Von Neumann

Componentes principales:

- CPU= UC + ALU
- Entrada/Salida
- Memoria (Instrucción, Datos); Contenido se direcciona indicando su posición sin considerar el tipo.

Repertorio de instrucciones: conjunto de instrucciones que se realizan en una CPU (Código máquina; Binario). Representado por un conj. de códigos de ensamblaje:

De operaciones: ADD, SUB, LOAD

De operandos: ADD BX, PEPE

Elementos de una instrucción: código de operación (*operación)*, referencia a operando fuentes (*cargar)*, referencia a operando resultado (*guardar)*, referencia a la siguiente instrucción.

Tipos de instrucciones:

- Procesamiento de datos (inst aritmético – lógicas)
- Almacenamiento de datos (inst de memoria)
- Transferencia de datos (inst de E/S)
- Control (ins de testeo y flujo de programa)

Dónde están almacenados los operandos?

- Memoria principal
- Registro de la CPU
- Dispositivo de E/S

Cuántas direcciones?

(4) Fuente, fuente, resultado, sig

(3) Programa secuencial → fuente, fuente, resultado

(2) Fuente, resultado → guardo res en un op (pierdo un op pero uso menos direcciones)

(1) Acumulador

(0) Máquina de pila → push, pop

Más direcciones por instrucción: *instrucciones* más complejas pero menos, más registros (operaciones entre registros son más rápidas)

Menos direcciones por instrucción: instrucciones menos complejas pero más, la captación/ejecución de las instrucciones es más rápida.

Tipos de operandos:

- Direcciones (indicar celda de memoria - es nº sin signo)
- Números (flotante o fijo?)
- Caracteres (ASCII -7bits, EBCDIC -8bits …)
- Datos lógicos (1 o 0)

Orden de los bytes:

Big endian: el byte más significativo en la dirección con valor numérico más bajo (Mayoría de los RISC)

Little endian: el byte menos significativo en la dirección con valor numérico más bajo

Tipo de operaciones

- **Transferencia de datos:** Especificar: ubicación fuente y destino; tamaño; modo de direccionamiento. Diferentes movimientos → diferentes instrucciones (reg-reg, reg-mem, mem-reg). O una instrucción y diferentes direcciones (MOV destino, fuente).

- **Aritméticas:** básicas: add (puedo hacer TODO), sub, mul, div; otras: inc, dec, neg, abs, shift left/right.

- **Lógicas – Conversión:** operaciones que manipulan bits individualmente. Booleanas: AND, OR, XOR, NOT. Otras: rotate left/right; Conversión de formatos.

- **Entrada/Salida:** IN o OUT: pocas pero específicas. Se pueden realizar utilizando: instrucciones de movimiento de datos MOVE o a través de un controlador aparte DMA.

- **Control de flujo:** modifican el valor contenido en el registro PC. Salto Incondicional, Salto Condicional, JZ, Salto con retorno o llamada a subrutina (CALL subrut) (retorno con RET)

- **Modos de direccionamiento:**

Inmediato: el operando está presente en la instrucción. La ventaja es que una vez captada la instrucción no debo acceder a memoria. La desventaja es que el tamaño del número está restringido a la longitud del campo de direcciones.
Directo: el campo de direcciones contiene la dirección efectiva del operando. Solo requiere una referencia a memoria. Proporciona un espacio de direcciones restringido.
Indirecto: el campo de direcciones referencia la dirección de una palabra de memoria que contiene la dirección completa del operando. Tiene dos accesos a memoria.
De registros: el campo de direcciones referencia a un registro.
Indirecto con registro: el campo de direcciones hace referencia a una posición de memoria o a un registro.
Con desplazamiento: requiere que las instrucciones tengan dos campos de direcciones, al menos uno de ellos explícito.
De pila: es implícito, se opera directamente sobre la cabecera de la pila.

  - Inmediato: al registro le sumo el valor que doy y lo pongo en otro registro.
  - Directo de memoria o Absoluto: doy la dirección de memoria donde está el operando.
  - Directo de registro: doy nombre del registro.
  - Indirecto de memoria (en desuso): pongo dirección de memoria y traigo la nueva dirección del operando.
  - Indirecto con registro: soy ubicación de registr5o que contiene la dirección de memoria.
  - Indirecto con Desplazamiento: basado, indexado o relativo al PC: PILA o relativo al SP.

Subrutinas: programa auto-contenido, puede invocarse desde cualquier punto de un programa mediante instrucción CALL. Brinda economía (código usado varias veces) y modularidad. Requiere pasaje de argumentos (parámetros) por valor o por referencia.
Pasaje de argumentos a subrutinas:

Vía registros: El número de registros es la principal limitación. Es importante documentar qué registros se usan.

Vía memoria Se usa un área definida de memoria (RAM). Difícil de estandarizar.

Vía Pila: El más usado. El verdadero "pasaje de parámetros". Independiente de memoria y registros. Debemos comprenderla porque la pila es usada por el usuario y por el sistema. El operando está en la cabeza de la pila. Se requiere SP (contiene la dirección de la cabeza de la pila). Operaciones sobre la pila PUSH y POP.

INTERRUPCIONES

Mecanismo mediante el cual se puede interrumpir el procesamiento normal de la CPU. Pueden ser de origen interno o externo a la CPU.

¿Porqué Interrumpir?

Por resultado de una ejecución de una instrucción (ej división por cero)

Por un temporizador interno del procesador (Permite al S.O. realizar ciertas funciones de manera regular).

Por una operación de E/S (ej para indicar la finalización normal de una operación)

Por un fallo de hardware (ej error de paridad en la memoria, pérdida de energía)

¿Qué hacer si interrumpen?

En casi todos los casos, implica transferir el control a otro programa (el GESTOR) que: salve el estado del procesador; corrija la causa que ocasionó la interrupción; restaure el estado original; retorne a la ejecución normal del programa.

Jerarquía de interrupciones

Si hay múltiples fuentes que pueden solicitar interrupción se establece cuales son mas importantes. Se consideran:

No Enmascarables: las que NO pueden ignorarse (eventos peligrosos o de alta prioridad).

Enmascarables: pueden ser ignoradas (Con instrucciones podemos inhibir la posible solicitud)

Interrupciones por hardware

Son las generadas por dispositivos de E/S. El sistema de cómputo tiene que manejar estos eventos externos "no planeados". No están relacionadas con el proceso en ejecución en ese momento. Son conocidas como *interrupt request*.

Traps/excepciones: Interrupciones por hardware creadas por el procesador en respuesta a ciertos eventos como: Condiciones excepcionales (overflow en ALU de punto Flotante); Falla de programa (tratar de ejecutar una instrucción no definida); Fallas de hardware (error de paridad de memoria); Accesos no alineados ó a zonas de memoria protegidos

Interrupciones por software

Muchos procesadores tienen instrucciones explícitas que afectan al procesador de la misma manera que las interrupciones por hardware. Generalmente usadas para hacer llamadas a funciones del SO. Esta característica permite que las subrutinas del sistema se carguen en cualquier lugar. No requieren conocer la dirección de la rutina en tiempo de ejecución.

Hay sistemas que no permiten hacer una llamada directa a una dirección de la función del SO, por estar en una zona reservada.

¿Qué pasa si no tuviera las int. por software? Debería escribir todas las funciones que necesito ó Al cargar un programa habría que "mirar" todas

las llamadas a funciones del BIOS y SO y reemplazar en el código las direcciones de todas estas funciones invocadas.

Interrupciones múltiples:

Interrupciones inhabilitadas: El procesador debe ignorar la señal de petición de interrupción si se produce una interrupción en ese momento. Si se hubiera generado una interrupción se mantiene pendiente y se examinará luego una vez que se hayan habilitado nuevamente. Ocurre una interrupción, se inhabilitan, se gestiona la misma y luego se habilitan otra vez. Por lo tanto las interrupciones se manejan en un orden secuencial estricto. Definir prioridades: Una interrupción de prioridad más alta puede interrumpir a un gestor de interrupción de prioridad menor. Cuando se ha gestionado la interrupción de prioridad más alta, el procesador vuelve a las interrupciones previas (de menor prioridad). Terminadas todas las rutinas de gestión de interrupciones se retoma el programa del usuario.

Reconocimiento de interrupciones

Interrupciones multinivel: Cada dispositivo que puede provocar interrupción tiene una entrada física de interrupción conectada a la CPU. Es muy sencillo, pero muy caro.

Línea de interrupción única: Una sola entrada física de pedido de interrupción a la que están conectados todos los dispositivos. Se debe "preguntar" a cada dispositivo si ha producido el pedido de interrupción (técnica Polling/encuesta).

Interrupciones vectorizadas: El dispositivo que quiere interrumpir además de la señal de pedido de interrupción, debe colocar en el bus de datos un identificador (vector). Lo coloca el periférico directamente ó Controlador de Interrupciones (que se ocupa de todo). Es el nexo entre tipo de interrupción (0...255) y el procedimiento designado para atenderla. Cada entrada es una doble palabra (4 bytes). Dirección del procedimiento que brinda el servicio. Ej: 0000yyyy, donde yyyy es la dirección lógica/física. Vectores preasignados

- Tipo 0 – finaliza ejecución de programa
- Tipo 3 – punto de parada para depuración/seguimiento
- Tipo 6 – lectura de entrada std. Requiere el uso de BX.
- Tipo 7 – escritura de salida std. Requiere BX y AL.

Controlador de Interrupciones **PIC**

Registros internos

EOI: para comandos: Para fin de int escribir 20H

IMR: máscara de int enmascara con 1

IRR: petición de int Indica con bit en 1

ISR: int en servicio Indica con bit en 1

INT0...INT7 c/u con su vector

ENTRADA / SALIDA

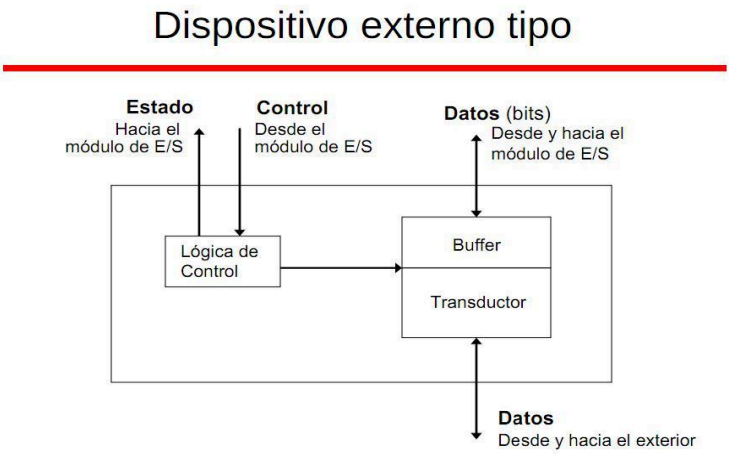
Problemas de Entrada/Salida

Gran variedad de periféricos con varios métodos de operación. Diferentes cantidades de datos, velocidades, formatos y tamaño. Todos más lentos que la CPU y la RAM.

Módulo de E/S

Realiza la interfaz entre el procesador y la memoria (bus) y los periféricos. Pueden manejar varios.

Procesador < controladores <bus> interfaz <puerto> periférico> usuario
Memoria ppal (datos, control, estado)
S.O + Drivers



Características de un puerto

- Interfase entre el periférico y el módulo de E/S
- Señales:
  - **Señal de Control**: función a realizar (INPUT ó READ, OUTPUT ó WRITE)
  - **Señal de Estado**: READY/NOT READY
  - **Control lógico**: manejo de direccionamiento
  - **Transductor**: conversión del datos
  - **Buffer**: adaptación (1, 8 o 16 bits)

Funciones de un módulo de E/S

- Control y temporización de varios periféricos
- Interpretar las órdenes de CPU y transmitirlas al periférico
- Comunicación con la CPU (registros), Memoria y periféricos
- Controlar las transferencias de datos entre CPU y el periférico (convertir formatos, adaptar velocidades)
- Informar a la CPU del estado del periférico
- Almacenamiento temporal (buffering) de datos
- Detección de errores

Capacidades de un módulo de E/S

- Ocultar las propiedades del dispositivo a la CPU (formatos, etc.)
- Ocuparse de varios dispositivos.
- Controlar las funciones del dispositivo:
  - Canales de E/S ó procesador de E/S ALTO NIVEL
  - Controlador de E/S ó controlador de dispositivo NIVEL BAJO

Operación de Entrada ó Salida

Requiere:

- **Direccionamiento**
  - E/S mapeada en memoria: dispositivos de E/S y memoria comparten un único espacio de direcciones. No hay órdenes específicas.

- E/S aislada: Espacios de direcciones separados. Necesidad de líneas especiales de E/S y de memoria. Órdenes específicas para E/S.

• **Transferencia de información**: Lectura ó escritura
• **Gestión de la transferencia**: Mecanismos de sincronización y control de la transferencia de datos

Técnicas de gestión de E/S

Programada

Intercambio de datos entre la CPU y el módulo.
Se produce bajo el control directo y continuo del programa que solicita la operación de E/S.
La CPU tiene control directo sobre la operación de E/S: Comprobación del estado. Envío de comandos de lectura/escritura. Transferencia de datos
Detalles
La CPU solicita la operación de E/S al módulo. Este realiza la operación y activa los bits de estado. La CPU comprueba el estado de esos bits, hasta que detecta que la operación fue completada.
Órdenes (comandos)
La CPU emite una dirección (especifica el módulo y el dispositivo si hay más de uno) La CPU da una orden
• Control: indica al módulo qué hacer
• Test: comprueba el estado (¿hubo algún error?) del módulo y sus periféricos
• Lectura/Escritura

La CPU espera que el módulo termine la operación por lo que permanece ociosa durante un tiempo (no deseable).

Con Interrupciones

La CPU puede seguir procesando. El módulo envía un pedido de interrupción a la CPU cuando está listo nuevamente.
¿Cómo darse cuenta quién interrumpe?
Diferentes líneas para cada módulo
- PC
- Limita el número de dispositivos

Consulta software (Poll)
- La CPU consulta quien fue el demandante. Lento.

Conexión en cadena
- La línea de pedido es compartida. Una vez enviada la confirmación de parte de la CPU el módulo responderá colocando un vector (palabra), en el bus, que lo identifica. La CPU emplea el vector como puntero para acceder a la rutina de servicio.

Interrupciones múltiples

Todas las líneas de interrupción tienen un orden de prioridad. Si existe un maestro del bus, solo él puede interrumpir.

Qué hace la CPU?
Envía una orden de lectura. El módulo obtiene los datos del periférico. **La CPU chequea si hay pedidos de interrupciones al final de cada ciclo de instrucción.** El módulo E/S emite un pedido de interrupción a la CPU. La CPU detecta el pedido, guarda el contexto, interrumpe el proceso y realiza la gestión de la interrupción. La CPU solicita los datos. El módulo E/S transfiere los datos.

Las operaciones de E/S mediante interrupciones son más efectivas que las programadas. Pero ambas necesitan la intervención directa de la CPU. La velocidad

de transferencia es limitada. La CPU permanece ocupada mucho tiempo durante la operación.
Con *Periférico lento* interrupciones reducen el tiempo. Con rápido, es lo mismo.

Acceso directo a memoria (DMA)

El controlador de DMA es un dispositivo capaz de controlar una transferencia de datos entre un periférico y memoria sin intervención de la CPU. El Controlador de DMA debe actuar como maestro del bus durante la transferencia DMA y debe ser capaz de:

- Solicitar el uso del bus
- Especificar la dirección de memoria sobre la que se realiza la transferencia
- Generar las señales de control del bus (Tipo de operación (lectura/escritura) y Señales de sincronización de la transferencia)

Etapas

Inicialización de la transferencia

La CPU debe enviar al interfaz del periférico y al DMAC los parámetros de la transferencia
Inicialización del interfaz

1. Nº de bytes a transferir
2. Tipo de transferencia (lectura/escritura)
3. Otra información de control

Inicialización controlador

Idem uno y dos

4. Dirección de memoria inicial para la transferencia
5. Nº de canal (para DMAs con varios canales)

Después de la inicialización la CPU retorna a sus tareas y ya no se preocupa más de la evolución de la transferencia.

Realización de la transferencia

- Cuando el periférico está listo para realizar la transferencia se lo indica al DMAC quien pide el control del bus y se realiza la transferencia entre el periférico y la memoria

Bus master: DMAC + Periférico - Bus slave: Memoria

Después de la transferencia de cada palabra se actualizan los registros del DMAC

- Nº de bytes o palabras a transferir
- Dirección de memoria

Finalización de la transferencia

- El DMAC libera el bus y devuelve el control a la CPU
- El DMAC suele activar una señal de interrupción para indicar la finalización.

Se puede degradar el rendimiento de la CPU si hace uso intensivo del bus. Si el DMAC toma control cuando la CPU no hace uso del bus el rendimiento del sistema no sufrirá degradación.
Con chaché El problema se reduce con el uso de memoria cache (La mayor parte del tiempo, la CPU lee instruc. de la cache, por lo que no necesita usar el bus de memoria. El DMAC puede aprovechar estos intervalos para realizar las transferencias).
Sin caché el procesador no utiliza el bus en todas las fases de la ejecución de una instrucción. El DMAC puede aprovechar las fases de ejecución de una instrucción en las que la CPU no utiliza el bus para realizar sus transferencias.

Se distinguen dos tipos de transferencias:

- **Por ráfagas (burst)**: Cuando la CPU concede el bus, el DMAC no lo libera hasta haber finalizado la transferencia de todo el bloque de datos completo. VENTAJAS:

La transferencia se realiza de forma rápida. DESVENTAJAS: Durante el tiempo que dura la transferencia la CPU no puede utilizar el bus con memoria.

• **Por robo de ciclo (cycle-stealing):** Cuando la CPU concede el bus al DMAC, se realiza la transferencia de una única palabra y después el DMAC libera el bus. El DMAC solicita el control del bus tantas veces como sea necesario hasta finalizar la transferencia del bloque completo. Para la CPU no es una interrupción. El procesador no debe guardar el contexto. Si bien el trabajo de la CPU es lento, no será tanto como si ella realizara la transferencia. Por lo tanto, para transferencia de E/S de múltiples palabras, es la técnica más eficiente. VENTAJAS: No se degrada el rendimiento del sistema. DESVENTAJAS: La transferencia tarda más tiempo en llevarse a cabo.

Canales de E/S

Los dispositivos de E/S son cada vez más sofisticados

Evolución: La CPU controla directamente los periféricos. Se agrega un módulo de E/S o controlador. Se agrega llamado de interrupción. El módulo de E/S provee el acceso directo a memoria (DMA). El módulo de E/S tiene su propio procesador con su pequeño conjunto de instrucciones. El módulo además tiene su memoria local o sea se convierte en una computadora en sí mismo.

Características de Canales de E/S

Representan una extensión al concepto de DMA (Tienen la habilidad de ejecutar instrucciones de E/S). Completo control de la transferencia de datos (la CPU no ejecuta instrucciones de E/S). Programa almacenado en memoria principal. La CPU inicia la transferencia de E/S (ordena ejecutar el programa que está en memoria. El programa especifica dispositivos, áreas de memoria a usar, prioridades y acciones ante errores).

Tipos de canales de E/S

Selector: Controla varios dispositivos de a uno, y se dedica a la transferencia de datos de ese. El canal selecciona un dispositivo y efectúa la transferencia. Los dispositivos son manejados por un controlador o módulo de E/S. Por lo tanto el canal de E/S ocupa el lugar de la CPU en el control de esos controladores.

Multiplexor: Puede manejar E/S con varios dispositivos a la vez.

Multiplexor de bytes: Acepta y transmite caracteres.

Multiplexor de bloques: Intercala bloques de datos desde distintos dispositivos.

SEGMENTACIÓN DE INSTRUCCIONES

Podemos dividir un procedimiento en varias etapas y ejecutar las mismas simultáneamente:

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

Segmentación de cauce:

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

Consiste en descomponer el proceso de ejecución de las instrucciones en fases o etapas que permitan una ejecución simultánea. Explota el **paralelismo** entre las instrucciones de un flujo secuencial. Es invisible al programador. Necesidad de uniformizar las etapas al **tiempo de la más lenta**. El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones.

Teórica: El máximo rendimiento es completar una instrucción con cada ciclo de reloj.
Si K es el número de etapas del cauce
Vel. procesador segmentado = Vel. secuencial x K

El incremento potencial de la segmentación del cauce es proporcional al número de etapas del cauce. Incrementa la productividad (throughput), pero no reduce el tiempo de ejecución de la instrucción

Suposiciones:
Todas las tareas duran el mismo tiempo. Las instrucciones siempre pasan por todas las etapas. Todas las etapas pueden ser manejadas en paralelo.
Problemas
No todas las instrucciones necesitan todas las etapas ni pueden ser manejadas en paralelo. No se tienen en cuenta los saltos de control.

Atascos de un cauce (stall)
Situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que le corresponde.
• **Estructurales**: Provocados por conflictos por los recursos (las instrucciones que están ingresando pueden querer lo mismo –acceder a memoria-)
• **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: una instrucción genera un dato que lee otra posterior
WAW: una instrucción escribe un dato después que otra posterior
WAR: una instrucción modifica un valor antes de que otra anterior que lo tiene que leer, lo lea.
• **Por dependencia de control**: una instrucción que modifica el valor del PC no lo ha hecho cuando se tiene que comenzar la siguiente.

POSIBLES SOLUCIONES A ATASCOS

Si resolvemos con paradas del cauce, disminuye el rendimiento teórico y CPI<1.

Soluciones a riesgos estructurales

Replicar, segmentar ó realizar turnos para el acceso a las unidades funcionales en conflicto.
• Duplicación de recursos hardware (en vez de duplicar la ALU pongo algunas réplicas que son necesarias): sumadores o restadores además de la ALU
• Separación en memorias de instrucciones y datos
• Turnar el acceso al banco de registros: Escrituras en la 1º mitad de los ciclos de reloj y Lecturas en la 2º mitad de los ciclos de reloj

Soluciones a riesgos de datos

Para riesgos **RAW**: se debe determinar cómo y cuándo aparecen esos riesgos. Será necesario: unidad de detección de riesgos y/o compilador más complejo

Dos soluciones:

• **Hardware**: Adelantamiento de operandos (forwarding) -> Consiste en pasar directamente el resultado obtenido con una instrucción a las instrucciones que lo necesitan como operando. Fácil de implementar si se identifican todos los adelantamientos y se comunican a los registros de segmentación correspondientes.

• **Software**: Instrucciones NOP (genera retardo) o reordenación de código (Máxima separación de instrucciones con dependencia RAW) sin afectar los resultados. Realizada por el compilador.

Soluciones a riesgos de control

Penalización por salto

• Instrucciones de salto

• **Incondicional**: La dirección de destino se debe determinar lo más pronto posible.

• **Condicional**: Introduce riesgo adicional por la dependencia entre la condición de salto y el resultado de una instrucción previa.

Modificación sencilla de la ruta de datos para reducir la cantidad de paradas a un solo ciclo.

• Adelantar la resolución de los saltos a la etapa D: En ella se decodifica y se sabe que es un salto; se puede evaluar la condición de salto (restador); se puede calcular la dirección de salto (sumador).

Técnica Hardware: Predicción de saltos para evitar la parada

Técnica Software: El compilador introduce instrucciones que se ejecutarán en cualquier caso después de la instrucción de salto.

Técnicas estáticas:

Predecir que nunca se salta: Asume que el salto no se producirá. Siempre capta la siguiente instrucción.

Predecir que siempre se salta: Asume que el salto se producirá. Siempre capta la instrucción destino del salto.

Técnicas dinámicas

Conmutador saltar/no saltar: Basado en la historia de las instrucciones. Eficaz para los bucles.

• Tabla de historia de saltos (branch-target buffer): Pequeña caché asociada a la etapa de búsqueda (F). Tres campos:

Dirección de una instrucción de bifurcación

Información de la instrucción destino (Dirección del destino ó Instrucción destino)

N bits de estado (historia de uso)

Salto retardado (Requiere reordenar las instrucciones)

Idea: Realizar trabajo útil mientras el salto se resuelve.

• Hueco o ranura de retardo de salto (delay-slot) es el período de penalización o parada luego de una instrucción de salto.

• El compilador trata de situar instrucciones útiles (que no dependan del salto) en los huecos de retardo. Si no es posible, se utilizan instrucciones NOP.

• Las instrucciones en los huecos de retardo de salto se captan siempre.

Otras soluciones hardware: Predecir según el código de operación (Hay instrucciones con más probabilidades de saltar). La tasa de acierto puede llegar a alcanzar un 75%.

Flujos múltiples: Varios cauces (uno por cada opción de salto). Precaptan cada salto en diferentes cauces. Se debe utilizar el cauce correcto. Desventajas: Provoca retardos en el acceso al bus y a los registros. Si hay múltiples saltos, se necesita un mayor número de cauces.

Precaptar el destino del salto: Se precapta la instrucción destino del salto, además de las instrucciones siguientes a la bifurcación. La instrucción se guarda hasta que se ejecute la instrucción de bifurcación.

Buffer de bucles: Memoria muy rápida. Gestionada por la etapa de captación de instrucción del cauce. Comprueba el buffer antes de hacer la captación de memoria. Muy eficaz para pequeños bucles y saltos.

RISC

El concepto de familia Separa la arquitectura de la implementación. Introduce el paralelismo en la naturaleza secuencial de los programas.

Computadoras de repertorio reducido de instrucciones

- Gran número de registros de uso general ó mejorar tecnología de compiladores para optimizar el uso de los registros.
- Repertorio de instrucciones limitado y sencillo.
- Énfasis en la optimización de la segmentación de instrucciones.

Finalidad del CISC: Facilitar el trabajo del escritor de compiladores. Dar soporte a HLL más complejos. Mejorar la eficiencia de la ejecución:

- Secuencias complejas de operaciones en microcódigo.

Inconvenientes del CISC: El software es mucho más caro que el hardware. El nivel del lenguaje era cada vez más complicado. Salto semántico: Diferencias entre operaciones HLL y operaciones de la Arquitectura

- Todo esto conduce a:
- Repertorios de instrucciones grandes
- Más modos de direccionamiento
- Varias sentencias de HLL implementadas en el Hardware (Por ejemplo, el CASE del VAX)

Características de la ejecución

Estudios sobre programas escritos en HLL

- Operaciones realizadas: Funcionamiento del procesador e interacción con memoria
- Operandos usados: Tipos y frecuencia de uso (Organización de la memoria y modos de direccionamiento)
- Secuenciamiento de la ejecución: Organización del control y del cauce

Estudios dinámicos: medir durante la ejecución

Operaciones

- Asignaciones: Movimiento de datos.
- Estamentos condicionales (IF, LOOP): Control secuencial.
- El procedimiento llamada/retorno consume mucho tiempo.
- Algunas instrucciones HLL conducen a muchas operaciones de código máquina.

Operandos: Principalmente variables escalares locales. La optimización debe concentrarse en el acceso a las variables locales.

Llamadas a procedimientos: Se consume mucho tiempo. Depende del número de parámetros tratados y del nivel de anidamiento.

- La mayoría de los programas no tienen una larga secuencia de llamadas seguida por la correspondiente secuencia de retornos.

• La mayoría de las variables son locales.
• Las referencias a operandos están muy localizadas.

Consecuencias: Se puede ofrecer mejor soporte para los HLL optimizando las prestaciones de las características más usadas y que más tiempo consumen.
• Usar un gran número de registros: Optimizar las referencias a operandos
• Prestar cuidadosa atención al diseño de los cauces de instrucciones: Predicción de bifurcaciones, etc.
• Es recomendable un repertorio con instrucciones simples (reducido).

Amplio banco de registros

Aproximación por Software:
• El compilador es necesario para asignar registros.
• Asignación de registros a las variables que se usen más en un período de tiempo dado.
• Requiere el uso de sofisticados algoritmos de análisis de programas.

Aproximación por Hardware:
• Utilización de más registros.
• De esta manera, más variables pueden mantenerse en registros durante periodos de tiempo más largos.

Registros para variables locales
• Muchos Registros=>Reducir el acceso a memoria.
• Por estudios anteriores=> Almacenar las variables escalares locales en registros.
Problema: Cada llamada de procedimiento/función cambia la *localidad*: Los parámetros deben ser pasados. Los resultados tienen que ser devueltos. Las variables de los programas de llamada tienen que ser restauradas.

Ventanas de registro
Por estudios realizados y con los registros: Se requieren pocos parámetros y variables locales en cada llamada. Hay limitación en la *profundidad* de llamadas. Utilización de múltiples conjuntos pequeños de registros para c/llamada distinta. La llamada cambia el conjunto de registros a usar. Los retornos vuelven a cambiar al anterior conjunto de registros utilizado. Tres áreas dentro de un conjunto de registros:
• Registros de parámetros.
• Registros de datos locales.
• Registros temporales.
Los registros temporales de un conjunto se solapan con los registros de parámetros del nivel más bajo adyacente (esto posibilita que los parámetros se pasen sin que exista transferencia de datos).

Variables globales
El compilador asigna posiciones de memoria a las variables (ineficiente para variables globales a las que se accede frecuentemente).
Incorporar al procesador un conjunto de registros para variables globales (dividir registros de la ventana en curso)

Amplio banco de registros vs cache

Banco de registros amplio	Cache
• Todos los datos escalares locales • Variables individuales • Variables globales asignadas por el compilador • Salvaguarda/restauración basadas en la profundidad de anidamiento	• Datos escalares locales recientemente usados • Bloques de memoria • Variables locales y globales usadas recientemente

• **Direccionamiento de registro**	• Salvaguarda/restauración basadas en el algoritmo de reemplazo • **Direccionamiento de memoria**

Optimización de uso de registros basada en el compilador

• Supongamos un pequeño número de registros (16 o 32)
• El uso optimizado es responsabilidad del compilador
• Los programas HLL no tienen referencias explícitas a los registros (Normalmente - pensando en C - registro int).
• Cada *cantidad* del programa candidata se asigna a un registro simbólico o virtual.
• Asignar el número ilimitado de registros simbólicos a un número fijo de registros reales.
• 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 *coloreado de grafos*.

¿Por qué CISC?
• ¿Simplificación del compilador? Ésta primera razón parece obvia. Instrucciones de máquina complejas son difíciles de aprovechar. La optimización es más difícil: *menos* tamaño, *más* velocidad.
• ¿Programas más pequeños? El programa ocupa menos memoria, pero la memoria hoy día es muy barata. El número de bits de memoria que ocupa no tiene porqué ser más pequeño al tener menos instrucciones (Más instrucciones necesitan códigos de operación más largos. Las referencias a registros necesitan menos bits).
• ¿Programas más rápidos? Propensión a usar las instrucciones más sencillas. Unidad de control más compleja. Memoria de control del microprograma más grande. Aumenta el tiempo de ejecución de las instrucciones simples. *No está nada claro que la tendencia hacia CISC fuera la apropiada.*

Características del RISC

• Una instrucción por ciclo.
• Operaciones registro a registro.
• Modos de direccionamiento sencillos.
• Formatos de instrucción sencillos.
• Diseño cableado (sin microcódigo).
• Formato de instrucción fijo.
• Mayor tiempo/esfuerzo de compilación.

RISC frente a CISC No existe una clara barrera diferenciadora. Muchos diseños incluyen características de ambos criterios.

- **Cuantitativa**: Comparación del 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.

Problemas de las comparaciones: No existe un par de máquinas RISC y CISC directamente comparables. No hay un conjunto de programas de prueba definitivo. Dificultad para separar los efectos del hardware de los del compilador. Mayoría de comparaciones con máquinas de "juguete", no con productos comerciales. La mayoría de las máquinas son una mezcla de ambas.

MEMORIA

CARACTERÍSTICAS CLAVES DE LOS SISTEMAS DE MEMORIA DE COMPUTADORAS
UBICACIÓN:
CPU
Interna
Externa

CAPACIDAD
Tamaño de palabra
Número de palabra

UNIDAD DE TRANSFERENCIA (número de bits que se leen o escriben en memoria a la vez)
Palabra
Bloque

MÉTODO DE ACCESO:
Acceso secuencial
Acceso directo
Acceso aleatorio
Acceso asociativo

PRESTACIONES
Tiempo de acceso

Memorias con acceso aleatorio: tiempo que transcurre desde que se presenta una dirección a la memoria hasta que el dato está disponible.

Memorias de otro tipo: tiempo que tarda en situar el mecanismo de W/R en la posición deseada.

Tiempo de ciclo: principalmente a las de acceso aleatorio; tiempo de acceso y algún tiempo más que se requiere antes de que pueda iniciarse un segundo acceso a memoria.

Velocidad de transferencia

Acceso aleatorio: velocidad a la que se pueden transferir datos a o desde una unidad de memoria.

Otras memorias:

Tmedio de N bits = Tacceso + N/R
Donde R es la velocidad de transferencia

DISPOSITIVO FÍSICO
Semiconductor
Soporte magnético
Soporte óptico
Magneto-óptico

CARACTERÍSTICAS FÍSICAS
No/Volátil
No/Borrable

• Los programadores desean acceder a cantidades ilimitadas de memoria rápida!!
• Solución: **Jerarquía de memoria**: organizada en niveles que son ubicados en distintos lugares físicos; fabricados con tecnologías diferentes que se gestionan de manera independiente.

Objetivo: la velocidad del sistema deberá ser, aproximadamente, la del nivel más rápido al costo del nivel mas barato. A medida que nos alejamos de la CPU, cada

nivel inferior es más grande, más lento y más barato que el nivel previo (o superior) en la jerarquía. Debe haber correspondencia de direcciones en los distintos niveles.

Propiedades a cumplir

• Inclusión: Los datos almacenados en un nivel han de estar almacenados en los niveles inferiores a él

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

¿Porqué funciona la jerarquía?

Principio de localidad de referencias

• Localidad Temporal: los elementos de memoria referenciados recientemente (datos o instrucciones), volverán a serlo en un futuro próximo => subo la palabra de nivel

• Localidad Espacial: los elementos de memoria cuyas direcciones están próximas a los últimos referenciados serán referenciados. => subo un bloque (con la palabra) de nivel

Memoria Cache

Cantidad pequeña de memoria rápida. Se ubica entre la memoria principal y la CPU. Puede localizarse en un chip o en módulo CPU.

Funcionamiento: La CPU solicita contenido de 1 dirección de memoria. La cache ¿tiene ese dato? Si es así, la obtiene de la cache (rápidamente). Si no está, se lee el bloque que contiene esa dirección desde la memoria principal y copia en la cache. Después, la cache entrega el dato requerido a la CPU. La cache incluye etiquetas para identificar qué bloque de la memoria principal está en cada una de sus líneas.

Acierto (hit): se encuentra en la caché el dato solicitado

Fallo (miss): no se encuentra en la caché el dato solicitado. Un bloque que contiene la palabra accedida se copia de la memoria principal a una línea de caché.

Tiempo para servir un fallo: depende de la latencia y ancho de banda de la memoria principal.

• Latencia: tiempo necesario para completar un acceso a memoria.

• Ancho de banda: cantidad de información por unidad de tiempo que puede transferirse desde/hacia la memoria.

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

Tiempo de acceso medio a memoria

Tacceso = Tacierto + Tfallos_memoria
Tfallos_memoria = Tasa de fallos x Penalización_fallo
Tacceso = Tacierto + TF x PF

Para mejorar las prestaciones

• Reducir el tiempo en caso de acierto (Tacierto)

• Reducir la tasa de fallos (TF)

• Reducir la penalización por fallo (PF)

MEMORIA PRINCIPAL SEMICONDUCTORA

Acceso aleatorio.

RAM: es posible W/R. Es volátil, debe estar continuamente alimentada, por ende, se usa solo como almacenamiento temporal. Se dividen en:

Dinámicas: requieren refrescos periódicos. Más simple, más pequeña, más densa y más económica.

Estáticas: retendrá los datos mientras esté alimentada. Más rápida

ROM: memoria de sólo lectura. Contiene un patrón permanente de datos que no puede alterarse. Existen derivaciones: PROM, EPROM, EEPROM, memorias flash.

ORGANIZACIÓN

El elemento básico de una memoria semiconductora es la celda de memoria. Presentan dos estados, puede escribirse en ellas para fijar su contenido, puede leerse para detectar estado.

Diseño de la cache

- **Organización (tamaño y cantidad)**
- **Política de ubicación: Tipo de función de correspondencia**

Correspondencia directa. Un bloque sólo puede estar almacenado en un lugar de la caché. Nº línea caché = Nº bloque ref. mod Nº líneas caché.

Simple. Poco costosa. Hay una posición concreta para cada bloque dado (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.)

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

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. Búsqueda costosa (en tiempo principalmente).

- Correspondencia asociativa por conjuntos. Un bloque puede almacenarse en un conjunto restringido de lugares en la caché. Un conjunto es un grupo de lineas de la caché. Nº conjunto = Nº bloque ref. mod Nº conjuntos caché.

Combina lo mejor de las otras correspondencias- La cache se divide en un grupo de conjuntos. Cada conjunto contiene un número de líneas N vías, con N=2, 4, 8... etc. Un bloque determinado corresponderá a cualquier línea de un conjunto determinado. El bloque B puede asignarse en cualquiera de las líneas del conjunto i.

- **Política de reemplazo: Algoritmo de sustitución**

En correspondencia directa: el que ocupa el lugar del nuevo. No hay elección. Sólo hay una posible línea para cada bloque. Se necesita una sustitución de esa línea (si o sí).

En correspondencia asociativa:

LRU (menos recientemente usado): controles de tiempo
FIFO (más antiguo): control de acceso
LFU (menos frecuentemente usado): controles de uso
Aleatoria: Se sustituye una línea al azar

Los algoritmos deben implementarse en hardware (para conseguir velocidad).

- **Política de escritura:** Se debe evitar inconsistencia de memorias en el caso de escrituras. Tener en cuenta: La CPU escribe sobre una línea de cache. El bloque de memoria principal correspondiente debe ser actualizado en algún momento. Un módulo E/S puede tener acceso directo a la memoria principal. En procesamiento paralelo, las múltiples CPU pueden tener caches individuales.

En acierto:

Write-through (Escritura inmediata). Se actualizan simultáneamente la posición de la caché y de la memoria principal: Con múltiples CPU, observar el tráfico a memoria principal para mantener actualizada cada cache local. Se genera mucho tráfico y retrasa la escritura.

Write-back (Post-escritura) La información sólo se actualiza en la caché. Se marca como actualizada -> bit de "sucio". La memoria principal se actualiza en el reemplazo y puede contener información errónea en algún momento.

En fallo:

Write allocate La información se lleva de la memoria principal a la caché. Se sobreescribe en la caché. Habitual con write-back

No-write allocate El bloque no se lleva a la memoria caché. Se escribe directamente en la memoria principal. Habitual con write-through.

La interpretación de la dirección física depende del tipo que se utilice. INDICE indicará la línea ó el conjunto que le corresponde. BO representa todas las direcciones que pertenecen al bloque.

ETIQUETA	INDICE	BO

BUSES DEL SISTEMA

Todas las unidades han de estar interconectadas. Existen distintos tipos de interconexiones para los distintos tipos de unidades:

Memoria:

Recibe y entrega datos.
Recibe direcciones (ubicación de trabajo).
Recibe señales de control:
- Leer
- Escribir
- Temporizar

Módulo de E/S:

E/S es funcionalmente similar a la memoria

Recibe y entrega datos del/al procesador (Envía y recibe datos al/del periférico)
Recibe direcciones (ubicación del periférico)
Recibe señales de control del procesador (Envía señales de control al periférico)
Envía señales de control al procesador (Interrupción)

Procesador

Lee instrucciones y datos.
Escribe datos (los procesados).
Envía señales de control a otras unidades.
Recibe (y utiliza) señales de interrupción.

Buses

Existe una serie de sistemas de interconexión. Las estructuras sencillas y múltiples son las más comunes. Ejemplo: control/dirección/bus de datos (PC); Ejemplo: unibus (DEC-PDP)

Qué es? Es un camino de comunicación entre dos o más dispositivos. Normalmente, medio de transmisión. Suele agruparse: Varios caminos de comunicación o líneas con función común. Un dato de 8 bits puede transmitirse mediante ocho líneas del bus.

Cómo son? Es un conjunto de conductores eléctricos paralelos. Líneas de metal. Poseen conectores para colocar *tarjetas*.

Bus de datos: Transmite datos. Recuerde que a este nivel no existe diferencia alguna entre "datos" e "instrucciones".El ancho del bus es un factor clave a la hora de determinar las prestaciones (8, 16, 32, 64 bits)

Bus de direcciones: Identifica la fuente o destino de un dato (cuando el procesador desea leer una palabra de una determinada parte en la memoria). El ancho del bus de direcciones determina la máxima capacidad de memoria posible en el sistema.

Bus de control: Transmite información de señales de control y temporización: Señal de escritura/lectura en memoria. Petición de interrupción. Señales de reloj.

Problemas de un único bus
Conectar gran número de dispositivos a un bus producen retardos de propagación. Si el control del bus pasa de un dispositivo a otro, puede afectar sensiblemente a las prestaciones. La mayoría de los sistemas utilizan varios buses para solucionar estos problemas. **Jerarquía de buses**

Tipos de buses
Dedicados: Uso de líneas separadas para direcciones y para datos.
- 16 líneas de direcciones
- 16 líneas de datos
- 1 línea de control de lectura ó escritura (r/w)

Multiplexados: Uso de las mismas líneas.
- 16 líneas de direcciones ó datos
- 1 línea de control de lectura ó escritura (r/w)
- 1 línea de control para definir direcciones ó datos (a/d)

Arbitraje del bus
El control del bus puede necesitar más de un módulo. Ejemplo: CPU y el controlador DMA. Sólo una unidad puede transmitir a través del bus en un instante dado. Los métodos de arbitraje se pueden clasificar como:

Centralizados: Un único dispositivo hardware es responsable de asignar tiempos en el bus: Controlador del bus ó Árbitro (puede estar en un módulo separado o ser parte del procesador)

Distribuidos: Cada módulo puede controlar el acceso al bus. Cada módulo dispone de lógica para controlar el acceso.

Temporización
Forma de coordinar los eventos en el bus.
Temporización síncrona:

La presencia de un evento está determinada por un reloj.
El bus incluye una línea de reloj.
Un intervalo desde un "uno" seguido de otro a "cero" se conoce como ciclo de bus.
Todos los dispositivos del bus pueden leer la línea de reloj.
Suele sincronizar en el flanco de subida.
La mayoría de los eventos se prolongan durante un único ciclo de reloj.

Bus PCI: Interconexión de Componente Periférico.

PROCESADORES SUPERESCALARES

Unidad segmentada: secuencia de etapas con la propiedad que nuevas operaciones pueden iniciarse mientras otras están en proceso. Esto aumenta el *paralelismo* e incrementa la *aceleración*. Dio lugar a procesadores ***superescalares*** y a procesadores ***segmentados***.

Implementación supersegmentada: muchas instrucciones no necesitan todo un ciclo de reloj, por esto subdividen el ciclo para obtener más prestaciones.

Implementación superescalar: (en gris están las soluciones) las instrucciones comunes pueden iniciar su ejecución simultáneamente y ejecutarse de manera independiente. Usan, por lo general, la predicción dinámica de saltos basada en las historia de los mismos. Para hacer esto posible, duplican algunas o todas las partes de la CPU/ALU. Políticas de emisión:

Emisión y finalización en orden: emitir en el orden exacto en el que lo haría una ejecución secuencial y escribirlos en ese mismo orden. **Ingenua!**

Emisión en orden y finalización desordenada: se usa en RISC para mejorar la velocidad. Puede haber cualquier número de instrucciones en la etapa de ejecución hasta alcanzar lo máximo del paralelismo. La emisión de instrucciones se para cuando hay una pugna por un recurso, una dependencia de datos o de procedimiento. Además, existe una nueva dependencia, la de salida.

Emisión y finalización desordenada: es necesario desacoplar las etapas del cauce de decodificación y ejecución, lo cual se hace gracias a la ventana de instrucciones: cuando se terminar de decodificar la coloca allí, hasta que se llene. El resultado es que el procesador tiene una capacidad de anticipación que permite identificar a las independientes.

Renombre de registros: (solución a la dependencia de salida y a la antidependencia) duplicación de recursos. El procesador asigna dinámicamente los registros que están asociados con los valores que necesitan las instrucciones en diversos tiempos. Cuando se crea un nuevo valor de registro (instrucción hace referencia a operando destino) se asigna un nuevo registro para ese valor. Las posteriores que accedan a él como fuente, tienen que sufrir un proceso de renombramiento: las referencias a registros de esas instrucciones han de revisarse para referenciar el registro que contiene el valor que se necesita.

Paralelismo de instrucciones: se refiere al grado en el que las instrucciones de un programa se pueden ejecutar en paralelo ya que son independientes entre si y por lo tanto pueden solaparse. Existe dependencia de datos (una instrucción necesita el resultado de la previa) y relativa al procedimiento (cuando hay saltos). Se generan conflictos en los recursos (dos o más quieren acceder al mismo). Hay dependencia de salida. Antidependencia.

Paralelismo de la máquina: medida de la capacidad del procesador para sacar partido al paralelismo a nivel de instrucciones. Depende del número de instrucciones que puedan captarse y ejecutarse al mismo tiempo y de la velocidad y sofisticación del mecanismo que usa el procesador para localizar instrucciones independientes (orden en que se captan, ejecutan y actualizan en memoria).

Ejecución: el procesador envía las instrucciones a una ventana de ejecución, donde ya no forman un flujo secuencial, sino que están estructuradas de acuerdo a sus dependencias de datos. Se ejecutan y luego se vuelven a poner en orden secuencial y sus resultados se registran.

Implementación:

Estrategias de captación simultánea de múltiples instrucciones, prediciendo los resultados de los saltos.

Lógica para determinar dependencias entre los valores de registros y mecanismos para comunicar esos valores.

Mecanismos para iniciar, o emitir múltiples instrucciones en paralelo.

Recursos para la ejecución en paralelo de múltiples instrucciones.

Mecanismos para entregar el estado del procesador en un orden correcto.

Consideraciones:

Influencia de las excepciones, ya que en ese momento hay varias instrucciones en ejecución. Para garantizar el funcionamiento correcto, la que origina la excepción y las siguientes se abortan. Luego del tratamiento, se continúa por la que generó el paro.

SUPERESCALARES Y MÁS VLIW

Hacia Pentium 4: CISC

Pentium – algún componente superescalar (2 unidades de ejecución de enteros separadas)
Pentium Pro – todo superscalar
Modelos subsequentes refinan y mejoran el diseño superscalar

Operación

Busca instrucciones en memoria en el orden del programa estático.
Traduce instrucciones en 1 o + instrucciones RISC (micro-operaciones)
Ejecuta las micro-ops en un cauce superscalar (pueden ser ejecutadas fuera de orden)
Entrega resultados de micro-ops a los registros en el orden de flujo de programa original
Caparazón CISC con núcleo RISC
Cauce del núcleo interno de al menos 20 etapas (Algunas micro-ops requieren múltiples etapas de ejecución)

Hacia IA64

Pentium 4 aparece como el último de la línea x86.
Desarrollo conjunto de Intel y HP
Nueva arquitectura
- 64 bit
- No es extensión de x86
- No adapta la arquitectura HP de sus RISC de 64 bits

Utiliza muchos circuitos a altas velocidades
Hace uso sistemático del paralelismo
Base superscalar

Motivación

Paralelismo a nivel instrucción – ILP
- Implícito en la instrucción de máquina
- No determinado por el procesador en tiempo de ejecución

Palabras de instrucción larga o muy larga (LIW/VLIW)
Predicación de saltos (no es predicción de saltos)
Carga especulativa (Intel-HP lo llaman EPIC Explicit Parallel Instruction Computing)
Arquitectura IA-64 pensada para implementación EPIC
Itanium es el primer producto de Intel

¿Porqué nueva arquitectura?

Hardware no compatible con x86 (IA32)
Muchos millones de transistores disponibles en chip
- Se pueden armar caches más grandes
- Se pueden colocar varios procesadores
- Se pueden agregar más unidades de ejecución
  - Aumenta superescalado, procesador más "ancho"
  - Se necesita más lógica para dirigir, mejor predicción de saltos, cauces más largos, más registros de "renaming"
  - Hay mayores penalidades por mala predicción
  - Retiro de hasta seis instrucciones por ciclo

Paralelismo explícito

Paralelismo de instrucción programado en tiempo de compilación. El procesador usa esa información para realizar ejecución paralela. Requiere circuitos menos complejos. El compilador tiene más tiempo para determinar posibles operaciones paralelas. El compilador mira todo el programa.

Características clave

Gran número de registros: Formato de instrucción de IA-64 asume 256 registros: 128 de 64 bit - para enteros, lógica y propósito general; 128 de 82 bit - para punto flotante y gráficos; 64 registros de 1 bit - para ejecución predicada. Soportan alto grado de paralelismo

Múltiples unidades de ejecución: Se espera 8 o más. Depende del número de transistores disponible. Ejecución paralela depende en hardware disponible (8 instrucciones en paralelo pueden dividirse en 2 lotes de 4 si hay 4)

Unidades de ejecución

I-Unit
Aritmética de enteros
Suma y desplazamientos
Lógica
Comparaciones
Operaciones multimedia de enteros

M-Unit
Load y Store (entre registro y memoria); alguna ALU de enteros

B-Unit
Instrucciones de salto

F-Unit
Instrucciones de punto flotante

Ejecución con predicados

Cualquier operación puede referirse a un registro de predicado

<PRi> operación

La operación se retirará (resultados visibles) sólo cuando el valor del predicado sea verdad (PRi=1). Si el valor se conoce cuando la instrucción se emite, la operación es ejecutada sólo si ese valor es verdadero. Si el valor no se conoce, la operación se inicia; si el valor se convierte en falso, la operación se descarta.

Si no se menciona registro de predicado, la operación es ejecutada y retirada incondicionalmente.

Saltos predicados

Es una técnica de compilación (para generar código con alto grado de ILP); Se basa en la ejecución con predicados de la IA-64; Se deja que ambas ramas de un salto condicional se ejecuten en paralelo (para explotar todo el potencial de paralelismo); Se eliminan los saltos y reemplazan por ejecución condicional (se requiere soporte de hardware)

Ubicación de operaciones de carga

Una instrucción LOAD (carga desde memoria) puede ser reubicada de modo de evitar la latencia de memoria (el valor debe estar cuando se necesite)

¿si se mueve de dentro de un salto? La carga se ejecuta para ambas ramas: Si hay recursos hardware disponibles, no habría problemas. Si no se necesita la rama, será tiempo perdido. Se debe monitorear si provoca una excepción (ej. fallo de cache) Interesa cuando la rama es útil por los recursos y el tiempo que la gestión de la misma necesita.

Carga especulativa

La instrucción LOAD en el programa original puede ser reemplazada por 2 instrucciones:
Carga especulativa (ej. LOAD.S)

Búsqueda en memoria
Detección de excepción generada. No se reporta al S.O.
Ésta instrucción es la que cambia de lugar

Chequeo (ej. CHK.S)

Reporta la excepción si la carga especulativa la detectó
Ésta instrucción queda en su lugar
Si no se detectó excepción, no pasa nada

Puedo manejar excepciones

Comparación

Superescalar	IA-64
• Instrucciones de línea RISC, una por palabra	• Instrucciones de línea RISC, empaquetadas en grupos de a 3
• Múltiples unidades de ejecución paralela	• Múltiples unidades de ejecución paralela
• Reordena y optimiza flujo de instr en tiempo de ejecución	• Reordena y optimiza flujo de instr en tiempo de compilación
• Predicción de salto con ejecución especulativa de un camino	• Ejecución especulativa a lo largo de ambos caminos de un salto
• Carga datos de memoria sólo cuando necesita e intenta en	• Carga especulativa de datos antes de necesitarlos e intenta en

PROCESAMIENTO PARALELO

Clases de computadores según FLYNN

SISD: un único procesador interpreta una única secuencia de instrucciones, para operar con los datos almacenados en una única memoria. *Computadores monoprocesador*.

SIMD: una única instrucción máquina controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada uno tiene una memoria asociada, de modo que cada instrucción es ejecutada por cada procesador con un conjunto de datos diferentes. *Procesadores vectoriales y matriciales*: (explicación) computadora con una única UC y una matriz de elementos; tipos de instrucciones: extensiones de las instrucciones escalares (se convierten en operaciones vectoriales ejecutadas en todos los procesadores de modo simultáneo).

MISD: se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. *Nunca implementada*.

MIMD: un conjunto de procesadores ejecuta simultáneamente secuencias de instrucciones diferentes con un conjunto de datos diferentes. *Los SMP, los clusters, y los sistemas NUMA*.

SMP: varios procesadores comparten una única memoria y las E/S mediante un mecanismo de interconexión. Cada uno posee UC, ALU, registros, y posibles cache. El tiempo de acceso a memoria principal es aproximadamente el mismo para cualquier procesador (UMA). Todos pueden desempeñar las mismas funciones. El sistema está controlado por un SO integrado, que proporciona la interacción entre los procesadores y sus programas. Ventajas:

Prestaciones: si el trabajo puede realizarse en paralelo.

Disponibilidad: un fallo en un procesador no detendrá la computadora.

Crecimiento incremental: se pueden aumentar las prestaciones del sistema añadiendo más procesadores.

Escalado: hay variedad de precios y prestaciones.

Desventajas:

La prestación está limitada por el tiempo de ciclo del bus. Cada procesador debería tener su propia cache. Se pueden producir problemas de coherencia de cache.

CLUSTERS: grupo de computadores completos interconectados que trabajan conjuntamente como un único recurso de cómputo. Cada uno se denomina *nodo*. Prestaciones y disponibilidad elevadas. Aplicaciones propias de un servidor. Ventajas: escalabilidad absoluta e incremental, mejor relación precio/prestaciones.

CLUSTER	SMP
Dan soporte a aplicaciones de alta demanda de recursos	

Disponibles comercialmente	
Superior escalabilidad incremental y abs	Fácil de configurar y administrar
Superior disponibilidad	Planificación
	Menos espacio, menos potencia
	Plataformas estables y establecidas

Todos los procesadores tienen acceso a toda la memoria
UMA: igual tiempo de acceso a todas las regiones y para todos los procesadores.
NUMA: el tiempo de acceso a diferentes zonas de memoria puede diferir. El objetivo es mantener una memoria transparente desde cualquier parte del sistema.
CC-NUMA: es un NUMA que mantiene coherencia de cache entre las caches de los distintos procesadores. Cada procesador tiene cache L1 y L2. Cada nodo tiene su memoria principal y están conectados. Es automático y transparente.

PROCESAMIENTO MULTIHEBRA: la secuencia se divide en secuencias más pequeñas llamadas *threads* que pueden ejecutarse en paralelo. Incluye un contexto de procesador (incluido PC y SP) y área de datos para su pila. Se ejecuta secuencialmente. Interrumpible (el procesador cambiaría a otra hebra). Conmutación de hebra: cambio de control del procesador entre hebras de un mismo proceso.

Explícito: ejecución concurrente de instrucciones de diferentes hebras explícitas. Mezcla de instrucciones de diferentes hebras en cauces compartidos o por ejecución paralela en cauces paralelos.

Implícito: ejecución concurrente de varias hebras extraídas de un único programa secuencial.

Su procesador: PC distinto para cada hebra que pueda ejecutarse concurrentemente. Se trata cada hebra separadamente. Aproximaciones con ejecución simultánea real.

CPU: debe hacer

Captar instrucción
Interpretar instrucción
Captar datos
Procesar datos
Escribir datos

Para todo esto, es obvio que necesita almacenar datos temporalmente. Debe recordar la posición de la última instrucción (para saber dónde buscar la siguiente). Está compuesta por **ALU**, **UC**, **MEMORIA INTERNA MÍNIMA CON REGISTROS** (de uso general, datos, direcciones, códigos de condición), **REGISTROS DE CONTROL** (PC, IR, MAR, MBR) y **DE ESTADO** (flags).

CICLO DE INSTRUCCIÓN
Captación: se lee una instrucción de memoria. **PC** contiene la dirección de la siguiente, la cual es llevada a **MAR** y puesta en el bus de direcciones. La **UC** solicita una lectura de memoria y el resultado se pone en el bus de datos, se copia en **MBR** y luego se lleva a **IR**.
Ejecución: depende de cuál de las diversas instrucciones máquina esté en **IR**.