Arquitectura y Organización de Computadores

Universidad Francisco de Vitoria
UFV Madrid

TEMA 5
Paralelismo

Objetivos

- **Objetivos generales**
  - Conocer los principios básicos de optimización aplicados al diseño de procesadores y computadores con el fin de mejorar su capacidad de cómputo.
- **Objetivos específicos**
  - Definir diferentes tipos y niveles de procesamiento paralelo y clasificar un sistema informático dentro de cada tipo
  - Explicar las diferentes técnicas de optimización que se han aplicado para la mejora de la capacidad de procesamiento de un sistema, indicando el tipo de paralelismo que explotan y proporcionando ejemplos de sistemas reales que las incorporan.
  - Distinguir, para cada técnica de optimización, si requiere de varios núcleos o procesadores y si es transparente o no tanto al sistema operativo como al software que se ejecuta en dicho sistema.

Índice

- Clasificación de Flynn
- Segmentación
- ILP: explotación mediante técnicas superescalares y VLIW
- Multithreading
- Sistemas multiprocesador

Bibliografía

- Apéndice A del libro Computer Architecture, de Hennessy – Patterson
- Capítulo 6 del libro Arquitectura de Computadores, de Hennessy-Patterson (1993)
- Capítulo 2 del libro Arquitectura de Computadores, de J. Ortega – M. Anguita – A. Prieto

Principios básicos de optimización

- ***Paralelismo***:

  incrementar el rendimiento del computador mediante la ejecución simultánea de varias instrucciones

- ***Jerarquía de memoria***:

  aprovechar el *principio de localidad* para incrementar el rendimiento del sistema de memoria, sustituyendo una memoria única por varias memorias con diferentes prestaciones y tamaños, organizadas de manera jerárquica

Clasificación de sistemas con paralelismo

- Taxonomía de Flynn
  - **SISD** (Single Instruction Single Data)
    - un flujo de instrucciones único trabaja sobre un flujo de datos único
    - El paralelismo a nivel de instrucción entra dentro de esta categoría (procesadores segmentados, superescalares y VLIW)
  - **SIMD** (Single Instruction Multiple Data):
    - un flujo de instrucciones único trabaja sobre múltiples flujos de datos (computadores vectoriales)
  - **MISD** (Multiple Instruction Single Data):
    - múltiples flujos de instrucciones trabajan sobre un flujo de datos único (no implementación real)
  - **MIMD** (Multiple Instruction Multiple Data)
    - múltiples flujos de instrucciones trabajan sobre múltiples flujos de datos (multiprocesadores y multicomputadores)

Clasificación de sistemas con paralelismo

- Ejemplo de paralelismo:

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

Este programa implica realizar **12 operaciones**:
4 sumas, 4 restas y 4 multiplicaciones (obviamos las de control del bucle)

Clasificación de sistemas con paralelismo

- Ejemplo (cont.):
  - Ejecución **SISD**: 12 operaciones secuencialmente

```
C[1] = A[1] + B[1];
D[1] = E[1] - F[1];
G[1] = K[1] * B[1];
C[2] = A[2] + B[2];
D[2] = E[2] - F[2];
G[2] = K[2] * B[2];
```

```
C[3] = A[3] + B[3];
D[3] = E[3] - F[3];
G[3] = K[3] * B[3];
C[4] = A[4] + B[4];
D[4] = E[4] - F[4];
G[4] = K[4] * B[4];
```

- Las instrucciones pueden ejecutarse simultáneamente gracias a la segmentación o las técnicas que explotan ILP, como las superescalares o VLIW, pero a efectos de la clasificación de Flynn, estos procesadores no se consideran paralelos

Principios básicos de optimización

- Ejemplo (cont.):
  - Ejecución **SIMD**: 3 operaciones secuencialmente, 4 en paralelo

```
C = A + B;
D = E - F;
G = K * B;
```

```
C[1]=A[1]+B[1]; C[2]=A[2]+B[2]; C[3]=A[3]+B[3]; C[4]=A[4]+B[4];
D[1]=E[1]-F[1]; D[2]=E[2]-F[2]; D[3]=E[3]-F[3]; D[4]=E[4]-F[4];
G[1]=K[1]*B[1]; G[2]=K[2]*B[2]; G[3]=K[3]*B[3]; G[4]=K[4]*B[4];
```

  - Los procesadores vectoriales están diseñados para trabajar con vectores y, por tanto, realizan la misma operación simultaneamente en los distintos componentes del vector

Principios básicos de optimización

- Ejemplo (cont.):
  - Ejecución **MIMD**: varias alternativas
    - 3 operaciones en secuencia, 4 en paralelo

Proc 1	Proc 2	Proc 3	Proc 4
`C[1]=A[1]+B[1];`	`C[2]=A[2]+B[2];`	`C[3]=A[3]+B[3];`	`C[4]=A[4]+B[4];`
`D[1]=E[1]-F[1];`	`D[2]=E[2]-F[2];`	`D[3]=E[3]-F[3];`	`D[4]=E[4]-F[4];`
`G[1]=K[1]*B[1];`	`G[2]=K[2]*B[2];`	`G[3]=K[3]*B[3];`	`G[4]=K[4]*B[4];`

Principios básicos de optimización

- Ejemplo (cont.):
  - Ejecución **MIMD**: varias alternativas
    - 4 operaciones en secuencia, 3 en paralelo

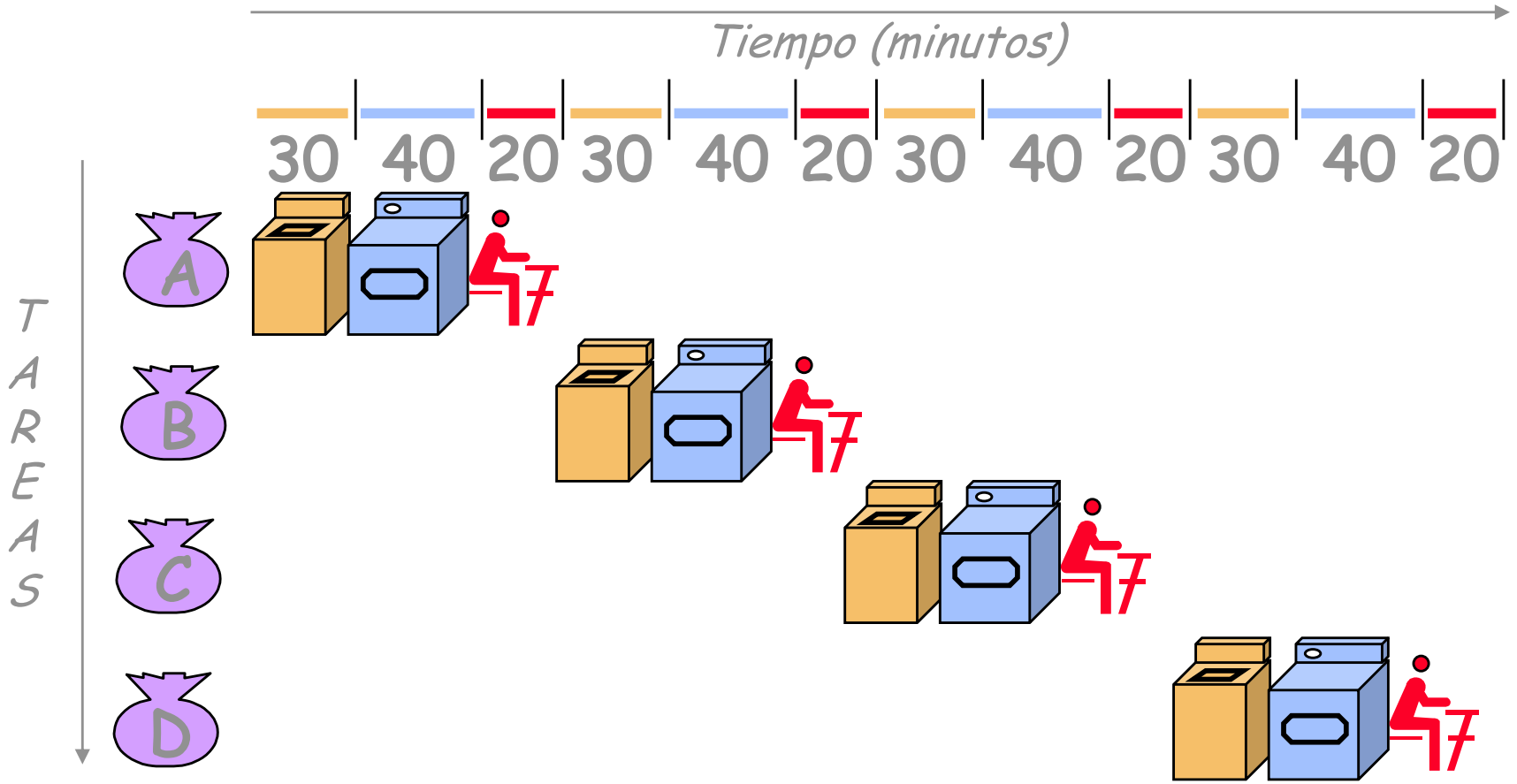
Proc 1	Proc 2	Proc 3
`C[1]=A[1]+B[1];`	`D[1]=E[1]-F[1];`	`G[1]=K[1]*B[1];`
`C[2]=A[2]+B[2];`	`D[2]=E[2]-F[2];`	`G[2]=K[2]*B[2];`
`C[3]=A[3]+B[3];`	`D[3]=E[3]-F[3];`	`G[3]=K[3]*B[3];`
`C[4]=A[4]+B[4];`	`D[4]=E[4]-F[4];`	`G[4]=K[4]*B[4];`

    - Estos dos últimos ejemplos ilustran, simplemente, que los sistemas MIMD son los más versátiles a la hora de repartir la carga de trabajo. Sin embargo, los programas se paralelizan a un nivel más grueso (a nivel de hebra normalmente) y no a nivel de instrucción.

Segmentación

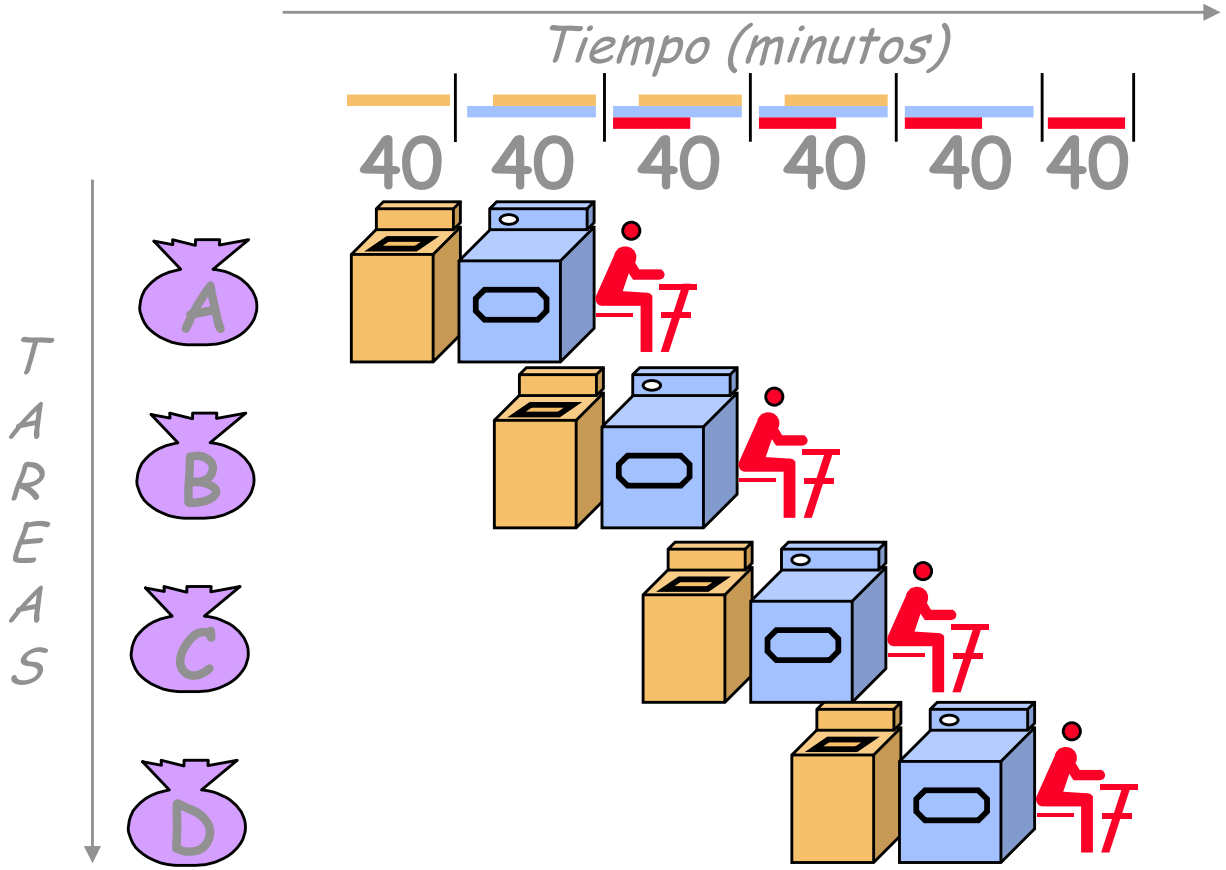
- La ***segmentación*** (*pipelining*) es una técnica de implementación mediante la cual múltiples instrucciones se ejecutan **simultáneamente** en el mismo procesador
- Es la técnica básica utilizada hoy en día para construir CPUs (core) rápidas
- Se empezó a utilizar a finales de los 50. El primer procesador segmentado de propósito general fue el IBM 7030 Stretch
- Se abandonó desde finales de los 60 hasta los 80, donde surgen las arquitecturas RISC, que permiten una segmentación más sencilla

Segmentación



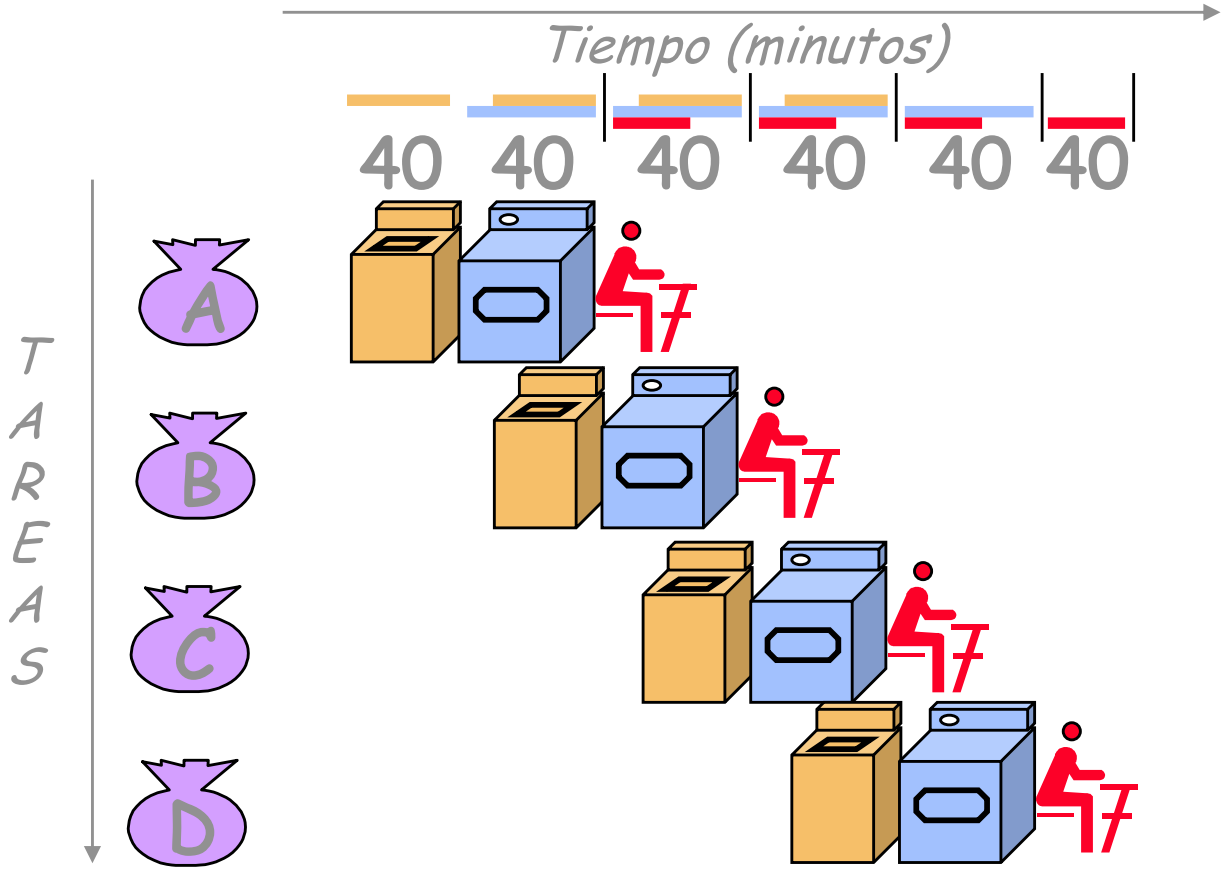
LAVANDERÍA SECUENCIAL: 4 coladas tardan 6 horas

Segmentación



LAVANDERÍA SEGMENTADA: 4 coladas tardan 4 horas

Segmentación

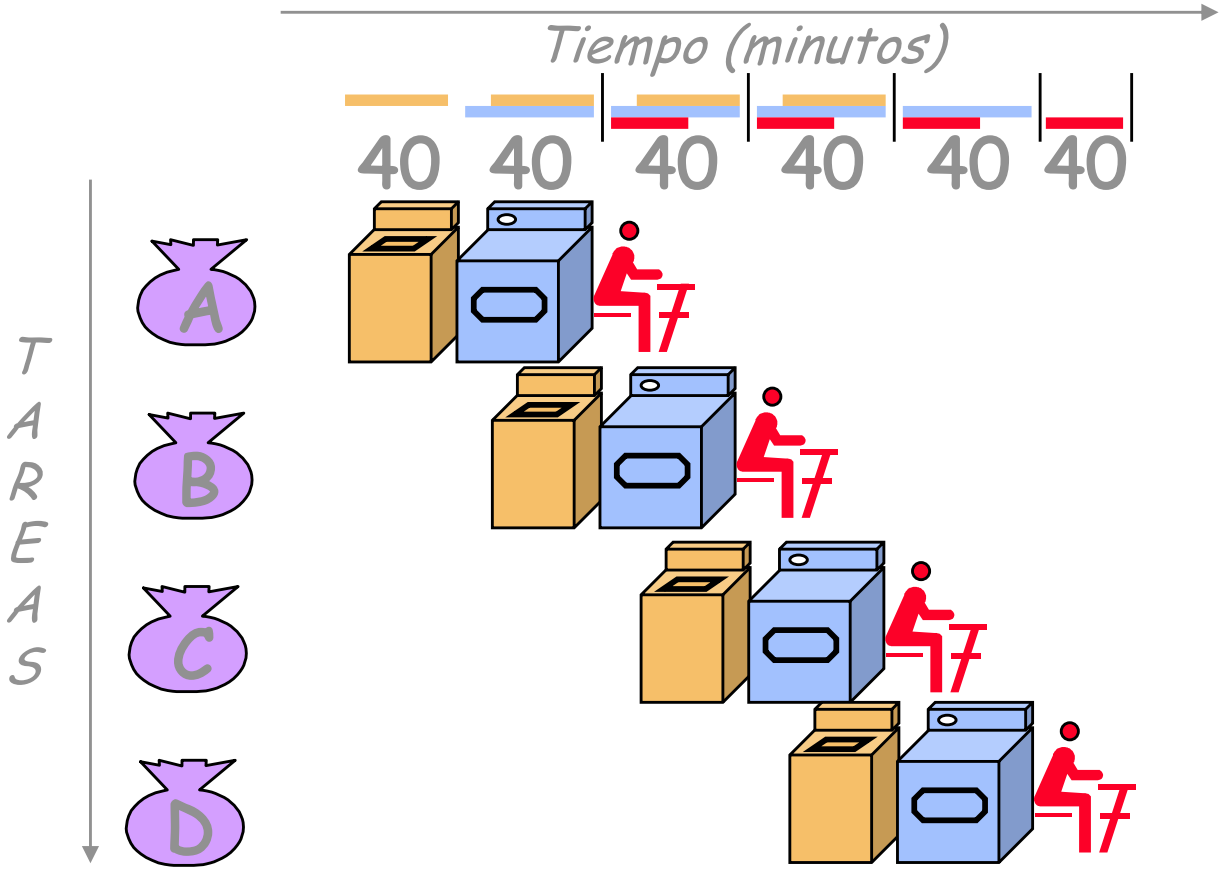


La segmentación no mejora el T a nivel instrucción, sino la productividad

La velocidad está limitada por la etapa más larga

A mayor diferencia de retardos entre etapas, menor ganacia

Segmentación

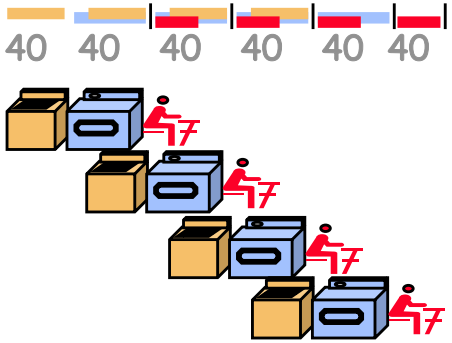


Ganancia (*Speedup*) potencial: número de etapas del **cauce** (*pipe*) segmentado

La Ganancia se reduce "irremediablemente" por el nº de etapas necesarias para llenar el cauce (problema inherente a la propia segmentación)

Segmentación

Ganancia con las segmentación



- Ejemplo:
  - ¿Cuál es la *ganancia* potencial en el caso
  - de la lavandería?

3

- ¿Cuál es la *ganancia* real para 4 coladas?

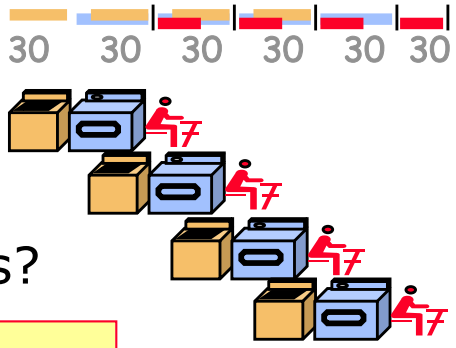
$$SpeedUp = \frac{T_{ejec.}\ no\ segmentado}{T_{ejec.}\ segmentado} = \frac{6}{4} = 1{,}5$$

- ¿Cuál es la *ganancia* real para 1000 coladas?

$$SpeedUp = \frac{T_{ejec.}\ no\ segmentado}{T_{ejec.}\ segmentado} = \frac{1500}{668} = 2{,}25$$

Segmentación

Ganancia con las segmentación



- Ejemplo:
  - ¿Cuál es la *ganancia* real para 4 coladas, si suponemos que todas las etapas tardan 30 minutos?

$$Speedup = \frac{T_{ejec}\text{no segmentado}}{T_{ejec}\text{segmentado}} = \frac{6h}{3h} = 2$$

  - ¿Cuál es la *ganancia* real para 1000 coladas?

$$Speedup = \frac{T_{ejec}\text{no segmentado}}{T_{ejec}\text{segmentado}} = \frac{1500h}{501h} \approx 3$$

Segmentación

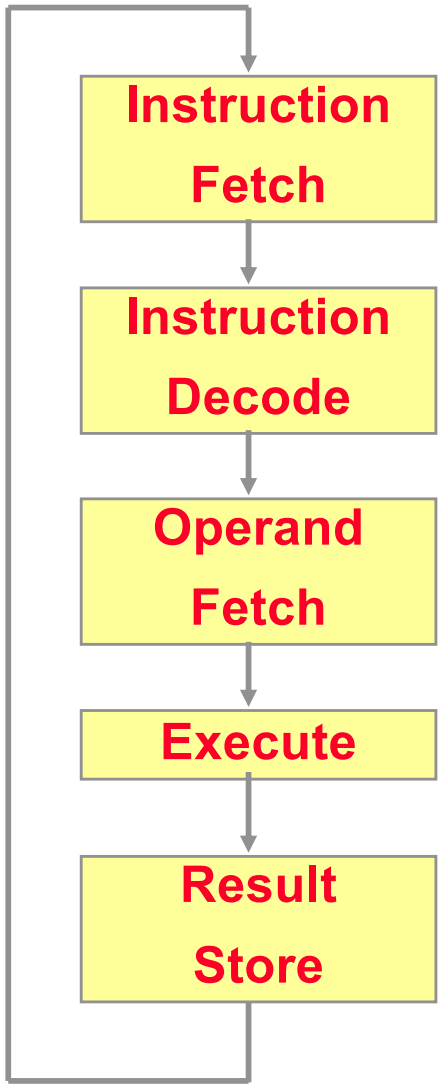
Conclusiones

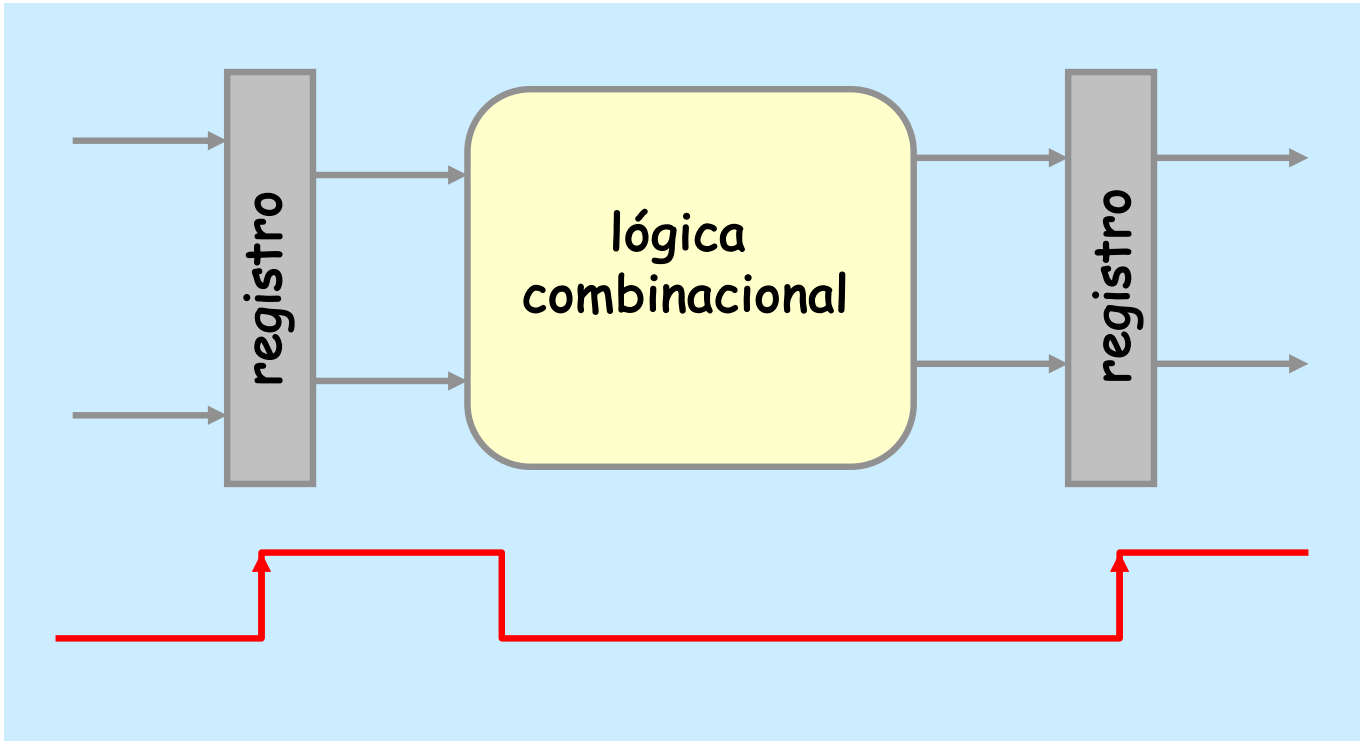
- La segmentación mejora más cuanto mayor número de tareas haya que realizar

- La segmentación mejora más cuanto más similares sean los retardos de cada una de las etapas
  - Con el fin de equilibrar los retardos, se puede:
    - Agrupar varias etapas "lógicas" en una etapa del cauce
    - Descomponer una etapa "lógica" en varias etapas del cauce

Segmentación básica de un procesador

- Etapas de ejecución de instrucciones

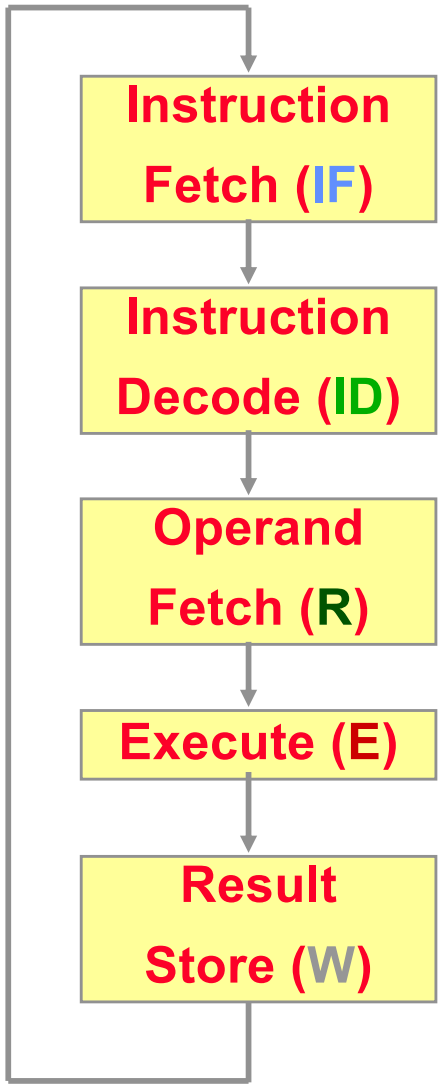


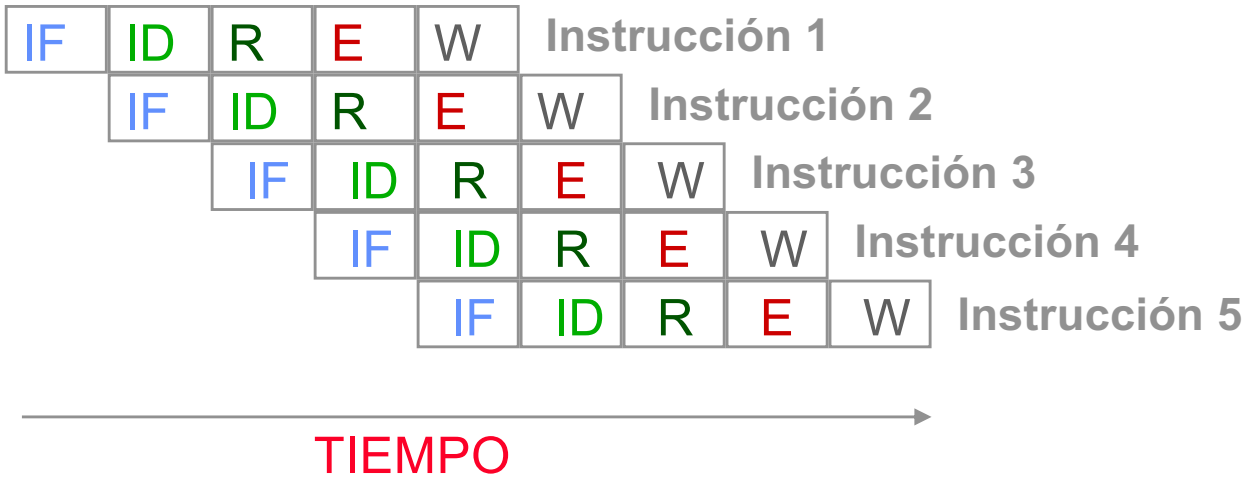


Duración de cada etapa (modelo): ciclo de reloj

Segmentación básica de un procesador

- Etapas de ejecución de instrucciones





Funcionamiento de un procesador segmentado

Segmentación básica de un procesador

- **Ganancia ideal**: número de etapas del cauce segmentado
  - No se alcanza debido a las penalizaciones por
    - llenado del cauce
    - diferencia de duración entre etapas
    - dependencias de datos y control entre las operaciones
    - conflictos por recursos
    - todas las instrucciones han de pasar por todas las etapas del cauce

- Efectos de la segmentación en el $\mathbf{T_{CPU}}$

$$T_{CPU} = NI \text{ x } CPI \text{ x } T_{ciclo}$$

  - No modifica el número de instrucciones (es una técnica a nivel de microarquitectura)
  - SÍ modifica el CPI, reduciéndolo **idealmente** hasta **CPI =1**
  - Puede incrementar el tiempo de ciclo, dado que el control del procesador segmentado es más complejo y esto añade una sobrecarga en tiempo de ejecución

ILP: explotación con técnicas superscalares y VLIW

Optimización del rendimiento del procesador

- Con **segmentación** se reduce el **CPI**, idealmente hasta 1.
- Para mejorar el rendimiento de un procesador hay que reducir el CPI por debajo de 1, es decir, lograr ejecutar más de una instrucción por ciclo.

- Para conseguir CPI < 1 hay que

  - **Encontrar** instrucciones en el código que puedan ejecutarse en paralelo

  - Disponer de recursos HW suficientes para poder **realizar el procesamiento simultáneo** de dichas instrucciones

ILP: explotación con técnicas superscalares y VLIW

Paralelismo entre instrucciones (ILP)

- Capacidad para poder ejecutar varias instrucciones de manera simultánea
- Hay paralelismo entre dos o más instrucciones si no existen entre ellas dependencias de datos ni de control
- El grado de paralelismo en el código es el número máximo de instrucciones que pueden ejecutarse en paralelo y supone un límite al CPI alcanzable
- Sólo es relevante el grado de paralelismo entre instrucciones próximas en un fragmento de código

ILP: explotación con técnicas superscalares y VLIW

- **Paralelismo entre instrucciones (ILP)**: Ejemplo
  - ¿Cuál es el grado de paralelismo en el fragmento de código? ¿Cuál es el número mínimo de ciclos necesarios para su ejecución? ¿Cuántos caminos de ejecución paralelos hacen falta para alcanzar el mínimo?

```
lb $1,d($2)
add $5,$6,$7
sub $4,$1,$4
mul $8,$9,$10
sb $4, d($11)
```

```
lb $1,d($2)
sub $4,$1,$4
sb $4,d($11)
```

```
add $5,$6,$7
```

```
mul $8,$9,$10
```

Grado de paralelismo: 3

```
lb $1,d($2)
sub $4,$1,$4
sb $4,d($11)
```

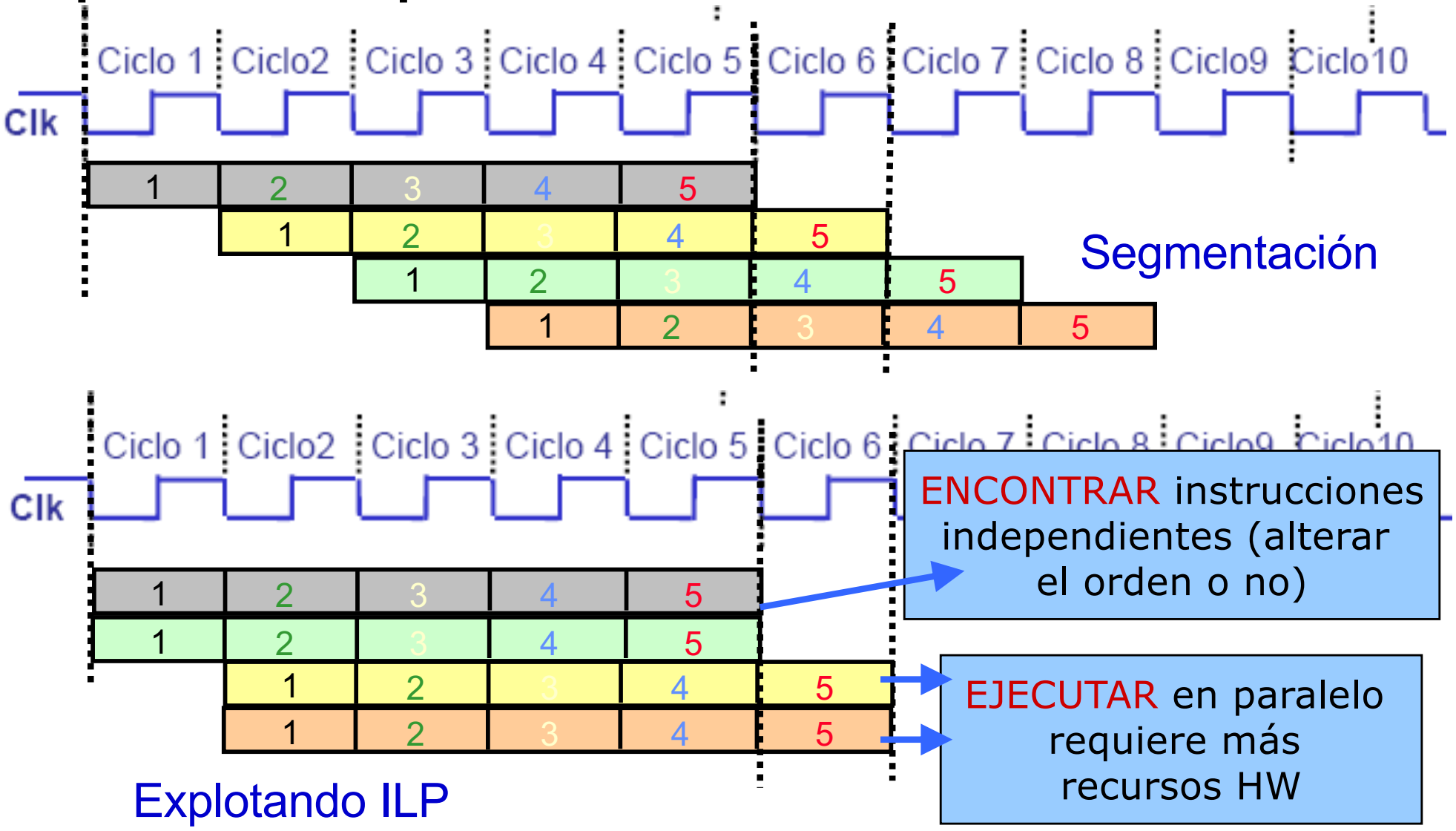
```
add $5,$6,$7
mul $8,$9,$10
```

Número mínimo de ciclos para ejecutarlo: 3 (uno por cada instrucción secuencial)

Dos cauces paralelos es suficiente

ILP: explotación con técnicas superscalares y VLIW

Explotación del paralelismo entre instrucciones



ILP: explotación con técnicas superscalares y VLIW

Explotación del paralelismo entre instrucciones

- Existen dos enfoques para explotar ILP, que se diferencian fundamentalmente en las técnicas aplicadas para encontrar e incrementar las instrucciones independientes que se ejecutan simultáneamente
  - **Procesadores superescalares = explotación dinámica**
    - Las instrucciones que se procesan en cada ciclo las decide el procesador, respetando el orden en que aparecen en el programa o no (en este caso se dice que hay *ejecución fuera de orden*).
    - Procesa un número variable de instrucciones por ciclo.
  - **Procesadores VLIW** (Very Long Instruction Word) **= explotación estática**
    - Es el compilador quien determina qué instrucciones se van a ejecutar en paralelo.
    - Varias instrucciones se codifican en lenguaje máquina como una única instrucción.
    - Procesa un número fijo de instrucciones por ciclo.

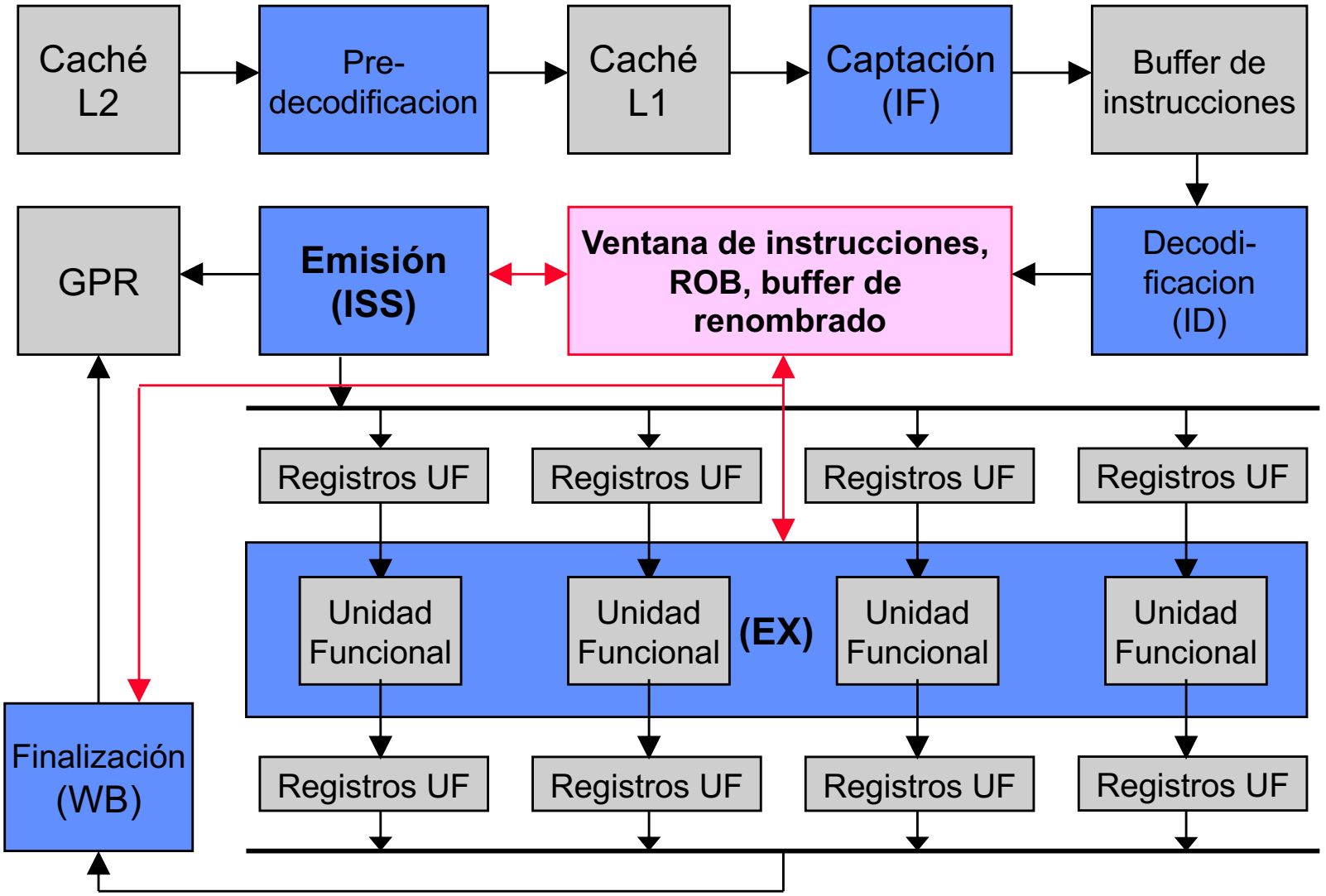
ILP: explotación con técnicas superscalares y VLIW

Etapas en el procesamiento de una instrucción:

- **Búsqueda** o captación de instrucciones: se accede a la caché de instrucciones L1 para la lectura simultánea de varias instrucciones.
- **Decodificación**: decodifica en paralelo varias instrucciones (normalmente el mismo número de instrucción captadas en la etapa anterior); esto implica identificar las instrucciones individuales, su tipo y analizar dependencias entre ellas. La complejidad de esta etapa depende del ISA, dividiéndose a veces en varias etapas para para ISAs de tipo CISC.
- **Emisión**: se encarga de determinar qué instrucciones pueden pasar a ejecución, entre las que tienen sus operandos listos y una unidad de ejecución libre, y enviarlas a dicha UF. La emisión de instrucciones puede hacerse *en orden o fuera de orden*.
- **Ejecución**: realiza la operación que indica la instrucción (una aritmética, lectura de un dato de memoria, etc.). Puede hacerse *en orden o fuera de orden*.
- **Finalización y escritura del resultado**: almacenamiento de los datos calculados en los registros, también varios por ciclo. Puede hacerse *en orden o fuera de orden*.

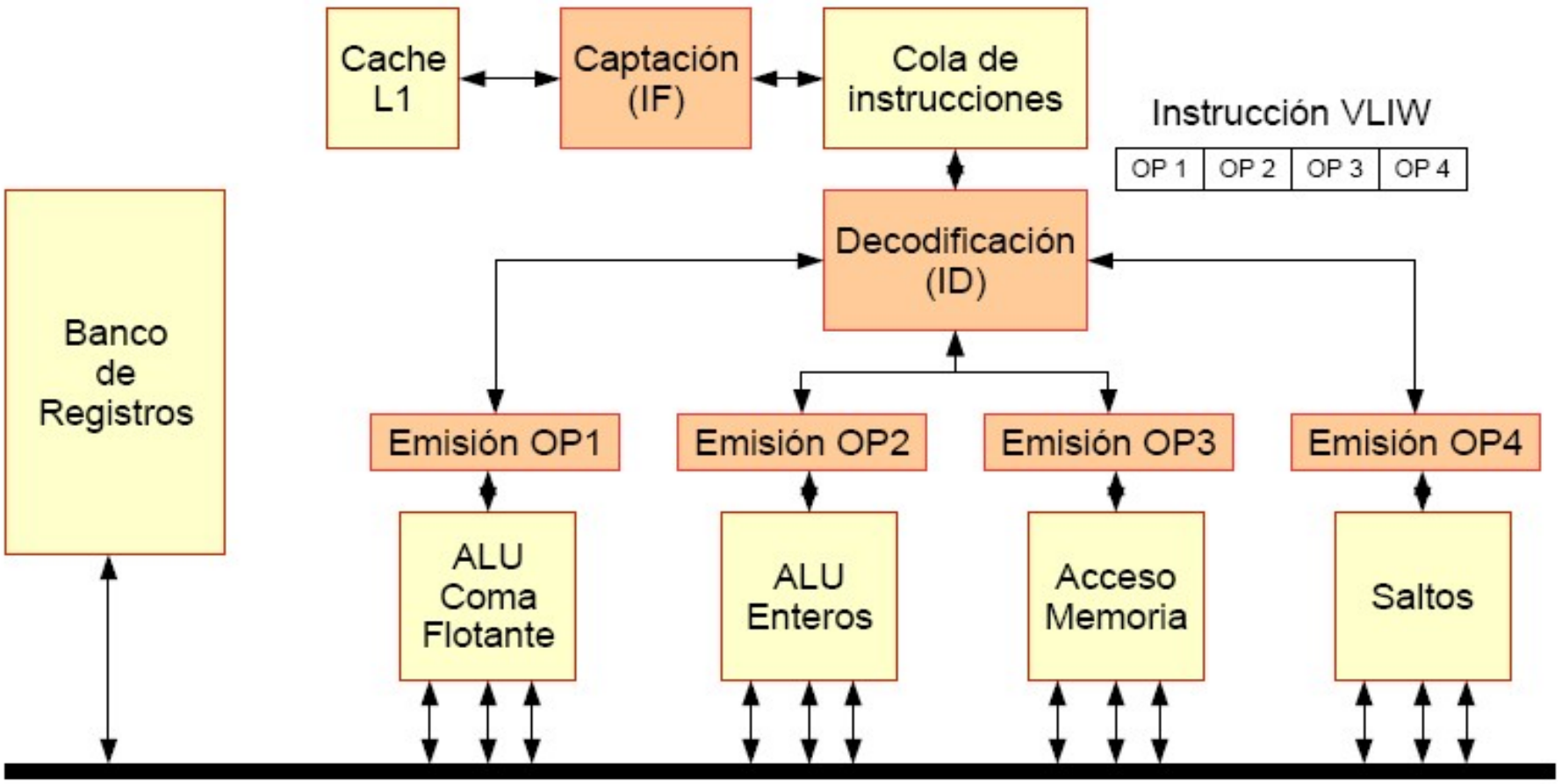
ILP: explotación con técnicas superscalares y VLIW

Organización básica de una microarquitectura superescalar



ILP: explotación con técnicas superscalares y VLIW

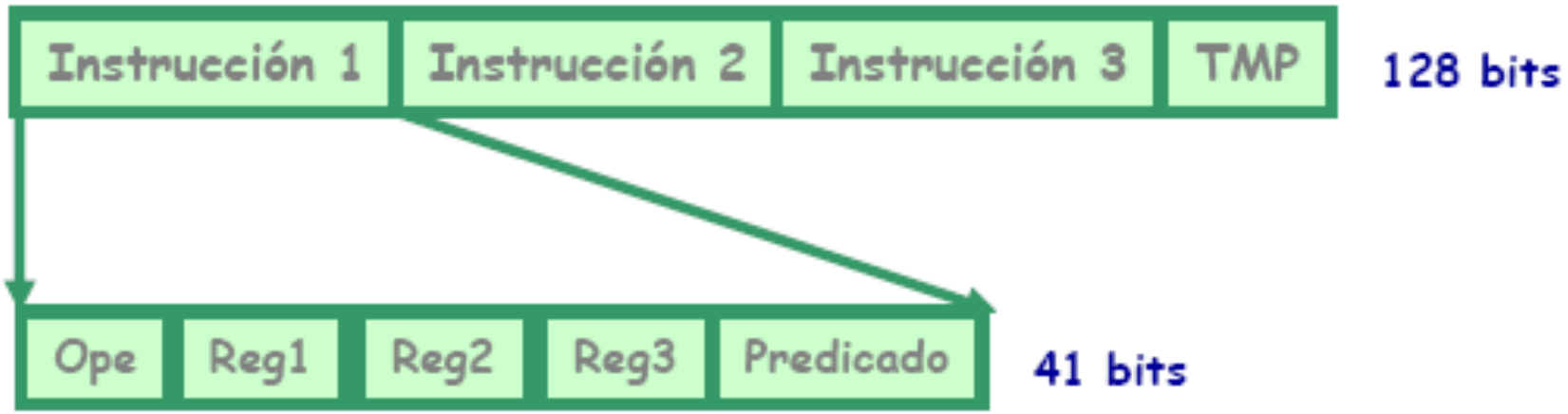
Organización básica de una microarquitectura VLIW



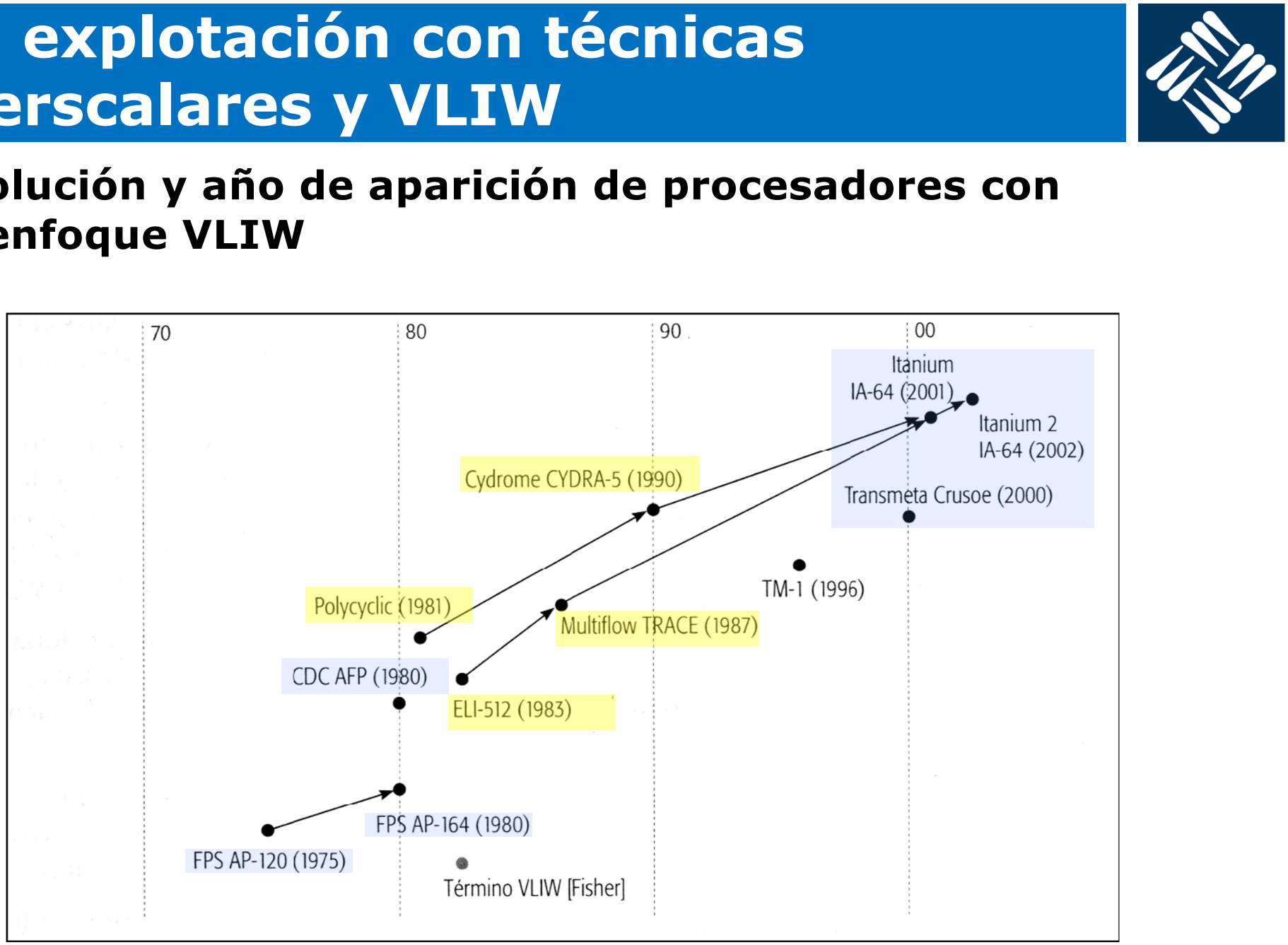
ILP: explotación con técnicas superscalares y VLIW

Procesadores VLIW. Ejemplo: Intanium- EPIC (Explicit Parallel Instruction Computation)

- Las instrucciones máquinas son de 41 bits, que incluyen:
  - Código de operación + Código de predicado
  - Registros fuente
  - Registro destino
- El compilador agrupa tres instrucciones en un paquete de 128 bits, donde se incluye además un campo para indicar el grado de paralelismo entre las instrucciones



ILP: explotación con técnicas superscalares y VLIW



Evolución y año de aparición de procesadores con enfoque VLIW

ILP: explotación con técnicas superscalares y VLIW

Procesadores superescalares:

- Son **procesadores segmentados que disponen de varias unidades** de ejecucion
- El análisis de las posibilidades de ejecución paralela de instrucciones lo realiza el **hardware del procesador**
- Instrucciones **simples** (cada instruccion codifica una operacion)
- Microarquitectura del procesador compleja que implica **altas densidades de integración**

Procesadores VLIM/EPIC:

- Son **procesadores segmentados que disponen de varias unidades** de ejecucion
- El análisis de las posibilidades de ejecución paralela de instrucciones lo realiza el **compilador** (puede contemplar un mayor número de instrucciones)
- Instrucciones **complejas** (cada instruccion codifica varias operaciones)
- Microarquitectura del procesador menos compleja, **menor densidad de integracion** ➔ menor coste y consumo

Multithreading

- Capacidad del sistema operativo de ejecutar diferentes partes de un programa, llamadas **threads** (hilos o hebras), de forma simultánea.
- Ejecutar dos o más threads de forma simultánea puede hacerse
  - Sobre un mismo procesador o core:
    - Cambiando de thread a intervalos fijos de tiempo
    - Cambiando de thread ante eventos dinámicos (estas dos técnicas son similares a la "**multiprogramación**")
    - Mezclando instrucciones de ambos threads a la vez
      - Ejemplo: **Hyper Threading** , tecnología de INTEL (https://www.youtube.com/watch?v=VcoVYfDVEww)
  - Sobre diferentes procesadores o cores: similar a técnicas de multiprocesamiento

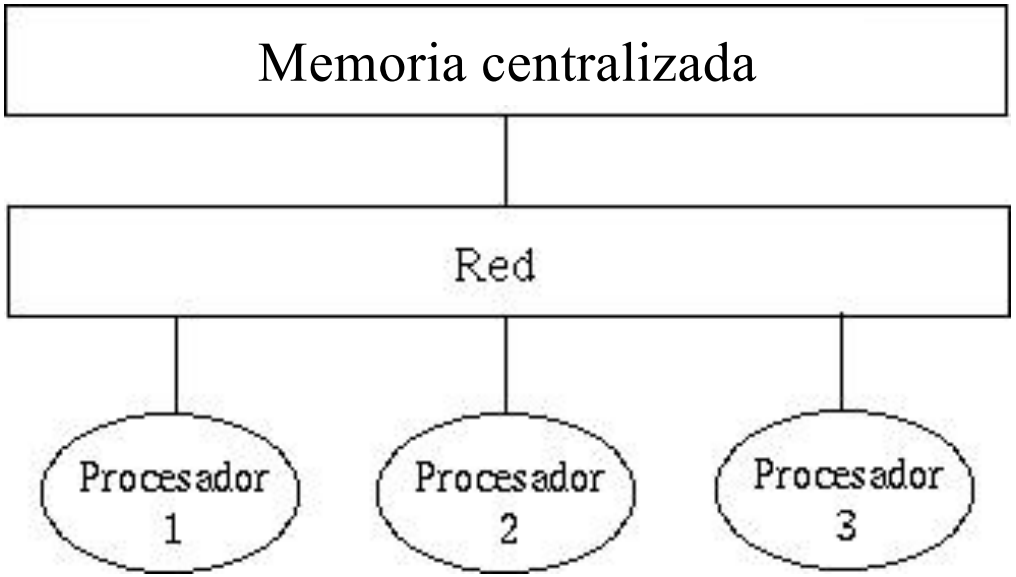
Sistemas multiprocesador

- Sistemas con múltiples procesadores conectados por una "red de interconexión"
  - Si los procesadores se integran dentro de un mismo chip se denominan cores o núcleos
- Mejoran la productividad del sistema, dado que el SO puede lanzar la ejecución simultánea de varias aplicaciones, cada una a un procesador diferente
- Pueden mejorar también el tiempo de respuesta para un programa, si éste se "paraleliza".
- Los aspectos clave para conseguir una buena ganancia al implementar esta técnica son:
  - La comunicación entre procesadores
  - Sincronización
  - Balanceo de carga
- Otro aspecto clave es la organización del sistema de memoria

Sistemas multiprocesador

Organización de la memoria

- Sistemas de **memoria centralizada**: existe una única memoria física, compartida por todo los procesadores
- Se utilizan con sistemas que tienen pocos procesadores

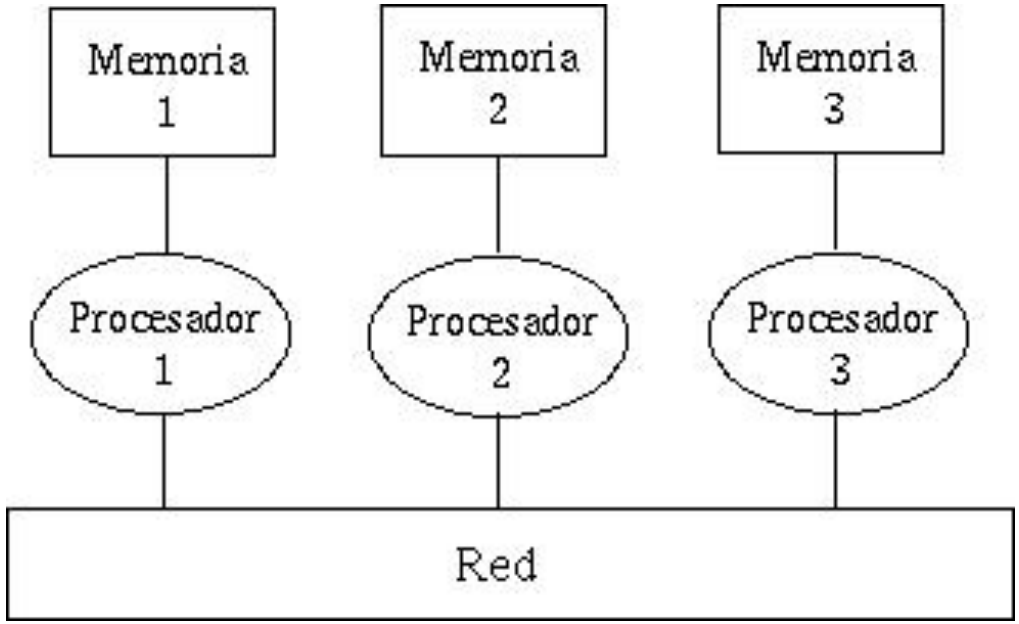


- Los procesadores suelen tener una memoria cache local, que mejora el tiempo de acceso a memoria.
- Cuidado: coherencia de las memorias caché

Sistemas multiprocesador

Organización de la memoria

- Sistemas de **memoria distribuida**: cada procesador dispone de su propia memoria, que puede compartir o no con el resto de procesadores
- Igualmente, es necesario cuidar la coherencia de memoria, uno de los principales problemas de este esquema



Sistemas multiprocesador

Organización de la memoria

- Sistemas de **memoria distribuida**: existen dos modelos de programación para estos sistemas
  - **Memoria compartida**: existe un único espacio de direcciones en el que todos los procesadores leen y escriben. Existen recursos HW para gestionar de forma opaca al procesador la comunicación entre distintas memorias
  - **Paso de mensajes**: la comunicación se realiza de forma explícita en el programa, mediante instrucciones que envían mensajes a otros procesadores