# Módulo 2
# Conceptos de Arquitectura de Computadoras

Polanis, Iván Valentín

# Índice

# 1. Segmentación de instrucciones

En una arquitectura **RISC**[1] la mayoría de las instrucciones son del tipo registro a registro, y un ciclo de instrucción tiene las dos fases siguientes:

- **I**: captación de instrucción
- **E**: ejecución. Realiza una operación de la ALU como entrada y salida.

Las operaciones de carga y almacenamiento necesitan tres fases:

- **I**: captación de instrucción
- **E**: ejecución. Calcula una dirección de memoria.
- **D**: memoria. Operación registro a memoria o memoria a registro.

Dada la simplicidad y regularidad de un repertorio de instrucciones RISC, el diseño de la organización en tres o cuatro etapas se realiza fácilmente.

## 1.1. Segmentación en el MIPS64

- **Búsqueda (F, Fetch)**
  - Se accede a memoria por la instrucción.
  - Se incrementa el Program Counter.
- **Decodificación (D, Decode)**
  - Se decodifica la instrucción, obteniendo operación a realizar en la ruta de datos.
  - Se accede al banco de registros por el/los operandos/s (si es necesario).
  - Se calcula el valor del operando inmediato con extensión de signo (si es necesario).
- **Ejecución (X, Execute)**
  - Se ejecuta la operación en la ALU.
- **Acceso a memoria (M, Memory Access)**
  - Si se requiere un acceso a memoria, se accede.
- **Almacenamiento (W, Writeback)**
  - Si se requiere volcar un resultado a un registro, se accede al banco de registros.

## 1.2. Segmentación de cauce

La segmentación de cauce (pipelining) es una forma particularmente efectiva de organizar el hardware de la CPU para realizar más de una operación al mismo tiempo.

Consiste en descomponer el proceso de ejecución de las instrucciones en fases o etapas que permitan una ejecución simultánea. Explota el paralelismo entre las instrucciones de un flujo secuencial.

La segmentación es una técnica de mejora de prestaciones a nivel de diseño hardware. La misma es invisible al programador.

Es necesario uniformizar las etapas al tiempo de la más lenta.

El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones.

La segmentación de cauce incrementa la productividad, pero no reduce el tiempo de ejecución de la instrucción.

[1] Reduced Instruction Set Computer

### 1.2.1. Atascos de un cauce

Son situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que le corresponde. Estas pueden ser:

- **Estructurales**

  Provocados por conflictos por los recursos.

- **Por dependencia de datos**

  Ocurren cuando dos instrucciones se comunican por medio de un dato.

- **Por dependencia de control**

  Ocurren cuando la ejecución de una instrucción depende de cómo se ejecute otra.

## 1.3. Optimización de la segmentación

Dada la naturaleza sencilla y regular de las instrucciones RISC, los esquemas de segmentación se pueden emplear eficientemente. Hay poca variación en la duración de la ejecución de instrucciones, y el cauce puede adaptarse para reflejar este hecho.

Para compensar las dependencias de datos, se han desarrollado técnicas de reorganización de código. Consideremos primero las instrucciones de salto. El *salto retardado*, que es una forma de incrementar la eficiencia de la segmentación, utiliza un salto que no tiene lugar hasta después de que se ejecute la siguiente instrucción. La posición de la instrucción inmediatamente después de la instrucción de salto se conoce como *espacio de retardo*.

Para saltos condicionales, el procedimiento no puede aplicarse a ciegas. Si la condición comprobada por la bifurcación puede alterarse por la instrucción inmediatamente precedente, el compilador ha de abstenerse de hacer el intercambio en su lugar, debe inserta un NOOP.

Un tipo de táctica similar, llamada *carga retardada*, se puede usar con las instrucciones LOAD. En las instrucciones LOAD, el procesador bloquea el registro destino de la carga. Después el procesador continúa la ejecución del flujo de instrucciones hasta que se alcanza una instrucción que necesite ese registro, deteniéndose hasta que la carga finalice. Si el compilador puede reorganizar las instrucciones de manera que se pueda hacer un trabajo útil mientras la carga está en el cauce, la eficiencia aumenta.

# 2. Soluciones a Atascos

## 2.1. Soluciones a riesgos estructurales

La solución es simple, se debe replicar, segmentar o realizar turnos para el acceso a las unidades funcionales en conflicto.

- Duplicación de recursos de hardware.

  Sumadores o restadores además de la ALU.

- Separación en memorias de instrucciones y datos.

- Turnar el acceso al banco de registro.

  Escrituras en la primera mitad del ciclo de reloj.

  Lecturas en la segunda mitad del ciclo de reloj.

## 2.2. Soluciones a riesgos de datos

Para riesgos **RAW** se debe determinar cómo y cuando aparecen esos riesgos. Para ello será necesario una unidad de detección de riesgos y/o un compilador más complejo.
Para este tipo de riesgo tenemos dos soluciones:

- **Hardware**

  Adelantamiento de operandos (Forwarding).

- **Software**

  Instrucciones NOP o reordenamiento de de codigo.

### 2.2.1. Adelantamiento de operandos (Forwarding)

Esta técnica consiste en pasar directamente el resultado obtenido con una instrucción a las instrucciones que lo necesitan como operando. Si el dato necesario está disponible a la salida de la ALU ($X_i$) se lleva a la entrada de la etapa correspondiente ($X_{i+1}$) sin esperar a la escritura ($M_i$ o $W_i$).

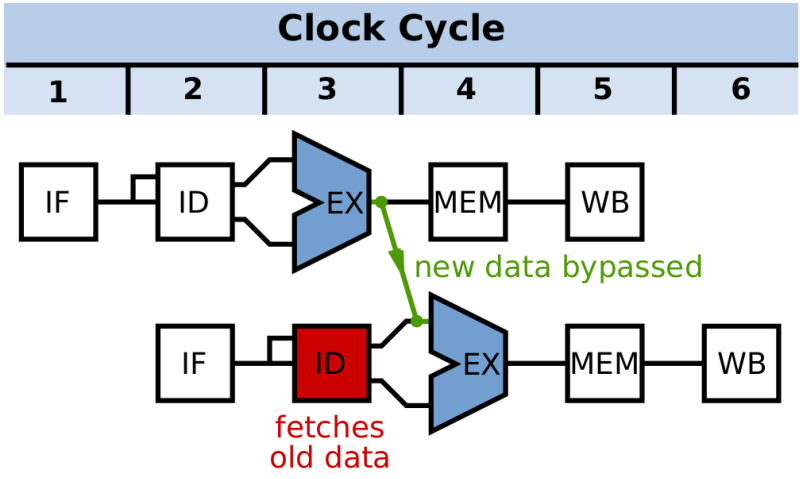


Figura 1: Adelantamiento de operandos.

### 2.2.2. Instrucciones NOP o reordenamiento de código

Evita los riesgos reordenando las instrucciones del código sin afectar el código. El compilador es el encargado de realizar esta tarea.

Aquí se introducen las instrucciones NOP, que son utilizadas para generar un retardo. Estas instrucciones no realizan ninguna operación, pero consumen un ciclo de reloj. Ademas, se reordenan instrucciones para evitar riesgos RAW.

## 2.3. Soluciones a riesgos de control

En los riesgos de control se introduce la penalización por salto. Cuando el salto es incondicional, la dirección de destino se debe determinar lo más pronto posible, dentro del cauce, para reducir la penalización. Cuando el salto es condicional, la penalización se debe a que el procesador no sabe si se debe tomar o no el salto hasta que se evalúe la condición.

Se utiliza una modificación sencilla de la ruta de datos para reducir la cantidad de paradas a un solo ciclo:

- **Adelantar la resolución de los saltos a la etapa D**
  - En ella se decodifica y se sabe que es un salto.
  - Se puede evaluar la condición de salto.
  - Se puede calcular la dirección de destino.

### 2.3.1. Técnicas de predicción de salto (Hardware)

Para este tipo de técnicas tenemos dos posibilidades:

- **Predicción estática**

  Se asume que el salto se toma o no se toma.

- **Predicción dinámica**

  Se utiliza un predictor de saltos.

#### Predicción estática

- **Predicción de salto tomado**

  Se asume que el salto se toma.

  Se adelanta la búsqueda de la dirección de destino.

- **Predicción de salto no tomado**

  Se asume que el salto no se toma.

  Se continúa con la ejecución secuencial.

#### Predicción dinámica

- **Conmutador saltar/no saltar**
  - Basado en la historia de las instrucciones.
  - Es eficaz para los bucles.
- **Tabla de historia de saltos (BTB)**
  - Pequeña cache asociada a la etapa de búsqueda (F).
  - Contiene tres campos.

    Dirección de una instrucción de bifurcación.
    Información de la instrucción destino.
    N bits de estado (histórico).

**Otras soluciones**

Una técnica que se utiliza es la **predicción según el código de operación**, ya que existen instrucciones con más probabilidades de saltar. La tasa de acierto es del 75 %.

Los **Flujos múltiples** utilizan varios cauces (uno por cada opción de salto). Precaptan cada salto en diferentes cauces. Las desventajas son que provoca retardos en el acceso al bus y a los registros. Si hay múltiples saltos, se necesita un mayor número de cauces.

La **Precaptación del destino de salto** consiste en precaptar la instrucción destino del salto, además de las instrucciones siguientes a la bifurcación. La instrucción se guarda hasta que se ejecute la instrucción de bifurcación.

En el **Buffer de bucles** se utiliza una memoria muy rápida gestionada por la etapa de captación de instrucción del cauce. Comprueba el buffer antes de hacer la captación de memoria. Esta técnica es muy eficaz para pequeños bucles y saltos.

### 2.3.2. Instrucciones de salto retardado (Software)

La idea es realizar trabajo útil mientras el salto se resuelve.

- Hueco o ranura de retardo de salto es el período de penalización o parada luego de una instrucción de salto.
- El compilador trata de situar instrucciones útiles (que no dependan del salto) en los huecos de retardo. Si no es posible, se utilizan instrucciones NOP.
- Las instrucciones en los huecos de retardo de salto se captan siempre.
- Requiere reordenamiento de código.

# 3. Computadoras de repertorio reducido de instrucciones

Algunos de los principales avances desde el nacimiento del computador son:

- **El concepto de familia:** introducido por IBM en su System/360 em 1964. El concepto de familia separa la arquitectura de una máquina de su implementación. Se ofrece un conjunto de computadores, con distintas características en cuanto a precio/prestaciones, que presentan al usuario la misma arquitectura.
- **Unidad de control micro programada:** propuesta por Wilkes en 1951, e introducida por iBM en la línea S/360 en 1964. La micro programación facilita la tarea de diseñar e implementar la unidad de control y da soporte al concepto de familia.
- **Memoria caché:** introducida en 1968 por IBM. La introducción de este elemento en la jerarquía de memoria mejoró las prestaciones de manera espectacular.
- **Segmentación de cauce:** una manera de introducir procesamiento simultaneo en la naturaleza esencialmente secuencial de un por instrucciones máquina.

La arquitectura RISC se aparta de manera drástica de la tendencia histórica en la arquitectura del procesador. Un análisis de la arquitectura RISC involucra la mayoría de los asuntos importantes de organización y arquitectura de computadores.

Los sistemas RISC están caracterizados por:

- Un gran número de registros de uso general, o el uso de tecnología de compiladores para optimizar la utilización de los registros.
- Un repertorio de instrucciones limitado y sencillo.
- Un énfasis en la optimización de la segmentación de instrucciones.

## 3.1. Características de la ejecución de instrucciones

Los repertorios complejos de instrucciones están pensados para:

- Facilitar el trabajo del escritor de compiladores.
- Mejorar la eficiencia de la ejecución, ya que las secuencias complejas de operaciones pueden implementarse en micro código.
- Dar soporte a HLL[2] aun más complejos y sofisticados.

Para comprender la línea de razonamiento de los partidarios de los RISC, comenzamos con una breve revisión de las características de la ejecución de instrucciones. Los aspectos cuyo cálculo tiene interés son los siguientes:

- **Operaciones realizadas:** determinan las funciones que lleva a cabo el procesador y su interacción con la memoria.
- **Operandos usados:** los tipos de operandos y su frecuencia de uso determinan la organización de memoria para almacenarlos y los modos de direccionamiento para acceder a ellos.
- **Secuenciamiento de la ejecución:** determina la organización del control y del cauce segmentado.

[2]High Level Language

**Operaciones**

Existe una gran concordancia en los resultados de esta mezcla de lenguajes y aplicaciones. Las sentencias de asignación predominan, lo cual indica que el sencillo movimiento de datos tiene mucha importancia. También predominan las sentencias condicionales (IF, LOOP). Estas sentencias se implementan en el lenguaje máquina con alguna instrucción del tipo comparar y saltar. Esto indica que el mecanismo incluido en el repertorio de instrucciones para el control del secuenciamiento es importante.

Estos resultados son instructivos para el diseñador del repertorio de instrucciones máquina, diciéndole qué tipo de sentencias tiene lugar más a menudo y por consiguiente deben ser implementadas de una forma óptima.

**Operandos**

El estudio de Patterson ya referenciado también consideró la frecuencia dinámica de aparición de distintas clases de variables. Los resultados, coherentes para programas en Pascal y en C, muestran que la mayoría de las referencias se hacen a variables escalares simples. Además, más del ochenta por ciento de los datos eran variables locales. Asimismo, las referencias a matrices/estructuras requieren una referencia previa a su índice o puntero, que nuevamente suele ser un dato escalar local. Por consiguiente, hay un predominio de referencias a operandos escalares, y estos están muy localizados.

Estos estudios reflejan la importancia de una arquitectura que se preste a un rápido acceso a operandos, dado que es una operación que se realiza con mucha frecuencia. El estudio de Patterson indica que un candidato fundamental a optimizar es el mecanismo de almacenamiento y acceso a variables escalares.

**Llamadas a procedimientos**

Las llamadas y retornos de procedimientos constituyen un aspecto importante de los programas en HLL. Los estudios indican que son las operaciones que consumen más tiempo en programas en HLL compilados.

Esto depende del número de parámetros tratados y del nivel de anidamiento. La mayoría de los programas no tienen una larga secuencia de llamadas seguida por la correspondiente secuencia de retornos. Además la mayoría de las variables suelen ser locales y las referencias a operandos están muy localizadas.

Partiendo del trabajo de varios investigadores, surgen tres elementos que, por lo general, caracterizan las arquitecturas RISC:

- **Utilización de un amplio banco de registros**
- **Cuidadosa atención al diseño de los cauces de instrucciones.**
- **Uso de un repertorio simplificado de instrucciones**

## 3.2. Utilización de un amplio banco de registros

Se necesita una estrategia que permita que los operandos a los que se acceda con mayor frecuencia se encuentren en registros y se minimicen las operaciones registro-memoria.

Son posibles dos aproximaciones básicas, una basada en software y la otra en hardware.

La aproximación por software consiste en confiar al compilador la maximización del uso de los registros. El compilador intentará asignar registros a las variables que se usen más en un periodo dado. Esta solución requiere el uso de sofisticados algoritmos de análisis de programas.

La aproximación por hardware consiste sencillamente en usar más registros de manera que puedan mantenerse en ellos más variables durante períodos de tiempo más largos.

### Ventanas de registros

A primera vista, el uso de un conjunto amplio de registros debería reducir la necesidad de acceder a memoria. La labor del diseño es organizar los registros de tal modo que se alcance esa meta.

Una llamada a un procedimiento hace que el procesador conmute automáticamente a una ventana de registros distinta de tamaño fijo, en lugar de salvaguardar los registros en memoria. Las ventanas de procedimientos adyacentes están parcialmente solapadas para permitir el paso de parámetros.

La ventana se divide en tres áreas de tamaño fijo. Los registros de parámetros contienen parámetros pasados al procedimiento actual desde el procedimiento que lo llamó, y los resultados a devolver a este. Los registros locales se usan para variables locales, según la asignación que realice el compilador. Los registros temporales se usan para intercambiar parámetros y resultados con el siguiente nivel más bajo (el procedimiento llamado por el procedimiento en curso). Los registros temporales de un nivel son físicamente los mismos que los registros de parámetros del nivel más bajo adyacente. Este solapamiento posibilita que los parámetros se pasen sin que exista una transferencia de datos real.

Para manejar cualquier patrón posible de llamadas y retornos, el número de ventanas de registros tendría que ser ilimitado. En lugar de eso, las ventanas de registros se pueden usar para contener unas pocas, las más recientes, activaciones de procedimientos. Las activaciones más antiguas han de salvaguardarse en memoria y restaurarse más tarde cuando la profundidad de anidamiento disminuya.

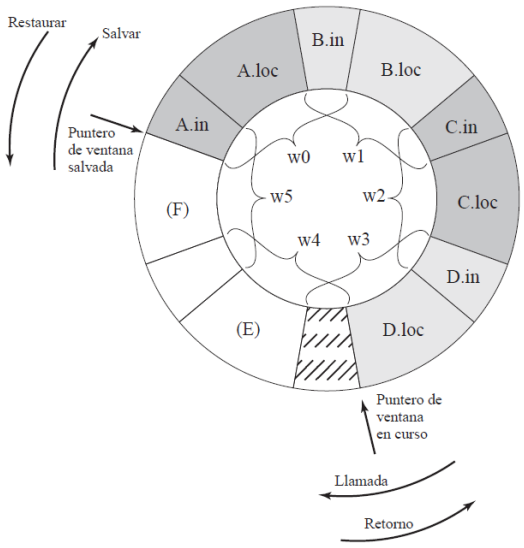


Figura 2: Organización de las ventanas solapadas con un buffer circular.

### Variables Globales

Para tratar variables globales se sugieren dos opciones.

La primera es que el compilador asigne posiciones de memoria a las variables declaradas como globales en un HLL, y que todas las instrucciones máquina que referencien esas variables usen operandos referenciados en memoria.

Una alternativa es incorporar al procesador un conjunto de registros globales. Estos registros serían fijos en cuanto a su número y accesibles por todos los procedimientos. Se puede usar un esquema de numeración unificado para simplificar el formato de instrucción. Esto conlleva un hardware añadido, que se encarga de adaptar la división en el direccionamiento de los registros.

**Un amplio banco de registros frente a una caché**

| Banco de registros amplio | Caché |
|---|---|
| Todos los datos escalares locales | Datos escalares locales recientemente usados |
| Variables individuales | Bloques de memoria |
| Variables globales asignadas por el compilador | Variables recientemente |
| Salvaguarda basadas en la profundidad de anidamiento | Salvaguarda basadas en el algoritmo de reemplazo |
| Direccionamiento a registros | Direccionamiento a memoria |

## 3.3. Optimización de registros basada en el compilador

Supongamos que ahora disponemos de una máquina RISC que contiene unicamente un pequeño número de registros. En ese caso, el uso optimizado de registros es responsabilidad del compilador. Los programas escritos en HLL no tienen referencias explicitas a los registros. El objetivo del compilador es mantener en registros en lugar de en memoria los operandos necesarios para tantos cálculos como sea posible, y minimizar las operaciones de carga y almacenamiento.

Por lo general se sigue el siguiente enfoque. Cada cantidad del programa candidata para residir en un registro se asigna a un registro simbólico o virtual. El compilador entonces asigna el número ilimitado de registros simbólicos a un número fijo de registros reales. Los registros simbólicos cuya utilización no se solape pueden compartir el mismo registro real. Si en una parte concreta del programa hay más cantidades a tratar que registros reales, algunas de las cantidades se asignan a posiciones de memoria.

## 3.4. Arquitectura de repertorio reducido de instrucciones

### ¿Por qué CISC?

La primera de las razones es, la simplificación de los compiladores. La labor del escritor de compiladores es generar una secuencia de instrucciones máquina para cada sentencia HLL. Si existen instrucciones máquina que se parezcan a sentencias del HLL, la tarea se simplifica.

La otra razón importante mencionada es la esperanza de que un CISC produzca programas más pequeños y rápido. Los programas más pequeños tienen dos ventajas. Como el programa ocupa menos memoria, hay un ahorro de este recurso, pero como la memoria hoy en día es tan barata, este aspecto no es primordial. Tiene mayor importancia el hecho de que programas más pequeños mejoren las prestaciones porque tienen menos instrucciones, lo que significa que hay que captar menos bytes de instrucciones. El segundo es que en un entorno paginado los programas más pequeños ocupan menos páginas, reduciendo los fallos de página. Sin embargo, el número de bits de memoria que ocupa, no tiene por qué ser más pequeño al tener menos instrucciones.

El segundo factor que motivaba repertorios de instrucciones cada vez más complejos era que la ejecución de instrucciones fuera más rápida. Parece tener sentido el que una operación compleja de un HLL se ejecute más rápido como una única instrucción máquina que como una sucesión de instrucciones más primitivas. Sin embargo, debido a la propensión a usar las instrucciones más sencillas, esto puede no ser así. La unidad de control completa debe hacerse más compleja, y/o la memoria de control del micro programa ha de hacerse más grande, para acomodar un repertorio de instrucciones más rico.

### Características de RISC

- Una instrucción por ciclo.
- Operaciones registro a registro.

- Modos de direccionamiento sencillos.
- Formatos de instrucción sencillos.

Un *ciclo máquina* se define como el tiempo que se tarda en captar dos operandos desde dos registros, realizar una operación de la ALU y almacenar el resultado en un registro. Así, las instrucciones máquina de un RISC no deberían ser más complicadas que las micro instrucciones de las máquinas CISC, y deberían comportarse más o menos igual de rápido.

Una segunda característica es que la mayoría de las operaciones deben ser del tipo **registro a registro** con la excepción de sencillas operaciones LOAD y STORE para acceder a memoria.

Casi todas las instrucciones RISC usan direccionamiento sencillo a registro. Se pueden incluir varios modos adicionales, como el desplazamiento y el relativo al contador de programa.

Por último, solo se usa un formato o unos pocos. La longitud de las instrucciones es fija y alineada en los limites de una palabra. Las posiciones de los campos, especialmente la del código de operación, son fijas.

# 4. Memoria

## 4.1. Diseño de una jerarquía de memoria básica

El sistema de memoria formado por una memoria principal única que se propuso en los primeros modelos de computador sencillo quedó descartado rápidamente por sus bajas prestaciones.

La alternativa a esta memoria única es una jerarquía de memoria organizada en niveles, cuanto más cercanos al procesador, más pequeños, rápidos y caros. Hoy en día esta jerarquía incluye, en casi todos los casos, tres tipos de memorias: memoria caché, memoria principal y memoria virtual.

- **Memoria Caché (MC):** Se ubica dentro del mismo chip que el procesador, fabricada con *RAM* estática (SRAM) y controlada por el controlador de caché incluido en el mismo chip. Hoy en día lo habitual es que haya varios niveles de caché.
- **Memoria Principal (MP):** Se ubica en un chip diferente al procesador, fabricada con RAM dinámica (DRAM) y controlada por el controlador de memoria principal. Este controlador es muy importante para el rendimiento de la jerarquía de memoria ya que se encarga de la planificación de los accesos a memoria principal. Hoy en día puede ubicarse en el mismo chip que el procesador y la memoria caché, o en otro chip como el el chipset norte o el MCH.
- **Memoria virtual (MV):** Ubicada en la actualidad en el disco duro, se fabrica por lo tanto con tecnología magnética y se controla desde el sistema operativo a través del controlador del disco duro.

El objetivo es conseguir una estructura de memoria de gran capacidad, con un coste casi tan bajo como el del nivel más barato de la jerarquía, pero con latencia comparable a la del nivel más rápido.

Hay dos propiedades que la jerarquía debe cumplir para que su funcionamiento sea adecuado:

- **Inclusión:** implica que cualquier información contenida en un nivel de la jerarquía debe estar también en los niveles superiores (en los que están más lejos del procesador a partir de él):
- **Coherencia:** garantiza que las copias de la misma información en los diferentes niveles de la jerarquía son coherentes entre sí, es decir, que almacenan los mismos valores.

Cuando el procesador realiza un acceso a memoria, primero se busca la palabra que hay que leer o escribir en la memoria caché **(MC)**. Si esta palabra se encuentra en la caché, ha ocurrido un acierto. Si por el contrario, no se encuentra a ocurrido un fallo.

En este último caso, se traerá de memoria principal **(MP)** un bloque que contendrá varias palabras, entre ellas, la que ha producido el fallo.

El **principio de localidad** tiene dos aspectos diferentes:

- **Localidad espacial:** es la tendencia del procesador a referenciar elementos de memoria cercanos a los últimos que han sido referenciados.
- **Localidad temporal:** es la tendencia del procesador a referenciar elementos de memoria que han sido referenciados recientemente.

Al acceder a memoria principal también pueden producirse aciertos y fallos, ya que no toda la información cabe al mismo tiempo en este nivel. La relación entre la memoria principal y la memoria virtual es similar a la que existe entre la memoria caché y la principal.

La memoria principal se divide en **páginas o segmentos**, de mayor tamaño que los bloques de caché. Cuando se produce un fallo con una página o segmento que no se encuentra en la memoria principal, deberá traerse de la memoria virtual.

La principal diferencia con los fallos de la memoria caché está en que la penalización por este tipo de fallos es bastante grande, ya que la memoria virtual es la más lenta de toda la jerarquía. Por ello el procesador suele pasar a realizar otro tipo de tareas, realizando un cambio de contexto, hasta que la página o segmento esté disponible en la memoria principal.

## 4.2. Mecanismo completo de acceso a memoria

En primer lugar, el procesador genera la dirección virtual de la palabra que se debe leer o escribir. Normalmente, lo primero que se hace es traducir esta dirección virtual a una dirección física comprensible para la jerarquía de memoria.

Si se consigue realizar esta traducción de dirección, es porque la palabra que está buscando se encuentra actualmente en la memoria principal. Con esta dirección se accede a la memoria caché, y si la palabra buscada se encuentra en este nivel, el acceso a memoria ya ha finalizado.

Si por el contrario, se produce un fallo, se busca la palabra en el siguiente nivel, en este caso en la memoria principal, (si existiese otro nivel de cache, se buscaría primero allí).

La resolución de un fallo siempre implica buscar el bloque necesario en la memoria principal.

Cuando se realice el acceso, todo el bloque en el que se incluye la palabra solicitada por el procesador se envía a la memoria caché para resolver su fallo, y el acceso puede completarse con éxito.

Si por el contrario, la palabra no se encuentra en memoria principal y desde un principio no se pudo traducir su dirección virtual a dirección física por este motivo, se debe resolver el fallo de página o segmento desde la memoria virtual. El sistema operativo realiza un cambio de contexto para que otro proceso pase a ejecutarse en el procesador mientras se trae desde memoria virtual la página o segmento, que hace falta para resolver el fallo. Una vez que esté en memoria principal esta página o segmento, ya puede llevarse el bloque correspondiente a memoria caché y cuando el proceso que había provocado el fallo vuelva a pasar a ejecución, el acceso puede completarse con éxito.

Para un procesador segmentado, los accesos a memoria en la etapa **MEM**, en su mayoría, sólo tienen en cuenta los accesos a caché, y si hay que resolver un fallo, el tiempo que se tarda en resolver este fallo se considera un tiempo extra que hay que sumar al tiempo que tarda en ejecutarse una instrucción.

## 4.3. Evaluación de prestaciones de la jerarquía de memoria

En el caso de la jerarquía de memoria también se puede definir una ecuación de prestaciones que proporciones una herramienta cuantitativa de evaluación de rendimiento.

En este caso lo que interesa saber es cuánto tiempo le cuesta en media al procesador realizar un acceso a memoria. Si la memoria caché fuera perfecta y no fallara nunca, el tiempo medio de acceso a memoria sería justo el tiempo de acierto a la memoria caché. Pero como se producen fallos, el tiempo medio de acceso a memoria se calcula teniendo en cuenta estos fallos y el tiempo que se invierte en resolverlos, lo que se ha denominado penalización por fallo:

$$t_{MEM} = t_{aciertoMC} + TF \cdot pF$$

$t_{aciertoMC}$: Tiempo de acierto de MC

$TF$: Tasa de fallos de MC $\rightarrow$ $TF = \dfrac{\text{número de fallos}}{\text{número total de accesos a memoria}}$

$pF$: Penalización por fallo en MC

Si lo que interesa es evaluar las prestaciones de la memoria principal o de la memoria virtual como niveles aislados de la jerarquía, las métricas que se suelen utilizarse son:

- **Latencia:** tiempo que transcurre desde que un acceso a memoria comienza hasta que finaliza. Está muy relacionada con la tecnología con la que está fabricada la memoria.
- **Ancho de banda:** cantidad de información por unidad de tiempo que puede transferirse desde/hacia la memoria. En este caso está muy relacionado con la organización de la memoria más que con la tecnología.

## 4.4. Niveles de la jerarquía de memoria

### 4.4.1. Diseño de la memoria caché

La memoria caché almacena en cada momento unos determinados bloques de información, por lo tanto se divide en marcos capaces de albergar estos bloques en su interior.

La caché no sólo se compone de marcos, ya que para determinar qué bloque está ocupando un determinado marco en un instante concreto se utilizan etiquetas que también deben almacenarse en la memoria. Esta etiquetas se comparan con la del bloque que se está buscando para determinar si éste se encuentra o no en la memoria caché.

#### Organización de la memoria caché

El primer aspecto importante a decidir es el tamaño de la cache, no puede ser demasiado pequeña ya que aumentaría el número de fallos porque no se capturaría bien la localidad.

Tampoco puede ser demasiado grande por dos motivos, el primero es que va integrada en el chip de la cpu y el segundo es que una cache muy grande requeriría una lógica más compleja, por lo que sería más lenta.

El segundo aspecto a tener en cuenta es el tamaño de marco que se va a manejar. Devuelta tenemos el mismo problema, si capturamos bloques muy grandes se captura mejor la localidad espacial, pero a su vez aumenta la penalización por fallo, ya que se necesitaría más tiempo para traer los bloques del siguiente nivel de la jerarquía de memoria.

En muchos casos, la segmentación del procesador obligará a dividir la caché de instrucciones y la de datos para evitar los riesgos estructurales entre las etapas de búsqueda de instrucción y las de acceso a memoria para la lectura o escritura de operandos/resultados.

Por último, destacar dentro de la organización de la caché uno de los aspectos que más influye en el rendimiento de la jerarquía de memoria: la implementación de cachés multinivel.

Normalmente se utilizan dos niveles de memoria caché:

- **L1:** este nivel es el más cercano al procesador, por lo tanto, esta memoria caché es pequeña y rápida.
- **L2:** es el siguiente nivel de la jerarquía, de mayor tamaño (por lo que aprovecha mejor el principio de localidad) y por lo tanto, más lento aunque con menos fallos de capacidad.

De esta manera, cuando se produzca un fallo en el nivel 1 de la memoria, la penalización de este fallo será menor que si sólo hubieran un nivel de caché, porque en lugar de ir a la memoria principal irá a la caché de nivel 2.

#### Política de ubicación

Hay tres tipos de ubicación:

- **Directo:** a cada bloque le corresponde un único marco de caché y sólo puede alojarse en este marco.
- **Asociativo:** un bloque puede alojarse en cualquier marco de la memoria caché.
- **Asociativo por conjuntos:** la memoria caché se divide en conjuntos con un número determinado de marcos por conjunto. A un determinado bloque le corresponde un único conjunto, pero dentro de él, puede alojarse en cualquier marco.

Las diferentes alternativas para la ubicación en la memoria caché llevan a diferentes interpretaciones de la dirección física desde el punto de vista de esta memoria. Según el tipo de ubicación de la memoria caché:

- **Directo:** el índice indica al marco que le corresponde a ese bloque de memoria.

- **Asociativo:** no existe este campo en la dirección, ya que el bloque puede alojarse en cualquier marco.
- **Asociativo por conjuntos:** el índice indica el conjunto que le corresponde a ese bloque de memoria.

### Política de reemplazo

Además de decidir la política de ubicación de la memoria caché, también es necesario definir qué ocurre cuando se produce un fallo. Si no se encuentra un determinado bloque de memoria en la caché, habrá que traerlo del siguiente nivel de la jerarquía de memoria. Pero, ¿qué bloque se reemplaza?

En el caso de una caché directa, no cabe ninguna duda, puesto que cada bloque sólo puede alojarse en un determinado marco. Pero en el caso de memorias con una asociatividad mayor, el reemplazamiento puede hacerse con distintos tipos de políticas.

- **Aleatoria:** se utiliza un generador de números aleatorios para escoger el bloque que se reemplaza.
- **Least Recently Used (LRU):** se escoge el bloque que lleva más tiempo sin utilizarse. Asi se minimiza la probabilidad de sustituir un bloque que vaya a necesitarse en el futuro.
- **First In, First Out (FIFO):** se reemplaza el bloque que lleva más tiempo en la caché.
- **Least Frequently Used (LFU):** esta política reemplaza el bloque que ha sido accedido menos veces en un período de tiempo determinado, es decir, a aquel que se accedió una menor cantidad de veces. Requiere controles de uso (hardware que lleve el conteo de cuantas veces se accedió a un dato de un bloque).

### Política de escritura

Las escrituras llevan más tiempo ya que no pueden solaparse con la comparación de las etiquetas como se hace con las lecturas. En muchos casos se recupera la información de un marco determinado de caché antes de saber si el bloque que está ubicado en él es el que se está buscando. Esto sólo puede hacerse con operaciones de lectura, ya que si finalmente el bloque no es el adecuado, no se ha modificado su contenido.

Pero una operación de escritura sólo puede hacerse si la comparación de etiquetas da como resultado que el bloque ubicado en ese marco es el que se estaba buscando; no se puede hacer trabajo en paralelo:

- **Escritura directa:** cuando se modifica una palabra con una instrucción de almacenamiento, se realiza la escritura en el primer nivel de caché y en el siguiente nivel de la jerarquía.
- **Post-Escritura:** cuando se modifica una palabra sólo se hace en el primer nivel de la memoria caché. Cuando este bloque modificado sea reemplazado, se actualizará el contenido del nivel siguiente. Para no actualizar la memoria con cada reemplazamiento que se haga, suele utilizarse un bit que indica si el bloque que está en cache ha sido modificado (sucio) y por lo tanto debe actualizarse en memoria, o no (limpio).

Por último, en el caso de algunas jerarquías de memoria, para evitar la penalización extra que las escrituras implican se ignoran los fallos de escritura, especialmente en el caso de las memorias de escritura directa.

Por lo tanto suelen encontrarse dos opciones:

- **Write Allocate:** los fallos de escritura se comportan como los fallos de lectura, se trae el bloque que ha provocado el fallo a la memoria caché y se realiza en función de la política utilizada. Habitual con Post-Escritura.

- **No-Write Allocate:** cuando se produce un fallo de escritura, el bloque se modifica directamente en el siguiente nivel de la jerarquía de memoria y no se trae a la caché. Habitual con Escritura directa.

# 5. Buses

Un computador es una red de módulos elementales. Por consiguiente, deben existir lineas para interconectar estos módulos.

Existen distintos tipos de interconexiones para los distintos tipos de unidades:

- **Memoria:** un módulo de memoria está constituido por N palabras de la misma longitud. A cada palabra se le asigna una única dirección numérica (entre 0 y N-1). Una palabra de datos puede leerse de o escribirse en memoria. El tipo de operación se indica mediante las señales de control *Read* y *Write*. La posición de memoria para la operación se especifica mediante una dirección.
- **Módulo de E/S:** Hay dos tipos de operaciones, leer y escribir. Además, un módulo de E/S puede controlar más de un dispositivo externo. Nos referiremos a cada una de estas interfaces con un dispositivo externo con el nombre de *puerto*, y se le asignara una dirección a cada uno. Por otra parte, existen líneas externas de datos para la entrada y la salida de datos por un dispositivo externo. Por último, un módulo de E/S puede enviar señales de interrupción al procesador.
- **Procesador:** el procesador lee instrucciones y datos, escribe datos una vez los ha procesado, y utiliza ciertas señales para controlar el funcionamiento del sistema. También puede recibir señales de interrupción.

## 5.1. Interconexión con buses

Un bus es un camino de comunicación entre dos o más dispositivos. Una característica clave de un bus es que se trata de un medio de transmisión compartido. Al bus se conectan varios dispositivos, y cualquier señal transmitida por uno de esos dispositivos está disponible para que los otros dispositivos conectados al bus puedan acceder a ella. Solo un dispositivo puede transmitir con éxito en un momento dado.

### 5.1.1. Estructura del bus

El **bus del sistema** está constituido, usualmente, por entre cincuenta y cien lineas. A cada línea se le asigna un significado o una función particular. Aunque existen diseños de buses muy diversos, se suelen clasificar en tres grupos funcionales:

#### Líneas de datos

Proporcionan un camino para transmitir datos entre los módulos del sistema. El conjunto constituido por estas líneas se denomina **bus de datos**. El bus de datos puede incluir entre 32 y cientos de lineas, cuyo número se conoce como *anchura* del bus de datos. Puesto que cada línea solo puede transportar un bit cada vez, el número de líneas determina cuántos bits se pueden transferir al mismo tiempo.

#### Líneas de dirección

Se utilizan para designar la fuente o el destino del dato situado en el bus de datos. Claramente, la anchura del bus de direcciones determina la máxima capacidad de memoria posible en el sistema. Además, las líneas de direcciones generalmente se utilizan también para direccionar los puertos de E/S dentro de un módulo. Usualmente, los bits de orden más alto se utilizan para seleccionar una posición de memoria o un puerto de E/S dentro de un módulo.

**Lineas de control**

Se utilizan para controlar el acceso y el uso de las líneas de datos y de direcciones. Puesto que las líneas de datos y de direcciones son compartidas por todos los componentes, debe existir una forma de controlar su uso. Las señales de control transmiten tanto órdenes como información de temporización entre los módulos. Las señales de temporización indican la validez de los datos y las direcciones. Las señales de órdenes especifican las operaciones a realizar.

### 5.1.2. ¿Cómo es un bus?

Físicamente, el bus de sistema es de hecho un conjunto de conductores eléctricos paralelos. Estos conductores son líneas de metal grabadas en una tarjeta. El bus se extiende a través de todos los componentes del sistema, cada uno de los cuales se conecta a algunas o a todas las líneas del bus.

Cada uno de los componentes principales del sistema ocupa una o varias tarjetas y se conecta al bus a través de esas ranuras. Los sistemas actuales tienden a tener sus componentes principales en la misma tarjeta y los circuitos integrados incluyen más elementos. Asi, el bus que conecta el procesador y la memoria caché se integra con el microprocesador junto con el procesador y la caché (on-chip), y el bus que conecta el procesador con la memoria y otros componentes se incluye en la tarjeta (on-board).

### 5.1.3. Jerarquías de búses múltiples

Si se conecta un gran número de dispositivos al bus, las prestaciones pueden disminuir. Hay dos causas principales:

- En general, a más dispositivos conectados al bus, mayor es el retardo de propagación. Este retardo determina el tiempo que necesitan los dispositivos para coordinarse en el uso del bus.
- El bus puede convertirse en un cuello de botella a medida que las peticiones de transferencia acumuladas se aproximan a la capacidad del bus.

Por consiguiente, la mayoría de los computadores utilizan varios buses, normalmente organizados jerárquicamente.

- **Bus del sistema y bus de memoria:** son los que conectan el procesador con el resto del sistema y la memoria principal con el controlador de memoria respectivamente. Se trata de buses rápidos y cortos, propietarios y optimizados para arquitecturas y diseños específicos. Esta optimización es posible ya que a estos buses se conectan un número fijo de dispositivos de prestaciones conocidas.
- **Buses de expansion:** se trata de buses más largos y lentos, abiertos, accesibles por el usuario y a los que se conectan un número indeterminado de dispositivos de prestaciones desconocidas muy diferentes entre sí.

Es posible conectar controladores de E/S directamente al bus de sistema. Una solución más eficiente consiste en utilizar uno o más buses de expansión. La interfaz del bus de expansión regula las transferencias de datos entre el bus de sistema y los controladores conectados al bus de expansión. Esta disposición permite conectar al sistema una amplia variedad de dispositivos de E/S y al mismo tiempo aislar el tráfico de información entre la memoria y el procesador del tráfico correspondiente a la E/S.

Esta arquitectura de buses tradicional es razonablemente eficiente, pero muestra su debilidad a medida que los dispositivos de E/S ofrecen prestaciones cada vez mayores. La respuesta común a esta situación, por parte de la industria, ha sido proponer un bus de alta velocidad que está estrechamente integrado con el resto del sistema, y requiere solo un adaptador (bridge) entre el bus del procesador y el bus de alta velocidad.

#### Tipos de buses

Los **buses dedicados**, usan líneas separadas para direcciones y para datos. Suelen tener 16 líneas de direcciones y 16 líneas de datos. Tienen una línea de control de lectura ó escritura.

Los **buses multiplexados**, usa las mismas líneas, tienen 16 líneas que pueden ser tanto para direcciones como para datos, una línea de control para escritura ó lectura y una línea de control para definir direcciones ó datos.

#### Arbitraje del bus

El control del bus puede necesitar más de un módulo. Solamente una unidad puede transmitir a través del bus en un instante dado. Los métodos de arbitraje se pueden clasificar en:

- **Centralizado:** un único dispositivo hardware, denominado *controlador del bus* o *árbitro*, es responsable de asignar tiempos en el bus. El dispositivo puede estar en un módulo separado o ser parte del procesador.
- **Distribuido:** no existe un controlador central. En su lugar, cada módulo dispone de lógica para controlar el acceso y los módulos actúan conjuntamente para compartir el bus.

En ambos métodos de arbitraje, el propósito es designar un dispositivo, el procesador o un módulo de E/S como maestro del bus. El maestro podría entonces iniciar una transferencia de datos con otro dispositivo, que actúa como esclavo en este intercambio concreto.

#### Temporización

El término temporización hace referencia a la forma en la que se coordinan los eventos en el bus. Los buses utilizan temporización síncrona o asíncrona.

- **Temporización síncrona:** La presencia de un evento está determinada por un reloj. El bus incluye una línea de reloj. Un intervalo desde un *uno* seguido de otro a *cero* se conoce como ciclo de bus. Todos los dispositivos del bus pueden leer la línea de reloj. Suele sincronizar en el flanco de subida. La mayoría de los eventos se prolongan durante un único ciclo de reloj.
- **Temporización asíncrona:** la presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo. El módulo de memoria correspondiente decodifica la dirección y responde proporcionando el dato en la línea de datos. Una vez estabilizadas las líneas de datos, el módulo de memoria activa la línea de *reconocimiento* para indicar al procesador que el dato está disponible. Cuando el maestro ha leído el dato de las líneas correspondientes, deshabilita la señal de lectura. Esto hace que el módulo de memoria libere las líneas de datos y reconocimiento. Por último, una vez se ha desactivado la línea de reconocimiento, el procesador quita la información de dirección de las líneas correspondientes.

## 5.2. Diseño de buses de E/S

Es necesario diseñar el protocolo de transferencia que gobierna la operación del bus y especifica cómo se utilizan las señales de datos, control y direcciones para realizar una transferencia de información completa.

Además es necesario decidir el tipo de protocolo que sincroniza las transferencias de información y en este casi si que existe un conjunto muy reducido de alternativas. En los buses sincrónicos las transferencias están gobernadas por una única señal de reloj compartida por todos los dispositivos que se conectan al bus, de manera que cada transferencia se realiza en un número fijo de ciclos de reloj.

Los protocolos sincrónicos son muy sencillos y sólo necesitan una señal de reloj. Pero hay que adaptar esta señal al dispositivo más lento y además es necesario distribuirla a todos los dispositivos conectados al bus. También existen confirmaciones en las transferencias por lo que es difícil implementar mecanismos de detección y corrección de errores. Con los protocolos asíncronos

no existen estos inconvenientes, pero son menos eficientes debido a la necesidad de intercambiar señales de control.

Por eso surgen los buses semisíncronos, que combinan las ventajas de los dos anteriores, se comportan como síncronos para dispositivos rápidos y como asíncronos para dispositivos lentos.

## 5.3. PCI

El bus **PCI** es un bus muy popular de ancho de banda elevado, independiente del procesador, que se puede utilizar como bus de periféricos o bus para una arquitectura de entreplanta. Comparado con otras especificaciones comunes de bus, el PCI proporciona mejores prestaciones para los subsistemas de E/S de alta velocidad. El PCI ha sido diseñado específicamente para ajustarse, económicamente a los requisitos de E/S de los sistemas actuales; se implementa con muy pocos circuitos integrados y permite que otros buses se conecten al bus PCI.

El PCI esta diseñado para permitir una cierta variedad de configuraciones basadas en microprocesadores, incluyendo sistemas tanto de uno como de varios procesadores. Por consiguiente, proporciona un conjunto de funciones de uso general. Utiliza temporización síncrona y un esquema de arbitraje centralizado.

## 5.4. Evolución jerarquía de buses

*Ver páginas 117 a 120 de Diseño y Evaluación de arquitectura de computadores.*

# 6. Procesadores Superescalares

Una implementación superescalar de la arquitectura de un procesador es aquella en la que las instrucciones comunes pueden iniciar su ejecución simultáneamente y ejecutarse de manera independiente. Estas implementaciones plantean complejos problemas de diseño relacionados con el cauce de instrucciones.

Lo esencial del enfoque superescalar es su habilidad para ejecutar instrucciones en diferentes cauces de manera independiente y concurrente. El concepto puede llevarse más lejos permitiendo que las instrucciones se ejecuten en un orden diferente al del programa.

## 6.1. Superescalar frente a supersegmentado

La supersegmentación aprovecha el hecho de que muchas etapas del cauce realizan tareas que requieren menos de medio ciclo de reloj. De este modo, doblando la velocidad de reloj interna se permite la realización de dos tareas en un ciclo de reloj externo.

Las etapas macro se dividen en el cauce segmentado en sub-etapas más pequeñas y se transmiten los datos a la mayor velocidad del ciclo de reloj.

El enfoque supersegmentado aumenta el grado de paralelismo e incrementa la aceleración percibida.

El enfoque superescalar permite llevar a cabo más de una instrucción de manera simultánea. Conlleva la duplicación de algunas o todas las partes de la CPU/ALU. Debe ser capaz de captar múltiples instrucciones al mismo tiempo. Ejecutar sumas y multiplicaciones simultáneamente. Ejecutar carga/almacenamiento, mientras se lleva a cabo una operación en ALU. El grado de paralelismo y, por tanto, la aceleración de la máquina aumenta, ya que se ejecutan más instrucciones en paralelo.

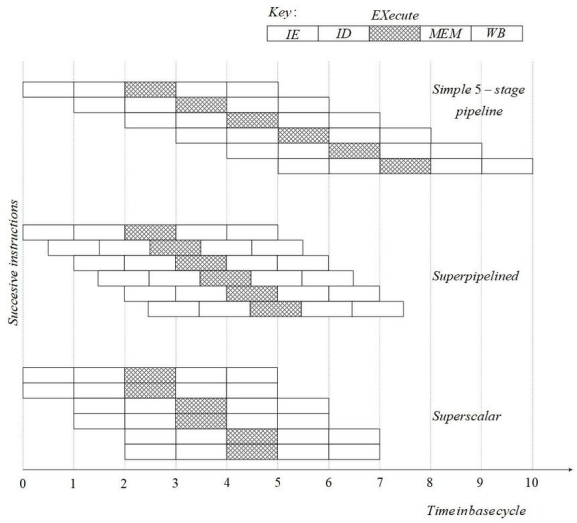


Figura 3: Superescalar contra Supersegmentado.

### Limitaciones

La aproximación superescalar depende de la habilidad para ejecutar múltiples instrucciones en paralelo. La expresión **paralelismo en las instrucciones** se refiere al grado en el que, en promedio, las instrucciones de un programa se pueden ejecutar en paralelo. Para maximizar el paralelismo en las instrucciones, se puede usar una combinación de optimizaciones realizadas por el compilador y de técnicas de hardware. Las principales limitaciones son:

- **Dependencia de datos verdadera:** es cuando una instrucción necesita el dato producido por la primera instrucción. Si no hay dependencias, se puede captar y ejecutar dos instruccio-

nes en paralelo. En caso de que exista esta dependencia de datos entre la primera y la segunda instrucción, se retrasa la segunda instrucción tantos ciclos de reloj como sea necesario para eliminar la dependencia.

- **Dependencia relativa al procedimiento:** la presencia de saltos en una secuencia de instrucciones complica el funcionamiento del cauce. Las instrucciones que siguen a una bifurcación tienen una dependencia relativa al procedimiento en ese bifurcación y no pueden ejecutarse hasta que se ejecute el salto.
- **Conflicto en los recursos:** Un conflicto en un recurso es una pugna de dos o más instrucciones por el mismo recurso al mismo tiempo. Desde el punto de vista del cauce segmentado, un conflicto en los recursos presenta el mismo comportamiento que una dependencia de datos. No obstante, hay algunas diferencias. Los conflictos en los recursos pueden superarse duplicando estos. Además, cuando una operación tarda mucho tiempo en finalizar, los conflictos en los recursos se pueden minimizar segmentando la unidad funcional apropiada.
- **Dependencia de salida.**
- **Antidependencia:** la restricción es similar a la de la dependencia verdadera pero a la inversa. En lugar de que la primera instrucción produzca un valor que usa la segunda instrucción, la segunda instrucción destruye un valor que utiliza la primera instrucción.

## 6.2. Cuestiones relacionadas con el diseño

### Paralelismo en las instrucciones y paralelismo de la máquina

El **paralelismo en las instrucciones** existe cuando las instrucciones de una secuencia son independientes y por tanto pueden ejecutarse en paralelo solapándose.

El paralelismo en las instrucciones depende de la frecuencia de dependencias de datos verdaderas y dependencias relativas al procedimiento que haya en el código. Estos factores dependen a su vez de la arquitectura del repertorio de instrucciones y de la aplicación.

El **paralelismo de la máquina** es una medida de la capacidad del procesador para sacar partido al paralelismo en las instrucciones. El paralelismo de maquina depende del número de instrucciones que pueden captarse y ejecutarse al mismo tiempo y de la velocidad y sofisticación de los mecanismos que usa el procesador para localizar instrucciones independientes.

### Políticas de emisión de instrucciones

Se utiliza el término **emisión de instrucciones** para referirse al proceso de iniciar la ejecución de instrucciones en las unidades funcionales del procesador y el término **política de emisión de instrucciones** para referirse al protocolo usado para emitir instrucciones.

El procesador intenta localizar instrucciones más allá del punto de ejecución en curso que puedan introducirse en el cauce y ejecutarse. Hay tres ordenaciones importantes:

- El orden en que se captan las instrucciones.
- El orden en que se ejecutan las instrucciones.
- El orden en que las instrucciones actualizan los contenidos de los registros y de las posiciones de memoria.

Cuanto más sofisticado sea el procesador, menos limitado estará por la estrecha relación entre estas ordenaciones. Para optimizar la utilización de los diversos elementos del cauce, el procesador tendrá que alterar uno o más de estos órdenes con respecto al orden que se encontraría en una ejecución secuencial estricta. La única restricción que tiene el procesador es que el resultado debe ser correcto. De este modo, el procesador tiene que acomodar las diversas dependencias y conflictos discutidos antes.

Podemos agrupar las políticas de emisión de instrucciones de los procesadores superescalares en las siguientes categorías:

- **Emisión en orden y finalización en orden:** es la política de emisión más sencilla. Consiste en emitir instrucciones en el orden exacto en que lo haría una ejecución secuencial y escribir los resultados en ese mismo orden. Ni siquiera los cauces escalares siguen una política tan ingenua. No obstante, es útil considerar esta política como base con la cual comparar otras aproximaciones más sofisticadas.

- **Emisión en orden y finalización desordenada:** la finalización desordenada se usa en los procesadores RISC escalares para mejorar la velocidad de las instrucciones que necesitan ciclos. Con finalización desordenada, puede haber cualquier número de instrucciones en la etapa de ejecución en un momento dado, hasta alcanzar el máximo grado de paralelismo e la máquina ocupando todas la unidades funcionales. La emisión de instrucciones se para cuando hay una pugna por un recurso, una dependencia de datos o una dependencia relativa al procedimiento. Surge la **dependencia de salida**.

- **Emisión desordenada y finalización desordenada:** para permitir la emisión desordenada, es necesario desacoplar las etapas del cauce de decodificación y ejecución. Esto se hace mediante un buffer llamado **ventana de instrucciones**. Con esta organización, cuando un procesador termina de decodificar una instrucción, la coloca en la ventada de instrucciones. Mientras el buffer no se llene, el procesador puede continuar captando y decodificando nuevas instrucciones. Cuando una unidad funcional de la etapa de ejecución queda disponible, se puede emitir una instrucción desde la ventana de instrucciones a la etapa de ejecución. La única restricción es que el programa funcione correctamente. Por esta política surge el termino **antidependencia**.

### Renombramiento de registros

Cuando varias instrucciones compiten por el uso de los mismos registros, generando restricciones en el cauce que reducen las prestaciones. Un método para hacer frente a este tipo de conflictos de almacenamiento se basa en una solución tradicional para los conflictos en los recursos: la duplicación de recursos.

Con el **renombramiento de registros**, el hardware del procesador asigna dinámicamente los registros, que están asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. Cuando se crea un nuevo valor de registro, se asigna un nuevo registro para ese valor. Las instrucciones posteriores que accedan a ese valor como operando fuente en ese registro tienen que sufrir un proceso de renombramiento: las referencias a registros de esas instrucciones han de revisarse para referenciar el registro que contiene el valor que se necesita.

### Ejecución superescalar

El proceso de captación de instrucciones, que incluye la predicción de saltos, se usa para formar un flujo dinámico de instrucciones. Se examinan las dependencias de este flujo, y el procesador puede eliminar las que sean artificiales. En esta ventana, las instrucciones ya no forman un flujo secuencial sino que están estructuradas de acuerdo a sus dependencias de datos verdaderas. El procesador lleva a cabo la etapa de ejecución de cada instrucción en un orden determinado por las dependencias de datos verdaderas y la disponibilidad de los recursos hardware. Por último, las instrucciones se vuelven a poner conceptualmente en un orden secuencial y sus resultados se almacenan.

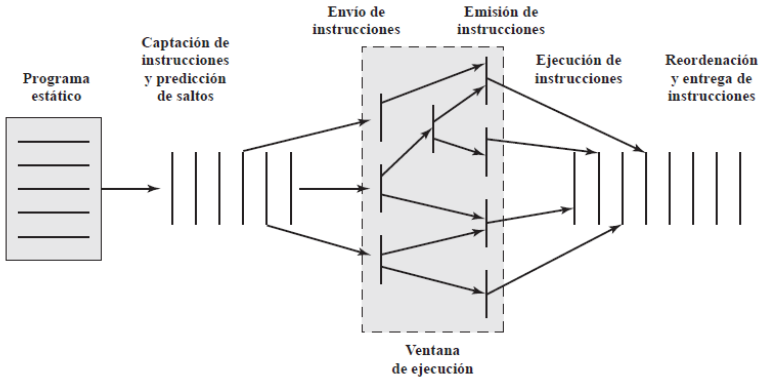


Figura 4: Representación conceptual del procesamiento superescalar

**Implementación superescalar**

- Estrategias de captación simultánea de múltiples instrucciones.
- Lógica para determinar dependencias verdaderas entre valores de registros y mecanismos para comunicar esos valores.
- Mecanismos para iniciar o emitir múltiples instrucciones en paralelo.
- Recursos para la ejecución en paralelo de múltiples instrucciones.
- Mecanismos para entregar el estado del procesador en un orden correcto.

**Consideraciones destacables en el procesamiento superescalar**

En el momento en que se produce una excepción hay varias instrucciones en ejecución. Si $I_1$ produce una excepción $\rightarrow$ ¿ha podido terminar $I_2$? $\rightarrow$ Estado inconsistente (excepciones imprecisas).

El comportamiento debería ser idéntico al que tendría la misma computadora no segmentada. Para garantizar un estado consistente:

- Instrucciones anteriores terminan correctamente.
- La que origina la excepción y siguientes se abortan.
- Tras la rutina de tratamiento se comienza por la que originó la excepción.

## 6.3. Pentium 4

Aunque el concepto de diseño superescalar se asocia generalmente con la arquitectura RISC, se pueden aplicar los mismos principios superescalares a una máquina CISC. El ejemplo más notable de ello tal vez sea el Pentium.

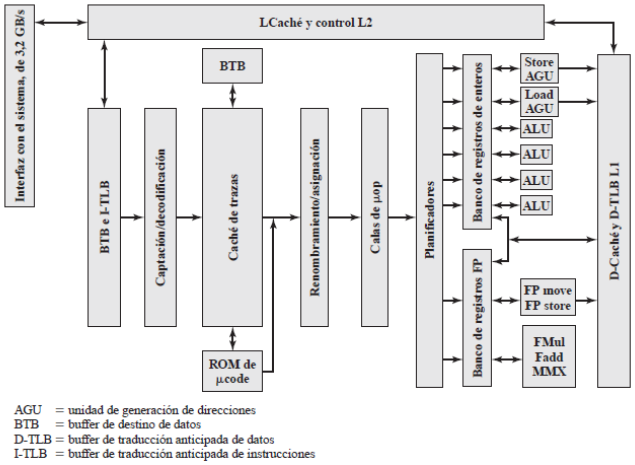


Figura 5: Diagrama de bloques del Pentium 4

El funcionamiento del Pentium 4 se puede resumir como sigue:

- El procesador capta instrucciones de memoria en el orden en que aparecen en el programa estático.
- Cada instrucción se traduce en una o más instrucciones RISC de tamaño fijo conocidas como **microoperaciones** o **micro-ops**.
- El procesador ejecuta las micro-ops en una organización de cauce superescalar, de modo se pueden ejecutar desordenadas.
- El procesador entrega los resultados de la ejecución de cada micro-op al conjunto de registros, en el orden del flujo del programa original.

En realidad, la arquitectura del Pentium 4 consta de una envoltura CISC con un núcleo RISC interno. Las micro-ops RISC internas pasan a través de un cauce con al menos veinte etapas; en algunos casos, las micro-ops necesitan múltiples etapas de ejecución, lo que se traduce en un cauce aún más largo.

## 6.4. Arquitectura IA-64

Intel se asocia con HP para desarrollar una nueva arquitectura de 64 bits, llamada IA-64. La arquitectura IA-64 no es una ampliación a 64 bits de la arquitectura de 32 bits x86 de Intel. La IA-64 aprovecha la enorme circuitería y la gran velocidad disponibles en las más recientes generaciones de microchips gracias a la utilización sistemática del paralelismo.

### 6.4.1. Motivación

Los conceptos básicos en los que se fundamenta la arquitectura IA-64 son los siguientes:

- Paralelismo en las instrucciones que queda explícito en las instrucciones máquina, en lugar de depender del procesador en tiempo de ejecución.
- Palabras de instrucción largas o muy largas (LIW[1]/VLIW[2]).
- Ejecución de saltos basada en predicados (concepto diferente al de predicción de saltos).

[1] long instruction word
[2] very long instruction word

- Carga especulativa.

Intel y HP hacen referencia a esta combinación de conceptos con el nombre de computación con instrucciones explícitamente paralelas (EPIC). La arquitectura IA-64 consiste en un repertorio de instrucciones real destinado a ser implementado usando la tecnología EPIC. El primer producto de Intel basado en IA-64 es conocido con el nombre de **Itanium**.

| **Superescalar** | **IA-64** |
|---|---|
| Instrucciones de tipo RISC, una por palabra | Instrucciones de tipo RISC puestas en grupos de tres |
| Multiples unidades de ejecución en paralelo | Múltiples unidades de ejecución en paralelo |
| Reordena y optimiza el flujo de instrucciones en tiempo de ejecución | Reordena y optimiza el flujo de instrucciones en tiempo de compilación. |
| Predicción de saltos con ejecución especulativa de un camino. | Ejecución especulativa de los dos caminos de una bifurcación. |
| Carga de datos desde memoria solo cuando es necesario, e intenta encontrar los datos primero en las cachés. | Carga datos especulativamente antes de que se necesiten, y sigue intentando encontrar los datos primero en las cachés. |

Intel y HP han propuesto un planteamiento de diseño global que permite la utilización eficaz de un procesador con muchas unidades de ejecución en paralelos. El corazón de este nuevo enfoque es el concepto de paralelismo explícito. En esta aproximación, el compilador planifica estáticamente las instrucciones en tiempo de compilación, en lugar de que lo haga dinámicamente el procesador en tiempo de ejecución. El compilador determina qué instrucciones pueden ejecutarse en paralelo e incluye esta información en la instrucción máquina. El procesador usa esa información para llevar a cabo la ejecución paralela. Una ventaja de esta aproximación es que el procesador EPIC no requiere tanta circuitería compleja como un procesador superescalar capaz de ejecutar instrucciones sin orden. Además, mientras que el procesador tiene que determinar la posibilidad de una potencial ejecución en paralelo en cuestión de nanosegundos, el compilador dispone de un plazo varios órdenes de magnitud mayor para examinar el código con detenimiento y estudiar el programa globalmente.

### 6.4.2. Organización general

Como cualquier arquitectura de procesador, la IA-64 puede implementarse con diversas organizaciones. Sus características más importantes son:

- **Gran numero de registros:** el formato de instrucción de la arquitectura IA-64 supone el empleo de 256 registros, 128 registros de 64 bits de uso general para uso con enteros, con datos lógicos y para propósito general, y 128 registros de 82 bits para su uso con coma flotante y gráficos. También hay 64 bits de predicado de un bit, usados para la ejecución con predicados.
- **Multiples unidades de ejecución:** una máquina superescalar típica puede tener cuatro cauces paralelos, empleando cuatro unidades de ejecución en paralelo tanto para la parte de enteros del procesador como para la parte de la coma flotantes. Se espera que la IA-64 se implemente en sistemas con ocho o más unidades paralelas.

En la arquitectura IA-64 se definen cuatro tipos de unidades de ejecución:

- **I:** para instrucciones aritméticas con enteros, de desplazamiento y suma, lógicas de comparación y multimedia con enteros.
- **M:** cargas y almacenamientos entre registros y memoria más algunas operaciones de la ALU con enteros.

- **B:** instrucciones de salto.
- **F:** instrucciones de coma flotante.

### Formato de instrucción

La arquitectura IA-64 define un **paquete** de 128 bits que contiene tres instrucciones, llamadas **silabas**, y un campo plantilla. El procesador puede captar uno o más paquetes de instrucciones a la vez; y cada captación indica qué instrucciones se pueden ejecutar en paralelo. La interpretación de este campo no se limita a un único paquete. Por el contrario, el procesador puede examinar varios paquetes para determinar qué instrucciones pueden ejecutarse en paralelo.

Las instrucciones agrupadas no tienen que estar en el orden original del programa. Además, debido a la flexibilidad del campo plantilla, el compilador puede mezclar instrucciones dependientes e independientes en el mismo paquete.

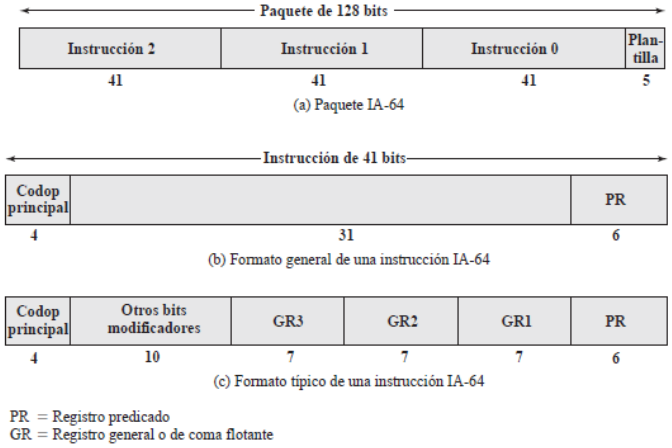


Figura 6: Formato de instrucción de la arquitectura IA-64

El valor de plantilla tiene dos propósitos:

- El campo especifica la correspondencia de las instrucciones con los tipos de unidades de ejecución. No son posibles todas las correspondencias de instrucciones con unidades.
- El campó indica la presencia de posibles **paradas**. Una parada indica al hardware que una o más instrucciones anteriores a la parada pueden tener ciertos tipos de dependencias de recursos con una o más instrucciones posteriores a la parada.

### Saltos predicados

Los saltos predicados son una técnica de compilación. El compilador elimina saltos del programa usando ejecución condicional, es necesario soporte de hardware. Se deja que ambas ramas de un salto condicional se ejecuten en paralelo.

Procedimiento para saltos predicados:

1. El procesador encuentra una instrucción de salto predicado y determina la dirección de destino de las dos posibles ramas del salto (la rama verdadera y la rama falsa).
2. Mientras se resuelve la condición del salto, el procesador especulativamente cumple las instrucciones de ambas ramas en paralelo, manteniendo un estado interno de ejecución especulativa.

3. Se realiza la evaluación de la condición del salto. Si la condición se determina como verdadera, el procesador descarta los resultados de la rama falsa y continúa la ejecución en la rama verdadera. Si la condición es falsa, se descartan los resultados de la rama verdadera y se continúa en la rama falsa.

4. Los resultados correctos de la rama seleccionada se reescriben en los registros y se continúa la ejecución secuencial a partir de la instrucción de destino correspondiente.

# 7. Procesamiento paralelo

## 7.1. Organizaciones con varios procesadores

### Tipos de sistemas paralelos

La taxonomía introducida por Flynn es todavía la forma más común de clasificar a los sistemas según sus capacidades de procesamiento paralelo. Flynn propuso las siguientes categorías o clases de computadoras:

- **Single Instruction Single Data (SISD):** un único procesador interpreta una única secuencia de instrucciones para operar con los datos almacenados en una única memoria. Los computadores monoprocesador caen dentro de esta categoría.
- **Single Instruction Multiple Data (SIMD):** una única máquina controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada elemento de proceso tiene una memoria asociada, de forma que cada instrucción es ejecutada por cada procesador con un conjunto de datos diferentes. Los procesadores vectoriales y los matriciales pertenecen a esta categoría.
- **Multiple Instruction Single Data (MISD):** se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Esta estructura nunca ha sido implementada.
- **Multiple Instruction Multiple Data (MIMD):** un conjunto de procesadores ejecuta simultáneamente secuencias de instrucciones diferentes con conjuntos de datos diferentes. Los SMP, los *clusters* y los sistemas NUMA son ejemplos de esta categoría.

En la organización MIMD los procesadores son de uso general; cada uno es capaz de procesar todas las instrucciones necesarias para realizar las transformaciones apropiadas de los datos. Los computadores MIMD se pueden subdividir además según la forma que tienen los procesadores para comunicarse. Si los procesadores comparten una memoria común, entonces cada procesador accede a los programas y datos almacenados en la memoria compartida, y los procesadores se comunican unos con otros a través de esa memoria. La forma más común de este tipo de sistema se conoce como **multiprocesador simétrico (SMP)**. En un SMP, varios procesadores comparten una única memoria mediante un bus compartido u otro tipo de mecanismo de interconexión. Una característica distintiva de estos sistemas es que el tiempo de acceso a memoria principal es aproximadamente el mismo para cualquier procesador. Un desarrollo más reciente es la organización con **acceso no uniforme a memoria (NUMA)**. Como el propio nombre indica, el tiempo de acceso a zonas de memoria diferentes puede diferir en un computador NUMA.

Un conjunto de computadores monoprocesador independientes o de SMP pueden interconectarse para formar un *cluster*. La comunicación entre los computadores se realiza mediante conexiones fijas o mediante algún tipo de red.

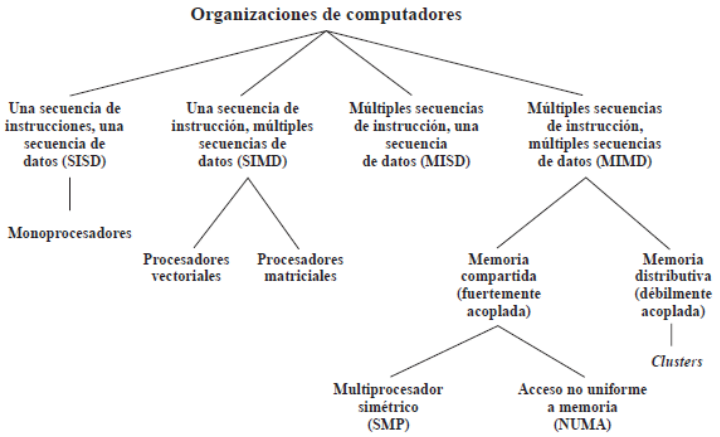


Figura 7: Taxonomía de las arquitecturas paralelas.

## 7.2. Multiprocesadores simétricos

### 7.2.1. SMP

El término SMP se refiere as la arquitectura hardware del computador y también al comportamiento del sistema operativo que utiliza dicha arquitectura. Un SMP puede definirse como un computador con las siguientes características:

- Hay dos o más procesadores similares de capacidades comparables.
- Estos procesadores comparten la memoria principal y las E/S y están interconectados mediante un bus u otro tipo de sistema de interconexión, de forma que el tiempo de acceso a memoria es aproximadamente el mismo para todos los procesadores.
- Todos los procesadores comparten los dispositivos de E/S, bien a través de los mismos canales o mediante canales distintos que proporcionan caminos de acceso al mismo dispositivo.
- Todos los procesadores pueden desempeñar las mismas funciones.
- El sistema está controlado por un sistema operativo integrado que proporciona la interacción entre los procesadores y sus programas a los niveles de trabajo, tarea, fichero y datos.

El sistema operativo de un SMP planifica la distribución de procesos o hilos (*threads*) entre todos los procesadores. Un SMP tiene las siguientes ventajas potenciales con respecto a una arquitectura monoprocesador:

- **Prestaciones:** si el trabajo a realizar por un computador puede organizarse de forma que partes del mismo se puedan realizar en paralelo, entonces un sistema con varios procesadores proporcionará mejores prestaciones que uno con un solo procesador del mismo tipo.
- **Disponibilidad:** en un multiprocesador simétrico, debido a que todos los procesadores pueden realizar las mismas funciones, un fallo en un procesador no hará que el computador se detenga.
- **Crecimiento incremental:** se pueden aumentar las prestaciones del sistema añadiendo más procesadores.
- **Escalado:** los fabricantes pueden ofrecer una gama de productos con precios y prestaciones diferentes en función del número de procesadores que configuran el sistema.

Una característica atractiva de un SMP es que la existencia de varios procesadores es transparente al usuario. El sistema operativo se encarga de la sincronización entre los procesadores y de la planificación de los hilos o de los procesos, asignándolos a los distintos procesadores.

**Bus de tiempo compartido**

La organización más común en los computadores personales, estaciones de trabajo y servidores es el bus de tiempo compartido. El bus de tiempo compartido es el mecanismo más simple para construir un sistema multiprocesador. La estructura y las interfaces son básicamente las mismas que las de un sistema de un único procesador que utilice un bus para la interconexión. El bus consta de líneas de control, dirección y datos. Para facilitar las transferencias de DMA con los procesadores de E/S, se proporcionan los elementos para el:

- **Direccionamiento:** debe ser posible distinguir los módulos del bus para determinar la fuente y el destino de los datos.
- **Arbitraje:** cualquier módulo de E/S puede funcionar temporalmente como un *maestro*. Se proporciona un mecanismo para arbitrar entre las peticiones que compiten por el control del bus, utilizando algún tiempo de esquema de prioridad.
- **Tiempo compartido:** cuando un módulo está controlando el bus, los otros módulos no tienen acceso al mismo y deben, si es necesario, suspender su operación hasta que dispongan del bus.

La organización del bus presenta varias características atractivas:

- **Simplicidad:** es la aproximación más simple para organizar el multiprocesador. La interfaz física y la lógica de cada procesador para el direccionamiento, el arbitraje y para compartir el tiempo del bus es el mismo que de un sistema con un solo procesador.
- **Flexibilidad:** es generalmente sencillo de expandir el sistema conectando más procesadores al bus.
- **Fiabilidad:** el bus es esencialmente un medio pasivo, y el fallo de cualquiera de los dispositivos conectados no provocaría el fallo de todo el sistema.

La principal desventaja de la organización de bus son las prestaciones. Todas las referencias a memoria pasan por el bus. En consecuencia, la velocidad del sistema está limitada por el tiempo de ciclo. Para mejorar las prestaciones, es deseable equipar a cada procesador con una memoria caché.

El uso de cachés introduce algunas consideraciones de diseño nuevas. Puesto que cada caché local contiene una imagen de una parte de la memoria, si se altera una palabra en una caché, es concebible que eso podría invalidar una palabra en otra caché. Para evitarlo, se debe avisar a los otros procesadores de que se ha producido una actualización de memoria. Este problema se conoce como *coherencia de caché*.

### 7.2.2. Clusters

Los **clusters** constituyen la alternativa a los multiprocesadores simétricos (SMP) para disponer de prestaciones y disponibilidad elevadas, y son particularmente atractivos en aplicaciones propias de un servidor. Se puede definir un **cluster** como un grupo de computadores completos interconectados que trabajan conjuntamente como un único recurso de cómputo, creándose la ilusión de que se trata de una sola maquina.

Los objetivos de los clusters son:

- **Escalabilidad absoluta:** es posible configurar clusters grande que incluso superan las prestaciones de los computadores independientes más potentes. Un cluster puede tener decenas de máquinas, cada una de las cuales puede ser un multiprocesador.
- **Escalabilidad incremental:** un cluster se configura de forma que sea posible añadir nuevos sistemas al cluster en ampliaciones sucesivas.

- **Alta disponibilidad:** puesto que cada nodo del cluster es un computador autónomo, el fallo de uno de los nodo no significa la perdida del servicio.
- **Mejor relación precio-prestaciones:** al utilizar elementos estandarizados, es posible configurar un cluster con mayor o igual potencia de cómputo que un ordenador independiente mayor, a mucho menos costo.

### Arquitectura de los clusters

Los computadores se conectan a través de una red de área loca de alta velocidad o mediante un conmutador. Cada computador puede trabajar de forma independiente. Además, en cada computador se instala una capa software intermedia que permite el funcionamiento de todos los computadores como un único cluster. El *middleware del cluster* proporciona al usuario una imagen unificada conocida como **imagen de sistema único**. El *middleware* también es responsable de proporcionar alta disponibilidad, distribuyendo la carga y respondiendo a los fallos de los componentes. Se pueden enumerar los siguientes servicios y funciones deseables en la capa de middleware:

- Punto de entrada único.
- Jerarquía de ficheros única.
- Punto de control único.
- Red virtual única.
- Espacio de memoria único.
- Sistema de gestión de trabajos único.
- Interfaz de usuario única.
- Espacio de E/S único.
- Espacio de procesos único.
- Puntos de chequeo.
- Migración de procesos.

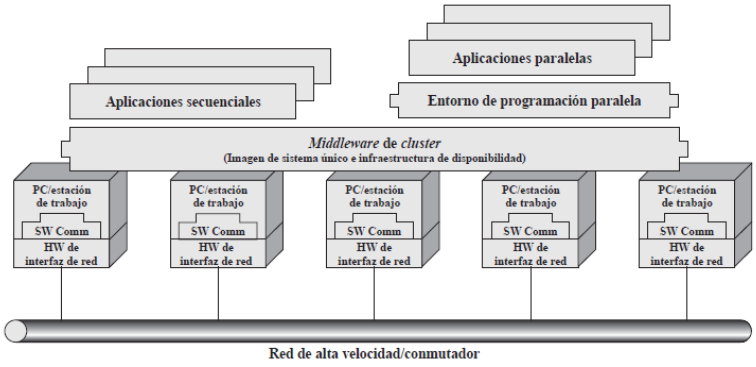


Figura 8: Arquitectura de computador de cluster

**Clusters frente a sistemas SMP**

La principal ventaja de un SMP es que resulta más fácil gestionar y configurar que un cluster. El SMP está mucho más cerca del modelo de computador de un solo procesador para el que están disponibles casi todas las aplicaciones. El principal cambio que se necesita para pasar de un computador monoprocesador a un SMP se refiere al funcionamiento del planificador. Otra ventaja es que los SMP ocupan menos espacio físico y y consume menos energía.

Con el tiempo los clusters dominarán el mercado de servidores de altas prestaciones. Los cluster son superiores a los SMP en términos de escalabilidad absoluta e incremental, y además también son superiores en términos de disponibilidad.

## 7.3. Coherencia de caché

En los sistemas multiprocesador contemporáneos es común disponer de uno o dos niveles de caché asociados a cada procesador. Esta organización es esencial para conseguir unas prestaciones razonables. No obstante, origina un problema conocido como **coherencia de caché**. La esencia del problema es esta: pueden existir varias copias del mismo dato simultáneamente en cachés diferentes, y si los procesadores actualizan sus copias, puede producirse una visión inconsistente de la memoria.

El objetivo de un protocolo de coherencia de caché, es situar las variables locales utilizadas recientemente en la caché apropiada y mantenerlas allí para las distintas escrituras y lecturas, al mismo tiempo que se mantiene la consistencia de las variables compartidas que pudieran encontrarse en varias cachés al mismo tiempo.

### 7.3.1. Protocolos de directorio

Los protocolos de directorio son utilizados para recopilar y mantener información sobre la ubicación de las copias de líneas en un sistema. Hay un controlador centralizado y un directorio almacenado en la memoria principal. El controlador centralizado verifica y emite instrucciones para transferir datos entre la memoria y las cachés. También se encarga de mantener actualizada la información de estado. Antes de que un procesador pueda escribir en una copia local de una línea, debe solicitar acceso exclusivo al controlador. Este controlador envía un mensaje a los procesadores con copias de la línea, obligándolos a invalidar sus copias. Después de recibir confirmación de los procesadores, se concede acceso exclusivo al procesador solicitante. Si otro procesador intenta leer una línea en acceso exclusivo, se envía una notificación al controlador, quien ordena al procesador poseedor de la línea volver a escribirla en memoria principal. Aunque los esquemas de directorio tienen desventajas como un cuello de botella central y un costo de comunicación elevado, son efectivos en sistemas de gran escala con múltiples buses o complejas interconexiones.

### 7.3.2. Protocolos de sondeo

Los protocolos de sondeo distribuyen la responsabilidad de mantener la coherencia de caché entre los controladores de caché en un sistema multiprocesador. Estos protocolos se adaptan bien a sistemas basados en un bus compartido, ya que el bus proporciona una forma sencilla de difusión y sondeo.

Existen dos enfoques principales en los protocolos de sondeo: *invalidar-si-escritura* y *actualizar-si-escritura* o *difundir-escritura*. En el enfoque de *invalidar-si-escritura*, cuando una caché realiza una escritura en una línea compartida, envía una notificación que invalida la línea en las otras cachés, asegurando que la línea sea exclusiva para la caché que realizó la escritura. En el enfoque de *actualizar-si-escritura*, cuando una caché desea escribir en una línea compartida, la palabra actualizada se distribuye a las demás cachés que contienen esa línea, permitiendo que todas las cachés la actualicen.

No hay un enfoque que sea mejor en todas las situaciones, ya que las prestaciones dependen del número de cachés locales y del patrón de escrituras y lecturas en la memoria. Es importante tener en cuenta que aunque los protocolos de sondeo permiten mantener la coherencia de caché,

también pueden aumentar el tráfico en el bus compartido, lo cual puede afectar los beneficios de las cachés locales.

En resumen, los protocolos de sondeo son utilizados para mantener la coherencia de caché en sistemas multiprocesador, distribuyendo la responsabilidad entre los controladores de caché. Hay dos enfoques principales: *invalidar-si-escritura* y *actualizar-si-escritura*. La elección del enfoque depende del número de cachés y del patrón de acceso a memoria. Es importante considerar el impacto en el tráfico del bus compartido al implementar estos protocolos.

## 7.4. Acceso no uniforme a memoria

- **Uniform Memory Access (UMA):** todos los procesadores pueden acceder a toda la memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso de un procesador a cualquier región de la memoria es el mismo.
- **Nonuniform Memory Access (NUMA):** todos los procesadores tiene acceso a todas las partes de la memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso a memoria de un procesador depende de la región a la que se acceda.
- **Cache-Coherent NUMA (CC-NUMA):** un computador NUMA en el que la coherencia de caché se mantiene en todas las cachés de los distintos procesadores.

### Motivación

En un SMP existe un límite práctico en el número de procesadores que pueden utilizarse. Un esquema de caché eficaz reduce el tráfico en el bus entre los procesadores y la memoria principal. La degradación de las prestaciones parece que limita el número de procesadores en una configuración SMP entre 16 y 64 procesadores.

Precisamente el límite de procesadores en un SMP es uno de los motivos para el desarrollo de los clusters. Sin embargo, en un cluster cada nodo tiene su propia memoria principal privada y las aplicaciones no ven la memoria global. La coherencia se mantiene mediante software en lugar de mediante hardware.

El objetivo de un computador NUMA es mantener una memoria transparente desde cualquier parte del sistema, al tiempo que se permiten varios nodos de multiprocesador, cada uno con su propio bus u otro sistema de interconexión interna.

### Organización

Cada nodo de un sistema CC-NUMA incluye cierta cantidad de memoria principal. Sin embargo, desde el punto de vista de los procesadores, existe un único espacio de memoria direccionable en el que a cada posición se asocia una única dirección válida para todo el sistema. Cuando un procesador inicia un acceso a memoria, si la posición solicitada no se encuentra en la caché del procesador, entonces la caché L2 inicia una operación de captación. Si la línea deseada está en una posición remota de la memoria principal, la línea se capta a través del bus local. Si la línea solicitada está en una porción remota de la memoria, entonces se envía una petición automática para captar dicha línea a través de la red de interconexión, se proporciona a través del bus local a la caché que lo solicitaba en dicho bus.

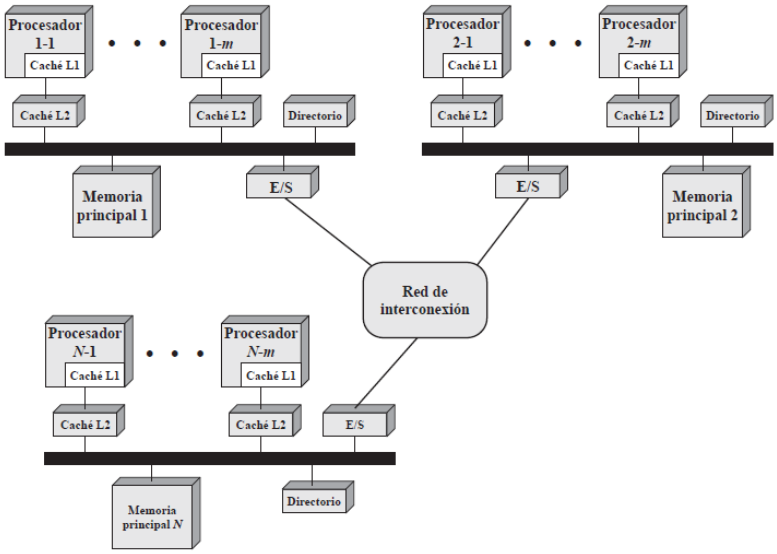


Figura 9: Organización CC-NUMA

## 7.5. Procesamiento multihebra

Una alternativa, que permite un paralelismo entre instrucciones elevado sin incrementar ni la complejidad de los circuitos ni el consumo de potencia, es el procesamiento multihebra. Básicamente, la secuencia de instrucciones se divide en secuencias más pequeñas, denominadas hebras.

### 7.5.1. Procesamiento multihebra implicito y explícito

El concepto de hebra utilizado para estudiar los procesadores multihebra puede no ser el mismo que el concepto de hebra en un sistema operativo multiprogramado. Por lo tanto, es útil definir brevemente una seria de termino:

- **Proceso:** un programa en ejecución en un computador.
  - *Propiedad de recursos:* espacio de direcciones virtuales para almacenar la imagen de un proceso.
  - *Planificación/Ejecución:* hay un camino de ejecución (traza).
- **Conmutación de proceso:** operación que cambia el proceso que se está ejecutando en el procesador por otro.
- **Hebra:** una unidad de trabajo dentro de un proceso que se puede asignar al procesador. Incluye un contexto de procesador (con el contador de programa y el puntero de pila) y su propia área de datos para la pila.
- **Conmutación de hebra:** el control del procesador pasa de una hebra a otra dentro de un mismo proceso.
- **Multihebra explícito:** se realiza una ejecución concurrente de instrucciones de diferentes hebras explícitas. Se mezclan instrucciones de diferentes hebras en cauces compartidos o se ejecuta de manera paralela en cauces paralelos. Todos los procesadores comerciales lo usan.
- **Multihebra implícito:** se realiza una ejecución concurrente de varias hebras extraídas de un único programa secuencial. Se definen estáticamente por el compilador o dinámicamente por el hardware.

En un procesador multihebra el PC es distinto para cada hebra, permitiendo la ejecución concurrente. Cada hebra es tratada por separado, predicción de saltos, renombre de registros para optimizar ejecución.