operando fuente.</li><li>•Codificar un número de registro y una dirección de memoria en cada instrucción puede restringir el número de registros.</li><li>•Los ciclos de instrucción varían según los operandos</li></ul>
M-M	<ul style="list-style-type: none"><li>•No se emplean registros para temporales.</li><li>•Código más compacto.</li></ul>	<ul style="list-style-type: none"><li>•Gran variación en el tamaño de las instrucciones.</li><li>•Gran variación en el trabajo por instrucción.</li><li>•Los accesos a memoria crean cuellos de botella en memoria.</li></ul>

## 3.2 Características

**Tema 3. Diseño del repertorio de instrucciones**

**Arquitectura de computadores**

## 3.2.1 Direccionamiento de la memoria

Introducción

**Características**

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### 3.2 Características del repertorio

- ◆ 3.2.1 Direccionamiento de la memoria
- ◆ 3.2.2 Tipo y tamaño de los operandos
- ◆ 3.2.3 Operaciones

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

- ➊ Arquitecturas direccionan por bytes y proporcionan acceso a bytes (8 bits), medias palabras (16 bits), palabras (32 bits) y dobles palabras (64 bits)
- ➋ **a. Ordenación de los bytes**
  - ➌ Dos convenios para ordenar los bytes de una palabra: "**LittleEndian**" y "**BigEndian**"
  - ➍ Estos términos provienen de un famoso **artículo de Cohen[1981]** que establece una **analogía** entre la discusión sobre por que extremo de byte comenzar y la discusión **de los Viajes de Gulliver sobre que extremo del huevo abrir**

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

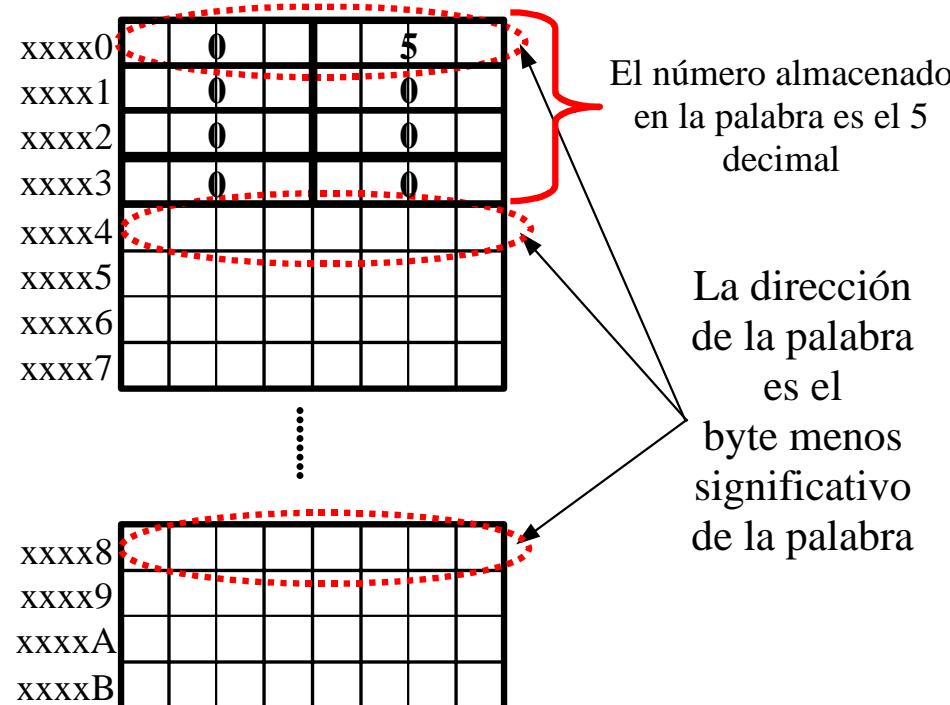
Programación

Ejemplos

Diseño del repertorio de instrucciones

### a. Ordenación de los bytes

- La ordenación **Little endian** (extremo pequeño)
  - La dirección de un dato es la del **byte menos significativo**
  - DEC PDP11, VAX y 80x86 siguen el modelo Little endian



## 3.2.1 Direccionamiento de la memoria

Introducción

Características

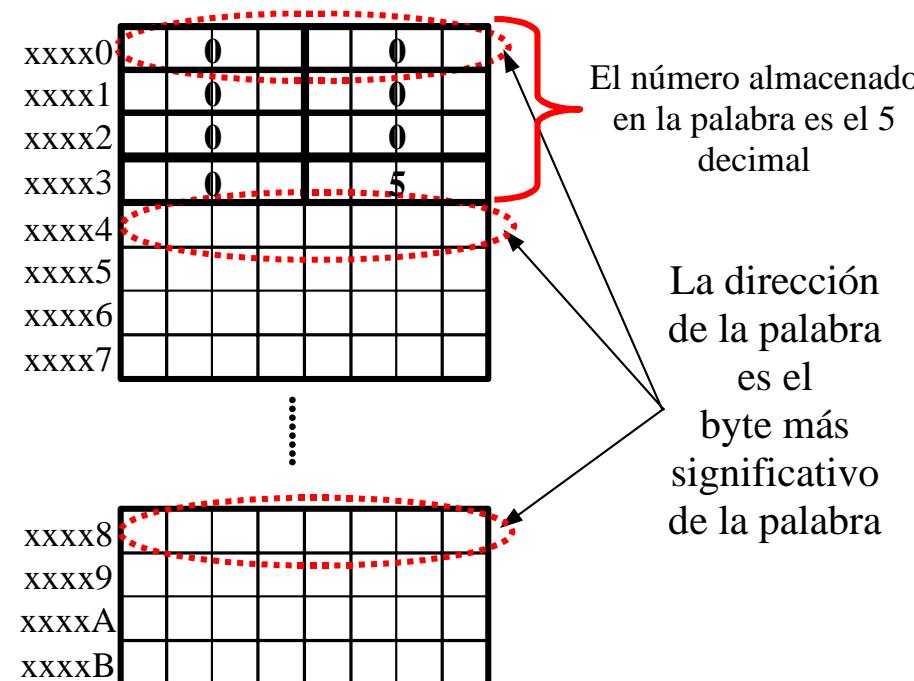
Programación

Ejemplos

Diseño del repertorio de instrucciones

### a. Ordenación de los bytes

- La ordenación **Big endian** (extremo grande)
  - La dirección de un dato es la del byte más significativo
  - IBM 360/370, los Motorola 680x0 siguen el modelo Big endian



## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

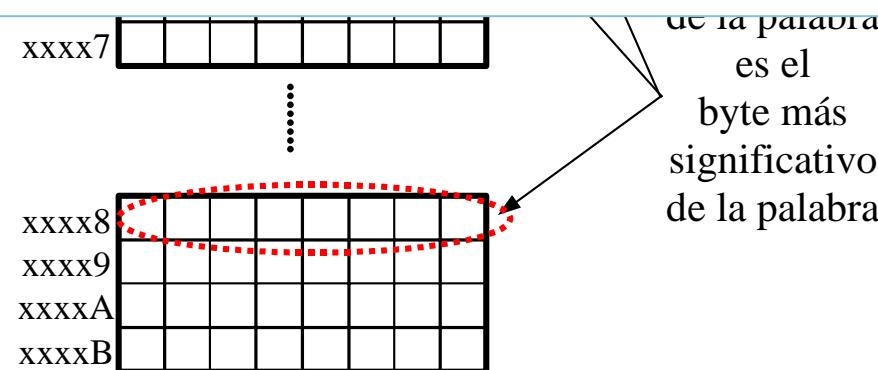
Diseño del repertorio de instrucciones

### a. Ordenación de los bytes

#### La ordenación **Big endian** (extremo grande)

- La dirección de un dato es la del **byte más significativo**

**La ordenación de los bytes puede ser problema cuando se intercambian datos entre máquinas con diferentes ordenaciones**



### 3.2.1 Direccionamiento de la memoria

Introducción

Características

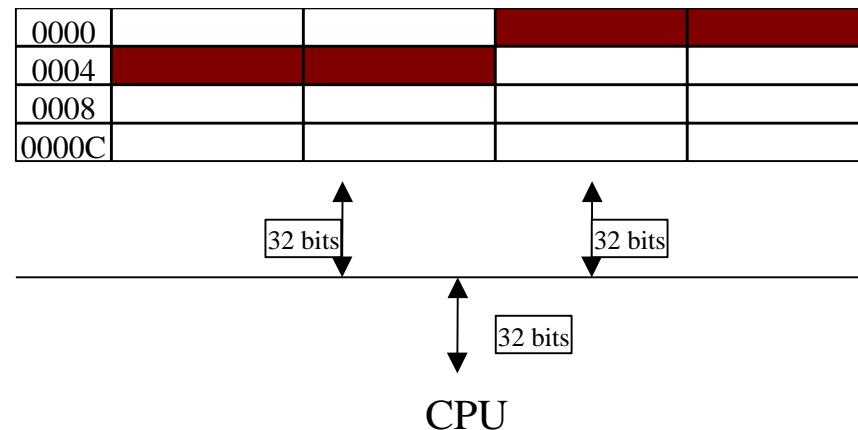
Programación

Ejemplos

Diseño del repertorio de instrucciones

#### Alineamiento de los accesos a los objetos de memoria

- ◆ Un acceso a un objeto mayor de un byte en la **dirección A** y en una memoria de tamaño **n** bytes (**ancho de palabra**) en su **bus de datos**, esta **alineado**, si la dirección **A mod n = 0**
- ◆ El **acceso no alineado** a los datos **puede empeorar el tiempo** de ejecución del programa debido a la necesidad de realizar varios accesos a memoria para completar un acceso.
- ◆ **Ejemplo:** Que ocurre en un sistema con un bus de datos de 32 bits al acceder a una palabra no alineada.



### 3.2.1 Direccionamiento de la memoria

Introducción

Características

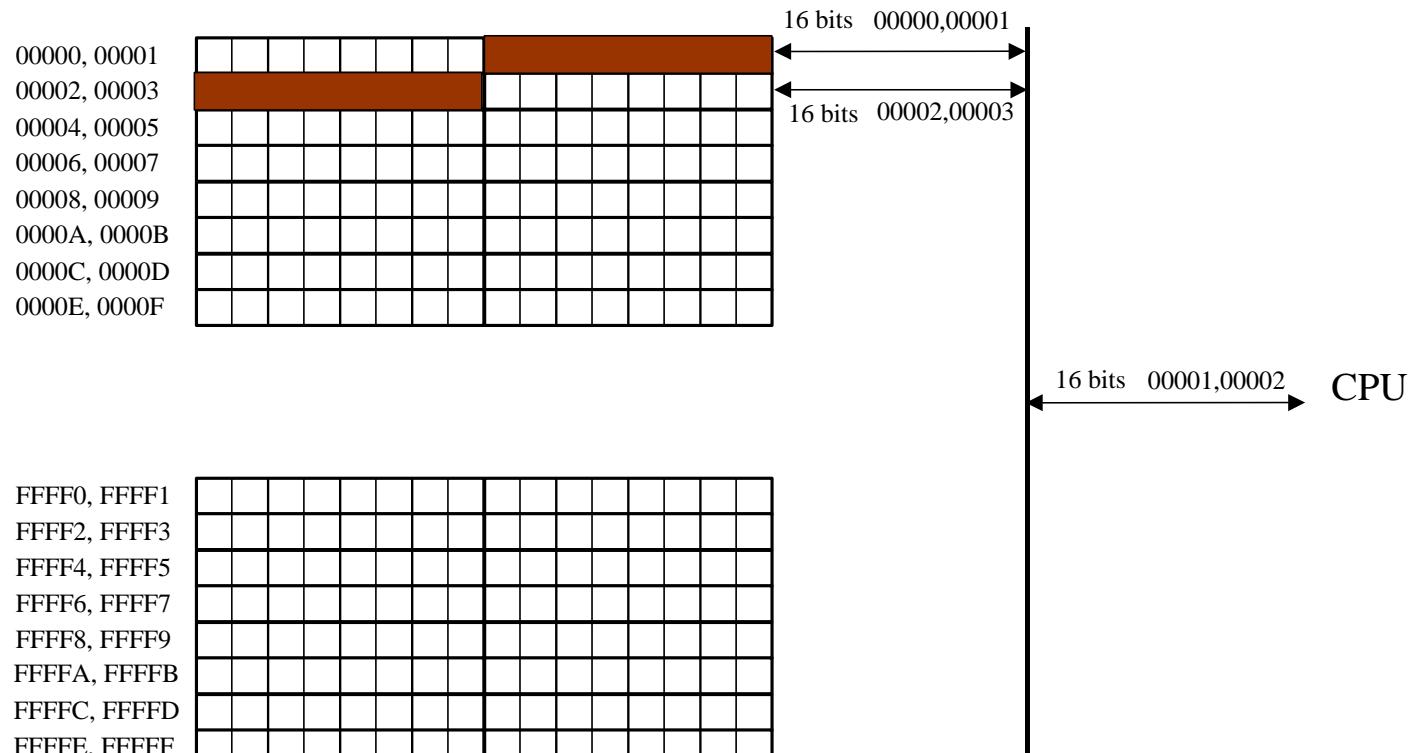
Programación

Ejemplos

Diseño del repertorio de instrucciones

#### Alineamiento de los accesos a los objetos de memoria

- ◆ **Ejemplo:** Que ocurre en el 80x86 cuando se realiza un acceso a una palabra no alineada (sistema con un bus de 16 bits a memoria)



## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### b. Modos de direccionamiento

- ➊ **Modos de direccionamiento:** forma en que las arquitecturas especifican la dirección de un objeto
- ➋ En las arquitecturas GPR **un modo de direccionamiento puede especificar:**
  - ➌ **Constante, registros o posiciones de memoria**
  - ➍ En caso de ser una posición de memoria, la dirección real especificada por el modo de direccionamiento se denomina **dirección efectiva**.

### 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

#### b. Modos de direccionamiento

- Los nombres de los modos de direccionamiento de la tabla pueden diferir entre arquitecturas

Modo de direccionamiento	Ejemplo	Significado	Cuando se usa
Registro	Add R4, R3	$R4 \leftarrow R4 + R3$	Cuando un valor está en un registro
Inmediato o literal	Add R4, #3	$R4 \leftarrow R4 + 3$	Para constantes. En algunas máquinas, literal e inmediato son dos modos diferentes de direccionamiento
Desplazamiento	Add R4, 100(R1)	$R4 \leftarrow R4 + M[100+R1]$	Acceso a variables locales
Registro diferido o indirecto	Add R4, (R1)	$R4 \leftarrow R4 + M[R1]$	Acceso utilizando un puntero o una dirección calculada
Indexado	Add R3, (R1+R2)	$R3 \leftarrow R3 + M[R1+R2]$	A veces útil en direccionamiento de arrays- R1 base del array; R2 índice.
Directo o absoluto	Add R1, (1001)	$R1 \leftarrow R1 + M[1001]$	A veces útil para acceder a datos estáticos; la constante que especifica la dirección puede necesitar ser grande

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### b. Modos de direccionamiento

- Los nombres de los modos de direccionamiento de la tabla pueden diferir entre arquitecturas

Modo de direccionamiento	Ejemplo	Significado	Cuando se usa
Indirecto o diferido de memoria	Add R1, @(R3)	$R1 \leftarrow R1 + M[R3]$	Si R3 es la dirección de un puntero p, entonces el modo obtiene *p
Autoincremento	Add R1, (R2)+	$R1 \leftarrow R1 + M[R2]$ $R2 \leftarrow R2 + d$	Util para recorridos de arrays en un bucle. R2 apunta al principio del array; cada referencia incrementa R2 en el tamaño de un elemento, d.
Autodecremento	Add R1, -(R2)	$R2 \leftarrow R2 - d$ $R1 \leftarrow R1 + M[R2]$	El mismo uso que autoincremento. Autoincremento/decremento también puede utilizarse para realizar una pila mediante introducir y sacar (push y pop)
Escalado o índice	Add R1, 100 (R2)[R3]	$R1 \leftarrow R1 + M[100 + R2 + R3 * d]$	Usado para acceder a arrays por índice. Puede aplicarse a cualquier modo de direccionamiento indexado en algunas máquinas.

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### b. Modos de direccionamiento (ejemplos)

Dir.	Valor	Registros
1000	1012	R1=1012
1004	4	R2=1004
1008	8	R3=2
1012	1004	<b>d = 4 (palabras de 32 bits)</b>
1016	1020	
1020	16	
1024	0	

	Mod dir	efectiva	→	valor
Desp	8(R1):	1020	→	16
Autoinc	(R1)+:	1012	→	1004 (después R1 sería igual a 1016)
Indir	@(R1):	1004	→	4
Esca	4(R2)[R3]:	1016	→	1020

### 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

#### ◆ b. Modos de direccionamiento

- ◆ Los **modos de direccionamiento reducen el RI pero complican la implementación** pudiendo incrementar el CPI medio
- ◆ El arquitecto de computadores debe **elegir que modos de direccionamiento** incluir **en base a estudios de frecuencia de utilización**

### 3.2.1 Direccionamiento de la memoria

Introducción

Características

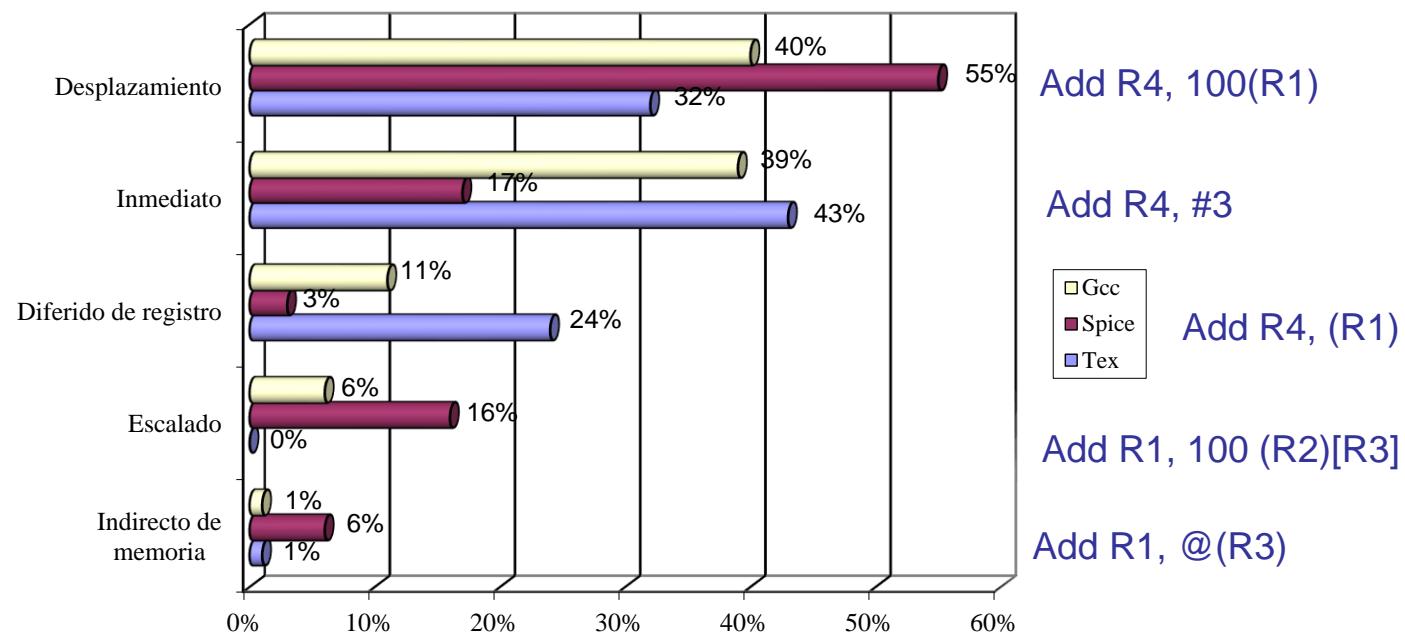
Programación

Ejemplos

Diseño del repertorio de instrucciones

#### b. Modos de direccionamiento

- ◆ Frecuencia de utilización de los modos de direccionamiento. SPEC (gcc, spice, Tex) en el VAX. Medidas independientes de arquitectura
- ◆ El **direccionamiento inmediato y desplazamiento dominan la utilización de los modos de direccionamiento**



### 3.2.1 Direccionamiento de la memoria

Introducción

Características

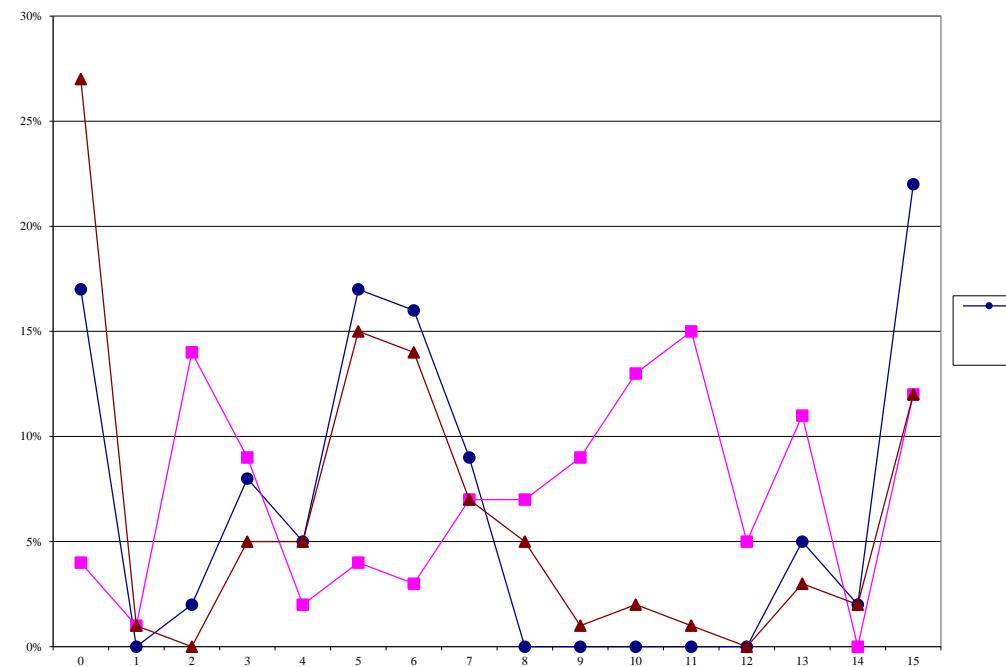
Programación

Ejemplos

Diseño del repertorio de instrucciones

#### c. Modo de direccionamiento desplazamiento

- ¿Cuál es el **rango más frecuente de desplazamientos** en este modo de direccionamiento?
- La respuesta indicará que tamaño soportar** (afecta a la longitud de la instrucción)



### 3.2.1 Direccionamiento de la memoria

Introducción

Características

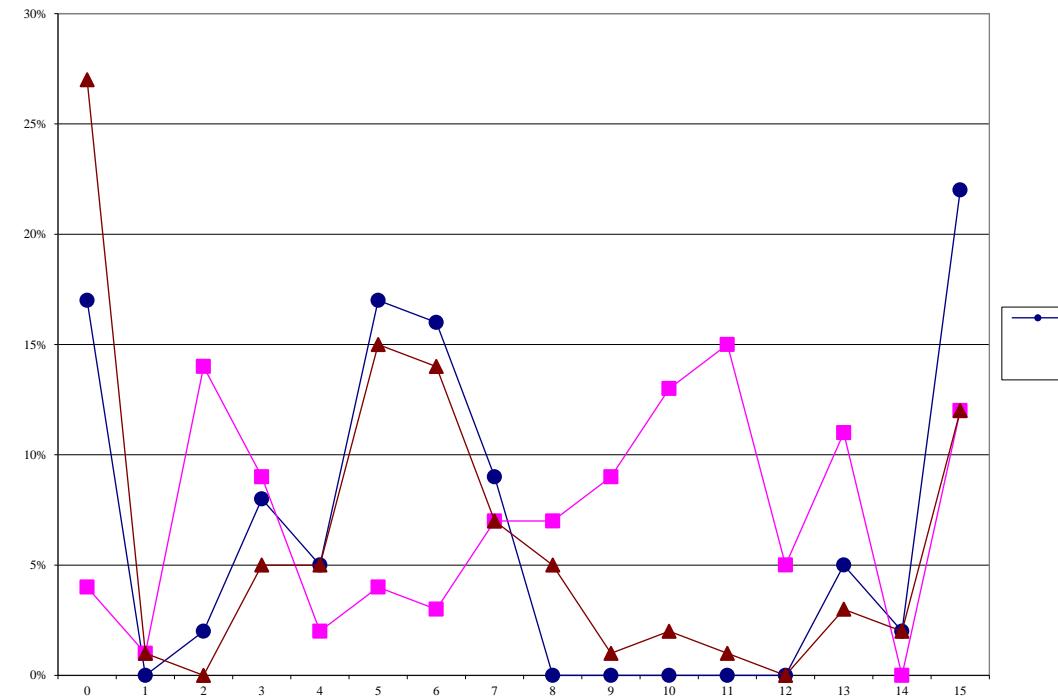
Programación

Ejemplos

Diseño del repertorio de instrucciones

#### c. Modo de direccionamiento desplazamiento

- Los **desplazamientos** están **ampliamente distribuidos**
- eje x **log2 desplazamiento (tamaño campo desplazamiento)**
- VAX (8,16,32); IBM360 (12) ; DLX (16); 80x86 (8,16).



### 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

#### d. Modo de direccionamiento literal o inmediato

- ➊ Los **inmediatos se utilizan** frecuentemente en:
  - ➊ **Operaciones aritméticas** (ADD R1,R2,#3)
  - ➋ **Comparaciones** (principalmente para saltos) (CMP R1,#0)
  - ➌ **Transferencias** para poner una constante en un registro. (MOV R1,#1)
    - ➊ **Constantes** escritas en el código que tienden a ser pequeñas
    - ➋ Constantes de **direcciones** que pueden ser grandes
- ➋ Dos cuestiones:
  - ➊ ¿**Que operaciones** necesitan **soportar inmediatos**?
  - ➋ ¿**Qué rango** de valores es necesario para los inmediatos?

### 3.2.1 Direccionamiento de la memoria

Introducción

Características

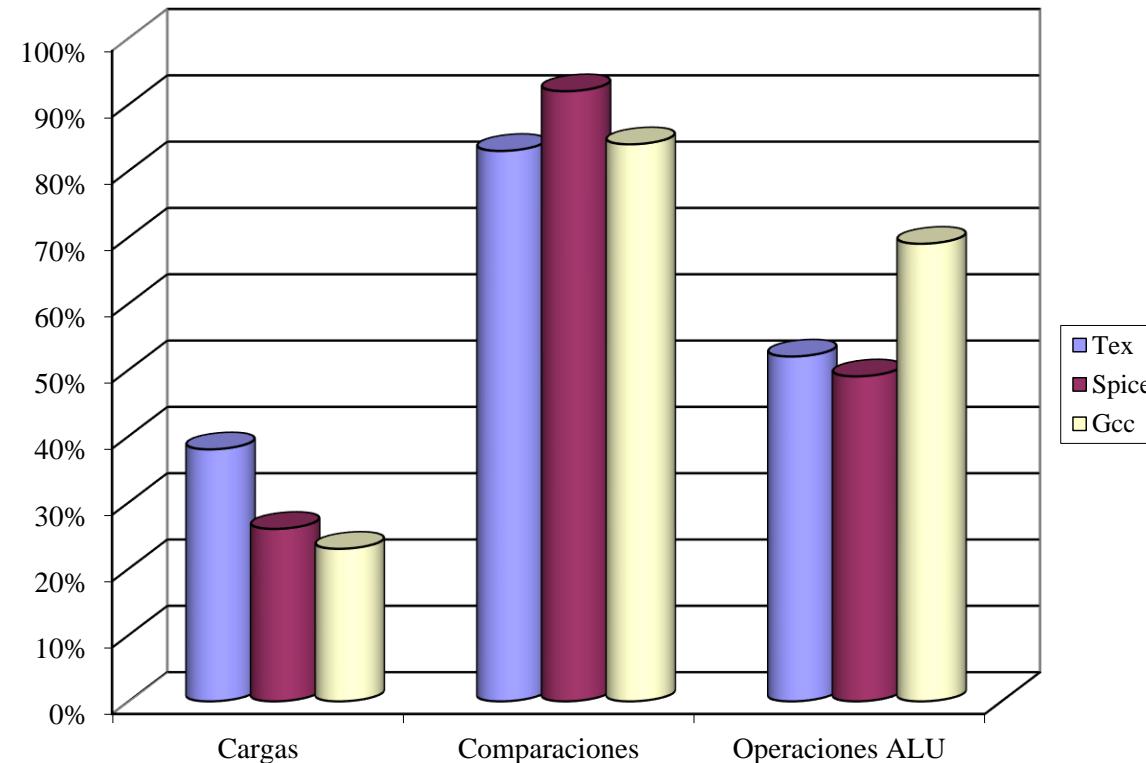
Programación

Ejemplos

Diseño del repertorio de instrucciones

#### d. Modo de direccionamiento literal o inmediato

- ¿Que operaciones necesitan soportar inmediatos?



## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

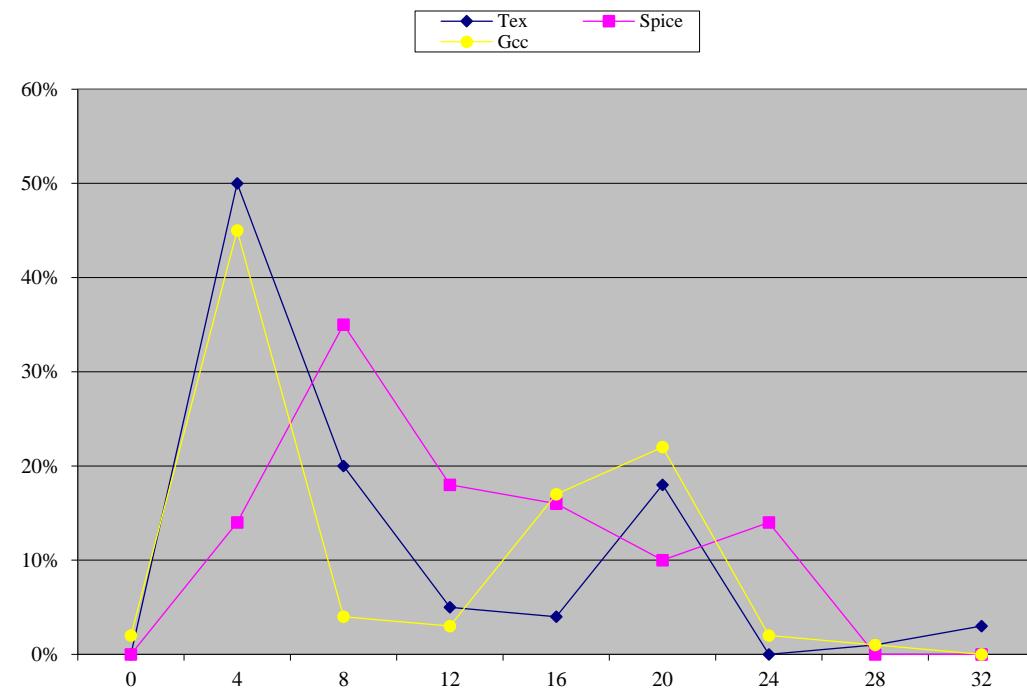
Ejemplos

Diseño del repertorio de instrucciones

### d. Modo de direccionamiento literal o inmediato

- ¿Qué **rango** de valores es necesario para los inmediatos?
  - El **tamaño** de los inmediatos **afecta a la longitud de la instrucción**
  - Distribución de valores inmediatos: Los **inmediatos pequeños** son los **más utilizados**, aunque se usan inmediatos grandes en el cálculo de direcciones.

- VAX (8,16,32)
- IBM360 (8)
- DLX (16)
- 80x86 (8,16)



## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

**Resumen modos:** Appendix A. Pg. 536 Hennessy 5th ed.

**1.-** Because of their popularity, we would expect a new architecture to support **at least** the following addressing modes: **displacement, immediate, and register indirect**. They represent 75% to 99%

**2.-** We would expect the **size of the address for displacement mode** to be at least **12 to 16 bits**, since these sizes would capture 75% to 99% of the displacements

**3.-** We would expect the **size of the immediate field** to be at least **8 to 16 bits**.

Diseño del  
repertorio de  
instrucciones

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### e. Codificación de los modos de direccionamiento

- **Codificación incluida en el código de operación:** Para un pequeño número de combinaciones modo de direccionamiento/código de operación, el modo de direccionamiento puede codificarse en el código de operación
- **Especificador de direcciones separado para cada operando:** En muchas ocasiones se necesita este especificador para indicar el modo de direccionamiento que está usando cada operando

### ◆ El arquitecto debe equilibrar

- El **interés de disponer del mayor número posible de registros y modos de direccionamiento**
- El **impacto del tamaño de los campos de los registros y de los modos de direccionamiento en el tamaño medio de la instrucción**
- El **interés de tener instrucciones codificadas en longitudes fáciles de manejar e implementar**

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### e. Codificación de los modos de direccionamiento

a) Variable (VAX, Intel 80x86)

Operación y nº de operandos	Especificador de dirección 1	Campo de dirección 1	.....	Especificador de dirección 1	Campo de dirección 1
-----------------------------	------------------------------	----------------------	-------	------------------------------	----------------------

b) Fijo (Alpha, ARM, MIPS, PowerPC, SPARC )

Operación	Campo de dirección 1	Campo de dirección 2	Campo de dirección 3
-----------	----------------------	----------------------	----------------------

c) Hibrido (IBM 360,370)

Operación	Especificador de dirección	Campo de dirección	
Operación	Especificador de dirección	Campo de dirección 1	Campo de dirección 2

◆ **Variable:** cualquier modo de direccionamiento con cualquier operador.

◆ Interesante con número alto de modos de direccionamiento y operaciones. Consigue menor RI pero las instrucciones individuales varían en talla y cantidad de trabajo. Ejemplo VAX.

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### e. Codificación de los modos de direccionamiento

a) Variable (VAX, Intel 80x86)

Operación y nº de operandos	Especificador de dirección 1	Campo de dirección 1	.....
-----------------------------	------------------------------	----------------------	-------

Especificador de dirección 1	Campo de dirección 1
------------------------------	----------------------

b) Fijo (Alpha, ARM, MIPS, PowerPC, SPARC )

Operación	Campo de dirección 1	Campo de dirección 2	Campo de dirección 3
-----------	----------------------	----------------------	----------------------

c) Hibrido (IBM 360,370)

Operación	Especificador de dirección	Campo de dirección
-----------	----------------------------	--------------------

Operación	Especificador de dirección	Campo de dirección 1	Campo de dirección 2
-----------	----------------------------	----------------------	----------------------

- ◆ **Fija:** Combina la operación y el modo de direccionamiento en el código de operación.
- ◆ Tamaño único para todas las instrucciones. Interesante con número reducido de modos de direccionamiento y operaciones. Fáciles de decodificar e implementar pero conducen a RI altos. Ejemplo MIPS.

## 3.2.1 Direccionamiento de la memoria

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### e. Codificación de los modos de direccionamiento

a) Variable (VAX, Intel 80x86)

Operación y nº de operandos	Especificador de dirección 1	Campo de dirección 1	.....	Especificador de dirección n	Campo de dirección n
-----------------------------	------------------------------	----------------------	-------	------------------------------	----------------------

b) Fijo (Alpha, ARM, MIPS, PowerPC, SPARC )

Operación	Campo de dirección 1	Campo de dirección 2	Campo de dirección 3
-----------	----------------------	----------------------	----------------------

c) Híbrido (IBM 360,370)

Operación	Especificador de dirección	Campo de dirección
-----------	----------------------------	--------------------

Operación	Especificador de dirección	Campo de dirección 1	Campo de dirección 2
-----------	----------------------------	----------------------	----------------------

- ◆ **Híbrida:** Esta alternativa reduce la variabilidad en talla y trabajo proporcionando varias longitudes de instrucción.
- ◆ Es una alternativa intermedia que persigue las ventajas de las anteriores: reducir recuento de instrucciones y formato sencillo de fácil implementación. Ejemplo IBM 360.

## 3.2.2 Tipo y tamaño de los operandos

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

**Forma de designar el tipo de operando.** Dos alternativas:

- ◆ **En el código de operación**

- El tipo de operando se expresa en el código de operación. Es el método utilizado con más frecuencia

- ◆ **Datos identificados o autodefinidos**

- El dato se anota con identificadores que especifican el tipo de cada operando y que son interpretados por el hardware
- Son extremadamente raras. Arquitecturas de Burroughs. Symbolics para implementaciones LISP

## 3.2.2 Tipo y tamaño de los operandos

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### Tamaños más comunes de los operandos:

- **Byte** (8 bits)
- **Media palabra** (16 bits)
- **Palabra** (32 bits)
- **Doble palabra** (64 bits)

## 3.2.2 Tipo y tamaño de los operandos

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### Codificaciones más comunes de los operandos:

#### • Carácteres

- **EBCDIC**: usado en las arquitecturas de grandes computadores IBM
- **ASCII**: (128 ASCII estandar y 256 ASCII extendido). Muy difundido
- **16-bit Unicode**: usado en Java → gana popularidad

#### • Enteros: Representación en **complemento a 2** muy difundida

#### • Punto flotante: **754 de IEEE** (estándar más difundido)

- **Precisión simple**: 32 bits (1+8+23 signo, exponente, mantisa).
- **Precisión doble**: 64 bits (1+11+52 signo, exponente, mantisa).
- **Precisión simple extendida**
- **Precisión doble extendida**
- **Formatos extendidos**: para evitar errores y desbordamientos en operaciones intermedias aumentando el número de bits de mantisa y exponente. Dependen de implementaciones

## 3.2.2 Tipo y tamaño de los operandos

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### Codificaciones más comunes de los operandos:

- **Cadenas de caracteres:** Algunas arquitecturas soportan operaciones sobre **cadenas de caracteres ASCII** (comparaciones, desplazamientos...)
- **Decimales:** Algunas arquitecturas soportan un formato denominado habitualmente **decimal empaquetado (BCD)**. Se utilizan 4 bits para codificar los valores 0-9, y en cada byte se empaquetan dos dígitos decimales

For business applications, some architectures support a decimal format, usually called *packed decimal* or *binary-coded decimal*—4 bits are used to encode the values 0 to 9, and 2 decimal digits are packed into each byte. Numeric character strings are sometimes called *unpacked decimal*, and operations—called *packing* and *unpacking*—are usually provided for converting back and forth between them.

One reason to use decimal operands is to get results that exactly match decimal numbers, as some decimal fractions do not have an exact representation in binary. For example,  $0.10_{10}$  is a simple fraction in decimal, but in binary it requires an infinite set of repeating digits:  $0.0001100110011\dots_2$ . Thus, calculations that are exact in decimal can be close but inexact in binary, which can be a problem for financial transactions. (See Appendix J to learn more about precise arithmetic.)

## 3.2.2 Tipo y tamaño de los operandos

Introducción

Características

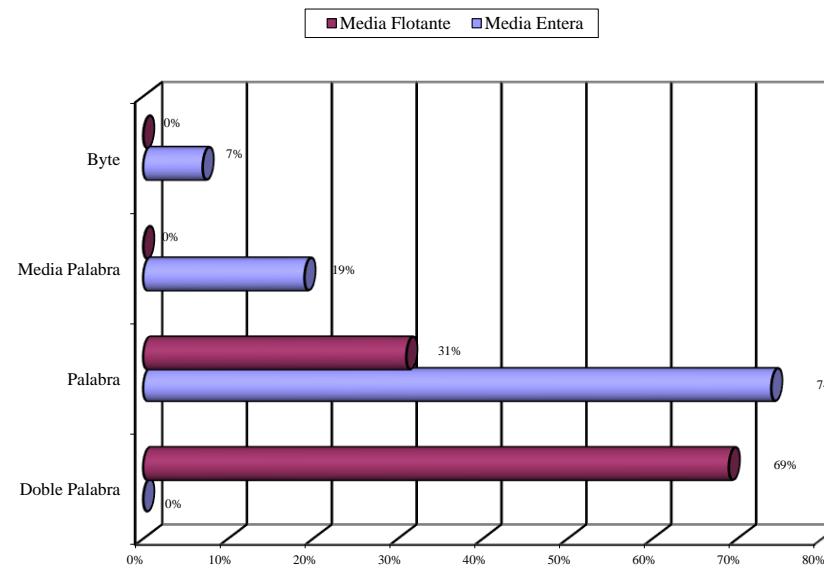
Programación

Ejemplos

Diseño del repertorio de instrucciones

### Distribución de los accesos a los datos por tamaños:

- Los accesos a los tipos principales de datos (palabra y doble palabra) dominan claramente
- Predominio de operandos enteros de 32 bits y operandos en coma flotante de 64 bits (IEEE 754).



## 3.2.3 Operaciones

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### a. Tipos de operaciones

Tipo de operación	Ejemplo
Aritmético y lógico	Operaciones lógicas y aritméticas enteras: suma, and, resta, or...
Transferencias de datos	Cargas y almacenamientos
Control	Salto, bifurcación, llamada y retorno de procedimiento, traps
Sistema	Llamada al sistema operativo, instrucciones de gestión de memoria virtual
Punto flotante	Operaciones de punto flotante: suma, multiplicación
Decimal	Suma, multiplicación decimal, conversiones de decimal a caracteres
Cadenas	Transferencia, comparación de cadenas, búsqueda de cadenas
Gráficos	Operaciones sobre pixels, operaciones de compresión descompresión

- ➊ Tres primeras categorías. Todas las máquinas proporcionan un repertorio completo de este tipo de operaciones
- ➋ Funciones del sistema: El soporte varía entre arquitecturas
- ➌ Punto flotante: frecuente incluso en repertorios reducidos

## 3.2.3 Operaciones

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### a. Tipos de operaciones

Tipo de operación	Ejemplo
Aritmético y lógico	Operaciones lógicas y aritméticas enteras: suma, and, resta, or...
Transferencias de datos	Cargas y almacenamientos
Control	Salto, bifurcación, llamada y retorno de procedimiento, traps
Sistema	Llamada al sistema operativo, instrucciones de gestión de memoria virtual
Punto flotante	Operaciones de punto flotante: suma, multiplicación
Decimal	Suma, multiplicación decimal, conversiones de decimal a caracteres
Cadenas	Transferencia, comparación de cadenas, búsqueda de cadenas
Gráficos	Operaciones sobre pixels, operaciones de compresión descompresión

- ➄ Tres últimas categorías pueden no estar presentes en algunas arquitecturas. Las arquitecturas de repertorio extenso CISC pueden contener un amplio repertorio en estas categorías

### 3.2.3 Operaciones

#### Regla de comportamiento común a todas las arquitecturas

- ➊ Las instrucciones utilizadas más extensamente de un conjunto de instrucciones son las operaciones simples

	Instrucciones 80x86	Promedio
1	Load	22%
2	Salto condicional	20%
3	Comparación	16%
4	Store	12%
5	Add	8%
6	And	6%
7	Sub	5%
8	Move reg-reg	4%
9	Call	1%
10	Return	1%
	Total	96%

- ➋ **Ejemplo:** 10 instrucciones simples del 80x86 que contabilizan el 96% de las instrucciones ejecutadas. El diseñador debe esforzarse en hacer rápidas estas instrucciones.

## 3.2.3 Operaciones

Introducción

Características

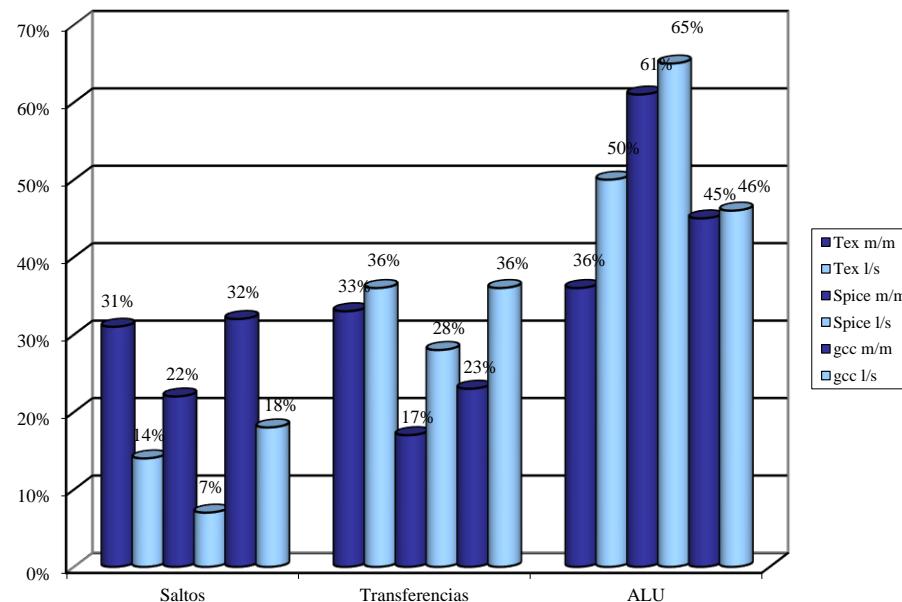
Programación

Ejemplos

Diseño del repertorio de instrucciones

### b. Rep. Inst. M-M vs R-R (carga/almacenamiento, I/s)

- ◆ Frecuencias para una arquitectura I/s (MIPS) y una M-M (VAX)
  - ◆ Referencias a memoria
  - ◆ Operaciones de la ALU
  - ◆ Instrucciones de flujo de control (saltos y bifurcaciones)



- ◆ Máquina R-R mayor porcentaje de movimientos de datos
- ◆ Frecuencia relativa más baja para saltos en R-R

## 3.2.3 Operaciones

Introducción

Características

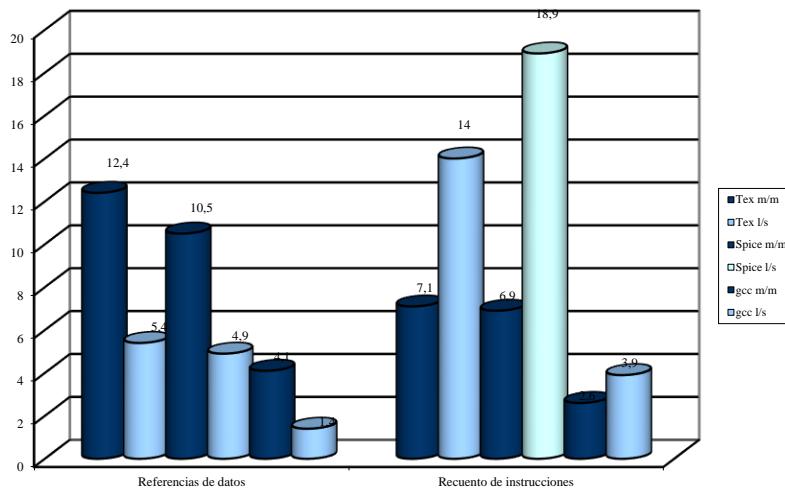
Programación

Ejemplos

Diseño del repertorio de instrucciones

### b. Rep. Inst. M-M vs R-R (carga/almacenamiento, I/s)

- Recuento absoluto de instrucciones ejecutadas
- Referencias a datos en memoria (cargas, almacenamientos, ALU mem)
- Máquina I/s requiere más instrucciones
- Podríamos deducir: de los RI y las frecuencias de operaciones de transferencia que el número de referencias a datos en I/s es mayor
- Datos indican lo contrario (más referencias a datos en M-M que I/s)
- En M-M referencias a datos no sólo con operaciones de transferencia sino con las ALU



## 3.2.3 Operaciones

Introducción

Características

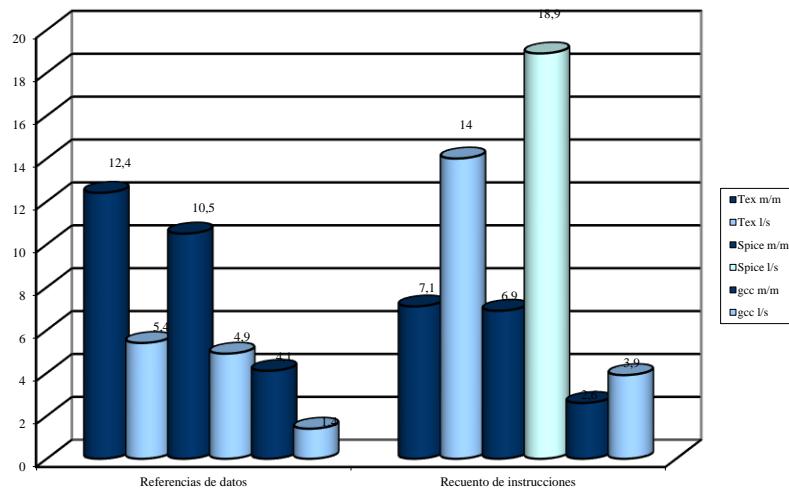
Programación

Ejemplos

Diseño del repertorio de instrucciones

### b. Rep. Inst. M-M vs R-R (carga/almacenamiento, I/s)

- Recuento absoluto de instrucciones ejecutadas
- Referencias a datos en memoria (cargas, almacenamientos, ALU mem)
- Máquina I/s requiere más instrucciones
- Podríamos deducir: de los RI y las frecuencias de operaciones de transferencia que el número de referencias a datos en I/s es mayor



- Diferencia las referencias a datos consecuencia de mejores posibilidades de ubicación de registros de I/s
- Diferencias entre las referencias a datos de mem-mem y I/s equilibra la diferencia entre referencias a instrucciones

## 3.2.3 Operaciones

Introducción

Características

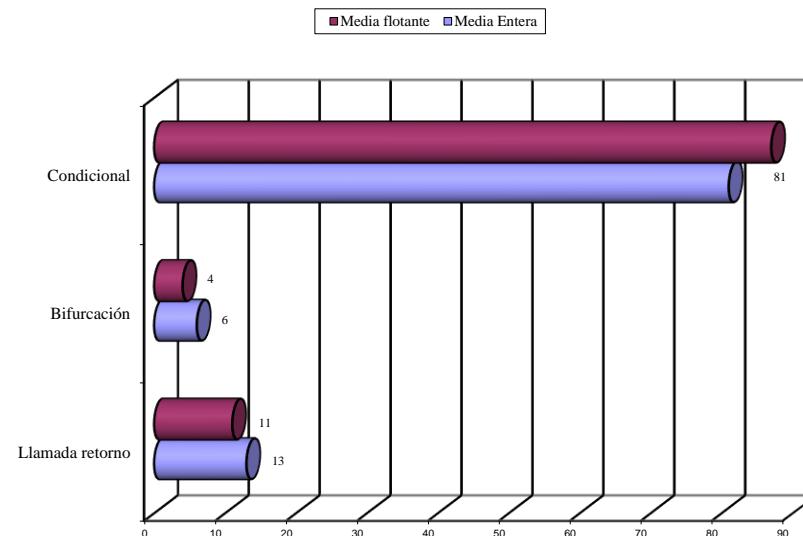
Programación

Ejemplos

Diseño del repertorio de instrucciones

### c. Instrucciones de control

- ◆ Cuatro tipos de cambios del flujo de control
  - ◆ Saltos condicionales
  - ◆ Bifurcaciones incondicionales
  - ◆ Llamadas a procedimientos
  - ◆ Retornos de procedimiento
- ◆ Frecuencia de instrucciones de flujo de control para máquina I/s



- ◆ Los saltos condicionales son los que más se utilizan

## 3.2.3 Operaciones

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### c. Instrucciones de control

- ◆ **Formas de especificar el destino del salto**

- ◆ **Explícitamente** (lo más frecuente) excepción (retorno de procedimiento)
  - ◆ JE ET, JMP ET, CALL ET, RET

- ◆ **Saltos relativos al PC**

- ◆ Dirección especificada mediante desplazamiento sumado al PC
  - ◆ Normalmente la posición destino del salto es cercana a la actual (pocos bits)

- ◆ **Saltos no relativos al PC**

- ◆ Para saltos a direcciones concretas de destino no conocido en tiempo de compilación. Necesario especificarlo dinámicamente.
  - ◆ Se puede nombrar un registro que contenga la dirección del destino
  - ◆ Alternativamente, se puede utilizar cualquier modo de direccionamiento

## 3.2.3 Operaciones

Introducción

Características

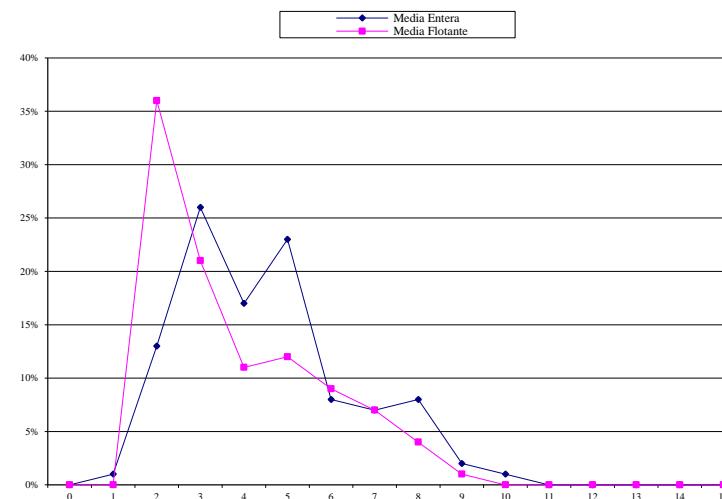
Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### c. Instrucciones de control

- ◆ Diseñador debe conocer magnitud de los desplazamientos para ver como afecta a la longitud y codificación de la instrucción
- ◆ Observamos distancias de los saltos relativos al PC. Número de instrucciones entre el destino y la instrucción de salto



- ◆ Saltos más frecuentes en programas enteros tienen entre 3 y 5 bits (25%)
- ◆ En programas en punto flotante 2 bits son los más frecuentes
- ◆ Los campos de desplazamientos cortos son suficientes 8 bits cubren el 93%

## 3.2.3 Operaciones

Introducción

Características

Programación

Ejemplos

### c. Instrucciones de control

#### ◆ Formas de especificar la condición del salto

#### ◆ Código de condición

- ◆ Los saltos examinan bits especiales inicializados por las operaciones de la ALU
  - ◆ **Ejemplo:**
    - ◆ SUB R1, R2, R3;                     $R1=R2-R3$
    - ◆ CMP R1, #0;                        Si  $R1=0$  el indicador  $z=1$
    - ◆ BEQ eti;                             Salta si  $z=1$
  - ◆ **Ventaja:** Las comparaciones pueden eliminarse en algún caso.
  - ◆ **Inconveniente:** Problemas en máquinas segmentadas derivados de la posible utilización simultanea de  $z$  desde varias instrucciones.

## 3.2.3 Operaciones

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### c. Instrucciones de control

#### ◆ Formas de especificar la condición del salto

#### ◆ Registro de condición

- ◆ Los saltos examinan registros arbitrarios con el resultado de una comparación

- ◆ **Ejemplo:**

- ◆ SUB R1, R2, R3;               $R1=R2-R3$

- ◆ SEQ R10, R1, #0;              Si  $R1=0$  se actualiza R10 con un 1

- ◆ BNEZ R10, eti;                Salta si  $R10 \neq 0$

- ◆ **Ventaja:** Independencia entre la operación y el registro implicado

- ◆ **Inconveniente:** Se consume un registro

## 3.2.3 Operaciones

Introducción

Características

Programación

Ejemplos

### c. Instrucciones de control

#### ◆ Formas de especificar la condición del salto

#### ◆ Comparación y salto

- ◆ La comparación es parte del salto, permitiendo saltar con una sola instrucción, si bien puede ser demasiado trabajo por instrucción

#### ◆ Ejemplo:

- ◆ SUB R1, R2, R3;                     $R1=R2-R3$
- ◆ C&B R1, #0, eti;                    Si  $R1=0$  salta a etiqueta.

- ◆ **Ventaja:** Reducción del recuento de instrucciones

- ◆ **Inconveniente:** Puede ser demasiado trabajo para una instrucción, aumentando el CPI o el clk

## 3.2.3 Operaciones

Introducción

Características

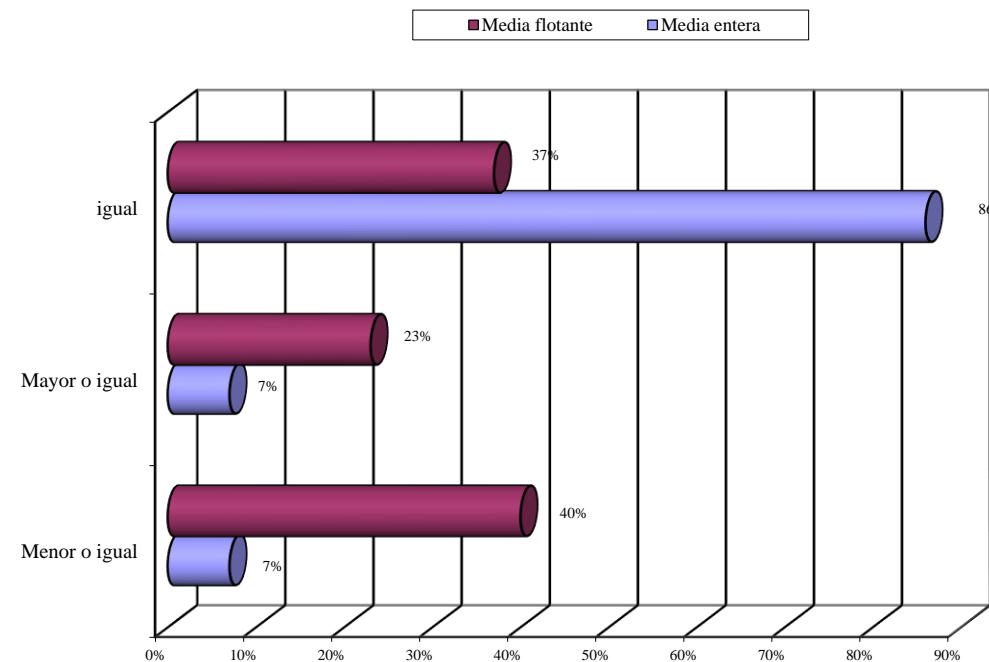
Programación

Ejemplos

Diseño del repertorio de instrucciones

### c. Instrucciones de control

- La mayor parte de las comparaciones son test de igualdad desigualdad y un gran número son comparaciones con 0 (aproximadamente un 50% son test de igualdad con 0)



## 3.3 Evolución computadores

**Tema 3. Diseño del repertorio de instrucciones**

**Arquitectura de computadores**

### 3.3.1 Introducción

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

- ➊ Inicialmente, las decisiones de diseño de la arquitectura se realizaban para facilitar la programación en lenguaje ensamblador (CISC)
- ➋ La aparición de los RISC (y por la madurez de los compiladores) lleva a que los compiladores deban realizar las operaciones eficientemente
- ➌ Actualmente, la mayor parte de la programación se realiza en lenguajes de alto nivel para computadores de escritorio, servidores y clusters
  - ➍ La mayoría de instrucciones ejecutadas son salida de un compilador
  - ➎ La arquitectura a nivel lenguaje máquina es un **objeto del compilador**
  - ➏ Decisiones de diseño afectan a la **calidad del código** que puede ser generado por un compilador y la **complejidad de construir** un buen **compilador**

## 3.3.2 Arquitectura como objeto del compilador

Introducción

Características

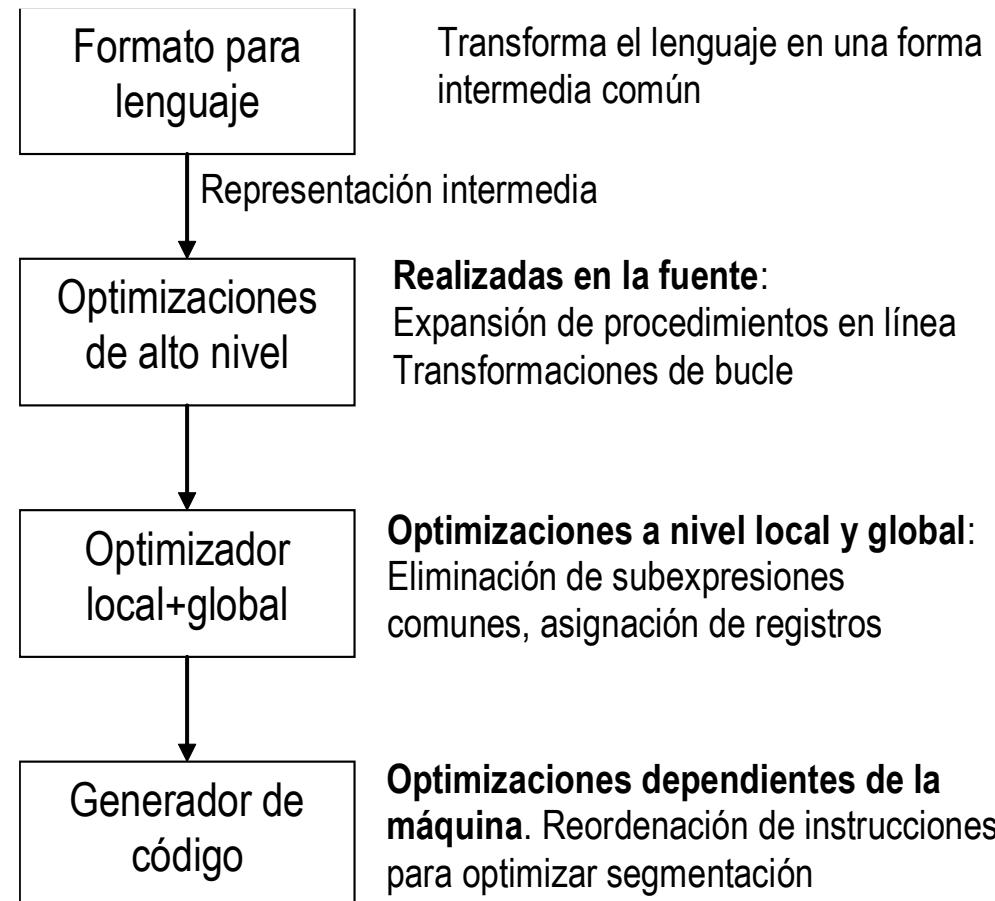
Programación

Ejemplos

Diseño del repertorio de instrucciones

### Estructura de compiladores

- ◆ Pasos de los compiladores para transformar representaciones de alto nivel en representaciones de bajo nivel



## 3.3.2 Arquitectura como objeto del compilador

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### Asignación de registros

- **¿Cuantos registros se necesitan para ubicar las variables?**
  - Óptima ubicación de variables en registros depende del número de registros de propósito general y de estrategia de ubicación
  - Ubicación de registros influye en aceleración del código (acceso a registros frente a acceso a memoria) como en mejorar optimizaciones del compilador (eliminación subexpresiones comunes)
- **Coloreado de grafos:** Algoritmo de ubicación de variables en registros
  - **Problema NP-Completo** pero hay **técnicas heurísticas** que funcionan bien, a partir de 16 reg.
  - El funcionamiento mejora con **al menos 16 registros** (preferiblemente más) de propósito general, para ubicación de variables enteras y análogamente para variables de punto flotante.  
**Ejemplo MIPS** 32 enteros y 32 para trabajo en punto flotante

## 3.3.2 Arquitectura como objeto del compilador

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

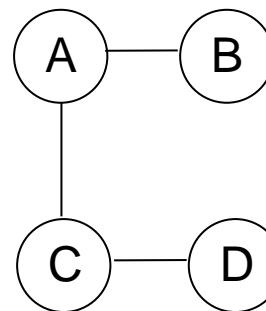
### Coloreado de grafos

- Programa: Grafo cuyos nodos son las variables y cuyos arcos muestran el solapamiento en su utilización
- Colorear grafo utilizando número de colores igual al de registros disponibles
- Dos nodos adyacentes no pueden usar el mismo color

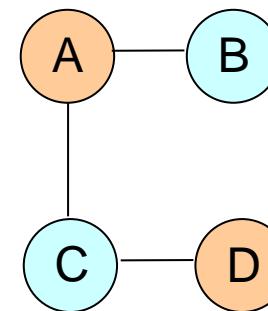
◆ Programa

A=  
B=  
...  
...B...  
C=  
...A...  
D=...  
...D...  
...C...

◆ Grafo



◆ Grafo Coloreado



◆ Programa registr

R1=  
R2=  
...  
...R2...  
R2=  
...R1...  
R1=...  
...R1...  
...R2...

## 3.3.2 Arquitectura como objeto del compilador

Introducción

Características

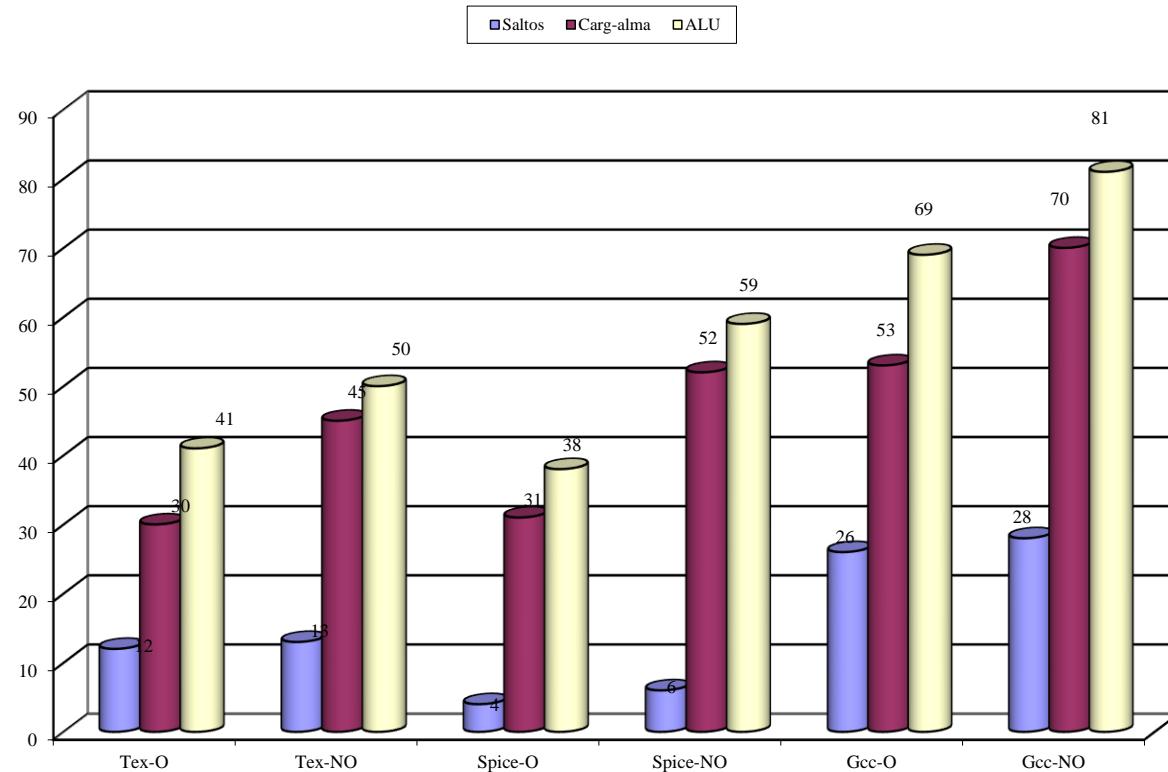
Programación

Ejemplos

Diseño del repertorio de instrucciones

### Influencia de la optimización en la mezcla de instrucciones

- ◆ Efecto inmediato de optimización es reducción del RI
- ◆ Las estructuras de control son las más difíciles de reducir



## 3.3.2 Arquitectura como objeto del compilador

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### Propiedades que ayudan al diseñador de compiladores

#### ➊ Ortogonalidad

- Tres componentes principales de un repertorio de instrucciones, operaciones, tipos de datos y modos de direccionamiento deben ser independientes

#### ➋ Proporcionar primitivas y no soluciones

- Intentos de soportar lenguajes de alto nivel no han tenido éxito

#### ➌ Proporcionar información de las secuencias alternativas de código de rendimiento óptimo

- Tarea del escritor de compiladores: imaginar secuencias de instrucciones óptimas para cada segmento de código
- El número de instrucciones o el tamaño del código no son representativas

### 3.3.3 VLIW

Introducción

Características

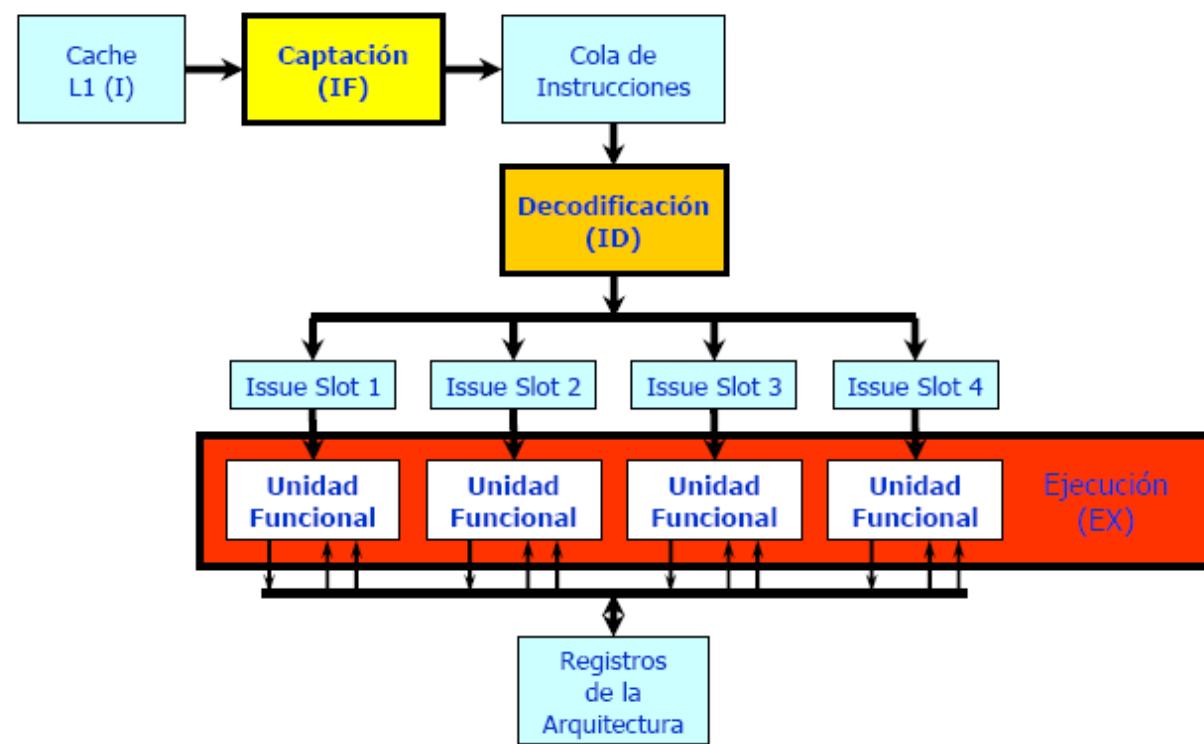
Programación

Ejemplos

Diseño del repertorio de instrucciones

## Evolución de la tecnología de computadores ha permitido:

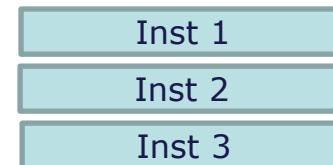
- Disponer de varias unidades de ejecución dentro del mismo procesador
- Los computadores actuales pueden ejecutar varias operaciones simultáneamente en esas unidades de ejecución



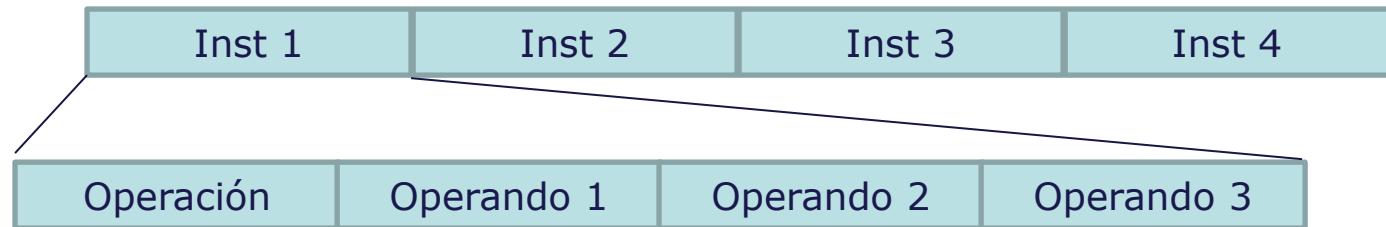
### 3.3.3 VLIW

## Superescalares vs VLIW

- En los procesadores **superescalares**, la organización es la encargada de descubrir el paralelismo que permita aprovechar las instrucciones que se van captando de memoria



- En los procesadores **Very Large Instruction Word (VLIW)**, el paralelismo es implícito en las instrucciones (Intel Itanium-2)
  - Cada instrucción incluye las operaciones que se realizan **simultáneamente**.



Diseño del repertorio de instrucciones

### 3.3.3 VLIW

## Procesamiento VLIW

- La arquitectura VLIW utiliza **varias unidades funcionales independientes**
- En lugar de enviar varias instrucciones independientes a las unidades funcionales, empaqueta varias instrucciones en una única instrucción (112-128 bits)
- La **decisión** de qué instrucciones se deben ejecutar simultáneamente corresponde al **compilador**
- Las ventajas aumentan a medida que se pretenden emitir más instrucciones por ciclo

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### 3.3.3 VLIW

## El papel del compilador

### ● Planificación estática (VLIW)

- Más esfuerzo del compilador: renombrado de registros, reorganizaciones de código,... para mejorar el uso de los recursos disponibles
- El compilador construye **paquetes de instrucciones sin dependencias**, de forma que el procesador no necesita comprobarlas explícitamente

### ● Planificación dinámica (Superescalar)

- Menos asistencia del compilador pero más coste hardware (organización) aunque facilita la portabilidad de código entre la misma familia de procesadores

Diseño del repertorio de instrucciones

## 3.4 Ejemplos característicos

**Tema 3. Diseño del repertorio de instrucciones**

**Arquitectura de computadores**

# Ejemplos característicos

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

- **VAX** de DEC (ha durado 10 años, década de los 80, y cientos de miles de unidades).
- **IBM 360** (ha durado 25 años décadas 70 y 80, y cientos de miles de unidades).
- **Intel 8086** Es el computador de propósito general más popular del mundo. Decada de los 80 y 90.
- **DLX** Maquina genérica de carga almacenamiento muy popular desde finales de los 80

## 3.4.1 DEC VAX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### ◆ Objetivos

- Facilitar la tarea de escritura de compiladores y sistemas operativos proporcionando una **arquitectura altamente ortogonales**
- Las **demás arquitecturas** que estudiaremos son **subconjuntos del VAX** en términos de instrucciones y modos de direccionamiento
- El **VAX** es una **máquina de registros de propósito general**

## 3.4.1 DEC VAX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### ◆ Memoria

- move transfiere datos entre dos posiciones direccionables cualesquiera:
- Cargas: **reg-mem**
- Almacenamientos: **mem-reg**
- Transferencias r-r: **reg-reg**
- Transferencias m-m: **mem-mem**

## 3.4.1 DEC VAX

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### Tipos de datos

- ◆ Inicial del tipo de dato utilizada para completar un nombre de código de operación. Ejemplo mov transfiere un operando del tipo de dato indicado
- ◆ MOVB, MOVW, MOVL, MOVQ, MOVO, MOVF, MOVG, MOVD, ...

Bits	Tipo de dato	Nuestro nombre	Nombre de DEC
8	Entero	Byte	Byte (B)
16	Entero	Media palabra	Palabra (W)
32	Entero	Palabra	Palabra larga (L)
64	Entero	Doble palabra	Cuad palabra (Q)
128	Entero	Cuad palabra	Octa-palabra (O)
32	Punto flotante	Simple precisión	F_flotante (F)
64	Punto flotante	Doble precisión	D_flotante, G_flotante (D,G)
128	Punto flotante	Huge (Enorme)	H-flotante (H)
4n	Decimal	Empaquetado	Empaquetado (P)
8n	Cadena numérica	Desempaquetado	Cadenas numéricas (S)
8n	Cadenas de caracteres	Carácter	Carácter (C)

## 3.4.1 DEC VAX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### ◆ Modos de direccionamiento

- Una **instrucción VAX** de tres operandos puede incluir **desde 0 a tres referencias a memoria**, cada una de las cuales puede utilizar cualquier modo de direccionamiento

Modo de direccionamiento	Sintaxis
Literal	#valor
Inmediato	#valor
Registro	$R_n$
Registro diferido	$(R_n)$
Desplazamiento de byte/palabra/largo	Desplazamiento $(R_n)$
Desplazamiento diferido de byte/palabra/largo	@Desplazamiento $(R_n)$
Escalado (indexado)	Modo base $[R_x]$
Autoincremento	$(R_n) +$
Autodecremento	$-(R_n)$
Autoincremento diferido	@ $(R_n) +$

## 3.4.1 DEC VAX

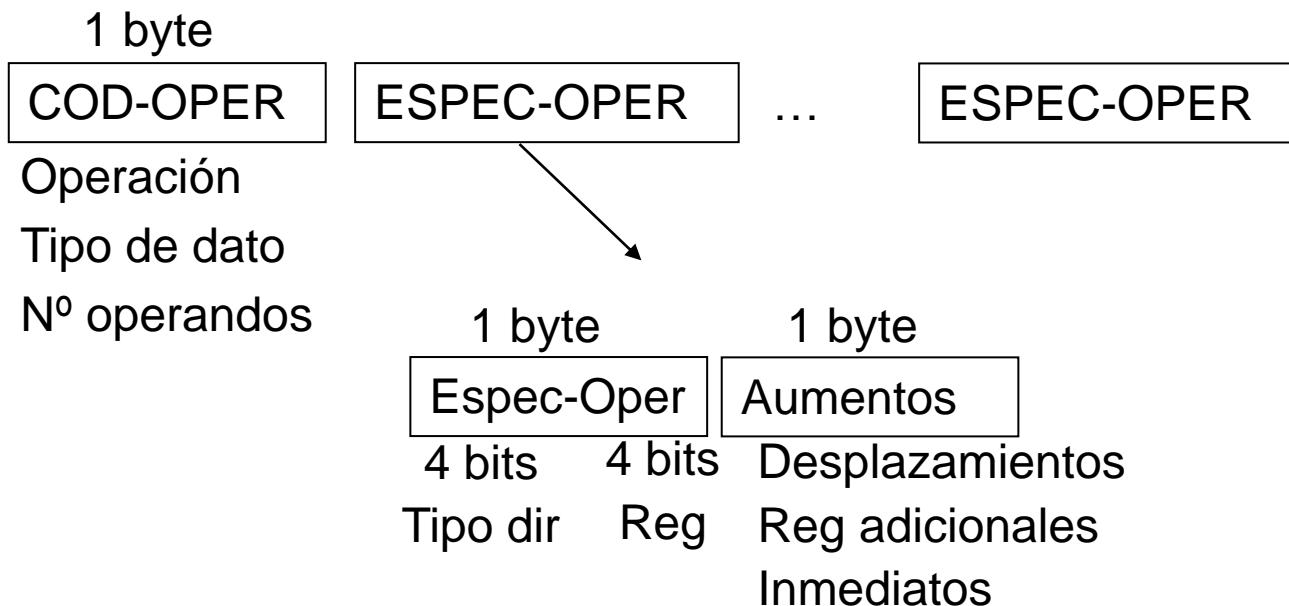
Introducción

Características

Programación

Ejemplos

### • Codificación (Variable)



## 3.4.1 DEC VAX

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### ◆ Codificación (Variable)

Modo de direccionamiento	Sintaxis	Longitud en bytes
Literal	#valor	1 byte
Inmediato	#valor	1 + longitud del inmediato
Registro	R <sub>n</sub>	1
Registro diferido	(R <sub>n</sub> )	1
Desplazamiento de byte/palabra/largo	Desplazamiento (R <sub>n</sub> )	1 + longitud del desplazamiento
Desplazamiento de byte/palabra/largo	@Desplazamiento (R <sub>n</sub> )	1 + longitud del desplazamiento
Escalado (indexado)	Modo base [R <sub>x</sub> ]	1 + longitud del modo de direccionamiento base
Autoincremento	(R <sub>n</sub> )+	1
Autodecremento	-(R <sub>n</sub> )	1
Autoincremento diferido	@(R <sub>n</sub> )+	1

## 3.4.1 DEC VAX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### ◆ Codificación (Variable)

#### ◆ Ejemplo

◆ **ADDL3 R1, 737(R2), #456**

↑      ↑      ↑      ↑  
1 + 1 + (1+2) + (1+4)

### ◆ Operaciones del VAX (CISC)

- ◆ Transferencias de datos
- ◆ Aritmética lógica
- ◆ Control
- ◆ Procedimiento
- ◆ Carácter decimal de campo de bits
- ◆ Punto flotante
- ◆ Sistema
- ◆ Otras

## 3.4.2 IBM 360/370

Introducción

Características

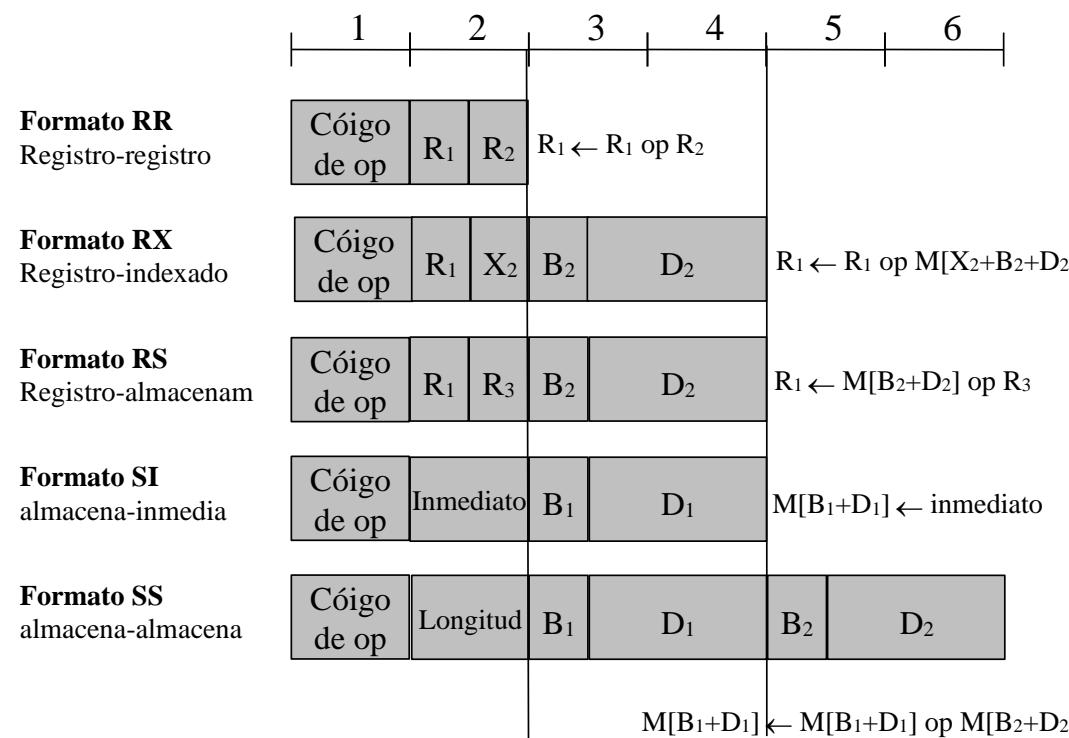
Programación

Ejemplos

Diseño del repertorio de instrucciones

### Objetivos

- ◆ Máquina de propósito general con muchos tipos de datos y facilidades para los sistemas operativos
- ◆ Compatibilidad del lenguaje máquina



## 3.4.2 IBM 360/370

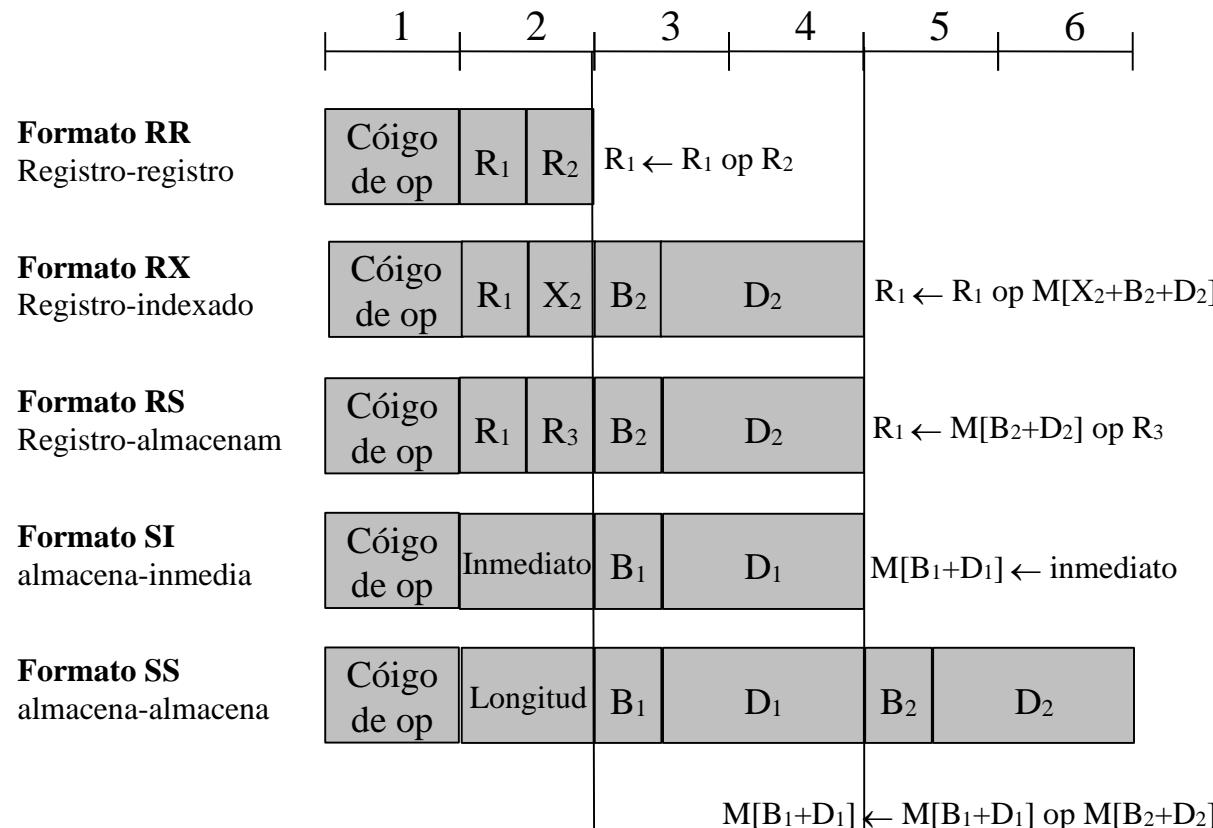
Introducción

Características

Programación

Ejemplos

### ◆ Modos de direccionamiento y formatos de instrucción



- ◆ **RR (Registro-registro).** Ambos operandos son el contenido de los registros. El primer operando fuente es también destino

## 3.4.2 IBM 360/370

Introducción

Características

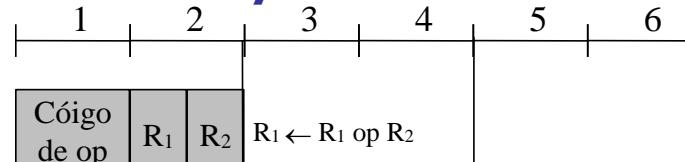
Programación

Ejemplos

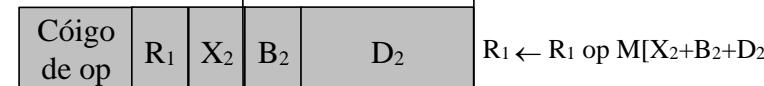
Diseño del  
repertorio de  
instrucciones

### ◆ Modos de direccionamiento y formatos de instrucción

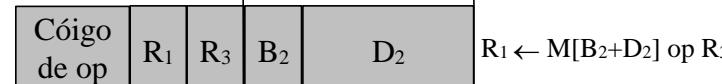
**Formato RR**  
Registro-registro



**Formato RX**  
Registro-indexado



**Formato RS**  
Registro-almacenam



**Formato SI**  
almacena-inmedia



**Formato SS**  
almacena-almacena



M[B<sub>1</sub>+D<sub>1</sub>] ← M[B<sub>1</sub>+D<sub>1</sub>] op M[B<sub>2</sub>+D<sub>2</sub>]

### ◆ RX (Registro-indexado)

- Primer operando (fuente y destino) es un registro
- Segundo operando posición de memoria

D2 desplazamiento de 12 bits

B2 contenido del registro B2

X2 contenido del registro X2

## 3.4.2 IBM 360/370

Introducción

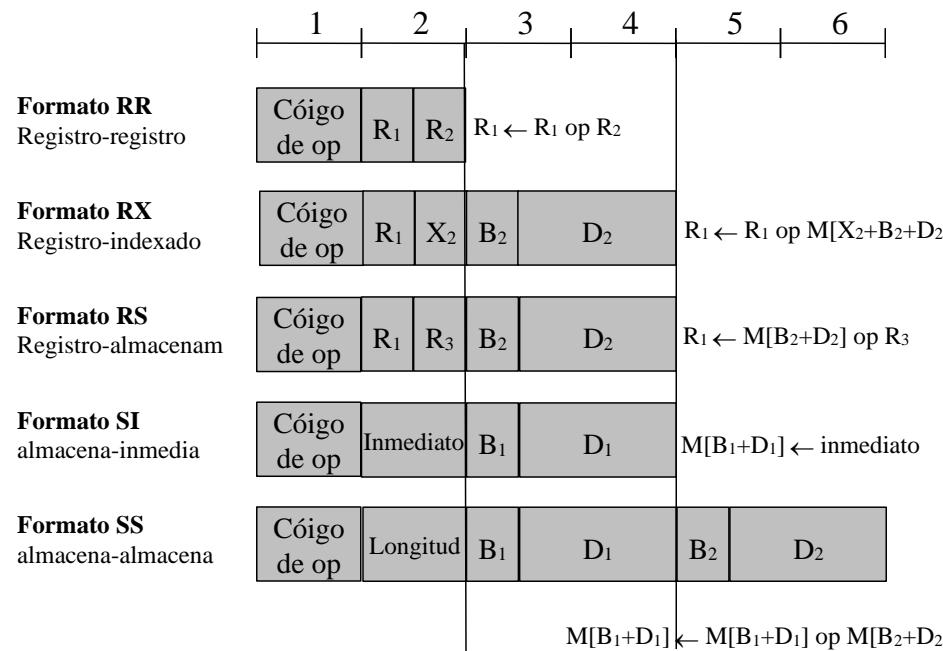
Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### ◆ Modos de direccionamiento y formatos de instrucción



### ◆ RS (Registro-memoria)

- ◆ Primer operando es el registro destino
  - ◆ Tercer operando registro como segunda fuente
  - ◆ Segundo operando posición de memoria
- D2: campo desplazamiento de 12 bits  
B2: contenido del registro B2

## 3.4.2 IBM 360/370

Introducción

Características

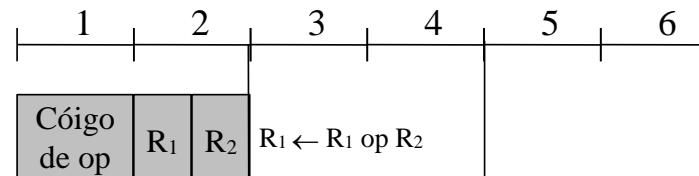
Programación

Ejemplos

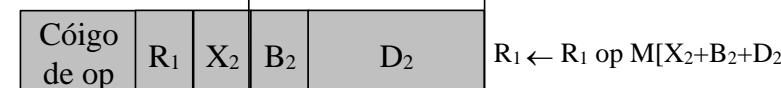
Diseño del  
repertorio de  
instrucciones

### ◆ Modos de direccionamiento y formatos de instrucción

**Formato RR**  
Registro-registro



**Formato RX**  
Registro-indexado



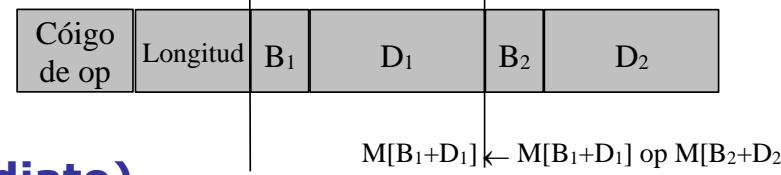
**Formato RS**  
Registro-almacenam



**Formato SI**  
almacena-inmedia



**Formato SS**  
almacena-almacena



### ◆ SI (memoria-inmediato)

- El destino es un operando de memoria dado por la suma de
- B1: contenido del registro B1
- D1: valor del desplazamiento D1.
- Segundo operando, un campo inmediato de 8 bits es la fuente

## 3.4.2 IBM 360/370

Introducción

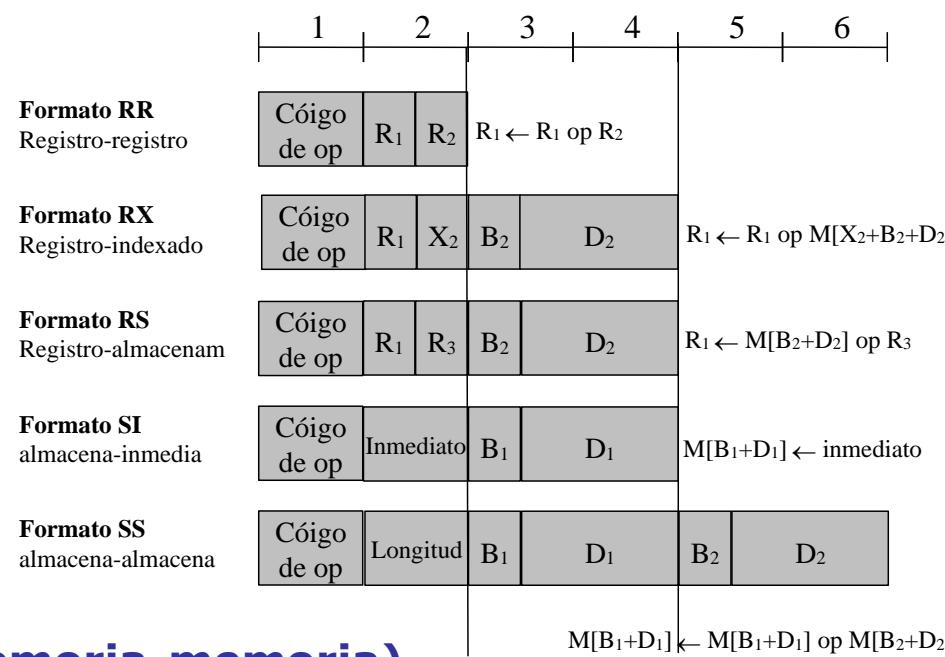
Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### ◆ Modos de direccionamiento y formatos de instrucción



### ◆ SS (memoria-memoria)

- ◆ Las direcciones de los dos operandos de memoria son la suma del contenido de un registro base Bi y un desplazamiento Di
- ◆ El primer operando es fuente y destino

## 3.4.2 IBM 360/370

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### ◆ Operaciones del 360/370 (CISC)

- ◆ **Control**
- ◆ **Aritmético, lógica**
- ◆ **Transferencia de datos**
- ◆ **Punto flotante**
- ◆ **Cadena decimal**

### 3.4.3 Intel 8086

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

- La arquitectura 8086 extensión del 8088 (máquina acumulador)
- El 8086 amplió el banco de registros
- Arquitectura de 16 bits
- Memoria segmentada
- Los diseñadores lograron un espacio de direcciones de 20 bits mediante la segmentación de la memoria

### 3.4.3 Intel 8086

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

#### ◆ La familia x86

- **Los 80186, 80286, 80386, 80486, Pentium (Pro, II, III, M, 4), Core 2, Core i3/i5/i7** son extensiones compatibles del 8086
- **El 80186** extendió el repertorio original. Sistema de 16 bits.
- **El 80286** amplió el espacio de direcciones a 24 bits. Multitarea y memoria virtual
- **El 80386** (1985) verdadera máquina de 32 bits (registros de 32 bits) (espacio de direcciones de 32 bits). Nuevo conjunto de modos de direccionamiento y de operaciones

### 3.4.3 Intel 8086

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

#### ◆ La familia x86

- **El 80486 (1989)** más instrucciones. Incremento del rendimiento. Segmentación del cauce (5 etapas) y cache más sofisticada
- **Pentium.** Introducción de técnicas superescalares, varias instrucciones en paralelo. (varias unidades de ejecución)
- **Pentium Pro (1995):** Profundiza sobre técnicas superescalares
- **Pentium II:** tecnología MMX (procesamiento eficiente de video audio y gráficos). Se utilizan registros de la pila del coprocesador

### 3.4.3 Intel 8086

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

#### ◆ La familia x86

- ◆ **Arquitectura del Pentium II (similar a la del Pentium Pro)** consta de una envoltura CISC con un núcleo RISC
  1. El procesador capta instrucciones de memoria
  2. Cada inst se traduce en varias inst RISC tamaño fijo (microops)
  3. El procesador ejecuta las microops con organización superescalar
  4. Datos escriben en BR en orden establecido por programa
- ◆ **Pentium III:** Instrucciones adicionales en punto flotante para procesamiento eficiente de gráficos 3D. SSE (Streaming SIMD Extension) 8 nuevos registros de 128 bits
- ◆ **Pentium 4.** Supersegmentada de 20 etapas. Duplica ALUs (2 unidades enteras). Nuevas instrucciones SSE

### 3.4.3 Intel 8086

#### ◆ La familia x86

- ◆ **Core 2 (2006):** Arquitectura de 64 bits. Duo: 2 cores
- ◆ **i3,i5,i7 (2008):** Quad-core

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

## 3.4.3 Intel 8086

Introducción

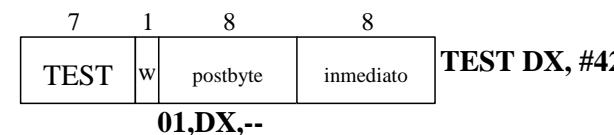
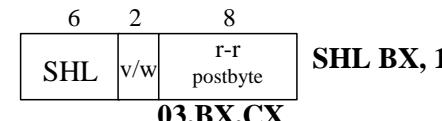
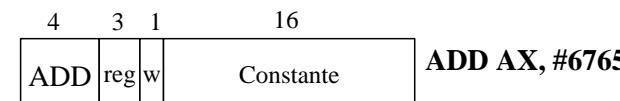
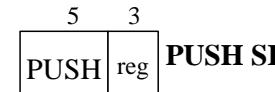
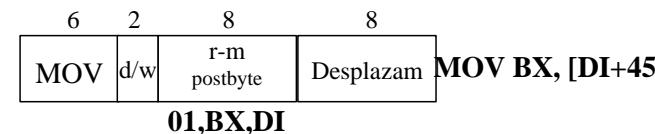
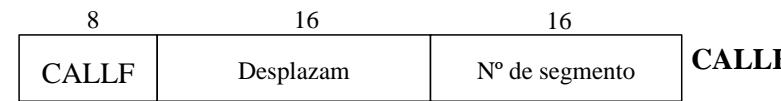
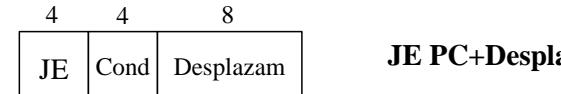
Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

### ◆ Formatos de instrucción



### 3.4.3 Intel 8086

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

#### ◆ Formatos de instrucción

Código (8 bits)	Post-byte(8 bits)	Des	Val
mod(2b) reg(3b) rm(3b)			

- ◆ **Código:** 1er byte, es el único que existe siempre, el resto pueden aparecer o no.
- ◆ **Post-byte:** Refleja los operandos de la instrucción
  - 1er operando:** mediante mod y rm. Puede ser un registro o una posición de memoria. mod=tipo de direccionamiento. rm=registro de direccionamiento.
  - 2º operando** mediante reg: Debe ser un registro.
- ◆ **Des:** componente desplazamiento de una dirección de memoria. 1 o 2 bytes
- ◆ **Val:** valor inmediato. 1 o 2 bytes

### 3.4.3 Intel 8086

Introducción

Características

Programación

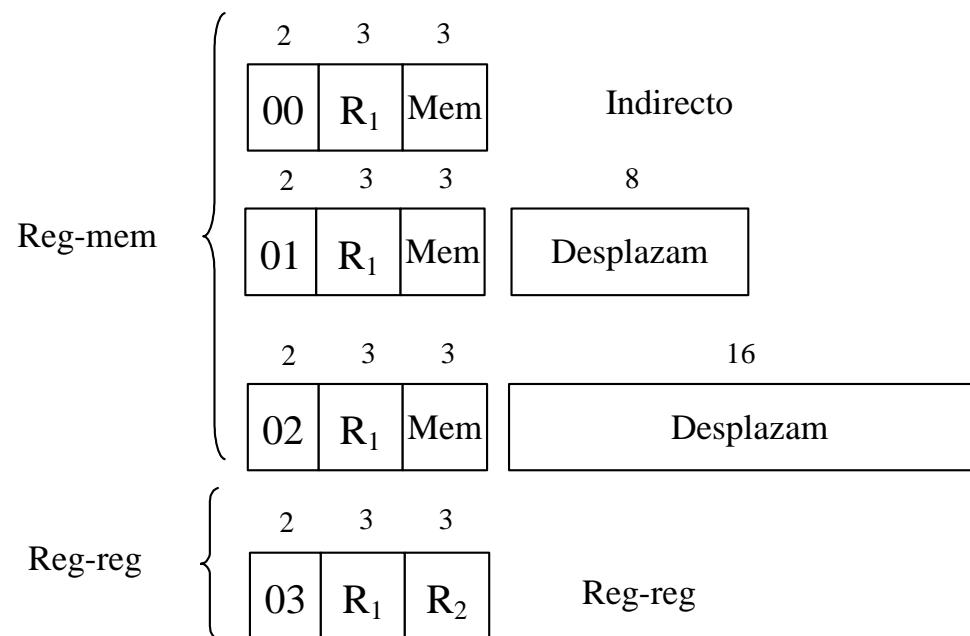
Ejemplos

Diseño del  
repertorio de  
instrucciones

#### ◆ Formatos de instrucción

Código (8 bits)	Post-byte(8 bits)	Des	Val
mod(2b) reg(3b) rm(3b)			

#### ◆ Hay cuatro codificaciones posibles para el postbyte:



## 3.4.4 DLX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### • **La arquitectura DLX**

- **DLX es una sencilla arquitectura de carga almacenamiento.**
- **Nombre promedio varias máquinas próximas a DLX en romanos**
- AMD 29K, DECstation 3100, HP 850, IBM 801, Intel i860, MPS M/102<sup>a</sup>, MIPS M/1000, Motorola 88K, RISC I, SGI 4D/60, SPARCstation-1, Sun-4/110, Sun-4/260.
- **La arquitectura DLX se escogió basándose en las observaciones sobre las primitivas más frecuentes utilizadas en los programas**
- **Las funciones más sofisticadas se implementaban a nivel software con múltiples instrucciones**

## 3.4.4 DLX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### • **La arquitectura DLX**

- **DLX hace énfasis en:**
- **Un sencillo repertorio de instrucciones de carga almacenamiento**
- **Diseño de segmentación eficiente (pipelining)**
- **Un repertorio de instrucciones fácilmente decodificables**
- **Eficiencia como objeto del compilador**

## 3.4.4 DLX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### • **Características**

### • **Los registros**

- La arquitectura tiene 32 registros de propósito general GPR de 32 bits; el valor de R0 siempre es 0.
- Registros de punto flotante (FPR), se pueden utilizar como 32 registros de simple precisión (32 bits), o como parejas de doble precisión F0 , F2, ...., F28 , F30 .
- Registros especiales para acceder a la información sobre el estado, que se pueden transferir a y desde registros enteros (ej. Registro de estado de punto flotante)

## 3.4.4 DLX

Introducción

Características

Programación

Ejemplos

Diseño del  
repertorio de  
instrucciones

### • **Características**

### • **La memoria**

- La memoria es direccionable por bytes en el modo <<Big Endian>> con una dirección de 32 bits.
- Todas las referencias a memoria se realizan a través de cargas o almacenamientos entre memoria y los GPR o FPR.
- Los accesos que involucran a los GPR pueden realizarse a un byte, a media palabra y a una palabra.
- Los accesos que involucran a los FPR pueden realizarse a palabras en simple o doble precisión.
- Los accesos a memoria deben estar alineados
- Todas las instrucciones son de 32 bits y deben estar alineadas

## 3.4.4 DLX

### Operaciones

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

Tipo de instrucción. Cód de oper	Significado de la instrucción
<b>Transferencia de datos</b>	<b>Transfieren datos entre registros y memoria, o entre registros enteros y FP o registros especiales.</b>
LB, LBU, SB	Carga byte, carga byte sin signo, <b>almacena</b> byte
LH, LHU, SH	Carga med pal, carga med pal sin signo, <b>almacena</b> med pal
LW, SW	Carga palabra, <b>almacena</b> palabra
LF, LD, SF,SD	Carga punto flotante SP, carga punto flotante DP, almacena punto flotante SP, almacena punto flotante DP
MOVI2S, MOVS2I	<b>Transfiere</b> desde/ a GPR a/ desde un registro especial
MOVF, MOVD	Copia un registro de punto flotante a un par en DP
MOVFP2I, MOVI2FP	Transfiere 32 bits desde/a registros FP a/ desde registros enteros
<b>Aritmético-lógicas</b>	<b>Operaciones sobre datos enteros o lógicos en GPR.</b>
ADD, ADDI, ADDU, ADDUI	<b>Suma, suma inmediato</b> (todos los inmediatos son de 16 bits)
SUB, SUBI, SUBU, SUBUI	<b>Resta, resta inmediato</b> con y sin signo
MULT, MULTU, DIV, DIVU	<b>Multiplica y divide</b> , con signo y sin signo, los operandos deben estar en registros de punto flotante
AND, ANDI	<b>And, and inmediato</b>
OR, ORI, XOR, XORI	<b>Or, or inmediato, or exclusiva, or exclusiva inmediata</b>
LHI	Carga inmediato superior, carga la mitad superior de registro con inmediato
SLL, SRL, SRA, SLLI, SRRI, SRAI	<b>Desplazamientos, lógicos</b> dere izqu, aritméticos derecha

## 3.4.4 DLX

### Operaciones

Introducción

Características

Programación

Ejemplos

Diseño del repertorio de instrucciones

Tipo de instrucción. Cód de oper	Significado de la instrucción
Control	<b>Saltos y bifurcaciones condicionales; relativos al PC o mediante registros.</b>
BEQZ, BNEZ	Salto GPR igual/no igual a cero, despla 16 bits
BFPT, BFPF	Test de bit de comparación reg estado FP y salto, despla 16
J, JR	Bifurcaciones: desplazamiento de 26 bits
JAL, JALR	Bifurcación y enlace
TRAP	Transfiere a S.O. a una dirección vectorizada
RFE	Volver a código de usuario desde una excepción
Punto flotante	<b>Operaciones en punto flotante en formatos DP y SP</b>
ADDD, ADDF	Suma números DP, SP
SUBD, SUBF	Resta números DP, SP
MULTD, MULTF	Multiplica punto flotante DP, SP
DIVD, DIVF	Divide punto flotante DP, SP
CVTF2D, CVTF2I, CVTD2F, CVTD2I, CVTI2F, CVTI2D	Convierte instrucciones
_____D, _____F	Compara DP, SP

## 3.4.4 DLX

Introducción

Características

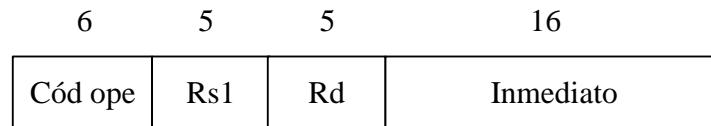
Programación

Ejemplos

Diseño del repertorio de instrucciones

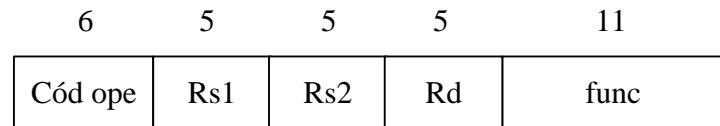
### ◆ Codificación de las instrucciones

#### Instrucción tipo I



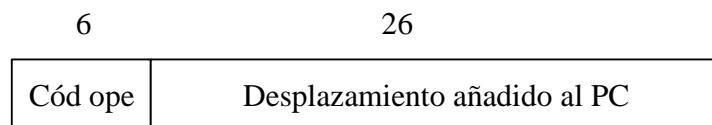
- Cargas y almacenamientos (byte, media palabra, palabra)
- ALUs con operandos inmediatos
- Instrucciones de salto condicional (BEQZ, BNEZ)
  - Rs1 registro implicado Rd no se utiliza
  - Saltos a registro
    - Rd=0; Inmediato=0; Rs1=destino

#### Instrucción tipo R



- Aritméticas y lógicas entre registros
  - Rs1= fuente1
  - Rs2= fuente2
  - Rd= Registro destino
  - Fun.= operación del flujo de datos

#### Instrucción tipo J



- Instrucciones de salto
  - Desplazamiento 26 bits con signo añadido al PC
    - JAL Salto incondicional y enlace (R31)
    - J Salto incondicional
    - Trap Interrupciones

# Tema 4. Segmentación

Arquitectura de los Computadores

# Tema 4. Segmentación

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## Objetivos:

- ◆ Mostrar al alumno conceptos relativos a segmentación.
- ◆ Proporcionar una clasificación de las arquitecturas segmentadas.
- ◆ Proponer varios niveles de aplicación de la segmentación.
- ◆ Profundizar en la segmentación del repertorio de instrucciones, utilizando la arquitectura MIPS como caso de estudio y manteniendo la continuidad con respecto a temas anteriores.
- ◆ Estudiar los cauces aritméticos
- ◆ Introducir las técnicas de optimización de unidades segmentadas
- ◆ Introducir conceptos de superescalares

# Tema 4. Segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **1. Introducción**
- ➋ **2. Segmentación del repertorio de instrucciones**
- ➌ **3. Cauces aritméticos**
- ➍ **4. Optimización de unidades segmentadas**
- ➎ **5. Superescalares**

## 4.1 Introducción

Tema 4. Segmentación

Arquitectura de los Computadores

# Segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ 4.1. INTRODUCCIÓN

- ◆ Concepto de segmentación
- ◆ Clasificación de las arquitecturas segmentadas Niveles de aplicación.
- ◆ Análisis de prestaciones

# Concepto de segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ La segmentación es una de las claves que permite **aumentar el rendimiento** en los computadores.
- ➋ Analogía con una cadena de montaje industrial
  - ➌ La ejecución de un tarea se divide en etapas, cada elemento de la cadena se especializa en realizar una operación concreta.
  - ➍ Explota el **parallelismo temporal**
    - ➎ Opera de forma serie para una pieza determinada
    - ➏ Ejecución de varias tareas simultáneas en diferentes etapas

# Concepto. Secuencial

Introducción

Segmentación del repertorio

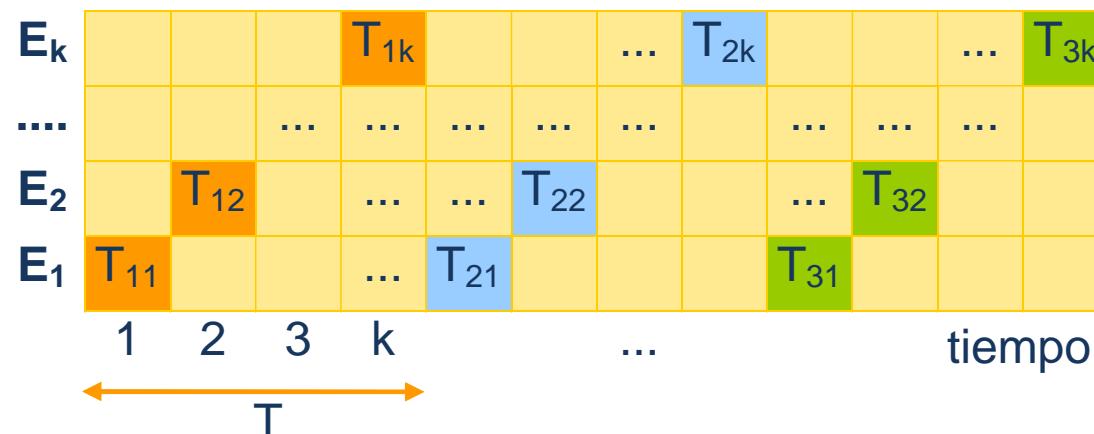
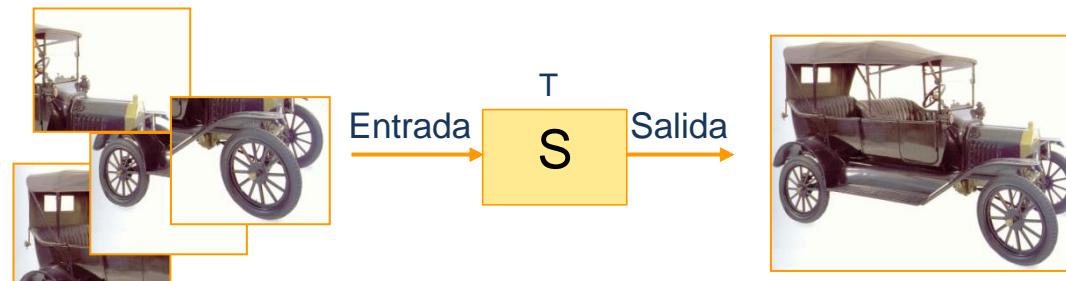
Cauces aritméticos

Optimización

Superescalares

Segmentación

- Una tarea puede realizarse cuando todos los **recursos** que necesita estén **disponibles**



- Equipo de trabajo, **trabajadores especializados** que descansan hasta que vuelvan a entrar los materiales
- **Sólo un trabajador** (un recurso)

# Concepto. Secuencial

Introducción

Segmentación del repertorio

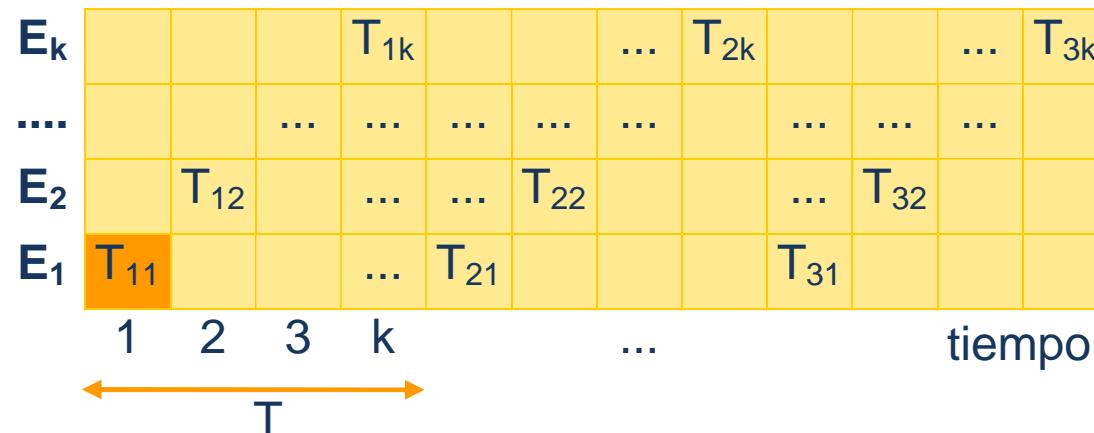
Cauces aritméticos

Optimización

Superescalares

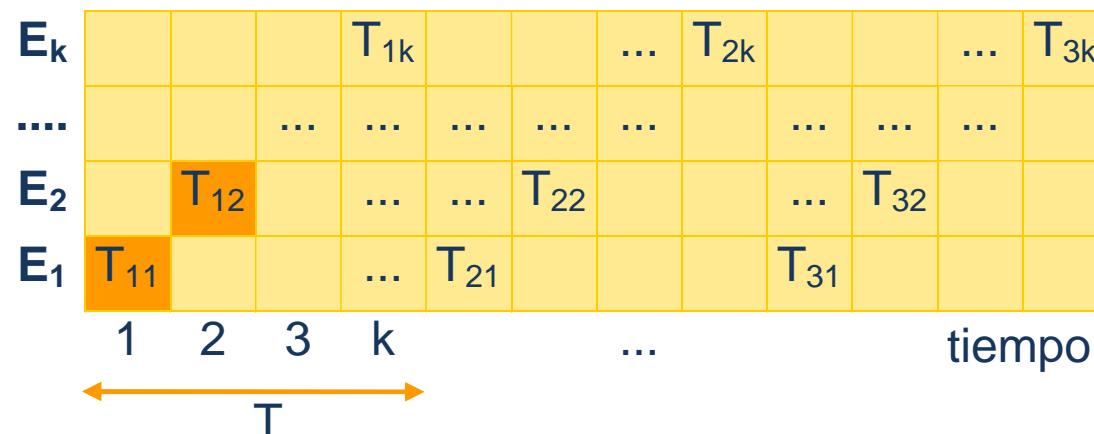
Segmentación

- Una tarea puede realizarse cuando todos los **recursos** que necesita estén **disponibles**



## Concepto. Secuencial

- ➊ Una tarea puede realizarse cuando todos los **recursos** que necesita estén **disponibles**



# Concepto. Secuencial

Introducción

Segmentación del repertorio

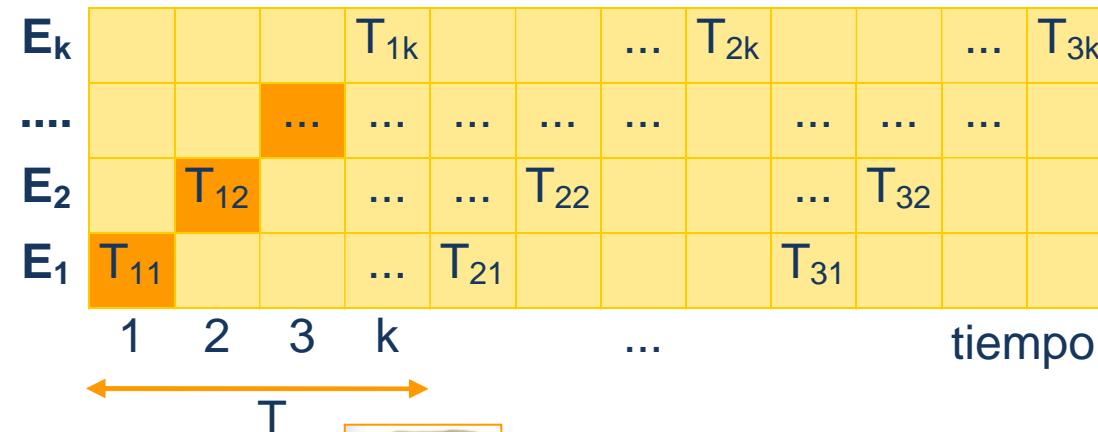
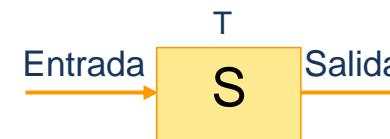
Caucos aritméticos

Optimización

Superescalares

Segmentación

- Una tarea puede realizarse cuando todos los **recursos** que necesita estén **disponibles**



# Concepto. Secuencial

Introducción

Segmentación del repertorio

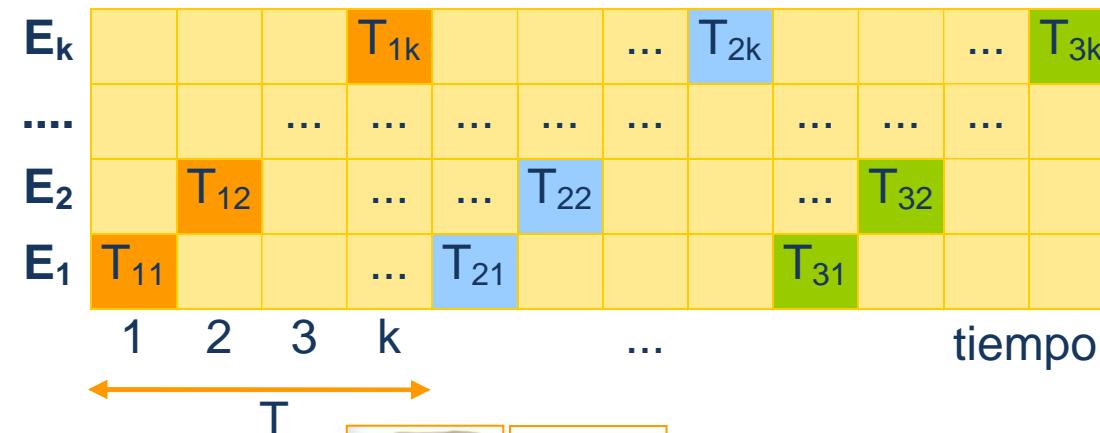
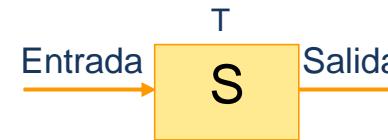
Cauces aritméticos

Optimización

Superescalares

Segmentación

- Una tarea puede realizarse cuando todos los **recursos** que necesita estén **disponibles**



# Concepto. Con segmentación

Introducción

Segmentación  
del repertorio

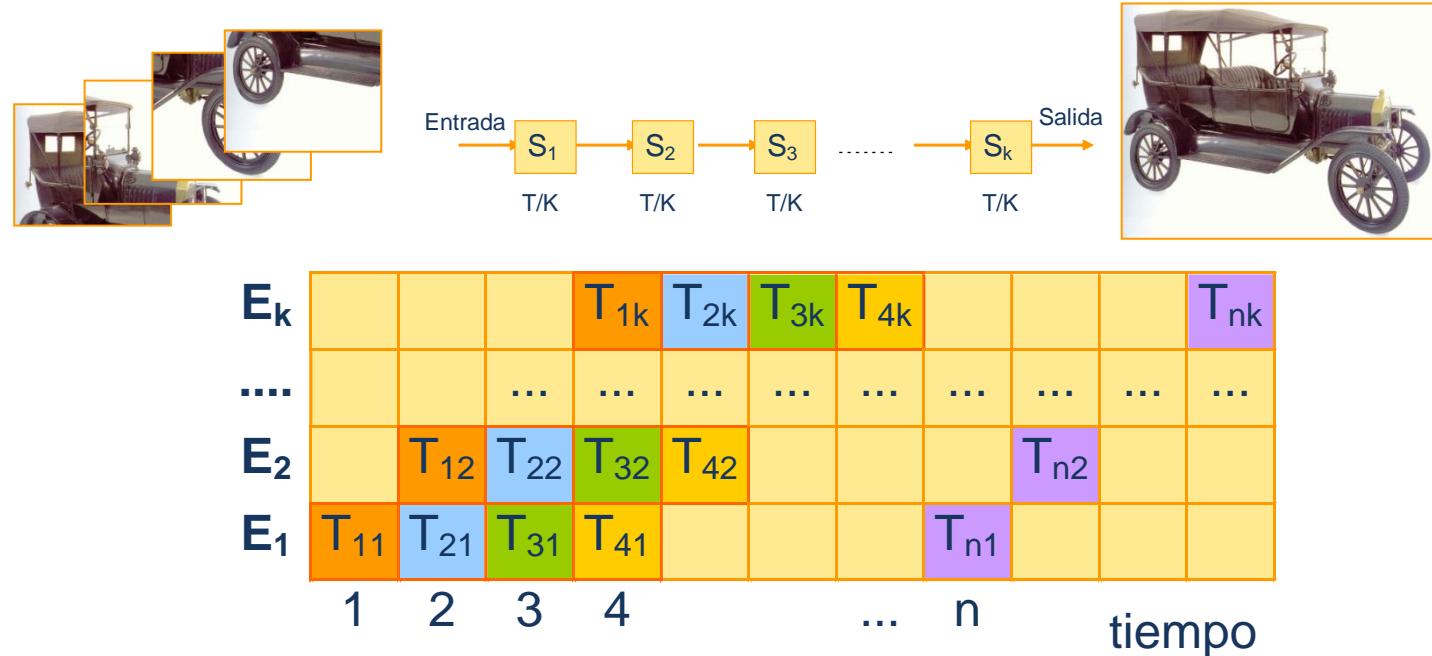
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- El comienzo de una tarea en una etapa sólo requiere la finalización de la tarea anterior en esa etapa



- Cada trabajador pasa al siguiente trabajador de la cadena el estado actual del trabajo mientras puede comenzar a trabajar en el siguiente vehículo

# Concepto. Con segmentación

Introducción

Segmentación  
del repertorio

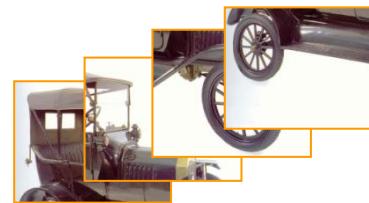
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- El comienzo de una tarea en una etapa sólo requiere la finalización de la tarea anterior en esa etapa



$E_k$				$T_{1k}$	$T_{2k}$	$T_{3k}$	$T_{4k}$				$T_{nk}$
....			...	...	...	...	...	...	...	...	...
$E_2$		$T_{12}$	$T_{22}$	$T_{32}$	$T_{42}$				$T_{n2}$		
$E_1$	$T_{11}$	$T_{21}$	$T_{31}$	$T_{41}$				$T_{n1}$			

1    2    3    4    ...    n    tiempo ciclos



# Concepto. Con segmentación

Introducción

Segmentación  
del repertorio

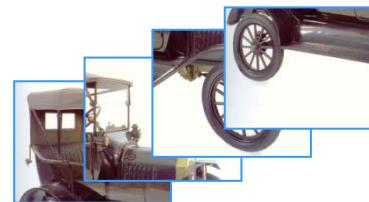
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ El comienzo de una tarea en una etapa sólo requiere la finalización de la tarea anterior en esa etapa



$E_k$				$T_{1k}$	$T_{2k}$	$T_{3k}$	$T_{4k}$				$T_{nk}$
....			...	...	...	...	...	...	...	...	...
$E_2$		$T_{12}$	$T_{22}$	$T_{32}$	$T_{42}$				$T_{n2}$		
$E_1$	$T_{11}$	$T_{21}$	$T_{31}$	$T_{41}$				$T_{n1}$			

1    2    3    4    ...    n    tiempo ciclos



# Concepto. Con segmentación

Introducción

Segmentación  
del repertorio

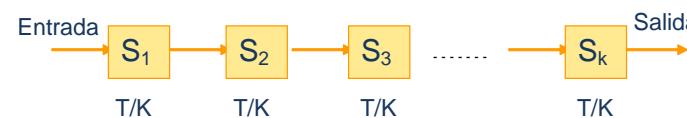
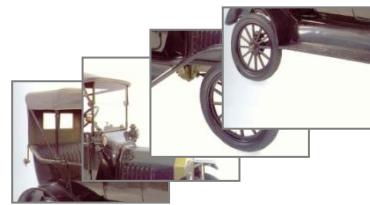
Cauces  
aritméticos

Optimización

Superescalares

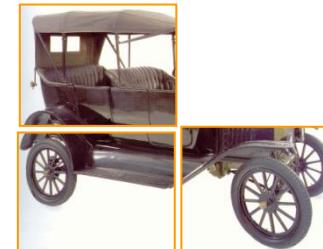
Segmentación

- El comienzo de una tarea en una etapa sólo requiere la finalización de la tarea anterior en esa etapa



$E_k$				$T_{1k}$	$T_{2k}$	$T_{3k}$	$T_{4k}$				$T_{nk}$
....			...	...	...	...	...	...	...	...	...
$E_2$		$T_{12}$	$T_{22}$	$T_{32}$	$T_{42}$					$T_{n2}$	
$E_1$	$T_{11}$	$T_{21}$	$T_{31}$	$T_{41}$				$T_{n1}$			

1    2    3    4    ...    n    tiempo ciclos



# Concepto. Con segmentación

Introducción

Segmentación  
del repertorio

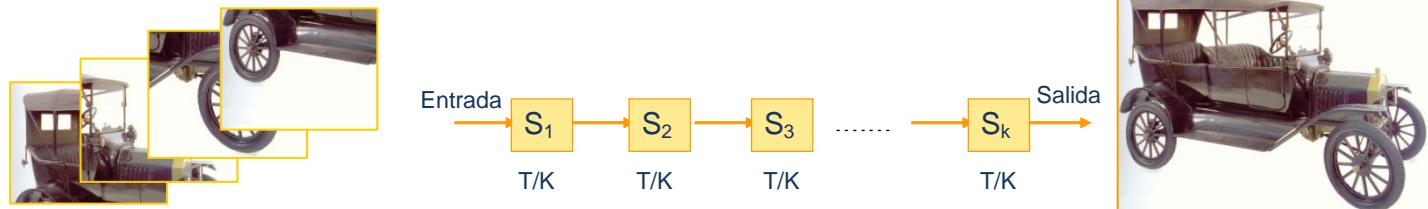
Caucos  
aritméticos

Optimización

Superescalares

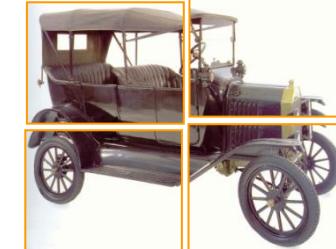
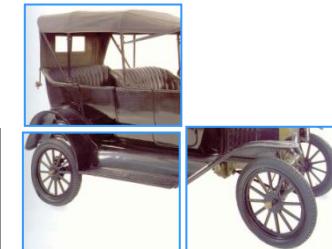
Segmentación

- El comienzo de una tarea en una etapa sólo requiere la finalización de la tarea anterior en esa etapa



$E_k$				$T_{1k}$	$T_{2k}$	$T_{3k}$	$T_{4k}$				$T_{nk}$
....			...	...	...	...	...	...	...	...	...
$E_2$		$T_{12}$	$T_{22}$	$T_{32}$	$T_{42}$				$T_{n2}$		
$E_1$	$T_{11}$	$T_{21}$	$T_{31}$	$T_{41}$				$T_{n1}$			

1    2    3    4    ...    n         tiempo ciclos



# Concepto. Con segmentación

Introducción

Segmentación  
del repertorio

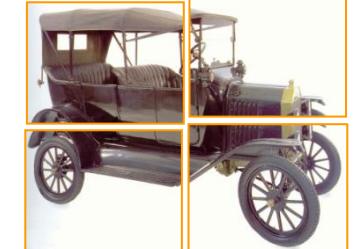
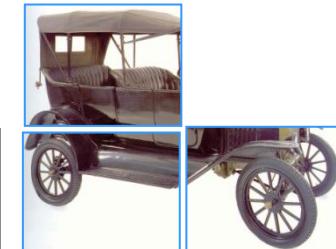
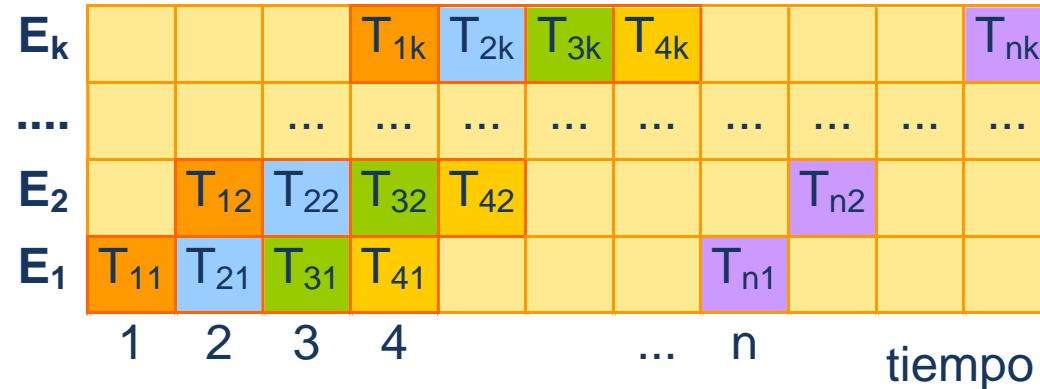
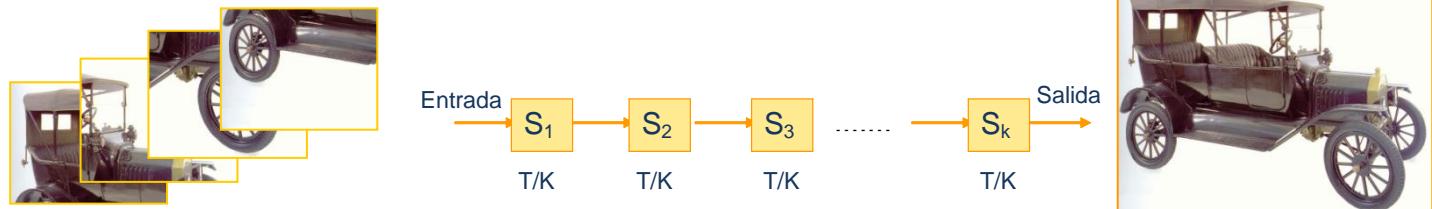
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- El comienzo de una tarea en una etapa sólo requiere la finalización de la tarea anterior en esa etapa



# Concepto. Con segmentación

Introducción

Segmentación  
del repertorio

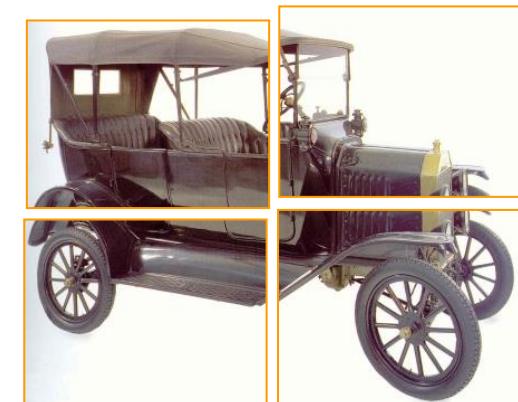
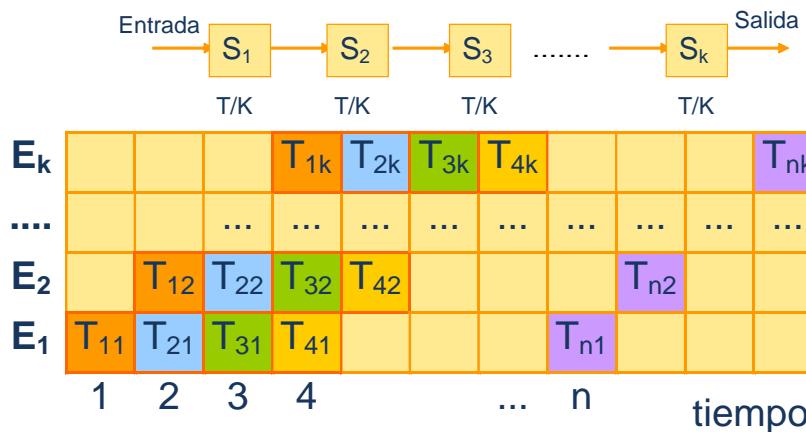
Caucos  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ Factor determinante: descomposición tarea a realizar en etapas
  - ◆ Distribución uniforme del tiempo (caso ideal)
    - ◆ Etapa más lenta actúa como cuello de botella
    - ◆ Ajustar el ritmo de trabajo a la etapa más lenta
- ◆ Es necesario contemplar:
  - ◆ Para que cada trabajador pueda pasar el trabajo necesita tiempo para preparar y distribuir la parte del vehículo que lleva fabricada
  - ◆ También tener en cuenta que la nueva organización necesaria para controlar el proceso puede ser muy compleja



# Clasificación procesadores segmentados

Introducción

Segmentación del repertorio

Cauces aritméticos

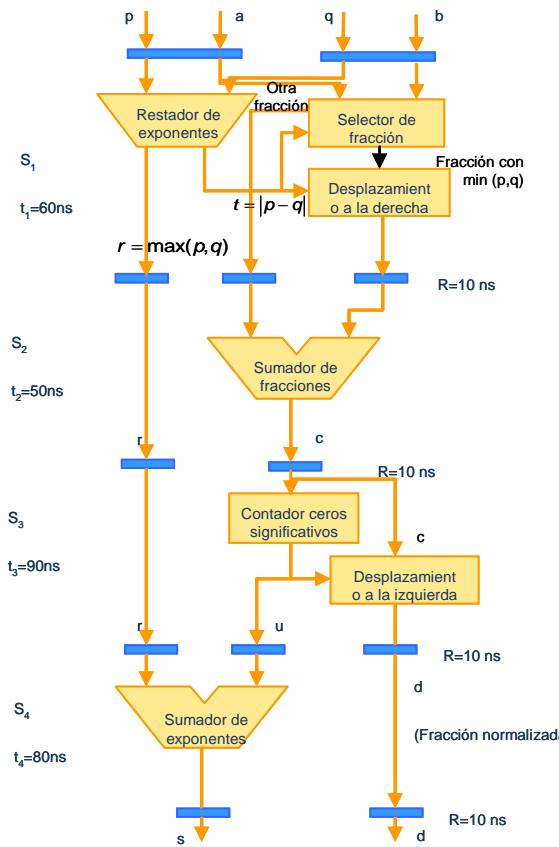
Optimización

Superescalares

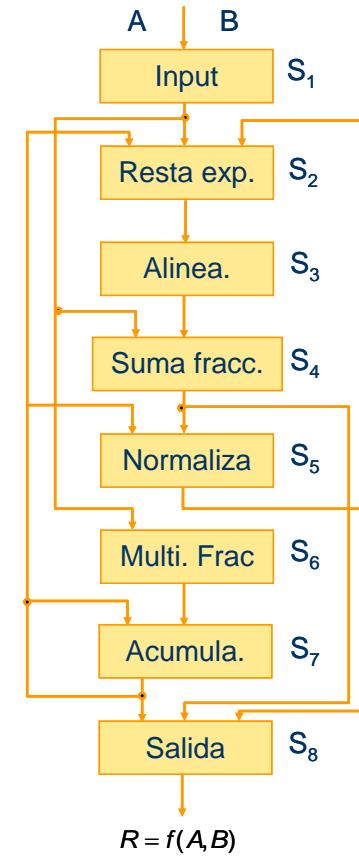
Segmentación

## Por tipo de cauce:

• **Unifunción:** ejecutan un único proceso (ejemplo: sumador).



• **Multifunción:** pueden ejecutar varios procesos (ejemplo: TI-ASC)



# Clasificación procesadores segmentados

Introducción

Segmentación del repertorio

Cauces aritméticos

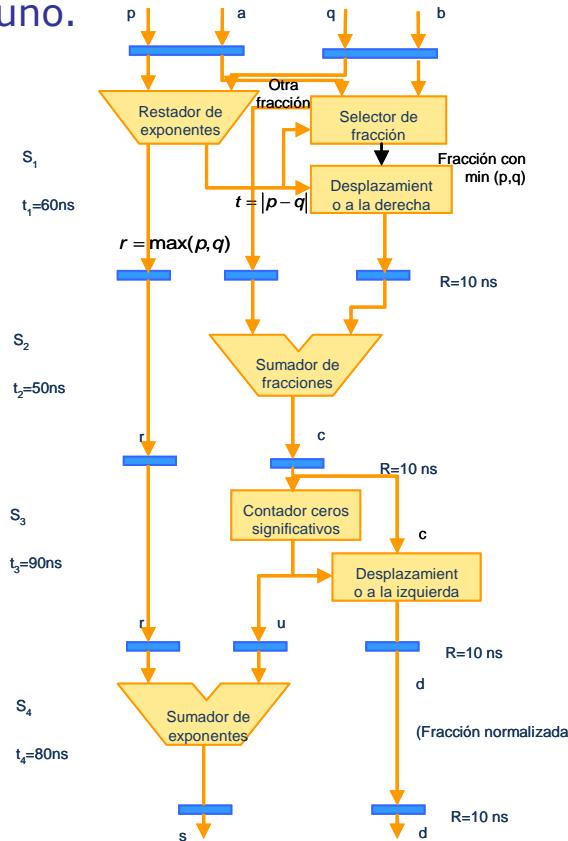
Optimización

Superescalares

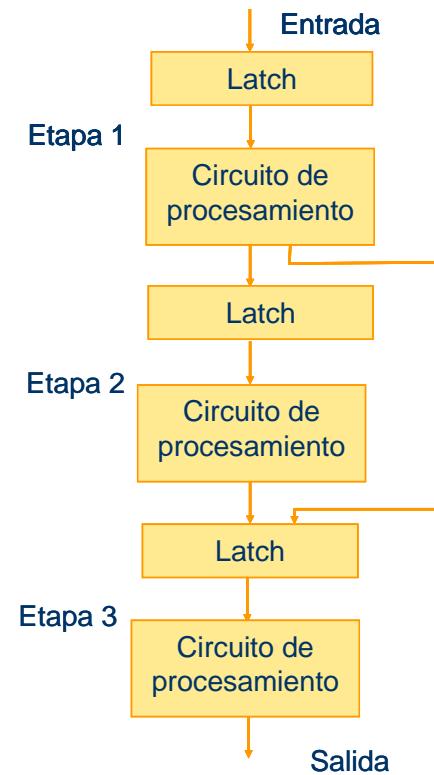
Segmentación

## Por tipo de cauce (multifunción):

- Estáticos: en un instante determinado sólo pueden ejecutar uno.



- Dinámicos: pueden ejecutar simultáneamente varios procesos.



# Clasificación procesadores segmentados

Introducción

Segmentación del repertorio

Cauces aritméticos

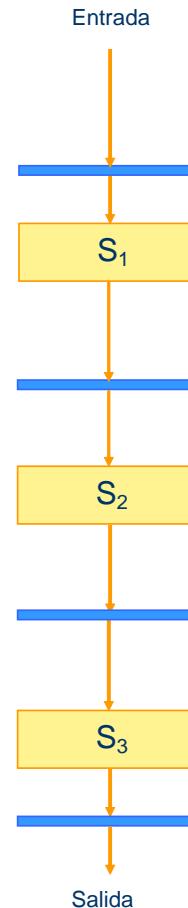
Optimización

Superescalares

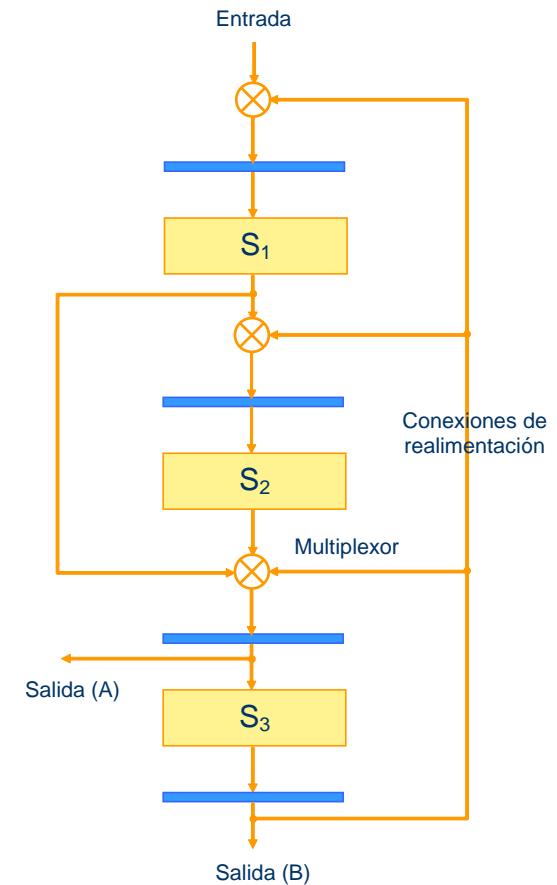
Segmentación

## Por tipo de cauce:

- **Lineal:** a cada etapa sólo le puede seguir otra etapa concreta.



- **No lineal:** se pueden establecer recorridos complejos de las etapas.



# Clasificación procesadores segmentados

Introducción

Segmentación del repertorio

Cauces aritméticos

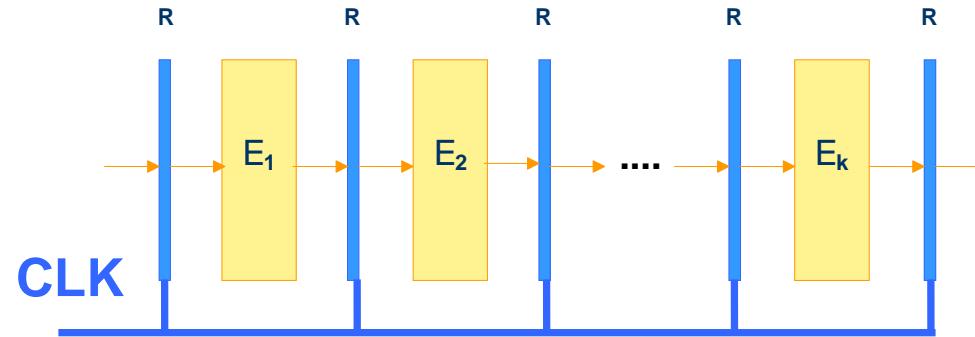
Optimización

Superescalares

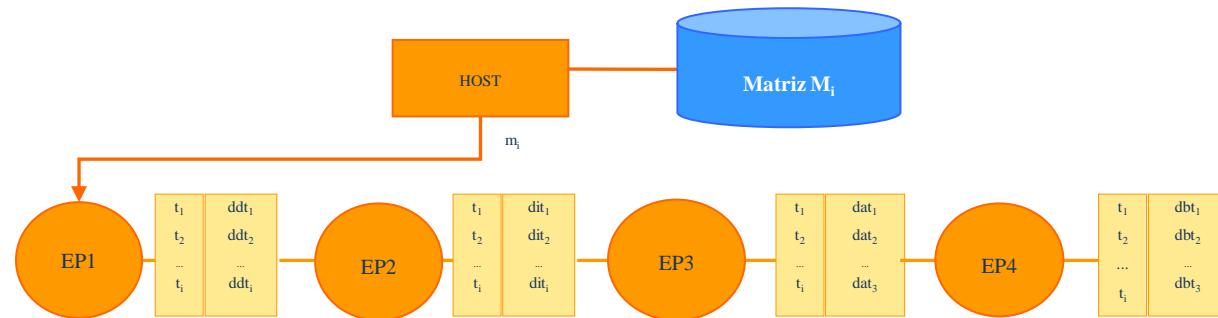
Segmentación

## Por tipo de cauce:

- ◆ **Síncrono:** Una señal de reloj.



- ◆ **Asíncrono:** Cada etapa funciona de forma autónoma.



- ◆ Protocolo comunicación entre etapas (semáforos, paso de mensajes).

# Clasificación procesadores segmentados

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Por niveles de aplicación:

### Segmentación aritmética

- ◆ Ejecutan una o varias operaciones de la ALU
- ◆ Pueden ser lineales (sumas) o no lineales (división). En este caso suelen ser cíclicos (bucles).
- ◆ Los procesadores actuales incluyen varias ALUs segmentadas y cada una se puede ocupar de varias operaciones.

### Segmentación de instrucciones

- ◆ Suelen ser cauces lineales
- ◆ Algunas de sus fases pueden a su vez sub-segmentarse (uso de una ALU segmentada para la fase de ejecución)

### Segmentación de procesadores

# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Se descompone cada operación aritmética en distintas etapas y se realiza un diseño encauzado de la ALU
- ➋ Técnica empleada por la mayoría de los computadores.
  - ➌ Unidad de punto flotante de tres etapas del clásico del MC68040.
  - ➌ La unidad de punto flotante del IBM 360/Model 91 con dos cauces. Un sumador de 2 etapas y un multiplicador de 6.
  - ➌ El cauce multifunción de 8 etapas del TI ASC.
  - ➌ Operaciones vectoriales del Cray-1 desde 1 a 14 etapas dependiendo del tipo de operación
  - ➌ Supercomputadores NEC SX2, Fujitsu VP400, Hitachi S820, Cray Y-MP
  - ➌ Procesadores NEC SX6 utilizado en los 640 nodos del Earth Simulator
  - ➌ Unidades punto flotante, MMX y SSE de microprocesadores de Intel y AMD

# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

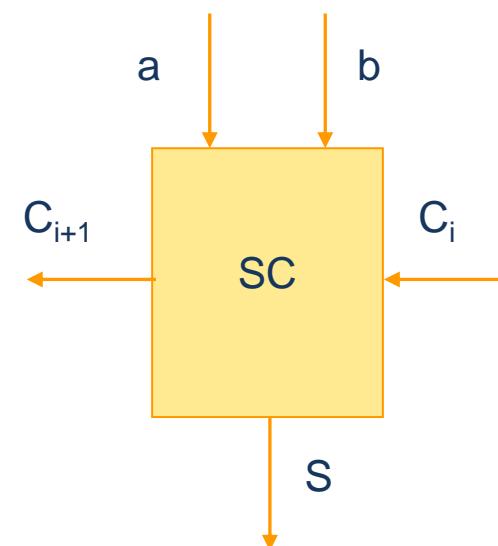
Optimización

Superescalares

Segmentación

- ➊ Sumador de tres bits con propagación de acarreo
  - ➋ Compararemos el caso secuencial frente al segmentado

## Sumador completo



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

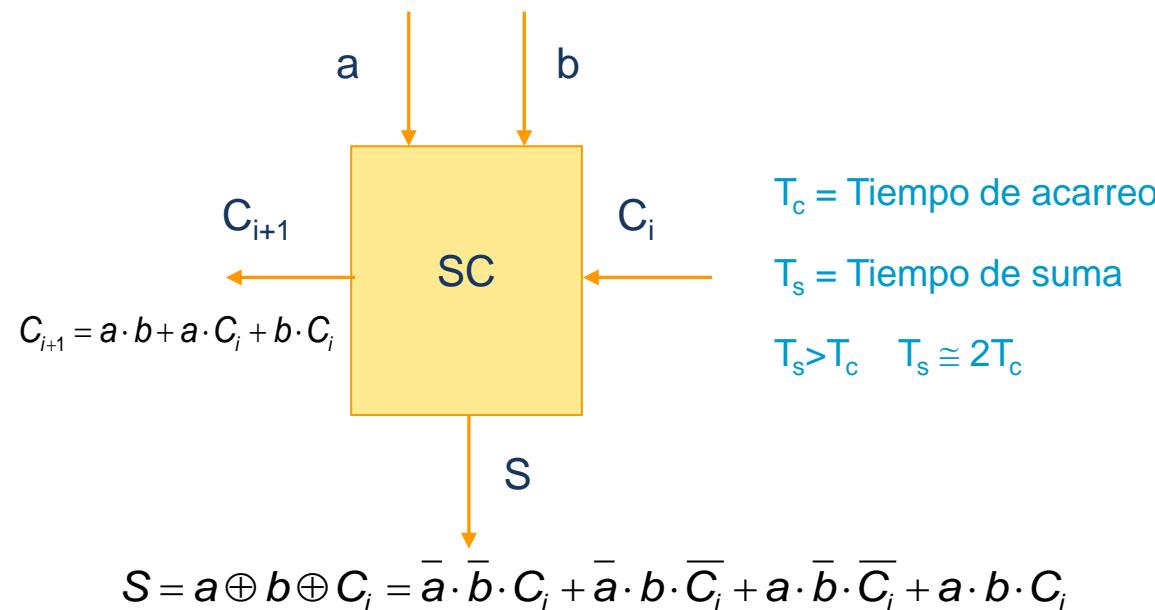
Superescalares

Segmentación

## Sumador de tres bits con propagación de acarreo

- Compararemos el caso secuencial frente al segmentado

### Sumador completo



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

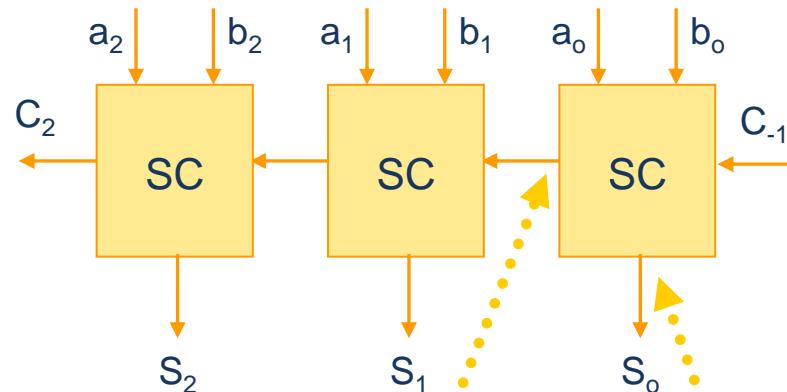
Optimización

Superescalares

Segmentación

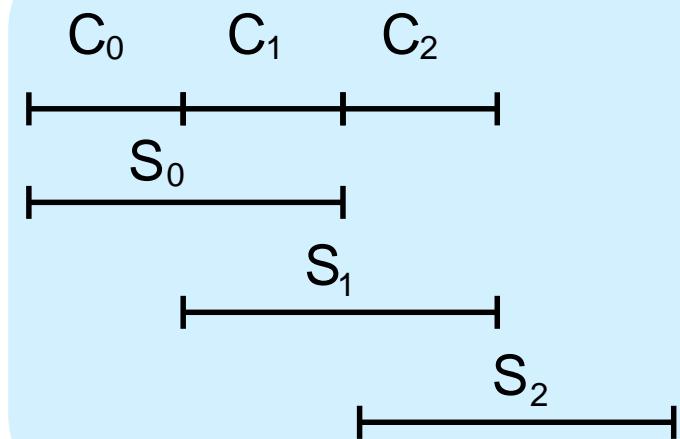
- Sumador de tres bits con propagación de acarreo

## Caso secuencial



1º El  
acarreo

2º La  
suma



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

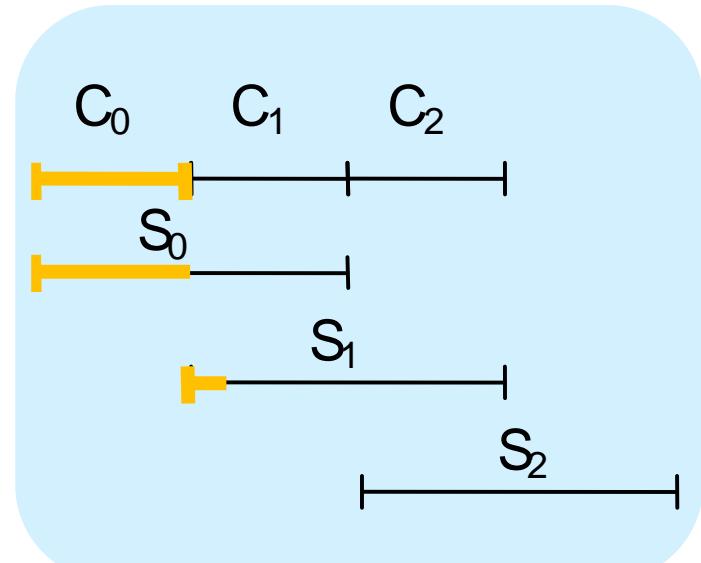
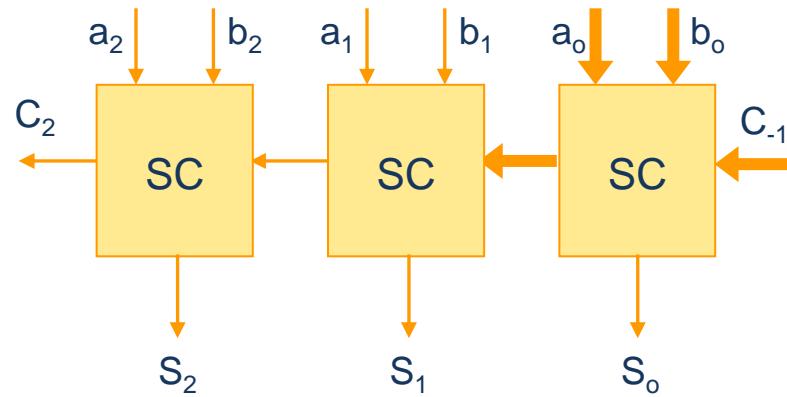
Optimización

Superescalares

Segmentación

- Sumador de tres bits con propagación de acarreo

## Caso secuencial



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

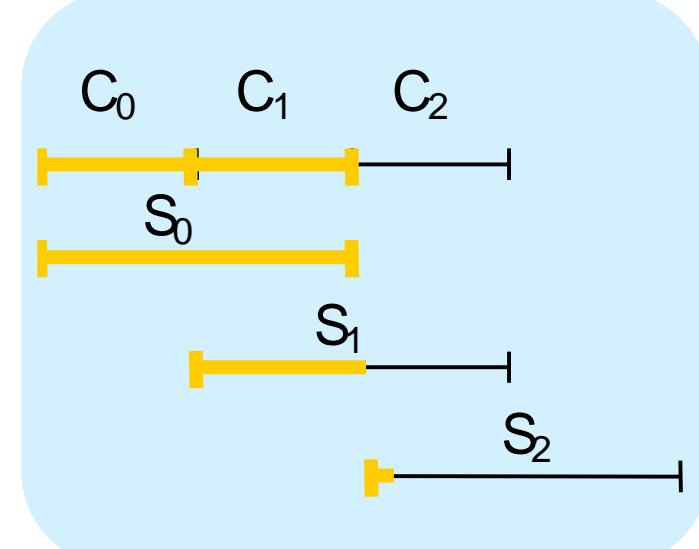
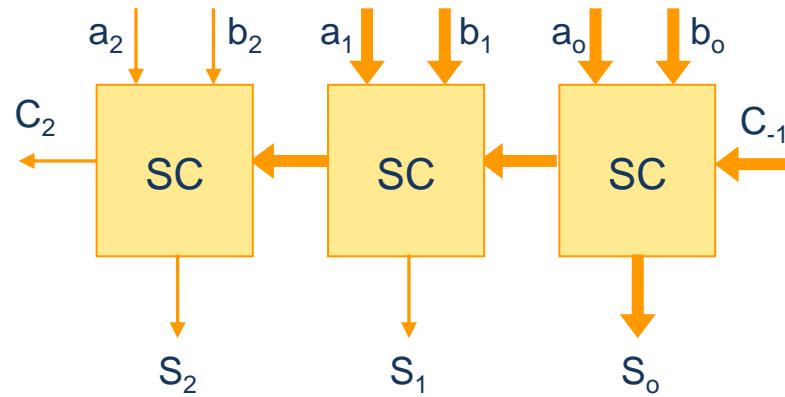
Optimización

Superescalares

Segmentación

- Sumador de tres bits con propagación de acarreo

## Caso secuencial



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

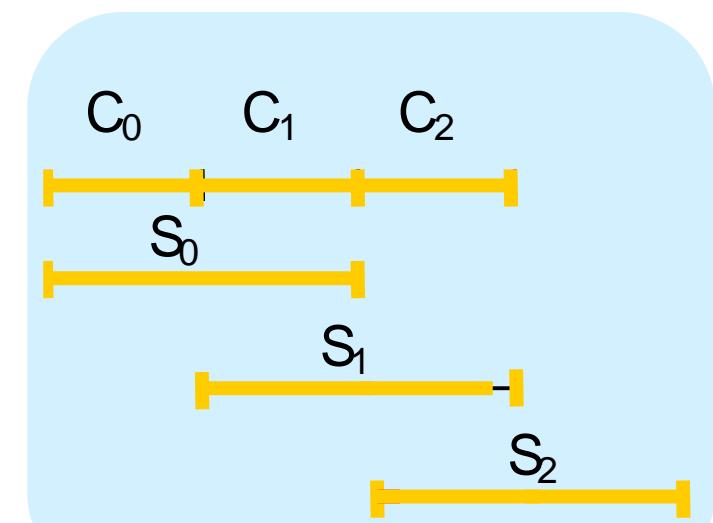
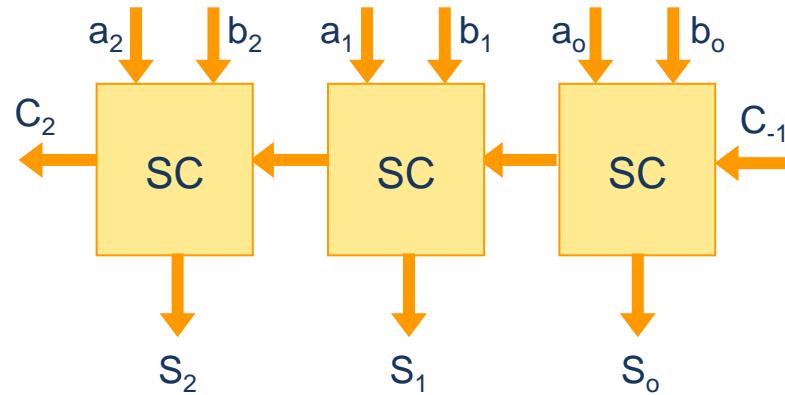
Optimización

Superescalares

Segmentación

- Sumador de tres bits con propagación de acarreo

## Caso secuencial



$$T_{\text{suma}} = 2 T_c + T_s$$

# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

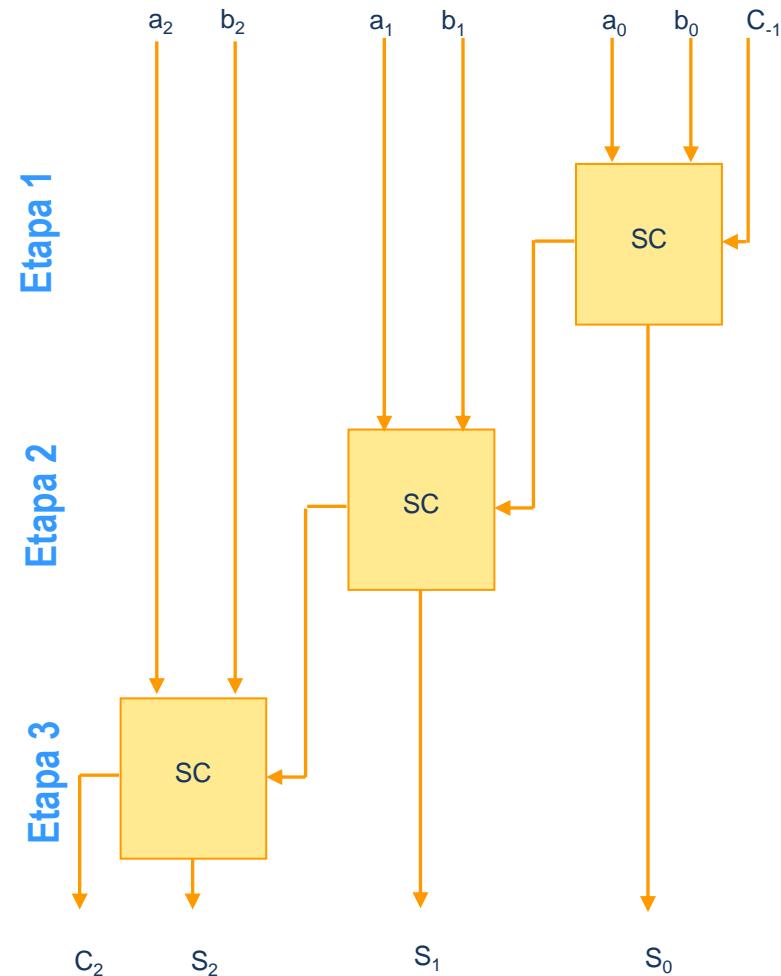
Superescalares

Segmentación

## Sumador de tres bits con propagación de acarreo

### Caso segmentado

- Bloque constructivo SC 1 bit
- Dividir en etapas cada una de las sumas parciales de los bits de los números



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

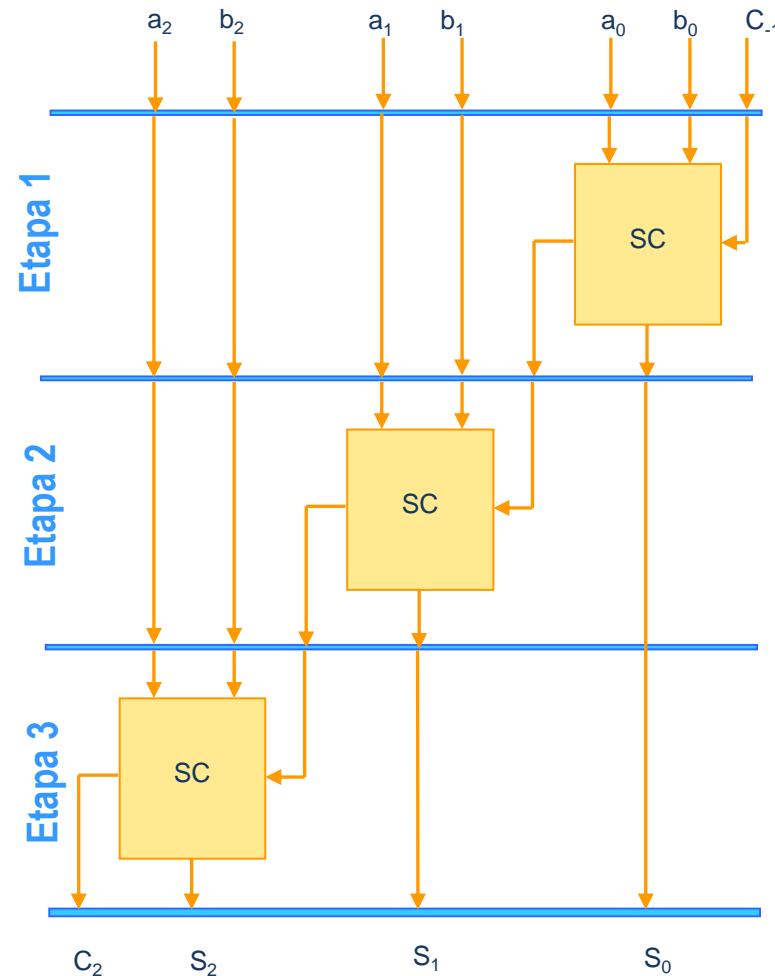
Superescalares

Segmentación

## Sumador de tres bits con propagación de acarreo

### Caso segmentado

- Bloque constructivo SC 1 bit
- Dividir en etapas cada una de las sumas parciales de los bits de los números
- Introducir registros intermedios entre cada una de las etapas para pasar datos (secuencial)



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

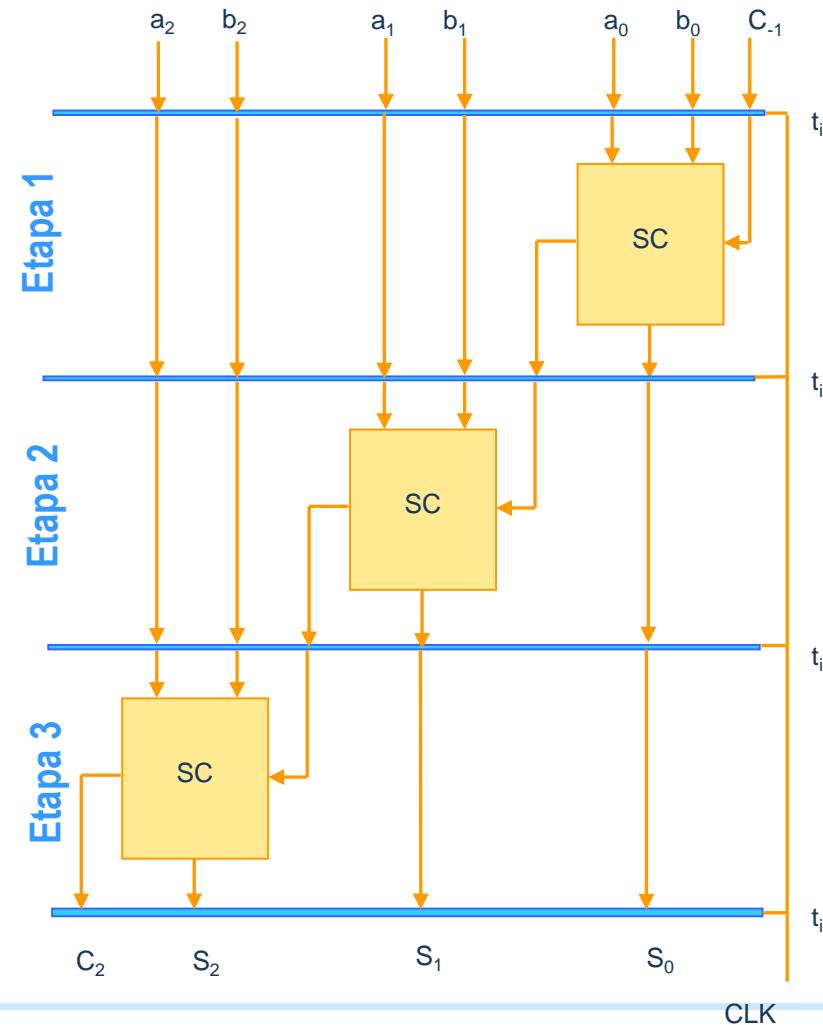
Segmentación

- Sumador de tres bits con propagación de acarreo

## Caso segmentado

- Bloque constructivo SC 1 bit
- Dividir en etapas cada una de las sumas parciales de los bits de los números
- Introducir registros intermedios entre cada una de las etapas para pasar datos (secuencial)
- (Ritmo) Duración del ciclo de reloj clk: tiempo de la etapa más lenta,  $T_s$

$$T_{\text{suma}} = 3 (T_s + t_i)$$



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

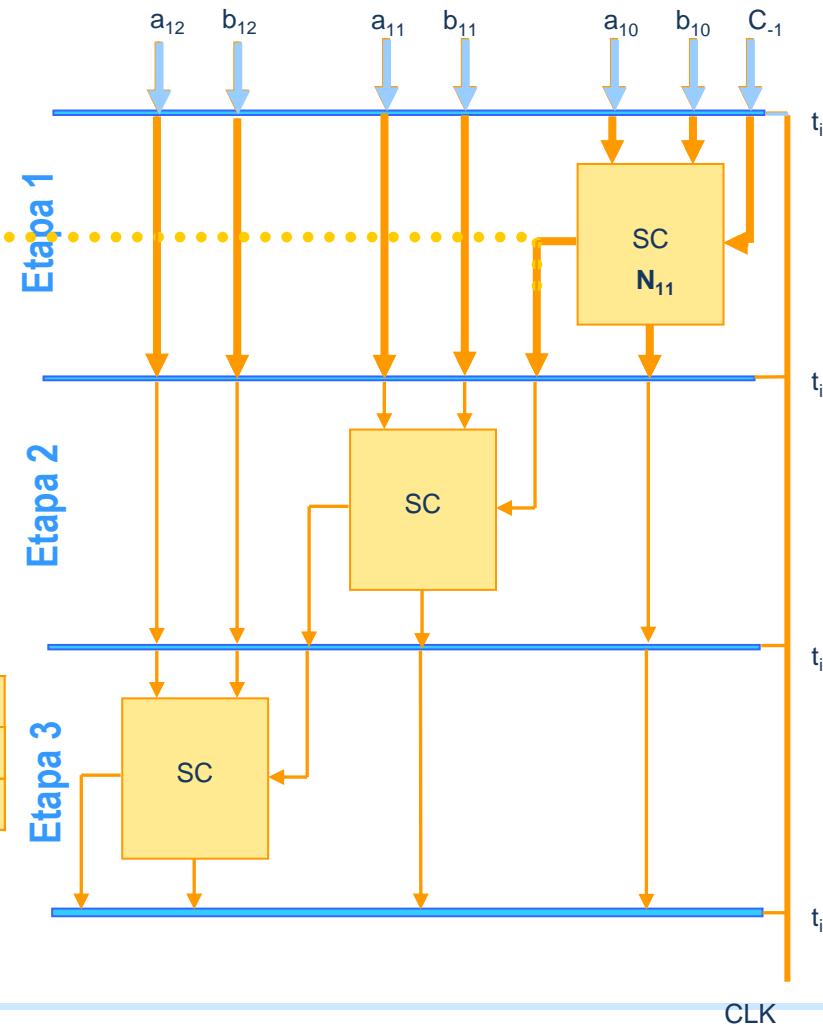
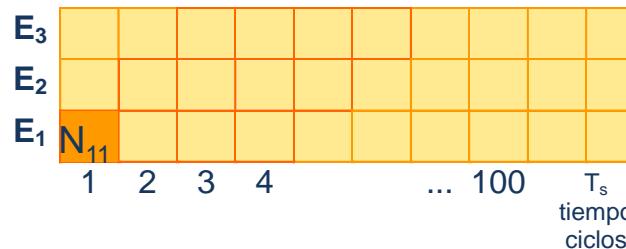
Superescalares

Segmentación

- Sumador de tres bits con propagación de acarreo

## Caso segmentado

Primer bit, primer número



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

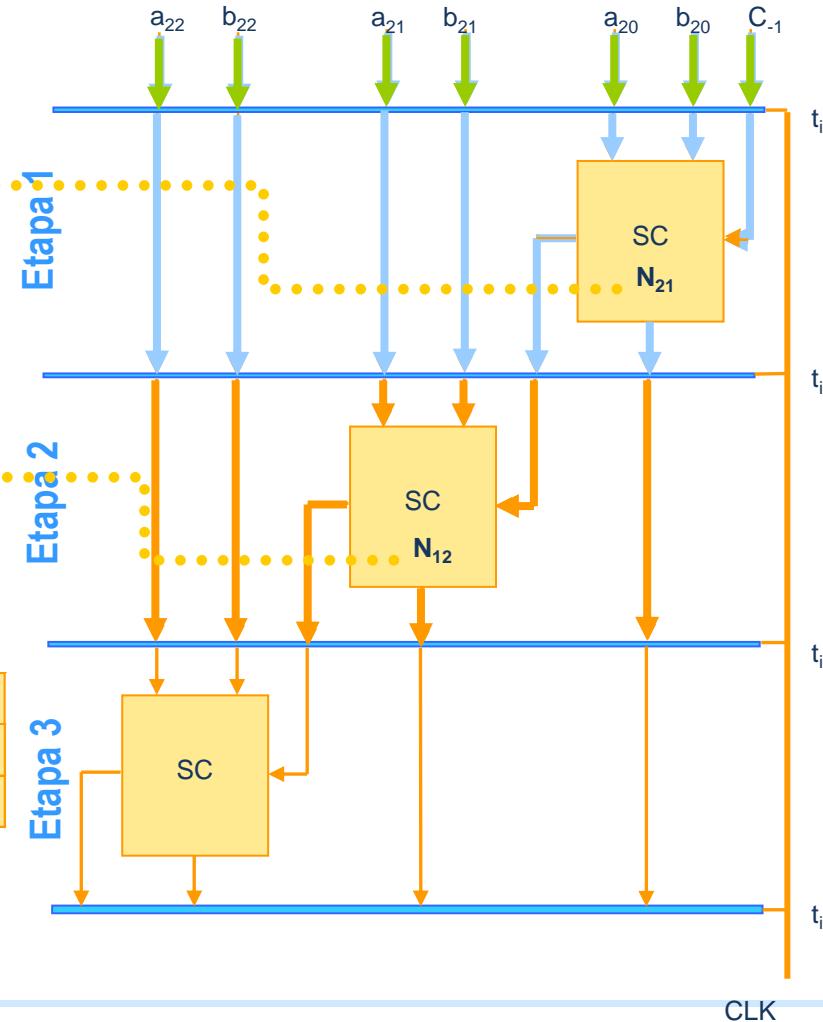
Superescalares

Segmentación

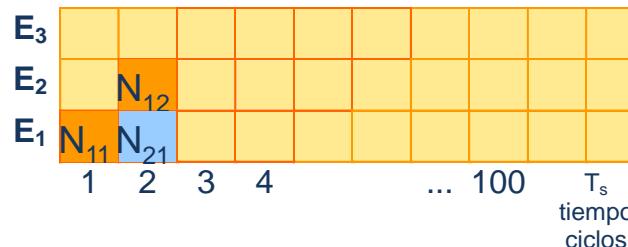
- Sumador de tres bits con propagación de acarreo

## Caso segmentado

Primer bit, segundo número



Segundo bit, primer número



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

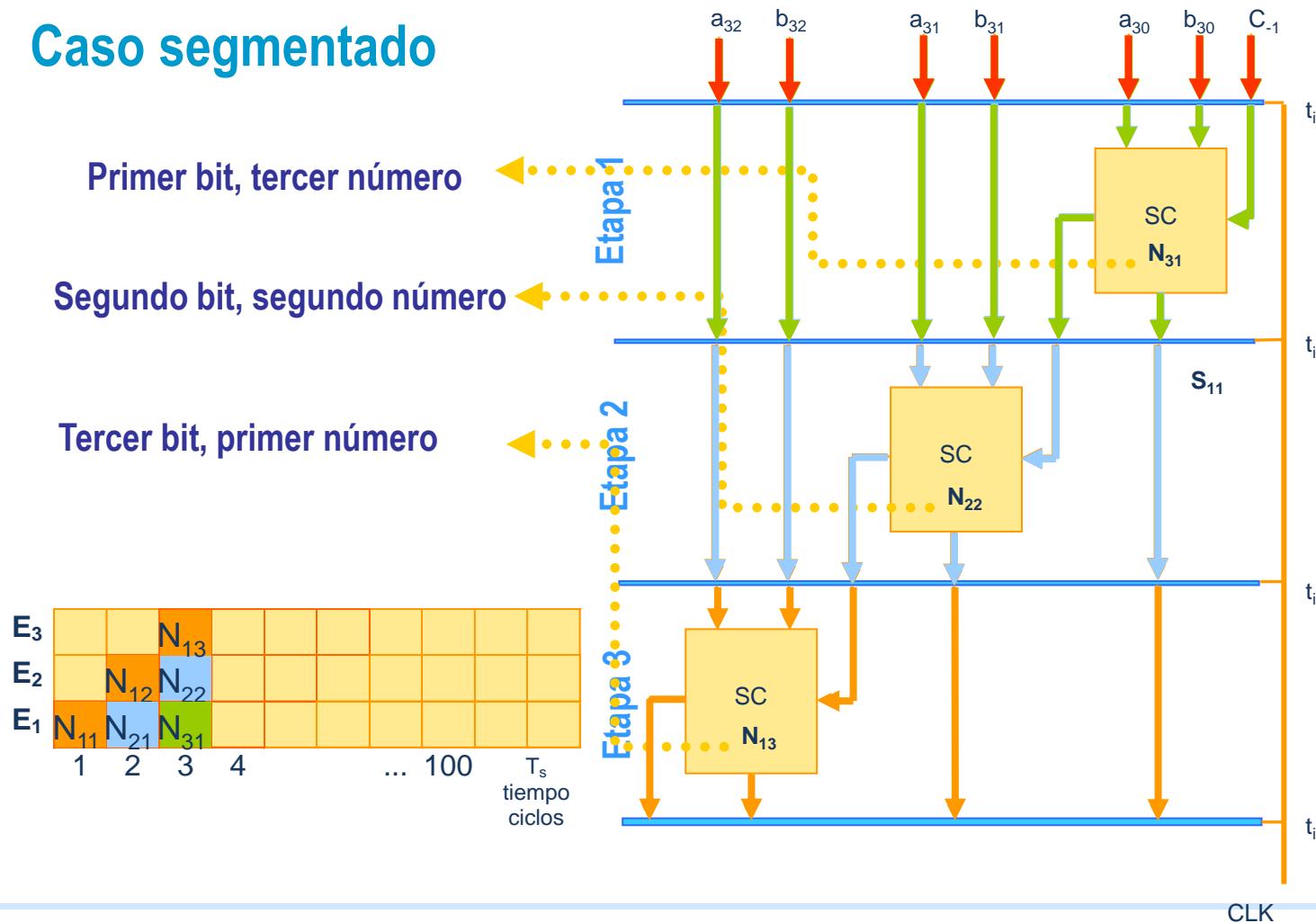
Optimización

Superescalares

Segmentación

- Sumador de tres bits con propagación de acarreo

## Caso segmentado



# Segmentación aritmética

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Sumador de tres bits con propagación de acarreo

- El coste temporal de una suma aislada es incluso mayor.
- Tiempo secuencial para 100 números:

$$T_{\text{Secuencial}} = 100(2T_c + T_s) = 200T_s$$

$$\text{Si } T_c \approx \frac{1}{2}T_s$$

- Tiempo segmentado para 100 números:

$$T_{\text{segmentado}} = 3T_s + 99T_s = 102T_s$$

- Aceleración de aproximadamente 1,96 para 100 números

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ La instrucción a ejecutar se descompone en fases y se encauzan
- ➋ Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ La instrucción a ejecutar se descompone en fases y se encauzan
- ➋ Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

## Hitos

1955      1960      1965      1970      1975      1980      1985      1990



**IBM 709 (1958): primer computador segmentado.** Computador de tubos de vacío y memorias de núcleo de ferrita

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

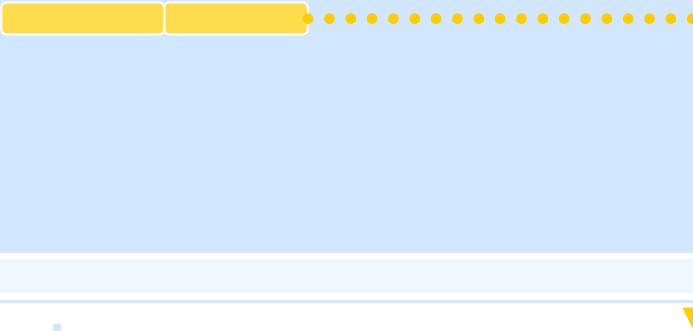
Superescalares

Segmentación

- ➊ La instrucción a ejecutar se descompone en fases y se encauzan
- ➋ Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

## Hitos

1955      1960      1965      1970      1975      1980      1985      1990



IBM 7030 Stretch (1962): aparece por primera vez el término segmentación. Primer procesador de propósito general segmentado.

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ La instrucción a ejecutar se descompone en fases y se encauzan
- ➋ Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

## Hitos



Finales 60: la **segmentación** pasa a un **nivel secundario** dado el énfasis que se realiza en las máquinas orientadas a simplificar el software (VAX).

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- La instrucción a ejecutar se descompone en fases y se encauzan
- Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

## Hitos



Años 80: las **arquitecturas RISC** retoman la segmentación. Las características favorecen el diseño del cauce. Se diseñaron para ser procesadores segmentados

- MIPS R2000 de 5 etapas

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cuces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ La instrucción a ejecutar se descompone en fases y se encauzan
- ➋ Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

## Hitos



Mediados de los 80: los **microprocesadores** incorporan **segmentación de cauce internamente de forma generalizada**

- Intel 80286 con 4 etapas
- Intel 80386 de 6 etapas
- Intel 80486 de 5 etapas

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

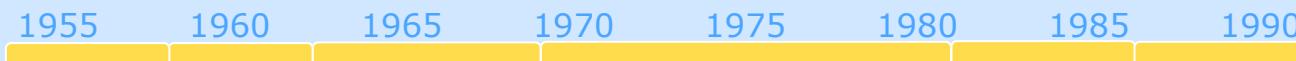
Optimización

Superescalares

Segmentación

- La instrucción a ejecutar se descompone en fases y se encauzan
- Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

## Hitos



Los 90: PCs traducen instrucciones **CISC a microoperaciones RISC** superescalares

- Pentium PRO, Pentium II, III y 4 de Intel
- K5, K6, K7, Athlon de AMD

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ La instrucción a ejecutar se descompone en fases y se encauzan
- ➋ Esta técnica o su fundamento, se emplea en la mayoría de los procesadores para aumentar el rendimiento de la CPU

**Los 90:** reutilización de la técnica en microprocesadores **embebidos**.

- MIPS R4300 (1997). Fabricado por NEC. Gran cantidad de aplicaciones.
  - **Nintendo 64.**
  - Utiliza las 5 etapas MIPS.
- ARM de Advanced Risc Machines y SH de Hitachi. Rediseño de la arquitectura RISC para proporcionar un cociente prestaciones/potencia
  - **PDAs, TDTs** y dispositivos similares.
- Se introducen **más recursos** para trabajar de forma paralela
  - MIPS R4000: uno de los primeros en incorporar un cauce profundo, supersegmentado de 8 etapas
  - Alpha 21064: similar al MIPS R4000 para tratar enteros y una mayor segmentación para punto flotante
  - Procesadores más modernos (MIPS R10000/12000, Sun Ultra Sparc III, PowerPC 603, G3 y G4, Alpha 21264.)

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Procesador MIPS (DLX)

- ◆ **MIPS64** procesador **diseñado** expresamente para **trabajar de forma segmentada.**
- ◆ Su **simplicidad** hace más fácil demostrar los principios
  - ◆ Sencilla arquitectura de **carga almacenamiento**.
  - ◆ Sencillo repertorio de **instrucciones fácilmente decodificables**
  - ◆ Diseño de segmentación **eficiente**
  - ◆ Tres tipos de instrucciones: ALU, carga/almacenamiento, saltos

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **Paso IF.** Búsqueda de instrucción
- ➋ **Paso ID.** Decodificación de instrucción y búsqueda de registros
  - ➌ Formato permite leer los registros ya que siempre están ubicados en las mismas posiciones
  - ➌ Instrucción de salto: calcular la dirección y hacer la comprobación de los registros (dependerá de la implementación)
- ➌ **Paso EX.** Ejecución y cálculo de direcciones efectivas
  - ➌ Los operandos en memoria sólo en las operaciones de carga y almacenamiento por lo que en esta fase se puede calcular la dirección de memoria.
- ➌ **Paso MEM.** Acceso a memoria
  - ➌ Instrucción de carga se accede a memoria para la lectura del dato.
  - ➌ Instrucción almacenamiento se guarda el valor del registro en memoria.
- ➌ **Paso WB.** Postescritura, escritura del resultado en un registro

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

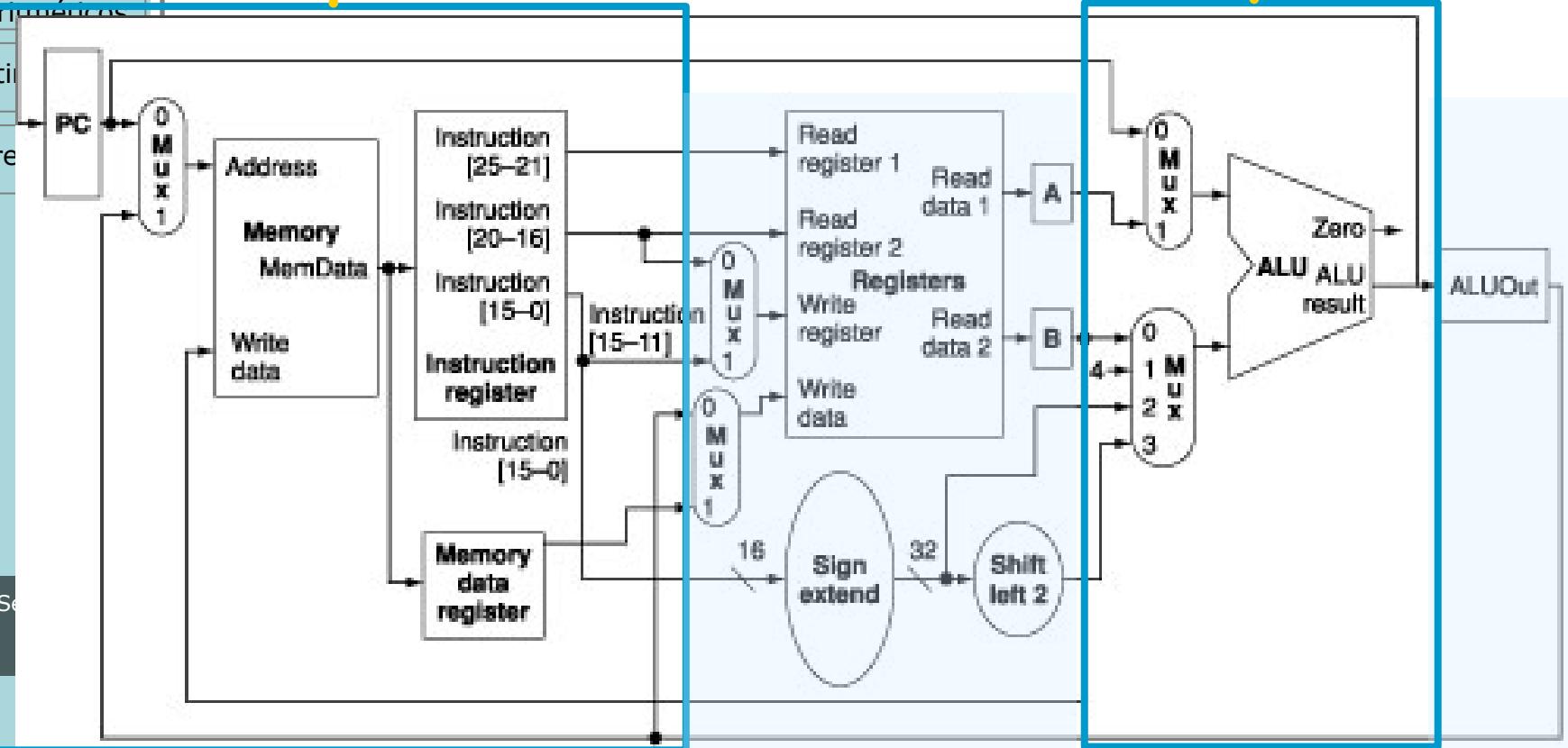
Opciones

Supere

IF: Búsqueda de instrucción

IR  $\leftarrow M[PC]$ ; PC  $\leftarrow PC + 4$

## Implementación multiciclo



# Segmentación de instrucciones

Introducción

Segmentación  
del reportorio

Cauces

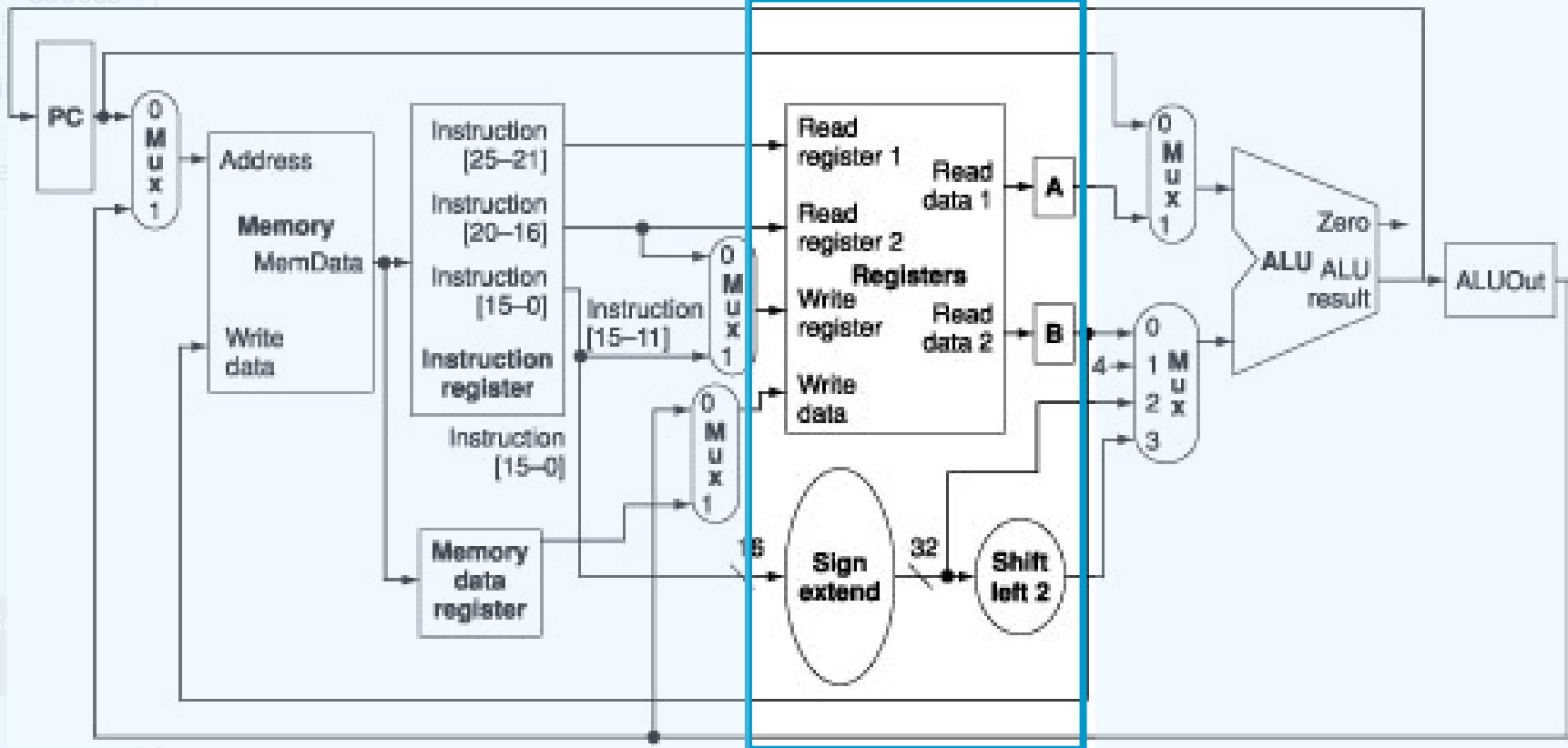
Opcionales

Superestructuras

## Implementación multiciclo

ID: Decodificación de instrucción y búsqueda de registros

$A \leftarrow R_s1; B \leftarrow R_s2;$

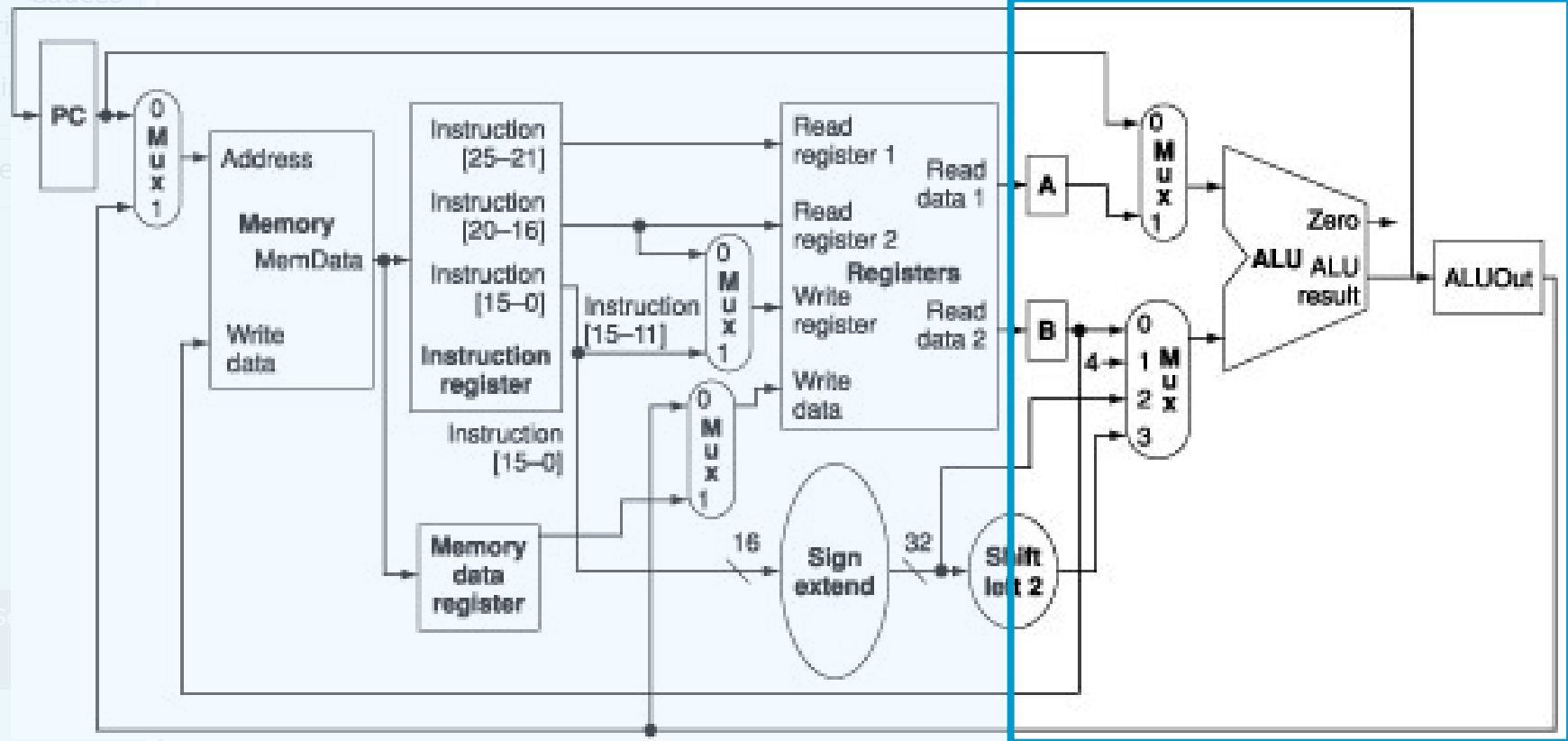


# Segmentación de instrucciones

Introducción

Segmentación  
del reportorio

## Implementación multiciclo



EX: Cálculo de direcciones efectivas

$$\text{ALUOut} \leftarrow A + (\text{IR16})16\# \text{IR16..31};$$

# Segmentación de instrucciones

Introducción

Segmentación  
del reportorio

Cauces

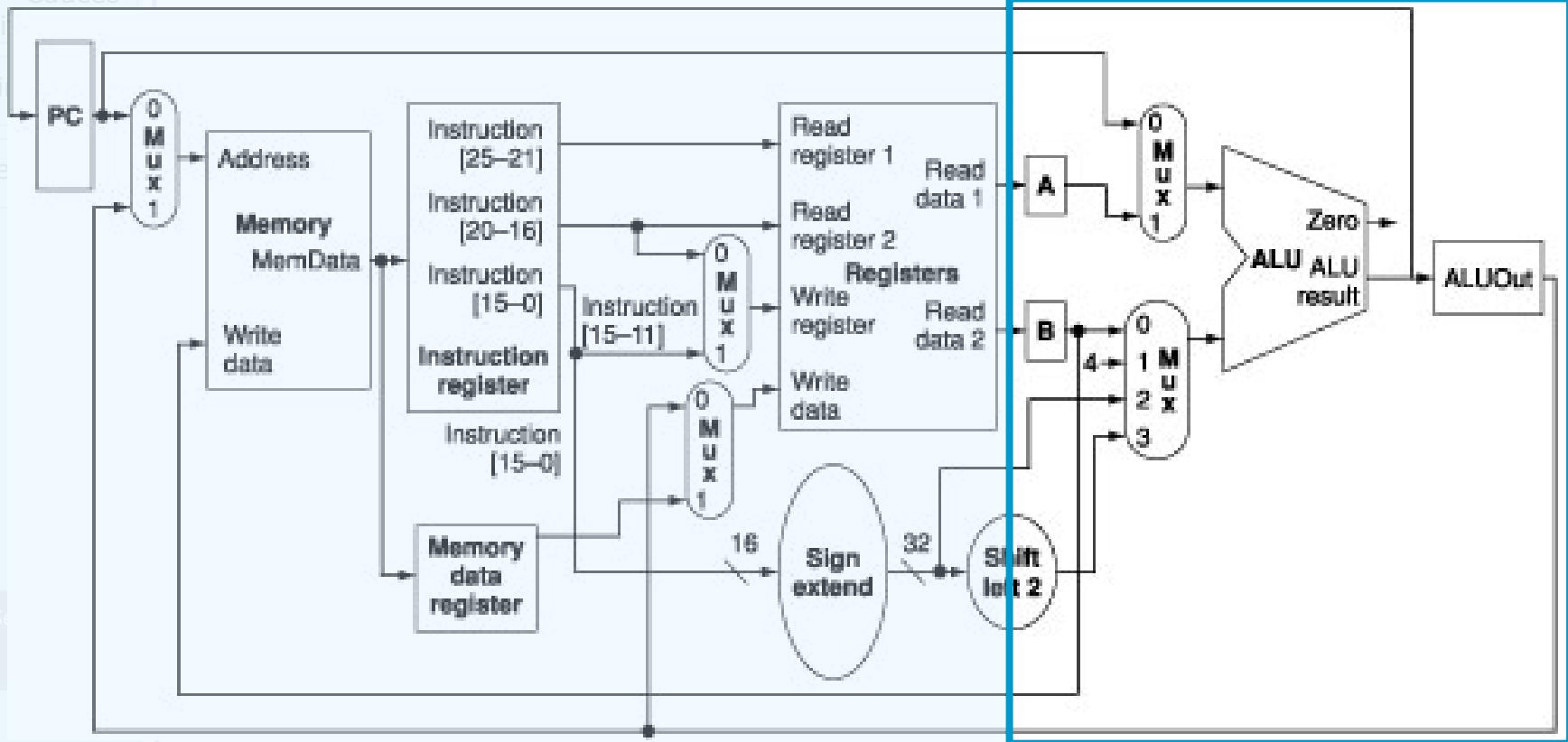
Opciones

Supere

## Implementación multiciclo

EX: Ejecución

$$\text{ALUoutput} \leftarrow A \text{ op } (B \text{ or } (\text{IR16})16\#\#\text{IR16..31})$$



# Segmentación de instrucciones

Introducción

Segmentación  
del reportorio

Cauces

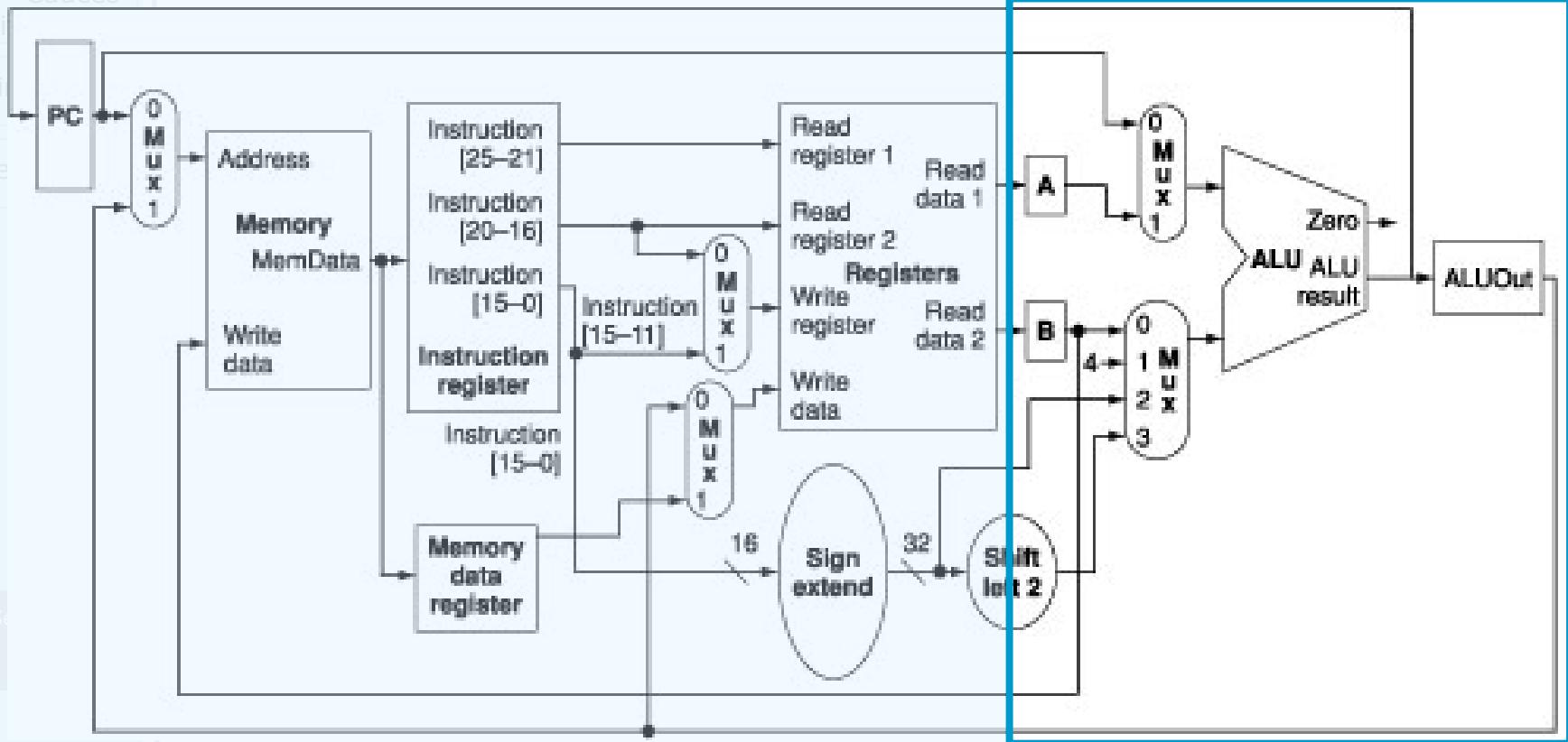
Optimizac.

Supere

## Implementación multiciclo

EX: Salto/Bifurcación

$$\text{ALUoutput} \leftarrow \text{PC} + (\text{IR16})16\#\# \text{IR16.31}$$

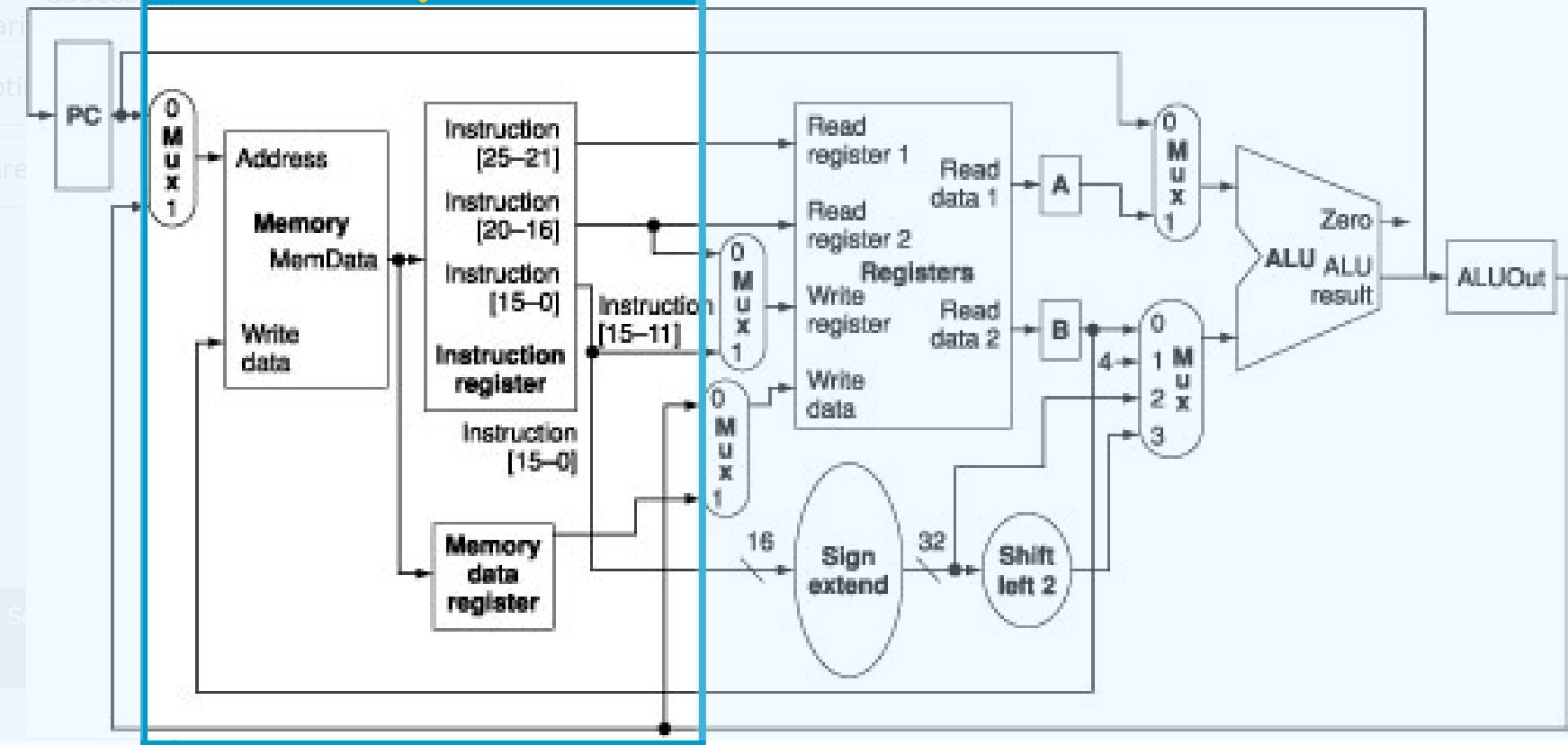


# Segmentación de instrucciones

Introducción

Segmentación  
del reportorio

## Implementación multiciclo



MEM: Referencia memoria

MDR  $\leftarrow$  M [ALUOut]

M[ALUOut]  $\leftarrow$  B

## Introducción

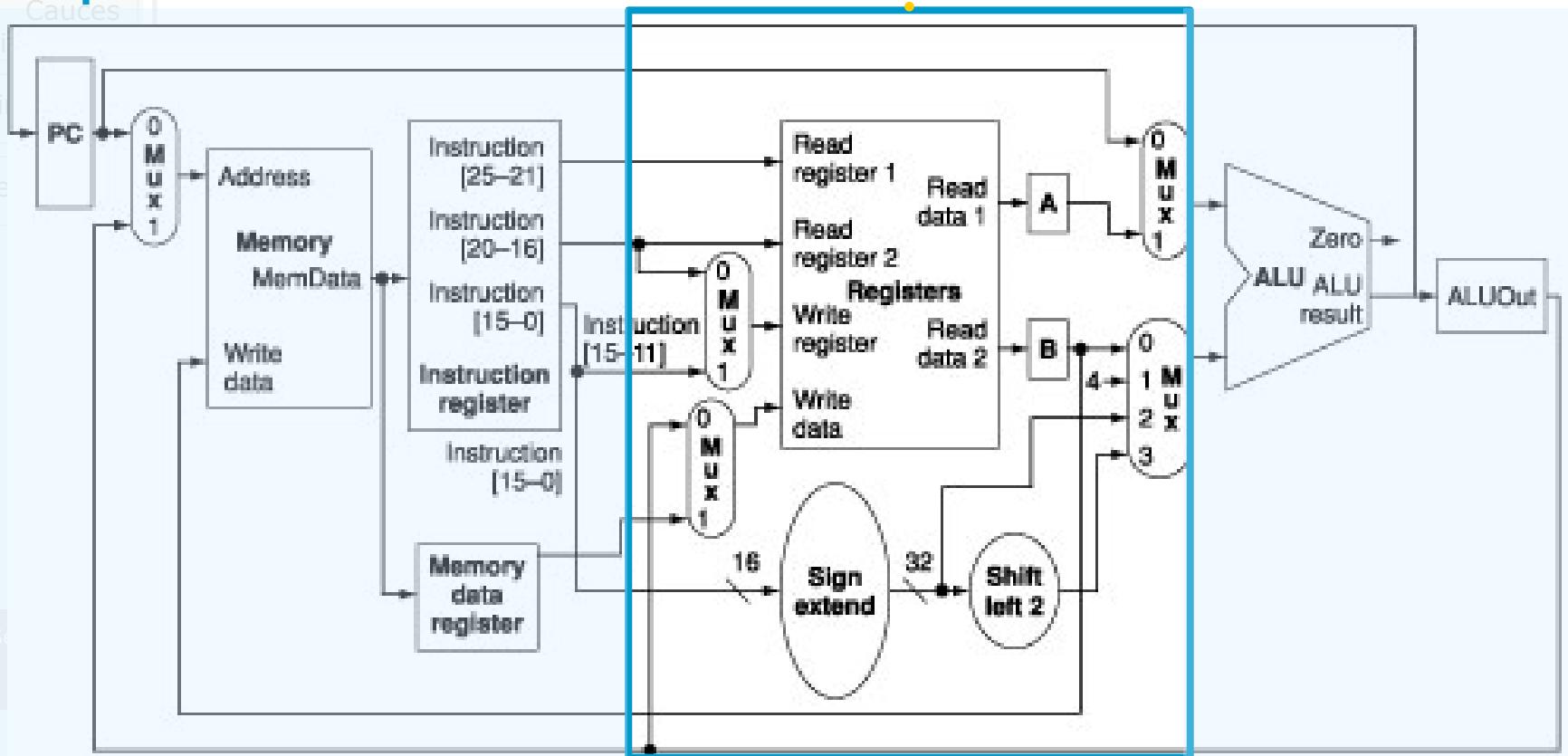
# Segmentación del repertorio

Cautes

Opti

**Supere**

# Implementación multiciclo



# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

- La segmentación consiste en solapar la ejecución de las instrucciones

## Implementación multiciclo



# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

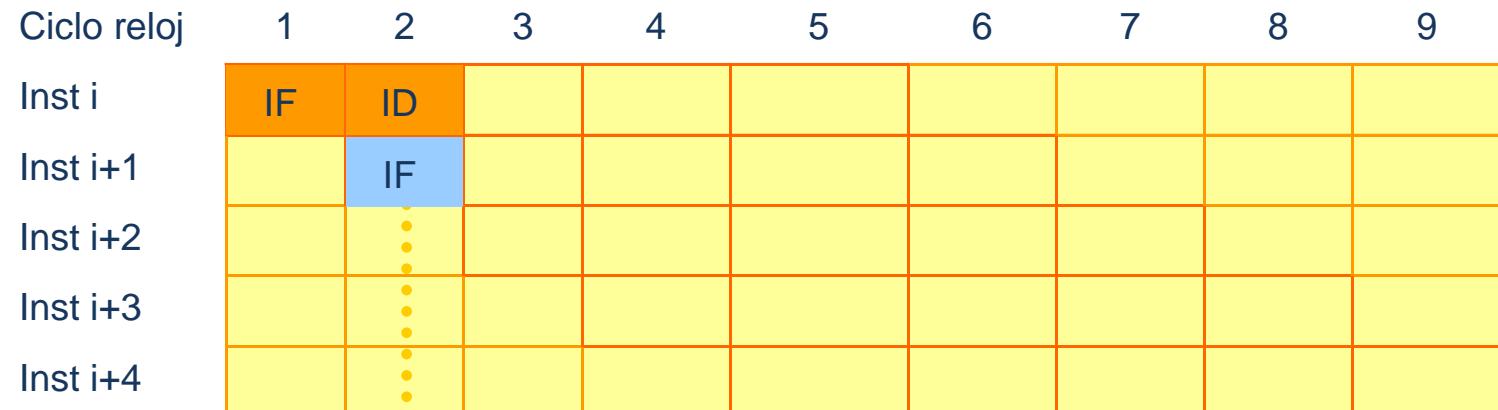
Cauces  
aritméticos

Optimización

Superescalares

- La segmentación consiste en solapar la ejecución de las instrucciones

## Implementación multiciclo



Segmentación

En el segundo ciclo se carga una nueva y se decodifica la anterior

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

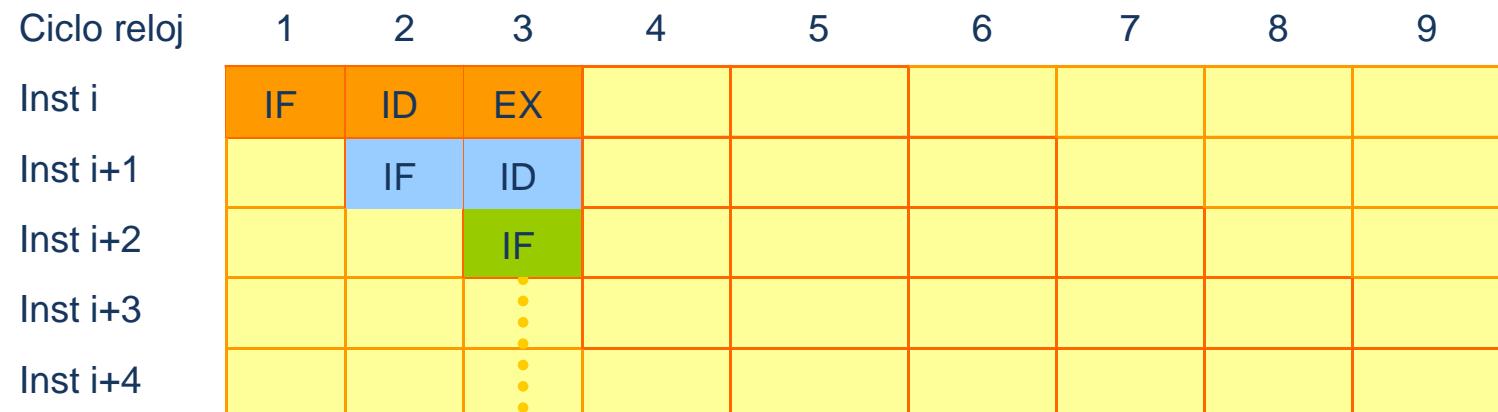
Optimización

Superescalares

Segmentación

- La segmentación consiste en solapar la ejecución de las instrucciones

## Implementación multiciclo



En el tercer ciclo se carga una nueva, se decodifica la anterior y se ejecuta la primera

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

- La segmentación consiste en solapar la ejecución de las instrucciones

## Implementación multiciclo

Ciclo reloj	1	2	3	4	5	6	7	8	9
Inst i	IF	ID	EX	MEM					
Inst i+1		IF	ID	EX					
Inst i+2			IF	ID					
Inst i+3				IF					
Inst i+4									

Segmentación

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- La segmentación consiste en solapar la ejecución de las instrucciones

## Implementación multiciclo



Cuando se llena el cauce se ejecuta una instrucción cada ciclo

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

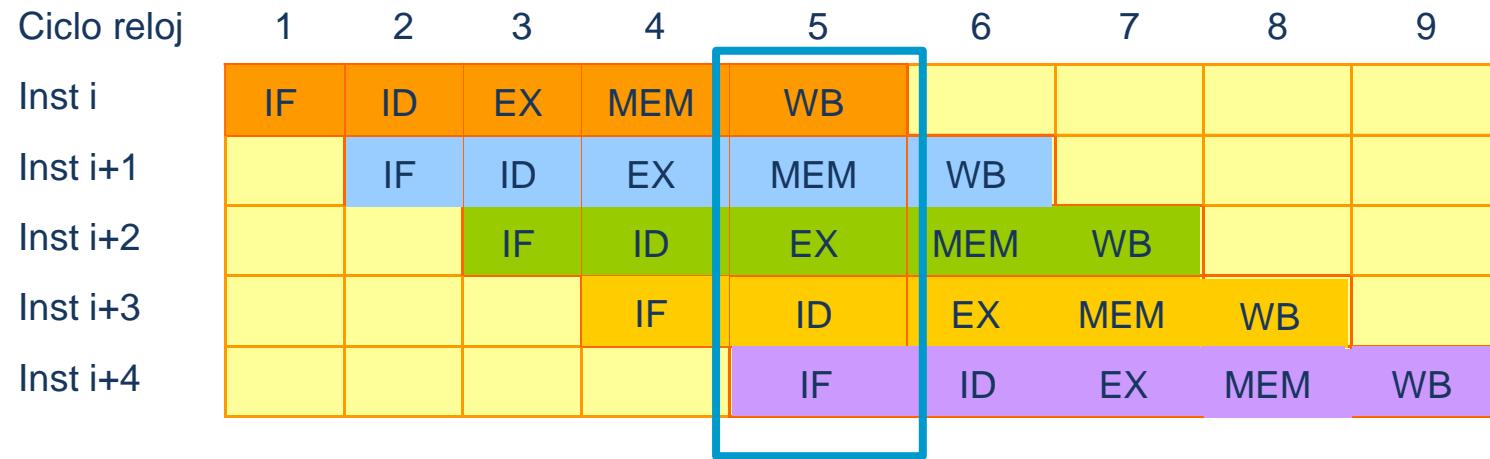
Optimización

Superescalares

Segmentación

- La segmentación consiste en solapar la ejecución de las instrucciones

## Implementación multiciclo



- Cada paso constituye una etapa de la segmentación.
- Cada ciclo, cinco instrucciones en ejecución
- La segmentación incrementa la **productividad** sin reducir el tiempo de ejecución de una instrucción individual.
- Los programas se ejecutan más rápido

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Ejemplo:

Considerar una máquina no segmentada con 5 pasos de ejecución cuyas duraciones son 50ns, 50ns, 60ns, 50ns, 50ns.

Suponer que debido al tiempo de preparación y sesgo de reloj, segmentar la máquina añade 5 ns de gasto a cada etapa de ejecución.

¿Cuánta velocidad se ganará con la segmentación en la frecuencia de ejecución de las instrucciones?

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

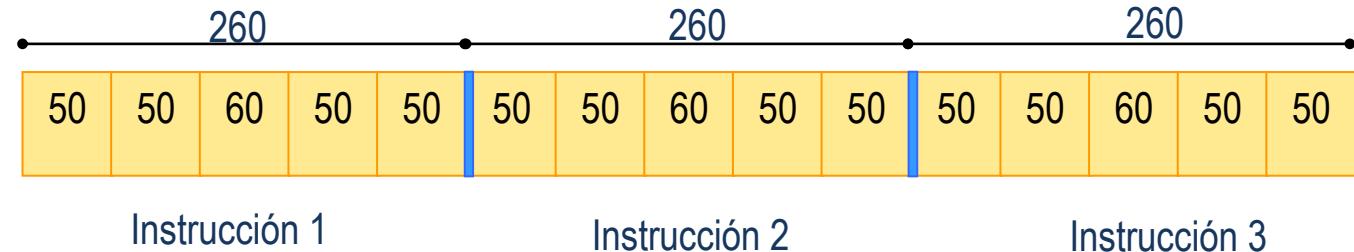
Cauces  
aritméticos

Optimización

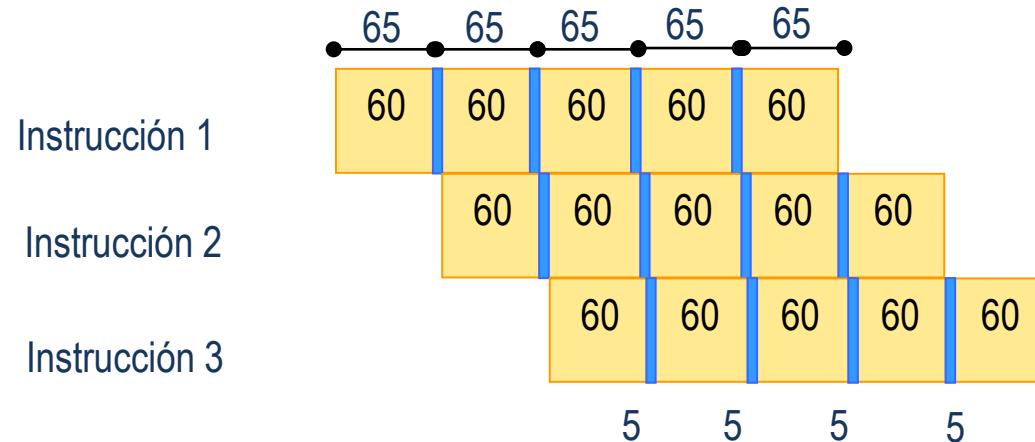
Superescalares

Segmentación

## Ejecución secuencial



## Ejecución segmentada



$$\text{Aceleración} = \frac{\text{Tiempo medio}_{\text{sin segmentación}}}{\text{Tiempo medio}_{\text{con segmentación}}} = \frac{260}{65} = 4$$

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Riesgos de segmentación

- ◆ Riesgos estructurales. Surgen de conflictos de los recursos cuando el hardware no puede soportar todas las combinaciones de instrucciones en ejecución (p.ej. acceso a memoria, banco registros, etc).
- ◆ Riesgos por dependencias de datos. Surgen cuando una instrucción depende de los resultados de una instrucción anterior.
- ◆ Riesgos de control. Surgen de la segmentación de los saltos y otras instrucciones que cambian el PC.

# Segmentación de instrucciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Riesgos de segmentación

- ◆ Impiden que se ejecute la siguiente instrucción
- ◆ Pueden hacer necesario detener el cauce
- ◆ Cuando una instrucción se detiene las instrucciones posteriores a esta también lo hacen
- ◆ Detención disminuye la ganancia con respecto a la ideal

# Segmentación de procesadores

Introducción

Segmentación  
del repertorio

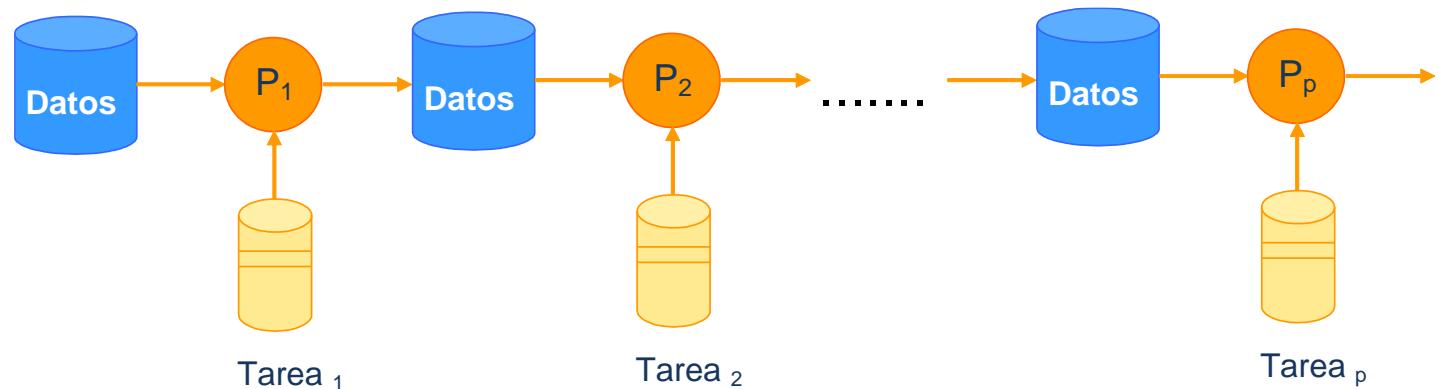
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- Las etapas del cauce están formadas por distintos procesadores que realizan, distintas operaciones sobre el flujo de datos
- Propio de la computación sistólica y utilizada en la implementación de algoritmos de alto coste.



# Segmentación de procesadores

Introducción

Segmentación  
del repertorio

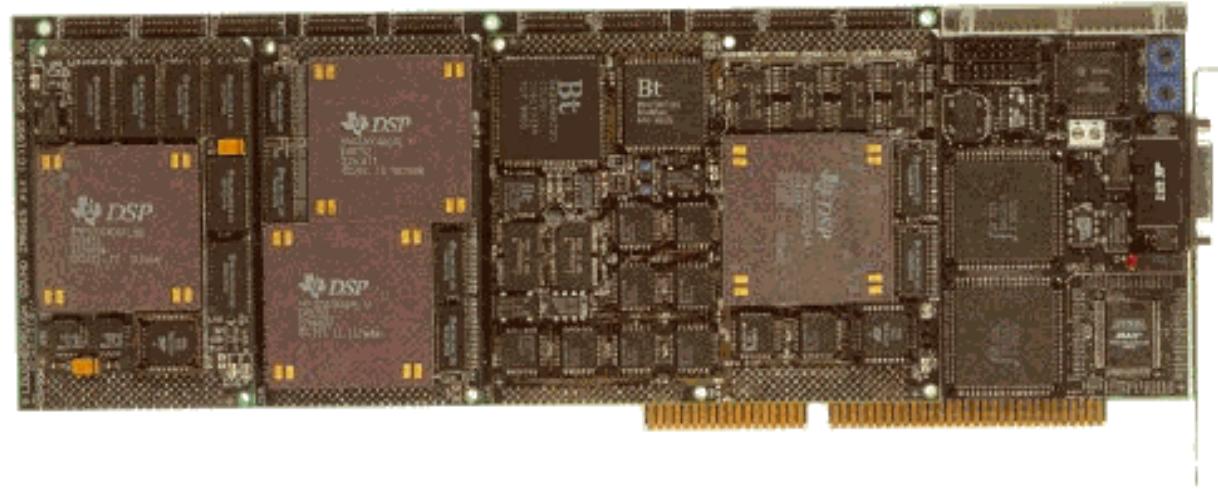
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ Ejemplo: Implementación de filtros de morfología matemática sobre DSPs (Digital Signal Processor). (Plataforma con 4 DSPs)



- ◆ La morfología matemática proporciona una herramienta para extraer componentes de la imagen tales como contornos, esqueletos y formas convexas.

# Análisis de prestaciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ **Unidad segmentada lineal síncrona**

### ◆ **Video**

◆ <https://drive.google.com/file/d/1ymeOjcmSUXmRUICVgHVMqp1oC5PwomNz/view?usp=sharing>

# Análisis de prestaciones

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

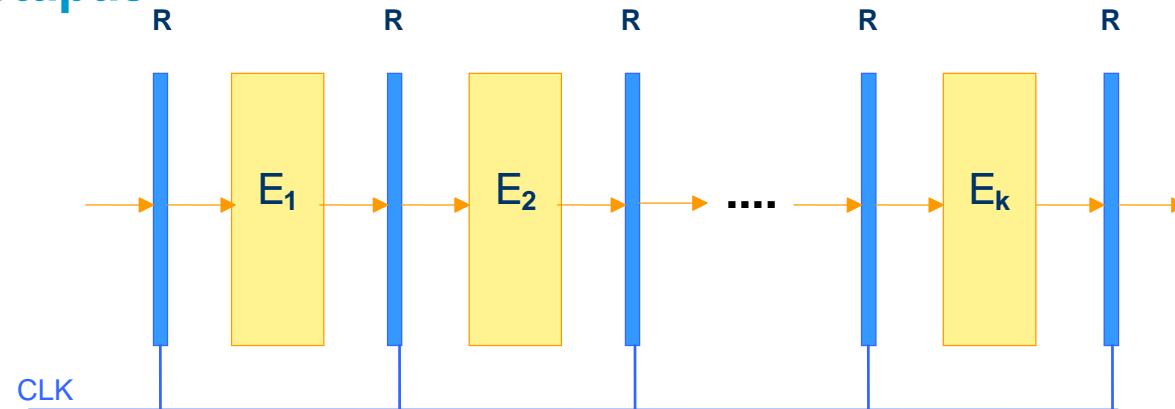
Optimización

Superescalares

Segmentación

## Unidad segmentada lineal síncrona

K etapas



**R** = Registros de almacenamiento intermedio

**E<sub>i</sub>** = Circuitos combinacionales para operaciones

**CLK** = Señal de reloj que controla el flujo de datos.

**t<sub>i</sub>** = Retardo temporal de cada etapa  $E_i$

**t<sub>r</sub>** = Es el retardo de cada registro R.

# Análisis de prestaciones

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

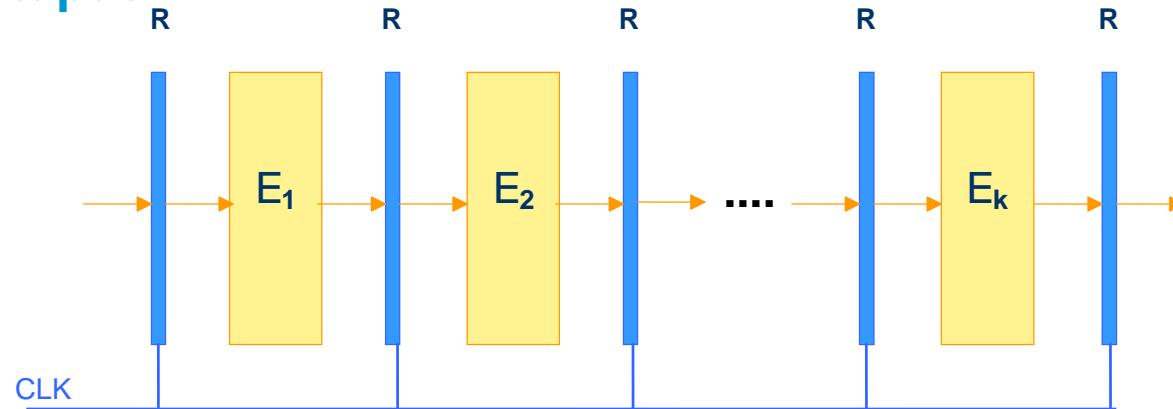
Optimización

Superescalares

Segmentación

## Unidad segmentada lineal síncrona

K etapas



### Periodo de reloj del cauce

$$CLK = \max\{t_i\}_{i=1}^k + t_r$$

- Sesgo de reloj (retardo del pulso). El instante de comienzo de las etapas debe incluir el retardo del pulso de reloj

$$CLK \geq \max\{t_i\}_{i=1}^k + t_r + s$$

# Análisis de prestaciones

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

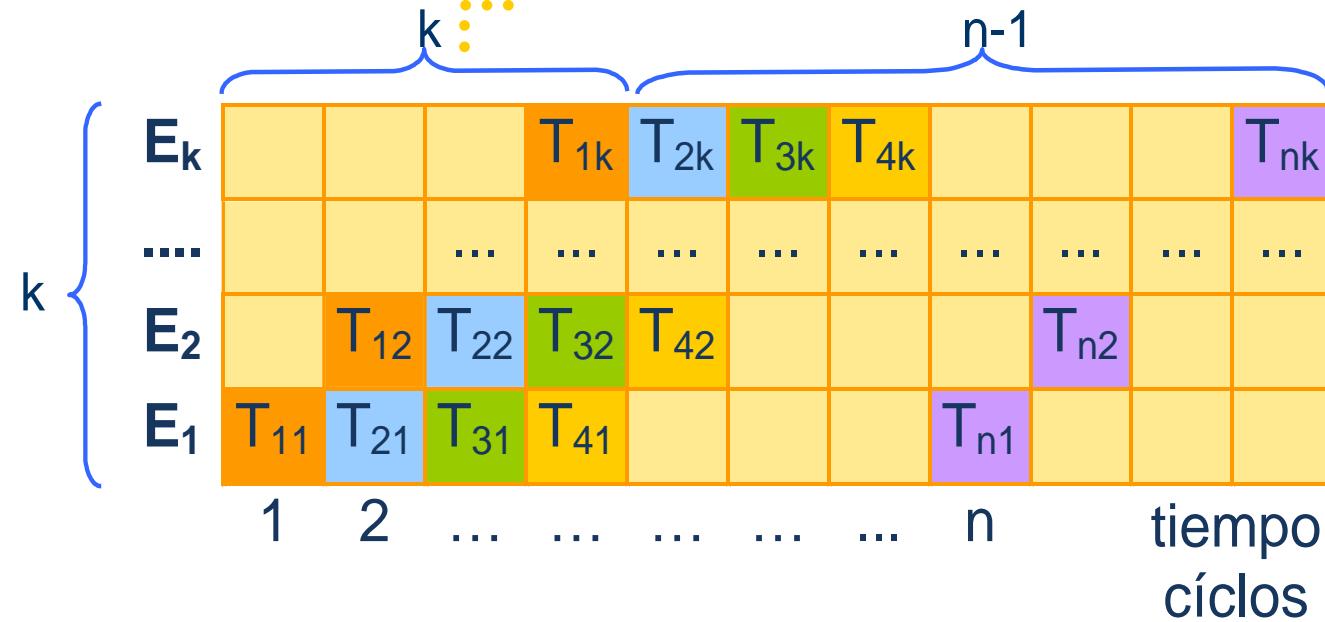
Superescalares

Segmentación

## Unidad segmentada lineal síncrona

### Tiempo para procesar n tareas

$$T_{SEG} = k \cdot CLK$$



# Análisis de prestaciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

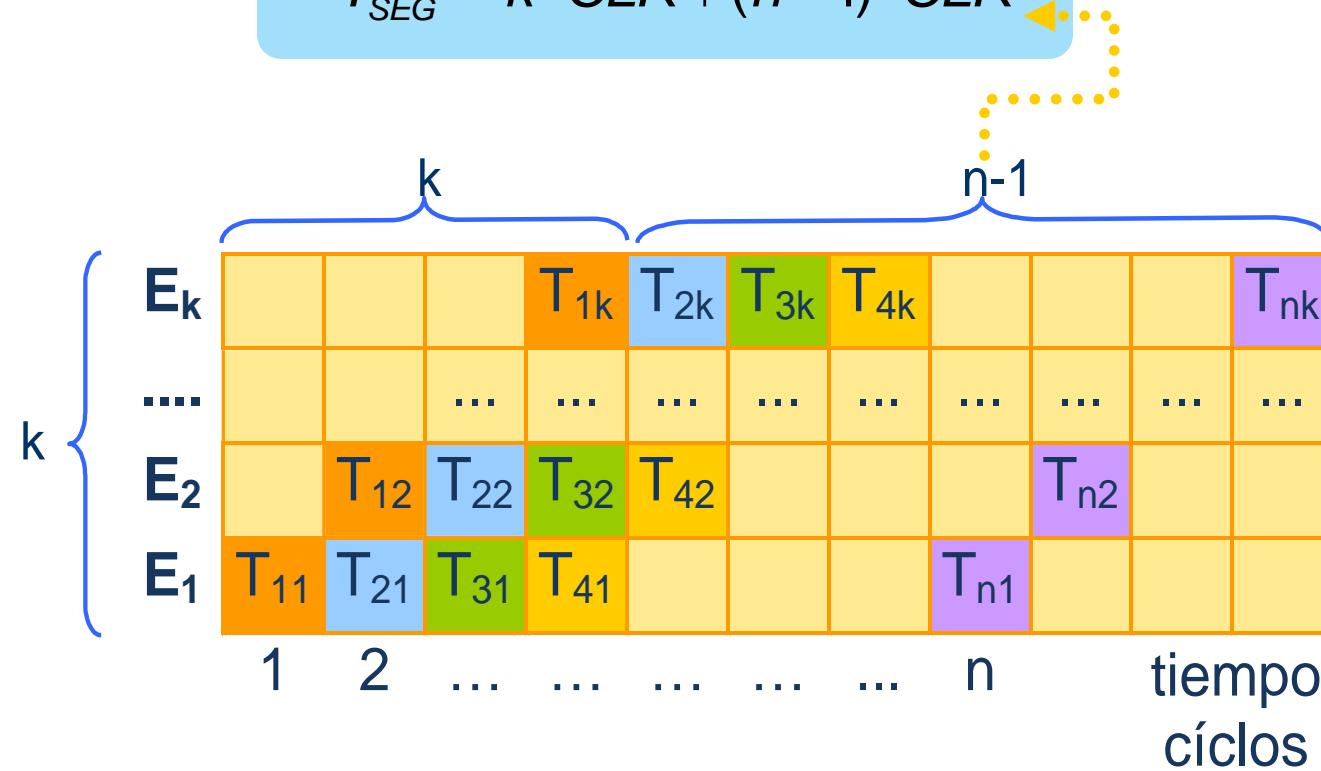
Superescalares

Segmentación

## Unidad segmentada lineal síncrona

### Tiempo para procesar n tareas

$$T_{SEG} = k \cdot CLK + (n-1) \cdot CLK$$



# Análisis de prestaciones

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

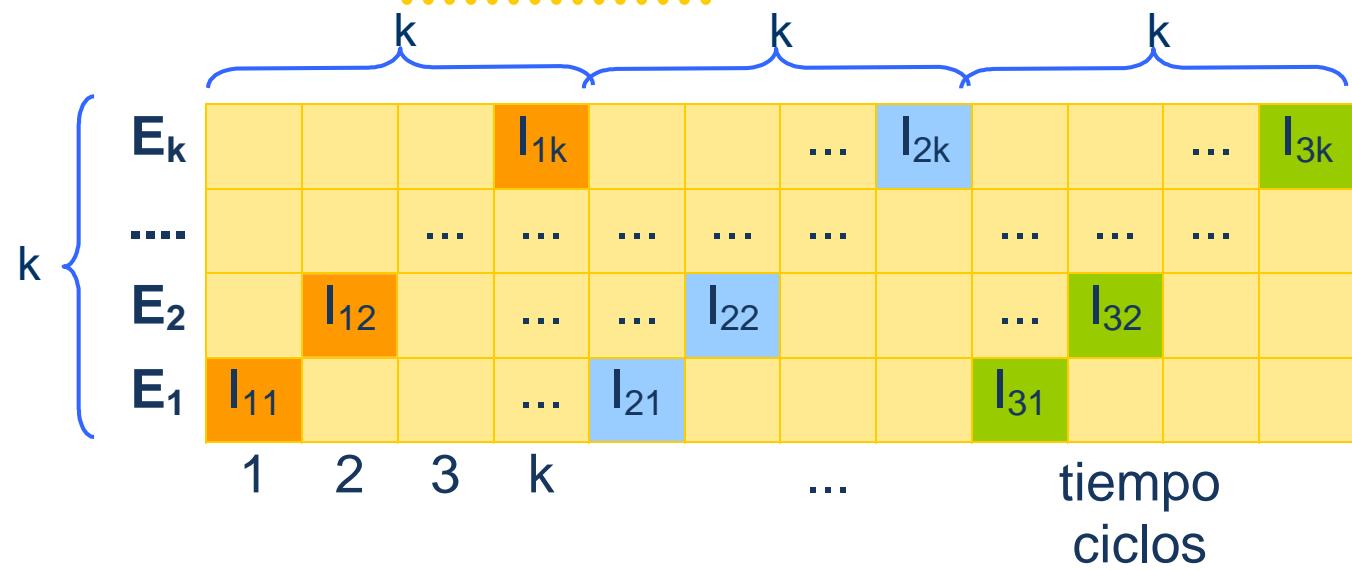
Superescalares

Segmentación

## Unidad segmentada lineal síncrona

Tiempo equivalente para un proceso no encauzado

$$T_{SEC} = K \cdot CLK \cdot n$$



# Ganancia de velocidad

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Incremento de velocidad de un cauce de k etapas respecto del proceso secuencial

$$G_k = \frac{T_{SEC}}{T_{SEG}} = \frac{n \cdot k \cdot CLK}{(k + n - 1)CLK} = \frac{n \cdot k}{k + n - 1}$$

- ➋ Cuando  $n \rightarrow \infty$

$$\lim_{n \rightarrow \infty} G_k = \lim_{n \rightarrow \infty} \frac{n \cdot k}{k + n - 1} = k$$

- ➌ En la práctica diversos condicionantes hacen que  $G_k < k$ 
  - ➍ Imposibilidad de la descomposición óptima de la tarea en k subtareas.
  - ➎ Dependencias entre datos e instrucciones.
  - ➏ Bifurcaciones en el programa.

# Eficiencia

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

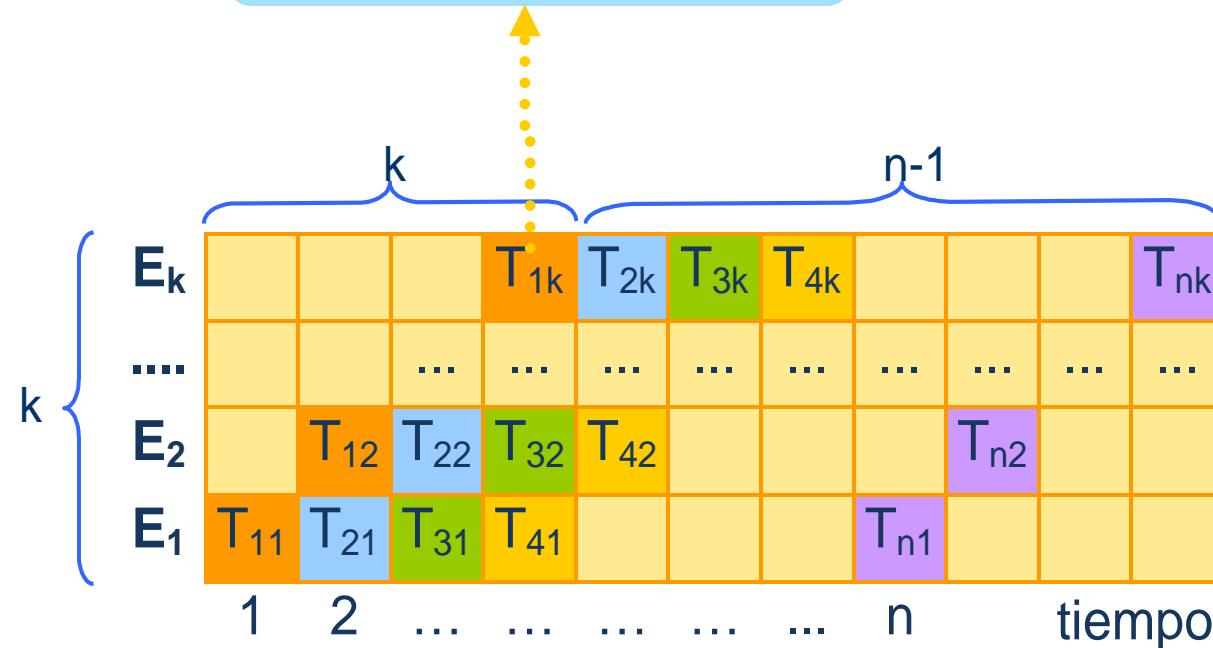
Optimización

Superescalares

Segmentación

- ◆ Nos ofrece idea del grado de utilización de los recursos
- ◆ Relación entre el número de tramos temporales ocupados y el número total de tramos en dicho periodo de tiempo T

$$T_{ocupado} = k \cdot n \cdot clk$$



# Eficiencia

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

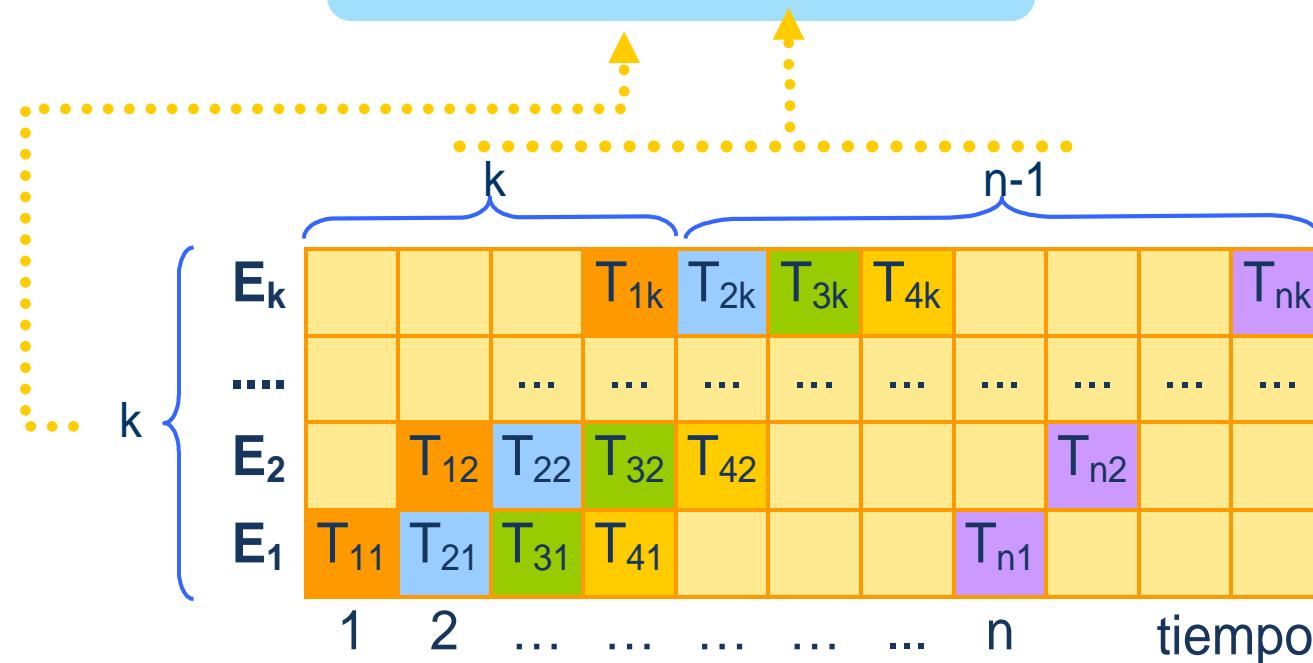
Optimización

Superescalares

Segmentación

- ◆ Nos ofrece idea del grado de utilización de los recursos
- ◆ Relación entre el número de tramos temporales ocupados y el número total de tramos en dicho periodo de tiempo T

$$T_{total} = k \cdot (k + n - 1) \cdot clk$$



# Eficiencia

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Nos ofrece idea del grado de utilización de los recursos
- ➋ Relación entre el número de tramos temporales ocupados y el número total de tramos en dicho periodo de tiempo T

$$E_k = \frac{k \cdot n \cdot CLK}{k(k+n-1)CLK} = \frac{n}{k+n-1} = \frac{G_k}{k}$$

- ➌ Cuando  $n \rightarrow \infty$

$$\lim_{n \rightarrow \infty} E_k = \lim_{n \rightarrow \infty} \frac{n}{k+n-1} = 1$$

- ➍ En la práctica problemas hacen que  $E_k \ll 1$

# Productividad

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ Número de datos o instrucciones que puede procesar por unidad de tiempo

$$P_k = \frac{n}{(k + n - 1)CLK} = \frac{E_k}{CLK}$$

- ◆ Cuando  $n \rightarrow \infty$

$$\lim_{n \rightarrow \infty} P_k = \lim_{n \rightarrow \infty} \frac{n}{(k + n - 1)CLK} = \frac{1}{CLK}$$

- ◆ La cota máxima coincide con la frecuencia de funcionamiento y corresponde a la aparición de un resultado cada periodo de reloj

## **4.2 Segmentación del repertorio de instrucciones**

---

**Tema 4. Segmentación**

**Arquitectura de los Computadores**

# Segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## 4.2. SEGMENTACIÓN DEL REPERTORIO DE INSTRUCCIONES

- ◆ 4.2.1. Segmentación básica en la arquitectura MIPS
- ◆ 4.2.2. Resolución de la problemática de segmentación para MIPS
- ◆ 4.2.3. Riesgos de la segmentación
- ◆ 4.2.4. Ruta de datos sementada
- ◆ 4.2.5. Ejemplos

# Arquitectura MIPS (DLX)

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Formato de las instrucciones

### ◆ Tipo R (Aritmético-lógicas)

6 (0..5)	5 (6..10)	5 (11..15)	5 (16..20)	11 (21..31)	
Cód ope	Rs1	Rs2	Rd	func	add R1,R2,R3

### ◆ Tipo I (acceso a memoria, saltos condicionales, inmediatas)

6 (0..5)	5 (6..10)	5 (11..15)	16 (16..31)	
Cód ope	Rs1	Rd	Inmediato	lw R1,100(R2) sw R2, 50(R4) add R1, R2,#100 beqz R1, ET

### ◆ Tipo J (salto incondicional)

6 (0..5)	26 (6..31)	
Cód ope	Desplazamiento añadido al PC	J Etiqueta

# Arquitectura MIPS (DLX)

Introducción

Segmentación  
del repertorio

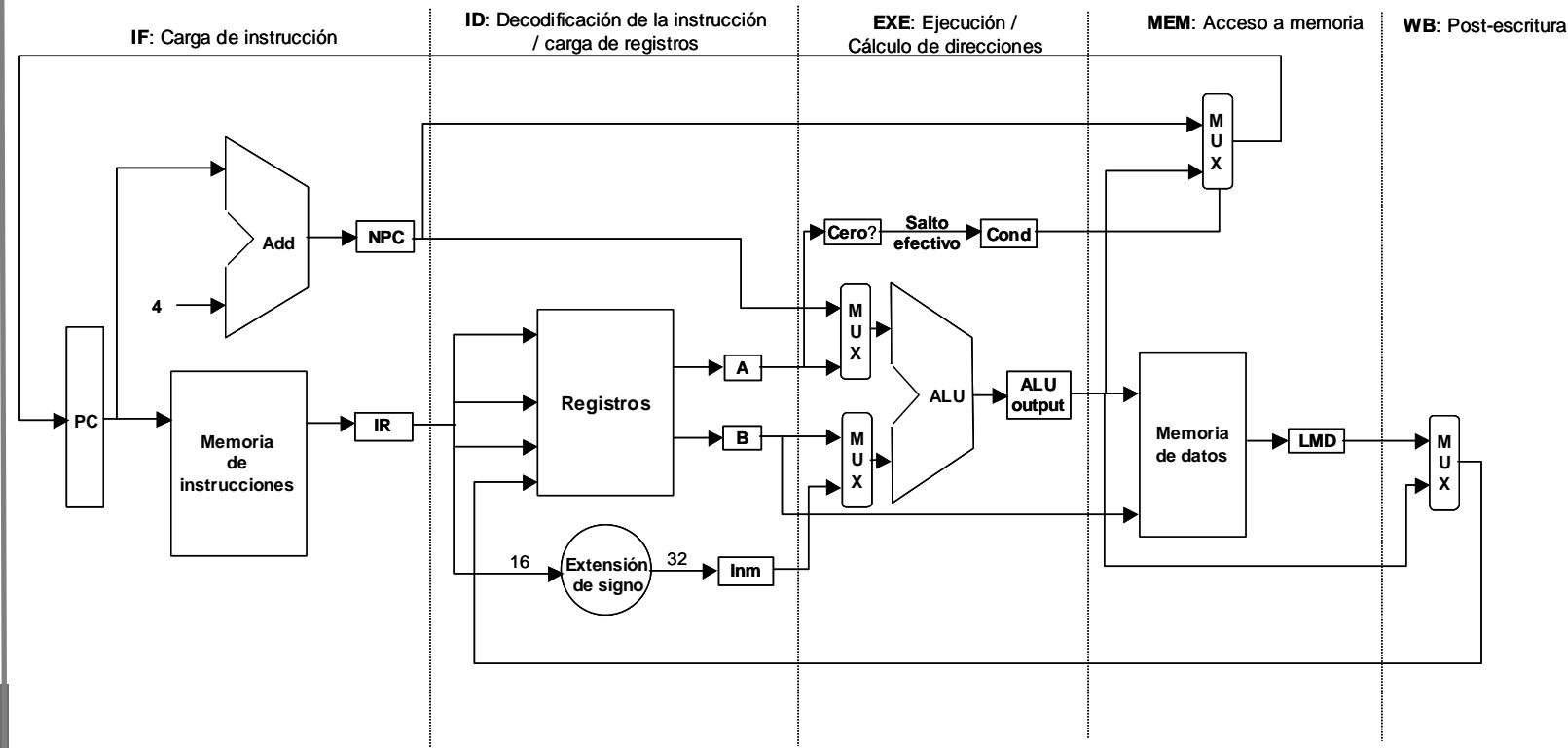
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Ruta de datos no segmentada



# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cuces  
aritméticos

Optimización

Superescalares

Segmentación

## ➊ 1. Ciclo de búsqueda de instrucción

$IR \leftarrow MEM [PC]$

$NPC \leftarrow PC + 4$

➊ **Operación:** Transfiere el PC y ubica la instrucción de memoria en el registro de instrucciones. Incrementa el PC en 4 para apuntar la siguiente instrucción de la secuencia.

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cauca-  
aritmético

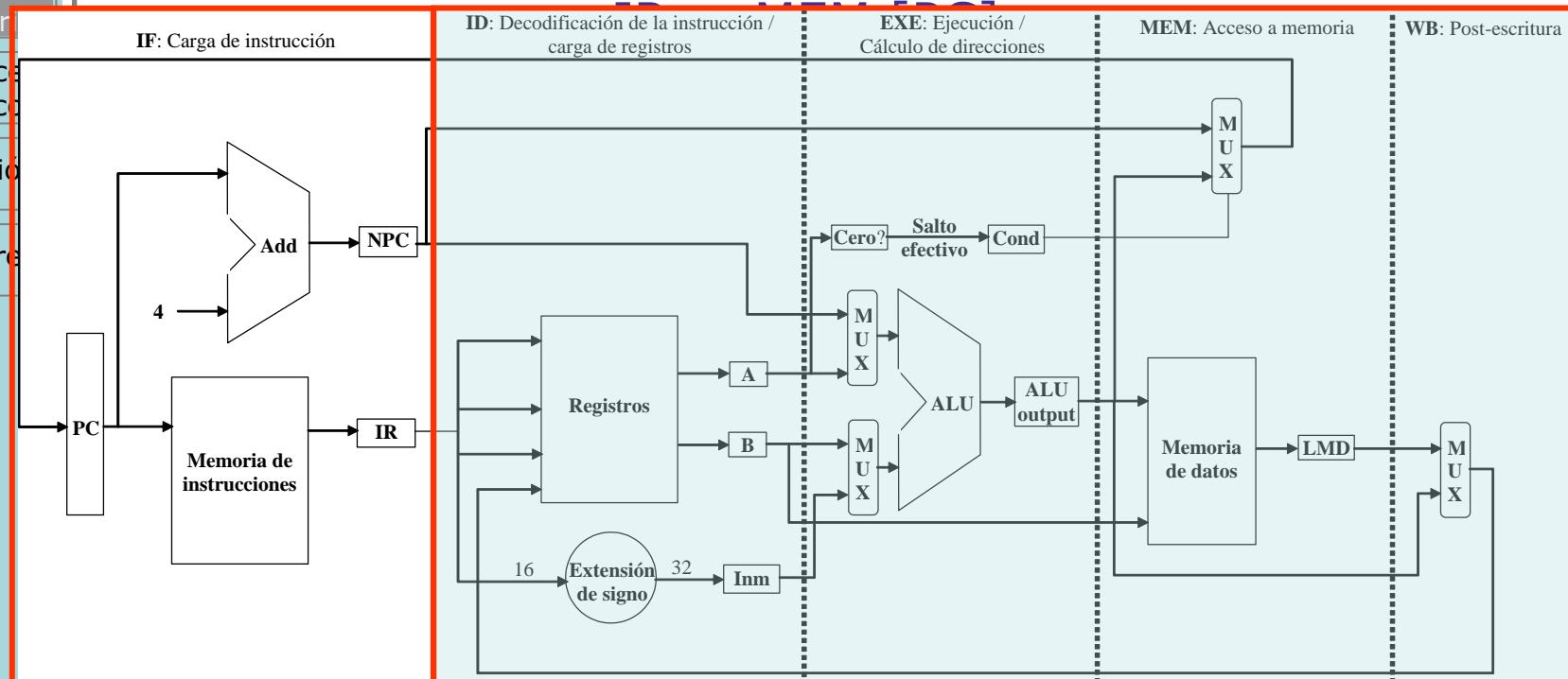
Optimizacio-

Superescalare

Segmentación



## 1. Ciclo de búsqueda de instrucción



$$\begin{aligned} \text{IR} &\leftarrow \text{MEM} [\text{PC}] \\ \text{NPC} &\leftarrow \text{PC} + 4 \end{aligned}$$

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ➊ 2. Ciclo de decodificación de la instrucción/carga de registros

$$A \leftarrow \text{Regs } [\text{IR}_{6..10}]$$

$$B \leftarrow \text{Regs } [\text{IR}_{11..15}]$$

$$\text{Inm} \leftarrow ((\text{IR}_{16})^{16} \# \# \text{IR}_{16..31})$$

- ➊ **Operación:** Decodifica la instrucción y accede al banco de registro para leer los registros. Las salidas de los dos registros de propósito general se cargan en dos registros temporales (A y B) para su uso posterior.
- ➋ Se extiende el signo a los 16 bits bajos del IR y se almacena el resultado en el registro temporal Inm

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ 2. Ciclo de decodificación de la instrucción/carga de registros

$$A \leftarrow \text{Regs } [\text{IR}_{6..10}]$$

$$B \leftarrow \text{Regs } [\text{IR}_{11..15}]$$

$$\text{Inm} \leftarrow ((\text{IR}_{16})^{16} \# \# \text{IR}_{16..31})$$

- ◆ **Decodificación de campo fijo:** Decodificación en paralelo con la lectura de los registros posible porque los campos ocupan posiciones fijas en el formato de instrucción DLX.
- ◆ Inmediato situado en posiciones idénticas en todos los formatos de instrucción DLX, el inmediato de signo extendido se calcula también en este ciclo.

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

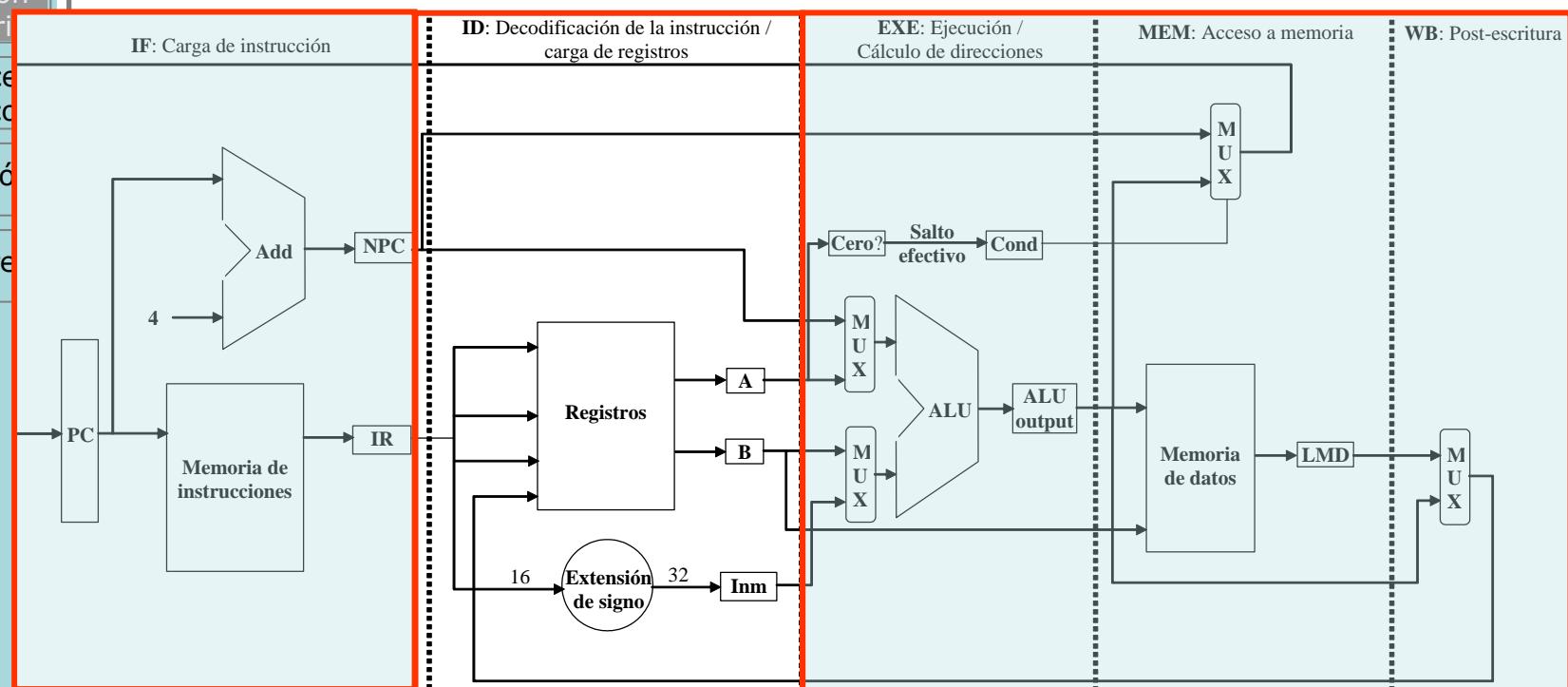
Cauce  
aritmético

Optimización

Superescalare

Segmentación

## 2. Ciclo de decodificación de la instrucción/carga de registros



$$\begin{aligned} A &\leftarrow \text{Regs [IR6..10]} \\ B &\leftarrow \text{Regs [IR11..15]} \\ \text{Inm} &\leftarrow ((\text{IR16})16 \# \text{IR16..31}) \end{aligned}$$

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## 3. Ciclo de ejecución / dirección efectiva

- La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.

## Referencia a memoria

- ALUoutput  $\leftarrow A + \text{Inm}$**
- Operación:** La ALU suma los operandos para formar la dirección efectiva.

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cauce  
aritmético

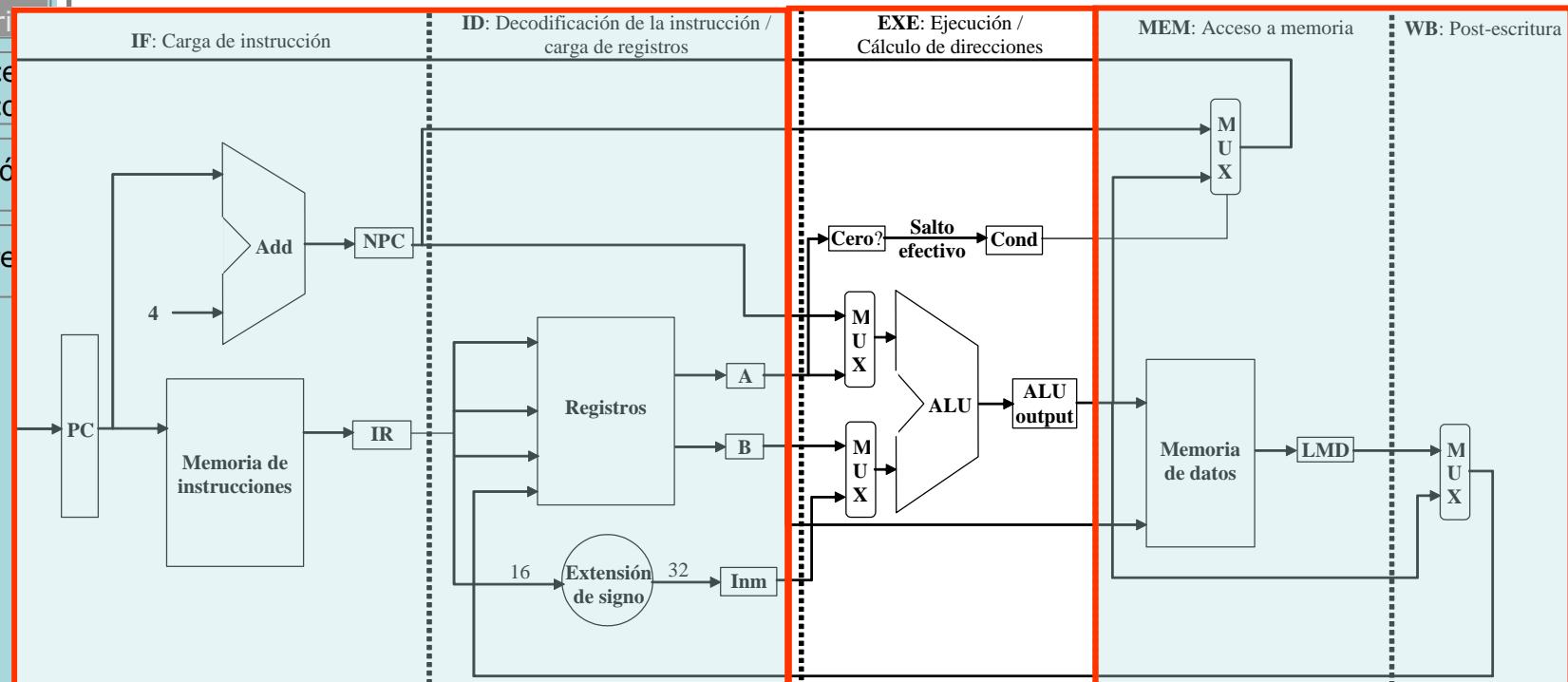
Optimización

Superescalare

Segmentación

## 3. Ciclo de ejecución / dirección efectiva

### Referencia a memoria.



$$\text{ALUoutput} \leftarrow A + \text{Inm}$$

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## 3. Ciclo de ejecución / dirección efectiva

- La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.

### Instrucción ALU registro-registro

- ALUoutput ← A func B**
- Operación:** La ALU realiza la operación especificada por el código de operación sobre el valor de A y sobre el valor de B. El resultado se coloca en el registro temporal ALUoutput.

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cauce  
aritmético

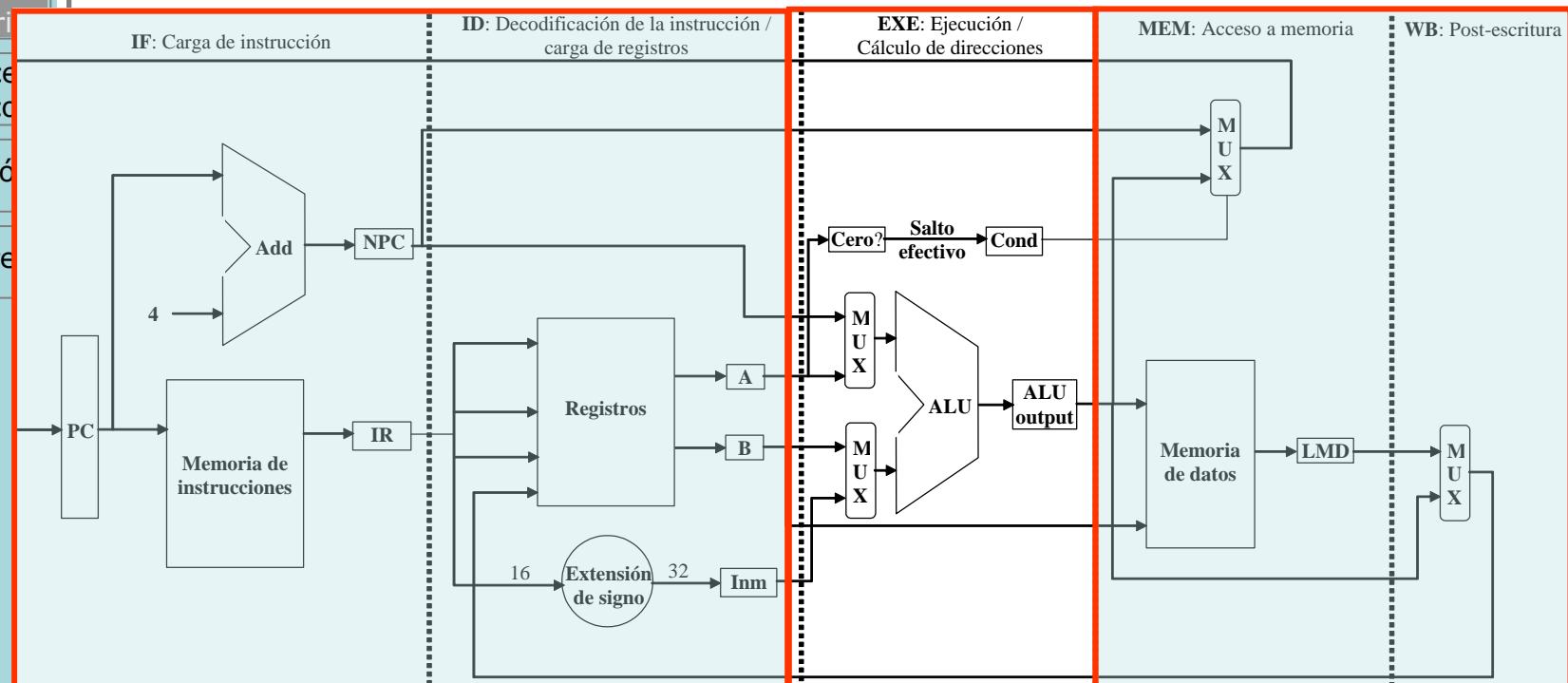
Optimización

Superescalare

Segmentación

## 3. Ciclo de ejecución / dirección efectiva

### Instrucción ALU registro-registro



ALUoutput  $\leftarrow$  A func B

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ➊ 3. Ciclo de ejecución / dirección efectiva

- ➊ La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.

## ➋ Instrucción ALU registro-inmediato

- ➋ **ALUoutput**  $\leftarrow$  A op Inm
- ➋ **Operación:** La ALU realiza la operación especificada por el código de operación sobre el valor de A y sobre el valor del registro Inm. El resultado se coloca en el registro temporal ALUoutput.

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cauce  
aritmético

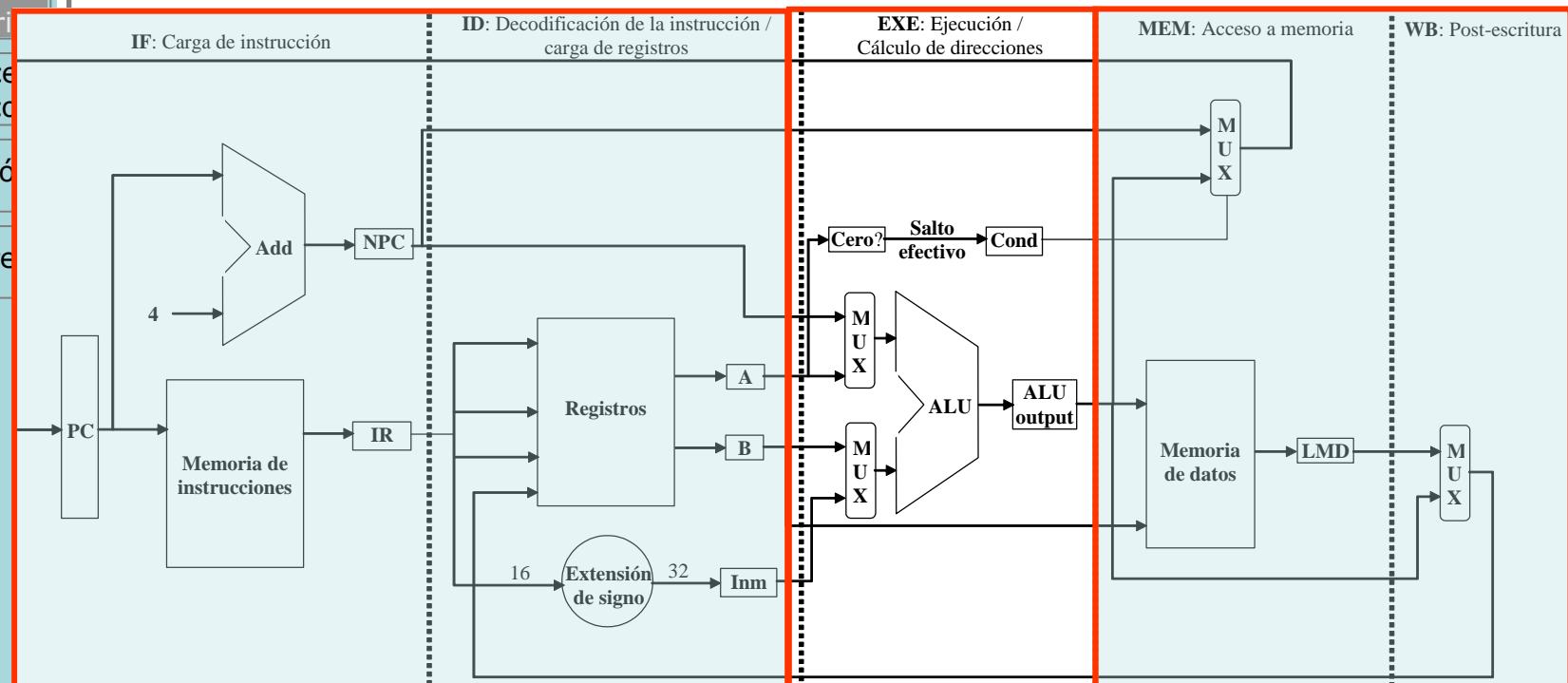
Optimización

Superescalare

Segmentación

## 3. Ciclo de ejecución / dirección efectiva

### Instrucción ALU registro-inmediato



$$\text{ALUoutput} \leftarrow A \text{ op } B$$

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ➊ 3. Ciclo de ejecución / dirección efectiva

- ➊ La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.

## ➋ Salto / bifurcación

- ➌  $\text{ALUoutput} \leftarrow \text{NPC} + \text{Imm}$

- ➌  $\text{cond} \leftarrow (\text{A op 0})$

- ➌ Operación:

- ➌ La ALU suma el NPC al valor inmediato de signo extendido para calcular la dirección destino del salto.
- ➌ El registro A se examina para decidir si se realiza el salto o no. Op es el operador relacional determinado por el código de operación, (Eje: op es = para la instrucción BEQZ)

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cauce  
aritmético

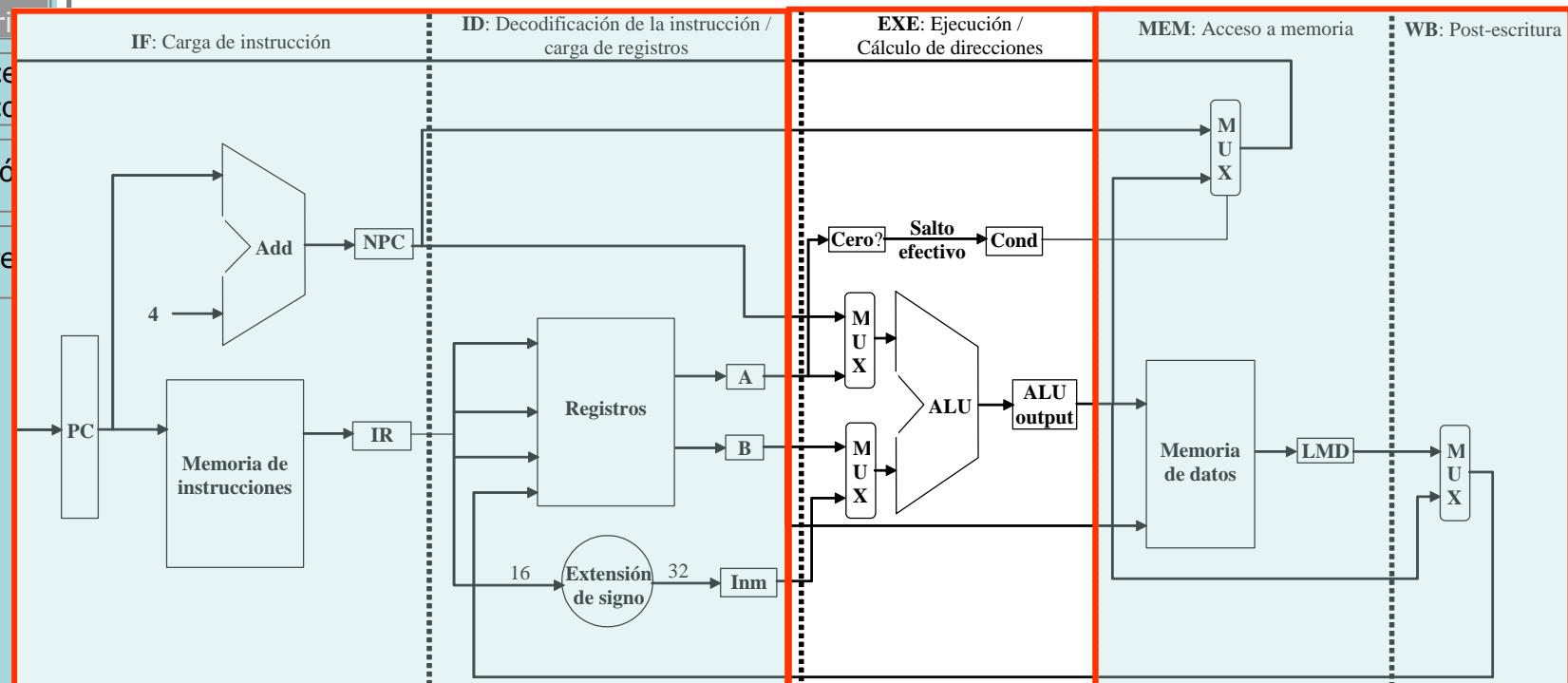
Optimización

Superescalare

Segmentación

## 3. Ciclo de ejecución / dirección efectiva

### Salto/ bifurcación



$$\begin{aligned} \text{ALUoutput} &\leftarrow \text{NPC} + \text{Inm} \\ \text{cond} &\leftarrow (\text{A op } 0) \end{aligned}$$

# Implementación simple de DLX

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ➃ 4. Paso de acceso a memoria / completar salto:

- ➃ Las únicas instrucciones DLX activas en este paso son cargas, almacenamientos y saltos.

## ➃ Referencia a memoria

- ➃ **LMD  $\leftarrow$  MEM [ALUoutput]** o
- ➃ **MEM [ALUoutput]  $\leftarrow$  B**
- ➃ **Operación:** Accede a memoria si es necesario.
- ➃ **Si la instrucción es una carga,** se lee el dato de memoria y se carga en LMD (load memory data);
- ➃ **Si es un almacenamiento,** escribe el dato del registro B en memoria.
- ➃ En cualquier caso la dirección utilizada es la calculada durante el paso anterior y almacenada en ALUoutput.

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cádeas  
aritméticas

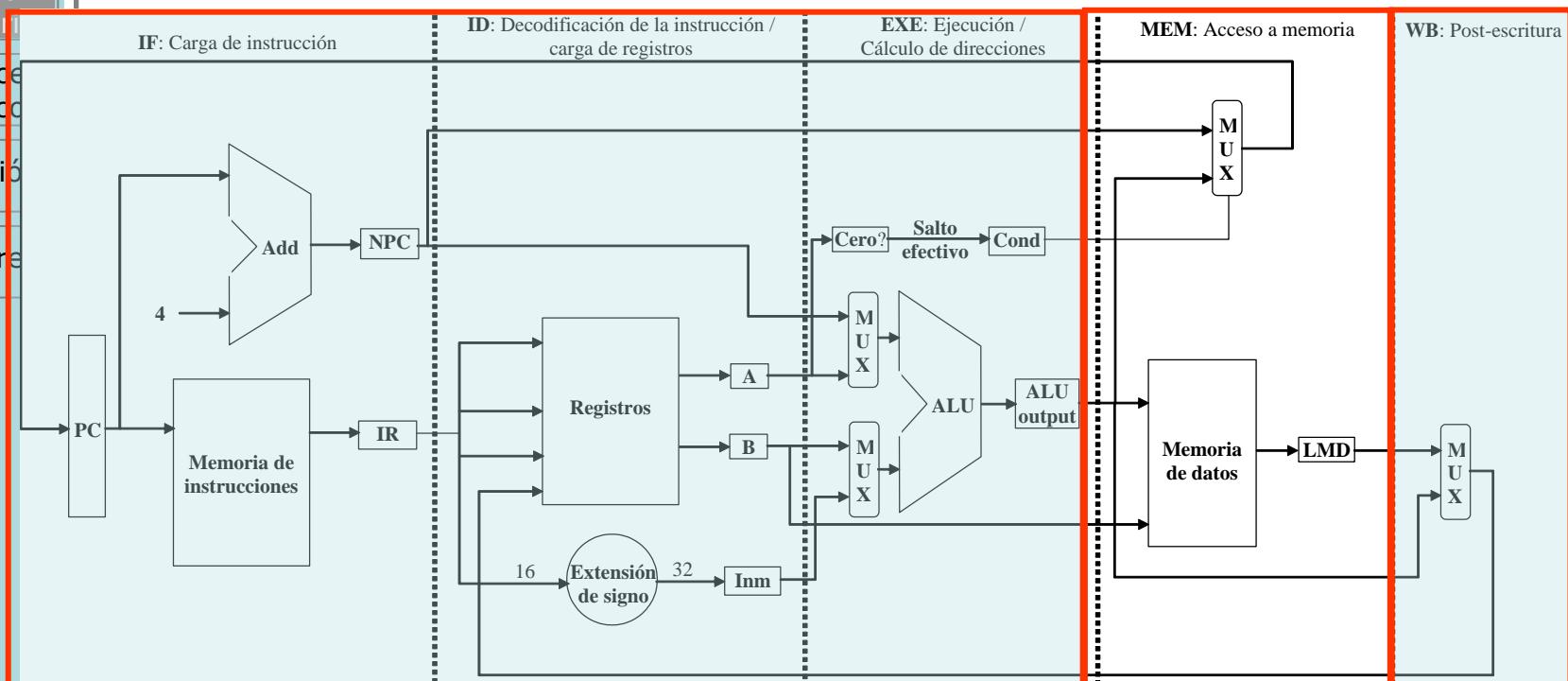
Optimización

Superescalares

Segmentación

## 4. Paso de acceso a memoria / completar salto:

### Referencia a memoria



$PC \leftarrow NPC$   
 $LMD \leftarrow MEM [ALUoutput] \text{ o }$   
 $MEM [ALUoutput] \leftarrow B$

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Cáuces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ 4. Paso de acceso a memoria / completar salto:

- ◆ Las únicas instrucciones DLX activas en este paso son cargas, almacenamientos y saltos.

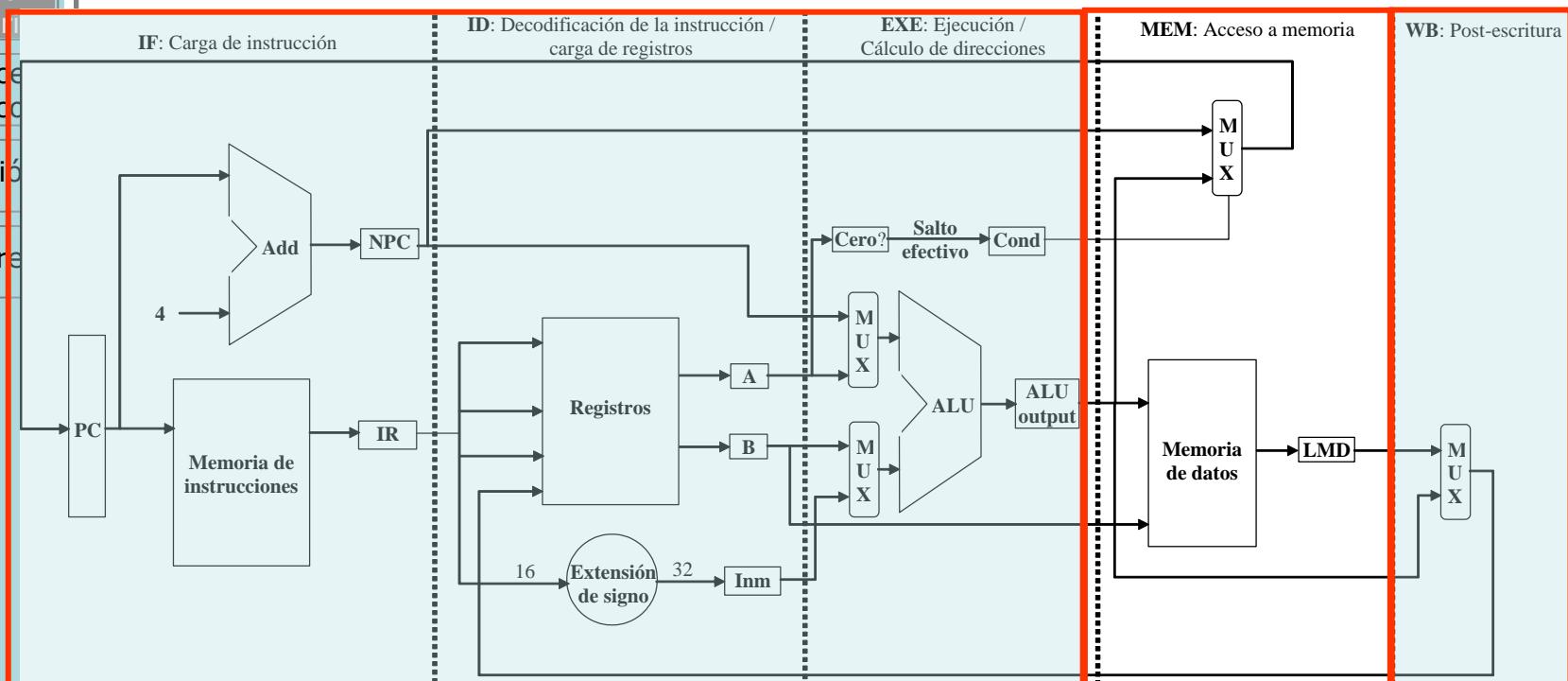
### ◆ Salto

- ◆ **If (cond) then PC  $\leftarrow$  ALUoutput else PC  $\leftarrow$  NPC**
- ◆ **Operación:** Si la instrucción salta, el PC es sustituido por la dirección destino del salto del registro ALUoutput, en caso contrario, se reemplaza con el contador de programa incrementado en el registro NPC.

# Implementación simple del MIPS

## 4. Paso de acceso a memoria / completar salto:

### Salto



Segmentación

If (cond)  $PC \leftarrow ALUoutput$   
else  $PC \leftarrow NPC$

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ 5. Paso de postescritura (write-back)

### ◆ Instrucciones ALU registro-registro:

- ◆ Regs [IR<sub>16..20</sub>] ← ALUoutput

### ◆ Instrucciones ALU registro-inmediato:

- ◆ Regs [IR<sub>11..15</sub>] ← ALUoutput

### ◆ Instrucciones load

- ◆ Regs [IR<sub>11..15</sub>] ← LMD

◆ Operación: Escribir el resultado en el banco de registros tanto si proviene del sistema de memoria (en LMD) como de la ALU (en ALUoutput).

◆ El campo del registro destino puede estar en dos posiciones dependiendo del código de operación.

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

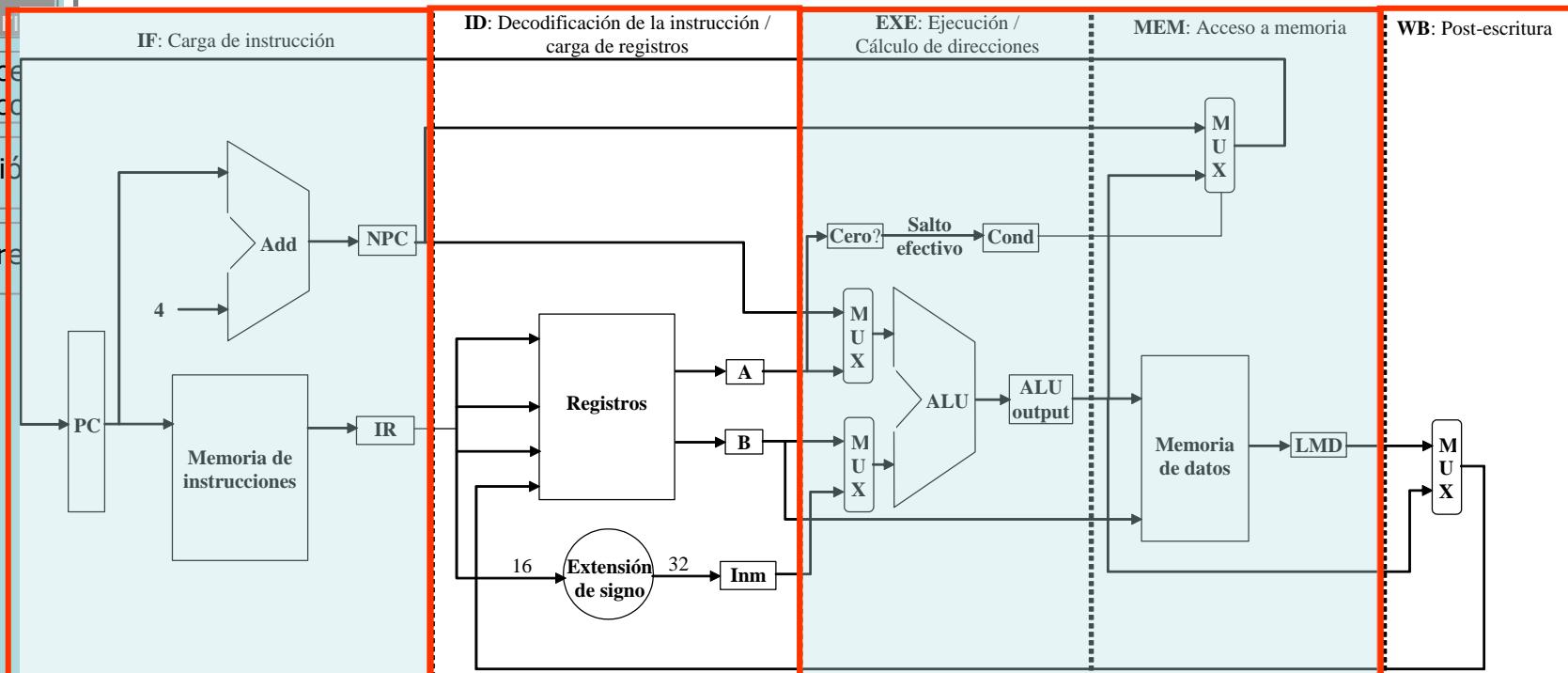
Cádeas  
aritméticas

Optimización

Superescalares

Segmentación

## 5. Paso de postescritura (write-back)



### Instrucciones ALU R-R:

$$\text{Regs } [\text{IR}_{16..20}] \leftarrow \text{ALUoutput}$$

### Instrucciones ALU R-I:

$$\text{Regs } [\text{IR}_{11..15}] \leftarrow \text{ALUoutput}$$

### Instrucciones load

$$\text{Regs } [\text{IR}_{11..15}] \leftarrow \text{LMD}$$

# Arquitectura MIPS (DLX)

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Ejecución de las instrucciones (resumen)

Etapa	Aritmético-lógicas	Carga/almacenamiento	Salto
IF		$IR \leftarrow MEM[PC]$ $NPC \leftarrow PC + 4$	
ID		$A \leftarrow Regs[IR_{6..10}]$ $B \leftarrow Regs[IR_{11..15}]$ $Inm \leftarrow ((IR_{16})^{16} \# \# IR_{16..31})$	
EX	$ALUoutput \leftarrow A \text{ op } B$ O bien $ALUoutput \leftarrow A \text{ op } Inm$	$ALUoutput \leftarrow A + Inm$	$ALUoutput \leftarrow NPC + Inm$ $cond \leftarrow (A \text{ op } 0)$
MEM	$PC \leftarrow NPC$	$PC \leftarrow NPC$  $LMD \leftarrow MEM[ALUoutput]$ O bien $MEM[ALUoutput] \leftarrow B$	If (cond) $PC \leftarrow ALUoutput$ else $PC \leftarrow NPC$
WB	$Regs[IR_{16..20}] \leftarrow ALUoutput$ O bien $Regs[IR_{11..15}] \leftarrow ALUoutput$	$Regs[IR_{11..15}] \leftarrow LMD$	

# Implementación simple del MIPS

Introducción

Segmentación  
del repertorio

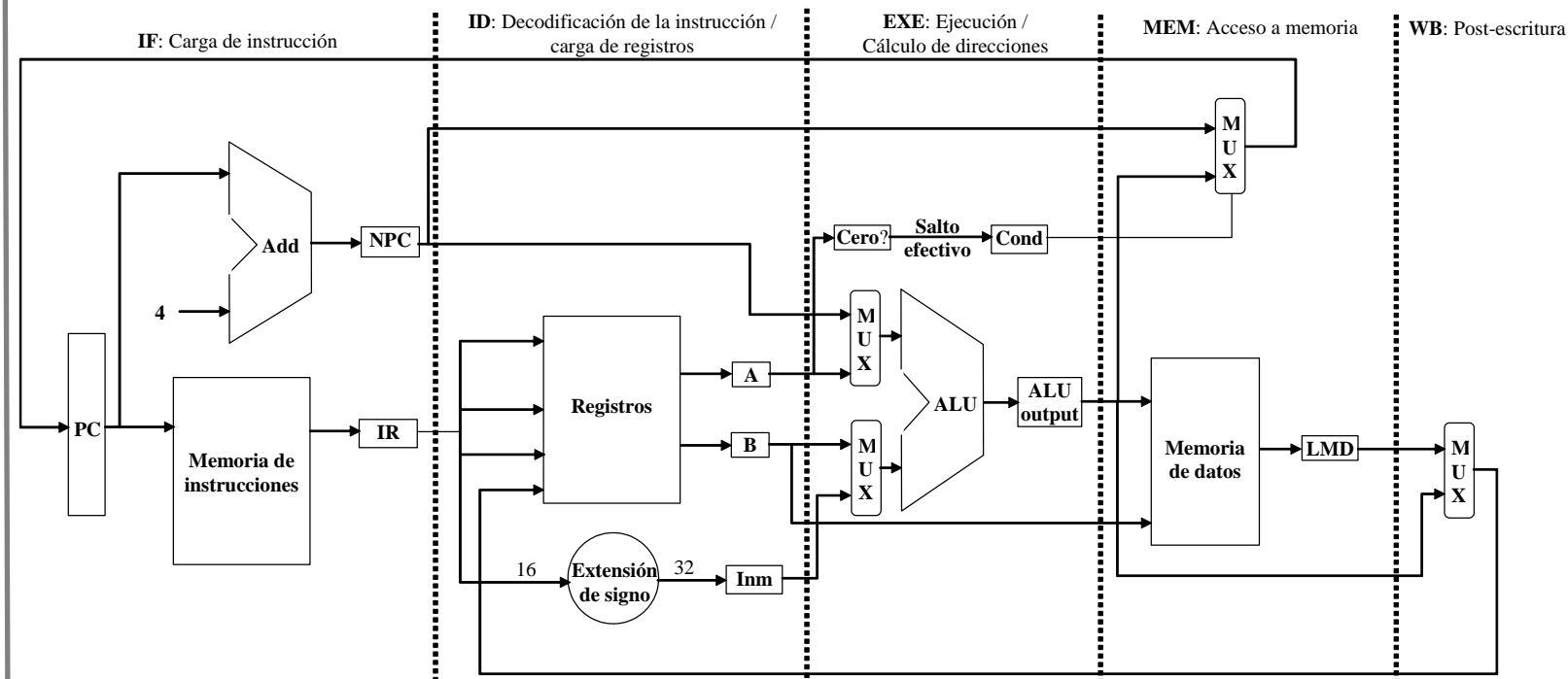
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Evolución de una instrucción en el camino de datos



- ◆ Observamos el PC y Registros en el ciclo de reloj en el que son leídos
- ◆ En ambos casos las escrituras en etapas posteriores se indican mediante multiplexores en las salidas (en acceso a memoria y post-escritura) que retornan resultados al PC o a los registros

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ Podemos segmentar el camino de datos sin más que iniciar una nueva instrucción en cada ciclo de reloj
- ◆ Cada ciclo de reloj se convierte en una etapa del cauce.

Número de instrucción	1	2	3	4	Ciclo 5	Reloj 6	7	8	9
Inst i	IF	ID	EX	MEM	WB				
Inst i+1		IF	ID	EX	MEM	WB			
Inst i+2			IF	ID	EX	MEM	WB		
Inst i+3				IF	ID	EX	MEM	WB	
Inst i+4					IF	ID	EX	MEM	WB

- ◆ Durante un ciclo de reloj el hardware ejecuta alguna parte de cinco instrucciones diferentes.
- ◆ Si comienza una instrucción nueva cada ciclo de reloj, el rendimiento mejora hasta cinco veces con respecto a una máquina no encauzada

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Podemos segmentar el camino de datos sin más que iniciar una nueva instrucción en cada ciclo de reloj
- ➋ Cada ciclo de reloj se convierte en una etapa del cauce

Número de instrucción	1	2	3	4	Ciclo 5	Reloj 6	7	8	9
Inst i	IF	ID	EX	MEM	WB				
Inst i+1		IF	ID	EX	MEM	WB			
Inst i+2			IF	ID	EX	MEM	WB		
Inst i+3				IF	ID	EX	MEM	WB	
Inst i+4					IF	ID	EX	MEM	WB

- ➌ Iniciar una instrucción cada ciclo introduce **problemas**
- ➍ Tenemos que determinar que ocurre en cada ciclo de reloj de la máquina y asegurarnos que no estamos intentando realizar dos operaciones diferentes con el mismo recurso
- ➎ Ejemplo: una simple ALU calculando una dirección efectiva y una operación de resta al mismo tiempo

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

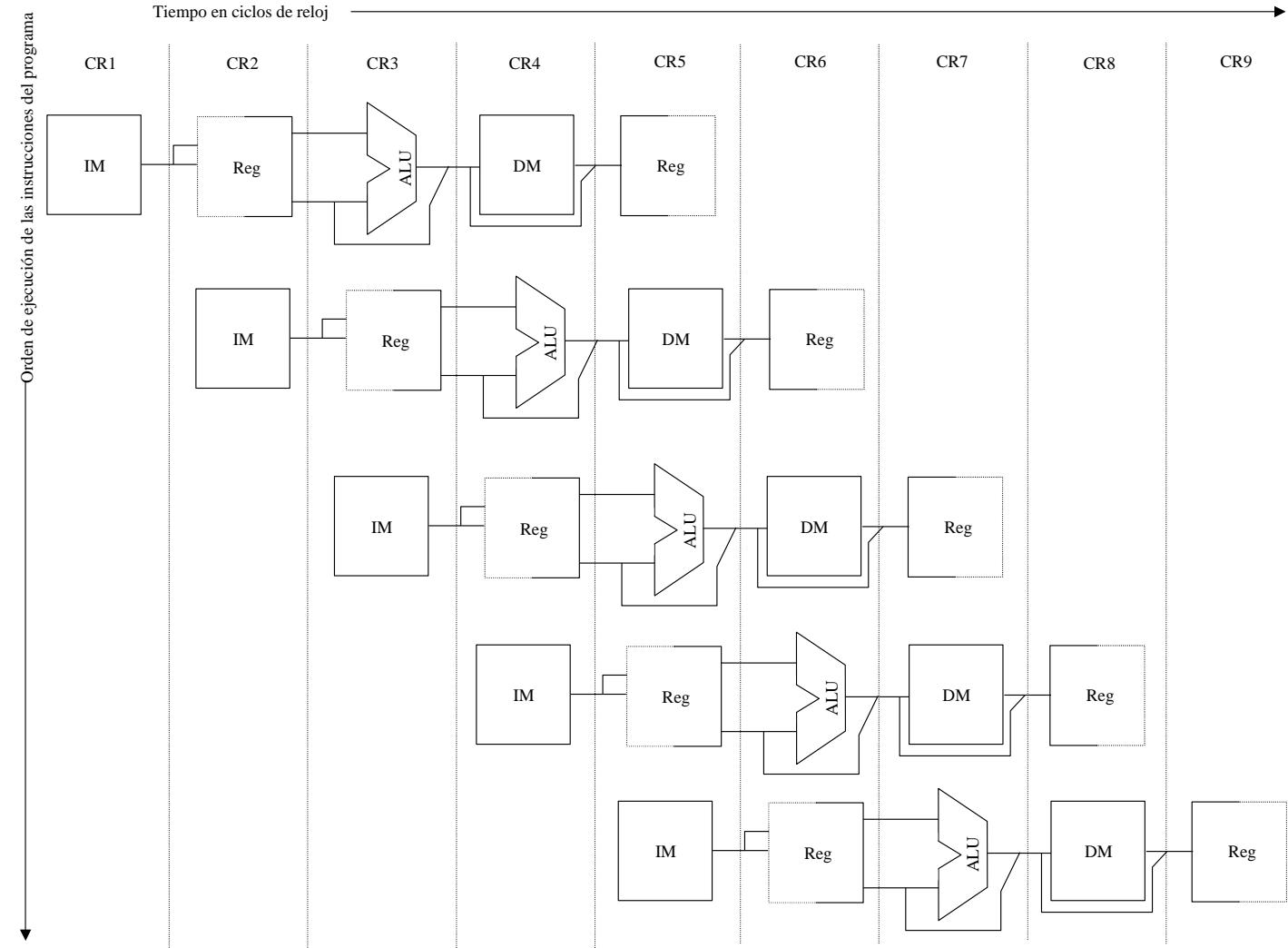
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Evaluación de conflictos entre recursos para el cauce DLX



# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

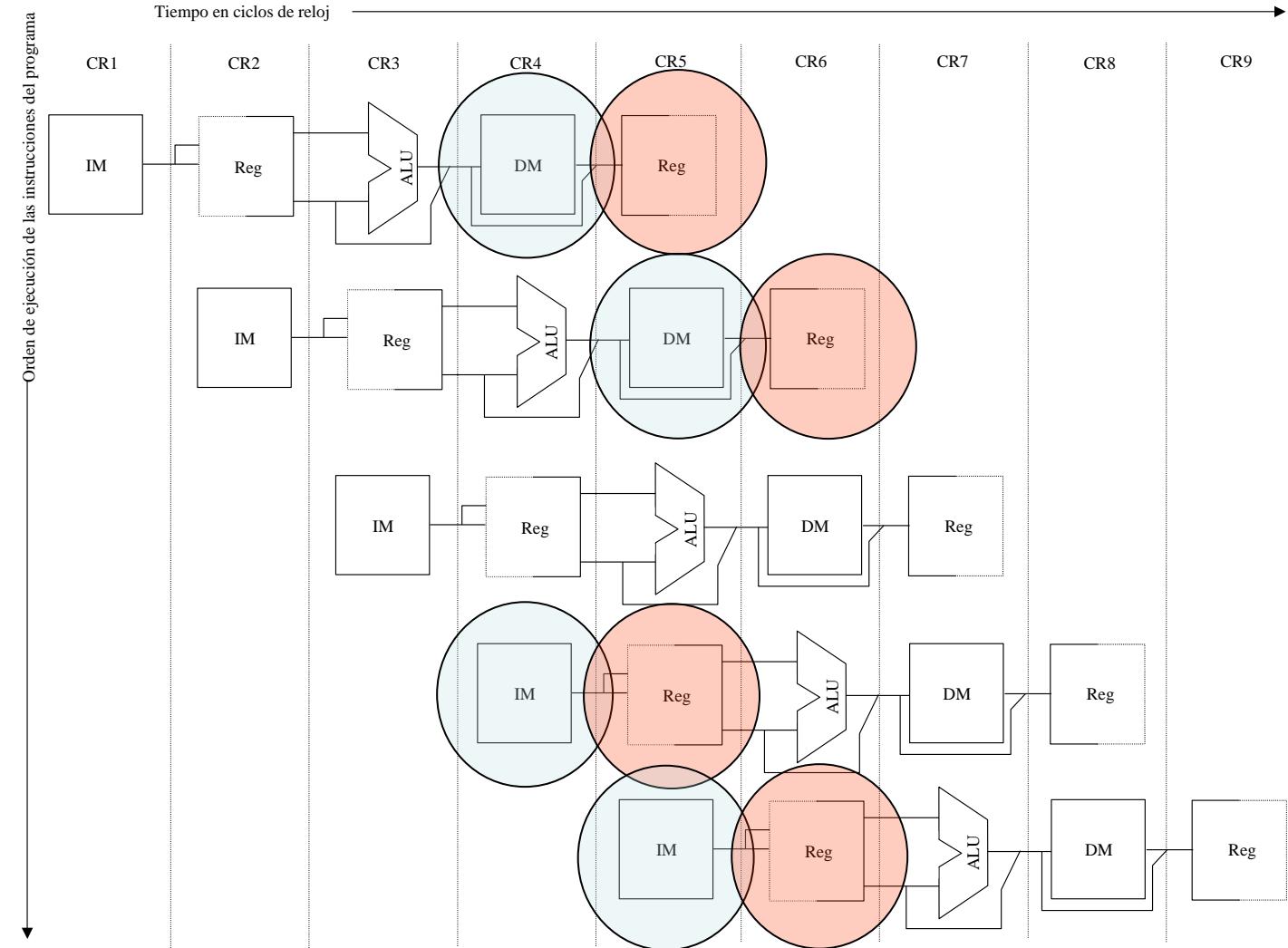
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Evaluación de conflictos entre recursos para el cauce DLX



# Segmentación básica del MIPS

Introducción

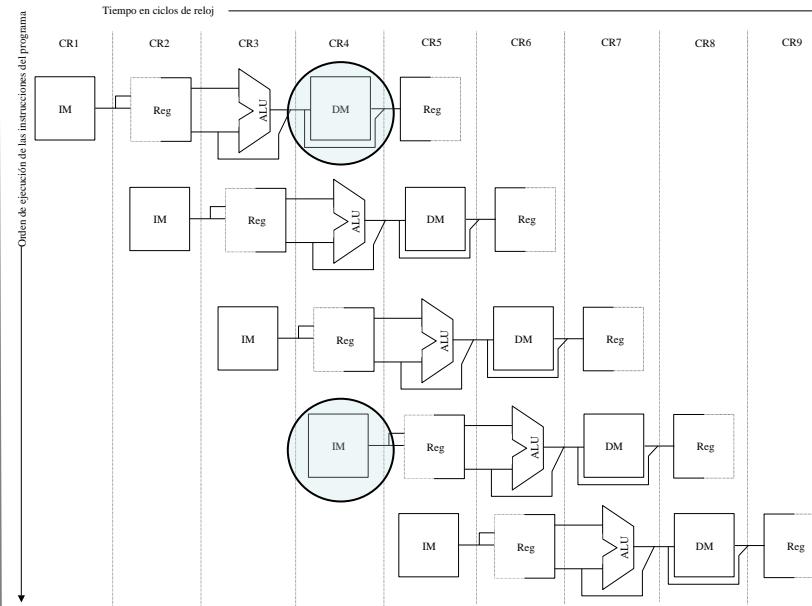
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



## La memoria:

- Memorias de datos e instrucciones separadas con caches separadas de datos e instrucciones
- Elimina conflicto entre la búsqueda de una instrucción y acceso a los datos

## La memoria

- Si la máquina encauzada tiene un ciclo de reloj como el de la máquina no encauzada, el sistema de memoria debe proporcionar un ancho de banda cinco veces mayor
- Máquina no encauzada (2 accesos cada cinco ciclos)
- Máquina encauzada (2 accesos cada ciclo)

# Segmentación básica del MIPS

Introducción

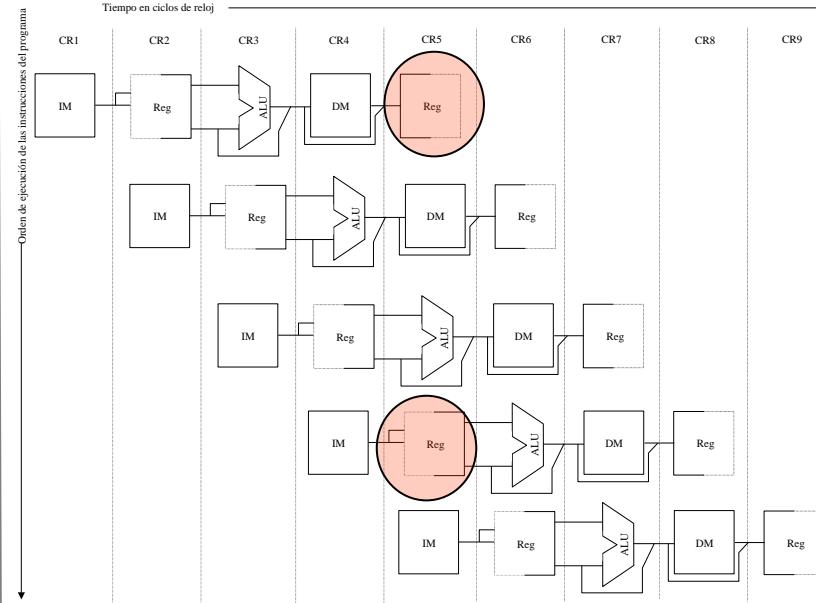
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



## El banco de registros

- El banco de registros se usa en dos etapas
- Para leer en ID
- Para escribir en WB
- Dos lecturas y una escritura en cada ciclo

## El banco de registros

- ¿Qué ocurre si una lectura y una escritura se realizan sobre el mismo registro?

# Segmentación básica del MIPS

Introducción

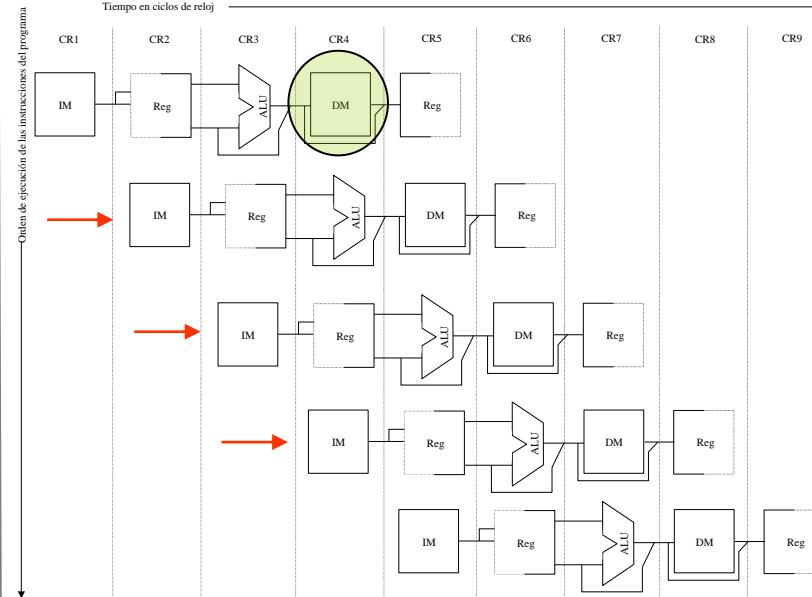
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



El PC

- Iniciar una nueva instrucción cada ciclo de reloj requiere incrementar y guardar el PC en cada ciclo, durante la etapa IF
- Problema con los saltos, que cambian el PC, pero no lo hacen hasta la etapa MEM

El PC

- En el caso encauzado, organizaremos el camino de datos para escribir el PC en IF bien con el PC incrementado en 4 o el valor del destino del salto de una instrucción de salto ejecutada previamente

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

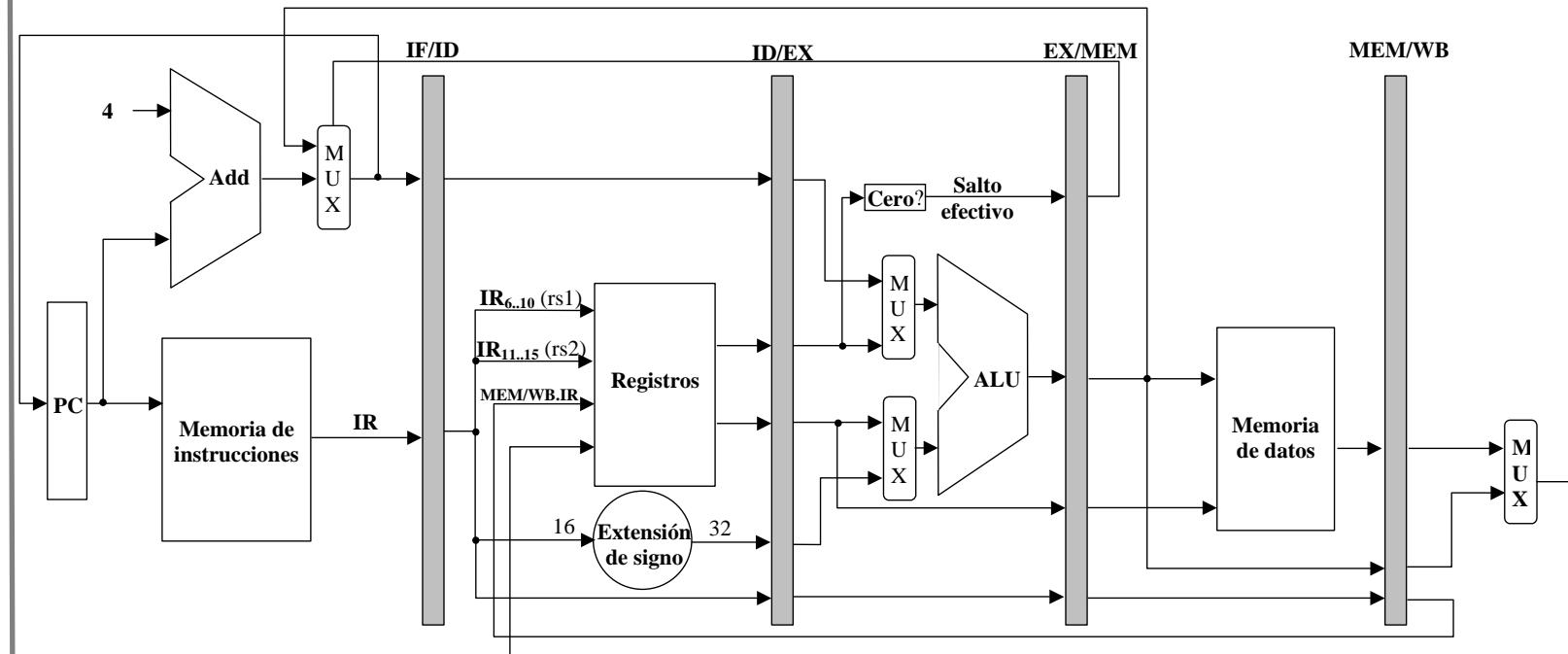
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Los registros intermedios



- ◆ Los registros intermedios sirven para transportar valores e información de control desde una etapa a la siguiente
- ◆ **Ejemplo**, campo de IR que apunta al operando registro destino de una operación load o ALU se proporciona desde el registro intermedio MEM/WB en lugar de obtenerlo de IF/ID.

# Segmentación básica del MIPS

Introducción

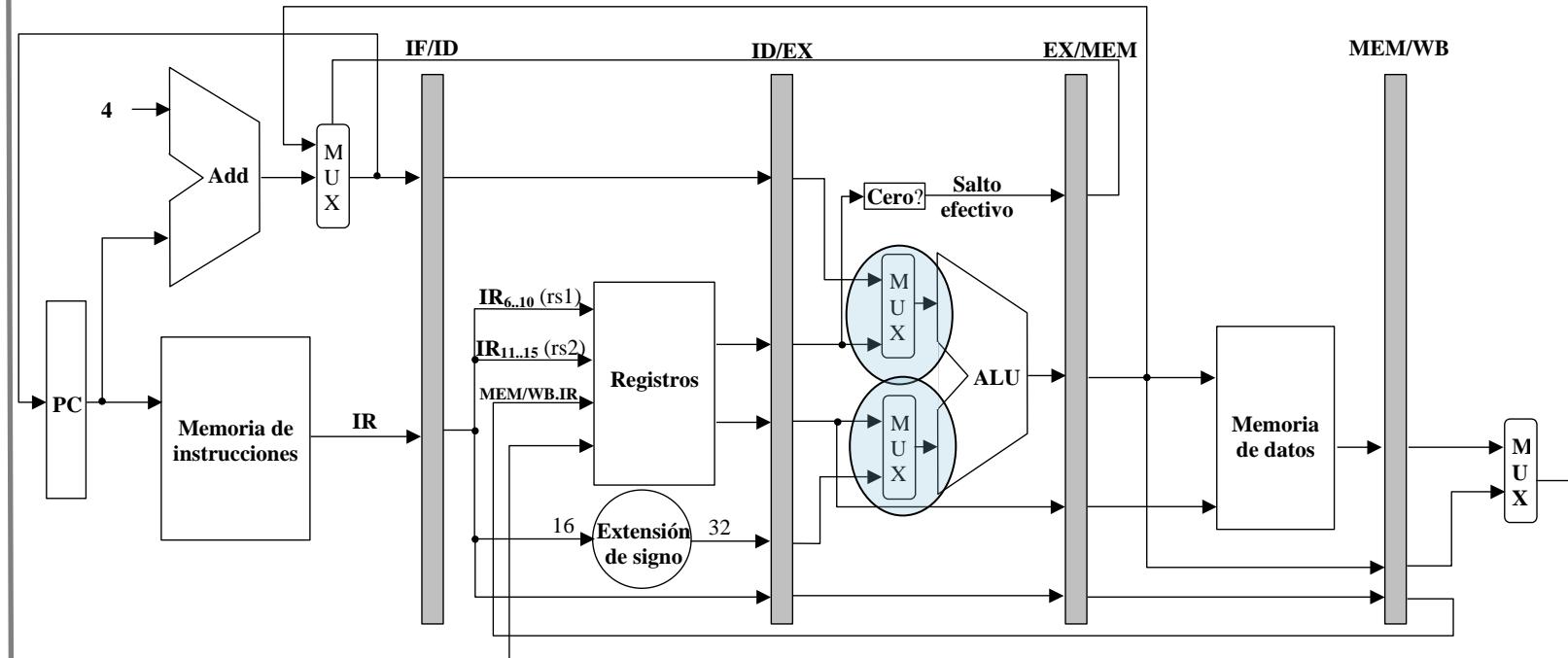
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



## Los multiplexores

- **Los dos multiplexores de la ALU:**
  - Las entradas se establecen dependiendo del tipo de instrucción, que se encuentra en el campo IR del registro intermedio ID/EX
  - **Multiplexor alto:** se establece según la instrucción sea un salto o no
  - **Multiplexor bajo:** se establece según la instrucción sea ALU registro-registro o cualquier otro tipo de operación

# Segmentación básica del MIPS

Introducción

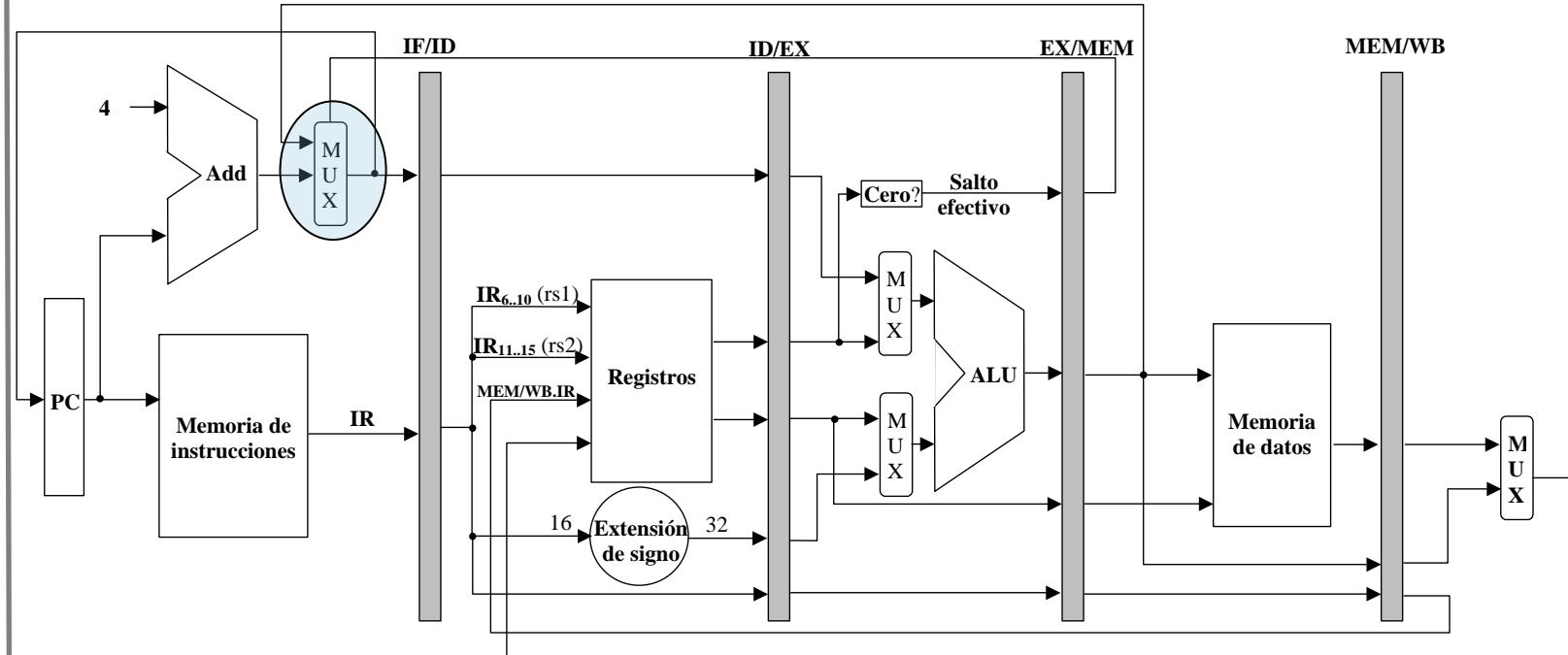
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



## Los multiplexores

- **El multiplexor en la etapa IF**
- El multiplexor que selecciona el valor que será escrito en el PC se ha movido a la etapa IF, de forma que el PC sólo puede ser escrito durante la etapa IF
- Selecciona el uso del PC en curso o el valor de EX/MEM.ALUoutput como dirección de instrucción. Este multiplexor se controla mediante el campo EX/MEM.cond

# Segmentación básica del MIPS

Introducción

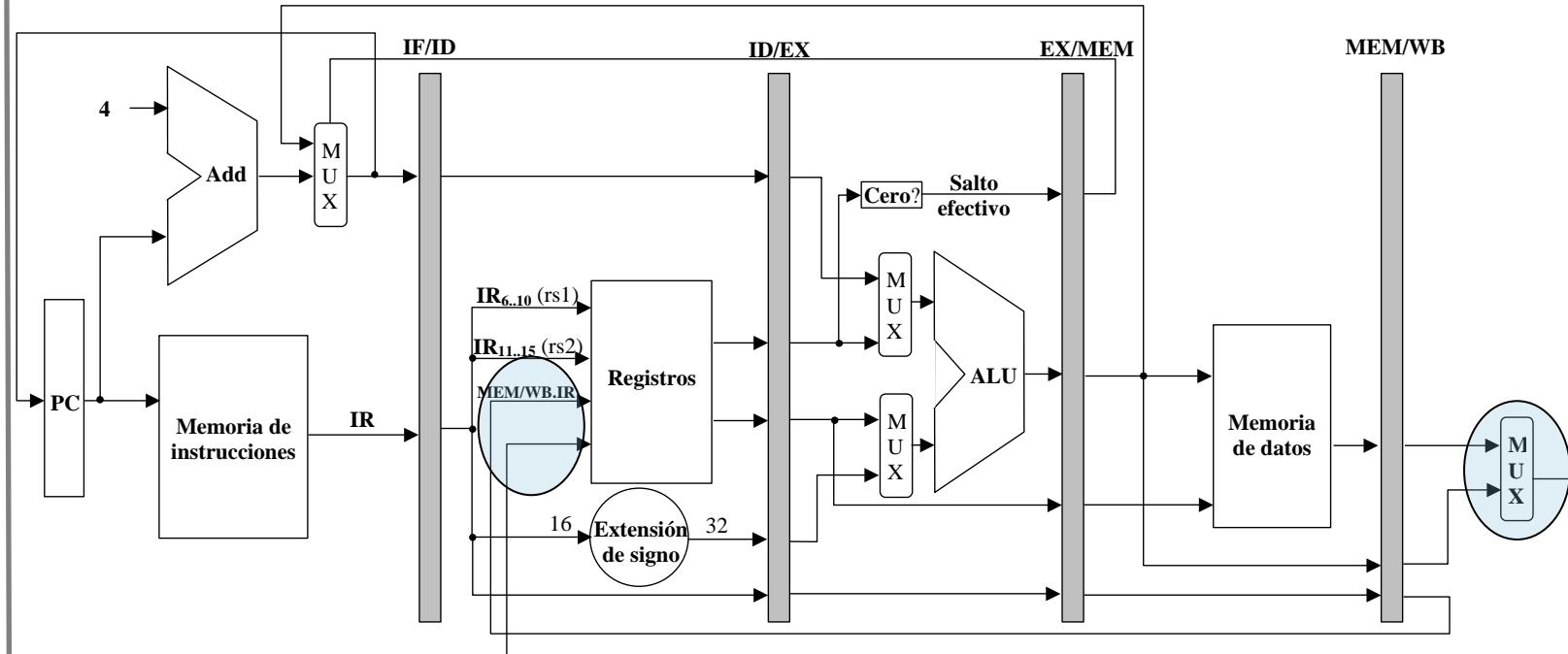
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



## Los multiplexores

- El cuarto multiplexor
  - Se controla según la instrucción en la etapa WB sea una operación ALU o una load.
  - Es necesario un multiplexor más para controlar si el campo de IR que expresa el registro destino está en una u otra posición según la instrucción sea ALU registro-registro frente a ALU inmediato o load.

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

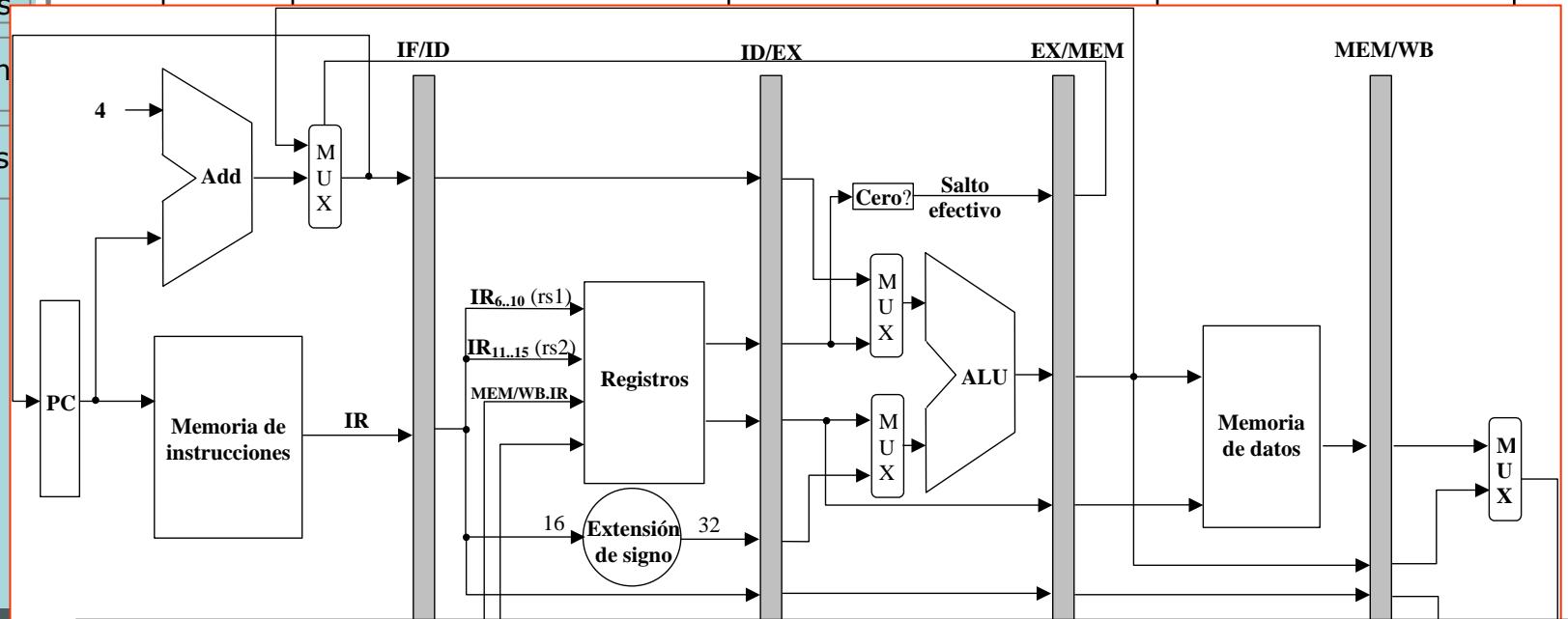
Segmentaci

**IF**

**IF/ID.IR**  $\leftarrow$  Mem[PC];  
**IF/ID.NPC, PC**  $\leftarrow$  ( if EX/MEM.cond {EX/MEM.ALUoutput} else {PC+4} )

**ID**

**ID/EX.A**  $\leftarrow$  Regs [IF/ID.IR<sub>6..10</sub>]; **ID/EX.B**  $\leftarrow$  Regs [IF/ID.IR<sub>11..15</sub>];  
**ID/EX.NPC**  $\leftarrow$  IF/ID.NPC;  
**ID/EX.IR**  $\leftarrow$  IF/ID.IR  
**ID/EX.Inm**  $\leftarrow$  (IF/ID.IR<sub>16</sub>)<sup>16</sup> ## IF/ID.IR<sub>16..31</sub>



Instrucción tipo I

Iw R1,100(R2) (sw)  
add R1,R2,#100  
beqz R1,ET

6

5

5

Inmediato

Instrucción tipo R

add R1,R2,R3

6

5

5

func

Instrucción tipo J

J ET

26

Cód ope

Desplazamiento añadido al PC

# Segmentación básica del MIPS

Introducción

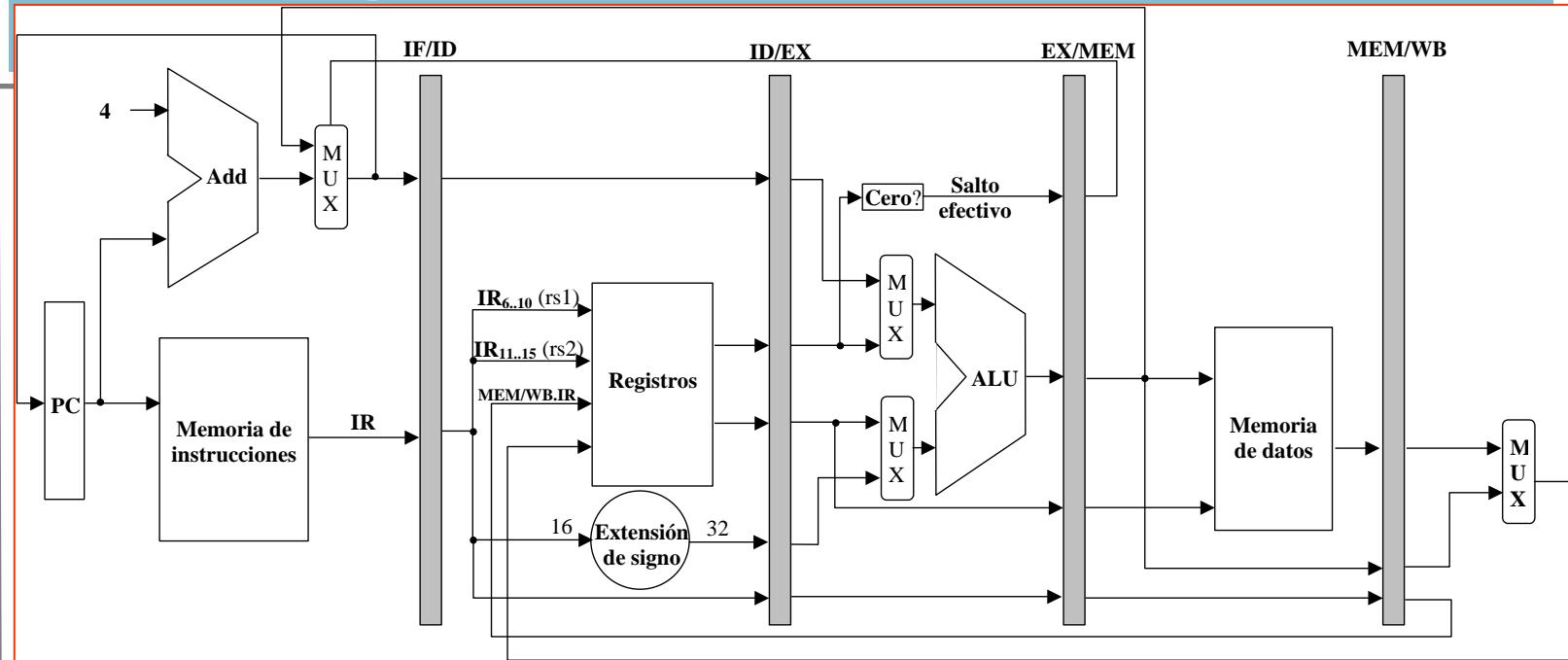
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



	Instrucción ALU	Instrucción load o store	Instrucción de salto
EX	<b>EX/MEM.IR</b> $\leftarrow$ ID/EX.IR; <b>EX/MEM.ALUoutput</b> $\leftarrow$ ID/EX.A func ID/EX.B; O <b>EX/MEM.ALUoutput</b> $\leftarrow$ ID/EX.A op ID/EX.Inm; <b>EX/MEM.Cond</b> $\leftarrow$ 0;	<b>EX/MEM.IR</b> $\leftarrow$ ID/EX.IR; <b>EX/MEM.ALUoutput</b> $\leftarrow$ ID/EX.A + ID/EX.Inm; <b>EX/MEM.cond</b> $\leftarrow$ 0; <b>EX/MEM.B</b> $\leftarrow$ ID/EX.B;	<b>EX/MEM.ALUoutput</b> $\leftarrow$ $\text{ID/EX.NPC} + \text{ID/EX.Inm};$ <b>EX/MEM.cond</b> $\leftarrow$ $(\text{ID/EX.A op 0});$

Instrucción tipo I	Instrucción tipo R	Instrucción tipo J
<i>lw R1,100(R2) (sw)</i> <i>add R1,R2,#100</i> <i>beqz R1,ET</i>	<i>add R1,R2,R3</i>	<i>J ET</i>
6      5      5      16 Cód ope    Rs1    Rd      Inmediato	6      5      5      11 Cód ope    Rs1    Rs2    Rd      func	6      26 Cód ope      Desplazamiento añadido al PC

# Segmentación básica del MIPS

Introducción

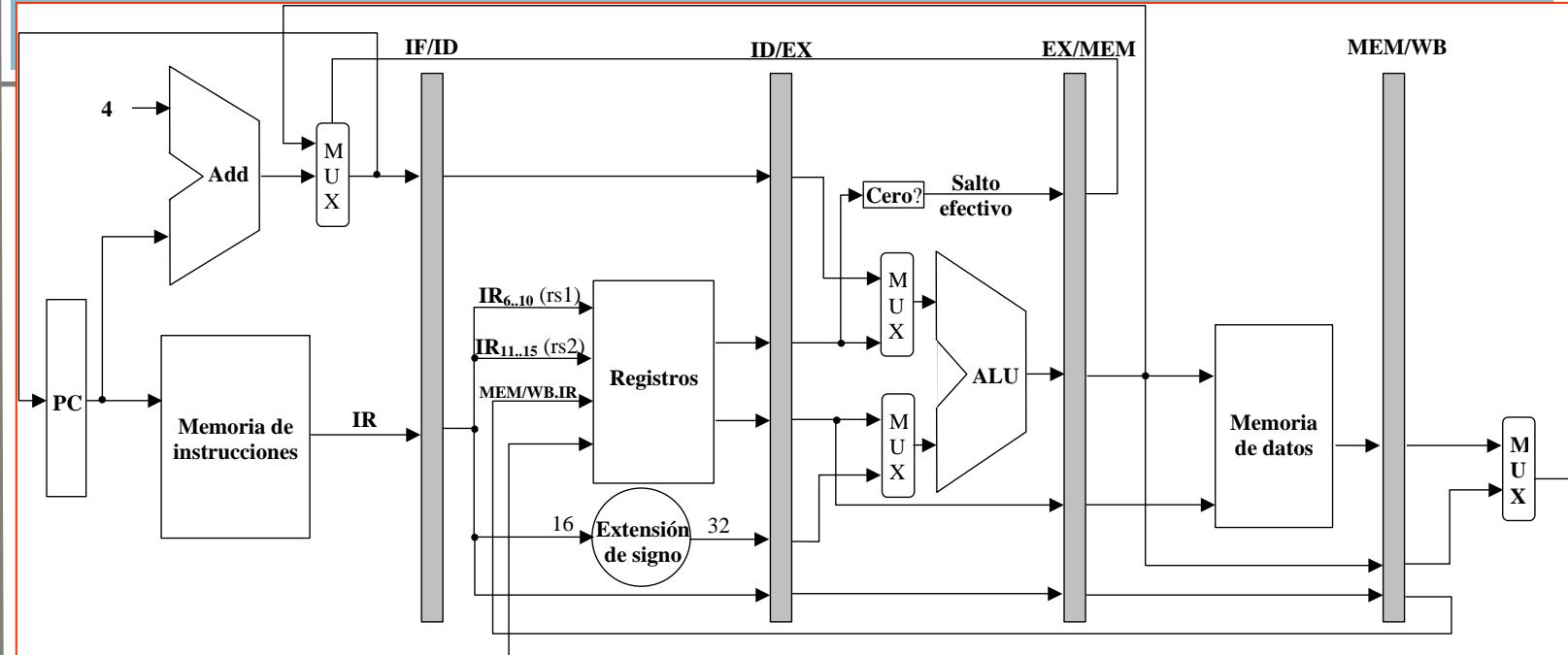
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



	Instrucción ALU	Instrucción load o store	Instrucción de salto
MEM	$\text{MEM/WB.IR} \leftarrow \text{EX/MEM.IR};$ $\text{MEM/WB.ALUoutput} \leftarrow \text{EX/MEM.ALUoutput};$	$\text{MEM/WB.IR} \leftarrow \text{EX/MEM.IR};$ $\text{MEM/WB.LMD} \leftarrow \text{Mem}[\text{EX/MEM.ALUoutput}];$ $\text{O}$ $\text{Mem}[\text{EX/MEM.ALUoutput}] \leftarrow \text{EX/MEM.B},$	$\text{MEM/WB.IR} \leftarrow \text{EX/ME M.IR};$

Instrucción tipo I

Iw R1,100(R2) (sw)  
add R1,R2,#100  
beqz R1,ET

6 5 5

Cód ope

Rs1

Rd

Inmediato

Instrucción tipo R

add R1,R2,R3

6

5

5

11

Cód ope

Rs1

Rs2

Rd

func

Instrucción tipo J

J ET

6

26

Cód ope

Desplazamiento añadido al PC

# Segmentación básica del MIPS

Introducción

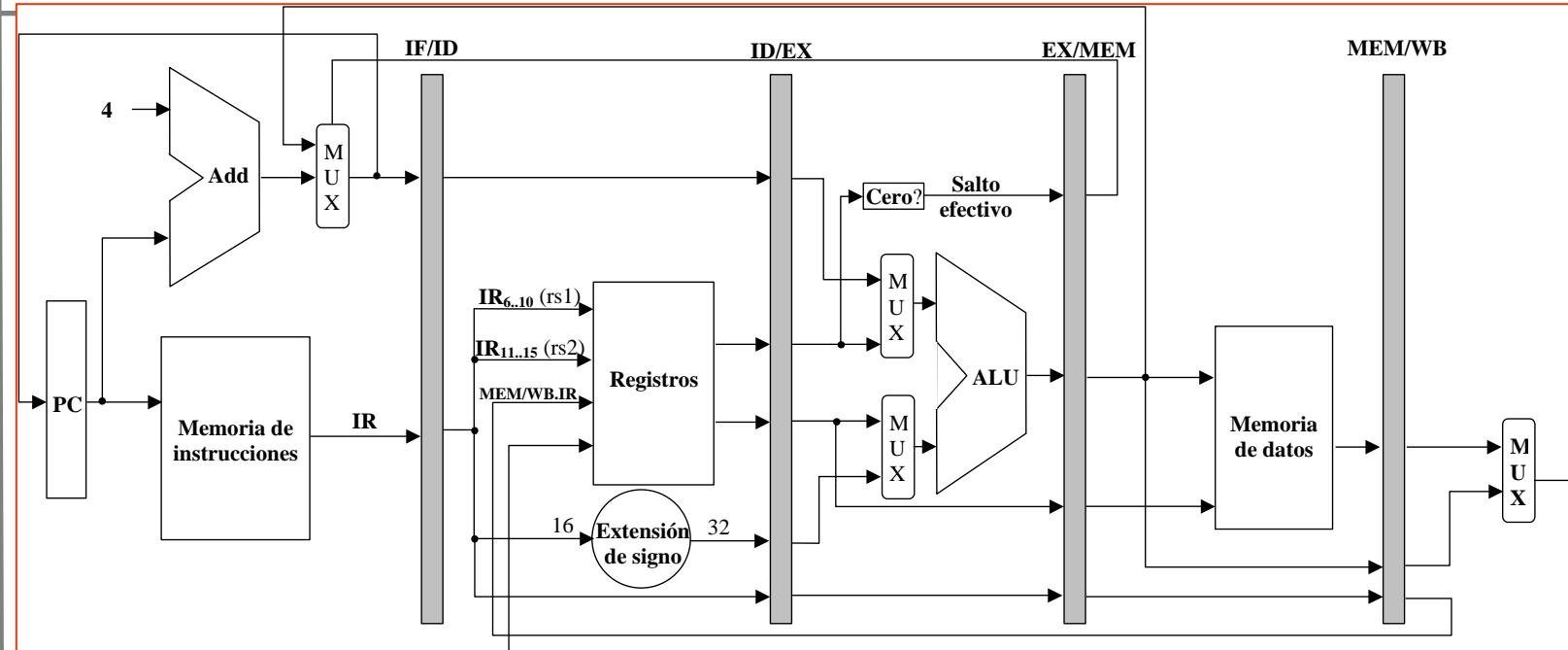
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



	Instrucción ALU	Instrucción load o store	Instrucción de salto
WB	<b>Regs[MEM/WB.IR<sub>16..20</sub>] ←</b> MEM/WB.ALUoutput; O <b>Regs[MEM/WB.IR<sub>11..15</sub>] ←</b> MEM/WB.ALUoutput;	<b>Regs[MEM/WB.IR<sub>11..15</sub>] ←</b> MEM/WB.LMD;	

Instrucción tipo I

lw R1,100(R2) (sw)  
 add R1,R2,#100  
 beqz R1,ET

6

5

5

16

Instrucción tipo R

add R1,R2,R3

6

5

5

11

Instrucción tipo J

J ET

6

26

Cód ope

Rs1

Rd

Inmediato

Cód ope

Rs1

Rs2

Rd

func

Cód ope

Desplazamiento añadido al PC

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **Rendimiento segmentado**
- ➋ La segmentación **incrementa la productividad** de las instrucciones de la CPU (número de instrucciones completas por unidad de tiempo) pero **no reduce** el tiempo de **ejecución de una instrucción individual**
- ➌ Normalmente se ve incrementado el tiempo de ejecución de instrucciones individuales debido a la sobrecarga en el control de la segmentación
- ➍ El incremento en la productividad de instrucciones significa que los **programas se ejecutan más rápido** y tienen **menor tiempo total de ejecución**

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **Ejemplo.** Supongamos que los tiempos requeridos para las cinco unidades funcionales, que operan en cada uno de los cinco ciclos son: 10ns, 8ns, 10ns, 10ns y 7ns. Suponemos que la segmentación añade 1ns de sobrecarga. Buscar la ganancia frente a la versión monociclo.

## Monociclo

- ➊ Máquina no segmentada ejecuta todas las instrucciones en un único ciclo de reloj
- ➋ Tiempo medio por instrucción es el tiempo del ciclo de reloj que es la suma de los tiempos para cada paso de la ejecución
- ➌ Tiempo de ejecución medio por instrucción

$$T_{emi} = 10 + 8 + 10 + 10 + 7 = 45 \text{ ns}$$

# Segmentación básica del MIPS

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **Ejemplo.** Supongamos que los tiempos requeridos para las cinco unidades funcionales, que operan en cada uno de los cinco ciclos son: 10ns, 8ns, 10ns, 10ns y 7ns. Suponemos que la segmentación añade 1ns de sobrecarga. Buscar la ganancia frente a la versión monociclo.

## Segmentada

- ➊ El tiempo del ciclo de reloj en la máquina segmentada debe ser el tiempo más largo de cada una de las etapas (10ns) más la sobrecarga de 1 ns, sumando un total de 11ns.
- ➋ CPI=1,
- ➌ Tiempo de ejecución medio por instrucción de 11ns.

$$G_s = \frac{\text{tiempo medio instrucción sin segmentación}}{\text{tiempo medio instrucción con segmentación}} = \frac{45\text{ns}}{11\text{ns}} = 4.1$$

# Segmentación básica del MIPS

- ➊ **Ejemplo.** Supongamos que los tiempos requeridos para las cinco unidades funcionales, que operan en cada uno de los cinco ciclos son: 10ns, 8ns, 10ns, 10ns y 7ns. Suponemos que la segmentación añade 1ns de sobrecarga. Buscar la ganancia frente a la versión monociclo.

**La segmentación puede entenderse como una mejora del CPI, que es lo que típicamente entendemos o como una reducción del ciclo de reloj**

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Hay situaciones, llamadas riesgos, que impiden que se ejecute la siguiente instrucción del flujo de instrucciones. Estos riesgos reducen la ganancia.

## ➊ Tres clases de riesgos:

### ➊ Riesgos estructurales

- ➌ Surgen de conflictos de los recursos cuando el hardware no puede soportar todas las combinaciones de instrucciones en ejecución.

### ➊ Riesgos por dependencias de datos

- ➌ Surgen cuando una instrucción depende de los resultados de una instrucción anterior.

### ➊ Riesgos de control

- ➌ Surgen de la segmentación de los saltos y otras instrucciones que cambian el PC.

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Pueden hacer necesario detener la segmentación.
- ➋ Cuando una instrucción se detiene, las instrucciones posteriores también lo hacen y no se buscan instrucciones nuevas. Las anteriores pueden continuar
- ➌ Una detención disminuye la ganancia con respecto a la ideal

# Riesgos de la segmentación

## ➊ Rendimiento de la segmentación con detenciones

$$\begin{aligned} G_s &= \frac{\text{tiempo medio instrucción sin segmentación}}{\text{tiempo medio instrucción con segmentación}} = \\ &= \frac{CPI \text{ sin segmentación} \cdot \text{Ciclo de reloj sin segmentación}}{CPI \text{ con segmentación} \cdot \text{Ciclo de reloj con segmentación}} = \\ &= \frac{\text{Ciclo de reloj sin segmentación}}{\text{Ciclo de reloj con segmentación}} \times \frac{CPI \text{ sin segmentación}}{CPI \text{ con segmentación}} \end{aligned}$$

- ➊ La segmentación puede entenderse como una mejora del CPI o como una reducción del ciclo de reloj.

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Rendimiento de la segmentación con detenciones

### ◆ Segmentación como una mejora del CPI

- ◆ Suponed que el CPI ideal de un cauce es 1, el CPI de un cauce se calcular como:

$$\begin{aligned} \text{CPI con segmentación} &= \text{CPI}_{\text{ideal}} + \text{Ciclos reloj detención de segmentación por instrucción} \\ &= 1 + \text{Ciclos de reloj de detención de la segmentación por instrucción} \end{aligned}$$

- ◆ Ejemplo: Supongamos un 25% instrucciones de acceso a memoria que generan una detención:

$$\bullet \text{ CPI} = 1 + 0,25 * 1 = 1,25$$

$$\bullet \text{ CPI} = 0,75 * 1 + 0,25 * 2 = 0,75 + 0,5 = 1,25$$

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Rendimiento de la segmentación con detenciones

### ◆ Segmentación como una mejora del CPI

- ◆ Suponed que el CPI ideal de un cauce es 1, el CPI de un cauce se calcular como:

$$\begin{aligned} \text{CPI con segmentación} &= \text{CPI}_{\text{ideal}} + \text{Ciclos reloj detención de segmentación por instrucción} \\ &= 1 + \text{Ciclos de reloj de detención de la segmentación por instrucción} \end{aligned}$$

- ◆ Si ignoramos el incremento potencial en el ciclo de reloj debido a la segmentación, y asumimos que las etapas están equilibradas, podemos igualar el ciclo de reloj de las dos máquinas:

$$G_s = \frac{\text{Ciclo de reloj sin segmentación}}{\text{Ciclo de reloj con segmentación}} \times \frac{\text{CPI sin segmentación}}{\text{CPI con segmentación}}$$

$$G_s = \frac{\text{CPI sin segmentación}}{1 + \text{ciclos de reloj de detención de la segmentación por instrucción}}$$

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Rendimiento de la segmentación con detenciones

- ◆ Un caso simple e importante: todas las instrucciones tardan el mismo número de ciclos que además es igual al número de etapas del cauce (profundidad de la segmentación). CPI sin segmentación es igual a la profundidad del cauce y por tanto:

$$G_s = \frac{\text{Profundidad de la segmentación}}{1 + \text{ciclos de reloj de detención de la segmentación por instrucción}}$$

- ◆ Si no hubiese detenciones esto llevaría al resultado intuitivo de que la segmentación puede mejorar el rendimiento en la magnitud de la profundidad de la segmentación.

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Rendimiento de la segmentación con detenciones

### ◆ Segmentación como una reducción del ciclo de reloj.

- ◆ Alternativamente si entendemos la segmentación como una mejora del ciclo de reloj, entonces podemos suponer que el CPI de la máquina no segmentada así como el de la segmentada vale 1. Esto nos lleva a:

$$G_s = \frac{CPI \text{ sin segmentación}}{CPI \text{ con segmentación}} \times \frac{\text{Ciclo de reloj sin segmentación}}{\text{Ciclo de reloj con segmentación}}$$

$$= \frac{1}{1 + \text{ciclos de detención por instrucción}} \times \frac{\text{Ciclo de reloj sin segmentación}}{\text{Ciclo de reloj con segmentación}}$$

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ **Rendimiento de la segmentación con detenciones**

- ◆ En el caso de que las etapas del cauce estén equilibradas y no exista sobrecarga, el ciclo de reloj en la máquina segmentada es menor que el ciclo de reloj de la máquina no segmentada en un factor igual a la profundidad del cauce.

$$\text{Ciclo de reloj con segmentación} = \frac{\text{Ciclo de reloj sin segmentación}}{\text{profundidad de la segmentación}}$$

$$\text{profundidad de la segmentación} = \frac{\text{Ciclo de reloj sin segmentación}}{\text{Ciclo de reloj con segmentación}}$$

# Riesgos de la segmentación

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Rendimiento de la segmentación con detenciones

- ◆ Esto nos lleva a

$$G_s = \frac{1}{1 + \text{ciclos de detención por instrucción}} \times \frac{\text{Ciclo de reloj sin segmentación}}{\text{Ciclo de reloj con segmentación}}$$

$$= \frac{1}{1 + \text{ciclos de detención por instrucción}} \times \text{profundidad de la segmentación}$$

- ◆ De nuevo nos encontramos con el resultado intuitivo de que la segmentación puede mejorar el rendimiento en la magnitud de la profundidad de la segmentación, si no hubiese detenciones

# Riesgos estructurales

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Si alguna combinación de instrucciones no puede darse en el cauce por conflicto de recursos, se dice que la máquina tiene un riesgo estructural.
- ➋ Cuando una máquina se segmenta, la ejecución solapada de instrucciones necesita la segmentación de unidades funcionales y la duplicación de recursos para permitir todas las posibles combinaciones de instrucciones en el cauce.

# Riesgos estructurales

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ➊ Dos formas de encontrar riesgos estructurales

### ➊ Unidades funcionales que no están completamente segmentada

- ➊ Los casos más comunes de riesgos estructurales surgen cuando alguna unidad funcional no está completamente segmentada. En ese caso una secuencia de instrucciones usando esta unidad no segmentada no puede proceder a razón de una instrucción por ciclo.
- ➊ **Ejemplo:** Unidad aritmética en punto flotante que no permite segmentación.

# Riesgos estructurales

## ◆ Dos formas de encontrar riesgos estructurales

### ◆ Recurso que no se ha duplicado lo suficiente

- ◆ Otra forma habitual de encontrar riesgos estructurales es cuando un recurso no se ha duplicado lo suficiente para soportar todas las combinaciones.
- ◆ **Ejemplo:** una máquina puede tener un solo puerto de escritura en el banco de registros. La segmentación puede necesitar realizar dos escrituras en un único ciclo
- ◆ Cuando una secuencia de instrucciones encuentre este riesgo, el cauce detendrá una de las instrucciones hasta que la unidad requerida este libre. Estas detenciones incrementarán el CPI ideal de valor 1.

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

# Riesgos estructurales

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

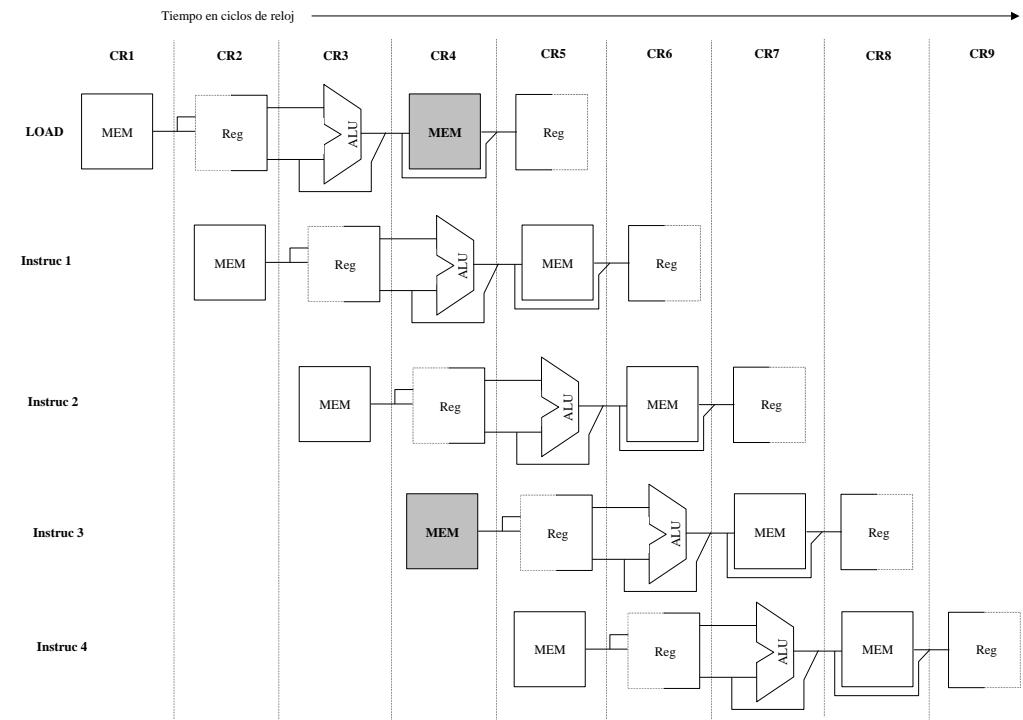
Optimización

Superescalares

Segmentación

◆ **Ejemplo:** Algunas máquinas segmentadas comparten una única memoria para datos e instrucciones.

◆ Cuando una instrucción contiene una referencia a un dato de memoria, entrará en conflicto con la referencia a instrucción de una instrucción posterior.



# Riesgos estructurales

Introducción

Segmentación  
del repertorio

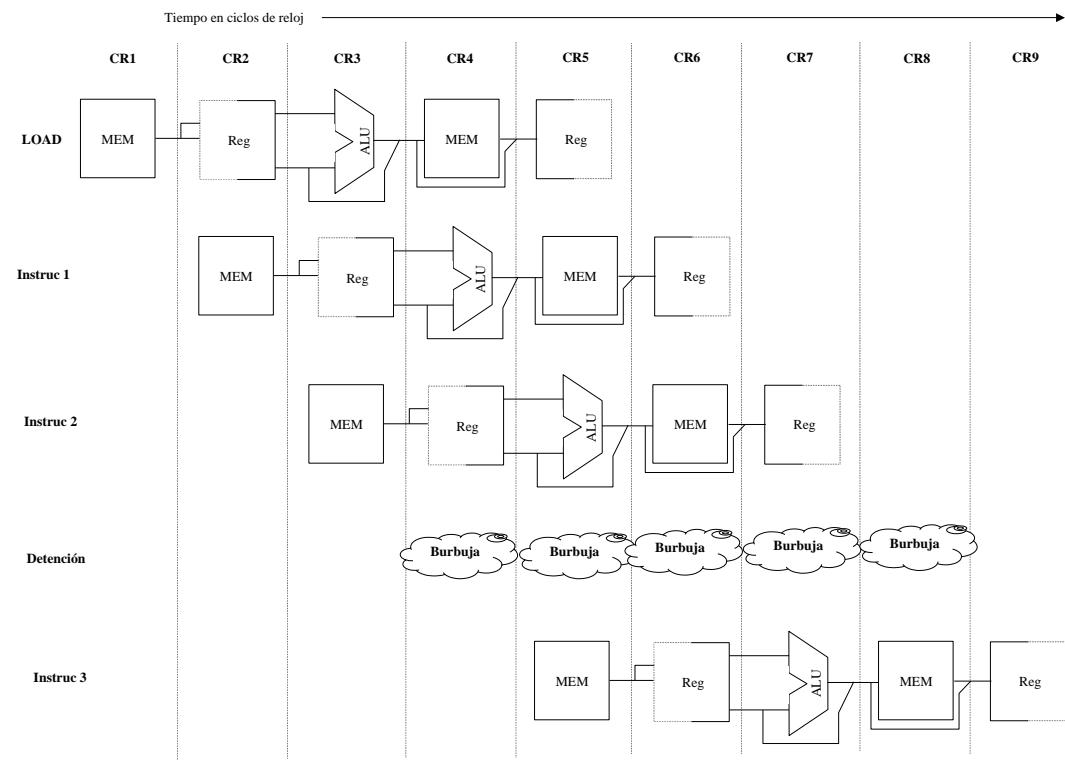
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Detenemos la segmentación durante un ciclo de reloj cuando ocurre el acceso a los datos de memoria.
- ➋ A las detenciones se les llama burbujas, ya que flotan en el cauce ocupando espacio pero sin realizar trabajo.
- ➌ Ninguna instrucción terminará durante el ciclo de reloj 8,
- ➍ Asumimos que la instrucción 1 no es load ni store, en ese caso la instrucción 3 no puede comenzar su ejecución.



# Riesgos estructurales

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- En lugar de dibujar el camino de datos segmentado cada vez, los diseñadores a menudo indican el comportamiento de las detenciones usando diagramas simples con los nombres de las etapas.
- La figura muestra la detención indicando el ciclo donde no ocurren acciones y desplazando la instrucción 3 a la derecha

Nº ciclo de reloj										
Instrucción	1	2	3	4	5	6	7	8	9	10
Load	IF	ID	EX	MEM	WB					
Instrucción i+1		IF	ID	EX	MEM	WB				
Instrucción i+2			IF	ID	EX	MEM	WB			
Instrucción i+3				Deten	IF	ID	EX	MEM	WB	
Instrucción i+4						IF	ID	EX	MEM	WB
Instrucción i+5							IF	ID	EX	MEM
Instrucción i+6								IF	ID	EX

# Riesgos estructurales

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- El efecto de la burbuja es ocupar los recursos para el hueco de una instrucción como si esta cruzase todo el cauce.
- El impacto sobre el rendimiento es que la inst. 3 no se completa hasta el ciclo 9 y ninguna inst. se completa durante el 8.

Nº ciclo de reloj											
Instrucción	1	2	3	4	5	6	7	8	9	10	
Load	IF	ID	EX	MEM	WB						
Instrucción i+1		IF	ID	EX	MEM	WB					
Instrucción i+2			IF	ID	EX	MEM	WB				
Instrucción i+3				Deten	IF	ID	EX	MEM	WB		
Instrucción i+4						IF	ID	EX	MEM	WB	
Instrucción i+5							IF	ID	EX	MEM	
Instrucción i+6								IF	ID	EX	

# Riesgos estructurales

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **Ejemplo.** Veamos cuanto puede costar el riesgo estructural load. Supongamos que las referencias a datos constituyen un 40% de la mezcla, y que el CPI ideal de la máquina segmentada, ignorando el riesgo estructural es 1. Suponemos que la máquina con el riesgo estructural tiene una frecuencia de reloj que es 1.05 veces mayor que la frecuencia de reloj que la máquina sin el riesgo. Descartando otras perdidas de rendimiento. ¿La segmentación con el riesgo estructural es más o menos rápida que sin él, y en que magnitud?.

Calculamos el tiempo medio por instrucción:

$$\text{Tiempo medio por instrucción} = \text{CPI} \times \text{Ciclo de reloj}$$

El tiempo medio por instrucción para la máquina ideal es (al no tener detenciones):

$$\text{Tiempo medio por instrucción ideal} = \text{Ciclo de reloj ideal}$$

# Riesgos estructurales

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **Ejemplo.** Veamos cuanto puede costar el riesgo estructural load. Supongamos que las referencias a datos constituyen un 40% de la mezcla, y que el CPI ideal de la máquina segmentada, ignorando el riesgo estructural es 1. Suponemos que la máquina con el riesgo estructural tiene una frecuencia de reloj que es 1.05 veces mayor que la frecuencia de reloj que la máquina sin el riesgo. Descartando otras perdidas de rendimiento. ¿La segmentación con el riesgo estructural es más o menos rápida que sin él, y en que magnitud?.

Tiempo medio por instrucción para la máquina con riesgo estructural es:

$$\text{Tiempo medio por instrucción} = \text{CPI} \times \text{Ciclo de reloj}$$

$$(1+0.4 \cdot 1) \cdot \frac{\text{Ciclo de reloj}_{ideal}}{1.05} = 1.3 \cdot \text{Ciclo de reloj}_{ideal}$$

La máquina sin el riesgo estructural es 1.3 veces más rápida

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

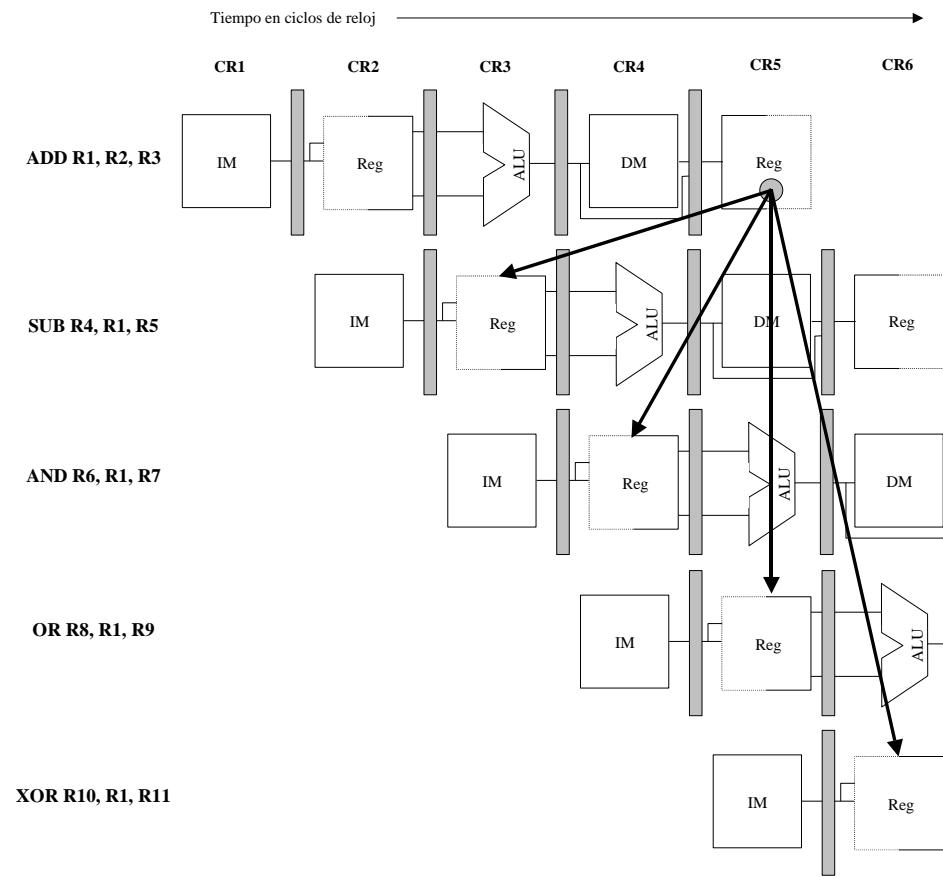
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ Los riesgos de datos ocurren cuando la segmentación cambia el orden de los accesos lectura/escritura a los operandos



# Riesgos por dependencia de datos

Introducción

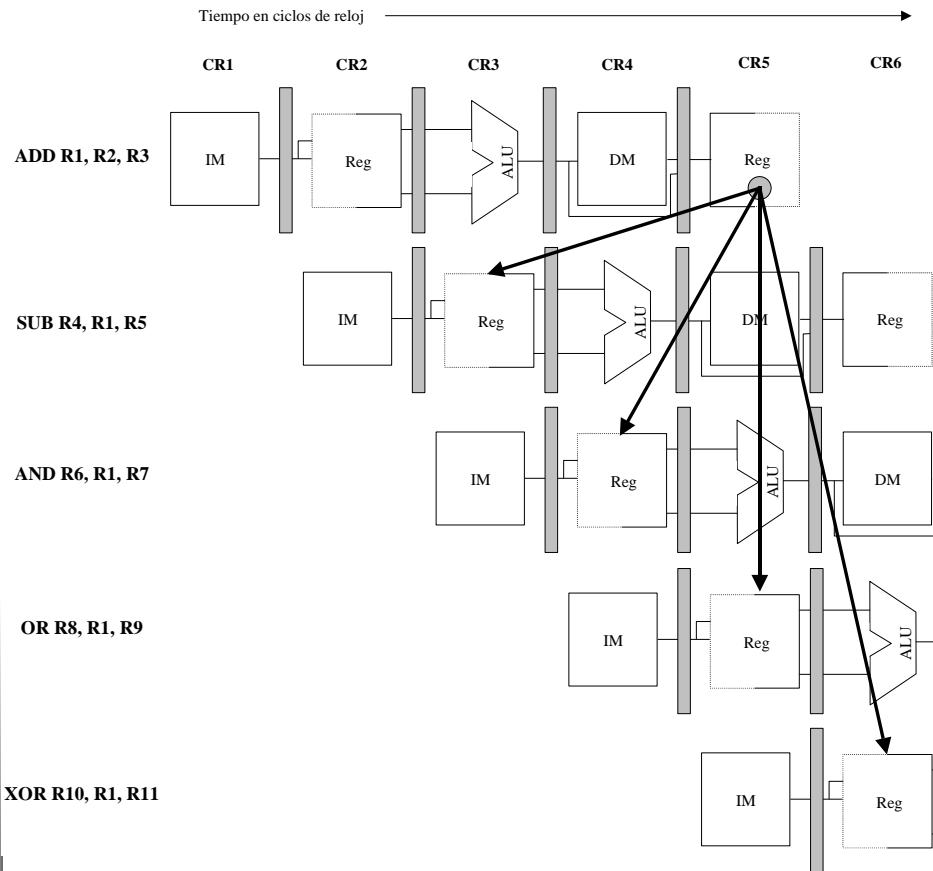
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



- Segunda instrucción (SUB)
- La instrucción ADD escribe el valor de R1 en la etapa WB, pero la instrucción SUB lee el valor durante su etapa ID. La instrucción SUB leerá el valor erróneo e intentará utilizarlo.
- El valor usado por la instrucción SUB ni siquiera es determinístico

# Riesgos por dependencia de datos

Introducción

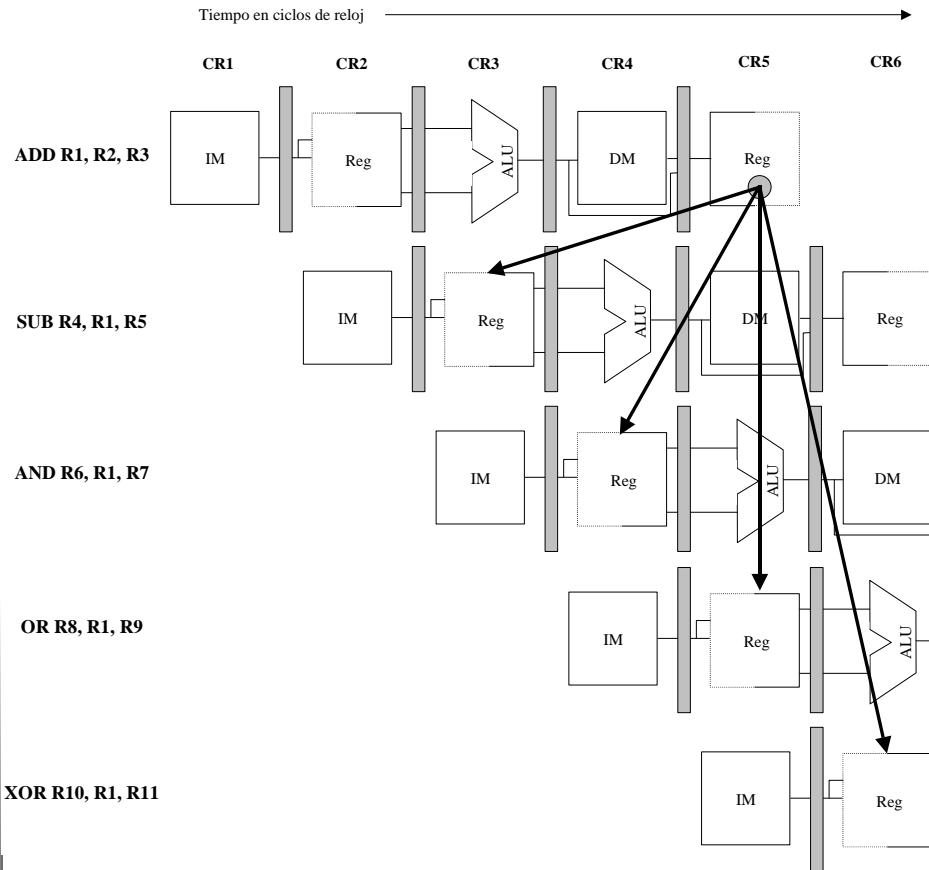
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



- ➊ Riesgo de datos de la tercera instrucción (AND)
- ➋ La instrucción AND. la escritura de R1 no se completa hasta el final del ciclo 5. Por lo tanto la instrucción AND que lee el registro durante el ciclo 4 recibirá resultados erróneos.

# Riesgos por dependencia de datos

Introducción

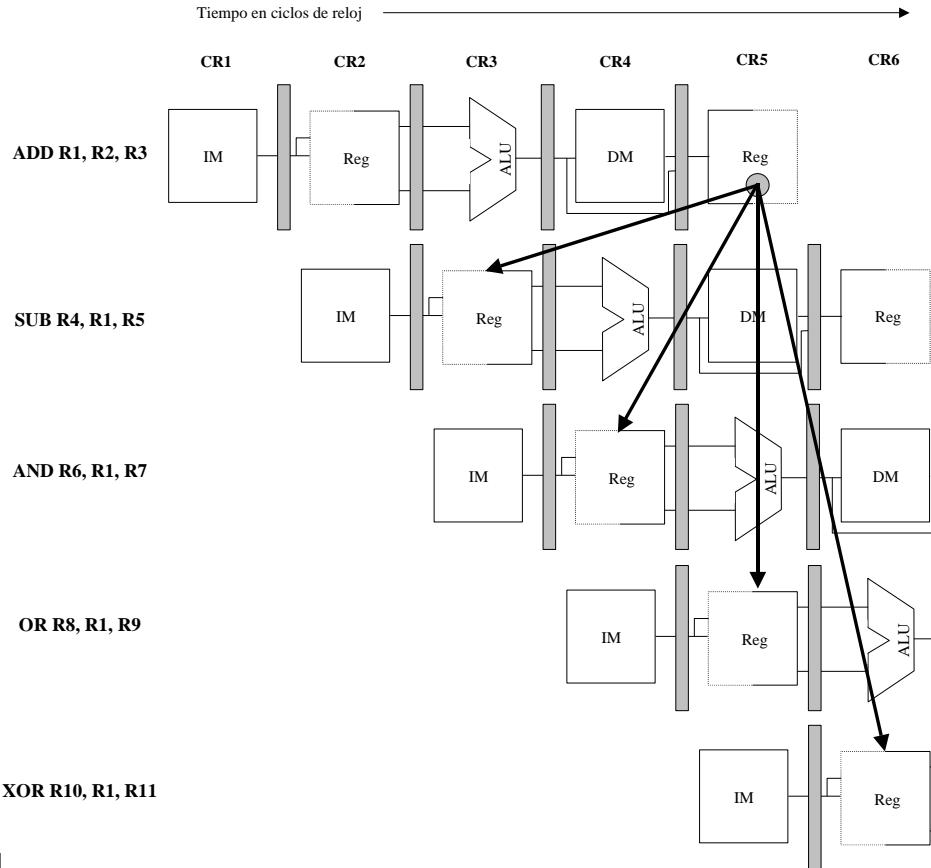
Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación



- ➊ Riesgo de datos de la cuarta instrucción (OR)
- ➋ La instrucción OR puede hacerse funcionar sin que incurra en un riesgo con una técnica de implementación simple. Las lecturas del banco de registros en la segunda mitad del ciclo y las escrituras en la primera mitad.
- ➌ La instrucción XOR funciona adecuadamente.

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ La técnica del adelantamiento

- ◆ Forwarding, bypassing, short-circuiting
- ◆ **La clave:** el resultado no se necesita realmente en la instrucción SUB hasta después de que la ADD realmente los produce
- ◆ Si el resultado puede moverse desde donde lo produce la ADD (el registro EX/MEM) hasta donde lo necesita la SUB (los multiplexores de entrada de la ALU) entonces la necesidad de detención puede evitarse

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Caucos  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ **La técnica del adelantamiento**

- ◆ **El adelantamiento trabaja** de la siguiente forma:
  - ◆ **1.** El resultado de la ALU del registro EX/MEM siempre se realimenta a los multiplexores de entrada de la ALU.
  - ◆ **2.** Si el hardware de adelantamiento detecta que la instrucción ALU previa ha escrito el registro correspondiente a una fuente de la operación ALU en curso, la lógica de control selecciona el valor adelantado como valor de entrada a la ALU en lugar del valor leído en el banco de registros
- ◆ **Completamente determinístico**
- ◆ Si la SUB se detiene, la ADD se completará y el adelantamiento no será activado

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

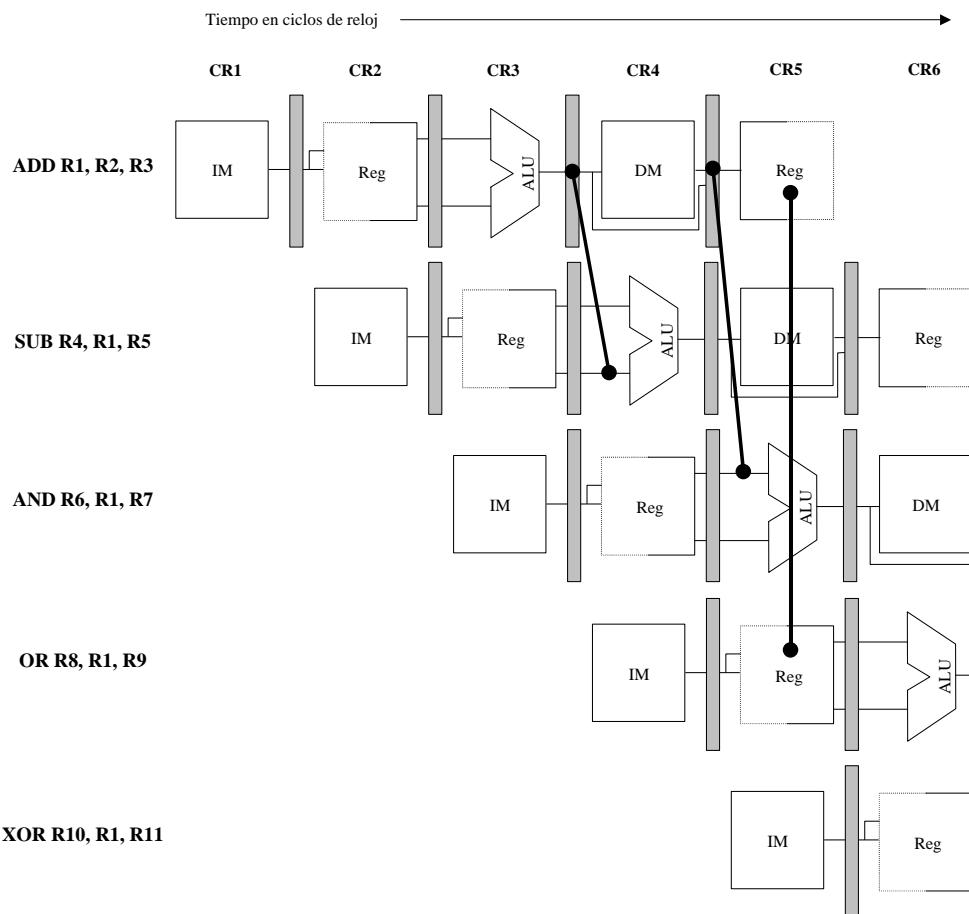
Optimización

Superescalares

Segmentación

## ◆ La técnica del adelantamiento

- ◆ La figura siguiente muestra el ejemplo con las rutas de adelantamiento



- ◆ Las entradas para las instrucciones SUB y AND se adelantan desde los registros intermedios EX/MEM y MEM/WB
- ◆ La OR recibe el resultado mediante adelantamiento a través del banco de registros

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

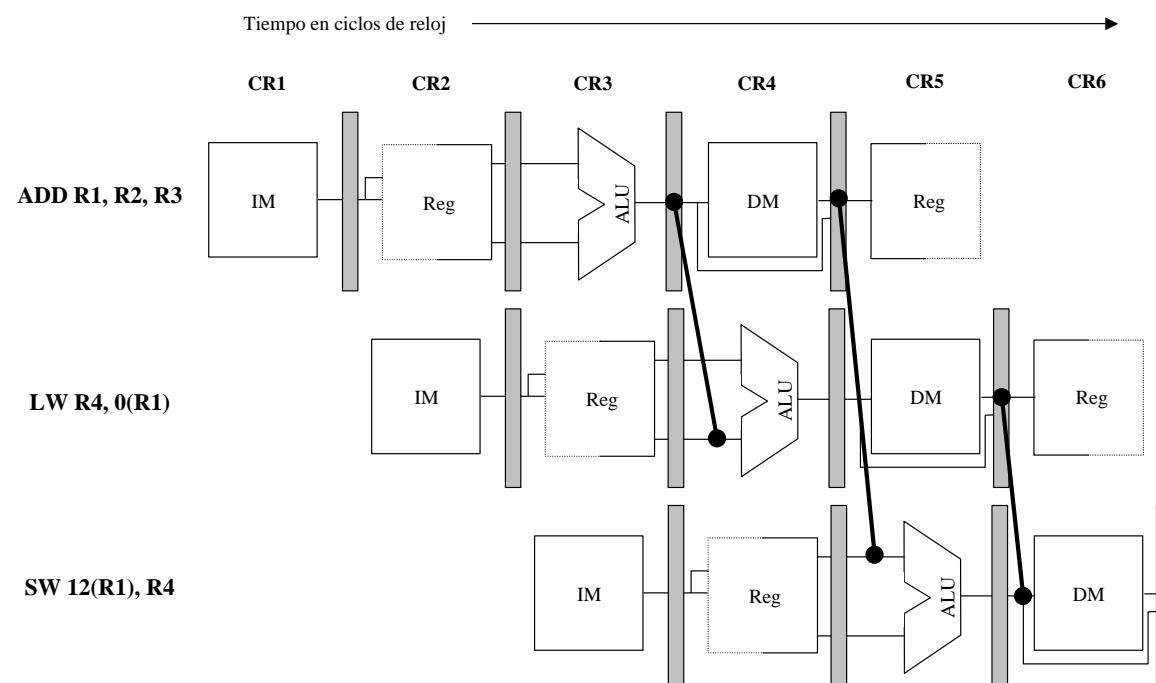
Optimización

Superescalares

Segmentación

## ◆ Generalización del adelantamiento

- ◆ El adelantamiento puede generalizarse para incluir el paso de resultados desde la salida de una unidad a la entrada de otra



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

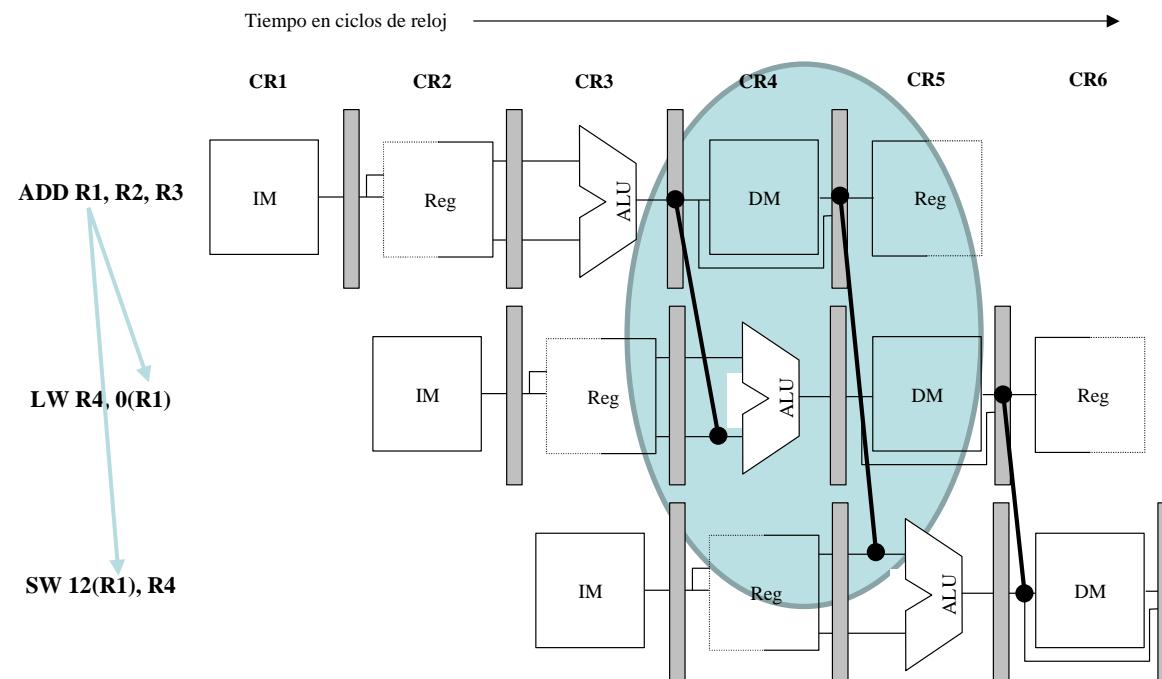
Optimización

Superescalares

Segmentación

## ◆ Generalización del adelantamiento

- ◆ La salida de la ALU se adelanta a la entrada de la ALU para el cálculo de la dirección tanto en la LOAD como en la STORE



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

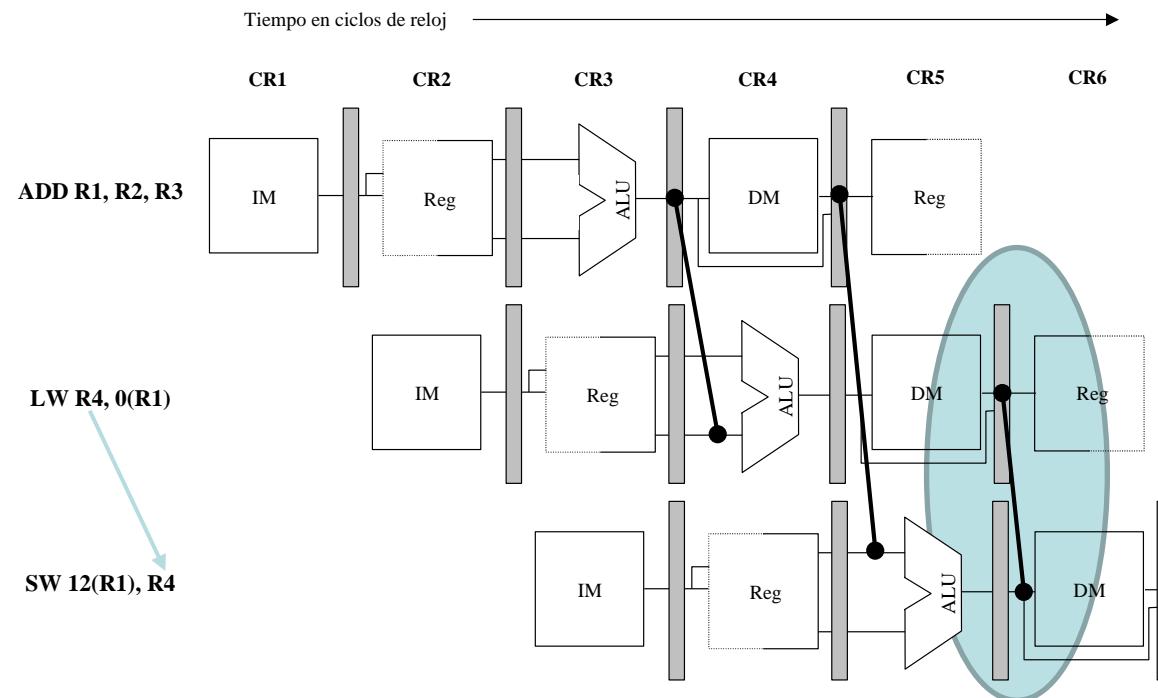
Optimización

Superescalares

Segmentación

## ◆ Generalización del adelantamiento

- ◆ STORE requiere un operando durante MEM. El resultado de la instrucción LOAD se adelanta desde la salida de la memoria en MEM/WB a la entrada de la memoria para ser almacenado.



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Generalización del adelantamiento

- ◆ En MIPS, podemos requerir una ruta de adelantamiento desde cualquier registro intermedio hacia la entrada de cualquier unidad funcional.
- ◆ Como tanto la ALU como la memoria de datos aceptan operandos, se necesitan rutas de adelantamiento hacia sus entradas desde los registros intermedios EX/MEM y MEM/WB.
- ◆ MIPS usa una unidad de detección de cero que opera durante el ciclo EX, y el adelantamiento a esa unidad puede ser también necesario.

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

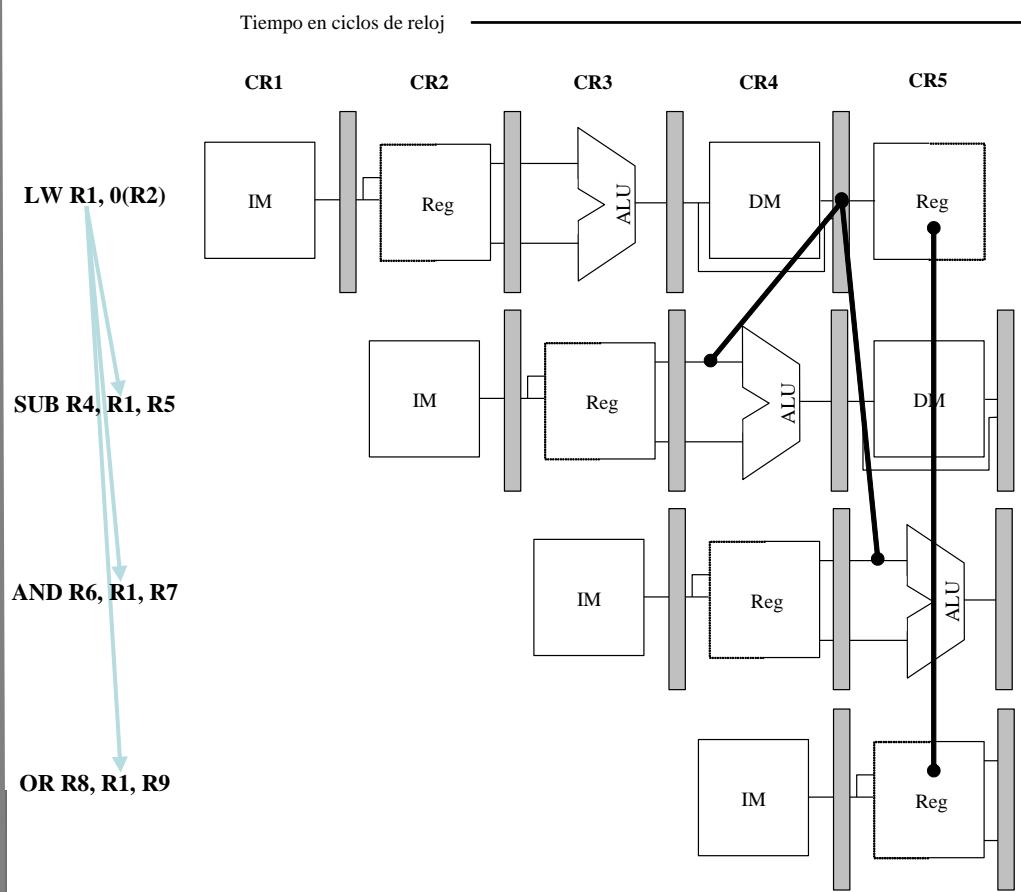
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Riesgos de datos que requieren detenciones



- ◆ LW no tiene el dato hasta el final del ciclo 4 (su ciclo MEM), mientras que la instrucción SUB necesita el dato en el inicio de ese ciclo
- ◆ Podemos adelantar a la ALU desde los registros MEM/WB para AND
- ◆ OR no tiene problema, ya que recibe el valor a través del banco de registros

# Riesgos por dependencia de datos

## ◆ **Hardware de interbloqueo**

- ◆ Necesitamos **añadir hardware de interbloqueo** del cauce, para preservar el patrón de ejecución correcto.
- ◆ En general, el hardware de interbloqueo, detecta un riesgo y detiene el cauce hasta que el riesgo desaparece.
- ◆ En este caso, el interbloqueo detiene el cauce, empezando por la instrucción que quiere usar el dato hasta que la instrucción fuente los produce.
- ◆ Este interbloqueo del cauce introduce una detención o burbuja, como se hacía para los riesgos estructurales.

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

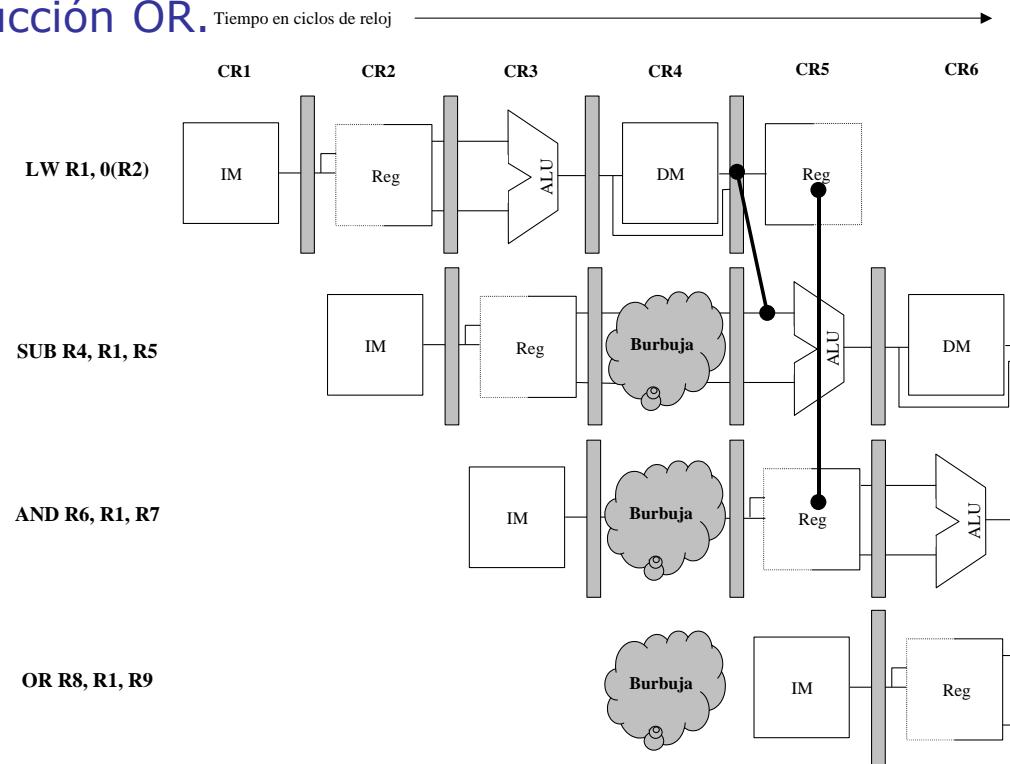
Optimización

Superescalares

Segmentación

## ◆ Hardware de interbloqueo

- ◆ Observamos el cauce con la detención y el adelantamiento legal.
- ◆ El adelantamiento a la instrucción AND va ahora a través del banco de registros y no se necesita adelantamiento para la instrucción OR.



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

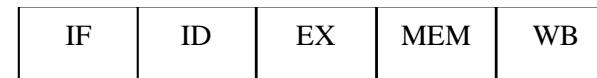
Superescalares

Segmentación

## ◆ Planificación del compilador para los riesgos de datos

- ◆ Muchos tipos de detenciones son muy frecuentes.
- ◆ El patrón general de generación de código para una sentencia del tipo  $A=B+C$  produce una detención para la carga del valor del segundo dato C.
- ◆ El almacenamiento de A no provoca otra detención, ya que el resultado de la suma puede adelantarse a la memoria de datos para que sea usado por la store.

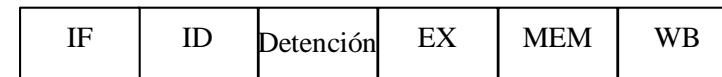
LW R<sub>1</sub>, B



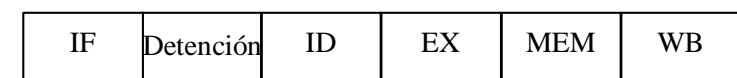
LW R<sub>2</sub>, C



ADD R<sub>3</sub>, R<sub>1</sub>, R<sub>2</sub>



SW A, R<sub>3</sub>



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

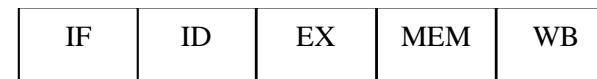
Superescalares

Segmentación

## ◆ Planificación del compilador para los riesgos de datos

- ◆ En lugar de permitir que el cauce se detenga, el compilador puede intentar planificar la segmentación para evitar estas detenciones mediante la reorganización del código para eliminar los riesgos.
- ◆ Por ejemplo, el compilador puede intentar evitar generar código con una instrucción load seguida por el uso inmediato del registro destino de la load. Esta técnica se llama planificación del cauce o planificación de las instrucciones.

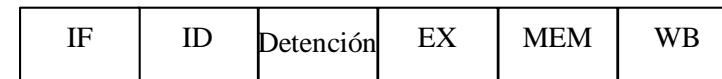
LW R<sub>1</sub>, B



LW R<sub>2</sub>, C



ADD R<sub>3</sub>, R<sub>1</sub>, R<sub>2</sub>



SW A, R<sub>3</sub>



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

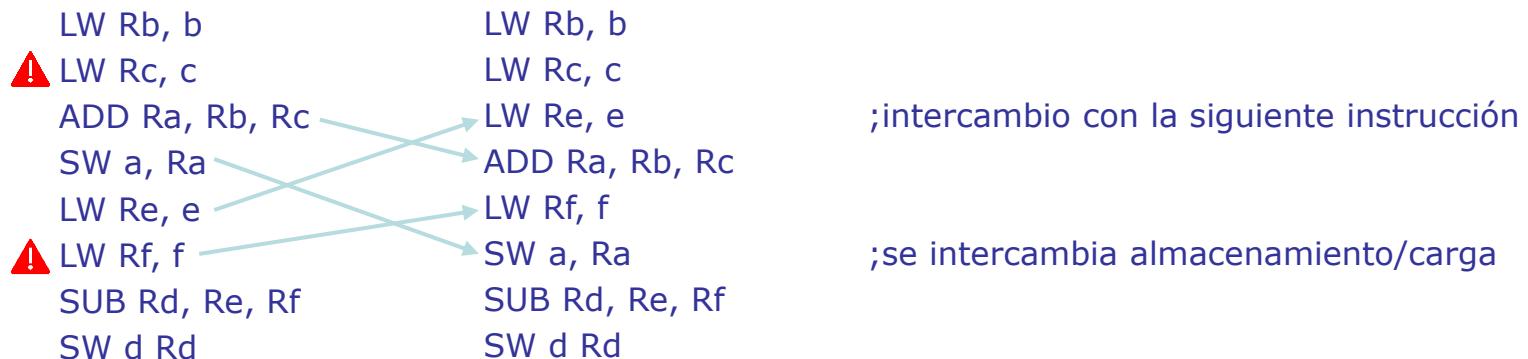
Segmentación

## Ejemplo

Generar código de DLX que evite detenciones del cauce para la siguiente secuencia:

$$a = b + c$$

$$d = e - f$$



Evitamos los interbloqueos:

$$\begin{aligned} \text{LW Rc,c} &\rightarrow \text{ADD Ra, Rb, Rc} \\ \text{LW Rf,f} &\rightarrow \text{SUB Rd, Re, Rf} \end{aligned}$$

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ **Implementación de la detección de riesgos por dependencias de datos**

- ◆ **Emisión:** proceso de evolucionar desde la fase ID a la EX
- ◆ Para DLX todos los riesgos de datos pueden detectarse durante ID
- ◆ En caso de riesgo se detiene la emisión de la instrucción
- ◆ Podemos determinar que adelantamiento será necesario durante ID y fijar el control apropiado

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ **Situaciones detectables por el hardware de detección de riesgos al comparar los destinos y fuentes de instrucciones.**
- ◆ La tabla muestra que la única comparación requerida es entre el destino y las fuentes de las dos instrucciones siguientes

Situación	Secuencia de código ejemplo	Acción
No dependencia	LW $\mathbf{R}_1, 45(\mathbf{R}_2)$ ADD $\mathbf{R}_5, \mathbf{R}_6, \mathbf{R}_7$ SUB $\mathbf{R}_8, \mathbf{R}_6, \mathbf{R}_7$ OR $\mathbf{R}_9, \mathbf{R}_6, \mathbf{R}_7$	No hay riesgo. No existe dependencia sobre $\mathbf{R}_1$ en las tres instrucciones siguientes.
Dependencia que requiere detención	LW $\mathbf{R}_1, 45(\mathbf{R}_2)$ ADD $\mathbf{R}_5, \mathbf{R}_1, \mathbf{R}_7$ SUB $\mathbf{R}_8, \mathbf{R}_6, \mathbf{R}_7$ OR $\mathbf{R}_9, \mathbf{R}_6, \mathbf{R}_7$	Los comparadores detectan el uso de $\mathbf{R}_1$ en ADD y detienen ADD (y SUB y OR) antes que ADD comience EX
Dependencia superada por adelantamiento	LW $\mathbf{R}_1, 45(\mathbf{R}_2)$ ADD $\mathbf{R}_5, \mathbf{R}_6, \mathbf{R}_7$ SUB $\mathbf{R}_8, \mathbf{R}_1, \mathbf{R}_7$ OR $\mathbf{R}_9, \mathbf{R}_6, \mathbf{R}_7$	Los comparadores detectan el uso de $\mathbf{R}_1$ en SUB y adelantan el resultado de la carga a la ALU en el instante en que SUB comienza EX.
Dependencia con accesos en orden	LW $\mathbf{R}_1, 45(\mathbf{R}_2)$ ADD $\mathbf{R}_5, \mathbf{R}_6, \mathbf{R}_7$ SUB $\mathbf{R}_8, \mathbf{R}_6, \mathbf{R}_7$ OR $\mathbf{R}_9, \mathbf{R}_1, \mathbf{R}_7$	No se requiere acción porque la lectura de $\mathbf{R}_1$ por OR se presenta en la segunda mitad de la fase ID, mientras que la escritura del dato cargado se presentó en la primera mitad.

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

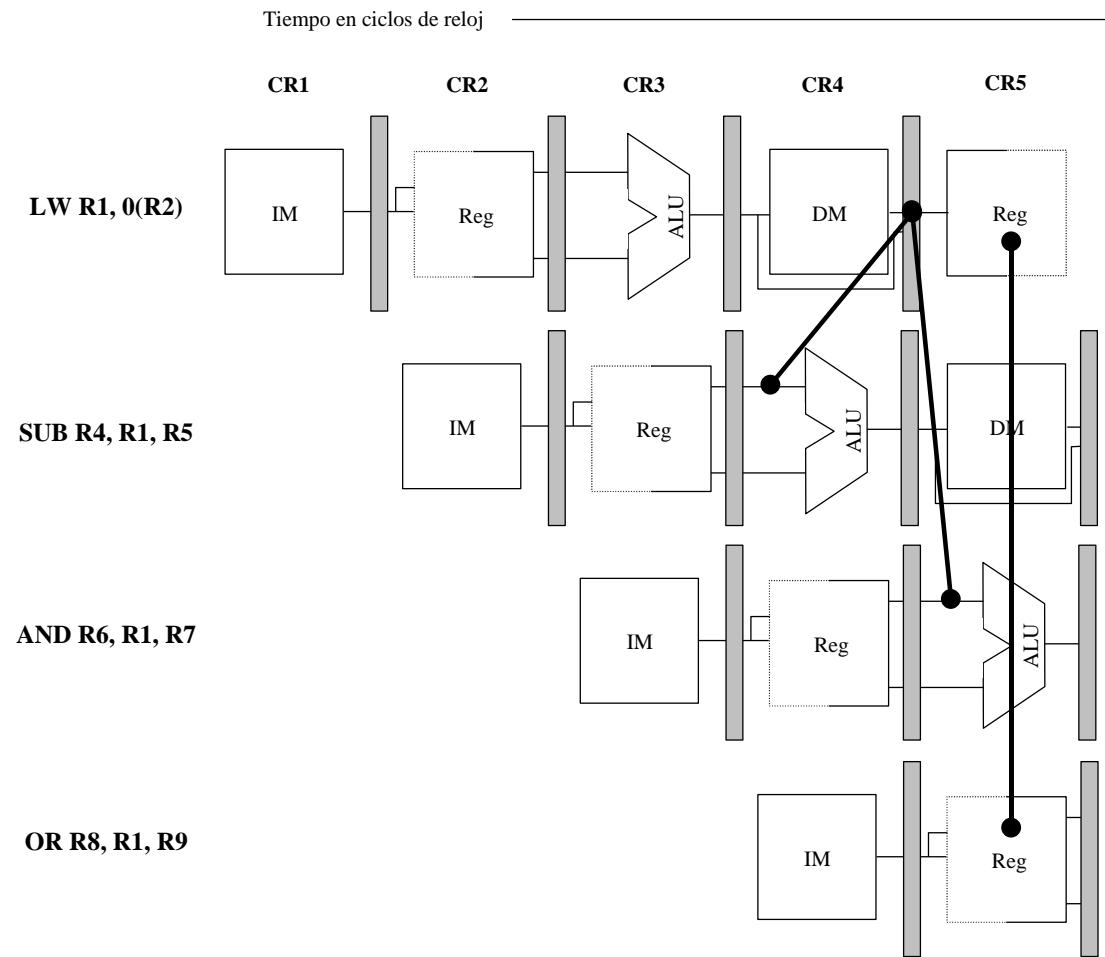
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Implementación del interbloqueo load



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

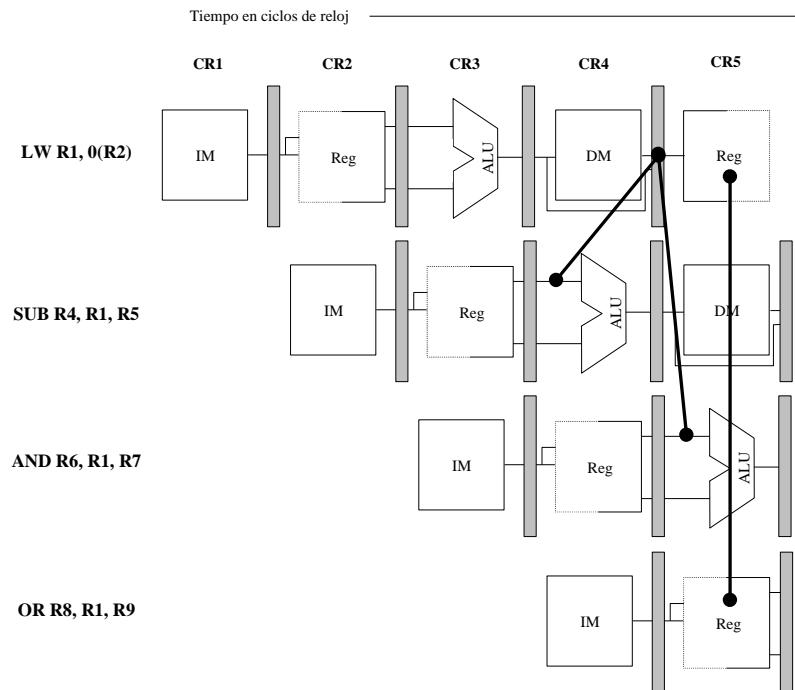
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Implementación del interbloqueo load



Campo código de operación de ID/EX (ID/EX.IR <sub>0..5</sub> )	Campo código de operación de IF/ID (IF/ID.IR <sub>0..5</sub> )	Comparación de campos de operandos
Load	ALU registro registro	ID/EX.IR <sub>11..15</sub> ==IF/ID.IR <sub>6..10</sub>
Load	ALU registro registro	ID/EX.IR <sub>11..15</sub> ==IF/ID.IR <sub>11..15</sub>
Load	Load, store, ALU inm, o salto	ID/EX.IR <sub>11..15</sub> ==IF/ID.IR <sub>6..10</sub>

### Instrucción tipo I

6 5 5 16

Cód ope Rs1 Rd Inmediato

### Instrucción tipo R

6 5 5 5 11

Cód ope Rs1 Rs2 Rd func

### Instrucción tipo J

6 26

Cód ope Desplazamiento añadido al PC

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

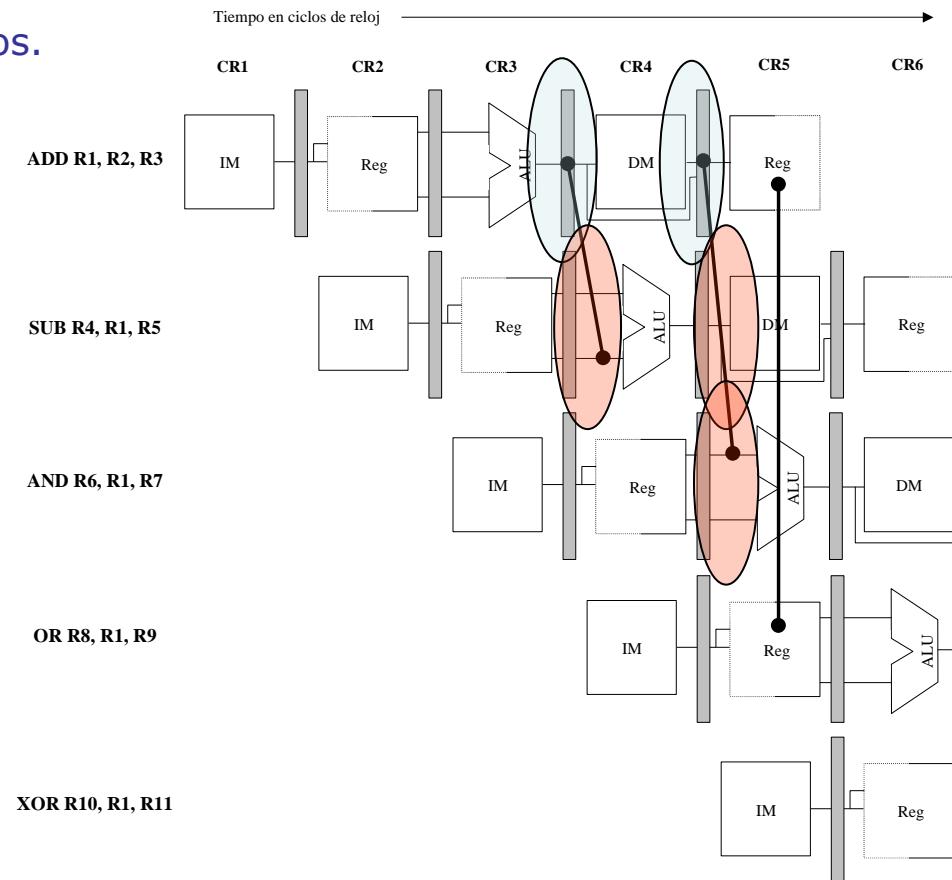
Optimización

Superescalares

Segmentación

## Implementación de la lógica de adelantamiento

- Todo adelantamiento, lógicamente, ocurre:
  - Desde la salida de la ALU o la memoria de datos
  - Hacia la entrada de la ALU, la memoria de datos o la unidad de detección de ceros.



# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

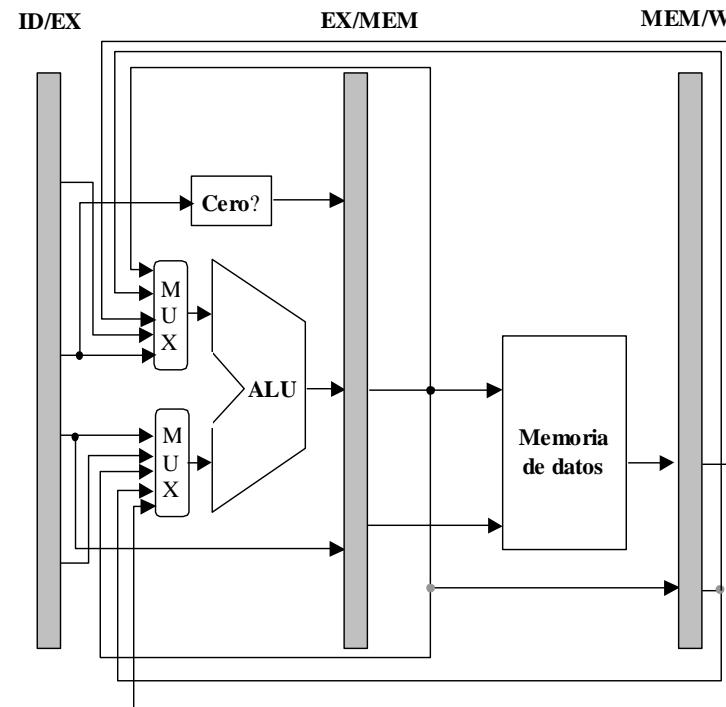
Caucos  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ Adelantar los resultados a la ALU requiere la incorporación de
  - ➏ Tres entradas adicionales en cada multiplexor de la ALU
  - ➏ Tres rutas a las nuevas entradas. Las rutas corresponden al adelantamiento
    - ➏ (1) la salida de la ALU al final de EX
    - ➏ (2) la salida de la ALU al final de la etapa MEM
    - ➏ (3) la salida de la memoria al final de la etapa MEM



# Riesgos por dependencia de datos

Introducción

Segmentación del repertorio

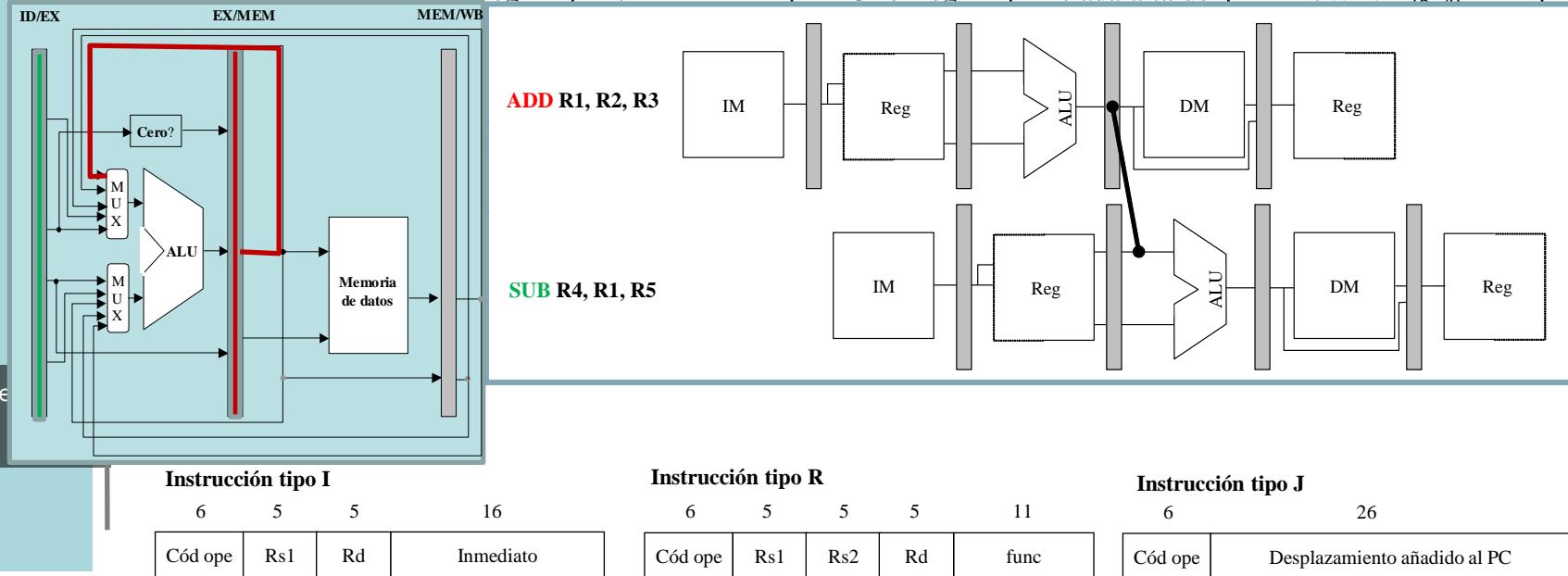
Cauces aritméticos

Optimización

Superescalares

## Implementación de la lógica de adelantamiento

Registro intermedio con la instrucción fuente	Código de operación de la instrucción fuente	Registro intermedio con la instrucción destino	Código de operación de la instrucción destino	Destino del resultado adelantado	Comparación (si igual entonces adelantar)
EX/MEM	ALU rer-reg	ID/EX	ALU rer-reg ALU inm Load, store Branch	Entrada alta de la ALU	EX/MEM.IR <sub>16..20</sub> = ID/EX.IR <sub>6..10</sub>
EX/MEM	ALU rer-reg	ID/EX	ALU rer-reg	Entrada baja de la ALU	EX/MEM.IR <sub>16..20</sub> = ID/EX.IR <sub>11..15</sub>
MEM/WB	ALU rer-reg	ID/EX	ALU rer-reg	Entrada alta de	MEM/WB.IR <sub>16..20</sub> =



# Riesgos por dependencia de datos

## Implementación de la lógica de adelantamiento

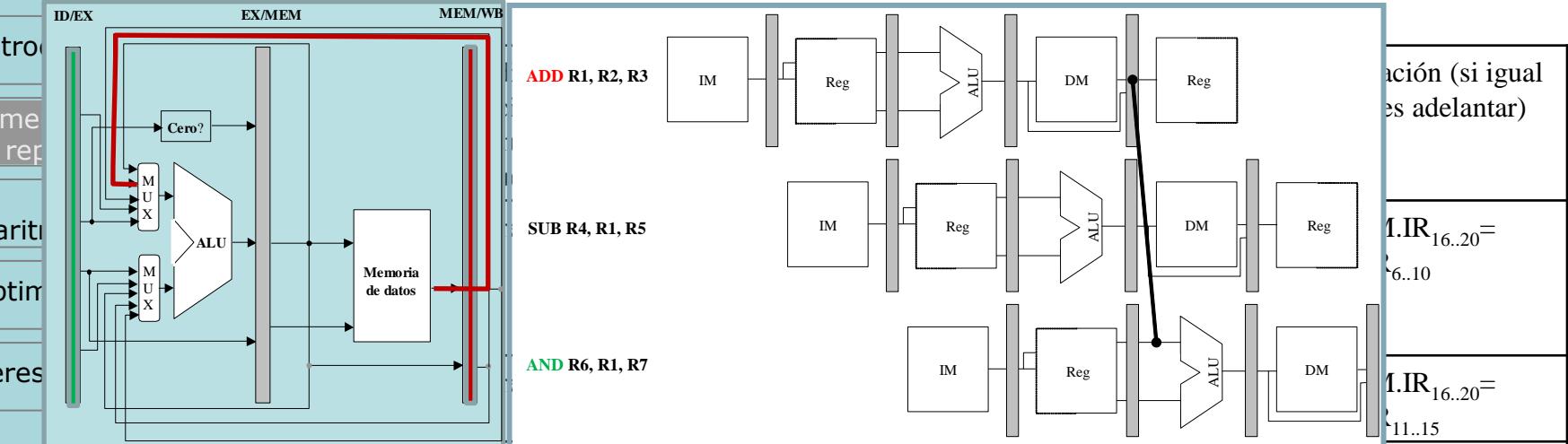
Introducción

Segmentación del espacio de memoria

aritmética

Optimización

Superscalar



	MEM/WB	ALU rer-reg	ID/EX	ALU rer-reg ALU imm Load, store Branch	Entrada alta de la ALU	MEM/WB.IR <sub>16..20</sub> = ID/EX.IR <sub>6..10</sub>
	MEM/WB	ALU rer-reg	ID/EX	ALU rer-reg	Entrada baja de la ALU	MEM/WB.IR <sub>16..20</sub> = ID/EX.IR <sub>11..15</sub>

Segmentación

Instrucción tipo I

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

Instrucción tipo R

6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

Instrucción tipo J

6	26
Cód ope	Desplazamiento añadido al PC

# Riesgos por dependencia de datos

## Implementación de la lógica de adelantamiento

Introducción

Segmentación  
del repertorio

Cuces  
aritméticos

Optimización

Superescalares

Segmentación

Registro intermedio con la instrucción fuente	Código de operación de la instrucción fuente	Registro intermedio con la instrucción destino	Código de operación de la instrucción destino	Destino del resultado adelantado	Comparación (si igual entonces adelantar)
EX/MEM	ALU imm	ID/EX	ALU rer-reg ALU imm Load, store Branch	Entrada alta de la ALU	EX/MEM.IR <sub>11..15</sub> = ID/EX.IR <sub>6..10</sub>
EX/MEM	ALU imm	ID/EX	ALU rer-reg	Entrada baja de la ALU	EX/MEM.IR <sub>11..15</sub> = ID/EX.IR <sub>11..15</sub>
MEM/WB	ALU imm	ID/EX	ALU rer-reg ALU imm Load, store Branch	Entrada alta de la ALU	MEM/WB.IR <sub>11..15</sub> = ID/EX.IR <sub>6..10</sub>
MEM/WB	ALU imm	ID/EX	ALU rer-reg	Entrada baja de la ALU	MEM/WB.IR <sub>11..15</sub> = ID/EX.IR <sub>11..15</sub>

Instrucción tipo I

6      5      5      16

Cód ope	Rs1	Rd	Inmediato
---------	-----	----	-----------

Instrucción tipo R

6      5      5      5      11

Cód ope	Rs1	Rs2	Rd	func
---------	-----	-----	----	------

Instrucción tipo J

6      26

Cód ope	Desplazamiento añadido al PC
---------	------------------------------

# Riesgos por dependencia de datos

Introducción

Segmentación  
del repertorio

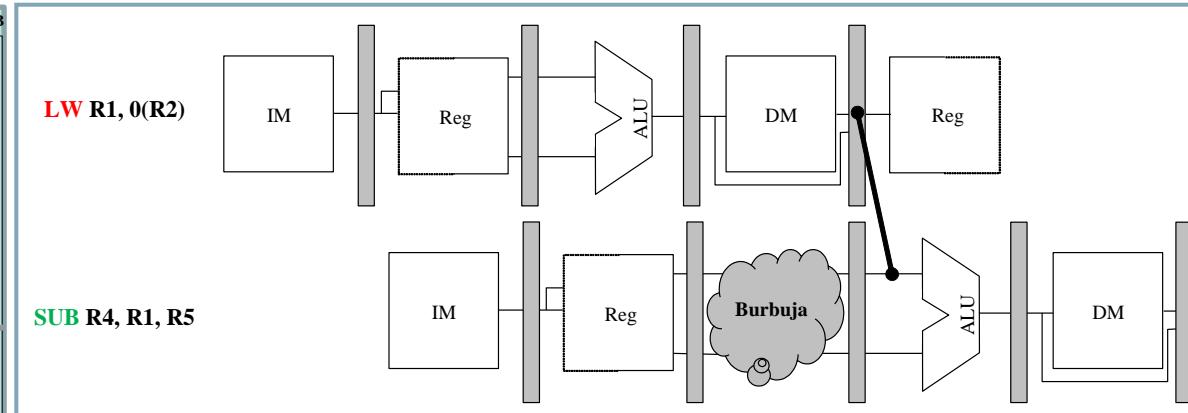
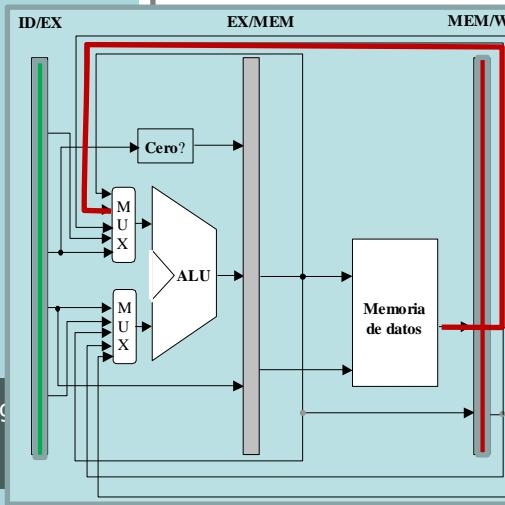
Cauces  
aritméticos

Optimización

Superescalares

## Implementación de la lógica de adelantamiento

Registro intermedio con la instrucción fuente	Código de operación de la instrucción fuente	Registro intermedio con la instrucción destino	Código de operación de la instrucción destino	Destino del resultado adelantado	Comparación (si igual entonces adelantar)
MEM/WB	Load	ID/EX	ALU rer-reg ALU imm Load, store Branch	Entrada alta de la ALU	MEM/WB.IR <sub>11..15</sub> = ID/EX.IR <sub>6..10</sub>
MEM/WB	Load	ID/EX	ALU rer-reg	Entrada baja de la ALU	MEM/WB.IR <sub>11..15</sub> = ID/EX.IR <sub>11..15</sub>



Instrucción tipo R

16

Cód ope | Rs1 | Rd | Inmediato

6 5 5 11

Cód ope | Rs1 | Rs2 | Rd | func

Instrucción tipo J

26

Cód ope | Desplazamiento añadido al PC

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

- ➊ Pueden provocar mayor pérdida de rendimiento que los riesgos por dependencia de datos.
- ➋ Cuando se ejecuta un salto pueden ocurrir dos cosas, en función de la evaluación de la condición de salto (realizada en fase EX):
  - ➌ **El salto es efectivo:** El PC cambia su valor por una dirección calculada por la ALU al final de MEM, tras calcular la dirección y la comparación
$$\text{ALUoutput} \leftarrow \text{PC} + ((\text{IR}_{16})^{16}\# \#\text{IR}_{16..31})$$
  - ➍ **El salto no es efectivo:** El PC cambia su valor por PC+4

Segmentación

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

- ◆ **El método más simple:** detener el cauce tan pronto como detectemos el salto antes de alcanzar la etapa MEM, que determina el nuevo PC.

Instrucción de salto	IF	ID	EX	MEM	WB				
Instrucción i+1	Detención	Detención	Detención	IF	ID	EX	MEM	WB	
Instrucción i+2	Detención	Detención	Detención	IF	ID	EX	MEM	WB	
Instrucción i+3	Detención	Detención	Detención	IF	ID	EX	MEM		
Instrucción i+4	Detención	Detención	Detención	IF	ID	EX			
Instrucción i+5	Detención	Detención	Detención	IF	ID				
Instrucción i+6	Detención	Detención	Detención	IF					

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

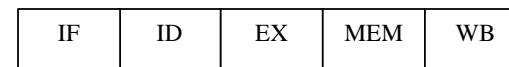
Optimización

Superescalares

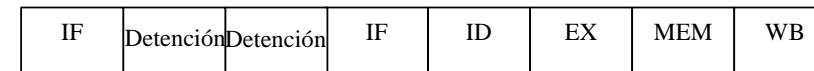
Segmentación

- La detención no ocurre hasta después de la etapa ID (no queremos parar el cauce hasta que sepamos que la instrucción es un salto).

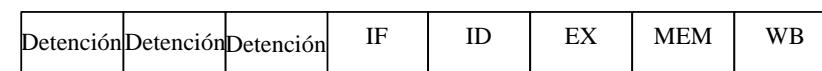
Instrucción de salto



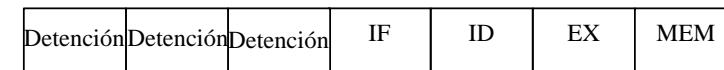
Instrucción i+1



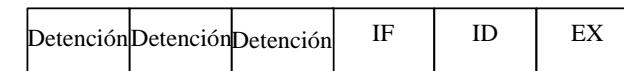
Instrucción i+2



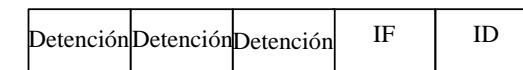
Instrucción i+3



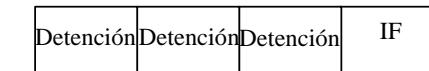
Instrucción i+4



Instrucción i+5



Instrucción i+6



- El ciclo IF de la instrucción siguiente al salto debe repetirse en cuanto conocemos el resultado del salto. Por eso, el primer ciclo IF es esencialmente una detención, porque nunca realiza trabajo útil.

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

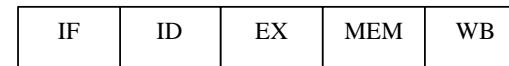
Optimización

Superescalares

Segmentación

- La detención no ocurre hasta después de la etapa ID (no queremos parar el cauce hasta que sepamos que la instrucción es un salto).

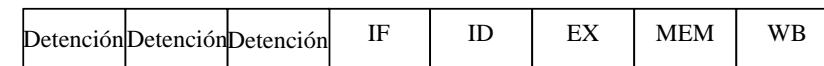
Instrucción de salto



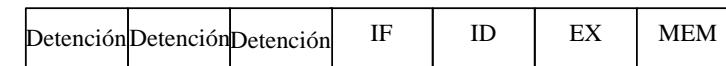
Instrucción i+1



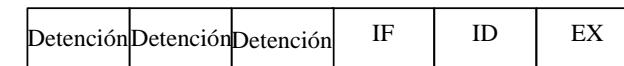
Instrucción i+2



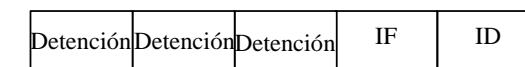
Instrucción i+3



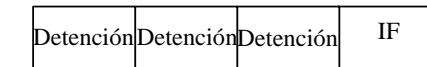
Instrucción i+4



Instrucción i+5



Instrucción i+6



- Un salto causa una detención de tres ciclos en el cauce MIPS: Un ciclo corresponde la repetición de IF y dos ciclos están inactivos.

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

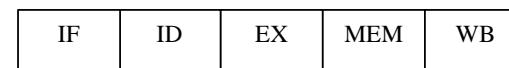
Optimización

Superescalares

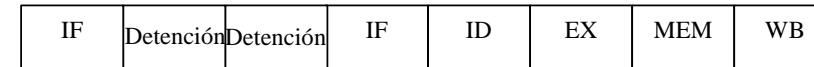
Segmentación

- ◆ La detención no ocurre hasta después de la etapa ID (no queremos parar el cauce hasta que sepamos que la instrucción es un salto).

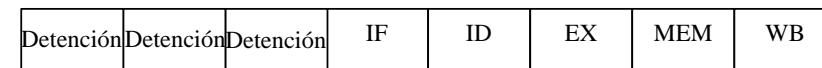
Instrucción de salto



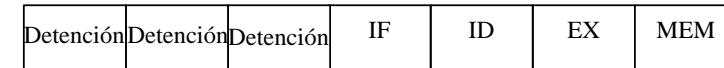
Instrucción i+1



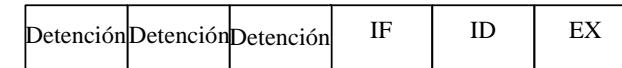
Instrucción i+2



Instrucción i+3



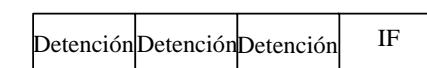
Instrucción i+4



Instrucción i+5



Instrucción i+6



- ◆ Esta detención puede implementarse fijando el registro IF/ID a cero en los tres ciclos. Debemos resaltar que si el salto no es efectivo, entonces la repetición de la etapa IF es innecesaria.

# Riesgos de control

## Ejemplo

Supongamos una frecuencia de salto de un 30% y un CPI ideal de 1, ¿Qué relación de velocidad encontramos entre la máquina segmentada ideal y la máquina con detenciones por saltos?.

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

# Riesgos de control

## Ejemplo

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

Supongamos una frecuencia de salto de un 30% y un CPI ideal de 1,  
¿Qué relación de velocidad encontramos entre la máquina segmentada  
ideal y la máquina con detenciones por saltos?.

$$\text{CPI}_{\text{ideal}} = 1$$

$$\text{CPI}_{\text{máquina con detenciones por saltos}} = 0.3*4+0.7*1= 1.2 + 0.7 = 1.9$$

$$\text{Relación de rendimiento} = 1.9/1$$

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ **Reducción del número de ciclos de detención de salto**
  - ➌ La relación de rendimiento entre la máquina ideal y la máquina con detenciones por saltos es 1.9, por lo tanto la reducción del número de detenciones es crítica
- ➋ **Dos pasos** para la reducción del número de detenciones por salto:
  - ➌ 1. Evaluar la condición lo antes posible en el cauce
  - ➌ 2. Calcular el PC efectivo lo antes posible en el cauce
- ➌ Para optimizar el comportamiento del salto, deben realizarse ambas cosas

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ En MIPS los saltos (BEQZ y BNEZ) requieren comprobar si un registro es igual a cero.
- ➋ Es posible completar esta decisión al final del ciclo ID moviendo el test cero a este ciclo.
- ➌ Para aprovechar este adelantamiento de la decisión de salto, ambos PCs (efectivo y no efectivo) deben calcularse previamente.
- ➍ Computar el destino del salto durante ID requiere un sumador adicional porque la ALU, no está disponible hasta EXE.

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

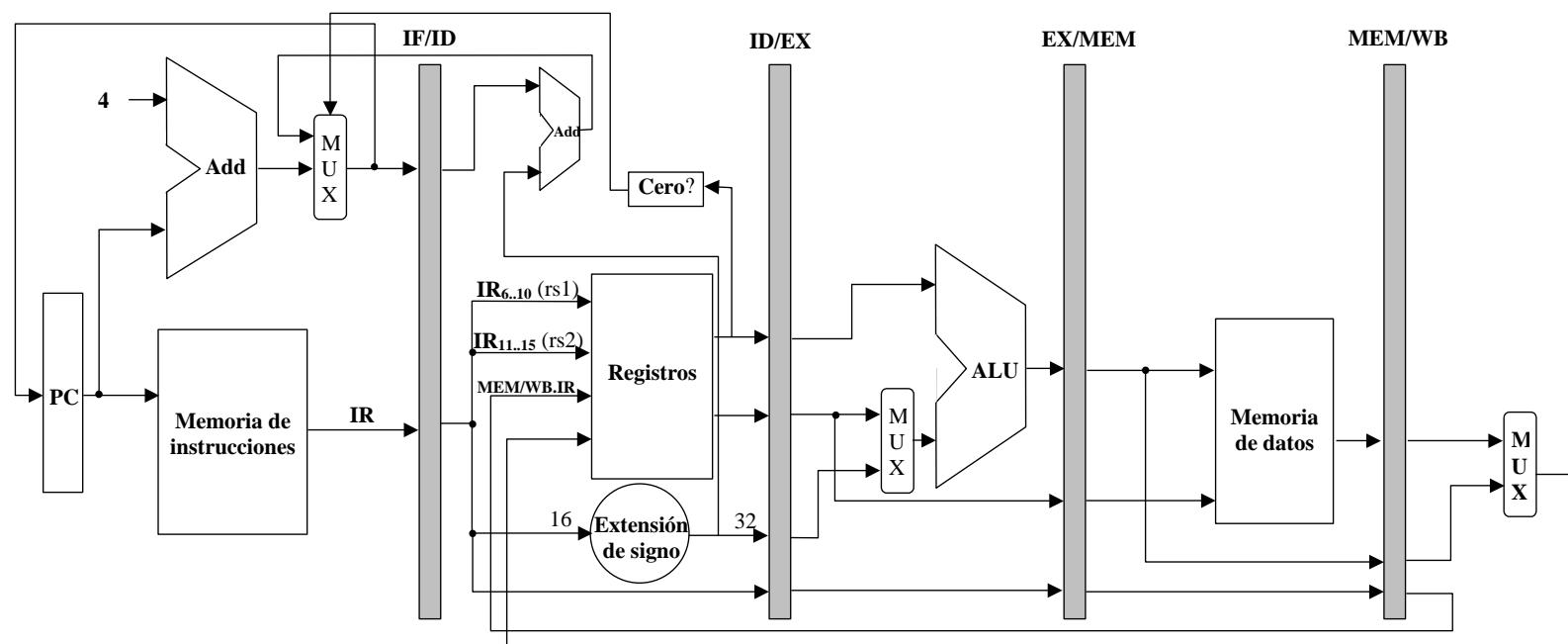
Optimización

Superescalares

Segmentación

## Riesgos de control

- Estructura revisada de la segmentación, se muestra el uso de un sumador separado para calcular la dirección destino del salto:



# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

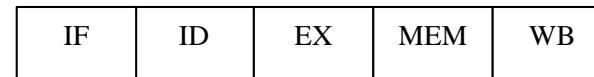
Optimización

Superescalares

Segmentación

- ◆ En estas condiciones sólo es necesario un ciclo de detención en los saltos

Instrucción de salto



Instrucción i+1



Instrucción i+2



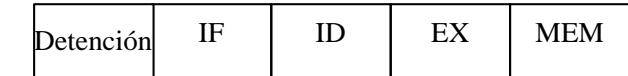
Instrucción i+3



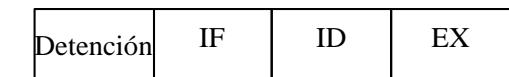
Instrucción i+4



Instrucción i+5



Instrucción i+6



# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

- ◆ Tanto el cálculo de la dirección de salto como la evaluación de la condición se realiza para todas las instrucciones.
- ◆ Sólo se sustituye el PC en caso de evaluación positiva de la condición y de que la instrucción sea de salto.

Etapa	Cualquier instrucción
IF	$IF/ID.IR \leftarrow Mem[PC];$ $IF/ID.NPC, PC \leftarrow ( \text{if } (Regs[IF/ID.IR}_{6..10}]) \text{ op } 0 )$ $\{ IF/ID.NPC + (IF/ID.IR}_{16})^{16} \# IF/ID.IR_{16..31} \}$ $\text{else } \{ PC+4 \});$
ID	$ID/EX.A \leftarrow Regs [IF/ID.IR}_{6..10}; ID/EX.B \leftarrow Regs [IF/ID.IR}_{11..15};$ $ID/EX.IR \leftarrow IF/ID.IR$ $ID/EX.Inm \leftarrow (IF/ID.IR}_{16})^{16} \# IF/ID.IR_{16..31}$

Segmentación

- ◆ En general, cuanto más profundo es el cauce, peor es la penalización en ciclos de retardo para los saltos.

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Reducción de las penalizaciones de saltos en la segmentación

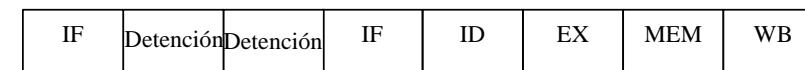
### Congelación de la segmentación:

- El esquema más sencillo es detener todas las instrucciones después del salto, hasta conocer el destino correcto. La sencillez del esquema es su principal atractivo.

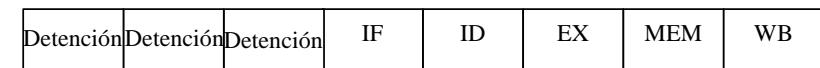
Instrucción de salto



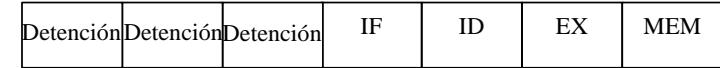
Instrucción i+1



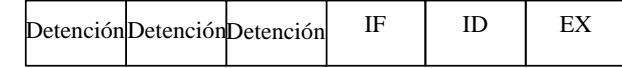
Instrucción i+2



Instrucción i+3



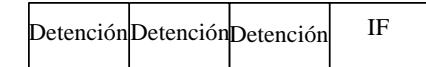
Instrucción i+4



Instrucción i+5



Instrucción i+6



# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- **Reducción de las penalizaciones de saltos en la segmentación**
  - **Predecir el salto como no efectivo**
  - En este esquema permitimos que el hardware continúe como si el salto no se ejecutase.
  - La complejidad radica en no cambiar el estado de la máquina hasta que no se conozca el resultado del salto. Esto supone el conocimiento de cuando una instrucción cambia el estado y como deshacer el cambio.
  - Se implementa continuando la búsqueda de instrucciones como si no ocurriese nada extraordinario. Si el salto es efectivo detenemos la segmentación, recomendamos las búsquedas y deshacemos los cambios de estado (una posibilidad simple es limpiar la segmentación).

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

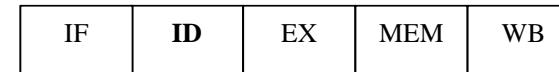
Superescalares

Segmentación

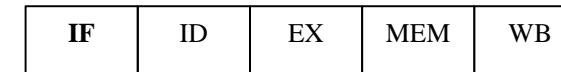
## ◆ Reducción de las penalizaciones de saltos en la segmentación

- ◆ Predecir el salto como no efectivo
- ◆ Cuando el salto no es efectivo: Se determina en ID, simplemente continuamos

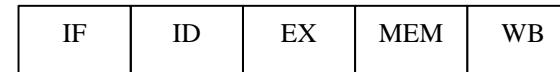
Instrucción de salto no efectivo



Instrucción i+1



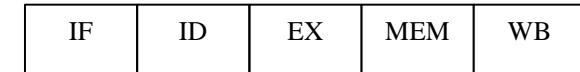
Instrucción i+2



Instrucción i+3



Instrucción i+4



# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

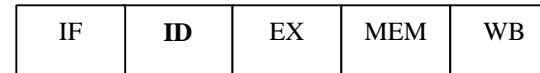
Superescalares

## Reducción de las penalizaciones de saltos en la segmentación

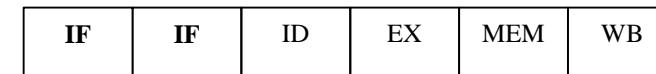
### Predecir el salto como no efectivo

- When the jump is effective during ID, we resume the search at the destination of the jump. This makes all instructions that follow the jump stop for one clock cycle.

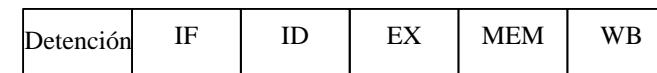
Instrucción de salto efectivo



Instrucción i+1



Instrucción i+2



Instrucción i+3



Instrucción i+4



Segmentación

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Reducción de las penalizaciones de saltos en la segmentación

- ◆ **Predecir el salto como efectivo**
- ◆ Una vez decodificado el salto y calculada la dirección destino, suponemos que el salto se va a realizar y comenzamos la búsqueda y ejecución en el destino.
- ◆ En MIPS no se conoce la dirección destino antes de conocer la evaluación del salto, por lo tanto esta estrategia no es útil.
- ◆ En máquinas con códigos de condición más potentes (más lentas) el destino del salto se conoce antes que la evaluación del salto, es en esta situación donde el esquema tiene sentido.

# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Reducción de las penalizaciones de saltos en la segmentación

### Saltos retardados

- ➊ Ejecutar instrucciones **independientes del salto** durante los ciclos de retardo (delay-slot)
- ➋ Estas instrucciones se ejecutarán **siempre** (tanto si el salto es tomado como si no)
- ➌ Trabajo del compilador: hacer que las instrucciones sean válidas y útiles
- ➍ El número de instrucciones situadas a continuación de un salto que se ejecutan puede variar con la arquitectura

# Riesgos de control

Introducción

Segmentación  
del repertorio

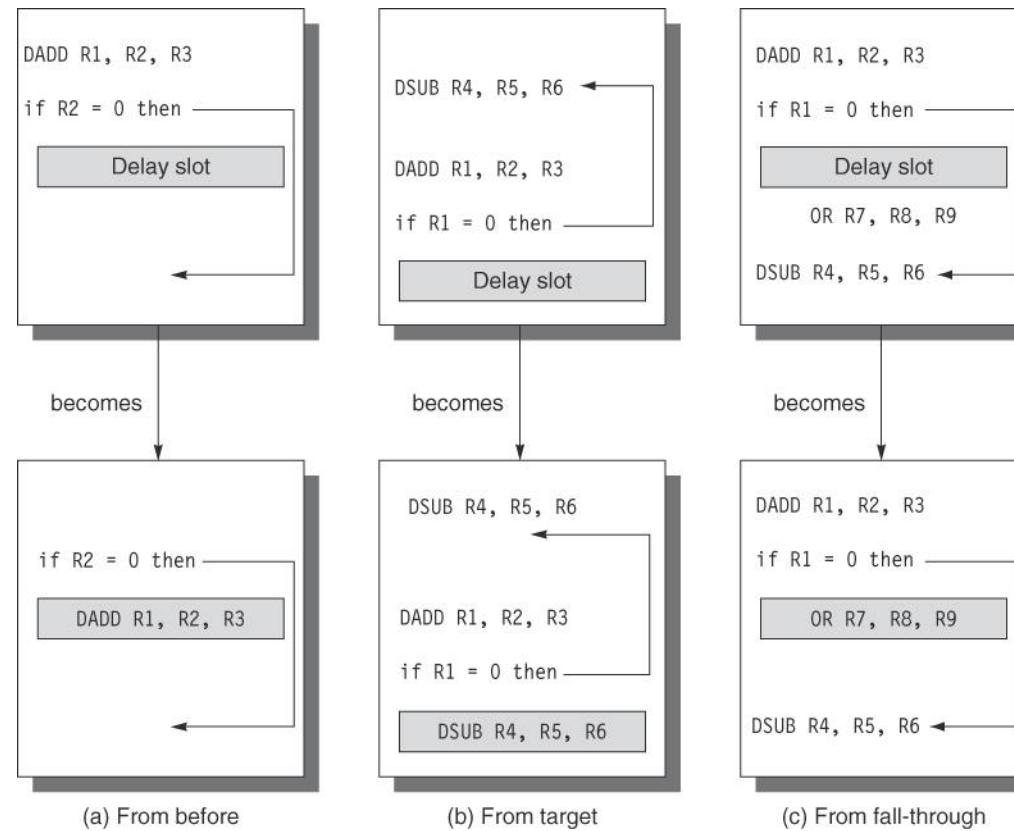
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Planificación del salto retardado



# Riesgos de control

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## Planificación del salto retardado

- ◆ **Si las instrucciones se toman de antes del salto:**
  - ◆ no debe haber dependencia entre ellas y el salto
  - ◆ siempre mejora el rendimiento
- ◆ **Si las instrucciones se toman del destino del salto:**
  - ◆ no debe afectar al programa el que se ejecuten si el salto no se toma
  - ◆ puede haber ocasiones en las que el código no se copie, sino que se duplique (al destino se llega desde varios puntos)
  - ◆ mejora el rendimiento si el salto se toma
- ◆ **Si las instrucciones se toman a continuación del salto:**
  - ◆ no debe afectar al programa el que se ejecuten si el salto se toma
  - ◆ mejora el rendimiento si el salto no se toma

**Tienen bastante utilidad en cauces sencillos, pero en cauces más complejos su efectividad es poca**

# Ejemplo: MIPS R4000

Introducción

Segmentación  
del repertorio

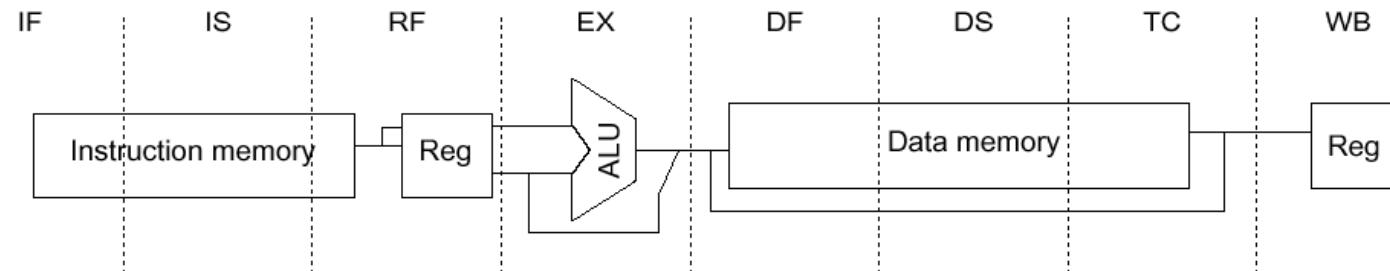
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ Representativo de arquitecturas que aparecieron a partir de los 80
- ◆ Máquina de 64 bits
- ◆ Implementa el repertorio MIPS-3
- ◆ Cauce con 8 etapas (supersegmentación)
- ◆ Esto le permitía utilizar una frecuencia alta (100-200 MHz)
- ◆ Utilizado por NEC, Nintendo, Silicon Graphics, Sony...



# Ejemplo: MIPS R4000

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

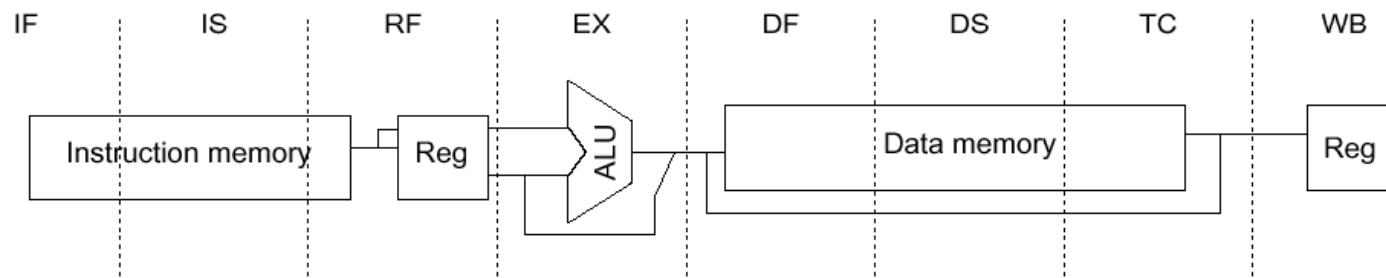
Optimización

Superescalares

Segmentación

- ➊ Segmentación: 8 etapas
- ➋ IF: Primera mitad de búsqueda de instrucción. Inicia el acceso a cache de instrucciones. Actualización del PC.
- ➌ IS: Segunda mitad de búsqueda de instrucción. Completa el acceso a cache.
- ➍ RF: Comprobación de acierto en cache. Decodificación y búsqueda de registros. Chequeo de riesgos.
- ➎ EX: Ejecución; operación de la ALU, calculo de dirección efectiva, evaluación de condición de salto y cálculo de destino (puede durar varios ciclos)
- ➏ DF: Inicio de la búsqueda de datos
- ➐ DS: Segunda mitad de la búsqueda de datos
- ➑ TC: Chequeo de etiquetas, determinación de acierto en la cache
- ➒ WB: Postescritura

Los datos se encuentran disponibles al final de DS, aunque no se ha comprobado el acierto en la cache



# Ejemplo: MIPS R4300

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ Procesador de 64 bits utilizado para aplicaciones empotradas
- ◆ La segmentación del R4300 implementa el punto flotante mediante la extensión de la longitud del pipeline a través de la adición de múltiples etapas EX para las operaciones de punto flotante. Esto introduce una complejidad: la posibilidad de completar instrucciones fuera de orden. Una instrucción entera puede completarse y escribir su resultado antes de que termine una instrucción en punto flotante.
- ◆ Cuatro versiones:

Clock frequency (MHz)	Performance SPECInt 92	Performance SPECFP 92	Power consumption (W)
80	48	36	1.5
100	60	45	1.8
133	80	60	1.9
167	100	75	2.4

# Ejemplo: Power PC 604

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ El PowerPC 604 se introdujo en diciembre de 1994.
- ➋ El 604 es un procesador superescalar capaz de emitir cuatro instrucciones simultáneamente ( 3 enteras y una en coma flotante). El 604 cuenta con un cauce de 6 etapas y con seis unidades de ejecución (dos para operaciones enteras, dos para operaciones en coma flotante y dos para procesamiento de saltos) que pueden trabajar en paralelo, puede terminar hasta seis instrucciones en cada ciclo
- ➌ El PowerPC 604 contiene 3,6 millones de transistores y fue fabricado por IBM y Motorola. Funcionan a velocidades de entre 100 y 180 MHz.

# Ejemplo: Pentium Pro

Introducción

Segmentación  
del repertorio

Cáculos  
aritméticos

Optimización

Superescalares

Segmentación

- ➊ El Pentium Pro se introdujo en 1995.
- ➋ El Pentium Pro es un procesador superescalar capaz de emitir tres instrucciones simultáneamente. El Pentium Pro cuenta con una pipeline de 6 etapas y con seis unidades de ejecución que pueden trabajar en paralelo, puede terminar hasta 5 instrucciones en cada ciclo.
- ➌ El Pentium Pro en vez de segmentar las instrucciones de tamaño variable 80x86, lo que hace es que la unidad de decodificación del Pentium Pro traduce las instrucciones Intel a microoperaciones de 72 bits de longitud fija y envía estas microoperaciones al buffer de reordenación y a las estaciones de reserva.

# PowerPC 604 y Pentium Pro

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

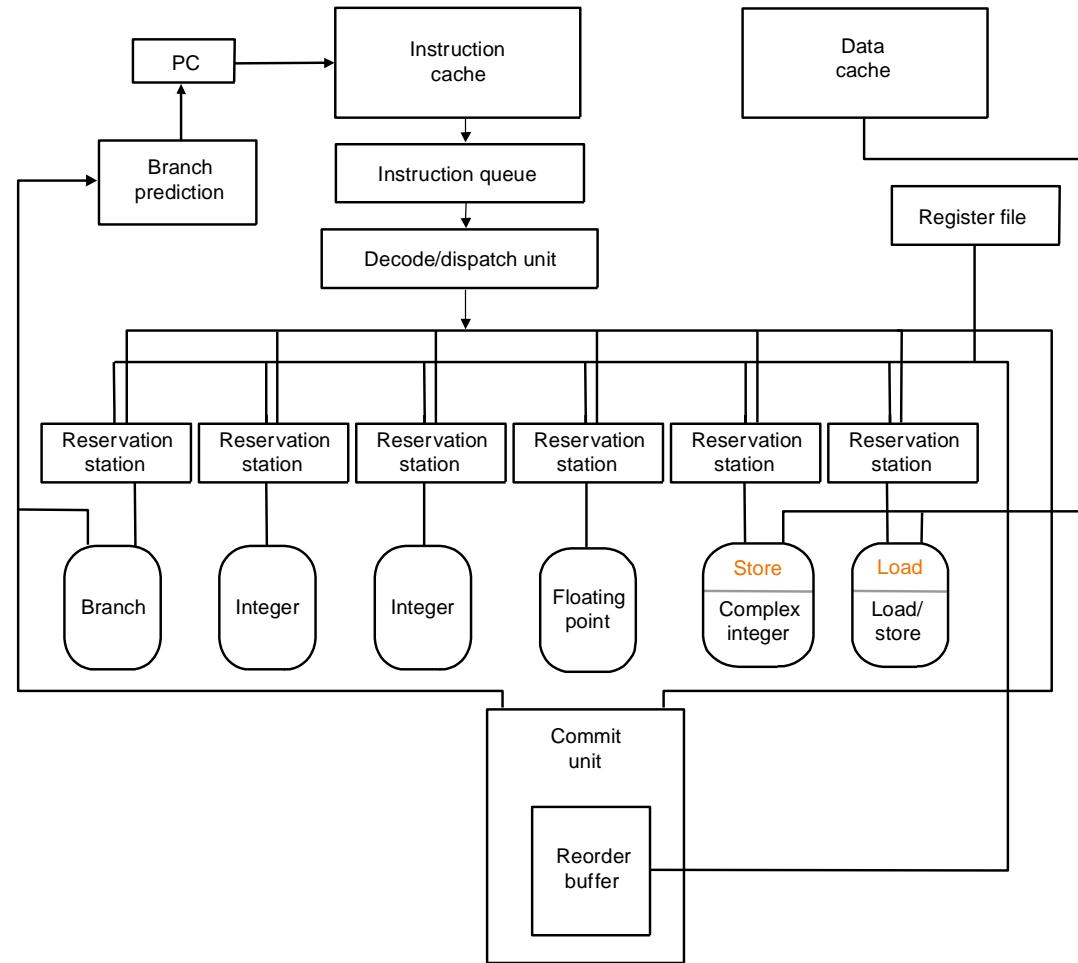
Optimización

Superescalares

Segmentación

## ◆ Organización genérica del pipeline de los proceadores

### Pentium pro y Power PC 604



# PowerPC 604 y Pentium Pro

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- La cache de instrucciones busca 16 bytes de instrucciones y los envía a una cola de instrucciones; cuatro para el PowerPC y un número de instrucciones variables para el Pentium.
- Después unas cuantas de ellas se decodifican
- Ambos utilizan una tabla de historia de saltos de 512 entradas para predecir los saltos.
- La unidad de envío manda cada instrucción y sus operandos a la estación de reserva de una de las seis unidades funcionales. Esta unidad también reserva una entrada en el buffer de reordenación de la unidad de commit para esta instrucción.

# Ejemplo: Pentium 4

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

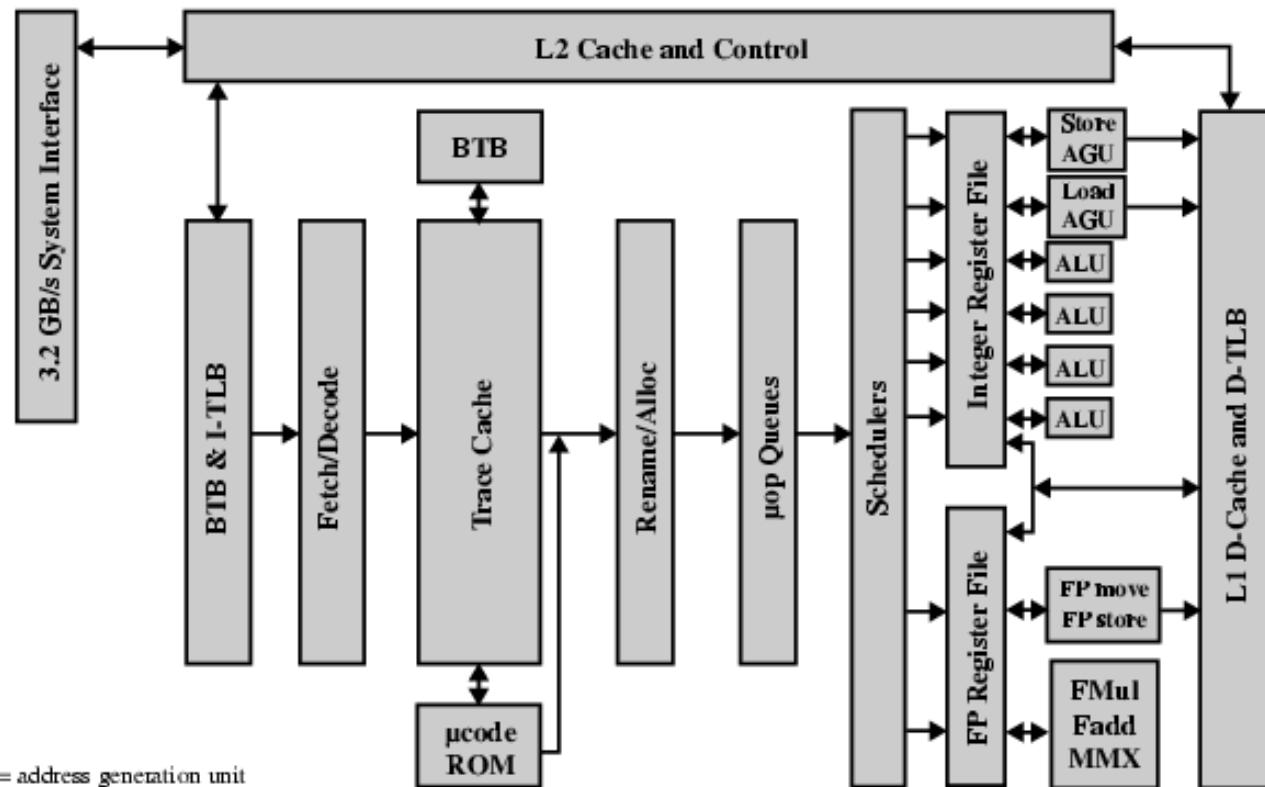
Superescalares

Segmentación

- ➊ Pentium 4
- ➋ Introducido en el 2001 con ciclo de reloj de más de 1GHz
- ➌ El cauce del Pentium 4 tiene 20 etapas
- ➍ Tiene registros XMM (extended multi media) de 128 bits para dos operaciones de coma flotante empaquetadas de 64bits cada una
- ➎ Los registros de 128 bits manejan también enteros grandes
- ➏ Está equipado con un conjunto operaciones SIMD
- ➐ Revisions
  - ➑ Northwood (1/2002) – 21 etapas
  - ➒ Prescott (2/2004) – 31 etapas

# Ejemplo: Pentium 4

## Diagrama de bloques del Pentium 4



AGU = address generation unit

BTB = branch target buffer

D-TLB = data translation lookaside buffer

I-TLB = instruction translation lookaside buffer

Introducción

Segmentación del repertorio

Cauces aritméticos

Optimización

Superescalares

Segmentación

# Ejemplo: Pentium 4

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ◆ Operación del Pentium 4:

- ◆ Busca instrucciones de memoria en orden
- ◆ Traduce las instrucciones en una o mas instrucciones RISC de longitud fija (micro operaciones)
- ◆ Ejecuta las micro operaciones en un pipeline superescalar.
  - ◆ Las micro operaciones pueden ejecutarse fuera de orden
- ◆ Almacena los resultados de las micro operaciones en el conjunto de registros en el orden del flujo del programa original.
- ◆ Capa externa CISC con núcleo interno RISC
- ◆ El cauce del núcleo interno RISC tienen al menos 20 etapas
  - ◆ Algunas micro operaciones requieren múltiples etapas de ejecución (pipeline más larga)

# Ejemplo: Procesadores ARM

Introducción

Segmentación  
del repertorio

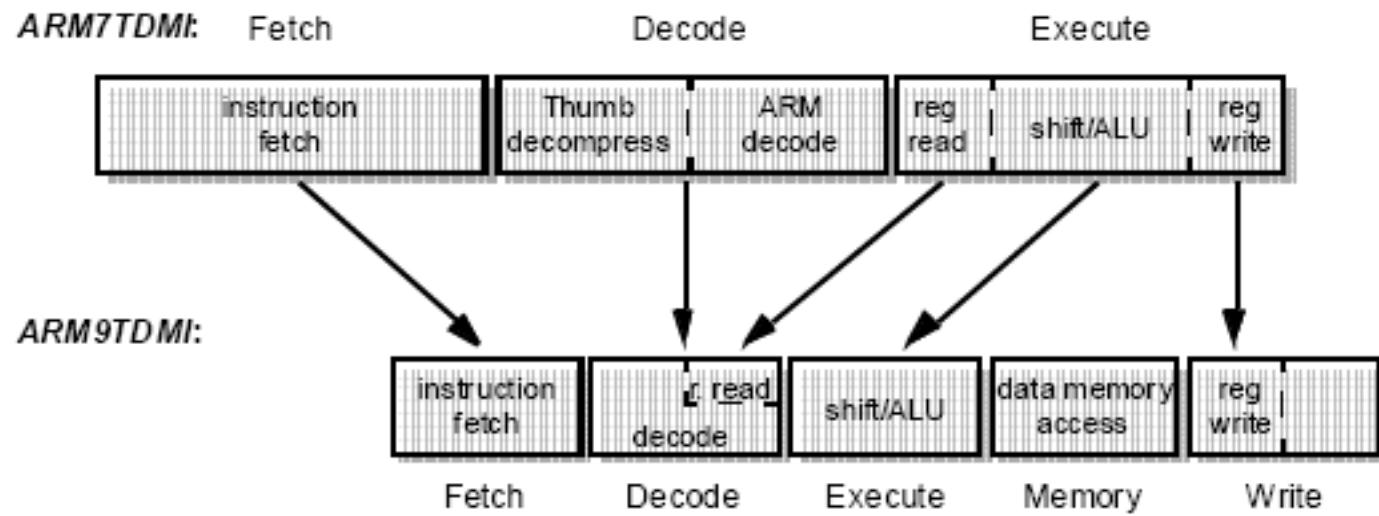
Cauces  
aritméticos

Optimización

Superescalares

Segmentación

- ◆ **ARM:** Advances RISC Machines
- ◆ La familia de procesadores ARM se dirige fundamentalmente al mercado de los sistemas embebidos
- ◆ El primer ARM se desarrolló en Inglaterra entre 1983 y 1985
- ◆ Han aparecido 5 versiones del repertorio de instrucciones ARM
- ◆ Hasta la versión ARM7 inclusive se implementaban con una cauce de 3 etapas, después pasaron a 5 etapas. Posteriormente se introdujo un cauce de 6 etapas para poder utilizar ciclos de reloj más pequeños.



# Ejemplo: Procesadores ARM

Introducción

Segmentación  
del repertorio

Cauces  
aritméticos

Optimización

Superescalares

Segmentación

## ARM7: Lanzamiento en 1994

- ◆ Objetivos: Teléfonos móviles, agendas, impresoras, cámaras, PDAs,
- ◆ ARM7TDMI(-S) :Pipeline de 3 etapas.
- ◆ ARM7EJ-S Pipeline de 5 etapas.

## ARM9: Lanzamiento en 1997

- ◆ Objetivos: Teléfonos móviles, buscas, smartphones, decodificadores de TV,
- ◆ ARM946E-S (1999) Pipeline de 5 etapas.

## ARM11 : lanzamiento en 2000

- ◆ Objetivos: Cámaras digitales, smartphones, e-book readers, media centers, ...
- ◆ ARM1136J(F)-S (2002) Pipeline de 8 etapas.

# Tema 5. Rendimiento de la jerarquía de memoria

Arquitectura de los Computadores

# Rendimiento de la jerarquía de memoria

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## Objetivos

- ➊ Analizar conceptos sobre la jerarquía de memoria
- ➋ Reflexionar sobre la optimización de los accesos a memoria

# Rendimiento de la jerarquía de memoria

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## Contenido

- ◆ **5.1. Introducción**
- ◆ **5.2. Jerarquía de memoria**
- ◆ **5.3. Memoria caché**
- ◆ **5.4. Mejoras del rendimiento de la memoria principal**
- ◆ **5.5. Memoria Virtual**

## **5.1. Introducción**

---

**Tema 5 Rendimiento de la jerarquía de memoria**

**Arquitectura de los Computadores**

# Introducción

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## INTRODUCCIÓN

- 5.1.1 Principio de localidad

# Introducción

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Los usuarios desean memorias grandes, rápidas y baratas.  
Sin embargo:
  - ➊ a menor tiempo de acceso, mayor coste por bit
  - ➋ a mayor capacidad, menor coste por bit
  - ➌ a mayor capacidad, mayor tiempo de acceso
- ➋ Es necesario equilibrar el gasto, el tamaño y la velocidad de las memorias utilizadas.
- ➌ Se utilizan diferentes tipos de memoria.

# Principio de localidad

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ➊ Principio de localidad:

- ➊ Los programas acceden a una porción relativamente pequeña del espacio de direcciones en cualquier instante de tiempo.

## ➋ Dos dimensiones:

- ➊ **Localidad temporal:** si se referencia un elemento, tenderá a ser referenciado pronto
- ➊ **Localidad espacial:** los elementos cercanos al elemento referenciado tenderán a ser referenciados pronto

## **5.2. Jerarquía de memoria**

---

**Tema 5 Rendimiento de la jerarquía de memoria**

**Arquitectura de los Computadores**

# Jerarquía de memoria

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## JERARQUÍA DE MEMORIA

- 5.2.1 Definiciones

# Jerarquía de memoria

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria



- Según se desciende en la jerarquía:
  - disminuye el coste por bit
  - aumenta la capacidad
  - aumenta el tiempo de acceso
  - disminuye la frecuencia de accesos por parte de la CPU
- La información se ubica en un nivel dependiendo de su probabilidad de uso.
- Como ésta varía durante la ejecución de un programa, se produce un **tráfico continuo** de información entre los niveles.

# Definiciones (I)

Introducción

Jerarquía de memoria

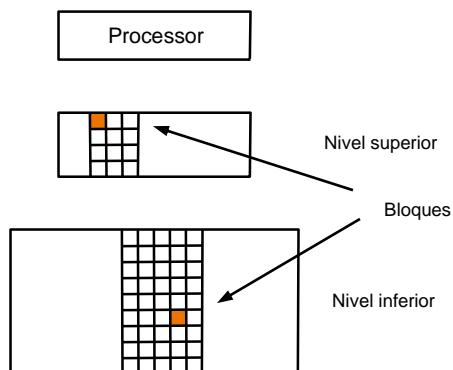
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Una jerarquía de memoria consta de varios **niveles**, pero en cada momento se gestiona entre dos niveles.
  - ➌ **Nivel superior**: el más cercano a la CPU, corresponde a la memoria más rápida, pequeña y cara
  - ➌ **Nivel inferior**: el más alejado a la CPU, corresponde a la memoria más lenta, grande y barata.
- ➋ Todos los datos del nivel superior se encuentran también el nivel inferior.
- ⌁ **Bloque**: la mínima unidad de información que se puede referenciar en la jerarquía de dos niveles.
  - ➌ Está compuesto de palabras de memoria



# Definiciones (II)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Un **acuerdo** (hit) es un acceso con éxito a la memoria del nivel superior, en caso contrario se produce un **fallo** (miss).
- ➋ La frecuencia o tasa de aciertos (hit ratio) es el porcentaje de aciertos en accesos a memoria en el nivel superior.
- ➌ La tasa de fallos (miss ratio) se define como:  
$$1 - \text{tasa de aciertos}$$

Principal razón de la jerarquía de memoria: rendimiento



La velocidad de aciertos y fallos es importante

# Definiciones (III)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

◆ El **tiempo de acierto** (TA) es el tiempo necesario para acceder al nivel superior cuando se produce un acierto:

**t. empleado en determinar si la información está en ese nivel**

+

**t. empleado en acceder a esa información**

# Definiciones (III)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

◆ La **penalización de fallo** (PF) es el tiempo consumido en obtener la información cuando se produce un fallo:

**t. empleado en sustituir un bloque del nivel superior por el bloque correspondiente del nivel inferior**

+

**tiempo en proporcionar el bloque al dispositivo que lo ha pedido (CPU)**

◆ Se distingue:

- ◆ **tiempo de acceso**: para acceder a la primera palabra del bloque
- ◆ **tiempo de transferencia**: para transferir el resto del bloque.  
Depende del ancho de banda entre las memorias.

# Definiciones (IV)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

◆ Una **dirección de memoria** especifica, por completo, la localización de un elemento de información dentro de un nivel determinado de la jerarquía de memoria. Consta de:

- ◆ **dirección de la estructura de bloque:** identifica un bloque dentro de ese nivel.
- ◆ **dirección del desplazamiento del bloque:** identifica el elemento dentro de un bloque.

Dirección de la  
estructura de  
bloque

Dirección del  
desplazamiento  
de bloque

01011010001000001001010	111001110
-------------------------	-----------

- ◆ Direccionamiento de 32 bits
- ◆  $2^9=512 \Rightarrow$  tamaño del bloque
- ◆  $2^{23}$  bloques

# Definiciones (y V)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

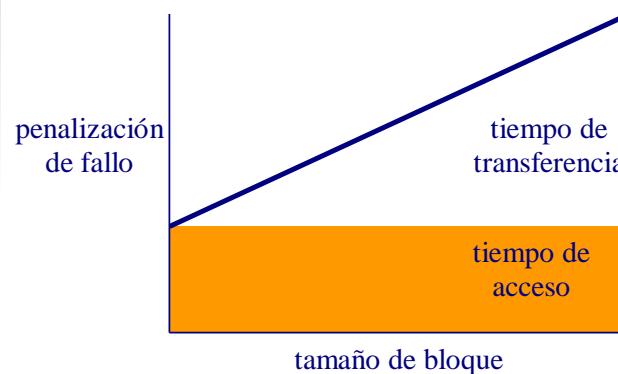
Memoria Virtual

Memoria

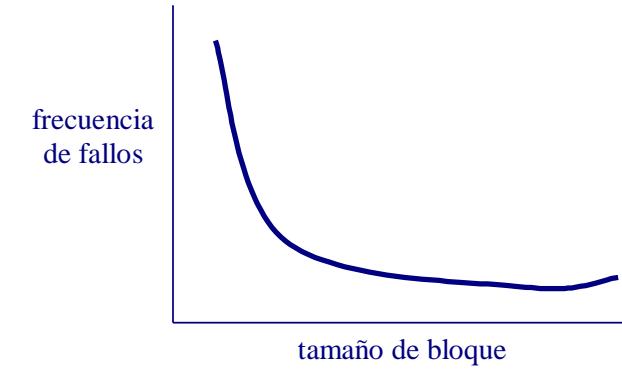
- El **tiempo medio de acceso a memoria** (TMA) se define como:

$$\text{tiempo de acierto} + \text{tasa de fallos} * \text{penalización de fallo}$$

- PF se puede medir en ciclos de reloj
- Existe una relación entre el TMA y el tamaño del bloque.



tiempo promedio de acceso



tamaño de bloque

## **5.3. Memoria caché**

---

**Tema 5 Rendimiento de la jerarquía de memoria**

**Arquitectura de los Computadores**

# Memoria caché

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## MEMORIA CACHÉ

- 5.3.1 Organización de la caché
- 5.3.2 Ubicación de bloque
- 5.3.3 Estrategias de reemplazo
- 5.3.4 Operación de la caché
- 5.3.5 Políticas de escritura
- 5.3.6 Rendimiento

# Memoria caché (I)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ La memoria de la CPU es **mucho más rápida** que la memoria principal.
- ➋ Lo ideal sería que la memoria principal fuera de la misma tecnología que los registros de la CPU, pero debido a su alto coste, se tiende a **soluciones intermedias**.
- ➌ Aprovechando el principio de localidad se coloca una memoria muy rápida (**memoria caché**) entre la CPU y la memoria principal.
- ➍ La CPU accede más veces a memoria caché que a memoria principal.
- ➎ Contiene una copia de aquellas posiciones de memoria principal que están siendo utilizadas por la CPU.

# Memoria caché (y II)

Introducción

Jerarquía de memoria

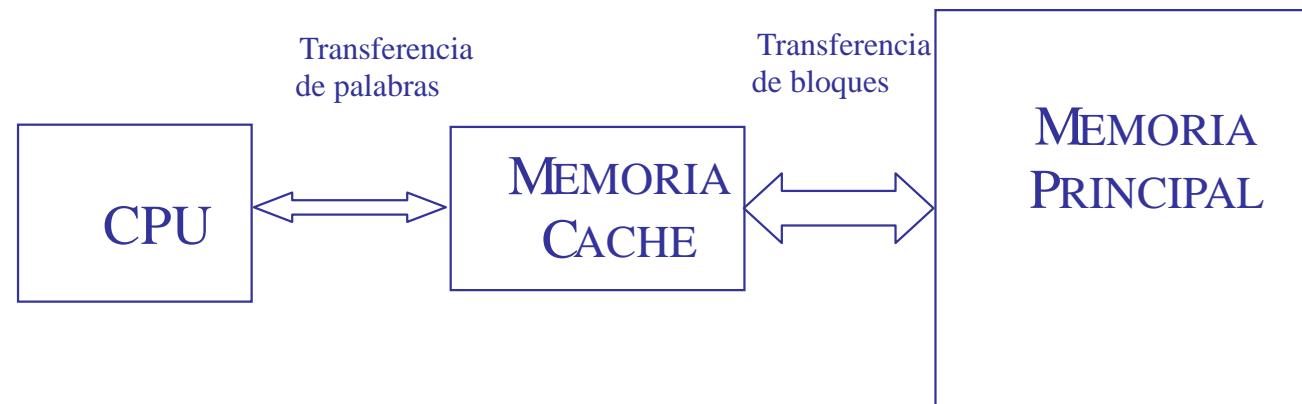
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- El funcionamiento de la memoria caché se basa en la **transferencia de bloques** entre memoria principal y caché, y en la **transferencia de palabras** entre memoria caché y CPU.



## Cuestiones a tener en cuenta:

- ¿Dónde se ubica un bloque en la caché?
- ¿Cómo se encuentra un bloque en la caché?
- ¿Qué bloque debe reemplazarse en caso de fallo?
- ¿Qué ocurre en una escritura?

# ¿Dónde se ubica un bloque en la caché?

Introducción

Jerarquía de memoria

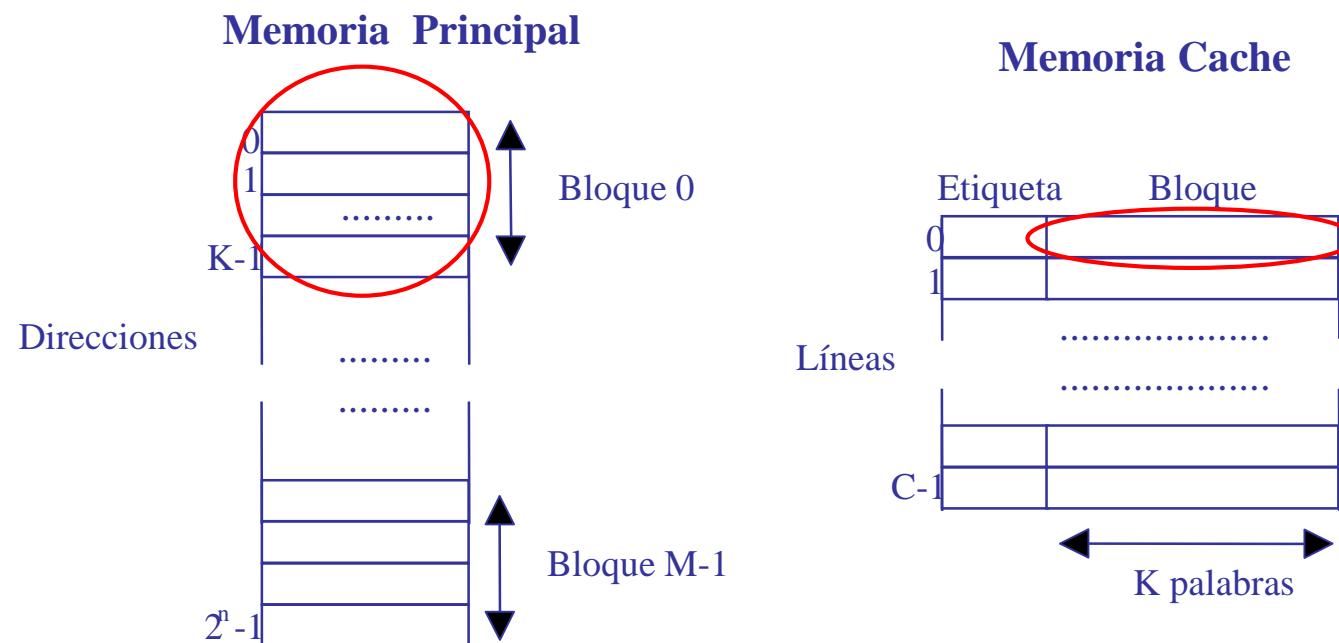
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- La memoria caché se divide en **Líneas**, que poseen el mismo tamaño que los bloques de memoria principal.



- A partir de la dirección de memoria suministrada por la CPU, se convierte en la dirección de caché correspondiente de acuerdo con el **método de ubicación de bloques**.

# ¿Dónde se ubica un bloque en la caché?

Introducción

Jerarquía de memoria

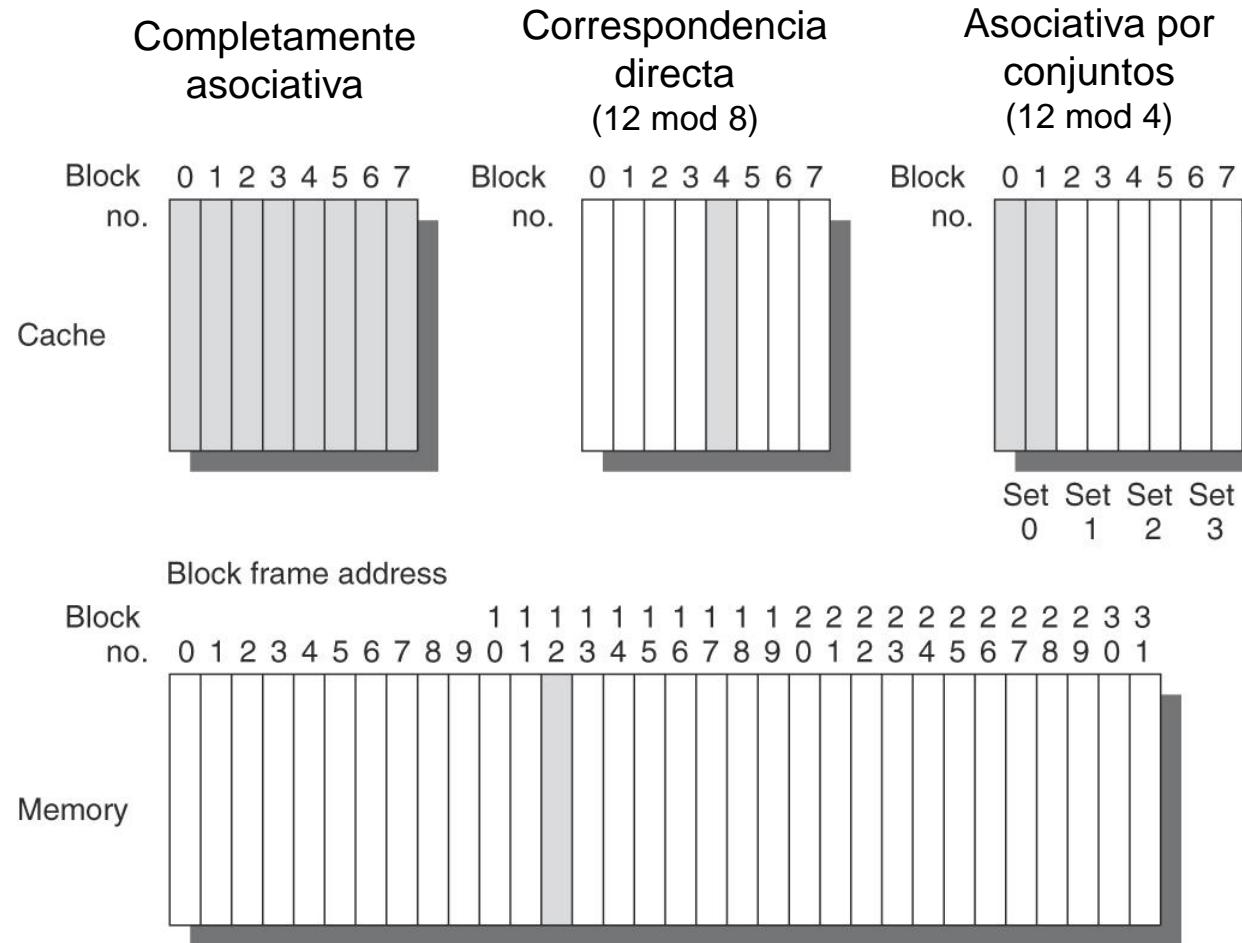
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- Las restricciones de ubicación del bloque dan lugar a tres categorías en la organización de la caché:



# ¿Dónde se ubica un bloque en la caché?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Es necesaria una **función de correspondencia** que haga corresponder bloques de memoria principal con líneas de memoria caché.
  - ➌ Correspondencia directa: Cada bloque sólo puede aparecer en un lugar en la caché.  
línea = dirección de la estructura de bloque **mod** n° líneas
  - ➍ Correspondencia asociativa por conjuntos: Un bloque se puede colocar en un conjunto restringido de lugares en la caché.  
conjunto = dirección de la estructura de bloque **mod** n° de conjuntos  
línea = [(líneasxconjunto)\*conjunto .. (líneasxconjunto)\*(conjunto+1)-1]
  - ➎ Correspondencia completamente asociativa: Un bloque se puede colocar en cualquier parte de la caché.  
línea = [0.. n° líneas -1]

# ¿Dónde se ubica un bloque en la caché?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## Ejemplo:

- ➊ El tamaño de la memoria **caché** es de **4Kb**
- ➋ **Bloques de 16 bytes** para transferir entre MP y caché
- ➌ Esto indica que la **caché** tiene  $(4096/16)$  **256 líneas**
- ➍ **Memoria principal** tiene **64Kb** por lo que el **bus de direcciones** es de **16 bits**. Luego, la memoria principal posee **4096 bloques**.

# ¿Cómo se encuentra un bloque en la caché?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Las cachés incluyen una etiqueta de dirección en cada línea que identifica la dirección de la estructura del bloque.
- ➋ Esta etiqueta se comprueba para ver si coincide con la dirección de la estructura del bloque requerido por la CPU.
- ➌ Como la velocidad es esencial, todas las posibles etiquetas se buscan en paralelo
- ➍ Se añade un **bit de validez** a la etiqueta para indicar si la línea de la cache contiene información válida.

# Correspondencia directa (I)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

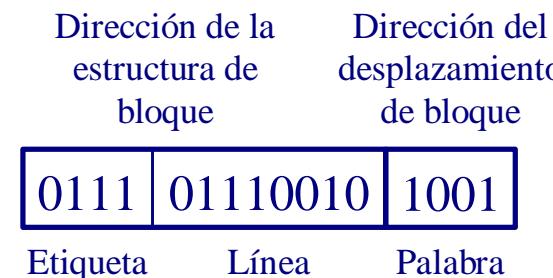
Memoria Virtual

Memoria

- Un bloque sólo puede colocarse en una línea de la caché, siguiendo la expresión:

**Línea = dirección de la estructura de bloque mod nº de líneas**

- Cada dirección de memoria principal puede verse dividida en 3 campos:



- La dirección de la estructura de bloque se divide en **etiqueta** y **línea**.
  - Con los bits de **línea** se puede averiguar directamente en qué línea de la caché se debe guardar ese bloque.
  - Cuando un bloque se almacena en una línea es necesario **etiquetarla** para diferenciarlo del resto de bloques que podrían estar en dicha línea.

# Correspondencia directa (II)

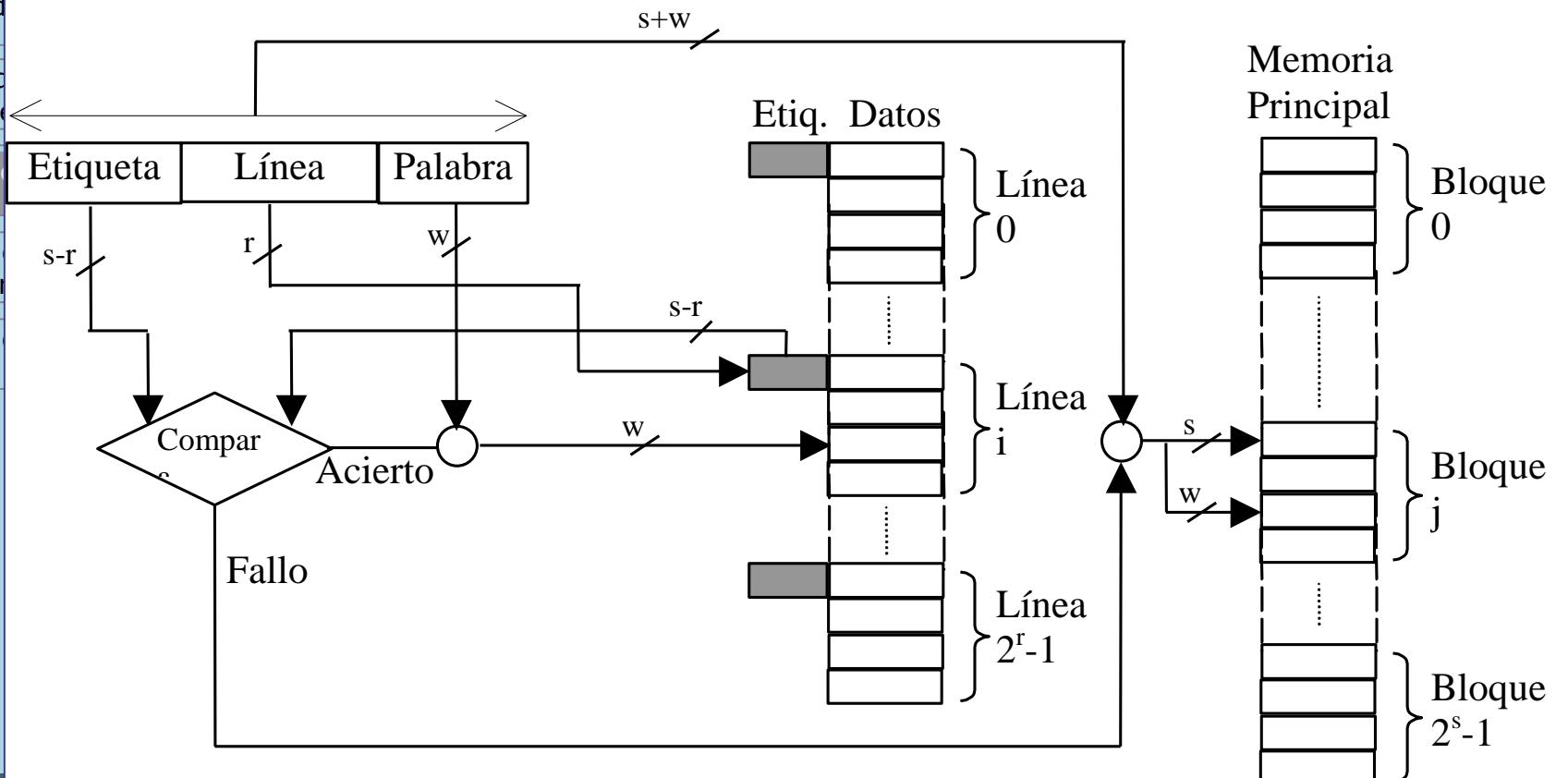
Introd

Jerard  
mem

Mem

Mem  
Pr

Mem



Memoria

# Correspondencia directa (III)

Introducción

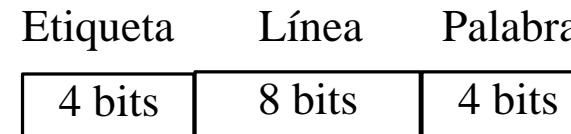
Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria



Memoria Principal

Dirección

0000	12
0001	34
0002	56
000F	78
...	...
F800	AB
F801	0F
F802	FF
F80F	54
...	...
FFF0	FF
FFF0	21
FFFFE	33
FFFF	55

Bloque 0

Bloque 3968

Bloque 4095

Memoria Cache

Nº Línea	Etiqueta	Datos
0 (00)	0	123456.....78
1 (01)	.....	.....
.....	.....	.....
128 (80)	F	AB0FFF.....54
.....	.....	.....
255 (FF)	F	FF.....213355

4 bits

16 bytes

8 bits

# Correspondencia directa (y IV)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ◆ Ventajas:

- ◆ Técnica muy simple
- ◆ Fácil de implementar

## ◆ Desventajas:

- ◆ Existe una gran limitación al poder ubicar un bloque de memoria principal en una única línea de caché.

# Correspondencia asociativa (I)

Introducción

Jerarquía de memoria

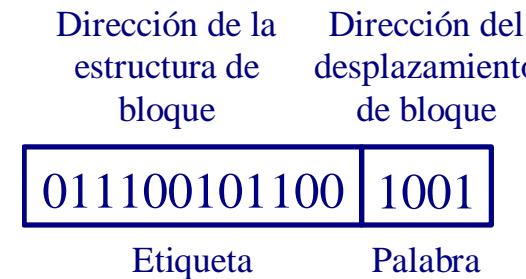
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- Un bloque se puede ubicar en **cualquier posición** de la caché.
- Las direcciones de memoria están formadas por una **etiqueta** y una **palabra**.



- El campo de **etiqueta** se utiliza para identificar el bloque que se encuentra almacenado en esa línea.

# Correspondencia asociativa (II)

Introducción

Jerarquía de memoria

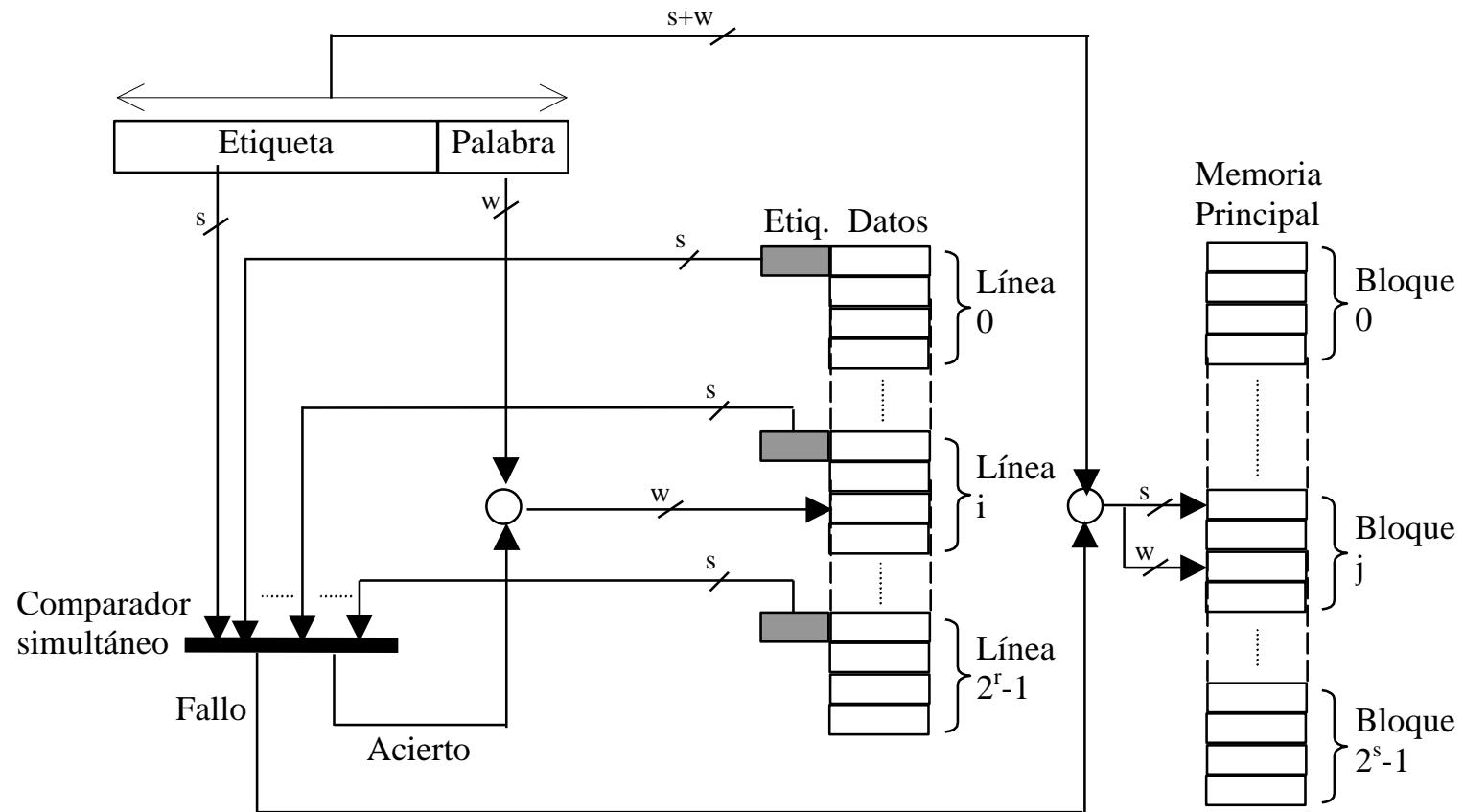
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- Para determinar si un bloque está en caché, se examinan en **paralelo** todas las etiquetas de las líneas de memoria caché.



# Correspondencia asociativa (III)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

Etiqueta Palabra

12 bits

4 bits

Memoria Principal

Dirección Dato

0000	12
0001	34
0002	56
000F	78
...	...
F800	AB
F801	0F
F802	FF
F80F	54
...	...
FFF0	FF
FFFD	21
FFFE	33
FFFF	55

Bloque 0

Bloque 3968

Bloque 4095

Memoria Cache

Etiqueta	Datos	Nº Línea
000	123456.....78	0
.....	.....	1
.....	.....	.....
F80	AB0FFF.....54	128
.....	.....	.....
FFF	FF.....213355	255

12 bits

16 bytes

# Correspondencia asociativa (y IV)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ◆ Ventaja:

- ◆ Cualquier combinación de bloques de memoria principal puede estar en la caché en un determinado instante.

## ◆ Desventaja:

- ◆ Se necesita una circuitería muy compleja para realizar la búsqueda de un bloque.

# Correspondencia asociativa por conjuntos

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Combina las 2 técnicas anteriores.
- ➋ Un **conjunto** es un grupo de dos o más líneas de caché.
- ➌ Al número de líneas que forman un conjunto se le denomina **vías**.
- ➍ Se dice que existe correspondencia asociativa por conjuntos de N vías.
- ➎ Un **bloque de memoria** principal se corresponde con un **conjunto**, pudiéndose ubicar en cualquiera de las líneas que lo componen.
- ➏ El conjunto se escoge de forma similar a la correspondencia directa:  
**conjunto = dirección de la estructura de bloque mod n° de conjuntos**
- ➐ Dentro de ese conjunto, se puede situar el bloque en cualquier línea.

# Correspondencia asociativa por conjuntos (II)

Introducción

Jerarquía de memoria

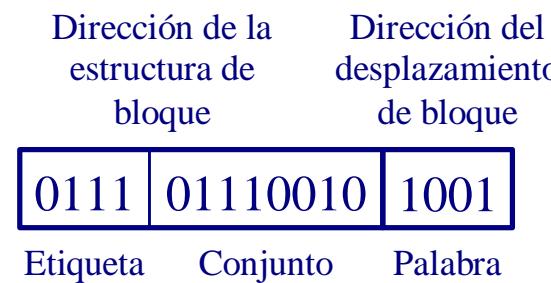
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- Cualquier tipo de correspondencia puede expresarse como correspondencia asociativa por conjuntos de 1 vía (directa), P vías (totalmente asociativa) o N vías.
- La dirección de memoria se divide en 3 campos:



- Si se incrementa el grado de asociatividad (N) vías, aumenta el nº de líneas por conjunto, disminuyendo el número de conjuntos y aumentando la etiqueta.

# Correspondencia asociativa por conjuntos (III)

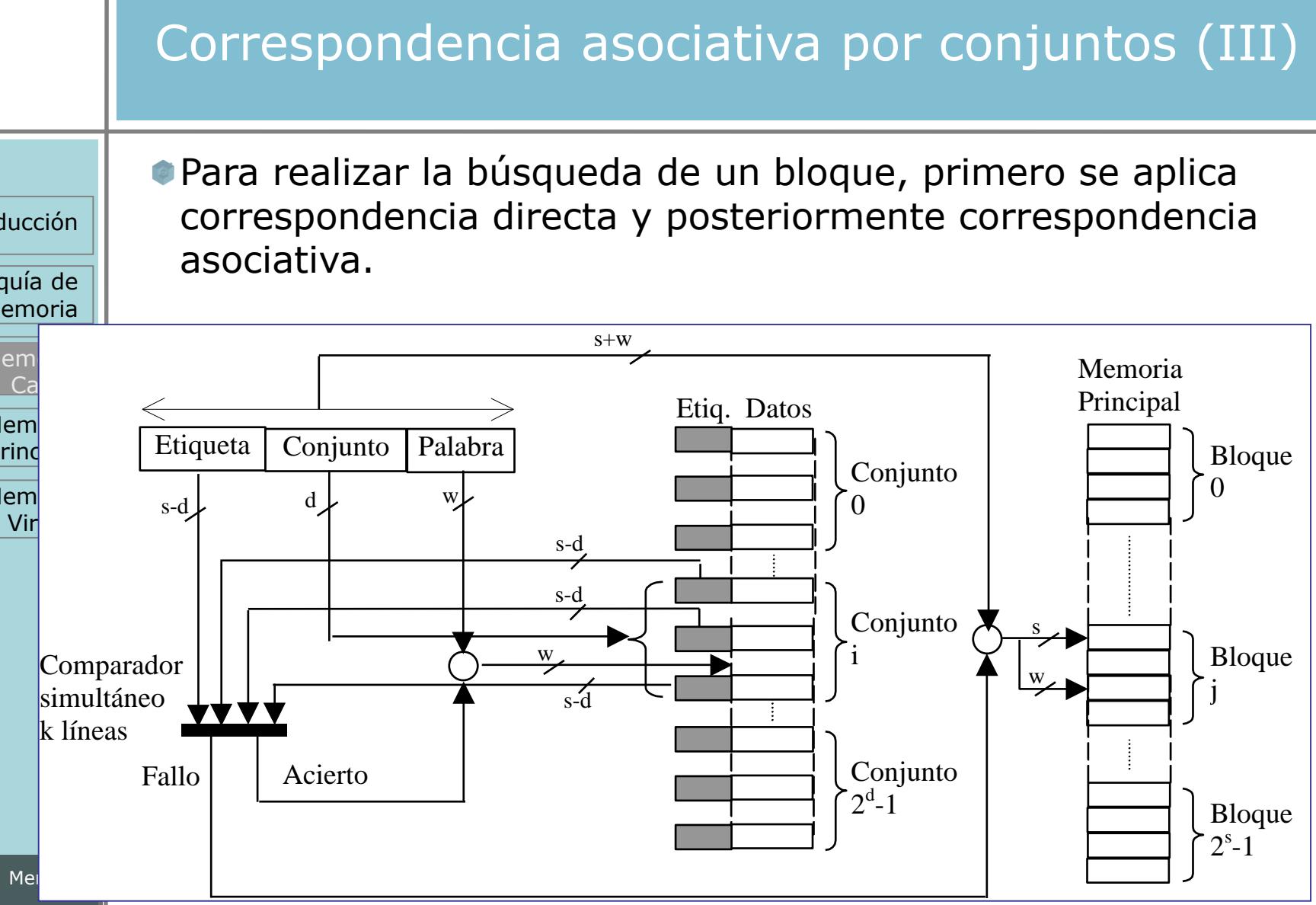
Introducción

Jerarquía de memoria

Mem  
Ca

Mem  
Princ

Mem  
Vir



# Correspondencia asociativa por conjuntos (IV)

Introducción

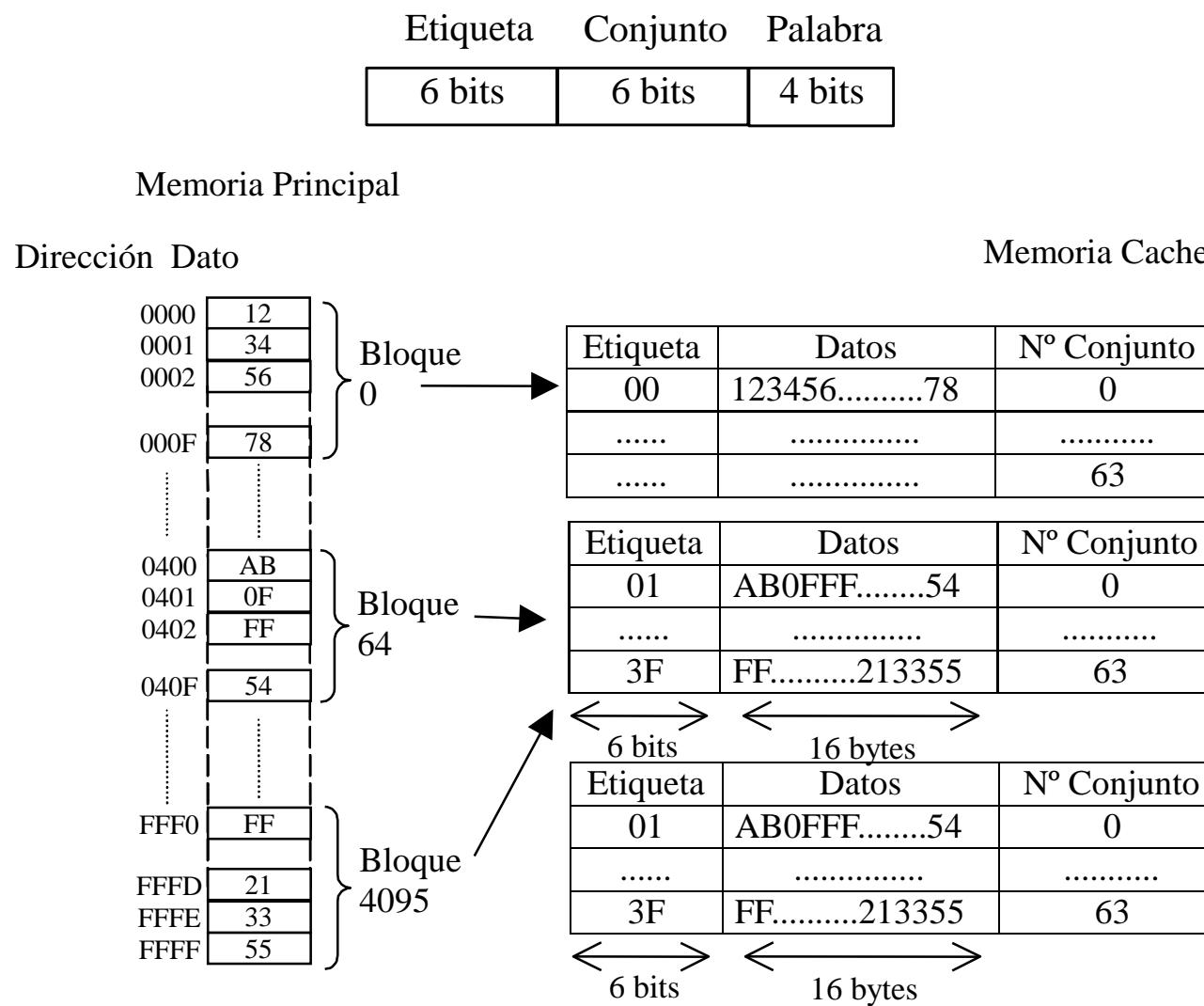
Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria



# Correspondencia asociativa por conjuntos (y V)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

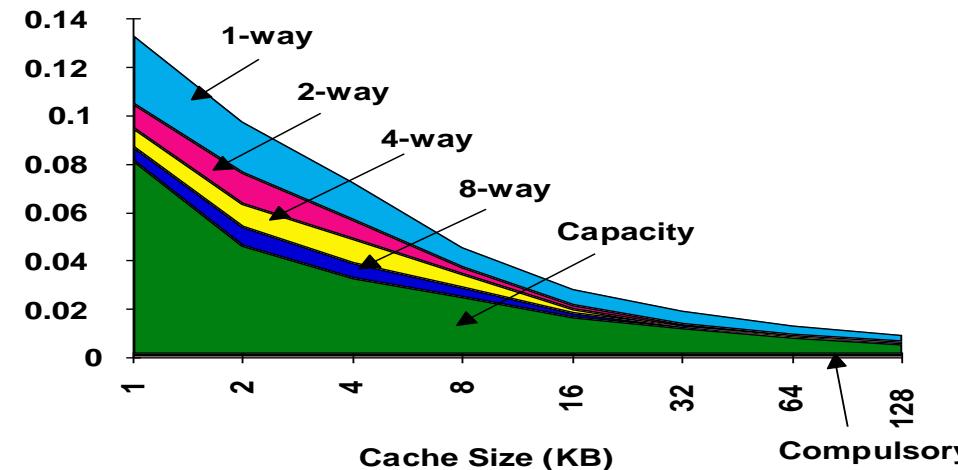
Memoria

## Ventajas:

- ◆ Aúna ventajas de los otros dos métodos.
- ◆ Usualmente se emplean conjuntos de 2 vías, mejorando las tasas de acierto frente a la correspondencia directa.
- ◆ También se usan las 4 vías, que proporciona una relativa mejora a un coste moderado.

## Desventajas:

- ◆ A partir de 4 vías, el beneficio de mayor número de vías no compensa debido al mayor coste.



# Ejercicio

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

Un diseñador está pensando, para un procesador sencillo, en dotarlo de una memoria caché. El diseñador escoge como estrategia de ubicación de bloques la correspondencia directa. El sistema anterior posee una memoria principal de 64KB y se escoge un tamaño de bloque de 256 palabras. El tamaño de la memoria caché será de 4KB.

- a) ¿Cuántas líneas tendrá la caché?
- b) ¿Cuántos bloques tendrá la memoria principal?
- c) Partiendo de una caché vacía y tras darse la siguiente secuencia ordenada de accesos de a estas direcciones de memoria:  
1º 0xF321, 2º: 0xC743, 3º: 0xD327 y 4º: 0x7135 indica:
  1. La línea o líneas donde se ha producido un reemplazo de bloque (si se ha dado el caso, ya que no se considera reemplazo cuando la línea está vacía)
  2. La etiqueta de bloque que finalmente contendrá la línea 7 de la caché
  3. La etiqueta de bloque que finalmente contendrá la línea 3 de la caché

# ¿Qué bloque debe reemplazarse en caso de fallo?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Cuando se introduce un nuevo bloque en la cache, debe sustituirse uno de los bloques existentes.
- ➋ Para el caso de correspondencia directa sólo hay una posible línea para cada bloque.
- ➌ Para las técnicas asociativas se requieren **algoritmos de sustitución**. Estos algoritmos deben implementarse por hardware.

# ¿Qué bloque debe reemplazarse en caso de fallo?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ◆ Método aleatorio

- ◆ Se escoge aleatoriamente uno de los bloques candidatos.
- ◆ Implementación hardware sencilla.

## ◆ Método LRU (Least recently used)

- ◆ Se sustituye el bloque que hace más tiempo que no fue referenciado
- ◆ Fácil de implementar para asociativas por conjuntos de dos vías
- ◆ Cuando se referencia una línea se pone a uno su bit de uso y a cero el de la otra línea del mismo conjunto

# ¿Qué bloque debe reemplazarse en caso de fallo?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ◆ **Método FIFO (First in first out):**

- ◆ Se reemplaza el bloque que hace más tiempo que está en caché
- ◆ Puede implementarse mediante una técnica de buffer circular

## ◆ **Método LFU:**

- ◆ Se elimina el bloque que ha sido menos referenciado
- ◆ Se debe asociar un contador a cada línea

# Operación de la caché (I)

## 1. Inicialización

- ➊ Cuando se enciende el computador:
  - ➌ La memoria principal es cargada por el disco con una serie de programas

## 2. Lectura

- ➋ La CPU solicita una palabra de memoria:
  - ➌ Se lee primero de caché (se puede leer al mismo tiempo que se compara la etiqueta)
  - ➌ Si allí no se encuentra la palabra requerida (fallo de caché) se acude a MP, de donde se lee la palabra, y se emplaza el bloque que contiene esa palabra en la caché.
- ➋ Las lecturas dominan los accesos a la caché, por lo que se debe **optimizar la caché para las operaciones de lectura**.

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

# Operación de la caché (y II)

## 3. Escritura

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ La CPU pide reemplazar una porción de bloque (palabra)
- ➋ Esto implica la secuencia de:
  - ➌ leer el bloque original
  - ➍ modificar una parte de él
  - ➎ escribir de nuevo el bloque
- ➏ La modificación del bloque no puede comenzar hasta que no se haya comprobado la etiqueta → más lenta que lectura

# ¿Qué ocurre en una escritura?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ➊ Asegurar coherencia entre caché y memoria

### ➊ Aciertos:

- ➊ Escritura directa (write-through)
- ➊ Postescritura (write-back)

### ➋ Fallos

- ➋ no ubicar en escritura (write-non-allocate)
- ➋ ubicar en escritura (write-allocate)

# Escritura directa (write through)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ **Acierto:** la información se escribe en la caché y en la MP en paralelo.
- ➋ **Fallo:** Se escribe en MP. Usualmente el bloque no es cargado en la caché (**no ubicar en escritura**). Menos frecuente es cargar el bloque en caché (**ubicar en escritura**).
- ➌ **Ventaja:**
  - ➏ Los contenidos de MP y caché están siempre actualizados
- ➍ **Desventajas:**
  - ➏ Mucho tráfico de datos entre MP y CPU.
  - ➏ La **CPU debe detenerse** en las escrituras ya que siempre tiene que acceder a MP.

# Postescritura (write back)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ **Acierto:** la información se escribe sólo en el bloque de caché. Este bloque sólo se escribe en MP al ser reemplazado.
  - ➋ **Fallo:** Se suele utilizar la **ubicación en escritura** (similar a un fallo de lectura). Las escrituras posteriores no provocan fallo de caché.
  - ➌ Se utiliza un **bit de modificación** para saber si ese bloque está **limpio** o **sucio**.
- ➍ **Ventajas:**
- ➎ Las escrituras se realizan a la **velocidad de la memoria caché**.
  - ➏ La postescritura utiliza **menos ancho de banda** de memoria ya que varias escrituras requieren una única escritura del bloque en MP. Esto reduce la espera de CPU.
- ➐ **Desventaja:**
- ➑ La caché y la MP son inconsistentes.

# Múltiples niveles de caché

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ El esquema descrito se puede extender a más de un nivel de memoria caché.
- ➋ Cuando una memoria caché recibe una petición y el dato no está presente en la memoria, el dato se le pide a la memoria del siguiente nivel
- ➌ La memoria de mayor nivel (comenzando por el 1 al lado del procesador) tiene menor tamaño
- ➍ Los procesadores suelen tener el primer nivel de memoria dentro del encapsulado
- ➎ En ciertos modelos también encontramos un segundo y tercer nivel (L2 Cache, L3 Cache) que puede encontrarse dentro o fuera del chip

# Caché Pentium

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ● Caché Instrucciones

- ◆ P y Ppro: 8 ó 16 KB
- ◆ P4: 12000 microinstrucciones

## ● Caché de nivel 1

- ◆ P: 16 KB
- ◆ Ppro: 8 ó 16 KB
- ◆ PIII: 8 KB
- ◆ PIV: 8 KB

## ● Caché de nivel 2

- ◆ P: 256 ó 512 KB
- ◆ Ppro: 128, 256, 512 KB ó 1 MB
- ◆ PIII: 256 KB
- ◆ PIV: 256, 512 KB ó 1 MB

## ● Caché de nivel 3

- ◆ PIV: 2MB

# Rendimiento de la caché (I)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ◆ Frecuencia de fallos de la caché (FF)

- ◆ Es independiente de la velocidad del hardware.
- ◆ Es engañosa, ya que no tiene en cuenta la penalización de fallos (PF)
- ◆ El objetivo de una jerarquía de memoria es reducir el tiempo de ejecución, no los fallos.

## ◆ Tiempo medio de acceso a memoria (TMA)

- ◆ Contempla tanto la FF como la PF:

$$TMA = TA + FF * PF$$

## ◆ Tiempo de ejecución (de una tarea o programa) ( $T_{CPU}$ )

- ◆ Medida real del rendimiento

$$T_{CPU} = NI * CPI * T_{reloj} = NI * (CPI_{ejec} + CPI_{mem}) * T_{reloj}$$

# Rendimiento de la caché (II)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

◆  $CPI_{MEM}$ : ciclos en espera de CPU por referencias a memoria

◆ Debidos a fallos de caché.

$$CPI_{mem} = \frac{\text{ciclos de detención debido a fallos}}{NI} = CPI_{mem}(\text{lecturas}) + CPI_{mem}(\text{escrituras})$$

$$CPI_{mem}(\text{lecturas}) = \frac{\text{fallos en lecturas} * PF_{\text{lecturas}}}{NI} = \frac{\text{lecturas} * FF_{\text{lecturas}} * PF_{\text{lecturas}}}{NI}$$

$$CPI_{mem}(\text{escrituras}) = \frac{\text{fallos en escrituras} * PF_{\text{escrituras}}}{NI} = \frac{\text{escrituras} * FF_{\text{escrituras}} * PF_{\text{escrituras}}}{NI}$$

$$CPI_{mem} = \frac{NM * FF * PF}{NI}$$

# Rendimiento de la caché (y IV)

$$T_{CPU} = NI * (CPI_{ejec} + CPI_{mem}) * T_{reloj}$$

$$T_{CPU} = NI * \left( CPI_{ejec} + \frac{NM}{NI} * FF * PF \right) * T_{reloj}$$

- Algunos diseñadores prefieren medir los fallos por instrucción en lugar de fallos por acceso a memoria:

$$T_{CPU} = NI * \left( CPI_{ejec} + \frac{\text{fallos}}{\text{instrucción}} * PF \right) * T_{reloj}$$

- Puesto que el CPI global depende tanto de la FF como de la PF, **el impacto de la caché sobre el rendimiento global ha de ser analizado en función de las características de la MP.**
- Los cambios en la FF (diseño de caché) puede causar cambios sustanciales o despreciables, dependiendo de la PF (diseño o velocidad de MP)

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

# Ejemplo

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- Suponer que la penalización de fallos de la caché es de 200 ciclos de reloj y que todas las instrucciones normalmente emplean 1.0 ciclos de reloj (ignorando las detenciones de memoria). Suponer que la frecuencia de fallos es de 2%, hay una media de 1.5 referencias a memoria por instrucción y que el número medio de fallos a la caché por 1000 instrucciones es 30. ¿Cuál es el impacto en el rendimiento cuando se incluye el comportamiento de la caché? Calcula el impacto utilizando tanto los fallos por instrucción como la frecuencia de fallos.

$$T_{CPU} = NI * \left( CPI_{ejec} + \frac{NM}{NI} * FF * PF \right) * T_{reloj}$$

$$T_{CPU} = NI * \left( CPI_{ejec} + \frac{\text{fallos}}{\text{instrucción}} * PF \right) * T_{reloj}$$

# Ejemplo

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- Suponer que la penalización de fallos de la caché es de 200 ciclos de reloj y que todas las instrucciones normalmente emplean 1.0 ciclos de reloj (ignorando las detenciones de memoria). Suponer que la frecuencia de fallos es de 2%, hay una media de 1.5 referencias a memoria por instrucción y que el número medio de fallos a la caché por 1000 instrucciones es 30. ¿Cuál es el impacto en el rendimiento cuando se incluye el comportamiento de la caché? Calcula el impacto utilizando tanto los fallos por instrucción como la frecuencia de fallos.

$$T_{CPU} = NI \times (1.0 + (30/1000 \times 200)) * T_{reloj} = NI \times 7.0 \times T_{reloj}$$

$$T_{CPU} = NI \times (1.0 + (1.5 \times 2\% \times 200)) * T_{reloj} = NI \times 7.0 \times T_{reloj}$$

$T_{CPU}$  se incrementa en 7 con una caché con fallos.

Sin jerarquía de memoria el CPI se incrementaría en  $1.0 + 200 \times 1.5$  o 301

- Cuanto más bajo  $CPI_{ejec}$  más pronunciado el impacto
- Una frecuencia mayor de reloj lleva a una PF mayor

# Ejemplo

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ ¿Cuál es el impacto de dos organizaciones cachés diferentes en el rendimiento de un procesador? Suponer que el CPI con una caché perfecta es 1.65 con una duración del ciclo de reloj es 0.35ns; hay 1.4 referencias a memoria por instrucción, el tamaño de ambas cachés es de 128KB y ambas tienen un tamaño de bloque de 64bytes. Un caché es de correspondencia directa y la otra es asociativa por conjunto de dos vías. Como la velocidad de la CPU está ligada directamente a la velocidad de un acierto en la caché, suponer que la duración del ciclo de reloj debe alargarse 1.35 veces para acomodar la selección en la caché asociativa por conjuntos. Para la primera aproximación, la penalización de fallos es de 65ns para cualquier organización de la caché (En la práctica se debe redondear hacia arriba o hacia abajo un número entero de ciclos de reloj). Primero calcular el tiempo medio de acceso a memoria y después el rendimiento de la CPU. Suponer que el tiempo de acierto es 1 ciclo de reloj, la frecuencia de fallos de la caché de correspondencia directa es de 2.1% y la de la caché asociativa por conjuntos es de 1.9%.

# Ejemplo

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ Tiempo medio de acceso a memoria:

$$TMA = TA + FF * PF$$

$$TMA_{1\text{-vía}} = 0.35 + (0.021 \times 65) = 1.72 \text{ ns}$$

$$TMA_{2\text{-vías}} = 0.35 \times 1.35 + (0.019 \times 65) = 1.71 \text{ ns}$$

- ◆ Rendimiento del procesador:

$$T_{CPU} = NI * \left( CPI_{ejec} + \frac{NM}{NI} * FF * PF \right) * T_{reloj}$$

Se sustituye 65ns por PF\*T<sub>CLK</sub>

$$T_{CPU\ 1vía} = NI * (1.65 \times 0.35 + (0.021 \times 1.4 \times 65)) = 2.47 \times NI$$

$$T_{CPU\ 2vía} = NI * (1.65 \times 0.35 \times 1.35 + (0.019 \times 1.4 \times 65)) = 2.49 \times NI$$

El rendimiento relativo:

$$\frac{T_{CPU\ 2vía}}{T_{CPU\ 1vía}} = \frac{2.49 \times NI}{2.47 \times NI} = 1.01$$

# Optimizaciones

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ Tiempo medio de acceso a memoria:

$$TMA = TA + FF \cdot PF$$

- ◆ Optimizaciones:

- ◆ Reducir Frecuencia de fallos: tamaño del bloque más grande, tamaño de la caché más grande y mayor asociatividad.
- ◆ Reducir Penalización de fallos: caché multinivel y dar prioridad a las lecturas sobre las escrituras.
- ◆ Reducir el tiempo de acierto: evitar la traducción de la dirección cuando se indexa a la caché.

# Optimizaciones

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

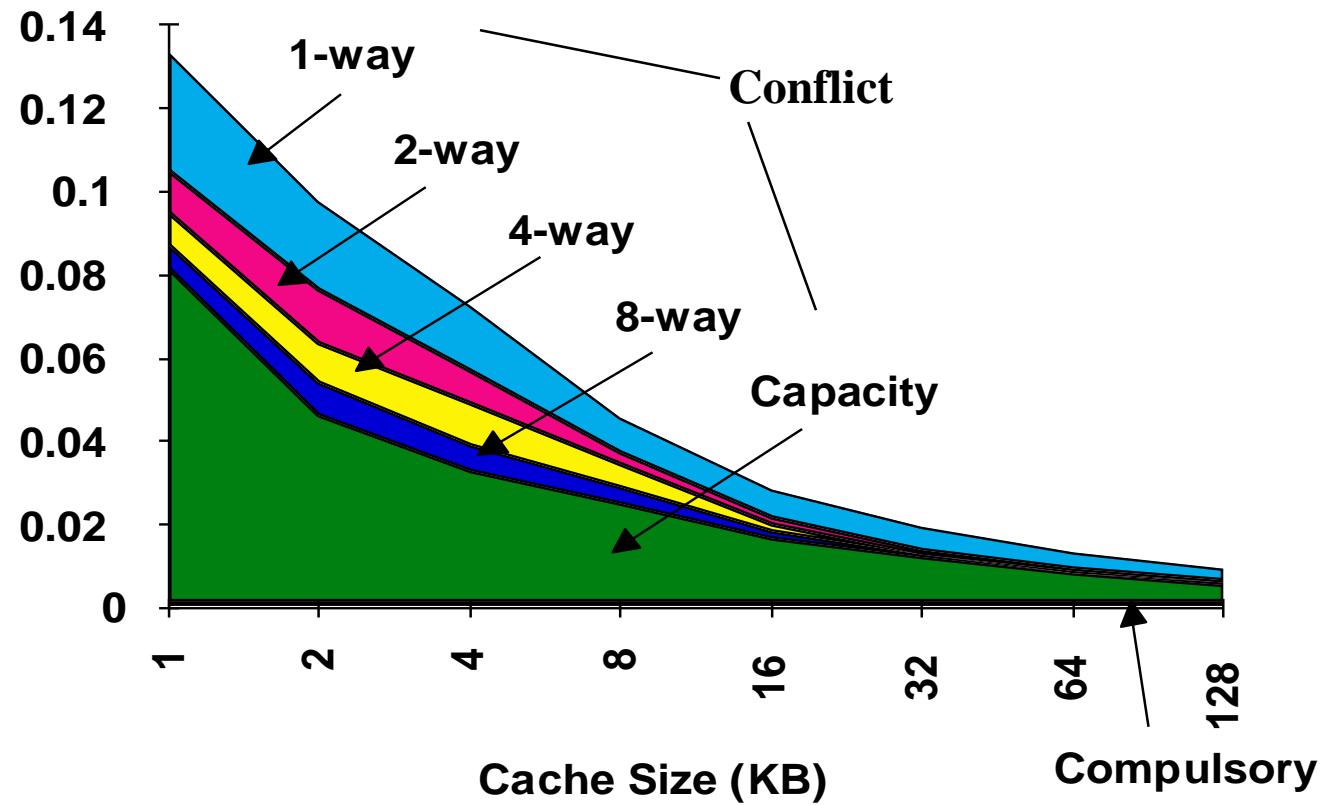
## ◆ Tipos de fallos

- ◆ Por carga inicial o forzosos (*Compulsory*)
  - ◆ Carga inicial de la caché.
  - ◆ Fallos incluso en una caché infinita
- ◆ De capacidad (*Capacity*)
  - ◆ Si la caché no puede almacenar todos los bloques de un programa en ejecución.
- ◆ Por conflicto (*Conflict*)
  - ◆ Aún habiendo líneas libres no se puede cargar un bloque
  - ◆ Suele suceder en cachés con un grado de asociatividad (correspondencia directa o asociativa por conjuntos de pocas vías)

$$\text{Tasa de Fallo} = \text{Nº Fallos} / \text{Nº Accesos}$$

# Optimizaciones

## Tipos de fallos



Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

# Optimizaciones

Introducción

Jerarquía de memoria

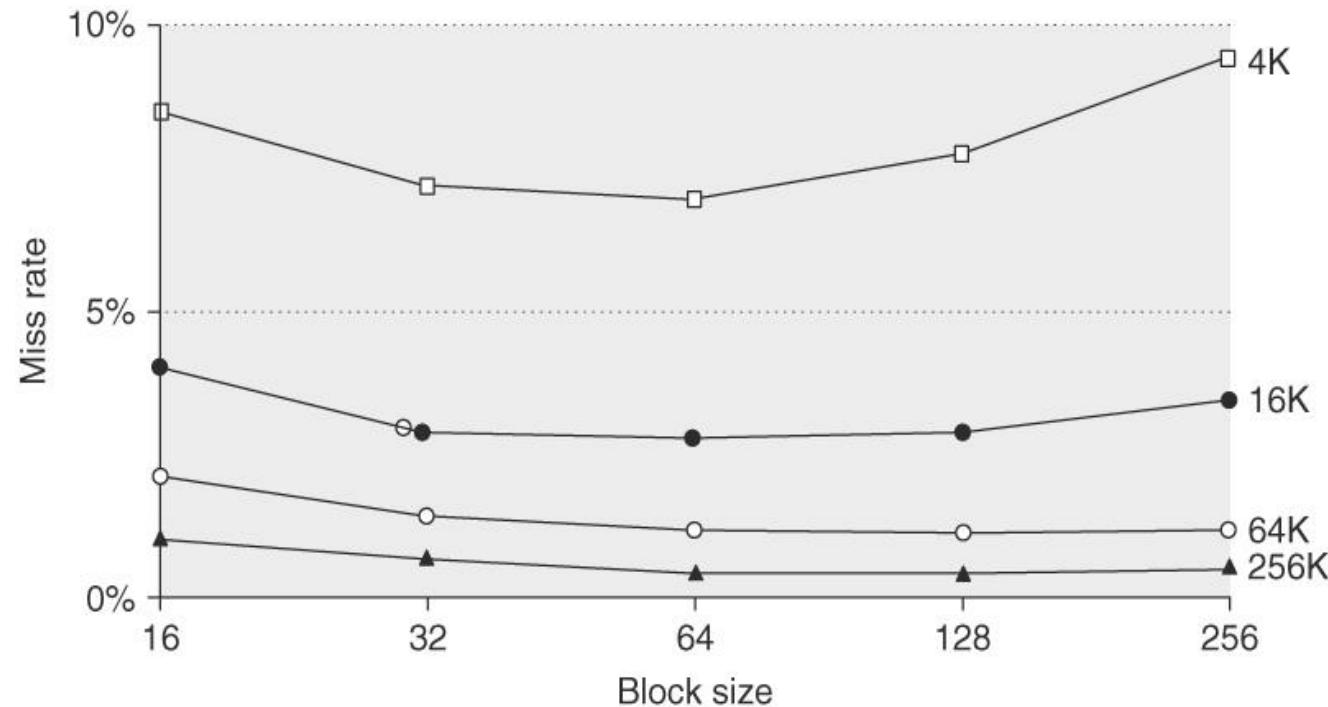
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ Reducir frecuencia de fallos aumentando el tamaño del bloque



- ◆ Se reducen los fallos por carga inicial pero aumentan los fallos por capacidad y conflicto
- ◆ Los fallos aumentan si el tamaño del bloque es demasiado grande en relación al tamaño de la caché.
- ◆ Aumentar el tamaño del bloque incrementa la penalización por fallos.

# Optimizaciones

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

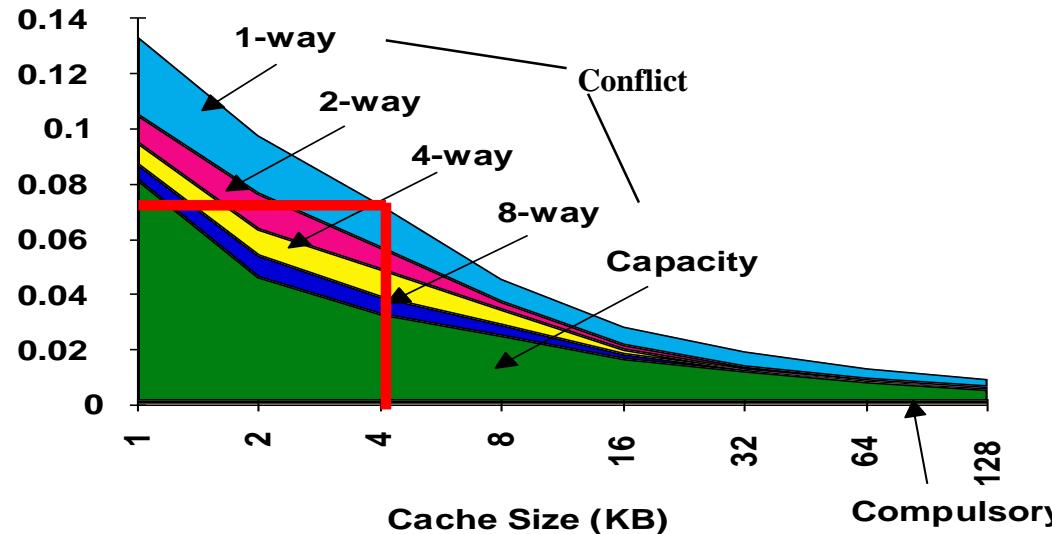
Memoria Virtual

Memoria

## Reducir frecuencia de fallos aumentando el tamaño de la caché

- Potencialmente se alarga el tiempo de acierto y aumenta el coste.

## Reducir frecuencia de fallos aumentando la asociatividad



- Una caché de correspondencia directa de tamaño N tiene la misma frecuencia de fallos que una asociativa por conjunto de 2 vías de tamaño N/2.
- Aumentar la asociatividad puede aumentar el coste e incrementar el tiempo de acierto.

# Optimizaciones

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## ➊ Reducir la penalización de fallos con cachés multinivel

- ➌ La caché de primer nivel pequeña para ajustarse al tiempo de ciclo de reloj del procesador rápido.
- ➌ La caché de segundo nivel lo suficientemente grande para que haya pocos accesos a la MP.

$$TMA = TA_{L1} + FF_{L1} * PF_{L1}$$

$$PF_{L1} = TA_{L2} + FF_{L2} * PF_{L2}$$

## ➋ Hay que diferenciar:

- ➌  $FF\_local = n^o \text{ de fallos} / n^o \text{ de accesos a la caché}$
- ➌  $FF\_global = n^o \text{ de fallos} / n^o \text{ total de accesos realizados por la CPU}$

## ➌ En general se cumple: $FF\_local \geq FF\_global$

## ➍ Y en particular:

- ➌  $FF\_local_{L1} = FF\_global_{L1}$
- ➌  $FF\_local_{L2} > FF\_global_{L2}$

# Ejemplo

- ◆ Suponed que en 1000 referencias a memoria hay 40 fallos en la caché de primer nivel y 20 fallos en la caché de segundo nivel. ¿Cuáles son las distintas frecuencias de fallos? Asumid que la penalización de fallos de la caché L2 a la memoria es de 200 ciclos de reloj, el tiempo de acierto a la caché L2 es de 10 ciclos de reloj, el tiempo de acierto a la caché L1 es de 1 ciclos de reloj y que hay 1.5 referencias a memoria por instrucción. ¿Cuál es el tiempo medio de acceso?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

# Ejemplo

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ Suponed que en 1000 referencias a memoria hay 40 fallos en la caché de primer nivel y 20 fallos en la caché de segundo nivel. ¿Cuáles son las distintas frecuencias de fallos? Asumid que la penalización de fallos de la caché L2 a la memoria es de 200 ciclos de reloj, el tiempo de acierto a la caché L2 es de 10 ciclos de reloj, el tiempo de acierto a la caché L1 es de 1 ciclos de reloj y que hay 1.5 referencias a memoria por instrucción. ¿Cuál es el tiempo medio de acceso?

$$FF_{localL1} = FF_{globalL1} = FF_{L1} = 40/1000 = 0.04 \text{ (4\%)}$$

$$FF_{localL2} = 20/40 = 0.5 \text{ (50\%)}$$

$$FF_{globalL2} = 20/1000 = 0.02 \text{ (2\%)}$$

$$\begin{aligned} TMA &= TA_{L1} + FF_{L1} * (TA_{L2} + FF_{L2} * PF_{L2}) = \\ &= 1 + 4\% \times (10 + 50\% \times 200) = 5.4 \text{ ciclos de reloj} \end{aligned}$$

# Optimizaciones

Introducción

Jerarquía de memoria

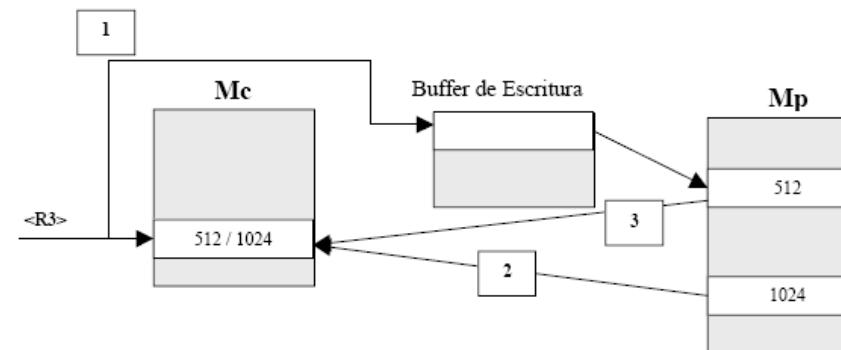
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ Reducir la penalización de fallos dando prioridad a las lecturas sobre las escrituras
  - ◆ Se realiza la lectura antes de que la escritura se haya completado.
  - ◆ Ejemplo: Caché de correspondencia directa y escritura directa mejorada con un buffer de escritura de tamaño apropiado. Se hace corresponder los bloques en los que se encuentran las direcciones 512 y 1024 sobre la misma línea de la caché. Supongamos que se ejecuta el siguiente programa:
    - (1) SW R3, 512(R0) ; M[512] <- <R3> // escritura sobre M[512]
    - (2) LW R1, 1024(R0); R1 <- M[1024] // lectura de M[1024]
    - (3) LW R2, 512(R0); R2 <- M[512] // lectura de M[512]



# Optimizaciones

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Reducir la penalización de fallos dando prioridad a las lecturas sobre las escrituras

(1) SW R3, 512(R0) ;      M[512] <-- <R3> // escritura sobre M[512]

(2) LW R1, 1024(R0);      R1 <-- M[1024] // lectura de M[1024]

(3) LW R2, 512(R0);      R2 <-- M[512] // lectura de M[512]

- ➋ Dos soluciones para evitar que la lectura 3 no lea un dato inconsistente en memoria:

- ➌ Esperar a que se vacíe el buffer. Esto incrementará la penalización de fallos de lectura en un factor de 1.5.
- ➍ En un fallo de lectura chequear el contenido del buffer de escritura y continuar la lectura si no hay conflicto.

## **5.4. Mejoras del rendimiento de la memoria principal**

---

**Tema 5 Rendimiento de la jerarquía de memoria**

**Arquitectura de los Computadores**

# Memoria principal

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## 5.4. MEJORAS DEL RENDIMIENTO DE LA MEMORIA PRINCIPAL

5.4.1 Técnicas de mejora del rendimiento

5.4.2 Anchura de la memoria

5.4.3 Memoria entrelazada

5.4.4 Bancos de memoria independientes

5.4.5 Evitar conflictos entre bancos de memoria

# Introducción

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ La memoria principal satisface las demandas de las cachés y unidades vectoriales y sirve como interfaz de E/S
- ➋ Las medidas de rendimiento en la MP hacen énfasis en la latencia y el ancho de banda:
  - ➌ La latencia es la preocupación primordial de la caché (afecta a la penalización de fallos)
  - ➍ El ancho de banda es la preocupación primordial de las E/S y de las unidades vectoriales
- ➎ La latencia se expresa utilizando:
  - ➏ Tiempo de acceso: tiempo desde que se pide una lectura hasta que llega la palabra deseada.
  - ➐ Duración del ciclo: tiempo mínimo entre peticiones a memoria.

# Organizaciones para mejorar el rendimiento

Introducción

Jerarquía de memoria

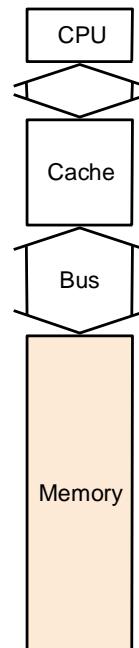
Memoria Caché

Memoria Principal

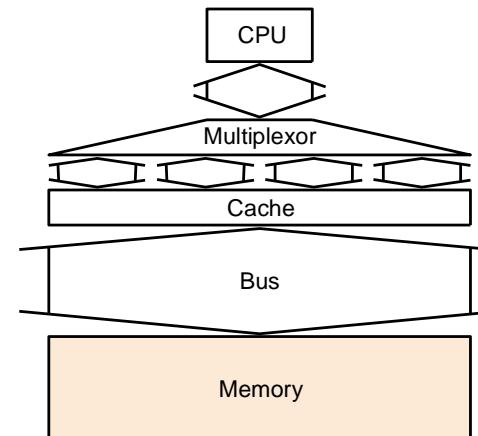
Memoria Virtual

Memoria

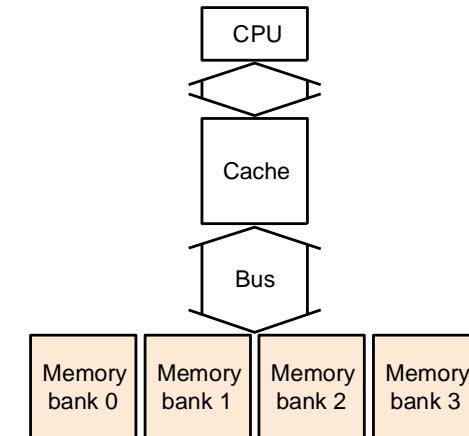
- Generalmente resulta más fácil mejorar el ancho de banda que reducir latencia.
- Una mejora en el ancho de banda permite incrementar tamaño de bloques sin incremento de penalización de fallos.



a. One-word-wide  
memory organization



b. Wide memory organization



c. Interleaved memory organization

# Memoria principal más ancha

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Las cachés están organizadas normalmente con una anchura de una palabra
- ➋ La MP tiene la anchura de una palabra para que coincida con anchura de la caché.
- ➌ Duplicar o cuadruplicar el ancho de la memoria, duplicará o cuadruplicará el ancho de banda de memoria.

# Memoria principal más ancha

## ◆ Inconvenientes:

- ◆ Hay coste en el bus más ancho. Se necesita un multiplexor entre caché y CPU ya que se accede a una palabra cada vez. Si la caché es más rápida que el bus se puede colocar el multiplexor entre la caché y el bus.
- ◆ Inconveniente al expandir la memoria por el usuario ya que el incremento mínimo se duplica o cuadriplica.
- ◆ Dificultades en la corrección de errores. Las memorias con corrección de errores tienen dificultades con las escrituras en una parte del bloque protegido. Muchos diseños de memoria más ancha separan la corrección de errores cada 32 bits, ya que la mayor parte de las escrituras tienen ese tamaño.

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

# Memoria entrelazada

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Los chips de memoria se pueden organizar en bancos para leer o escribir múltiples palabras a la vez en lugar de una sola palabra.
- ➋ Los bancos son de una palabra de ancho para que la anchura del bus y de la caché no necesiten cambiar pero se envían direcciones a varios bancos lo que les permite a todos leer simultáneamente.
- ➌ Los bancos útiles también en las escrituras, permiten un ciclo de reloj para cada escritura, siempre que no estén destinadas al mismo banco. (Si hay muchas escrituras seguidas, normalmente tendrán que esperar que acaben las escrituras anteriores)

# Memoria entrelazada

Introducción

Jerarquía de memoria

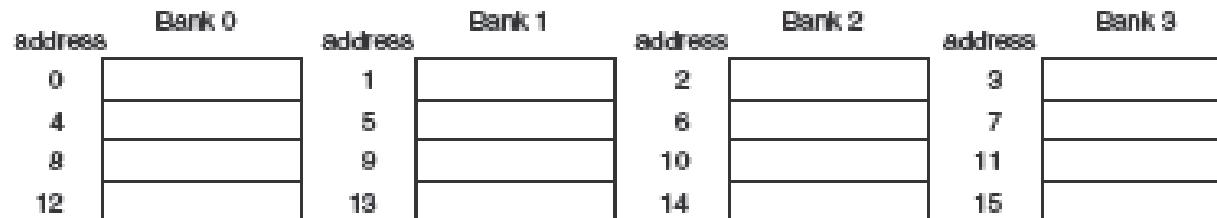
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ La correspondencia entre direcciones y bancos afecta al comportamiento del sistema de memoria. Esta correspondencia se llama *factor de entrelazado*.
- ➋ El entrelazado optimiza los accesos secuenciales a la memoria
  - ➌ Un fallo de lectura de la caché es un caso ideal para una lectura entrelazada a nivel de palabra ya que las palabras se leen secuencialmente.
  - ➍ Las cachés de postescritura realizan lecturas y escrituras de forma secuencial.
- ➎ En la figura 4 bancos entrelazados a nivel de palabra.



# Memoria entrelazada

## ◆ Inconvenientes.

- ◆ Al aumentar la capacidad por chip de memoria habrá menos chips en un sistema de memoria del mismo tamaño, con lo que tener múltiples bancos será más caro.
- ◆ La dificultad de expansión de la memoria principal. El hardware de control necesitará bancos del mismo tamaño, por lo que el incremento mínimo será probablemente duplicar la memoria.

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

# Ejemplo 1

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

◆ Estudiemos estas organizaciones mostrando cómo se satisface un fallo de caché.

◆ Supongamos el siguiente rendimiento para una organización básica de memoria:

- ◆ 4 ciclo de reloj para enviar dirección
- ◆ 24 ciclos de reloj para tiempo de acceso por palabra
- ◆ 4 ciclos de reloj para enviar una palabra de datos
- ◆ Bloque de caché de 4 palabras
- ◆ Palabra de 32 bits

◆ Para la organización de una palabra de ancho, la penalización de fallos es de  $4 \times (4 + 24 + 4) = 128$  ciclos de reloj por bloque, con un ancho de banda de memoria de 1/8 de byte (16/128) por ciclo de reloj.

# Ejemplo 1

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ Supongamos el siguiente rendimiento para una organización básica de memoria:

- ◆ 4 ciclo de reloj para enviar dirección
- ◆ 24 ciclos de reloj para tiempo de acceso por palabra
- ◆ 4 ciclos de reloj para enviar una palabra de datos
- ◆ Bloque de caché de 4 palabras
- ◆ Palabra de 32 bits

- ◆ Para la organización de memoria ancha con un ancho de dos palabras, la penalización de fallos es de  $2 \times (4 + 24 + 4) = 64$  ciclos de reloj por bloque con un ancho de banda de 1/4 byte/ciclo de reloj. Con un ancho de 4 palabras, la penalización de fallos será 1X32 ciclos de reloj por bloque con un ancho de banda de 1/2 byte/ciclo de reloj.

# Ejemplo 1

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Supongamos el siguiente rendimiento para una organización básica de memoria:
  - ➊ 4 ciclo de reloj para enviar dirección
  - ➋ 24 ciclos de reloj para tiempo de acceso por palabra
  - ➌ 4 ciclos de reloj para enviar una palabra de datos
  - ➍ Bloque de caché de 4 palabras
  - ➎ Palabra de 32 bits
  
- ➋ Para la organización de memoria entrizada con cuatro bancos, enviando una dirección a los cuatro bancos a la vez, la penalización de fallos es de  $4+24+4\times4=44$  ciclos de reloj por bloque con un ancho de banda de unos 0,4byte/ciclo de reloj.

# Ejemplo 2

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

Considerar la siguiente descripción de una máquina y su rendimiento:

- Tamaño del bloque: 1 palabra
- Anchura del bus de memoria: 1 palabra
- Frecuencia de fallos: 3%
- Accesos a memoria por instrucción: 1.2
- Penalización de fallos de caché: 32 ciclos
- Ciclos medios por instrucción (ignorando los fallos de la caché): 2

Si cambiamos el tamaño de bloque a dos palabras, la frecuencia de fallos cae al 2%, y un bloque de cuatro palabras tiene una frecuencia de fallos del 1%. ¿Cuál es la mejora del rendimiento al entrelazar dos y cuatro vías frente a duplicar el ancho de memoria y del bus, suponiendo los tiempos de acceso del ejemplo anterior?

# Solución ejemplo 2

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- EL CPI para la máquina base utilizando bloques de una palabra es:

$$CPI = CPI_{ejec} + \frac{\text{Accesos a memoria (NM)}}{\text{Instrucción (NI)}} \cdot FF * PF$$

$$CPI = 2 + (1.2 \cdot 3\% \cdot 32) = 3.15$$

- Como la duración del ciclo de reloj y el recuento de instrucciones n puede cambiar en este ejemplo, podemos calcular la mejora de rendimiento comparando el CPI.
- Incrementar el tamaño de bloque a dos palabras:
  - Bus y memoria de 32 bits, sin entrelazado  $\rightarrow CPI = 2 + [1.2 \cdot 10\% \cdot (2 \cdot 32)] = 3.54$
  - Bus y memoria de 32 bits, con entrelazado  $\rightarrow CPI = 2 + [1.2 \cdot 2\% \cdot (4 + 24 + 8)] = 2.86$
  - Bus y memoria de 64 bits, sin entrelazado  $\rightarrow CPI = 2 + [1.2 \cdot 2\% \cdot (1 \cdot 32)] = 2.77$
- Duplicar el tamaño del bloque ralentiza la implementación, mientras que el entrelazado o la memoria ancha lo hace más rápido, 1.10 y 1.14 veces más rápido respectivamente.

# Solución ejemplo 2

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ◆ EL CPI para la máquina base utilizando bloques de una palabra es:

$$CPI = CPI_{ejec} + \frac{\text{Accesos a memoria (NM)}}{\text{Instrucción (NI)}} \cdot FF * PF$$

$$CPI = 2 + (1.2 \cdot 3\% \cdot 32) = 3.15$$

- ◆ Incrementar el tamaño de bloque a 4 palabras:

- ◆ Bus y memoria de 32 bits, sin entrelazado  $\rightarrow CPI = 2 + [1.2 \cdot 1\% \cdot (4 \cdot 32)] = 3.54$
- ◆ Bus y memoria de 32 bits, con entrelazado  $\rightarrow CPI = 2 + [1.2 \cdot 1\% \cdot (4 + 24 + 16)] = 2.53$
- ◆ Bus y memoria de 64 bits, sin entrelazado  $\rightarrow CPI = 2 + [1.2 \cdot 1\% \cdot (2 \cdot 32)] = 2.77$

- ◆ De nuevo, un tamaño del bloque más grande disminuye el rendimiento del caso simple, aunque el el entrelazado de 32 bits es ahora el más rápido, 1.25 veces más rápido frente a 1.14 veces más rápido de la memoria y bus más ancho.

# Bancos de memoria independientes

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Una generalización del entrelazado es permitir múltiples accesos independientes, donde múltiples controladores de memoria permitan a los bancos operar independientemente.
- ➋ Cada banco necesita líneas de dirección separadas y posiblemente un bus de datos separado.
- ➌ Por ejemplo, un dispositivo de entrada puede utilizar un controlador y un banco, la lectura en caché puede utilizar otro banco y la escritura en caché un tercer banco.
- ➍ Se utiliza el término *superbanco* para referirse a toda la memoria activa en una transferencia de un bloque y el término banco para referirse a la porción dentro de un superbanco que está entrelazada a nivel de palabra.

# Evitar conflictos en bancos de memoria

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Si el sistema de memoria se diseña para soportar múltiples peticiones independientes, la efectividad del sistema va a depender de la frecuencia con que las peticiones independientes vayan a distintos bancos.
- ➋ Para evitar problemas de conflicto al acceder al mismo banco hay dos posibles soluciones:
  - ➌ Solución Software: El compilador puede realizar optimizaciones. Puede por ejemplo expandir el tamaño de los arrays para que no sean potencias de 2, forzando a que las direcciones pertenezcan a distintos bancos.
  - ➍ Solución Hardware: Una solución es tener un número primo de bancos. Esto puede complicar el cálculo de la dirección del módulo pero hay muchas técnicas rápidas para calcular el módulo.

## **5.5. Memoria Virtual**

---

**Tema 5 Rendimiento de la jerarquía de memoria**

**Arquitectura de los Computadores**

# Memoria virtual

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## 5.5. MEMORIA VIRTUAL

5.3.1 Organización de la caché

5.3.2 Ubicación de bloque

5.3.3 Estrategias de reemplazo

5.3.4 Operación de la caché

5.3.5 Políticas de escritura

5.3.6 Rendimiento

# Introducción

Introducción

Jerarquía de memoria

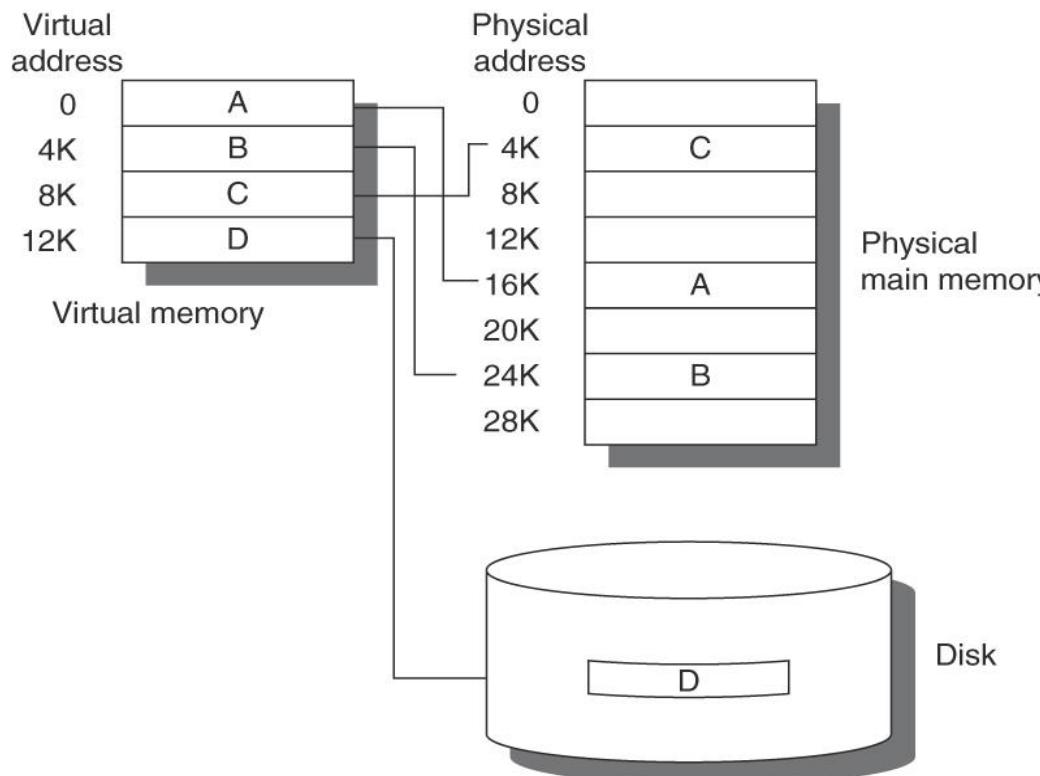
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- La memoria virtual gestiona automáticamente los niveles de la jerarquía de memoria representada por la memoria principal y la secundaria.



# Conceptos generales

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ La memoria virtual permite:
  - ➊ En un instante de tiempo los computadores estén corriendo múltiples procesos, cada uno de ellos con su propio espacio de direcciones.
  - ➋ La existencia de un esquema de protección de la memoria para cada proceso.
  - ➋ Las necesidades de memoria de todos los programas de un sistema sean superiores a la memoria física disponible.
  - ➋ Compartir una cantidad de memoria física pequeña entre muchos procesos.
  - ➋ Simplifica la carga del programa para su ejecución (reubicación).

# Conceptos generales

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ La memoria virtual divide la memoria física en bloques (**páginas**) y los asigna a diferentes procesos.
- ➋ Cada proceso tiene un espacio virtual propio dividido en páginas.
- ➌ Las páginas pueden estar en memoria principal o en disco.
- ➍ La CPU genera **direcciones virtuales**
- ➎ Un combinación de hardware y software las traduce a direcciones físicas.
- ➏ Cuando una página no está en memoria principal se produce una **fallo** de página (o segmento) y debe traerse desde disco.
- ➐ El reemplazo de páginas está controlado principalmente por el sistema operativo.

# Conceptos generales

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Dos clases de memoria virtual:
  - ➊ Bloques de tamaño fijo denominados **páginas**. Típicamente entre 4096 bytes y 8192 bytes.
  - ➋ Bloques de tamaño variable denominados **segmentos**. El segmento mayor varía entre  $2^{16}$  bytes hasta  $2^{32}$  bytes; el más pequeño e de 1 byte.
- ➋ Debido a que el reemplazo es difícil pocas máquinas utilizan la segmentación pura.
- ➋ Algunas máquinas utilizan un enfoque híbrido llamado de segmentos paginados.

# ¿Dónde puede ubicarse un bloque en MP?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ La penalización de fallos es bastante alta puesto que acceden a disco.
- ➋ Es muy importante reducir los fallos de página.
- ➌ Los SO permiten la asignación completamente asociativa.

# ¿Cómo se encuentra un bloque si está en MP?

Introducción

Jerarquía de memoria

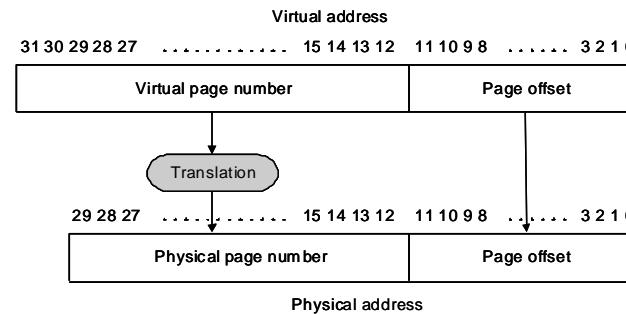
Memoria Caché

Memoria Principal

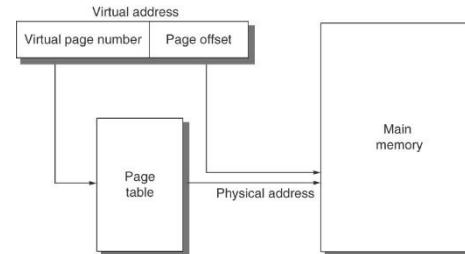
Memoria Virtual

Memoria

- La dirección virtual se descompone en dos campos: número de página virtual y desplazamiento de página.
- La dirección física se obtiene concatenando la dirección física de la página con el desplazamiento de página.



- Se utiliza una estructura de datos (tabla de páginas) que contiene la dirección física de la página y es indexada por la dirección virtual.



- La tabla de páginas reside en memoria principal.

# Tabla de páginas

Introducción

Jerarquía de memoria

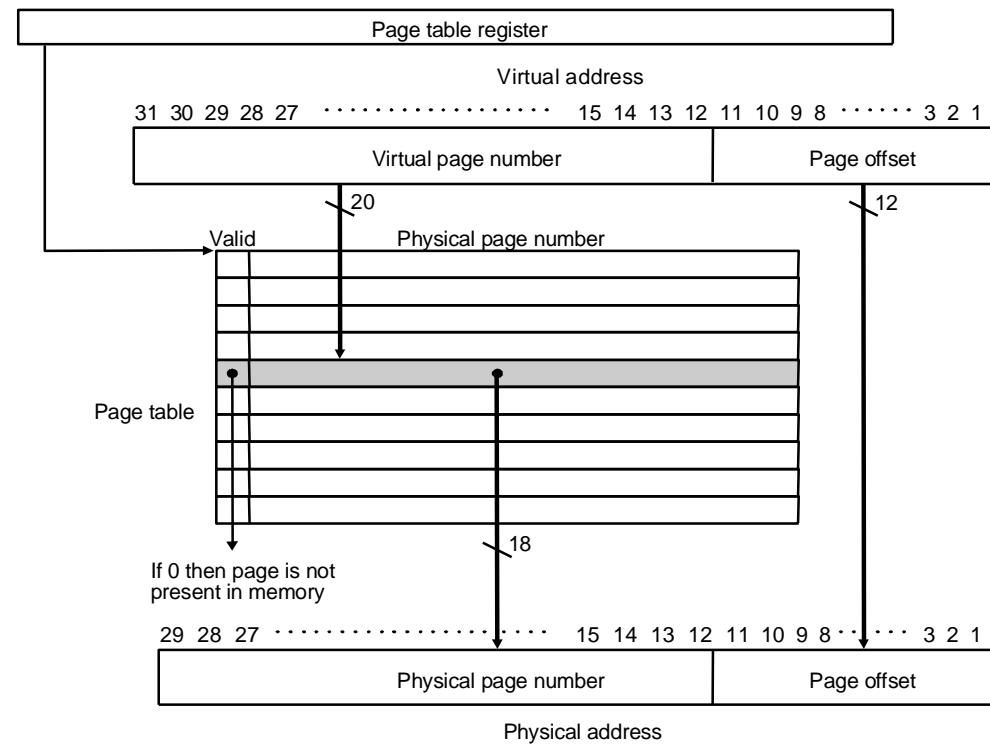
Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

## Correspondencia entre dirección virtual y física



# ¿Qué bloque debe sustituirse en un fallo de memoria virtual?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Se pretende minimizar los fallos de página.
- ➋ Casi todos los SO intentan sustituir el bloque LRU (menos recientemente utilizado)
- ➌ Muchas máquinas proporcionan un bit de uso o referencia para este propósito.

# ¿Qué ocurre en una escritura?

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

- ➊ Usar escritura directa (write-through) es demasiado caro por tanto se usará postescritura (writeback).
- ➋ Los sistemas de memoria virtual incluyen un bit de modificación (dirty).

# Traducción rápida de direcciones

Introducción

Jerarquía de memoria

Memoria Caché

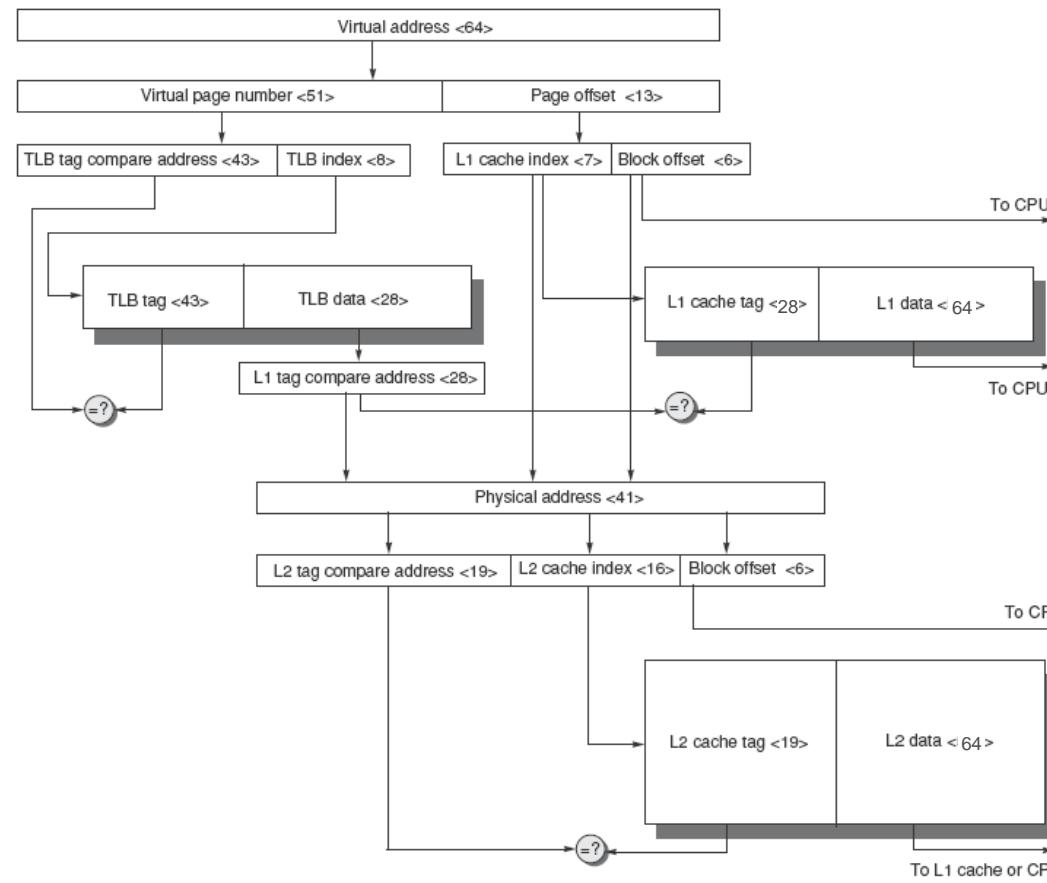
Memoria Principal

Memoria Virtual

Memoria

- ➊ Disponer de una caché para la traducción de direcciones: translation lookaside buffer (TLB)
- ➋ Una entrada a la TLB es como una entrada de la caché donde la etiqueta contiene parte de la dirección virtual y la parte del dato contiene un número de estructura de página, campo de protección, bit de uso y bit de modificación.
- ➌ La dirección virtual debe ir primero a la TLB antes que la dirección física pueda acceder la caché. Lo que significa que el tiempo de acierto se alarga.
- ➍ Se puede reducir el tiempo de acierto al acceder a la caché con el desplazamiento de página. La comparación de direcciones se realiza entre la dirección física del TLB y la etiqueta de la caché.
  - ➎ El inconveniente es que una caché con correspondencia directa no puede ser mayor de una página.

# Ejemplo de jerarquía de memoria



# Ejemplos: Pentium Pro y PowerPC 604

Introducción

Jerarquía de memoria

Memoria Caché

Memoria Principal

Memoria Virtual

Memoria

Características	Pentium Pro	PowerPc 604
Dirección virtual	32 bits	52 bits
Dirección física	32 bits	32 bits
Tamaño de página	4KB, 4MB	4KB, 256MB
Organización TLB	TLB para datos TLB para instrucciones Asociativas de 4 vías Reemplazo pseudo LRU TLB-i 32 entradas TLB-d 64 entradas Fallos TLB tratados por Hardware	TLB para datos TLB para instrucciones Asociativas de 2 vías Reemplazo LRU TLB-i 128 entradas TLB-d 128 entradas Fallos TLB tratados por Hardware
Organización caché	Cachés de datos e instrucciones separadas	Caches de datos e instrucciones separadas
Tamaño caché	8KB cada una	16KB cada una
Asociatividad	Asociativa de 4 vías	Asociativas de 4 vías
Reemplazo	LRU aproximado	LRU
Tamaño bloque	32 byte	32 byte
Actualización	Aplazada	Aplazada o inmediata