CAC - Resumen Teoría

Conceptos de Arquitectura de Computadoras, Ingeniería en Computación, UNLP

Autor: Tomás Badenes

Resumen de teoría para el examen de promoción, hecho a partir de las dispositivas como guía del contenido necesario y de la bibliografía recomendada como fuente del contenido.

Bibliografía:

- *Organización y Arquitectura de Computadores, W. Stallings, 5ta edición*
- *Diseño y Evaluación de Arquitecturas de Computadores, M. Pardo y A. Guzmán*

Clases 1-3

No entraron en primera fecha. Temas:

- Repaso, Von Neumann
- Interrupciones
- Entrada/Salida

Clase 4: Segmentación de instrucciones

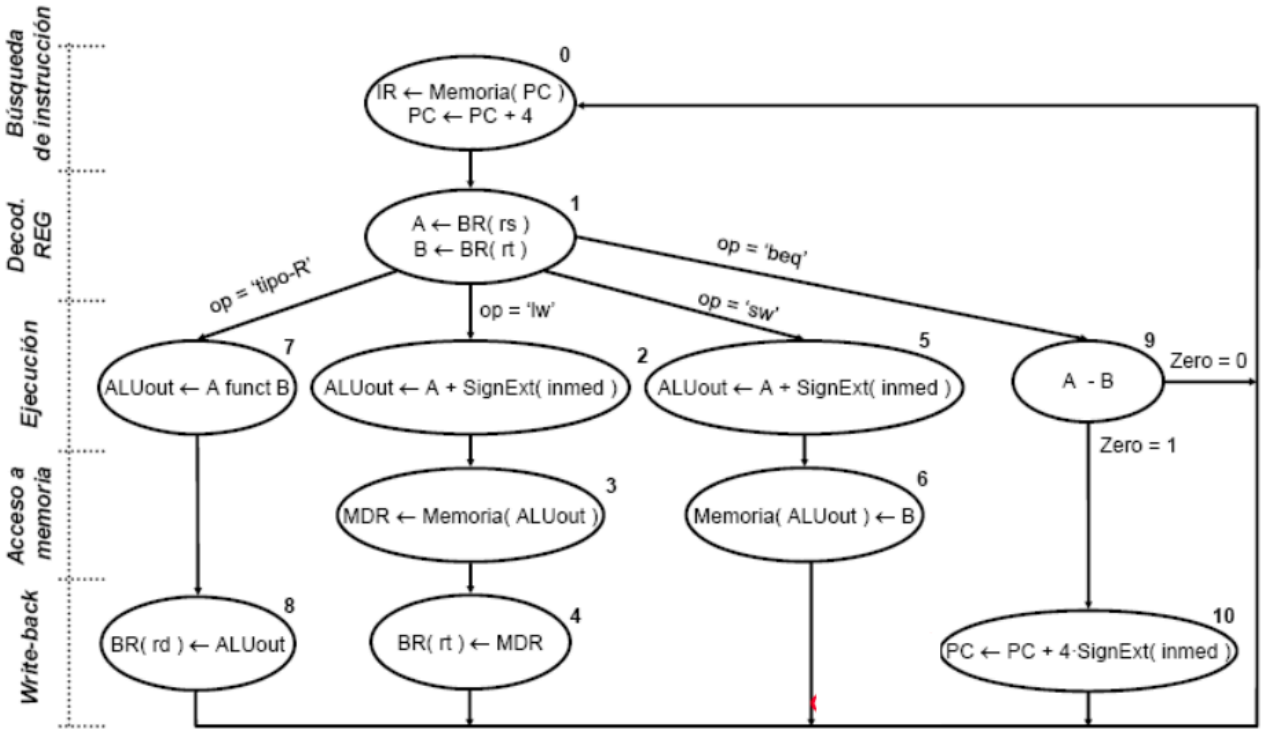
Organización y Arquitectura de Computadores, W. Stallings, Capítulo 11

Diseño y evaluación de arquitecturas de computadores, M. Pardo y A. Guzmán, Capítulo 1. 1º ed. Sección 1.5

Previo: Procesador multiciclo

- Cada instrucción puede tardar más de un ciclo en ejecutarse (se aumenta la frecuencia de reloj). Una instrucción puede ser más corta que otra, y este tiempo puede ser aprovechado ya que ya no se ejecutan las instrucciones en un tiempo fijo.
- No se usan porque la ventaja obtenida es poca comparado con el aumento en complejidad.
- Hardware:
    - Registros intermedios de etapa y para operandos de la ALU
    - Unidad de control global para enviar señales de latching a los registros nuevos y sincronizar las etapas. Máquina de estados (ocupar área, más complicada) o microprogramada (más flexible, menos área, pero más lenta porque se requiere miniciclos para varias microinstrucciones por ciclo que deben estabilizarse):

Diagrama de estados del controlador



Procesador segmentado

Forma de reorganizar el hardware para realizar más de una operación al mismo tiempo. Invisible al programador.

Consiste en descomponer el proceso de ejecución de las instrucciones en fases o etapas que permitan una ejecución simultánea.

Paralelismo a nivel de instrucción (ILP). Explota el paralelismo entre las instrucciones de un flujo secuencial.

Se aceptan instrucciones antes de que se terminen las anteriores. Idealmente se termina una instrucción en cada ciclo de reloj, por más que las instrucciones tarden varios ciclos de reloj.

- \+ Mejor throughput
- \- Mayor precio (más hardware: registros de segmentación y complejidad de control)

Se introducen los **registros de segmentación** para evitar que las etapas no hagan conflicto con otras

Criterios para segmentar en etapas

- La duración de cada etapa se adaptará al tiempo de la más lenta, así todas duran lo mismo, por lo que se dividen las etapas para que duren aproximadamente el mismo tiempo de reloj.
- En cada etapa de cauce hay gasto asociado a transferencia de datos de buffer a buffer y otras funciones, por lo que hay un límite en el que el beneficio de aumentar la cantidad de etapas disminuye.
- Cuantas más etapas más lógica de control y registros de segmentación se necesitan, lo que aumenta el costo.
- El diseño tiene dependencia del repertorio de instrucciones.
- Por las razones anteriores, en general más de 6 a 9 etapas son contraproducentes (fuente: Stallings).

Evaluación de prestaciones:

- Ecuación de prestaciones: $t_{CPU} = I \cdot CPI \cdot T$ (Instrucciones, CPI, Periodo de reloj), Un CISC puede tener I menor, pero cada I requiere mucho más tiempo al ser más compleja → mayor CPI
- El tiempo que van a tardar las instrucciones en ejecutarse individualmente es mayor por la mayor complejidad

Riesgos

- Estructurales: Dos o más instrucciones deben acceder al mismo hardware. Ejemplo: Mem y Fetch si la memoria es unificada. Más comunes en la práctica fueron coincidencias de la misma etapa (por etapas de ejecución extendidas).
- Por dependencia de datos: Condición en la que los operandos fuente o destino de una instrucción no están disponibles en el momento en que se necesitan en una etapa determinada del cauce.
    - RAW. Dependencia verdadera
    - WAW. Dependencia en salida. Solo si se deja que las instrucciones se adelanten a otras. Aparecen en procesadores multifuncionales (ruta de datos no lineal o bifurcada)
    - WAR. Antidependencia. Una instrucción modifica un valores antes que otra que lo debe leer lo lea

  WAR y WAW no hay en cauces simples ya que cada instrucción dura lo mismo y los datos pasan por las mismas etapas. Aparecen en procesadores multifuncionales (ruta de datos no lineal o bifurcada)
- Por dependencia de control (o “de instrucción”). No se sabe cual es la siguiente instrucción. Saltos condicionales.

Clase 5: Atascos y soluciones

Organización y Arquitectura de Computadores, W. Stallings, Capítulo 11

Diseño y evaluación de arquitecturas de computadores, M. Pardo y A. Guzmán, Capítulo 1. 1º ed. Sección 1.5.2

Solución de riesgos en procesadores segmentados

Riesgos estructurales

Dos o más instrucciones deben acceder al mismo hardware. Ejemplo: Mem y Fetch

- Replicar o duplicar hardware
- Turnos de acceso a registros y memorias. Escrituras en la primera mitad, lecturas en la segunda.
- Separar memoria de instrucción y memoria de datos.
- Por software: Programar mejor.

Riesgos de datos

RAW:

- Unidad de detección de riesgos y/o compilador más complejo
- Software: Instrucciones NOP o reordenación de código (compiler complejo ayuda). Esto es evitar los riesgos, no solucionarlos.
- Hardware: Forwarding (adelantamiento de operandos, cortocircuito): La mejor, pero no siempre evita todas las paradas. Consiste en utilizar los registros de segmentación de forma inteligente con una unidad de adelantamiento + algunos multiplexores estudiando todas las posibles combinaciones de riesgos RAW para saber de donde a donde enviar datos. Ej, el adelantamiento en MIPS es siempre de X/M→ D/X o

M/W→X/M. No es taan complicado de implementar si ya tenías un segmentado con registros intermedios de etapa.

WAR y RAW se suele optar por resolver con paradas y listo. Algoritmos más complejos son demasiado caros de implementar en hardware y tienen beneficio reducido para la poca frecuencia con la que se producen.

Riesgos de control

Sea condicional o incondicional siempre hay **penalización por salto.** Para los incondicionales pasa porque hay que encontrar la dirección destino, aunque no haya que evaluar una condición.

Adelantamiento del cálculo

Una fácil: Adelantar la resolución de saltos añadiendo hardware para el chequeo de condición y para cálculo de destino en la etapa D (esto también requiere de forwarding en la etapa D para cuando hay RAW con uno de los registros operando de condición de salto). Aun así siempre se tiene que parar un ciclo (en vez de 2).

Delay slot / salto retardado / relleno de ranura de retardo

Software. Se incluyen instrucciones independientes del salto que se ejecutan siempre después de la instrucción de salto. A veces se puede mezclar con la predicción para anular la ejecución de la instrucción en ranura pero es una técnica rara vez usada y complejiza el control.

Flujos múltiples

Solución burda pero que funciona, duplicar las primeras etapas cosa que se empiecen a procesar ambas ramas. Hubo dos computadoras de IBM con esta característica.

Prefetch del destino de salto

Precaptar ambos y estar listo. Sigue habiendo retardo.

Buffer de bucles

Mini caché para saltos donde se guardan las últimas n instrucciones captadas. Cuando se determina la condición, se busca en el buffer si la instrucción ya existe y así no se debe buscar en memoria. Si se combina con prefetch, el buffer puede incluir la incorrecta, que puede servir luego para la siguiente iteración.

Si es lo suficientemente grande para almacenar todas las instrucciones del bucle, estas se buscan una sola vez.

A diferencia de la caché de instrucciones, solo guarda instrucciones consecutivas y es de menor tamaño (y costo).

Predicción de datos

Solo se toma una penalización de salto si la predicción falla.

Se tiene que parar la ejecución de la etapa predicha antes de que haga cambios en registros / memoria si la condición es aún indeterminada (no llega a ser un problema en el WinMips).

- Técnicas estáticas:
    - Siempre captar la instrucción correspondiente a la predicción de siempre saltar o nunca saltar. La predicción es siempre la misma.
    - Se elige siempre saltar/no saltar dependiendo del codop / condición de salto / longitud de salto o mediante un análisis a alto nivel del programa apuntando a los comportamientos o estilos de programación más habituales.
- Técnicas dinámicas
    - Conmutador saltar/no saltar: Se le asocia a la instrucción de salto ciertos bits que indican las últimas veces que saltó, y si le erra ciertas veces cambia la predicción.

Puede guardarse en la caché junto con las instrucciones (si se borra de la caché se pierde la historia) o en una tabla asociativa:
    - BTB (Branch Target Buffer), BHT (Branch History Table): Tabla con instrucción, bits de historia e info de la instrucción destino (la dirección de la instrucción o toda la instrucción)

Clase 6: Computadoras RISC

Organización y Arquitectura de Computadores, W. Stallings, Capítulo 12. 5º edición.

CISC vs RISC

La finalidad del CISC era

- Facilitar el desarrollo de compiladores → compiladores más simples (hardware más complejo)
- Mejorar eficiencia de ejecución mediante secuencias complejas de microoperaciones
- Dar soporte a lenguajes de alto nivel (HLL) más complejos
    - Esto lleva al salto semántico. Disparidad entre lo que soporta el hardware y las sentencias de alto nivel (ej: se inventa el switch/case en HLL y no existe una implementación nivel de arquitectura que lo soporte explícitamente, lo debe hacer el compilador. Algunas computadoras lo implementan. Lo mismo con docenas de modos de direccionamiento, etc.).
- Como el software era “más caro” que el hardware complejizan el hardware (los programas son más cortos con CISC y por lo tanto más livianos y baratos de almacenar. Además, había escasez de programadores todo el tiempo).

Todo esto lleva a:

- Microcódigo complejo → Unidad de control compleja, memoria de control más grande y ejecucion más lenta
- Repertorios extensos de instrucciones complejas
    - Instrucciones que son combinación de operaciones
    - Instrucciones de muchos bits
    - Instrucciones complejas rara vez utilizadas
- Muchos modos de direccionamiento
- A veces sentencias de HLL integradas en el hardware

Si se estudia los tipos de operandos y los modos de direccionamiento más utilizados al momento de compilar el código o al momento de ejecución (estudios dinámicos) se determina que las CISC tienen ciertos problemas:

- Las asignaciones en HLL eran 15% del total, pero de las instrucciones que ejecutaba la máquina representan cerca del 40% del programa. Se lleva y trae cosas a los registros internos y a memoria todo el rato
    - Y si ponemos muchísimos registros más?
- Las instrucciones call son demasiadas y usan mucho acceso a memoria cada vez que se llaman
    - Realmente necesitamos tanta complejidad al llamar a subrutinas? Tanta profundidad, tanto pasaje de argumentos por memoria? Podemos acelerar el llamado a subrutinas si evitamos los PUSH/POP de SP y otros registros. Las subrutinas usan demasiado tiempo
    - La profundidad suele ser no demasiada en bajo nivel, aunque en el HLL parezca que sí. 4 o 5 anidamientos como mucho. 4 parámetros como mucho
- Los if/loops se ejecutan mucho en bajo nivel y con mucho acceso a memoria comparado con lo que aparenta HLL. Una instrucción HLL no se transforma a una sola de máquina.
    - Comparar y saltar es muy importante
- Operandos son principalmente variables escalares locales

    - Se deben poder optimizar las locales para evitar acceder tanto a memoria
- Conclusión: Las sentencias de alto nivel no se traducen como aparentan a bajo nivel, y luego de que el compilador hace su trabajo el approach al diseño CISC no parece no haber sido la más apropiada. Se puede dar mucho mejor soporte a los HLL con mayor número de registros, mejor diseño de cauces (predicción de bifurcaciones, etc) e instrucciones simples.
  Soluciones:
    - Por hardware, más registros. Ventanas de registros
    - Por software, optimización de compiladores para asignar registros a variables locales, requiriendo algoritmos sofisticados de análisis de los programas.
    - Pocos parámetros y por registro, menos profundidad
    - Convención de uso de registros, para no guardar siempre todos sino permitirse usar algunos temporales y no preocuparse en sobreescribirlos. También ventanas de registros, se solapan parcialmente en los llamados.
    - En general es más común usar instrucciones simples

Nota sobre ventana de registro: Nosotros no usamos. Buffer circular.

Otra cosa: Que el programa CISC sea más corto no siempre lleva a que use menos memoria principal.

Esto nos lleva al RISC:

- Conjunto reducido de instrucciones: No necesariamente pocas instrucciones, sino instrucciones ortogonales. Repertorio reducido y sencillo. Instrucciones cortas de formato fijo o pocos formatos de instrucción.
    - Programas más largos que requieren de más instrucciones. Pero, ahora el almacenamiento es barato.
    - CPI más cercano a 1.
    - Diseño cableado, sin microcódigo (más rápido). Consecuencia de una unidad de control más simple.
- Enfoque en la optimización de segmentación de instrucciones. Con un conjunto más sencillo de instrucciones es más simple optimizar la segmentación del cauce.
- Gran número de registros de uso general
    - Las operaciones registro a registro no son un problema, porque son las que más se usan
- Confía en la tecnología de compiladores para optimizar el uso de los registros y minimizar el acceso a memoria
    - El compilador debe asignar registros eficientemente
    - Más tiempo y esfuerzo de compilación

OJO: No existe una barrera diferenciadora. Muchos procesadores tienen aspectos de uno y de otro. No hay pares directamente comparables o prueba definitiva

Comparación cuantitativa:

- RISC: Más rápido.
- CISC: Programas de menor tamaño

Comparación cualitativa:

- RISC: Uso óptimo de recursos (útil en VLSI, very large scale integration)
- CISC: Soporte más simple para HLL

Banco de registros amplio vs Cache:

Banco de registros amplio	Caché
Datos escalares locales en registro	Datos escalares recientemente usados en caché

Solo variables individuales	Bloques de memoria
Direccionamiento de registro	Direccionamiento de memoria
Instrucciones más cortas	
Mayor velocidad	
Cómo ventana, menos eficiente en espacio (siempre guarda x registros aunque no necesite)	

No hay una elección clara a excepción de que la caché es notablemente más lenta que los registros y que en espacio los bancos de registros almacenan notablemente menos información. Se debe balancear.

En resumen, los estudios del comportamiento de la ejecución de programas escritos en alto nivel compilados para procesadores CISC llevaron a cambiar la orientación en el desarrollo del procesador para crear RISC.

Clase 7: Jerarquía de memoria

Organización y Arquitectura de Computadoras, W. Stallings, Capítulos 4. 5º edición.

Diseño y evaluación de arquitecturas de computadores, M. Pardo y A. Guzmán, Capítulo 2 (secciones 2.1 a 2.4) y algo del capítulo 4. 1º edición.

Conceptos

Niveles en distintos lugares físicos, fabricados con tecnologías diferentes.

La jerarquía es invisible al programador.

Objetivo: La velocidad del sistema debe ser la del nivel mas rapido al costo del más barato

A cumplir:

- **Inclusión:** Datos en nivel tienen que estar reflejados en niveles superiores (más lejos del CPU)
- **Coherencia:** Copias de la misma información en distintos niveles deben tener los mismos valores
- Correspondencia de direcciones

Principio de localidad de referencia. Las referencias tienden a formar agrupaciones de direcciones cercanas en periodos cortos de tiempo

- **Localidad temporal:** Los elementos referenciados recientemente volverán a ser referenciados en un futuro próximo → Subir esa palabra de nivel (la probabilidad de aciertos es buena)
- **Localidad espacial:** Loe elementos próximos en memoria a los últimos referenciados serán referenciados proximamente → Subir al bloque de nivel (aumenta la probabilidad de aciertos)

A medida que se desciende en la jerarquía disminuye el coste por bit pero aumenta el tiempo de acceso

El procesador accede a palabras, la unidad natural de memoria. Suele coincidir con el tamaño de los datos o el tamaño de las instrucciones.

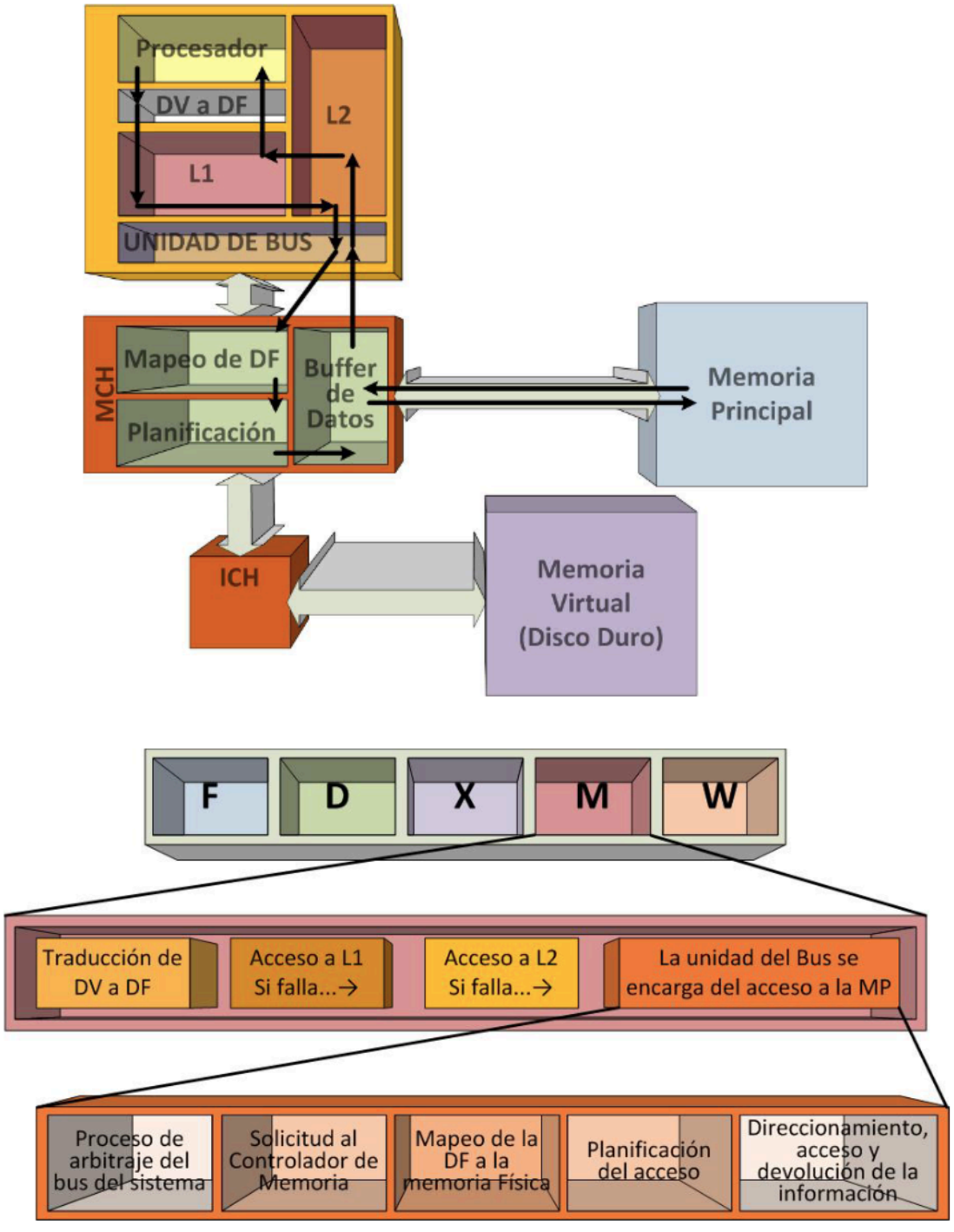
La caché almacena bloques de palabras. Es volátil.

La memoria principal está dividida en páginas o segmentos de mayor tamaño que los bloques de caché. Es volátil.

La memoria virtual está dividida en secciones más grandes que las páginas o segmentos. El sistema de direccionado virtual depende del OS y permite direccionar más memoria que la disponible físicamente utilizando el almacenamiento externo. La resolución de fallos de página (*page fault*) dependen del sistema operativo y requiere cambios de contexto del procesador. Para evitar la hiperpaginación se usan algoritmos de pre búsqueda de páginas.

Datos interesantes de hardware: La caché y la memoria principal son RAM (tiempo de acceso de lectura *y de escritura* constantes), pero una es SRAM (estática) y la otra es DRAM (dinámica). La memoria estática está hecha de flip flops (almacenamiento biestable, por eso estática), más rápida, menos densa. La DRAM es más simple, más densa y hecha con capacitores que como tienden a descargarse deben ser refrescados, por eso dinámica.

Mecanismo de acceso a memoria



1. El procesador genera la dirección virtual.
2. Se traduce la dirección virtual a una dirección física comprensible para la jerarquía de memoria

3. Se accede a la caché y se analizan las etiquetas. Comparación entre la etiqueta buscada y las etiquetas en la caché. Si está el bloque de palabras está la palabra.

Si la palabra buscada se encuentra en este nivel (acierto, hit), termina el acceso a memoria. Si la etiqueta del bloque buscada no está en la caché (fallo, miss):

1. Sea cual sea el tipo de fallo se busca la palabra en el siguiente nivel. Puede ser otro nivel de caché o la memoria principal.
2. Si se quiere buscar el bloque necesario en la memoria principal, se pasa por el controlador de memoria (puede o no estar en el chip) que mapea la dirección física a la ubicación física y también planifica el acceso (dispositivos de E/S que también quieran entrar). MMU (memory management unit), puede ser dentro del MCH (Memory Controller Hub) o en su propio chip, traduce a una dirección de los chips de DRAM o de memoria virtual en disco. El direccionado virtual depende del sistema operativo. Se debe visitar la tabla de paginados que similar al cache usa una forma directa, asociativa o mixta de organizar las páginas.
3. Cuando se realiza el acceso a la memoria principal, todo el bloque que incluye a la palabra buscada se mueve a la memoria caché para resolver el fallo, identificando cual es el bloque a sobreescribir. Al mismo tiempo se sirve la palabra buscada al CPU.

Si la palabra no se encuentra en memoria principal (en ningún momento se pudo traducir la dirección virtual a física, *page fault*):

1. El sistema operativo trae desde memoria virtual la página o segmento necesario para resolver el fallo. En general también cambia de contexto al procesador para entretenerlo.
2. Una vez la página o segmento está en memoria, se sube el bloque a caché y la palabra al procesador

Tipos de fallos

- **Iniciales:** Es la primera vez que se referencia una palabra y esta no está en la caché
- **De capacidad:** En la caché no caben todos los bloques correspondientes al programa completo y deben realizarse desplazamientos.
- **De conflicto:** Los bloques, según el algoritmo de reemplazo, tienen la misma localización y se sobreescriben unos a otros.

Los fallos de caché se gestionan mediante hardware y detienen al procesador hasta que el dato está disponible.

Optimización de acceso

Todo el mecanismo de acceso a memoria debe ocurrir en la etapa de memoria en un procesador segmentado. Se calcula el tiempo en base al tiempo de acceso a caché y si se resuelve el fallo se extiende a ejecución.

Caché no bloqueante: Permite recibir accesos durante un miss.

Evaluación de prestaciones

Métricas:

- **Latencia:** Tiempo desde que se inicia la transferencia hasta que finaliza
- **Ancho de banda:** Información por unidad de tiempo desde/hacía memoria
- **Tiempo para servir un fallo:** Tiempo desde que se determina que el dato no está en la caché, hasta que el bloque es copiado a la caché y la palabra se sirve al CPU. Depende de la latencia y ancho de banda. Surge la penalización por fallo como consecuencia del tiempo para servir un fallo.
- **Tiempo de acceso medio a memoria:** $t_{MEM} = t_{aciertoMC} + TF \cdot pF$ (tiempo de acceso medio a memoria, tiempo de acierto en memoria caché, tasa de fallo, penalización por fallo)

Para mejorar las prestaciones:

- Mejorar la tecnología (latencia, ancho de banda) para reducir el tiempo de acceso de acierto y la penalización por fallo
- Mejorar el tiempo de acceso reduciendo la tasa de fallos.

Jerarquía perfecta: No hay fallos (los datos siempre están) y el $t_{acierto}$ es despreciable (eso no es tan poco realista porque es muy bajo)

Jerarquía real: Tasa de fallo >0% (en ejemplos dados, 4%), penalización por fallo >0s (en ejemplos, 100ns), $t_{acierto}$ puede considerarse 0s.

Diseño de la caché

Organización

La memoria caché es una memoria pequeña pero de más altas prestaciones, que en el camino de transferencia de datos está entre el CPU y la memoria principal. En general está integrado en el mismo chip del CPU.

La caché se organiza en líneas o marcos. Cada línea contiene una etiqueta y un bloque de datos con K palabras. Debe elegirse el tamaño de línea. Un tamaño mayor de bloque aumenta los aciertos por localidad, pero con bloques muy grandes hay menos cantidad de bloques en caché y se sobreescriben seguido, aumentando los fallos.

Es importante el factor de tamaño de la caché, ya que una muy pequeña tendrá muchos fallos y una más grande será más compleja y por lo tanto más lenta.

Las etiquetas sirven para identificar si el dato que se busca está en la memoria caché. Usualmente es una porción de la dirección en memoria principal.

Es normal tener separada una caché para instrucciones y otra caché para datos (muchos RISC, el Pentium IV). La caché unificada tiene ventajas si se balancea la cantidad de datos y la cantidad de instrucciones y solo se debe implementar una sola, pero se tiende a tener cachés separadas, particularmente para superescalares y otros diseños con segmentación de cauce.

Política de ubicación / Función de correspondencia

Al haber menos líneas de caché que bloques en una memoria se necesita de un algoritmo para determinar dónde copiar cada bloque a la caché.

Correspondencia

- **Directa:** Cada bloque puede corresponder sólo a una línea de caché. Ej. línea = N° de bloque de memoria. ppal. % cant. líneas. Fácil de implementar usando la dirección de memoria (los bits menos significativos pueden indicar la palabra dentro del bloque (índice) y los bits más significativos el bloque).
    - Ventajas: Lectura y escritura en tiempo constante. Simple y poco costosa de implementar. Etiqueta corta
    - Desventajas: Si el programa referencia constantemente palabras pertenecientes a dos bloques diferentes con la misma correspondencia al caché, la cantidad de aciertos baja mucho.
- **Totalmente asociativa:** Cada bloque puede cargarse en cualquier línea. La dirección de memoria (sin los últimos bits que indican la línea dentro del bloque) es la etiqueta.
    - Ventajas: Alta tasa de aciertos. Mucha flexibilidad al reemplazar, permitiendo algoritmos más sofisticados.
    - Desventajas: Etiquetas grandes, mecanismo de comparación complejo (buscar en todas las etiquetas simultáneamente).
- **Asociativa por conjuntos:** Compromiso. La caché se divide en v conjuntos (de correspondencia directa) de k líneas (de correspondencia asociativa). El conjunto al que corresponde el bloque = N° bloque % v, y el bloque puede asignarse a cualquiera de las

líneas del conjunto. Así, se examinan en simultáneo k vías por vez. La dirección tiene parte etiqueta parte índice, con menos bits de índice en general que la directa.
    - Ventajas: Etiqueta más corta que con totalmente asociativa, buena tasa de aciertos, control no extremadamente complejo.

Política de reemplazo

Cuando se produce un fallo hay que traer el bloque de la memoria a la caché (por principio de localidad temporal y espacial).

Algoritmos necesarios únicamente para correspondencias mediante asociatividad. Para gran velocidad, los algoritmos se implementan en hardware.

- **Aleatorio:** Se elige el bloque a sustituir al azar. Prestaciones ligeramente inferiores a los algoritmos basados en utilización.
- **LRU, Least-Recently Used:** Más efectivo. Se sustituye el bloque que haya sido utilizado hace más tiempo. Cada línea incluye los bits necesarios para representar el orden en que fueron usados y se actualizan con cada acceso. El accedido cambia a cero mientras que los demás se reordenan. Se asume que es más probable referenciar los bloques que fueron referenciados recientemente.
- **FIFO, First In-First Out:** Se reemplaza el que lleva más tiempo en la caché, en forma de cola o buffer circular (sin actualizar cuando se referencia un bloque).
- **LFU, Least-Frequently Used:** Se cuenta la cantidad de referencias que tuvo el bloque en vez de hace cuanto se utilizó. Se reemplaza el que se haya utilizado menos veces.

Política de escritura

Se debe mantener la coherencia de cache al escribir. La CPU escribe sobre una línea de cache, por lo que el bloque de memoria ppal debe ser actualizado, pero también un dispositivo de E/S puede escribir la memoria y la caché se debe actualizar. En procesamiento paralelo, cada CPU puede tener caches individuales que se deben sincronizar.

Escrituras en acierto:

- **Write-through/escritura inmediata:** Al CPU alterar la caché se actualiza inmediatamente la memoria. El bus de memoria se monitorea para que cualquier alteración de la memoria externa al CPU se refleje en la caché.
    - Ventajas: Simple
    - Desventajas: Mucho tráfico a la memoria.
- **Write-back/post-escritura:** Se actualiza solo la caché. Cuando se actualiza una línea se activa un bit que indica que fue actualizada. Al momento de reescribir el bloque, si fue actualizado en caché se actualiza la memoria. Los módulos de E/S a veces deben interactuar con la caché si la memoria no es válida, lo que complejiza el circuiterío.
    - Ventajas: Minimiza el tráfico a memoria.
    - Desventajas: Lógica de control compleja.

Escrituras en fallo (se quiere escribir y la línea no está en la caché):

- **No write allocate:** Se lleva la escritura directamente a la memoria principal, luego se actualiza la caché. Habitual con writethrough.
- **Write allocate:** La info se lleva de memoria principal a la caché sobreescribiendo en caché, y después el CPU hace su escritura en caché. Habitual con writeback.

Para sistemas con memoria compartida entre diferentes dispositivos en el mismo bus, generalmente 2+ CPUs con su caché, tenemos varios sistemas para mantener la coherencia de caché (ver capítulo 16 para mas info, pero no lo deberían tomar):

- Vigilancia de bus con escritura inmediata: Cada controlador de caché monitorea los buses de direcciones. Si otro maestro escribe sobre memoria, se invalida la línea en todas las demás cachés. Solo para escritura inmediata.

- Transparencia de hardware: Hardware adicional que sincroniza todas las cachés y la memoria de forma inmediata.
- Memoria excluida de caché: Solo una porción de memoria se comparte con más de un procesador a la vez y no puede ser transferida a caché

Anexo Clase 7: Buses

Organización y Arquitectura de Computadores, W. Stallings, Capítulo 3. 5º edición.

Diseño y evaluación de arquitecturas de computadores, M. Pardo y A. Guzmán, Capítulo 2 (sección 2.8) y Capítulo 4 (secciones 4.2 a 4.4). 1º edición.

Buses

Un computador está constituido por un conjunto de módulos de tres tipos elementales (procesador, E/S, memoria) que se comunican entre sí, haciendo de la computadora una red de dispositivos. Las conexiones entre módulos se denomina **estructura de interconexión**.

Los buses permiten describir al computador mediante la estructura de interconexión y los controles necesarios para gestionar el uso de dicha estructura.

Definición: Medio de transmisión compartido que interconecta dos o más dispositivos y permite que se establezca entre ellos una correcta comunicación (*diseño y evaluación de arquitectura de computadoras*).

Un maestro inicia y dirige la transferencia y un esclavo obedece las órdenes.

Interconexiones entre (uno o más de cada uno)

- Memoria/s. Las palabras de datos pueden leerse y escribirse, y para eso deben identificarse dentro de la memoria.
- Módulo/s de E/S. Un módulo de E/S puede controlar a más de un dispositivo externo mediante puertos. Un dispositivo de E/S también puede enviar señales de interrupción al procesador.
- Procesador/es. Lee instrucciones y datos, y devuelve resultados. Utiliza señales para controlar el sistema y recibe señales de interrupción.

Funciones

- Recibir y entregar datos
- Recibir direcciones
- Recibir señales de control (lectura, escritura, temporización)

Durante los años se experimentó con diversas estructuras. Se llegó a estructuras de buses múltiples como la más ampliamente utilizadas. También hubo estructuras de único bus, pero las prestaciones disminuyen a medida que se conectan más dispositivos y se requiere de más ancho de banda.

Estructura del bus

Un bus es un camino de comunicación entre dos o más dispositivos y generalmente el medio de transmisión de información. Si uno transmite una señal, los otros dispositivos conectados la reciben. Si dos se transmiten a la vez, las señales se solapan y distorsionan.

Son conductores eléctricos paralelos con conectores o conexiones en sus extremos.

Un bus está constituido por líneas: varios caminos de comunicación. Cada línea transmite un 1 o un 0 a la vez. A lo largo del tiempo se puede enviar una secuencia de 1s y 0s. Varias líneas permiten la transmisión de varios dígitos en paralelo.

Un bus suele estar constituido por entre 50 y 100 líneas cada una con una función particular. Suele ser un grupo de líneas de un mismo tipo. La cantidad de cierto tipo de un bus determina el ancho del bus. Las líneas pueden clasificarse en cuatro grupos:

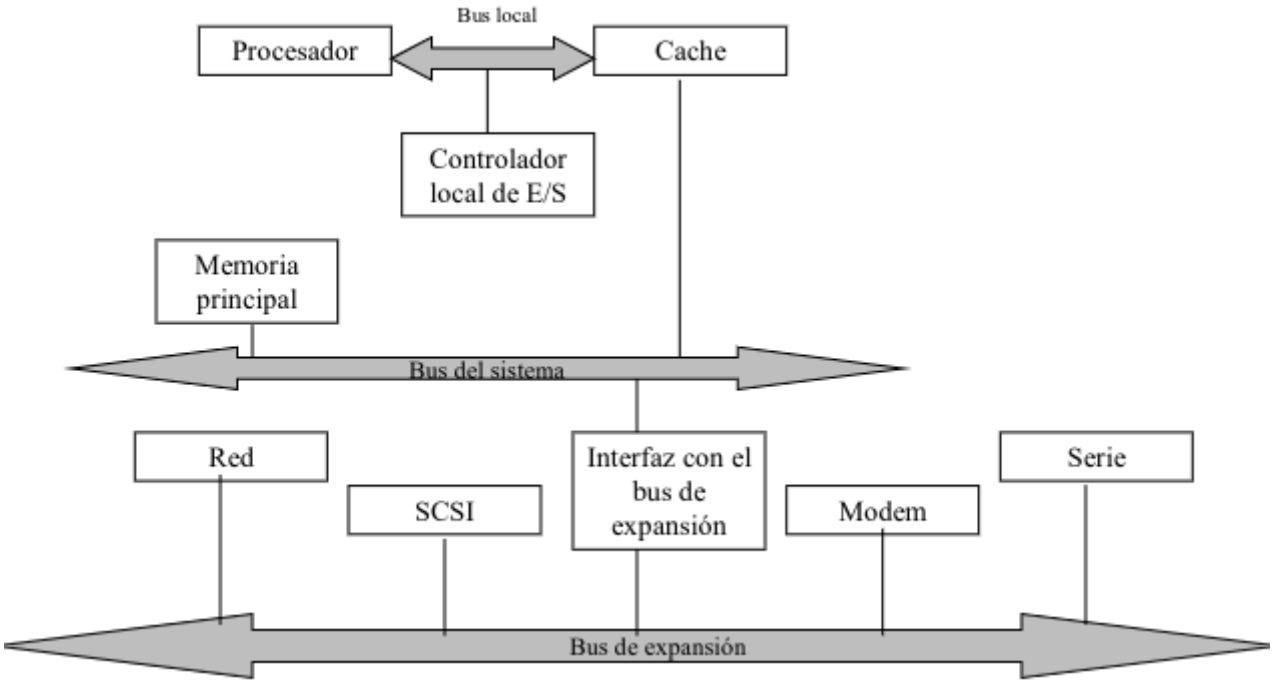
- **Líneas de datos:** El conjunto de líneas de datos se denomina bus de datos. El ancho del bus indica la cantidad de bits de información que se transfieren en paralelo.
- **Líneas de dirección:** Utilizadas para indicar la fuente o destino del dato al que se accederá por el bus de datos. El ancho indica el rango de direcciones que se pueden referenciar, indicando la cantidad máxima de memoria que se puede conectar al bus. También se usan para direccionar puertos o módulos de E/S (en general los bits más altos indican el puerto o el módulo y los demás la dirección dentro del módulo).
- **Líneas de control:** En general utilizadas para controlar el acceso y uso a líneas de datos y direcciones. Transfieren órdenes de lectura/escritura de memoria/E/S, señales reconocimiento de transferencia, interrupción, peticiones de bus y/o señales de temporización (reloj).
- **Líneas de alimentación:** Suministran energía a los módulos conectados.

Jerarquía de buses

Si se conecta un gran número de dispositivos a un bus las prestaciones disminuyen debido a

- Mayor retardo de propagación: Tiempo que necesitan los dispositivos para coordinar el uso del bus. Si el control se pasa frecuentemente los retardos de propagación afectan las prestaciones.
- Se convierte en un cuello de botella cuando las peticiones de transferencia aproximan la capacidad del bus. Al tener que transferir grandes cantidades de datos, el bus es bloqueado.

Por eso se organizan los buses jerárquicamente. Una estructura tradicional es:



Normalmente todas las jerarquías tienen

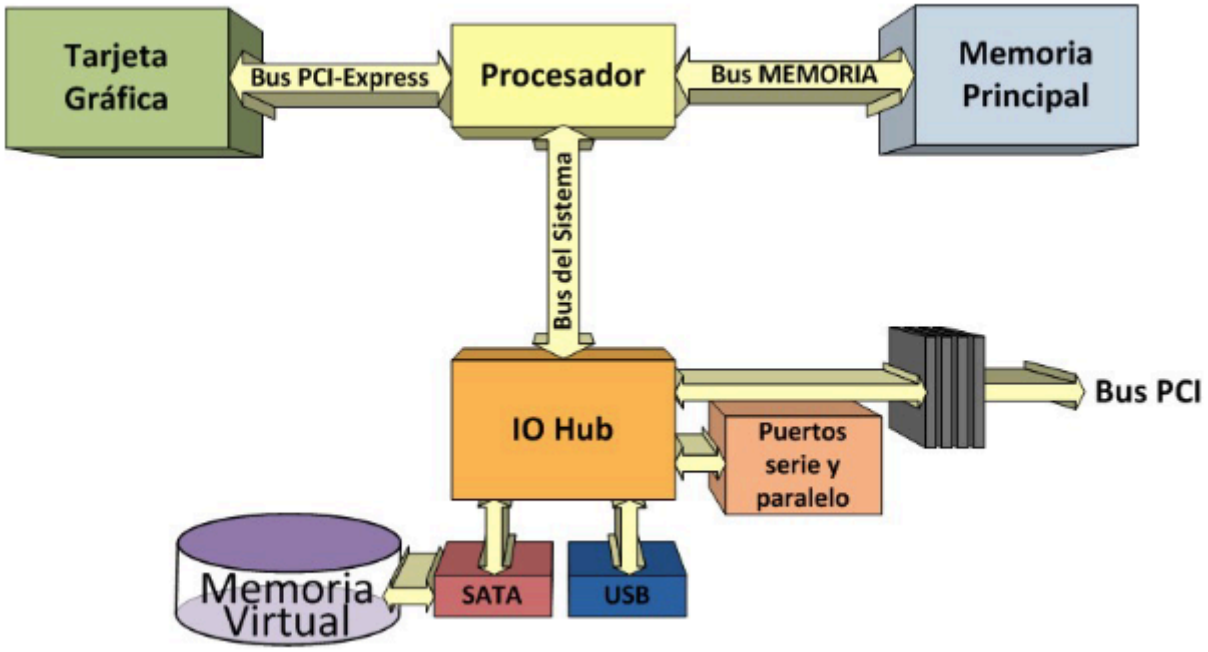
- **Bus de sistema y bus de memoria:** Conecta al procesador con el resto del sistema y la memoria principal con el controlador de memoria. Rápidos, cortos en general no estandarizados y optimizados para diseños y arquitecturas específicos. Controla al flujo de datos hacia y desde memoria
- **Buses de expansión:** Buses más largos, lentos, estandarizados, accesibles por el usuario para conectar dispositivos desconocidos al momento de diseñar la arquitectura. Aísla el tráfico entre dispositivos de E/S y la memoria y procesador y permite la conexión de variedad de dispositivos sin afectar gravemente el funcionamiento de los módulos más importantes (CPU, memoria)
- Bus local: Otros buses, no siempre aparecen, son extremadamente cortos. Controla los datos entre el procesador, el controlador local de E/S (para ciertos dispositivos privilegiados) y el

controlador de caché, que se conecta con el bus del sistema para acceder a la memoria en el ejemplo dado.

Esto es eficiente, pero a medida que los dispositivos de E/S son cada vez más rápidos y requieren de mayor ancho de banda se propuso una “arquitectura de altas prestaciones” con red, SCSI (Small Computer System Interface, interfaz estándar para transferir datos entre dispositivos. Ahora USB), gráficos, etc. en un bus de alta velocidad conectado al caché/adaptador, del que parte el bus local al procesador y el bus del sistema a la memoria. Al bus de alta velocidad se conecta la interfaz del bus de expansión donde se conectan dispositivos de menores prestaciones para que no afecten a los de mayores prestaciones. Con el paso del tiempo se fueron agregando más buses de expansión.

Evolución de la jerarquía de buses

De *diseño y evaluación de arquitectura de computadoras*



Breve historia:

1. El procesador se comunica con el controlador de memoria en el chipset norte mediante el bus del sistema. Al chipset norte se conecta la memoria y la tarjeta gráfica mediante AGP (se trata diferente a los demás módulos de E/S por ser de mayores prestaciones). El resto de los dispositivos se conectan a través de PCI u otros buses e interfaces, o al chipset sur al que se conecta almacenamiento, E/S, etc y se comunica con el chipset norte mediante PCI.
2. El chipset norte evolucionó a MCH, Memory Controller Hub. A este se conectan por separado el procesador mediante el bus del sistema, la gráfica mediante AGP, la memoria principal mediante el bus de memoria, y el ICH, IO Controller Hub, que sustituye al chipset sur. A este se conectan todos los dispositivos misceláneos y un bus PCI para variedad de dispositivos. Mejora el rendimiento de los dispositivos conectados a PCI al no interferir con el ICH directamente sino estar conectados a él
3. Los procesadores comienzan a venir con controlador de memoria y surge el PCI Express. Surgen tres alternativas:
    a. Al Graphics Tunnel se conecta el procesador por el bus del sistema, la memoria por bus de memoria, la gráfica por PCI-e, y se conecta el IO Hub (ex ICH) ahora con PCI-e
    b. Al procesador se conecta la memoria por el bus de memoria. El IO-Hub se conecta al procesador por el bus del sistema. La gráfica va al IO Hub mediante PCI-e.
    c. (Imagen de arriba). La gráfica habla con el procesador en vez del IO Hub.

Elementos de diseño del bus

Tipos de buses

- **Dedicados:** Una línea de un bus dedicado tiene asignada siempre la misma función o subconjunto físico de componentes del computador. Un ejemplo común son líneas separadas para direcciones y para datos (*x* líneas de direcciones, *y* líneas de datos y una línea de control de lectura y escritura por ejemplo)
- **Multiplexados:** Las líneas pueden tener varias funciones. Ahorran espacio y costo, pero necesita circuiterio más complejo para aparentar que tiene más líneas (se usan las mismas líneas para direcciones o para datos, y hay una línea que indica si los valores corresponden a direcciones o datos, por ejemplo).

Método de arbitraje

La gran mayoría de sistemas con más de un módulo puede necesitar tener el control de un bus. El método de arbitraje indica qué módulo es el maestro a la vez. La clasificacion de metodos de arbitraje aproximadamente es:

- **Centralizado:** Un único dispositivo de hardware (controlador, árbitro) asigna los tiempos del bus. Los dispositivos se conectan al árbitro.
- **Distribuido:** No hay controlador central. Cada módulo tiene lógica de control para indicar cuando toma control del bus y cuando lo libera, y cooperan para compartirlo.

Temporización

Es la forma en que se coordinan los eventos del bus

- **Síncrono:** La presencia de eventos se indica con pulsos de reloj. Para esto incluye una línea de reloj. Todos los dispositivos pueden leer la línea de reloj y la mayoría de los eventos se prolongan por un único ciclo de reloj. Las señales en las líneas cambian generalmente entre el flanco descendente y el ascendente, el ascendente indicando el inicio del ciclo (se sincroniza en el franjo ascendente). Simple de implementar y probar. Como todos los dispositivos dependen de la misma frecuencia de reloj, no aprovecha al máximo las prestaciones de estos.
- **Asíncrono:** Los eventos dependen de eventos previos. Un cambio de señal indica algo, que produce otro evento en otra señal, y así. Más flexible que la temporización síncrona pero más compleja.
- **Semisíncrono:** Inicialmente se impone una velocidad, pero dispositivos más lentos pueden funcionar de forma asíncrona si lo requieren. Tiene grandes ventajas sobre los primeros dos pero requieren de lógica más compleja y más líneas.
- **Ciclo partido / robo de ciclo:** El bus se libera entre la petición por parte del maestro y la contestación del esclavo. Tiene grandes ventajas sobre los primeros dos pero requieren de lógica más compleja y más líneas.

Anchura

- **Dirección:** A más ancho, más direcciones se pueden referenciar.
- **Datos:** A más ancho, más datos a la vez se pueden enviar. Menos llamadas al bus para datos grandes.

Tipo de transferencia de datos

- **Lectura:** Dato de esclavo a maestro
- **Escritura:** Dato de maestro a esclavo
- **Lectura-Modificación-Escritura:** Combinación de las dos en un mismo bus utilizando una dirección una vez y luego siempre leer un dato y escribir un dato.

- **Lectura después de escritura:** Al revés del anterior. La operación de lectura luego de escribir se puede utilizar para comprobar el resultado
- **Bloque:** Un ciclo de dirección es seguido por varios ciclos de datos.

Bus PCI

Peripheral Component Interconnect, interconexión de componente periférico. Elevado ancho de banda, independiente del procesador. Creado por Intel en 1990, liberado más tarde.

El bus PCI se usa para conectar el controlador de memoria (al que se conecta el conjunto CPU, Caché, DRAM) con audio, video, red, el adaptador del bus de expansión, etc. O tambien a traves de un adaptador PCI conectar al bus del sistema adaptadores al bus de expansión y múltiples dispositivos o incluso adaptadores PCI-PCI

PCI Express: Hay ahora un switch al que se conectan todos los dispositivos PCI-e que gestiona los paquetes, parecido a un network chip. Síncrono o semisíncrono. Soporta diferentes tipos de paquetes.

Clase 8: Procesadores Superescalares

Organización y Arquitectura de Computadoras, William Stallings, Capítulo 13 en 5º ed. Capítulo 14 en 7º ed.

Diseño y evaluación de arquitecturas de computadores, M. Pardo y A. Guzmán, Capítulo 3. 1º ed.

Procesador Supersegmentado

Procesador escalar: Ejecución secuencial de instrucciones. En un procesador escalar segmentado, las instrucciones se ejecutan en todas las etapas a la vez. El WinMips.

La evolución de los procesadores escalares y cauces segmentados lleva a los procesadores supersegmentados y superescalares.

El enfoque supersegmentado se basa en:

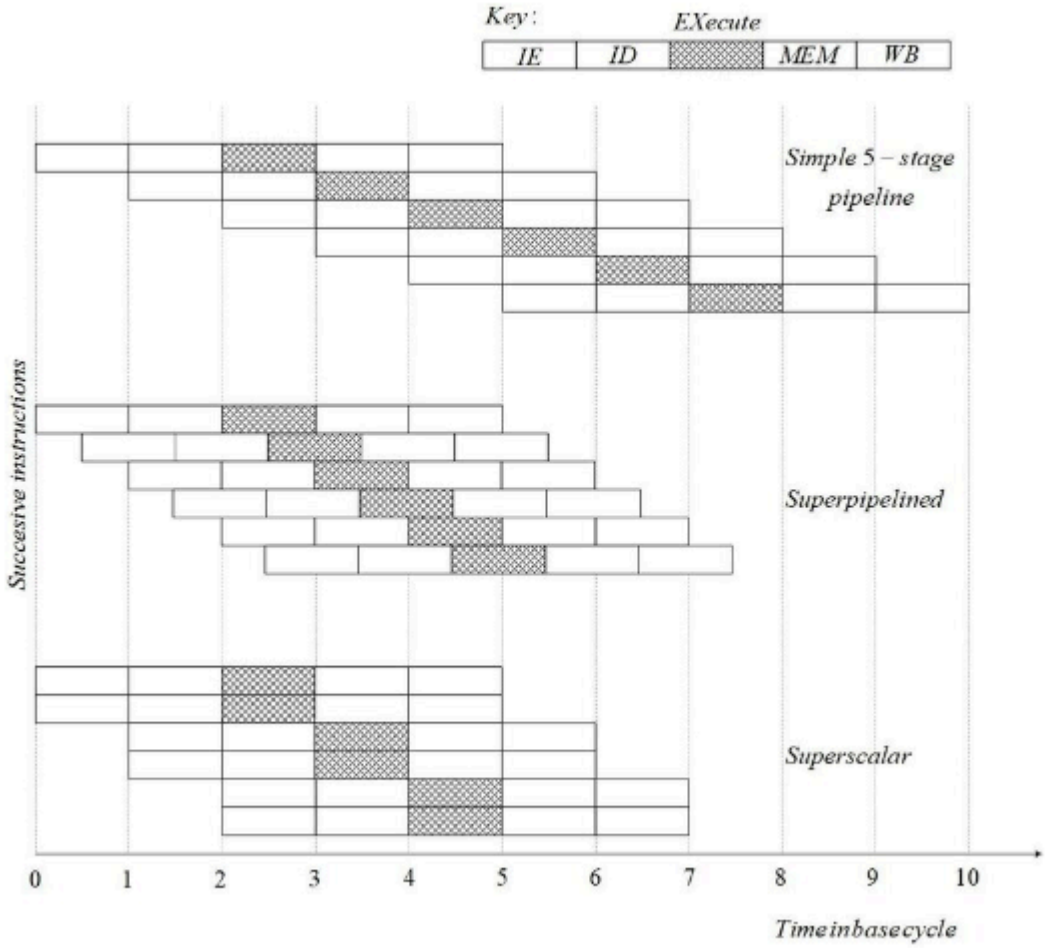
- Muchas operaciones no requieren de todo un ciclo de reloj
- Se pueden dividir los ciclos en subintervalos → Mayor frecuencia de reloj
- División de etapas en subetapas.
- El tiempo de instrucciones individuales no varía pero aumenta el paralelismo y se percibe mayor velocidad

Procesador Superescalar

Permite llevar a cabo instrucciones diferentes en diferentes cauces

- Duplica algunas o todas las partes del CPU/ALU
- Es capaz de captar múltiples instrucciones al mismo tiempo
- Operaciones aritméticas simultáneas
- Carga o escritura de datos mientras se realiza una operación en la ALU

La aproximación superescalar puede aplicarse tanto a CISC como a RISC, pero es mucho más común en RISC por las facilidades que provee el repertorio de instrucciones.



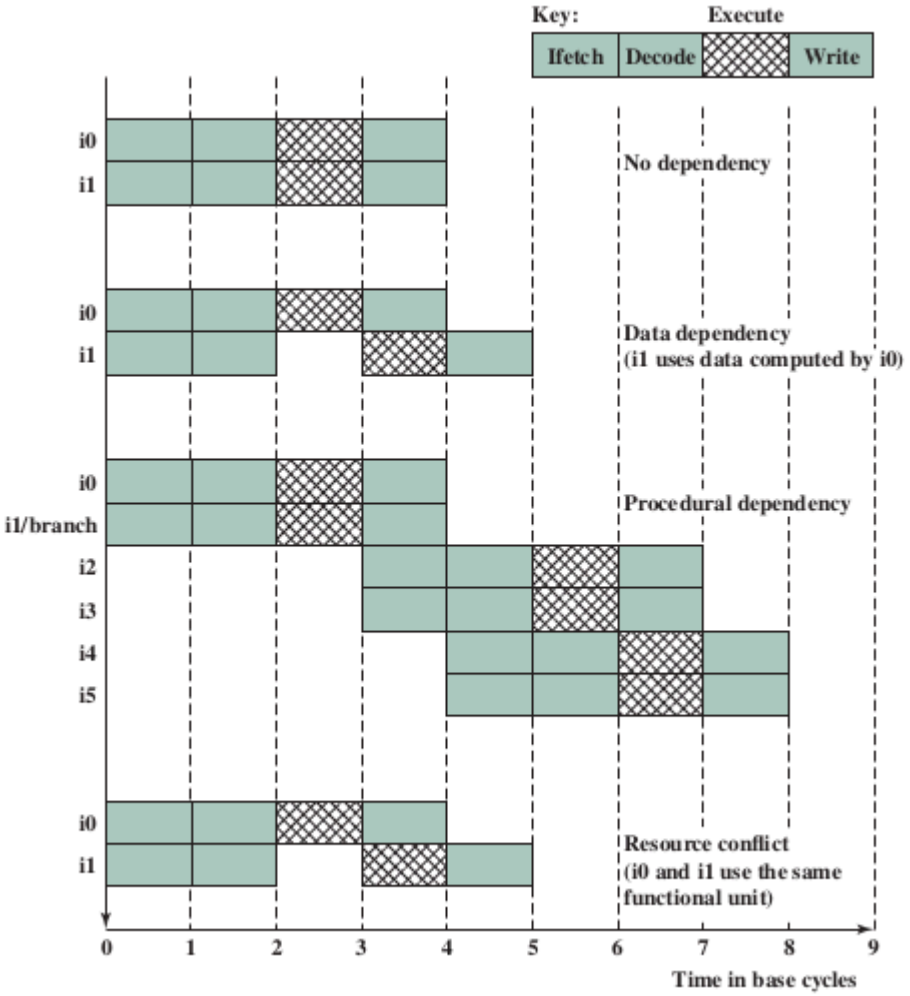
Paralelismo a nivel de instrucción

Limita la ejecución paralela de instrucciones, al igual que con los procesadores segmentados no superescalares.

Hay paralelismo a nivel de instrucción cuando dos instrucciones se pueden ejecutar independientemente sin dependencia una de la otra.

Limitaciones:

- Riesgos estructurales (conflicto de recursos): Mismas instrucciones usan el mismo hardware
- Dependencias de control (dependencia relativa al procedimiento): Se debe conocer una condición antes de captar la siguiente instrucción
- Dependencias de datos: RAW, WAR, WAW entre las instrucciones a ejecutar en paralelo



Una máquina con instrucciones de longitud fija (un RISC) tendrá un alto paralelismo a nivel de instrucciones.

Paralelismo a nivel de máquina

Capacidad del procesador para sacar provecho del paralelismo de las instrucciones.

Depende del número de instrucciones que se pueden captar y ejecutar en el mismo instante, y de los mecanismos para localizar instrucciones independientes. A veces un programa puede no tener el paralelismo a nivel de instrucciones necesario para sacar provecho al paralelismo a nivel de máquina, y a veces un programa es fácilmente paralelizable pero no hay suficiente paralelismo a nivel de máquina para sacarle provecho.

Políticas de emisión de instrucciones

Deben localizarse las instrucciones independientes y definir su orden de captación y ejecución para hacer uso óptimo del cauce atendiendo a las dependencias y lograr un gran paralelismo a nivel de instrucción.

El procesador debe identificar el paralelismo y organizar la captación, decodificación y ejecución de instrucciones en paralelo.

Emisión de instrucciones: Proceso de iniciar la ejecución de instrucciones en unidades funcionales del procesador

Política de emisión de instrucciones: Protocolo usado para emitir instrucciones

Es importante:

- El orden de captación
- El orden de ejecución

- El orden en que las instrucciones actualizan registros y memoria

Cuanto más sofisticado el procesador (mas paralelismo a nivel máquina), menos limitado estará por las ordenaciones.

Para optimizar la ejecución el procesador debe alterar el orden en que se ejecutarán las instrucciones en un proceso estrictamente secuencial.

Emisión y finalización en orden

La forma más sencilla. Se emiten las instrucciones en el mismo orden que en una ejecución secuencial (emisión en orden), y se escriben los resultados en el mismo orden. Ni siquiera los escalares siguen esta política, pero sirve de base para otros métodos.

Emisión en orden y finalización desordenada

Se usa en los RISC escalares para mejorar la velocidad de instrucciones que necesitan de muchos ciclos.

Puede haber cualquier número de instrucciones en etapa de ejecución hasta alcanzar el paralelismo máximo de máquina utilizando todas las unidades funcionales.

La emisión se frena cuando hay conflictos, hasta entonces decodifica instrucciones.

De esta manera surgen las **dependencias de salida** WAW.

Es más difícil ocuparse de las interrupciones y excepciones.

Emisión y finalización desordenada

Se desacoplan las etapas de cauce de decodificación y ejecución utilizando un buffer llamado **ventana de instrucciones**. La ventana no es una etapa adicional del cauce. Que la instrucción esté en la ventana indica que se tiene suficiente información sobre ella para saber si se puede emitir.

Cuando se termina de decodificar la instrucción se coloca en la ventana de instrucciones. Mientras el buffer no se llena, se sigue decodificando. Cuando una unidad funcional de la etapa de ejecución queda disponible, se emite una instrucción desde la ventana de instrucciones a la etapa de ejecución, siempre que necesite de esa unidad funcional y que no la bloquee la dependencia de datos.

Así el procesador tiene la capacidad de anticipar bloqueos.

Surge la **antidependencia** (WAR).

Emisión desordenada y finalización ordenada

Ver consideraciones más abajo. Evita excepciones imprecisas, se utiliza renombramiento de registros y agrega la etapa de escritura temporal.

Renombramiento de registros

La emisión y/o finalización desordenada genera dependencias de salida y antidependencias, que surgen porque los valores de los registros no pueden reflejar la secuencia de valores dictada por el flujo del programa.

Cuando se desordenan las instrucciones no se puede conocer el valor de los registros en un punto en el tiempo basándose únicamente en la secuencia de instrucciones del programa. Además, por las técnicas de optimización (ver sección RISC) que maximizan el uso de registros.

Como solución se emplea de vuelta la duplicación de recursos. El hardware del procesador asigna dinámicamente los registros asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. Cuando se ejecutan unas instrucciones que tiene un registro como operando destino se asigna un nuevo registro para ese valor. Las instrucciones posteriores que accedan a ese valor sufren un proceso de renombramiento que desvía las referencias a registros de las

instrucciones para referenciar el registro que contiene el valor que necesitan. Así las referencias a valores antes y después de modificar el registro pueden desordenarse más fácilmente.

Implementación del procesador superescalar

En conclusión tenemos que para un procesador escalar se necesitan

- Estrategias de captación simultánea de instrucciones
- Lógica de detección de dependencias entre valores de registros y mecanismos para comunicar los valores
- Mecanismos para iniciar o emitir múltiples instrucciones en paralelo
- Recursos para ejecución en paralelo
- Mecanismos para entregar el estado del procesador en orden correcto

Recursos de hardware para mejorar las prestaciones:

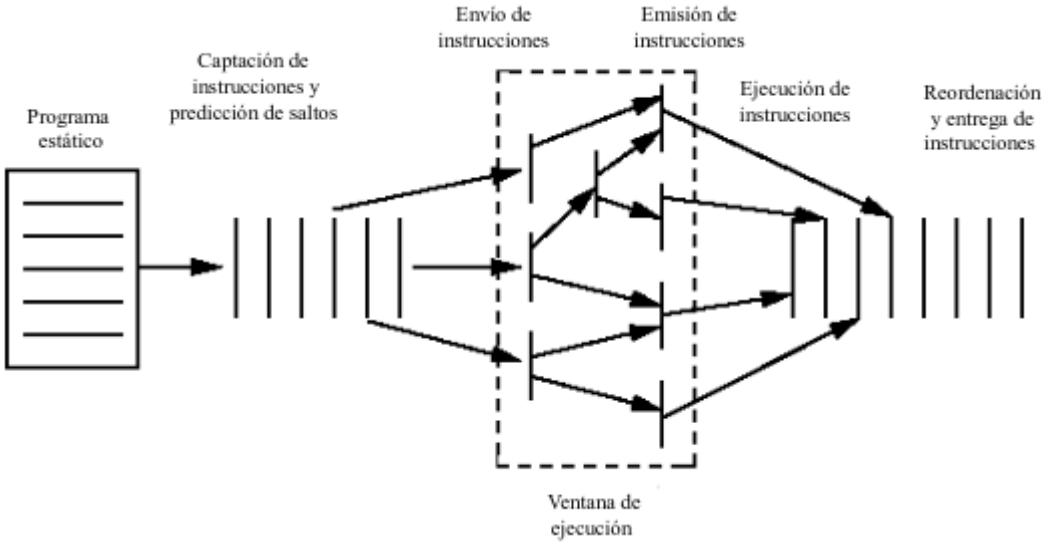
- Duplicación de recursos
- Emisión desordenada
- Renombramiento

Las tres técnicas tienen relación y solo aumentan de forma apreciable las prestaciones si se utilizan en conjunto.

Todo esto se complementa con la predicción de saltos y el salto retardado para minimizar los conflictos de control, aunque el salto retardado perdió interés debido a que hay que ejecutar múltiples instrucciones en un superescalar y genera muchos problemas de dependencia. En general (en 1996, Stallings de quinta edición), se volvió a técnicas simples como predicción estática o técnicas basadas en la historia de saltos (switch, BTB).

El compilador tiene gran influencia en el aprovechamiento del procesador superescalar.

Ejecución superescalar



1. El programa es una secuencia lineal de instrucciones. Programa estático
2. El proceso de captación de instrucciones (incluyendo predicción de saltos) se usa para formar un flujo dinámico de instrucciones.
3. Se examinan las dependencias y se eliminan las artificiales (usando renombramiento)
4. Se envían las instrucciones a una ventana de instrucciones.
5. Se ejecutan las instrucciones de cada instrucción en cierto orden determinado por las dependencias verdaderas y la disponibilidad de recursos. Los resultados son almacenados en registros temporarios (etapa de escritura temporal) (no se mezclan los resultados con el almacenamiento permanente o los registros visibles hasta el siguiente paso)
6. Las instrucciones se reordenan conceptualmente en orden secuencial y se registran los resultados (*commit or retire*)

a. Si se ejecuta una instrucción captada por el mecanismo de predicción de saltos y resulta que no se debía saltar, el resultado se descarta

Excepciones

Durante la ejecución pueden surgir excepciones con varias instrucciones en ejecución, y generan problemas cuando la finalización es desordenada. Si una finalización termina antes que otra que genera una excepción, se tiene excepciones imprecisas, estado inconsistente.

Si una instrucción primera produce una excepción, esta instrucción y las siguientes deben ser abortadas. Luego se comienza por la que origina la excepción. Esto asegura un estado consistente.

Para tener un estado consistente con excepciones imprecisas se debe retardar la escritura de datos para que los resultados se presenten en orden y sea posible abortarlos en caso necesario.

Las interrupciones externas son excepciones precisas, la unidad de emisión deja de emitir y se cancela la cola, pero las instrucciones pendientes se completan y comienza el procesamiento de la interrupción.

Consideraciones

Ejecución desordenada → Libera las unidades de ejecución rápidamente

Completar las instrucciones en orden → Reducen las excepciones imprecisas

Solución: **Emisión desordenada y finalización ordenada** con **etapa de escritura temporal** (renombramiento de registros).

Clase 9: Superescalares y más; VLIW

Organización y Arquitectura de Computadoras, W.Stallings, Capítulo 13 en 5º ed. ó Capítulos 14 y 15 en 7º ed.

Diseño y evaluación de arquitecturas de computadores, M. Pardo y A. Guzmán, Capítulo 3.3. 1º ed.

Pentium IV

El diseño superescalar se suele aplicar a procesadores RISC, aunque también se ha implementado en procesadores CISC. El Pentium traduce las instrucciones CISC a instrucciones más simples (microoperaciones o micro-ops) que lo llevan a parecerse a un RISC, lo que permite la segmentación de instrucciones.

Pentium: 2 unidades de ejecución de enteros separadas. Pentium Pro: 100% superescalar.

Último de la línea x86.

IA-64

Colaboración Intel-HP.

arquitectura de 64bit no basada en x86 ni adapta la arquitectura RISC de HP.

Base superescalar.

Motivaciones: Aprovechar el ILP. No se determina por el procesador en tiempo de ejecución sino por el compilador, y las múltiples instrucciones se incluyen en una LIW/VLIW (Long Instruction Word / Very Long Instruction Word) → Paralelismo explícito.

Se le llama EPIC (Explicit Parallel Instruction Computing) a esta organizacion, y a IA-64 la implementación. Itanium es el primer producto (y la línea Itanium fue la única en utilizarla), y tiene características superescalares y EPIC.

Gran número de registros (IA-64 asume 256) + 64 registros de 1 bit para ejecución predicada.

Múltiples unidades de ejecución (8 o más). I-Unit (Integer unit), M-Unit (memory unit), B-Unit (branch unit), F-Unit (floating point unit).

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.

Paralelismo explícito

Paralelismo de instrucciones programado en tiempo de compilación, incluido con la instrucción de máquina. Lleva al compilador a ser más complejo para poder sacar provecho de la operación paralela y detectar el paralelismo a alto nivel examinando todo el programa.

El procesador usa esa información para realizar la ejecución paralela, simplificando circuiterío.

Predicación de saltos

Cualquier instrucción puede referirse a un registro de predicado. La operación se retirará cuando el predicado sea verdad. Si el valor se conoce aunado se emite la instrucción, se decide en base al valor si se ejecuta o no. Si no se conoce, se inicia la operación y se descarta si el valor se vuelve falso.

La comparación para determinar la condición de salto altera al predicado.

Es una técnica de compilación para generar código con alto grado de ILP.

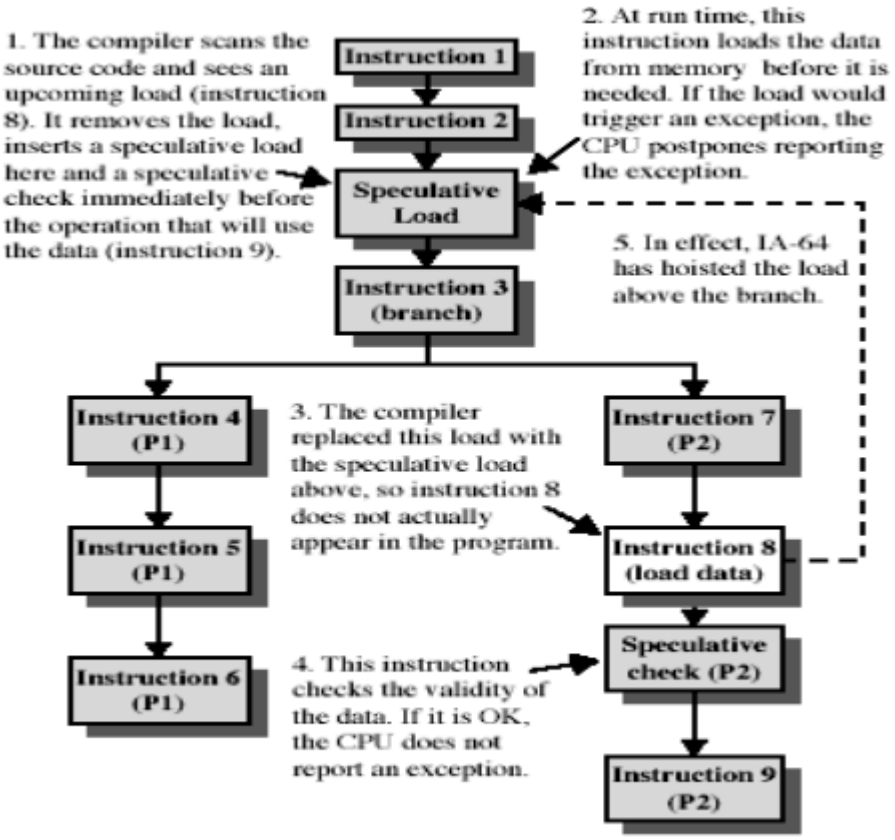
Logra que ambas ramas de un salto condicional se ejecuten en paralelo.

Carga especulativa

La instrucción load se puede reubicar para evitar la latencia. Si se mueve dentro de un salto se ejecuta para ambas ramas → Si ocupa hardware que podría haber sido utilizado para otra cosa es tiempo perdido. Por eso la carga especulativa.

La instrucción load se reemplaza por

- Carga especulativa (LOAD.S): Busca en memoria, detecta una excepción (como fallo de caché). es la instrucción que se puede cambiar de lugar
- Chequeo (CHK.S): Justo antes de la instrucción que utiliza el dato. Reporta si se detectó una excepción en la carga especulativa. Permite manejar excepciones



(b) Speculative loading

VLIW

Very Long Instruction Word de IA-64

- Bundle de 3 slots de instrucciones (128)
- Instrucción de formato de 41 bit con un registro predicado, 3 de registro general o FP y otros bits para cosas varias.

Clase 10: Procesamiento paralelo

Organización y Arquitectura de Computadoras, William Stallings, Capítulo 16 de 5ta edición ó Capítulo 18 de 7ma

edición.

Diseño y evaluación de arquitecturas de computadoras, M. Beltrán y A. Guzmán, Capítulo 5 de 1ra edición.

Notar que hay multiprocesadores on-chip con memoria compartida y memoria distribuida (estos últimos necesitan de una red on-chip, NoC).

Hay un montón de información en *diseño y evaluación de arquitectura de computadoras* sobre las redes de interconexión.

Se usa un switch para manejo de paquetes en redes con arbitraje centralizado.

Taxonomía de Flynn

Clasificación usada por Stallings.

Clasificación de las arquitecturas paralelas basado en la explotación del paralelismo de datos y el paralelismo funcional. Es la más común para clasificar a los sistemas según sus capacidades de procesamiento paralelo.

SISD: Single Instruction, Single Data

Un único procesador interpreta una única secuencia de instrucciones para operar con los datos almacenados en una única memoria. Monoprocesador.

SIMD: Single Instruction, Multiple Data

Una única máquina controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos del 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 (una operación se aplica para todos los elementos de un vector o matriz. Se extienden las operaciones escalares a operaciones vectoriales).

MISD: Multiple Instruction, Single Data

Una secuencia de datos se transmite a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. No existe, solo de relleno.

MIMD: Multiple Instruction, Multiple Data

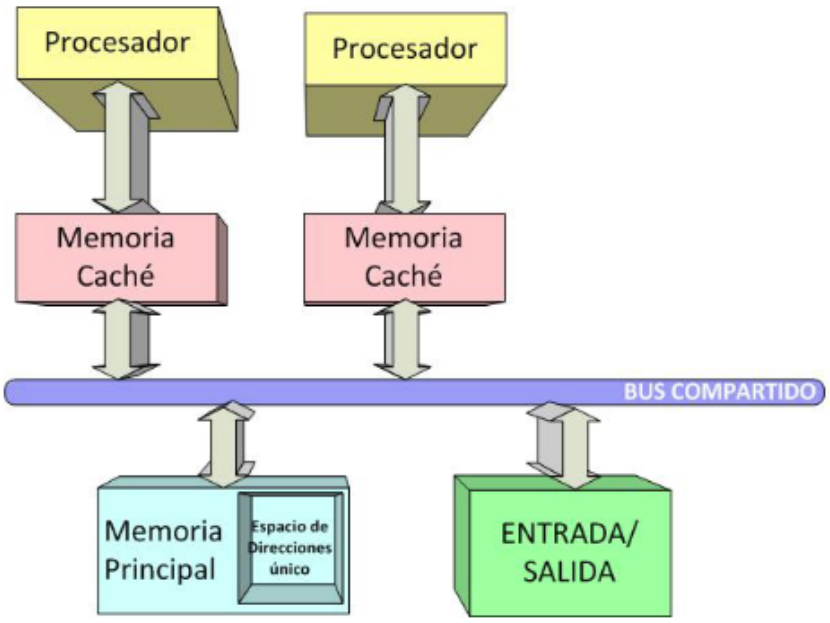
Un conjunto de procesador ejecuta simultáneamente secuencias de instrucciones diferentes con datos diferentes. SMP, clusters, NUMA.

Los procesadores en MIMD son de uso general capaz de procesar todas las instrucciones del conjunto de instrucciones. Se subdividen según la forma que tienen de comunicarse.

Memoria distribuida (débilmente acoplada) (multicomputador): Clusters

Memoria compartida (fuertemente acoplada) (multiprocesador): SMP y NUMA

SMP: Symmetric MultiProcessing



SMP se refiere tanto a la arquitectura de hardware de la arquitectura como al comportamiento del OS que la utiliza

- Computadora autónoma **MIMD**
- Dos o más procesadores similares de capacidades comparables (mejor si son iguales, gemelos). Cada uno es autónomo.
- Comparten la memoria principal y la E/S
- Interconectados mediante un bus u otro tipo de sistema de interconexión → **Bus de tiempo compartido**

- Tiempo de acceso a memoria similar para todos los procesadores (**UMA**, Uniform Memory Access)
- Todos los procesadores pueden desempeñar las mismas funciones
- Sistema operativo integrado común a todos los procesadores (antes Windows NT separado de los demás. Ahora todos sirven para SMP) → Este punto marca la mayor diferencia con los clusters o sistemas débilmente acoplados. **La existencia de varios procesadores es transparente al usuario.**
- La interacción se puede producir a través de elementos de datos individuales y puede haber un elevado grado de cooperación entre procesadores. En general se comunican por memoria pero puede pasar que se envíen mensajes.

El OS planifica la distribución de threads entre todos los procesadores.

A veces cada procesador puede tener su propia memoria y E/S privada además de los recursos compartidos.

Ventajas frente al monoprocesador (potenciales, no garantizados):

- **Prestaciones:** Si el trabajo es paralelizable, un sistema con varios procesadores funciona con mejores prestaciones
- **Disponibilidad:** Un fallo en un procesador no hace que el computador se detenga
- **Crecimiento incremental:** Se puede aumentar las prestaciones añadiendo más procesadores
- **Escalado:** Los fabricantes ofrecen distintos productos en base a la cantidad de procesadores
- **La existencia de varios procesadores es transparente al usuario.**

Desventajas:

- Prestaciones limitadas por el tiempo de ciclo del bus
- Cada procesador debe tener su caché para reducir el número de accesos a memoria (y no abrumar al bus)
- Problemas de coherencia de caché → Se resuelve por hardware con protocolos de sondeo y protocolos de directorio

Límite práctico en la cantidad de procesadores: entre 16 y 64 por degradación de prestaciones

Organización del SMP

Hay mucha más información sobre SMP en el Stallings. Consideraciones en el diseño de un sistema operativo multiprocesador, más tipos de interconexión, organización de hardware, coherencia de caché, etc. Es un tema muy profundo.

Bus de tiempo compartido

Estructura e interfaces similares a un sistema monoprocesador con un bus de interconexión.

Para facilitar las transferencias DMA hay elementos para el

- **Direccionamiento:** Debe ser posible distinguir los módulos del bus fuente y destino de los datos
- **Arbitraje:** Cada módulo puede funcionar temporalmente como maestro. Se manejan las peticiones con algún tipo de esquema de prioridad.
- **Tiempo compartido:** Sólo un módulo a la vez usa el bus, y los demás si es necesario suspenden operación

Ventajas:

- **Simplicidad:** Es la aproximación más simple para organizar el multiprocesador. La interfaz fisica y logica de cada procesador para direccionamiento, arbitraje, y para compartir el tiempo del bus son las mismas que para un monoprocesador
- **Fiabilidad:** El bus es esencialmente un medio pasivo y un fallo de un módulo no provoca un fallo del sistema
- **Flexibilidad:** Es en general sencillo conectar más procesadores al bus

Desventajas:

- **Prestaciones:** Todas las referencias a memoria, E/S pasan por el bus. El tiempo de ciclo limita la velocidad
- **Escalabilidad:** Debido al bus de tiempo compartido, cuando más procesadores se añaden al sistema se reduce el beneficio obtenido.

Memoria multipuerto

En lugar de un único bus, cada procesador y módulo de E/S se conecta a la memoria por un puerto separado.

Requiere modificaciones en la lógica de control de memoria para resolver conflictos, pero el único requerimiento del lado del procesador es que utilice write-through (luego, la interfaz física y eléctrica es idéntica, como si se tratase de una memoria con un solo puerto).

Una forma de resolver conflictos es asignar prioridades fijas a cada puerto.

Ventajas:

- **Prestaciones** al evitar problemas de control de bus
- **Seccionamiento de memoria** en zonas privadas para uno o más procesadores o modelos de E/S,

Desventajas:

- **Complejidad** del control de memoria

Unidad de control central

El controlador puede almacenar temporalmente peticiones y realizar arbitraje del bus. Cada módulo se conecta a la unidad de control central que encauza las secuencias de datos entre módulos independientes.

Las interfaces de E/S, memoria y procesador no sufren grandes cambios.

Ventajas:

- **Flexibilidad** y
- **Simplicidad** del enfoque de bus

Desventajas:

- **Complejidad:** De la unidad de control central
- **Prestaciones:** Puede convertirse en un cuello de botella.

Es común en mainframes.

Clusters

Grupo de **computadores completos** (cada computador podría funcionar por sí solo. Son los *nodos interconectados* que trabajan conjuntamente como un único recurso de cómputo, dando la ilusión de ser una sola máquina.

Memoria compartida distribuida. Los nodos no 'ven' la memoria global.

Coherencia de caché mantenida por software y no por hardware.

Cada computador tiene una capa de software intermedia que permite que el cluster funcione como un sistema único.

Beneficios:

- **Escalabilidad absoluta:** Se puede configurar clusters grandes con decenas de máquinas, pudiendo ser estas multiprocesador.
- **Escalabilidad incremental:** Es posible añadir nuevos nodos en ampliaciones sucesivas sin tener que sustituir el sistema por uno nuevo.

- **Alta disponibilidad:** Al ser cada nodo autónomo, un fallo de uno no significa la pérdida del servicio. El software suele implementar tolerancia a fallos.
- **Mejor relación precio/prestaciones:** Utiliza elementos estandarizados, por lo que el costo de armar un cluster con prestaciones equivalentes a un SMP es menor.

Se pueden conectar los nodos en LAN o WAN u otro sistema de conexión para el envío de mensajes.

Los clusters pueden compartir un mismo dispositivo de almacenamiento.

Requiere de load balancing, necesario para cumplir con la escalabilidad incremental.

Cluster vs SMP

Ambos

- Dan soporte a aplicaciones de alta demanda de recursos
- Disponibles comercialmente

SMP:

- Más fácil de administrar y configurar
- Más cercano a los sistemas de un solo procesador
    - La diferencia provincial siendo el scheduling

Cluster:

- Escalabilidad incremental y absoluta
- Disponibilidad (redundancia)

Clasificación según jerarquía de memoria

El usado en *diseño y evaluación de arquitectura de computadoras*

Utilizada cuando todos los procesadores tienen acceso a toda la memoria

El objetivo es tener sistemas NUMA.

A tener en cuenta con sistemas de memoria compartida: La coherencia, la consistencia y la sincronización.

UMA: Uniform Memory Access

Un SMP es UMA.

Igual tiempo de acceso a todas las regiones de memoria.

Igual tiempo de acceso a memoria para los diferentes procesadores.

Todos los procesadores pueden acceder a toda la memoria.

NUMA: Non-Uniform Memory Access y CC-NUMA: Cache-Coherent NUMA

Novedoso

Todos los procesadores tienen acceso a toda la memoria principal, pero el tiempo depende de la región a la que se acceda.

Motivación:

- Límite práctico en la cantidad de procesadores de SMP
- Incapacidad de los nodos de un cluster de ver toda la memoria
- Coherencia de cache por software en un cluster

Tiene características SMP manteniendo la escalabilidad del clúster.

Objetivo: Mantener la memoria transparente a todo el sistema y permitir acceso desde cualquier nodo, cada uno con su propio bus u otro sistema de conexión interna

CC-NUMA: NUMA con coherencia de caché entre todas las cachés de todos los procesadores.

Cada nodo tiene algo de memoria principal, pero desde el punto de vista de los procesadores hay un único espacio de memoria direccionable. La memoria local actúa como caché de la memoria remota.

Ventajas:

- CC-NUMA: Grandes prestaciones con mayores niveles de paralelismo que SMP sin cambios importantes de software

Desventajas:

- Si se realizan muchos acceso a memoria a nodos remotos las prestaciones se empiezan a reducir. El impacto es disminuido por el uso de múltiples niveles de caché y si el programa tiene gran localidad espacial.
- No es tan transparente como un SMP.
- Se necesitan cambios de software para adaptar el OS de un SMP (paginación, planificación, load balancing)

Multithreading (procesamiento multithebra)

Explota el paralelismo de instrucciones que pertenecen a diferentes hilos de ejecución. Aprovecha el paralelismo a nivel de thread (TLP) además del ILP.

Un **proceso** es un programa corriendo en una computadora, con sus **recursos** y camino de ejecución (**traza**). Existe el proceso switch (**conmutación de proceso**), más costoso que el thread switch.

Un **thread** tiene su propia pila y PC pero comparte código, variables globales, archivos y demás contexto con el resto de los threads del mismo proceso, por lo que los cambios de contexto (thread switch, **conmutación de hebra**) entre threads son más sencillos. Es interrumpible, ya que el procesador puede cambiar a la otra hebra.

Se ejecutan al mismo tiempo dos o más threads de un programa, permitiendo que cada uno de los threads sea planificado de la manera más conveniente para aprovechar al máximo todos los recursos.

SMT (Symmetric MultiThreading) o HyperThreading permiten la emisión en el mismo ciclo de instrucciones de dos instrucciones de diferentes ciclos. CMT (Coarse MultiThreading) solo alterna threads en un mismo cauce secuencial.

¿Cómo se comparten los recursos?

- **Recurso replicado:** Cada thread tiene su propio PC, SP, registros, etc.
- **Recurso repartido:** Un único recurso se divide de manera estática entre todos los threads y no se cambia en tiempo de ejecución. Cola de planificación, buffer de reordenamiento, etc.
- **Recurso compartido:** Se comparte de forma dinámica entre los threads según la planificación. Unidades de ejecución, registros de uso general, caché.

Los recursos compartidos mejoran el rendimiento sobre los superescalares tradicionales al permitir aprovechar recursos que de otra manera no se utilizarían, pero limita la ganancia obtenida al tener solo un procesador (nunca hay un bump de 100% por tener dos threads).

Multihebra explícito

Ejecución concurrente de instrucciones de diferentes hebras de forma explícita (indicado explícitamente por la instrucción).

Se mezclan instrucciones de diferentes hebras en el cauce compartido.

Todos los procesadores comerciales lo usan.

Multihebra implicito

Ejecución concurrente de instrucciones de varias hebras extraídas de un único programa secuencial. Se definen estrictamente por el compilador o dinámicamente por el hardware.

No hubo demasiados comerciales. La gran mayoría siempre fueron explícitos.

Anexo Clase 10: Ejemplos comerciales

Ejemplos muy interesantes de la realidad, con gráficos muy interesantes y no muy desactualizados (hay hasta el iphone 6!).