2. Incorporar en el procesador un conjunto de registros para variables globales accesible en todos los niveles. Este banco de registros no está mostrado en la ventana de registros.

Ventana de registros vs. Memoria cache

Banco de registros amplio

-Todos los datos son escalares y locales

- Acceso individual a variables

- Variables globales asignadas por el compilador

-Salvaguarda/restauración basadas en la profundidad de anidamiento

-Direccionamiento de registro

Cache

-Datos escalares locales reciente-mente usados

-Acceso a bloques de memoria

-Variables locales y globales usadas recientemente

-Salvaguarda/restauración basadas en el algoritmo de reemplazo

-Direccionamiento de memoria

Procesadores RISC - Técnicas por software

Técnicas de optimización por software

-También se puede mejorar las prestaciones en los procesadores RISC, optimizando el uso de los registros.

-Pero los lenguajes HLL no tienen referencias explícitas a los registros.

-Como la asignación de registros se hace en la compilación, el uso optimizado de registros es responsabilidad del compilador (no del programador).

-Las estrategias de optimización tratan de asignar los pocos registros a las variables más usadas o las que más tiempo permanecerán en el registro.

Una estrategia consiste en suponer que se disponen de infinitos registros (llamados “virtuales”). A cada variable del programa se le asigna un registro virtual (que son ilimitados).

-El compilador mapea (asigna) el número ilimitado de registros simbólicos a un número fijo de registros reales.

-Los registros simbólicos que no se solapan pueden compartir el registro real. Si se agotan los registros reales, algunas de las variables se asignan a posiciones de memoria.

-En la optimización se usa una técnica denominada ‘coloreado de grafos’.

Características relevantes de los procesadores RISC

-Formatos de instrucción sencillos (fijo).

-Modos de direccionamiento sencillos.

-Diseño de la Unidad de control del tipo “cableado” (sin microcódigo).

-Operaciones registró a registro. ¬ Instrucciones a memoria únicamente LOAD y STORE.

-Una instrucción por ciclo (segmentación eficiente).

-Típicamente máquinas tipo Harvard.

-Mayor tiempo/esfuerzo de compilación.

Ventajas y desventajas de los procesadores CISC

-Ventaja: El compilador es mas sencillo por disponer de un repertorio amplio de instrucciones y modos de direccionamiento

-Desventaja: las instrucciones de máquina complejas son difíciles de aprovechar por el compilador, es decir las usa poco nada.

-Desventaja: la optimización es más difícil de realizar.

-Ventaja: los programas son más pequeños, tiene menos instrucciones, lo que probablemente implique que ocupa menos memoria. Sin embargo, la memoria hoy día es muy barata, por lo que esta ventaja es muy relativa.

-El número de bits de memoria que ocupa no tiene porqué ser más pequeño al tener menos instrucciones

-Desventaja: al tener repertorio de instrucciones más amplio, los campos de código de operación son más largos y aumentan el tamaño de la instrucción.

-Las referencias a registros necesitan menos bits.

Procesadores RISC y CISC

Velocidad de ejecución

-El objetivo principal en el desarrollo de los procesadores es mejorar la velocidad de ejecución de los programas.

-Se supone que aumentar la complejidad del procesador debería mejorara su velocidad en ejecutar los programas.

-Pero:

-Los procesadores CISC casi no usan las instrucciones más complejas.

-El repertorio de instrucciones complejo exige una Unidad de control más compleja y lenta. En particular la memoria de control en la Unidad de control (microprogramada) es muy grande y “lenta”.

-Una Unidad de control más lenta aumenta el tiempo de ejecución de las instrucciones simples, es decir, penaliza las instrucciones que podrían hacerse más rápido.

-En definitiva, no está comprobado que la tendencia hacia CISC fuera la apropiada.

Comparación entre RISC y CISC

Se han realizado numerosas mediciones comparativas entre procesadores RISC y CISC.

Pero las comparaciones tienen varios problemas:

-No existe un par de máquinas RISC y CISC directamente comparables.

-No hay un conjunto de programas de prueba definitivo.

-EN los análisis, es muy difícil separar los efectos del hardware de los del compilador.

-En muchos casos, las comparaciones se realizan usando prototipos o simulaciones en un “ambiente” controlado, y pocas veces con productos comerciales.

-La mayoría de las máquinas son una mezcla de ambas.

Por otra parte es necesario definir qué parámetros se van a medir. En general, las evaluaciones pueden ser:

-Cuantitativas: básicamente comparando los tamaño de los programas y su velocidad de ejecución

-Cualitativa: revisión de soporte de lenguajes de alto nivel y uso óptimo de los recursos VLSI.

Conclusiones:

-No existe una marcada diferencia de performance entre uno y otro

-No está clara la barrera que separa uno u otro estilo.

-Muchos diseños incluyen características de ambos criterios, por ejemplo PowerPC y Pentium II.

***Clase 7 Memoria***

Subsistema de Memoria

En el diseño de la Memoria existe un compromiso entre:

-Capacidad

-Velocidad

-Costo

Una Memoria ideal sería aquella que:

-Es infinitamente grande

-El tiempo de acceso sumamente pequeño (casi 0)

-El costo relativamente bajo

Durante años se ha cumplido que:

-La velocidad del procesador aproximadamente se ha duplicado cada 18 meses (casi sin variar su precio), medido en la cantidad de instrucciones ejecutadas por segundo.

-Se ha cuadruplicado el tamaño de la memoria cada 36 meses (al mismo precio). Pero la velocidad aumenta a razón de un 10% anual.

-El desbalance entre la velocidad del procesador y la de la memoria ha generado una brecha que ha crecido a lo largo del tiempo.

En la figura anterior se pueden ver las curvas de performance de la CPU y la memoria:

-La velocidad del procesador creció según una curva (conocida como ley de Moore).

-La velocidad de la memoria creció más lentamente.

-Estas diferencias crean un desbalance entre ambos subsistemas.

-Para compensar este desbalance sin impactar fuertemente en el costo del subsistema, se implementa la Memoria como una organización jerárquica compuesta por varios tipos de dispositivos.

-En una computadora típica hay distintos tipos de memorias, desde las rápidas y caras (Ej.: registros) hasta las lentas y baratas (Ej.: discos).

-En las computadoras actuales los diferentes tipos de memorias actúan coordinadamente y no separadas.

-Esa interacción permite un comportamiento global equivalente al que tendría con una memoria única, grande y rápida.

Jerarquía de Memoria

-La forma en que se organizan coordinadamente los distintos tipos de memoria se conoce con el nombre de Jerarquía de Memoria

-La Jerarquía de memoria se puede pensar como una pirámide de múltiples capas o niveles, de diferentes tamaños y velocidades.

Jerarquía de Memoria es:

-Un método de administración del almacenamiento de la información (Memoria) estructurado en varios niveles ubicados físicamente en distintos lugares, con tecnologías, costos, tamaños y velocidades distintas.

-De esta manera los programadores “creen” disponer de cantidades casi “ilimitadas” de memoria a un costo accesible, y con velocidades cercanas a las de una memoria “ultrarápida”.

-Niveles principales de la Jerarquía de Memoria:

-Registros

-Memoria cache (RAM de muy alta velocidad)

-Memoria principal (RAM de alta velocidad)

-Memoria virtual o secundaria (medios de almacenamiento magnético/óptico)

-A medida que nos alejamos de la CPU, los niveles son más grandes, más lentos y más baratos que los niveles previos (o superiores) en la jerarquía, de ahí la forma de pirámide.

Los principales objetivos de una Jerarquía de memoria son:

-Maximizar tamaño: idealmente disponer de una “capacidad ilimitada”, equiparada al tamaño del nivel más grande.

-Optimizar velocidad: simular que se dispone de un banco de memoria “ultrarápida”, próximo a la velocidad del nivel más rápido

-Minimizar el costo total: implementar una memoria a un costo cercano al del nivel más lento.

Para que la memoria se comporte como una jerarquía (integrada) debe cumplir las siguientes propiedades:

-Inclusión: los datos almacenados en un nivel han de estar también almacenados en los niveles inferiores a él.

-Coherencia: las copias de la misma información en los distintos niveles deben contener los mismos valores.

El manejo de la jerarquía de memoria es administrado por:

-Nivel registros: el compilador. Se puede decir que el programador no interviene en la administración de este recurso porque en los lenguajes de programación no son visibles (con algunas excepciones)

-Cache: la administración se hace por hardware

-Memoria principal: la administración la pueden hacer:

-Hardware

-Sistema operativo

-Programador (archivos)

Memoria Cache

-En la Jerarquía de memoria ya se vieron anteriormente los niveles de Registros y Memoria Principal. Ahora se va a analizar el nivel de la Memoria cache.

-La memoria Caché es una memoria pequeña y muy rápida, que se ubica entre la memoria principal y la CPU.

-Puede localizarse en un chip separado o dentro de la CPU, o en ambos lugares.

-Contiene algunos sectores (bloques) de la MP.

-La información contenida en la Caché se organiza en bloques (también llamadas ranuras) de longitud fija (Ej.: 8 bytes, 16 bytes, 32 bytes, etc.).

-En los bloques de la Caché se copian algunos bloques (de idéntico tamaño) de la Memoria principal.

-La cantidad de bloques copiados depende del tamaño de la memoria Caché y del bloque.

-Cada ranura tiene asociada una etiqueta para identificar el bloque de la Memoria que tiene copiado.

-El conjunto de etiquetas forma el directorio de la cache.

Funcionamiento:

-Cuando la CPU necesita un dato, genera 1 dirección de memoria.

-La cache “intercepta” esa dirección y determina si tiene ese dato. Pueden ocurrir 2 situaciones:

-ACIERTO: si lo tiene, se lo envía a la CPU (a la velocidad de la cache)

-FALLO: Si no lo tiene, se trae el bloque que contiene esa dirección desde la memoria principal, y la cache entrega el dato requerido a la CPU.

-Si el dato que busca la CPU está en la cache, la velocidad del acceso depende del tiempo de acceso de la cache (relativamente muy corto).

-Si el dato no está en la cache, la velocidad del acceso depende del tiempo de acceso de la memoria principal (relativamente largo).

-Así, la eficiencia del uso de la cache depende de la cantidad de veces que “acierta” a la cache.

-La cantidad de veces que se acierta (la “tasa de aciertos”) no necesariamente tiene que ser proporcional al tamaño de la caché, que es miles de veces mas chico que el de la Memoria principal. La razón de esto tiene que ver con el comportamiento de los programas.

Principios en los que se basa la memoria caché

Los programas tiene un comportamiento basado en 2 principios (que son de carácter empírico).

-Principio de localidad temporal de las referencias: Es altamente probable que los elementos de memoria referenciados recientemente (datos o instrucciones) vuelvan a ser referenciados en el corto tiempo.

-Principio de localidad espacial de las referencias: Es altamente probable que los próximos elementos de memoria referenciados estén en las proximidades de los últimos referenciados.

Acceso a memoria en un procesador segmentado

En un procesador con segmentación del cauce, se dispone de un ciclo de reloj para el acceso a memoria. Ese tiempo debe ser suficiente para el acceso a la memoria caché. En caso de fallo el acceso a la memoria principal requiere de varios ciclos extra.

Tal como se vio, cuando la CPU busca un dato se pueden dar 2 situaciones:

-Acierto (se conoce como “hit”): se encuentra en la caché el dato solicitado

-Fallo (se conoce como “miss”): no se encuentra en la caché el dato solicitado

-Cuando ocurre un fallo, el bloque que contiene la palabra accedida se copia de la memoria principal a una línea de caché.

-Los fallos de caché se gestionan mediante hardware y causan que el procesador se detenga hasta que el dato esté disponible.

-Esta acción requiere un tiempo determinado.

-El tiempo para servir un fallo depende de la latencia y ancho de banda de la memoria principal.

-La latencia es el tiempo necesario para completar un acceso a memoria (depende de la memoria).

-El ancho de banda es la velocidad a la cual se puede transferir el dato, es decir, es la cantidad de información por unidad de tiempo que puede transferirse desde/hacia la memoria (depende de la velocidad del bus).

-El tiempo de acceso promedio de la CPU es el promedio del tiempo que tarda en obtener los datos buscados en memoria, compuestos por accesos a la caché y accesos a la memoria principal.

Organización de la cache

Consideraciones sobre la cache

Para el diseño de la caché hay que tener en cuenta varias consideraciones.

1. Tamaño de la memoria caché

-Debe ser suficientemente grande para contener la mayor cantidad posible de información.

-Pero no demasiado grande porque el tamaño tiene impacto en la velocidad (es decir, en el taccesoMC) y en el costo.

1. Tamaño del bloque o ranura

-El tamaño del bloque es muy importante en la tasa de aciertos.

-El bloque o ranura debe ser suficientemente grande para aprovechar al máximo las referencias cercanas. Al aumentar el tamaño mejora la tasa de aciertos, hasta un cierto tamaño del bloque. Después de eso, aumentar el tamaño no mejora la tasa de aciertos.

-Por otra parte, al aumentar el tamaño del bloque hay menos bloques de la memoria principal en la caché, lo que tiende a aumentar la tasa de fallos, y la penalización por fallos porque son más palabras a transferir entre la memoria principal y la caché.

3. Costo

-El costo crece fuertemente con el tamaño de la memoria.

-Y este costo es significativamente grande cuando la caché está incluida en el chip que contiene el procesador

1. Niveles de la caché

-La memoria caché pude ser una sola (1 nivel) o estar dividida en varias unidades (multiples niveles). Los múltiples niveles de caché por lo general tienen distintos tamaños.

1. Separación de la caché de instrucciones y operandos

-El mecanismo de acceso a las instrucciones es distinto al del acceso de los datos. Es por ello que las estrategias para obtener una alta tasa de aciertos es distinta en un caso que en el otro.

-Teniendo en cuenta esto, es posible mejorar la tasa de aciertos general si la caché se divide en una caché de instrucciones y una de datos, con distintas características.

-Por ejemplo, ambas caches pueden tener distintos tamaños y políticas de acceso y reemplazo.

Análisis cuantitativo de los fallos en la caché

-Para entender el impacto en la velocidad de ejecución de un programa, de los fallos en los accesos a la Caché, supongamos el siguiente problema.

-Se tiene un procesador con:

-Frecuencia de reloj de 2 GHz

-CPI=1 (ciclos de reloj por instrucción)

-Caches de instrucciones y datos separadas

-Cuánto tarda en ejecutar 100 instrucciones?

Análisis cuantitativo de los fallos en la caché – Conclusiones

-En el análisis anterior se ha considerado que:

-Tasas de fallo típicos de instrucciones y datos

-Los tiempos de accesos a las memorias caché se han despreciado respecto de las penalizaciones por fallo

-Las penalizaciones por fallo en los accesos a las cache de instrucciones y datos producen un tiempo de ejecución más de 10 veces superior al requerido por la CPU. Es decir, el tiempo gastado por accesos a la memoria principal (caso de fallos a accesos a las caché) es mucho mayor del propio de la CPU.

Diseño de la caché

En el diseño de la caché se deben definir:

-Organización: tamaño de la caché, cantidad y tamaño de de bloques

-Política de asignación: es decir, el tipo de función de correspondencia entre los bloques de la Memoria Principal y los bloques o ranuras de la Caché.

-Política de reemplazo: se refiere a los algoritmos para reemplazar bloques en la memoria Caché.

-Política de escritura: son los mecanismos de escritura en Memoria Principal.

Políticas de asignación

Las políticas de asignación son las funciones (de mapeo) que definen la forma en que se van a asignar los bloques de la Memoria Principal en la Memoria Caché.

Las políticas más empleadas son:

-Correspondencia totalmente asociativa: Un bloque puede almacenarse libremente en cualquier lugar de la caché.

-Correspondencia directa: Un bloque sólo puede estar almacenado en un lugar fijo de la caché.

-Correspondencia asociativa por conjuntos: Un bloque puede almacenarse en un conjunto restringido de lugares en la caché.

Política de asignación Asociativa

-La dirección de memoria está compuesta de 2 campos

-Tag: es el número de bloque en la memoria principal

-Word: es el número de palabra dentro del bloque

-El tag es comparado simultáneamente con todas las etiquetas de la caché, que identifican qué bloques de la MP están asignados a la caché. Las etiquetas de la memoria caché forman el “directorio” de la caché.

-Si la comparación da un acierto, el dato se busca en la caché. Si la comparación da una falla, el dato se trae de la memoria principal.

Política de asignación Asociativa – Conclusiones

-Un bloque de memoria principal puede colocarse en cualquier línea de la cache.

-La etiqueta identifica unívocamente un bloque de memoria.

-Todas las etiquetas de las líneas se examinan para buscar una coincidencia. Para esa búsqueda se require una memoria del tipo asociativa (es decir, una memoria de acceso por contenido CAM) para implementar el directorio.

-La implementación del directorio de la caché es compleja y costosa.

-Esta política permite una libertad absoluta para la asignación y reemplazo de los bloques.

Política de asignación Directa

-La dirección de memoria está compuesta de 3 campos:

-Tag: es el número de grupo en la memoria principal

-Line: en el número de bloque dentro del grupo

-Word: es el número de palabra dentro del bloque

-El tag es comparado con la etiqueta de la caché correspondiente al número de línea. Hay tantas líneas en la caché como bloques por grupo en la memoria principal. Las etiquetas de la memoria caché forman el “directorio” de la caché.

-Si la comparación da un acierto, el dato se busca en la caché. Si la comparación da una falla, el dato se trae de la memoria principal.

Política de asignación Directa – Conclusiones

-Un bloque de memoria principal puede colocarse en una única línea de la cache.

-Nº línea caché = Nº bloque ref. “en módulo” Nº líneas caché

-La etiqueta solo contiene el número de grupo de la memoria principal asignado a esa línea de la caché.

-Esta política de asignación es muy simple de implementar y poco costosa.

-Tiene un rendimiento acceptable, aunque a veces puede ser malo. Por ejemplo, si un programa accede a dos bloques que se corresponden a la misma línea (diferentes bloques de memoria principal) de forma repetida, las pérdidas de cache (desaciertos) serán muy grandes.

Política de asignación Asociativa por conjuntos – Conclusiones

-Un bloque de memoria principal puede colocarse en bloques determinados de la cache. La cache se divide en un número de conjuntos N (N vías, con N=2, 4, 8 ... etc.)

-Cada conjunto contiene un número de líneas o ranuras.

-Un bloque determinado corresponderá a alguna línea o ranura de un conjunto determinado.

-En general, la función asociativa por conjuntos combina lo mejor de las otras correspondencias (asociativa y directa).

Políticas de reemplazo

-Cuando hay un fallo de acceso a la memoria cache, se debe traer un bloque desde la Memoria principal y almacenar un bloque existente en la memoria cache.

-El lugar a donde va a ser ubicado el bloque a traer desde la memoria principal requiere reemplazar un bloque existente.

-Las diferentes estrategias de reemplazo de los bloques se conocen como Políticas de reemplazo de bloque.

Políticas de reemplazo - Correspondencia directa

-En la función de mapeado directo, la asignación de bloques de la memoria principal a bloques de la cache es fija, no hay elección.

-Sólo hay una posible línea para cada bloque.

-Por lo tanto si se requiere traer un nuevo bloque desde la memoria principal, indefectiblemente reeemplazará el que está usando esa ranura actualmente.

Políticas de reemplazo - Correspondencia asociativa y asociativa por conjuntos

-Para las asignaciones asociativa y asociativa por conjuntos, hay varios algoritmos de reemplazo:

1. LRU

2. FIFO

3. LFU

4. Aleatorio

-Todos los algoritmos deben implementarse en hardware (por razones de velocidad)

1.- Menos usado recientemente (LRU)

-Se reemplaza el bloque que lleva más tiempo sin utilizarse.

-Requiere controles de tiempos.

-Aprovecha la localidad temporal.

-Válido en Asociativas por conjunto, donde por cada ranura se agrega un bit de USE para identificar el usado recientemente.

2.- Primero en entrar - primero en salir (FIFO).

-Se reemplaza el bloque que entró antes en la caché.

-Requiere controles de acceso para identificar el orden en que ingresaron.

3.- Menos frecuentemente usado (LFU)

-Se sustituye aquella línea que ha experimentado menos referencias.

-Requiere controles de uso.

4.- Aleatoria

-Se sustituye una línea al azar

Políticas de escritura

Políticas de escritura

-Cuando la CPU tiene que almacenar (“escribir”) un resultado en la memoria puede hacerlo tanto en un acierto (la dirección donde se va a guardar el dato está en un bloque de la cache) como en un fallo (la dirección donde se va a guardar el dato NO está en un bloque de la cache).

-En cualquiera de las 2 situaciones, se debe evitar inconsistencia de información entre las memorias principal y cache, durante los procesos de escrituras. Es decir que, aún escribiéndose el dato en la cache, el correspondiente bloque de la memoria principal debe ser actualizado en algún momento.

-Además, a veces un módulo E/S puede tener acceso directo a la memoria principal y requerir información que fue modificada en la cache. En arquitecturas complejas (procesadores paralelo) múltiples CPU pueden tener caches individuales.

-Las políticas de escritura son distintas en aciertos que en fallos.

-Cuando la dirección de memoria a donde tiene que escribir un dato está en la memoria caché, se usan las políticas de escritura en acierto.

-Cuando la dirección de memoria a donde tiene que escribir un dato no está en la memoria caché, se usan las políticas de escritura en fallo.

Políticas de escritura en aciertos

Hay 2 estrategias:

-Write-through (Escritura inmediata)

-Se actualizan simultáneamente la posición de la caché y de la memoria principal.

-Aumenta el tráfico con la memoria principal.

-Pueden haber retrasos durante múltiples escrituras.

-Write-back (Post-escritura)

-La información sólo se actualiza en la caché y se escribe la memoria cuando se reemplaza el bloque. El bloque requiere de un bit de “sucio” para indicar cuando se lo escribió.

-Como la memoria principal se actualiza en el reemplazo, puede contener información errónea en algún momento.

Políticas de escritura en fallos

Hay 2 estrategias:

-No-write allocate

-Se escribe directamente en la memoria principal. La caché se usa solo en las lecturas. El bloque no se lleva a la memoria caché mientras no se lo tenga que leer.

-Habitual con write-through.

-Write allocate

-El bloque requerido primero se copia en la cache, y luego se escribe (en la cache).

-Habitual con write-back.

Caché de Pentium 4

Caché de Pentium 4

-2 niveles de caché: L1 y L2

-Ambas caché integradas en el chip del procesador.

-Ancho de banda de las transferencias: 48 GBytes/s.

-La arquitectura admite un tercer nivel de caché (L3) en el mismo chip (para aplicaciones en servidores).

-Cache L1: integrada por 2 cache, una para datos y otra para instrucciones

-Cache de datos

-Tamaño: 8 KB

-128 Bloques de 64 bytes

-Organización: Asociativa por conjuntos de 4 vías.

-Política de Escritura: Write-through (inmediata)

-Velocidad: acceso a los datos enteros en dos ciclos de reloj.

-Caché de instrucciones:

-Tamaño: 12 kB

-Almacena segmentos de caminos de ejecución de instrucciones decodificadas (trazas).

-Cache L2 (interna) unificada para datos e instrucciones

-Capacidad: 256 KBytes.

-Bloques de 128 bytes.

-Función de mapeo: Asociativa por conjuntos de 8 vías.

-Política de escritura: Post-escritura.

-Latencia de acceso de 7 ciclos de reloj

***Clase 8 Procesadores Superescalares***

Evolución de la segmentación

Primera generación – ejecución secuencial

-La primer generación de microprocesadores se caracterizó por un proceso de ejecución completamente secuencial de instrucciones.

-No empezaba una nueva instrucción hasta que se completaba la corriente.

-Ejemplos: i4004,i8008/80, MC6800, MCS6502, Z80, F8 …

Segunda generación – ejecución segmentada

-En la segunda generación se empezó a segmentar el cauce, aunque en forma limitada y en muy pocas etapas.

-Nuevas instrucciones podían iniciarse mientras otras estaban en proceso.

-Ejemplos: MC68000, i8086/286, Z8000

Tercera generación – ejecución segmentada aumentada

-En la tercera generación la segmentación se profundizó, aumentando significativamente la cantidad de etapas.

-Nuevas instrucciones podían iniciarse mientras otras estaban en proceso.

-Ejemplos: MC68020, i80386, R2000/3000

Limitaciones de procesadores segmentados

-En general, un procesador de k etapas tiene una productividad teórica máxima igual a k

-productividad teórica máxima = k

-Pero esa productividad tiene varias limitaciones:

-La máxima capacidad teórica es 1 instrucción por ciclo (CPI o IPC =1). Es decir, solo puede completar 1 instrucción por ciclo de reloj.

-Hay 1 solo pipeline para los diferentes tipos (de datos).

-Los atascos en el cauce producen burbujas innecesarias que reducen la productividad.

Rendimiento de procesadores escalares

-El rendimiento de un procesador escalar segmentado depende de:

-IPC: número de instrucciones por ciclo

-f: Frecuencia del reloj (T= 1 / f)

-k: cantidad de etapas del cauce

-El tiempo de ejecución es igual a: CPU time= N° instr. X IPC x T (donde: T= tiempo de ciclo)

-Para estos procesadores, en el mejor de los casos IPC=1, y solo se puede mejorar este tiempo si se aumenta f.

Limitaciones en procesadores escalares segmentados

-Las limitaciones en el rendimiento de un procesador escalar segmentado se originan en 2 aspectos:

1. El IPC es igual a 1, en el mejor de los casos.

2. Los posibles conflictos que pueden producir atascos en el cauce debidos a:

-Dependencias de datos: hay 3 tipos de dependencias de datos: Verdadera, de salida, y antidependencia.

-Penalizaciones en saltos.

-Conflictos por uso de recursos

Procesadores supersegmentados y superescalares

1) Procesadores supersegmentados

En la ejecución supersegmentada de instrucciones cada ciclo se divide en fracciones más chicas, en las que se inician nuevas instrucciones. Por ejemplo, un procesador supersegmentado de grado 2 acepta una nueva instrucción en cada semiciclo de reloj.

-La ejecución supersegmentada consiste en subdividir cada segmento en partes más pequeñas.

-Como muchas operaciones no necesitan todo un ciclo de reloj, se puede hacer más de una tarea en ese ciclo, subdividiendo el ciclo de reloj en sub-intervalos, lo que es equivalente a usar una mayor frecuencia de reloj (menor ciclo de reloj).

-El tiempo para las instrucciones individuales no varía, pero aumenta el grado del paralelismo de la máquina temporal.

-El pipeline se hace más profundo, pero está limitado por la tecnología (es decir, por la frecuencia máxima).

2) Procesadores superescalares

En la ejecución superescalar de instrucciones, en cada ciclo se inician más de 1 instrucción. Por ejemplo, un procesador superescalar de grado 2 inicia 2 nuevas instrucciones en cada ciclo de reloj.

-La aproximación superescalar se basa en poder ejecutar varias instrucciones en diferentes cauces de manera independiente y concurrente.

-Algunos ejemplos de procesadores superescalares son: MC68040, i80486, MC88110, i80860,PA-RISC, Sparc, R6000.

-Se pueden llevar a cabo (completar) 2 o más instancias de cada etapa de una instrucción simultáneamente.

-Para poder iniciar/ejecutar 2 o más instrucciones simultáneamente, se requiere la duplicación de algunas o todas las partes de la CPU/ALU, por ejemplo:

-Captación de múltiples instrucciones al mismo tiempo.

-Ejecución (sumas y multiplicaciones) simultánea.

-Ejecución de carga/almacenamiento, mientras se lleva a cabo una operación en ALU.

-El grado de paralelismo y, por tanto, la aceleración de la máquina aumentan, ya que se ejecutan más instrucciones en paralelo.

La aproximación superescalar presenta paralelismo de máquina de 2 tipos: temporal y espacial.

Esquema básico

Un procesador superescalar dispone, básicamente, de multiples unidades funcionales, cada una implementada como un cauce segmentado, que admite la ejecución paralela de varias instrucciones.

Los conflictos en procesadores superescalares son los mismos que en procesadores segmentados.

Paralelismo

-El paralelismo es la capacidad para ejecutar varias tareas en el mismo intervalo de tiempo (en “paralelo”).

-Hay 2 tipos de paralelismo:

-Paralelismo a nivel de instrucciones: se refiere a las posibilidades que tiene un programa para ejectar instrucciones en paralelo

-Paralelismo a nivel de máquina: es la capacidad que tiene una máquina para encontrar y ejecutar tareas en paralelo.

Paralelismo a nivel de programa

-El grado de paralelismo de un programa se mide usando un parámetro conocido como IPL.

-El IPL es la cantidad, en promedio, de instrucciones que pueden ejecutarse en paralelo.

-Las instrucciones pueden ejecutarse en paralelo si son independientes.

Paralelismo a nivel de máquina

-El grado de paralelismo de una máquina se mide usando un parámetro conocido como MPL, que es la capacidad que tiene una máquina para aprovechar el paralelismo de un programa.

-El MPL es función de:

-Número de instrucciones captadas por ciclo.

-Número de unidades funcionales.

-Mecanismos para localizar y ejecutar instrucciones independientes.

Procesadores superescalares

Tratamiento de las instrucciones

-En los procesadores superescalares el objetivo fundamental es localizar instrucciones que puedan ser introducidas al pipeline y ejecutadas. En este proceso de detección, es necesario distinguir:

-El orden en que se captan las instrucciones.

-El orden en que se ejecutan las instrucciones.

-El orden en que las instrucciones actualizan los registros y las posiciones de memoria.

-Cuanto más sofisticado es el procesador, menos restricciones impone a estos ordenamientos. Incluso puede alterar cualquier ordenamiento, respecto del estrictamente secuencial. La única condición es que el resultado debe ser correcto.

Políticas de emisión

Tratamiento de las instrucciones

-Las políticas de emisión de instrucciones son los protocolos usados para el envío de las instrucciones a las unidades funcionales, es decir, definen la forma en que se captan, ejecutan y finalizan (escriben los resultados) las instrucciones.

-En función del orden en que se captan, ejecutan y terminan las instrucciones, existen 3 políticas de emisión de instrucciones:

-Emisión y finalización ordenada

-Emisión ordenada y finalización desordenada

-Emisión y finalización desordenada

Políticas de emisión de instrucciones

-Para analizar las políticas de emisión de instrucciones, vamos a considerar un procesador superescalar con un cauce de 3 segmentos compuesto por:

-Búsqueda de instrucción y decodificación: IF+D

-Ejecución: Ex

-Escritura de resultados: W

Además, el procesador dispone de:

-2 unidades de decodificación D1 y D2.

-3 unidades de ejecución E1, E2 y E3.

-2 copias de la etapa de escritura W1 y W2.

-Políticas de ejecución de instrucciones

-Como tiene 2 unidades de búsqueda y decodificación D1 y D2, 2 instrucciones pueden ser leídas y decodificadas al mismo tiempo.

-Solo se puede leer un nuevo par de instrucciones cuando se liberan ambas unidades D1 y D2.

-Para garantizar la correcta ejecución, los conflictos se resuelven parando la instrucción (“stall”) hasta que el conflicto desaparece.

Se va a analizar el comportamiento del procesador superescalar para una secuencia de 6 instrucciones con las siguientes características:

-I1 tarda dos ciclos en ejecutarse

-I2 tarda 1 ciclo

-I3 e I4 tardan 1 ciclo y usan la misma unidad funcional (E3)

-I5 e I6 tardan 1 ciclo y usan la misma unidad funcional (E2)

-I5 depende del valor producido por I4

Emisión y finalización ordenada

En esta política, las instrucciones se emiten exactamente como están en el programa (como si fuera “secuencial”), y se escriben los resultados en el mismo orden.

-En el ejemplo anterior se puede ver que:

-Las instrucciones se captan y decodifican de a 2.

Las instrucciones I5 e I6 deben esperar hasta que se desocupen ambos decodificadores D1 y D2. ¬ Dado que el orden en que se escriben los resultados debe ser el mismo en que se captan, I2 no puede ser escrita hasta que I1 no se haya completado y esté lista para guardarse (W1 o W2).

-I4 tiene que esperar que I3 libere E3. Idem I5 e I6.

-I5 tiene que esperar que se ejecute I4.

-Se requieren 8 ciclos para ejecutar las 6 instrucciones.

Emisión ordenada y finalización desordenada

En esta política, las instrucciones se emiten exactamente como está en el programa, pero los resultados se escriben en “cualquier orden”

-Las instrucciones se captan y decodifican de a 2. Por eso, las instrucciones I5 e I6 deben esperar hasta que se desocupen ambos decodificadores D1 y D2.

-Dado que el orden en que se escriben los resultados puede ser distinto del que se captan, I2 puede ser escrita antes que I1 se haya completado y esté lista para guardarse.

-Al permitir la finalización desordenada, I3 puede ser ejecutada anticipadamente, ganándose 1 ciclo.

-Ahora se requieren 7 ciclos para ejecutar las 6 instrucciones.

Emisión ordenada y finalización desordenada – Conclusiones

-Las instrucciones entran a las unidades de ejecución a medida que se van decodificando y hay unidades de ejecución disponibles.

-Las emisiones de las instrucciones están limitadas por los posibles conflictos de recursos, dependencias de datos y de saltos.

-Al permitir la finalización desordenada, aparecen nuevas fuentes de conflictos por dependencia de datos.

-En el siguiente ejemplo se pueden apreciar los nuevos conflictos que aparecen por la política de finalización desordenada.

Emisión ordenada y finalización desordenada – Conflictos

Consideremos el siguiente programa:

R3:= R2 op R5 (I1)

R4:= R3 + 1 (I2)

R3:= R5 + 1 (I3)

R7:= R3 op R4 (I4)

Los riesgos que aparecen en el ejemplo anterior son:

1) RAW (dependencia verdadera) I1 escribe el resultado en R3, que es un operando de la I2.

2) WAW (dependencia de salida) I1 escribe el resultado en R3 igual que la I3. Si I3 se escribe antes que I1, entonces la I4 usará un valor incorrecto de R3.

3) WAR (antidependencia) I3 escribe en R3. Pero I2 necesita el valor previo de R3. I3 no puede escribir R3 hasta que I2 haya leído el valor.

-La finalización desordenada requiere detectar estas posibles nuevas dependencias, y evitar que se ejecuten instrucciones en forma desordenada que puedan alterar el resultado del programa.

-Por tal motivo, la lógica de emisión de instrucciones es más compleja que la empleada en máquinas con política de finalización ordenada.

-Con la política de emisión ordenada, el procesador solo puede decodificar instrucciones hasta el punto de dependencia o conflicto. Es decir, no analiza si hay nuevas instrucciones que pueden emitirse sin producir conflicto.

Emisión y finalización desordenada

-Para permitir que el procesador busque más allá del punto de conflicto por nuevas instrucciones, se require usar una política de emisión desordenada.

-La política de emisión desordenada require separar, mediante un buffer, llamado Ventana de instrucciones, las unidades de decodificación y las de ejecución.

-Cada instrucción decodificada se coloca en un buffer intermedio, desde donde se emiten las que no presentan conflictos. Mientras no se llene el buffer, nuevas instrucciones son decodificadas y colocadas para su emisión a las unidades de ejecución. Las instrucciones emitidas no deben tener conflictos por recursos ni por dependencias de datos.

-La Ventana de instrucciones no es una etapa del cauce, es un buffer que retiene la información necesaria para emitirla.

-En el siguiente ejemplo se muestra el esquema funcional de una máquina con emisión y finalizacion desordenada.

-Se intercala una buffer (ventana de instrucciones) entra las unidades de decodificación y las de ejecución.

-Las instrucciones se emiten y los resultados se guardan en forma desordenada.

Ejemplo:

-Las instrucciones se captan y decodifican de a 2, pero ahora la ventana de instrucciones permite, en el ciclo 3, decodificar las instrucciones I5 e I6.

-Como I6 no tiene dependencias y está lista para ejecutarse, puede ser emitida anticipadamente, incluso antes de I5. La ventana le permite “ver” al procesador más adelante de las instrucciones en conflicto, pudiendo detectar nuevas instrucciones (la I6) que no genera conflicto.

-Ahora se requieren 6 ciclos para ejecutar las 6 instrucciones.

Renombrado de registros

-Al permitir finalización y/o emisión desordenada surgen los nuevos problemas de dependencia de datos (WAW y WAR).

-Las dependencias de salida y antidependencias surgen porque los valores de los registros pueden no reflejar la secuencia de valores dictada por el flujo del programa (en realidad es un conflicto de almacenamiento: hay instrucciones que compiten por un mismo registro).

-Tal como se vio en clases anteriores, estos nuevos conflictos pueden resolverse con detenciones en la etapa del cauce, pero esa solución produce pérdidas de tiempo que pueden llegar a ser inaceptables.

Supongamos la siguiente secuencia de instrucciones de un programa:

R3:= R3 op R5 (I1)

R4:= R3 + 1 (I2)

R3:= R5 + 1 (I3)

R7:= R3 op R4 (I4)

En la secuencia anterior se pueden observar las 3 dependencias:

RAW entre la I1 y la I2

WAR entre la I2 y la I3

WAW entre la I1 y la I3

-Dado que en definitiva, es un conflicto de recursos (los registros), se puede resolver duplicando los recursos.

-Esta técnica se conoce como Renombrado de Registros.

-Los registros en las instrucciones del programa, son registros “lógicos”. Los registros físicos (es decir, los reales) los asigna el procesador.

-Cada vez que se guarda un dato en un registro el procesador le asigna un nuevo registro físico. Las instrucciones siguientes que referencien a ese registro, deben renombrarse para referenciar el registro físico que tiene el dato correcto.

Consideremos el ejemplo anterior (donde los nombres de registros son referencias “lógicas”): R3:= R3 op R5 (I1)

R4:= R3 + 1 (I2)

R3:= R5 + 1 (I3)

R7:= R3 op R4 (I4)

El procesador va asignando nuevos registros en cada escritura, quedando las siguientes referencias “físicas”:

R3b:= R3a op R5a (I1)

R4a:= R3b + 1 (I2)

R3c:= R5a + 1 (I3)

R7a:= R3c op R4a (I4)

-En el ejemplo anterior, en la instrucción I1 se asigna un nuevo registro R3b distinto del usado anteriormente R3a.

-La I2 debe renombrarse para usar R3b (y no R3a), y crea R4a (que no existía previamente).

-La I3 asigna un nuevo registro R3c, distinto del usado anteriormente (R3b).

-La I4 debe renombrarse para usar R3c (y no R3b), y crea R7a (que no existía previamente).

-De esta manera se eliminan las dependencias WAW y WAR (antidependencia y dependencia de salida) quedando únicamente la dependencia verdadera RAW.

Ejecución superescalar

-Las instrucciones son captadas a partir del programa estático, y decodificadas para la detección temprana de dependencias y predicción de saltos.

-Las instrucciones, todavía ordenadas, son enviadas a la ventana de ejecución, donde se las redistribuye en función de las dependencias verdaderas, esto es, las que están en condiciones de ser ejecutas o las que necesitan disponer de datos de otras instrucciones.

-Las instrucciones son ejecutadas en un orden basado en la disponibilidad de unidades de ejecución y en las dependencias verdaderas.

-Las instrucciones ejecutadas son nuevamente reordenadas secuencialmente, y los resultados guardados en el orden que corresponde.

-Este último paso es para:

-Reordenar el almacenamiento de acuerdo al programa original

Eliminar todas las ejecuciones inservibles (aquellas que se hicieron por predicción de salto o ejecución especulativa) y que se deben descartar.

-Mejorar la respuesta a las interrupciones.

-De esta manera, las instrucciones ejecutadas producen resultados almacenados temporalmente, hasta que se graban los resultados válidos en los registros reales.

En resumen, la ejecución superscalar involucra:

-Diferentes 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.

Estudios sobre ejecuciones superescalares

En el estudio anterior, se han comparado las mejoras obtenidas (o “speedup” en el eje vertical) para los siguientes casos:

-2 situaciones: sin (gráfico de la izquierda), y con (gráfico de la derecha), renombrado de registros .

-3 tamaños de ventanas de instrucciones: 8, 16 y 32 instrucciones.

-4 tipos de máquinas:

-Máquina base: sin duplicación de unidades funcionales

-Máquina base + ld/st: duplicando las unidades de acceso a la memoria

-Máquina base + ALU: duplicando las ALU

-Máquina base + ld/st + ALU: duplicando las unidades de acceso a la memoria y las ALU

Conclusiones

-¬ En la máquina sin renombrado de registros se observa que:

-Duplicar las unidades funcionales de acceso a la memoria y las ALU (barras “both”) tiene poco impacto en la aceleración obtenida (el eje vertical pasa de 2 a 2.5).

-Disponer de una Ventana de instrucciones mas grande (8 a 32) no modifica significativamente los resultados.

-En la máquina con renombrado de registros se observa que:

-Duplicar las ALU y también las unidades de acceso a la memoria mejoran fuertemente la aceleración obtenida (el je vertical pasa de 2.5 a 4).

-El tamaño de la Ventana de instrucciones tiene un impacto significativo cuando cambia de 8 a 16 instrucciones.

-La gran diferencia entre los resultados con y sin renombrado de registros se debe a un elemento fundamental en el proceso de ejecución de instrucciones:

- En las máquinas sin renombrado de registros, todos los conflictos por las dependencias de datos (RAW, WAR, WAW) se resuelven mediante detenciones del cauce (“stall”).

-En las máquinas con renombrado de registro se eliminan los conflictos por dependencias de salida y antidependencia (WAW, WAR) quedando únicamente la dependencia directa o verdadera (RAW).

Ejemplo procesador superescalar

Un esquema de un procesador superescalar se muestra en la figura siguiente.

La unidad de búsqueda de instrucciones (F) busca y carga en un buffer (cola de instrucciones), las instrucciones leídas.

La unidad de decodificación/emisión puede tomar 2 instrucciones de la cola y decodificarlas.

Hay 2 unidades de ejecución (ALU), una de punto fijo y la otra de punto flotante.

Si las 2 instrucciones que capta, una es de enteros y la otra de coma flotante (segmentada), y no existen riesgos, entonces pueden emitirse y ejecutarse a la vez

Es responsabilidad del compilador generar el código apropiado para el máximo aprovechamiento del procesador supersescalar.

Interrupciones en procesadores superescalares

-En los procesadores superescalares el tratamiento de interrupciones internas o excepciones (rupturas por ejecución de instrucciones) y externas (rupturas por eventos externos) es más complicado que en los procesadores convencionales.

-La razón es que en el momento en que se produce una interrupción hay varias instrucciones en ejecución (y además, desordenadas).

-El problema que se plantea es si producida una excepción en una instrucción dada I2, en qué estado quedaron las instrucciones anteriores (por ejemplo I1) y posteriores (por ejemplo I3) dado que con la política de ejecución desordenada pudieron haber quedado en distintos estados.

Interrupciones internas o excepciones

Existen 2 estrategias para el tratamiento de excepciones:

Excepciones precisas: las instrucciones que preceden a la que produjo la excepción se completan, y las que le suceden se reinician. Es decir, el comportamiento es “idéntico” al que tendría la misma computadora no segmentada

Excepciones imprecisas: no se respetan las condiciones exactas de la máquina no segmentada.

-Para garantizar un estado consistente (preciso) se requiere:

-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

-Para que las interrupciones sean precisas los resultados se deben almacenar en el orden en que aparecen las instrucciones.

Para ello, se requiere retardar las escrituras de los resultados de tal manera que queden en el orden en el que estaba el programa secuencial.

Interrupciones externas

-Las interrupciones externas pueden tener distinta grado de “precisión”, dado que no necesariamente tienen que ver con las instrucciones en ejecución (por ejemplo operaciones de I/O).

-Interrupciones imprecisas: se completa lo que estaba en ejecución.

-Interrupciones “algo” imprecisas: el software (la rutina de servicio de la interrupción) resuelve algunas inconsistencias.

-Interrupciones precisas: solo se completan las instrucciones previas a la interrupción, para lo cual se requiere implementar la escritura con buffer de salida. La unidad de emisión deja de emitir, se cancela la cola, todas las instrucciones pendientes se completan, y comienza el procesamiento de la interrupción.

Conclusiones

En general se tiene:

En los sistemas con finalización desordenada:

-Excepciones imprecisas

-Mayor performance

-En los sistemas con finalización ordenada (es decir, con buffer de reordenamiento):

-Excepciones precisas

-Menor performance