

1. Explica la utilidad del buffer de renombrado y enumera los tipos que hay según su direccionamiento.

El **buffer de renombrado** es una técnica para **evitar el efecto de las dependencias WAR y WAW**.

Existen dos tipos de buffer de renombrado:

- Con **acceso asociativo**.

- Permite varias escrituras pendientes a un mismo registro.
- Se utiliza el bit último para marcar cuál ha sido la escritura más reciente.

- Con **acceso indexado**.

- Sólo permite una escritura pendiente a un mismo registro.
- Se mantiene la escritura más reciente.

2. Explica la diferencia entre predicción dinámica explícita y predicción dinámica implícita.

- **Predicción dinámica explícita.**

Para cada instrucción de salto existen unos bits específicos que codifican la información de historia de dicha instrucción de salto.

- **Predicción dinámica implícita.**

No hay bits de historia propiamente dichos, sino que se almacena la dirección de la instrucción que se ejecutó después de la instrucción de salto en cuestión.

3. Justifica la diferencia que existe entre multicomputadores y multiprocesadores en términos de latencia y escalabilidad.

En los **multicomputadores**, la **latencia** en el acceso a memoria es **menor que** en los **multiprocesadores**. Esto es debido a que el acceso concurrente a memoria compartida por parte de los multiprocesadores provoca que el tiempo de lectura y escritura en memoria aumente.

En los **multiprocesadores**, la **escalabilidad** respecto al número de núcleos es **menor que** en los **multicomputadores**. Esto es debido a que el rendimiento de los programas paralelos no aumenta en la misma proporción que el aumento de núcleos, debido a los conflictos de acceso a memoria.

4. ¿Cuál es la característica distintiva de las redes de interconexión dinámicas frente a otros tipos de redes de interconexión?

A diferencia de otros tipos de redes de interconexión, las **redes de interconexión dinámicas** se caracterizan porque **pueden variar su topología** durante la ejecución de los procesos o entre la ejecución de dos procesos.

5. Indica a qué topología de la izquierda corresponden los conceptos de la derecha ($N = \text{número de nodos}$). Subraya las topologías que sean directas y a la vez irregulares.

A. Malla Illiac

1. Grado = 2

B. Barril

2. $F_{-1}(i) = (i - 1) \bmod r + (i \bmod r) \cdot r, N = r \cdot r$

C. Anillo

3. Ortogonal

D. Crossbar

4. $F_{-1}(i) = (i - 1) \bmod r <> 0, N = r \cdot r$

E. Toro

5. Diámetro = $q - 1, N = q \cdot q$

F. Malla abierta

6. Grado = 3

G. Hipercubo

7. Diámetro = $\frac{n}{2}, n = \log N$

H. Árbol binario

8. Indirecta

A → 5, B → 7, C → 1, D → 8, E → 2, F → 4, G → 3, H → 6

* En rojo las topologías **directas e irregulares**.

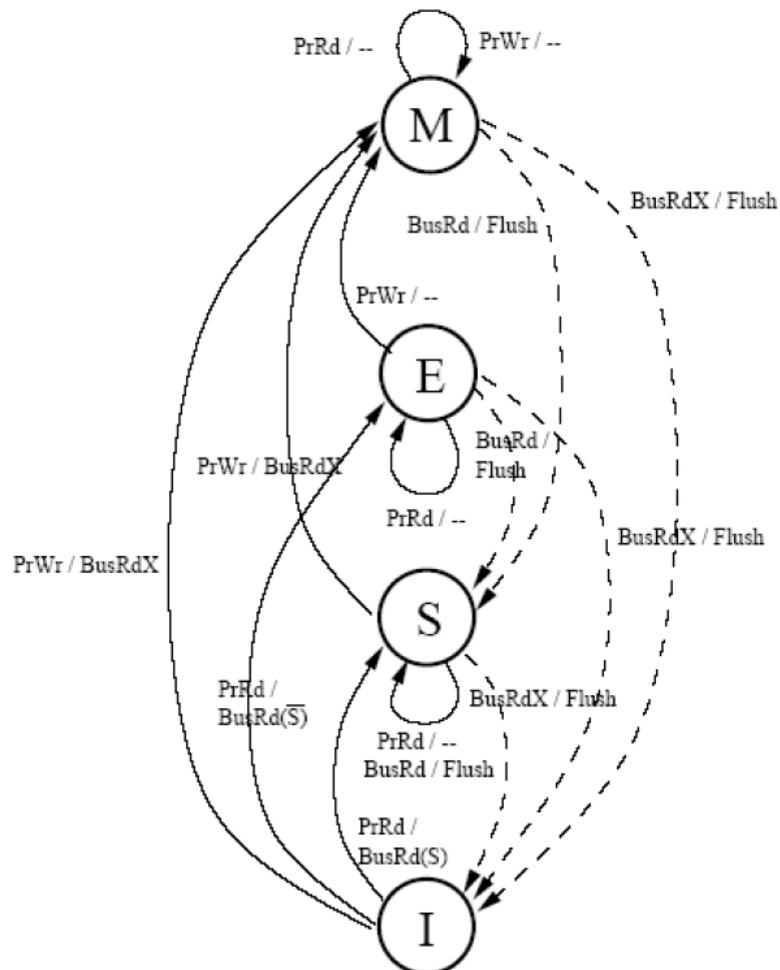
6. Explica en qué consiste una arquitectura vectorial.

Una arquitectura vectorial es una arquitectura orientada al procesamiento de vectores.

Esta arquitectura utiliza un repertorio de instrucciones especializado y se caracteriza por:

- Cálculo de los componentes del vector de forma independiente, obteniendo buenos rendimientos.
- Cada operación vectorial codifica gran cantidad de cálculos, reduciendo el número de instrucciones y evitando riesgos de control.
- Se optimiza el uso de memoria, usando entrelazado de memoria y organizaciones S y C.

7. Dibuja el diagrama de transiciones del protocolo de coherencia de caché MESI.



Julio 2013

1. Explica la utilidad de la ventana de instrucciones y explica brevemente cómo puede ser el orden de emisión y el alineamiento de la ventana.

La **ventana de instrucciones** se emplea para **almacenar las instrucciones pendientes**. Las instrucciones se cargan en la ventana una vez decodificadas y se utiliza un bit para indicar si un operando está disponible o no. Una instrucción puede ser emitida cuando tiene todos sus operandos disponibles y la unidad funcional donde se procesará.

El orden de emisión de las instrucciones puede ser:

- **Emisión Ordenada.** Las instrucciones se emiten en el mismo orden en el que entran a la ventana. Se espera hasta que la instrucción esté preparada para ser emitida.
- **Emisión Desordenada.** Las instrucciones se emiten sin orden alguno. Se emite la que esté preparada. Sin esperar a una en particular.

El alineamiento de la ventana de instrucciones puede ser:

- **Emisión Alineada.** No se reciben nuevas instrucciones hasta que no se vacía la ventana de instrucciones.
- **Emisión Desalineada.** Se pueden recibir nuevas instrucciones siempre que haya sitio.

2. Explica brevemente las estructuras que permiten la gestión de predicción de los saltos dinámica explícita.

- **Branch Target Buffer (BTB)**: bits acoplados.

Almacena la dirección de destino de los últimos saltos tomados y los bits de predicción de ese salto. Los campos de la BTB se actualizan después de ejecutar el salto, cuando se conoce si el salto fue tomado o no y la dirección de destino del salto. La predicción implícita no emplea bits de predicción, el problema de esto es que sólo se pueden predecir aquellas instrucciones de salto que están en la BTB.

- Tabla de historia de saltos (BHT): bits desacoplados.

2 tablas:

- BTAC. Almacena la dirección de destino de los últimos saltos tomados.
- BHT. Almacena los bits de predicción de todas las instrucciones de salto condicional.

La ventaja de esta estructura es que puede predecir instrucciones que no están en la BTAC, sin embargo, necesita más hardware.

- Bits de predicción en la caché.

Cuando se capta una instrucción de la caché, si se trata de una instrucción de salto condicional, accede en paralelo a los bits de predicción y si el salto se predice como tomado se accede a la instrucción destino del salto. Para acceder a la instrucción destino del salto se usa una BTB independiente a la que se añade el índice sucesor a la l-cache.

Las ventajas de esta estructura son que se puede predecir instrucciones que no están en la BTB y no añade una cantidad extra de hardware excesiva.

3. Describe ejemplos de problemas, cuya solución consista en un conjunto de procesos o hebras que se comunican mediante cada uno de los siguientes patrones:

- a) Difusión (*broadcast*).
- b) Dispersión (*scatter*).

La difusión o *broadcast* consiste en un ordenador maestro que envía un mismo mensaje a todas las máquinas hijo. Un ejemplo de esto sería las actualizaciones del sistema operativo que un ordenador maestro envía a los ordenadores de sus clientes por medio de una OTA (*Over-The-Air*).

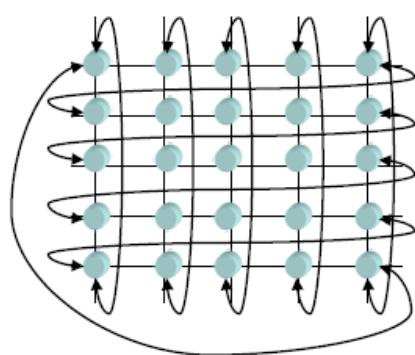
La dispersión o *scatter* consiste en un ordenador maestro que divide un mensaje en partes y envía una parte diferente a cada una de las máquinas hijo. Un ejemplo de esto sería la suma de las componentes de dos vectores, de esta forma, cada máquina hijo tomaría una componente de cada vector y realizaría la suma.

4. ¿Cuál es la característica distintiva de las redes de interconexión directas frente a los otros tipos de redes de interconexión (indirectas y de medio compartido)?

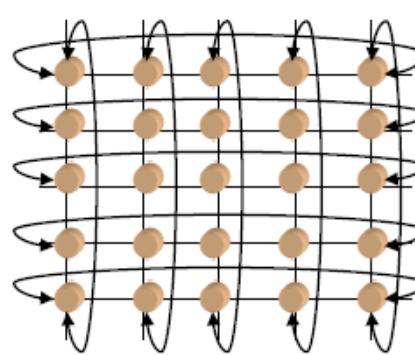
A diferencia de otros tipos de redes de interconexión, las **redes de interconexión directas** emplean enlaces directos fijos entre los nodos mientras que las redes de interconexión indirectas y de medio compartido están formadas por enlaces no permanentes que se reconfiguran en función de la demanda.

5. Indica el nombre de dos topologías de interconexión que cumplan las siguientes características, y dibuja el grafo de cada una de ellas:

- Directa
- Grado 4
- Regular



Malla Illiac



Toro

6. Explica brevemente qué tipos de paralelismo existen y en qué niveles puede aplicarse.

Existen dos tipos de paralelismo:

- **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.
- **Paralelismo funcional.** Varias funciones, tareas, instrucciones, etc (iguales o distintas) 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). 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.

7. ¿Qué diferencia existe entre el protocolo de coherencia de caché MESI y MSI? ¿Qué ventajas tiene uno de ellos sobre el otro?

- MSI. Protocolo de invalidación básico que usa tres estados necesarios en cualquier caché post-escritura para distinguir bloques válidos que no han sido modificados de aquellos que han sido modificados.
Está compuesto por 3 estados: Inválido (I), Compartido (S) y Modificado (M).
- MESI. Protocolo de invalidación de 4 estados. Refinado para aplicaciones “secuenciales” que corren en multiprocesadores. MESI añade el estado Exclusivo (E), que indica que el bloque es la única copia (exclusiva) del sistema multiprocesador y que no está modificado, ningún otro procesador tiene el bloque en la caché y la memoria principal está actualizada.

La ventaja de MESI sobre MSI es que, al ser exclusivo, es posible realizar una escritura o pasar al estado modificado sin ninguna transición en el bus, al contrario que en el caso de estar en el estado compartido; pero no implica pertenencia, así que al contrario que en el estado modificado la caché no necesita responder al observar una petición de dicho bloque. En el MSI el programa cuando lee y modifica un dato tiene que generar 2 transacciones incluso en el caso de que no exista compartición (sólo presente en una caché) del dato.

9. Dos grupos de ingenieros están trabajando en la paralelización de una determinada aplicación. Ambos grupos resuelven que el 20% de la aplicación no es paralelizable. Sin embargo, el primer grupo decide que, para el resto, el 25% es paralelizable para cualquier número de nodos y el 75% sólo hasta la mitad del número máximo de nodos disponible. El segundo grupo de ingenieros decide, que para la parte paralelizable, se puede paralelizar el 30% para cualquier número de nodos y el 70% sólo hasta un tercio del número de nodos máximo.

Suponiendo que la aplicación va a correr en un multicomputador con N nodos iguales, y que la sobrecarga de comunicación es del 15% del tiempo de ejecución en paralelo en cada caso, decida cuál es la mejor solución en función de la eficiencia (argumentando así que se trata de la máquina que mejor compromiso tiene entre número de nodos y ganancia en velocidad).

$$S_P = \frac{T(1)}{T(P) + Q(P)}$$

Grupo 1:

$$\begin{aligned} Spedup_1 &= \frac{T_s}{\frac{1}{5} \cdot T_s + \frac{4}{5} \cdot \left(\frac{1}{4} \cdot \frac{T_s}{N} + \frac{3}{4} \cdot \frac{1}{2} \cdot \frac{T_s}{N} \right) + \frac{3}{20} \cdot \frac{T_s}{N}} \rightarrow \frac{T_s}{\frac{1}{5} \cdot T_s + \frac{4}{20} \cdot \frac{T_s}{N} + \frac{6}{20} \cdot \frac{T_s}{N} + \frac{3}{20} \cdot \frac{T_s}{N}} \rightarrow \frac{T_s}{T_s \cdot \left(\frac{1}{5} + \frac{4}{20N} + \frac{6}{20N} + \frac{3}{20N} \right)} \\ &\rightarrow \frac{1}{\frac{4N+13}{20N}} = \boxed{\frac{20N}{4N+13}} \end{aligned}$$

Grupo 2:

$$Spedup_2 = \frac{T_s}{\frac{1}{5} \cdot T_s + \frac{4}{5} \cdot \left(\frac{3}{10} \cdot \frac{T_s}{N} + \frac{7}{10} \cdot \frac{1}{3} \cdot \frac{T_s}{N} \right) + \frac{3}{20} \cdot \frac{T_s}{N}} \rightarrow \frac{T_s}{\frac{1}{5} \cdot T_s + \frac{12}{50} \cdot \frac{T_s}{N} + \frac{28}{150} \cdot \frac{T_s}{N} + \frac{3}{20} \cdot \frac{T_s}{N}}$$
$$\rightarrow \frac{T_s}{T_s \cdot \left(\frac{1}{5} + \frac{12}{50N} + \frac{28}{150N} + \frac{3}{20N} \right)} \rightarrow \frac{1}{\frac{60N + 72 + 56 + 45}{300N}} = \frac{300N}{60N + 173}$$

Para $N = 4$:

Grupo 1:

$$\frac{20 \cdot 4}{4 \cdot 4 + 13} = 2.7586$$

$$Eficiencia = \frac{2.7586}{4} = 0.6897$$

Tras calcular el speedup y la eficiencia obtenida por cada grupo podemos concluir que el grupo 2 obtiene un mayor speedup y una mayor eficiencia que el grupo 1, por tanto, podemos concluir que la máquina del grupo 2 tiene mejor compromiso entre número de nodos y ganancia en velocidad que la máquina del grupo 1.

Grupo 2:

$$\frac{300 \cdot 4}{60 \cdot 4 + 173} = 2.9056$$

$$Eficiencia = \frac{2.9056}{4} = 0.7264$$

Enero 2014

1. Explica brevemente cuáles son los tipos de paralelismo que podemos encontrar en un sistema informático.

- **Paralelismo funcional.** Se obtiene a través de la reorganización de la estructura lógica de una aplicación. Existen diferentes niveles de paralelismo funcional según las estructuras que se reorganicen.
- **Paralelismo de datos.** Implícito en operaciones con estructuras de datos tipo vector o matriz. Está relacionado con operaciones realizadas sobre grandes volúmenes de datos que sean independientes entre sí.
 - **Paralelismo explícito.** Paralelismo no presente de forma inherente en las estructuras de programación y que se debe indicar expresamente.
 - **Paralelismo implícito.** Paralelismo presente (aunque puede no estar ejecutándose de forma paralela) debido a la propia estructura de los datos (vectores) o de la aplicación (bucle).

2. ¿Existe alguna diferencia entre el acceso a memoria concurrente y el acceso simultáneo en una máquina vectorial? Si es así, explícalo muy brevemente.

La diferencia entre el acceso a memoria concurrente y el acceso simultáneo radica en que en el acceso concurrente cada módulo puede trabajar por separado, de forma asíncrona, leyendo direcciones diferentes y cada módulo tiene su propio registro de dirección y puede ir trabajando paralelamente a los demás.

El acceso simultáneo se basa en simultanejar los accesos en todos los módulos y, tratar de superponer ese tiempo con la salida de los datos de memoria.

3. ¿Cuáles son las similitudes y las diferencias entre el buffer de renombrado y el buffer de reorden?

Similitudes:

En los procesadores con finalización desordenada, se pueden producir riesgos de dependencias WAR o WAW (*Write After Read* o *Write after Write*), y para minimizar el impacto que provocarían en la ejecución del programa haremos uso del renombrado de registros y reorden de los mismos, y dentro de ellos existen dos tipos:

- **Implementación Dinámica.** Se realizan durante la ejecución y requieren circuitería adicional.
- **Implementación Estática.** Se realiza durante la compilación.

Diferencias:

El buffer de renombrado se encarga de modificar el nombre de las instrucciones, en cambio, el buffer de reorden se encarga de modificar el orden de ejecución de las instrucciones.

4. Explica en qué consiste el procesamiento especulativo de los saltos.

El procesamiento especulativo es una manera de gestionar los saltos condicionales no resueltos. Se realiza una tarea que podría no ser necesaria; la idea consiste en llevar a cabo un trabajo antes de saber si será necesario con la intención de evitar el retraso que supondría realizarlo después de saber que sí es necesario. Si el trabajo en cuestión resulta ser innecesario, se revierten los cambios realizados por ese trabajo y se ignoran los resultados obtenidos.

5.

a) Defina y enumere las diferencias entre proceso y una hebra.

Proceso. Es una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.

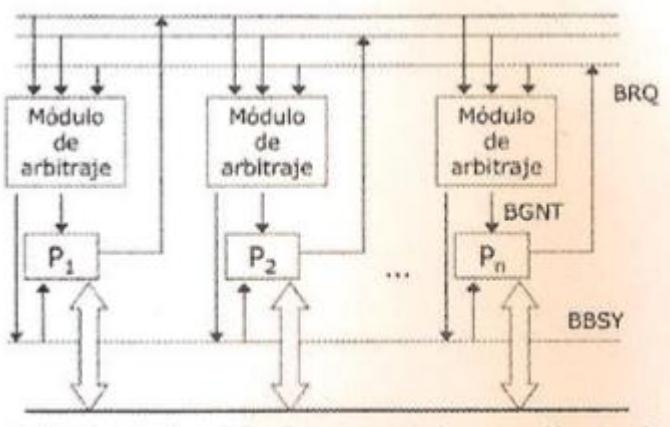
Cada proceso tiene un espacio de direcciones virtuales propio.

Hebra. Secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo.

Las hebras comparten direcciones virtuales, se crean y destruyen más rápido y la comunicación también es más rápida que en los procesos.

- b) Hablando de paralelismo, ¿qué tipo de computador paralelo está asociado a cada uno de estos dos conceptos? ¿Por qué?
- c) ¿Puede paralelizar una aplicación usando al mismo tiempo distintos procesos y distintas hebras? Indique, si es el caso, un ejemplo real de arquitectura que dé soporte a estos niveles de paralelismo.

6. La imagen de la figura indica un tipo de arbitraje para un cierto tipo de redes de interconexión.



a) ¿De qué tipo de redes se trata? ¿La prioridad es estática o dinámica? ¿Por qué?

Se trata de una red de **medio compartido** (arbitraje del bus), en concreto, se trata de **autoarbitraje** (distribuido-paralelo).

La prioridad de la red es estática.

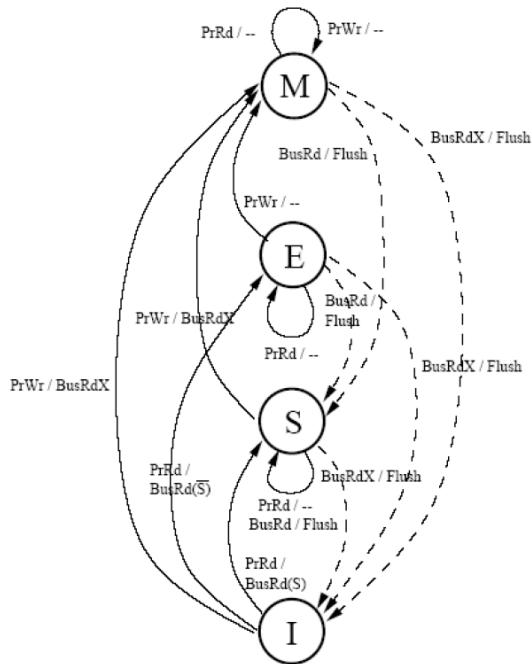
b) Explique pormenorizadamente la función de cada una de las señales.

7. Deducza el diámetro de una red directa de tipo árbol binario balanceado con 511 nodos.

$$2^k - 1 = 511 \text{ nodos} \rightarrow 2^k = 512 \rightarrow k = \log_2 512 \rightarrow k = 9$$

$$\text{Diámetro} = 2 \cdot (k - 1) = 2 \cdot (9 - 1) = \boxed{16}$$

8. ¿Qué tipo de operaciones del procesador o transiciones del bus hacen falta para pasar directamente (en un solo salto o transición) del estado M al estado E en el protocolo MESI? Si es el caso, ponga un ejemplo.



No existe ninguna operación que nos permita ir desde el estado M al estado E en un solo salto o transición. Lo único que podemos es ir desde el estado E al estado M por medio de la operación PrWr.

9. Calcula el tiempo que tarda en transmitirse un paquete formado por 34 bytes (30 bytes de datos + 4 bytes de cabecera) en un toro 4D de 65536 nodos desde el nodo 215 hasta el nodo 5063. El tiempo de enruteamiento es 14 ms y el tiempo de transmisión entre nodos es de 300 bytes/s. Compara los tiempos que se obtienen utilizando una estrategia "Wormhole" frente a una estrategia "Store and Forward".

9.)

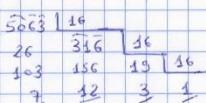
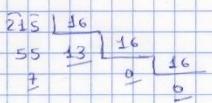
Paquete: 34 bytes \rightarrow 30 bytes datos ($L = 240 \text{ bits}$)
 \rightarrow 4 bytes cabecera ($W = 32 \text{ bits}$)

Toro 4D; $N = 65536$ nodos; $tr = 14 \text{ ms}$

Nodo origen = 215 = (0, 0, 13, 7)
Nodo destino = 5063 = (1, 3, 12, 7)

Transmisión entre nodos = $300 \frac{\text{bytes}}{\text{s}} \cdot \frac{8 \text{ bits}}{1 \text{ byte}} = 2400 \frac{\text{bits}}{\text{s}}$

$N = r^4 \rightarrow r = \sqrt[4]{N} = \sqrt[4]{65536} \rightarrow r = 16$



$D = |\text{origen} - \text{destino}| = |(0-1, 0+3, 13-12, 7-7)| = (1, 3, 1, 0)$
 $= 1+3+1+0 = 5$

$tw = \frac{32 \text{ bits}}{2400 \frac{\text{bits}}{\text{s}}} = 13'33 \text{ ms}$

Store and Forward:

$$TAR = D \cdot \left(tr + tw \cdot \left(\frac{L}{W} + 1 \right) \right) = 5 \cdot \left(14 + 13'33 \cdot \left(\frac{240}{32} + 1 \right) \right)$$

$$= 5 \cdot (14 + 13'33 \cdot 8'5) = \boxed{636'53 \text{ ms}}$$

Wormhole:

$$Tw = D \cdot (tr + tw) + tw \cdot \frac{L}{W} = 5 \cdot (14 + 13'33) + 13'33 \cdot \frac{240}{32}$$

$$= 5 \cdot 27'33 + 13'33 \cdot 7'5 = \boxed{236'63 \text{ ms}}$$

1. Explica brevemente qué es el paralelismo en computación. ¿Siempre mejora el rendimiento? (Explícalo poniendo ejemplos).

El paralelismo es una forma de computación en la cual varios cálculos pueden realizarse simultáneamente, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños, que son posteriormente solucionados en paralelo.

El paralelismo en computación no siempre mejora el rendimiento, cuando hay cargas computacionales de bajo coste, realizar la paralelización ralentiza el tiempo respecto a la ejecución del programa en secuencial.

2. ¿Para qué sirve el buffer de renombrado? ¿En qué etapa o etapas se usa? Pon un ejemplo descriptivo sencillo de su funcionamiento.

El buffer de renombrado sirve para evitar el efecto de las dependencias WAR, o Antidependencias (en la emisión desordenada) y WAW, o Dependencias de Salida (en la ejecución desordenada).

El buffer de renombrado se usa en la etapa de decodificación para leer los registros fuente que estén renombrados y para renombrar el registro destino. Al acabar la ejecución se escriben los resultados en el buffer de renombrado, y al acabar se escriben los resultados en los registros correspondientes.

3. ¿En qué consiste la predicción dinámica implícita? Explica cómo funciona poniendo un ejemplo.

La predicción dinámica consiste en que cada vez que se predice una misma instrucción de salto la predicción puede ser diferente según la historia previa (si se han tomado o no las anteriores ejecuciones).

La predicción implícita no contiene información en forma de bits de historia, sino que almacena la dirección destino de salto de la instrucción que se ejecutó después del salto.

4. ¿Cuál es la principal diferencia entre multicamputadores y multiprocesadores?

La principal diferencia entre un multiprocesador y un multicamputador es el número de equipos implicados en cada uno.

Un **multiprocesador** es una máquina que opera con varias CPU que comparten el **mismo espacio de memoria**.

Un **multicamputador** es un conjunto de ordenadores interconectados entre sí para funcionar como un único equipo. Cada procesador tiene su propio espacio de memoria.

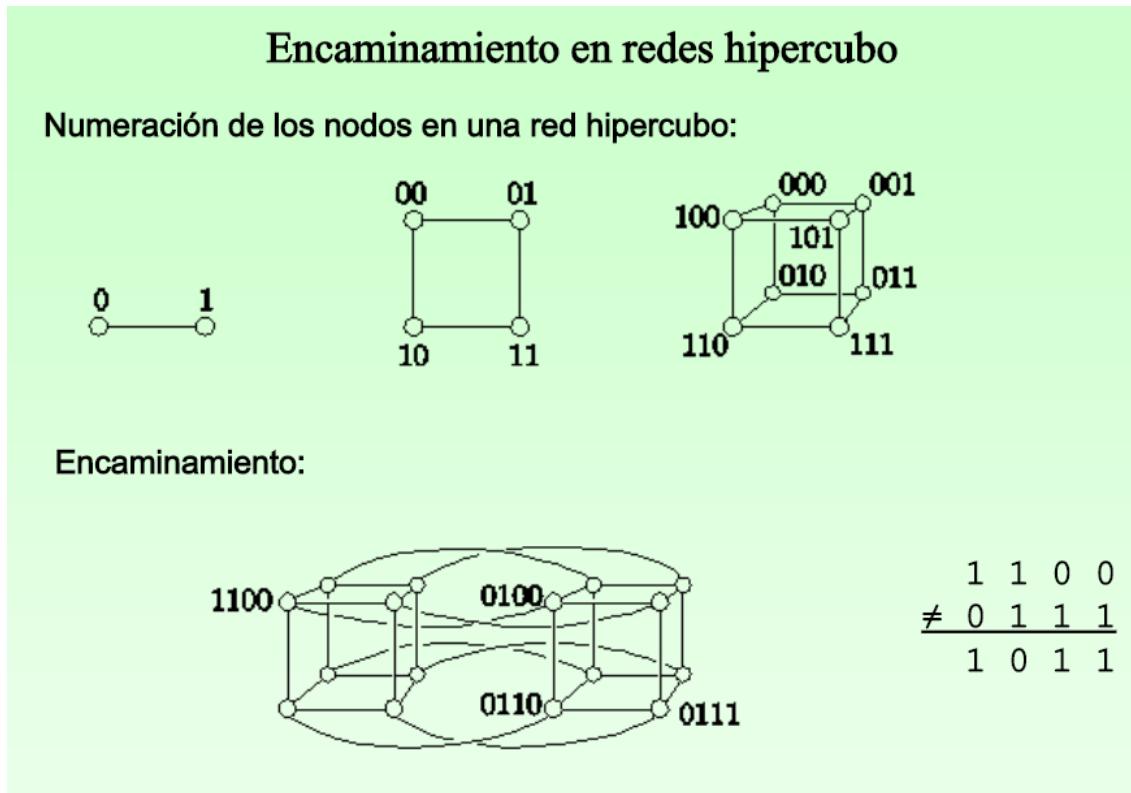
Multiprocesadores	Multicamputadores
Mayor latencia	Menor latencia
Poco escalable	Mayor escalabilidad
Comunicación: variables compartidas (datos no duplicados en memoria principal)	Comunicación: paso de mensajes (datos duplicados en memoria)
Necesita implementar primitivas de sincronización	Sincronización mediante mecanismos de comunicación
No necesita distribuir código y datos	Distribución de código y datos entre procesadores
Programación más sencilla	Programación más difícil

5. Explica brevemente los diferentes tipos de paralelismo que se pueden encontrar.

Existen dos tipos de paralelismo:

- **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.
- **Paralelismo funcional.** Varias funciones, tareas, instrucciones, etc (iguales o distintas) 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). 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.

6. Explica mediante un ejemplo detallado cómo funciona el algoritmo de encaminamiento de un hipercubo.



7. Explica cómo se distribuyen los bits necesarios para mantener la coherencia utilizando un protocolo basado en directorios. ¿Para qué sirve cada bit?

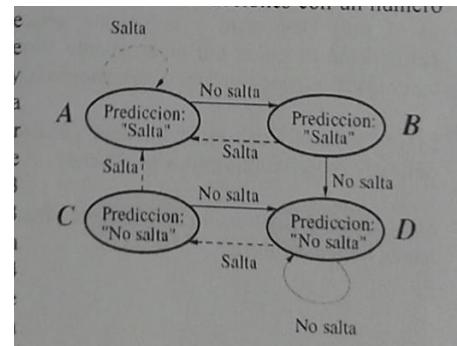
En el directorio: cada entrada de bloque tiene unos bits de presencia por cada caché y un bit de inconsistencia única:

- **Bits de presencia.** Especifican la presencia en las cachés de copias del bloque de memoria.
- **Bit de inconsistencia única.** Cuando este bit está activado, sólo uno de los bits de presencia está a uno, es decir, sólo existe una caché con la copia de ese bloque o línea, con lo que sólo esa caché tiene permiso para actualizar la línea.

Por cada caché:

- **Bits de validación (v).** Indica si la copia es válida o no.
- **Bit de privacidad (p).** Indica si la copia tiene permiso de escritura, es decir, cuando este bit es uno entonces es la única copia que existe de esta línea en las cachés y, por tanto, tiene permiso para escribir.

8. Suponer un computador superescalar que dispone un buffer de reorden, que permite resolver los riesgos WAR y WAW, y una ventana de instrucciones con un número de entradas suficiente. La arquitectura tiene adelantamientos. El procesador es capaz de decodificar 4 instrucciones por ciclo, emitir y completar 3 instrucciones por ciclo. Además, la emisión de las instrucciones puede ser desordenada. Para las tareas de ejecución, se dispone de las siguientes unidades segmentadas: 3 FP mul/div (5c), 3 FP add (2c), 3 ALU int (1) y 3 load/store (3). Finalmente, se dispone de un predictor de saltos dinámico que utiliza BTB de 4 entradas y 2 bits de predicción. Cuando se añade una nueva entrada en el BTB, su primera predicción siempre sería de estado A (salto efectivo).



En el computador se ejecuta el siguiente fragmento de programa:

```

; r1 almacena la dirección de a
; r2 almacena la dirección de b
    addi r3, r1, #80 ; condición de final
    addi r1, r1, #8 ; inicialización de los índices
    addi r2, r2, #8 ;
    ld f0, coef ; cargar coeficiente
loop: ld f2, -8 (r1) ; cargar a[i-1]
    ld f4, 0 (r1) ; cargar a[i]
    muld f8, f2, f0 ; a[i-1] * coef
    adddd f4, f8, f4 ; a[i-1] * coef + a[i]
    sd 0 (r2), f4 ; almacenar b[i]
    addi r1, r1, #8 ; incrementar índices
    addi r2, r2, #8
    slt r4, r1, r3
    bnez r4, loop

```

- a) Suponer que inicialmente $r1 = 0$ y $r2 = 100$. Planificar las instrucciones hasta la primera iteración del bucle (sin realizar el salto) utilizando una tabla como la siguiente.

Instr.	IF	ID/ISS	EX	ROB	WB	Comentario
addi r3,r1,#80	1	2	3	4	5	
addi r1,r1,#8	1	2	3	4	5	
addi r2,r2,#8	1	2	3	4	5	
ld f0,coef	2	3	4-6	7	8	
loop: ld f2, -8 (r1)	2	3	4-6	7	8	
ld f4,0 (r1)	2	3	4-6	7	8	
muld f8,f2,f0	3	4-7	8-12	13	14	
adddd f4,f8,f4	3	4-13	14-15	16	17	
sd 0 (r2),f4	3	4-16	17-19	20	21	
addi r1,r1,#8	4	5	6	7	21	
addi r2,r2,#8	4	5	6	7	21	

- b) Realizar una traza de ejecución del código, mostrando el contenido de la BTB, (BTB inicialmente vacía) para todas las iteraciones del bucle.

Dir. salto.	Dir. destino	Bits predicción	Comentario
bnez r4, loop	loop	A	Resto de iteración
bnez r4, loop	loop	B	Última iteración

- c) Determinar el número total de ciclos que tardaría en ejecutarse todo el código.

9. Un estudiante de prácticas de Ingeniería de los Computadores ha paralelizado un programa para aprovechar la paralelización inherente a un multicomputador con todos los nodos iguales y sin pérdidas en su ejecución paralela basado en una red hipercubo 4-dimensional. El programa sin paralelizar tarda 1.5 minutos en ejecutarse en un solo nodo. El estudiante ha concretado que el 75% de dicho programa se puede paralelizar para su ejecución paralela en múltiples unidades de cómputo y el resto debe ser forzosamente ejecutado en un solo nodo. El tiempo de ejecución obtenido es de 18 segundos empleando 4 nodos. Al ver su nota, comprueba que ha obtenido una muy mala calificación en la práctica.

Se pide:

- a) Demuestre y argumente aplicando sus conocimientos de paralelismo la mala calificación obtenida por el estudiante. Nota: Es obligado usar el concepto de “ganancia en velocidad” o “speed-up” en su argumentación.

$$75\% \text{ paralelizado}; \quad n = 4 \text{ nodos}; \quad T_s = 1.5 \text{ minutos} \cdot \frac{60 \text{ s}}{1 \text{ minuto}} = 90 \text{ s}$$

$$\text{Speedup} = \frac{\text{Tiempo antiguo}}{\text{Tiempo nuevo}}$$

$$\text{Tiempo nuevo} = \frac{1}{4} \cdot T_s + \frac{3}{4} \cdot T_p = \frac{1}{4} \cdot T_s + \frac{3}{4} \cdot \frac{T_s}{n} = T_s \cdot \left(\frac{1}{4} + \frac{3}{4} \cdot \frac{1}{n} \right) \rightarrow \text{Tiempo nuevo} = 90 \cdot \left(\frac{1}{4} + \frac{3}{4} \cdot \frac{1}{4} \right)$$

$$\rightarrow \boxed{\text{Tiempo nuevo} = 39.375 \text{ s}}$$

$$\text{Speedup} = \frac{\text{Tiempo antiguo}}{\text{Tiempo nuevo}} = \frac{90}{39.375} \approx 2.29$$

Tras paralelizar el 75% del código con 4 nodos obtenemos una ganancia de velocidad de 2.29

Tras realizar los cálculos, obtenemos un tiempo de **39.375 s** y no **18 s** como decía el alumno, por tanto, los cálculos del estudiante son incorrectos, lo cual justifica la mala calificación que este ha obtenido.

- b) ¿Cuál es la máxima ganancia teórica en velocidad que el estudiante puede obtener? ¿Con cuántos nodos se obtiene?

La máxima ganancia teórica en velocidad se obtiene cuando se consigue paralelizar el 100% del código y se emplean todos los nodos disponibles.

Como se trata de un hipercubo, el número de nodos dependerá de las dimensiones del hipercubo. En este caso, como nuestro hipercubo tiene 4 dimensiones, podemos concluir que el número de nodos será:

$$\text{num nodos} = 2^{\text{dimensión}} \rightarrow \boxed{\text{num nodos} = 2^4 = 16 \text{ nodos}}$$

$$\text{Speedup}_{\max} = \frac{\text{Tiempo antiguo}}{\text{Tiempo nuevo}} = \frac{T_s}{0 \cdot T_s + 1 \cdot T_p} = \frac{T_s}{T_p} = \frac{T_s}{\frac{T_s}{n}} = \frac{1}{\frac{1}{n}} = n \rightarrow \boxed{\text{Speedup}_{\max} = 16}$$

La máxima ganancia teórica en velocidad que se puede obtener con 16 nodos es igual a 16. En este hipotético caso, nuestro programa paralelizado sería 16 veces más rápido que el programa secuencial.

c) Si el ancho de banda de bisección de la red es de 8000 Mbits/seg, ¿qué ancho de banda tiene un enlace?

$$ABB = \text{Ancho Banda Bisección}$$

$$ABE = \text{Ancho Banda Enlace}$$

$$n = \text{número nodos}$$

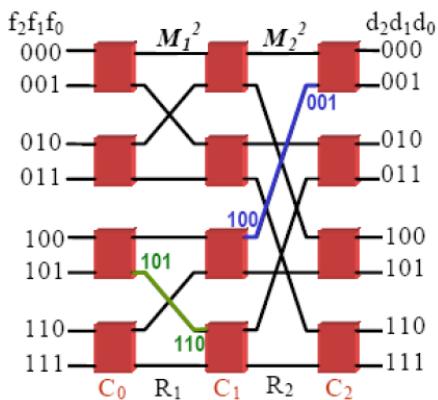
$$ABB = n \cdot ABE \rightarrow ABE = \frac{ABB}{n} \rightarrow ABE = \frac{8000 \text{ Mbits/seg}}{4 \text{ nodos}} = 2 \frac{\text{Gbits}}{\text{s}}$$

Por tanto, podemos concluir que el ancho de banda de cada enlace es de 2 Gbits/seg.

Enero 2016

1. Contesta a las siguientes preguntas:

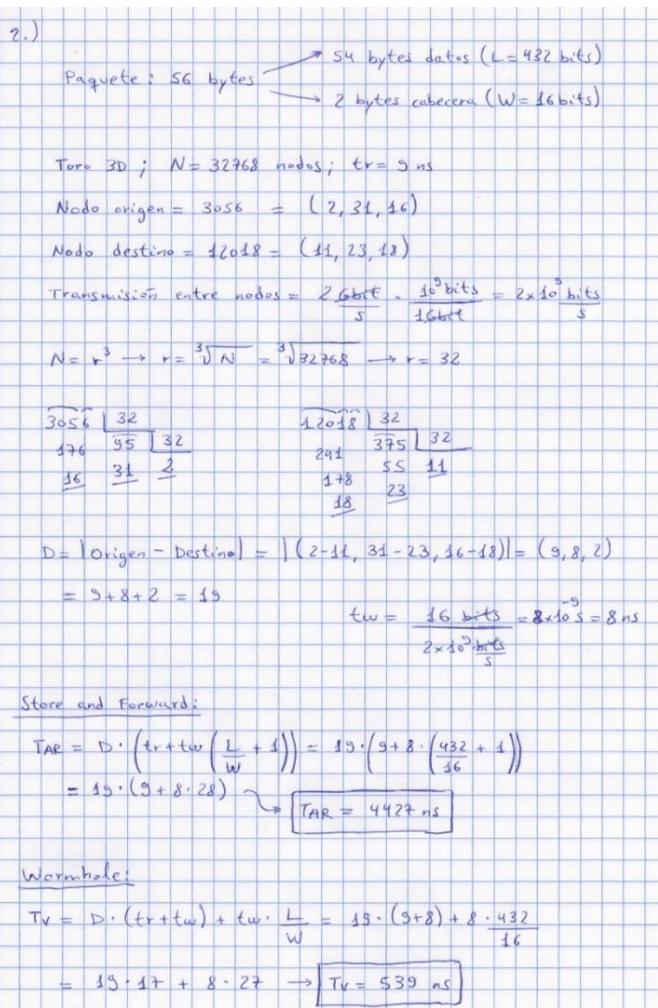
a) Dibuja una red mariposa de 8 entradas y 8 salidas con conmutadores 2x2.



b) En qué consiste la predicción dinámica implícita. Pon un ejemplo.

La predicción dinámica implícita consiste en predecir una instrucción de salto que puede variar en cada ejecución según la historia previa de saltos tomados/no-tomados. En el caso de la implícita no hay bits de historia, sino que se almacena la dirección de la instrucción que se ejecutó después de la instrucción de salto en cuestión.

2. Un banco ha adquirido un supercomputador formado por 32768 nodos conectados mediante una red toro 3D cuyos enlaces tienen una velocidad de 2 Gbit/s. Para terminar de analizar el rendimiento del supercomputador se desea saber cuánto tardará un paquete formado por 56 bytes (incluyendo la cabecera) que se envía desde el nodo 3056 al nodo 12018. El tiempo de enrutamiento es de 9 ns. Calcula los tiempos de envío tanto utilizando "store and forward" como "Wormhole". Nota: la cabecera del paquete está formada por 2 bytes.



3. Conteste a las siguientes preguntas:

- a) Un sistema multiprocesador compuesto por N nodos monoprocesador con caché utiliza el protocolo MESI para mantener la coherencia de sus cachés. Una determinada línea de caché de uno de sus procesadores está inicialmente en estado M. Indique para las siguientes órdenes en el bus que observa la controladora de caché para dicha línea de caché: cambio de estado (indicando qué significa cada estado y en qué estado está la memoria principal con respecto a la/s línea/s de caché) y el flujo de información (qué memorias o cachés se actualizan y quién recibe o da la información).

- 1) BusRdX
- 2) BusRd

- b) ¿Puede una línea de caché del anterior computador pasar del estado S al estado E mediante una orden de procesador (PrWr o PrRd)? Explique razonadamente y pormenorizadamente su respuesta.

4. Un estudiante de IC ha paralelizado mediante MPI una cierta aplicación. Después del proceso de paralelización ha visto que el 10% de la aplicación no se puede paralelizar. El restante 90% está definido por un gran ciclo for que sí es paralelizable para cualquier número de nodos. Se pide (explique razonadamente cada apartado):

- a) ¿A partir de qué número de procesadores obtenemos ganancias estrictamente mayores que 4? Suponga que el tiempo de sobrecarga es nulo.

$$\text{Speedup} = \frac{T_s}{\frac{1}{10} \cdot T_s + \frac{9}{10} \cdot \frac{T_s}{n}} = \frac{T_s}{T_s \cdot \left(\frac{1}{10} + \frac{9}{10} \cdot \frac{1}{n} \right)} = \frac{1}{\frac{n+9}{10 \cdot n}} = \frac{10 \cdot n}{n+9}$$

$$\text{Speedup} > 4$$

$$\frac{10 \cdot n}{n+9} > 4 \rightarrow 10 \cdot n > 4 \cdot (n+9) \rightarrow 10 \cdot n > 4 \cdot n + 36 \rightarrow 6n > 36 \rightarrow n > 6$$

Tal y como podemos ver en el siguiente ejemplo, a partir de **7 procesadores** obtenemos ganancias mayores que 4.

Ejemplo:

$$n = 7$$

$$\text{Speedup} = \frac{10 \cdot n}{n+9} = \frac{10 \cdot 7}{7+9} = \frac{70}{16} \rightarrow \text{Speedup} = 4.375$$

- b) Demuestre que si suponemos un tiempo de sobrecarga linealmente dependiente del número de nodos con los que paraleliza (p), el tiempo de cómputo paralelo no siempre mejora con p , sino que necesariamente debe llegar a degradarse a partir de un cierto número de nodos (p).

5. Conteste a las siguientes preguntas:

- a) Explica en qué consiste el encadenamiento de operaciones en las máquinas vectoriales.

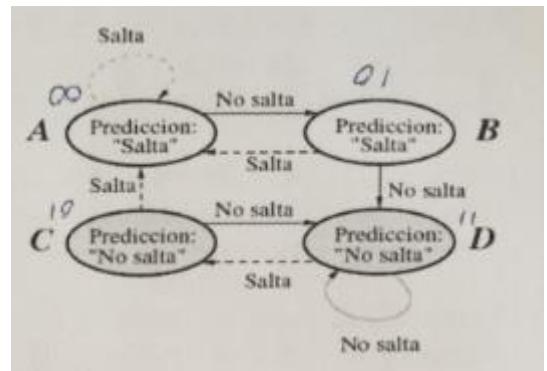
El encadenamiento permite que una operación vectorial comience tan pronto como los elementos individuales de su operando vectorial fuente estén disponibles, es decir, los resultados de la primera unidad funcional de la cadena se adelantan a la segunda unidad funcional. Naturalmente deben ser unidades funcionales diferentes, de lo contrario surge un conflicto temporal.

Si las unidades están completamente segmentadas, basta retrasar el comienzo de la siguiente instrucción durante el tipo de arranque de la primera unidad.

- b) ¿Para qué se utilizan las estructuras buffer de renombrado y buffer de reorden de los procesadores superescalares?

Tanto el buffer de renombrado como el buffer de reorden de los procesadores superescalares se utiliza para evitar el efecto de las dependencias WAR, o Antidependencias (en la emisión ordenada) y WAW, o Dependencias de Salida (en la ejecución desordenada).

6. Suponer un computador superescalar que dispone un buffer de reorden, que permite resolver los riesgos WAR y WAW, y una cola y ventana de instrucciones con un número de entradas suficiente. El procesador es capaz de captar 3 instrucciones por ciclo, decodificar, emitir y completar 2 instrucciones por ciclo. La ejecución implementa adelantamiento. Además, la emisión y finalización de las instrucciones puede ser desordenada. Para las tareas de ejecución, se dispone de las siguientes unidades segmentadas: 2 FP mul/div (5c), 2 FP add (2c), 2 ALU int (1) y 2 load/store (3). Finalmente, se dispone de un predictor de saltos dinámico que utiliza BTB de 3 entradas y 2 bits de predicción. Cuando se añade una nueva entrada en el BTB, su primera predicción sería de estado A (salto efectivo) si el salto es hacia atrás y de estado D (salto no efectivo) si el salto es hacia adelante.



En el computador se ejecuta el siguiente fragmento de programa:

```

0x01      addi r3, r0, #4 ; r3 = 4
0x02      add r4,r0,r3 ; r4 = r3
0x03      loop: subi r3,r3,#1 ; r3 = r3 - 1
0x04          beqz r3, end; si r3=0 saltar end
0x05          ld f1,-8 (r1) ; cargar a[i-1]
0x06          ld f2,0 (r1) ; cargar a[i]
0x07          ld f3,0 (r2) ; cargar b[i]
0x08          muld f4,f1,f2 ; a[i-1] * a[i]
0x09          muld f4,f4,f3 ; a[i-1] * a[i] * b[i]
0x0A          sd 0(r1), f4 ; almacenar a[i]
0x0B          addi r1,r1,#8 ; r1 = r1 + 8
0x0C          addi r2, r2, #8 ; r2 = r2 + 8
0x0D          bnez r4, loop
0x0E      end: addd f4, f4, f4

```

- a) Planificar las instrucciones utilizando una tabla como la siguiente hasta la primera iteración del bucle (sin realizar el salto). Suponer que inicialmente r1=0 y r2=100.

Instr.	IF	ID/ISS	EX	ROB	WB	Comentario
addi r3,r0,#4	1	2	3	4	5	
add r4,r0,r3	1	2-3	4-5	6	7	
loop: subi r3,r3,#1	1-2	3	4	5	6	
beqz r3,end;	2	3-4	5	6	7	
ld f1,-8 (r1)	2-3	4	5-7	8	9	
ld f2,0 (r1)	3-4	5	6-8	9	10	
ld f3,0 (r2)	3-4	5-7	8	9	10	
muld f4,f1,f2	4-5	6-8	9-13	14	15	
muld f4,f4,f3	5-7	8-13	14-18	19	20	
sd 0(r1), f4	5-8	9-18	19-20	21	22	
addi r1,r1,#8	6-13	14	15	16	17	
addi r2, r2, #8	8-14	15	16	17	18	
bnez r4,loop	9-15	16	17	18	19	

- b) Realizar una traza de ejecución del código, mostrando el contenido de la BTB, (BTB inicialmente vacía) para todas las iteraciones del bucle.

1. Contesta a las siguientes preguntas:

- a) Explica qué dos tipos de buffer de renombrado podemos encontrar y cuáles son las ventajas de cada tipo.

Existen dos tipos de buffer de renombrado:

- **Buffer de renombrado con acceso asociativo.**

- Permite varias escrituras pendientes a un mismo registro.
- Se utiliza el bit último para marcar cual ha sido la más reciente.

Cada línea de buffer tiene cinco campos: asignación válida, registro de destino, contenido, contenido válido y bit de asignación última.

- **Asignación válida.** Indica si la línea en cuestión se ha utilizado para nombrar algún registro, es decir, si los restantes campos tienen información válida.
- **Registro de destino.** Se indica el número de registro de la arquitectura que se ha renombrado utilizando la línea en cuestión.
- **Contenido.** Almacenará los datos correspondientes al registro que se ha renombrado hasta que esos datos se actualicen en el registro de la arquitectura correspondiente.
- **Contenido válido.** Se utiliza como bit de validez del contenido, indicando, si está a cero, que alguna instrucción que se ha emitido o enviado a escribir su resultado en el campo del contenido de línea.
- **Bit de asignación última.** Está a 1 en la línea del buffer de renombrado en la que se haya hecho la última asignación a un registro dado.

- **Buffer de renombrado con acceso indexado.**

- Sólo permite una escritura pendiente a un mismo registro.
- Se mantiene la escritura más reciente.

En cada registro de la arquitectura existe un índice que apunta a la línea del buffer que se utiliza para renombrar ese registro. Junto con ese índice también existe un campo de asignación válida que indica si se ha hecho o no el renombrado (y si el contenido del campo de índice es válido). El buffer de renombrado propiamente dicho únicamente tiene el campo de contenido y el de contenido válido.

- b) Explica cómo se realiza el encaminamiento en una red CCC (no hace falta dibujarla).

Primero procedemos a enumerar los nodos de forma recursiva partiendo de un 1-cubo. Un 1-cubo tendrá dos nodos a los que numeraremos con 0 y 1. Cada vez que se añada una nueva dimensión, se duplicará el número de nodos existentes: numeraremos los nuevos nodos con el mismo número que los anteriores añadiendo por la izquierda un nuevo bit con valor 1. Se procederá sucesivamente de esta forma para numerar los nodos de una red n-CCC de cualquier dimensión.

Una vez que hemos numerado los nodos, procedemos con el algoritmo de encaminamiento: se comparan los números de los nodos origen y destino de la comunicación. A partir de ahí, se envía el mensaje por los enlaces de las direcciones que corresponden a los bits diferentes en ambos números.

Por último, hemos de tener en cuenta que, para cambiar de dimensión, en cada vértice hay que recorrer un enlace por el anillo y, en el anillo final, hay que llegar hasta el nodo deseado por el lado más corto.

2. Una universidad ha adquirido un supercomputador formado por 32768 nodos conectados mediante una red malla abierta 3D cuyos enlaces tienen una velocidad de 4 Gbit/s. Para terminar de analizar el rendimiento del supercomputador se desea saber cuánto tardará un paquete formado por 24 bytes (incluyendo la cabecera) que se envía desde el nodo 1015 al nodo 22222. El tiempo de enrutamiento es de 27 ns. Calcula los tiempos de envío tanto utilizando "Store and Forwarding" como "Wormhole".

Nota: la cabecera del paquete está formada por 4 bytes.

2.)

Paquete: 24 bytes → 20 bytes datos ($L = 160 \text{ bits}$)
 Paquete: 24 bytes → 4 bytes cabecera ($W = 32 \text{ bits}$)

Malla abierta 3D; $N = 32768$ nodos; $tr = 27 \text{ ns}$

Nodo origen = 1015 = (0, 31, 23)

Nodo destino = 22222 = (21, 22, 14)

$$\text{Transmisión entre nodos} = 4 \frac{\text{Gbit}}{\text{s}} \cdot \frac{10^9 \text{ bits}}{1 \text{ Gbit}} = 4 \times 10^9 \frac{\text{bits}}{\text{s}}$$

$$N = r^3 \rightarrow r = \sqrt[3]{N} = \sqrt[3]{32768} \rightarrow r = 32$$

$$\begin{array}{r|rr} 10 & 1 & 32 \\ 5 & 3 & 1 \\ \hline 2 & 3 & 1 \\ \hline 0 & & 0 \end{array}$$

$$\begin{array}{r|rr} 22 & 2 & 32 \\ 3 & 0 & 2 \\ \hline 1 & 4 & 2 \\ \hline 1 & 2 & 2 \\ \hline 0 & & 0 \end{array}$$

$$D = |\text{Origen} - \text{Destino}| = |(0-21, 31-22, 23-14)| = (21, 9, 9)$$

$$= 21 + 9 + 9 = 39$$

$$tw = \frac{32}{4 \times 10^9} = 8 \times 10^{-9} \text{ s} = 8 \text{ ns}$$

Store and Forward:

$$TAR = D \cdot \left(tr + tw \cdot \left(\frac{L+1}{W} \right) \right) = 39 \cdot \left(27 + 8 \cdot \left(\frac{160+1}{32} \right) \right) = \boxed{2925 \text{ ns}}$$

Wormhole:

$$TV = D \cdot (tr + tw) + tw \cdot \frac{L}{W} = 39 \cdot (27+8) + 8 \cdot \frac{160}{32} = \boxed{1405 \text{ ns}}$$

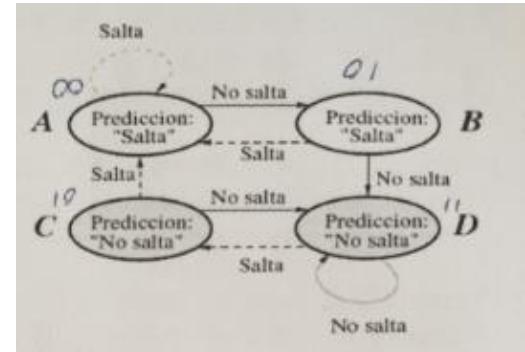
3. Suponer un computador superescalar que dispone un buffer de reorden, que permite resolver los riesgos WAR y WAW, y una ventana de instrucciones con un número de entradas suficiente. El procesador es capaz de decodificar, emitir y completar 3 instrucciones por ciclo. Además, la emisión de las instrucciones puede ser desordenada y dispone de unidad de adelantamiento. Para las tareas de ejecución, se dispone de las siguientes unidades segmentadas: 2 FP mul/div (5c), 2 FP add (2c), 2 ALU int (1) y 2 load/store (3). Finalmente, se dispone de un predictor de saltos dinámico que utiliza BTB de 4 entradas y 2 bits de predicción. Cuando se añade una nueva entrada en el BTB, su primera predicción siempre sería de estado A (salto efectivo).

En el computador se ejecuta el siguiente fragmento de programa:

```

; r1 almacena la dirección de a
; r2 almacena la dirección de b
    addi r3, r1, #80 ; condición de final
    addi r1, r1, #8 ; inicialización de los índices
    addi r2, r2, #8 ;
    ld f0, coef ; cargar coeficiente
loop: ld f2, -8 (r1) ; cargar a[i-1]
    ld f4, 0 (r1) ; cargar a[i]
    beqz r5, fin
    muld f8, f2, f0 ; a[i-1] * coef
    divd f9, f2, f0 ;
    addd f4, f8, f4 ; a[i-1] * coef + a[i]
    sd 0 (r2), f4 ; almacenar b[i]
    addi r1, r1, #8 ; incrementar índices
    addi r2, r2, #8
    subi r5, r5, #1
    slt r4, r1, r3
    bnez r4, loop
fin: subd f2, f1, f3

```



- a) Planificar las instrucciones utilizando una tabla como la siguiente hasta la primera iteración del bucle (sin realizar el salto). Suponer que inicialmente r1=0 y r2=100.

Instrucción	IF	ID/ISS	EX	ROB	WB	Comentario
addi r3,r1,#80	1	2	3	4	5	alu1
addi r1,r1,#8	1	2	3	4	5	alu2
addi r2,r2,#8	1	2-3	4	5	6	alu1 (todas alus ocupadas, esperar)
addi r5,r1,#3	2	3	4	5	6	alu2
ld f0, coef	2	3	4-6	7	8	load/store 1
loop: ld,-8(r1)	2	3	4-6	7	8	load/store 2
ld f4,0(r1)	3	4-6	7-9	10	11	load/store 1 (todas load/store ocupadas, esperar)
beqz f5,fin	3	4	5	6-10	11	alu1
muld f8,f2,f0	3	4-6	7-11	12	13	mult/div 1 (esperar a que f0 se libere, usar adelantamiento)
divd f9,f2,f0	4	5-6	7-11	12	13	mult/div 2 (adelantamiento en uso en ciclo 8, esperar 1 ciclo más)
addd f4,f8,f4	4	5-11	12-13	14	15	add 1 (esperar a que f8 se libere, usar adelantamiento)
sd 0(r2),f4	4	5-13	14-16	17	18	load/store 1 (esperar a que se libere f4 de instrucción anterior)
addi r1,r1,#8	5	6	7	8-17	18	alu1
addi r2,r2,#8	5	6	7	8-17	18	alu2
subi r5,r5,#1	5	6-7	8	9-18	19	alu1 (todas alus ocupadas, esperar)
slt r4,r1,r3	6	7	8	9-18	19	alu2

bnez r4,loop	6	7-8	9	10-18	19	alu1 (todas alus ocupadas, esperar)
fin: subd f2,f1,f3	x					

- b) Realizar una traza de ejecución del código, mostrando el contenido de la BTB (BTB inicialmente vacía) para todas las iteraciones del bucle.

Iteración 1:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	A
bnez r4, loop	loop	A

Iteración 2:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	B
bnez r4, loop	loop	A

Iteración 3:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	D
bnez r4, loop	loop	A

Iteración 4:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	D
bnez r4, loop	loop	A

Iteración 5:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	D
bnez r4, loop	loop	A

Iteración 6:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	D
bnez r4, loop	loop	A

Iteración 7:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	D
bnez r4, loop	loop	A

Iteración 8:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	D
bnez r4, loop	loop	A

Iteración 9:

Dir. salto.	Dir. destino	Bits predicción
beqz r5, fin	fin	D
bnez r4, loop	loop	B

- c) ¿Existe alguna penalización en la ejecución del código? Si es así, indica con qué instrucción y cuando.

Sí, con la instrucción **beqz r5,fin** ya que en la primera iteración la predicción por defecto es que salta pero esta instrucción no salta y en la segunda iteración es aun estando en el estado B la predicción para este estado es saltar pero la instrucción no salta.

- d) Determinar el número de ciclos que tardaría en ejecutarse el código.

Iteración 1: 19 ciclos

Iteraciones 2 – 8: 18 ciclos

Iteración 9: 11 ciclos

Total: x ciclos

4. Explica la diferencia entre predicción dinámica explícita y predicción dinámica implícita.

Predicción dinámica implícita. No se guarda ninguna información explícita que represente el comportamiento pasado de la instrucción. Se almacena únicamente la dirección de la instrucción que se ejecutó tras la instrucción de salto la última vez que se captó esta. La dirección puede ser: la dirección de destino del salto (lo que equivale a predecir que se produce el salto), o la dirección de la instrucción siguiente a la del salto (si se predice que no se produce el salto).

Predicción dinámica explícita. Para cada instrucción de salto condicional, existe un conjunto de bits que codifican la información relativa al comportamiento pasado de la instrucción en cuestión. Esos bits se denominan bits de historia. El número de bits de historia que se guardan para cada instrucción depende del tipo de esquema de predicción dinámica explícita que se haga.

5. En la paralelización de una aplicación orientada a una máquina paralela de memoria distribuida un ingeniero ha descompuesto dicha aplicación en 12 tareas, que nombramos como T1, T2...T12, donde el subíndice indica el orden de ejecución de cada tarea en la versión secuencial. Seguidamente nos indica que los grupos de tareas T2 a T6 (ambas incluidas) y T8 a T10 (ambas incluidas) son independientes entre sí.

Por último, nos informa del porcentaje de tiempo que toma cada tarea por separado en la versión secuencial (ver tabla adjunta). Suponga que disponemos de 2 multicamputadores con 2 y 4 nodos respectivamente (todos los nodos iguales), conectados entre sí con una red cuya sobrecarga puede modelar con $T_{overhead}(p) = 0.05 * p$ (donde p es el número de procesadores).

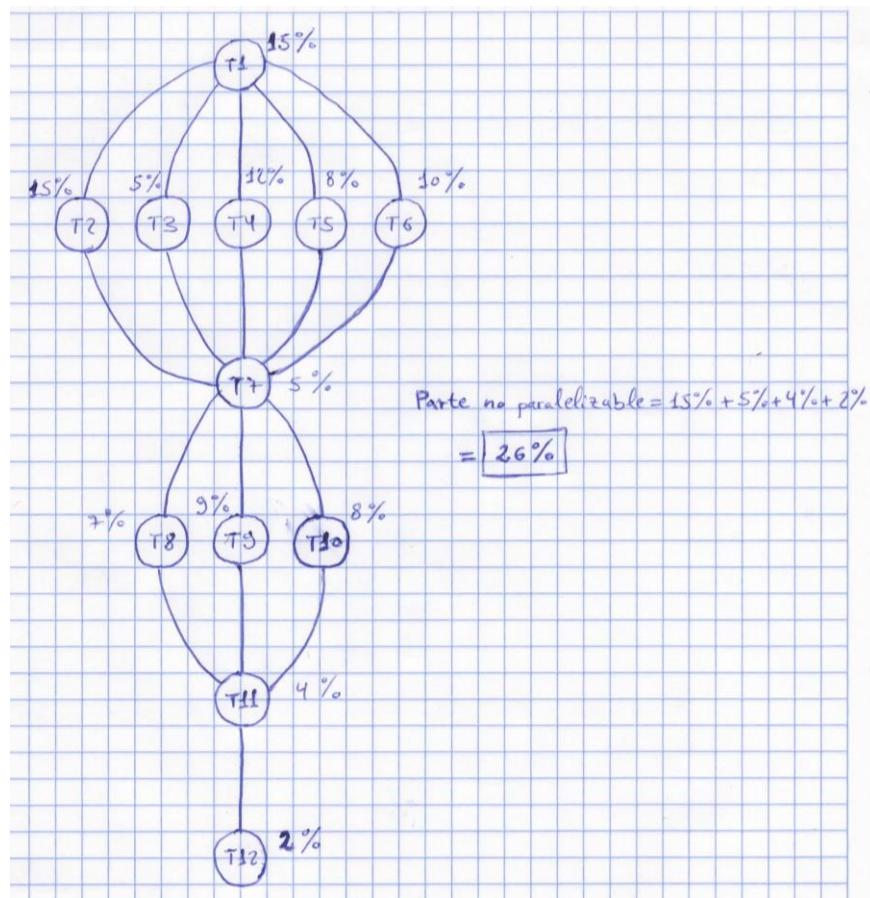
Tarea	Porcentaje tiempo
T1	15%
T2	15%
T3	5%
T4	12%
T5	8%
T6	10%
T7	5%
T8	7%
T9	9%
T10	8%
T11	4%
T12	2%

VERDE: Grupo 1

AZUL: Grupo 2

Se pide:

- a) Dibuje el grafo de precedencia entre tareas, calcule la fracción no paralelizable del problema.



b) ¿Cuál es la máxima ganancia en velocidad que se puede obtener con cada uno de los multicomputadores?

2 nodos:

$$\begin{aligned}Parte1 &= \text{Máx } \{T2, T4\} + \text{Máx}\{T5, T6\} + T3 = \text{Máx}\{15\%, 12\%\} + \text{Máx}\{8\%, 10\%\} + 5\% \\&= 15\% + 10\% + 5\% = 30\%\end{aligned}$$

$$Parte2 = \text{Máx } \{T9, T10\} + T8 = \text{Máx}\{9\%, 8\%\} + 7\% = 9\% + 7\% = 16\%$$

$$\begin{aligned}T_p(p = 2) &= \text{Porcentaje no paralelizable} + Parte1 + Parte2 + T_{overhead}(p) \\&= 26\% + 30\% + 16\% + 0.05 \cdot 2 = 72.1\%\end{aligned}$$

$$Speedup(p = 2) = \frac{100\%}{72.1\%} = \boxed{1.38}$$

4 nodos:

$$Parte1 = \text{Máx } \{T2, T4, T5, T6\} + T3 = \text{Máx}\{15, 12, 8, 10\} + 5\% = 15\% + 5\% = 20\%$$

$$Parte2 = \text{Máx } \{T8, T9, T10\} = \text{Máx}\{7\%, 9\%, 8\%\} = 9\%$$

$$\begin{aligned}T_p(p = 4) &= \text{Porcentaje no paralelizable} + Parte1 + Parte2 + T_{overhead}(p) \\&= 26\% + 20\% + 9\% + 0.05 \cdot 4 = 55.2\%\end{aligned}$$

$$Speedup(p = 4) = \frac{100\%}{55.2\%} = \boxed{1.812}$$

c) ¿En qué cluster es más eficiente la ejecución de nuestra aplicación con la paralelización propuesta?

2 nodos:

Por tanto, la ejecución de nuestra aplicación paralelizada es más eficiente en el multicomputador de 2 nodos.

$$E_p(p = 2) = \frac{1.38}{2} = 0.69$$

4 nodos:

$$E_p(p = 4) = \frac{1.812}{4} = 0.453$$

6. Una cierta línea de caché de uno de los nodos de un multiprocesador equipado con un sistema de caché que implementa el **protocolo MESI** pasa al estado I.

Indique todo lo que podemos saber (mecanismo ha llevado a la línea a cambiar de estado, porqué, posible estado o estados previos, ...) y qué consecuencias tiene cambiar al estado I (¿en qué casos se ha de compartir y con quién el contenido de dicha línea)?

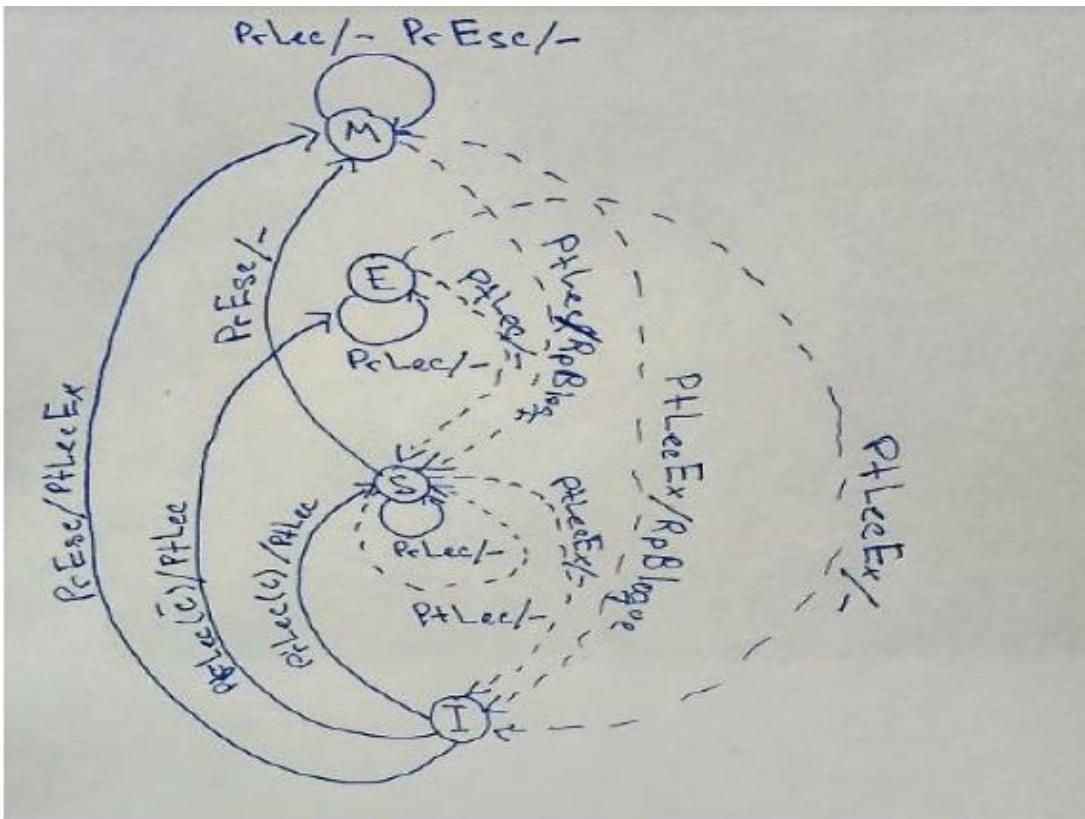
Lo que podemos saber es lo siguiente: se ha producido un **fallo de escritura al no estar el bloque en caché**, es decir, el procesador escribe (PrEsc) y el bloque no está en caché. El controlador de caché del procesador que escribe donde un paquete de petición de acceso exclusivo al bloque (PtLecEx). El estado del bloque en la caché después de la escritura será de modificado (**M**). El paquete PtLecEx provoca los siguientes efectos:

- Si una caché tiene el bloque en estado **Modificado (M)**, bloquea la lectura de memoria y deposita el bloque en el bus. El bloque pasa en esta caché a estado **Inválido (I)**.
- Si una caché tiene el bloque en estado **Exclusivo (E)** o **Compartido (S)**, pasa a estado **Inválido (I)**.

Possíveis estados:

- Modificado (M).** Significa que es el único componente con una copia válida del bloque. El resto tienen una copia no actualizada.
- Exclusivo (E).** Significa que es el único componente con una copia válida del bloque. El resto tienen una copia no válida y la memoria tiene una copia actualizada del bloque.
- Compartido (S).** Supone que el bloque es válido en esta caché en memoria y al menos en otra caché.
- Inválido (I).** Supone que el bloque no está físicamente en la caché, o si se encuentra ha sido invalidado por el contador como consecuencia de la escritura en la copia del bloque situada en la otra caché.

Estado	Evento	Acción	Siguiente Estado
Modificado (M)	Procesador lee (PrLec)	-	M
	Procesador escribe (PrEsc)	-	M
	Paquete de lectura (PtLec)	Genera paquete respuesta bloque (RpBloque)	S
	Paquete de acceso exclusivo al bloque (PtLecEx)	Genera paquete respuesta bloque (RpBloque) Invalida copia local	I
	Reemplazo	Genera paquete postescritura bloque (PtPEsc)	I
Exclusivo (E)	Procesador lee (PrLec)	-	E
	Procesador escribe (PrEsc)	-	M
	Paquete de lectura (PtLec)	-	S
	Paquete de acceso exclusivo al bloque (PtLecEx)	Invalida la copia local	I
Compartido (S)	Procesador lee (PrLec)		S
	Procesador escribe (PrEsc)	Genera paquete (PtLeeEx)	M
	Paquete de lectura (PtLec)		S
	Paquete exclusivo de acceso al bloque (PtLeeEx)	Invalida copia local	I
Inválido (I)	Procesador lee (PrLec)	(Copia en otras cachés, C = 1) Genera paquete PtLec	S
	Procesador lee (PrLec)	(No hay copias en otras cachés, C = 0) Genera paquete PtLec	E
	Procesador escribe (PrEsc)	Genera paquete PtLeeEx	M
	Paquete de lectura (PtLee)		I
	Paquete de acceso exclusivo al bloque (PtLeeEx)		I



Julio 2017

1. Contesta a las siguientes preguntas:

- a) Explica para qué sirve un buffer de reorden y en qué estados pueden estar las instrucciones que se almacenan en este buffer.

El buffer de reorden es una técnica que se utiliza para **evitar el efecto de las dependencias WAR, o Antidependencias** (en la emisión desordenada) y **WAW, o Dependencias de Salida** (en la ejecución desordenada).

Las instrucciones que se almacenan en el buffer de reorden pueden encontrarse en uno de los siguientes estados:

- Emitida.
- Ejecución.
- Finalizada la ejecución.

- b) Explica cómo se realiza el encaminamiento en una red tipo mariposa (no hace falta dibujarla).

Para emitir un mensaje entre dos procesadores empleando una red tipo mariposa usaremos un paquete con la siguiente forma:

Encabezamiento	Carga útil	Remolque
----------------	------------	----------

Encabezamiento. Contiene la dirección de destino del mensaje.

Carga útil. Contenido del mensaje.

Remolque. Suma de verificación.

Al llegar a un nodo de conmutación, se selecciona uno de los dos enlaces de salida en base al bit más significativo de la dirección de destino. Si ese bit es cero, se selecciona el enlace izquierdo. Si ese bit es uno, el enlace correcto está seleccionado. Posteriormente, este bit se elimina de la dirección de destino en el paquete transmitido a través del enlace seleccionado. Este paso se realizará tantas veces como bits tenga el encabezamiento.

Una vez que el encabezamiento esté vacío habremos llegado a nuestro destino. Finalmente, este destino recibirá sólo el contenido del mensaje y la suma de verificación.

2. Una empresa ha adquirido un supercomputador formado por 4096 nodos conectados mediante una red toro 3D cuyos enlaces tienen una velocidad de 2 Gbit/s. Para terminar de analizar el rendimiento del supercomputador se desea saber cuánto tardará un paquete formado por 30 bytes (incluyendo la cabecera) que se envía desde el nodo 105 al nodo 4000. El tiempo de enrutamiento es de 20 ns. Calcula los tiempos de envío utilizando "Wormhole" y comparando entre dos configuraciones diferentes, una en la que hay buffers tanto en la entrada como en la salida de los conmutadores y otra en la que solo hay buffers a la entrada de los conmutadores.

Nota: la cabecera del paquete está formada por 2 bytes.

2.)

Paquete: 30 bytes → 28 bytes datos ($L = 224$ bits) → 2 bytes cabecera ($W = 16$ bits)

Toro 3D; $N = 4096$ nodos; $t_r = 20$ ns

Nodo origen = 105 = (0, 6, 9)

Nodo destino = 4000 = (15, 10, 0)

Transmisión entre nodos = $2 \frac{\text{Gbit}}{\text{s}} \cdot \frac{224 \text{ bits}}{1 \text{ Gbit}} = 2 \times 10^3 \frac{\text{bits}}{\text{s}}$

$N = r^3 \rightarrow r = \sqrt[3]{N} = \sqrt[3]{4096} \rightarrow r = 16$

Diagrama de los nodos:

$\overbrace{105}^{9} \mid \overbrace{16}^{6} \mid \overbrace{16}^{0}$	$\overbrace{4000}^{15} \mid \overbrace{16}^{10} \mid \overbrace{16}^{0}$ 80 250 16 00 90 15 0 10
---	--

$D = |\text{nodo origen} - \text{nodo destino}| = |(0-15, 6-10, 9-0)| = (15, 4, 9) \rightarrow (1, 4, 7)$

$= 1+4+7 = 12$

$t_w = \frac{W}{\text{Velocidad}} = \frac{16}{2 \times 10^3} = 8 \text{ ns}$

Diagrama de la transmisión:

Wormhole (buffer sólo en entradas de conmutadores)

$t_v = D \cdot (t_r + t_w) + t_w \cdot \frac{L}{W} = 12 \cdot (20 + 8) + 8 \cdot \frac{224}{16} \rightarrow \boxed{t_v = 448 \text{ ns}}$

Wormhole (buffer en entradas y salidas)

$t_v = D \cdot (t_r + t_s + t_w) + \max(t_s, t_w) \cdot \frac{L}{W} = 12 \cdot (20 + t_s + 8) + \max(t_s, 8) \cdot \frac{224}{16}$

$\hookrightarrow t_v = 12 \cdot (28 + t_s) + 14 \cdot \max(t_s, 8)$

4.7. Métodos de encaminamiento

Los métodos de encaminamiento son mecanismos, hardware o software, que permiten establecer la ruta entre los nodos origen y destino de una comunicación. El algoritmo de encaminamiento también debe efectuar la elección de la ruta cuando haya varias posibles, también debe gestionar los conflictos entre las informaciones que quieran tomar el mismo camino. La facilidad en el encaminamiento puede ser una de las razones para elegir uno u otro tipo de red. Estudiaremos los métodos de encaminamiento en algunas de las redes estudiadas con anterioridad.

4.7.1. Encaminamiento en redes hipercubo

Para entender el encaminamiento en las redes de tipo k -cubo es necesario saber como se numeran los nodos en este tipo de red. Lo haremos de forma recursiva partiendo de un 1-cubo. Un 1-cubo tendrá dos nodos a los que numeraremos con 0 y 1. Cada vez que se añada una nueva dimensión, se duplicará el número de nodos existentes: numeraremos los nuevos nodos con el mismo número que los anteriores añadiendo por la izquierda un nuevo bit con valor 1. Se procederá sucesivamente de esta forma para numerar los nodos de un hipercubo de cualquier dimensión.

Tabla 4.2. Resumen comparativo de diferentes redes de interconexión

	Hipercubo	Bus	Líneas cruzadas	Multietapa
Costo	Medio	Bajo	Alto	Medio
Velocidad	Media	Baja	Alta	Alta
Complejidad	Media	Baja	Alta	Media
Escalabilidad	Media	Alta	Media	Media

Sabiendo esto, el algoritmo de encaminamiento es sencillo: Se comparan los números de los nodos origen y destino de la comunicación. A partir de ahí, se envía el mensaje por los enlaces de las direcciones que corresponden a los bits diferentes en ambos números.

4.7.2. Encaminamiento en redes n -CCC

El encaminamiento en estas redes es muy parecido al de las redes hipercubo. La diferencia estriba en que en cada vértice hay que recorrer un enlace por el anillo, para cambiar de dimensión, y en el anillo final hay que llegar hasta el nodo deseado por el lado más corto.

4.7.3. Encaminamiento en redes omega

El principio del encaminamiento en las redes omega radica en que, como se estudió con anterioridad, la permutación *perfect shuffle* conecta cada nodo con el que resulta de rotar su número un lugar a la izquierda y, por otra parte, si nos fijamos en el primer nivel, cada conmutador intercambia los nodos que difieren en el bit de mayor orden: esto equivale a complementar ese bit de cara al encaminamiento. Reuniendo ambos hechos se llega fácilmente al algoritmo de encaminamiento. Para efectuarlo, se parte del nodo origen, esto nos llevará a un módulo conmutador. Este módulo se invertirá si los números de los nodos origen y destino, difieren en el bit de mayor orden. Esto nos conducirá a otro conmutador que se invertirá si los números de ambos nodos difieren en el bit siguiente, se continuará así hasta llegar a la última etapa en que alcanzaremos el nodo destino.

4.7.4. Encaminamiento en redes delta

En una red delta el algoritmo de encaminamiento tiene una característica curiosa: sólo depende del número del nodo de destino: En una red delta de $a^n \times b^n$, si se pone el número del nodo de destino en base b , partiremos del nodo origen y haremos que el primer conmutador conecte la entrada que proviene del nodo origen con la salida correspondiente al valor del dígito más significativo del número del nodo destino (escrito en base b), procederemos así con sucesivos dígitos en las siguientes etapas por las que el paquete de información vaya pasando (véase la figura 4.20).

4.7.5. Encaminamiento en redes de linea base

Para efectuar el encaminamiento en redes de línea base se debe proceder según los pasos siguientes:

1. Se invierte el orden de los bits del número del nodo origen (en binario).
2. Se hace la operación OR EXCLUSIVO del resultado del paso anterior con el número del nodo destino, también en binario.

3. El resultado de la operación anterior nos indicará si cada conmutador se dejará en conexión directa (cuando el bit sea 0) o en conexión cruzada (cuando el bit sea 1), respectivamente.

1. Un programa tarda 40 segundos en ejecutarse en un SMP (multiprocesador), durante el 20% del tiempo se ha ejecutado en 4 procesadores, el 60% del tiempo en 3 y el 20% restante en 1. Consideramos que la carga se ha distribuido por igual. ¿Cuándo tardaría el programa en ejecutarse en un solo procesador? Hallar la ganancia y la eficiencia.

20% 4 procs.

$$40 = 40*0.2 + 40*0.6 + 40*0.2$$

60% 3 procs.

$$T_{\text{ptotal}} = T_{4\text{procs}} + T_{3\text{procs}} + T_{1\text{procs}}$$

20% 1 proc.

Tsec?

Eficiencia?

SpeedUp?

TIEMPO PARALELIZADO

$$T_{4\text{procs}} = 40*0.2 = 8 \text{ s}$$

$$T_{3\text{procs}} = 40*0.6 = 24 \text{ s}$$

$$T_{1\text{procs}} = 40*0.2 = 8 \text{ s}$$

TIEMPO SIN PARALELIZAR

$$T_{\text{sec}}(4 \text{ procs}) = 8*4 = 32 \text{ s}$$

$$T_{\text{sec}}(3 \text{ procs}) = 24*3 = 72 \text{ s}$$

$$T_{\text{sec}}(1 \text{ proc}) = 8*1 = 8 \text{ s}$$

$$T_{\text{sec}} = 32+72+8 = 112 \text{ s}$$

SPEED UP (GANACIA)

$$SP = \frac{T_s}{T_p}$$

$$SP = \frac{112}{40} = 2.8 \leq 4 :)$$

EFICIENCIA

$$E = \frac{SP}{N}$$

$$E = \frac{2.8}{4} = 0.7 \leq 1 :)$$

2. Un programa tarda 20 segundos en ejecutarse en un procesador p1 y 30 en otro p2. Quiero utilizar P1 y P2 para ejecutar el mismo programa en paralelo. ¿Qué tiempo tarda en ejecutarse el programa si la carga de trabajo se distribuye por igual entre los dos procesadores? [No se tiene en cuenta la sobrecarga]. Calcular el Speed-Up y la eficiencia.

Si repartimos de forma equivalente el trabajo cada sección paralela tardara:

- $T_{pp1} = 1/2 * 20 = 10 \text{ s}$
- $T_{pp2} = 1/2 * 30 = 15 \text{ s}$

Para calcular cuánto tarda el programa el ejecutarse en total tendremos que hacer lo siguiente:

- $T_{p_{N=2}} = \text{MAX}(T_{pp1}, T_{pp2}) = 15 \text{ s}$

Lo cual se puede explicar diciendo que si repartimos el trabajo de forma equivalente sin tener en cuenta las especificaciones de cada procesador, el más lento será nuestro factor condicionante.

A continuación calculamos el Speed-Up y la eficiencia, en este caso para el tiempo secuencial cogeremos el más rápido de los dos.

GANANCIA

$$SP = \frac{Ts}{Tp}$$

$$SP = \frac{20}{15} = 1.33 \leq 2 :)$$

EFICIENCIA

$$E = \frac{SP}{N}$$

$$E = \frac{1.33}{2} = 0.66 \leq 1 :)$$

Nota: Ganancia máxima teórica == numero de nodos

- b) Que distribución de carga entre los dos procesadores P1 y P2 permite el menor tiempo de ejecución utilizando los dos procesadores en paralelo. Calcula el tiempo.

$$x_1 = x \quad tpp_1(x) = tpp_2(1-x)$$

$$x_2 = 1 - x \quad 20(x) = 30(1-x)$$

$$5x = 3 \rightarrow x = 3/5$$

A tpp_1 se le dan $3/5$ del problema y a tpp_2 se le dan $2/5$ del problema. tpp_1 procesa mas parte del problema ya que es más rápido. Al haber distribuido la carga, ambos procesadores tardarán lo mismo a la hora de resolucionar el problema:

$$3/5 * 20 = 12 \text{ s}$$

$$2/5 * 30 = 12 \text{ s}$$

3. Cuál es la fracción de código paralelo de un programa secuencial que, ejecutado en paralelo en 8 procesadores tarda un tiempo de 100 ns, asumiendo que durante 50 ns utiliza un único procesador y durante los otros 50 los 8 procesadores. Todos los procesadores son iguales. Hallar el tiempo secuencial.

Lo primero que haremos será hallar el tiempo secuencial.

$$t_{sec} = 8 * 50 + 50 = 450 \text{ ns}$$

Una vez tenemos el tiempo secuencial total podemos hacer lo mismo que hicimos en el ejercicio anterior:

$$T_p = 100 \text{ ns} = x * 450 + ((1-x) * 450) / 8$$

La x es el valor que queremos hallar por lo tanto despejamos:

$$x = 1/9$$

Ahora bien, ese $1/9$ sería la fracción de código sin parallelizar, pues para hallar la fracción de código paralelizado haremos:

$$1 - 1/9 = 8/9$$

b) Calcula la ganancia y la eficiencia en paralelo.

$$SP = T_{sec} / T_p$$

$$SP = 450 / 100 = 4.5 \leq 8 :)$$

$$E = SP / N$$

$$E = 4.5 / 8 = 0.5625 \leq 1 :)$$

4. El 25% de un programa no se puede parallelizar y el resto se puede distribuir por igual entre cualquier número de procesadores.
- ¿Cuál es el máximo valor de ganancia en velocidad que se podría conseguir al parallelizarlo en p procesadores?
 - ¿Y con infinitos?
 - A partir de qué numero de procesadores podríamos conseguir ganancias superiores o iguales a 2?
 - Calcula la eficiencia
 - Máxima eficiencia que puedo obtener cuando p tiende a ∞

a) $SP = tsec/tpp = tsec / ((0.25)*tsec + (0.75*(1/p))*tsec) =$
 $= 4p/(3+p)$

Para 4 nodos por ejemplo: $4*4/(3+4) = 16/7 = 2.28 \geq 4$:)

b) $\lim_{(p \rightarrow \infty)} SP = 4p/(3+p) = 4\infty/\infty = 4$

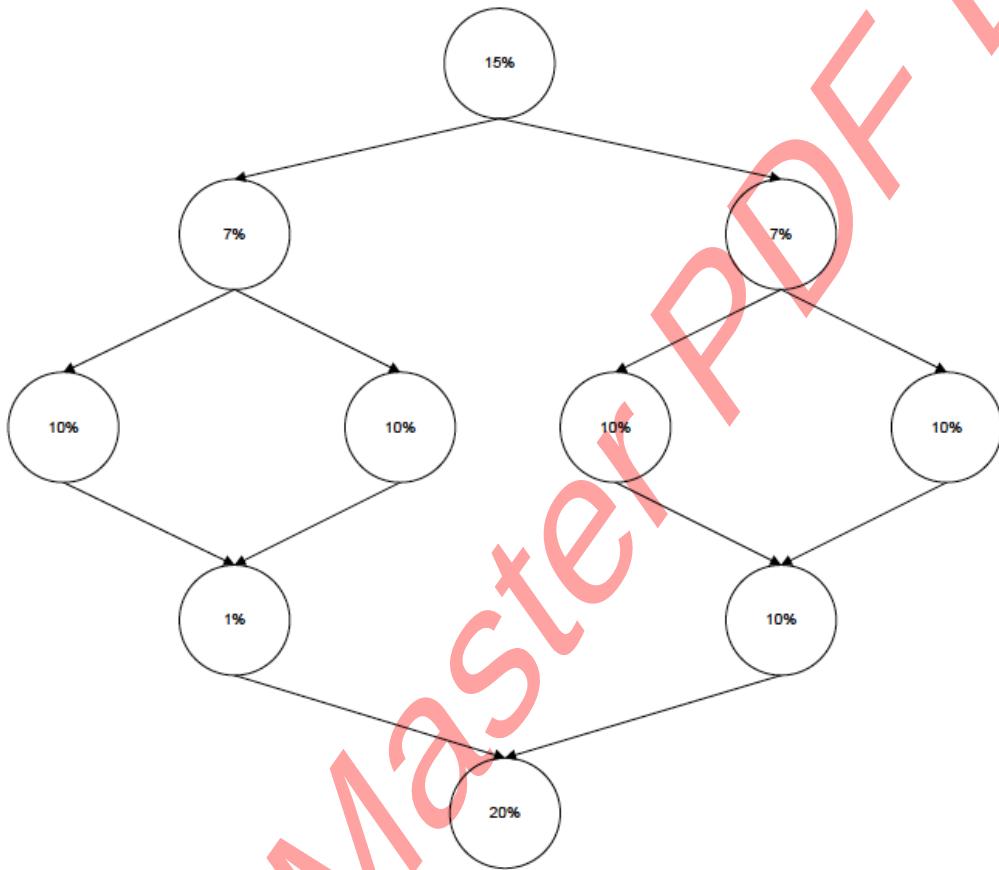
c) $4p/(3+p) = 2 \rightarrow \text{Despejamos } p \rightarrow p = 3$

d) $E = SP/N = (4p/(3+p))/p$

e) $\lim_{(p \rightarrow \infty)} E = 0$

Si metemos nodos infinitos la eficiencia acabará siendo 0.

5. Tenemos un programa que hemos dividido en 10 tareas, las cuales tardan en ejecutarse 5 segundos. En la siguiente figura podemos ver como se divide el cómputo entre las distintas tareas y el orden de precedencia de las mismas. Si disponemos de 4 procesadores, se pide calcular el tiempo de ejecución de la versión paralela del programa, así como la ganancia en velocidad obtenida al realizar la parallelización.



Tiempo de ejecución en paralelo:

$$Tp(n) = Ts \cdot (0.15 + \max(0.07, 0.07) + \max(0.1, 0.1, 0.1, 0.1) + \max(0.1, 0.1) + 0.2)$$

$$Tp(n) = Ts \cdot (0.15 + 0.07 + 0.1 + 0.1 + 0.2) = Ts \cdot 0.62 = 5 \cdot 0.62 \rightarrow Tp(n) = 3.1 \text{ s}$$

Ganancia:

$$S(p,n) = Ts / Tp(n) = 5 / 3.1 \rightarrow S(p,n) = 1.61$$

6. Se quiere parallelizar el siguiente trozo de código:

```
// {Cálculos antes del bucle} | t1
for(int i = 0; i < w; i++) {
    // iteración i | ti
}
// {Cálculos después del bucle} | t2
```

Los cálculos después del bucle supone t_2 , los de antes del bucle t_1 . Cada iteración del bucle supone un tiempo t_i . En la ejecución paralela, la inicialización de p procesos supone un tiempo con k_1 constante ($k_1 \cdot p$). La comunicación y sincronización supone un tiempo k_2 ($k_2 \cdot p$).

- a) Hallar $tp(p)$
- b) Hallar el Speed Up de p
- c) ¿Existe un mínimo de p para la ejecución en paralelo?

a) Llamamos t' al tiempo que no se puede parallelizar

$$t' = t_1 + t_2$$

Al tener overhead lo dejamos listo así.

$$K_1 p \rightarrow t_0(p) = (k_1 + k_2)p = k'p; \text{ siendo } k' = k_1 + k_2$$

Tenemos w iteraciones paralelizables por p procesadores, por lo tanto:

$$tp(p) = t' + k'p + \lceil \frac{w}{p} \rceil * t_i$$

$tp(p) = t$. no parallelizable + overhead + tiempo paralelizable.
 $\lceil \frac{w}{p} \rceil$ implica coger el siguiente entero ya que no podemos tener números no enteros (ver OMP práctico).

b) $SP(p) = t_{sec}/tpp = (t' + t_i \cdot w) / tp(p)$

- c) Para simplificar $w \cdot t_i = k''$
 Si $tp(p) = t' + k'p + k''/p$

Derivamos

$$tp'(p) = 0 + k' - k''/p^2$$

Igualamos la derivada a 0 para sacar el punto de inflexión.

$$k' = k''/p^2;$$

$$p = +\sqrt{(k''/k')} \rightarrow p = +\sqrt{((w \cdot t_i)/k')}$$

(Cogemos el resultado positivo $+\sqrt{\cdot}$ ya que no puede haber un número de procesadores negativo)

Hayamos la segunda derivada y comprobamos que efectivamente sí que existe un mínimo

$$tp''(p) = +k''/p^3 \rightarrow tp''(p) = + (w \cdot t_i)/p^3$$

Sustituimos el punto crítico dado en la 1^a derivada en la 2^a

$$tp''(p) = (w \cdot t_i)/(+\sqrt{((w \cdot t_i)/k')})^3$$

Como podemos observar no cabe posibilidad alguna que el resultado de la segunda derivada sea menor que 0. Por lo tanto obtendríamos un mínimo.

$X > 0 \rightarrow \text{Mínimo}$

$X < 0 \rightarrow \text{Máximo}$

$X = 0 \rightarrow \text{Punto de inflexión}$

1. Para un hipercubo de dimensión 6, ¿Cuántos nodos tiene el nodo 13 a distancia 2?

Nodo 13 = 001101 al ser 6 dimensiones, trataremos con 6 dígitos

Por lo tanto, queremos calcular todos los nodos que estén a distancia hamming de 2. Para ello:

$$\binom{n}{d} = \frac{n!}{d!(n-d)!}$$

n = Número de dimensiones
d = Distancia buscada

$$\binom{6}{2} = \frac{6!}{2!(6-2)!} = 15$$

El nodo 13 del hipercubo tiene 15 nodos a distancia 2.

2. Un multicomputador utiliza una red de comunicación en la que los enlaces son de 1Gb/s. La técnica de comunicación es almacenamiento y reenvío. Mandar un paquete de 32 bytes a una distancia de 6 cuesta $1.56\mu s$. ¿Cuántas veces sería más rápida la comunicación si la técnica de comunicación/comutación fuera Virtual Cut-Through (VCT)? Se supone tráfico 0, flits de 8 bits y un flit de cabecera.

$$s\&f \rightarrow t_{s\&f} = D(tr + tw) + D*tw[L/W]$$

$$vct \rightarrow t_{vct} = D(tr + tw) + tw[L/W]$$

$$t_{s\&f} = 1.56\mu s \rightarrow 1560 \text{ ns}$$

$$D = 6$$

$$tr ??$$

$$tw ?? \rightarrow (\text{numero bits del flit}) * 1/\text{AnchoBanda}$$

$$1Gb/s \rightarrow 10^9 \text{ bits/s}$$

$$tw (\text{para un flit}) = 8 * (1/10^9) = 8 \text{ ns};$$

De los 32 bytes que enviamos omitimos la cabecera, trabajamos en bytes ya que t_r y t_w es tiempo por flit, y un flit son 8 bits $\rightarrow 1$ byte.

$$1560 \text{ ns} = 6(tr + 8\text{ns}) + 6 * 8 * (31)$$

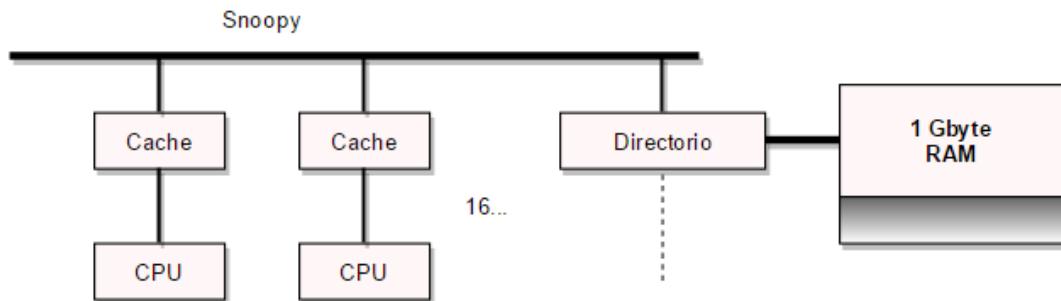
$$tr = 4 \text{ ns}$$

¿Ahora bien cuánto es más rápido?

$$t_{vct} = 6(4 + 8) + 8*(31) = 320 \text{ ns}$$

$$SP = t_{s\&f} / t_{vct} \rightarrow 1560 / 320 \approx 4$$

3. Tengo una arquitectura SMP p+1 centralizado con caches. Disponemos de 32 clusters snoopy (1 por directorio) en los cuales tenemos 16 procesadores por clúster. Los clusters disponen de 1GByte de memoria y 1 Mbyte de caché, siendo las líneas de ésta última de 128 bytes, además necesitaremos 2 bits por cada línea de caché para que ésta pueda ser referenciada. ¿Cuál será la sobrecarga de memoria necesaria para mantener este sistema?



EN CACHÉ:

- Líneas de caché: $\frac{1 \text{ Mbyte}}{128 \text{ bytes}} = \frac{2^{20}}{2^7} = 2^{13} = 2^{10} + 2^3 = 8\text{K líneas}$
- Número de cachés: $32 * 16 = 512$. **Cada procesador tiene su caché.**
- $512 \text{ caches} * 8\text{k líneas} * 2 \text{ bits/lineacache} = 2^{23} = 1\text{MByte extra por cache.}$

EN DIRECTORIO:

- **P+1 = 32+1 (un bit por nodo + 1 adicional) = 33 bits por directorio**

Característica del p+1 centralizado, un bit por nodo más uno adicional. Recordamos que estamos en el directorio.

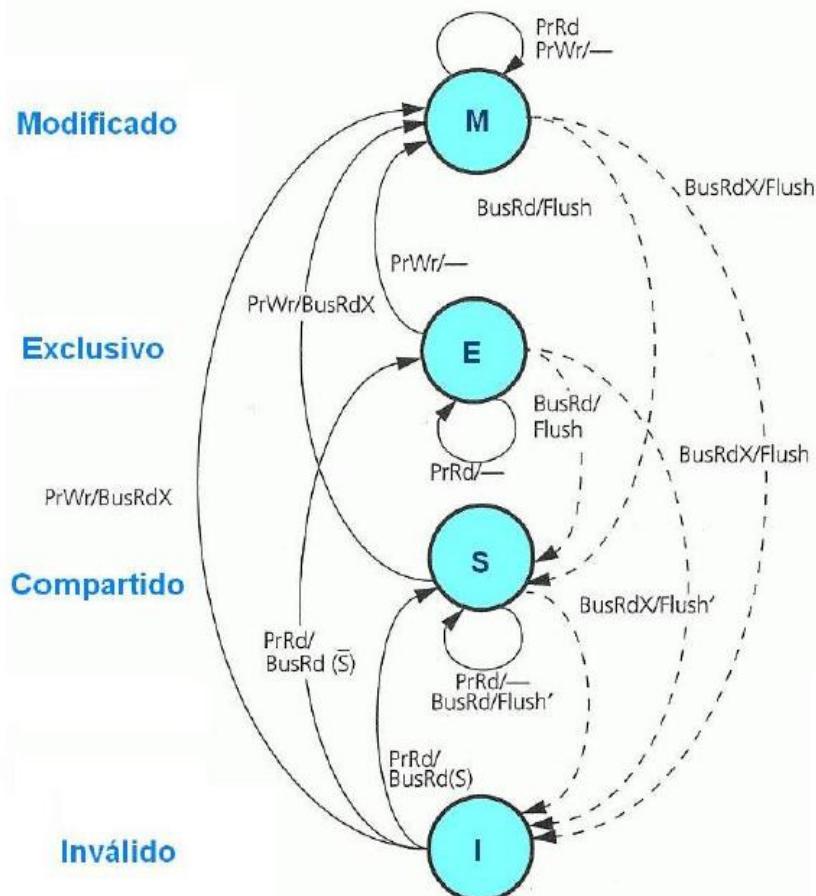
- Bloques de RAM: $\frac{1 \text{ Gbyte}}{128 \text{ bytes}} = 8 \text{ MBloques por directorio.}$
- $8 \text{ MB/directorio} * 33 \text{ bits/directorio} * 32 \text{ directorios} = 8448 \text{ Mbits}$

4. Disponemos de un multiprocesador con 4 nodos, cada nodo tiene caché, el protocolo snoopy (o de sondeo) elegido es el Illinois (*NOTA: Llamado también MESI, pero tened cuidado y aprenderos los dos nombres*). Las caches pueden alojar un máximo de 2 palabras, siendo el tamaño de bloque de 1 palabra y la política de reemplazo aleatoria. Dadas las siguientes referencias a memoria, indique cómo evoluciona el estado de las cachés y de la memoria principal.

```
rd1 @1; wr1 (@1, 22);
wr3 (@2, -11); rd2 @4
```

Cache 1			Cache 2			Cache 3			Cache 4			MEMORIA PRINCIPAL		
@1	5	E	@1	-3	I	@2	9	S	@2	9	S	@1	5	@2 9
@2	9	S	@5	2	E	@3	6	M	@3	10	M	@3	10	@4 10
									@4	10	M	@5	2	@6 9

Lo primero, dibujamos el diagrama MESI



1. Acceso 1. rd1 @1

Se accede a la línea 1 de la caché 1. ¿Se produce fallo de caché?

Cache 1			Cache 2			Cache 3			Cache 4			MEMORIA PRINCIPAL			
@1	5	E	@1	-3	I	@2	9	S	@2	9	S	@1	5	@2	9
@2	9	S	@5	2	E	@3	6	M	@4	10	M	@3	10	@4	10

→ Acierto en lectura

Pr1Rd/- → P1 lee, no se inyecta nada en el bus

Pasa de estado E a estado I.

2. Acceso 2. Wr1 (@1, 22)

Se accede a la línea 1 de la caché 1. ¿Se produce fallo de caché? Se escribe en la línea 1 de la caché 1 el valor 22. (El @ indica la línea, el numero detrás de la instrucción el número de caché)

Cache 1			Cache 2			Cache 3			Cache 4			MEMORIA PRINCIPAL			
@1	22	M	@1	-3	I	@2	9	S	@2	9	S	@1	22	@2	9
@2	9	S	@5	2	E	@3	6	M	@4	10	M	@3	10	@4	10

Al actualizar la caché 1, se actualiza también la memoria principal.

3. Acceso 3. Wr3 (@2, -11)

Se accede a la línea 1 de la caché 3 y se escribe un -11.

Cache 1			Cache 2			Cache 3			Cache 4			MEMORIA PRINCIPAL			
@1	22	M	@1	-3	I	@2	-11	M	@2	9	I	@1	22	@2	-11
@2	9	I	@5	2	E	@3	6	M	@4	10	M	@3	10	@4	10

4. Acceso 4. Rd2 @4

Se accede a la línea 4 de la caché 2 y se lee el contenido de la misma. Como la línea 4 en la caché 2 no existe, tenemos que referencia a la política de reemplazo expuesta en el enunciado, en este caso, aleatoria.

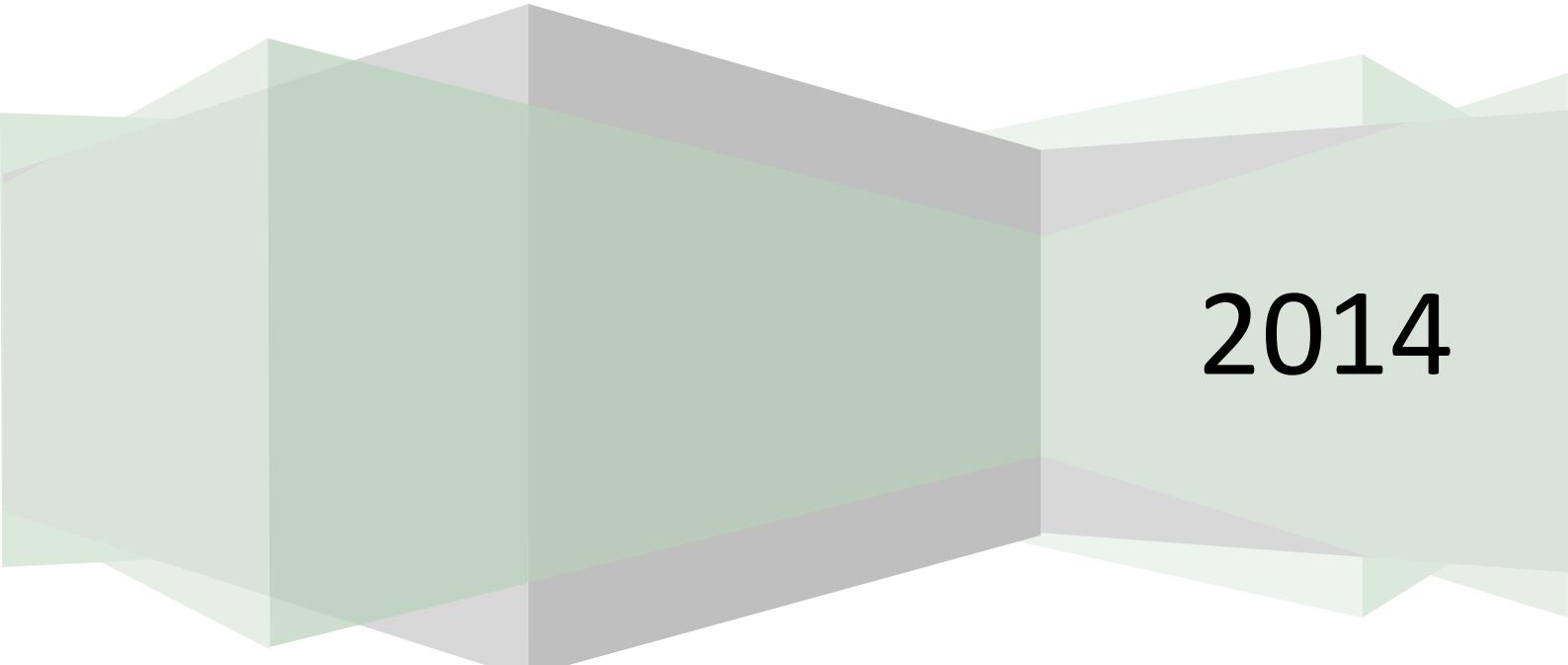
Cache 1			Cache 2			Cache 3			Cache 4			MEMORIA PRINCIPAL			
@1	22	M	@4	10	S	@2	-11	M	@2	9	I	@1	22	@2	-11
@2	9	I	@5	2	E	@3	6	M	@4	10	S	@3	10	@4	10

Juan José Conejero Serna

Kit de supervivencia para aprobar IC en Julio

Sólo para valientes

Grado Ingeniería Informática UA



2014

SUPERESCALARES

CAUCE

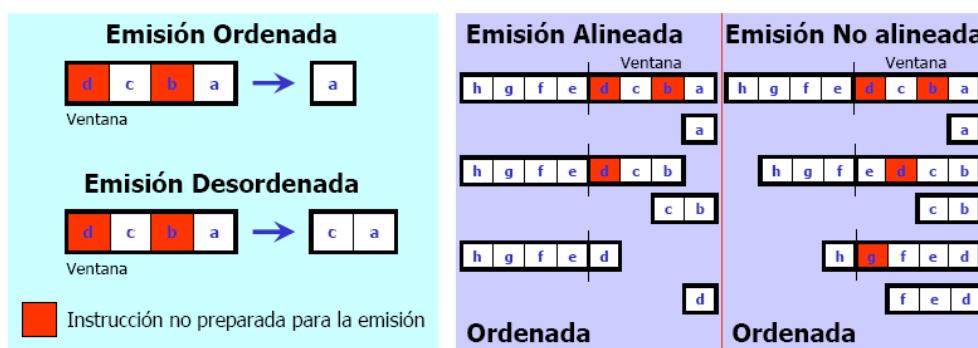
En una arquitectura superescalar, aparte de las etapas comunes de una máquina normal, (IF, ID, EX, WEB) se añade la etapa de Emisión de instrucciones (ISS), que controla que puedan existir más de una operación realizándose a la vez.

EMISIÓN DE INSTRUCCIONES (ISS)

La ventana de instrucciones almacena las instrucciones pendientes ya decodificadas y se utiliza un bit para indicar si esa instrucción está disponible. Por su naturaleza, la emisión de las instrucciones depende del *alineamiento* y del *orden*:

La emisión es **alineada** si no pueden introducirse nuevas instrucciones en la ventana de instrucciones hasta que ésta no esté totalmente vacía. En la emisión **no alineada**, mientras que exista espacio en la ventana, se pueden ir introduciendo instrucciones para ser emitidas.

En la emisión **ordenada** se respeta el orden en que las instrucciones se han ido introduciendo en la ventana de instrucciones; si una instrucción incluida en la ventana de instrucciones no puede emitirse, las instrucciones que la siguen tampoco podrán emitirse, aunque tengan sus operandos y la unidad que necesitan esté disponible. En cambio, en la emisión **desordenada** no existe este *bloqueo*, ya que pueden emitirse todas las instrucciones que dispongan de sus operandos y de la correspondiente unidad funcional.



ESTRUCTURAS

RENOMBRADO DE REGISTROS

En los procesadores con finalización desordenada, se pueden producir riesgos de dependencias WAR o WAW (Write After Read o Write After Write), y para ello haremos uso del **renombrado de registros y reorden de los mismos**, y dentro de ellos existen dos tipos: **Implementación Dinámica**, que se realizan durante la ejecución y requieren circuitería adicional, y la **Implementación Estática**, que se realiza durante la compilación y es la que vemos a continuación:

3 Kit de supervivencia para aprobar IC en Julio

BUFFER RENOMBRAMIENTO

- Buffer de Renombrado de **Acceso Asociativo**
 - Permite varias escrituras pendientes a un mismo registro
 - Se utiliza el último bit para marcar cual ha sido la más reciente
- Buffer de Renombrado de **Acceso Indexado**
 - Sólo permite una escritura pendiente a un mismo registro
 - Se mantiene la escritura más reciente.

BUFFER REORDEN (ROB)

Una vez que hemos realizado todas las operaciones, los resultados se encuentran en el buffer de renombramiento, el **problema** surge al decidir el momento en que los resultados se escriben en los registros. Esto, puede coincidir o no con el **orden** en que las instrucciones estén en el programa, pero en cualquier caso, debe respetarse la *semántica* del programa. Para ello utilizaremos el **ROB**.

Además el **ROB** permite gestionar correctamente el procesamiento especulativo de las instrucciones de **salto** y las interrupciones; además que también se puede utilizar para el renombramiento.

PROCESAMIENTO DE INSTRUCCIONES DE SALTO

El efecto en los saltos de los superescalares es más pernicioso que en el resto, ya que en cada ciclo puede haber una instrucción de salto. Y también se compone de diferentes etapas:

- **Detección de la instrucción de salto:** Si se detecta la instrucción de un salto condicional en el momento de la captación y se conoce la dirección de destino en ese mismo ciclo, en el siguiente ciclo se pueden captar instrucciones a partir de esa dirección sin penalización de tiempo. Las posibilidades son:
 - **Detección paralela.**
 - **Detección anticipada**
 - **Detección integrada por captación**
- **Gestión de saltos condicionales no resueltos:** Si en el momento de evaluar la instrucción aun no se ha evaluado. Y se puede o utilizar un proceso especulativo o una comprobación directa.
- **Acceso a las instrucciones de destino del salto:** hay que implementar procesos que permitan el acceso más rápido a la secuencia de instrucciones.

Dentro de éste tipo de predicciones podemos encontrar dos tipos: predicción fija y predicción verdadera, que ésta segunda a su vez se descompone en predicción estática y dinámica.

Predicción Fija		
Siempre No Tomado		<ul style="list-style-type: none"> ▪ Toda condición de salto no resuelta, se predice que no da lugar a un salto. ▪ Después se evalúa si era buena.
Siempre Tomado		<ul style="list-style-type: none"> ▪ Toda condición de salto no resuelta, se predice que da lugar a un salto. ▪ Después se evalúa si era buena.
Predicción Verdadera		
Predicción Estática		<ul style="list-style-type: none"> ▪ Para ciertos códigos de operación, se predice que el salto se toma, y para otros que el salto no se toma. ▪ Saltos hacia atrás (bucles) y saltos hacia delante (if,then,else)
Predicción Dinámica	<i>Pr. Dinámica Implícita</i>	Predecir hacer lo mismo que ocurrió la última vez que se ejecutó esa instrucción
	<i>Pr. Dinámica Estática</i>	Para cada instrucción de salto condicional, existe unos bits de historia que codifican la información del pasado de la instrucción.

PARALELISMO

CONCEPTOS BÁSICOS

- **Procesamiento distribuido:** Ejecución de múltiples aplicaciones al mismo tiempo utilizando múltiples recursos, situados en distintas localizaciones físicas.
- **Procesamiento paralelo:** división de aplicaciones en unidades independientes, y su ejecución en varios procesadores.

Y la clasificación de los computadores paralelos es:

- **Multiprocesadores:** Comparten el mismo espacio de memoria.
 - Mayor Latencia
 - Poco Escalable
 - Variables compartidas
 - Prog. Sencilla
- **Multicomputadores:** Cada procesador (o nodo) tiene su propio espacio de direcciones.
 - Menor Latencia
 - Mayor Escalabilidad
 - Paso de mensajes
 - Prog. Complicada

TIPOS DE PARALELISMO

Paralelismo de Datos

Implícito en operaciones con estructuras de datos tipo *vector* o *matriz*. Grandes volúmenes de datos independientes entre sí (superescalares y máquinas segmentadas).

Paralelismo Funcional

Reorganización de la estructura lógica de una aplicación. Nivel de Programas, Funciones, Bloques en funciones u Operaciones (granularidad de gruesa a fina).

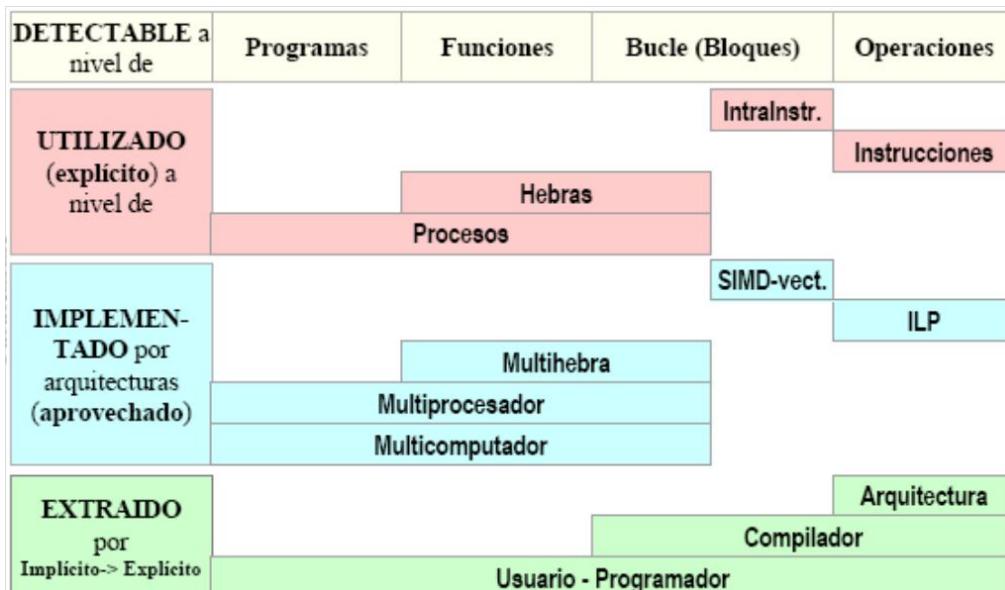
- *Explícito*: No presente de forma inherente en las estructuras de programación y que se debe indicar expresamente.
- *Implícito*: Presente debido a la propia estructura de los datos (vectores) o de la aplicación.

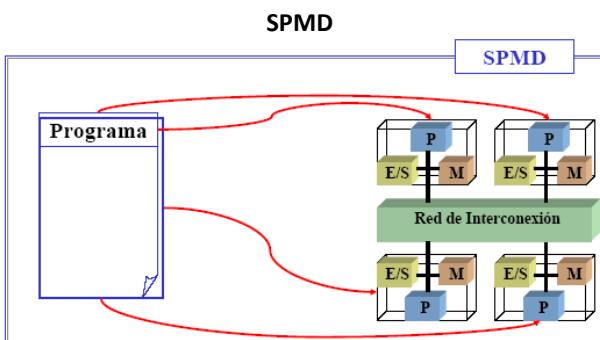
UNIDADES DE EJECUCIÓN

Hardware: Gestiona la ejecución de instrucciones (a nivel de procesador).

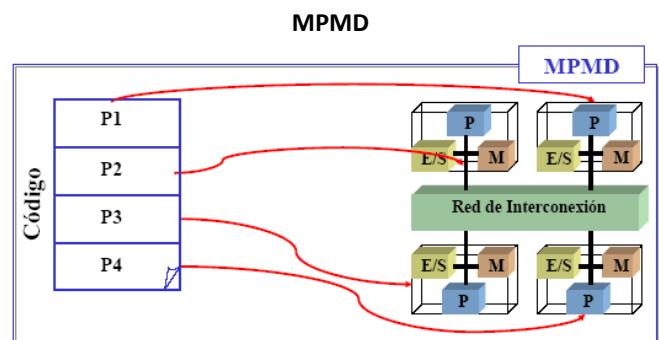
Software: Gestiona la ejecución de hilos y procesos.

- *Proceso*: espacio de direcciones virtuales propio.
- *Hilos*: comparten direcciones virtuales, se crean y destruyen rápido y su comunicación también.



Modos de programación paralela:

Cada procesador realiza un bloque de código (if, for)

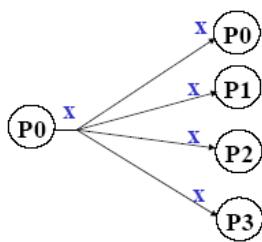


Se divide el programa en trozos y cada uno hace una función.

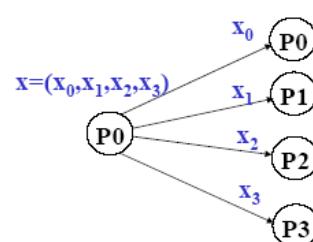
Comunicación entre estructuras paralelas:

Uno a todos

Difusión (*broadcast*)

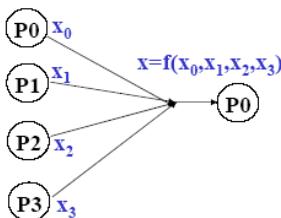


Dispersión (*scatter*)

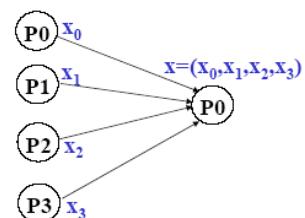


Todos a uno

Reducción



Acumulación (*gather*)



PROBLEMAS DE PARALELISMO

LEY DE AMDHAL

Un principio básico: Hacer rápidas las funciones frecuentes --> Gastar recursos donde se gasta el tiempo.

Ley de Amdahl: Permite caracterizar este principio

Permite la evaluación del speedup que se obtendrá al aplicar una cierta mejora, M, que permite ejecutar una parte del código x veces más rápido.

$$\text{Speedup}(E) = \frac{\text{TEj sin M}}{\text{TEj con M}} = \frac{\text{Performance con M}}{\text{Performance sin M}}$$



Si la mejora sólo acelera la ejecución de un fracción F de la tarea, el tiempo de ejecución del resto permanece sin modificación. Por tanto es muy importante el porcentaje de la tarea que es acelerada.

$$F = \frac{t_B}{t_A + t_B + t_C}$$

LEY DE AMDHAL

$$\text{TEj}_{\text{nuevo}} = \text{TEj}_{\text{antiguo}} \times [(1 - \text{Fraccion}_{\text{mejora}}) + \text{Fraccion}_{\text{mejora}} / X]$$

$$\text{Speedup} = \text{Tej}_{\text{antiguo}} / \text{Tej}_{\text{nuevo}} = 1 / [(1 - \text{Fraccion}_{\text{mejora}}) + \text{Fraccion}_{\text{mejora}} / X]$$

Ejemplo 1: El 10% del tiempo de ejecución de mi programa es consumido por operaciones en PF. Se mejora la implementación de la operaciones PF reduciendo su tiempo a la mitad

$$\text{TEj}_{\text{nuevo}} = \text{TEj}_{\text{antiguo}} \times (0.9 + 0.1 / 2) = 0.95 \times \text{TEj}_{\text{antiguo}} \quad \text{Speedup} = 1 / 0.95 = 1.053$$

Mejora de sólo un 5.3%

Ejemplo 2: Para mejorar la velocidad de una aplicación, se ejecuta el 90% del trabajo sobre 100 procesadores en paralelo. El 10% restante no admite la ejecución en paralelo.

$$\text{TEj}_{\text{nuevo}} = \text{TEj}_{\text{antiguo}} \times (0.1 + 0.9 / 100) = 0.109 \times \text{TEj}_{\text{antiguo}} \quad \text{Speedup} = 1 / 0.109 = 9.17$$

El uso de 100 procesadores sólo multiplica la velocidad por 9.17

REDES DE INTERCONEXIÓN

Elemento fundamental en arquitecturas paralelas con varios elementos de proceso que se comunican.

Los *parámetros básicos* que tienen estos elementos son:

- **Número de nodos (N)**
- **Grado del nodo (D)**: Número de canales de entrada Y salida.
 - Nodos unidireccionales: Grado de salida y Grado de entrada
- **Diámetro de la red**: Longitud máxima del camino más corto entre dos nodos cualquiera

CLASIFICACIÓN DE REDES ESTÁTICAS (DIRECTAS)

Estrictamente Ortogonales: Cada nodo tiene al menos un enlace en cada dimensión, y supone un desplazamiento en una dimensión.

No ortogonales: estructura de árbol.

Propiedades:

- Grado
- Diámetro
- Simetría (se ve semejante desde cualquier nodo)
- Regularidad (todos los nodos tienen el mismo grado)

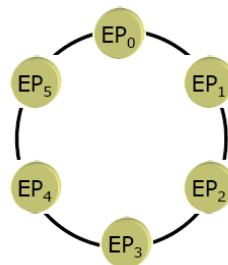
TIPOS DE REDES ESTÁTICAS

Anillo Unidireccional

➤ F. interconexión: $F+1(i) = (i+1) \bmod N$

➤ Grado de entrada/salida: 1/1

➤ Diámetro: $N-1$



Malla Abierta

F. interconexión:

➤ $F+1(i) = (i+1) \bmod r$ si $i \bmod r < r-1$

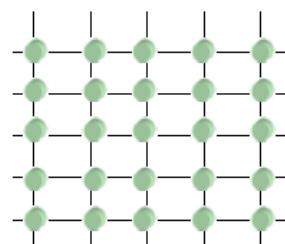
➤ $F-1(i) = (i-1) \bmod r$ si $i \bmod r > 0$

➤ $F+r(i) = (i+r) \bmod r$ si $i \bmod r > 0$

➤ $F-r(i) = (i-r) \bmod r$ si $i \bmod r < r-1$

➤ Grado: 4

➤ Diámetro: $2(r-1)$, donde $N=r^2$



Malla Illiac

F. interconexión:

➤ $F+1(i) = (i+1) \bmod N$

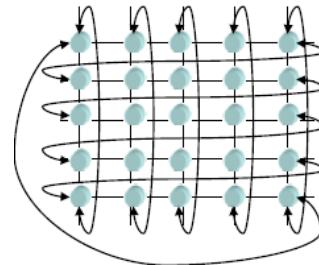
➤ $F-1(i) = (i-1) \bmod N$

➤ $F+r(i) = (i+r) \bmod N$

➤ $F-r(i) = (i-r) \bmod N$

➤ Grado: 4

➤ Diámetro: $(r-1)$, donde $N=r^2$

**Malla toro**

➤ n dimensiones, k nodos

➤ F. interconexión toro 2D:

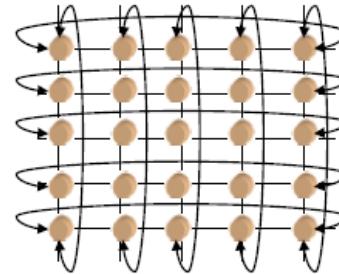
➤ $F+1(i) = (i+1) \bmod r + (i \text{ DIV } r) \cdot r$

➤ $F-1(i) = (i-1) \bmod r + (i \text{ DIV } r) \cdot r$

➤ $F+r(i) = (i+r) \bmod N$

➤ $F-r(i) = (i-r) \bmod N$

➤ Grado: 4



➤ Diámetro: $2 \cdot \frac{r}{2}$, donde $N=r^2$

Desplazador Barril

➤ F. interconexión:

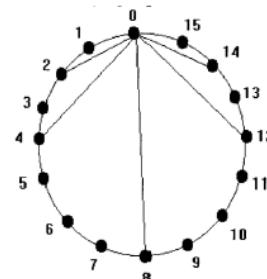
➤ $B+k(i) = (i+2^k) \bmod N$

➤ $B-k(i) = (i - 2^k) \bmod N$

➤ $K=0 \dots n-1$, $n=\log N$, $i=0 \dots N-1$

➤ Grado: $2n - 1$

➤ Diámetro: $n/2$

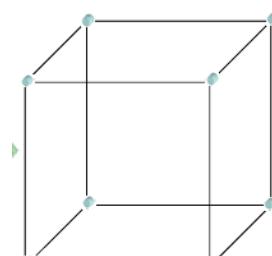
**Hipercubo**

➤ F. interconexión:

➤ $F_i(h_{n-1}, \dots, h_i, \dots, h_0) = h_{n-1}, \dots, h_i, \dots, h_0$

➤ Grado: n ($n=\log N$)

➤ Diámetro: n



Árbol Binario

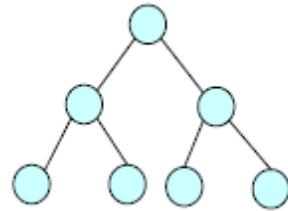
➤ Balanceado: todas las ramas del árbol tienen la misma longitud

➤ Cuello de botella → nodo raíz

➤ N (balanceado) = $2^k - 1$ (k = niveles del árbol)

➤ Grado: 3

➤ Diámetro: $2(k-1)$

**CLASIFICACIÓN DE REDES DINÁMICAS (INDIRECTAS)**

Uso de conmutador: Las redes dinámicas se caracterizan por tener conmutadores (interruptores que desvían la red a un lado o a otro) para la interconexión entre nodos.

Modelo $G(N, C)$:

- N , conjunto conmutadores
- C , enlaces entre conmutadores

Distancia entre nodos: distancia entre los conmutadores que conectan los nodos + 2

TIPOS DE REDES DINÁMICAS

<u>Crossbar</u>	<p>➤ Conexión directa nodo-nodo ➤ Gran ancho de banda y capacidad de interconexión ➤ Conexión Proc. – Mem. → limitado por los accesos a memoria (columnas) ➤ Conexión Proc(N) – Proc(N) → máximo de N conexiones ➤ Coste elevado: $O(N \cdot M)$</p>	
<u>Redes Min</u>	<p>➤ Conectan dispositivos de entrada con dispositivos de salida mediante un conjunto de etapas de conmutadores, donde cada conmutador es una red crossbar. ➤ Concentradores → nº entradas > nº salidas ➤ Expansores → nº salidas > nº entradas ➤ Conexión de etapas adyacentes → Patrón de conexión ➤ Patrón basado en permutaciones: conmutadores con el mismo número de entradas y salidas. ➤ Número de entradas a_n y número de salidas b_n (red $a_n \times b_n$) ➤ n etapas de conmutadores (C_0, C_1, \dots, C_{n-1}) ➤ Conmutadores $a \times b$ ➤ $a_{n-1-i} \times b_i$ conmutadores en la etapa C_i ➤ Funcionalidad de los conmutadores: barras cruzadas, reducción, difusión ➤ Subred de interconexión entre etapas: R_0, R_1, \dots ➤ Tipos de canales: unidireccionales, bidireccionales</p>	<p style="text-align: center;"> Red Omega Red Mariposa Red Cubo Red Delta </p>

TÉCNICAS DE CONMUTACIÓN

Existen 5 técnicas de conmutación, pero solo veremos las resaltadas en negrita: **Store & Forward**, **Womhole**, Cut-Through y Conmutación de Circuitos.

Store & Fordward

- El conmutador almacena el **paquete completo** antes de ejecutar el algoritmo de encaminamiento.
- La unidad de transferencia (paquete) entre interfaces ocupa **sólo un canal** en cada instante.
- Almacenamiento en conmutadores: **múltiplos de un paquete**.
- El número de enlaces ociosos (libres) influye en el ancho de banda: para un tamaño de *buffer* 1, un paquete bloqueado deja ocioso un canal.

Latencia de transporte

$$T_{AR} = D \cdot \left[T_r + T_w \cdot \left(\frac{L}{W} + 1 \right) \right]$$

- **1 flit** = w bits
 - **Cabecera** = 1 flit
 - Tamaño total del paquete = L bits + w bits (cabecera)
 - **D**= Distancia fuente-destino = D parejas conmutador-enlace
 - **T_w** = tiempo para que un phit atravesese una etapa conmutador/enlace
 - **T_r** = tiempo de encaminamiento (routing)
- 

Wormhole

- En cuanto llega la cabecera al conmutador se ejecuta el algoritmo de encaminamiento y se reenvía.
- La unidad de transferencia es el mensaje
- La transferencia se hace a través de un camino segmentado. La unidad de transferencia puede ocupar varios canales.
- Almacenamiento en conmutadores: múltiplos de un flit.
- El número de enlaces ociosos influye en el ancho de banda: para un tamaño de buffer 1, un paquete bloqueado deja ociosos varios canales.

Latencia de transporte

$$T_V = D \cdot (T_r + T_w) + T_w \cdot \left[\frac{L}{W} \right]$$

T_V: T_{cabecera}+T_{resto}

MEMORIA

TIPOS

La clasificación de procesadores atendiendo a la distribución de memoria es:

UNA (UNIFORM MEMORY ACCESS)

- **Memoria centralizada**, compartida entre procesadores
- **Dependencia funcional** entre procesadores (*fuertemente acoplado*)
- Cada procesador dispone de un **caché**
- Periféricos **compartidos** entre procesadores
- **Sincronización** entre procesadores utilizando variables compartidas.

NUMA (NON-UNIFORM MEMORY ACCESS)

- **Memoria compartida** con tiempo de acceso dependiente de la ubicación de procesadores
- Cada procesador dispone de una **memoria local** (*débilmente acoplado*)
- Sistema de **acceso global** a memoria, **también**.
- **Ventajas**: Escalado de memoria con + coste/rendimiento y reducción de latencia.

COMA (CACHÉ-ONLY MEMORY ARCHITECTURE)

- Sólo se usa una **caché** como memoria
- Caso particular de *NUMA* donde **las memorias distribuidas se convierten en cachés**
- Las cachés forman un mismo **espacio global de direcciones**
- Acceso a las cachés por **directorio distribuido**

CONSISTENCIA DE MEMORIA

*"Un modelo de consistencia de memoria especifica el orden en el cual las operaciones de acceso a memoria deben **parecer** haberse realizado".*

Se debe garantizar:

1. Cada lectura de una dirección proporcione el **último valor** escrito en dicha dirección
2. Si se **escribe varias** veces en esa dirección se debe retornar el último valor escrito.
3. Si se escribe donde se ha leído anteriormente, **no se debe obtener la lectura previa** a la escritura
4. No se puede escribir en una dirección si la escritura depende de una **condición** que **no se cumple**.

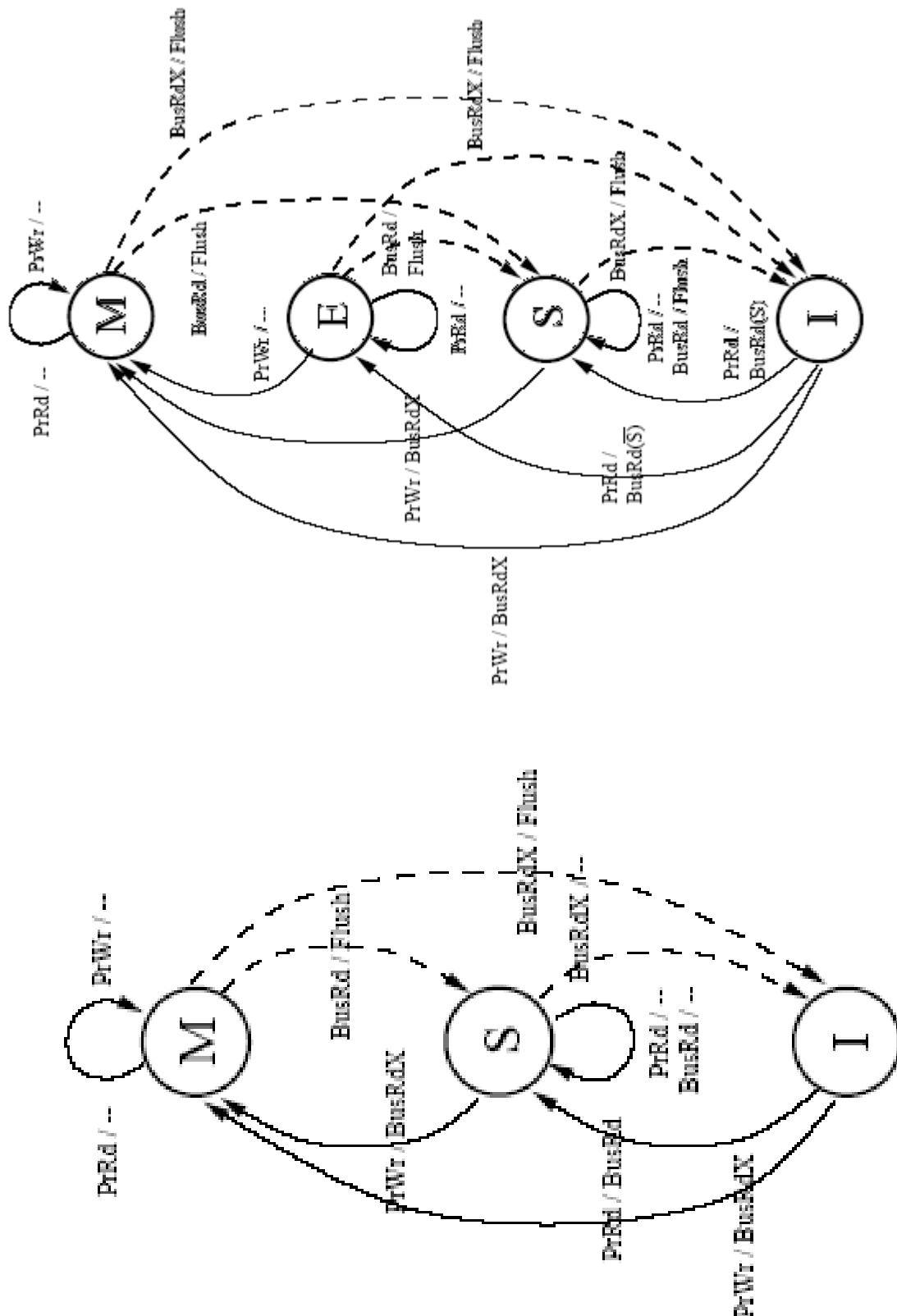
Un sistema de memoria es coherente si:

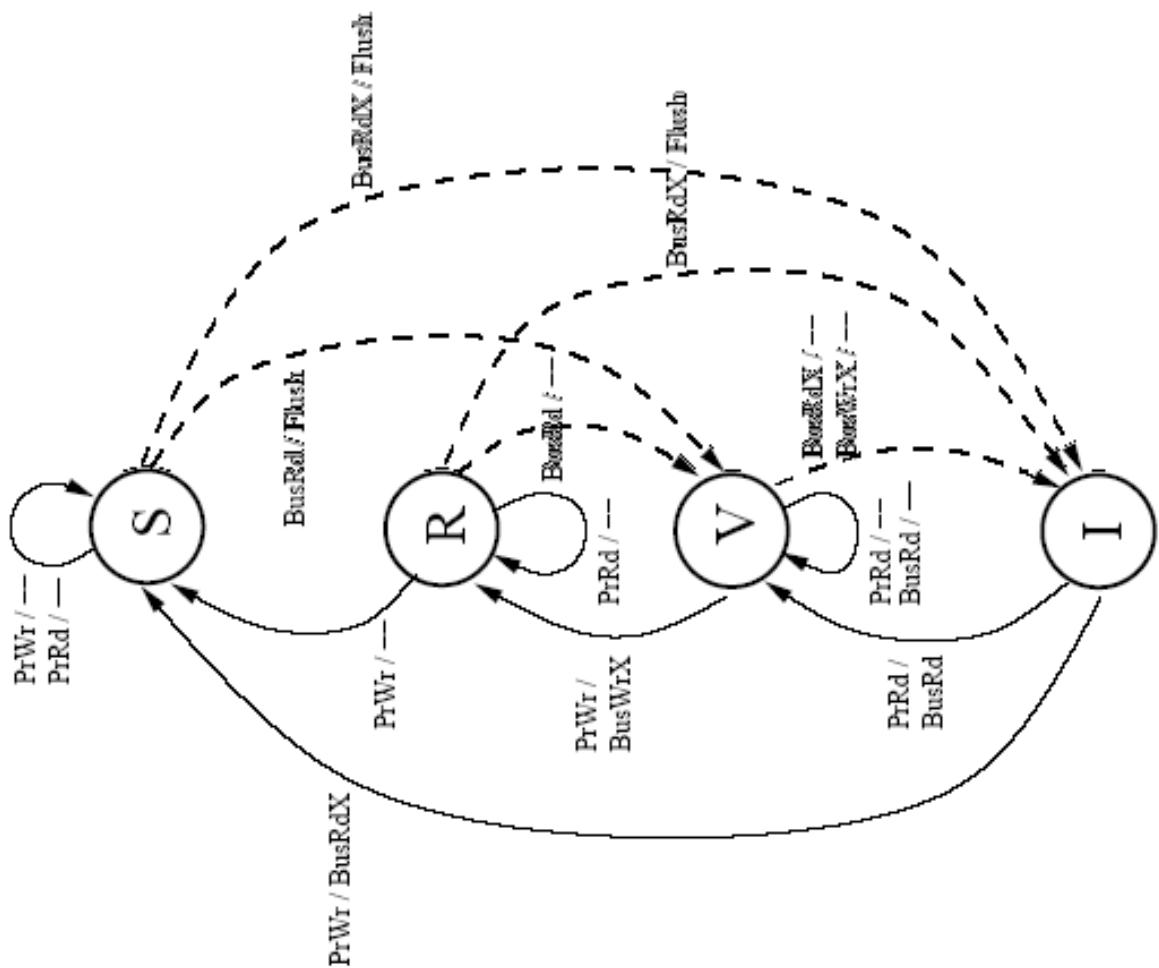
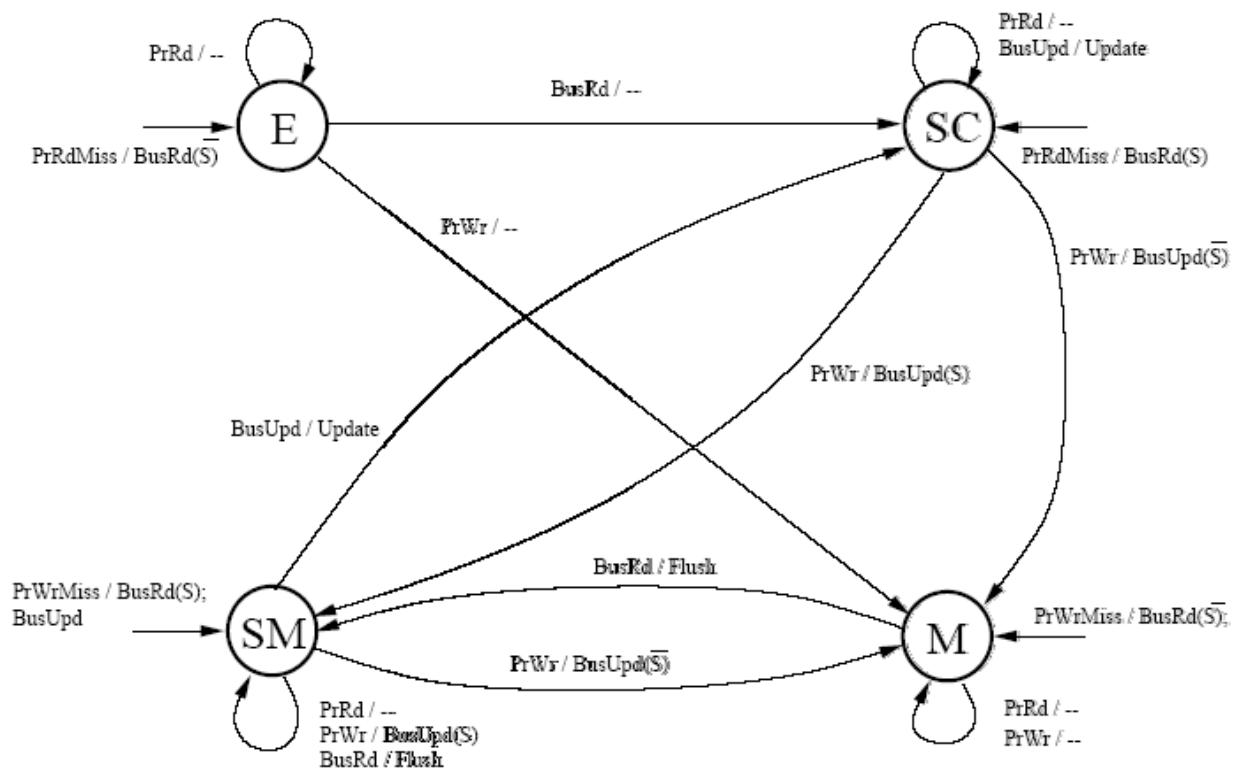
1. Cualquier lectura de un dato devuelve el valor más reciente escrito de ese dato.
2. El resultado de cualquier ejecución de un programa es tal que, para cada localización es posible construir una ordenación secuencial de las operaciones realizadas en el cual:
 - o Las operaciones emitidas ocurren en la secuencia indicada y en el orden en el que dicho procesador las emite.
 - o El valor devuelto por cada operación de lectura es el valor escrito por la última escritura.

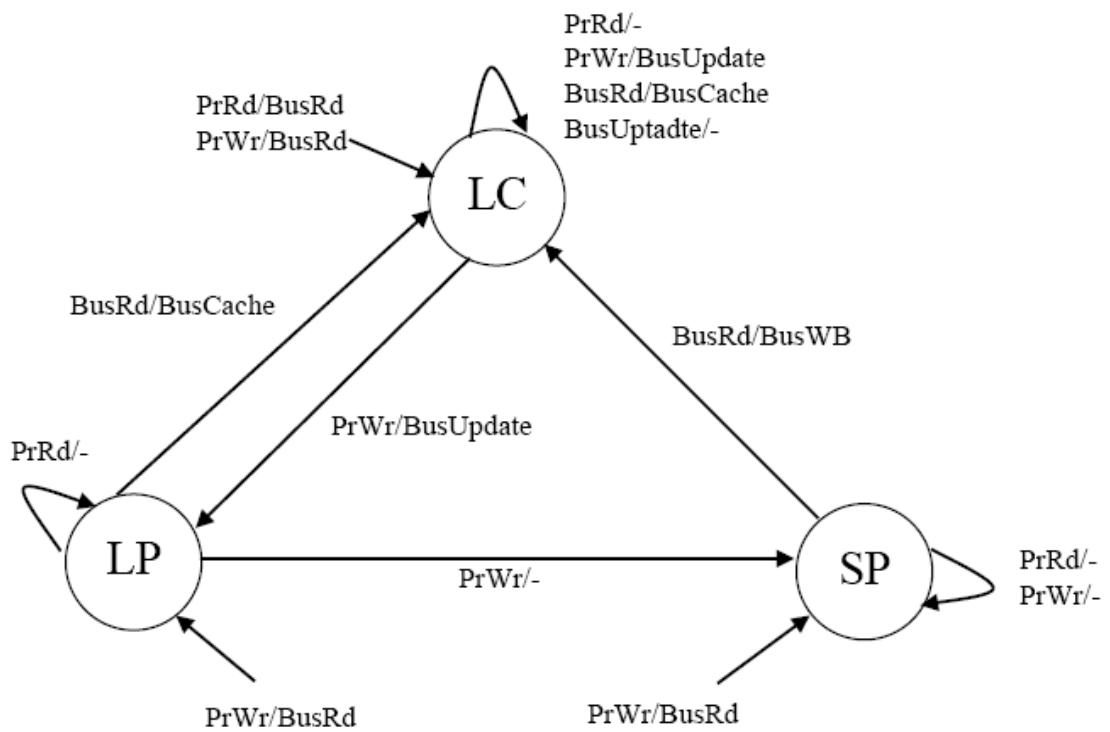
COHERENCIA DE MEMORIA

DIAGRAMAS

Sólo diagramas, ya que en casi todos los exámenes piden o corregir o dibujar: (A continuación en grande).







CONCLUSIÓN

Mierda de computadores. Quien cojones me mandaría a mi meterme en la carrera.

Ingeniería de los Computadores

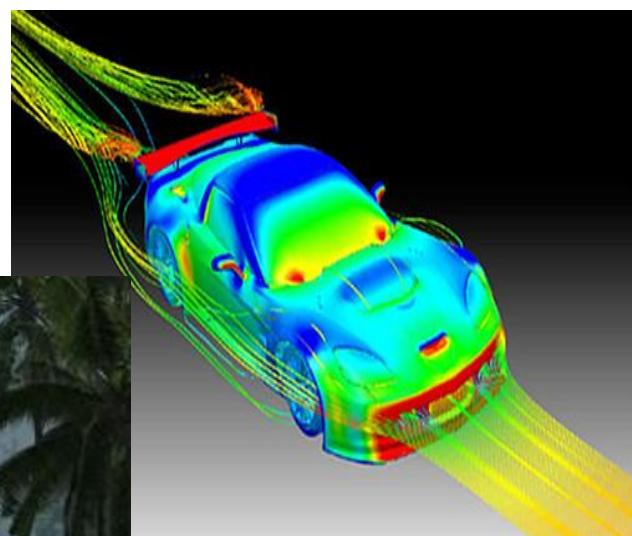
Sesión 1. Introducción

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

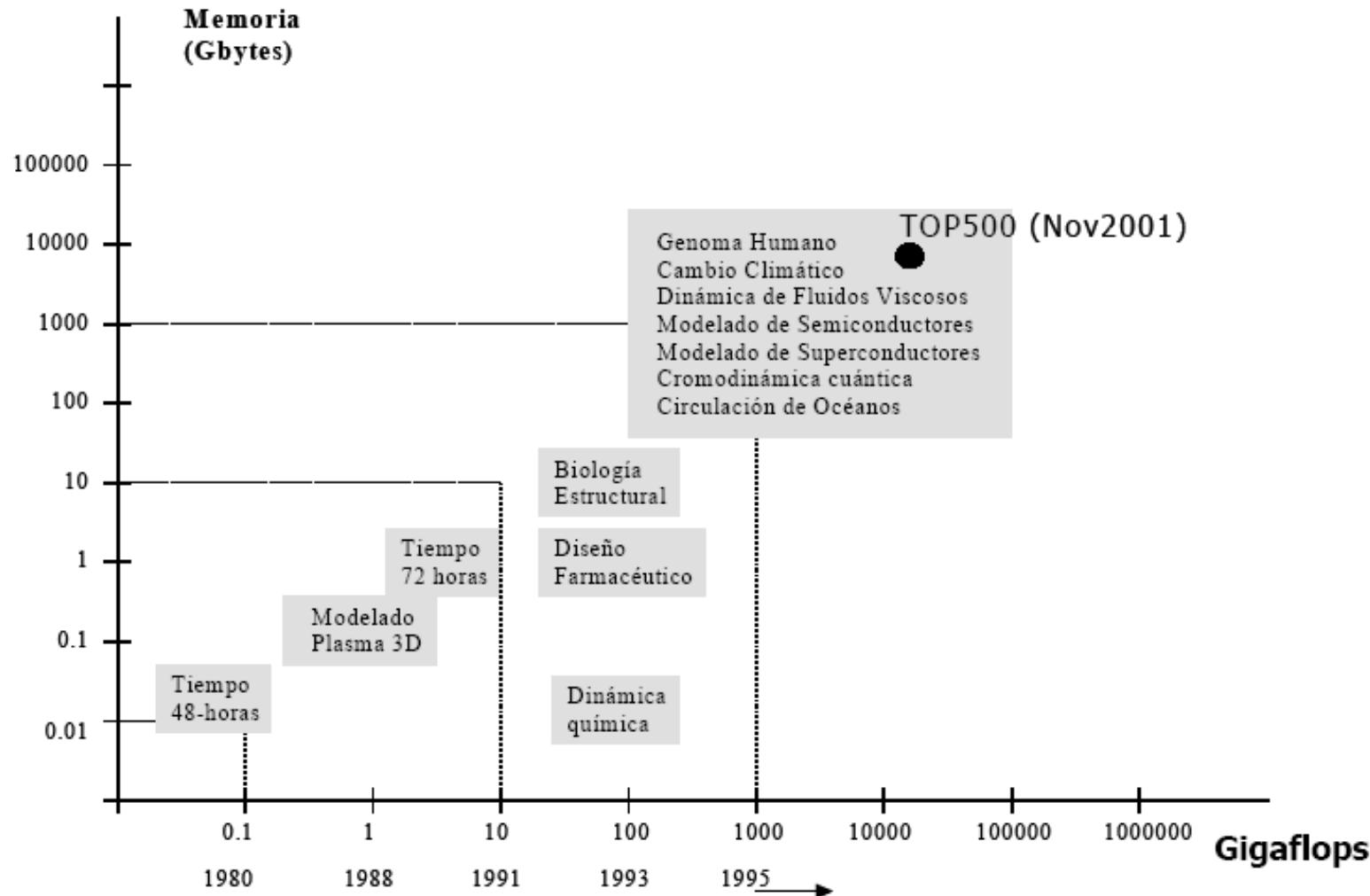
- Evolución de la informática  Necesidades computacionales



Ingeniería de los Computadores

Sesión 1. Introducción

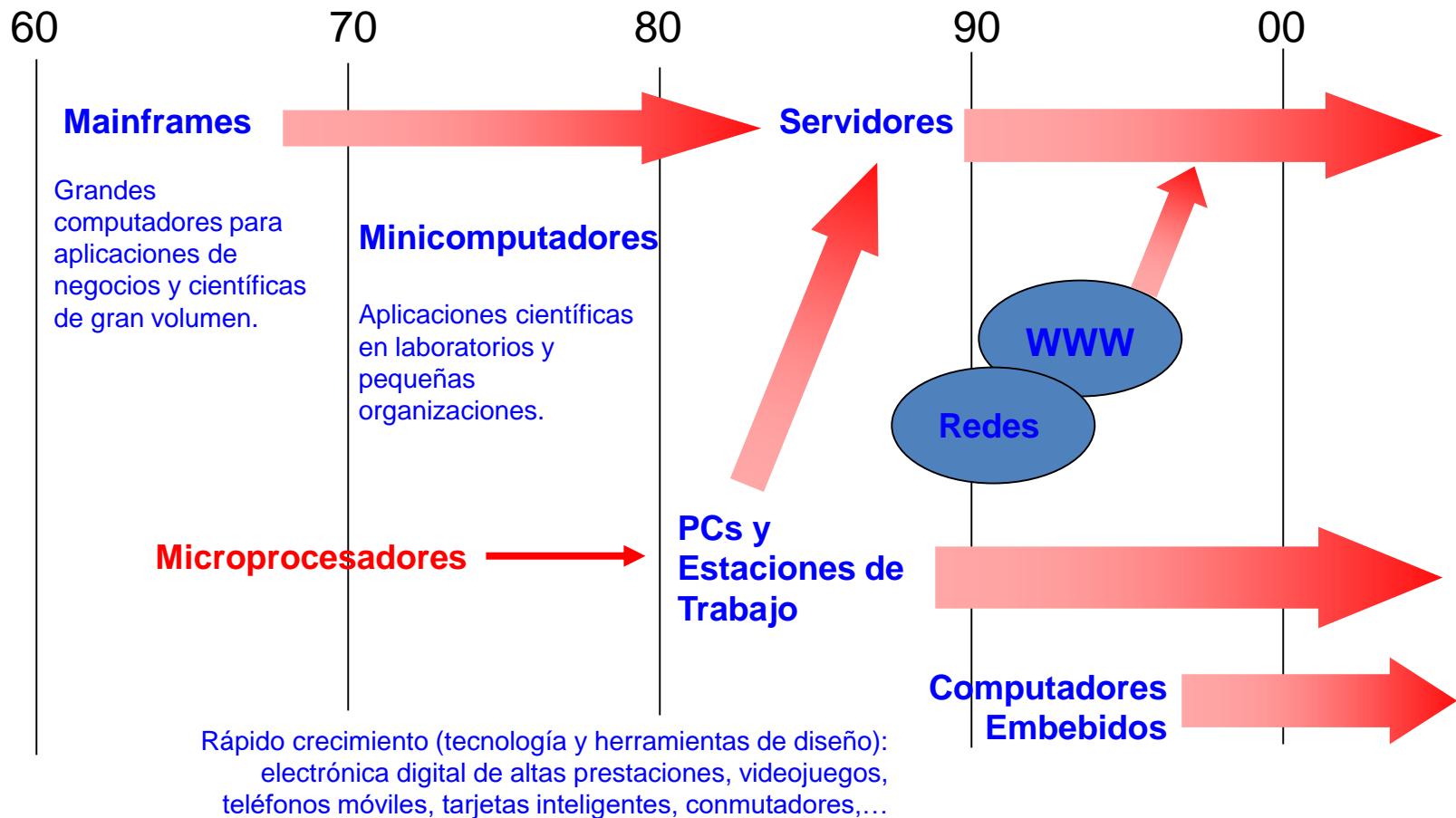
¿Qué?



Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

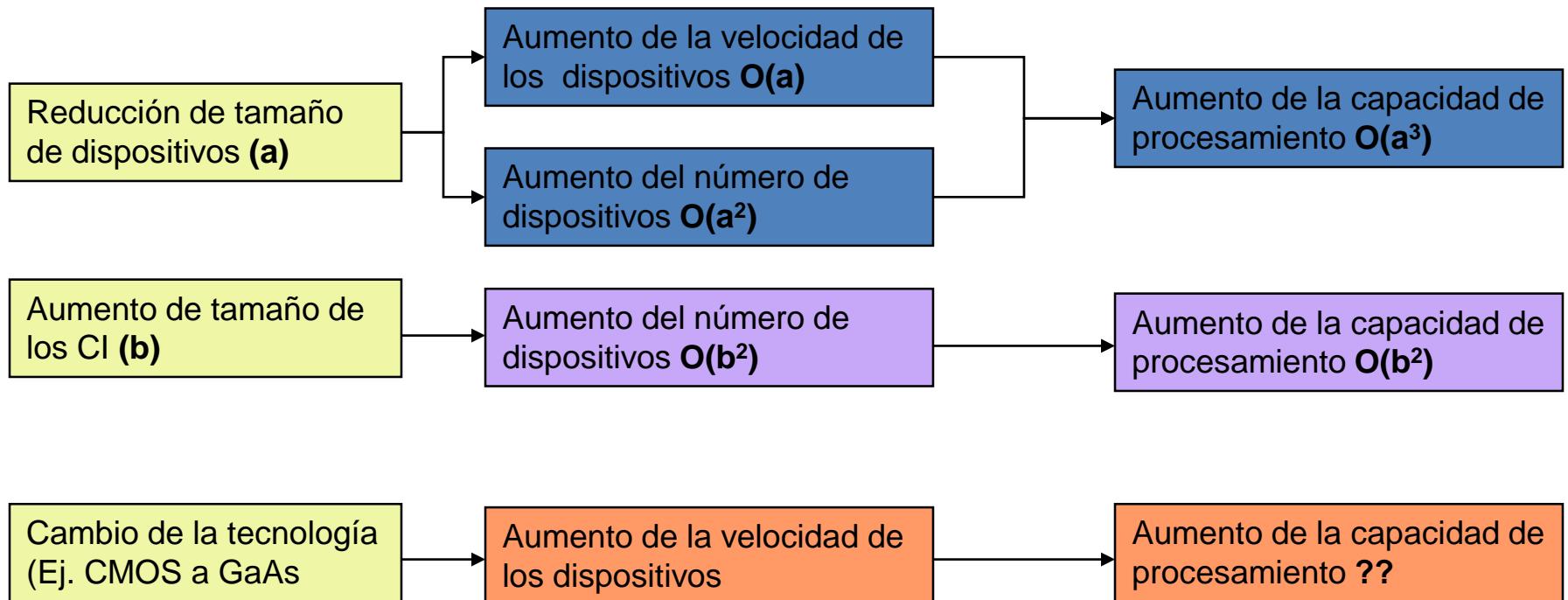


Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Mejora de prestaciones
 - Avances en tecnologías (límites físicos: calor, ruido, etc.)



Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Mejora de prestaciones
 - Avances en arquitecturas
 - **Paralelismo:**
 - Segmentación de cauces.
 - Repetición de elementos: Utilizar varias unidades funcionales, procesadores, módulos de memoria, etc. para distribuir el trabajo.
 - **Localidad:** Acercar datos e instrucciones al lugar donde se necesitan para que el acceso a los mismos sea lo más rápido posible (jerarquía de memoria).

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Desarrollo de las arquitecturas de computadores - Objetivo

MAYOR CAPACIDAD COMPUTACIONAL

Iniciativas mayor peso software

- Repertorio de instrucciones (RISC, CISC, ...)
- Arquitecturas VLIW
- Extensiones SIMD (MMX, SSE, 3DNOW, ...)

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Desarrollo de las arquitecturas de computadores - Objetivo

MAYOR CAPACIDAD COMPUTACIONAL

Iniciativas mayor peso hardware

- Arquitecturas segmentadas
- Arquitecturas vectoriales
- Arquitecturas superescalares
- Arquitecturas paralelas o de alto rendimiento

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Arquitectura de Computadores

“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]

- En Ingeniería de Computadores veremos:

- Arquitecturas superescalares
- Arquitecturas paralelas: multicomputadores y multiprocesadores

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Ámbito de la arquitectura de computadores
 - El lenguaje máquina del computador, la microarquitectura del procesador y la interfaz para los programas en lenguaje máquina (lenguaje máquina y arquitectura concreta del procesador).
 - Los elementos del computador y como interactúan (es decir la arquitectura concreta del computador, la estructura y organización).
 - La interfaz que se ofrece a los programas de alto nivel y los módulos que permiten controlar el funcionamiento del computador (sistema operativo y la arquitectura abstracta del computador).
 - Los procedimientos cuantitativos para evaluar los sistemas (benchmarking).
 - Las alternativas posibles y las tendencias en su evolución

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Niveles estructurales de Bell y Newell
 - 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)

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Niveles de interpretación de Levy
 - 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

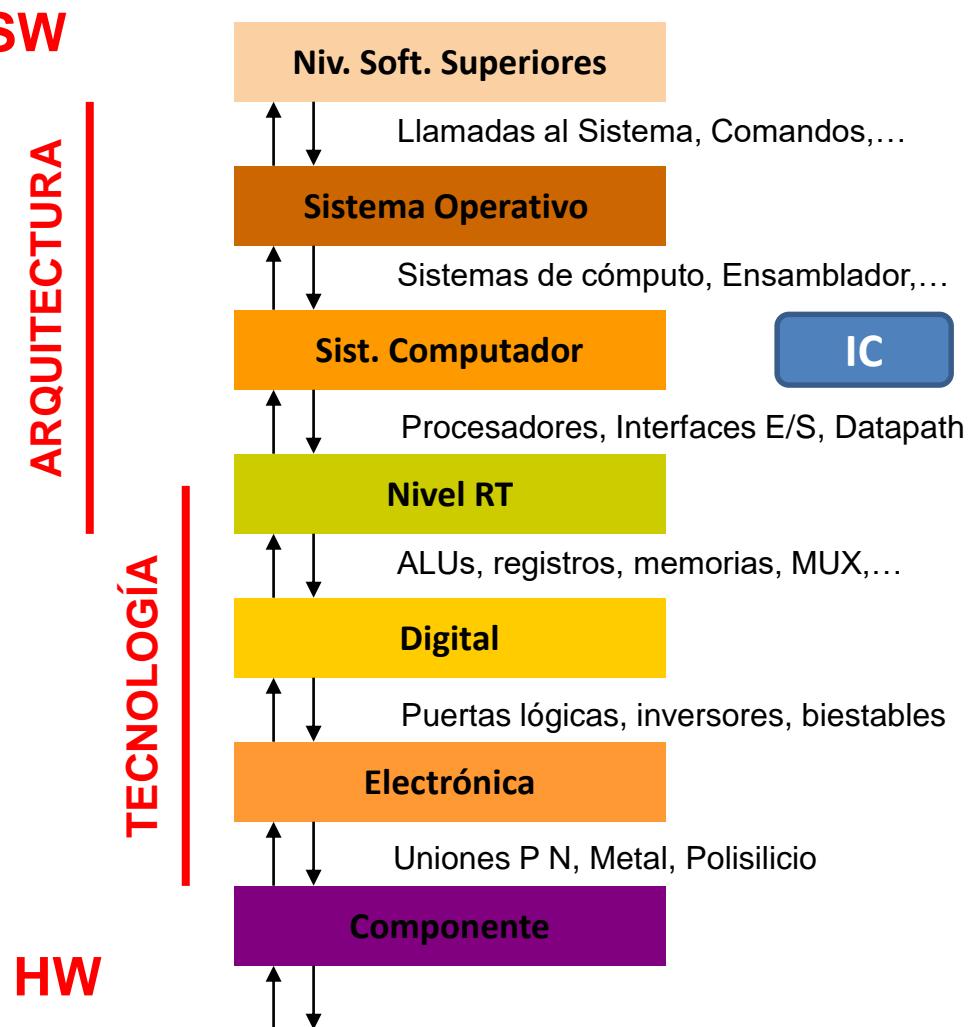
Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Niveles de abstracción de 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.



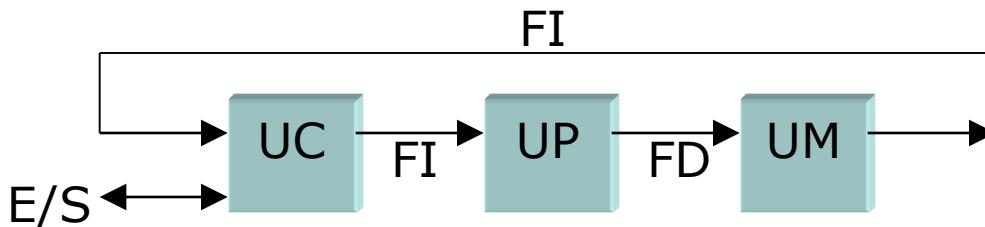
Ingeniería de los Computadores

Sesión 1. Introducción

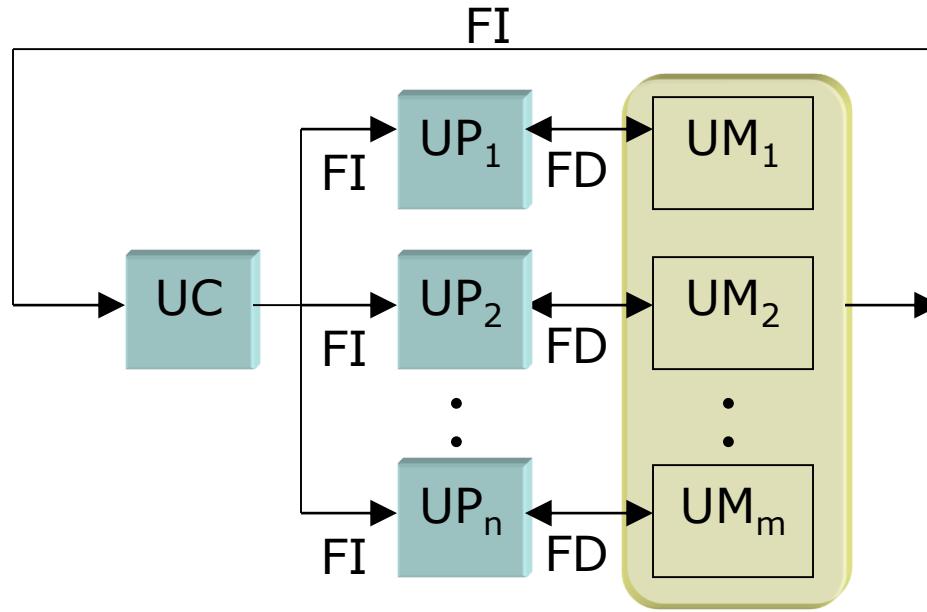
¿Qué?

- Taxonomía de Flynn

➤ SISD



➤ SIMD

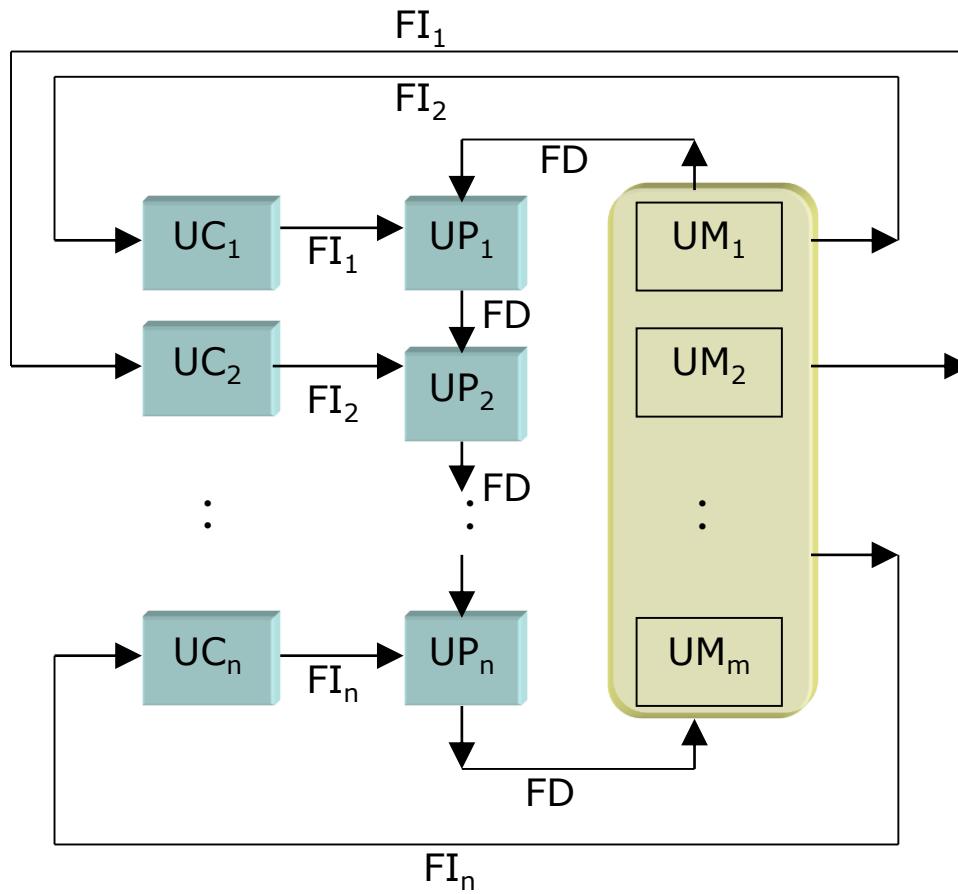


Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Taxonomía de Flynn
- MISD

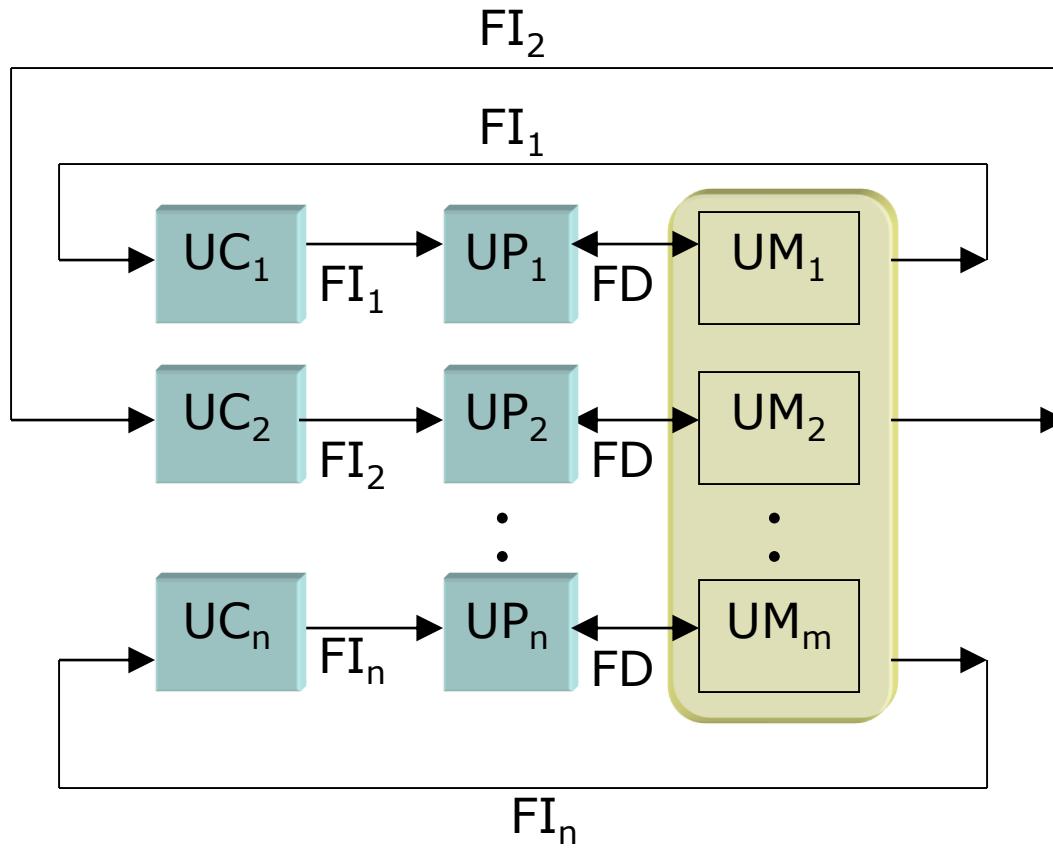


Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Taxonomía de Flynn
- MIMD

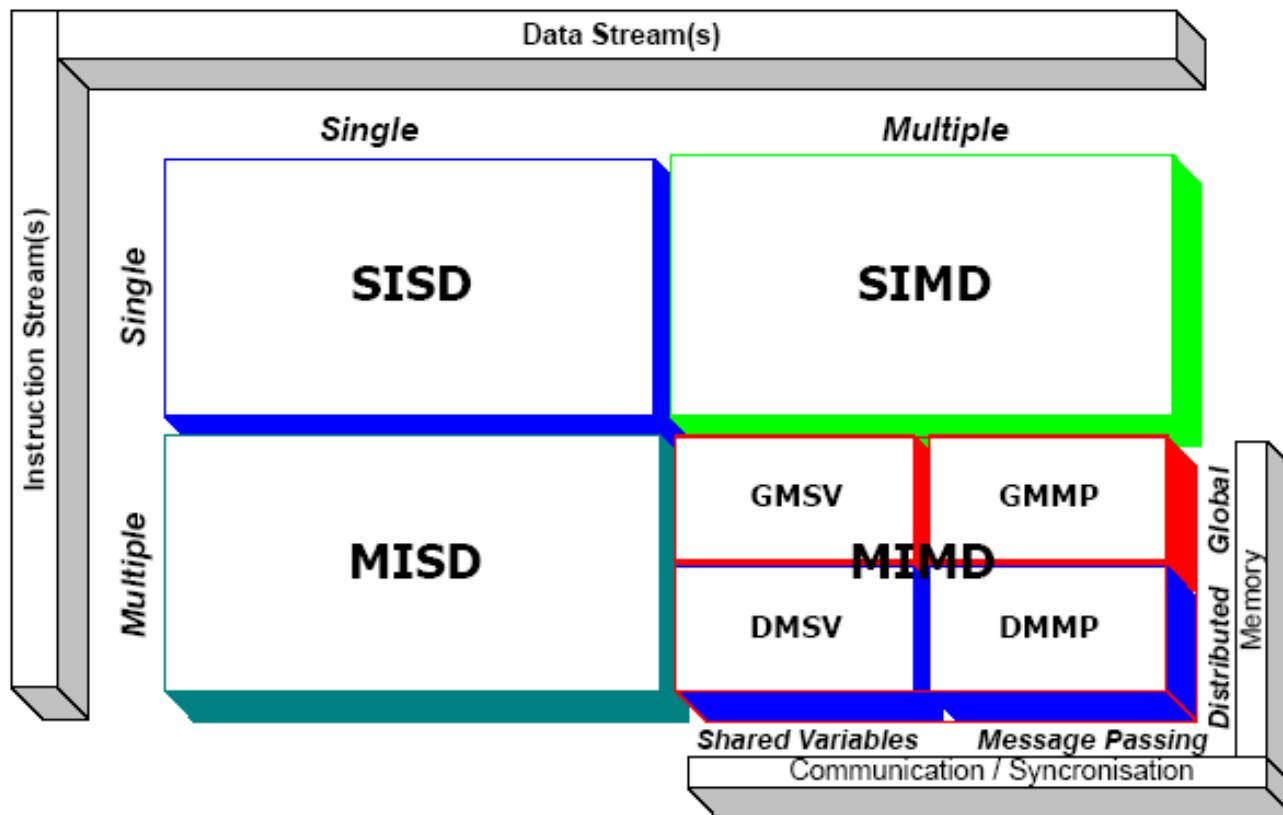


Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Taxonomía de Flynn-Johnson



Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

- Tipos de paralelismo
 - **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.
 - **Paralelismo funcional:** Varias funciones, tareas, instrucciones, etc. (iguales o distintas) 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) –distintos procedimientos que constituyen un programa se ejecutan simultáneamente. Grano medio.
 - Nivel de programa – la plataforma ejecuta en paralelo programas diferentes que pueden corresponder, o no, a una misma aplicación. Granularidad gruesa.

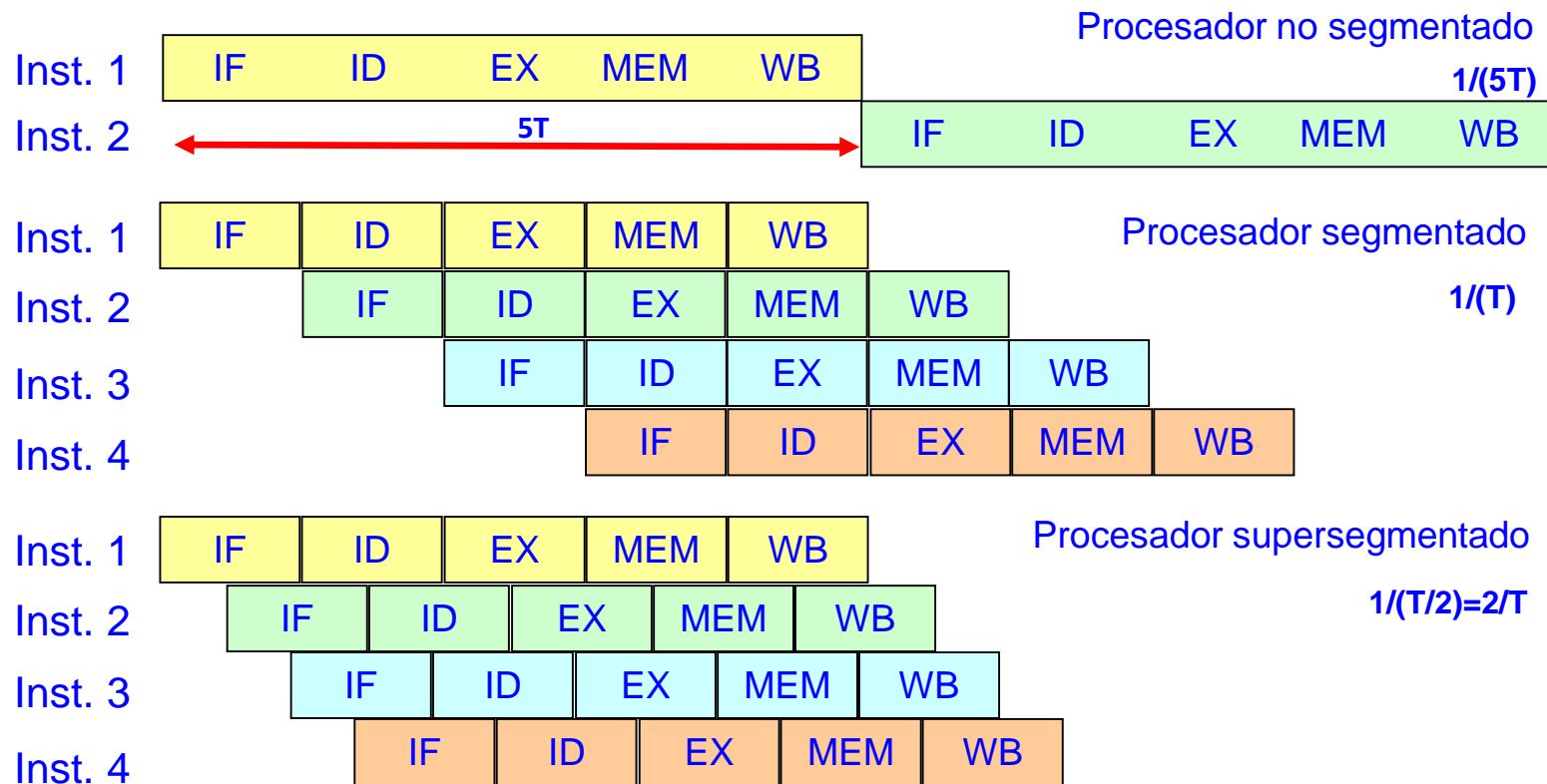
Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

- Segmentación: ILP

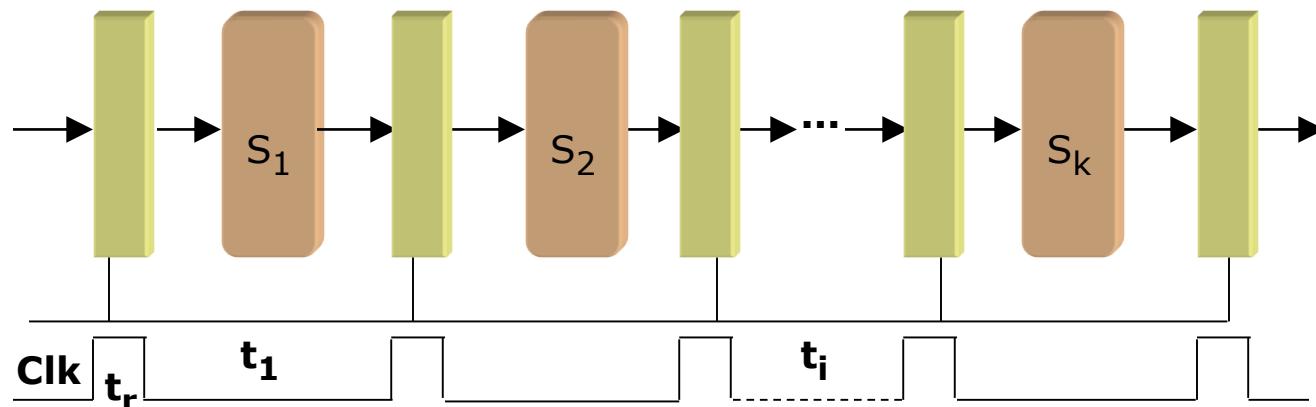


¿Qué?

Segmentación

- Segmentación

- Identificación de fases en el procesamiento de una tarea.
- Rediseño para implementar cada fase de forma independiente al resto.
- Paralelismo por etapas (el sistema procesa varias tareas al mismo tiempo aunque sea en etapas distintas).
- Se aumenta el número de tareas que se completan por unidad de tiempo.



Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

- Segmentación

Ganancia. Suponemos que TLI (tiempo de latencia de inicio) = T, tiempo que tarda en ejecutarse una operación en una unidad sin segmentar.

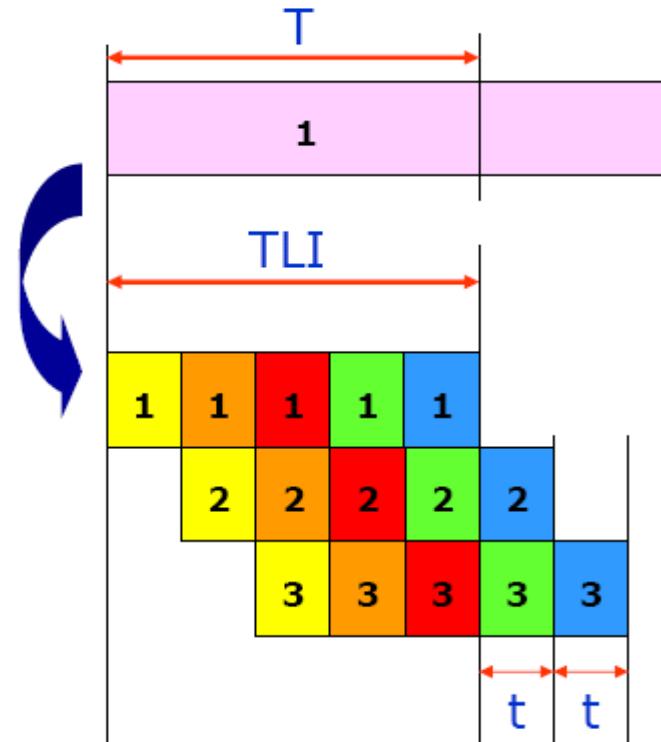
TLI = $k \cdot t$, siendo **k** el nº de etapas del cauce, y **t** la duración de cada etapa

$$G_k = \frac{T_1}{T_k} = \frac{k \cdot n \cdot t}{k \cdot t + (n-1) \cdot t} =$$

$$= \frac{k \cdot n}{k + n - 1}$$

$$\lim_{n \rightarrow \infty} G_k = k$$

Normalmente, ¿ $TLI > T$ ó $TLI < T$?



Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

- Ganancia real

$$G_k = \frac{T_{\text{sin_segmentar}}}{T_{\text{segmentado}}} = \frac{n \times T}{TLI + (n - 1) \times t}$$

$$G_{\max} = \lim_{n \rightarrow \infty} \frac{n \times T}{(k \times t) + (n - 1) \times t} = \frac{T}{t} < k$$

TLI = kt

T < TLI = kt

- Tipos de riesgos (detección del cauce)
 - **Riesgos de datos.** Se producen por dependencias entre operandos y resultados de instrucciones distintas.
 - **Riesgos de control.** Se originan a partir de instrucciones de salto condicional que, según su resultado, determinan la secuencia de instrucciones que hay que procesar tras ellas.
 - **Riesgos estructurales o colisiones.** Se producen cuando instrucciones diferentes necesitan el mismo recurso al mismo tiempo

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

- Riesgos de datos



RAW (Read After Write)

$$\begin{array}{l} R2 := R1 + R2 \\ \quad\quad\quad\circlearrowleft \\ R1 := R2 + R3 \\ \quad\quad\quad\circlearrowleft \end{array}$$



WAR (Write After Read)

$$\begin{array}{l} R2 := R1 + R2 \\ \quad\quad\quad\circlearrowleft \\ R1 := R2 + R3 \\ \quad\quad\quad\circlearrowleft \end{array}$$



WAWS (Write After Write)

$$\begin{array}{l} R2 := R1 + R2 \\ \quad\quad\quad\circlearrowleft \\ R2 := R4 + R3 \\ \quad\quad\quad\circlearrowleft \end{array}$$

Ingeniería de los Computadores

Sesión 1. Introducción

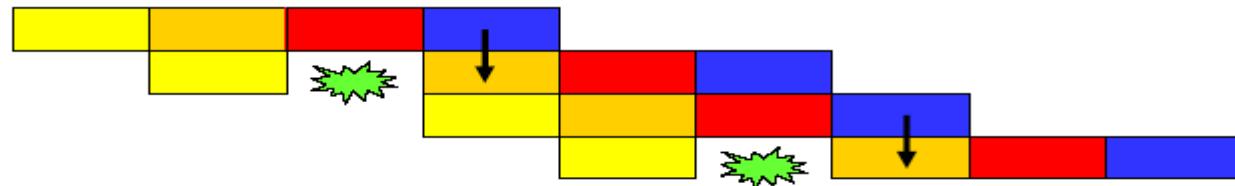
¿Qué?

Segmentación

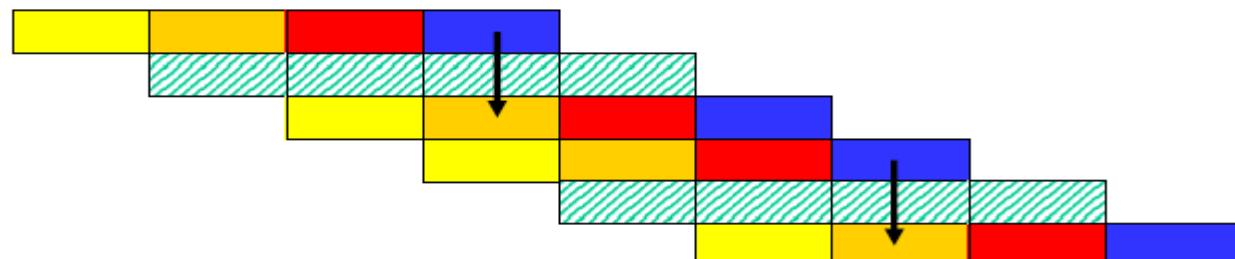
- Soluciones a los riesgos de datos

- Reorganización de código (intercambio de instrucciones e inserción de NOP)

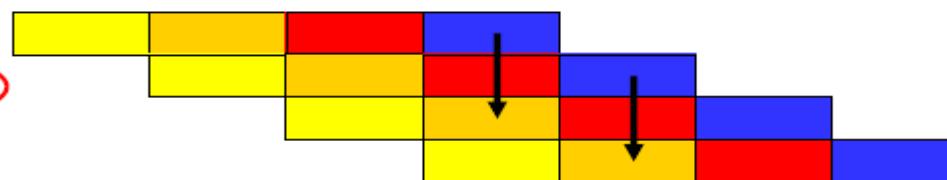
$$\begin{aligned} R3 &= R3 + R1 \\ R5 &= R5 - R3 \\ R4 &= R4 + R1 \\ R6 &= R6 - R4 \end{aligned}$$



$$\begin{aligned} R3 &= R3 + R1 \\ \textcolor{red}{nop} \\ R5 &= R5 - R3 \\ R4 &= R4 + R1 \\ \textcolor{red}{nop} \\ R6 &= R6 - R4 \end{aligned}$$



$$\begin{aligned} R3 &= R3 + R1 \\ \textcolor{red}{R4} &= R4 + R1 \\ \textcolor{red}{R5} &= R5 - R3 \\ R6 &= R6 - R4 \end{aligned}$$



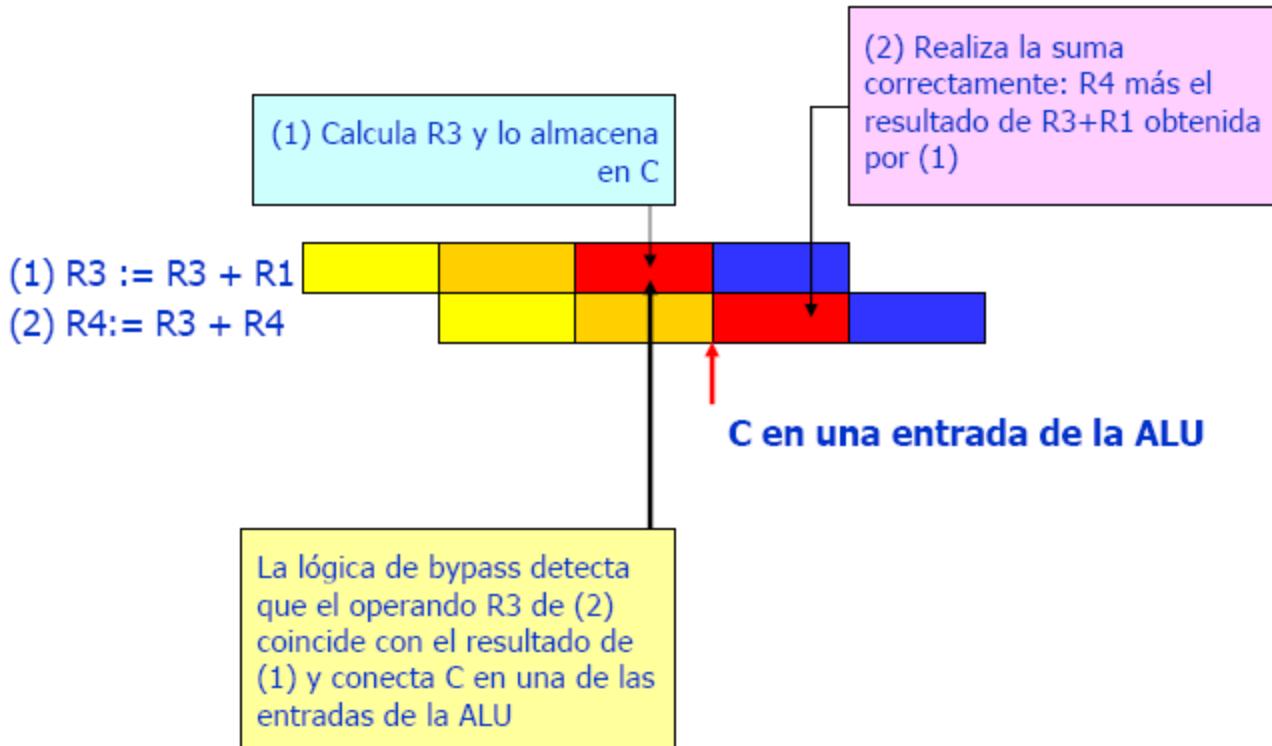
Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

- Soluciones a los riesgos de datos
 - Forwardings



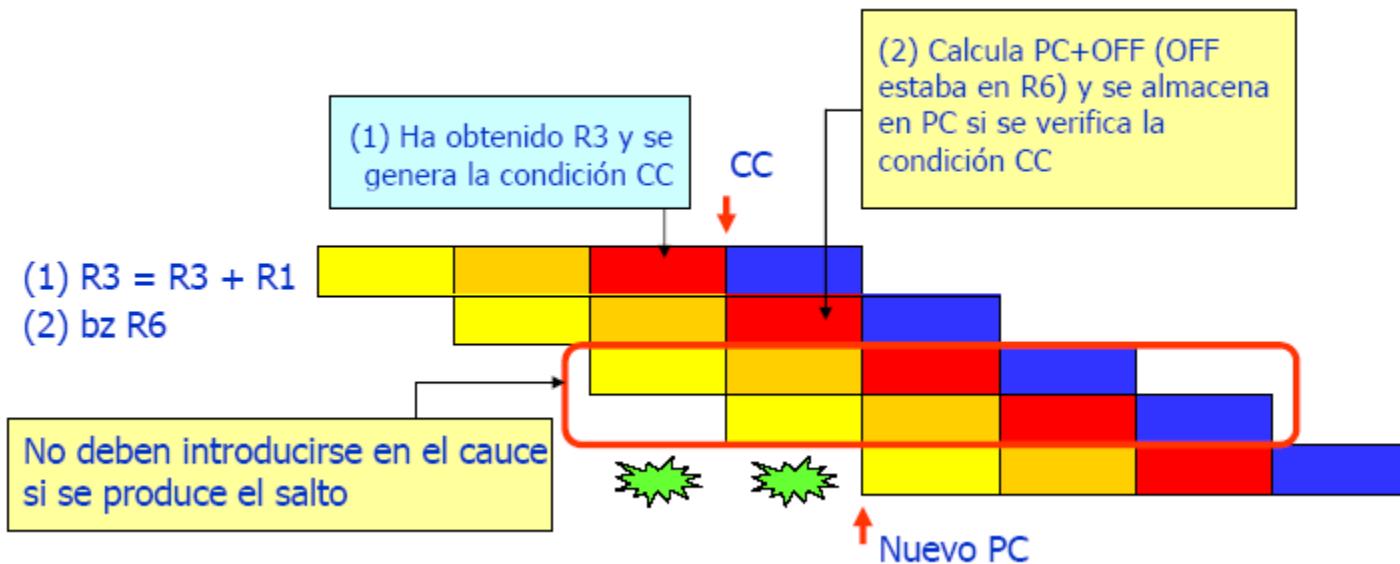
Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

- Soluciones a los riesgos de control
 - Abortar operaciones



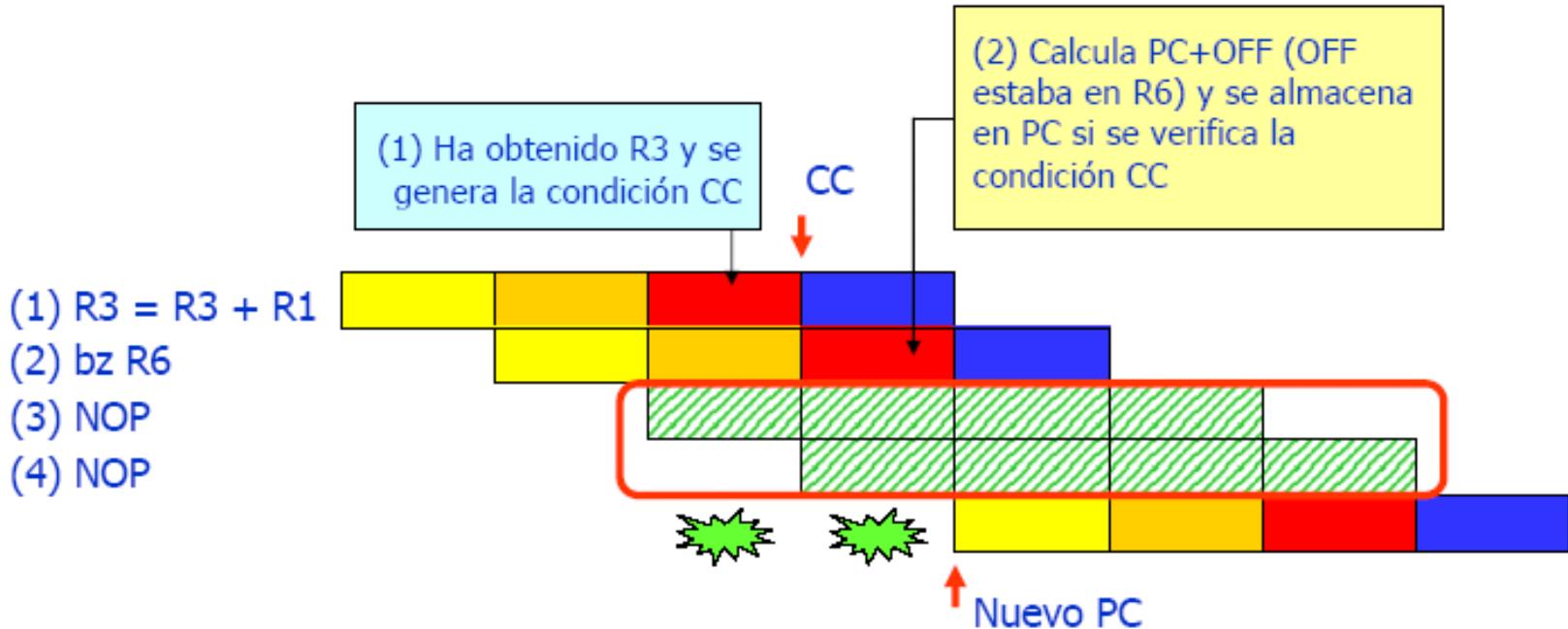
Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

- Soluciones a los riesgos de control
 - Bloqueos o uso de NOP



Ingeniería de los Computadores

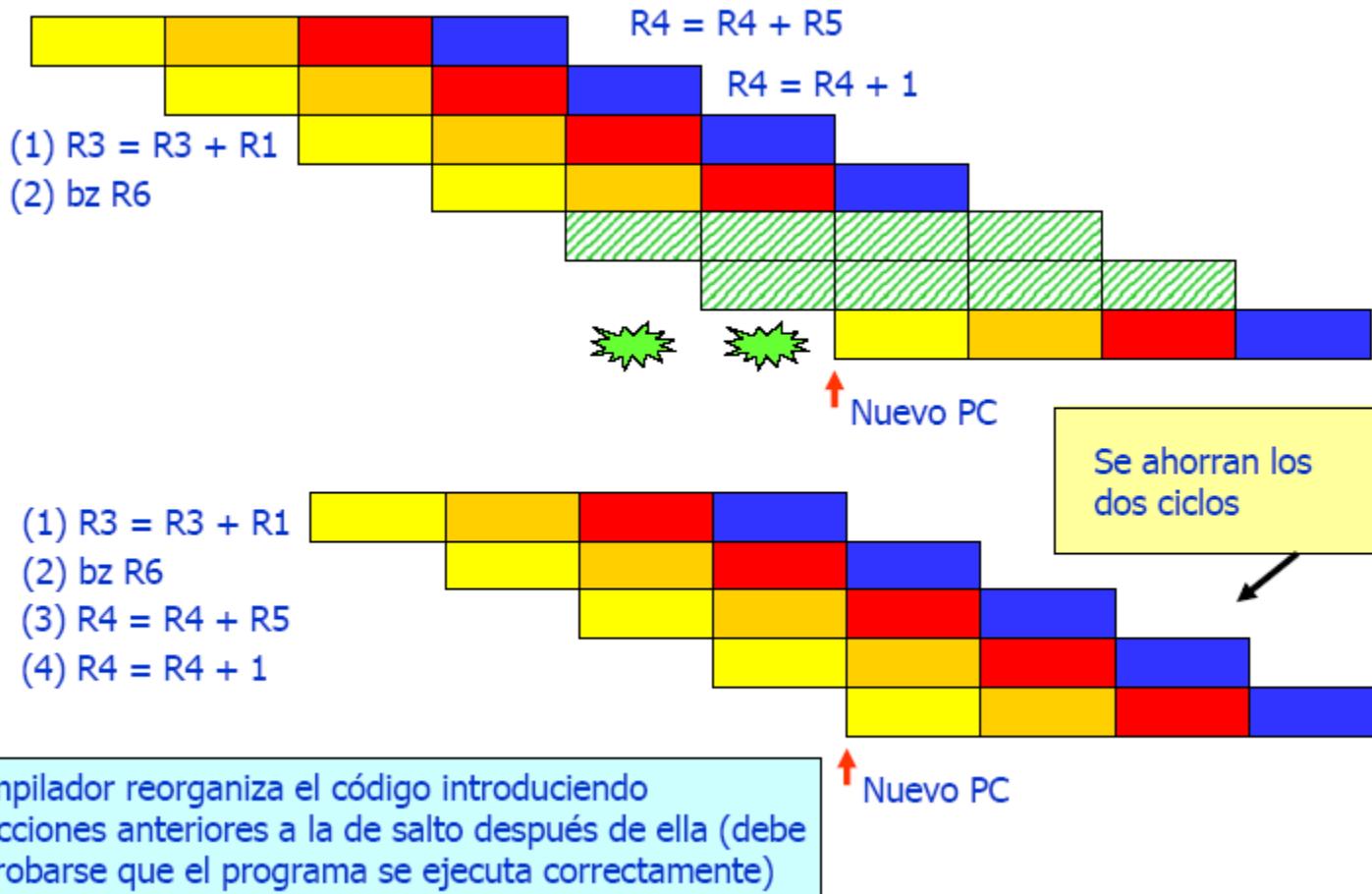
Sesión 1. Introducción

¿Qué?

Segmentación

- Soluciones a los riesgos de control

- Delayed branch



Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

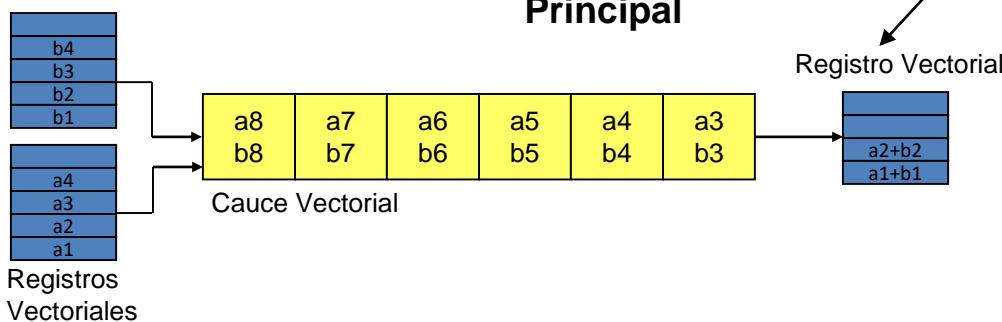
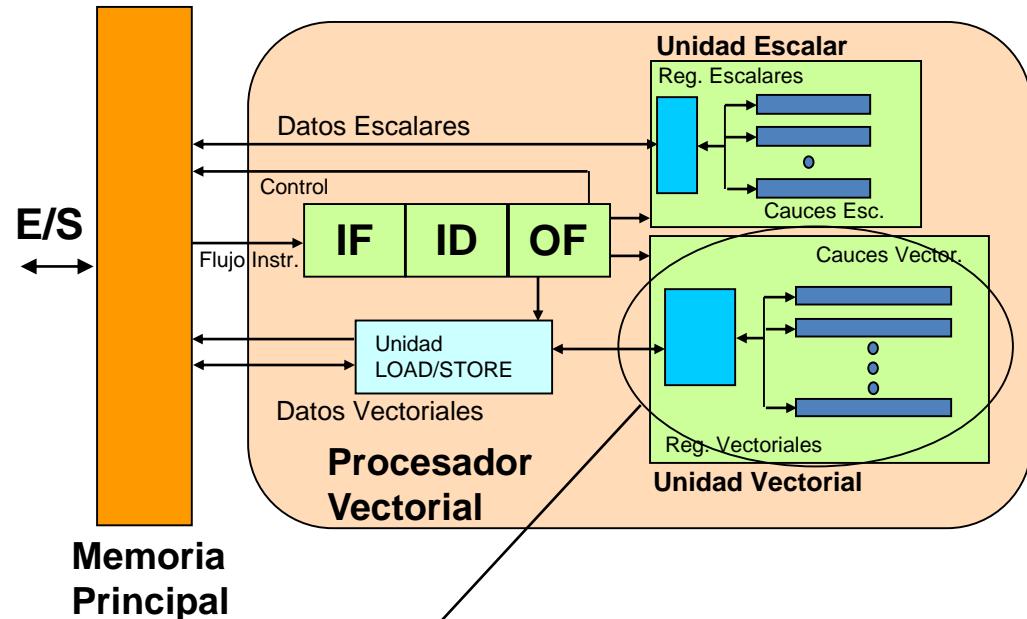
Segmentación

Vectoriales

- Arquitecturas vectoriales: ILP y paralelismo de datos

El procesamiento de instrucciones está segmentado y se utilizan múltiples unidades funcionales.

Paralelismo de datos: cada instrucción vectorial codifica una operación sobre todos los componentes del vector.



Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Arquitectura orientada al procesamiento de vectores (suma de vectores, productos escalares, etc.)
- Repertorio de instrucciones especializado
- Características
 - Cálculo de los componentes del vector de forma independiente (buenos rendimientos)
 - Cada operación vectorial codifica gran cantidad de cálculos (se reduce el número de instrucciones y se evitan riesgos de control)
 - Se optimiza el uso de memoria (entrelazado de memoria y organizaciones S y C)

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Ejemplo: Sumar dos vectores de 100 elementos.

Pseudo-código escalar

```
for i:= 1 to 100 do c(i)=b(i)+a(i)
```

Ensamblador escalar (con bucle de 100 iteraciones)

```
LOADI R5, BASEa  
LOADI R6, BASEb  
LOADI R7, BASEc  
LOADI R1, 0  
INI ADDRI R5, R5, 1  
ADDRI R6, R6, 1  
ADDRI R7, R7, 1  
ADDMR R8, R5, R6  
STORE R7, R8  
INC R1  
COMP R1, 100  
JUMP NOT.EQUALINI
```

Pseudo-código vectorial

$$c(1:100:1) = a(1:100:1) + b(1:100:1)$$

Ensamblador vectorial

ADDV c, a, b, 1, 100

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Vector instruction	Operands	Function
ADDV	V1, V2, V3	Add elements of V2 and V3, then put each result in V1.
ADDSV	V1, F0, V2	Add F0 to each element of V2, then put each result in V1.
SUBV	V1, V2, V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS	V1, V2, F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV	V1, F0, V2	Subtract elements of V2 from F0, then put each result in V1.
MULTV	V1, V2, V3	Multiply elements of V2 and V3, then put each result in V1.
MULTSV	V1, F0, V2	Multiply F0 by each element of V2, then put each result in V1.
DIVV	V1, V2, V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS	V1, V2, F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV	V1, F0, V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1, R1	Load vector register V1 from memory starting at address R1.
SV	R1, V1	Store vector register V1 into memory starting at address R1.
LVWS	V1, (R1, R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i*R2.
SVWS	(R1, R2), V1	Store V1 from address at R1 with stride in R2, i.e., R1+i*R2.
LVI	V1, (R1+V2)	Load V1 with vector whose elements are at R1+V2 (i), i.e., V2 is an index.
SVI	(R1+V2), V1	Store V1 with vector whose elements are at R1+V2 (i), i.e., V2 is an index.
CVI	V1, R1	Create an index vector by storing the values 0, 1*R1, 2*R1, ..., 63*R1 into V1.
S_V	V1, V2	Compare (EQ, NE, GT, LT, GE, LE) the elements in V1 and V2. If condition is true put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S_SV performs the same compare but using a scalar value as one operand.
S_SV	F0, V1	
POP	R1, VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MOVI2S	VLR, R1	Move contents of R1 to the vector-length register.
MOVSI2	R1, VLR	Move the contents of the vector-length register to R1.
MOVEF2S	VM, F0	Move contents of F0 to the vector-mask register.
MOVFS2F	F0, VM	Move contents of vector-mask register to F0.

Ingeniería de los Computadores

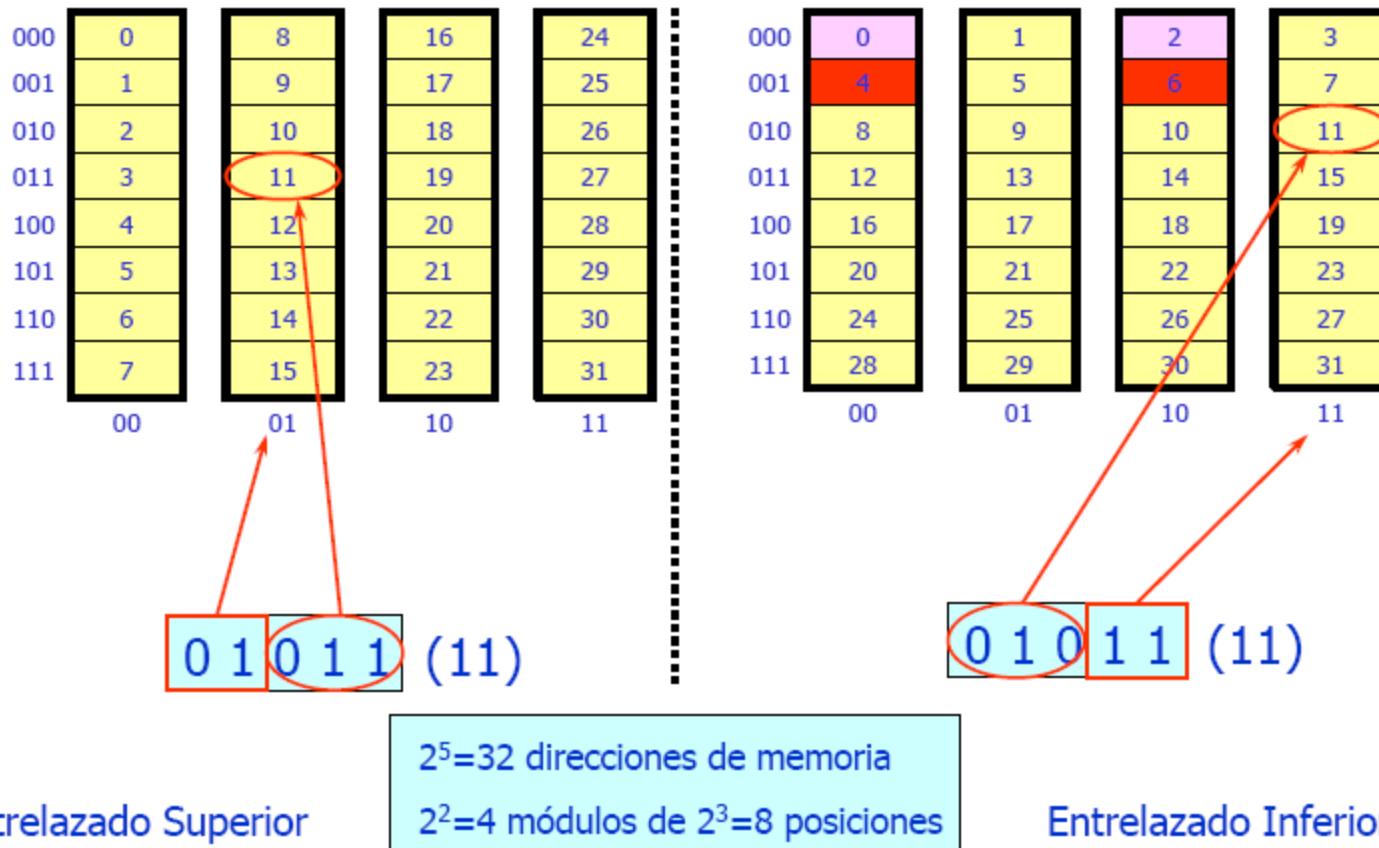
Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Entrelazado de memoria



Ingeniería de los Computadores

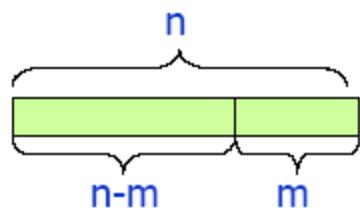
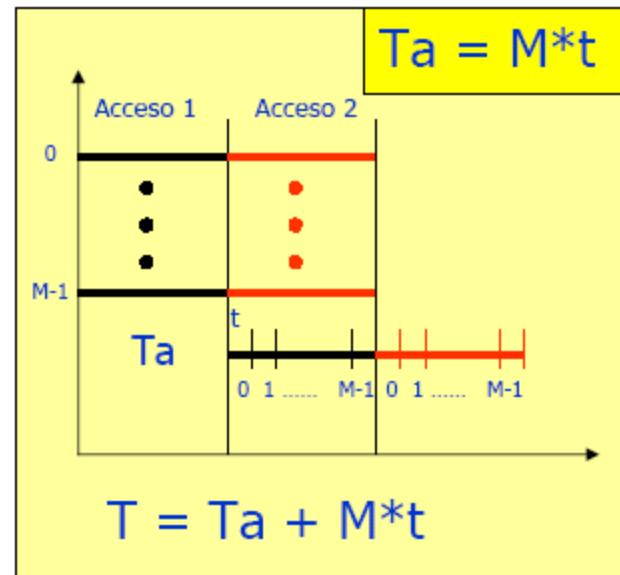
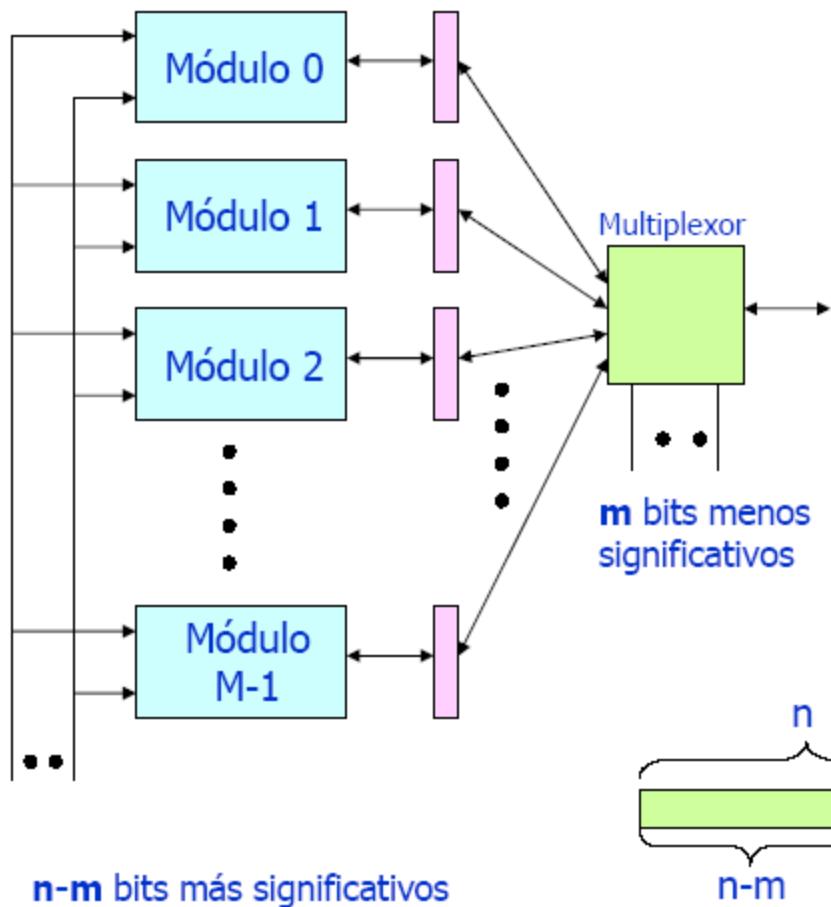
Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Acceso a memoria simultáneo o tipo S



Con Entrelazado Inferior

N=2ⁿ direcciones

M=2^m módulos

2^(n-m) direcciones/módulo

Ingeniería de los Computadores

Sesión 1. Introducción

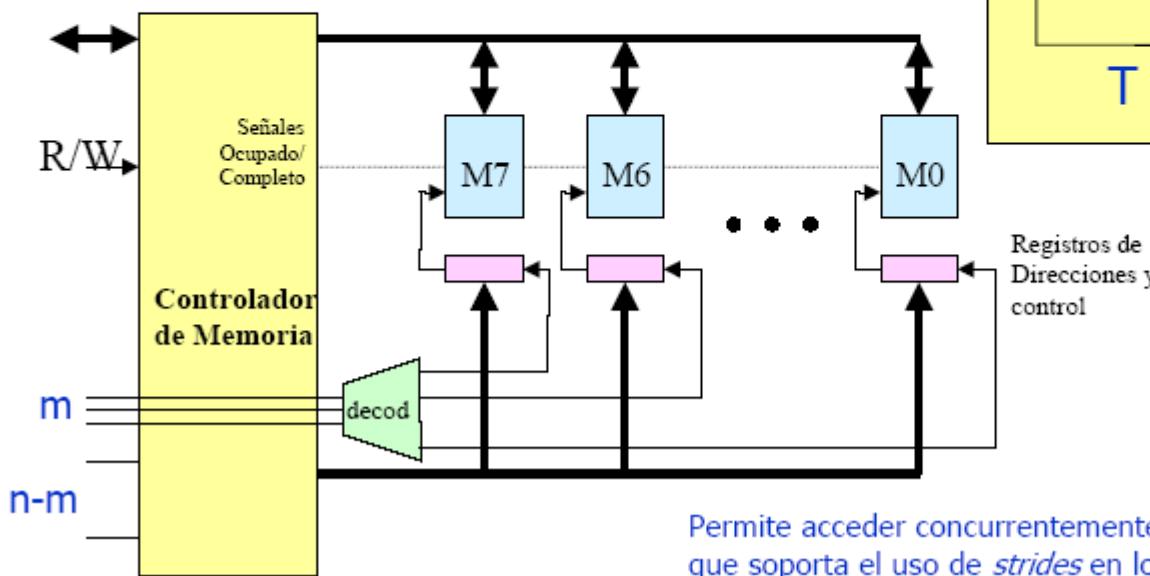
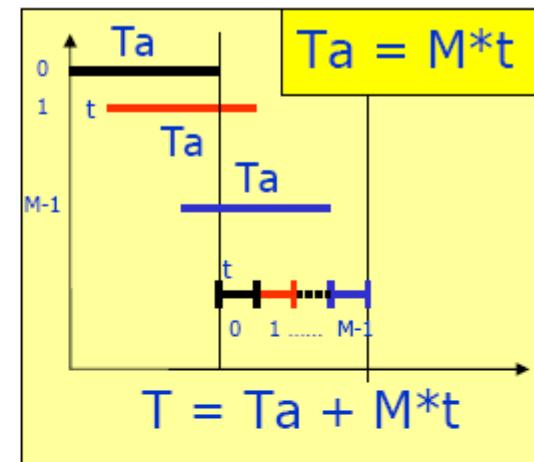
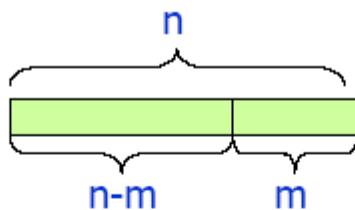
¿Qué?

Segmentación

Vectoriales

- Acceso a memoria concurrente o tipo C

Con Entrelazado Inferior
 $N = 2^n$ direcciones
 $M = 2^m$ módulos
 $2^{(n-m)}$ direcciones/módulo



Permite acceder concurrentemente a M direcciones, con lo que soporta el uso de *strides* en los accesos a memoria

Ingeniería de los Computadores

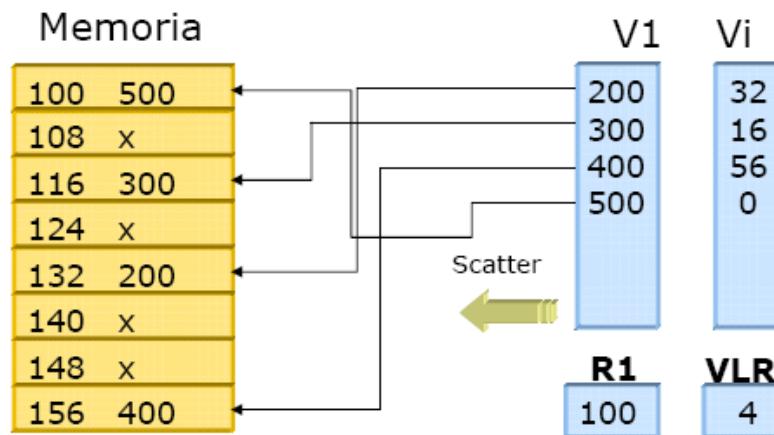
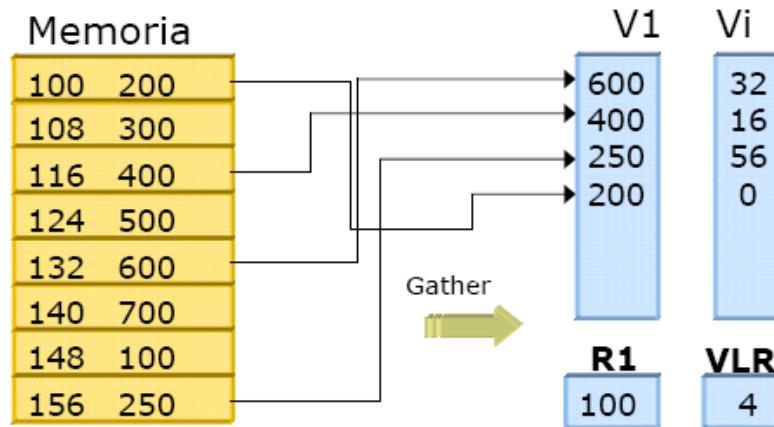
Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Operaciones gather-scatter



Ingeniería de los Computadores

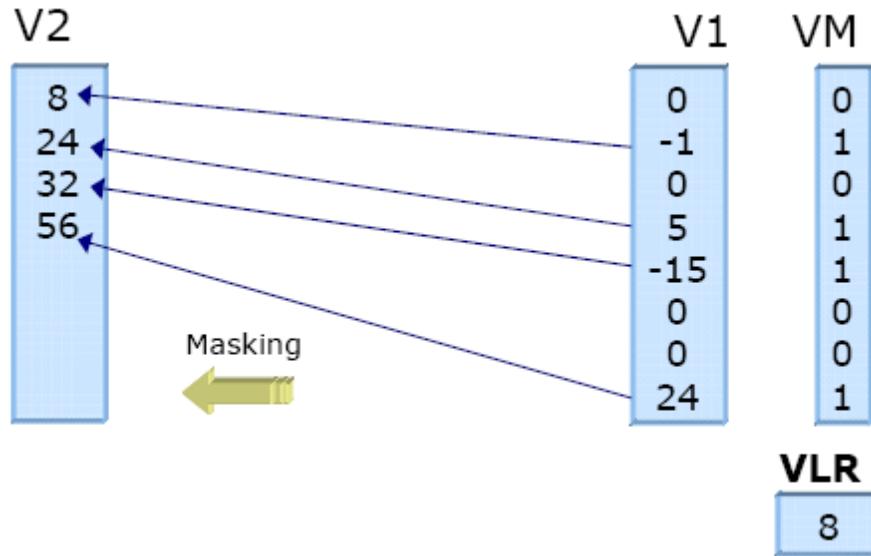
Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Enmascaramiento (gestión de matrices dispersas)



Ingeniería de los Computadores

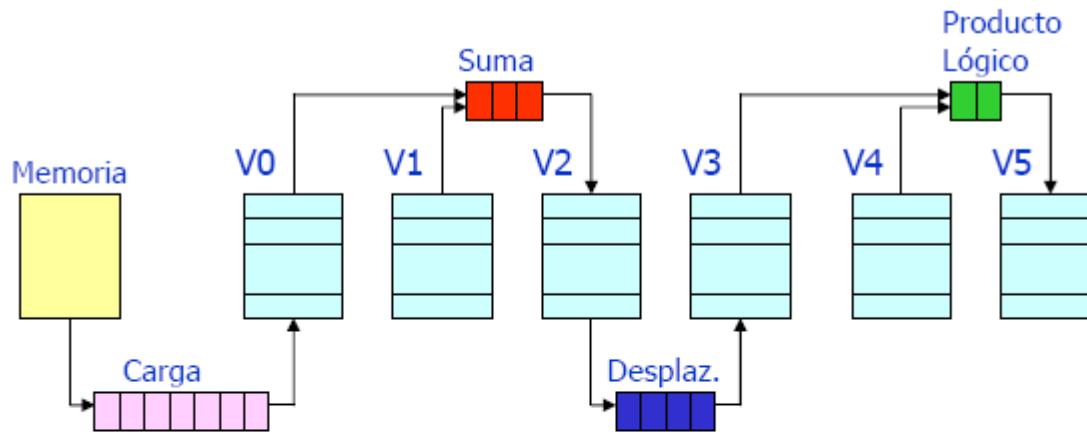
Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Rendimiento: encadenamiento de cauce



$V_0 = \text{Load}(\text{Memoria})$	(TLI=7)
$V_2 = V_0 + V_1$	(TLI=3)
$V_3 = V_1 < A_3$	(TLI=4)
$V_5 = V_3 \wedge V_4$	(TLI=2)

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

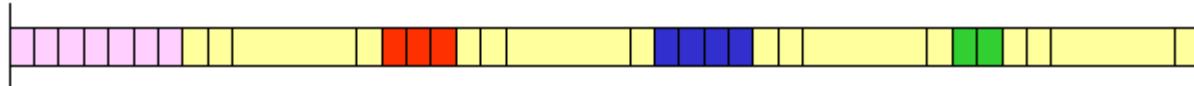
Vectoriales

- Rendimiento: encadenamiento de cauce

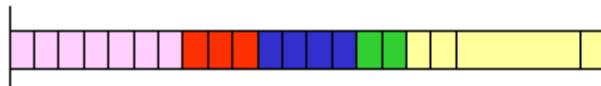


Si se espera que termine una operación vectorial para que empiece otra

$$TCV = TLI(\text{carga}) + TLI(\text{suma}) + TLI(\text{desplaz.}) + TLI(\text{Prod.Log}) + 4*K$$



$$TCV = TLI(\text{carga}) + TLI(\text{suma}) + TLI(\text{desplaz.}) + TLI(\text{Prod.Log}) + K$$



Si se encadenan los cauces de las distintas operaciones

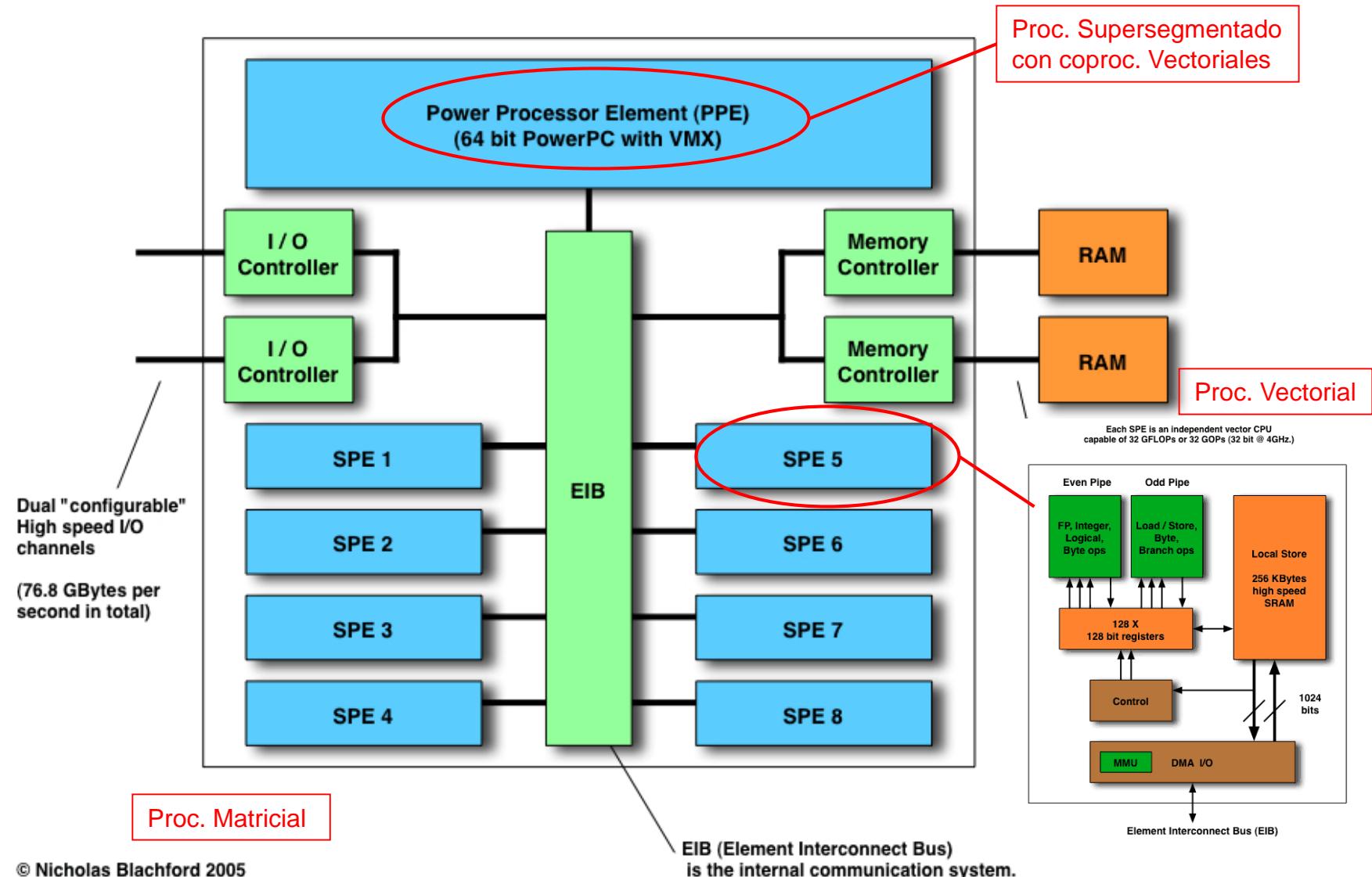
Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

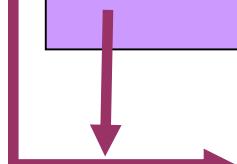
Vectoriales



- Tiempo de ejecución de un programa
 - Tiempo de CPU (usuario y sistema)
 - Tiempo de E/S (comunicaciones, acceso a memoria, visualización, etc.)

$$\text{Tiempo de CPU (T}_{\text{CPU}}\text{)} = \text{Ciclos_del_Programa} \times T_{\text{CICLO}} = \frac{\text{Ciclos_del_Programa}}{\text{Frecuencia_de_Reloj}}$$

$$\text{Ciclos por Instrucción (CPI)} = \frac{\text{Ciclos_del_Programa}}{\text{Número_de_Instrucciones (NI)}}$$



$$T_{\text{CPU}} = \text{NI} \times \text{CPI} \times T_{\text{CICLO}}$$

$$T_{\text{CICLO}} = 1/F$$

F=frecuencia

- Tiempo para arquitecturas capaces de emitir a ejecución varias instrucciones por unidad de tiempo

$$T_{CPU} = NI \times (CPE / IPE) \times T_{CICLO}$$

CPI

CPE = ciclos entre inicio de emisión de instrucciones.

IPE = instrucciones que pueden emitirse (empezar la ejecución) cada vez que se produce ésta.

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

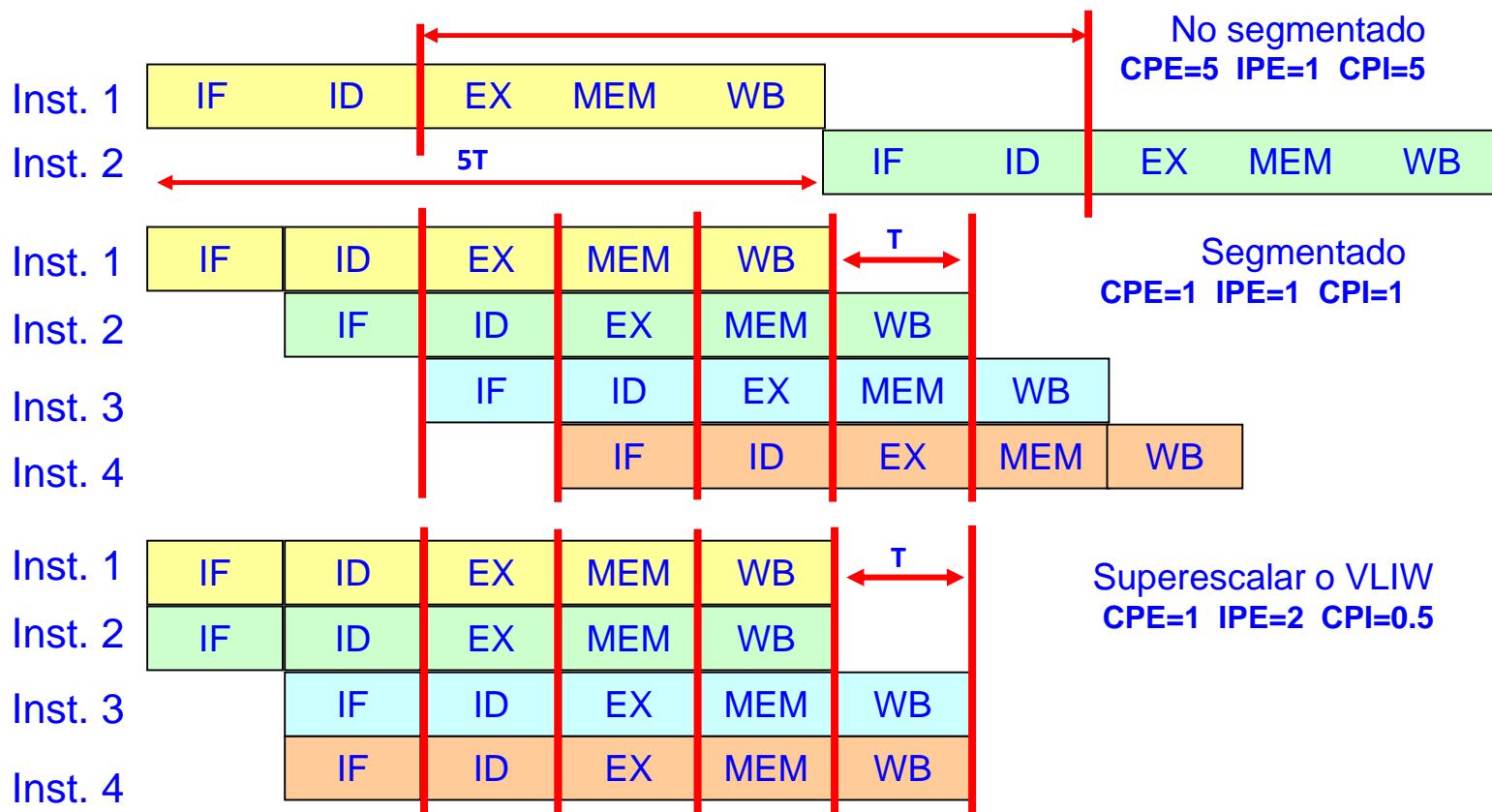
Segmentación

Vectoriales

Rendimiento

Ejemplo:

$$\text{CPI} = \text{CPE}/\text{IPE}$$



- Procesadores que codifican varias operaciones en una instrucción (VLIW)

$$T_{CPU} = (\underbrace{Noper / Op_instr}_{NI}) \times CPI \times T_{CICLO}$$

Noper = número de operaciones que realiza el programa.

Op_instr = número de operaciones que puede codificar una instrucción.

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

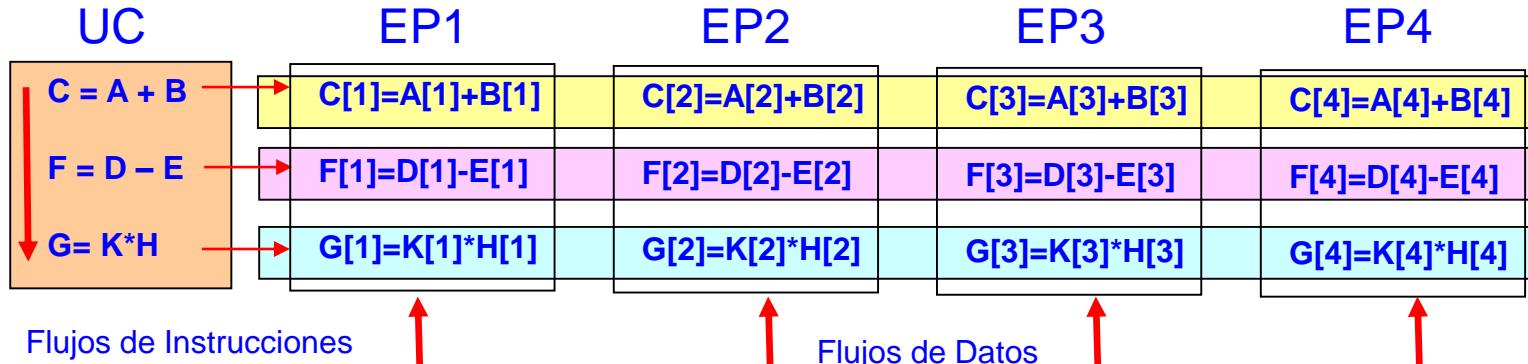
Rendimiento

Ejemplo:

$$NI = \text{Noper}/\text{Op_instr}$$

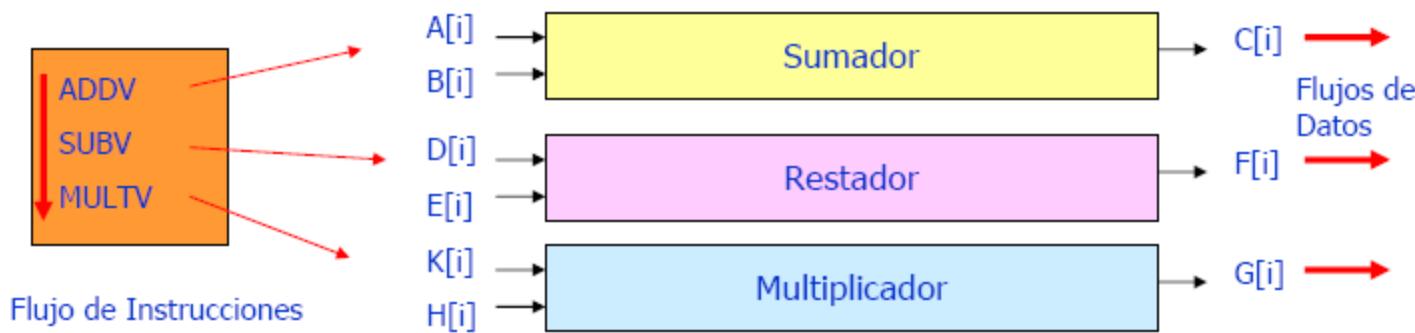
Ejemplo paralelismo datos

Procesador Matricial



Ejemplo paralelismo instrucciones

Procesador Vectorial



- Evaluación del rendimiento: benchmarks
- Tipos de benchmarks
 - Aplicaciones reales. Compiladores, Word, MatLab, ...
 - Kernels. Trozos de aplicaciones reales seleccionados para evaluar características específicas.
 - Simples (Toys). Pequeños programas fáciles de programar y cuyo resultado es conocido (Quicksort).
 - Sintéticos. Reproducen porcentajes de instrucciones y situaciones de carga reales.
- Suites. Conjuntos de benchmarks que miden las prestaciones de los computadores a través de un conjunto de aplicaciones distintas. Las limitaciones de un benchmark se suplen con la presencia de otros. Se cambian periódicamente para evitar optimizaciones realizadas con el único objetivo de mejorar los resultados del conjunto de benchmark.

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

- SPEC CPU2006
 - Eneros: <http://www.spec.org/cpu2006/CINT2006/>
 - Coma flotante: <http://www.spec.org/cpu2006/CFP2006/>
- Benchmarks para computadores de altas prestaciones
 - SPEC MPI2007: <http://www.spec.org/mpi2007/docs/index.html>
 - Linpack: <http://www.top500.org/project/linpack>

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

- Medidas de rendimiento:

➤ Ganancia

$$G_P = \frac{T_1}{T_P}; \quad G_P \leq P$$

➤ Eficiencia

$$E_P = \frac{G_p}{P}; \quad E_P \leq 1$$

➤ Productividad

Ingeniería de los Computadores

Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

Kilobyte (KB)	10^3 bytes	Kibibyte (KiB)	1024 bytes (2^{10} bytes)
Megabyte (MB)	10^6 bytes	Mebibyte (MiB)	2^{20} bytes
Gigabyte (GB)	10^9 bytes	Gibibyte (GiB)	2^{30} bytes
Terabyte (TB)	10^{12} bytes	Tebibyte (TiB)	2^{40} bytes

Ingeniería de los Computadores

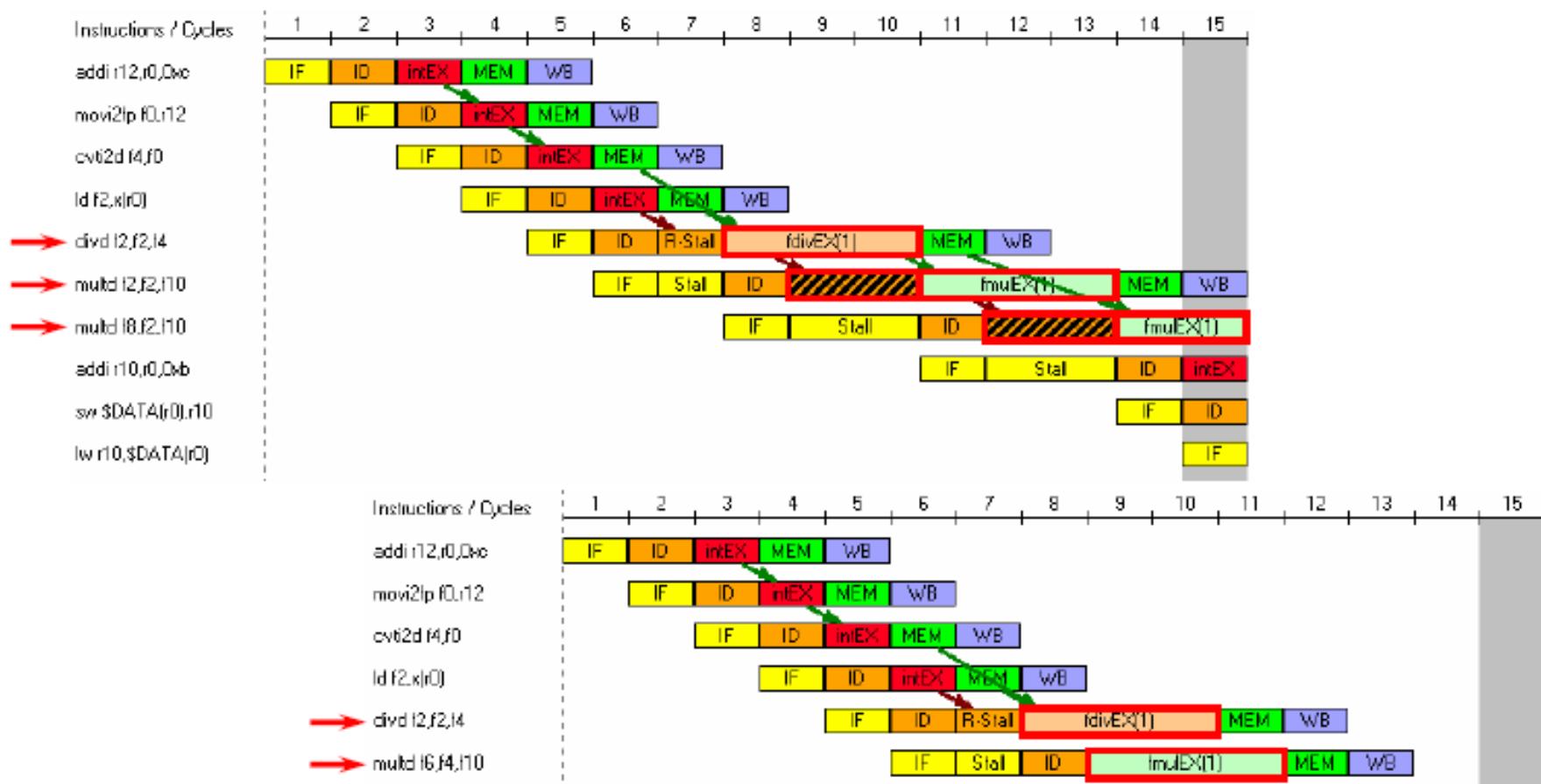
Sesión 2. Superescalares: motivación
y cauce

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

- Dependencias estructurales provocan pérdidas de ciclos
 - Ejemplo de una unidad FP vs. Varias unidades FP

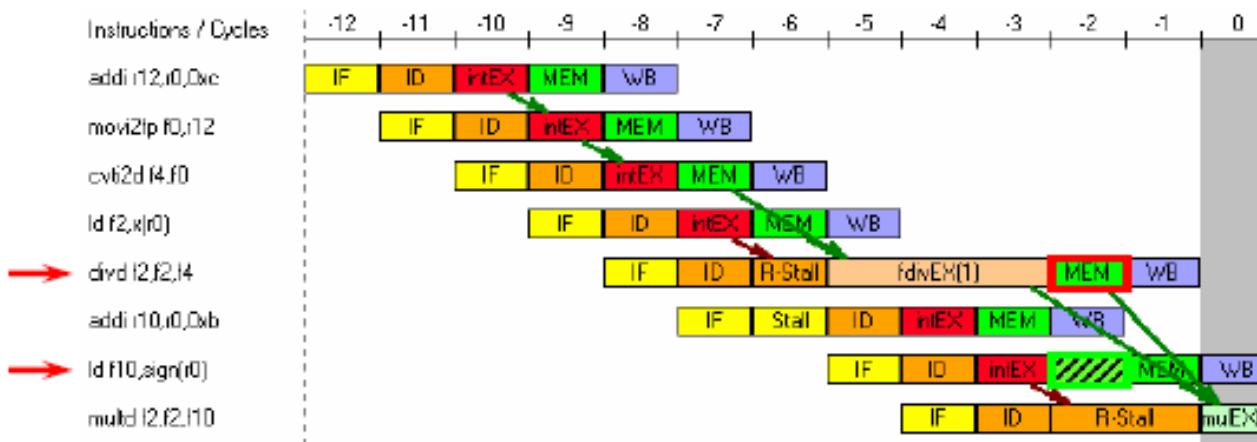


Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

- Varias unidades funcionales permiten la ejecución fuera de orden
 - Validar riesgos WAR y WAW



- Se obtienen mejores prestaciones si se pueden procesar varias instrucciones en la misma etapa → **procesamiento superescalar**

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

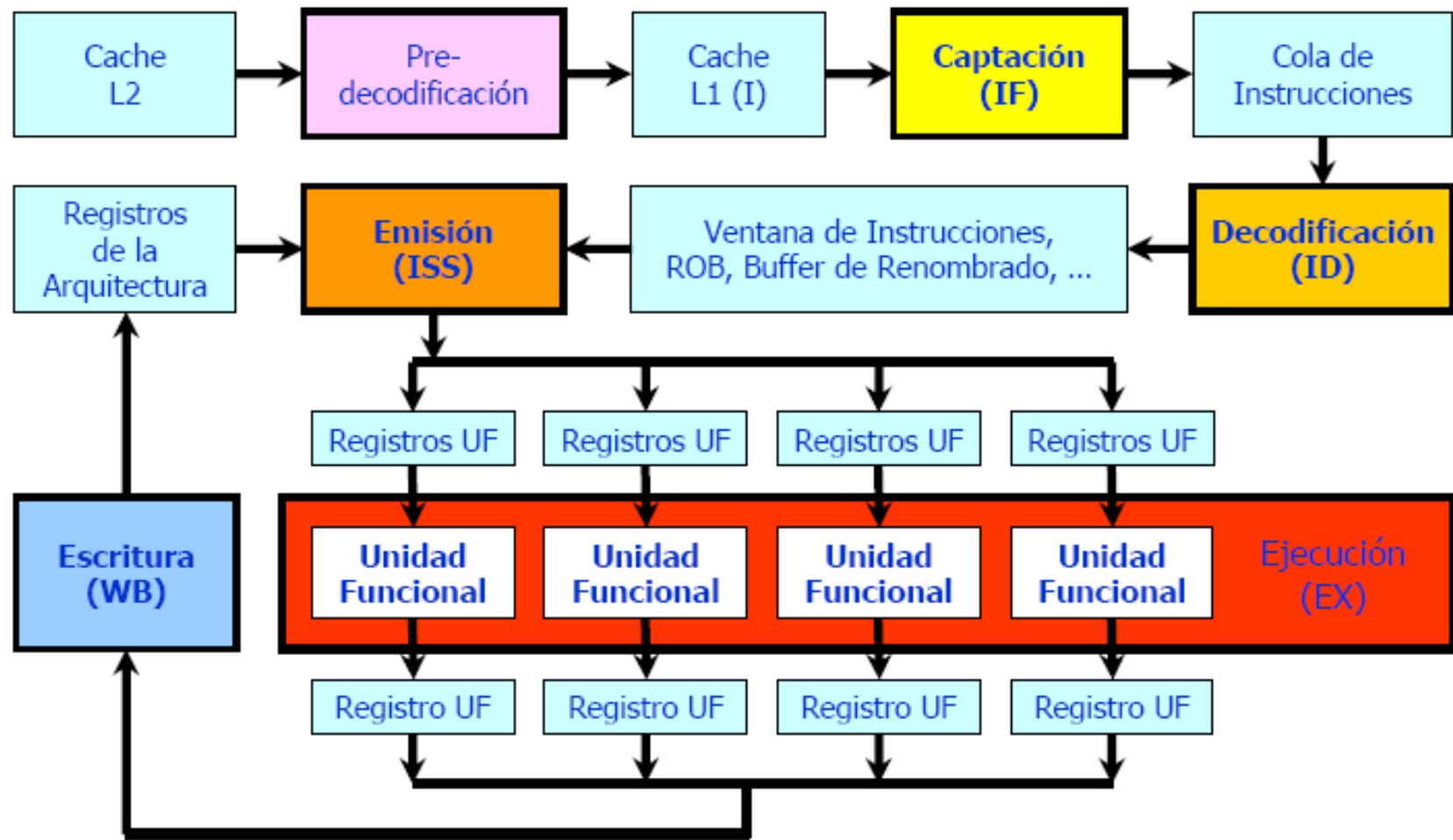
- Etapas
 - Captación de instrucciones (IF)
 - Decodificación de instrucciones (ID)
 - Emisión de instrucciones (ISS)
 - Ejecución de instrucciones (EX) – Instrucción finalizada o “finish”
 - Escritura (WB) – Instrucción completada o “complete”
- Características del procesamiento superescalar
 - Diferentes tipos de órdenes: orden de captación, orden de emisión, orden de finalización
 - Capacidad para identificar ILP existente y organizar el uso de las distintas etapas para optimizar recursos

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

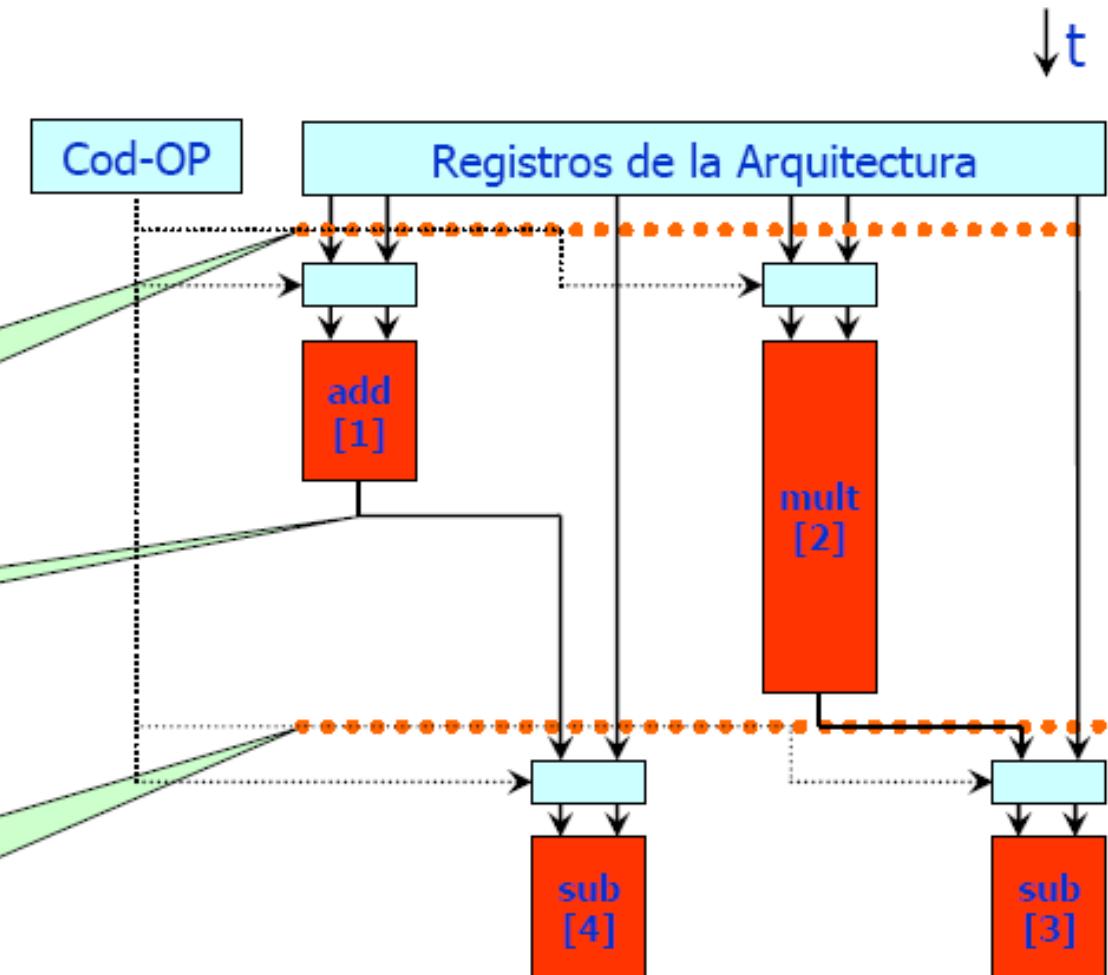
- Emisión ordenada

<i>i</i>	Instr. <i>i</i>	Latencia
[1]	add r4,r1,r2	(2)
[2]	mult r5,r1,r5	(5)
[3]	sub r6,r5,r2	(2)
[4]	sub r5,r4,r3	(2)

Emisión de [1] y [2] (tienen sus operandos)

Aunque [4] tiene sus operandos debe esperar

Emisión de [3] (tiene sus operandos al terminar [2]) y [4]



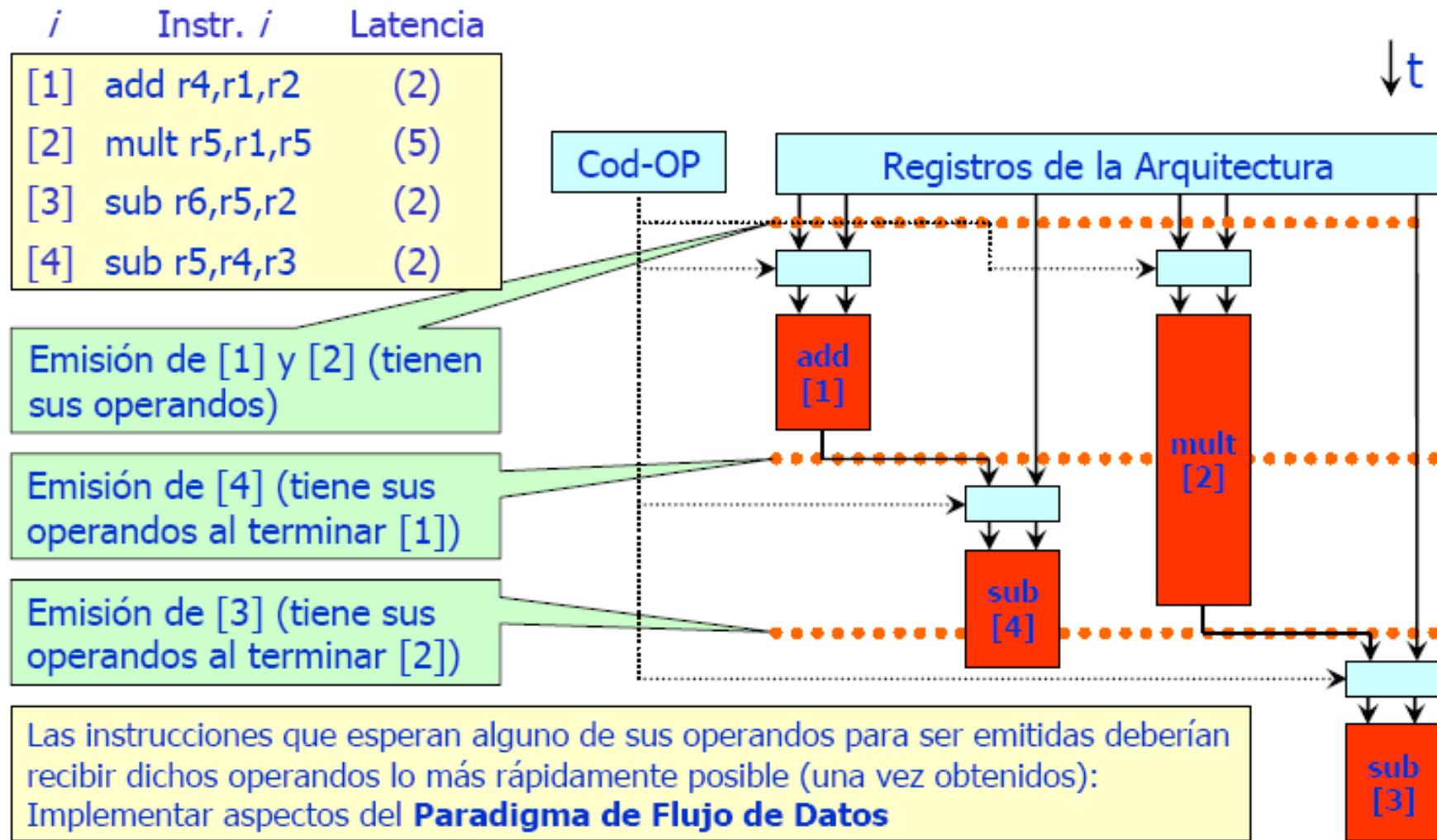
Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

- Emisión desordenada



Ingeniería de los Computadores

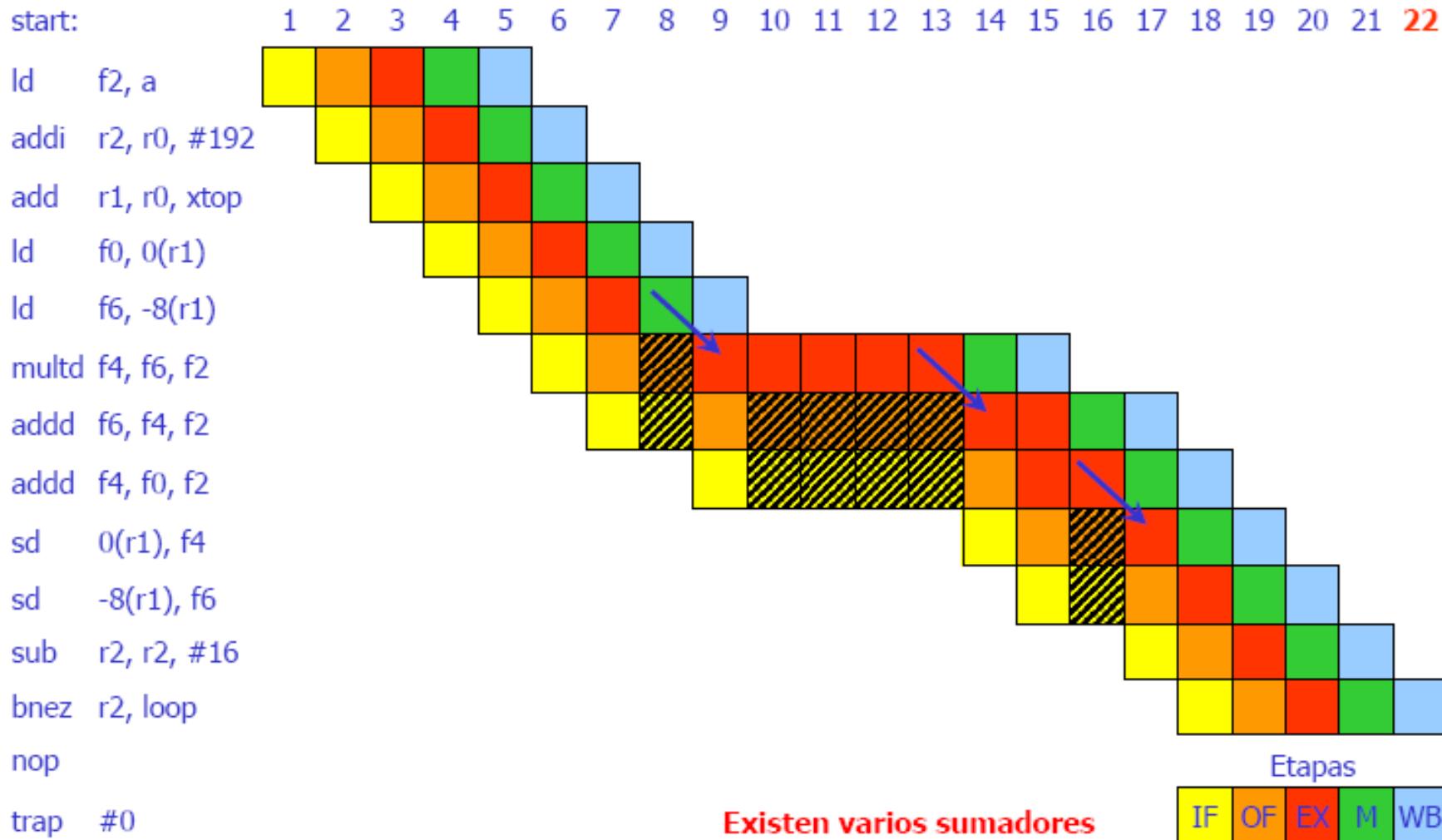
Sesión 2. Superescalares

Motivación

Cause

- Procesamiento segmentado

start:



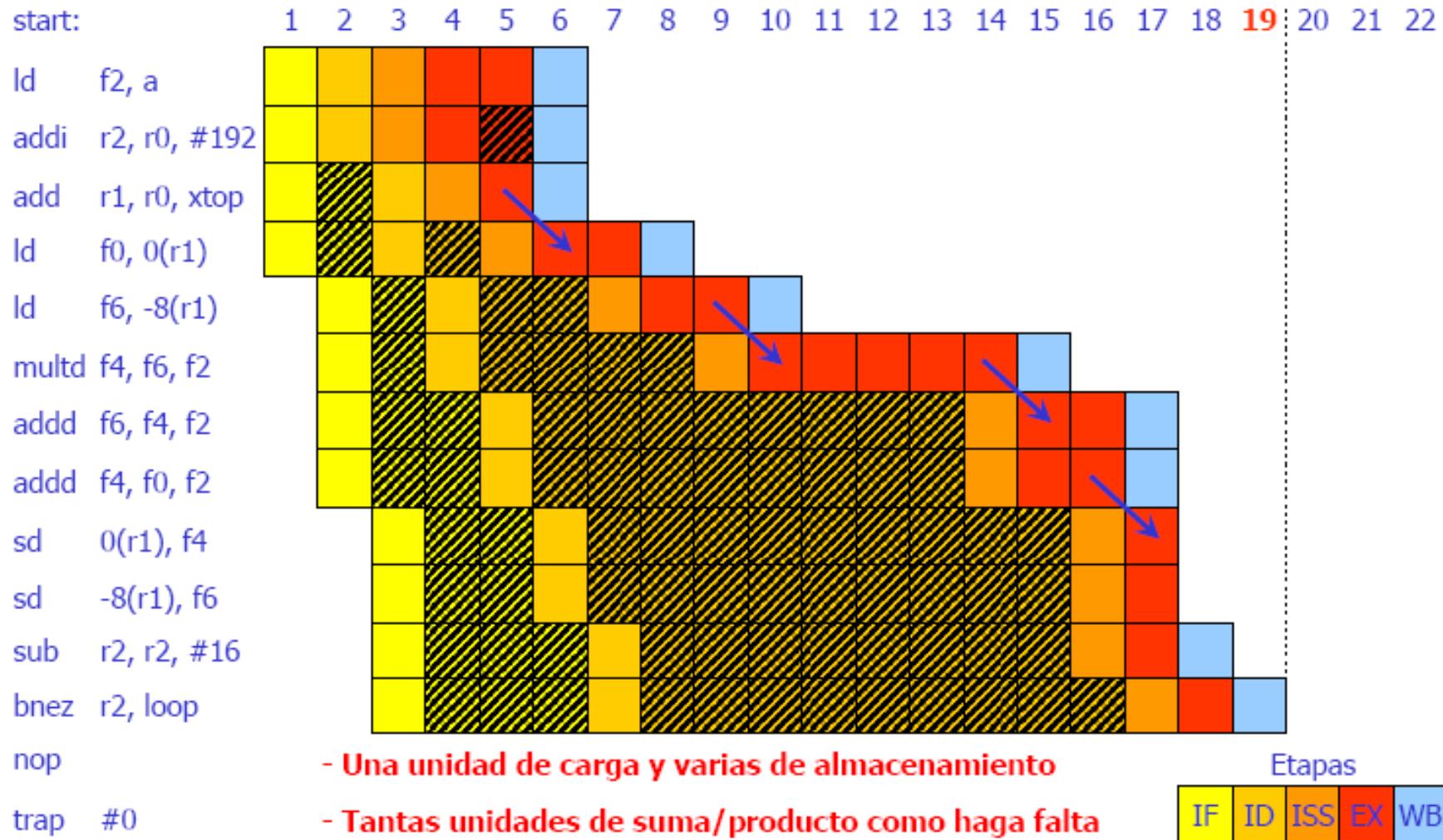
Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

- Superescalar: emisión ordenada/finalización ordenada



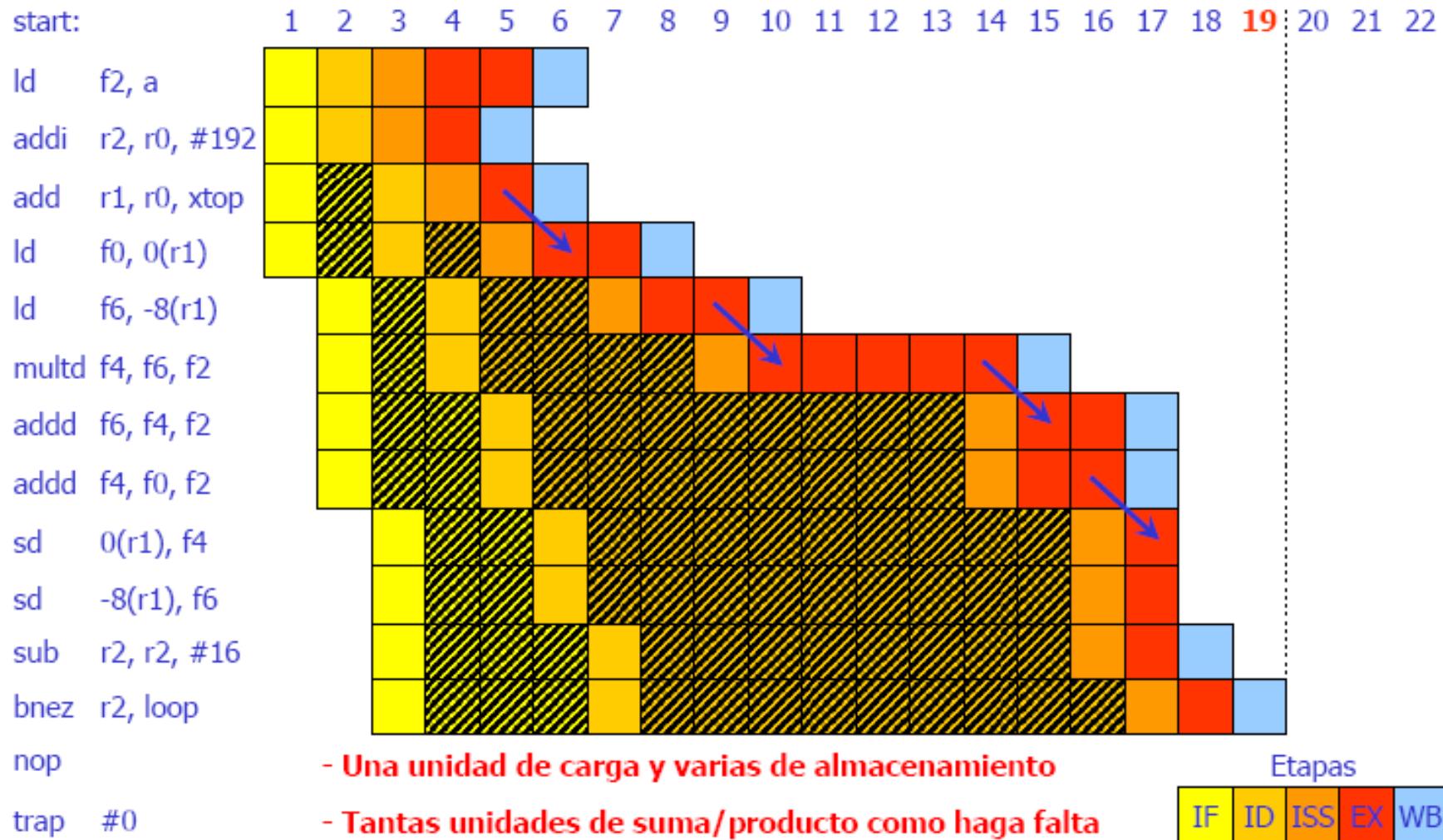
Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

- Superescalar: emisión ordenada/finalización desordenada



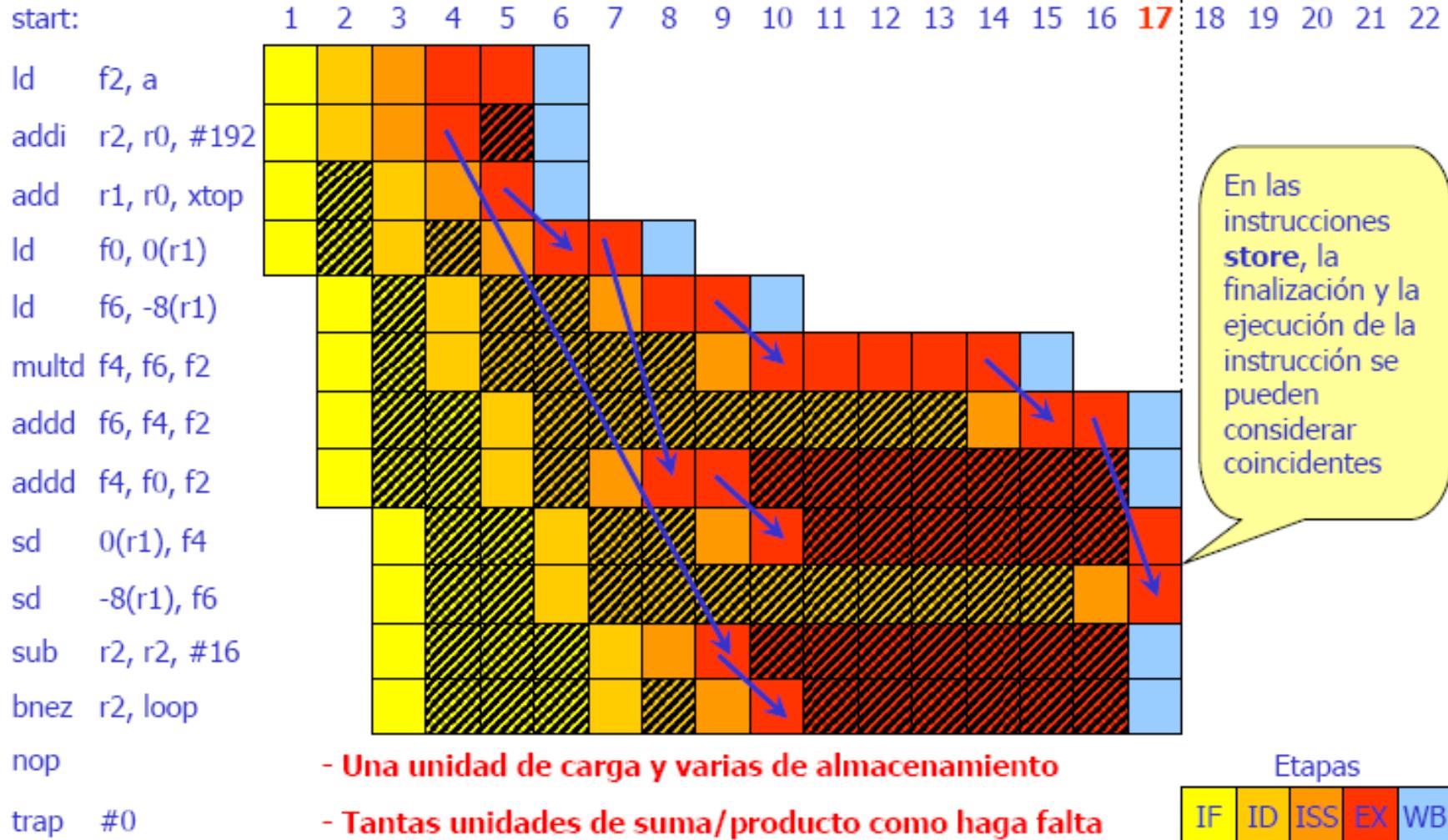
Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

- Superescalar: emisión desordenada/finalización ordenada



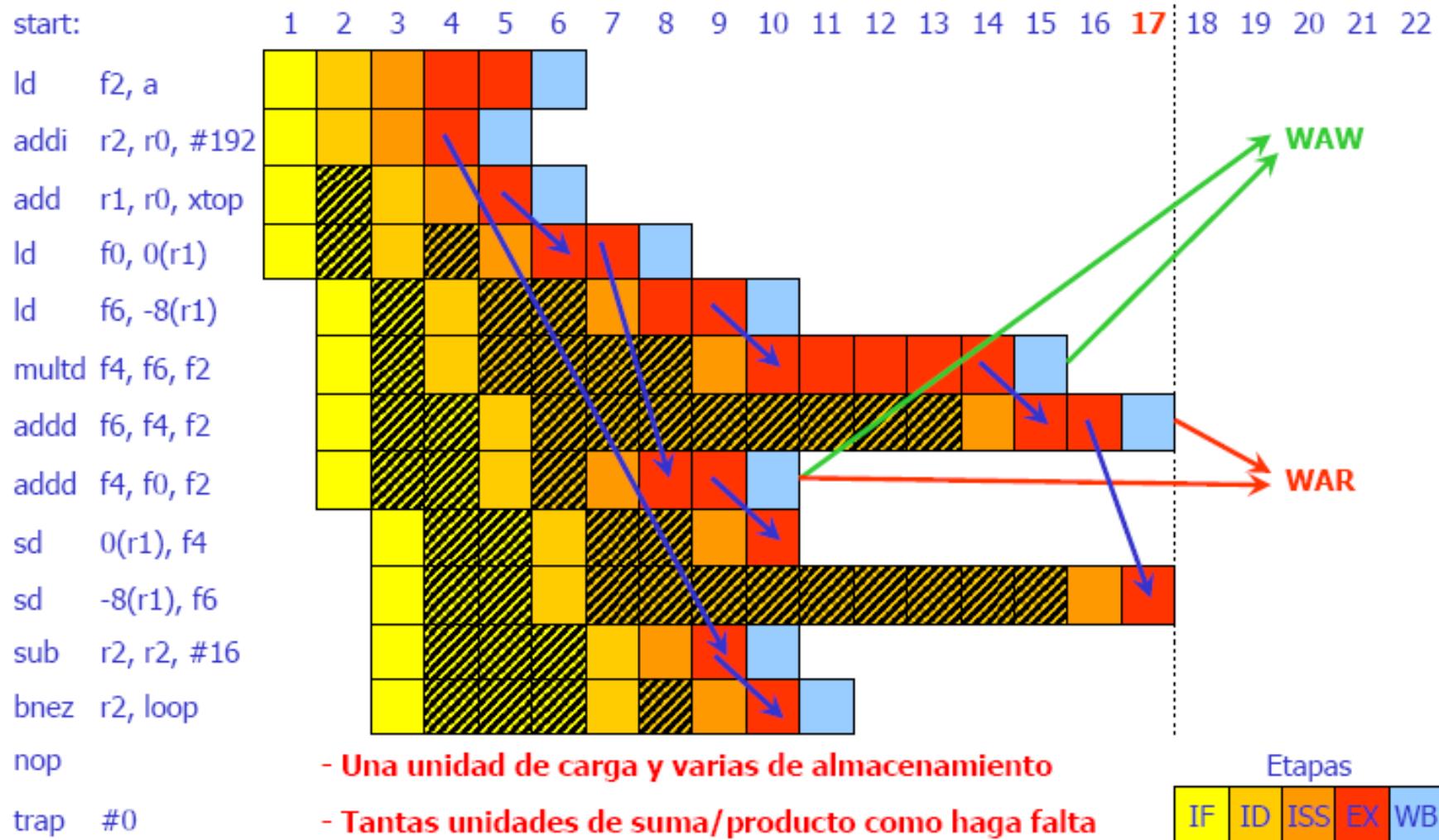
Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cause

- Superescalar: emisión desordenada/finalización desordenada



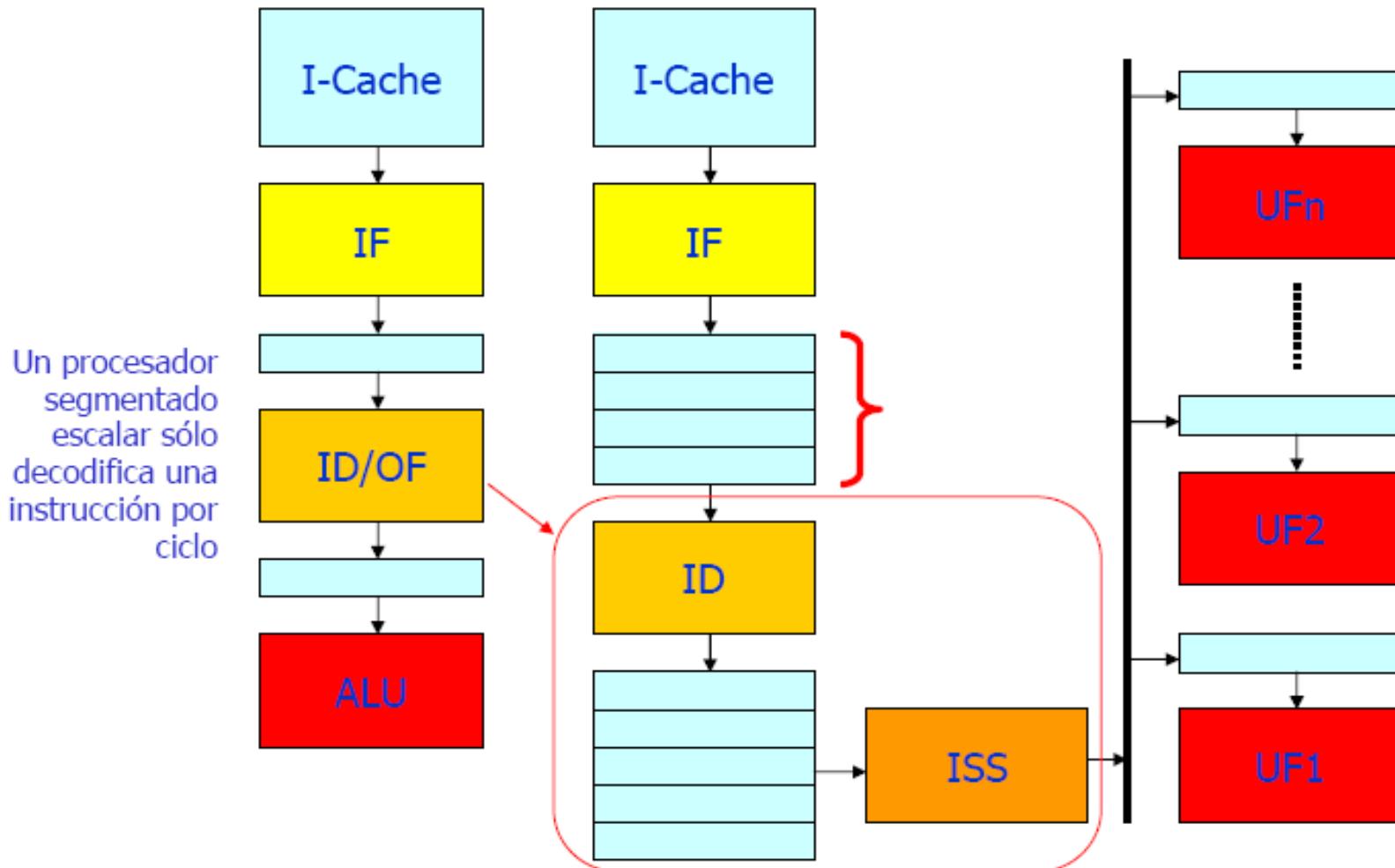
Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación



En un procesador superescalar se han de decodificar varias instrucciones por ciclo (y comprobar las dependencias con las instrucciones que se están ejecutando)

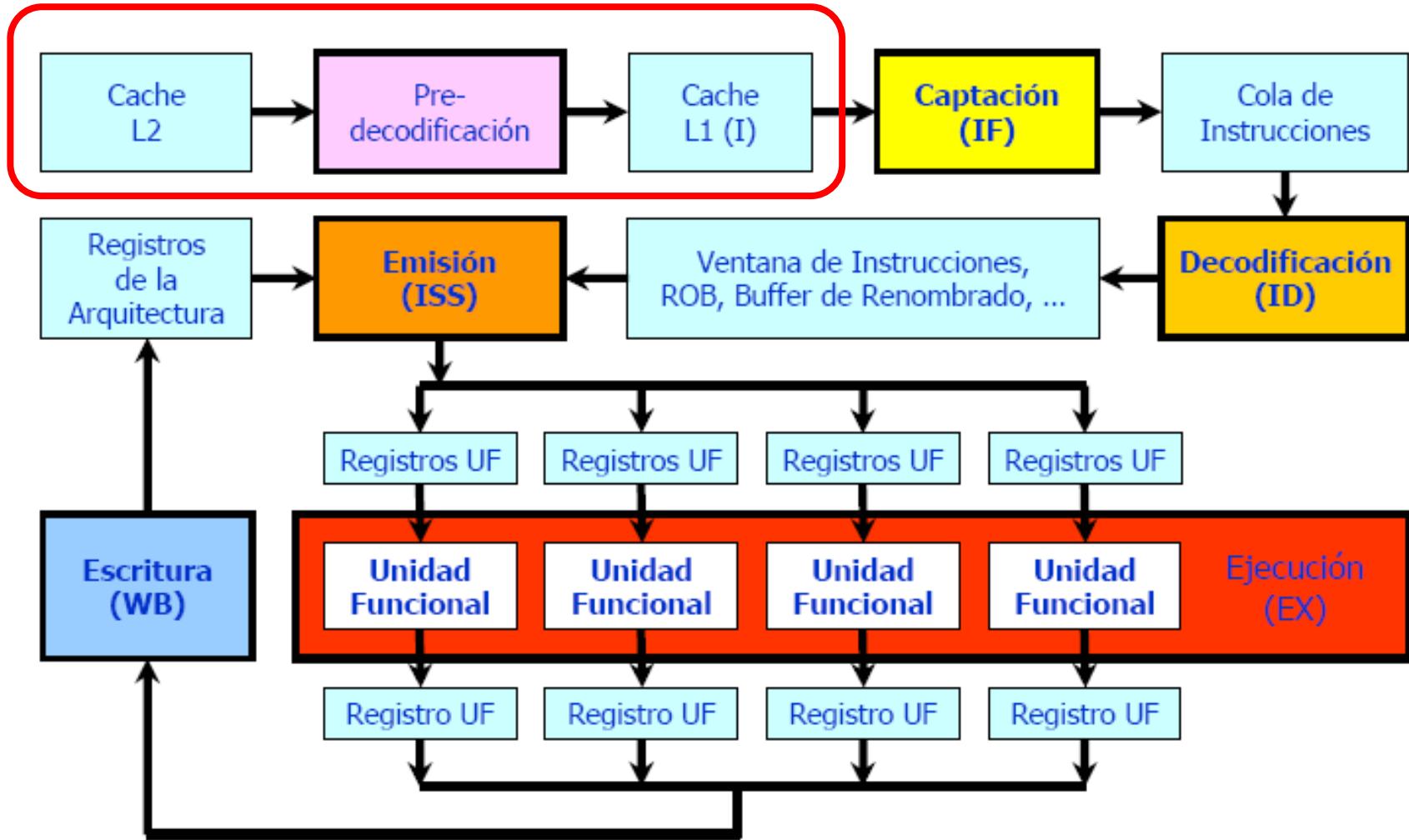
Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

- Bits de predecodificación pueden indicar:
 - Si es una instrucción de salto o no (se puede empezar su procesamiento antes)
 - El tipo de unidad funcional que va a utilizar (se puede emitir más rápidamente si hay cauces para enteros o coma flotante...)
 - Si hace referencia a memoria o no

Ingeniería de los Computadores

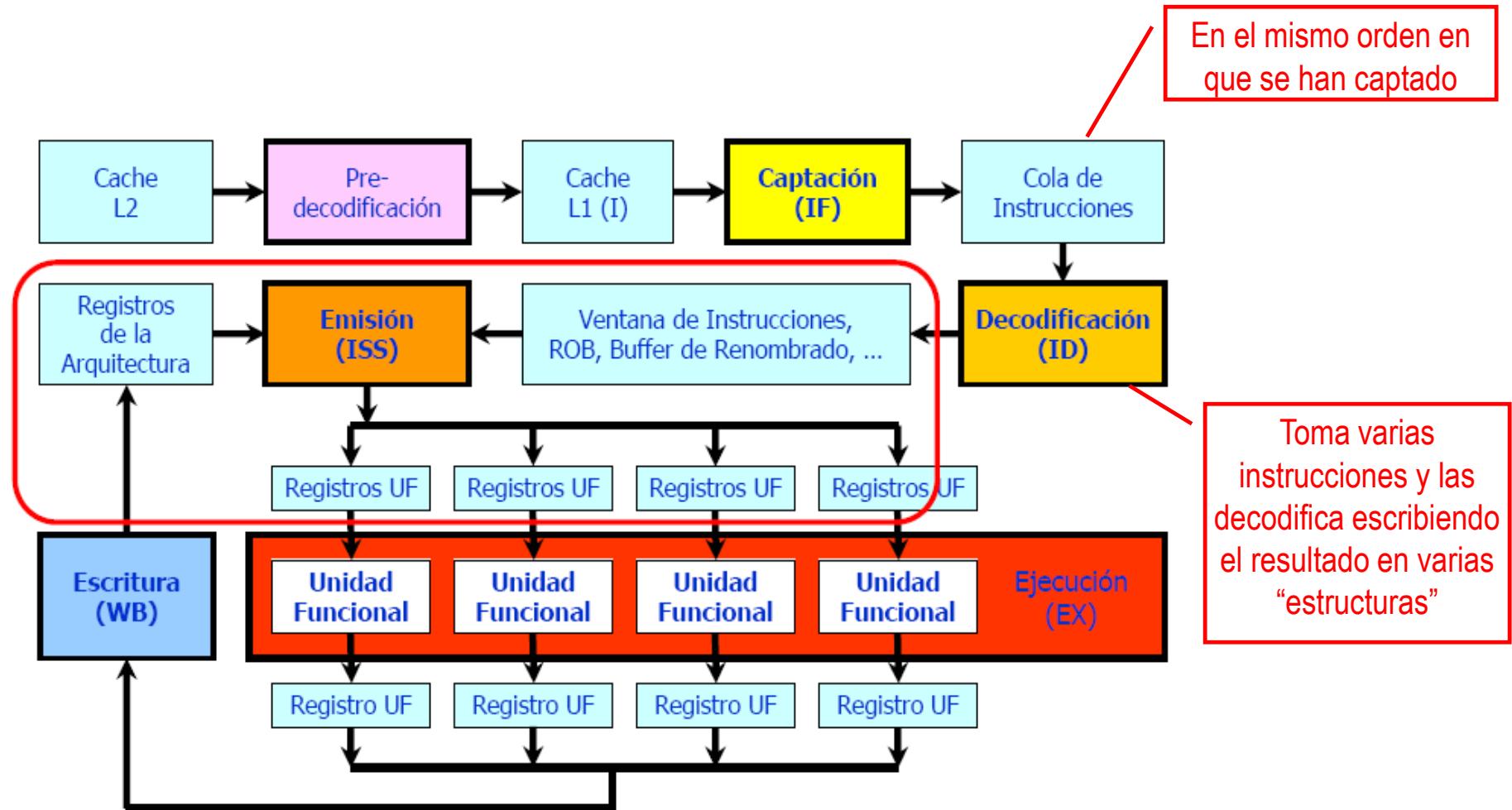
Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

- Ventana de instrucciones:
 - La ventana de instrucciones almacena las instrucciones pendientes (todas, si la ventana es centralizada o las de un tipo determinado, si es distribuida)
 - Las instrucciones se cargan en la ventana una vez decodificadas y se utiliza un bit para indicar si un operando está disponible (se almacena el valor o se indica el registro desde donde se lee) o no (se almacena la unidad funcional desde donde llegará el operando)
 - Una instrucción puede ser emitida cuando tiene todos sus operandos disponibles y la unidad funcional donde se procesará. Hay diversas posibilidades para el caso en el que varias instrucciones estén disponibles (características de los buses, etc.)

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

Ejemplo de Ventana de Instrucciones

#	opcode	address	rb_entry	operand1	ok1	typ1	operand2	ok2	typ2	pred
2	MULTD	loop + 0x4	2	1	0	FPD	0	0	FPD	-
1	LD	loop	1	0	0	INT	0	1	IMM	-

Lugar donde se
almacenará el resultado

Dato no válido
(indica desde dónde se recibirá el dato)

Dato válido
(igual a 0)

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

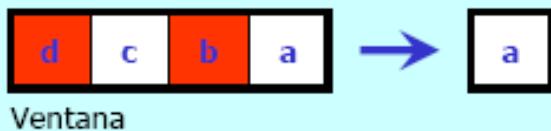
- **Orden:** Emisión Ordenada o Desordenada
- **Alineamiento:** Emisión Alineada o No alineada

Ejemplos:

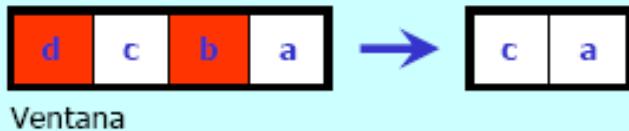
Alineada: SuperSparc (92), PowerPC (93, 95, 96), PA8000 (96), Alpha (92, 94, 95), R10000 (96)

No Alineada: MC88110 (93), PA7100LC (93), R8000 (94), UltraSparc (95)

Emisión Ordenada

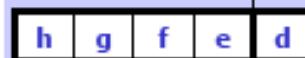


Emisión Desordenada



Instrucción no preparada para la emisión

Emisión Alineada



Ordenada

Emisión No alineada



Ordenada

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

[4] puede emitirse pero debe esperar a la [3]

add/sub: 2
mult: 1

Ventana de Registros

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(7)	(8)-(9)

Ha terminado [2]: pueden emitirse [3] y [4]

sub r5 [r4] 1 - [r3] 1 -
sub r6 [r5] 1 - [r2] 1 -

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

[4] puede emitirse pero debe esperar a la [3]

add/sub: 2
mult: 1

Ventana de Registros

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(7)	(8)-(9)

Ha terminado [2]: pueden emitirse [3] y [4]

sub r5 [r4] 1 - [r3] 1 -
sub r6 [r5] 1 - [r2] 1 -

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -



Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

add/sub: 2
mult: 1

Ventana de
Registros

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(4)	(5)-(6)



Se ha emitido [4] y ha terminado
[2]: puede emitirse [3]



sub r6 [r5] 1 - [r2] 1 -

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

↓ Se han emitido [1] y [2]

add/sub: 1
mult: 1

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(10)	(11)-(12)

Ha terminado [2] y se ha emitido [3].
Cuando la unidad quede libre se emitirá [4]

sub r5 [r4] 1 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



sub r5 [r4] 1 - [r3] 1 -

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -		
sub r6 r5 0 mult [r2] 1 -		
mult r5 [r1] 1 - [r5] 1 -		
add r4 [r1] 1 - [r2] 1 -		



Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -		
sub r6 r5 0 mult [r2] 1 -		



Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -		
sub r6 r5 0 mult [r2] 1 -		



sub r6 [r5] 1 - [r2] 1 -

Ventana de Registros

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(4)	(5)-(6)

Se ganan 3 ciclos

Se ha emitido [4] y ha terminado [2]: puede emitirse [3]

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Cola de
Instrucciones

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)



sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Cola de
Instrucciones

ID/ISS

sub r6 r5 0 mult [r2] 1 -
add r4 [r1] 1 - [r2] 1 -

Estaciones de Reserva
(con capacidad para dos
instrucciones)

mult r5 [r1] 1 - [r5] 1 -

Ingeniería de los Computadores

Sesión 2. Superescalares

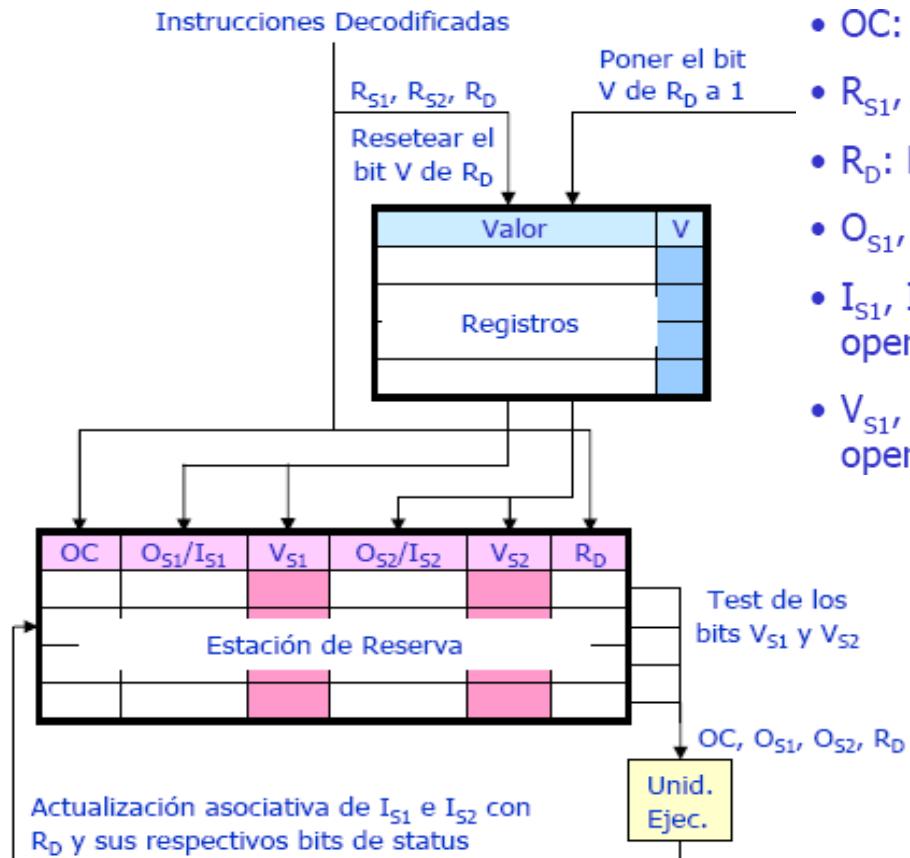
Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva



- OC: Código de operación
- R_{S1}, R_{S2}: Registros fuente
- R_D: Registro de destino
- O_{S1}, O_{S2}: Operandos fuente
- I_{S1}, I_{S2}: Identificadores de los operandos fuente
- V_{S1}, V_{S2}: Bits válidos de los operandos fuente

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

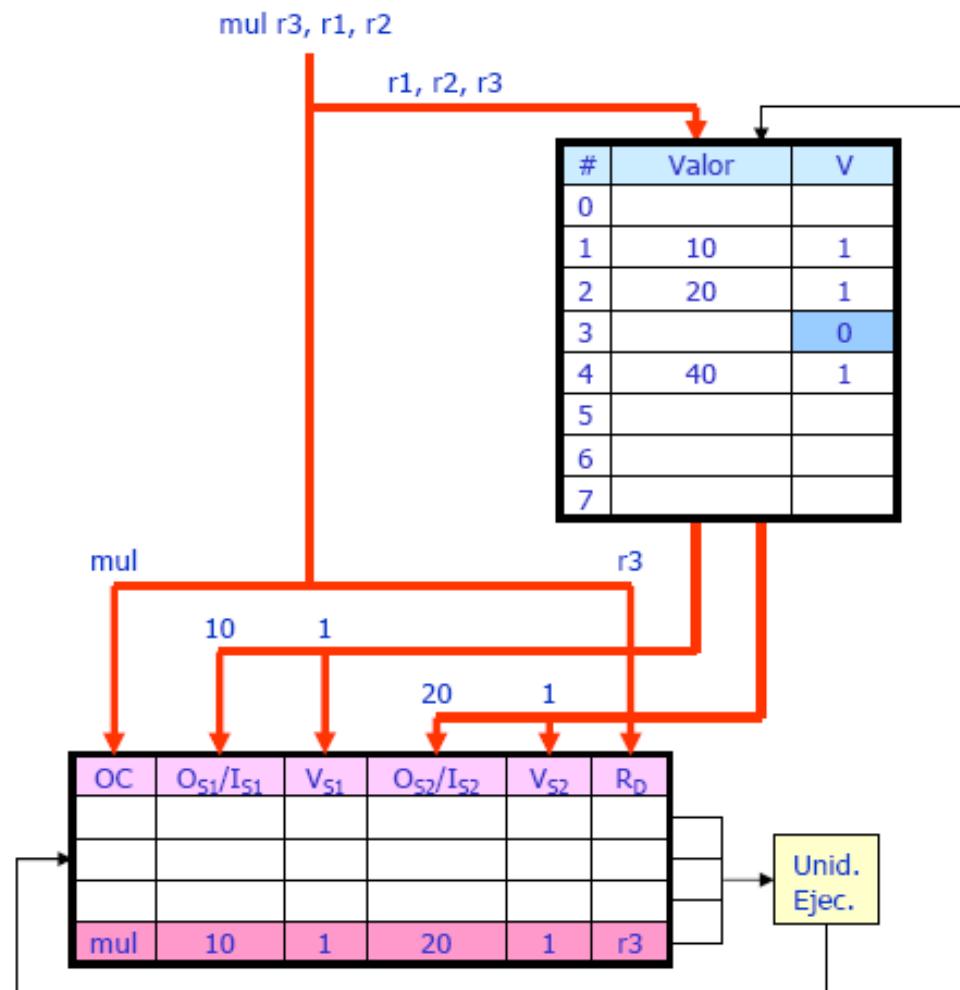
Emisión

- Estaciones de reserva. Ejemplo de uso (1)

Ciclo i:	mul r3, r1, r2
Ciclo i+1:	add r5, r2, r3
	add r6, r3, r4

Ciclo i:

- Se emite la instrucción de multiplicación, ya decodificada, a la estación de reserva
- Se anula el valor de r3 en el banco de registros
- Se copian los valores de r1 y r2 (disponibles) en la estación de reserva



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (2)

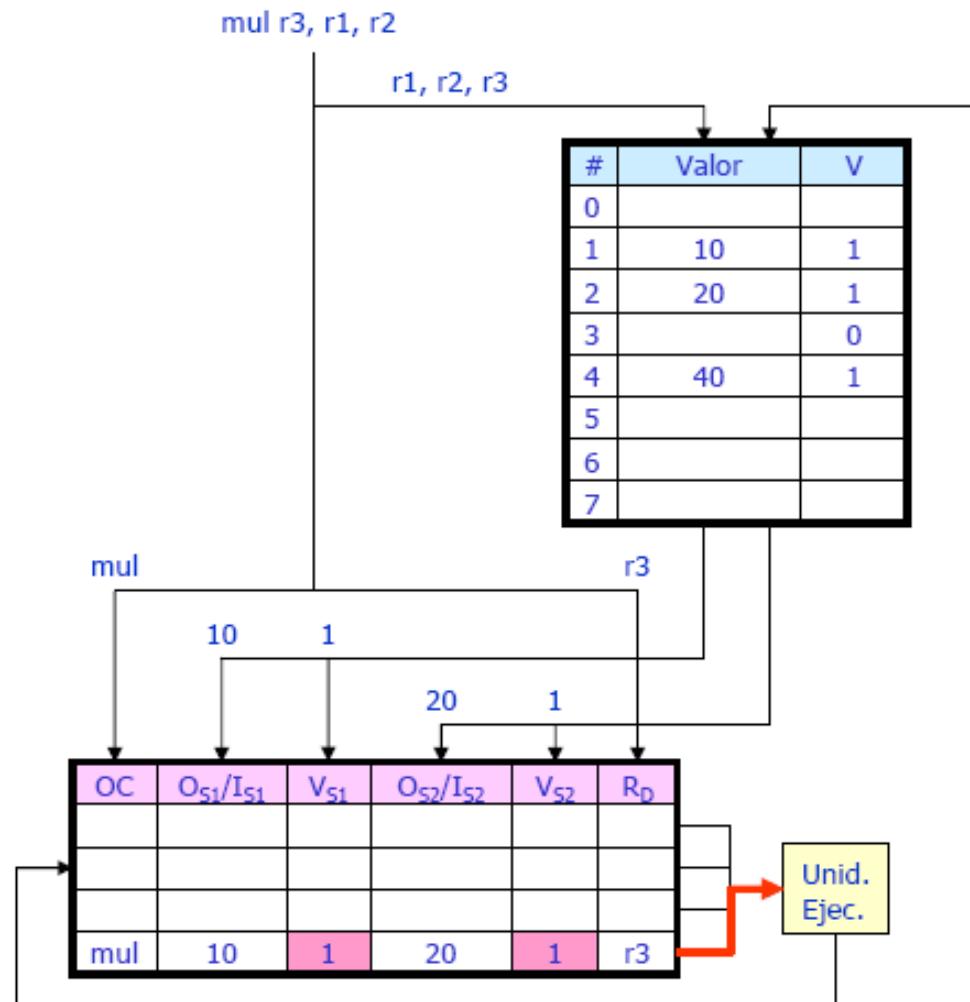
Ciclo i: mul r3, r1, r2

Ciclo i+1: add r5, r2, r3

add r6, r3, r4

Ciclo i + 1;

- La operación de multiplicación tiene sus operadores preparados ($V_{S1} = 1$ y $V_{S2} = 1$)
 - Así que puede enviarse a la unidad de ejecución



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

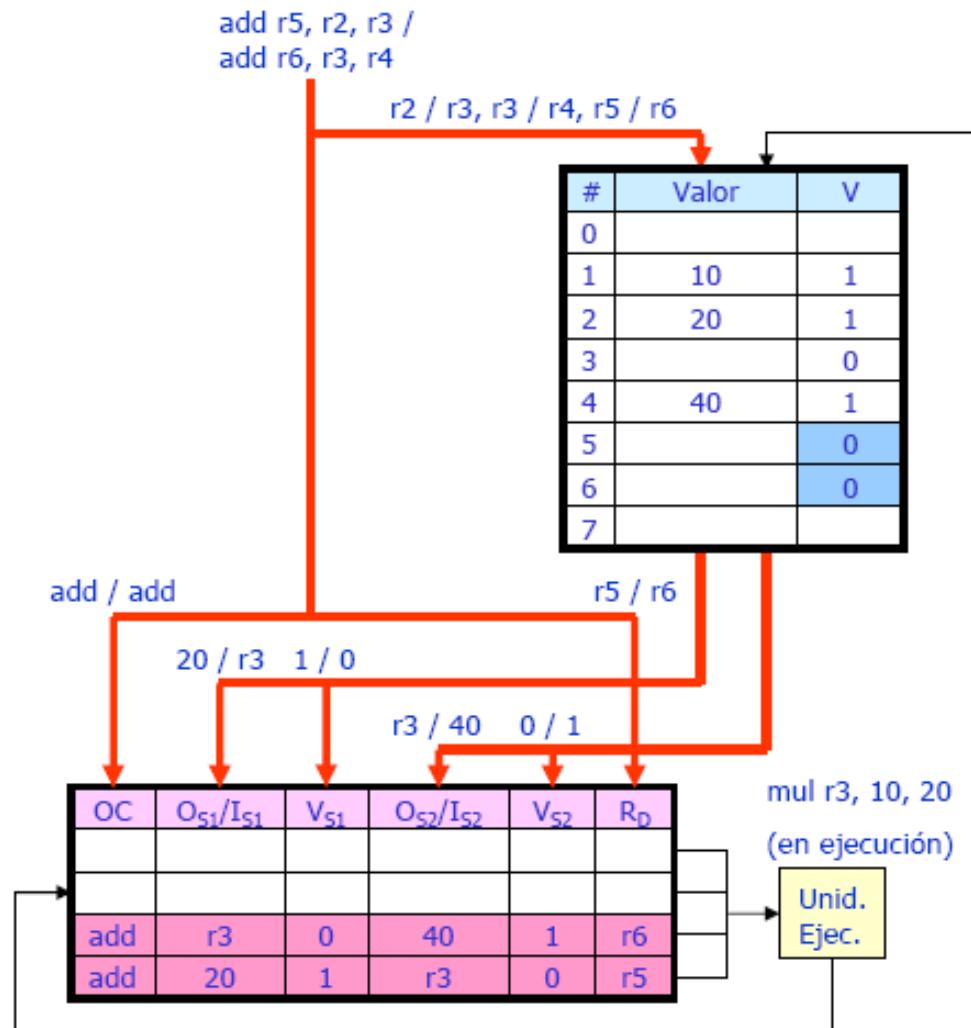
Emisión

- Estaciones de reserva. Ejemplo de uso (3)

Ciclo i: mul r3, r1, r2
 Ciclo i+1: add r5, r2, r3
 add r6, r3, r4

Ciclo i + 1 (*cont.*):

- Se emiten las dos instrucciones de suma a la estación de reserva
- Se anulan los valores de r5 y r6 en el banco de registros
- Se copian los valores de los operandos disponibles y los identificadores de los operandos no preparados



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

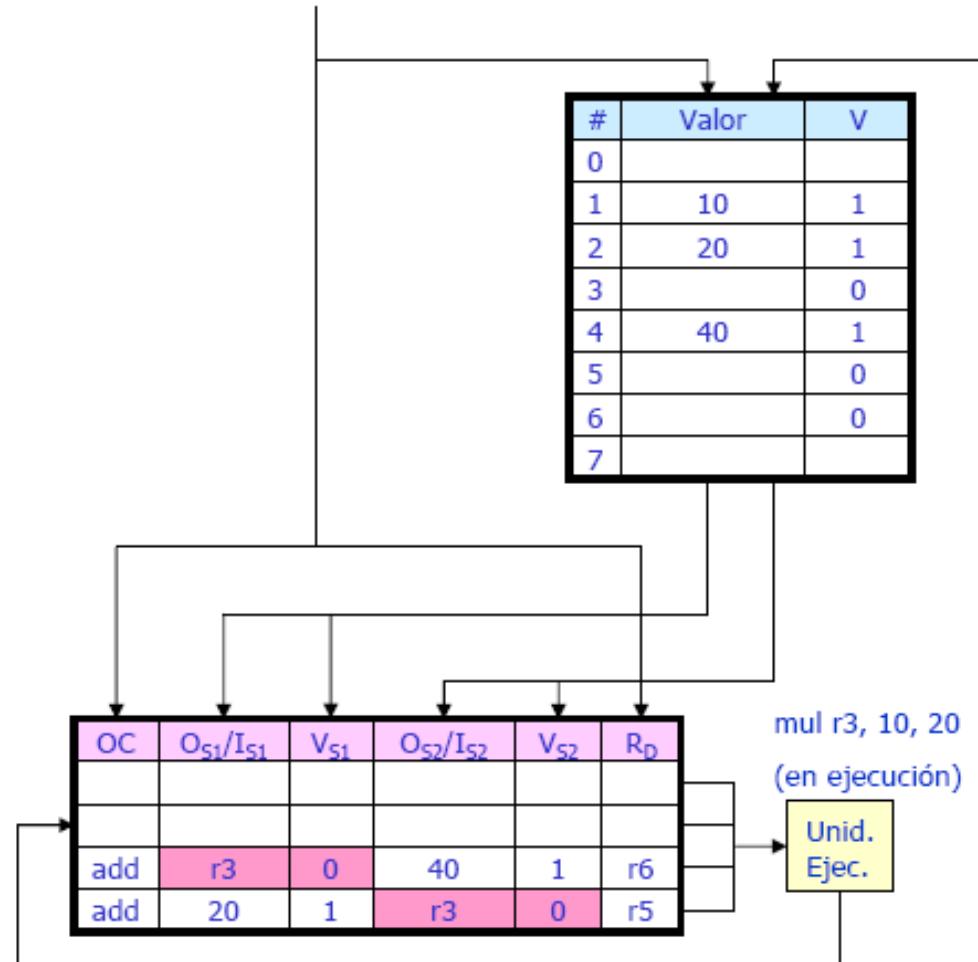
Emisión

- Estaciones de reserva. Ejemplo de uso (4)

Ciclo i: mul r3, r1, r2
 Ciclo i+1: add r5, r2, r3
 add r6, r3, r4

Ciclos i + 2 .. i + 5:

- La multiplicación sigue ejecutándose
- No se puede ejecutar ninguna suma hasta que esté disponible el resultado de la multiplicación (r3)



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

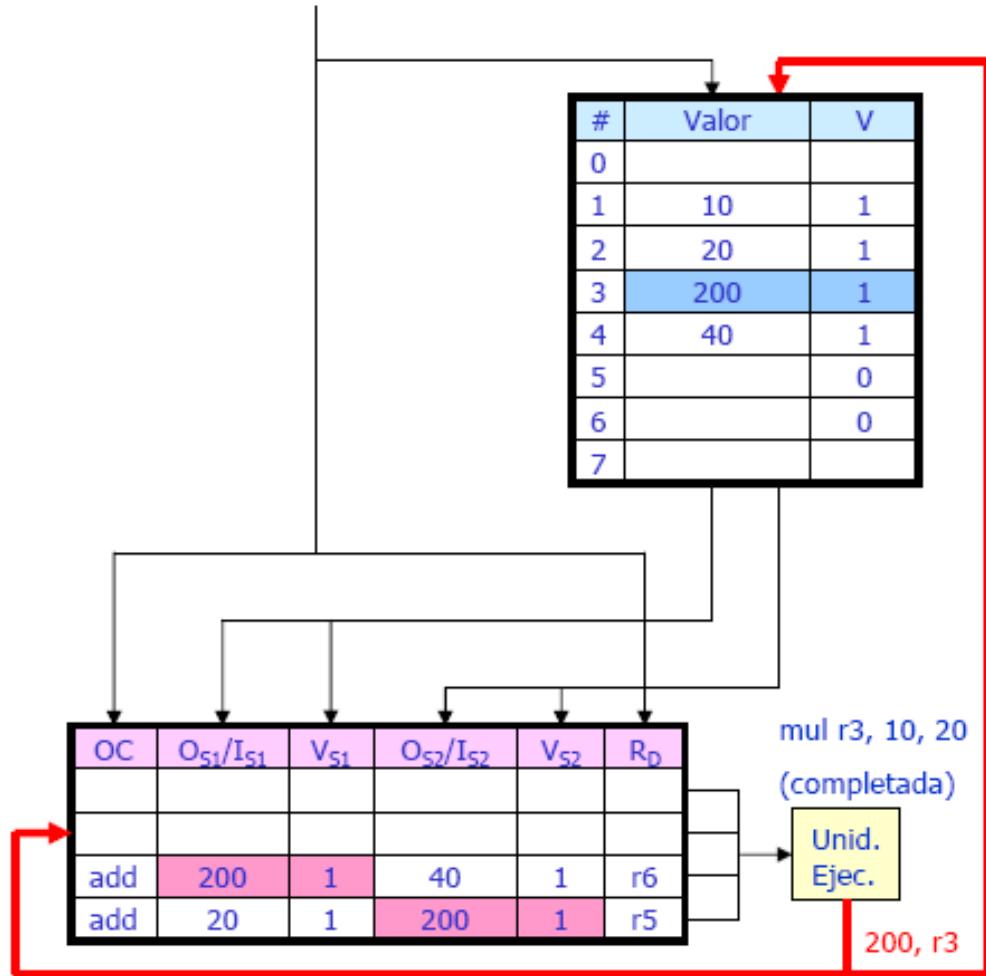
Emisión

- Estaciones de reserva. Ejemplo de uso (5)

Ciclo i: mul r3, r1, r2
 Ciclo i+1: add r5, r2, r3
 add r6, r3, r4

Ciclo i + 6:

- Se escribe el resultado de la multiplicación en el banco de registros y en las entradas de la estación de reserva
- Se actualizan los bits de disponibilidad de r3 en el banco de registros y en la estación de reserva



Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

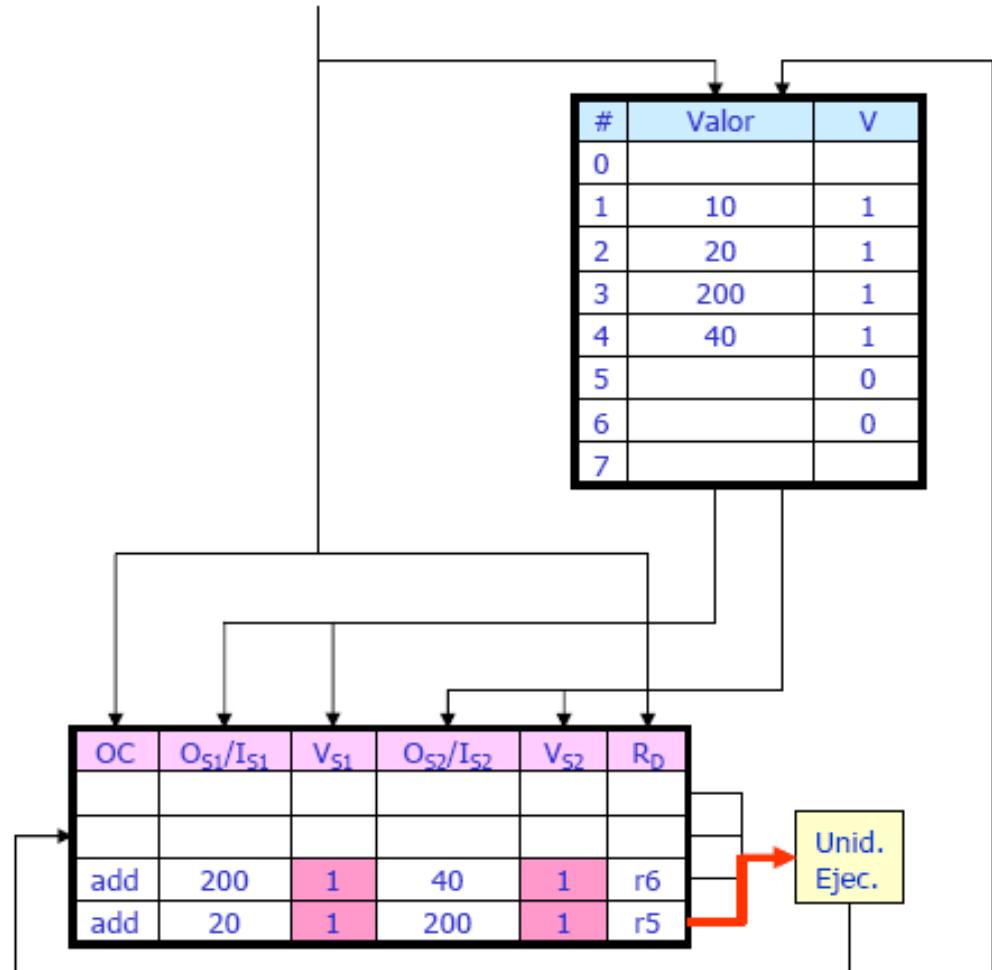
Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (6)

Ciclo i: mul r3, r1, r2
Ciclo i+1: add r5, r2, r3
add r6, r3, r4



Ciclo i + 6 (*cont.*):

- Las sumas tienen sus operadores preparados ($VS_1 = 1$ y $VS_2 = 1$)
- Así que pueden enviarse a la unidad de ejecución

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

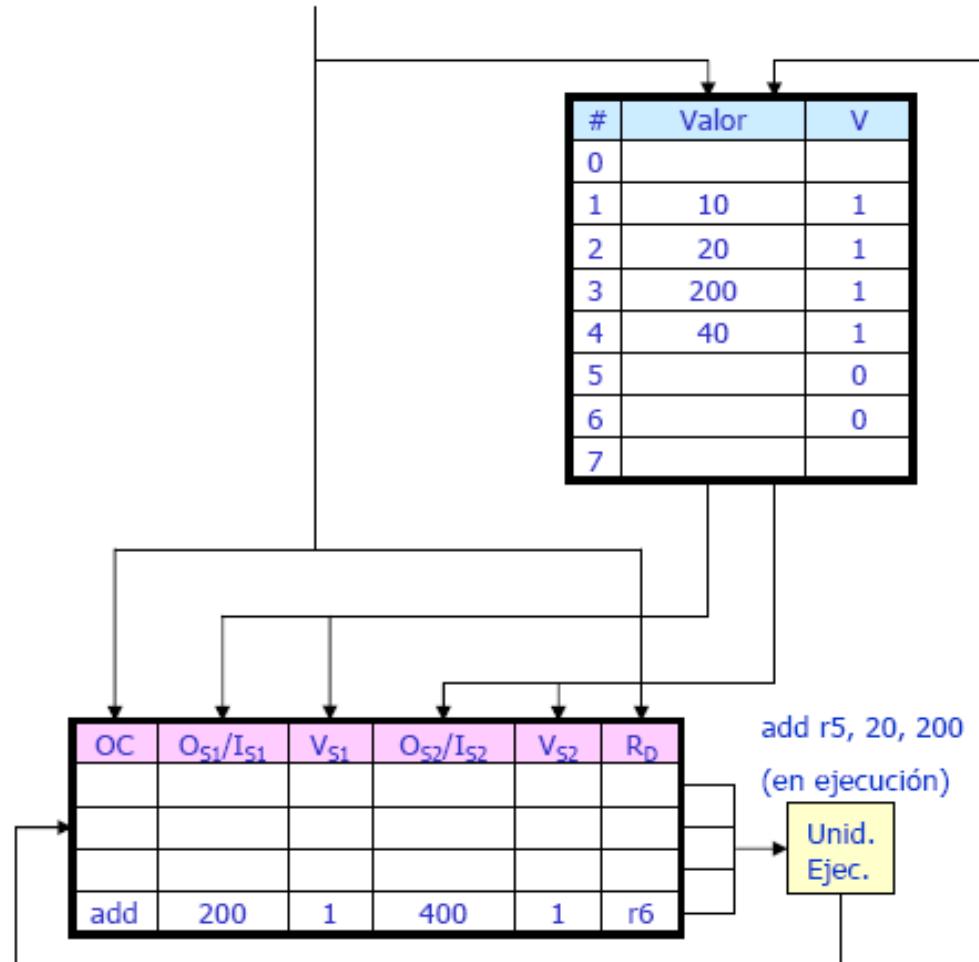
Emisión

- Estaciones de reserva. Ejemplo de uso (7)

Ciclo i: mul r3, r1, r2

Ciclo i+1: add r5, r2, r3

add r6, r3, r4



Ciclo $i + 6$ (*cont.*):

- Como sólo hay una unidad de ejecución, se envía la instrucción más antigua de la estación de reserva, la primera suma

Ingeniería de los Computadores

Sesión 2. Superescalares

Motivación

Cauce

Decodificación

Emisión

- Ejercicio
 - Emisión ordenada/desordenada
 - Recursos: 1 lw/sw, 1 add/sub, 1 mult/div
 - Recursos: 1 lw/sw, 2 add/sub, 2 mult/div
 - Latencia: lw/sw (5 ciclos), add/sub (2 ciclos), mult/div (5 ciclos)

```
Lw r1,0(r2)
Add r2,r1,r3
Mult r3,r1,r2
Sub r4,r1,r2
Add r4,r1,r2
Div r4,r5,r6
```

Ingeniería de los Computadores

Sesión 3. Superescalares: estructuras

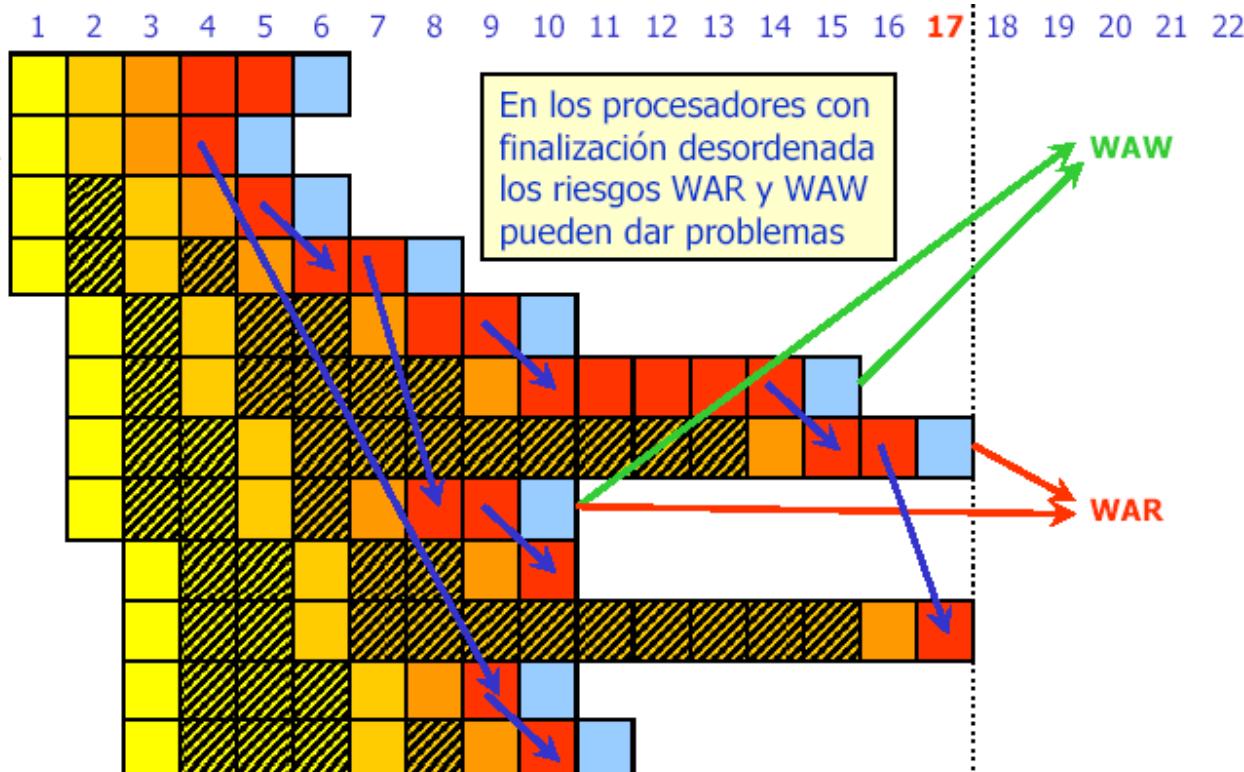
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

- Renombrado de registros (motivación)

start:
Id f2, a
addi r2, r0, #192
add r1, r0, xtop
Id f0, 0(r1)
Id f6, -8(r1)
multd f4, f6, f2
addd f6, f4, f2
addd f4, f0, f2
sd 0(r1), f4
sd -8(r1), f6
sub r2, r2, #16
bnez r2, loop
nop
trap #0



- Una unidad de carga y varias de almacenamiento
- Tantas unidades de suma/producto como haga falta



Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

- Técnica para **evitar el efecto de las dependencias WAR, o Antidependencias** (en la emisión desordenada) y **WAW, o Dependencias de Salida** (en la ejecución desordenada).

```
R3 := R3 - R5  
R4 := R3 + 1  
R3 := R5 + 1  
R7 := R3 * R4
```

Cada escritura se asigna
a un registro físico distinto

```
R3b := R3a - R5a  
R4a := R3b + 1  
R3c := R5a + 1  
R7a := R3c * R4a
```

Sólo RAW

R.M. Tomasulo (67)

Implementación Estática: Durante la Compilación

Implementación Dinámica: Durante la Ejecución (circuitería adicional y registros extra)

Características de los Buffers de Renombrado

- **Tipos de Buffers** (separados o mezclados con los registros de la arquitectura)
- **Número de Buffers de Renombrado**
- **Mecanismos para acceder a los Buffers** (asociativos o Indexados)

Velocidad del Renombrado

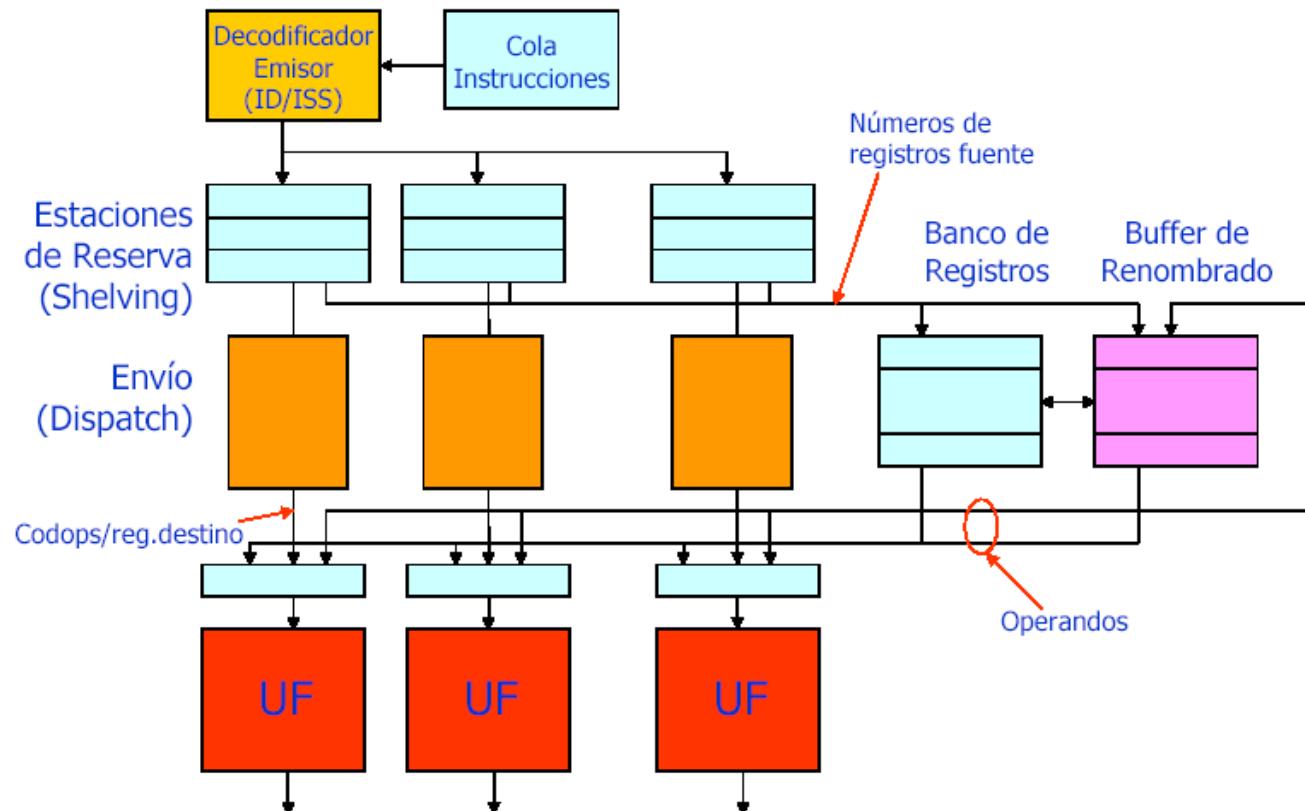
- **Máximo número de nombres asignados por ciclo** que admite el procesador

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

- Renombrado de registros
 - Estaciones de reserva + buffer de renombrado



Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

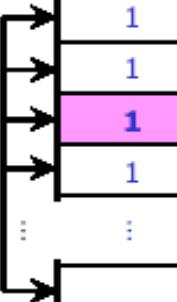
- Tipos de buffers de renombrado

Buffer de Renombrado con Acceso Asociativo

Búffer de Renombrado

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
1	5	50	1	1
1	12	1200	1	1
1	2	20	1	1
1	1	3	1	1
:	:	:	:	:

Búsq. asoc. de r2



Buffer de Renombrado con Acceso Indexado

Índices

Búsq.
de r2 →

Entrada Válida	Índice
0	2
1	5
2	3
3	12
:	:

Búffer de renombrado

Valor	Valor Válido
45	1
320	1
30	1
20	1
:	:

- Permite varias escrituras pendientes a un mismo registro
- Se utiliza el bit último para marcar cual ha sido la más reciente

- Sólo permite una escritura pendiente a un mismo registro
- Se mantiene la escritura más reciente

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (I)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	0			
5	0			
6	0			
7	0			
8	0			
9	0			
10	0			
11	0			
12	0			
13	0			
14	0			

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

$$\begin{aligned}r2a &= r0a * r1a \\r3a &= r1a + r2a \\r2b &= r0a - r1a\end{aligned}$$

Situación Inicial:

- Existen dos renombrados de r1 en las entradas 2 y 3 del buffer
- La última está en la entrada 3

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (II)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	1
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				

 r0: 0, "válido"
 r1: 15, "válido"
 4

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

Ciclo i:

- Se emite la multiplicación
- Se accede a los operandos de la multiplicación, que tienen valores válidos en el buffer de renombrado
- Se renombra r2 (el destino de mul)

Renombrado

Ejemplo de Renombrado (III)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	1
5	3		0	1
6				
7				
8				
9				
10				
11				
12				
13				
14				

r1: 15, "válido"

r2: "no válido"

5

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo $i + 1$:

- Se emite la suma
- Se accede a sus operandos, pero r2 no estará preparado hasta que termine la multiplicación
- Se renombra r3 (destino de add)

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (IV)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	0
5	3		0	1
6	2		0	1
7	0			
8	0			
9	0			
10	0			
11	0			
12	0			
13	0			
14	0			

 r0: 0, "válido"
 r1: 15, "válido"
 6

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

Ciclo i + 2:

- Se emite la resta
- Se accede a sus operandos
- Se vuelve a renombrar r2 (destino de sub)

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (V)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2	0	1	0
5	3		0	1
6	2		0	1
7				
8				
9				
10				
11				
12				
13				
14				

r1: 15, "válido" r2: 0, "válido"

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

Ciclo i + 5:

- Termina la multiplicación
- Se actualiza el resultado en el buffer de renombrado
- Ya se puede ejecutar la suma con el valor de r2 de la entrada 4
- Cuando termine la resta escribirá otro valor para r2 en la entrada 6
- Sólo se escribirá en el banco de registros el último valor de r2

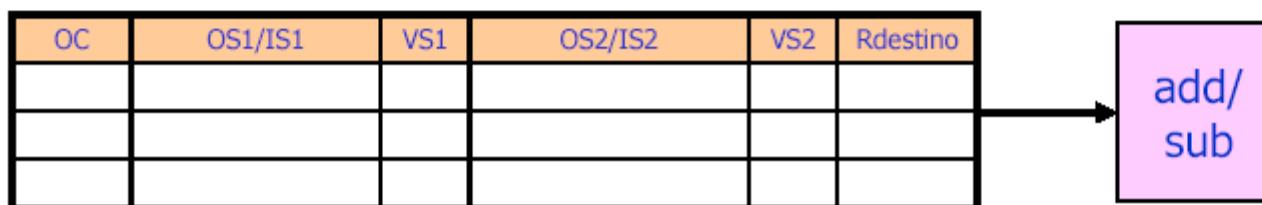
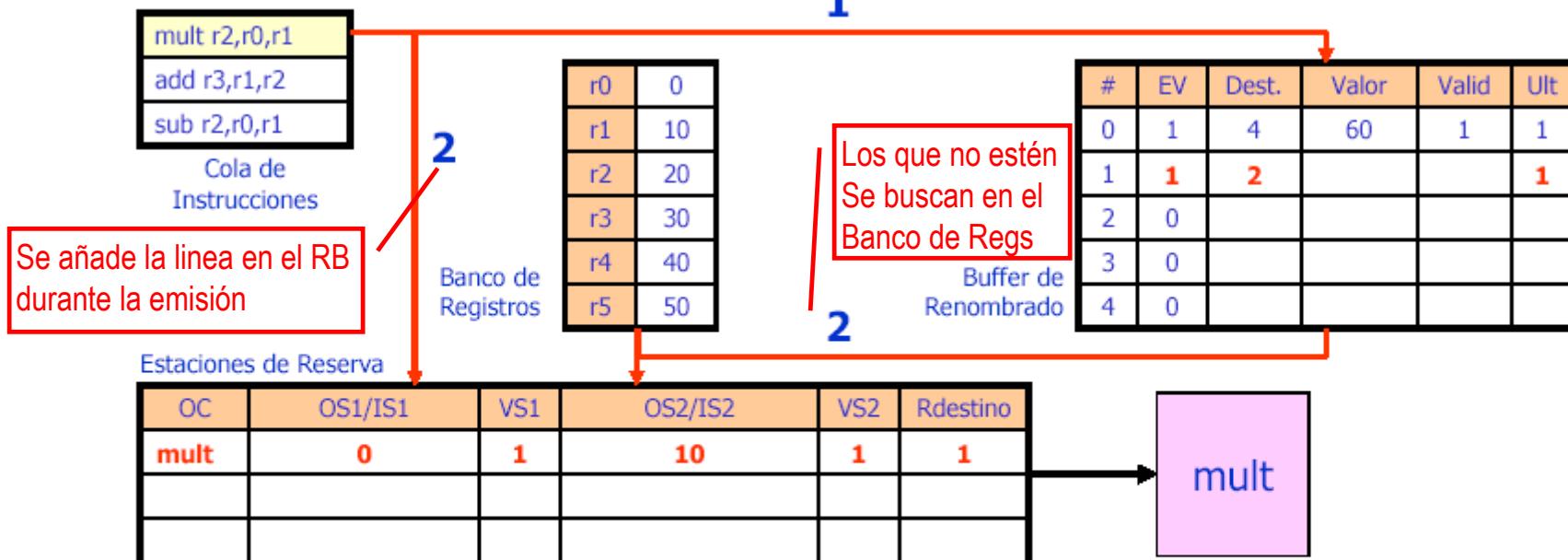
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva

Emisión de la multiplicación



Ingeniería de los Computadores

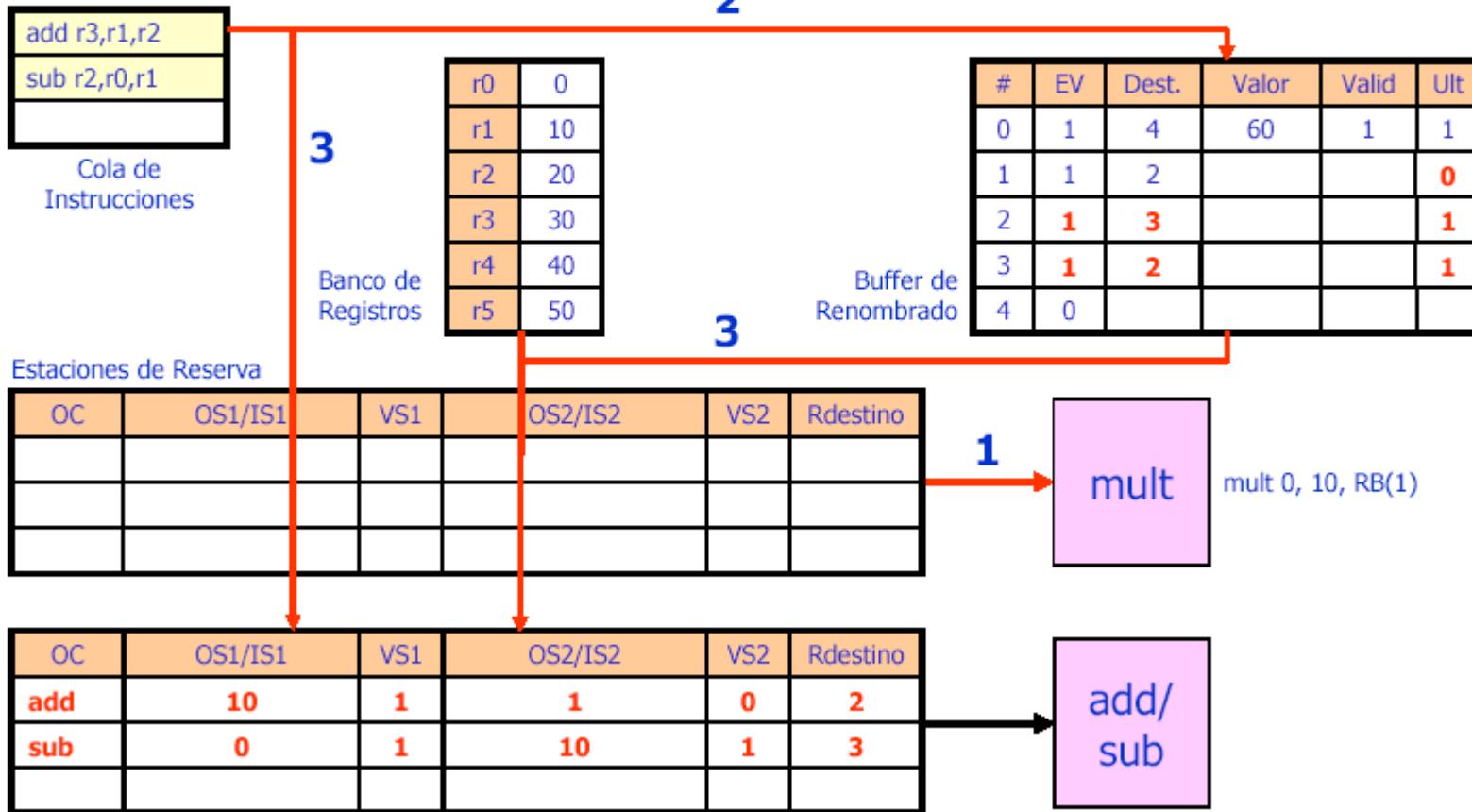
Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (II)

Envío de la multiplicación y emisión de las suma y la resta

2



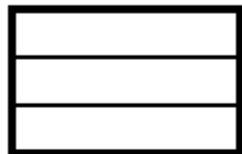
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (III)

Envío de la resta

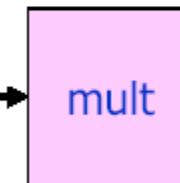


r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2			1
4	0				

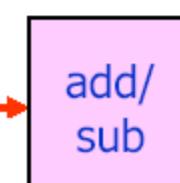
Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



sub 0, 10, RB(3)

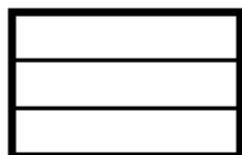
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (IV)

Termina la resta



Cola de
Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

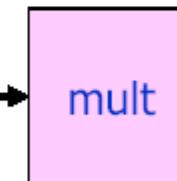
Banco de
Registros

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2	-10	1	1
4	0				

Buffer de
Renombrado

Estaciones de Reserva

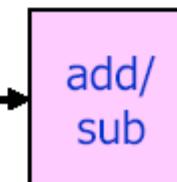
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



sub 0, 10, RB(3)

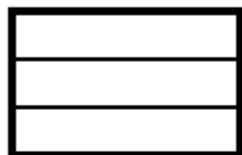
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (V)

Termina la multiplicación



Cola de
Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

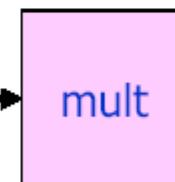
Banco de
Registros

Buffer de
Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva

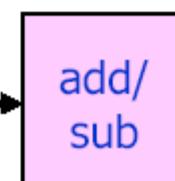
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	0	1	2



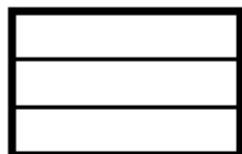
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VI)

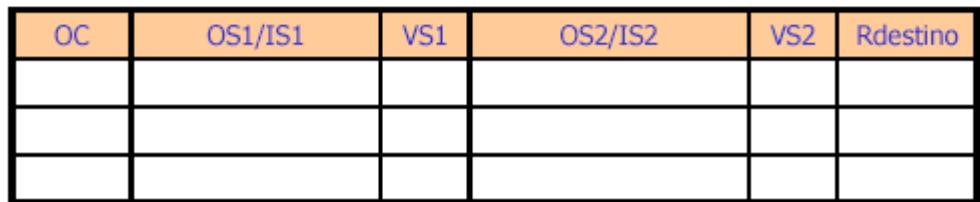
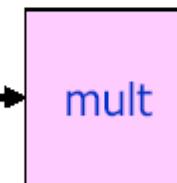
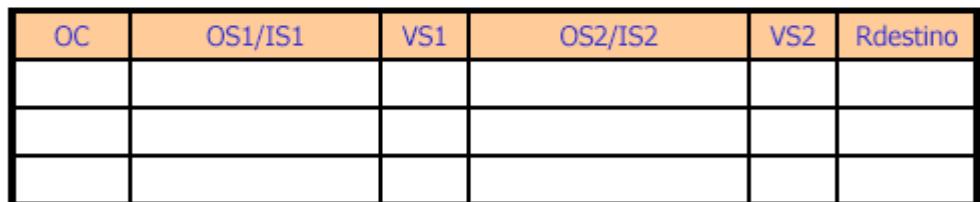
Envío de la suma



r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva



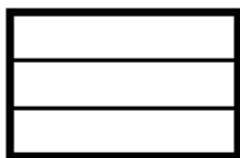
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Termina la suma



Cola de Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Banco de Registros

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3	10	1	1
3	1	2	-10	1	1
4	0				

Buffer de Renombrado

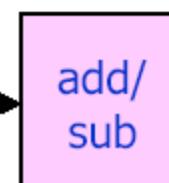
Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



add/
sub

add 10, 0, RB(2)

1

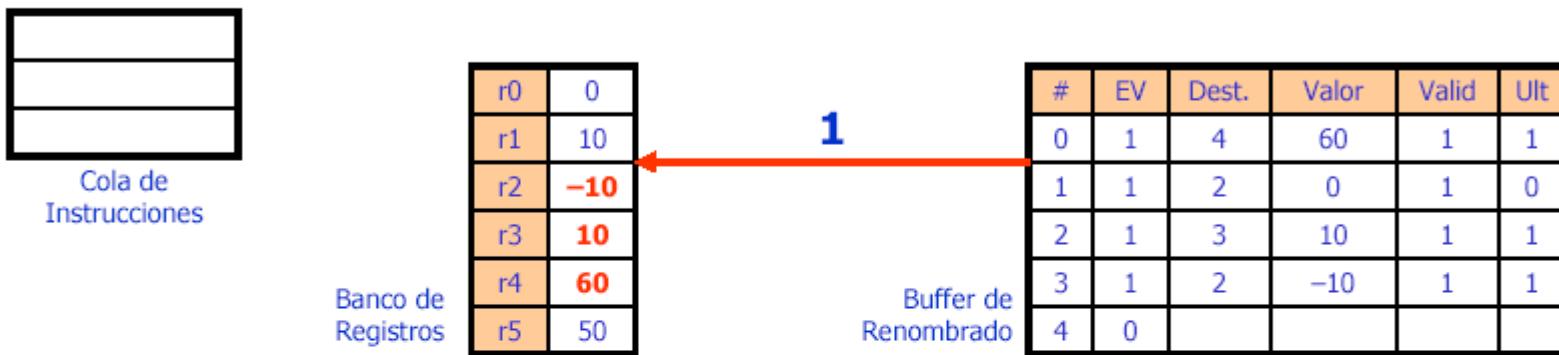
Ingeniería de los Computadores

Sesión 3. Superescalares

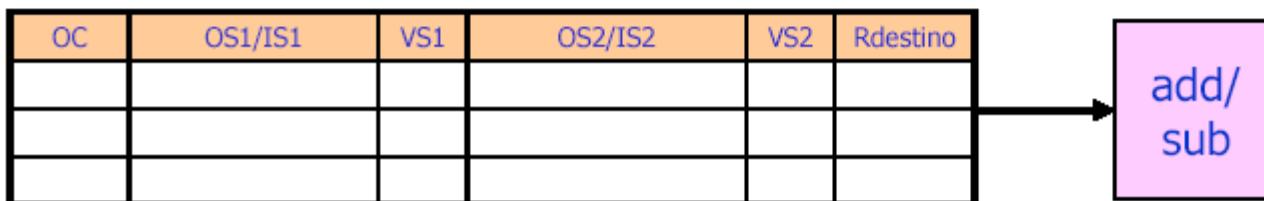
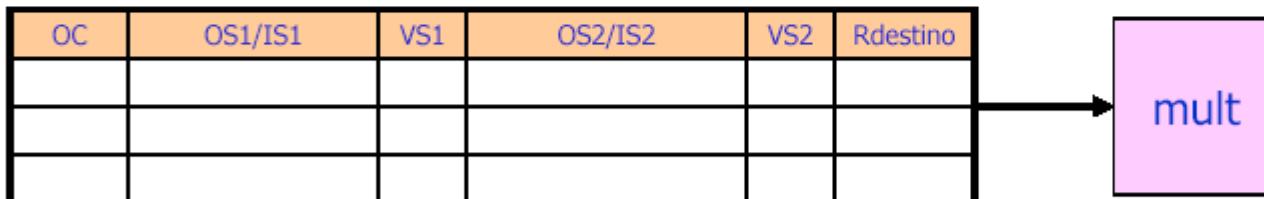
Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Se actualizan los registros



Estaciones de Reserva



Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

1				
2	instr(n)	f	
3	instr(n+1)	x	
4	instr(n+2)	f	
5	instr(n+3)	x	
6	instr(n+4)	x	
7	instr(n+5)	i	
8	instr(n+6)	i	
9				
10				

Cola
(Las instrucciones se retiran desde aquí)

Cabecera
(Las instrucciones que se decodifican se introducen a partir de aquí)

La gestión de interrupciones y la ejecución especulativa se realizan fácilmente mediante el ROB

- El puntero de cabecera apunta a la siguiente posición libre y el **puntero de cola a la siguiente instrucción a retirar**.
- Las instrucciones **se introducen en el ROB en orden de programa estricto** y pueden estar marcadas como **emitidas (issued, i)**, **en ejecución (x)**, o **finalizada su ejecución (f)**
- Las **instrucciones sólo se pueden retirar** (se produce la finalización con la escritura en los registros de la arquitectura) **si han finalizado**, y todas las que les preceden también.
- La **consistencia se mantiene porque sólo las instrucciones que se retiran del ROB se completan** (escriben en los registros de la arquitectura) y se retiran en el orden estricto de programa.

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

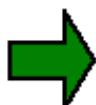
Ejemplo de Uso del Buffer de Reorden (I)

I1: mult r1, r2, r3

I2: st r1, 0x1ca

I3: add r1, r4, r3

I4: xor r1, r1, r3



Dependencias:

RAW: (I1,I2), (I3,I4)

WAR: (I2,I3), (I2,I4)

WAW: (I1,I3), (I1,I4), (I3,I4)

I1: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r2** y **r3**)

I2: Se envía a la unidad de almacenamiento hasta que esté disponible **r1**

I3: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r4** y **r3**)

I4: Se envía a la estación de reserva de la ALU para esperar a **r1**

Estación de Reserva (Unidad de Almacenamiento)

codop	dirección	op1	ok1
st	0x1ca	3	0

Estación de Reserva (ALU)

codop	dest	op1	ok1	op2	ok2
xor	6	5	0	[r3]	1

Líneas del ROB

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (II)

Ciclo 7

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	-	0	x	9
6	xor	10	r1	int_alu	-	0	i	-

Ciclo 9 No se puede retirar **add** aunque haya finalizado su ejecución

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	-	0	x	10

Ciclo 10 Termina **xor**, pero todavía no se puede retirar

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (III)

Ciclo 12

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	33	1	f	12
4	st	8	-	store	-	1	f	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Ciclo 13

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

- Se ha supuesto que se pueden retirar dos instrucciones por ciclo.
- Tras finalizar las instrucciones **mult** y **st** en el ciclo 12, se retirarán en el ciclo 13.
- Después, en el ciclo 14 se retirarán las instrucciones **add** y **xor**.

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

Problemas

1. Sea un computador superescalar capaz de decodificar 2inst/c, emitir 2inst/c, escribir 2inst/c en los registros correspondientes y retirar 2 inst/c. Dispone de buffer de reorden con número de entradas suficiente, la emisión (ejecución) puede ser desordenada y la finalización ordenada. El siguiente fragmento de programa aplica un filtro lineal sobre un vector A y almacena el resultado en un vector B:

```
for (i = 1; i < 100; i = i + 1) //99 iteraciones !!
{
    b[i] = coef*a[i-1] + a[i] ;
}
```

El compilador lo traduce al siguiente código:

```
; r1 almacena la dirección de a
; r2 almacena la dirección de b
addi r3,r1,#800 ; condicion de final
addi r1,r1,#8 ; inicialización de los indices
addi r2,r2,#8 ;
ld f0,coef ; cargar coeficiente
loop: ld f2,-8(r1) ; cargar a[i-1]
    ld f4,0(r1) ; cargar a[i]
    muld f8,f2,f0 ; a[i-1]*coef
    adddd f4,f8,f4 ; a[i-1]*coef + a[i]
    sd 0(r2),f4 ; almacenar b[i]
    addi r1,r1,#8 ; incrementar indices
    addi r2,r2,#8
    slt r4,r1,r3
    bnez r4,loop
```

Se dispone de las siguientes unidades segmentadas: 2 FP mul/div (4c), 2 FP add (2c), 2 ALU int (1) y 2 load/store (2). Se dispone de un predictor de saltos estático que predice como tomados los saltos hacia atrás y como no tomados los saltos hacia adelante.

- a) Planificar las instrucciones en la tabla 1

Tabla 1.

inst	IF	ID / ISS	EX	ROB	WB	renombrado en el ROB

- b) Desenrollar el bucle para obtener 3 copias del cuerpo (haciendo especial hincapié en **eliminar los cálculos redundantes**). Planificarlo asumiendo las latencias anteriores Nota: Las instrucciones de salto no pueden lanzarse a ejecución junto con una instrucción posterior a este.
- c) Calcular la ganancia de velocidad de b) respecto de a)

a) **(0,75) planificación sin errores 0,75 /errores menores 0,6 /errores mayores 0**

b) **(0,5) Desenrollado correcto 0,3 / planificación 0,2**

Se realizan 3 copias y se agrupan las instrucciones para aprovechar el paralelismo (0,1)

Se eliminan cargas redundantes, se ajustan los offset (0,1)

Se modifica la condición de finalización del bucle o los incrementos de punteros (0,1)

c) **(0,25) Correcto 0,25**

- a) Como no se dice nada se supone que la ventana de instrucciones tiene el tamaño suficientemente

a.1) En esta solución se tiene en cuenta que no hay adelantamientos ; los operandos se leen del ROB

inst	IF	ID/ISS	EX	RB	WB	comentario
addi r3,r1,#800	1	2	3 alui1	4	5	
addi r1,r1,#8	1	2	3 alui2	④	5	
addi r2,r2,#8	2	3	4 alui1	5	6	
ld f0,coef	2	3	4-5 ld1	6	7	
loop: ld f2,#-8(r1)	3	④	5-6 ld2	⑦	8	
ld f4,0(r1)	3	4-5	6-7 ld1	8	9	Ld ocupadas
muld f8,f2,f0	4	5-7	8-11	⑫	13	
addir f4,f8,f4	4	5-12	13-14	⑯	16	
sd 0(r2),f4	5	6-15	16-17	18	19	
addi r1,r1,#8	5	6	7 alui1	⑧	19	
addi r2,r2,#8	6	7	8 alui1	9	20	Se renombra r2
slt r4,r1,r3	6	7-8	9 alui1	⑩	20	
bnez r4,loop	7	8-10	11	12	21	Acierta la predicc.
sig	7	8-11	-			Se anula
loop: ld f2,#-8(r1)	8	9	10-11 ld1	12	21	
ld f4,0(r1)	8	9	10-12 ld2	12	22	

a.2) En esta solución se utilizan adelantamientos

inst	IF	ID/ISS	EX	RB	WB	comentario
addi r3,r1,#800	1	2	3 alui1	4	5	
addi r1,r1,#8	1	2	3 alui2	④	5	
addi r2,r2,#8	2	3	4 alui1	5	6	
ld f0,coef	2	3	4-5 ld1	6	7	
loop: ld f2,#-8(r1)	3	4	5-6 ld2	⑦	8	
ld f4,0(r1)	3	4-5	6-7 ld1	8	9	Ld ocupadas
muld f8,f2,f0	4	5-6	7-10	⑪	12	
addir f4,f8,f4	4	5-10	11-12	⑬	14	
sd 0(r2),f4	5	6-12	13-14	15	16	
addi r1,r1,#8	5	6	7 alui1	⑧	16	
addi r2,r2,#8	6	7	8 alui1	9	17	Se renombra r2
slt r4,r1,r3	6	7	8 alui2	⑨	17	
bnez r4,loop	7	8	9	10	18	Acierta la predicc.
sig	7	8--	--	--	--	Se anula
loop: ld f2,#-8(r1)	8	9	10-11 ld1	12	20	
ld f4,0(r1)	8	9	10-12 ld2	12	20	

a.3) En esta solución, ventana de instrucciones de 2 entradas y alineada (adelantamientos)

inst	IF	ID / ISS	EX	RB	WB	comentario
addi r3,r1,#800	1	2	3 alui1	4	5	
addi r1,r1,#8	1	2	3 alui2	4	5	
addi r2,r2,#8	2	3	4 alui1	5	6	
ld f0,coef	2	3	4-5 ld1	6	7	
loop: ld f2,#-8(r1)	3	4	5-6 ld2	7	8	
ld f4,0(r1)	3	4-5	6-7 ld1	8	9	Ld ocupadas
muld f8,f2,f0	4-5	6	7-10	11	12	Ventana alineada
addir f4,f8,f4	4-5	6-10	11-12	13	14	Ventana alineada
sd 0(r2),f4	5-10	11-12	13-14	15	16	
addi r1,r1,#8	5-10	11	12 alui1	13	16	
addi r2,r2,#8	6-12	13	14 alui1	15	17	Se renombra r2 en RB
slt r4,r1,r3	6-12	13	14 alui2	15	17	
bnez r4,loop	7-13	14	15	16	18	Acierta la predicc.
sig	7-13	14--	--	--	--	Se anula
loop: ld f2,#-8(r1)	8					
ld f4,0(r1)	8					

b) Desenrollamos el bucle 2 veces.

```

addi r3,r1,#800 ; condición modificada(800-3*8)
addi r1,r1,#8 ; inicialización de los indices
addi r2,r2,#8 ;
loop: ld f0,coef ; cargar coeficiente
      ld f2,-8(r1) ; cargar a[i-1]
      ld f4,0(r1) ; cargar a[i]
      ld f5,8(r1); cargar a[i+1]
      ld f6,16(r1); cargar a[i+2]
      muld f8,f2,f0 ; a[i-1]*coef
      muld f9,f4,f0 ; a[i]*coef
      muld f10,f5,f0 ; a[i+1]*coef
      addd f4,f8,f4 ; a[i-1]*coef + a[i]
      addd f5,f9,f5 ; a[i]*coef + a[i+1]
      addd f6,f10,f6 ; a[i+1]*coef + a[i+2]
      sd 0(r2),f4 ; almacenar b[i]
      sd 8(r2),f5 ; almacenar b[i+1]
      sd 16(r2),f6 ; almacenar b[i+2]

      addi r1,r1,#24 ; incrementar indices
      addi r2,r2,#24
      slt r4,r1,r3
      bnez r4,loop
    
```

inst	IF	ID/ISS	EX	RB	WB	comentario
addi r3,r1,#800	1	2	3 alui1	4	5	alui1
addi r1,r1,#8	1	2	3 alui2	④	5	alui2
addi r2,r2,#8	2	3	4 alui1	5	6	alui1
ld f0,coef	2	3	4-5 ld1	6	7	ld1
loop: ld f2,-8(r1)	3	4	5-6 ld2	⑦	8	ld2
ld f4,0(r1)	3	4-5	6-7 ld1	⑧	9	Ld1
ld f5,8(r1)	4	5-6	7-8 ld2	9	10	ld2
ld f6,16(r1)	4	5-7	8-9 ld1	10	11	Ld1
muld f8,f2,f0	5	6	7-10 mull	⑪	12	Mul1
muld f9,f4,f0	5	6-7	8-11 mul2	⑫	13	Mul2
muld f10,f5,f0	6	7-10	11-14 mull	⑮	16	Mul ocupadas mull
addd f4,f8,f4	6	7-10	11-12 sum1	⑬	16	sum1
addd f5,f9,f5	7	8-11	12-13 sum2	⑭	17	Sum2
addd f6,f10,f6	7	8-14	15-16 sum1	⑯	18	sum1
sd 0(r2),f4	8	9-12	13-14 sd1	15	18	Sd1
sd 8(r2),f5	8	9-13	14-15 sd2	16	19	Sd2
sd 16(r2),f6	9	10-16	17-18 sd1	19	20	Sd1
addi r1,r1,#24	9	10	11 alui1	⑯	20	Alu1
addi r2,r2,#24	10	11	12 alui2	13	21	Alu2
slt r4,r1,r3	10	11	12 alui1	⑯	21	Alu1 / solo 2 en ROB
bnez r4,loop	11	11-13	14	15	22	Acierta la predicc.
--	11	--				
loop	12					

Ingeniería de los Computadores

Sesión 5. Riesgos de control

Ingeniería de los Computadores

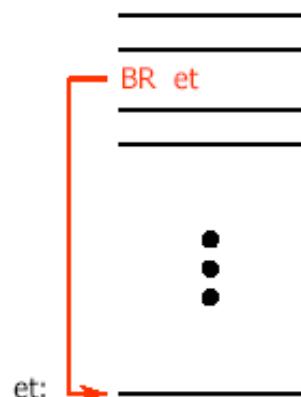
Sesión 5. Superescalares

Riesgos

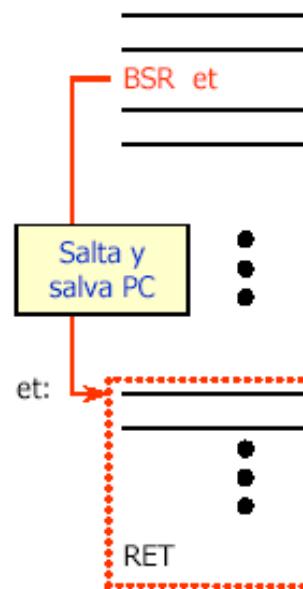
Clasificación de los Saltos

Saltos Incondicionales

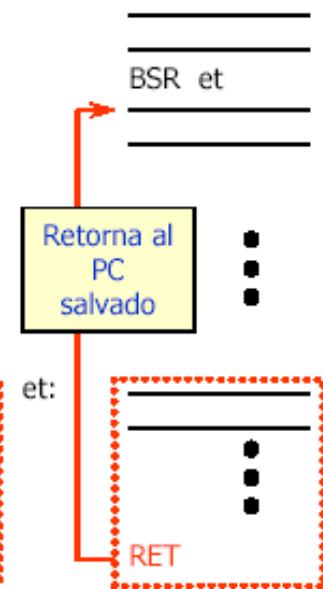
Salto Incondicional



Llamada a Subrutina

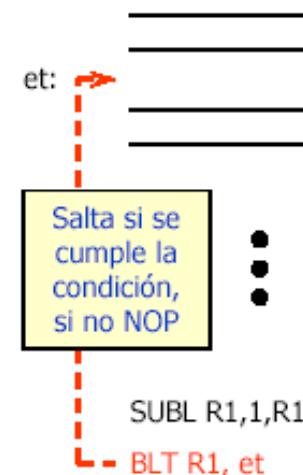


Retorno de Subrutina

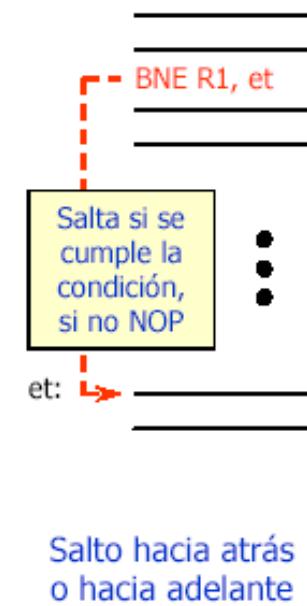


Saltos Condicionales

Condición de Bucle



Otro Salto Condicional



Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

El efecto de los saltos en los procesadores superescalares es más pernicioso ya que, al emitirse varias instrucciones por ciclo, prácticamente en cada ciclo puede haber una instrucción de salto.

El salto retardado no tiene mucho interés porque la unidad de emisión decide las instrucciones que pasan a ejecutarse teniendo en cuenta las dependencias.

- **Detección de la Instrucción de Salto**

Cuanto antes se detecte que una instrucción es de salto menor será la posible penalización. Los saltos se detectan usualmente en la fase de decodificación.

- **Gestión de los Saltos Condicionales no Resueltos**

Si en el momento en que la instrucción de salto evalúa la condición de salto ésta no se haya disponible se dice que *el salto o la condición no se ha resuelto*. Para resolver este problema se suele utilizar el **procesamiento especulativo del salto**.

- **Acceso a las Instrucciones destino del Salto**

Hay que determinar la forma de acceder a la secuencia a la que se produce el salto

- **La implementación física de las Unidades de Salto**

Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

- Detección

- En etapa de decodificación



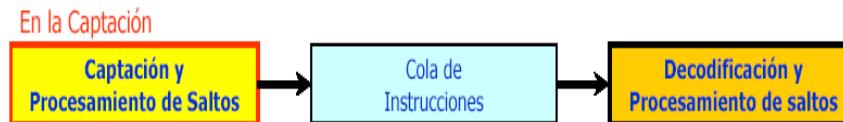
- Detección temprana ('early branch detection')
 - Detección paralela a decodificación



- Detección anticipada ('look-ahead branch detection')



- Detección integrada en captación



Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

Gestión

Uso de los ciclos que siguen a la inst. de salto condicional	Salto Retardado	Se utilizan los ciclos que siguen a la captación de una instrucción de salto para insertar instrucciones que deben ejecutarse independientemente del resultado del salto (Primeras arquitecturas RISC y posteriores)
Gestión de Saltos Condicionales no Resueltos (Una condición de salto no se puede comprobar si no se ha terminado de evaluar)	Bloqueo del Procesamiento del Salto	Se bloquea la instrucción de salto hasta que la condición esté disponible (68020, 68030, 80386)
	Procesamiento Especulativo de los Saltos	La ejecución prosigue por el camino más probable (se especula sobre las instrucciones que se ejecutarán). Si se ha errado en la predicción hay que recuperar el camino correcto. (Típica en los procesadores superescalares actuales)
	Múltiples Caminos	Se ejecutan los dos caminos posibles después de un salto hasta que la condición de salto se evalúa. En ese momento se cancela el camino incorrecto. (Máquinas VLIW experimentales: Trace/500 , URPR-2)
Evitar saltos condicionales	Ejecución Vigilada (<i>Guarded Exec.</i>)	Se evitan los saltos condicionales incluyendo en la arquitectura instrucciones con operaciones condicionales (IBM VLIW, Cydra-5, Pentium, HP PA, Dec Alpha)

Esquemas de Predicción de Salto

Predicción Fija

Se toma siempre la misma decisión: el salto siempre se realiza, '*taken*', o no, '*not taken*'

Predicción Verdadera

La decisión de si se realiza o no se realiza el salto se toma mediante:

- **Predicción Estática:**

Según los atributos de la instrucción de salto (el código de operación, el desplazamiento, la decisión del compilador)

- **Predicción Dinámica:**

Según el resultado de ejecuciones pasadas de la instrucción (historia de la instrucción de salto)



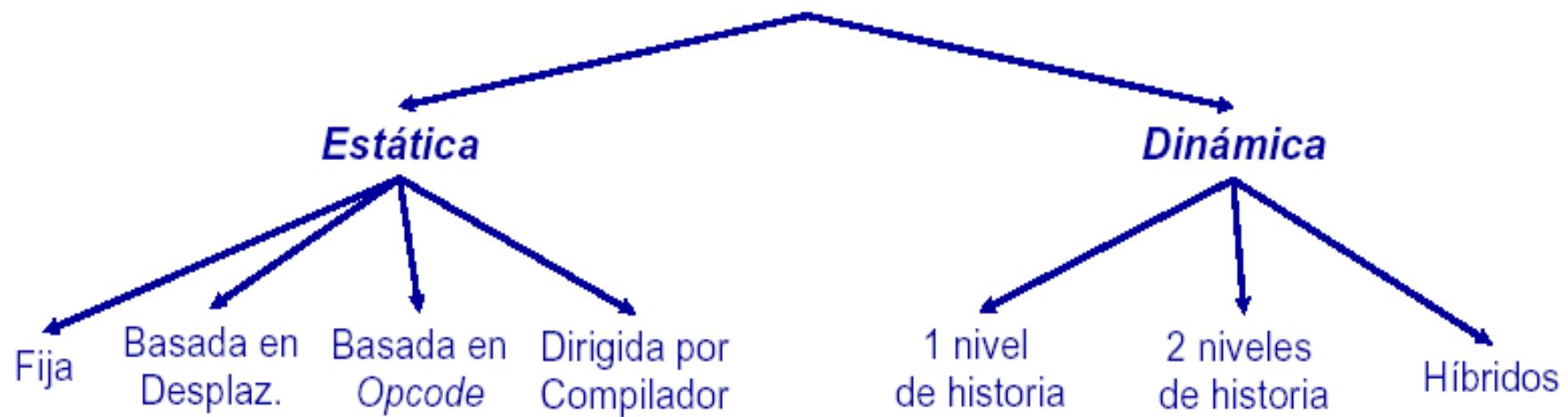
Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

Gestión

Predicción de saltos



Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

Gestión

- Predicción fija

Aproximación 'Siempre No Tomado'

- Toda condición de salto no resuelta se predice que no da lugar a un salto
- Se continúa la ejecución por donde iba aunque se puede adelantar algo el procesamiento de la secuencia de salto (calculo de la dirección de salto, BTA)
- Cuando se evalúa la condición se comprueba si la predicción era buena.
- Si la predicción era buena el procesamiento continúa y se borra la BTA, y si era mala se abandona el procesamiento de la secuencia predicha (no se considera su efecto) y se captan instrucciones a partir de la BTA

* Es más fácil de implementar que la aproximación de 'Siempre Tomado'

SuperSparc (1992)
(TP: 1; NTP: 0)

Alpha21064
Power I (1990)
(TP: 3; NTP: 0)

Power 2 (1993)
(TP: 1; NTP: 0)

Aproximación 'Siempre Tomado'

- Toda condición de salto no resuelta se predice que da lugar a un salto
- En previsión de error de predicción se salva el estado de procesamiento actual (PC) y se empieza la ejecución a partir de la dirección de salto.
- Cuando se evalúa la condición de salto se comprueba si la predicción era buena
- Si la predicción es correcta se continúa, y si es errónea se recupera el estado almacenado y no se considera el procesamiento de la secuencia errónea

MC68040 (1999)
(TP: 1; NTP: 2)

* Necesita una implementación más compleja que la aproximación anterior aunque suele proporcionar mejores prestaciones

Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

Gestión

- Predicción estática

Predicción basada en el Código de Operación

Para ciertos códigos de operación (ciertos saltos condicionales específicos) se predice que el salto se toma, y para otros que el salto no se toma

MC88110 (93)
PowerPC 603(93)

Ejemplo: Predicción Estática en el MC88110

Formato	Instrucción		Predicción
	Condición Especificada	Bit 21 de la Instr.	
bcnd <i>(Branch Conditional)</i>	$\neq 0$	1	Tomado
	$= 0$	0	No Tomado
	> 0	1	Tomado
	< 0	0	No Tomado
	≥ 0	1	Tomado
	≤ 0	0	No Tomado
	bb1 <i>(Branch on Bit Set)</i>		Tomado
	bb0 <i>(Branch on Bit Clear)</i>		No Tomado

Ingeniería de los Computadores

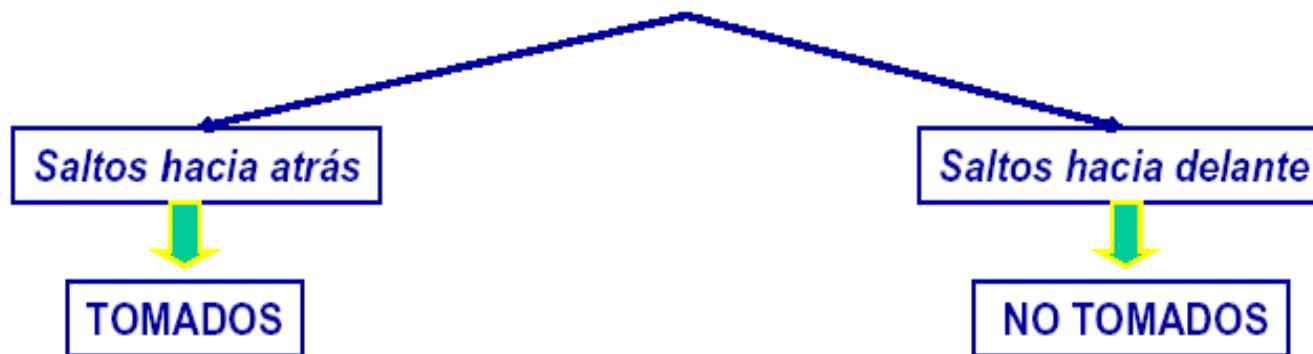
Sesión 5. Superescalares

Riesgos

Gestión

- Predicción estática

Predicción basada en la DIRECCIÓN del salto



La mayoría de saltos hacia atrás corresponden a bucles

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

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

EJEMPLOS

- Alpha 21064 (1992) (Opción seleccionable)
- PowerPC 601/603 (1993)

Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

Gestión

- Predicción estática



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

EJEMPLOS

- AT&T 9210 Hobbit (1993)
- PowerPC 601/603 (1993)
- PA 8000 (1996)

- Predicción dinámica

- La predicción para cada instrucción de salto puede cambiar cada vez que se va a ejecutar ésta según la historia previa de saltos tomados/no-tomados para dicha instrucción.
- El presupuesto básico de la predicción dinámica es que es más probable que el resultado de una instrucción de salto sea similar al que se tuvo en la última (o en las n últimas ejecuciones)
- Presenta mejores prestaciones de predicción, aunque su implementación es más costosa

- **Predicción Dinámica Implícita**

No hay bits de historia propiamente dichos sino que se almacena la dirección de la instrucción que se ejecutó después de la instrucción de salto en cuestión

- **Predicción Dinámica Explícita**

Para cada instrucción de salto existen unos bits específicos que codifican la información de historia de dicha instrucción de salto

Ingeniería de los Computadores

Sesión 5. Superescalares

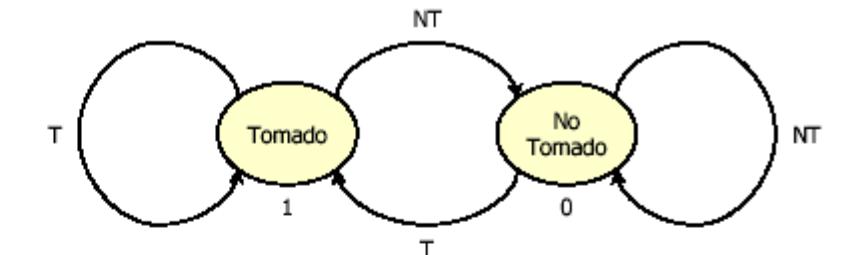
Riesgos

Gestión

- Predicción dinámica explícita

Predicción con 1 bit de historia

La designación del estado, Tomado (1) o No Tomado (0), indica lo que se predice, y las flechas indican las transiciones de estado según lo que se produce al ejecutarse la instrucción (T o NT)

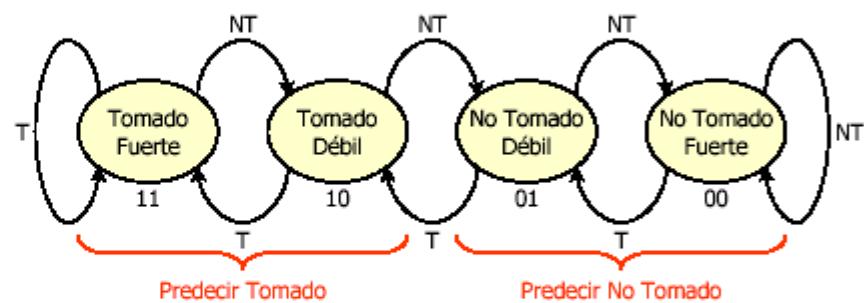


Predicción con 2 bits de historia

Existen cuatro posibles estados. Dos para predecir Tomado y otros dos para No Tomado

La primera vez que se ejecuta un salto se inicializa el estado con predicción estática, por ejemplo 11

Las flechas indican las transiciones de estado según lo que se produce al ejecutarse la instrucción (T o NT)

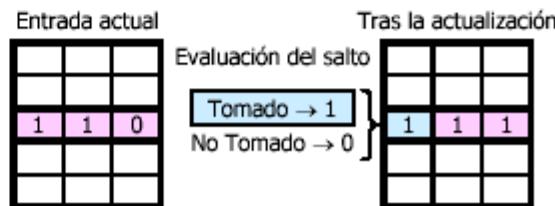


Predicción con 3 bits de historia

Cada entrada guarda las últimas ejecuciones del salto

Se predice según el bit mayoritario (por ejemplo, si hay mayoría de unos en una entrada se predice salto)

La actualización se realiza en modo FIFO, los bits se desplazan, introduciéndose un 0 o un 1 según el resultado final de la instrucción de salto



- Predicción dinámica explícita

Implementación de los Bits de Historia

• En la Cache de Instrucciones	Alpha 21064 (92) (2k x 1 bit) Alpha 21064A (94) (4k x 2 bits) UltraSparc (92) (2k x 2 bits)
• En una Tabla de Historia de Salto (BHT)	PA8000(96) (253 x 3bits) PowerPC 620(95) (2K x 2bits) R10000 (96) (512 x 2bits)
• En una Cache para las Instrucciones a las que se produce el Salto (BTIC) o, • En una Cache para las Direcciones a las que se produce el Salto (BTAC)	Pentium (94) (256 x 2 bits) (BTAC) MC 68069 (93) (256 x 2bits) (BTAC)

- Predicción dinámica explícita

1) Branch Target Buffer (BTB): bits acoplados

La BTB almacena

- La dirección destino de los últimos saltos tomados
- Los bits de predicción de ese salto

Actualización de la BTB

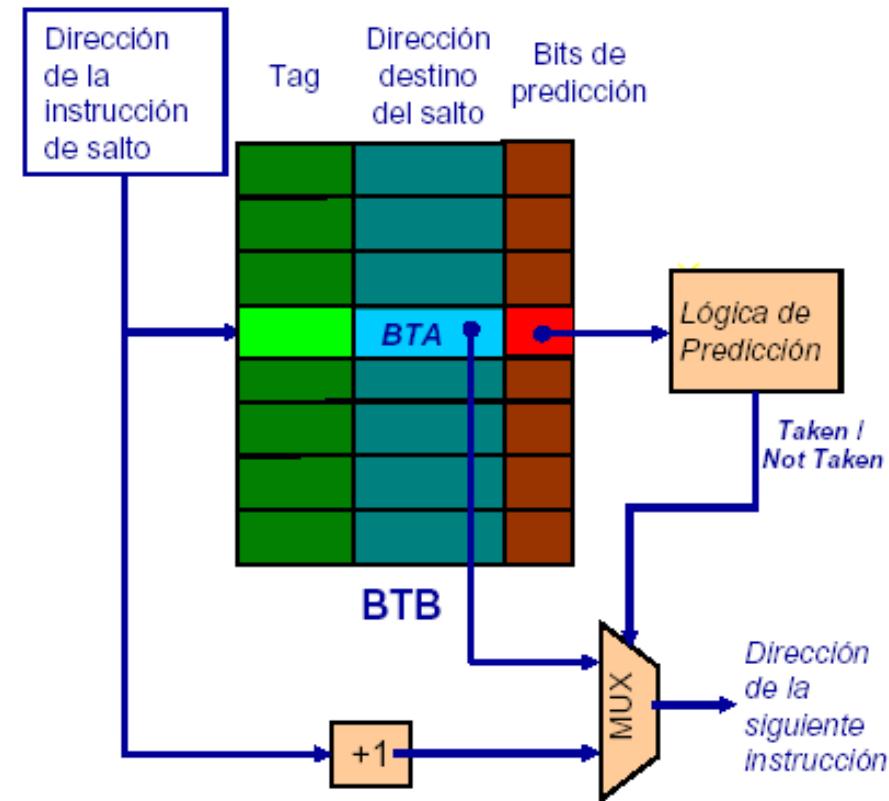
Los campos de la BTB se actualizan después de ejecutar el salto, cuando se conoce:

- Si el salto fue tomado o no \Rightarrow Actualizar bits de predicción
- La dirección destino del salto \Rightarrow Actualizar BTA

Predicción Implícita (sin bits de predicción)

Aplicable con un sólo bit de predicción

- Sí la instrucción de salto está en la BTB
 \Rightarrow El salto se predice como tomado
- Sí la instrucción de salto no está en la BTB
 \Rightarrow El salto se predice como no tomado



DESVENTAJA: Sólo se pueden predecir aquellas instrucciones de salto que están en la BTB

Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

Gestión

- Predicción dinámica explícita

2) Tabla de historia de saltos (BHT): bits desacoplados

Existen dos tablas distintas:

- La BTAC, que almacena la dirección destino de los últimos saltos tomados
- La BHT, que almacena los bits de predicción de todas las instrucciones de salto condicional

Ventaja

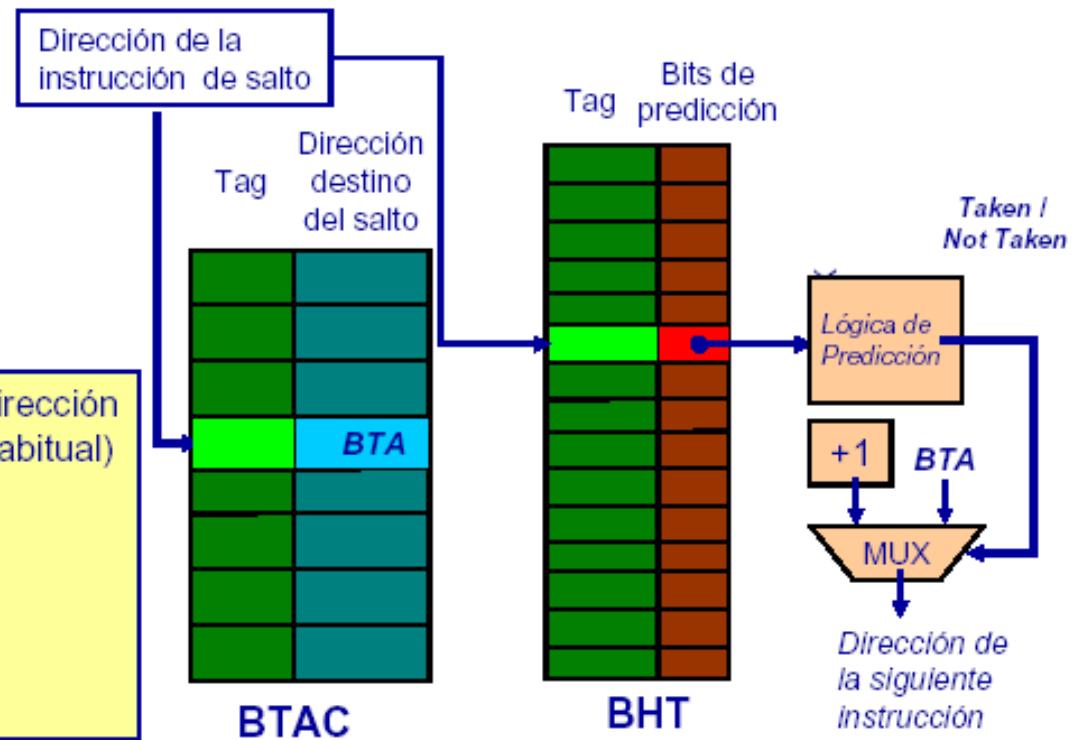
Puede predecir instruc. que no están en la BTAC (más entradas en BHT que en BTAC)

Desventaja

Aumenta el hardware necesario
⇒ 2 tablas asociativas

Acceso a la BHT

- Usando los bits menos significativos de la dirección
 - Sin TAGs ⇒ Menor coste (opción + habitual)
 - Compartición de entradas
 - ⇒ Se degrada el rendimiento
- Asociativa por conjuntos
 - Mayor coste ⇒ Tablas pequeñas
 - Para un mismo coste hardware
 - ⇒ Peor comportamiento



- Predicción dinámica explícita

3) Bits de predicción en la I-cache

Funcionamiento

Cuando se capta la instrucción de la cache
Si se trata de una instrucción de salto condicional

- Se accede en paralelo a los bits de predicción
- Si el salto se predice como tomado se accede a la instrucción destino del salto

Acceso a la instrucción destino del salto

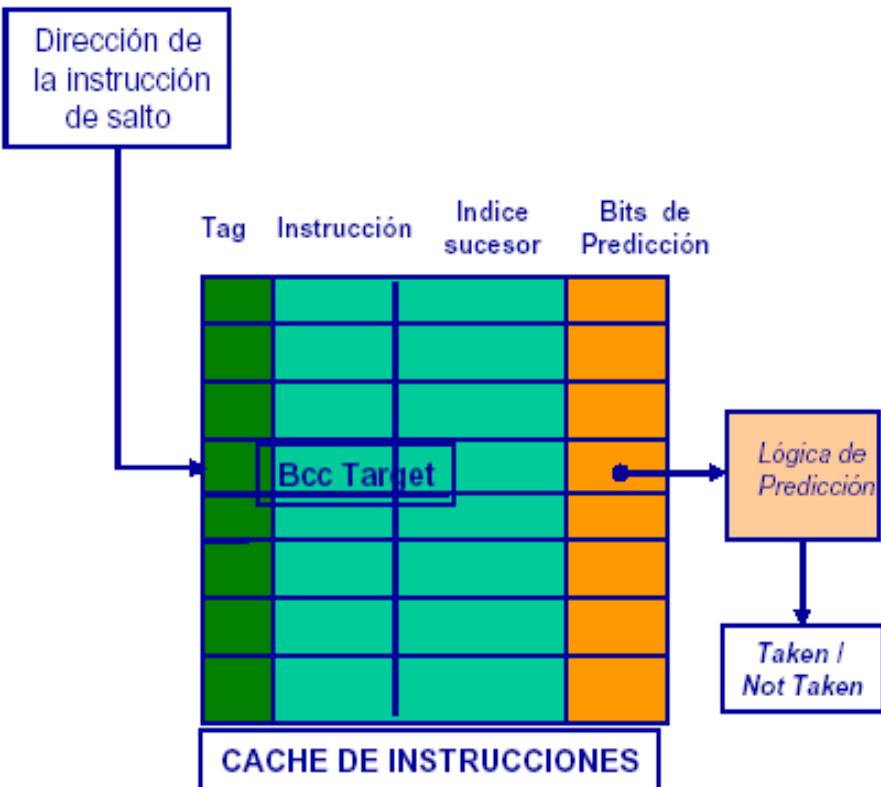
- BTB independiente
- Añadir índice sucesor a la I-cache

Alternativas de diseño

- Bits de predicción por cada instrucción de la cache
- Bits de predicción por cada línea de cache

Ventajas

- Puede predecir instrucciones que no están en la BTB
- No añade una cantidad extra de hardware excesiva



Ingeniería de los Computadores

Sesión 5. Superescalares

Riesgos

Gestión

Extensión del Procesamiento Especulativo

- Tras la predicción, el procesador continúa ejecutando instrucciones especulativamente hasta que se resuelve la condición.
- El intervalo de tiempo entre el comienzo de la ejecución especulativa y la resolución de la condición puede variar considerablemente y ser bastante largo.
- En los procesadores superescalares, que pueden emitir varias instrucciones por ciclo, pueden aparecer más instrucciones de salto condicional no resueltas durante la ejecución especulativa.
- Si el número de instrucciones que se ejecutan especulativamente es muy elevado y la predicción es incorrecta, la penalización es mayor.

Así, cuanto mejor es el esquema de predicción mayor puede ser el número de instrucciones ejecutadas especulativamente.



- **Nivel de Especulación:** Número de Instrucciones de Salto Condicional sucesivas que pueden ejecutarse especulativamente (si se permiten varias, hay que guardar varios estados de ejecución). Ejemplos: Alpha21064, PowerPC 603 (1); Power 2 (2); PowerPC 620 (4); Alpha 21164 (6)
- **Grado de Especulación:** Hasta qué etapa se ejecutan las instrucciones que siguen en un camino especulativo después de un salto. Ejemplos: Power 1 (Captación); PowerPC 601 (Captación, Decodificación, Envío); PowerPC 603 (Todas menos la finalización)

Ingeniería de los Computadores

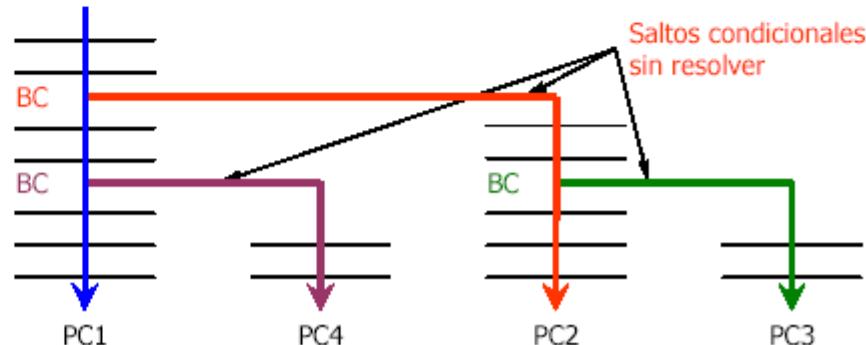
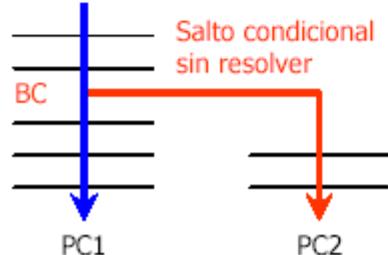
Sesión 5. Superescalares

Riesgos

Gestión

- Ramificación multicamino

- Se siguen las dos secuencias de instrucciones que aparecen a partir de una instrucción de salto (la correspondiente al salto efectuado y al salto no efectuado). Una vez resuelto el salto la secuencia incorrecta se abandona
- Para implementar esta técnica hacen falta varios contadores de programa
- Se necesitan gran cantidad de recursos hardware y el proceso para descartar las instrucciones que se han ejecutado incorrectamente es complejo



- Ejecución condicional

Instrucciones de Ejecución Condicional (*Guarded Execution*)

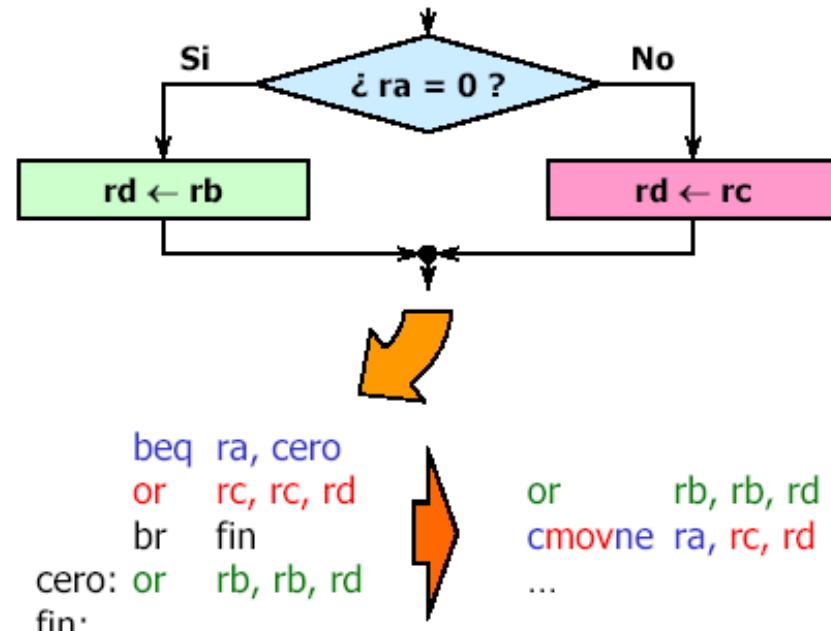
- Se pretende **reducir el número de instrucciones de salto** incluyendo en el **repertorio máquina instrucciones con operaciones condicionales** ('conditional operate instructions' o 'guarded instructions')
- Estas instrucciones tienen dos partes: la **condición** (denominada *guardia*) y la parte de **operación**

Ejemplo: cmovxx de Alpha

cmovxx ra.rq, rb.rq, rc.wq

- xx** es una condición
- ra.rq, rb.rq** enteros de 64 bits en registros ra y rb
- rc.wq** entero de 64 bits en rc para escritura
- El registro ra se comprueba en relación a la condición xx y si se verifica la condición rb se copia en rc.

Sparc V9, HP PA, y Pentium ofrecen también estas instrucciones.



Ingeniería de los Computadores

Sesión 5. Superescalares

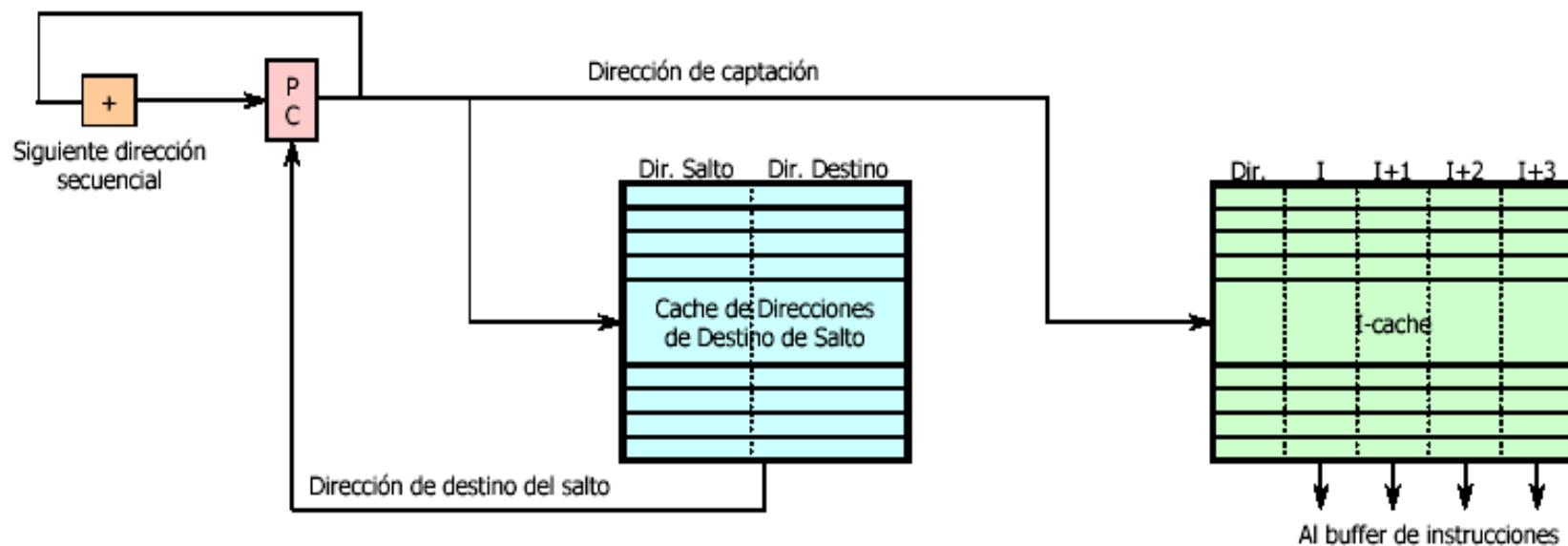
Riesgos

Gestión

Estructuras

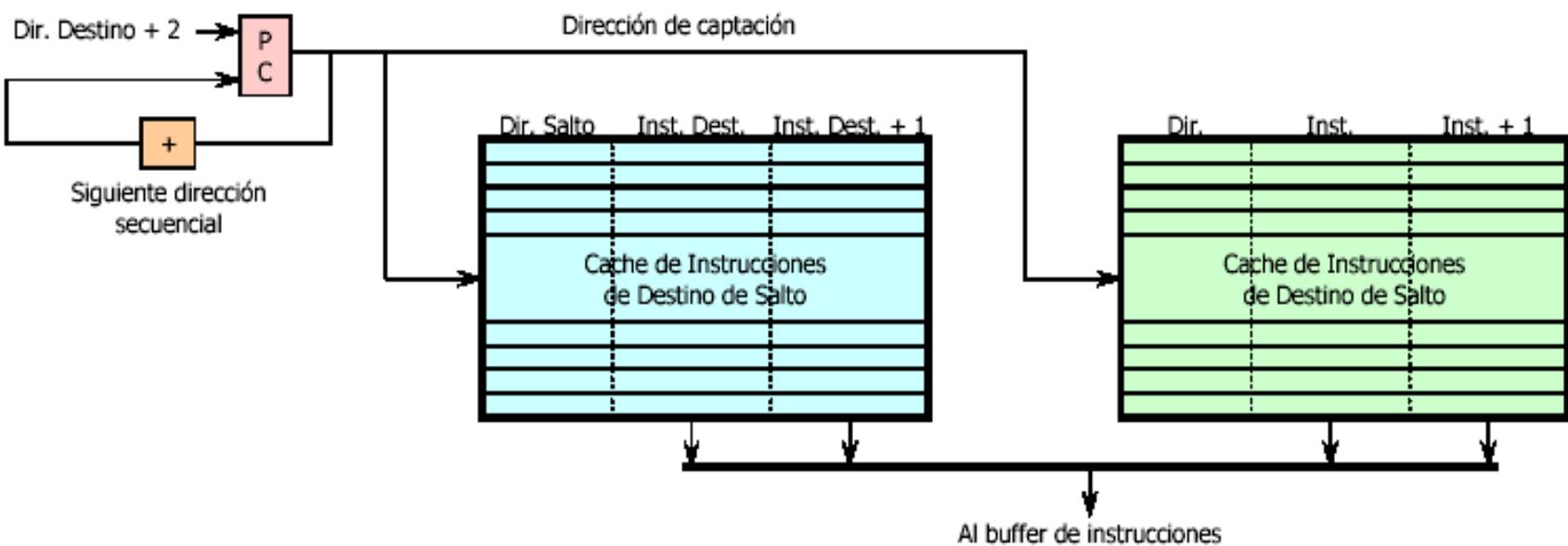
Esquema de cache de direcciones de destino de salto (BTAC)

- Se añade una cache que contiene las direcciones de las instrucciones destino de los saltos, junto con las direcciones de las instrucciones de salto.
- Se leen las direcciones al mismo tiempo que se captan las instrucciones de salto.



Esquema de cache de instrucciones de destino de salto (BTIC)

- Se añade una cache que contiene las instrucciones siguientes a la dirección de destino de los saltos, junto con las direcciones de las instrucciones de salto.
- Sólo tiene sentido si la cache de instrucciones tiene una latencia muy alta.
- Mientras se procesan estas instrucciones se calcula la dirección de las siguientes.



Ingeniería de los Computadores

Sesión 5. Superescalares

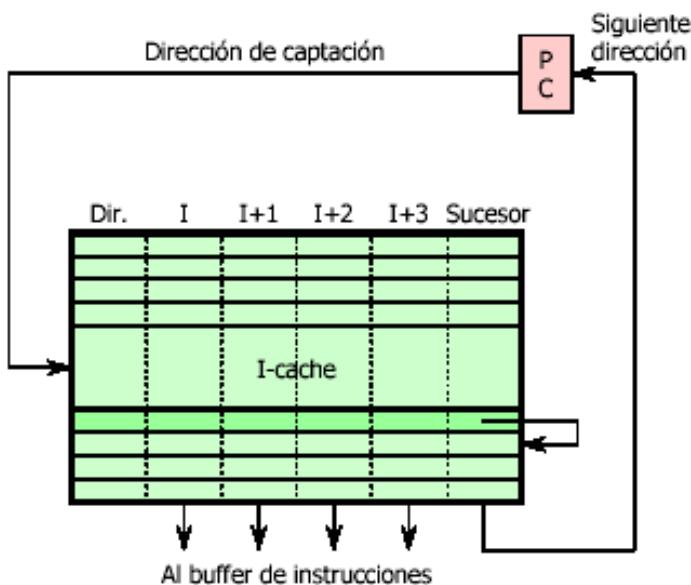
Riesgos

Gestión

Estructuras

Esquema de índice sucesor en la cache de instrucciones

- La cache de instrucciones contiene un índice sucesor que apunta a la siguiente línea de la cache de instrucciones que hay que captar (la siguiente, o la que se predice que se debe captar si hay una instrucción de salto condicional en esa línea).



Ejemplos y evolución de los esquemas de Acceso

Calcular/captar	BTIC	BTAC	Índice sucesor
i486 (1989)	→	Pentium (1993)	
MC68040 (1990)	→	MC 68060 (1993)	
		Am 29000 (1988) →	Am 29000 superscalar (1995)
Sparc CYC 600 (1992) SuperSparc (1992)		→	UltraSparc (1995)
R4000 (1992) R10000 (1996)		→	R8000 (1994)
PowerPC 601 (1993) PowerPC 603 (1993)	→	PowerPC 604 (1995) PowerPC 620 (1996)	

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

- Técnica para **evitar el efecto de las dependencias WAR, o Antidependencias** (en la emisión desordenada) y **WAW, o Dependencias de Salida** (en la ejecución desordenada).

```
R3 := R3 - R5  
R4 := R3 + 1  
R3 := R5 + 1  
R7 := R3 * R4
```

Cada escritura se asigna
a un registro físico distinto

```
R3b := R3a - R5a  
R4a := R3b + 1  
R3c := R5a + 1  
R7a := R3c * R4a
```

Sólo RAW

R.M. Tomasulo (67)

Implementación Estática: Durante la Compilación

Implementación Dinámica: Durante la Ejecución (circuitería adicional y registros extra)

Características de los Buffers de Renombrado

- **Tipos de Buffers** (separados o mezclados con los registros de la arquitectura)
- **Número de Buffers de Renombrado**
- **Mecanismos para acceder a los Buffers** (asociativos o Indexados)

Velocidad del Renombrado

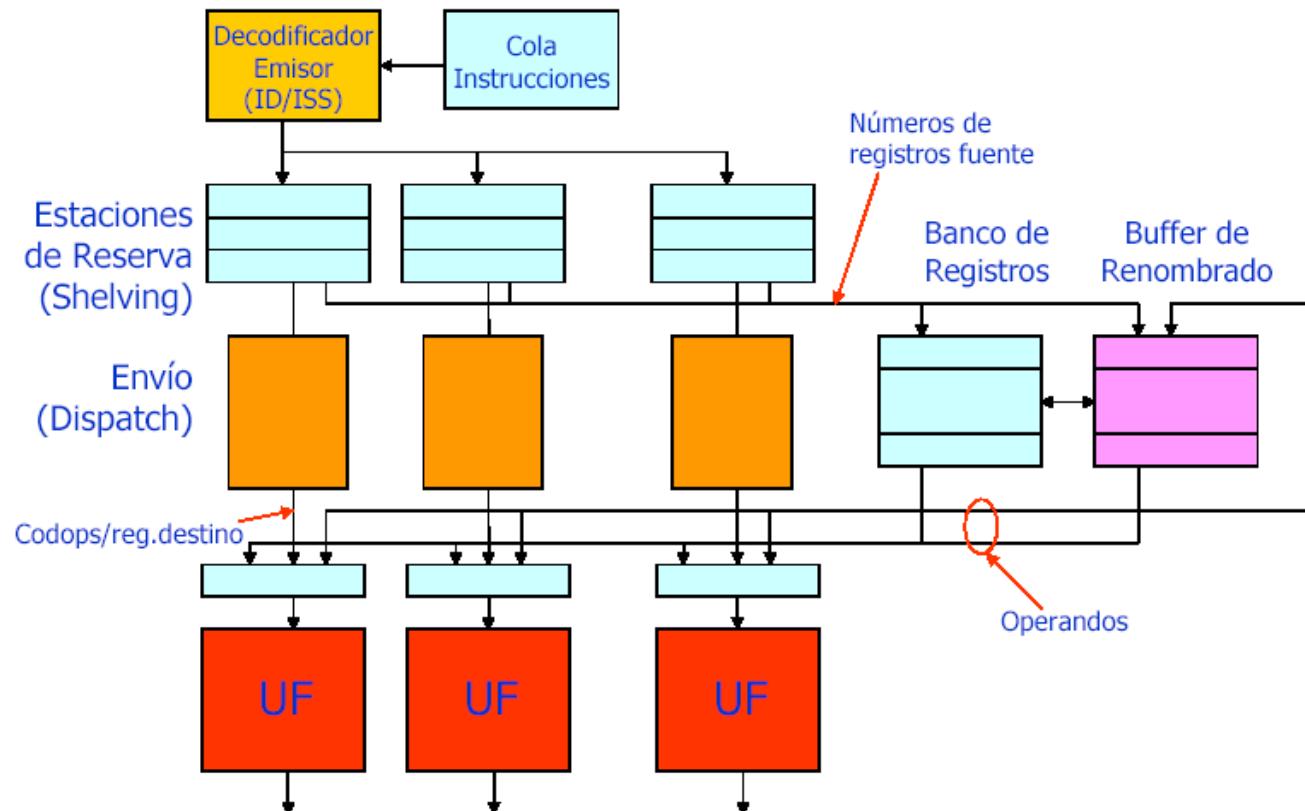
- **Máximo número de nombres asignados por ciclo** que admite el procesador

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

- Renombrado de registros
 - Estaciones de reserva + buffer de renombrado



Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

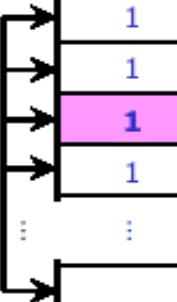
- Tipos de buffers de renombrado

Buffer de Renombrado con Acceso Asociativo

Búffer de Renombrado

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
1	5	50	1	1
1	12	1200	1	1
1	2	20	1	1
1	1	3	1	1
:	:	:	:	:

Búsq. asoc. de r2



Buffer de Renombrado con Acceso Indexado

Índices

Búsq.
de r2 →

Entrada Válida	Índice
0	2
1	5
2	3
3	12
:	:

Búffer de renombrado

Valor	Valor Válido
45	1
320	1
30	1
20	1
:	:

- Permite varias escrituras pendientes a un mismo registro
- Se utiliza el bit último para marcar cual ha sido la más reciente

- Sólo permite una escritura pendiente a un mismo registro
- Se mantiene la escritura más reciente

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (I)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	0			
5	0			
6	0			
7	0			
8	0			
9	0			
10	0			
11	0			
12	0			
13	0			
14	0			

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

$$\begin{aligned}r2a &= r0a * r1a \\r3a &= r1a + r2a \\r2b &= r0a - r1a\end{aligned}$$

Situación Inicial:

- Existen dos renombrados de r1 en las entradas 2 y 3 del buffer
- La última está en la entrada 3

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (II)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	1
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				

 r0: 0, "válido"
 r1: 15, "válido"
 4

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

Ciclo i:

- Se emite la multiplicación
- Se accede a los operandos de la multiplicación, que tienen valores válidos en el buffer de renombrado
- Se renombra r2 (el destino de mul)

Renombrado

Ejemplo de Renombrado (III)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	1
5	3		0	1
6				
7				
8				
9				
10				
11				
12				
13				
14				

r1: 15, "válido"

r2: "no válido"

5

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo $i + 1$:

- Se emite la suma
- Se accede a sus operandos, pero r2 no estará preparado hasta que termine la multiplicación
- Se renombra r3 (destino de add)

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (IV)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	0
5	3		0	1
6	2		0	1
7	0			
8	0			
9	0			
10	0			
11	0			
12	0			
13	0			
14	0			

r0: 0, "válido" **r1: 15, "válido"** **6**

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

Ciclo $i + 2$:

- Se emite la resta
- Se accede a sus operandos
- Se vuelve a renombrar r2 (destino de sub)

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado (V)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2	0	1	0
5	3		0	1
6	2		0	1
7				
8				
9				
10				
11				
12				
13				
14				

r1: 15, "válido" r2: 0, "válido"

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

Ciclo i + 5:

- Termina la multiplicación
- Se actualiza el resultado en el buffer de renombrado
- Ya se puede ejecutar la suma con el valor de r2 de la entrada 4
- Cuando termine la resta escribirá otro valor para r2 en la entrada 6
- Sólo se escribirá en el banco de registros el último valor de r2

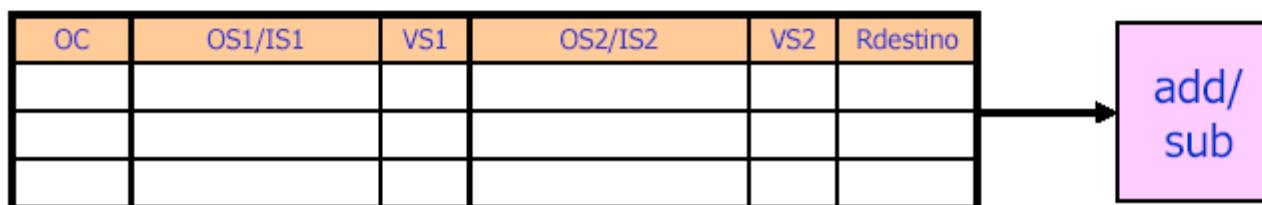
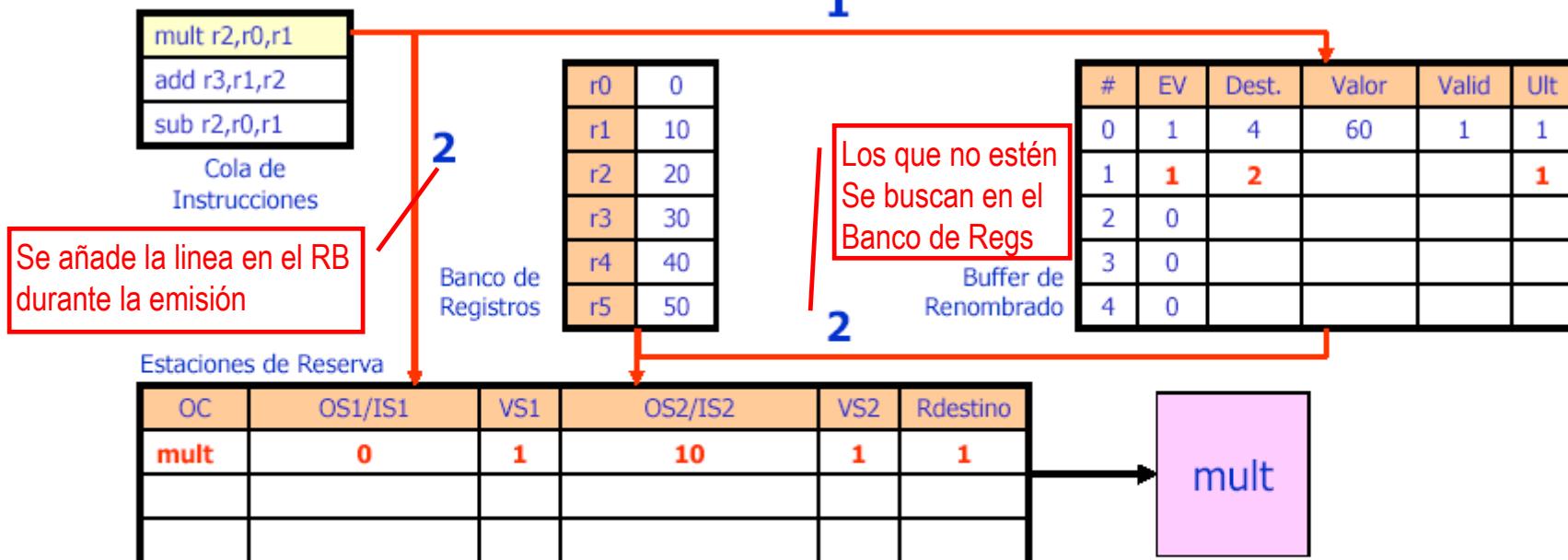
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva

Emisión de la multiplicación



Ingeniería de los Computadores

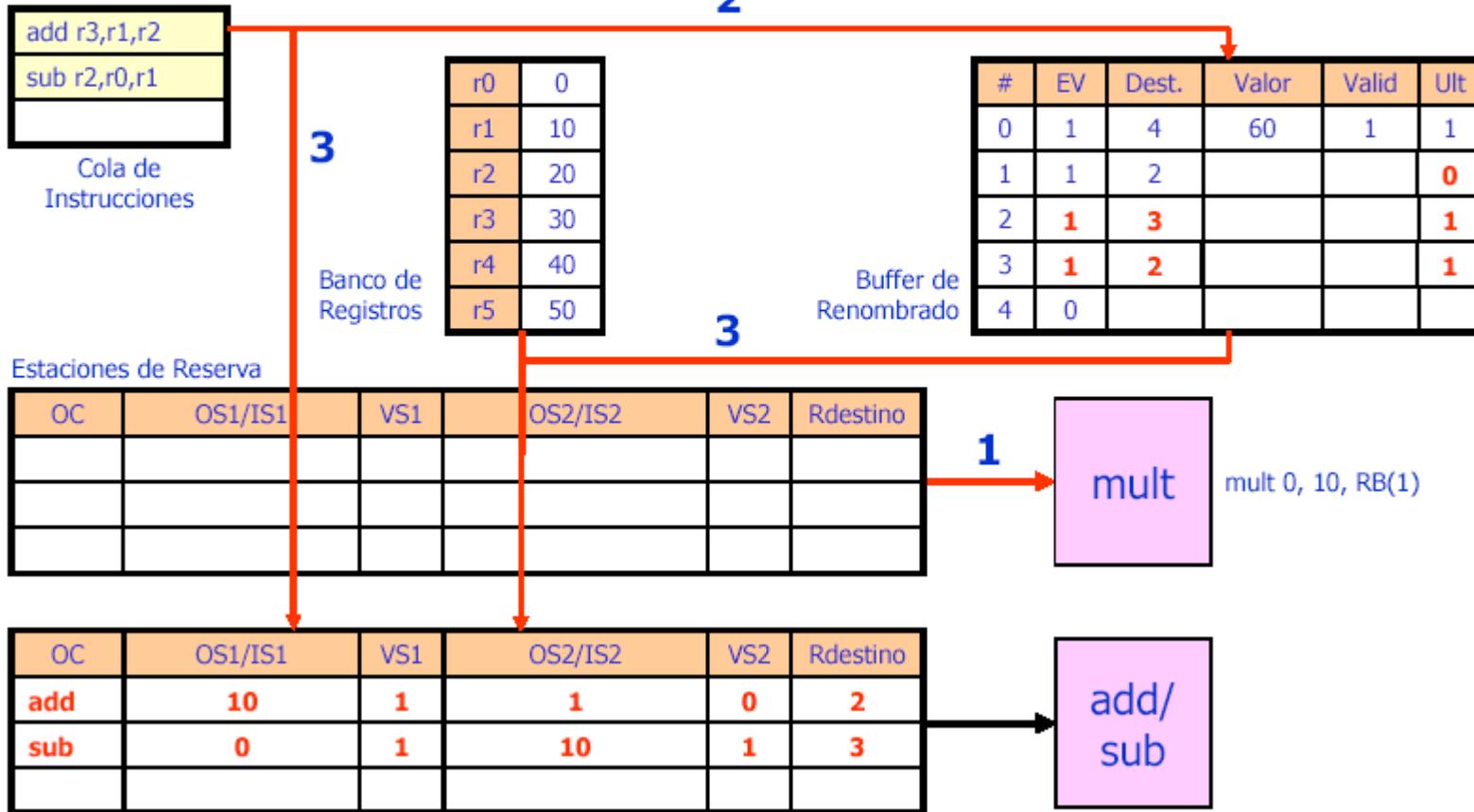
Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (II)

Envío de la multiplicación y emisión de las suma y la resta

2



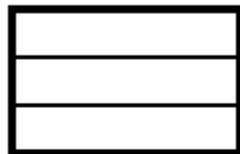
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (III)

Envío de la resta

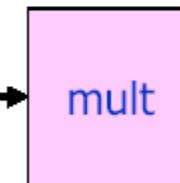


r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2			1
4	0				

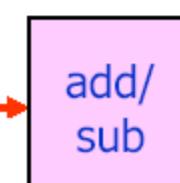
Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



sub 0, 10, RB(3)

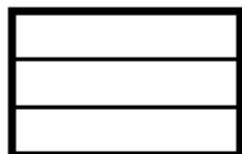
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (IV)

Termina la resta



Cola de Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

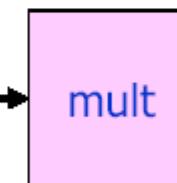
Banco de Registros

Buffer de Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva

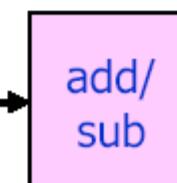
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



sub 0, 10, RB(3)

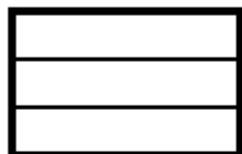
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (V)

Termina la multiplicación



Cola de
Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

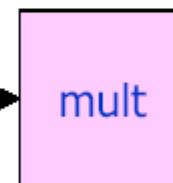
Banco de
Registros

Buffer de
Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva

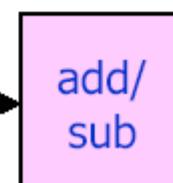
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	0	1	2



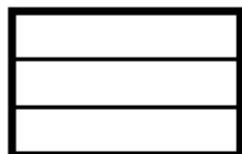
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VI)

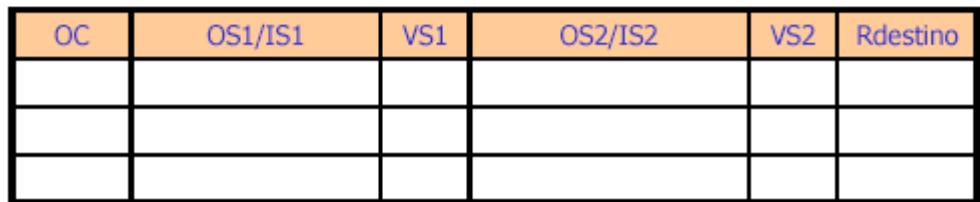
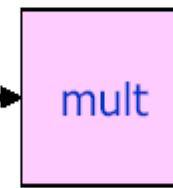
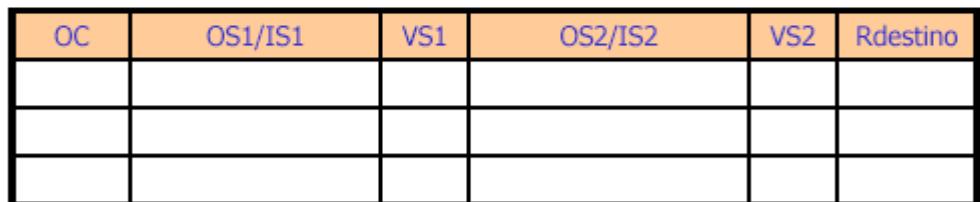
Envío de la suma



r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva



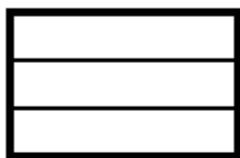
Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Termina la suma



Cola de Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

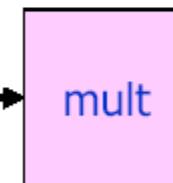
Banco de Registros

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3	10	1	1
3	1	2	-10	1	1
4	0				

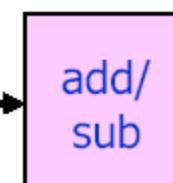
Buffer de Renombrado

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



add 10, 0, RB(2)

1

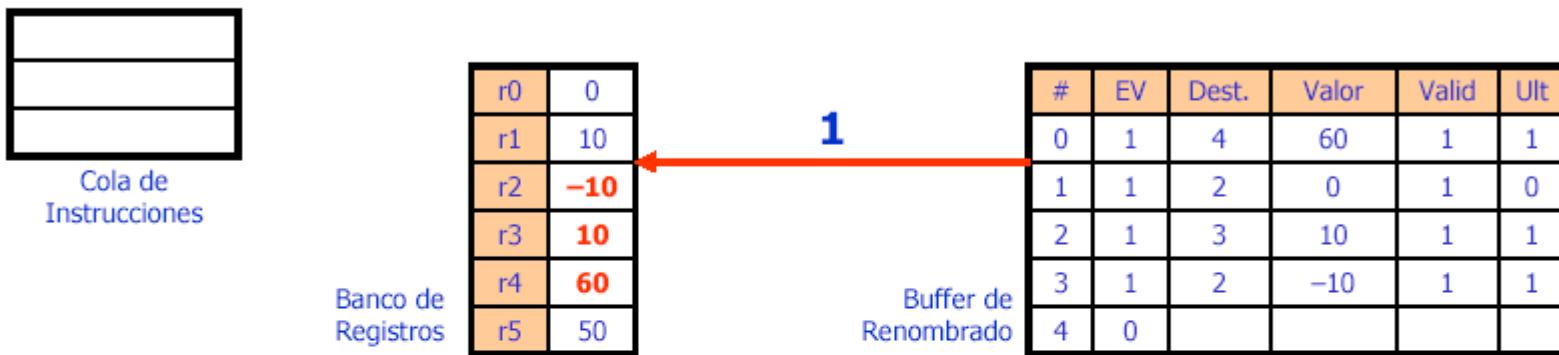
Ingeniería de los Computadores

Sesión 3. Superescalares

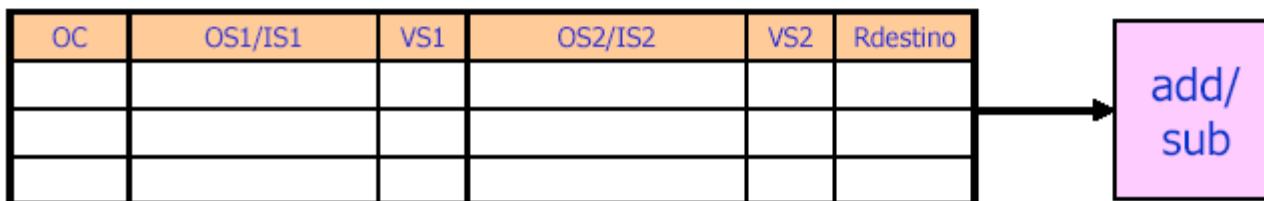
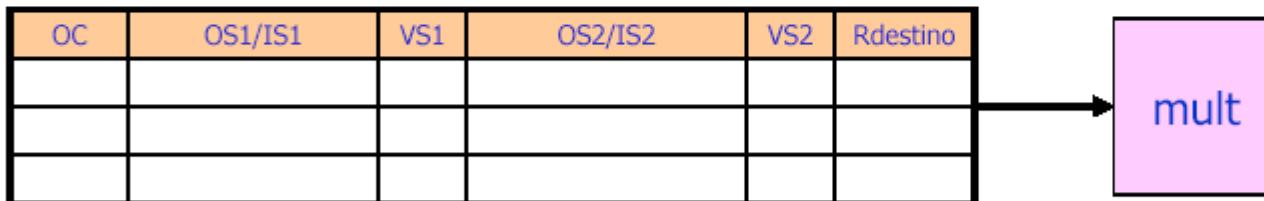
Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Se actualizan los registros



Estaciones de Reserva



Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

1									
2	instr(n)	f						
3	instr(n+1)	x						
4	instr(n+2)	f						
5	instr(n+3)	x						
6	instr(n+4)	x						
7	instr(n+5)	i						
8	instr(n+6)	i						
9									
10									

Cola
(Las instrucciones se retiran desde aquí)

Cabecera
(Las instrucciones que se decodifican se introducen a partir de aquí)

La gestión de interrupciones y la ejecución especulativa se realizan fácilmente mediante el ROB

- El puntero de cabecera apunta a la siguiente posición libre y el **puntero de cola a la siguiente instrucción a retirar**.
- Las instrucciones **se introducen en el ROB en orden de programa estricto** y pueden estar marcadas como **emitidas (issued, i)**, **en ejecución (x)**, o **finalizada su ejecución (f)**
- Las **instrucciones sólo se pueden retirar** (se produce la finalización con la escritura en los registros de la arquitectura) **si han finalizado**, y todas las que les preceden también.
- La **consistencia se mantiene porque sólo las instrucciones que se retiran del ROB se completan** (escriben en los registros de la arquitectura) y se retiran en el orden estricto de programa.

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

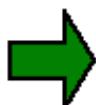
Ejemplo de Uso del Buffer de Reorden (I)

I1: mult r1, r2, r3

I2: st r1, 0x1ca

I3: add r1, r4, r3

I4: xor r1, r1, r3



Dependencias:

RAW: (I1,I2), (I3,I4)

WAR: (I2,I3), (I2,I4)

WAW: (I1,I3), (I1,I4), (I3,I4)

I1: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r2** y **r3**)

I2: Se envía a la unidad de almacenamiento hasta que esté disponible **r1**

I3: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r4** y **r3**)

I4: Se envía a la estación de reserva de la ALU para esperar a **r1**

Estación de Reserva (Unidad de Almacenamiento)

codop	dirección	op1	ok1
st	0x1ca	3	0

Estación de Reserva (ALU)

codop	dest	op1	ok1	op2	ok2
xor	6	5	0	[r3]	1

Líneas del ROB

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (II)

Ciclo 7

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	-	0	x	9
6	xor	10	r1	int_alu	-	0	i	-

Ciclo 9 No se puede retirar **add** aunque haya finalizado su ejecución

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	-	0	x	10

Ciclo 10 Termina **xor**, pero todavía no se puede retirar

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (III)

Ciclo 12

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	33	1	f	12
4	st	8	-	store	-	1	f	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Ciclo 13

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

- Se ha supuesto que se pueden retirar dos instrucciones por ciclo.
- Tras finalizar las instrucciones **mult** y **st** en el ciclo 12, se retirarán en el ciclo 13.
- Después, en el ciclo 14 se retirarán las instrucciones **add** y **xor**.

Ingeniería de los Computadores

Sesión 3. Superescalares

Renombrado

Reorden

Problemas

Ingeniería de los Computadores

Sesión 6. Paralelismo. Conceptos y
técnicas

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

- #1 TOP500 (6/2012) - Sequoia

- Manufacturer: IBM
- Núcleos: 1572864
- OS: Linux
- Interconexión: a medida
- Rendimiento máximo: 16324,8 TF
- Rendimiento pico: 20132,7 TF
- Consumo: 7890 kW
- Lugar: Lawrence Livermore National Laboratory (LLNL)
- <https://computing.llnl.gov/>
- Especificaciones detalladas:
https://computing.llnl.gov/?set=resources&page=OCF_resources#sequoia

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

- #5 TOP500 (6/2012) – Tianhe 1A
 - #1 TOP500 (11/2010)
 - Manufacturer: NUDT (National University of Defense Technology)
 - Núcleos: 186368
 - OS: Linux
 - Interconexión: propietaria
 - Rendimiento máximo: 2566 TF
 - Rendimiento pico: 4701 TF
 - Consumo: 4040 kW
 - Lugar: National Supercomputing Center in Tianjin
 - <http://www.nscc-tj.gov.cn/en/>
 - Especificaciones detalladas: http://www.nscc-tj.gov.cn/en/resources/resources_1.asp

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

- #177 TOP500 (6/2012) – Barcelona Supercomputing Center
 - Manufacturer: Bull
 - Núcleos: 5544
 - OS: Linux
 - Interconexión: Infiniband
 - Rendimiento máximo: 103,2 TF
 - Rendimiento pico: 182,9 TF
 - Consumo: 81,50 kW
 - Lugar: Barcelona Supercomputing Center
 - <http://www.bsc.es/>
 - Especificaciones detalladas: <http://www.bsc.es/marenostrum-support-services/res>

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

- Y todo esto...¿para qué?
 - Defensa
 - Terrorismo
 - Programa armamentístico
 - Programa espacial
 - Energía
 - Combustible basado en hidrógeno
 - Prospección de energías (carbón, gas, petroleo)
 - Ingeniería
 - Nanotecnología
 - Modelado biológico
 - Análisis de imagen

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Uso de un supercomputador

- Procesamiento paralelo

Estudia los aspectos relacionados con la división de una aplicación en unidades independientes y su ejecución en múltiples procesadores para reducir tiempo y/o aumentar la complejidad del problema.

- Procesamiento distribuido

Estudia los aspectos relacionados con la ejecución de múltiples aplicaciones al mismo tiempo utilizando múltiples recursos (procesadores, memorias, discos y bases de datos) situados en distintas localizaciones físicas.

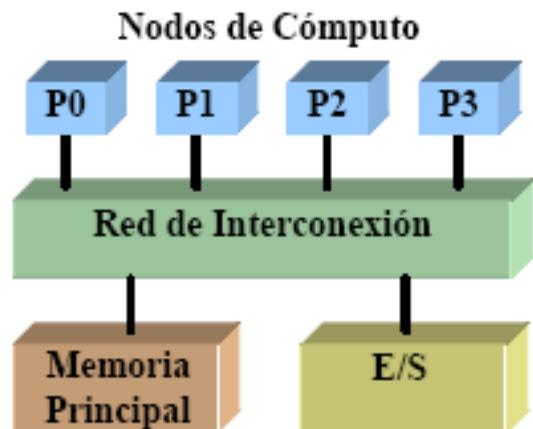
Ingeniería de los Computadores

Sesión 6. Paralelismo

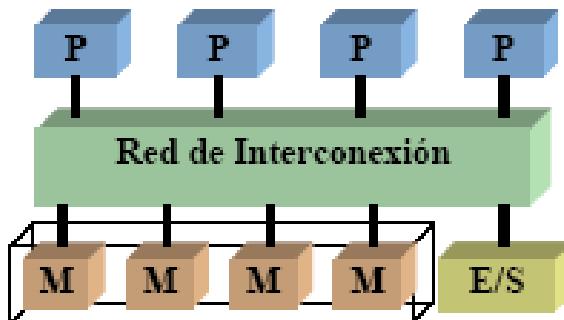
Introducción

Conceptos

- Clasificación de computadores paralelos (según el sistema de memoria)
 - Multiprocesadores. Comparten el mismo espacio de memoria (el programador no necesita saber dónde están los datos)
 - Multicomputadores. Cada procesador (nodo) tiene su propio espacio de direcciones (el programador necesita saber dónde están los datos)
- Diseño de un computador paralelo
 - Nodos de cómputo
 - Sistema de memoria
 - Sistema de comunicación
 - Sistema de entrada/salida



- Multiprocesador con memoria centralizada (SMP – Symmetric MultiProcessor)
 - Mayor latencia
 - Poco escalable
 - Comunicación mediante variables compartidas (datos no duplicados en memoria principal)
 - Necesita implementar primitivas de sincronización
 - No necesita distribuir código y datos
 - Programación más sencilla (generalmente)



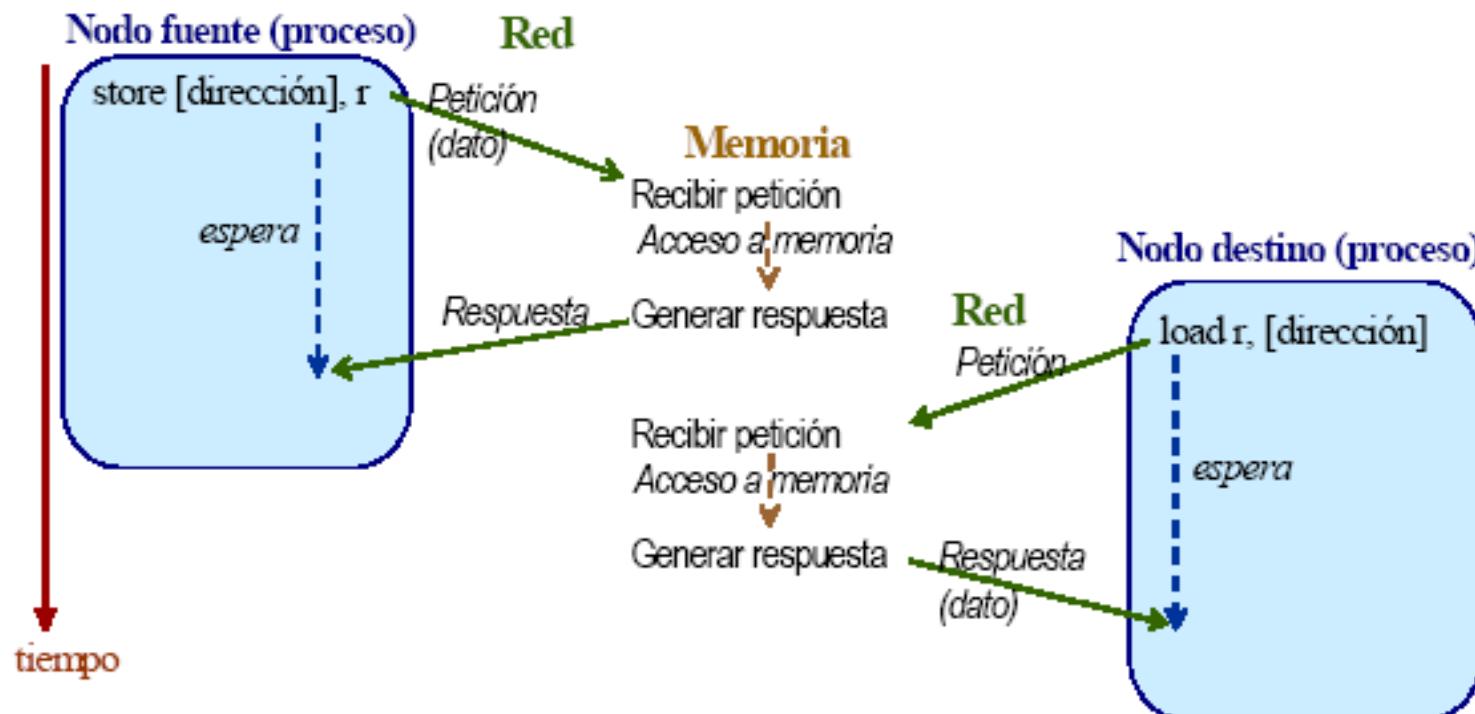
Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Comunicación en un multiprocesador



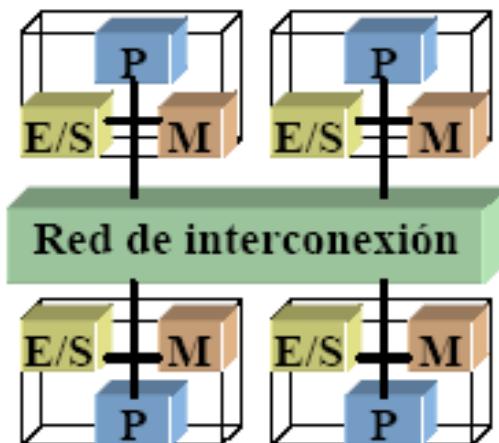
Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Multicomputador
 - Menor latencia
 - Mayor escalabilidad
 - Comunicación mediante paso de mensajes (datos duplicados en memoria)
 - Sincronización mediante mecanismos de comunicación
 - Distribución de carga de trabajo (código y datos) entre procesadores
 - Programación más difícil



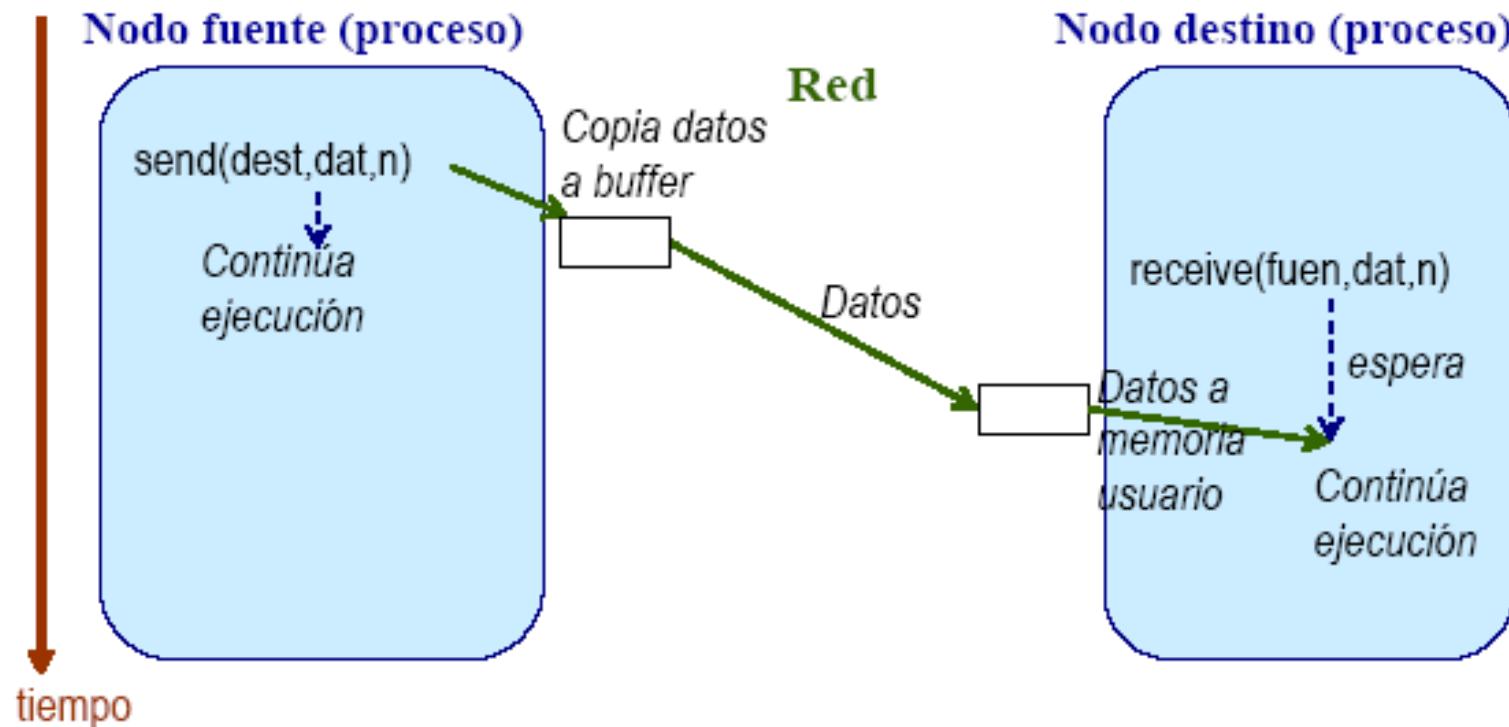
Ingeniería de los Computadores

Sesión 6. Paralelismo

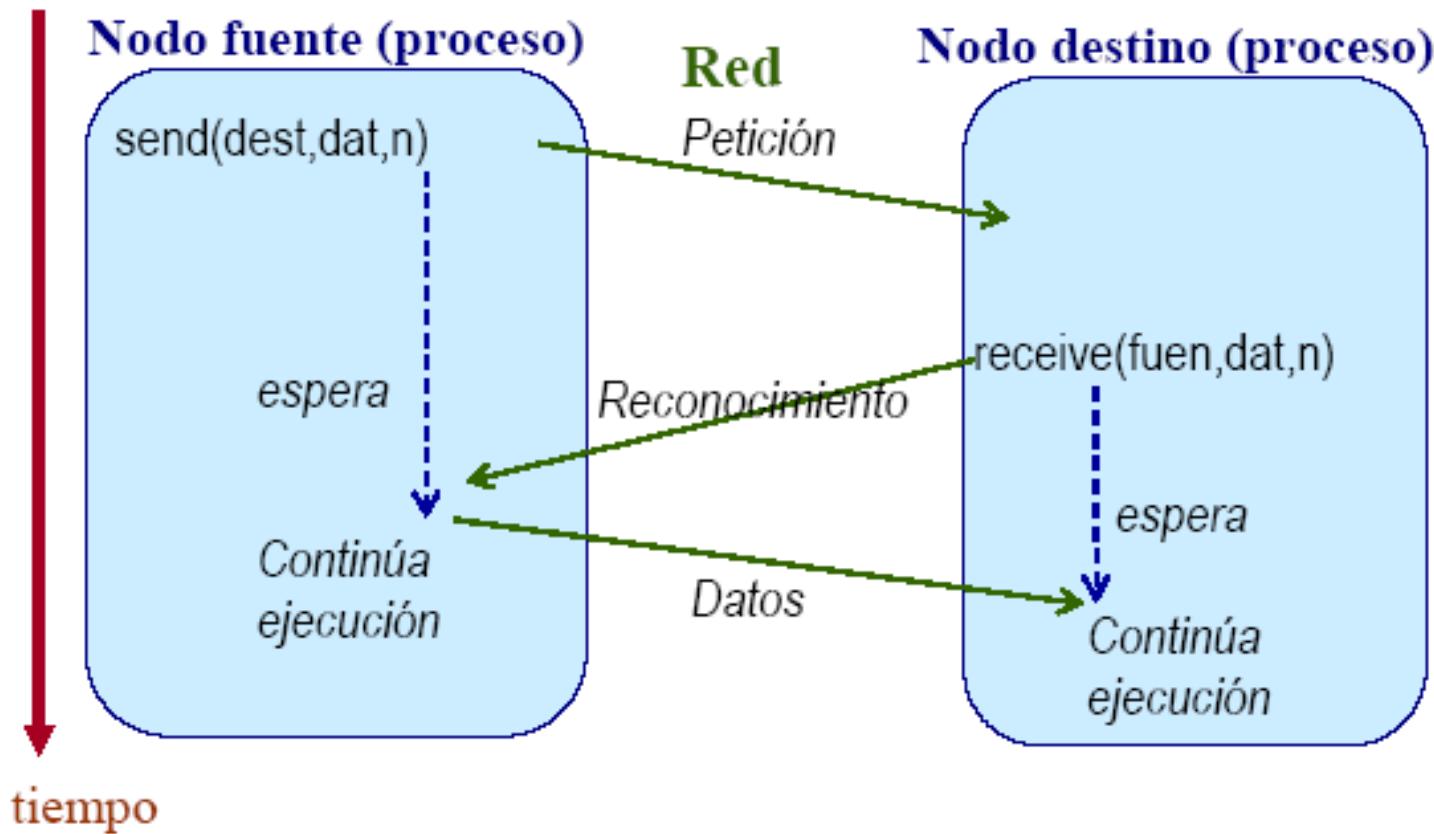
Introducción

Conceptos

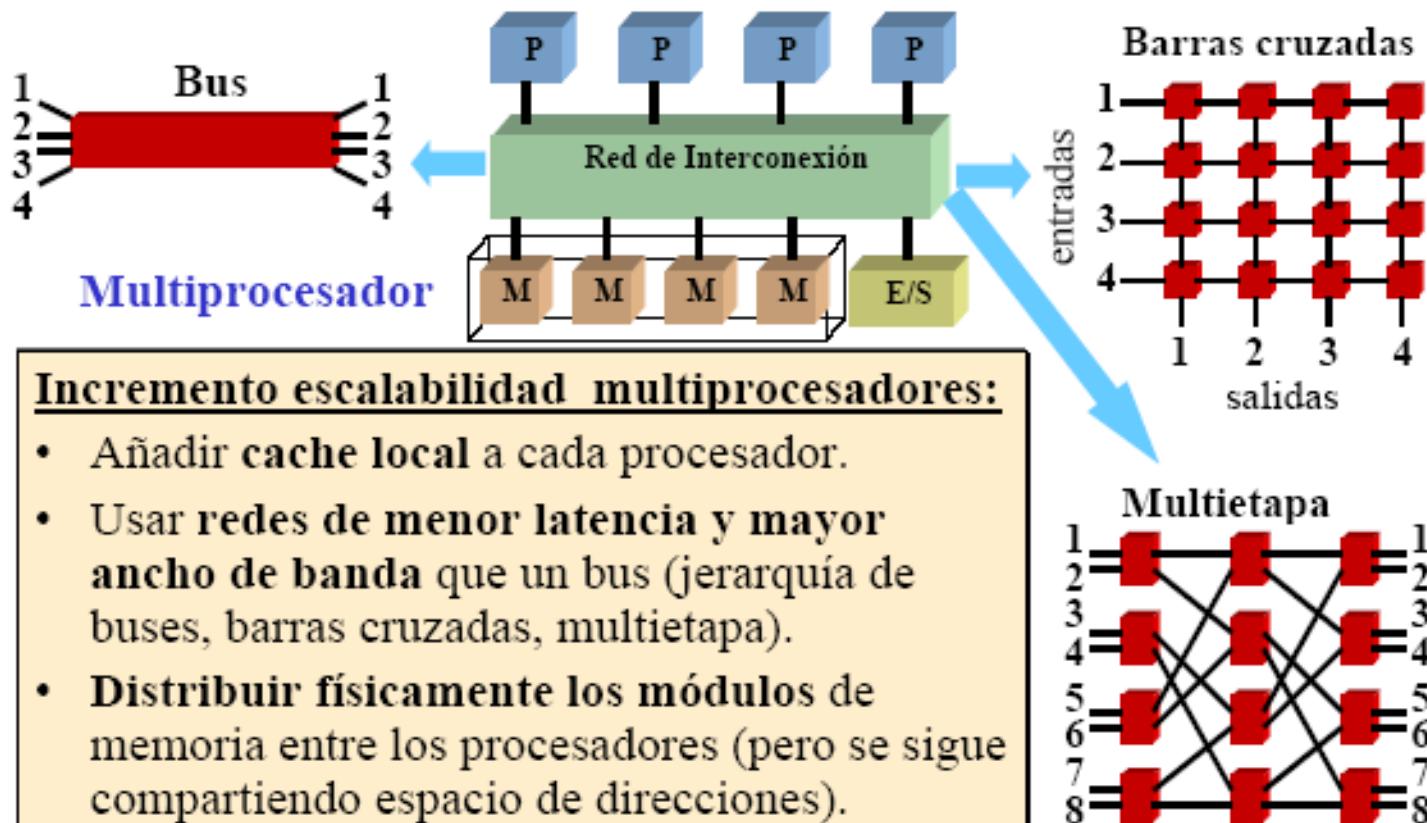
- Comunicación asíncrona en un multicomputador



- Comunicación síncrona en un multicomputador



- Redes de interconexión en multiprocesadores



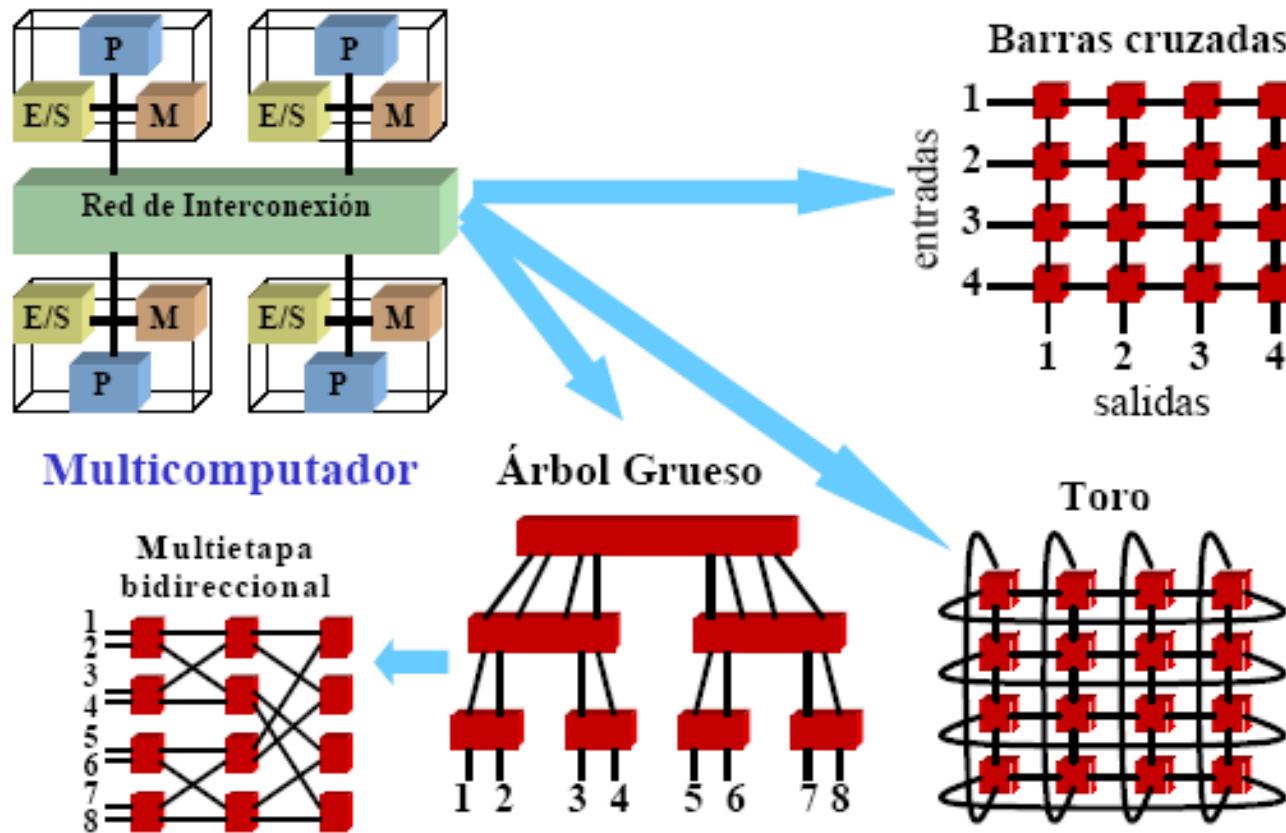
Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Redes de interconexión en multicomputadores



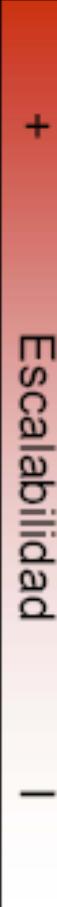
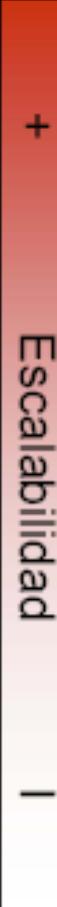
Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Clasificación de computadores paralelos

<p>Multicomputadores Memoria no compartida Múltiples espacios de direcciones.</p>	IBM-SP2 HP AlphaServer SC45 (cluster de SMP)			<p>Memoria físicamente distribuida</p> <p>Escalables</p>	
<p>Multiprocesadores Memoria compartida Un único espacio de direcciones.</p>	NUMA <i>(Non-Uniform Memory Access)</i>	NUMA	Cray T3E, Cray X1		
	CC-NUMA		Origin 3000 HP 9000 Superdome		
<p>UMA <i>(Uniform Memory Access)</i></p>	COMA		KSR-1	<p>Memoria físicamente centralizada</p>	
	SMP		WS, servidores básicos		

- Otros tipos de computadores paralelos
 - **MPP (Massive Parallel Processor)**. Número de procesadores superior a 100. Sistemas con redes diseñadas a medida.
 - **Cluster**. Computadores completos (PC's, servidores, etc.) conectados con alguna red comercial tipo LAN que se utiliza como recurso de cómputo único. El tráfico de la red se reduce al ocasionado por la aplicación ejecutada (no hay tráfico externo)
 - **Cluster Beowulf**. Cluster con sistema operativo libre (Linux) y hardware y software de amplia difusión
 - **Constelaciones**. Cluster de SMP con un nº de procesadores en un nodo mayor que el nº de nodos del cluster

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Otros tipos de computadores paralelos
 - **Redes de computadores.** Conjunto de computadores conectados por una LAN. Por la red circula tanto tráfico de la aplicación paralela como externo
 - **GRID.** Conjunto de recursos autónomos distribuidos geográficamente conectados por infraestructura de telecomunicaciones y que conforman un sistema de altas prestaciones virtual

- Tipos de paralelismo
 - Paralelismo funcional

Se obtiene a través de la reorganización de la estructura lógica de una aplicación.

Existen diferentes niveles de paralelismo funcional según las estructuras que se reorganicen.

- Paralelismo de datos

Implícito en operaciones con estructuras de datos tipo vector o matriz. Está relacionado con operaciones realizadas sobre grandes volúmenes de datos que sean independientes entre si.

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Paralelismo funcional

Niveles:

Programas

Programa1

Programa2

.....

Funciones

Func10 {
...
}

Func20 {
...
}

Func30 {
...
}

Bucle
(bloques)

for (){
...
}

while(){
...
}

Operaciones



Granularidad:

Grano Grueso

Grano
Medio

Grano
Medio-Fino

Grano Fino

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

- Tipos de paralelismo
 - Paralelismo explícito

Paralelismo no presente de forma inherente en las estructuras de programación y que se debe indicar expresamente.

- Paralelismo implícito

Paralelismo presente (aunque puede no estar ejecutándose de forma paralela) debido a la propia estructura de los datos (vectores) o de la aplicación (bucle)

Ingeniería de los Computadores

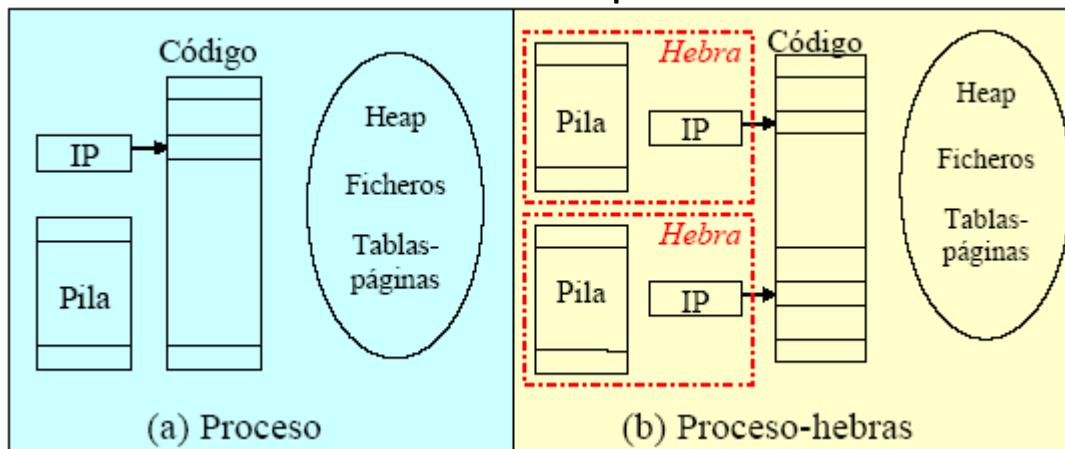
Sesión 6. Paralelismo

Introducción

Conceptos

- Unidades de ejecución: hilos y procesos

- Hardware (procesador). Gestiona la ejecución de las instrucciones
- Software (SO). Gestiona la ejecución de hilos y procesos
 - Proceso: espacio de direcciones virtuales propio
 - Hilos: comparten direcciones virtuales, se crean y destruyen más rápido y la comunicación también es más rápida

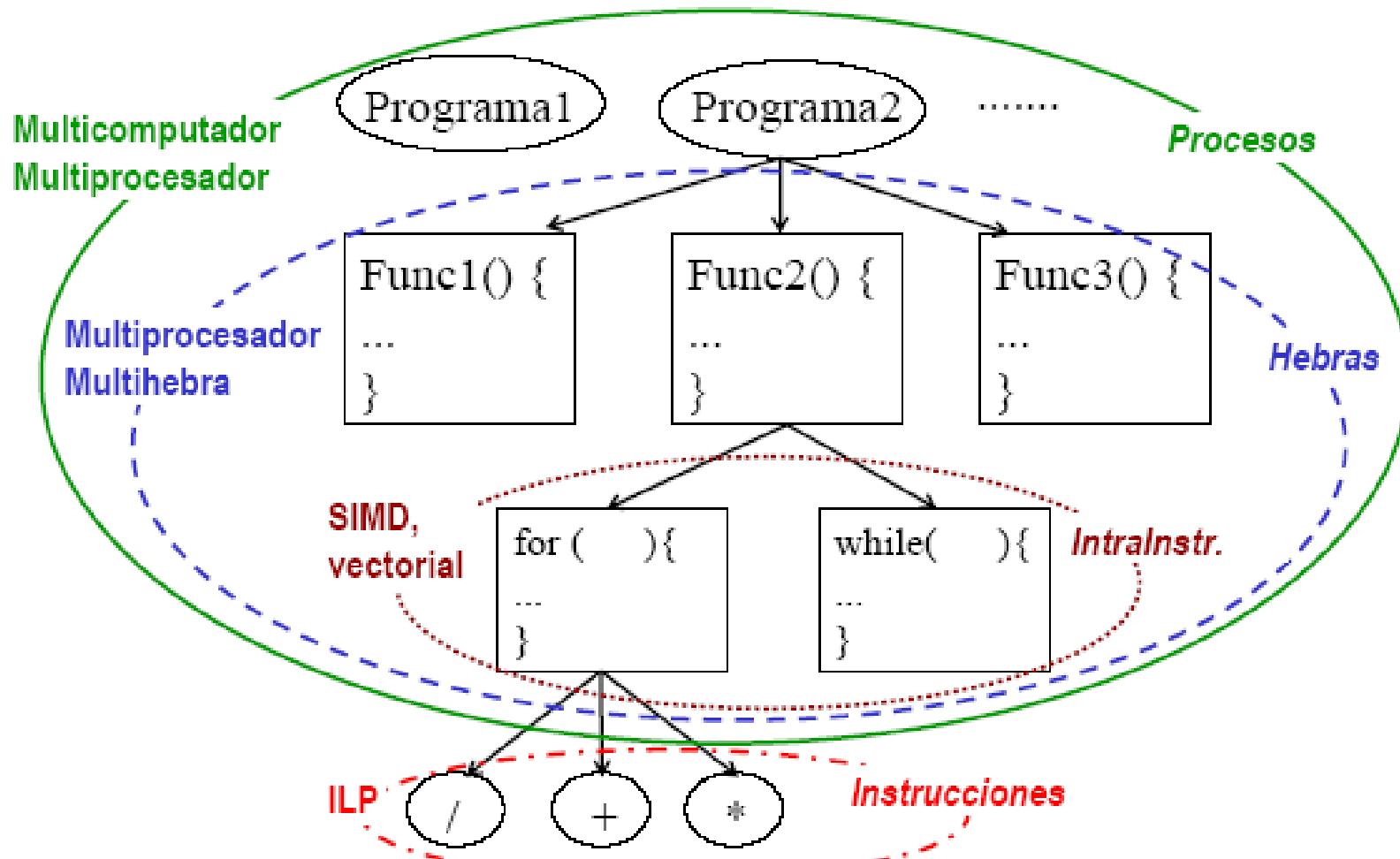


Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos



Ingeniería de los Computadores

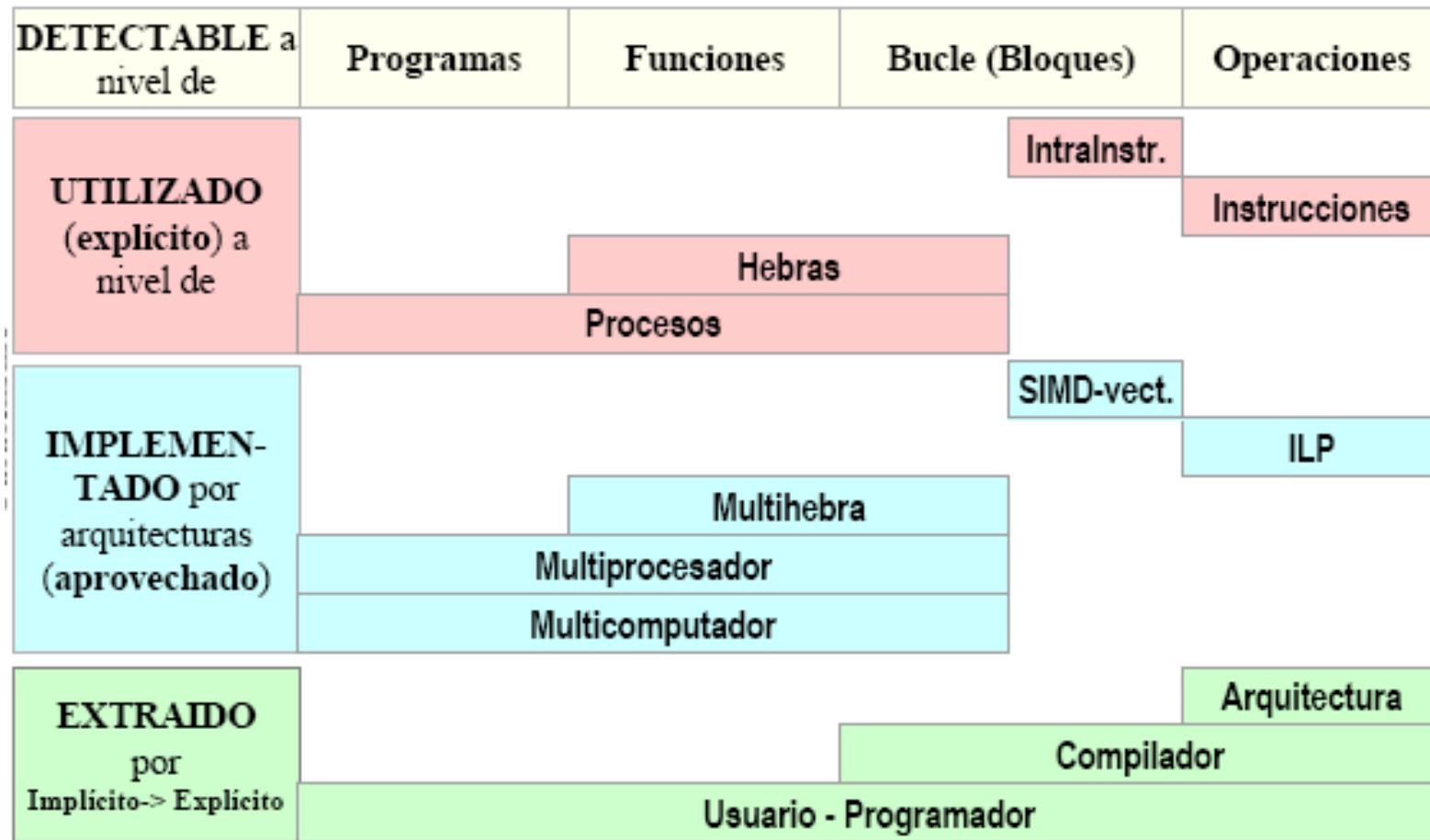
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Detección y extracción de paralelismo



Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Problemas introducidos por la programación paralela
 - División en unidades de cómputo independientes (tareas)
 - Agrupación de tareas (código y datos) en procesos/hebras
 - Asignación a procesadores
 - Sincronización y comunicación
- Situación inicial (normalmente)
 - Se parte de una versión secuencial (no paralela)
 - Se parte de una descripción de la aplicación
 - Elementos de apoyo:
 - Programa paralelo que resuelva un problema semejante
 - Librerías de funciones paralelas (BLAS, LAPACK, OpenMP)

Ingeniería de los Computadores

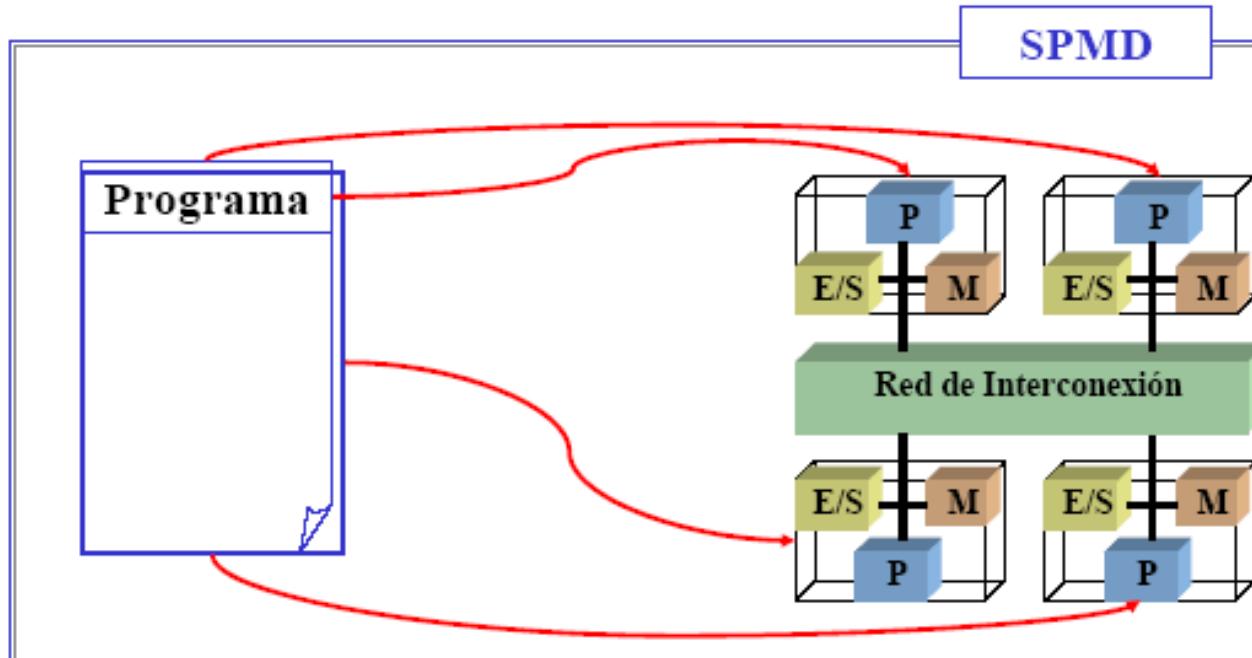
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Modos de programación paralela
 - SPMD (Single Program Multiple Data)



Ingeniería de los Computadores

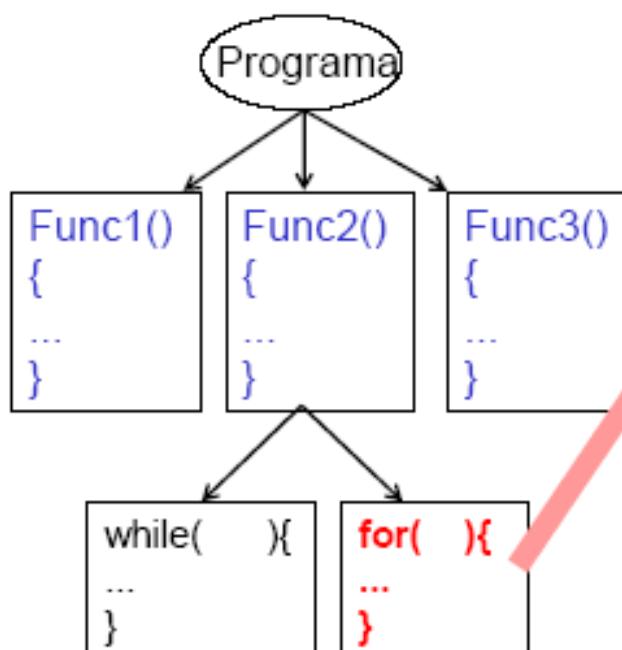
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Modos de programación paralela
 - SPMD (Single Program Multiple Data). Ejemplo



SPMD

```
Func1()\n{...}\n}\n\nFunc2()\n{...}\n}\n\n...\n\nfor (i=ithread;i<N;i=i+nthread){\n    código para la iteración i\n    ...\n}\n\nFunc3()\n{...}\n}\n\nMain () {\n    ...\n\n    switch (iproc) {\n        case 0: Func1(); break;\n        case 1: Func2(); break;\n        case 2: Func3(); break;\n    }\n    ...\n}
```

Ingeniería de los Computadores

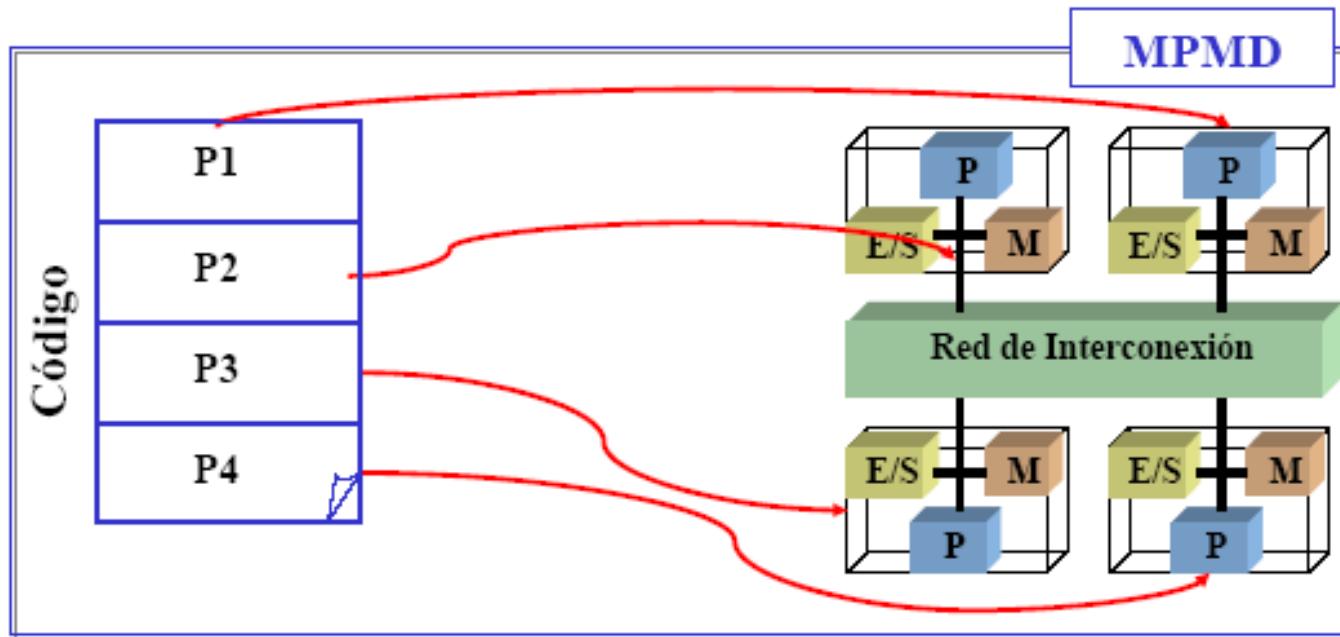
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Modos de programación paralela
 - MPMD (Multiple Program Multiple Data)



- Modo Mixto SPMD-MPMD

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Herramientas que facilitan la programación paralela
 - Compiladores paralelos. Extracción automática del paralelismo
 - Directivas del compilador (OpenMP): lenguaje secuencial + directivas
 - Lenguajes paralelos (HPF, Occam, Ada)
 - Bibliotecas de funciones (Pthread, MPI, PVM): lenguaje secuencial + funciones de biblioteca como interfaces

Ingeniería de los Computadores

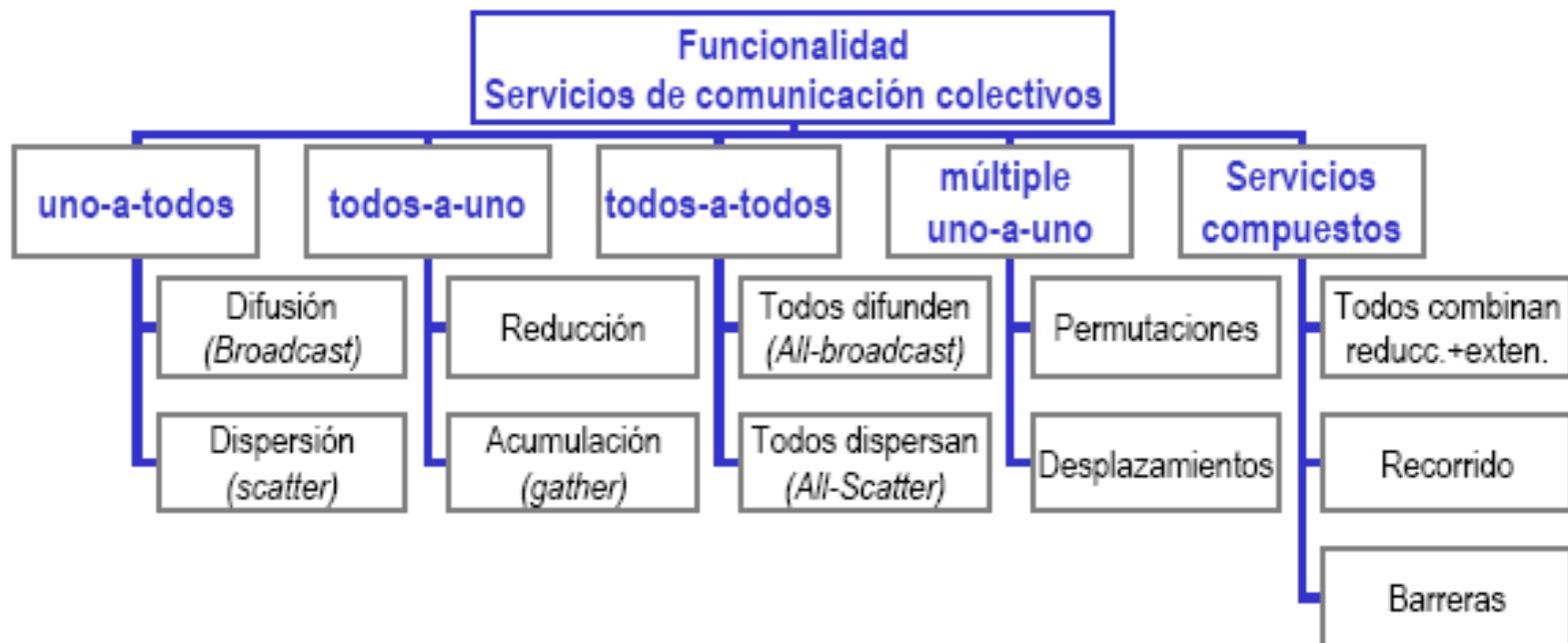
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Comunicación. Alternativas



Ingeniería de los Computadores

Sesión 6. Paralelismo

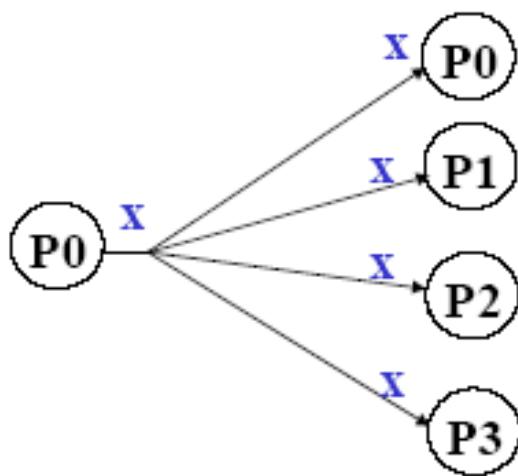
Introducción

Conceptos

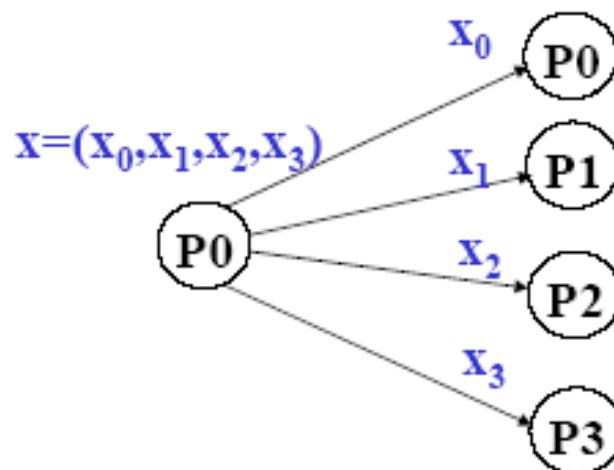
Prog. Paralela

- Comunicación: uno a todos

Difusión (*broadcast*)



Dispersión (*scatter*)



Ingeniería de los Computadores

Sesión 6. Paralelismo

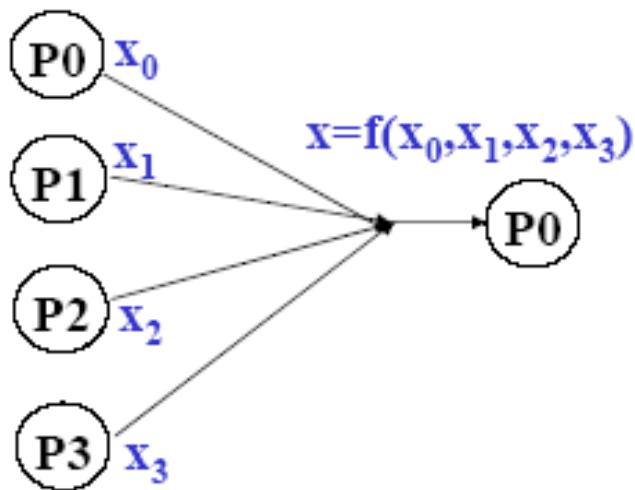
Introducción

Conceptos

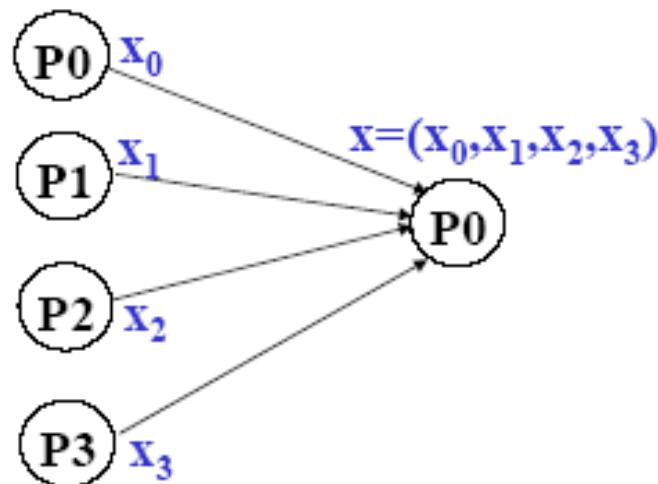
Prog. Paralela

- Comunicación: todos a uno

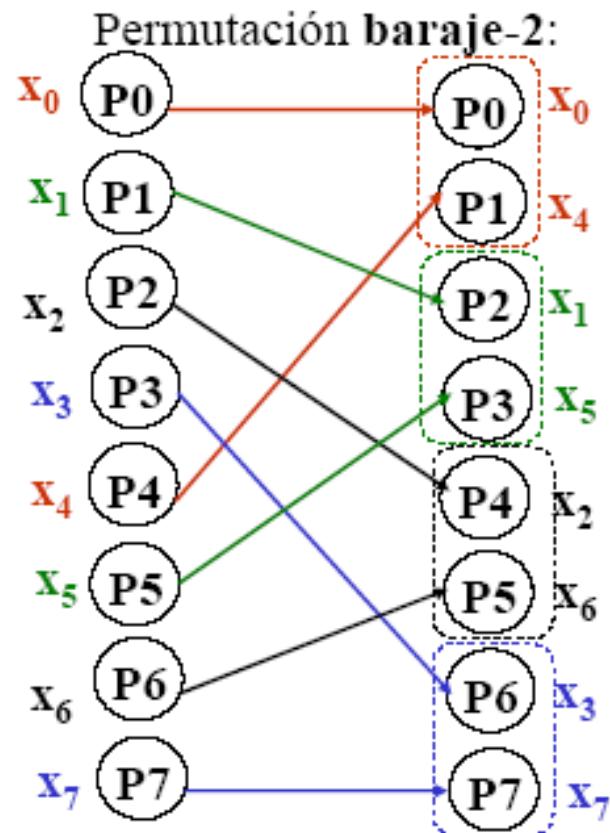
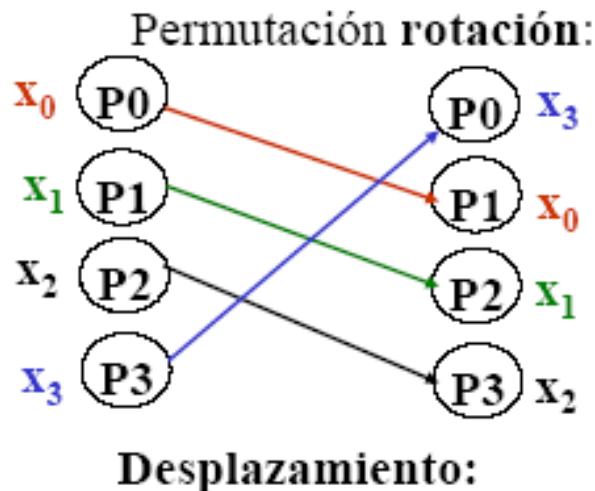
Reducción



Acumulación (*gather*)



- Comunicación: múltiple uno a uno



Ingeniería de los Computadores

Sesión 6. Paralelismo

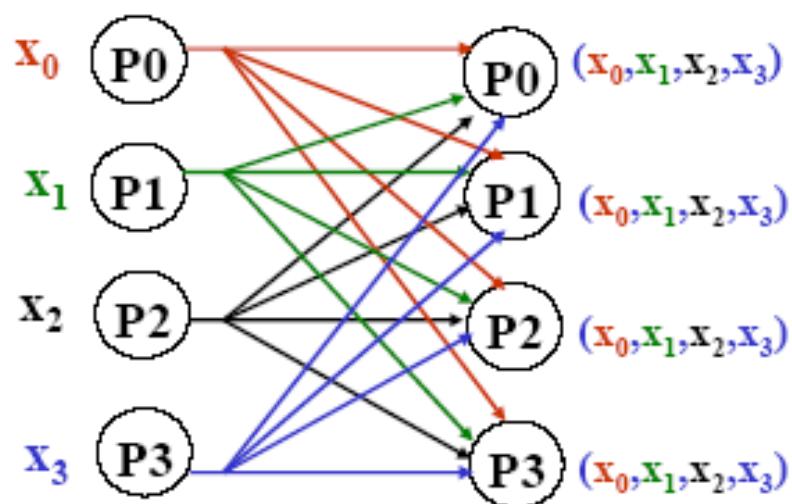
Introducción

Conceptos

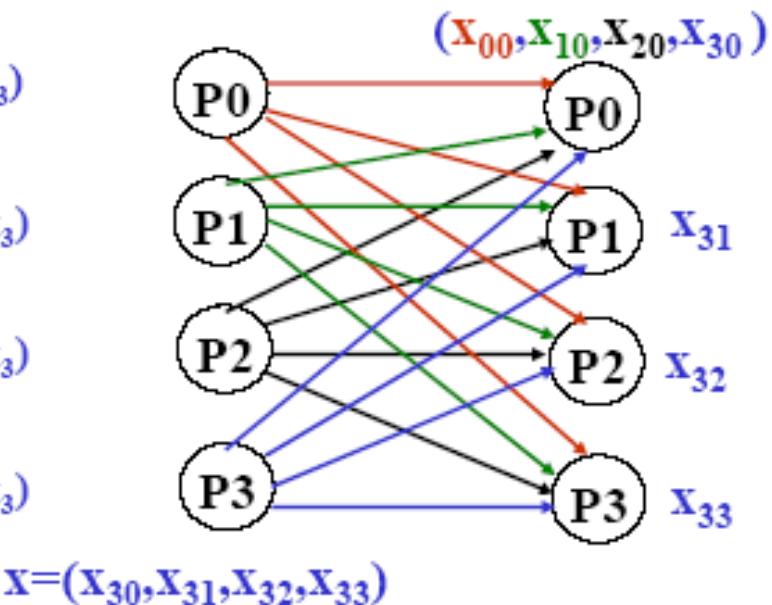
Prog. Paralela

- Comunicación: todos a todos

Todos Difunden (*all-broadcast*)
o chismorreo (*gossiping*)

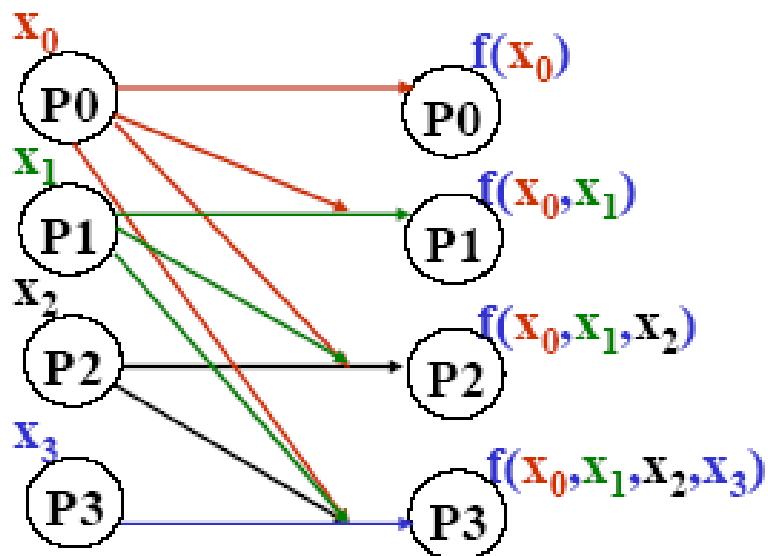


Todos Dispersan (*all-scatter*)

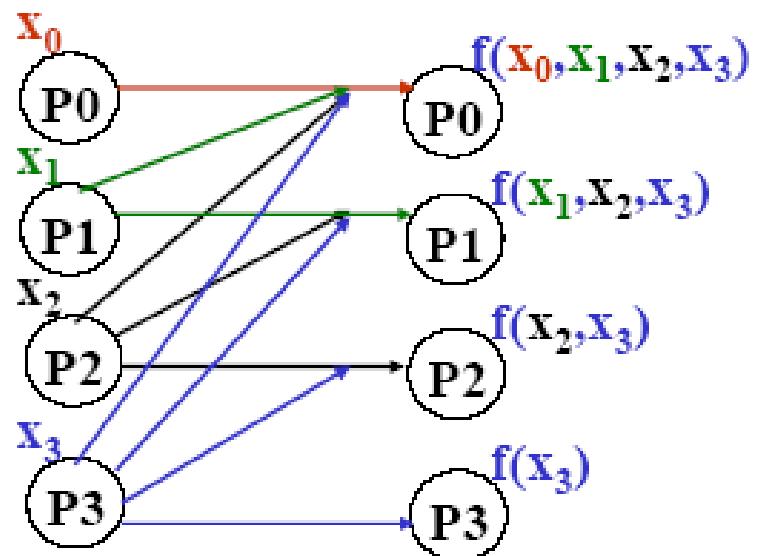


- Comunicación: servicios compuestos

Recorrido prefijo paralelo



Recorrido sufijo paralelo



Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Comunicación. Ejemplos de funciones colectivas usando MPI

Uno-a-todos	Difusión	MPI_Bcast()
	Dispersión	MPI_Scatter()
Todos-a-uno	Reducción	MPI_Reduce()
	Acumulación	MPI_Gather()
Todos-a-todos	Todos difunden	MPI_Allgather()
Servicios compuestos	Todos combinan	MPI_Allreduce()
	Barreras	MPI_Barrier()
Scan		MPI_Scan

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Estilos de programación paralela: paso de mensajes
 - Herramientas software
 - Lenguajes de programación (Ada) y librerías (MPI, PVM)
 - Distribución de carga de trabajo
 - Uso de librerías: explícita con construcciones del lenguaje secuencial (if, for, ...)
 - Uso de lenguaje paralelo: explícita con construcciones del propio lenguaje (Occam: sentencia par)
 - Primitivas básicas de comunicación : Send y Receive
 - Funciones de comunicación colectivas
 - Sincronización
 - Receive bloqueante. Barreras (MPI_BARRIER)
 - Send-receive bloqueante en ADA

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Estilo de programación paralela: variables compartidas
 - Lenguajes de programación (Ada), Funciones de librerías (OpenMP), directivas del compilador (OpenMP)
 - Distribución de la carga
 - Librerías: explícita con el lenguaje secuencial
 - Directivas: sincronización implícita (mayor nivel de abstracción)
 - Lenguajes: construcciones del lenguaje
 - Comunicación básica: load y store
 - Funciones de comunicación colectiva
 - Sincronización
 - Semáforos, cerros, variables condicionales

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Estilos de programación paralela: paralelismo de datos
 - Herramientas
 - Lenguaje de programación (HPF – High Performance Fortran)
 - Distribución de carga
 - Directivas para distribuir datos entre procesadores
 - Directivas para parallelizar bucles
 - Comunicación básica: implícita ($A(i)=A(i-1)$)
 - Funciones de comunicación colectivas
 - Ímplicitas en funciones usadas explícitamente por el programador (rotaciones – CSHIFT)
 - Sincronización: implícita

Ingeniería de los Computadores

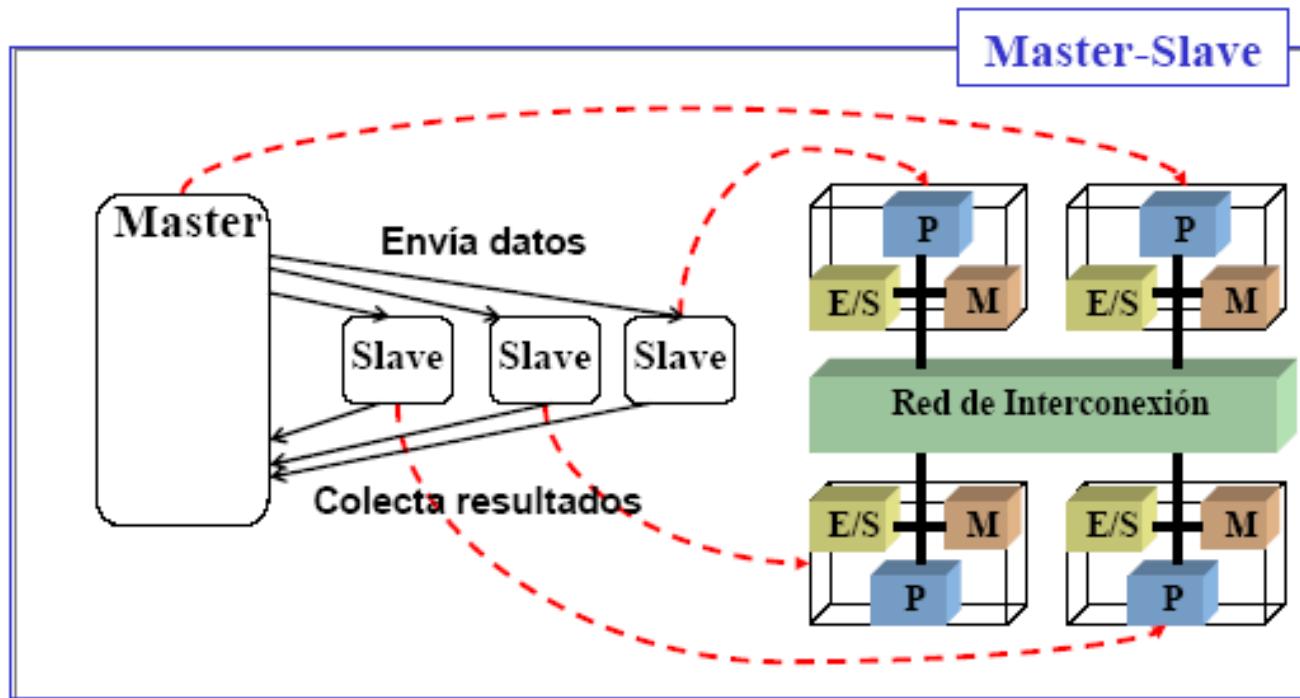
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Estructuras de paralelismo: master-slave



Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Estructuras de paralelismo: master slave

Master-Slave como MPMD– SPMD

```
main ()  
{   código Master  
}
```

```
main ()  
{   código Slaves  
}
```

Master-Slave como SPMD

```
main ()  
{ if (iproc=id_Master) {  
    código Master  
} else {  
    código Slaves  
}
```

Ingeniería de los Computadores

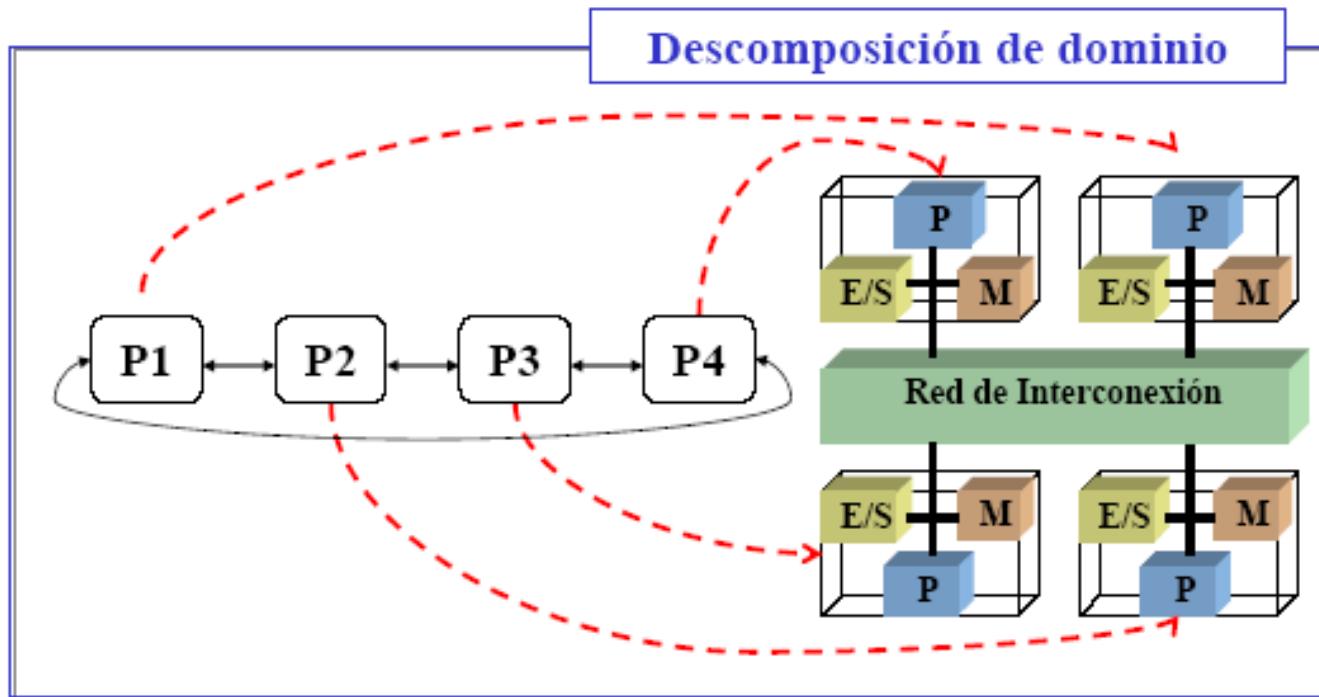
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Estructuras de paralelismo: descomposición de dominio (paralelismo de datos)



Ingeniería de los Computadores

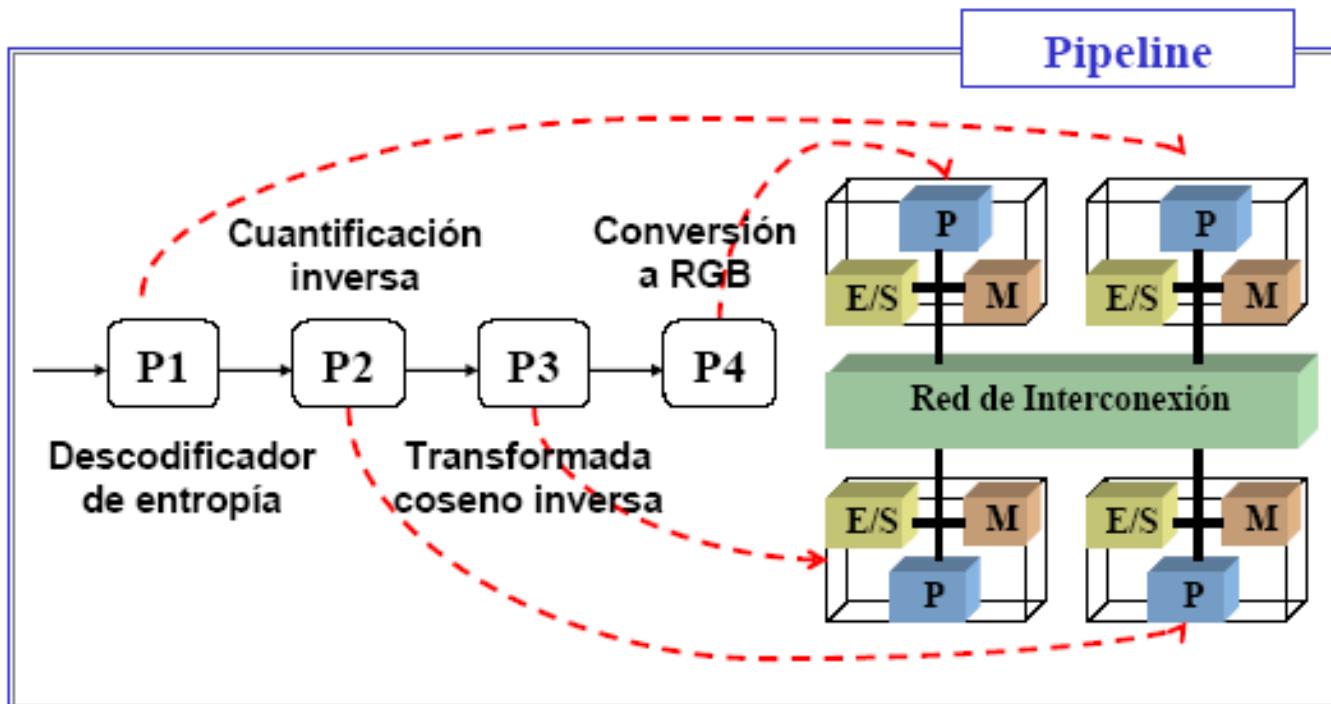
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Estructuras de paralelismo: segmentada (flujo de datos)



Ingeniería de los Computadores

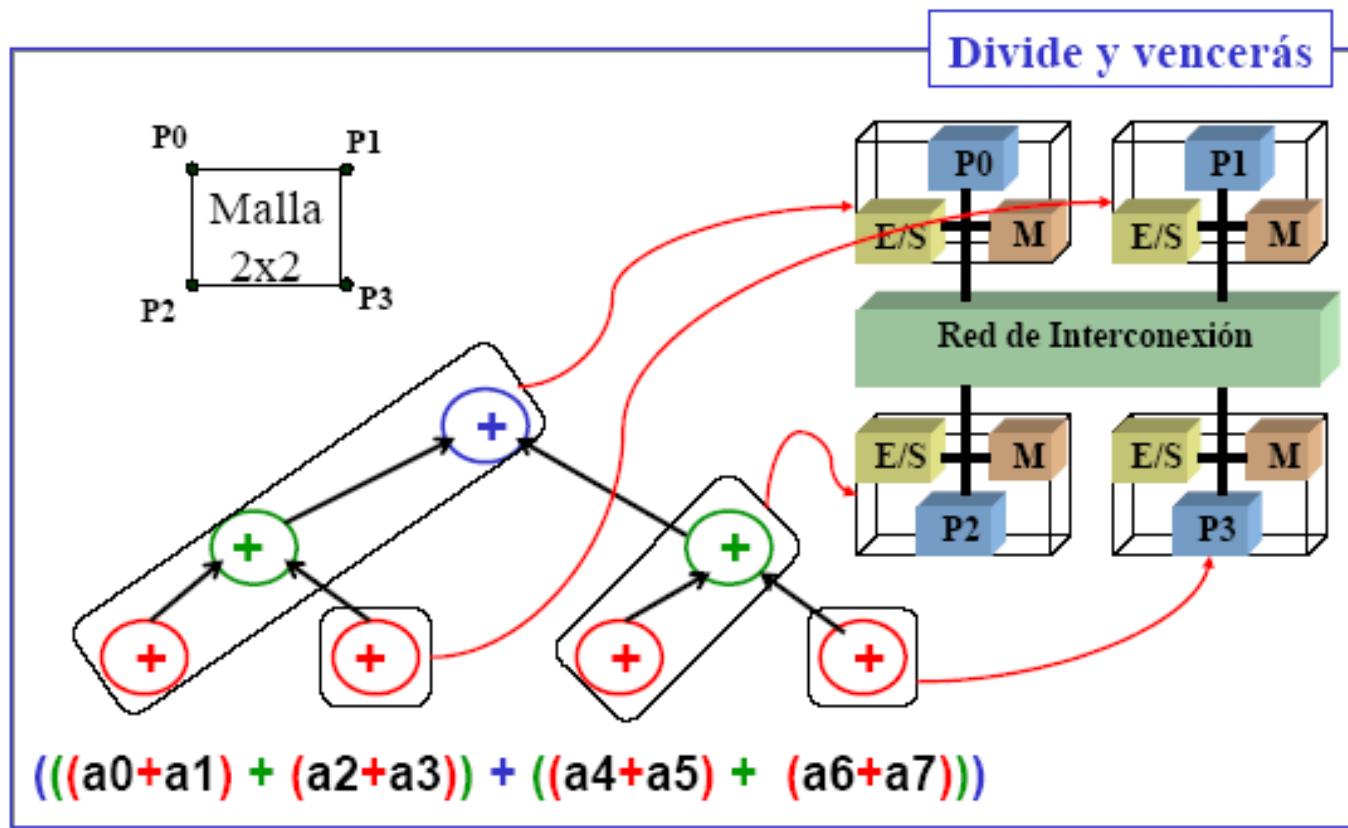
Sesión 6. Paralelismo

Introducción

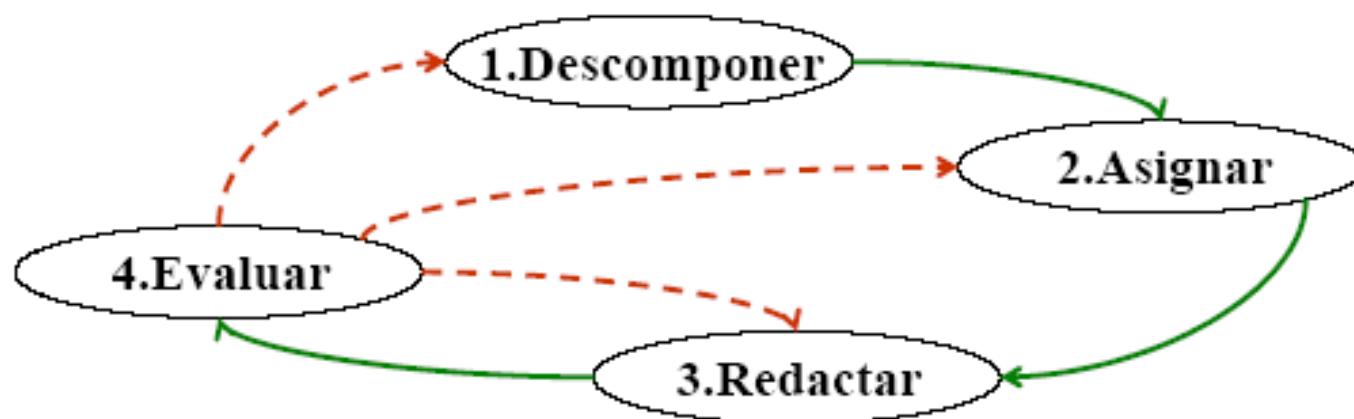
Conceptos

Prog. Paralela

- Estructuras de paralelismo: divide y vencerás (descomposición recursiva)



- Proceso de paralelización
 - Descomposición en tareas independientes
 - Asignación de tareas a procesos y/o hebras
 - Redacción de código paralelo
 - Evaluación de prestaciones



Ingeniería de los Computadores

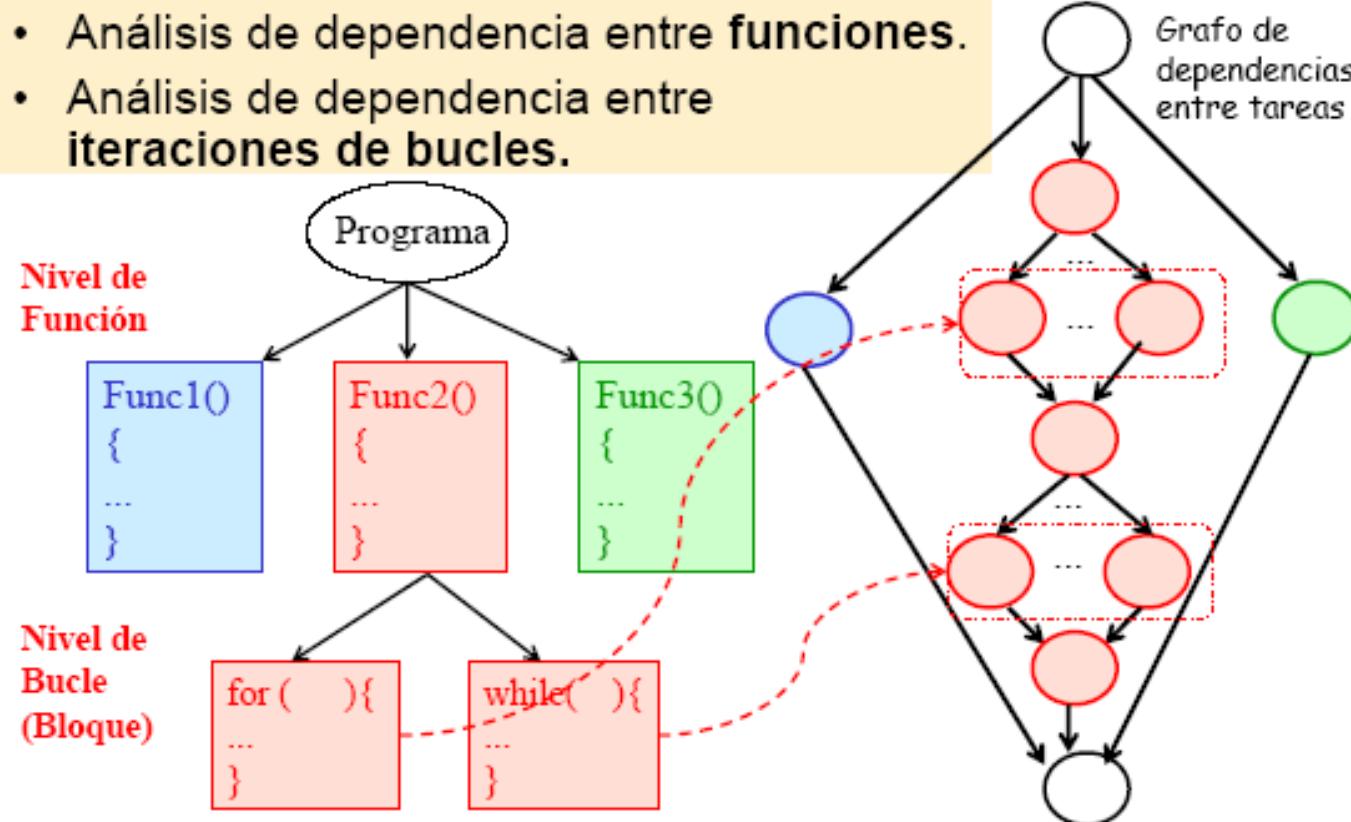
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Proceso de paralelización: descomposición en tareas independientes



Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Proceso de paralelización: asignación de tareas
 - Normalmente se asignan iteraciones de un ciclo a hebras y funciones a procesos
 - La granularidad depende de
 - El número de procesadores
 - El tiempo de comunicación/sincronización frente al tiempo de cálculo
 - Equilibrio en la carga de trabajo (que unos procesadores no esperen a otros)
 - Tipos de asignación
 - Dinámica (en tiempo de ejecución). Si no se conoce el número de tareas
 - Estática (programador o compilador)

Ingeniería de los Computadores

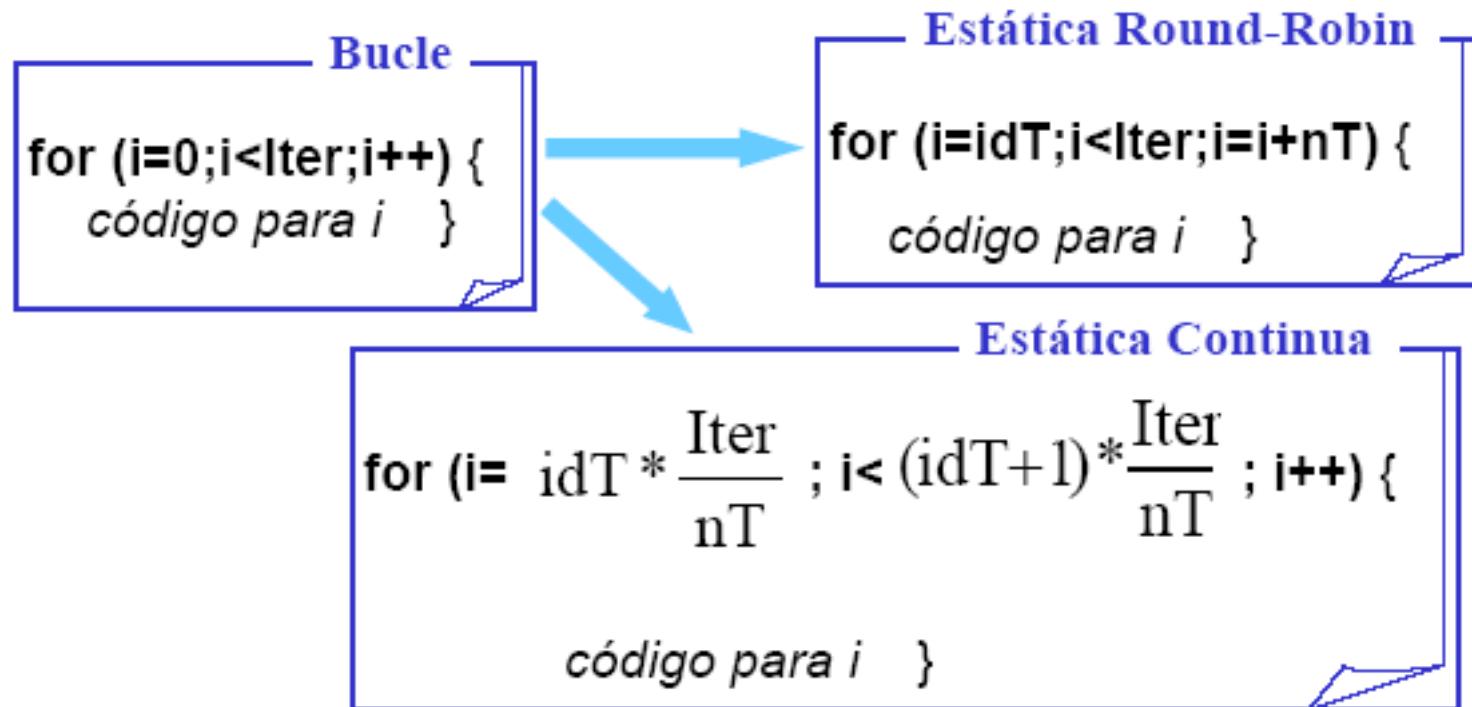
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Proceso de paralelización: asignación de tareas
 - Asignación estática



Ingeniería de los Computadores

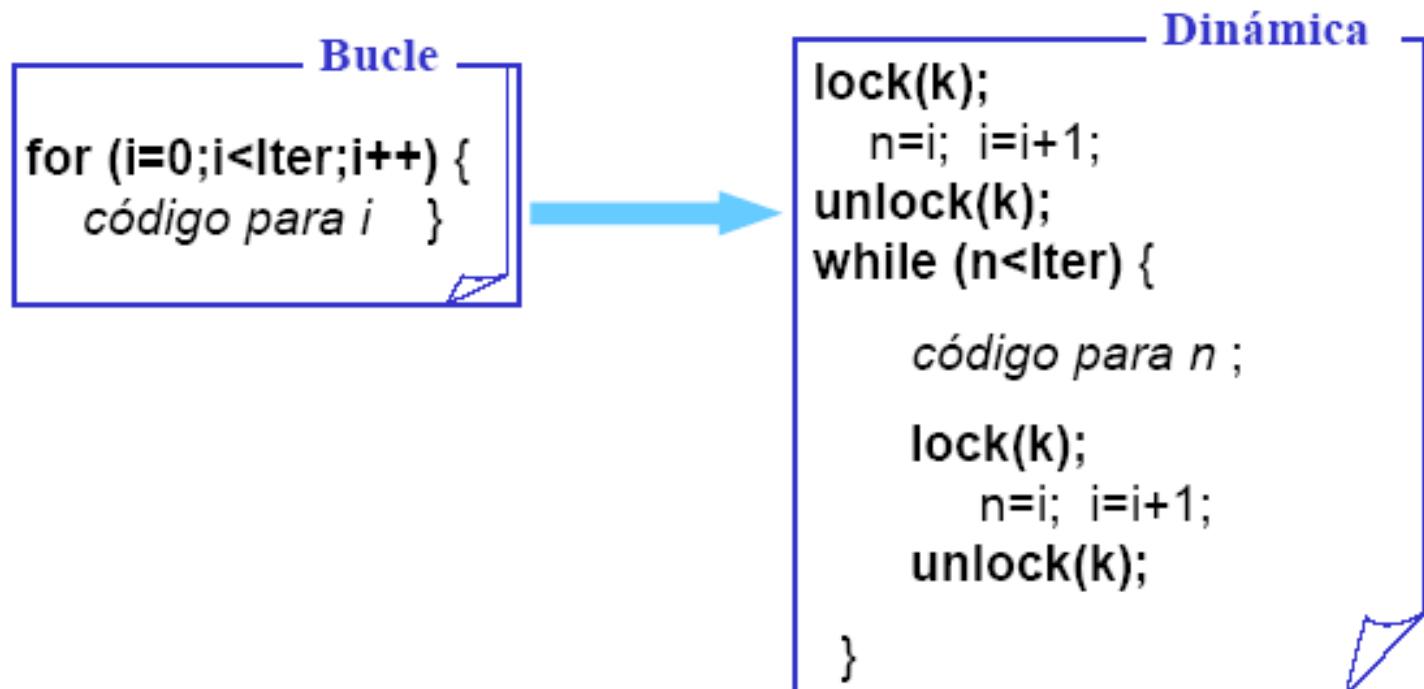
Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Proceso de paralelización: asignación de tareas
 - Asignación dinámica



Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

- Proceso de parallelización: redactar código paralelo
 - Depende de:
 - Estilo de programación: paso de mensajes, etc.
 - Modo de programación
 - Situación inicial
 - Herramienta utilizada para explicitar el paralelismo
 - Estructura del programa
 - Un programa paralelo debe incluir
 - Creación y destrucción de procesos/hebras
 - Asignación de carga de trabajo
 - Comunicación y sincronización

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

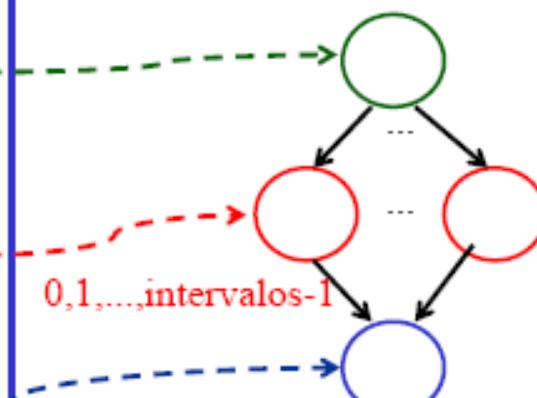
Ejemplo

- Cálculo de Pi por descomposición de tareas

```
main(int argc, char **argv) {
    double ancho, sum;
    int intervalos, i;

    intervalos = atoi(argv[1]);
    ancho = 1.0/(double) intervalos;
    for (i=0;i< intervalos; i++){
        x = (i+0.5)*ancho;
        sum = sum + 4.0/(1.0+x*x);
    }
    sum* = ancho;
}
```

Grafo de dependencias entre tareas



Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

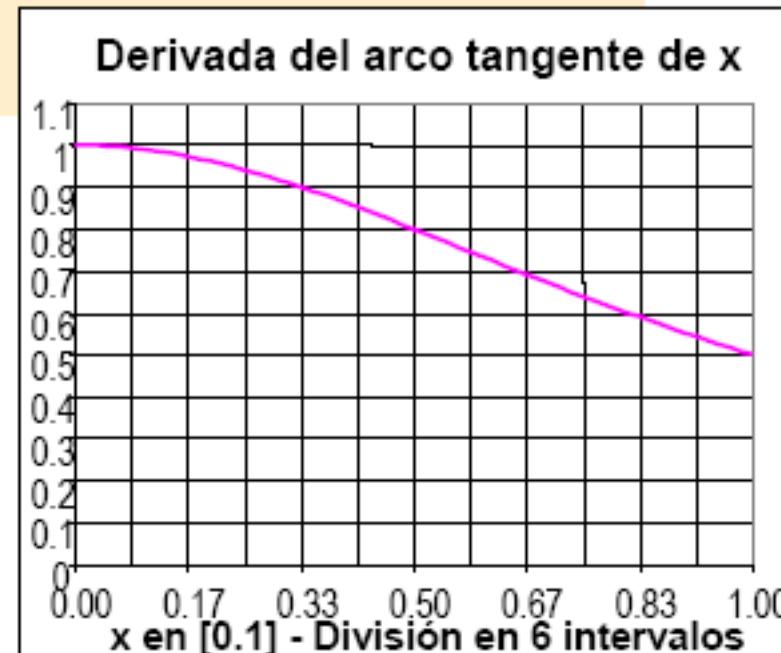
Conceptos

Prog. Paralela

Ejemplo

- Cálculo de Pi por descomposición de tareas

$$\left. \begin{array}{l} \arctg'(x) = \frac{1}{1+x^2} \\ \arctg(1) = \frac{\pi}{4} \\ \arctg(0) = 0 \end{array} \right\} \Rightarrow \int_0^1 \frac{1}{1+x^2} = \arctg(x) \Big|_0^1 = \frac{\pi}{4} - 0$$



- PI se puede calcular por integración numérica.

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

Ejemplo

- Cálculo de Pi por descomposición de tareas
 - Asignación estática de iteraciones del bucle (asignación Round Robin)
 - Redacción de código paralelo
 - Estilo de programación: paso de mensajes
 - Modo de programación: SPMD
 - Situación inicial: versión secuencial
 - Herramienta: MPI
 - Estructura del programa: paralelismo de datos o divide y vencerás

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

Ejemplo

- Cálculo de Pi por descomposición de tareas

```
#include <mpi.h>
main(int argc, char **argv) {
    double ancho,x, sum, tsum; int intervalos, i; int nproc, iproc;
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc); → Enrolar
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    intervalos = atoi(argv[1]); ancho = 1.0 / (double) intervalos; lsum = 0;
    for (i=iproc; i<intervalos; i+=nproc) { → Asignar/
        x = (i + 0.5) * ancho; lsum += 4.0 / (1.0 + x * x); → Localizar
    }
    lsum *= ancho;
    MPI_Reduce(&tsum, &sum, 1, MPI_DOUBLE, → Comunicar/
               MPI_SUM,0,MPI_COMM_WORLD); → sincronizar
    MPI_Finalize(); → Desenrolar
}
```

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

Ejemplo

- Cálculo de Pi por descomposición de tareas
 - Asignación dinámica de iteraciones del bucle
 - Redacción de código paralelo
 - Estilo de programación: directivas
 - Modo de programación: SPMD
 - Situación inicial: versión secuencial
 - Herramienta: OpenMP
 - Estructura del programa: paralelismo de datos o divide y vencerás

Ingeniería de los Computadores

Sesión 6. Paralelismo

Introducción

Conceptos

Prog. Paralela

Ejemplo

- Cálculo de Pi por descomposición de tareas

```
#include <omp.h>
#define NUM_THREADS 4
main(int argc, char **argv) {
    double ancho,x, sum; int intervalos, i;
    intervalos = atoi(argv[1]); ancho = 1.0/(double) intervalos;
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
        #pragma omp for reduction(+:sum) private(x)
            schedule(dynamic)
    Localizar
    for (i=0;i< intervalos; i++) {
        x = (i+0.5)*ancho; sum = sum + 4.0/(1.0+x*x);
    }
    sum* = ancho;
}
```

Crear/Terminar

Comunicar/sincronizar

Asignar

Ingeniería de los Computadores

Sesión 8. Redes de interconexión.
Conceptos y clasificación

Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

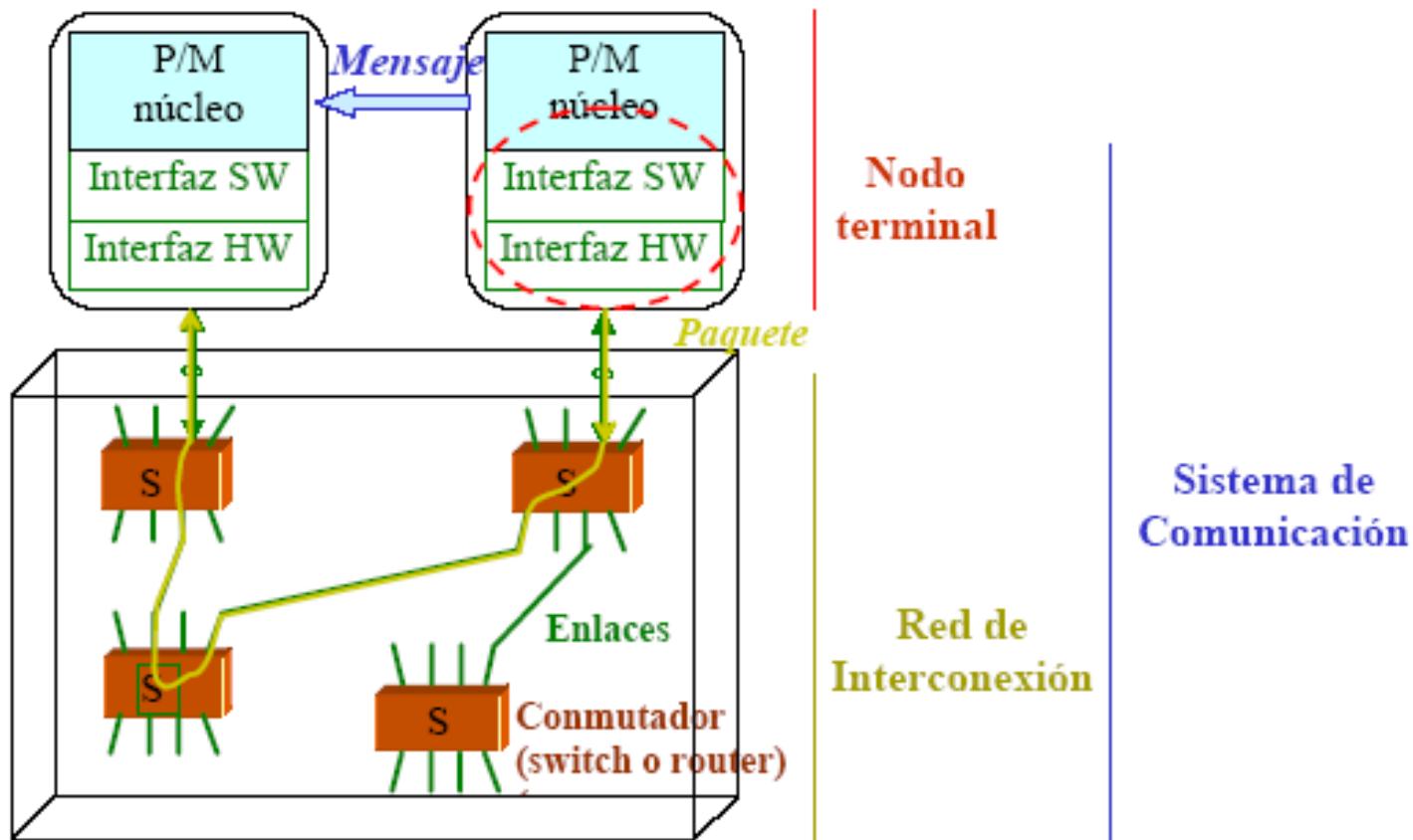
- Redes de Interconexión
 - Elemento fundamental en arquitecturas paralelas con varios elementos de proceso que se comunican
 - Eficiencia en la comunicación crítica: multiprocesadores, multicomputadores
 - Diseño de la red condiciona: escalabilidad de la arquitectura, complejidad, tolerancia a fallos, etc.
 - Aspectos relacionados: control de flujo y encaminamiento

Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

- Estructura general del sistema de comunicación



Conceptos

- Parámetros básicos
 - Tamaño de la red: número de nodos (EPs, memorias, computadores)
 - Grado del nodo (d – degree): número de canales de entrada y salida
 - Nodos unidireccionales: grado de salida y grado de entrada
 - Grado del nodo -> puertos de E/S (¿coste?)
 - Diámetro de red: longitud máxima del camino más corto entre dos nodos cualquiera de una red.

Conceptos

- Parámetros básicos
 - Anchura de la biseción (B): mínimo número de canales que, al cortar, separa la red en dos partes iguales
 - El número de cables que cruzan la biseción es una cota inferior de la densidad de cableado
 - Longitud del cable: efectos sobre la latencia
 - Simetría: Una red es simétrica si es isomorfa a ella misma independientemente del nodo considerado origen
 - Rendimiento
 - Funcionalidad. Indica cómo la red soporta el encaminamiento de datos, tratamiento de las interrupciones, sincronización.
 - Latencia. Indica el retraso de un mensaje

Conceptos

- Parámetros básicos
 - Rendimiento
 - Ancho de banda. Velocidad máxima de transmisión de datos
 - Complejidad hardware. Coste de implementación (cables, conmutadores, conectores, etc.)
 - Escalabilidad. Capacidad de la red para expandirse de forma modular
 - Capacidad de transmisión. Número total de datos que se pueden transmitir a través de la red en una unidad de tiempo. (Punto caliente)

Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

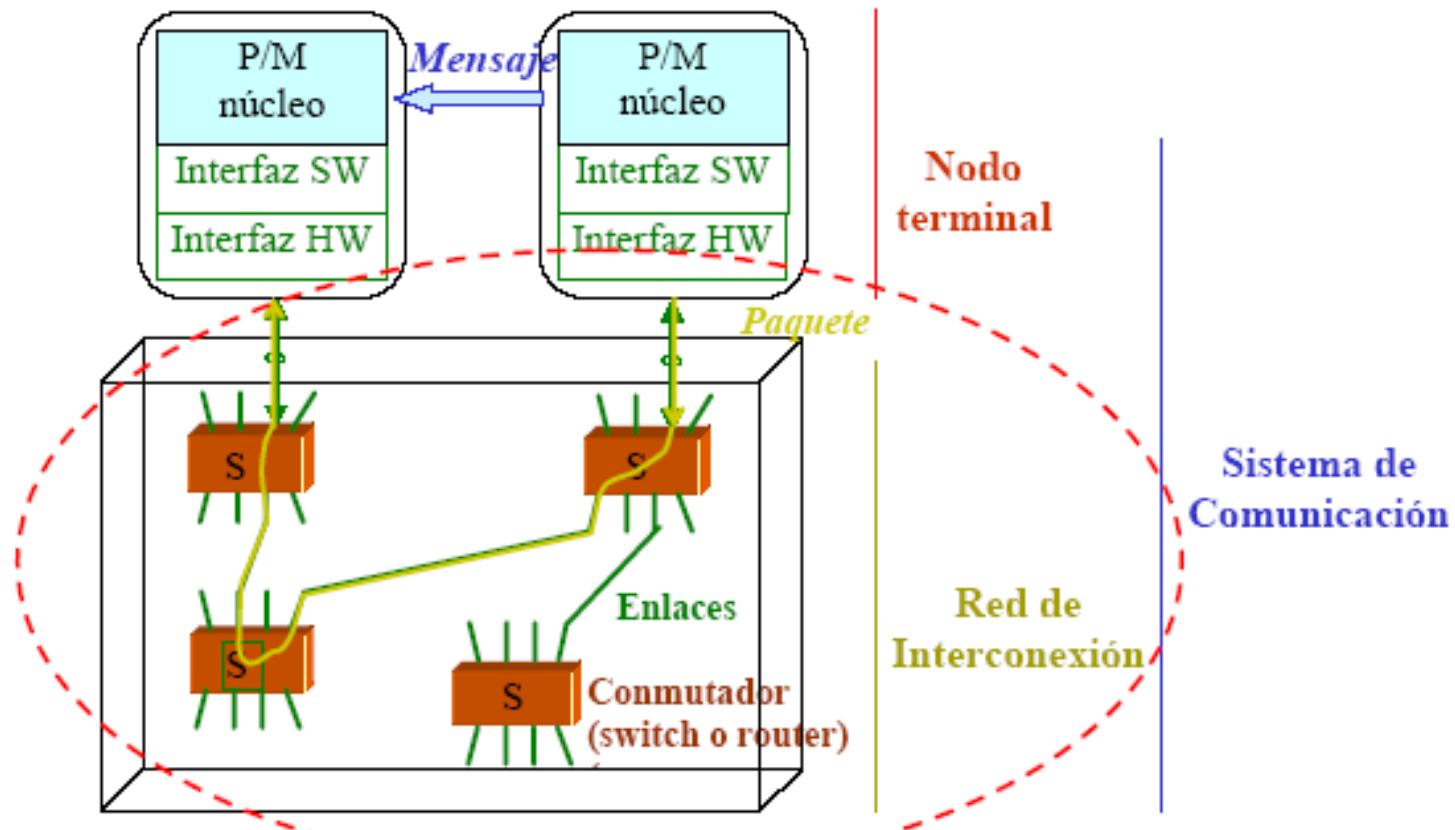
- Diseño de una red de interconexión
 - Topología -> grafo de interconexión
 - Control de flujo -> método usado para regular el tráfico en la red
 - Mensaje
 - Paquete
 - Flit (FLow control unIT)
 - Encaminamiento -> método usado por un mensaje para elegir un camino entre los canales de la red
 - Determinista
 - Adaptativo

Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

- Diseño de una red de interconexión



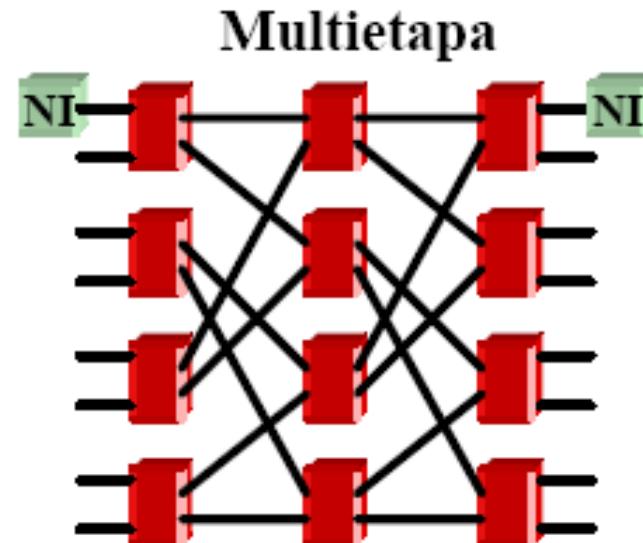
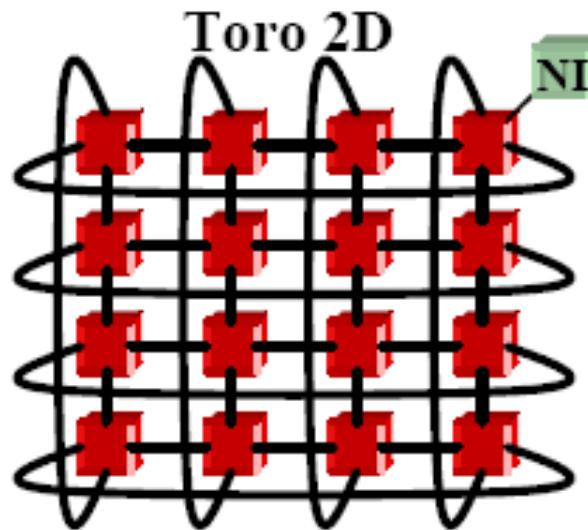
Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

- Diseño de una red de interconexión. Topología

- Estructura** de interconexión física de la red. Se puede modelar mediante un grafo cuyos **vértices** son *comunicadores* o *interfaces de red* (a nodos de cómputo, a módulos de memoria, o a dispositivos de E/S) y los **aristas** son los *enlaces*.



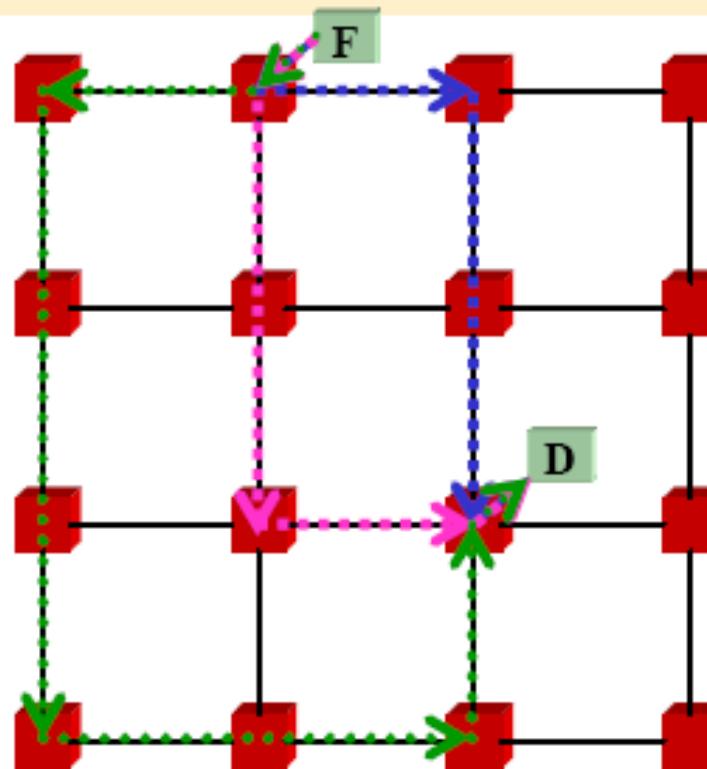
Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

- Diseño de una red de interconexión. Encaminamiento

- Determina el **camino** a seguir por un paquete desde el fuente al destino.

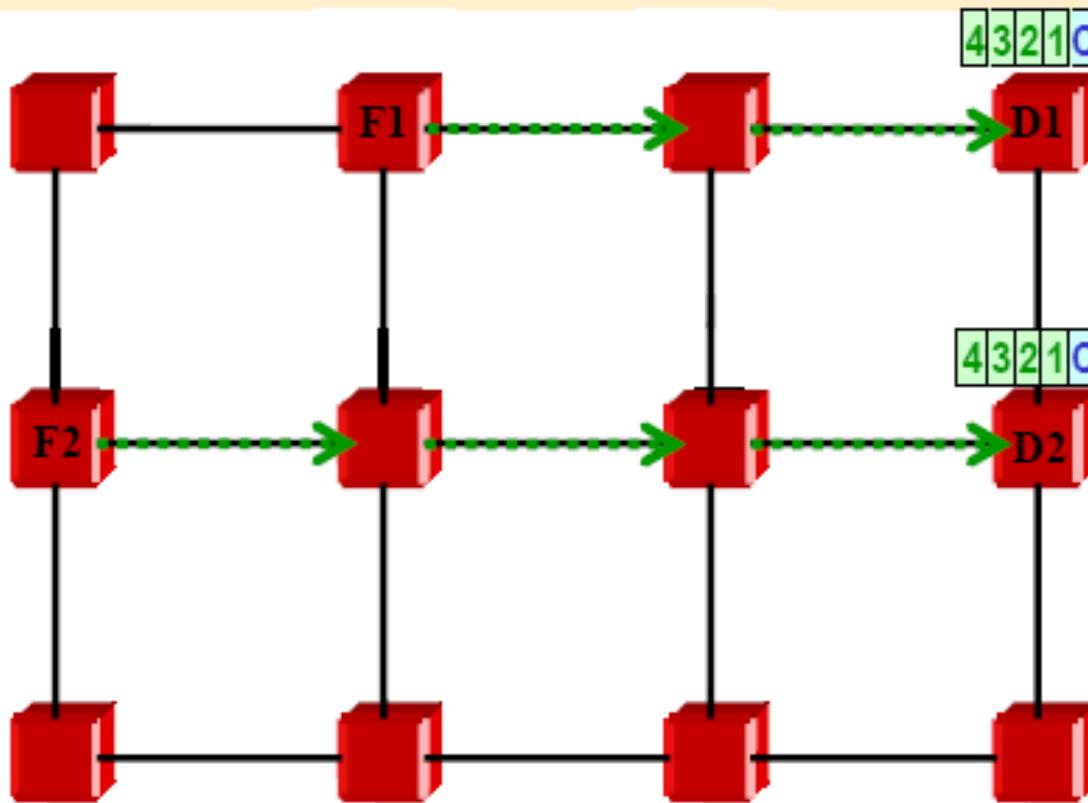


Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

- Diseño de una red de interconexión. Estrategia de conmutación
 - Determina **cómo** los datos en un paquete atraviesan el camino hacia el destino.



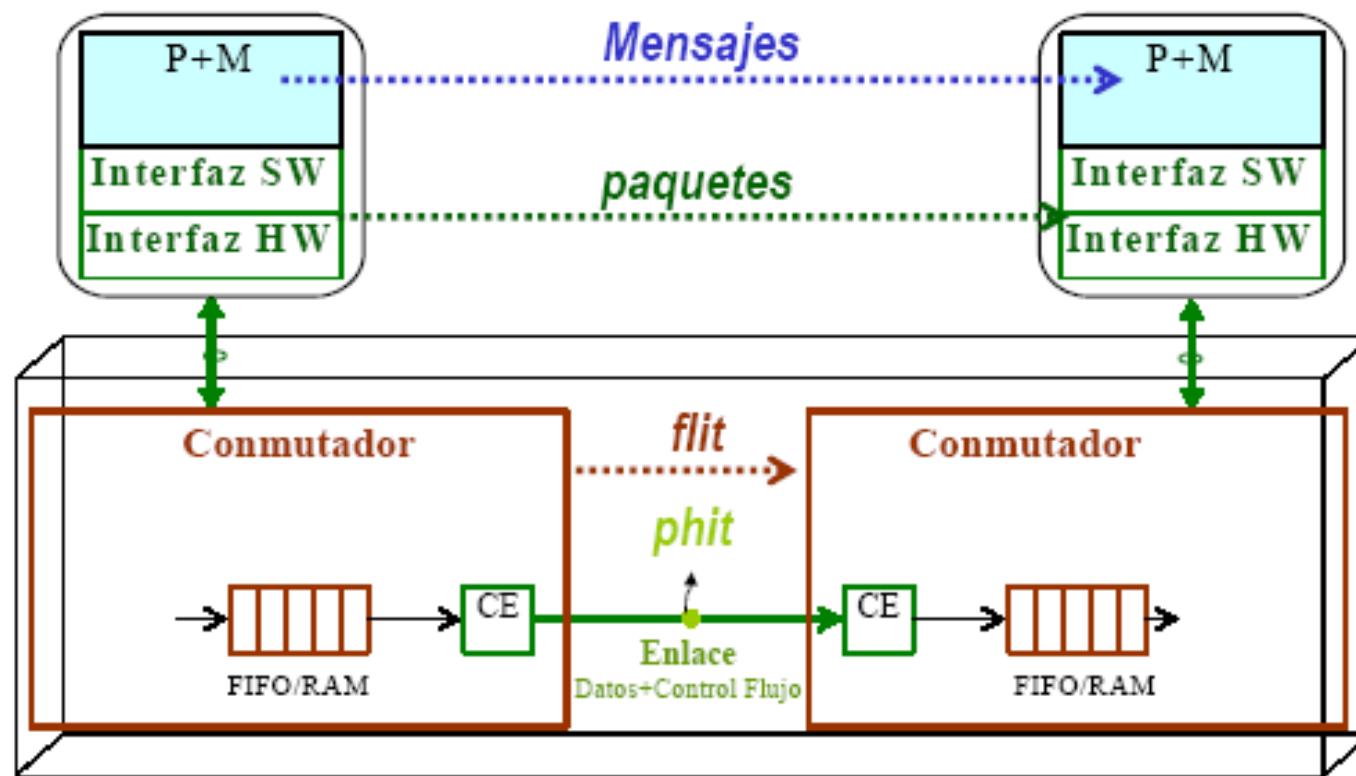
Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

- Diseño de una red de interconexión. Control de flujo

Determinan **cuándo** una unida se mueven entre componentes del Sist. Comunicación, avanzando hacia el destino. **Arbitra** ante colisiones. Determina cómo y cuándo se asignan recursos (intra- e inter-conmutadores)

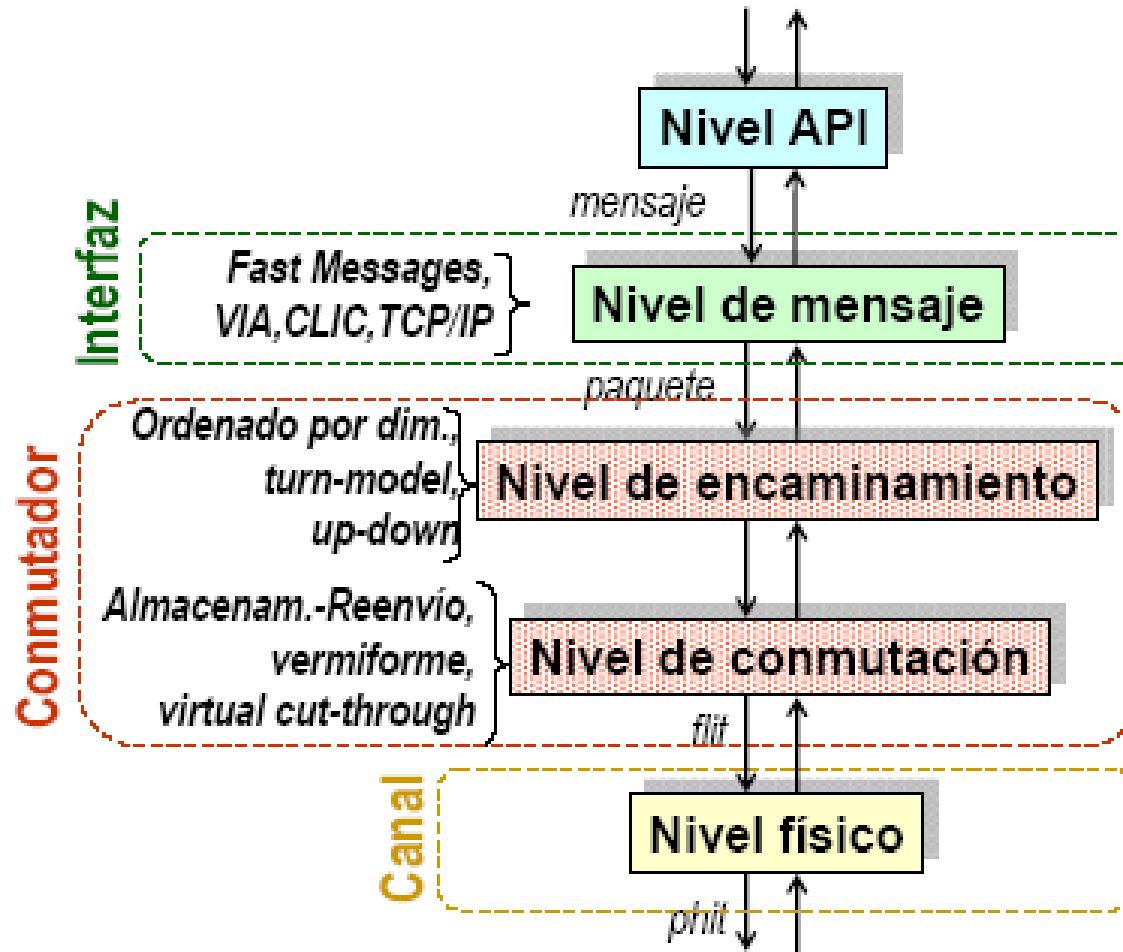


Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

- Diseño de una red de interconexión. Niveles de servicios



Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

Clasificación

- Clasificación de redes de interconexión

CLASES	Nº NODOS Y DISTANCIA	UTILIZACIÓN	DESARROLLO	EJEMPLOS
Diseñadas a medida	Nodos: unos pocos-decenas-cientos-miles	Multiprocesadores Multicomputadores Proc. matriciales	Arquitecturas de altas prestaciones.	-Cray X1 -Origin SGI -Sun Fire 15K
SAN: System Area Network	Nodos: decenas-cientos-miles Dist. decenas o cientos metros	Conecta comp. en habitación Interfaz software "ligera" (<i>lightweight</i>)	Redes a medida y LAN	-Estándares: SCI, Infiniband -OEM: Myrinet, QsNet
LAN: Local Area Network	Nodos: cientos Dist < decenas km	Conecta comp. en edificio o campus	Estaciones de trabajo	-Fast Eth. -Gigabit Eth.
WAN: Wide Area Network	Nodos: miles Dist. miles km	Conecta comp. a nivel mundial	Telecomunicaciones	-ATM

- Clasificación de redes de interconexión
 - Redes de medio compartido
 - Redes de área local
 - Bus de contención (Ethernet)
 - Token bus (Arcnet)
 - Token ring (IBM Token ring)
 - Bus de sistema (backplane bus) (Sun Gigaplane)
 - Redes directas (estáticas basadas en router)
 - Topologías ortogonales (Malla, Toro, Hipercubo)
 - Otras topologías (Árbol, CCC, Estrella, ...)
 - Redes indirectas (dinámicas basadas en conmutador)

- Clasificación de redes de interconexión

- Redes de medio compartido
- Redes directas
- Redes indirectas (dinámicas basadas en conmutador)
 - Topologías regulares
 - Barra cruzada (Crossbar)
 - Redes de interconexión multietapa (MIN)
 - Con bloqueos (unidireccionales y bidireccionales)
 - Sin bloqueos (red de Clos)
 - Topologías irregulares
 - Redes híbridas (redes jerárquicas)

- Redes de medio compartido

- Medio de transmisión compartido
- Arbitraje (resolución de conflictos)
- Sencillo Broadcast
- Ancho de banda limitado (escalabilidad limitada) -> cuello de botella
- Bus de sistema (arquitectura UMA: Proc -> Mem)
- Redes de área local
 - Ethernet (no determinista)
 - Token bus (determinista ☐ aplic. tiempo real)
 - Token ring (estructura en anillo)

Ingeniería de los Computadores

Sesión 8. Redes de interconexión

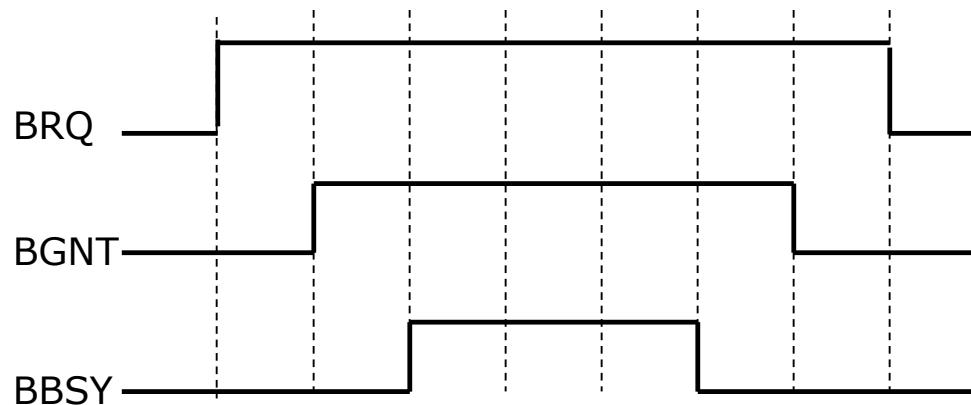
Conceptos

Clasificación

- Redes de medio compartido (arbitraje del bus)

➤ Prioridad estática. Señales de control:

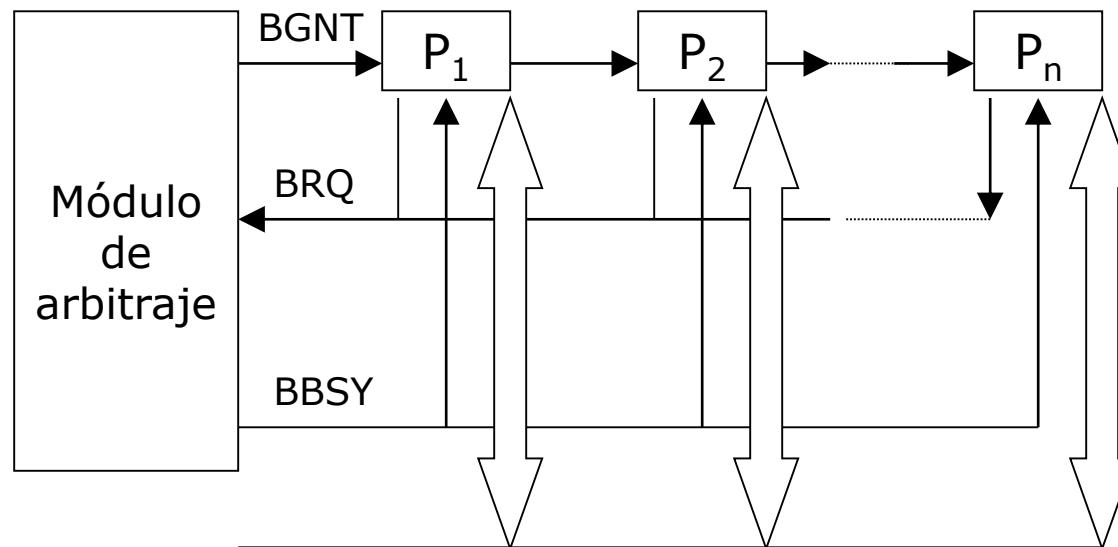
- BRQ
- BGNT
- BBSY común



- Redes de medio compartido (arbitraje del bus)

➤ Prioridad estática. Daisy Chain (centralizada-serie):

- BRQ común
- BGNT propagada
- BBSY común



Ingeniería de los Computadores

Sesión 8. Redes de interconexión

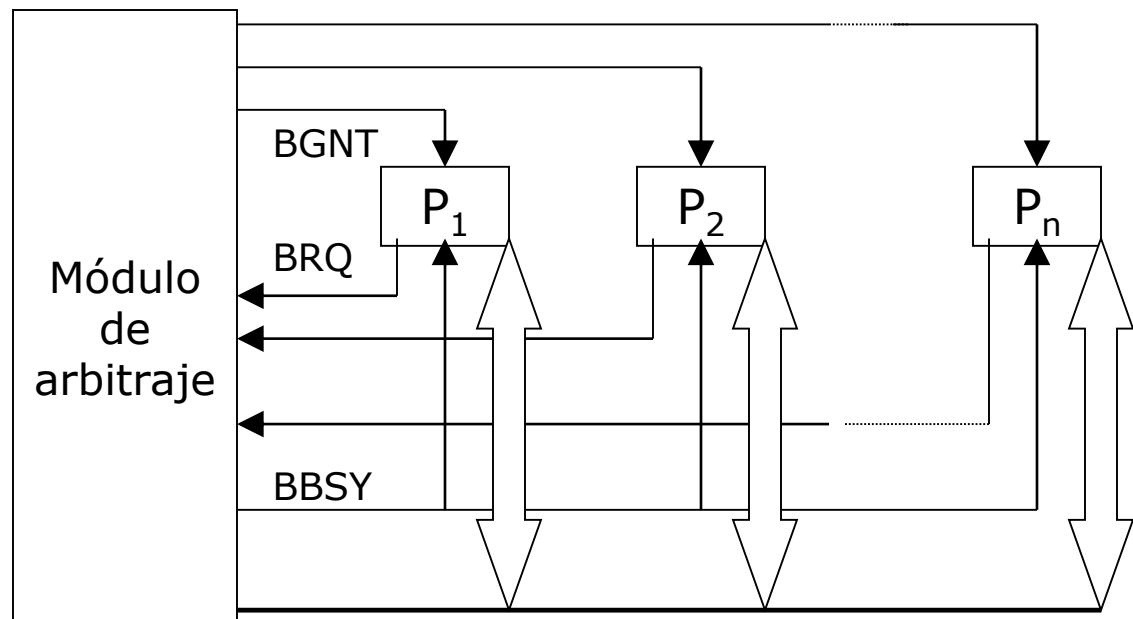
Conceptos

Clasificación

- Redes de medio compartido (arbitraje del bus)

➤ Prioridad estática. Codificador-decodificador de prioridad (centralizada-paralela)

- BRQ individual
- BGNT individual
- BBSY común



Ingeniería de los Computadores

Sesión 8. Redes de interconexión

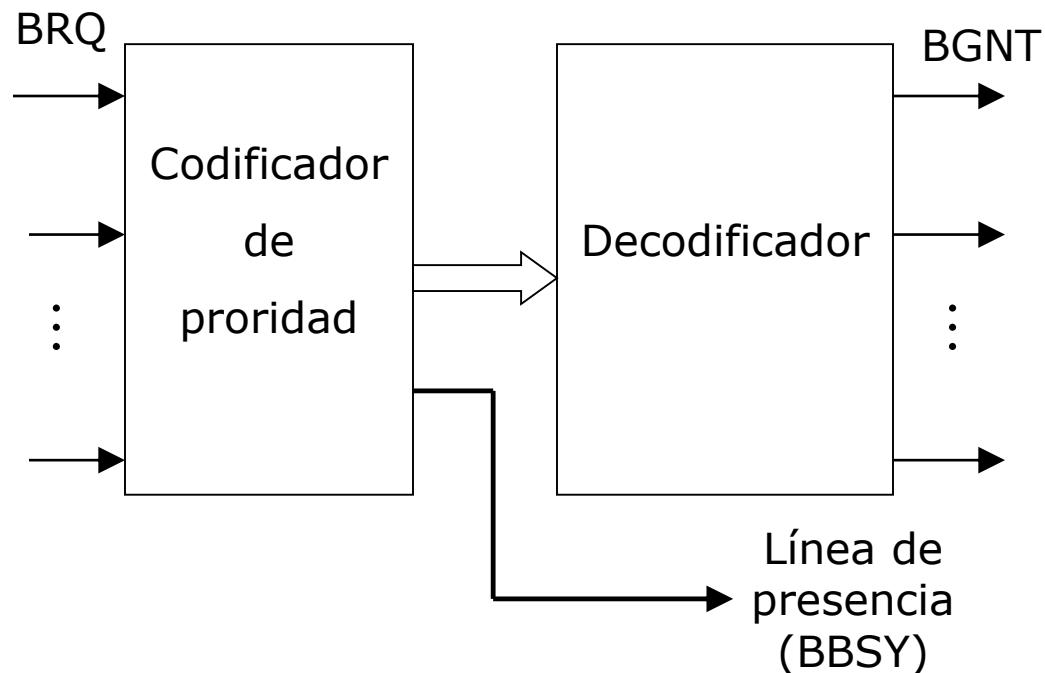
Conceptos

Clasificación

- Redes de medio compartido (arbitraje del bus)

➤ Prioridad estática. Codificador-decodificador de prioridad (centralizada-paralela)

- BRQ individual
- BGNT individual
- BBSY común



Ingeniería de los Computadores

Sesión 8. Redes de interconexión

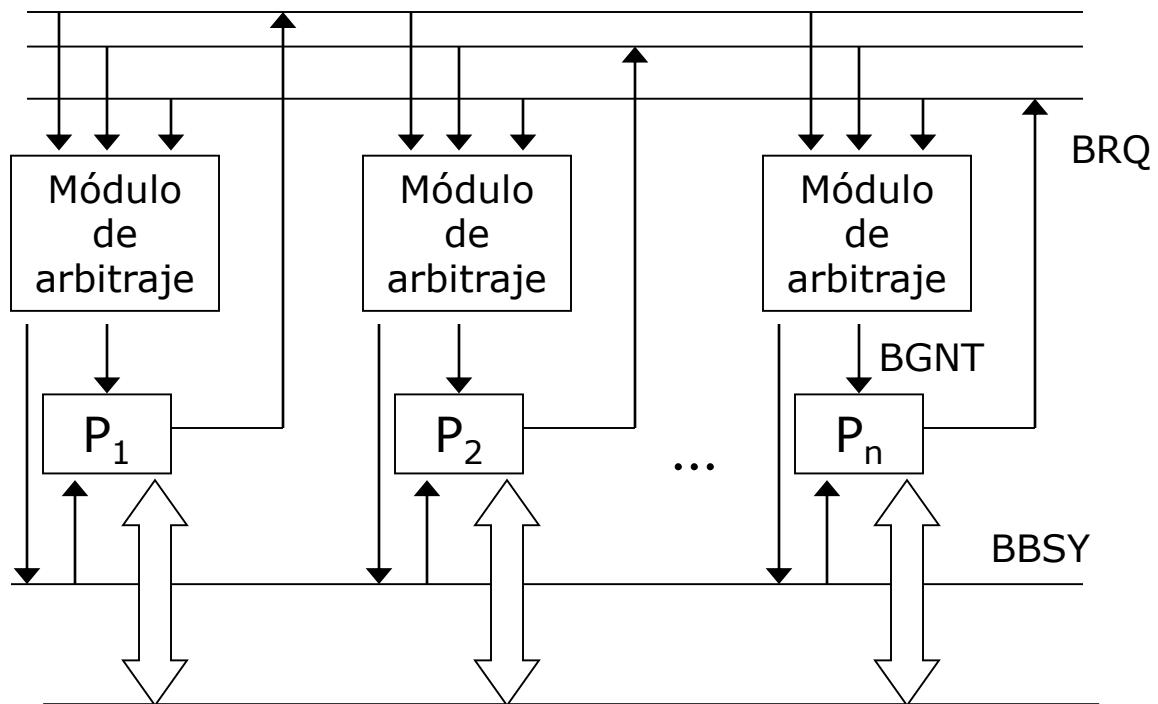
Conceptos

Clasificación

- Redes de medio compartido (arbitraje del bus)

➤ Prioridad estática. Autoarbitraje (distribuido-paralelo)

- BRQ individual
- BGNT individual
- BBSY común



- Redes de medio compartido (arbitraje del bus)
 - Multiplexación temporal
 - Ventajas
 - Asignación equitativa
 - Simplicidad
 - Inconvenientes
 - Infrautilización del ancho de banda
 - Prioridad dinámica
 - LRU
 - RDC (Rotating Daisy Chain)
 - FCFS

Ingeniería de los Computadores

Sesión 8. Redes de interconexión

Conceptos

Clasificación

- Redes de medio compartido (arbitraje del bus)
 - Prioridad dinámica (LRU)

P_0	P_1	P_2	P_3	Acción
0	1	2	3	P_0 utiliza bus
0	1	2	3	P_2 solicita bus
1	2	0	3	P_2 utiliza bus
1	2	0	3	P_1 y P_3 solicitan bus
2	3	1	0	P_3 utiliza bus

- Redes directas
 - Nodos conectados a subconjuntos de nodos
 - Escalabilidad
 - Router -> comunicación entre los nodos
 - Canales unidireccionales o bidireccionales
- Redes indirectas
 - Comunicación a través de conmutadores
 - Topologías regulares (matriciales) e irregulares (NOWs)
- Redes híbridas (combinación de las anteriores)
- Redes multibus
- Redes jerárquicas (jerarquía de buses conectados mediante routers)
- Redes basadas en clusters
 - Nodos conectados (buses – fácil broadcast) formando clusters
 - Clusters conectados entre sí (red directa - escalabilidad)

Ingeniería de los Computadores

Sesión 9. Redes de interconexión.
Topologías

- Redes estáticas o directas
 - Clasificación
 - Estrictamente ortogonales (malla, hipercubo, toro)
 - (Estrictamente) Cada nodo A tiene al menos un enlace en cada dimensión i
 - (Ortogonal) Cada enlace supone un desplazamiento en una dimensión
 - No ortogonales (árbol)
- Propiedades
 - Grado (número de enlaces con otros nodos)
 - Diámetro (máximo camino más corto entre dos nodos)
 - Regularidad (todos los nodos tienen el mismo grado)
 - Simetría (se ve semejante desde cualquier nodo)

Ingeniería de los Computadores

Sesión 9. Redes de interconexión

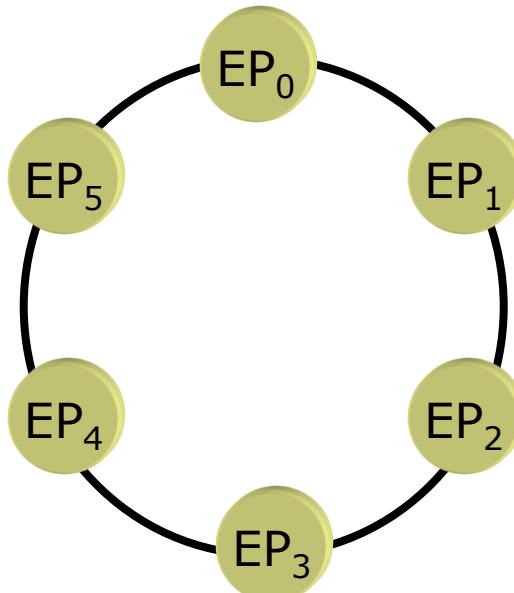
Topologías

Conceptos

Clasificación

Topologías

- Redes estáticas o directas. Anillo unidireccional
 - F. interconexión: $F_{+1}(i) = (i+1) \bmod N$
 - Grado de entrada/salida: 1/1
 - Diámetro: $N-1$



- ¿Anillo bidireccional?

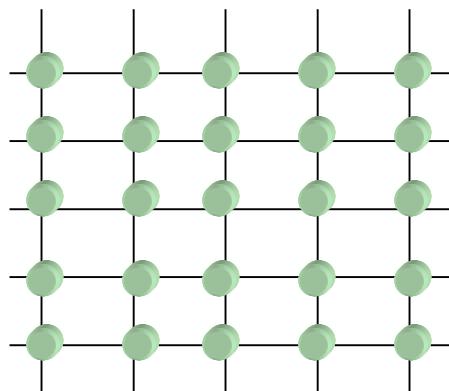
- Redes estáticas o directas. Malla abierta

➤ F. interconexión:

- $F_{+1}(i) = (i+1)$ si $i \bmod r <> r-1$
- $F_{-1}(i) = (i-1)$ si $i \bmod r <> 0$
- $F_{+r}(i) = (i+r)$ si $i \bmod r <> r-1$
- $F_{-r}(i) = (i-r)$ si $i \bmod r <> 0$

➤ Grado: 4

➤ Diámetro: $2(r-1)$, donde $N=r^2$



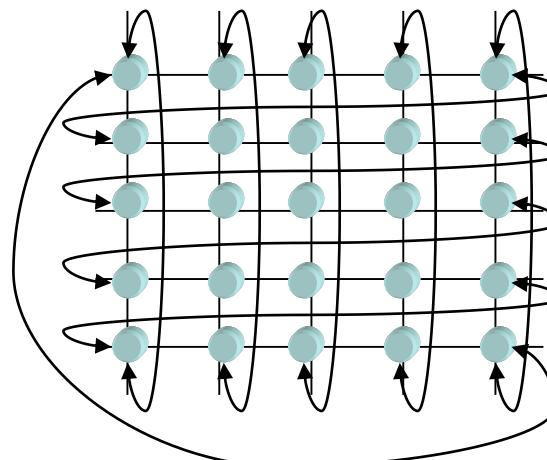
- Redes estáticas o directas. Malla Illiac

- F. interconexión:

- $F_{+1}(i) = (i+1) \bmod N$
 - $F_{-1}(i) = (i-1) \bmod N$
 - $F_{+r}(i) = (i+r) \bmod N$
 - $F_{-r}(i) = (i-r) \bmod N$

- Grado: 4

- Diámetro: $(r-1)$, donde $N=r^2$



Ingeniería de los Computadores

Sesión 9. Redes de interconexión

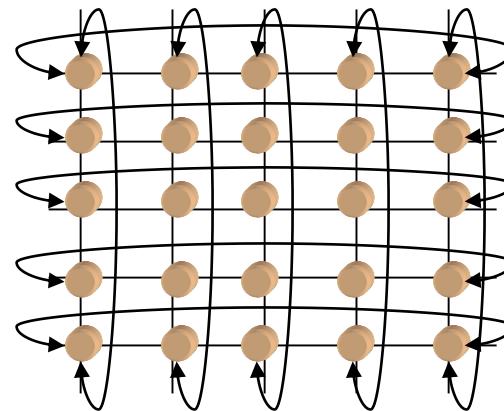
Topologías

Conceptos

Clasificación

Topologías

- Redes estáticas o directas. Redes n-cubos k-arias ó toros
 - n dimensiones, k nodos
 - F. interconexión toro 2D:
 - $F_{+1}(i) = (i+1) \bmod r + (i \text{ DIV } r) \cdot r$
 - $F_{-1}(i) = (i-1) \bmod r + (i \text{ DIV } r) \cdot r$
 - $F_{+r}(i) = (i+r) \bmod N$
 - $F_{-r}(i) = (i-r) \bmod N$
 - Grado: 4
 - Diámetro: $2 \cdot \left\lfloor \frac{r}{2} \right\rfloor$, donde $N=r^2$



Ingeniería de los Computadores

Sesión 9. Redes de interconexión

Topologías

Conceptos

Clasificación

Topologías

- Redes estáticas o directas. Desplazador barril

➤ F. interconexión:

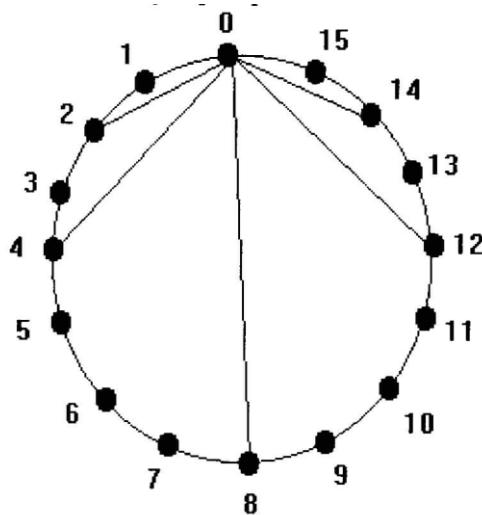
$$\triangleright B_{+k}(i) = (i + 2^k) \bmod N$$

$$\triangleright B_{-k}(i) = (i - 2^k) \bmod N$$

$$\triangleright K=0 \dots n-1, n=\log N, i=0 \dots N-1$$

➤ Grado: $2n - 1$

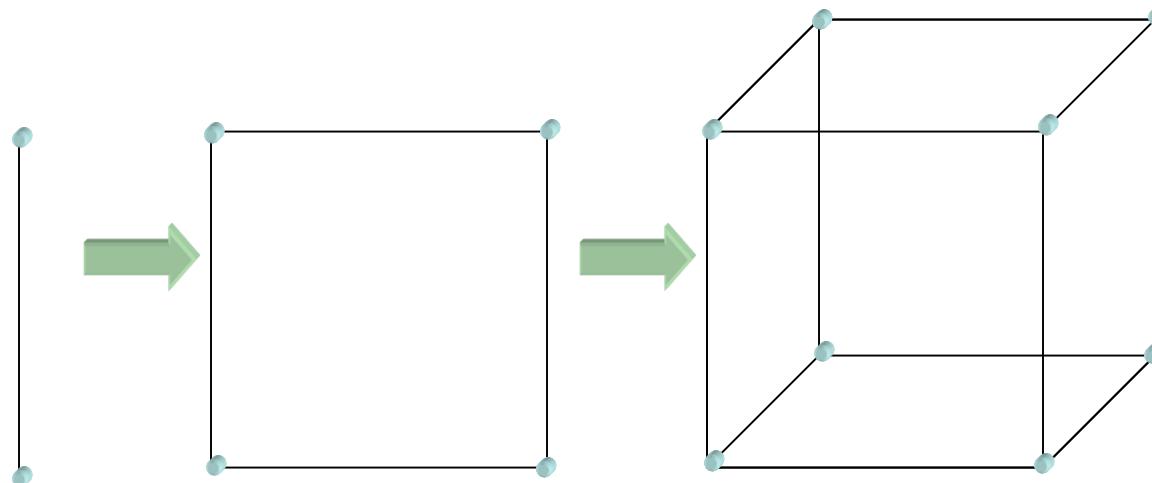
➤ Diámetro: $n/2$



- Redes estáticas o directas. Hipercubo

- F. interconexión:

- $F_i(h_{n-1}, \dots, h_i, \dots, h_0) = h_{n-1}, \dots, \bar{h}_i, \dots, h_0$
 - Grado: n ($n=\log N$)
 - Diámetro: n



Ingeniería de los Computadores

Sesión 9. Redes de interconexión

Topologías

Conceptos

Clasificación

- Redes estáticas o directas. Ciclo cubo conectado (CCC) (red jerárquica)



Ingeniería de los Computadores

Sesión 9. Redes de interconexión

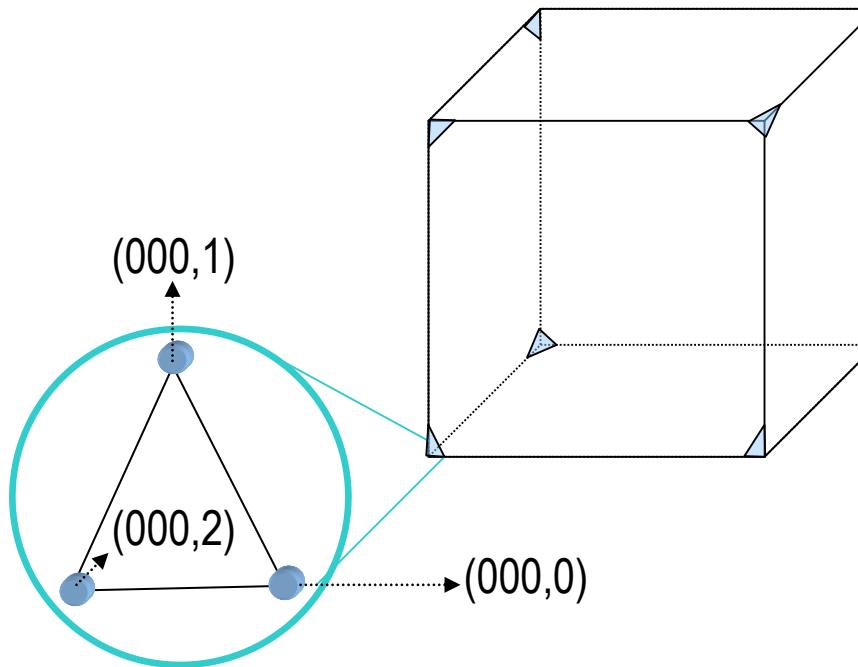
Topologías

Conceptos

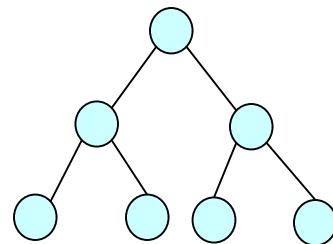
Clasificación

Topologías

- Redes estáticas o directas. Red CCC



- Redes estáticas o directas. Árbol binario
 - Balanceado: todas las ramas del árbol tienen la misma longitud
 - Cuello de botella → nodo raíz
 - N (balanceado) = $2^k - 1$ (k = niveles del árbol)
 - Grado: 3
 - Diámetro: $2(k-1)$



- Redes indirectas o dinámicas
 - Uso de conmutadores y árbitros
 - Ejemplos
 - Redes crossbar
 - Redes de conexión multietapa (MIN)
 - Modelo: $G(N,C)$
 - N, conjunto de conmutadores
 - C, enlaces (unidireccionales o bidireccionales) entre conmutadores
 - Canal bidireccional → dos canales unidireccionales
 - Un conmutador puede tener conectados 0, 1 ó más elementos (Procesadores, memorias, etc.)
 - Distancia entre dos nodos: distancia entre los conmutadores que conectan los nodos más 2.

Ingeniería de los Computadores

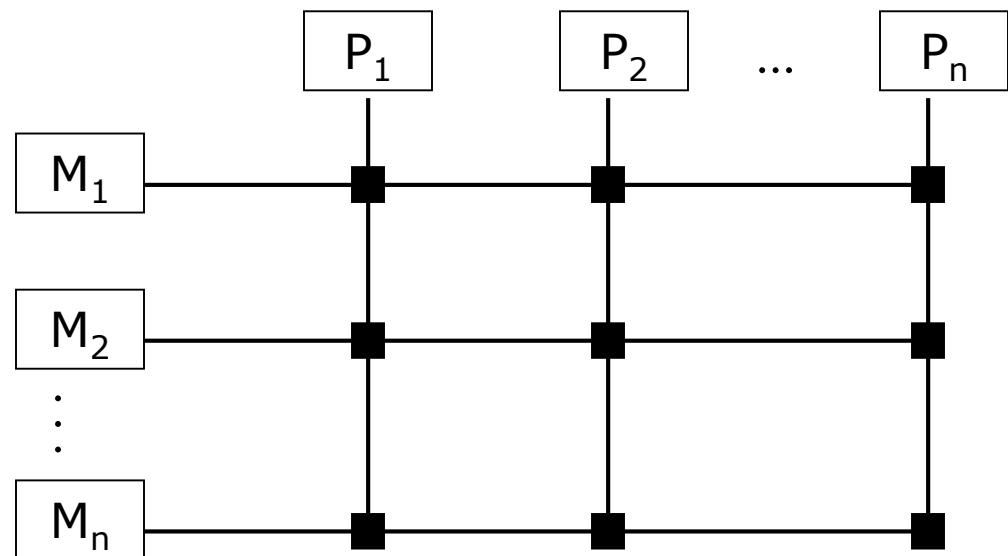
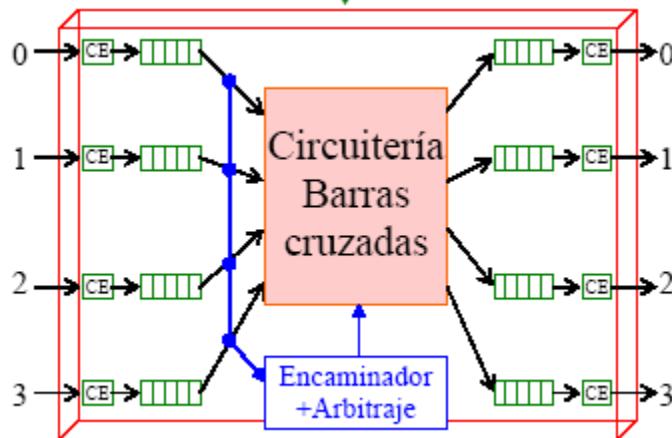
Sesión 9. Redes de interconexión

Topologías

Conceptos

Clasificación

- Redes indirectas o dinámicas. Redes crossbar
 - Conexión directa nodo-nodo
 - Gran ancho de banda y capacidad de interconexión
 - Conexión Proc. – Mem. → limitado por los accesos a memoria (columnas)
 - Conexión Proc(N) – Proc(N) → máximo de N conexiones
 - Coste elevado: $O(N \cdot M)$



Ingeniería de los Computadores

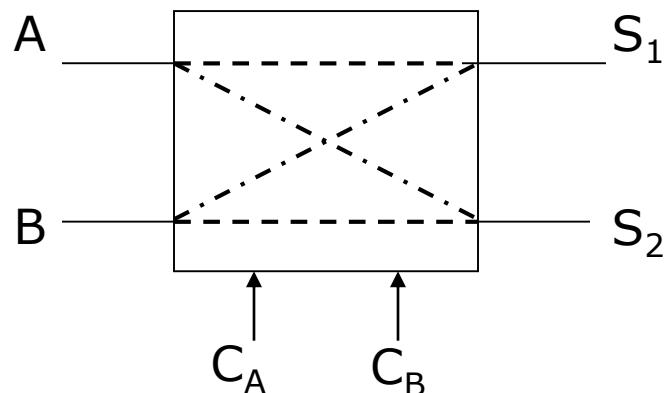
Sesión 9. Redes de interconexión

Topologías

Conceptos

Clasificación

- Redes indirectas o dinámicas. Redes MIN
 - Conectan dispositivos de entrada con dispositivos de salida mediante un conjunto de etapas de conmutadores, donde cada conmutador es una red crossbar.
 - Concentradores → nº entradas > nº salidas
 - Expansores → nº salidas > nº entradas



Ingeniería de los Computadores

Sesión 9. Redes de interconexión

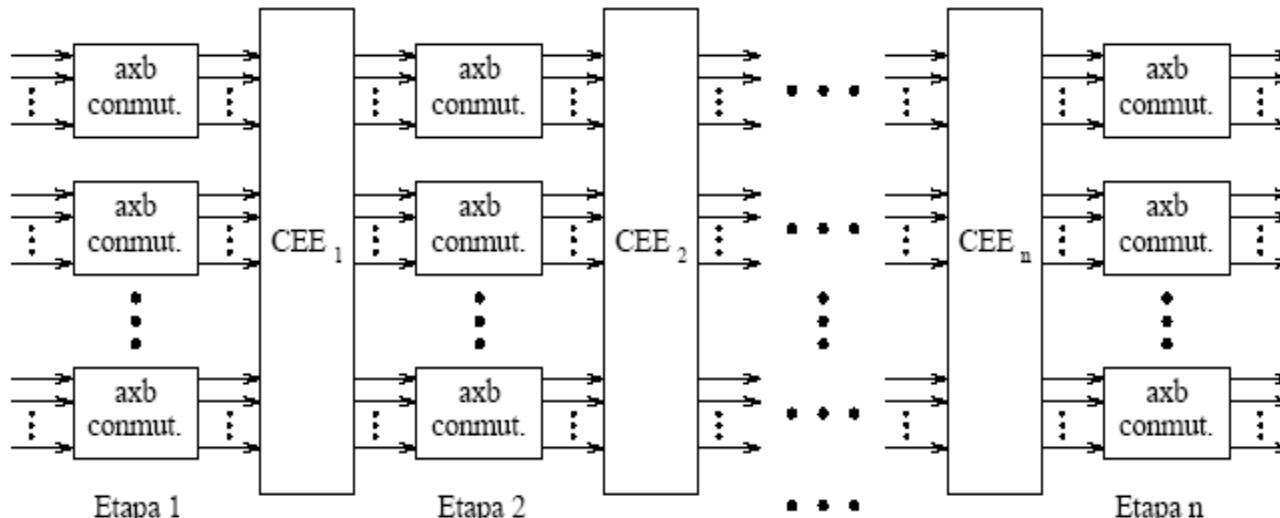
Conceptos

Clasificación

Topologías

- Redes indirectas o dinámicas. Redes MIN
 - Conexión de etapas adyacentes → Patrón de conexión
 - Patrón basado en permutaciones: conmutadores con el mismo número de entradas y salidas.
 - Ejemplo: barajado perfecto.

$$B(a_{n-1}, a_{n-2}, \dots, a_0) = (a_{n-2}, \dots, a_0, a_{n-1})$$



Ingeniería de los Computadores

Sesión 9. Redes de interconexión

Conceptos

Clasificación

Topologías

- Redes indirectas o dinámicas. Redes MIN
 - Número de entradas a^n y número de salidas b^n (red $a^n \times b^n$)
 - n etapas de conmutadores (C_0, C_1, \dots, C_{n-1})
 - Conmutadores $a \times b$
 - $a^{n-1-i} \times b^i$ conmutadores en la etapa C_i
 - Funcionalidad de los conmutadores: barras cruzadas, reducción, difusión
 - Subred de interconexión entre etapas: R_0, R_1, \dots
 - Tipos de canales: unidireccionales, bidireccionales

Ingeniería de los Computadores

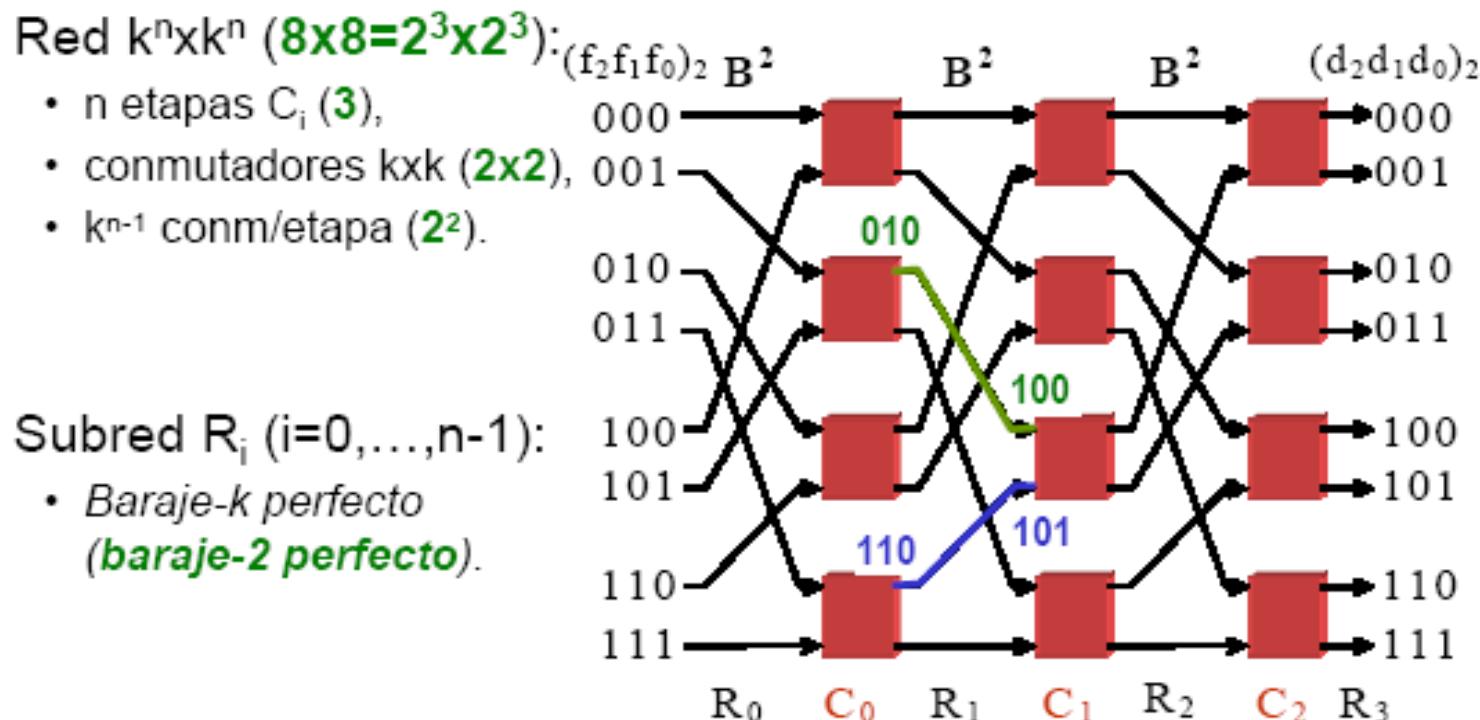
Sesión 9. Redes de interconexión

Conceptos

Clasificación

Topologías

- Redes indirectas o dinámicas. Redes MIN – red Omega
 - El patrón de conexión C_i es una permutación k-baraje perfecto a excepción del último (R_n) que es permutación 0



Subred R_i ($i=0, \dots, n-1$):

- Baraje-k perfecto
(baraje-2 perfecto).

$$B^k ((f_{n-1}, f_{n-2}, \dots, f_1, f_0)_k) = (f_{n-2}, \dots, f_1, f_0, f_{n-1})_k$$

$$B^2 ((f_2, f_1, f_0)_2) = (f_1, f_0, f_2)_2$$

Ingeniería de los Computadores

Sesión 9. Redes de interconexión

Conceptos

Clasificación

Topologías

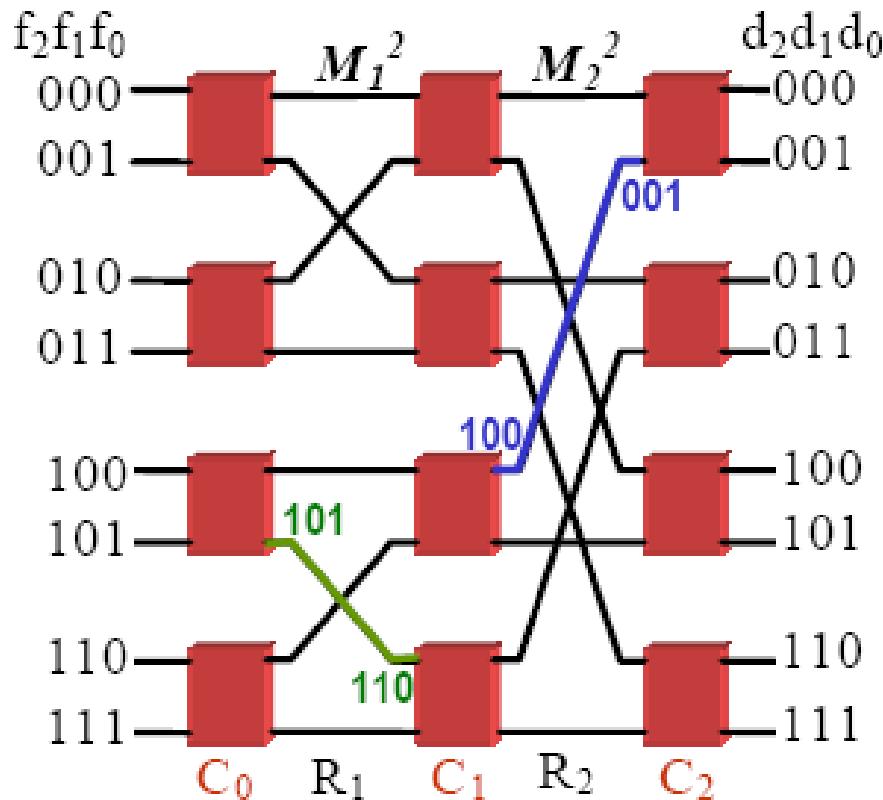
- Redes indirectas o dinámicas. Redes MIN – red mariposa

– Red $k^n \times k^n$ ($8 \times 8 = 2^3 \times 2^3$):

- n etapas C_i (3),
- comutadores $k \times k$ (2×2),
- k^{n-1} comm/etapa (2^2).

– Subred R_i ($i=0, \dots, n-1$):

- Mariposa M_i^k



$$M_i^k ((f_{n-1}, f_{n-2}, \dots, f_{i+1}, \underline{f_i}, f_{i-1}, \dots, f_1, \underline{f_0})_k) = (f_{n-1}, f_{n-2}, \dots, f_{i+1}, \underline{f_0}, f_{i-1}, \dots, f_1, \underline{f_i})_k$$

$$i=0, \dots, n-1$$

$$M_2^2 ((\underline{f_2}, f_1, \underline{f_0})_2) = (\underline{f_0}, f_1, \underline{f_2})_2$$

Ingeniería de los Computadores

Sesión 9. Redes de interconexión

Conceptos

Clasificación

Topologías

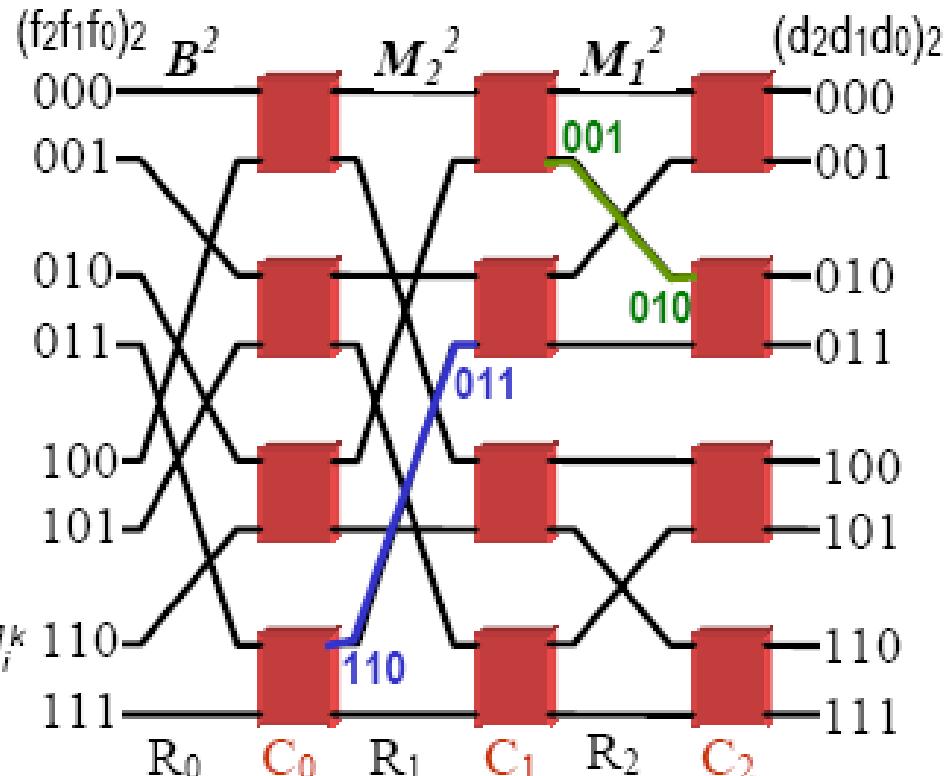
- Redes indirectas o dinámicas. Redes MIN – red cubo

- Red $k^n \times k^n$ ($8 \times 8 = 2^3 \times 2^3$):

- n etapas C_i (3),
- commutadores $k \times k$ (2x2),
- k^{n-1} comm/etapa (2^2).

- Subred R_i ($i=0, \dots, n-1$):

- R_0 : Baraje-k perfecto
(baraje-2 perfecto).
- R_{n-i} ($i=1, \dots, n-1$): Mariposa M_i^k



$$M_i^k ((f_{n-1}, f_{n-2}, \dots, f_{i+1}, \textcolor{blue}{f}_i, f_{i-1}, \dots, f_1, \textcolor{blue}{f}_0)_k) = (f_{n-1}, f_{n-2}, \dots, f_{i+1}, \textcolor{blue}{f}_0, f_{i-1}, \dots, f_1, \textcolor{blue}{f}_i)_k$$

$$i=0, \dots, n-1$$

$$M_1^2 ((f_2, \textcolor{blue}{f}_1, \textcolor{blue}{f}_0)_2) = (f_2, \textcolor{blue}{f}_0, f_1)_2$$

Ingeniería de los Computadores

Sesión 9. Redes de interconexión

Conceptos

Clasificación

Topologías

- Redes indirectas o dinámicas. Redes MIN – red delta

Red $a^n \times b^n$ ($16 \times 9 = 4^2 \times 3^2$):

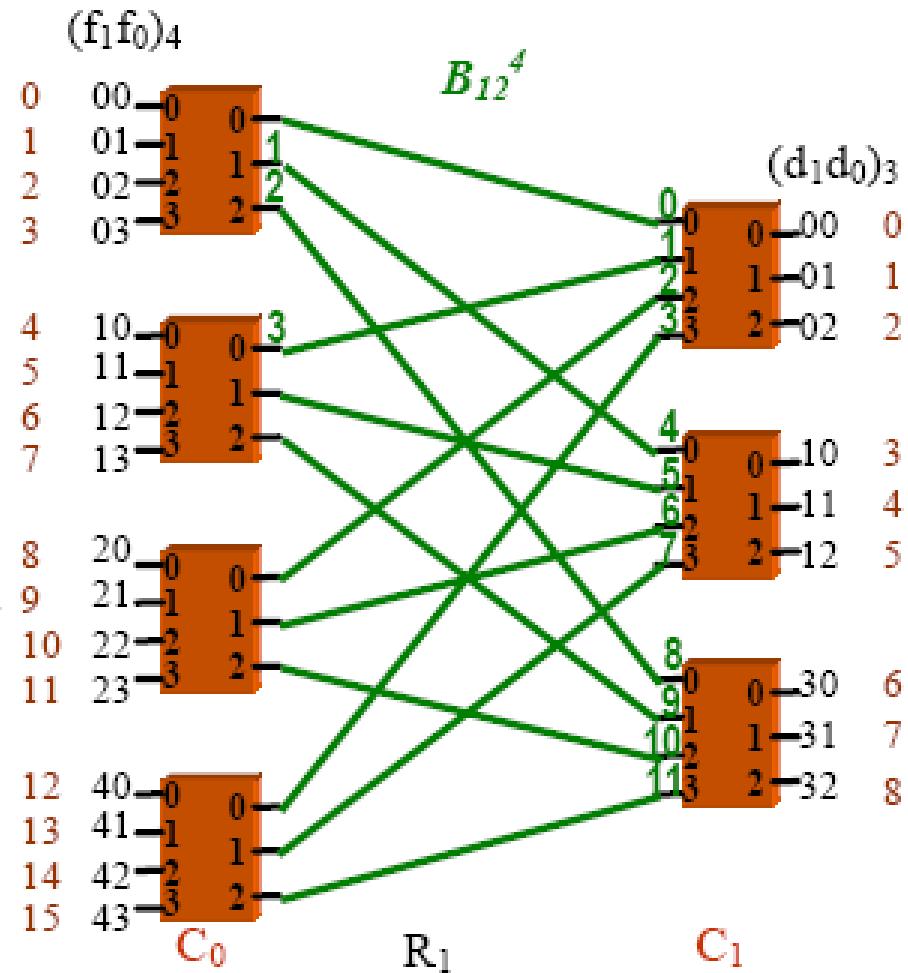
- n etapas C_i (2),
- conmutadores $a \times b$ (4×3),
- $a^{n-1-i} \cdot b^i$ conn / C_i (4, 3).

Subred R_i ($i=0 \text{ o } 1, \dots, n-1$):

- Baraje-a de c elementos
- R_1 : (**baraje-4 de 12 elementos**)

$$B_c^a(s) = \begin{cases} a \cdot s \bmod (c-1) & \text{si } s < c-1 \\ c-1 & \text{si } s = c-1 \end{cases}$$

$$B_{12}^4(s) = \begin{cases} 4 \cdot s \bmod (11) & \text{si } s < 11 \\ 11 & \text{si } s = 11 \end{cases}$$



Ingeniería de los Computadores

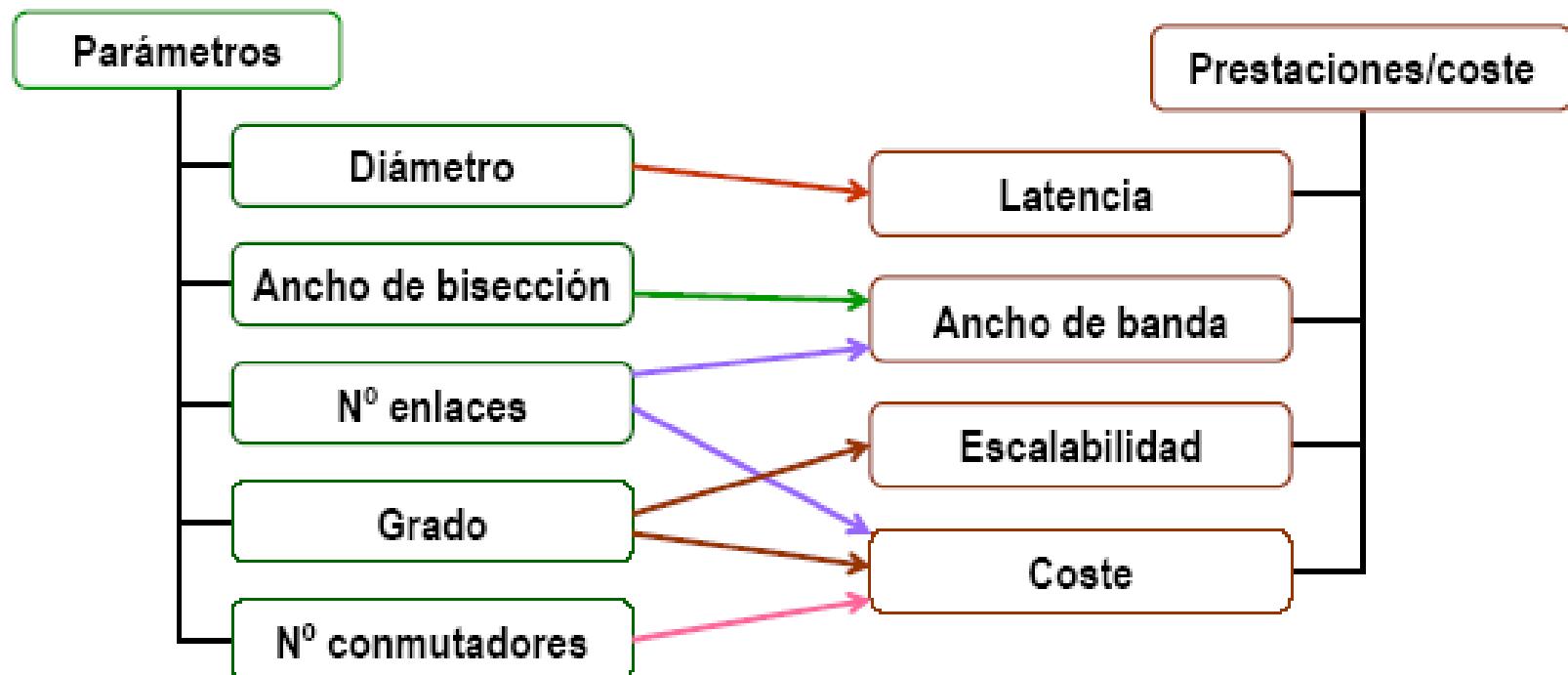
Sesión 9. Redes de interconexión

Conceptos

Clasificación

Topologías

- Prestaciones



Ingeniería de los Computadores

Sesión 10. Redes de interconexión.
Técnicas de conmutación

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

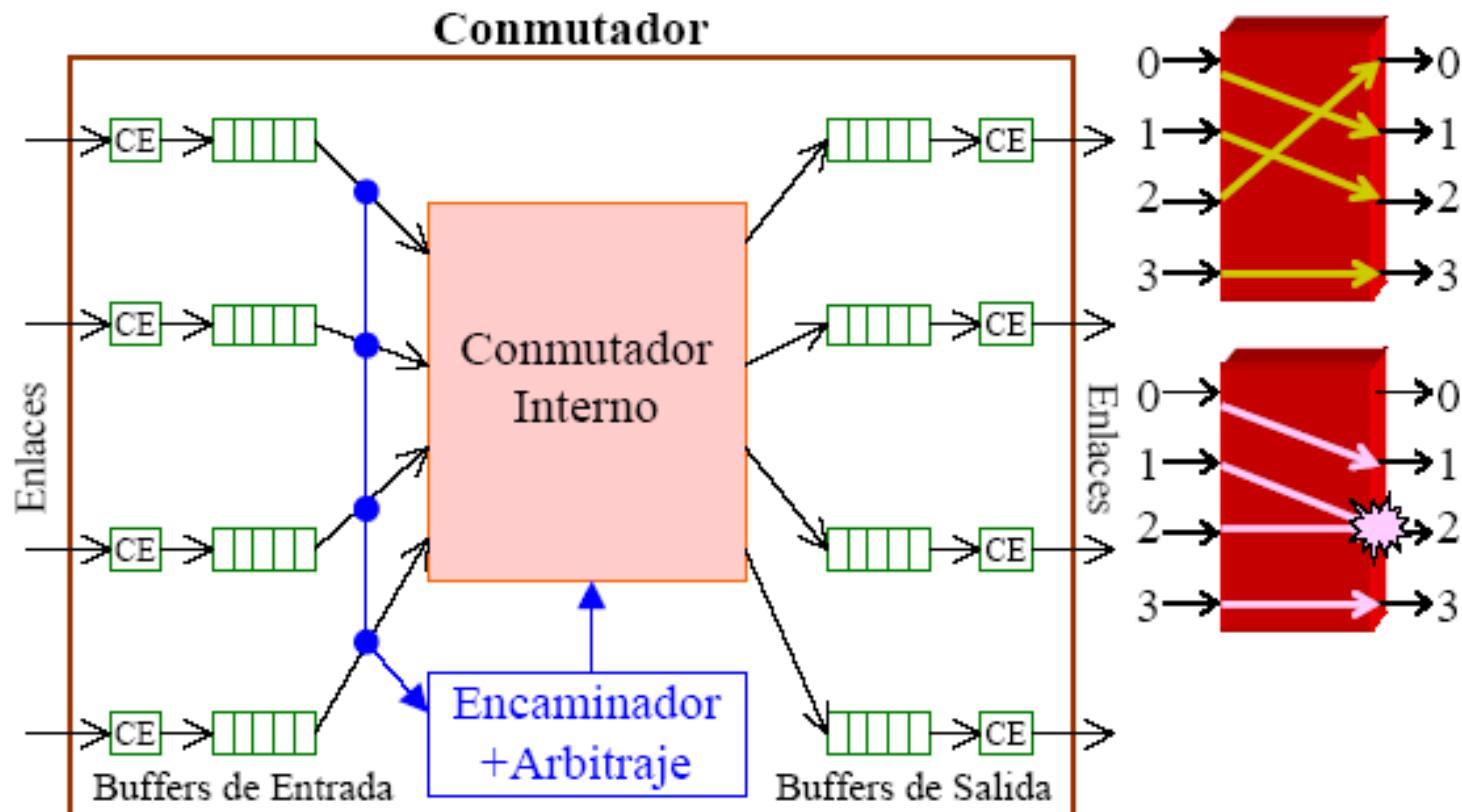
Conceptos

Clasificación

Topologías

Conmutación

- Básico en las topologías



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

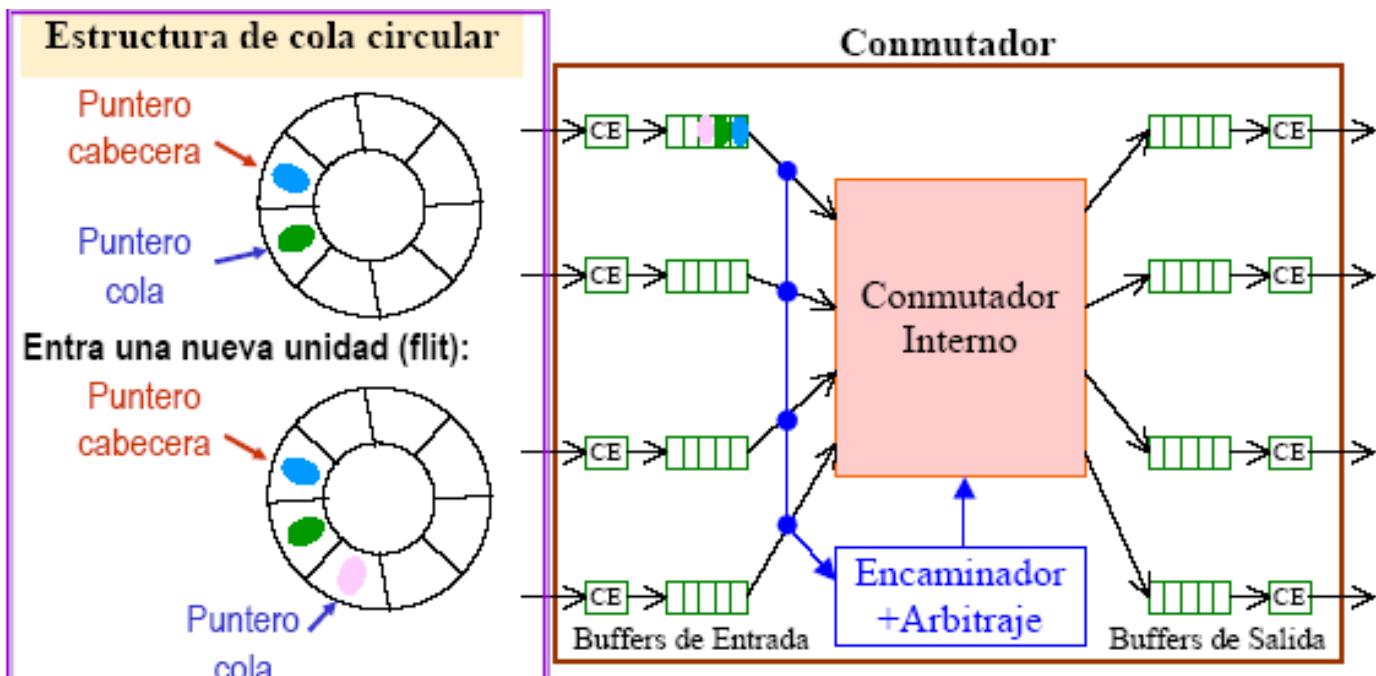
Conceptos

Clasificación

Topologías

Conmutación

- Buffers de entrada



Otra alternativa es una estructura de datos de lista enlazada

Coste de gestión mayor. Además de actualizar punteros de cabecera y cola hay que modificar punteros entre celdas y gestionar una lista con celdas libres.

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

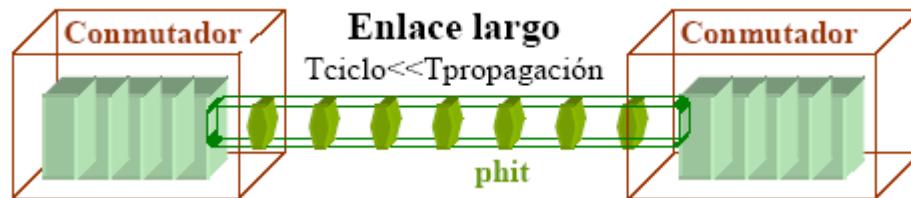
Conceptos

Clasificación

Topologías

Conmutación

- Enlaces y canales.
 - Infraestructura: hilos eléctricos (cobre), fibras ópticas, etc.
- Anchura
 - Anchos. Se transmite simultáneamente datos y control
 - Estrechos. Multiplexa en el tiempo datos y control
- Longitud
 - Cortos. 1 símbolo
 - Largos. Varios símbolos de forma simultánea



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

- Enlaces: longitud
 - Cortos
 - El ciclo de red depende del retardo de propagación
 - Largos
 - Ciclo de red << retardo de propagación
- Velocidad del canal depende:
 - Energía empleada para transmitir por una línea
 - Distancia a atravesar
 - Ruido
 - Desplazamiento entre líneas de un enlace
 - Tamaño del buffer destino (enlaces largos)

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

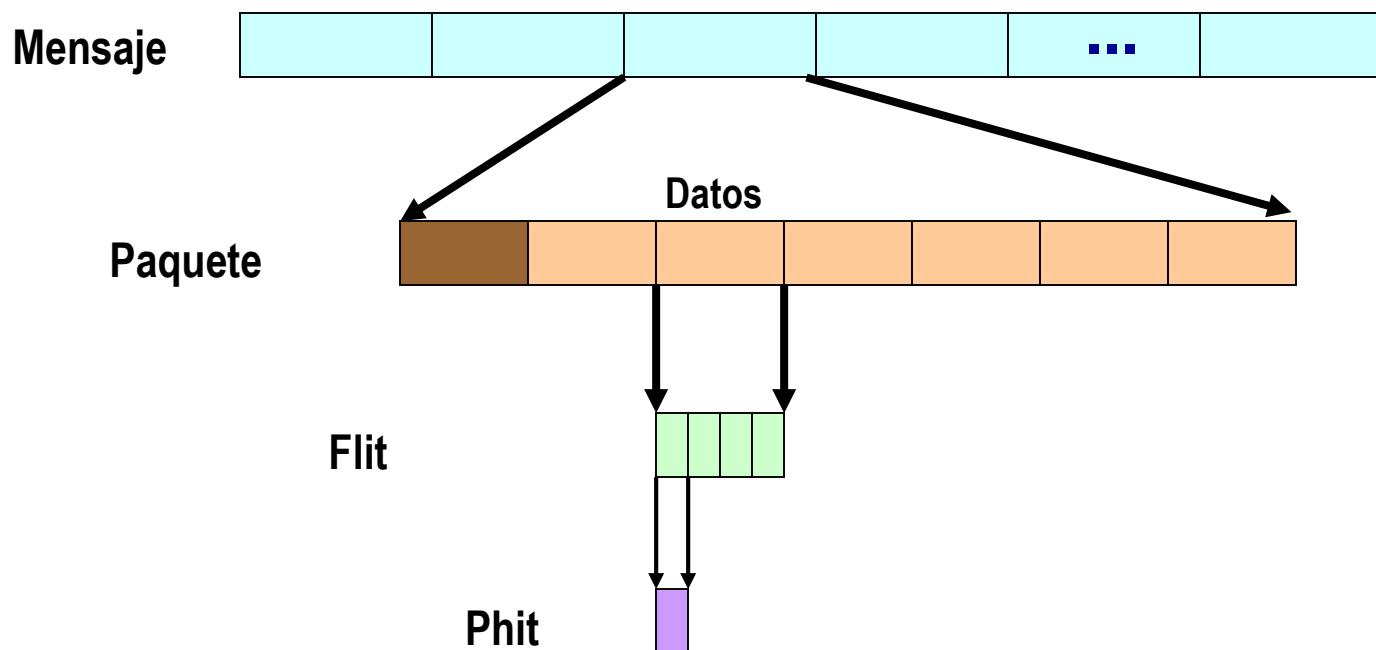
Conceptos

Clasificación

Topologías

Conmutación

- Técnicas de conmutación
 - Cuándo y cómo se conectan entradas y salidas de routers
 - Cuándo se transfiere el mensaje por los caminos



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

- Tipos de técnicas de conmutación
 - Almacenamiento y reenvío (S&F, Store and Forward)
 - Vermiforme (Wormhole)
 - Virtual Cut-Through (VCT)
 - Conmutación de circuitos (CC, Circuit Switching) (Origen en redes telefónicas)
 - Canales virtuales
- Comparación entre técnicas
 - Comparación cuantitativa: latencia de transporte
 - Comparación cualitativa: ancho de banda global
- Se considera:
 - $1 \text{ phit} = 1 \text{ flit} = w \text{ bits}$
 - Cabecera = 1 flit
 - Tamaño total del paquete = $L \text{ bits} + w \text{ bits}$ (cabecera)

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

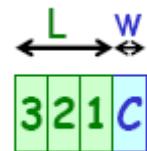
Conceptos

Clasificación

Topologías

Conmutación

- Se considera (a efectos de explicación teórica siguientes transparencias):
 - $1 \text{ phit} = 1 \text{ flit} = w \text{ bits}$
 - Cabecera = 1 flit
 - Tamaño total del paquete = L bits + w bits (cabecera)
 - Distancia fuente-destino = D parejas comutador-enlace
 - Comutadores con buffer independiente para cada entrada y salidas sin buffer
 - T_w = tiempo para que un phit atraviese una etapa comutador/enlace
 - T_r = tiempo de encaminamiento (routing)



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

- Store & forward
 - El conmutador almacena el paquete completo antes de ejecutar el algoritmo de encaminamiento y reenviar
 - La unidad de transferencia (paquete) entre interfaces ocupa sólo un canal en cada instante
 - Almacenamiento en conmutadores: múltiples de un paquete (mínimo 1 paquete)
 - Ancho de banda
 - El número de enlaces ociosos influye en el ancho de banda: para un tamaño de buffer mínimo (1 paquete), un paquete bloqueado deja ocioso un canal

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

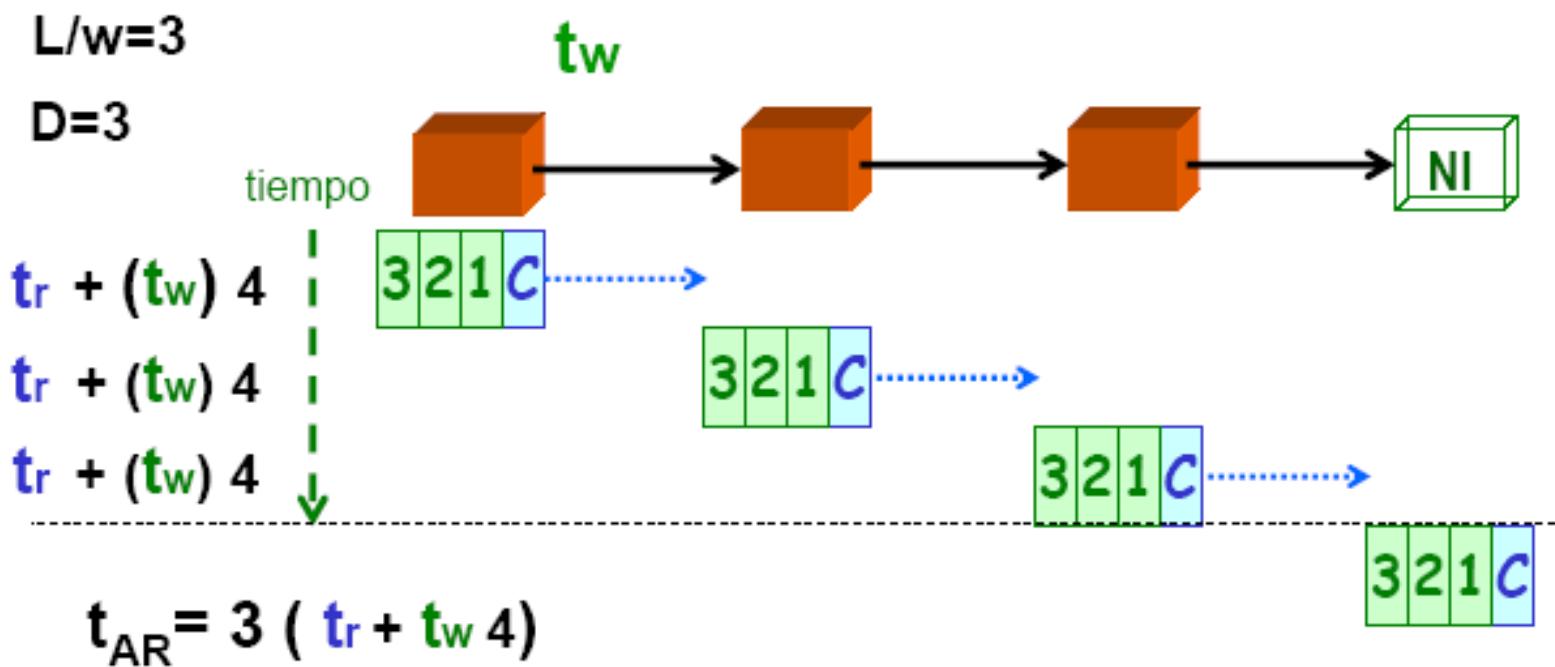
Topologías

Conmutación

- Store & forward

- Latencia de transporte:

$$t_{AR} = D \cdot \left[t_r + t_w \cdot \left(\left\lceil \frac{L}{W} \right\rceil + 1 \right) \right]$$



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

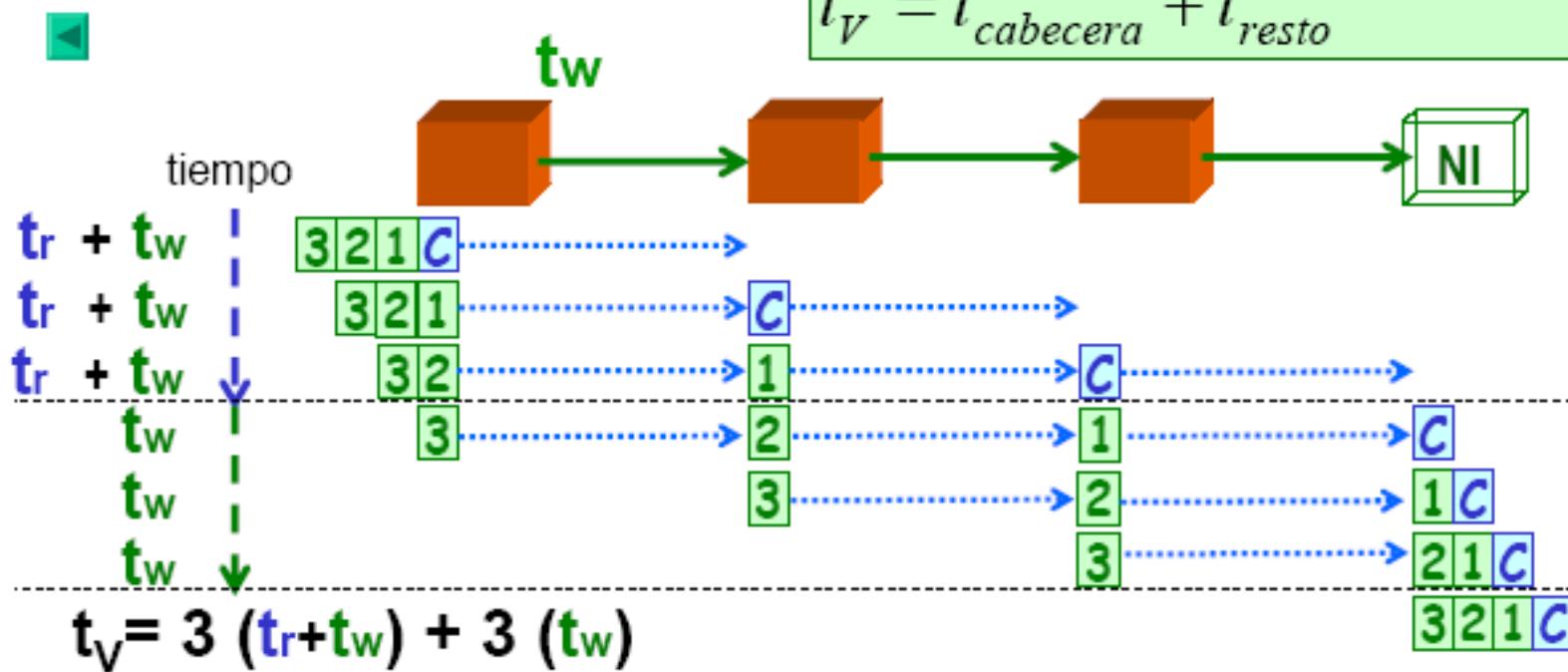
- Wormhole
 - En cuanto llega la cabecera al conmutador se ejecuta el algoritmo de encaminamiento y se reenvía
 - La unidad de transferencia es el mensaje
 - La transferencia se hace a través de un camino segmentado (nº etapas depende del nº de buffer). La unidad de transferencia puede ocupar varios canales en un instante
 - Almacenamiento en conmutadores: múltiples de un flit (mínimo 1 flit)
 - Ancho de banda
 - El número de enlaces ociosos influye en el ancho de banda: para un tamaño de buffer mínimo (1 flit), un paquete bloqueado deja ociosos varios canales

- Wormhole
 - *Latencia de transporte*

(buffer sólo en entradas de
conmutadores)

$$t_V = D \cdot (t_r + t_w) + t_w \cdot \left\lceil \frac{L}{W} \right\rceil$$

$$t_V = t_{cabecera} + t_{resto}$$



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

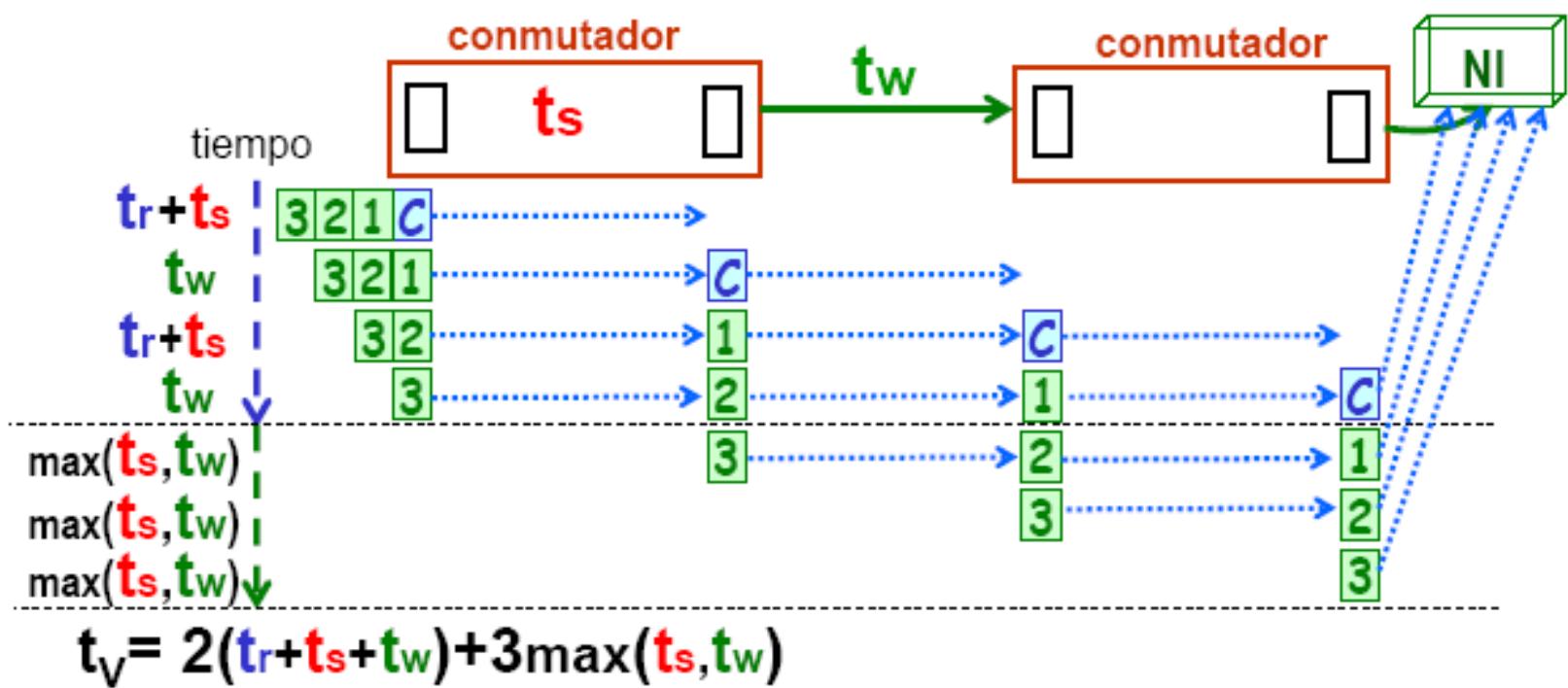
Conmutación

- Wormhole

- Latencia de transporte: (buffer en entradas y salidas)

$$t_V = D \cdot (t_r + t_s + t_w) + \max(t_s, t_w) \cdot \left\lceil \frac{L}{W} \right\rceil$$

$$t_V = t_{cabecera} + t_{resto}$$



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

- Virtual Cut-Through
 - En cuanto llega la cabecera al conmutador se ejecuta el algoritmo de encaminamiento y se reenvía
 - La unidad de transferencia es el paquete
 - La transferencia se hace a través de un camino segmentado (nº etapas depende del nº de buffer). La unidad de transferencia puede ocupar varios canales en un instante
 - Almacenamiento en conmutadores: múltiples de un paquete (mínimo 1 paquete)
 - Prestaciones
 - Latencia = wormhole
 - Ancho de banda = Store-and-Forward

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

- Conmutación de circuitos
 - Desde el fuente se envía una sonda (flit) que reserva el camino. El destino devuelve una señal de reconocimiento y el fuente comienza la transmisión
 - La unidad de transferencia es el mensaje
 - La transferencia se hace a través del canal entre fuente y destino (o un camino segmentado) reservado por la sonda
 - Almacenamiento en conmutadores: los buffers almacenan la sonda
 - Ancho de banda
 - Cuando la sonda queda bloqueada deja ociosos múltiples canales (tantos como la distancia del punto de bloqueo al fuente)

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

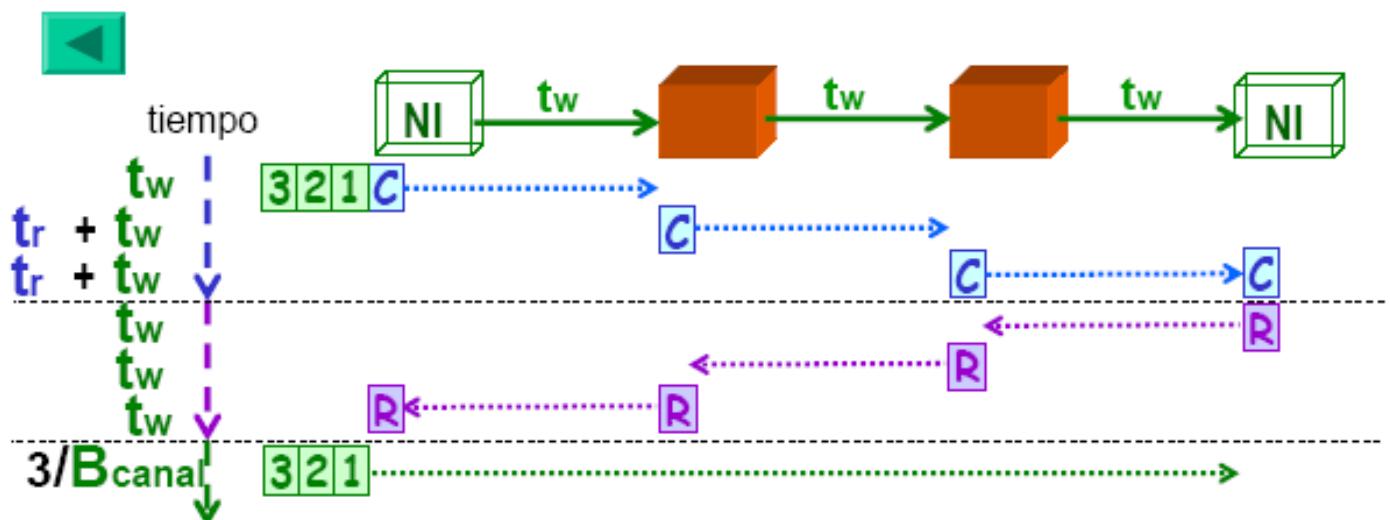
Conmutación

- Conmutación de circuitos

- *Latencia de transporte (si se establece 1 canal):*

$$t_{CC} = [t_w + D \cdot (t_r + t_w)] + [D \cdot (t_w) + t_w] + [1/B_{canal} \cdot \lceil L/W \rceil]$$

$$t_{CC} = t_{sonda} + t_{reconocimiento} + t_{datos}$$



$$t_w = (t_w + 2(t_r + t_w)) + (2(t_w) + t_w) + 3/B_{canal}$$

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

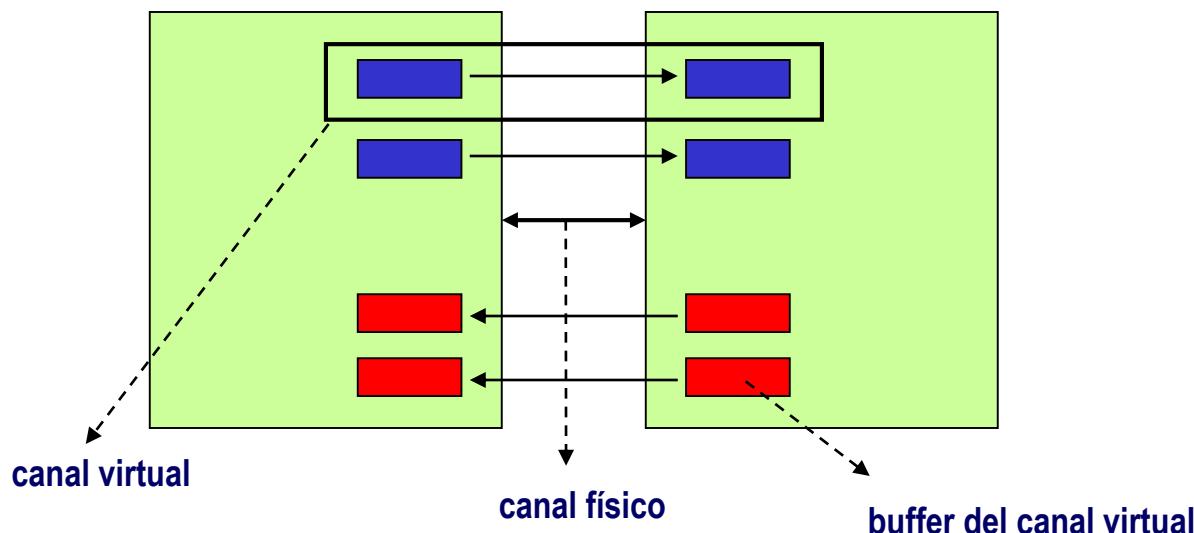
Conceptos

Clasificación

Topologías

Conmutación

- Canales virtuales
 - Permiten que varios paquetes compartan el mismo enlace (a nivel de flit)
 - Mejoran el ancho de banda y la latencia al disminuir la probabilidad de bloqueos
 - Se aplica con el resto de mecanismos



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

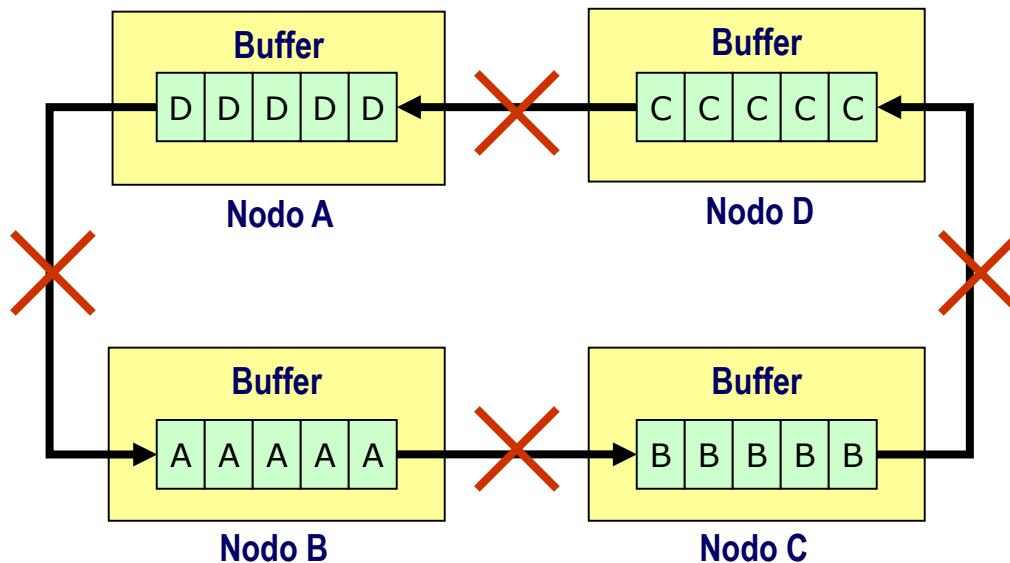
Clasificación

Topologías

Conmutación

- Bloqueos

- Algunos paquetes no pueden alcanzar el destino
- Capacidad de los buffers finita
- Canales ocupados



Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

- Bloqueos. Clasificación
 - Interbloqueos (deadlocks)
 - Recursos no disponibles para el avance de los paquetes
 - Buffers ocupados
 - Bloqueo permanente
 - Bloqueos activos (livelocks)
 - Los paquetes nunca llegan a su destino
 - Canales ocupados por otros paquetes
 - Sólo ocurre si se permiten caminos no mínimos
 - Inanición (los recursos siempre se asignan a otros paquetes)

Ingeniería de los Computadores

Sesión 10. Técnicas de conmutación

Conceptos

Clasificación

Topologías

Conmutación

- Bloqueos. Soluciones
 - Inanición
 - Emplear un esquema de asignación de recursos correcto
 - Cola circular con distinta prioridad
 - Bloqueos activos
 - Usar solo rutas mínimas
 - Usar rutas no mínimas restringidas
 - Dar mayor probabilidad a caminos mínimos respecto a no mínimos
 - Interbloqueos
 - Prevención
 - Evitación
 - Recuperación

Ingeniería de los Computadores

Sesión 11. Redes de interconexión.
Encaminamiento

- Encaminamiento. Los algoritmos de encaminamiento establecen el camino que sigue cada mensaje o paquete
- Propiedades derivadas
 - Conectividad: capacidad de encaminar desde cualquier nodo origen a cualquier nodo destino
 - Adaptabilidad: capacidad de encaminar a través de caminos alternativos
 - Evitación de bloqueos: capacidad de garantizar que los mensajes no se bloquearán en la red
 - Tolerancia a fallos: capacidad de encaminar en presencia de componentes defectuosos

- Algoritmos de encaminamiento. Criterios de clasificación

- El número de destinos
- Quién toma la decisión del encaminamiento
- Cómo se realiza la implementación
- La adaptabilidad
- La progresividad
- La minimalidad del encaminamiento
- El número de caminos proporcionados

- Algoritmos de encaminamiento. Clasificación
 - Según número de destinos:
 - Monodestino (unicast)
 - Multidestino (multicast)
 - Según decisión de encaminamiento:
 - Centralizados
 - En origen (El nodo fuente especifica el camino y la ruta se almacena en la cabecera del paquete) ↗ encaminamiento street-sign
 - Distribuidos (Los nodos intermedios deciden hacia dónde encaminar) → Idóneo para topologías irregulares
 - Multifase

- Algoritmos de encaminamiento. Clasificación
 - Según la implementación:
 - Tablas (encaminamiento por intervalos)
 - Máquinas de estados finitos (FSM) ☐ topologías ortogonales (encaminamiento por orden de dimensión)
 - Según adaptabilidad
 - Deterministas:
 - Siempre suministran el mismo camino
 - Rendimiento pobre si tráfico no uniforme
 - Adaptativos:
 - Consideran el estado de la red
 - Totalmente adaptativos: pueden usar todos los canales
 - Parcialmente adaptativos: usan un subconjunto

- Algoritmos de encaminamiento. Clasificación
 - Según progresividad
 - Progresivos
 - Backtracking: EPB (Exhaustive Profitable Backtracking)
 - Según minimalidad
 - Mínimos
 - ¿Algoritmos deterministas progresivos y mínimos?
 - No mínimos
 - Mayor flexibilidad
 - Encaminamiento tolerante a fallos

- Algoritmo determinista: encaminamiento por orden de dimensión
 - Topologías ortogonales
 - Selección de canales sucesivos con orden específico
 - Tipo determinístico
 - La diferencia en una dimensión se anula antes de pasar a la siguiente
 - Ejemplos:
 - Street-sign (fuente y sin tabla)
 - encaminamiento XY (distribuido y sin tabla)
 - encaminamiento e-cube
 - Intervalo (distribuido y con tabla de consulta)
 - Libre de interbloqueos en mallas e hipercubos (en toros es necesario usar canales virtuales y establecer un orden en su utilización)

- Encaminamiento XY (ordenado por dimensión)

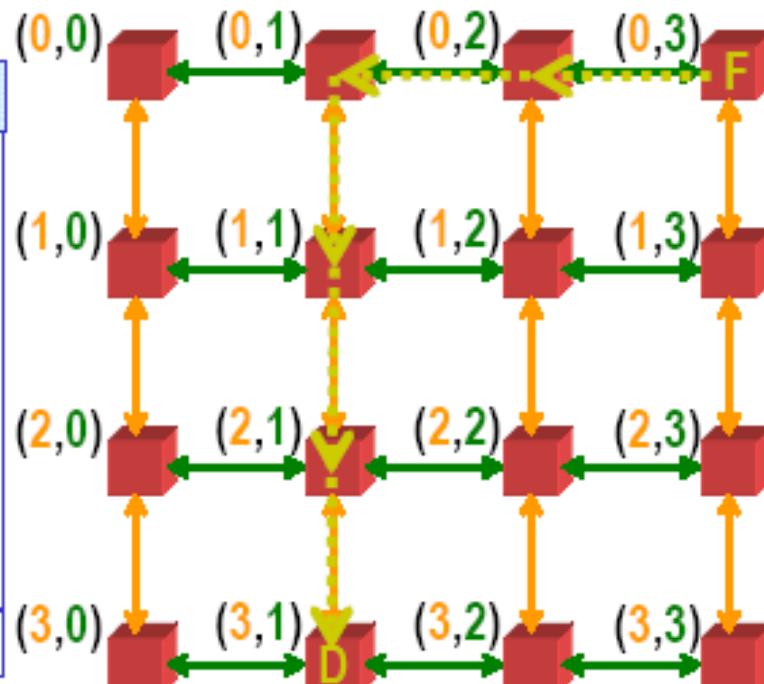
Ej. 3: malla 2D Ord. por dimensión 0-1

Entrada: actual $A = (a_1, a_0)_k$
 destino $D = (d_1, d_0)_k$

Salida: Canal $cs = (D0-, D0+, D1-, D1+, I)$.

Procedimiento:

```
dist0 =  $d_0 - a_0$ ; dist1 =  $d_1 - a_1$ ;
if ( dist0 < 0 ) cs = D0- ;
if ( dist0 > 0 ) cs = D0+ ;
if ( dist0 = 0 & dist1 < 0 ) cs = D1- ;
if ( dist0 = 0 & dist1 > 0 ) cs = D1+;
if ( dist0 = 0 & dist1 = 0 ) cs = I ;
```



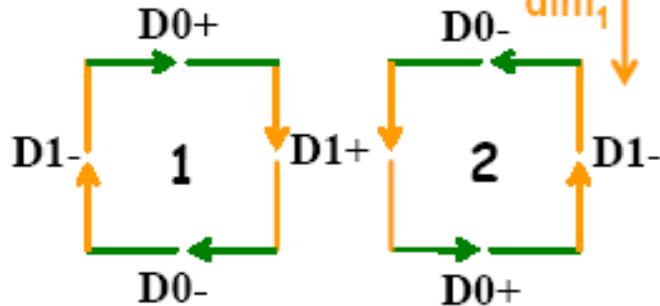
- Encaminamiento ordenado por dimensión en tablas



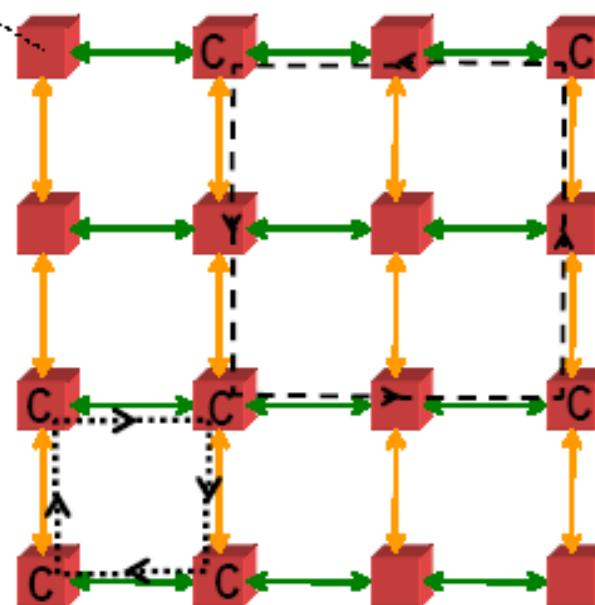
- Modelo de giros (turn-model)
 - Redes estáticas (topologías ortogonales) y redes dinámicas
 - Ejemplo:
 - West-First en mallas 2D (distribuido, sin tablas, parcialmente adaptativo y puede ser mínimo o no mínimo)
 - Interbloqueos
 - Ciclos que engloban varias direcciones → Se evitan prohibiendo al menos un cambio de dirección para cada ciclo
 - Ciclos sin cambio de dirección → Se evitan añadiendo canales virtuales y estableciendo un orden de uso

- Algoritmo West-First

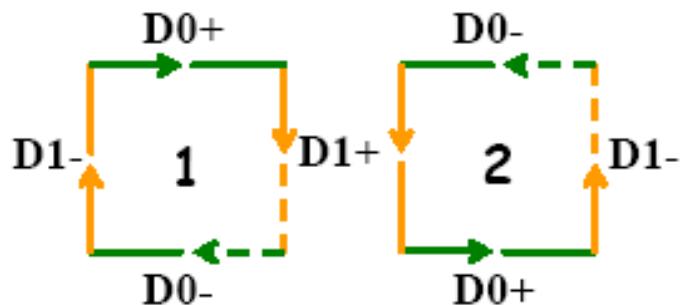
2 y 3. Identificar **cambios de dirección** y **ciclos**:



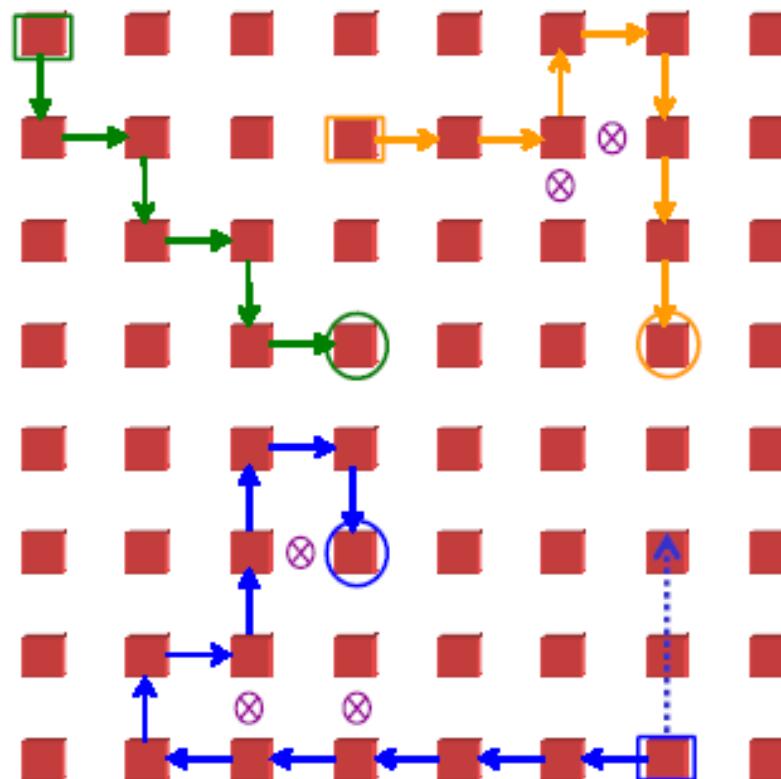
1. **Clasificación** de los enlaces de acuerdo a su **dirección**:



4. **Prohibir** un cambio de dirección en cada **ciclo**:



- Algoritmo West-First – implementación no mínima



- Algoritmo West-First para mallas 2D

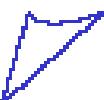
Entrada: Actual A = (a_1, a_0)

Destino D = (d_1, d_0)

Salida: Canal cs

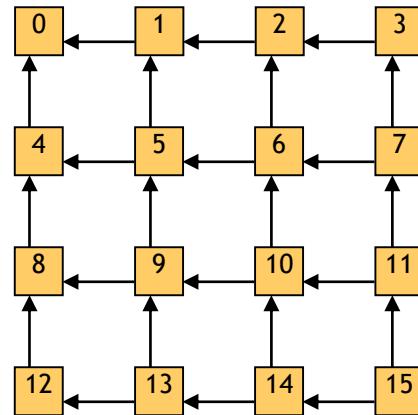
Procedimiento:

```
dist0 =  $d_0 - a_0$ ; dist1 =  $d_1 - a_1$ ;
if ( dist0 < 0 ) cs = D0-;
if ( dist0 > 0 & dist1 > 0 ) cs = Sel(D0+,D1+);
if ( dist0 > 0 & dist1 < 0 ) cs = Sel(D0+,D1-);
if ( dist0 > 0 & dist1 = 0 ) cs = D0+;
if ( dist0 = 0 & dist1 > 0 ) cs = D1+;
if ( dist0 = 0 & dist1 < 0 ) cs = D1-;
if ( dist0 = 0 & dist1 = 0 ) cs = I;
```

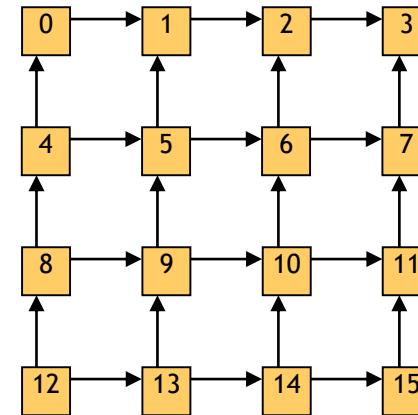


- Algoritmo totalmente adaptativo: redes virtuales

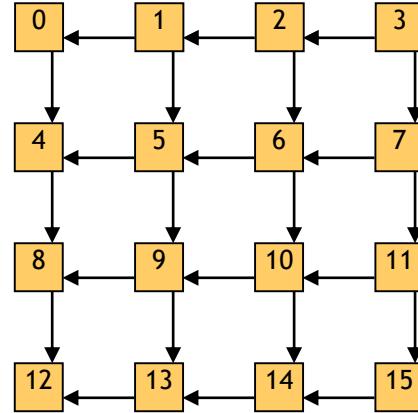
Red Virtual
 $X-Y+$



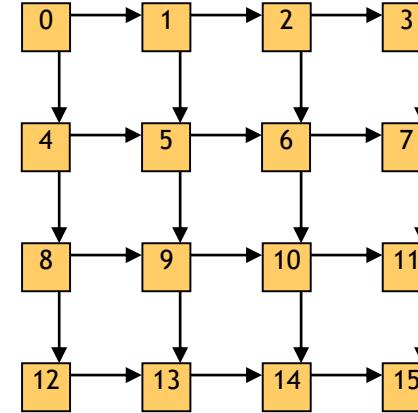
Red Virtual
 $X+Y+$



Red Virtual
 $X-Y-$



Red Virtual
 $X+Y-$





Ingeniería de los Computadores

Sesión 13. Memoria
(multiprocesadores)

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Clasificación de multiprocesadores atendiendo a la distribución de memoria
 - UMA (Uniform Memory Access)
 - NUMA (Non-Uniform Memory Access)
 - COMA (Cache-Only Memory Architecture)

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Tipo UMA
 - Memoria centralizada, uniformemente compartida entre procesadores (todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de la memoria)
 - Sistema fuertemente acoplado: alto grado de compartición de recursos (memoria) entre los procesadores → dependencia funcional entre ellos
 - Cada procesador puede disponer de caché
 - Periféricos compartidos entre procesadores
 - Aplicaciones de propósito general y de tiempo compartido por múltiples usuarios
 - Sincronización y comunicación entre procesadores utilizando variables compartidas
 - Acceso a memoria, periféricos y distribución de procesos S.O.:
 - Equitativo → Symmetric (shared-memory) Multi Processors (SMPs)
 - No equitativo → Asymmetric (shared-memory) Multi Processors (AMPs)
 - CC-UMA (Caché-Coherent Uniform Memory Access)

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Tipo NUMA
 - Memoria compartida con tiempo de acceso dependiente de la ubicación de procesadores
 - Sistema débilmente acoplado: cada procesador dispone de una memoria local a la que puede acceder más rápidamente
 - Sistema de acceso global a memoria: local, global, local de otros módulos
 - CC-NUMA (Caché-Coherent Non-Uniform Memory Access) → Memoria compartida distribuida y directorios de caché
 - Ventajas
 - Escalado de memoria con + coste/rendimiento
 - Reducción de latencia de acceso a memorias locales

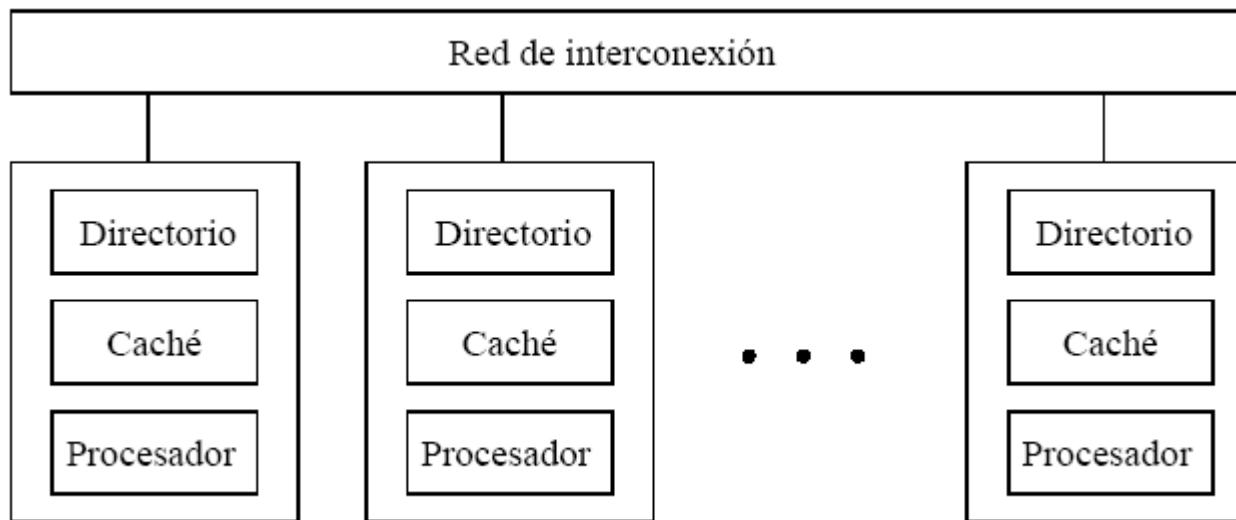
Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Tipo COMA

- Sólo se usa una caché como memoria
- Caso particular de NUMA donde las memorias distribuidas se convierten en cachés
- Las cachés forman un mismo espacio global de direcciones
- Acceso a las cachés por directorio distribuido



Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Consistencia de memoria

Problema: Transmisión de información entre procesadores en memoria compartida

Ejemplo:

```
/* El valor inicial de A y flag es 0 */
```

P1

```
A=1
```

```
flag=1
```

P2

```
while (flag == 0); /*bucle vacío*/
```

```
print A
```

P2 ha de realizar una espera activa hasta que *flag* cambie a ‘1’.

Después imprimirá A = ‘1’ (suponiendo que en P1 “A=1” es una instrucción anterior a “*flag=1*”) →

¿COHERENCIA DEL SISTEMA DE MEMORIA? Escrituras en orden de programa.

Pero ... y si “*flag=1*” se ejecuta antes de “A=1” ¿?

Es necesario un **modelo de consistencia de la memoria** en un espacio de direcciones compartido. Su objetivo es especificar restricciones en el orden de las operaciones en la memoria y proporcionar una visión uniforme de las mismas a los demás procesadores, logrando una abstracción de las operaciones de memoria independiente de la localización de las operaciones de memoria (módulos) y de los procesos involucrados.

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Consistencia de memoria
 - “Un modelo de consistencia de memoria especifica el orden en el cual las operaciones de acceso a memoria deben **parecer** haberse realizado (operaciones de lectura, escritura)”
 - En un procesador (o sistema uniprocesador), el orden en el que deben parecer haberse ejecutado los accesos a memoria es el orden secuencial especificado por el programador (o la herramienta de programación en el código que añade), denominado orden de programa.
 - Tanto el hardware como la herramienta de programación si que pueden alterar dicho orden, para mejorar las prestaciones, pero debe parecer en la ejecución del programa que no se ha alterado.

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Consistencia de memoria. Se debe garantizar que:
 - Cada lectura de una dirección, proporcione el último valor escrito en dicha dirección (**dependencia de datos lectura después de escritura**)
 - Si se escribe varias veces en una dirección se debe retornar el último valor escrito (**dependencia escritura después de escritura**)
 - Si se escribe en una dirección a la que previamente se ha accedido para leer, no se debe obtener en la lectura previa el valor escrito posteriormente en la secuencia del programa especificada por el programador (**dependencia escritura después de lectura**)
 - No se puede escribir en una dirección si la escritura depende de una condición que no se cumple (**dependencias de control**)

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

Consistencia de caché

- Ejemplo 1: 2P, caché write-through (cada vez que se escribe una línea de caché, se actualiza la memoria principal)

Secuencia	Acción	Caché A	Caché B	X en MP
1	CPU A lee X	1	-	1
2	CPU B lee X	1	1	1
3	CPU A escribe 0 en X	0	1	0

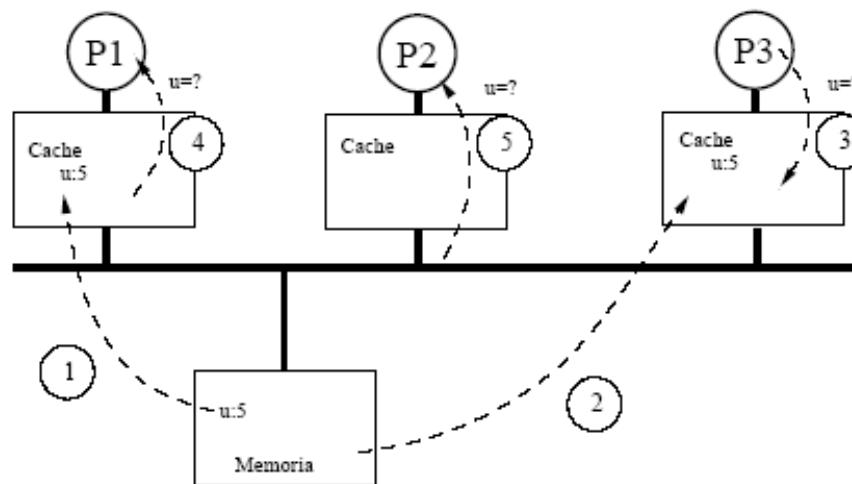
Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

Consistencia de caché

- Ejemplo 2: 3P, caché write-through / write-back (se actualiza la memoria principal escribiendo todo el bloque cuando se desaloja de la caché == aún más problemática)



Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

Consistencia de caché

- Un sistema de memoria es coherente si cualquier lectura de un dato devuelve el valor más reciente escrito de ese dato
- Aspectos críticos del sistema de memoria compartida: los datos devueltos por una lectura (coherencia) y cuándo un valor escrito será devuelto por una lectura (consistencia)
- Un sistema de memoria es coherente si cumple:
 - Preservación del orden del programa: una lectura por un procesador P de una posición X, que sigue a una escritura de P a X, sin que ningún otro procesador haya escrito nada en X entre la escritura y la lectura de P, siempre devuelve el valor escrito por P.
 - Visión coherente de la memoria: una lectura por un procesador de la posición X, que sigue a una escritura por otro procesador a X, devuelve el valor escrito si la lectura y escritura están suficientemente separados y no hay otras escrituras sobre X entre los dos accesos.
 - Serialización de operaciones concurrentes de escritura: Las escrituras a la misma posición por cualquiera dos procesadores se ven en el mismo orden por todos los procesadores.

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

Consistencia de caché

- Un sistema de memoria multiprocesador es coherente si el resultado de cualquier ejecución de un programa es tal que, para cada localización es posible construir una hipotética ordenación secuencial de todas las operaciones realizadas sobre dicha localización que sea consistente con los resultados de la ejecución y en el cuál:
 1. Las operaciones emitidas por un procesador particular ocurren en la secuencia indicada y en el orden en el que dicho procesador las emite al sistema de memoria
 2. El valor devuelto por cada operación de lectura es el valor escrito por la última escritura en esa localización en la secuencia indicada

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

Consistencia de caché

- Dos estrategias para abordar la coherencia de las cachés:
 - Resolución software: el compilador y el programador evitan la incoherencia entre cachés de datos compartidos.
 - Hardware: transparente al programador mediante la provisión de mecanismos hardware, esta es la solución más utilizada para mantener la coherencia.
- Políticas para mantener la coherencia:
 - Invalidación de escritura o coherencia dinámica (write invalidate): siempre que un procesador modifica un dato de un bloque en la caché, invalida todas las demás copias de ese bloque guardadas en las otras cachés de los procesadores.
 - Actualización en escritura (write-update ó write-broadcast): esta política lo que hace es actualizar la copia en las demás cachés en vez de invalidarla.

Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

Consistencia de caché

- Dos estrategias para abordar la coherencia de las cachés:
 - Resolución software: el compilador y el programador evitan la incoherencia entre cachés de datos compartidos.
 - Hardware: transparente al programador mediante la provisión de mecanismos hardware, esta es la solución más utilizada para mantener la coherencia.
- Políticas para mantener la coherencia:
 - Invalidación de escritura o coherencia dinámica (write invalidate): siempre que un procesador modifica un dato de un bloque en la caché, invalida todas las demás copias de ese bloque guardadas en las otras cachés de los procesadores.
 - Actualización en escritura (write-update ó write-broadcast): esta política lo que hace es actualizar la copia en las demás cachés en vez de invalidarla.

Ingeniería de los Computadores

Sesión 14. Coherencia de memoria
(multiprocesadores)

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy:
 - Utilizado en redes donde el broadcast (difusión) es posible
 - Redes basadas en bus
 - Cada caché monitoriza el estado del bus y las transacciones de las demás cachés
- Protocolos basados en directorio:
 - Utilizado en redes donde el broadcast no es posible a causa de la degradación
 - Redes multietapa
 - Se utiliza para ello un directorio centralizado o distribuido

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- **Protocolos de sondeo o snoopy**

- Objetivo: garantizar las transacciones necesarias en el bus para operaciones de memoria y que los controladores de caché observen y actúen en transacciones relevantes
- Todas las transacciones aparecen en el bus y son visibles para los procesadores en el mismo orden en el que se producen
- Utilizado en sistemas multiprocesador con bus y pocos elementos conectados
- Cada procesador indica el estado de cada línea de su caché
- La red de interconexión debe permitir broadcast (difusión)
- Cada caché monitoriza las transacciones de las demás cachés observando el bus
- Se utiliza un algoritmo distribuido representado como un conjunto de máquinas de estados finitos que cooperan entre sí:
 - Conjunto de estados asociado con los bloques de memoria en las cachés
 - Diagrama de transición entre estados
 - Acciones asociadas a las transiciones entre estados

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy – Protocolo de invalidación en escritura (write invalidate)
 - Constituyen las estrategias más robustas y extendidas
 - Basado en asegurar que un procesador tiene acceso exclusivo a un dato antes de que acceda a él
 - Se consigue invalidando todas la líneas de todas las cachés que contengan ese dato que está siendo escrito en ese momento
 - Cada controlador de caché observa las transacciones de memoria (observa el bus) de los otros controladores para mantener su estado interno
 - Control de coherencia de caché mediante transacciones de lectura y de lectura exclusiva
 - Cuando un procesador quiera leer un dato invalidado, falle su caché y tenga que ir a memoria a buscarlo
 - Protocolos MSI, MESI, Write Once y Brekeley

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy – Protocolo de actualización en escritura (write-update)
 - Menos utilizados que el anterior (write-invalidate)
 - Cada vez que un procesador escribe un dato, se actualizan las cachés que contienen el dato en los demás procesadores
 - Problemas para mantener el ancho de banda bajo control debido a la alta cantidad de transacciones que se generan
 - Mejora: escritura en una localización ↗ actualización de cachés relacionadas
 - Protocolos Dragon y Firefly

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy - MSI
 - Protocolo de invalidación básico para cachés write-back (MSI)
 - Estados: Inválido (I), Compartido (S), Modificado (M)
 - Inválido: no es válido el bloque
 - Compartido: el bloque esta presente en la caché y no ha sido modificado, la memoria principal esta actualizada y cero o más cachés pueden tener también una copia actualizada (compartida)
 - Modificado: únicamente este procesador tiene una copia válida, la copia de la memoria principal esta anticuada y ninguna otra caché puede tener una copia válida del bloque (ni en estado modificado ni compartido)

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy - MSI

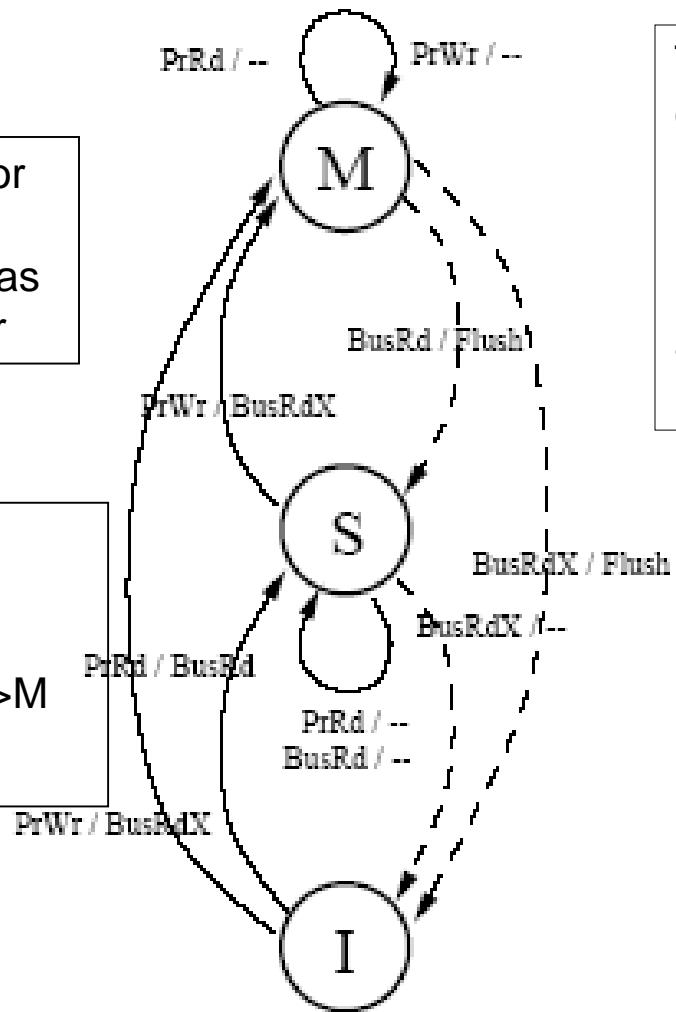
Operaciones del procesador
+ transacciones que se
generan en el bus por dichas
operaciones de procesador

Fallo lectura (I) -> M,S

Fallo escritura (I) -> M,S

Acierto escritura (S) -> S, (M)->M

Acierto lectura (S,M)



Transacciones en el bus
observadas por las cachés
para cambiar su estado
(monitorización)

Flush = dato en bus +
actualización de memoria
principal

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy - MESI
 - Ampliación del protocolo de invalidación de 3 estados. Refinamiento para aplicaciones “secuenciales” que corren en multiprocesadores (carga común usada en multiprocesadores de pequeña escala)
 - En el MSI el programa cuando lee y modifica un dato tiene que generar 2 transacciones incluso en el caso de que no exista compartición (solo presente en una caché) del dato (BusRd y BusRdX)
 - Estados: Modificado (M), *Exclusivo (E), Compartido (S), Inválido (I)
 - *Exclusivo (E): indica que el bloque es la única copia (exclusiva) del sistema multiprocesador y que no está modificado, ningún otro procesador tiene el bloque en la caché y la memoria principal está actualizada
 - Al ser exclusivo es posible realizar una escritura o pasar al estado modificado sin ninguna transacción en el bus, al contrario que en el caso de estar en el estado compartido; pero no implica pertenencia, así que al contrario que en el estado modificado la caché no necesita responder al observar una petición de dicho bloque (la memoria tiene una copia válida)
 - Conocido como protocolo Illinois (publicado por la U. de Illinois en 1984)

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- **Protocolos de sondeo o snoopy - MESI**

BusRd(S) = cuando la transacción de lectura en el bus ocurre, se activa la señal (S).

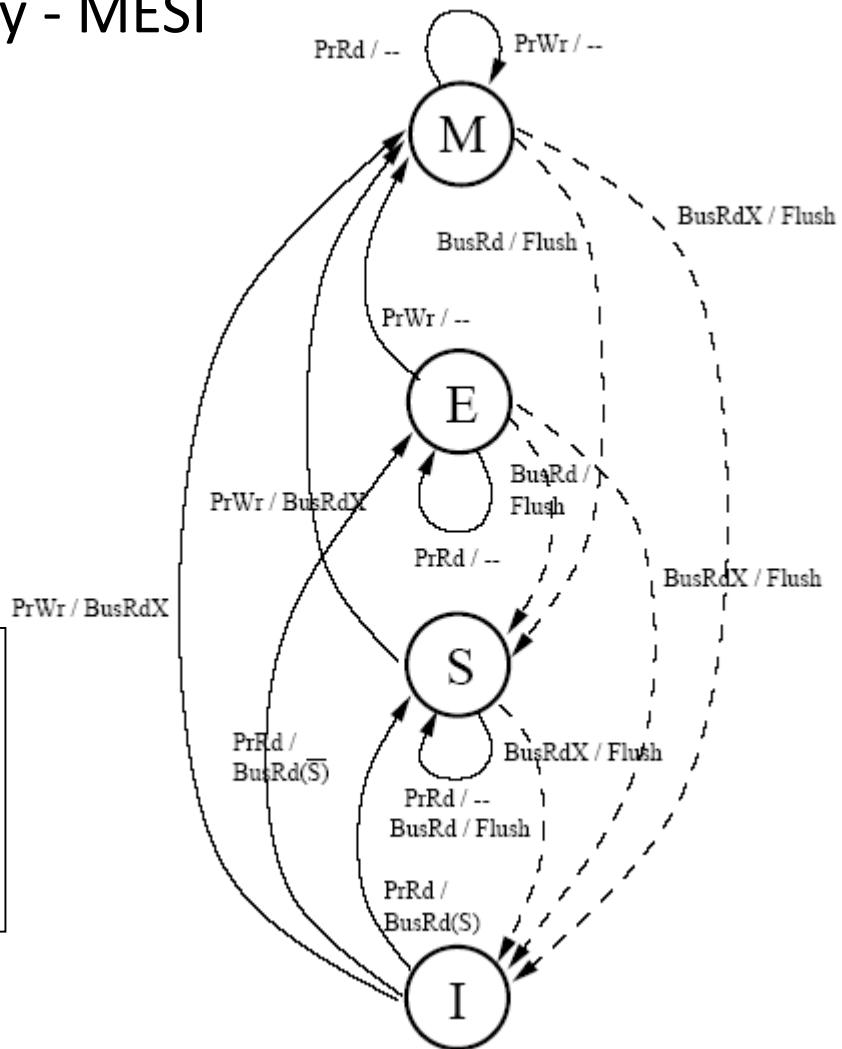
Este protocolo necesita una señal adicional que sea proporcionada por el bus (S) que esté disponible para que los controladores puedan determinar en una transacción BusRd si existe otra caché que tenga el mismo bloque

Fallo lectura (I) -> M,S,E

Fallo escritura (I) -> M,S,E

Acierto escritura (S) -> S, (M,E) -> O

Acierto lectura (S,M,E)



Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy – Write Once
 - Cada línea de caché tiene dos bits extra para almacenar el estado de esa línea
 - Líneas adicionales de control para inhibir la memoria principal
 - Estados líneas de caché:
 - Válida (V): la línea de caché, es consistente con la copia de memoria, ha sido leída de la memoria principal y no ha sido modificada
 - Inválida (I): la línea no se encuentra en la caché o no es consistente con la copia en memoria
 - Reservada (R): los datos han sido escritos una única vez desde que se leyó de la memoria compartida, la línea de caché es consistente con la copia en memoria que es la única otra copia
 - Sucia (S): la línea de caché ha sido escrita más de una vez, y la copia de la caché es la única en el sistema (por lo tanto inconsistente con el resto de copias)
 - Transiciones según operaciones sobre cachés: fallo de lectura, acierto de escritura, fallo de escritura, acierto de lectura y cambio de línea

Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

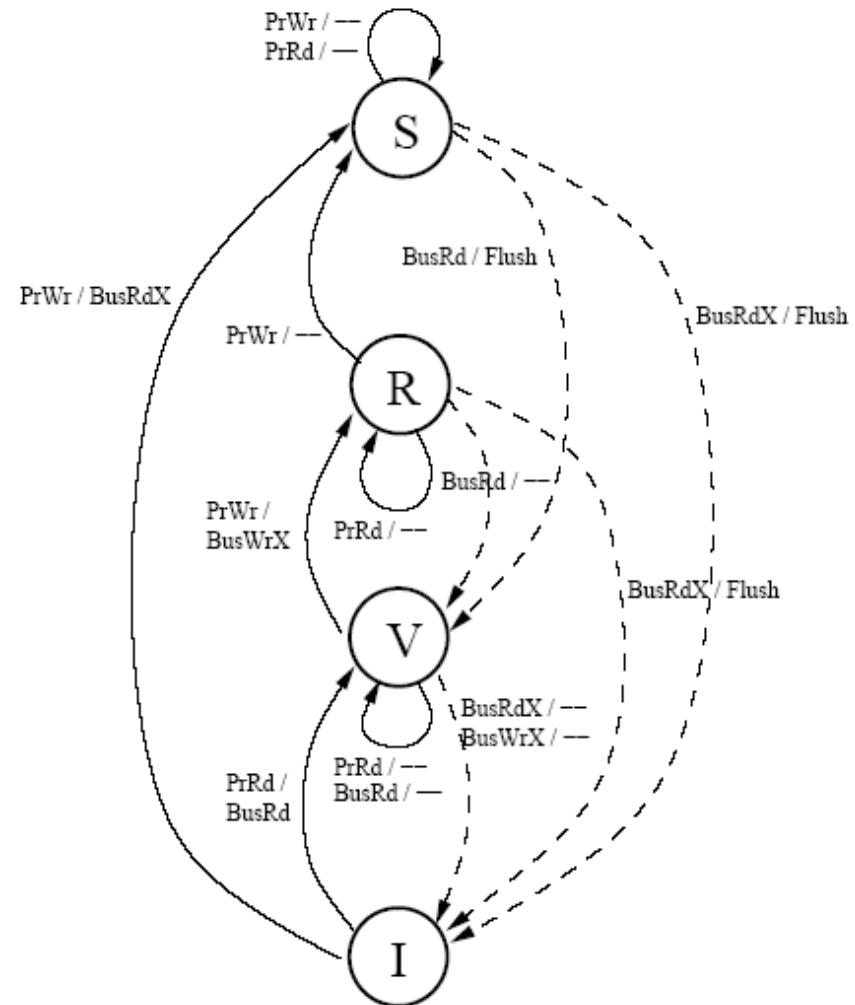
Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy – Write Once

- Válida (V): leída de la memoria principal y no ha sido modificada
- Inválida (I)
- Reservada (R): los datos han sido escritos una única vez. la línea de caché es consistente con la copia en memoria
- Sucia (S): la línea de caché ha sido escrita más de una vez



Ingeniería de los Computadores

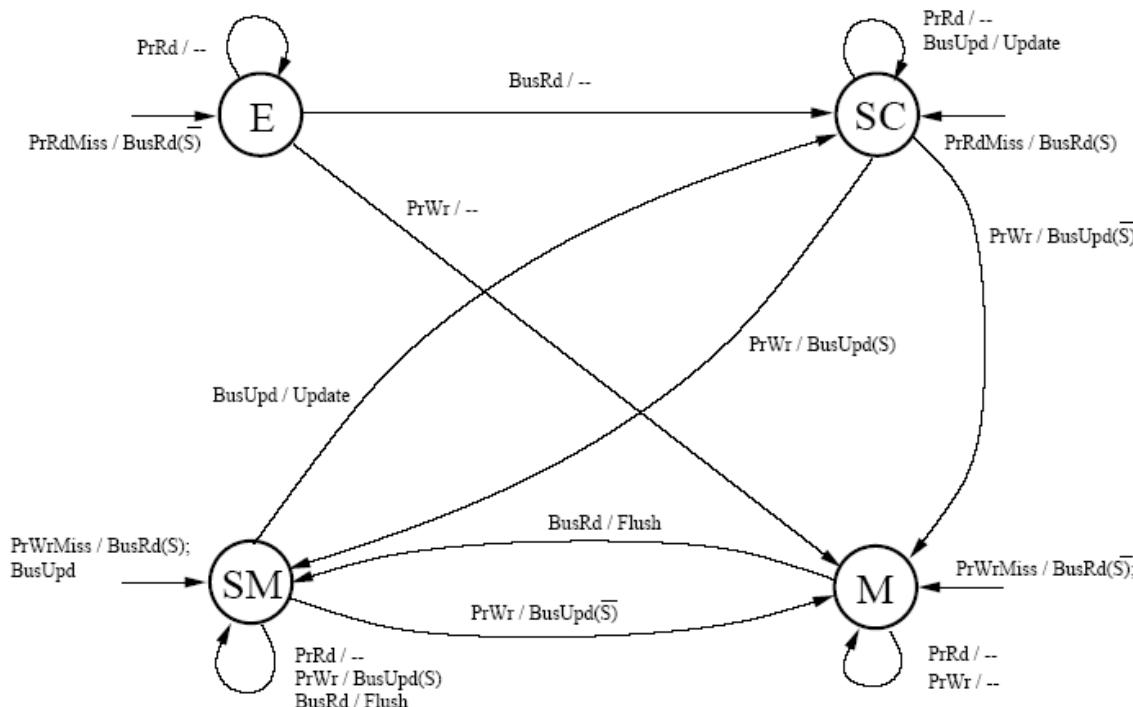
Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy - Dragon
 - Protocolo de actualización en escritura básico para cachés write-back (Dragón)
 - Estados: Exclusivo (E), Compartido (C), Compartido_Modificado (SM) y Modificado (M)
 - Transiciones según operaciones sobre cachés: fallo de lectura, escritura, reemplazo



Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

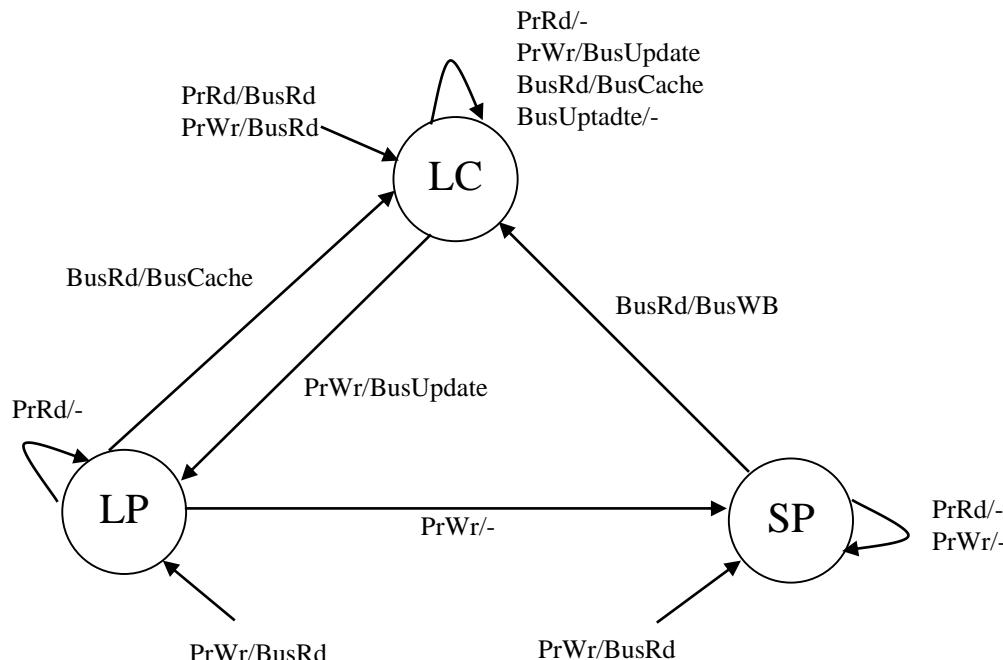
Conceptos

Consistencia de caché

Protocolos de coherencia

- **Protocolos de sondeo o snoopy - Firefly**

- Protocolo de actualización en escritura, que hace uso de una línea compartida especial de bus
- Estados de línea de caché: Lectura_Privada(LP) (Exclusive), Lectura_Compartida (LC) (Shared) y Sucia_Privada (SP) (Modified) (MESI)
- Transiciones según operaciones sobre cachés: fallo de lectura, escritura, reemplazo



Ingeniería de los Computadores

Sesión 14. Memoria. Coherencia

Conceptos

Consistencia de caché

Protocolos de coherencia

- Protocolos de sondeo o snoopy. Rendimiento
 - Consideraciones a tener en cuenta para evaluar el rendimiento de un protocolo:
 - Tráfico causado por fallos de caché
 - Tráfico de comunicación entre cachés
 - Diferencias de rendimiento entre protocolos de invalidación y actualización:
 - Varias escrituras a la misma palabra sin lecturas intermedias
 - Líneas de caché de varias palabras: los de invalidación trabajan sobre el bloque y los de actualización sobre palabras para aumentar la eficiencia
 - Retraso entre escritura de palabra en un procesador y la lectura por parte de otro; mejor en lo protocolos de actualización
 - Rendimiento según los requerimientos:
 - Gestión de ancho de banda y memoria crítica: para aprovechar mejor el ancho de banda del bus y la memoria, se utilizan los protocolos de invalidación
 - Migración de procesos o sincronización intensivas: cuando se requiera mucha migración de procesos o mucha sincronización, un protocolo de invalidación va a trabajar mucho mejor que uno de actualización