Unidad 7 - Arquitectura Multihilo y Multinucleo

1. Problema de la Arquitectura Superescalar

Las arquitectura se pueden clasificar según el número de instrucciones concurrentes (pipelines?) y flujos de datos disponibles.

- **Una Instrucción - Un Dato** (SISD): ejecución secuencial sin paralelismo de instrucción ni de datos. Aquí entrarían tanto la implementación monociclo como segmentada que hemos visto
- **Múltiples Instrucciones - Un Dato** (MISD): múltiples flujos de instrucciones sobre los mismos datos. No se suele usar salvo en contexto donde se requiere redundancia para comparar y verificar resultados.
- **Una instrucción - Múltiples Datos** (SIMD): un mismo flujo de instrucciones se ejecuta sobre múltiples datos. Suele darse en el contexto de procesamiento de gráficos (GPUs).
- **Múltiples Instrucciones - Múltiples Datos** (MIMD): varios procesadores ejecutando instrucciones diferentes sobre datos diferentes de forma simultanea. Es una arquitectura flexible donde cada procesador puede ejecutar su propio programa
- **Un Programa - Múltiples Datos** (SPMD): cada procesador ejecuta el mismo programa pero con diferentes rutas de ejecución. Es una forma de programación en paralelo y distribución de tareas.
- **Múltiples Programas - Múltiples Datos** (MPMD): múltiples procesadores trabajan de forma independiente sobre múltiples programas independientes.

Un procesador **superescalar** lee múltiples instrucciones al mismo tiempo para encontrar instrucciones que puedan ejecutarse **de manera independiente en paralelo en diferentes datapath**.
El **problema** de esta microarquitectura es que el uso de un único **hilo de ejecución** (thread) limita el paralelismo a nivel de instrucción, ya que no se pueden aprovechar todos los recursos disponibles. Esto se debe a que frecuentemente las unidades de ejecución deben esperar que otra instrucción complete una tarea para seguir su ejecución, y no se están aprovechando los recursos provistos por la arquitectura.

2. Microarquitectura Multihilo

Generalmente las computadoras ejecutan varios programas (hilos) al mismo tiempo utilizando **multiplexación** de tiempo. Las instrucciónes de los diferentes programas no tienen dependencias entre si, por lo que se pueden ejecutar de forma concurrente para mejorar el rendimiento, haciendo uso del paralelismo a nivel de hilo. De esta forma el procesador ejecuta varios hilos diferentes actuando como varios procesadores lógicos.

2.1. Asignación de Recursos

Para responder la pregunta de *cuándo y cómo asignar los recursos hilos disponibles* tenemos dos opciones:

- **Multihilo Temporal**: los recursos se asignan a cada hilo durante un tiempo (variable o fijo) que dependerá de los conflictos de recursos y dependencias de datos (**multihilo grueso**, **entrelazado** o **fino**).
- **Multihilo Simultaneo**: en cada ciclo de operación, los recursos se asignan a los hilos que puede ejecutarse, en función de los recursos disponibles y las dependencias y datos.

2.2. Multihilo Grueso

Esta es la implementación más sencilla del **multihilo temporal**, donde *se ejecuta un hilo hasta que es bloqueado por un evento de latencia prolongada y se conmuta a un hilo listo para ejecutar*.

Hay muchas variaciones del multihilo grueso, a su vez relacionadas con el algorítmo de conmutación de hilo. Este algorítmo se basa en múltiples factores, incluidos el tiempo de latencia y fallos de cache, entre otros.

La principal desventaja son las pérdidas de rendimiento en conmutaciones debidas a bloqueos cortos.

- Cuando se produce un bloqueo, el datapath debe vaciarse o congelarse.
- El nuevo hilo debe llenar el datapath antes de que se completen las instrucciones.

Por esto, es mejor para reducir la penalización si el tiempo de para es mayor al tiempo que tarda en inicializarse el datapath.

2.3. Multihilo Entrelazado

En esta implementación se *intercala la ejecución de diferentes hilos de manera rotativa, omitiendo hilos bloqueados*. El datapath contiene múltiples hilos y cambios de contexto que ocurren *entre* las etapas del datapath.
Esta implementación es muy costosa, ya que debe gestionarse simultaneamente todos los recursos y cada etapa del datapath tiene que identificar el hilo de la instrucción que está ejecutando. Los recursos compartidos además deben ser abundantes para evitar bloqueos y errores (cache, TLB, etc.).

2.4. Multihilo Simultaneo

Esta implementación se basa en que *cualquier instrucción de cualquier hilo puede ejecutarse en cualquier datapath*. Es decir, en un mismo datapath podemos estar ejecutando instrucciones de distintos hilos. Esta tecnología es la intel llama *Hyper-threading*.
Para lograr esto, se debe agregar al procesador un conjunto de registros más grande y la capacidad de **obtener instrucciones de varios hilos**.

- Ventajas
  - Supera las limitaciones producidas por hilos con bajo ILP.
  - Oculta los riesgos de control y otras latencias prolongadas.
- Desventajas
  - Sobrecarga de trabajo a la jerarquía de la memoria.
  - Aumento de la complejidad de la unidad de control.
  - Aumento de la cantidad de recursos (cache, Branch Prediction, TLB, etc.).

Implementaciones Multihilo

2.5. Modificaciones

Modificaciones necesarias para implementar esta microarquitectura:

- **Múltiples contadores de programas**, registros y un mecanismo para que una unidad de búsqueda seleccione un hilo en cada ciclo (política de fetch/issue).
- Una **pila de retorno para cada hilo**, para predecir las direcciones de retorno de cada subrutina.
- **Mecanismos fetch/issue** de instrucciones, **vaciado** de la cola de instrucciones y mecanismos de **captura** para cada hilo.
- Por cada entrada del buffer de destino de salto (BTB), un **identificador de hilo**, para evitar predicción de saltos fantasma.

Mejoras de desempeño:

- Registro de renombrado más grande, para admitir registros lógicos para cada hilo y etapas adicionales en el datapath.
- Mayor ancho de banda en las transacciones con la memoria principal.
- Buffer de traducción de Datos (TLB) más grande para compensar el aumento de traducciones de direcciones.
- Caché mejorada para compensar el uso compartido entre hilos y la reducción de la localidad resultante. De esto no se entiende nada porque se da en otro tema (memoria)

3. Multinucleo

Un procesador multinucleo está compuesto por varios nucleos (CPUs) independientes. Estos nucleos se integran en un solo circuito integrado y pueden estar acoplados entre sí de forma fuerte o debil, dependiendo de si comparten cache, comparten memoria o implementan métodos de comunicación basados en mensajes. Los nucleos además pueden implementar arquitecturas superescalar, procesamiento vectorial o multiproceso

Los elementos que se tienen que tener a consideración en las arquitecturas multinucleo son:

- Arquitectura de **memoria** utilizada

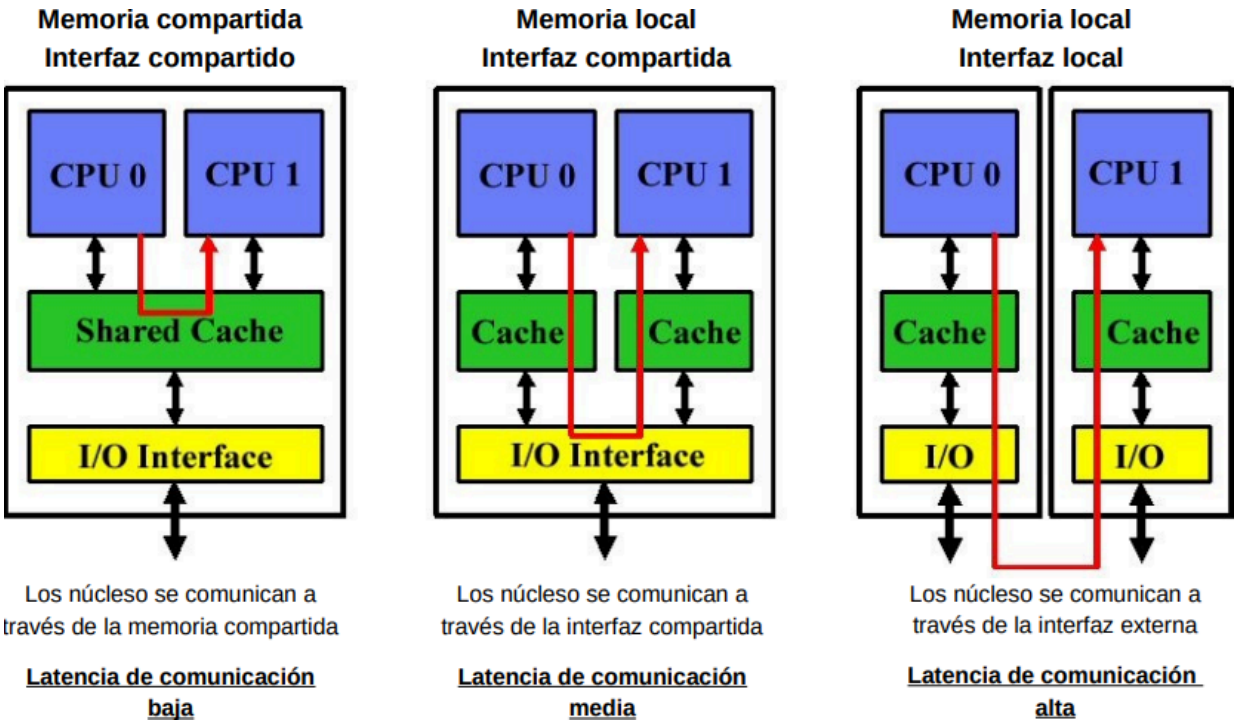
- **Tipos de elementos de procesamiento** (nucleos o unidades de procesamiento) implementados en el procesador, que determinarán las herramientas para programar en una arquitectura específica
- **Sistema de comunicación** entre los núcleos, por el que un nucleo sabrá cuándo puede acceder a los datos.

3.1. Arquitectura de Memoria

Las arquitecturas multinúcleo organizan la memoria de dos maneras:

- **Memoria Compartida**: existe una memoria única compartida por todos los núcleos.
- **Memoria Distribuida**: cada núcleo tiene su propia memoria local y su contenido no está replicado.

De todos modos, la caché de primer nivel (L1) siempre está dedicada a cada núcleo. Las cachés de nivel intermedio pueden ser compartidas entre todos los núcleos mientras que las de último nivel siempre son compartidas.



Memoria en Arquitectura Multinucleo

3.2. Elementos de Procesamiento

Según los tipos de núcleos, la microarquitectura multinucle se puede clasificar en:

- Arquitectura Homogenea: todos son iguales y de propósito general, por lo que cualquier tarea puede ser asignada a cualquier núcleo
- Arquitectura Heterogenea: cada núcleo está diseñado para tareas específicas, con un núcleo maestro que controla la operación. *Ventajas*: más eficiente en términos energéticos y con un mejor desempeño cundo el paralelismo de hilo es limitado. *Desventajas*: hay que diseñar más de un tipo de núcleo, la gestión de uso es complicada y cada núcleo tiene distintas demandas de recusos compartidos.

3.3. Sistemas de Comunicación

Las topologías de la red de interconexión pueden ser:

- Bus.
- Anillo.
- Malla.

- Bidimensional.
- Barra Transversal.