Unidad 1

Concepto de Arquitectura: La arquitectura de computadores se refiere a los atributos de un sistema que son visibles para un programador, o para aquellos atributos que tienen un impacto directo en la ejecución lógica de un programa. Para poner un ejemplo una cuestión de diseño arquitectónico es si el computador tendrá la instrucción de multiplicar, que conjunto de instrucciones lleva, número de bits usados para representar varios tipos de datos como números y caracteres, mecanismos de entrada salida y técnicas para direccionamiento de memoria.

Relación con Organización de Computadoras: la organización de computadores se refiere a las unidades funcionales y sus interconexiones, que dan lugar a especificaciones arquitectónicas. Entre los atributos de organización se incluyen aquellos detalles de hardware transparentes al programador, tales como señales de control, interfaces entre el computador y los periféricos y la tecnología de memoria usada. Para poner un ejemplo integrador se puede decir que una cuestión de diseño arquitectónico es si el computador tendrá la instrucción de multiplicar. Una cuestión de organización es si esa instrucción será implementada por una unidad especializada en multiplicar, o por un mecanismo que haga un uso iterativo de la unidad de suma del sistema. La decisión de organización puede estar basada en la frecuencia prevista del uso del a instrucción de multiplicar, la velocidad relativa de las dos aproximaciones y el coste y el tamaño físico de una unidad especializada en multiplicar.

Muchos fabricantes ofrecen una familia de modelos, todos con la misma arquitectura, pero con diferencias en cuanto a la organización. Consecuentemente, los diferentes modelos de la familia tienen precios y prestaciones distintas. Una arquitectura puede vivir muchos años pero su organización cambia con la evolución de la tecnología.
Los microcomputadores, la relación entre arquitectura y organización es muy estrecha. Los cambios en la tecnología no solo influyen en la organización sino que también dan lugar a la introducción de arquitecturas más ricas y potentes. Generalmente hay menos requisitos de compatibilidad, generacion a generacion, para estas pequeñas máquinas. Así hay más interacción entre las decisiones de diseño arquitectónicas y de organización. Un ejemplo interesante de esto son los computadores de repertorio reducido de instrucciones RISC.

Repaso del modelo de von Neumann

estructura general:

- una memoria principal, que almacena tanto datos como instrucciones.
- Una unidad aritmetico-logica(ALU), capaz de hacer operaciones con datos binarios.
- Una unidad de control, que interpreta las instrucciones en memoria y provoca su ejecución
- Un equipo de entrada-salida dirigido por la unidad de control.

Proposición de Von Neumann

1. **primero:** el dispositivo tendrá que realizar las operaciones aritméticas elementales muy frecuentementes. Estas son: suma, resta, multiplicacion y division. Por lo tanto es razonable que tenga elementos especializados solo en estas operaciones. Debe observarse sin embargo, que aunque este principio parece consistente, la manera específica de cómo se aplica requiere un examen cuidadoso. En cualquier caso, tendrá que existir la parte de aritmética central que constituye la primera parte especifica: CA (central arithmetic).

2. **Segundo:** el control lógico del dispositivo debe ser realizado eficientemente por un órgano central de control. Si este dispositivo de control tiene que servir en lo posible para todo uso, hay que diferenciar entre las instrucciones específicas que definen un problema particular y los órganos de control general que se ocupan de que se lleven a cabo dichas instrucciones, sean cuales sean. Las primeras deben almacenarse en un algún lugar. Las otras deben representarse definiendo partes operativas del dispositivo. Con el control central nos referimos solo a esta última función y los órganos que la realizan forman la segunda parte especifica CC(central control)
3. **tercero:** cualquier dispositivo que realice secuencias largas y complicadas de operaciones(concretamente de cálculo), debe tener una memoria considerable. Las instrucciones que gobiernan un problema complicado pueden constituir un material considerable sobre todo si el código es circunstancial. En cualquier caso, la memoria total es la tercera parte específica del dispositivo M (memory). Las tres partes corresponden a las neuronas asociativas del sistema nervioso. Queda por discutir los equivalentes a las neuronas sensoriales o aferentes. Estos son los órganos del dispositivo de entrada y salida. El dispositivo tiene que estar dotado con la habilidad de mantener contacto de entrada y salida con medios específicos de este tipo R (recording).
4. **Cuarto:** el dispositivo tiene que tener órganos para transferir, estos órganos forman su entrada ,la cuarta parte I input veremos que lo mejor es hacer todas las transferencias a partir de R hasta M y nunca directamente de C.
5. **Quinto:** el dispositivo tiene que tener órganos para transferir, información específica C y M hacia R. Estos forman la salida, la quinta parte O (output). Es mejor hacer todas las transferencias de M mediante O a R y nunca directamente a partir de C.

La unidad de control capta instrucciones de la memoria y ejecuta una a una. Para explicar esto, se necesita registros, definidos así:

- **registro temporal de memoria (MBR):** contiene una palabra que debe ser almacenada en la memoria, o es usado para recibir una palabra procedente de la memoria.
- **Registro de dirección de memoria (MAR):** especifica la dirección en memoria de la palabra que va a ser escrita o leída en MBR.
- **Registro de instrucción (IR):** contiene los 8 bit del código de operación de la instrucción que se va a ejecutar.
- **Registro temporal de instrucción (IBR):** empleado para almacenar temporalmente la instrucción contenida en la parte derecha de una palabra en memoria.
- **Contador de programa(PC):** contiene la dirección de su próxima pareja de instrucciones que van a ser captadas de la memoria.
- **Acumulador (AC) y multiplicador cociente (MQ):** se emplean para almacenar operando y resultados de operaciones de la ALU temporalmente. Los bit más significativos en AC y los menos en MQ.

Descripción del funcionamiento de un sistema basado en un microprocesador:
A medida que la densidad de elementos en los chips de memoria ha continuado creciendo, también lo ha hecho la densidad de elementos de procesamiento. Conforme el tiempo pasaba, en cada chip había más y más elementos, así que cada vez se necesitaban menos y menos chips para construir un procesador de un computador. En 1971. se hizo una innovación, cuando intel desarrolló su 4004. el primer chip que contenía todos los componentes de la cpu en un solo chip. Con el nace el microprocesador.

Buses, teoría de operación, buses sincrónicos y asincrónicos. Ejemplos:

El bus del sistema está constituido por entre 50 y 100 líneas. A cada línea se le asigna un significado o una función particular.las líneas se pueden clasificar en tres grupos funcionales:

Las líneas de datos proporcionan un camino para transmitir datos entre los módulos del sistema. El conjunto constituido por estas líneas se denomina bus de datos. Contra de 8, 16 o 32 líneas distintas, cuyo número se conoce como anchura del bus de datos. Pues cada línea solo puede transportar un bit cada vez, el numero de lineas determina cuántos bits se pueden transferir al mismo tiempo. La anchura del bus es un factor clave a la hora de determinar las prestaciones del conjunto del sistema.
por ejemplo si el bus de datos tiene una anchura de 8 bits y las instrucciones son de 16 bit, entonces el cpu debe acceder al módulo de memoria dos veces por cada ciclo de instrucción.

Las líneas de dirección se utilizan para designar la fuente o el destino del dato situado en el bus de datos. Si el procesador desea leer una palabra (8,16,32 bit) de datos de la memoria, sitúa la dirección de la palabra que desea en la línea de direcciones. Claramente la anchura del bus de direcciones determina la máxima capacidad de memoria posible en el sistema. Además, las líneas de direcciones generalmente se utilizan también para direccionar los puertos de E/S. Usualmente los bits de orden más alto se utilizan para seleccionar una posición de memoria o un puerto de E/S dentro de un módulo.

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

- escritura en memoria: hace que el dato del bus se escriba en la posición direccionada.
- Lectura de memoria: hace que el dato de la posición direccionada se sitúe en el bus
- escritura de e/s: hace que el dato del bus se transfiera a través del puerto de e/s direccionado.
- Lectura de e/s: hace que el dato del puerto de e/s direccionado se sitúe en el bus.
- Transferencia reconocida: indica que el dato se ha aceptado o se ha situado en el bus.
- Petición del bus: indica que un módulo necesita disponer del control del bus.
- Cesión del bus: indica que se cede el control del bus a un módulo que lo había solicitado.
- Petición de interrupción: indica si hay una interrupción pendiente.
- Interrupción reconocida: señala que la interrupción pendiente se ha aceptado.

- Reloj: se utiliza para sincronizar las operaciones.
- Inicio: pone los módulos conectado en su estado inicial.

Además pueden existir líneas de alimentación para suministrar energía a los módulos conectados al bus.

Si un módulo desea enviar un dato a otro debe hacer dos cosas: obtener el uso del bus, y transferir el dato a través del bus. Si un módulo desea pedir un dato a otro módulo debe obtener el uso del bus y transferir la petición al otro módulo mediante las líneas de control y dirección apropiadas. Después debe esperar a que el segundo módulo envíe el dato.
Físicamente, el bus de sistema es de hecho un conjunto de conductores eléctricos paralelos. Estos conductores son líneas de metal grabadas en una tarjeta de circuito.

Jerarquía de buses

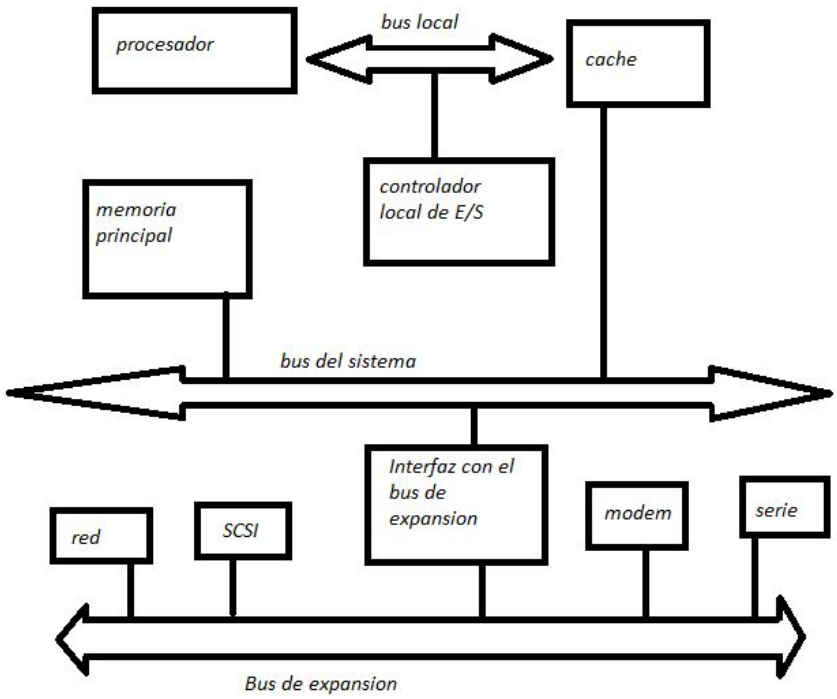
si se conecta un gran número de dispositivos al bus, las prestaciones pueden disminuir. Hay dos causas principales:

1. A más dispositivos conectados al bus, mayor es el retardo de propagación. Este retardo determina el tiempo que necesitan los dispositivos para coordinarse en el uso del bus. Si el control del bus pasa frecuentemente de un dispositivo a otro, los retardos de propagación pueden afectar sensiblemente a las prestaciones.

2. El bus puede convertirse en un cuello de botella a medida que las peticiones de transferencia acumuladas se aproximan a la capacidad del bus. Este problema se puede resolver en alguna medida incrementando la velocidad a la que el bus puede transferir los datos, y utilizando buses más anchos (por ejemplo incrementando el bus de datos de 32 a 64).sin embargo, puesto que la velocidad de transferencia que necesitan los dispositivos conectados al bus(controladores de gráfico y video, o interfaz de red) está incrementandose rápidamente, es un hecho que el bus único está destinado a dejar de utilizarse.

La mayoría de los computadores utilizan varios buses, normalmente organizados jerárquicamente. Una estructura típica se muestra en (a). Hay un bus local que conecta el procesador a una memoria caché, y al que pueden conectarse también uno o más dispositivos locales. El controlador de memoria caché conecta la caché no solo albus local, sino también al bus de sistema, donde se conectan todos los módulos de memoria principal. El uso de una caché alivia la exigencia de soportar los accesos frecuentes del procesador a memoria principal. De hecho la memoria principal puede pasar del bus local al bus de sistema. De esta forma, las transferencias de E/S con la memoria principal a través del bus de sistema no interfieren la actividad del procesador.

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



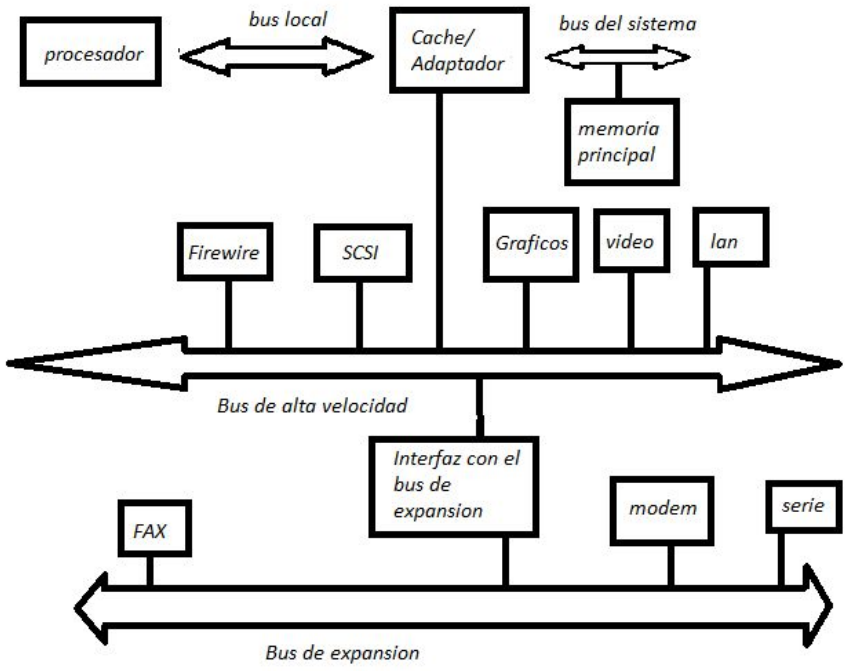
(a) Arquitectura de bus tradicional

Las conexiones a red incluyen conexiones a redes de área local, tales como una red ethernet de 10Mbps y conexiones a redes de área amplia(wide area networks), tales como red de conmutación de paquetes (packet-switching network). La interfaz SCSI (small computer system interface) es en sí un tipo de bus utilizado para conectar controladores de disco y otros periféricos. El puerto serie puede utilizarse para conectar una impresora o un escáner.

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

La figura (b) muestra un ejemplo típico de esta aproximación. De nuevo hay un bus local que conecta el procesador a un controlador de caché, que a su vez está conectado al bus de sistema que soporta a la memoria principal. El controlador de caché está integrado junto con el adaptador, o dispositivo de acoplo, que permite la conexión al bus de alta velocidad. Este bus permite la conexión de LAN de alta velocidad, tales como Fast ethernet a 100MBps, controladores de estaciones de trabajo específicos para aplicaciones gráficas y de video, y tambien controladores de interfaz para buses periféricos, tales como SCI y Firewire. Este último es un bus de alta velocidad diseñado específicamente para conectar dispositivos de E/S de alta capacidad. Los dispositivos de velocidad menor pueden conectarse al bus de expansión, que utiliza una interfaz para adaptar el tráfico entre el bus de expansión y el bus de alta velocidad.

La ventaja de esta organización es que el bus de alta velocidad acerca al procesador los dispositivos que exigen prestaciones elevadas y, al mismo tiempo, es independiente del procesador. Así, se pueden tolerar las diferencias de velocidad entre el procesador y el bus de altas prestaciones y las variaciones en la definición de las líneas de los buses. Los cambios en la arquitectura del procesador no afectan al bus de alta velocidad y viceversa.



(b) Arquitectura de altas prestaciones

Elementos de diseño de un bus

Aunque existe una gran diversidad de diseños de buses, hay unos pocos parámetros o elementos de diseño que sirven para distinguir y clasificar los buses:

- **Tipo:** Dedicado, Multiplexado.
- **Método de arbitraje:** Centralizado, Distribuido.
- **Temporización:** Síncrono, Asíncrono.
- **Anchura del bus:** Dirección, Datos.
- **Tipo de transferencia de datos:** Lectura(L), Escritura(E), L-Modificación-E, L-después de-E, Bloque.

Tipo de Buses

Las líneas del bus se pueden dividir en dos tipos genéricos: dedicadas y multiplexadas. Una línea de bus dedicada esta permanente asignada a una función o a un subconjunto físico de componente del computador. Un ejemplo de dedicación funcional, común en muchos buses es el uso de líneas separadas para direcciones y para datos. Sin embargo, no es esencial. Por ejemplo, la información de dirección y datos podría transmitirse a través del mismo conjunto de líneas sis se utiliza una línea de control de dirección valida. Al comienzo de la transferencia de datos, la dirección se sitúa en el bus y se activa la línea de dirección valida. En ese momento, cada módulo dispone de un periodo de tiempo para copiar la dirección y determinar si es el módulo direccionado. Después la

dirección se quita del bus y las mismas conexiones se utilizan para la subsecuente transferencia de lectura o escritura de datos. Este método de uso de las mismas líneas para usos diferentes se llama multiplexado en el tiempo.

La ventaja de multiplexado en el tiempo es el uso de menos lineas, cosa que ahorra espacio y normalmente costes. La desventaja es que se necesita una circuitería más compleja cada módulo. Además existe una posible reducción en las prestaciones debido a que los eventos deben compartir las mismas líneas no pueden producirse en paralelo. La dedicación física se refiere al uso de múltiples buses, cada uno de los cuales conecta solo un subconjunto de módulos. Un ejemplo típico es el uso de un bus de E/S para interconectar todos los módulos de E/S; este bus, a su vez se conecta al bus principal a través de algún tipo de módulo adaptador de E/S. La ventaja potencial de la dedicación física es su elevado rendimiento, debido a que hay menos disputas por el acceso al bus. Una desventaja es el incremento en el tamaño y el costo del sistema.

Método de arbitraje

En todos los sistemas, puede necesitar el control del bus más de un módulo. Por ejemplo, un módulo de E/S puede necesitar leer o escribir directamente en memoria, sin enviar el dato al procesador. Puesto que, en un instante dado, sólo una unidad puede transmitir a través del bus, se requiere algún método de arbitraje. Los diversos métodos se pueden clasificar como centralizados o distribuidos. En un esquema centralizado, un único dispositivo hardware, denominado controlador del bus, es responsable de asignar tiempos en el bus. El dispositivo puede estar en un módulo separado o ser parte del procesador. En un esquema distribuido, no existe un controlador central. En su lugar, cada módulo dispone de lógica para controlar el acceso y los módulos actúan conjuntamente para compartir el bus. En ambos métodos de arbitraje el propósito es designar un dispositivo, el procesador o un módulo de E/S, como maestro del bus. El maestro podría entonces iniciar una transferencia de datos (lectura o escritura) con otro dispositivo que actúa como esclavo en este intercambio concreto.

Temporización

Hace referencia a la forma en la que se coordinan los eventos en el bus. Síncrona, la presencia de un evento en el bus está determinada por un reloj. El bus incluye una línea de reloj a través de la que se transmite una secuencia en la que se alternan intervalos regulares de igual duración a uno y a cero. Un único intervalo a uno seguido de otro a cero se conoce como ciclo de reloj o ciclo de bus y define un intervalo de tiempo unidad (time slot). Todos los dispositivos del bus pueden leer la línea de reloj, y todos los eventos empiezan al principio del ciclo del reloj.

Con la temporización síncrona, la presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo. El procesador sitúa las señales de dirección y lectura en el bus. Después de un breve intervalo para que las señales se estabilice, activa la señal MSYN(master sync) indicando la presencia de señales de dirección y control válidas. El módulo de memoria responde proporcionando el dato y una señal SSYN(slave sync).

La temporización síncrona es más fácil de implementar y comprobar. Sin embargo, es menos flexible que la temporización asíncrona. Debido a que todos los dispositivos en un bus síncrono deben utilizar la misma frecuencia de reloj, el sistema no puede aprovechar las mejoras en las prestaciones de los dispositivos. Con la temporización asíncrona, pueden compartir el bus una mezcla de dispositivos lentos y rápidos, utilizando tanto las tecnologías más antiguas, como las más recientes.

Anchura del bus
El concepto e anchura del bus se ha presentado ya. La anchura del bus de datos afecta a las prestaciones del sistema: cuanto más ancho es el bus de datos, mayor es el número de bits que se transmiten a la vez. La anchura del bus de direcciones afecta a la capacidad del sistema:cuanto más ancho es el bus de direcciones, mayor es el rango de posiciones a las que se puede hacer referencia.

Tipos de transferencia de datos
Todos los buses permiten transferencias de escritura(dato maestro a esclavo), como de lectura(dato de esclavo a maestro). En el caso de un bus con direcciones y datos multiplexados, el bus se utiliza primero para especificar la dirección y luego para transferir el dato. En una operación de lectura, generalmente hay un tiempo de espera mientras el dato se está captando del dispositivo esclavo para situarlo en el bus. Tanto para la lectura como para la escritura, puede haber también un retardo si se necesita utilizar algún procedimiento de arbitraje para acceder al control del bus en el resto de la operación, es decir, tomar el bu para solicitar una lectura o una escritura y después tomar el bus de nuevo para realizar la lectura o la escritura.
En el caso de que haya líneas dedicadas a datos y a direcciones, la dirección se sitúa en el bus de direcciones y se mantiene ahí, mientras que el dato se ubica en el bus de datos. En una escritura, el maestro pone el dato en el bus de datos tan pronto como se han estabilizado las líneas de dirección y el esclavo ha podido reconocer su dirección. En una operación de lectura, el esclavo pone el dato en el bus de datos tan pronto como haya reconocido su dirección y disponga del mismo.
También son posibles operaciones combinadas. Lectura-modificación-escritura es una lectura seguida inmediatamente de una escritura en la misma dirección. La dirección se proporciona una sola vez al comienzo de la operación. La operación completa es generalmente indivisible, de cara a evitar cualquier acceso al dato por otros posibles maestros del bus. El objetivo de esta posibilidad es proteger los recursos de memoria compartida en un sistema con multiprogramación.
La lectura-despues de- escritura es una operación indivisible, que consiste en una escritura seguida inmediatamente de una lectura en la misma dirección. La operación de lectura se puede realizar con el propósito de comprobar el resultado.algunos buses ambien permiten transferencias de bloques de datos. En este caso, un ciclo de dirección viene seguido por n ciclos de datos.

Repaso de ejecución de instrucciones:
Al comienzo de cada ciclo de instrucción la CPU capta una instrucción de memoria. Se utiliza un registro llamado contador de programa (PC) para seguir la pista de la instrucción que debe captar a continuación. A no ser que indique otra cosa, la CPU siempre incrementa el PC después de captar cada instrucción de forma que captará la siguiente instrucción de la secuencia.
La instrucción captada se almacena en un registro de la CPU conocido como registro de instrucción (IR). La instrucción se escribe utilizando un código binario que especifica la acción que debe realizar la CPU. La CPU interpreta la instrucción y lleva a cabo la acción requerida, esta puede ser 4:

- **procesador-memoria:** deben transferirse datos desde la CPU a la memoria, o desde la memoria a la CPU.
- **Procesador-E/S:** deben transferirse datos a o desde el exterior mediante transferencias entre la CPU y un módulo de E/S.

- **Procesamiento de datos:** la CPU ha de realizar alguna operación aritmética o lógica con los datos.
- **Control:** una instrucción puede especificar que la secuencia de ejecución se altere.

Organización del procesador
para comprender la organización de la cpu, consideremos los requisitos fijados para la cpu, las cosas que debe hacer:

- *captar instrucción:* la cpu lee una instrucción de la memoria
- *interpretar instrucción:* la instrucción se decodifica para determinar que acción es necesaria
- *captar datos:* la ejecución de una instrucción puede exigir leer datos de la memoria o de un módulo de E/S
- *procesar datos:* la ejecución de una instrucción puede exigir llevar a cabo alguna operación aritmética o lógica con los datos
- *escribir datos:* los resultados de una ejecución pueden exigir escribir datos en la memoria o en un módulo de E/S

para hacer estas cosas, la cpu necesita almacenar algunos datos temporalmente. La posición de la última instrucción de forma que pueda saber a donde ir a buscar la siguiente. Necesita almacenar instrucciones y datos temporalmente mientras una instrucción está ejecutándose.

Los principales componentes de la cpu son la unidad aritmético lógica o ALU, y una unidad de control. La unidad de control controla las transferencias de datos hacia dentro y hacia fuera de la cpu y el funcionamiento de la ALU. Además una memoria interna mínima que consta de un conjunto de posiciones de almacenamiento llamadas registros.

Organización de los registros
un computador emplea una jerarquía de memoria. En los niveles más altos de la jerarquía, la memoria es mas rapida, mas pequeña y más cara. Dentro de la cpu hay un conjunto de registros que funciona como un nivel de memoria, por encima de la memoria principal y de la caché en la jerarquía. Los registros de la cpu son de dos tipos registros visibles para el usuario: permiten al programador de lenguaje máquina o ensamblador, minimizar las referencias a memoria principal cuando optimiza el uso de registros.

Registros de control y de estado: son utilizados por la unidad de control para controlar el funcionamiento de la cpu y por programas privilegiados del sistema operativo para controlar la ejecución de programas.

Registros visibles para el usuario: un registro visible para el usuario es uno que puede ser referenciado por medio del lenguaje máquina, que ejecuta la cpu. Podemos clasificarlos en las siguientes categorías:
uso general, datos, direcciones, códigos de condición.

Los registros de uso general pueden ser asignados por el programador a diversas funciones. Cualquier registro de uso general puede contener el operando para cualquier código de operación. Esto proporciona una utilización de registros de autentico uso

general. Con frecuencia, sin embargo existen restricciones. Por ejemplo, puede haber registros específicos para operaciones en coma flotante y operaciones de pila.

En algunos casos, los registros de uso general pueden ser utilizados para funciones de direccionamiento. En otros casos hay una separación parcial o total entre registros de datos y registros de direcciones.

Los registros de datos pueden usarse únicamente para contener datos, y no se pueden emplear en el cálculo de una dirección de operando.

Los registros de dirección pueden ser de uso más o menos general o pueden estar dedicados a un modo de direccionamiento particular. En otros, se pueden citar los siguientes ejemplos:

- punteros de segmento: un registro de segmento contiene la dirección de la base del segmento. Puede haber múltiples registros: por ejemplo, uno para el sistema operativo y otro para el proceso actual.
- Registros índice: se usan para direccionamiento indexado y pueden ser auto indexados.
- Puntero de pila: si existe direccionamiento a pila visible al usuario, la pila está normalmente en memoria y ha un registro dedicado que apunta a la cabecera de esta. Esto permite un direccionamiento implícito. Es decir apilar y desapilar y otras instrucciones de la pila que no necesitan contener un operando explícito referente a ella.

Hay aquí varias cuestiones de diseño a estudiar. Una importante, es si usar registros de uso completamente general o si especializar su uso. Con el uso de registros especializados, generalmente puede quedar implícito en el código de operación a que tipo de registro se refiere un determinado campo de operando. El campo de operando solo debe identificar uno de entre un conjunto de registros especializados en lugar de uno de entre todos los registros, lo cual ahorra bits. Por otra parte esta especialización limita la flexibilidad del programador. No hay una óptima y definitiva solución a este problema de diseño pero como se mencionó la tendencia parece ir hacia el uso de registros especializados.
Otro problema de diseño es el número de registros de uso general o de datos más direcciones que tienen que incluirse. Esto afecta al diseño del repertorio de instrucciones ya que más registros requieren más bits para el campo de operando. Parece óptimo alrededor de entre 8 y 32 registros. Menos registros se traducen en más referencias a memoria; más registros no reducen las referencias a memoria. Sin embargo una nueva aproximación, que saca partido al uso de cientos de registros se manifiesta en algunos sistemas risc.
Por ultimo, está la cuestión de la longitud de los registros. Los registros que han de contener direcciones, han de ser lo suficientemente grandes como para albergar la dirección más grande. Los registros de datos deben ser capaces de contener valores de la mayoría de tipos de datos. Algunas máquinas permiten que dos registros contiguos sean usados como uno solo para contener valores de doble longitud.
Una última categoría de registros que es al menos parcialmente visible al usuario contiene códigos de condición. Los códigos de condición son bits fijados por el hardware de la cpu como resultado de alguna operación.

Por ejemplo, una operación aritmética puede producir un resultado positivo, negativo, nulo o con desbordamiento. Además de almacenarse el propio resultado en un registro o en la memoria se obtiene también un código de condición. El código puede ser examinado con posterioridad ,como parte de una operación de bifurcación condicional.

Los bits de códigos de condición se reúnen en uno o más registros. Normalmente forman parte de un registro de control. Por lo general, las instrucciones máquina permiten que estos bits sean leidos por referencia implícita pero no pueden ser alterados por el programador.
En algunas máquinas, una llamada a una subrutina dará lugar a la salvaguarda automática de todos los registros visibles por el usuario que serán restablecidos en el retorno de la subrutina. La cpu lleva a cabo la salvaguarda y restablecimiento como parte de la ejecución de las instrucciones de llamada y retorno. Esto permite a cada subrutina usar independientemente los registros visibles por el usuario. En otras máquinas, es responsabilidad del programador guardar los contenidos de los registros visibles para el programador relevantes, antes de la llamada a subrutina, teniendo que incluir en el programa instrucciones para este fin.

Registros de control y de estado
hay diversos registros de la cpu que se emplean para controlar su funcionamiento. La mayoría de ellos, en la mayor parte de las máquinas, no son visibles para el usuario. Algunos de ellos pueden ser visibles a instrucciones máquina ejecutadas en un modo de control o de sistema operativo.
Son esenciales 4 registros para la ejecución de una función:

- *Contador de programa, PC:* contiene la dirección de la instrucción a captar
- *Registro de instrucción, IR:* contiene la instrucción captada más recientemente
- *Registro de dirección de memoria,MAR:* contiene la dirección de una posición de memoria.
- *Registro intermedio de memoria, MBR:* contiene la palabra de datos a escribir en memoria o la palabra leída más recientemente.

La cpu actualiza el contador de programa después de cada captación de instrucción de manera que siempre apunta a la siguiente instrucción a ejecutar. Una instrucción de bifurcación o salto también modificará el contenido de pc. La instrucción captada se carga en IR, donde son analizados el código de operación y los campos de operando. Se intercambian datos con la memoria por medio de MAR y de MBR,. En un sistema con organización de bus, MAR se conecta directamente al bus de direcciones y MBR directamente al bus de datos. Los registros visibles por el usuario intercambia repetidamente datos con MBR.
Los cuatro registros que se acaban de mencionar se usan para la transferencia de datos entre la cpu y la memoria. Dentro de la cpu, los datos tienen que ofrecer a la ALU para su procesamiento. La ALU puede tener acceso directo a MBR y a los registros visibles para el usuario. Como alternativa, puede haber registros intermedios adicionales en torno a la ALU, estos registros sirven como registros de entrada y salida de la ALU, e intercambian datos con MBR y los registros visibles para el usuario.
Todos los diseños de cpu incluye un registro o un conjunto de registros, conocidos a menudo como palabra de estado del programa(PSW) que contiene información de estado. PSW contiene normalmente códigos de condición. Además de otra información de estado.

Entre los campos o indicadores comunes se incluyen los siguientes:

- signo: contiene el bit de signo del resultado de la última operación aritmética
- cero: puesto a uno cuando el resultado es 0.
- acarreo: puesto a uno si una operación da lugar a un acarreo(suma) o adeudo(resta) del bits más significativo. Se usa en operaciones aritméticas multipalabra.
- Igual: puesto a uno si el resultado de una comparación lógica es la igualdad.
- Desbordamiento: usado para indicar un desbordamiento aritmético.
- Interrupciones habilitadas/inhabilitadas: usado para permitir o inhabilitar interrupciones.
- Supervisor: indica si la cpu funciona en modo supervisor o usuario. Únicamente en modo supervisor se pueden ejecutar ciertas instrucciones privilegiadas y se puede acceder a ciertas áreas de memoria.

Además de PSW puede existir un puntero a un bloque de memoria que contenga información de estado adicional. En las máquinas que usan interrupciones vectorizadas puede existir un registro de vector de interrupción. Si se utiliza una pila para llevar a cabo ciertas funciones, se necesita un puntero de pila del sistema. En un sistema de memoria virtual se usa un puntero a la tabla de páginas. Por último pueden emplearse registros para el control de operaciones de E/S.

En el diseño de la organización de los registros de control y estado entran en juego varios factores. Una cuestión importante es el soporte del sistema operativo. Algunos tipos de información de control son de utilidad específica para el sistema operativo. Si el diseñador del a cpu posee una comprensión funcional del sistema operativo que se va a utilizar, la organización de los registros puede adaptarse hasta cierto punto a ese s.o.

Otra decisión clave en el diseño es la distribución de información de control entre registros y memoria. Es frecuente dedicar los primero(más bajos) pocos cientos o miles de palabras de memoria para fines de control. El diseñador debe decidir cuánta información de control debiera estar en registros y cuana en memoria. Surge el habitual compromiso entre coste y velocidad.

Ejemplos de organizaciones de registros de microprocesadores

el mc68000 distribuye sus registros de 32 bits en ocho de datos y nueve de dirección. Los ocho registros de datos se usan principalmente para manipulación de datos y también se usan en direccionamiento como registros índice. El ancho de los registros permite operaciones con datos de 8,16,32 bits según determine el código de operación. Los registros de direcciones contienen direcciones de 32 bits(no hay segmentación); dos de estos registros se usan también como punteros de pila uno para los usuarios y el otro para el sistema operativo, dependiendo del modo de ejecución en curso. Los dos registros se referencian como 7 dado que solo uno de ellos se puede usar en un instante dado. El mc68000 también incluye un contador de programa de 32 bits y un registro de estado de 16bits.

El equipo de motorola hizo un repertorio de instrucciones sin registros de uso especial. Su interés por la eficiencia del código los condujo a dividir los registros en dos componentes funcionales, ahorrando un bit en cada campo de especificación de registro, esto parece un compromiso razonable entre generalidad total y compacidad del código.

El intel 8086 usa un enfoque diferente. Cada uno de los registros tiene un uso especial, aunque algunos registros se pueden emplear también para un uso general. El 8086 contiene cuatro registros de datos de 16 bits que son direccionables como registros de 16 bits o como registros de bytes y cuatro registros punteros e índices de 16bits. Los registros de datos pueden utilizarse como registros de uso general en algunas instrucciones. En otras los registros se usan implícitamente. Por ejemplo una instrucción de multiplicación siempre usa el acumulador. Los cuatro registros punteros se usan también en algunas operaciones: cada uno contiene un desplazamiento dentro de un segmento. Hay también cuatro registros de segmento de 16 bits. Tres de ellos cuatro registros de segmento se usan de una forma dedicada e implícita para apuntar al segmento de la instrucción en curso(útil para instrucciones de salto) a un segmento que contenga datos y a un segmento que contenga una piula. Estos usos dedicados e implicitos proporcionan una codificación compacta con el coste de una flexibilidad reducida. El 8086 incluye también un puntero de instrucción y un conjunto de indicadores de estado y control de 1 bit.

El ciclo de instrucción

un ciclo de instrucción incluye los siguiente subciclos:

- captación: llevar la siguiente instrucción de la memoria a la cpu
- ejecución: interpretar el código de operación y llevar a cabo la operación indicada.
- Interrupcion: si las interrupciones están habilitadas y ha ocurrido una interrupción, salvar el estado del proceso actual y atender la interrupción.

El ciclo indirecto

una instrucción puede involucrar a uno o más operandos en memoria, cada uno de los cuales requiere un acceso a memoria. Además, si se usa direccionamiento indirecto serán necesarios accesos a memoria adicionales.

Podemos considerar la captación de direcciones indirectas como un subciclo de instrucción más. La principal línea de actividad consiste en alternar las actividades de captación y ejecución de instrucciones. Después de que una instrucción sea captada, esta es examinada para determinar si implica algún direccionamiento indirecto. Si es así, los operandos requeridos se captan usando direccionamiento indirecto. Si es así, los operandos requeridos se captan usando direccionamiento indirecto. Tras la ejecución se puede procesar una interrupción santes de la captación de la siguiente instrucción.

Una vez que una instrucción es captada, deben identificarse sus campos de operando. Se capta entonces de la memoria cada operando de entrada, pudiendo requerir este proceso direccionamiento indirecto. Los operandos ubicados en registros no necesitan ser captados. Una vez que se ejecuta la operación, puede ser necesario un proceso similar para almacenar el resultado en la memoria principal.

Flujo de datos

durante el ciclo de captación se lee una instrucción de la memoria. Pc contiene la dirección de la siguiente instrucción que hay que captar. Esta dirección es llevada a MAR y puesta en el bus de direcciones. La unidad de control solicita una lectura de memoria y el resultado se pone en el bus de direcciones. La unidad de control solicita una lectura de memoria y el resultado se pone en el bus de datos, se copia en el MBR y después se lleva a IR. Mientras tanto, pc se incrementa en 1 como preparación para la siguiente captacion.

Una vez concluido el ciclo de captación, la unidad de controla examina los contenidos de IR para determinar si contiene un campo de operando que use direccionamiento indirecto. Si es así, se lleva a cabo un ciclo indirecto. Los N bits más a la derecha de MBR que contienen la dirección de referencia, se transfieren a MAR. Entonces la unidad de control solicita una lectura de memoria para llevar la dirección del operando deseada a MBR.

Los ciclos de captación e indirecto son sencillos y predecibles. El ciclo de ejecución adopta muchas formas, ya que depende de cual de las diversas instrucciones maquina este en IR. Este ciclo puede implicar transferencias de datos entre registros, lectura o escritura de memoria o E/S y o la invocación a la ALU.

Del mismo modo que los ciclos de captación e indirecto, el ciclo de interrupción es simple y predecible. El contenido actual de PC tiene que ser salvado de manera que la cpu pueda reanudar su actividad normal tras la interrupción. Así el contenido de pc se transfiere a MBR para ser escrito en memoria. La posición de memoria especial reservada para este propósito se cara en MAR desde la unidad de control. Podría ser por ejemplo un puntero de pila. Pc se carga con la dirección de la rutina de interrupción. Como resultado el siguiente ciclo de instrucción comenzará captando la instrucción oportuna.

Segmentación de instrucciones

estrategia de segmentación:
la segmentación de instrucciones es similar al uso de una cadena de montaje en una fábrica, trabajando sobre los productos en varias etapas simultáneamente. A este proceso se lo llama segmentación de cauce(pipelining), en un extremo se aceptan nuevas entradas antes de que algunas entradas aceptadas con anterioridad aparezcan como saldías en el otro extremo.
Una instrucción tiene varias etapas, que tienen lugar secuencialmente. Puede pensarse en la utilización de la segmentación.
Consideremos la subdivisión del procesamiento de una instrucción en dos etapas: captación de instrucción y ejecución de instrucción. Hay periodos en la ejecución de una instrucción en los que no se accede a memoria principal. Este tiempo podría utilizarse en capta la siguiente instrucción en paralelo con la ejecución de la actual. El cauce tiene dos etapas independientes. La primera etapa capta una instrucción y la almacena en un buffer, cuando la segunda etapa está libre, la primera le pasa la instrucción almacenada. Mientras que la segunda etapa ejecuta la instrucción. La primera etapa utiliza lagun ciclo de memoria no usado para captar y almacenar la siguiente instrucción. A esto se llama prebúsqueda o pre captación de instrucción o solapamiento de la captación.
Este proceso acelerará la ejecución de instrucciones. Si las etapas de captación y ejecución fueran de igual duración el tiempo de ciclo de instrucción se reduciría a la mitad. Sin embargo si miramos más atentamente a este cauce, veremos que esta duplicación de velocidad de ejecución es poco probable por dos razone:

1. el tiempo de ejecución será generalmente más largo que el tiempo de captación. La ejecución implica la lectura y almacenamiento de operando y la realización de alguna operación. Así, la etapa de captación puede tener que estará alguna tiempo antes de que pueda vaciar su buffer.

2.una instrucción de bifurcación condicional hace que la dirección de la siguiente instrucción a captar sea desconocida. La etapa de captación debe esperar hasta que reciba la dirección del a siguiente instrucción desde la etapa de ejecución. La etapa de ejecución puede entonces tener que esperar mientras se capta la siguiente instrucción.

La pérdida de tiempo debida a la segunda razón puede reducirse haciendo una estimación. Una regla simple es la siguiente: cuando una instrucción de bifurcación condicional pasa de la etapa de captación a la de ejecución, l a etapa de captación capta la instrucción de memoria que sigue a la instrucción de salto entonces si el salto no se produce, no se pierde tiempo si el salto se produce. Debe desecharse la instrucción captada y captarse una nueva instrucción.

Para conseguir una mayor aceleración , el cauce debe tener más etapas. Consideremos la siguiente descomposición.

- ***Captar instrucción,FI:*** leer la supuesta siguiente instrucción en un buffer.
- ***Decodificar instrucción,DI:*** determinar el código de operación y los campos de operando.
- ***Calcular operandos,CO:*** calcular la dirección efectiva de cada operando fuente. Esto puede involucrar direccionamiento mediante un desplazamiento, indirecto a través de registro, indirecto u otras formas de calcular la dirección.
- ***Captar operandos,FO:*** captar cada operando de memoria. Los operandos en registros no tienen que ser captados.
- ***Ejecutar instrucción,EI:*** realizar la operación indicada y almacenar el resultado, si lo hay, en la posición del operando destino especificada.
- ***Escribir operando,WO:*** almacenar el resultado en memoria.

con esta descomposición, un cauce de seis etapas, puede reducir el tiempo de ejecución de 9 instrucciones, de 54 a 14 unidades de tiempo.
La temporización se establece asumiendo que cada instrucción requiere las seis etapas.además supone que todas las etapas pueden funcionar en paralelo.
La mayoría de los sistemas de memoria no permitiran esto, no obstante el valor deseado puede estar en cache o las etapas FO o WO pueden ser nulas. De este modo, casi siempre los conflictos de memoria no reducirán la velocidad de cauce.

Algunos otros factores contribuyen a limitar la mejora de prestaciones. Si las seis etapas no son de igual duración habrá cierta espera en algunas etapas del cauce, como se discutió antes con el cauce de dos etapas. Otra dificultad es la instrucción de bifurcación condicional que puede invalidar varias captaciones de instrucción. Un evento impredecible similar es la llegada de una interrupción.hasta que no se ejecuta la instrucción no hay forma de saber qué instrucción vendrá a continuación. El cauce en este ejemplo simplemente carga la siguiente instrucción secuencialmente y continua. El salto no se efectúa y obtenemos el máximo provecho en cuanto a rendimiento de este diseño.

Se presentan además otros problemas que no aparecían en nuestra organización simple de dos etapas, la etapa CO puede depender del contenido de un registro que podría verse alterado por una instrucción previa que aun este en el cauce. Podrían ocurrir otros conflictos con registros y con memoria. El sistema tiene que incluir lógica para tener en cuenta este tipo de conflictos.

Puede parecer que cuanto mayor sea el número de etapas en el cauce, mas rapida sera la velocidad de ejecución. Hay dos factores que frustran este patrón de diseño de latas prestaciones:

1.en cada etapa del cauce hay algun gasto extra debido a la transferencia de datos de buffer a buffer, y a la realización de varias funciones de preparacion y distribucion. Este gasto adicional puede prolongar sensiblemente el tiempo de ejecución total de una instrucción aislada. Esto es importante cuando las instrucción secuenciales son lógicamente dependientes,bien a causa de un uso abundante de bifurcaciones, bien debido a dependencias de acceso a memoria.

2.la cantidad de logica de control neesaria para manejar dependencias de memoria y registros y para optimizar el uso del cauce aumenta enormemente con el numero de etapas. Esto puede llevar a una situación donde la lógica para controlar el paso entre etapas sea más compleja que las etapas controladas.
La segmentación de instrucciones es una poderosa técnica para aumentar las prestaciones pero requiere un diseño cuidadoso si se quieren obtener resultados óptimos con una complejidad razonable.

tratamiento de saltos
el principal obstáculo, es la instrucción de bifurcación condicional. Hasta que la instrucción no se ejecuta realmente, es imposible determinar si el salto se producirá o no. se considera varias aproximaciones en el tratamiento de bifurcaciones condicionales:

- flujos múltiples
- pre captar el destino del salto
- buffer de bucles
- predicción de saltos
- salto retardado

flujos múltiples: un cauce simple sufre penalización por las instrucciones de bifurcaciones, porque debe escoger una de las dos instrucciones a captar a continuación y puede hacer la elección equivocada., una solucion es duplicar las partes iniciales del cauce y dejar que éste capte las dos instrucciones utilizando los dos caminos. Esta aproximación tiene dos problemas:

1.con cauces múltiples hay retardos debidos a la competencia por el acceso a los registros y a la memoria.

2. pueden entrar en el cauce instrucciones de bifurcación adicionales antes de que se resuelva la decisión en la bifurcación original cada una de esas instrucciones exige un flujo adicional.

Pre Captar el destino del salto: cuando se identifica una instrucción de bifurcación condicional, se precapta la instrucción destino del salto, además de la siguiente a la bifurcación. Se guarda entonces esta instrucción hasta que se ejecute la instrucción de bifurcación. Si se produce el salto el destino ya habra sido precaptado.

Buffer de bucles: un buffer de bucles es una memoria pequeña de gran velocidad, gestionada por la etapa de captación de instrucción el cauce, que contiene secuencialmente, las n instrucciones captadas más recientemente. Si se va a producir un salto, el hardware comprueba en primer lugar si el destino del salto está en el buffer. En ese caso la siguiente instrucción se capta del buffer.

El buffer de bucles tiene tres utilidades:

1.con el uso de precaptacion el buffer de bucles se anticipa, almacenado algunas instrucciones que secuencialmente están después de la dirección de donde se capta la instrucción actual. De este modo las instrucciones que se capten secuencialmente estarán disponibles sin el tiempo de acceso a memoria habitual.

2.si ocurre un slato a un nuevo destino a solo unas pocas posicoines mas alla de la direcicon de la insturcion de bifurcacion, el destino estara en el buffer. Esto es útil para el caso bastante común de las secuencias if-then e if-then-else

3.esta estrategia se acomoda particularmente bien al tratamiento de bucles o iteraciones de ahí el nombre de buffer de bucles. Si el buffer de bucles es lo suficientemente grande como para contener todas las instrucciones de un bucle, entonces esas instrucciones solo necesitan ser captadas de la memoria una vez, durante la primera iteración. En las siguientes iteraciones, todas las instrucciones necesarias se encuentran ya en el buffer.

El buffer de bucles es similar a una caché de instrucciones. Las diferencias son que el buffer de bucles solo guarda instrucciones consecutivas y que es mucho más pequeño en tamaño y de menor coste

predicción de saltos
se pueden usar varias técnicas para predecir si un salto se va a producir:

- predecir que nunca se salta
- predecir que siempre se salta
- predecir según el código de operación
- conmutador saltar/no saltar
- tabla de historia de saltos.

Las tres primeras soluciones son estáticas, no dependen de la historia de la ejecución, las dos últimas aproximaciones son dinámicas, dependen de la historia de la ejecución.
Las dos primeras son las más simples, una asume que el salto no se producirá y continuará captando instrucciones secuencialmente y la otra que el salto se producirá y siempre captará la instrucción destino del salto.un estudio reveló que los saltos condicionales se producen más del 50% de las veces.
La ultima aproximación estática toma la decisión basándose en el código de operación de la instrucción de bifurcación. El procesador asume que el salto se producirá para ciertos códigos de operación de bifurcación y no para otros. Informa sobre tasas de acierto mayores del 75 por ciento con esta estrategia.
Las estrategias dinámicas intentan mejorar la exactitud de la predicción a cada instrucción de bifurcación pueden asociarse uno o más bits que reflejan su historia reciente. Estos bits son referenciados como un conmutador saltar/no saltar, que dirige al procesador a tomar una determinada decisión la próxima vez que encuentre la instrucción. Estos bits se guardan temporalmente en un almacenamiento de alta velocidad.otra posibilidad es mantener una pequeña tabla para las instrucciones de

bifurcación ejecutadas recientemente, con uno o más bits en cada elemento de la tabla el procesador prodira acceder a esa tabla asociativamente como a una caché o usando los bits de orden inferior de la dirección de la instrucción de bifurcación.

La utilización de bits de historia tiene un inconveniente: si la decisión que se toma es efectuar el salto, la instrucción destino no puede captarse hasta que su dirección que es un operando de la instrucción de bifurcación condicional sea codificada. Se podría lograr una mayor eficiencia si la instrucción captada pudiera iniciarse tan pronto como se tome la decisión de salto. Para ello se debe guardar más información en lo que se conoce como un buffer de destino de saltos o tabla de historia de saltos.
La tabla de historia de salto es una pequeña memoria caché asociada con la etapa de captación de instrucción del cauce. Cada elemento de la tabla consta de tres campos": la dirección de una instrucción de bifurcación, un determinado número de bits de historia que guardan el estado de uso de esta instrucción e información sobre la instrucción destino.

Salto retardado: se pueden mejorar las prestaciones de un cauce reordenando automáticamente las instrucciones de un programa, de forma que las instrucciones de salto tengan lugar después de lo realmente deseado.

Ejecución solapada ("pipeline"): PREGUNTAR A CÁTEDRA
Su aplicación en procesadores contemporáneos: PREGUNTAR A CÁTEDRA
Análisis de performance: PREGUNTAR A CÁTEDRA
Arquitecturas reconfigurables: conceptos: PREGUNTAR A CÁTEDRA

UNIDAD 2

Repaso de máquinas que ejecutan instrucciones.
Conforme el coste del hardware ha disminuido, el coste relativo del software ha ido creciendo. Junto con esto, la escasez permanente de programadores ha hecho subir los costes del software en términos absolutos. De ahí que el coste principal del ciclo de vida de un sistema sea el software, no el hardware. Otro que se añade al coste y a otros inconvenientes, es la falta de fiabilidad: es frecuente que los programas, tanto de sistema como de aplicación, continúen mostrando nuevos errores despues de años de funcionamiento.

Ejemplificación en procesadores típicos.
Análisis del conjunto de instrucciones de procesadores de uso comercial.

Concepto de máquinas CISC y RISC.
Los siguientes son algunos avances desde la invención del computador:

- el concepto de familia: introducido por ibm en su system/360 en 1964, y seguido poco después por dec, en su PDP-8. El concepto de familia separa la arquitectura de una máquina de su implementación. Se ofrece un conjunto de computadores, con distintas características en cuanto a precio y prestaciones, que presentan al usuario la misma arquitectura. Las diferencias en precio y prestaciones se deben a las distintas implementaciones de la misma arquitectura.
- unidad de control microprogramada, introducida por ibm en system 360 en 1964, la microprogramación facilita la tarea de diseñar e implementar la unidad de control y da soporte al concepto de familia
- memoria caché: una manera de introducir paralelismo en la naturaleza esencialmente secuencial de un programa constituido por instrucciones máquina. Dos ejemplos son la segmentación de instrucciones el procesamiento vectorial.
- múltiples procesadores esta categoría cubre diferentes organizaciones y objetivos.

A esta lista hay que anadir una de las más interesantes y potencialmente de las características: la arquitectura de computador de repertorio reducido de instrucciones (risc). Se aparta de manera drástica de la tendencia histórica en la arquitectura del procesador. Un análisis de la arquitectura risc involucra a la mayoría de los asuntos importantes de organización y arquitectura de computadores.
Aunque los sistemas risc se han definido y diseñado de diversas formas por parte de diferentes grupos, los elementos claves compartidos por la mayoría de los diseños son:

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

Características de la ejecución de instrucciones.
Uno de los aspectos relacionados con los computadores que ha evolucionado de manera más visible es el de los lenguajes de programación. Conforme el coste del hardware ha disminuido, el coste relativo del software ha ido creciendo. Junto con esto, la escasez permanente de programadores ha hecho subir los costes del software en términos absolutos. De ahí que el coste principal del ciclo de vida de un sistema sea el software ,

no el hardware. Otro elemento que se añade al coste y a otros inconvenientes, es la falta de fiabilidad: es frecuente que los programas, tanto de sistema como de aplicación, continúen mostrando nuevos errores después de años de funcionamiento.
La respuesta de los investigadores y de la industria fue desarrollar lenguajes de programación de alto nivel más potentes y complejos . Estos lenguajes de alto nivel permiten al programador expresar los algoritmos de manera más concisa, se encargan de gran parte de los pormenores,y a menudo apoyan de manera natural al programación estructurada o el diseño orientado a objetos.
La solución ocasionó otro problema conocido como salto semántico la diferencia entre las operaciones que proporcionan los High level Languages y las que proporciona la arquitectura del computador. Los supuestos síntomas de este salto o hueco incluían ineficiencia de la ejecución, tamaño excesivo del programa en lenguaje máquina y complejidad de los compiladores. Los diseñadores respondieron con arquitecturas que se proponen cerrar ese hueco. Sus características principales incluyen repertorios de instrucciones grandes, docenas de modos de direccionamiento y varias sentencias HLL implementadas en hardware. Un ejemplo de esto último es la instrucción máquina CASE del VAX. Tales repertorios complejos de instrucciones están pensados para:
facilitar el trabajo del escritor de compiladores.
Mejorar la eficiencia de la ejecución ya que las secuencias complejas de operaciones pueden implementarse en microcódigo.
Dar soporte a HLL aún más complejos y sofisticados.

Mientras se hicieron algunos estudios durante años para determinar las características y patrones de ejecución de las instrucciones maquina generadas por programas escritor en HLL. Los resultados de estos estudios movieron a algunos investigadores a buscar una aproximación diferente, a saber, realizar una arquitectura que diera soporte a los HLL más sencillos en lugar de los más complejos.

Para comprender la línea de razonamiento de los partidarios de los RISC, comenzamos con una breve revisión de las características de la ejecución de instrucciones. Los aspectos cuyo cálculo tiene interés son los siguientes:
operaciones realizadas: determinan las funciones que lleva a cabo el procesador y su interacción con la memoria.
Operandos usados: los tipos de operandos y su frecuencia de uso determina la organización de memoria para almacenarlos y los modos de direccionamiento para accederlos.
Secuenciamiento de la ejecución: determina la organización del control y del cauce.

Operaciones:
existe una gran concordancia en los resultados de esta mezcla de lenguajes y aplicaciones. Las sentencias de asignación predominan, lo cual indica que el sencillo movimiento de datos tiene mucha importancia. También predominan las sentencias condicionales (if,loop).estas sentencias se implementan en el lenguaje máquina con alguna instrucción del tipo comparar y saltar. Esto indica que el mecanismo incluido en el repertorio de instrucciones para el control del secuenciamiento es importante.
Estos resultados son instructivos para el diseñador del repertorio de instrucciones máquina, porque le dicen que tipos de sentencias tiene lugar más a menudo, y por consiguiente deben ser implementadas de una forma óptima. No obstante, estos resultados no revelan que sentencian consumen más tiempo en la ejecución de un programa típico. Es decir, dado un programa en lenguaje máquina compilado, que

sentencias del lenguaje fuente provocan la ejecución de la mayoría de las instrucciones en lenguaje máquina?

operandos

la mayoría de las referencias se hacen a variables escalares simples. más del 80% de los datos escalares eran variables locales. asimismo, las referencias a matrices/estructuras requieren una referencia previa a su índice o puntero que nuevamente suele ser un dato escalar local. de este modo, hay un predominio de referencias a operandos escalares, y estos están muy localizados.

la importancia de una arquitectura que se preste a un rápido acceso a operandos, dado que es una operación que se realiza con mucha frecuencia. un candidato fundamental a optimizar es el mecanismos de almacenamiento y acceso a variables escalares locales.

llamadas a procedimientos

hemos visto que las llamadas y retornos de procedimientos constituyen un aspecto importante de los programa en HLL. son las operaciones que consumen más tiempo en programas en HLL compilados. así, será ventajoso considerar formas de implementar estas operaciones eficientemente. hay dos aspectos importantes: el número de parámetros y variables que trata un procedimiento y la profundidad de anidamiento.

el estudio de tanenbaum encontró que al 98% de los procedimientos llamados dinámicamente se les pasaban menos de seis argumentos, y que el 92% de ellos usaban menos de seis variables escalares locales. el equipo de risc de berkeley presentó resultados parecidos. estos resultados muestran que el número de palabras necesarias por cada activación de un procedimiento no es muy grande. los resultados expuestos anteriormente indicaban que una alta proporción de referencias a operandos, se hacía a variables locales escalares. estos estudios muestran que aquellas referencias se limitan, de hecho a relativamente pocas variables.

el mismo grupo de berkeley también estudió los patrones seguidos por las llamadas y retornos de procedimientos de programas en HLL. encontraron que es poco común tener una larga secuencia ininterrumpida de llamadas a procedimientos seguida por la correspondiente secuencia de retornos. vierons, más bien que un programa permanece confinado en una ventana bastante estrecha de profundidad de invocación a procedimientos. tales resultados refuerzan la conclusión de que las referencias a operandos están muy localizadas.

consecuencias

intentar realizar una arquitectura de repertorio de instrucciones cercano al de los HLL, no es la estrategia de diseño más efectiva. en su lugar, se puede ofrecer mejor soporte para los HLL optimizando las prestaciones de las características de los programas típicos en HLL que más tiempo consumen.

los elementos que caracterizan las arquitecturas risc son:

1. un gran número de registros, o un compilador que optimice el tratamiento de estos. su finalidad es optimizar las referencias a operandos. los estudios recién discutidos muestran que hay varias referencias por instrucción de HLL, y que hay una alta proporción de sentencias de transferencia(asignación). esto asociado con la localidad y predominio de las referencias a datos escalares, sugiere que las prestaciones pueden mejorarse reduciendo las referencias a memoria, a costa de más referencias a registros. debido a la localidad de estas referencias, parece práctico un conjunto de registros ampliado.

2. hay que prestar una cuidadosa atención al diseño de los cauces de instrucciones. debido a la alta proporción de instrucciones de bifurcación condicional y de llamada a procedimientos, un cauce de instrucciones sencillo será ineficiente. esto se manifiesta debido a que una gran proporción de instrucciones de bifurcación condicional y de llamada a procedimientos, un cauce de instrucciones sencillo será ineficiente. esto se manifiesta debido a que una gran proporción de instrucciones se precaptan, pero nunca llegan a ejecutarse

3. es recomendable un repertorio de instrucciones simplificado.

Utilización de un amplio banco de registros

hemos visto que hay una gran proporción de sentencias de asignación en programas escritos en HLL, y muchas de estas son de la forma sencilla a<-b. además hay un número significativo de acceso a operandos por cada sentencia de HLL. si juntamos estos resultados con el hecho de que la mayoría de los accesos se hacen a datos escalares locales, se puede pensar en una fuerte dependencia de almacenamiento en registros.

la razón de que esté indicado ese almacenamiento en registros es que estos constituyen el dispositivo de almacenamiento más rápido disponible; más que la memoria principal y que la cache. el banco de registros es pequeño físicamente, esta en el mismo chip que la alu y la unidad de control, y emplea direcciones mucho más cortas que las de la caché y la memoria. por tanto, se necesita una estrategia que permita que los operandos a los que se accede con mayor frecuencia se encuentren en registros y que se minimicen las operaciones registro-memoria.

son posibles dos aproximaciones básicas: una basada en software y la otra en hardware . la aproximación por software consiste en confiar al compilador la maximización del uso de los registros. el compilador intentará asignar registros a las variables que se usen más en un periodo de tiempo dado. esta aproximación requiere el uso de sofisticados algoritmos de análisis de programas. la aproximación por hardware consiste sencillamente en usar más registros, de manera que mas variables puedan mantenerse en registros durante periodos de tiempo más largo.

ventanas de registros

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

ya que la mayoría de las referencias a operandos se hacen a datos escalares locales, la solución evidente es almacenar estos en registros, reservando tal vez varios registros para variables globales. el problema es que la definición de local varía con cada llamada y retorno de procedimiento , operaciones que suceden con frecuencia. en cada llamada, las variables locales deben guardarse desde los registros en la memoria, para que los registros puedan ser reutilizados por el programa llamado. además deben pasarse parámetros. en el retorno, las variables del programa padre tienen que ser restablecida(cargadas de nuevo en los registros) y hay que devolver los resultados al programa padre.

la solución en primer lugar, un procedimiento típico emplea solo unos pocos parámetros de llamada y variables locales. en segundo lugar, la profundidad de activación de procedimientos fluctúa dentro de un rango relativamente pequeño. para explotar estas propiedades, se usan múltiples conjuntos pequeños de registros, cada uno asignado a un procedimiento distinto. una llamada a un procedimiento hace que el procesador conmute automáticamente a una ventana de registros distinta de tamaño fijo, en lugar de salvaguardar los registros en memoria. las ventanas de procedimientos adyacentes están parcialmente solapadas para permitir el paso de parámetros.

en todo momento solo es visible una ventana de registros, y se direcciona como si fuera el único conjunto de registros, por ejemplo direcciones 0 a N-1,. la ventana se divide en tres áreas de tamaño fijo. los registros de parámetros contienen parámetros pasados al procedimiento en curso desde el procedimiento que lo llamo, y los resultados a devolver a este. los registros locales se usan para variables locales, según la asignación que realice el compilador. los registros temporales se usan para intercambiar parámetros y resultados con el siguiente nivel más bajo, el procedimiento llamado por el procedimiento en curso. los registros temporales de un nivel son físicamente los mismos que los registros de parámetros del nivel más bajo adyacente. este solapamiento posibilita que los parámetros se pasen sin que exista una transferencia de datos real.

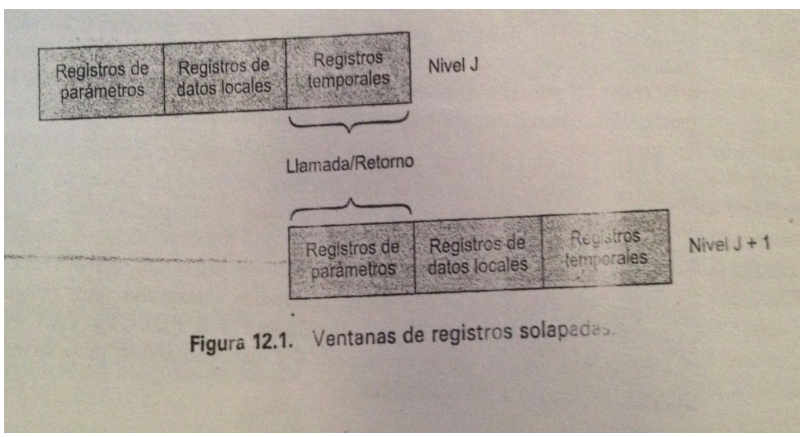


Figura 12.1. Ventanas de registros solapadas.

para manejar cualquier posible patrón de llamadas y retornos, el número de ventanas de registros tendría que ser ilimitado. en lugar de eso, las ventanas de registros se pueden usar para contener unas pocas,las más recientes, activaciones de procedimientos. las activaciones más antiguas han de salvaguardarse en memoria y restaurarse más tarde, cuando la profundidad de anidamiento disminuya. de este modo, la organización real del banco de registros es un buffer circular de ventanas solapadas.

esta organización se muestra en la figura 12.2 que representa un buffer circular de seis ventanas. el buffer está lleno hasta una profundidad de 4(A llamo a B; B llamo a C; y C ha llamado a D), siendo D el procedimiento activo. el puntero de ventana en curso (CWP) apunta a la ventana del procedimiento activo actualmente. las referencias de una instrucción máquina a registros se transforman con este puntero para determinar el registros físico real. el puntero de ventana salvada (SWP) identifica la ventana guardada en memoria más recientemente. si el procedimiento D llama ahora al procedimiento E, los argumento para E se colocan en los registros temporales de D(el solapamiento entre w3 y w2) y el CWP se avanza en una ventana.

si el procedimiento E hace después una llamada al procedimiento F, esta no puede llevarse a cabo con el estado actual del buffer. ello se debe a que la ventana de F se solapa con la de A. si F comienza a cargar sus registros temporales , como preparación para una llamada esto sobreescribirá los registros de parámetros de A . por tanto, cuando al incrementar CWP este llega a ser igual a SWP , tiene lugar una interrupción, y la ventana de A se guarda. solo hay que guardar las dos primeras partes. después se incrementa el SWP y la llamada a F sigue adelante. en los retornos ocurre una interrupción similar. por ejemplo, con posterioridad a la activación de F, cuando B retorna A, el CWP se decrementa y llega a ser igual a SWP. esto causa una interrupción, cuyo resultado es la restauración del a ventana de A.

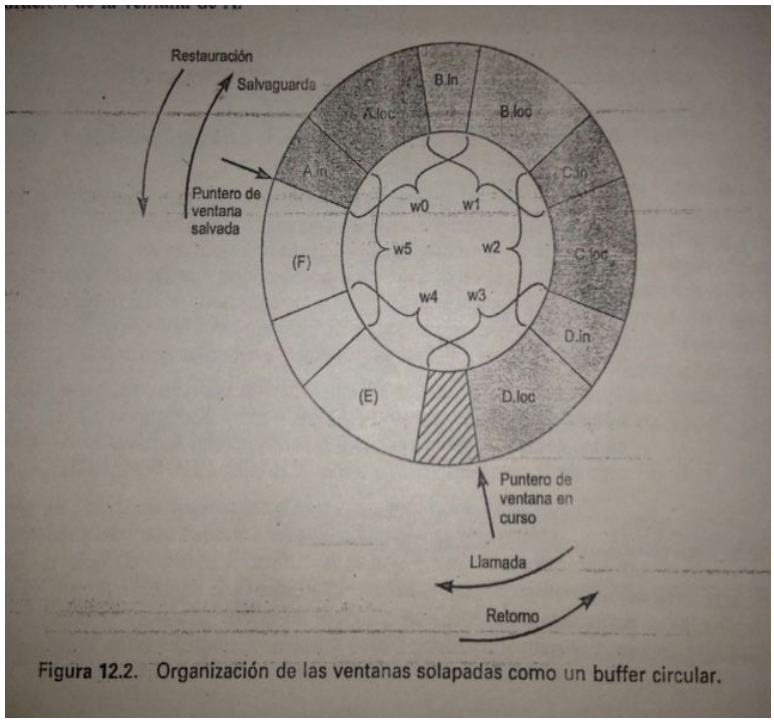


Figura 12.2. Organización de las ventanas solapadas como un buffer circular.

de lo anterior se puede deducir que un banco de registros de N ventana puede contener solo N-1 activaciones de procedimientos. el valor de N no tiene que ser muy grande. como mencionamos en el apéndice 4A, un estudio encontró que con 8 ventanas se requiere una salvaguarda o restauración solo en el 1 % de las llamadas o retornos. los computadores risc de berkeley usan 8 ventanas de 16 registros cada una. el computador pyramid emplea 16 ventanas de 32 registros cada una.

variables globales

el esquema de ventanas recién descrito proporciona una organización eficiente para el almacenamiento de variables escalares locales en registros. sin embargo, este esquema no trata la necesidad de almacenar las variables globales: aquellas a las que accede más de un procedimiento. se sugieren dos opciones. la primera es que el compilador asigna posiciones de memoria a las variables declaradas como globales en un HLL, y que todas las instrucciones máquina que referencian esas variables usen operandos referenciados en memoria. esto es sencillo desde los puntos de vista hardware y software(compilador). no obstante, para variables globales a las que se accede frecuentemente, este esquema es ineficiente.

una alternativa es incorporar al procesador un conjunto de registros globales. estos registros serían fijos en cuanto a su número, y accesibles a todos los procedimientos. se puede usar un esquema de numeración unificado para simplificar el formato de instrucción. por ejemplo, las referencias a los registros desde el 0 hasta el 7 pueden indicar registros globales únicos, y las referencias a los registros del 8 al 31 pueden

transformarse para seleccionar los registros físicos de la ventana en curso. esto conlleva un hardware añadido , que se encarga de adaptar la división en el direccionamiento de los registros. además, el compilador ha de decidir qué variables globales deben ser asignadas a registros.

un amplio banco de registros frente a una caché

el banco de registros organizado en ventanas funciona como un buffer pequeño y rápido que contiene un subconjunto de todas las variables que probablemente se usen mucho. desde este punto de vista, el banco de registros se comporta de manera muy similar a una memoria caché. surge por tanto la cuestión de si sería más sencillo y mejor, usar una caché y un tradicional banco de registro pequeno.

características de las organizaciones de un banco de registros amplio y de cache

banco de registros amplio:

- todos los datos escalares locales
- variables individuales
- variables globales asignadas por el compilador
- salvaguarda/restauración basadas en la profundidad de anidamiento
- direcccionamiento de registros

cache:

- datos escalares locales recientemente usados
- bloques de memoria
- variables globales usadas recientemente
- salvaguarda/restauración basadas en el algoritmo de reemplazo
- direccionamiento de memoria

la comparación tiene algunas características de las dos soluciones. el banco de registros basado en ventanas contiene todas las variables escalares locales, excepto en el poco frecuente de desbordamiento de ventana, de las N - 1 activaciones de procedimientos más recientes. la caché contiene una selección de las variables escalares usadas recientemente. el banco de registros debería ahorrar tiempo, ya que guarda todas las variables escalares locales. por otra parte, la caché puede hacer un uso más eficiente del espacio, ya que reacciona ante las situaciones dinámicamente. además, las caches generalmente tratan todas las referencias a memoria del mismo modo, incluyendo las instrucciones y otros tipos de datos. así, es posible un ahorro en estas otras áreas con la cache, y no con un banco de registros.

un banco de registros puede hacer un uso ineficiente del espacio, puesto que no todos los procedimientos necesitaran el espacio de ventana completo asignado a ellos. por otro

lado, la caché adolece de otro tipo de ineficiencia: los datos se leen en la cache por bloques. mientras que el banco de registros contiene sólo las variables en uso, la cache lee en un bloque de datos algo , o mucho, que no se usará.
la caché es capaz de manipular tanto variables globales como locales. por regla general, hay muchos datos escalares globales, pero , de ellos, solo unos pocos se usan mucho. una caché descubrirá dinámicamente esas variables, y las almacenara. si el banco de registros basado en ventanas se complementa con registros globales, también puede contener algunos datos escalares globales. sin embargo, para un compilador es difícil determinar que datos globales se usaran mucho.
con el banco de registros, las transferencias de datos entre registros y memoria queda determinada por la profundidad de anidamiento de los procedimientos. como esta profundidad normalmente fluctúa en un estrecho margen, el acceso a memoria es relativamente poco frecuente. casi todas las memorias caché son asociativas por conjuntos con un tamaño de conjunto pequeño. de este modo, existe el peligro de que otros datos o instrucciones sobrescriben las variables usadas con frecuencia.
si nos basamos en lo discutió hasta aquí, la elección entre un amplio banco de registros basado en ventanas y una caché no está clara. existe una característica, no obstante, en la cual la aproximación de los registros es claramente superior, y que nos indica que un sistema basado en caché será notablemente más lento. la distinción viene dada por la sobrecarga de direccionamiento experimentada por cada una de las dos aproximaciones. para referenciar un dato escalar local en un banco de registros basado en ventanas, se usan un número de registro virtual y un número de ventana. los dos pueden proporcionarse a un decodificador relativamente sencillo para seleccionar. los dos pueden proporcionarse a un decodificador relativamente sencillo para seleccionar uno de los registros físicos. para referenciar una posición de memoria en la caché, hay que generar una dirección de memoria completa. la complejidad de esta operación depende del modo de direccionamiento. en una cache asociativa por conjuntos , una parte de las direcion se usa para leer un número de palabras y etiquetas igual al tamaño del conjunto. otra parte del a dirección se compara con las etiquetas y se selecciona una de las palabras leídas. debe quedar claro que , incluso en el caso de que la caché sea tan rápida como el banco de registros, el tiempo de acceso será considerablemente más grande. por consiguiente, desde el punto de vista de las prestaciones, el banco de registros basado en ventanas es superior para datos escalares locales. se puede conseguir una mejora adicional en las prestaciones añadiendo una caché solo para instrucciones.

optimización de registros basada en el compilador

supongamos ahora que disponemos de una máquina risc que contiene únicamente un pequeño número de registros. en tal caso , el uso optimizado de registros es responsabilidad del compilador. un programa escrito en un lenguaje de alto nivel no tiene por supuesto , referencias explícitas a los registros. en su lugar, las cantidades son refredias simbólicamente en el programa. el objetivo del compilador es mantener en

registros, en lugar de en memoria, los operandos necesarios para tantos cálculos como sea posible y minimizar las operaciones de carga y almacenamiento.
por lo general se sigue el siguiente enfoque. cada cantidad del programa candidata para residir en un registro se asigna a un registro simbólico o virtual. el compilador entonces asigna el numero ilimitado de registros simbólicos a un número fijo de registros reales. los registros simbólicos cuya utilización no se solapen pueden compartir el mismo registro reales. los registros simbólicos cuya utilización no se solape pueden compartir el mismo registro real. si en una parte concreta del programa hay más cantidades de tratar que registros reales, algunas de las cantidades se asignan a posiciones de memoria. se usan las instrucciones de carga y almacenamiento para colocar temporalmente cantidades en registros en las operaciones del cálculo.
lo esencial de la labor de optimización es decidir qué cantidades tienen que ser asignadas a registros en cualquier punto determinado del programa. la técnica usada más comúnmente en los compiladores para risc se conoce como coloreado de grafos, una técnica que procede de la disciplina de topología.
el problema del coloreado de grafos es el siguiente. dado un grafo que consta de nodos y arcos, asignar colores a los nodos de manera que nodos adyacentes tengan colores diferentes, y hacerlo de tal forma que se minimice el número de colores distintos. este problema se adapta al problema de los compiladores de la siguiente forma. en primer lugar, el programa se analiza para construir un grafo de interferencias entre registros. los nodos del grafo son registros simbólicos. si dos registros simbólicos están vivos durante el mismo fragmento de programa, entonces unen por un arco para representar su interferencia. se intenta colorear el grafo con n colores, donde n es el número de registros. si este proceso no tiene éxito completo, los nodos que no se pueden colorear se colocan en memoria y se tienen que usar cargas y almacenamientos para crear espacio para las cantidades afectadas cuando estas se necesiten.
los investigadores variaron el número de registros de 16 a 128 y consideraron tanto la utilización de todos los registros para uso general, como la separación de registros para enteros y coma flotante. su estudio mostró que incluso con una optimizacion de registros sencilla, se saca poco provecho a la utilización de más de 64 registros. si se usan tecnicas de optimizacion de registros razonablemente sofisticadas, con más de 32 registros solo hay una mejora escasa de las prestaciones. por último, observaron que con un pequeño número de registros, por ejemplo 16, una máquina con una organización de registros compartidos funciona más rápido que una con una organización separada. se pueden extraer conclusiones similares de que expone un estudio centrado principalmente en la optimización del uso de un número pequeño de registros en un lugar de en la comparación del uso de grandes conjuntos de registros con esfuerzos en la optimización

arquitectura de repertorio reducido de instrucciones

porque cisc?

hemos mencionado la tendencia hacia repertorios de instrucciones más ricos, que incluyen un número mayor de instrucciones e instrucciones más complejas. esta tendencia ha sido motivada por dos razones principales: el deseo de simplificar los compiladores, y el deseo de mejorar las prestaciones. sirvió de base a estas razones el cambio de los programadores hacia los lenguajes de alto nivel(HLL), que se hizo que los arquitectos intentaran diseñar máquinas que proporcionaran mejor soporte para los HLL.
no es la intención de este capitulo decir que los diseñadores de cisc tomaron la dirección equivocada. verdaderamente, debido a que la tecnología continúa evolucionando, y a que existen arquitecturas a lo largo de todo un espectro en vez de en dos categorías puras, es poco probable que surja nunca una valoración del tipo blanco o negro. por tanto, los comentarios que siguen sólo pretenden señalar algunos de los peligros potenciales de la aproximación cisc, proporcionar cierta comprensión del a motivación de los partidarios de los risc.
la primera de las razones citadas, la simplificación de los compiladores parece obvia. la labor del escritor de compiladores es generar una secuencia de instrucciones máquina para cada sentencia de HLL. si existen instrucciones máquina que se parezcan a sentencias del HLL, la tarea se simplifica. este razonamiento ha sido puesto en duda por los investigadores de los risc. estos han encontrado que las instrucciones maquina complejas son con frecuencia difíciles de aprovechar, ya que el compilador debe descubrir aquellos casos que se ajustan perfectamente a la construcción. la tarea de optimizar el código generado para minimizar su tamaño reducir el número de instrucciones ejecutada s y mejorar la segmentación es mucho más difícil con un repertorio complejo de instrucciones como prueba de esto, los estudios citados antes en este capítulo indican que la mayoría de las instrucciones de un programa compilado son comparativamente las mas sencillas.

la otra razón importante mencionada es la esperanza de que un cisc produzca programas más pequeños y rápidos. examinemos los dos aspectos de esta afirmación:que los programas sean más pequeños y que se ejecuten más rápido.
los programas más pequeños tienen dos ventajas. en primer lugar, como el programa ocupa menos memoria, hay un ahorro de este recurso. como la memoria hoy día es tan barata la ventaja potencial ya no es primordial. tiene mayor importancia el hecho de que programas más pequeños mejoren las prestaciones, lo que ocurre por dos motivos. el primero es que el que haya menos instrucciones significa que hay que captar menos bytes de instrucciones. el segundo es que, en un entorno paginado los programas más pequeños ocupan menos páginas, reduciendo las faltas de paginas.
el problema de este razonamiento es que está lejos de ser cierto que un programa para un cisc sea más pequeño que el correspondiente programa para un risc. en muchos casos, el programa para el cisc, expresado en lenguaje maquina simbolico, puede ser

mas corto,tener menos instrucciones, pero el numero de bits de meoria que ocupa no tiene por que ser mas peuqeno.la tabla 12.6 muestra lso resultados de tres estudios que compararon el tamno de programas en c compilados en varias maquinas, incluyendo el risc 1, que tiene una arquitectura de repertorio reducido e insturcciones. observe que hay poco o ningún ahorro cuando se usa un cisc en lugar de un risc. es también interesante advertir que el vax, que tiene un repertorio de instrucciones mucho más complejo que el pdp-11, logra muy poco ahorro respecto a este último. los investigadores de ibm confirmaron estos resultados, constando que el bim 801,un risc, producía código cuyo tamaño era 0,9 veces del código de un ibm s/370. el estudio utilizó un conjunto de programas en pl/1.

Tabla 12.6. Tamaño del código, relativo al RISC I

	[PATT82a] 11 programas en C	[KATE83] 12 programas en C	[HEAT84] 5 programas en C
RISC I	1,0	1,0	1,0
VAX-11/780	0,8	0,67	
M68000	0,9		0,9
Z8002	1,2		1,12
PDP-11/70	0,9	0,71	

los compiladores en los cisc tienden a elegir las instrucciones más sencillas, de manera que la concisión de las instrucciones complejas raramente entra en juego. también, debido a que hay más instrucciones en un cisc, son precisos códigos de operación más largos, produciendo instrucciones más largas. por último, los risc, tienden acentuar las referencias a registros en lugar de a memoria y las primeras necesitan menos bits. un ejemplo de este último efecto se discute dentro de poco.

de este modo, las expectativas de que un cisc presente programas más pequeños, con las ventajas que implicaría, pueden no llevarse a cabo. el segundo factor que motiva repertorios de instrucciones cada vez más complejos era que la ejecución de instrucciones fuera mas rapida. parece tener sentido el que una operación compleja de un HLL se ejecute más rápido como una única instrucción máquina, que como una sucesión de instrucciones más primitivas. sin embargo, debido a la propensión a usar las instrucciones más sencillas, esto puede no ser así. la unidad de control completa debe hacerse más compleja, y/o la memoria de control del microprograma ha de hacerse más grande, para proveer un repertorio de instrucciones más rico. cualquiera de los dos factores aumenta el tiempo de ejecución de las instrucciones simples.

de hecho, algunos investigadores han encontrado que la aceleración en la ejecución de funciones complejas se debe, no tanto a la potencia de las instrucciones maquina complejas, como a su residencia en el rápido almacenamiento de control. en realidad, el almacenamiento de control actúa como una cache de instrucciones. de este modo, el arquitecto del hardware podría intentar determinar que subrutinas o funciones se usarán con mayor frecuencia, y asignarlas al almacenamiento de control, implementandolas en

microcódigo. la realidad no es muy alentadora. en los sistemas s/390, instrucciones tales como translate y extenden precicison flotatin point divide, dividir en coma flotante con precisión ampliada, residen en almacenamiento de alta velocidad, mientras que la secuencia involucrada en establecer llamadas a procedimientos o en iniciar un gestor de interrupción se encuentra en la memoria principal, que es más lenta.
por tanto no es nada claro que la tendencia hacia repertorios de instrucciones de complejidad creciente sea apropiada. ello ha llevado a varios grupos a seguir el camino opuesto.

características de las arquitecturas de repertorio reducido de instrucciones
aunque existen diferentes aproximaciones a la arquitectura de repertorio reducido de instrucciones, hay ciertas características comunes a todas ellas:

- una instrucción por ciclo
- operaciones registro a registro
- modos de direccionamiento sencillo
- formatos de instrucción sencillos.

ahora haremos una breve discusión de estas características. más adelante en este capítulo se examinarán ejemplos concretos.

la primera característica enumerada es que se ejecuta una instrucción máquina cada ciclo máquina. un ciclo máquina se define como el tiempo que se tarda en captar dos operandos desde dos registros, realizar una operación de la alu y almacenar el resultado en un registro. asi, las intrucciones maquina de un risc no deberían ser más complicadas que las microinstrucciones de las máquinas cisc, y deberían ejecutarse más o menos igual de rápido. con instrucciones sencillas y de un ciclo , hay poca o ninguna necesidad de microcódigo; las instrucciones máquina pueden estar cableadas. tales instrucciones deben ejecutarse más rápido que las intrucciones maquina comparables de otras maquinas, ya que no hay que acceder a la memoria de control de microprograma durante la ejecución de la instrucción.
una segunda característica es que la mayoría de las operación deben ser del tipo registro a registro, con la excepción de sencillas operaciones load y store, para acceder a memoria. esta forma simplifica el repertorio de instrucciones y por tanto, la unidad de control. por ejemplo, un repertorio de instrucciones risc puede incluir solo una o dos instrucción add, por ejemplo suma entre yt suma con acarreo; el vax tiene 25 instrucciones add diferentes. otra ventaja es que tal arquitectura fomenta la optimización del uso de registros, ya que los operando accedido frecuentemente, permanecen en el almacenamiento de alta velocidad.
este énfasis en las operaciones registro a registro es único de los diseños risc. otras máquinas contemporáneas tiene tales instrucciones, pero incluyen también operaciones memoria a memoria y operaciones mixtas registro/memoria. en los años setenta, antes de la aparición de los risc, se hicieron intentos de comparar estas aproximaciones. las

arquitecturas hipotéticas se evaluaron segun el tamano del programa y el tráfico de memoria, expresado en número de bits. resultados como éste llevaron a un investigador a sugerir que las arquitecturas futuras no deberían contener registros en absoluto. uno se pregunta qué habría pensado , en su momento, de la máquina risc comercializada por pyramid, con nada menos que 528 registros.

lo que se pasó por alto en estos estudios fue un reconocimiento de que hay un acceso frecuente a un número pequeño de datos escalares locales, y de que con un amplio banco de registros o un compilador optimizador, la mayoría de los operandos pueden permanecer en registros durante largos periodos de tiempo.

una tercera característica es el uso de modos de direccionamiento sencillos. casi todas las instrucciones risc usan el sencillo direccionamiento a registro. se pueden incluir varios modos adicionales, como el desplazamiento y el relativo al contador de programa. otros modos más complejos se pueden sintetizar por software a partir de los simples. nuevamente, esta característica de diseño simplifica el repertorio de instrucciones y la unidad de control

una última característica comunes el uso de formatos de instrucción sencillos. generalmente, solo se usa un formato o unos pocos . la longitud de las instrucciones es fija, y alineada en los límites de una palabra. Las posiciones de los campos, especialmente la del código de operación, son fijas. Este tipo de diseño tiene varias ventajas . Con campos fijos, en la decodificación del código de operación y el acceso a los operandos en registros pueden tener lugar simultáneamente. Los formatos sencillos simplifican la unidad de control. Se optimiza la captación de instrucciones ya que se captan unidades de una palabra de longitud. El alineamiento en el límite de una palabra significa también que una única instrucción no traspasa los límite de una página.

Estas características pueden evaluarse conjuntamente para determinar las ventajas potenciales de la aproximación risc. Las ventajas se pueden separar en dos categorías: las relacionadas con las prestaciones y las relacionadas con la implementación VLSI.

Respecto a las prestacione, se puede representar cierta cantidad de pruebas indiciarias . En primer lugar, se pueden desarrollar compiladores optimizadores más eficaces. Con instrucciones más primitivas, hay más ocasiones de sacar funciones fuera de bucles, reorganizar código buscando una mayor eficiencia, maximizar la utilización de registros, etc. Es posible incluso calcular partes de instrucciones complejas en tiempo de compilación. Por ejemplo la instrucción move character (MVC) del s/390, transfiere una cadena de caracteres de una posición a otra. Cada vez que se ejecuta, la transferencia dependerá de la longitud de la cadena, de sí las posiciones se solapan y en qué direcciones y de cuales son las caracteristicas de alineación. En la mayoría de los casos estos se conocerá en tiempo de compilación. Por tanto, el compilador podría producir una secuencia optimizada de instrucciones primitivas para esta función.

Un segundo punto, es que la mayoría de las instrucciones generadas por un compilador son , de todos modos, relativamente sencillas. Parece razonable que una unidad de control, construida expresamente para estas instrucciones y que usara poco o ningún microcódigo, podrá ejecutarlas más rapido que un cisc comparable.

Un tercer punto tiene que ver con el uso de segmentación de las instrucciones. Los investigadores de risc creen que la técnica de segmentación de las instrucciones se puede aplicar mucho más eficazmente a un repertorio reducido de instrucciones.

Un último punto , algo menos importante es que los programas risc deberían ser más sensibles a las interrupciones, ya que estas se comprueban entre operaciones bastante

elementales. Las arquitecturas con instrucciones complejas, o bien restringen las interrupciones a los límites de instrucción o bien tienen que definir puntos interrumpibles específicos e implementar mecanismos para reanudar la ejecución de una instrucción.
No está demostrado que aumenten las prestaciones con una arquitectura de repertorio reducido de instrucciones. Se han hecho varios estudios, pero no con máquinas de tecnología y potencia comparables. Ademas, la mayoria de los estudio no han intentado separar los efectos de un repertorio reducido de instrucciones de los de un banco de registros extenso. La prueba indiciaria, sin embargo es sugestiva.
La segunda hará con una ventaja potencial es más evidente y tiene que ver con la implementación VLSI. Cuando se usa VLSI, el diseño y la implementación del procesador son esencialmente distintos. Los procesadores tradicionales, tales como el ibm s/390 y el VAX, constan de una o más tarjetas de circuitos impresos, que contiene chips SSI y MSI estandarizados. Con la llegada de LSI y VLSI es posible meter un procesador complejo en un único chip. Con un procesador de un único chip hay dos motivaciones para seguir una estrategia risc. En primer lugar, está la cuestión de las prestaciones. Los retardos dentro del chip son mucho menores que los retador entre chips, de modo que tienen sentido dedicar el escaso espacio del chip a aquellas actividades que ocurren frecuentemente. Hemos visto que las instrucciones sencillas y el acceso a datos escalares locales, son, las actividades más frecuentes. Los chips risc de berkeley se diseñaron con esta consideración en mente. Mientras que un microprocesador de un único chip típico dedica aproximadamente la mitad de su área a la memoria de control de microcódigo , el risc 1 dedica solo el 6% de su área a la unidad de control
un segundo asunto relacionado con VLSI es el tiempo de diseño e implementación. Un procesador VLSI es difícil de desarrollar. En lugar de confiar en los componente SSI y MSI disponibles, el diseñador debe realizar el diseño, trazado y modelado del circuito en el nivel de dispositivos. Con una arquitectura de repertorio reducido de instrucciones, este procesdo es mucho mas facil.si ademas las prestaciones del chip risc son equivalentes a los microprocesadores cisc compara bles , las ventjaas de la aproximacion risc se hacen evidentes.

Características cisc frente a risc
tras el entusiasmo inicial por las máquinas risc, ha habido una creciente convicción de que los diseños risc pueden sacar provecho de la inclusión de algunas características cisc y de que los diseños cisc pueden sacar provecho de la inclusión de algunas características risc. El resultado es que los diseño risc más recientes especialmente el powerpc , no son ya risc puros y que los diseños cisc más recientes particularmente pentium 2 incorporan ciertas características risc.

Segmentación de cauce en risc
para aumentar las prestaciones se usa con frecuencia la segmentación de instrucciones. Reconsideremos este asunto en el contexto de la arquitectura risc. La mayoría de las instrucciones son del tipo registro a registro y un ciclo de instrucción tiene las dos siguientes fases:

- I:captación de instrucción.
- E:Ejecución.realiza una operación de la ALU con registros como entrada y salida.

Las operaciones de carga y almacenamiento requieren tres fases:

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

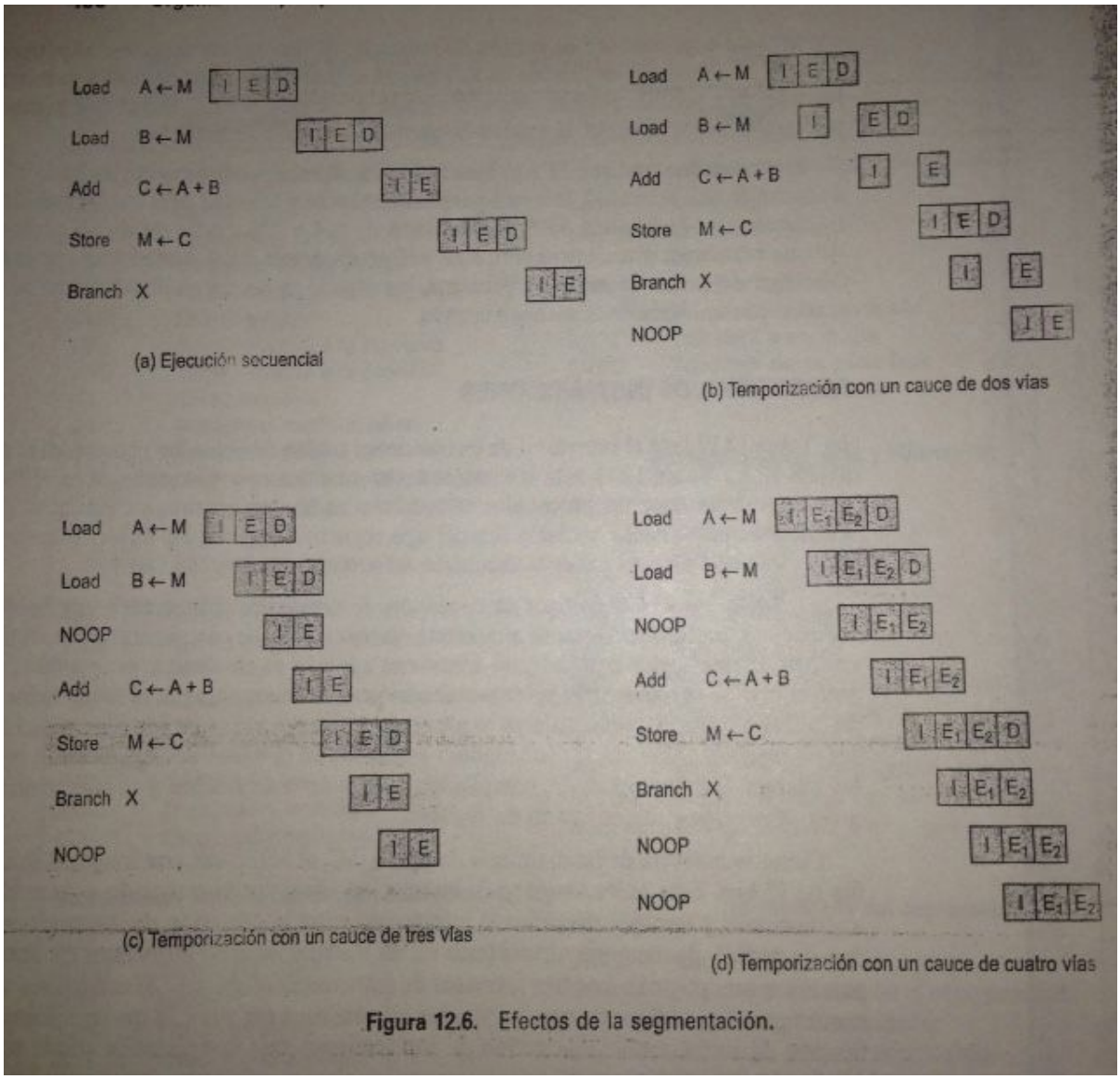


Figura 12.6. Efectos de la segmentación.

representa la temporalización de una secuencia de instrucciones que no usan segmentación. Sin duda se trata de un proceso dispendioso. Incluso una segmentación muy simple puede mejorar las prestaciones sustancialmente . La figura 12.6b muestra un esquema de segmentación de dos vías, en el cual las fases I y E de dos instrucciones diferentes se pueden ejecutar simultáneamente. Este esquema ofrece el doble de velocidad de ejecución que un esquema serie. Hay dos problemas que impiden que se consiga la máxima aceleración. El primero es que suponemos que se usa una memoria de un único puerto, y que solo es posible un acceso a memoria en cada fase. Esto requiere la inserción de un estado de espera en algunas instrucciones. El segundo problema consiste en que una instrucción de bifurcación interrumpa el flujo secuencial de ejecución. Para adaptarse a estos problemas con la mínima circuiteria, el compilador o ensamblador puede insertar una instrucción noop en el flujo de instrucciones.

La segmentación puede ser mejorada permitiendo dos accesos a memoria por fase. Esto produce la secuencia que se muestra en la figura 12.6c . ahora, hasta tres instrucciones pueden solaparse, y la mejora es como mucho un factor de tres. Nuevamente , las instrucciones de bifurcación hacen que la aceleración sea un poco menor que la máxima

posible. Hay que advertir también, que las dependencias de datos tienen su efecto. Si una instrucción necesita un operando que es modificado por la instrucción precedente , se necesita un retardo. También esto puede ser llevado a cabo por un NOOP.
La segmentación discutida hasta aquí funciona mejor si las tres fases son aproximadamente de la misma duración. Como la fase E usualmente implica una operación de la ALU, puede ser más larga. En ese caso, podemos dividirla en dos subfases:

- E1:lectura del banco de registros.
- E2:operación de la alu y escritura en el registro.

Dada la simplicidad y regularidad del repertorio de instrucciones, el diseño de la organización en tres o cuatro fases se realiza fácilmente. La figura 12.6d muestra el resultado con un cauce de cuatro vías. Hasta cuatro instrucciones pueden estar en curso en un momento dado y la aceleración potencial máxima es un factor de cuatro. Observe que de nuevo hay que usar NOOP para los retardos por bifurcaciones y dependencias de datos.

Optimización de la segmentación

dada la naturaleza sencilla y regular de las instrucciones risc, los esquemas de segmentación se pueden emplear eficientemente. Hay poca variación en la duración de la ejecución de instrucciones, y el cauce puede adaptarse para reflejar este hecho . Sin embargo, hemos visto que las dependencias de datos y bifurcaciones reducen la velocidad de ejecución global.
Para compensar estas dependencias se han desarrollado técnicas de reorganización del código. Consideremos primero las instrucciones de salto. El salto retardado, que es una forma de incrementar la eficacia de la segmentación, utiliza un salto que no tiene lugar hasta después de que se ejecute la siguiente instrucciones, de ahí que se llame retardado. Este curioso procedimiento se ilustra en la tabla 12.9 . en la columna salto normal, vemos un programa normal en lenguaje máquina con instrucciones simbólicas. Después de que se ejecute la instrucción de la dirección 102, la siguiente instrucción a ejecutar es la de la dirección 105. para regularizar el cauce, se inserta un NOOP después de este salto. No obstante, se pueden incrementar las prestaciones si se intercambian las instrucciones 101 y 102. la figura 12.7 muestra el resultado. La instrucción jump se capta antes de la instrucción add. Hay que observar, sin embargo, que la instrucción add se capta antes de que la ejecución de la instrucción jump tenga la oportunidad de modificar el contador de programa. Por consiguiente se conserva la semántica original del programa.
Este intercambio de instrucciones funcionara con éxito con saltos incondicionales, llamadas y retornos. Para bifurcaciones condicionales, el procedimiento no se puede aplicar a ciegas. Si la condición comprobada por la bifurcación condicional pueden optimizarse de esta forma.
Un tipo de táctica similar llamada carga retardada, se puede usar con las instrucciones load. En las instrucciones load, el procesador bloquea el registro destino de la carga. Después el procesador continúa la ejecución del flujo de instrucciones hasta que alcanza una instrucción que necesite ese registro, deteniéndose hasta que la carga finalice. Si el compilador puede reorganizar las instrucciones de manera que se pueda hacer un trabajo útil mientras la carga está en el cauce, la eficiencia aumentara.

Tabla 12.9. Salto normal y retardado

Dirección	Salto normal		Salto retardado		Salto retardado optimizado	
100	LOAD	X,A	LOAD	X,A	LOAD	X,A
101	ADD	1,A	ADD	1,A	JUMP	105
102	JUMP	105	JUMP	106	ADD	1,A
103	ADD	A,B	NOOP		ADD	A,B
104	SUB	C,B	ADD	A,B	SUB	C,B
105	STORE	A,Z	SUB	C,B	STORE	A,Z
106			STORE	A,Z		

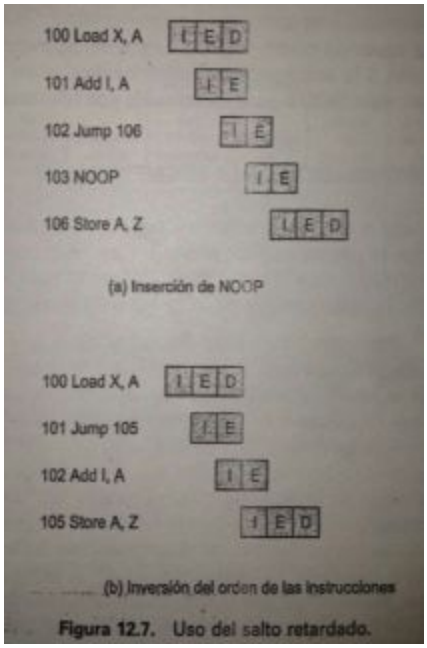


Figura 12.7. Uso del salto retardado.

como nota final, hay que señalar que el diseño del cauce de instrucciones no debe ser llevado a cabo independientemente de otras técnicas de optimización aplicadas al sistema. Por ejemplo, muestra que la planificación de instrucciones para el cauce y la asignación dinámica de registros tienen que considerarse conjuntamente para lograr la mayor eficiencia.

Lineamientos básicos en el diseño de un procesador RISC.
Análisis de prestaciones. Ejemplos.
Interrupciones: tratamiento general. Interrupciones por software y por hardware, vectores, descripción y tratamiento particular de cada una.
Relación entre las interrupciones y el manejo de operaciones de E/S.

UNIDAD 3

Concepto de E/S y su relación con la CPU, tipos de puertas.
Dispositivos externos
un dispositivo externo se conecta al computador mediante un enlace a un módulo de E/S. El enlace se utiliza para intercambiar señales de control, estado, y datos entre el módulo de E/S y el dispositivo externo.

Funciones de un módulo
principales funciones y requisitos de un módulo de E/S son las siguientes categorías:

- control y temporización
- comunicación con el procesador comunicación con los dispositivos
- almacenamiento temporal de datos
- detección de errores

en cualquier momento, el procesador puede comunicarse con uno o más dispositivos externos en cualquier orden, según las necesidades de E/S del programa. Los recursos internos, tales como la memoria principal y el bus del sistema, deben compartirse entre las distintas actividades, incluyendo la E/S de datos. Así, la función de E/S incluye ciertos requisitos de control y temporización para coordinar el tráfico entre los recursos internos y los dispositivos externos.

Ejemplo:
1.el procesador interroga al módulo de E/S para comprobar el estado del dispositivo conectado al mismo.
2.el módulo de E/S devie;ve e; estado de; dos[positivo.
3. si el dispositivo está operativo y preparado para transmitir, el procesador solicita la transferencia del dato mediante una orden al módulo de E/S.
4.el módulo de ES obtiene un dato, de 8 o 16 bits, del dispositivo externo.
5. los datos se transfieren desde el módulo de ES al procesador.

Si el sistema utiliza un bus, entonces cada una de las interacciones entre procesador y el módulo de ES implica uno o más arbitrajes del bus.
La comunicación con el procesador implica:
decodificación de órdenes: el módulo de E/S acepta órdenes del procesador. Estas órdenes se envían generalmente utilizando líneas del bus de control(read sector, write sector, scan record id)
datos: el procesador y el módulo de E/S intercambian datos a través del bus de datos.
Información de estado: puesto que los perifericos son lentos, es importante conocer el estado del módulo de E/S. Por ejemplo si se solicita a un módulo de ES que envíe datos al procesador, puede que no esté preparado por encontrarse atendiendo otra orden de ES previa. Esta situación puede indicarse con una señal de estado, como busy o ready.

Reconocimiento de dirección: tiene una dirección cada dispositivo. Un módulo de E/S puede reconocer una única dirección para cada uno de los periféricos que controla.

Los datos provenientes de la memoria se envían al modulo de ES en rafagas rapidas. Los datos se almacenan temporalmente en el módulo de ES y despues se envian al periferico a la velocidad de este. En el sentido contrario, los datos se almacenan para no mantener a la memoria ocupada en una operación de transferencia lenta. Así, el módulo de ES debe ser capaz de operar a las velocidad , tanto del dispositivo como de la memoria.
Un módulo de ES a menudo es responsable de la detección de errores, y de informar de estos errores al procesador. Una clase de errores son los defectos mecánicos y eléctricos en el funcionamiento del dispositivo, como papel atascado, o mal funcionamiento del disco, cambios accidentales en los bits al transmitirse desde el dispositivo al módulo de ES para detectar estos errores se utiliza algún tipo de código de detección de errores.

Estructura de un módulo de E/S

el módulo se conecta al resto del computador a través de un conjunto de lineas, lineas del bus del sistema.Los datos que se transfieren a y desde el módulo se almacenan temporalmente en uno o más registros de datos. Además puede haber uno o más registros de estado que proporcionan información del estado presente. Un registro de estado también puede funcionar como un registro de control para recibir información de control del procesador, la lógica que ha en el módulo interactúa con el procesador a través de una serie de líneas de control. Estas son las que utiliza el procesador para proporcionar las órdenes al módulo de E/s. Algunas de las líneas de control pueden ser utilizadas por el módulo de E/S, para señales de arbitraje y estado,. El módulo también debe ser capaz de reconocer y generar las direccione asociadas a los dispositivos que controla. Cada módulo de E/S tiene una dirección única o si controla más de un dispositivo externo, un conjunto único de direcciones. El módulo de E/S posee la lógica para la interfaz con cada uno de los dispositivos que controla.
Un módulo de E/S permite que el procesador vea a una amplia gama de dispositivos de una forma simplificada. Debe ocultar los detalles de temporización, formatos y electromecánica de los dispositivos externos, para que el procesador puede funcionar únicamente en términos de órdenes de lectura y escritura.
Un módulo de E/S que se encarga de los detalles del procesamiento, presentando al procesador una interfaz de alto nivel, se denomina canal de E/S o procesador de E/S. Un módulo que sea simple y requiera un control detallado, se denomina controlador de E/S o controlador de dispositivo.

E/S programada

son posibles tres técnicas para las operaciones de E/S. Con la E/S programada, los datos se intercambian entre el procesador y el módulo de E/S. El procesador ejecuta un programa que controla directamente la operación de E/S, incluyendo la comprobación del estado, el envío de una orden de lectura o escritura y la transferencia del dato. Cuando el procesador envía una orden al módulo de E/S, debe esperar hasta que la operación de E/S concluya. Si el procesador es más rápido que el módulo de E/S, el procesador desperdicia este tiempo. Con la E/S mediante interrupciones, el procesador proporciona la orden de E/S, continúa ejecutando otras instrucciones, y es interrumpido por el módulo de E/S cuando este ha terminado su trabajo. Tanto con E/S programada como con interrupciones, el procesador es responsable de extraer los datos de la memoria principal en una salida, y de almacenar los datos en la memoria principal en una entrada. La alternativa se conoce como acceso directo a memoria (DMA). En este

caso, el módulo de E/S y la memoria principal intercambian datos directamente, sin la intervención del procesador.

En resumen: cuando el procesador está ejecutando un programa y encuentra una instrucción relacionada con una E/S, ejecuta dicha instrucción mandando una orden al módulo de E/S apropiado. Con E/S programada, el módulo de E.S realiza la acción solicitada, y después activa los bits apropiados en el registro de estado, de E/S. El módulo de E/S no realiza ninguna otra acción para avisar al procesador. En concreto, no interrumpe al procesador, de esta forma, el procesador es responsable de comprobar periódicamente el estado del módulo de E/S hasta que encuentra que la operación ha terminado.

Ordenes de E/S
hay cuatro tipos de órdenes de E/S que puede recibir un módulo de E/S cuando es direccionado por el procesador:

1. Control: se utiliza para activar el periférico e indicarle que hacer.
2. Test: se utiliza para comprobar diversas condiciones de estado asociadas con el módulo de E/S y sus periféricos. Puede comprobar si el periférico está conectado y disponible.
3. Lectura: hace que el módulo de E/S capte un dato de un periférico y lo sitúe en un buffer interno. Después, el procesador puede obtener el dato solicitando que le módulo de E/S lo ponga en el bus de datos.
4. Escritura: hace que el módulo de E/S capte un dato del bus de datos y posteriormente lo transmita al periférico.

Instrucciones de E/S
las instrucciones se pueden hacer corresponder fácilmente con las órdenes de E/S y a menudo hay una simple relación de uno a uno. La forma de la instrucción depende de la manera de direccionar los dispositivos externos.
Normalmente habrá muchos dispositivos de E/S conectado al sistema a través de los módulos de E/S. Cada dispositivo tiene asociado un identificador único o dirección. Cuando el procesador envía una orden de E/S la orden contiene la dirección del dispositivo deseado. Así, cada módulo de E/S debe interpretar las líneas de dirección para determinar si la orden es para el.
Cuando el procesador, la memoria principal y las E/S comparten un bus común, son posibles dos modos de direccionamiento: asignado en memoria y aislado. Con las E/S asignadas en memoria, existe un único espacio de direcciones para las posiciones de memoria y los dispositivos de E/S, la CPU considera los registros de estado y de datos de los módulos de E/S como posiciones de memoria, y utiliza las mismas instrucciones maquina para acceder tanto a memoria como a los dispositivos de E/S.
Se necesita una sola línea de lectura y una sola línea de escritura en el bus. Alternativamente , el bus puede disponer de líneas de lectura y escritura en memoria junto con lineas para órdenes de entrada y salida. Las líneas de órdenes especifican si la dirección se refiere a una posición de memoria o a un dispositivo de E/S. El rango completo de direcciones está disponible para ambos.puesto que el espacio de direcciones de E/S está aislado del de memoria, este se conoce con el nombre de E/S aislada.

E/S mediante interrupciones
el problema con la E/S programada es que el procesador tiene que esperar un tiempo considerable a que el módulo de E/S en cuestión esté preparado para recibir o transmitir

los datos. El procesador, mientras espera, debe comprobar repetidamente el estado del módulo de E/S como consecuencia, se degrada el nivel de prestaciones de todo el sistema.
Una alternativa consiste en que el procesador, tras enviar una orden de E/S a un módulo, continúe realizando algún trabajo útil. Después , el módulo de E/S interrumpira al procesador para solicitar su servicio cuando esté preparado para intercambiar datos con el. El procesador ejecuta entonces la transferencia de datos como antes, y después continua con el procesamiento previo.
Desde el punto de vista del módulo de E/S para una entrada, el módulo de E/S recibe una orden READ del procesador, entonces el módulo de E/S procede a leer el dato desde el periférico asociado. Una vez que el dato está en el registro de datos del módulo, el módulo envía una interrupción al procesador a través de una línea control. Después el módulo espera hasta que el procesador solicite su dato. Cuando ha recibido la solicitud, el módulo sitúa su dato en el uso de datos y pasa a estar preparado para otra operación de E/S.
Desde el punto de vista del procesador, las acciones para una entrada son las que siguen: el procesador envía una orden READ de lectura. Entonces, pasa a realizar otro trabajos, el procesador puede estar ejecutando programas distintos al mismo tiempo. Al final de cada ciclo de instrucción el procesador comprueba las interrupciones. Cuando se pide la interrupción desde el módulo de E/S el procesador guarda el contexto, el contador de programa y los registros del procesador, del programa en curso, y procesa la interrupción. En este caso, el procesador lee la palabra de datos del módulo de E/.S y la almacena en memoria. Después recupera el contexto del programa que estaba ejecutando y continua su ejecución.
La E/S con interrupciones es más eficiente que la E/S programada, porque elimina las esperas innecesarias. No obstante, las E/S con interrupciones consumen gran cantidad del tiempo del procesador, puesto que cada palabra de datos que va desde la memoria al módulo de E/S o viceversa, debe pasar a través del procesador.

Procesamiento de interrupción
se produce la siguiente secuencia de eventos en el hardware:
1.el dispositivo envía una señal de interrupción al procesador
2.el procesador termina la ejecución de la instrucción en curso antes de responder a la interrupción
3. el procesador comprueba si hay interrupciones, determina que hay una y envía una señal de reconocimiento al dispositivo que originó la interrupción. La señal de reconocimiento hace que el dispositivo desactive su señal de interrupción.
4.el procesador necesita prepararse para transferir el control a la rutina de interrupción. Debe guardar la info necesaria para continuar el programa en curso en el punto en que se interrumpió. La info mínima que se precisa es el estado del procesador que se almacena en un registro llamado PSW program status word, y la posición de la siguiente instrucción a ejecutar contenida en el contador de programa. Estos registros se pueden introducir en la pila de control del sistema.
5.después el procesador carga el contador de programa con la posición de inicio del programa de gestión de la interrupción solicitada.
Una vez que el contador de programa se ha cargado. El procesador continúa con el ciclo de instrucción siguiente, captación de instrucción. El control se transfiere al programa de gestión de interrupción. La ejecución de este programa da lugar a las siguiente operaciones.

6.se deben guardar los contenidos de los registros del procesador,puesto que estos registros pueden ser utilizador por la rutina de interrupcion. La rutina de gestion de interrupción empezará almacenado en la pila los contenidos de todos los registros.
7.la rutina de gestion puede continuar procesando la interrupcion. Esto incluirá el examen de la info de estado a la operación de E/S, o cualquier otro evento que causara la interrupción. También puede implicar el envío al dispositivo de E/S de órdenes o señales de reconocimiento adicionales.
8.cuando el procesamiento de la interrupción ha terminado. Los valores de los registros almacenados se recuperan de la pila, y se vuelven a almacenar en los registros
9.el paso final es recuperar los valores del PSW y del contador de programa desde la pila. Como resultado, la siguiente instrucción que se ejecute pertenecera al programa previamente interrumpido.

Es importante almacenar toda la info del estado del programa interrumpido para que éste pueda reanudarse.

Cuestiones de diseño
hay 4 técnicas para identificar los dispositivos:

- múltiples líneas de interrupción
- consulta software(software polling)
- conexión en cadena (daysy chain),(consulta hardware, vectorizada)
- arbitraje de bus(vectorizada)

la aproximación más directa al problema consiste en proporcionar varias líneas de interrupción entre el procesador y los módulos de E/S. Sin embargo no resulta práctico, incluso si se utilizan varias líneas es probable que cada una se conecten varios módulos de E/S. Por eso se debe utilizar alguna de las otras tres técnicas en cada línea.
Consulta software. Cuando el procesador detecta una interrupción se produce una bifurcación a una rutina de servicio de interrupción que se encarga de consultar a cada módulo de E/S [para determinar el módulo que ha provocado la interrupción.como alternativa cada módulo de E/S podría disponer de un registro de estado direccionable. Entonces el procesador lee el estado del registro de cada módulo de E/S para identificar el módulo que solicitó la interrupción una vez identificado, se produce una bifurcación para que el procesador ejecute la rutina de servicio específica para ese dispositivo.
La desventaja de la consulta de software está en el tiempo que consume. Una técnica más eficiente consiste en utilizar la conexión en cadena, de los módulos de E/S que proporciona de hecho una consulta hardware. Todos los módulos de E/S comparten una línea común para solicitar interrupciones. La línea de reconocimiento de interrupción se conecta encadenando los módulos uno tras otro. Cuando el procesador recibe una interrupción, activa el reconocimiento de interrupción. Esta señal se propaga a través de la secuencia de los módulos de E/S hasta que alcanza un módulo que solicitó interrupción. Este módulo responde colocando una palabra en las líneas de datos. Esta palabra se denomina vector y es la dirección del módulo de E/.S. El procesador utiliza el vector como un puntero a la ruitnta de servicio de poisitivo apropiada.así se evita que tener que ejecutar una rutina de serivico general en primer lugar. Esta técnica se conoce con el nombre de interrupciones vectorizadas.
Hay otra técnica que hace uso de las interrupciones vectorizadas, se trata del arbitraje de bus. Con el arbitraje de bus un módulo de E.S debe, en primer lugar, disponer del control de bus antes de poder activar la línea de petición de interrupción. Así sólo un módulo puede activar la línea en un instante. Cuando el procesador detecta la

interrupción responde mediante la línea de reconocimiento de interrupción. Después el modulo que solicitó la interrupción sitúa su vector en las líneas de datos.

Las técnicas, sirven para identificar el módulo de E/S que solicita interrupción. Además, proporcionan una forma de asignar prioridades cuando más de un dispositivo está pidiendo que se sirva su interrupción. Con varias líneas de interrupción, el procesador selecciona la linea con mas prioridad. Con la consulta software, el orden en el que se consultan los módulos determina su prioridad. El orden de los módulos en la conexión en cadena determina su prioridad. El arbitraje del bus puede emplear un esquema de prioridad.

Acceso directo a memoria

inconvenientes de la E/S programada y con interrupciones:

la E/S con interrupciones, aunque más eficiente que la sencilla E/S programada, también requiere la intervención activa del procesador para transferir datos entre la memoria y el módulo de E/S y cualquier transferencia de datos debe seguir un camino a través del procesador. Por tanto ambas formas de E/S presentan dos inconvenientes inherentes:

1.la velcoidad de transferencia de E/S esta limitada por la velocidad a la cual el procesador puede comprobar y dar servicio a un dispositivo.

2.el procesador debe dedicarse a la gestión de las transferencias de E/S, se debe ejecutar cierto número de instrucciones por cada transferencia de E/S

existe un cierto compromiso entre estos dos inconvenientes. Considérese una transferencia de un bloque de datos utilizando E/S programada, el procesador se dedica a la tarea de la E/S y puede transferir datos a alta velocidad al precio de no hacer nada más. E/S con interrupciones libera en parte al procesador a expensas de reducir la velocidad de E/S. No obstante, ambos métodos tienen un impacto negativo tanto en la actividad del procesador como en la velocidad de transferencia de E/S.

Cuando hay que transferir grandes volúmenes de datos, se requiere una técnica más eficiente: acceso directo a memoria (DMA)

funcionamiento del DMA:

el DMA requiere un módulo adicional en el bus del sistema. El módulo de DMA, es capaz de imitar al procesador y es capaz de recibir el control del sistema cedido por el procesador. Necesita dicho control para transferir datos ,y desde, memoria a través del bus del sistema. Para hacerlo, el módulo de DMA debe utilizar el bus solo cuando el procesador no lo necesita, o debe forzar al procesador a que suspenda temporalmente su funcionamiento. Esta última técnica es la más común y se denomina robo de ciclo, puesto que en efecto el módulo de DMA roba un ciclo de bus.

Cuando el procesador desea leer o escribir un bloque de datos, envía una orden al módulo de DMA, incluyendo la siguiente información:

- si se solicita una lectura o una escritura, utilizando la línea de control de lectura o escritura entre el procesador y el módulo de DMA.
- La dirección del dispositivo de E/S en cuestión, indicada a través de las líneas de datos.
- La posición inicial de memoria a partir de donde se lee o se escribe indicada a través de las líneas de datos y almacenada por el módulo de dma en su registro de direcciones.

- El número de palabras a leer o escribir, también indicado a través de las líneas de datos y almacenado en el registro de cuenta de datos.

Después, el procesador continúa con otro trabajo. Ha delegado la operación de E/S al módulo de DMA, que se encargara de ella. El módulo de DMA transfiere el bloque completo de datos, palabra a palabra, desde o hacia la memoria, sin que tenga que pasar a través del procesador. Cuando la transferencia se ha terminado, el módulo de DMA envía una señal de interrupción al procesador. El procesador sólo interviene al comienzo y al final de la transferencia.
Para una transferencia e E/S de varias palabras el DMA es mucho más eficiente que la E/S mediante interrupciones o la programada. El número de ciclos de bus necesario puede reducirse sustancialmente si se integran las funciones de DMA y de E/S.
La lógica de DMA puede ser parte de un módulo de E/S o puede ser un módulo separado que controla a uno o más módulos de E/S. Este concepto se puede llevar algo más lejos conectando los módulos de E/S a un módulo de DMA mediante un bus de E/S. Esto reducirá uno el número de interfaces de E/S en el módulo de DMA, y permite una configuración fácilmente ampliable. En estos casos el bus del sistema que el módulo de DMA comparte con el procesador y la memoria, es usado por el módulo de DMA solo para intercambiar datos con la memoria. El intercambio de datos entre los módulos de DMA y E/S se produce fuera del bus del sistema.

Canales y procesadores de E/S
la evolución del funcionamiento de las E/S:
a medida que los computadores han evolucionado, la complejidad y sofisticacion de sus componentes se ha incrementado. En ningún lugar se hace más evidente que el funcionamiento de las E/S. Ya se ha considerado parte de esta evolución. Sus etapas se pueden resumir como sigue:
1. la cpu controla directamente al periférico. Esta situacion se observa en los dispositivos simples controlados por microprocesadores.
2.se anade un controlador o modulo.
3.se utiliza la misma configuracion del paso 2, pero ahora se emplean interrupciones. La cpu no necesita esperar a que se realice la operación de E/S, incrementandose la eficiencia.
4.el módulo de E/S tiene acceso directo a la memoria a través del DMA. Ahora se puede transferir un bloque de datos a, o desde la memoria sin implicar a la CPU, excepto al comienzo y al final de la transferencia.
5.el modo de E/S se mejora, haciendo que se comporte como un procesador en sí mismo, con un repertorio especializado de instrucciones orientado a las E/S. La cpu hace que el procesador de E.S ejecute un programa de E/S en memoria. El procesador de E/S capta y ejecuta sus instrucciones sin intervención de la cpu. Esto permite que la cpu pueda especificar una secuencia de actividades de E/S y ser interrumpida cuando se haya completado la secuencia entera.
6.el módulo de E/S tiene una memoria local propia y es, de hecho un computador en si mismo. Con esta arquitectura se puede controlar un conjunto grande de dispositivos de E/S con la misma intervención de la cpu. Un uso común de este tipo de arquitectura ha sido la comunicación con terminales interactivos. El procesador de E/S se ocupa de la mayoría de las tareas correspondientes al control de los terminales.

Características de los canales de E/S:
un canal de E/S puede ejecutar instrucciones de E/S lo que le confiere un control completo sobre las operaciones de E/S en un computador con tales dispositivos, la cpu

no ejecuta instrucciones de E/S. Dichas instrucciones se almacenan en memoria principal para ser ejecutadas por un procesador de uso específico contenido en el propio canal de E/S. La cpu inicia una transferencia de E/S, indicando al canal de E/S que debe ejecutar un programa de la memoria. El programa especifica el dispositivo o dispositivos, el área o áreas de memoria para almacenamiento, la prioridad y las acciones a realizar., el canal de E/s sigue estas instrucciones y controla la transferencia de datos.
Son comunes dos tipos de canales de E/S. Un canal selector controla varios dispositivos de velocidad elevada y, en un instante dado se dedica a transferir datos a uno de esos dispositivos. Es decir, el canal de E/S selecciona un dispositivo y efectúa la transferencia de datos. Cada dispositivo es manejado por un controlador, o módulo de E/S. Así el canal de E/S se utiliza en lugar de la cpu para controlar estos controladores de E/S. Un canal multiplexor puede manejar las E/S de varios dispositivos al mismo tiempo. Para dispositivos de velocidad reducida, un multiplexor de byte acepta o transmite caracteres tan rápido como es posible a varios dispositivos.

Concepto de puerta de Entrada y Salida paralelo.
Concepto de puerta de Entrada y Salida serie.
Tipos de transmisión serie.
Descripción del formato de transmisión serie asincrónica y sincrónica.
Descripción funcional de una puerta de E/S serie asincrónica, acceso a registros internos para control y determinación del estado de operación de la puerta.

Mapeado del subsistema E/S y la memoria.
Administración de las puertas por encuesta (polling) o por interrupción.
Tratamiento de la CPU de las operaciones de E/S, por interrupción o por software.
Transferencias de E/S por hardware, DMA, implementación.

Unidad 4
Subsistema Memoria.

Repaso de la organización jerárquica de la memoria, memoria principal y memoria secundaria.

la cuestión del tamaño es un tema siempre abierto. si se consigue hasta una cierta capacidad, probablemente se desarrollarán aplicaciones que la utilicen. la cuestión de la rapidez es en cierto sentido, fácil de responder. para conseguir las prestaciones óptimas, la memoria debe seguir al procesador. es decir, cuando el procesador ejecuta instrucciones, no e deseable que tenga que detenerse a la espera de instrucciones o de operandos. la última de las cuestiones anteriores también debe tenerse en cuenta. en la práctica, el coste de la memoria debe ser razonable con la relación a los otros componentes.

entre las tres características claves de coste, capacidad y tiempo de acceso. se emplean diversas tecnologías para realizar los sistemas de memoria. posibles tecnologías se cumplen las siguientes relaciones:

- a menos tiempo de acceso, mayor coste por bit.
- a mayor capacidad, menor coste por bit.
- a mayor capacidad, mayor tiempo de acceso.

el diseñador desearia utilizar tecnologías de memoria que proporcionan gran capacidad, tanto porque esta es necesaria, como porque el coste por bit es bajo. sin embargo, para satisfacer las prestaciones requeridas, el diseñador necesita utilizar memorias costosas, de capacidad relativamente baja y con tiempos de acceso reducidos.

la respuesta es no contar con un solo componente sino emplear una jerarquía de memoria.

cuando se desciende en la jerarquía ocurre:

- disminuye el coste por bit
- aumenta la capacidad
- aumenta el tiempo de acceso
- disminuye la frecuencia de accesos a la memoria por parte del procesador

memorias más pequeñas , más costosas y más rápidas, se complementan con otras mas grandes, mas economicas y más lentas. la clave del éxito de esta organización está en el ultimo item: la disminución de la frecuencia de acceso.

Memoria caché, concepto y descripción, análisis de prestaciones, métodos de implementación típicos, múltiples niveles. Ejemplos.

el objetivo de la memoria caché es lograr que la velocidad de la memoria sea lo más rápida posible, consiguiendo al mismo tiempo un tamano grande al precio de memorias semiconductoras menos costosas. hay una memoria principal relativamente grande y más lenta, junto con una memoria caché más pequeña y rápida. la caché contiene una copia de partes de la memoria principal. cuando el procesador intenta leer una palabra de memoria, se hace una comprobación para determinar si la palabra está en la caché. si es asi se entrega dicha palabra al procesador. si no, un bloque de memoria principal consistente en un cierto número de palabras se transfiere a la cache y después la palabra es entregada al procesador. debido al fenómeno de localidad de las referencias, cuando un bloque de datos es captado por la cache para satisfacer una referencia a memoria simple, es probable que se hagan referencias futuras a otras palabras del mismo bloque.

la estructura de un sistema de memoria caché principal. la memoria principal consta de hasta 2^n palabras direccionables, teniendo cada palabra una única dirección de n bits. esta memoria la consideramos dividida en un número de bloques de longitud fija, de k palabras por bloque. la caché consta de c líneas de k palabras cada una y eln umero de

líneas es considerablemente menos que el número de bloques de memoria principal. en todo momento un subconjunto de los bloques de memoria reside en líneas de la caché. ya que hay más bloques de líneas, una línea dada no puede dedicarse unívoca y permanentemente a un bloque. por consiguiente, cada línea incluye una etiqueta que identifica qué bloque particular está siendo almacenado, la etiqueta es usualmente una porción de la dirección de memoria principal, como describiremos más adelante en esta sección.
el procesador genera la dirección, RA , de una palabra a leer. si la palabra está en la caché es entregada al procesador. sino, el bloque que contiene dicha palabra se carga en la caché y la palabra es llevada después al procesador. la caché conecta con el procesador mediante líneas de datos, de control y de direcciones. las líneas de datos y de direcciones conectan también con buffer de datos y de direcciones que las comunican con un bus del sistema a través del cual se accede a la memoria principal. cuando ocurre un acierto de caché, los buffer de datos y de direcciones se inhabilitan, y la comunicación tiene lugar sólo entre procesador y caché, sin tráfico en el bus. cuando ocurre un fallo de caché la dirección deseada se carga en el bus del sistema, y el dato es llevado a través del buffer de datos tanto a la caché como al procesador.

diseño de caché
tamaño de caché: cuanto más grande el tamaño de caché, mayor es el número de puertas implicadas en el direccionar a la caché. el resultado es que caches grandes tienden a ser ligeramente más lentas que las pequeñas, incluso estando fabricadas con la misma tecnología de circuito integrado y con la misma ubicación en el chip o en la tarjeta de circuito impreso. el tamaño de caché está también limitado por las superficies disponibles de chip y de tarjeta.

función de correspondencia
se necesita un algoritmo que haga corresponder bloques de memoria principal a líneas de caché. además, se requiere algún medio de determinar qué bloque de memoria principal ocupa actualmente una línea dada de caché. la elección de la función de correspondencia determina cómo se organiza la cache. pueden utilizarse tres técnicas: directa, asociativa y asociativa por conjuntos.
en los tres casos usaremos los siguientes datos de ejemplo:

- la cache puede almacenar 64 kb
- los datos se transfieren entre memoria principal y la cache en bloques de 4 bytes. esto significa que la caché está organizada en 16k = 2^14 líneas de 4 bytes cada una
- la memoria principal es de 16 mbytes, con cada byte directamente direccionable mediante una dirección de 24bits (2^24 = 16m). así puede al objeto de realizar la correspondencia podemos considerar que la memoria principal consta de 4 m bloques de 4 bytes cada uno.

la técnica más simple, denominada correspondencia directa. consiste en hacer corresponder cada bloque de memoria principal a solo una línea posible de caché.
donde:
i=numero de linea de cache
j=número de bloque de memoria principal
m= número de líneas en la caché
se implementa utilizando la dirección. desde el dividida en tres campos. los w bits menos significativos identifican cada palabra dentro de un bloque de memoria principal; en la mayoría de las máquinas actuales el direccionamiento es a nivel de bytes. los s bytes restantes especifican uno de los 2 bloques de la memoria principal. la lógica de la caché

interpreta estos s bits como una etiqueta de s-r bits(parte más significativa) y un campo de línea de r bits. este último campo identifica una e las m=2^r líneas de la caché.

el uso de una parte de la dirección como número de línea proporciona una correspondencia o asignación única de cada bloque de memoria principal en la caché. cuando un bloque es realmente escrito en la línea que tiene asignada, es necesario etiquetarlo para distinguirlo del resto de los bloques que pueden introducirse en dicha línea. para ello se emplearán los s-r bits más significativos.
la técnica de correspondencia directa es simple y poco costosa de implementar. su principal desventaja es que hay una posición concret de caché para cada bloque dado. por ello, si un programa referencia repetidas veces a palabras de dos bloques diferentes asignados en la misma línea, dichos bloques se estarían intercambiando continuamente en la cache. y la tasa de aciertos sería baja.

la correspondencia asociativa supera la desventaja de la directa, permitiendo que cada bloque de memoria principal pueda cargarse en cualquier línea de la caché. la lógica de control de la caché interpreta una dirección de memoria simplemente como una etiqueta y un campo de palabra. el campo de etiqueta identifica unívocamente un bloque de memoria principal. para determinar si un bloque está en la caché su lógica de control debe examinar simultáneamente todas las etiquetas de líneas para buscar una coincidencia.
con la correspondencia asociativa hay flexibilidad para que cualquier bloque sea reemplazado cuando se va a escribir uno nuevo en la caché. los algoritmos de reemplazo o sustitución, discutidos más adelante en esta sección se diseñan para maximizar la tasa de aciertos.
la principal desventaja de la correspondencia asociativa es la compleja circuitería necesaria para examinar en paralelo las etiquetas de todas las líneas de caché.
la correspondencia asociativa por conjuntos es una solución de compromiso que recoge lo positivo de las correspondencias directa y asociativa, sin presentar sus desventajas. en este caso, la caché se divide en v conjuntos, cada uno de k líneas. las relaciones que se tienen son:

m=v x k
i=j módulo v

donde:
i= número de conjunto de caché
j=número de bloque de memoria principal
m=número de líneas de la caché

se denomina correspondencia asociativa por conjuntos de k vias. con la asignación asociativa por conjuntos, el bloque Bj puede asignarse en cualquiera de las k lineas del conjunto i. la lógica del control de control de la caché interpreta una dirección de memoria como tres campos: etiqueta, conjunto y palabra. los d bits de conjunto especifican uno de entre v=2^a conjuntos, los s bits de los campos de etiqueta y de conjunto especifican uno de los 2^s bloques de memoria principal. con la correspondencia asociativa por conjuntos de k vias. la etiqueta de una dirección de memoria es mucho más corta, se compara solo con las k etiquetas dentro de un mismo conjunto.

algoritmos de sustitución

cuando se introduce un nuevo bloque en la caché, debe sustituirse uno de los bloques existente. para el caso de correspondencia directa, solo hay una posible línea para cada bloque particular, y no hay elección posible. para las técnicas asociativas, se requieren algoritmos de sustitución. para conseguir alta velocidad, tales algoritmos deben implementarse en hardware. se han probado diversos algoritmos,; mencionaremos cuatro de los más comunes.LRU: se sustituye el bloque que se ha mantenido en la cache por más tiempo sin haber sido referenciado. esto es fácil de implementar para la asociativa por conjuntos de dos vías. cada línea incluye un bit USO. cuando una línea es referenciada, se pone a 1 su bit USO y a 0 el del a otra línea del mismo conjunto. cuando va a transferirse un bloque al conjunto se utiliza la línea cuyo bit USO es 0. ya que estamos suponiendo que son más probables de referencias las posiciones de memoria utilizadas más recientemente, el LRU debería dar la mejor tasa de aciertos. otra posibilidad es FIFO;se sustituye aquel bloque del conjunto que ha estado más tiempo en la caché, el algoritmo fifo puede implementarse fácilmente mediante una técnica cíclica tipo roun robin o buffer circular. otra posibilidad más es del LFU, se sustituye aquel bloque del conjunto que ha experimentado menos referencias, lfu podría implementarse asociando un contador a cada línea. una técnica no basada en el grado de utilización consiste simplemente en coger una línea al azar entre las posibles candidatas.

política de escritura

antes de ser reemplazado un bloque que está en una línea de cache, es necesario comprobar si ha sido alterado en cache pero no en memoria principal. si no lo ha sido, puede escribirse sobre la línea de caché. si ha sido modificado, esto significa que se ha realizado al menos una operación de escritura sobre una palabra de la línea correspondiente de la caché y la memoria principal debe actualizarse de acuerdo con ello. son posibles varias políticas de escritura, con distintos compromisos entre prestaciones y coste económico. hay dos problemas contra los que luchar. en primer lugar, más de un dispositivo puede tener acceso a la memoria principal. por ejemplo, un módulo de e/s puede escribir/leer directamente en/de memoria. si una palabra ha sido modificada solo en la caché, la correspondiente palabra de memoria no es válida. además, si el dispositivo de e/s ha alterado la memoria principal, entonces la palabra de cache no es válida. un problema más complejo ocurre cuando varios procesadores se conectan al mismo bus y cada uno de ellos tiene su propia caché local. en tal caso, si se modifica una palabra en una de las caches podría presumiblemente invalidar una palabra de otras caches.

la técnica más sencilla se denomina escritura inmediata. utilizando esta técnica, todas las operaciones de escritura se hacen tanto en caché como en memoria principal, asegurando que el contenido de la memoria principal siempre es válido. cualquiera otro módulo procesador-caché puede monitorizar el tráfico a memoria principal para mantener la coherencia en su propia caché. la principal desventaja de esta técnica es que genera un tráfico sustancial a memoria que puede originar un cuello de botella. una técnica alternativa, conocida como post-escritura minimiza las escrituras en memoria. con la post-escritura, las actualizaciones se hacen solo en la caché. cuando tiene lugar una actualización, se activa un bit actualizar asociado a la línea. después, cuando el bloque es sustituido, es post-escrito en memoria principal si y sólo si el bit actualizar esta activo. el problema de este esquema es que aveces porciones de memoria principal no son válidas y los accesos por parte de los módulo de e/s sólo podrían hacerse a través de la caché.esto complica la circuiteria y genera un cuello de botella potencia. la experiencia ha demostrado que el porcentaje de referencias a memoria para escritura es del orden del 15%.

en una estructura de bus en la que más de un dispositivo ,normalmente un procesador, tiene una caché y la memoria principal es compartida, se tropieza con un nuevo problema. si se modifican los datos de una caché, se invalida , no solamente la palabra correspondiente de memoria principal, sino también la misma palabra e otras caches, si coincide que otras cachés tengan la misma palabra. incluso si se utiliza una política de escritura inmediata, las otras cachés pueden contener dato son válidos. un sistema que evite este problema se dice que mantiene la coherencia de cache. entre las posibles aproximaciones a la coherencia de cache se incluyen:

- vigilancia del bus con escritura inmediata: cada controlador de caché monitoriza las líneas de direcciones para detectar operaciones de escritura en memoria por parte de otros maestros del bus. si otro maestro escribe en una posición de memoria compartida, que también reside en la memoria caché, el controlador de caché invalida el elemento de la caché. esta estrategia depende del uso de una política de escritura inmediata por parte de todos los controladores de caché.
- transparencia hardware: se utiliza hardware adicional para asegurar que todas las actualizaciones de memoria principal, vía cache, quedan reflejadas en todas las caches. así si un procesador modifica una palabra de su caché, esta actualización se escribe en memoria principal. además, de manera similar se actualizan todas las palabras coincidentes de otras caches.
- memoria excluida de cache: solo una porción de memoria principal se comparte por más de un procesador y esta se diseña como no transferible a cache. en un sistema de este tipo todos los accesos a memoria compartida son fallos de cache, porque la memoria compartida nunca se copia en la caché. la memoria excluida de caché puede ser identificada utilizando lógica de selección de chip o los bits más significativos de la dirección.

tamaño de línea

a medida que aumenta el tamaño de bloque, la tasa de aciertos primero aumenta debido al principio de localidad, el cual establece que es probable que los datos en la vecindad de una palabra referenciada sean referenciados en un futuro próximo. al aumentar el tamaño de bloque, más datos útiles son llevados a la cache. sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño de bloque se haga aún mayor y la probabilidad de utilizar la nueva información captada se haga menor que la de reutilizar la información que tiene que reemplazarse. dos efectos concretos entran en juego:

- bloques más grandes reducen el número de bloques que caben en la caché. dado que cada bloque captado se escribe sobre contenidos anteriores de la caché, un número reducido de bloques da lugar a que se sobreescribe sobre datos poco después de haber sido captados.
- a medida que un bloque se hace más grande, cada palabra adicional está más lejos de la requerida y por tanto es más improbable que sea necesaria a corto plazo.

la relación entre tamaño de bloque y tasa de aciertos es compleja, dependiendo de las características de localidad de cad programa particular, no habiéndose encontrado un valor óptimo definitivo. un tamaño entre 4 y 8 unidades direccionables(palabras o bytes) parece estar razonablemente próximo al óptimo.

número de caches

hay dos aspectos de diseño relacionados con este tema que son: el número de niveles de cache y el uso de caché unificada frente a cache partida. posible tener una caché en el mismo chip del procesador, cache on chip. reduce la actividad del bus externo del

procesador y los tiempos de ejecución e incrementa las prestaciones globales del sistema.
los diseños más actuales incluyen tanto cache on chip como externa. la estructura resultante se conoce como caché de dos niveles, siendo la caché interna el nivel 1 y la externa el nivel 2. si no hay caché 2 y el procesador hace una petición de acceso a una posición de memoria que no está en la cache 1 entonces el procesador debe acceder a la dram o la rom a través del bus. debido a la lentitud usual del bus y los tiempos de acceso de las memorias, se obtienen bajas prestaciones. si se utiliza una caché 2 sram entonces la información que falta puede recuperarse fácilmente. si la sram es suficientemente rápida para adecuarse a la velocidad el bus, los datos pueden accederse con cero estados de espera, el tipo de transferencia de bus más rápido.
la mejora potencial del uso de caché 2 depende de las tasas de aciertos en ambas cachés 1 y 2. el uso de un segundo nivel de caché mejora las prestacione.
recientemente se ha hecho normal separar la cache en dos: una dedicada a instrucciones y otra a datos
una caché unificada tiene ventajas

- tiene una tasa de aciertos mayor que una partida ya que nivela automáticamente la carga entre captaciones de instrucciones y de datos, si un patrón de ejecución implica muchas más captaciones de instrucciones que de datos la caché tenderá a llenarse con instrucciones y si el patrón de ejecución involucra relativamente más captaciones de datos, ocurrirá lo contrario
- solo se necesita diseñar e implementar una caché.

Conceptos de memoria virtual.

Organización y Arquitectura de Computadoras, William Stallings, Capítulo 4, 5ta ed.

- **Diseño y evaluación de arquitecturas de computadoras, M. Beltrán y A. Guzmán, Capítulo 2 Apartados 2.1 a 2.4, 1er ed.**

Unidad 5
Paralelismo y mejora de prestaciones.

Concepto de procesamiento paralelo. Paralelismo a nivel instrucción.
tipos de sistemas de paralelos

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

- una secuencia de instrucciones y una secuencia de datos (SISD, single instruction, single data):un único procesador interpreta una única secuencia de instrucciones, para operar con los datos almacenados en una única memoria. los computadores monoprocesador caen dentro de esta categoría.
- una secuencia de instrucciones y múltiples secuencias de datos (SIMD, single instruction multiple data): una única instrucción máquina controla paso a paso la ejecución simultánea y sincronizada de un cierto numero de elementos de proceso. cada elemento de proceso tiene una memoria asociada, de forma que cad instrucción es ejecutada por cada procesador, con un conjunto de datos diferentes. los procesadores vectoriales y los matriciales pertenece a esta categoría.
- múltiples secuencias de instrucciones y una secuencia de datos (MISD): se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. esta estructura nunca ha sido implementada.
- múltiples secuencias de instrucciones y múltiples secuencias de datos (MIMD):un conjunto de procesadores ejecuta simultáneamente secuencias de instrucciones diferentes con conjuntos de datos diferentes. los smp, los clusters, y los sistemas NUMA son ejemplos de esta categoría.

en la organización de MIMD, los procesadores son de uso general; cada uno es capaz de procesar todas las instrucciones necesarias para realizar las transformaciones apropiadas de los datos. los computadores MIMD se pueden subdividir, además según la forma que tienen los procesadores para comunicarse. si los procesadores comparten una memoria común, entonces cada procesador accede a los programas y datos almacenados en la memoria compartida, y los procesadores se comunican unos con otros atrás de esa memoria. la forma más común de este tipo de sistemas se conoce como multiprocesador simétrico (SMP).

en un SMP , varios procesadores comparten una única memoria mediante un bus compartido u otro tipo de mecanismo de interconexión.una característica distintiva de estos sistemas es que el tiempo de acceso a memoria principal es aproximadamente el mismo para cualquier procesador. un desarrollo más reciente es la organización con acceso no uniforme a memoria ((NUMA), como el propio nombre indica, el tiempo de acceso a zonas de memoria diferentes puede diferir en un computador NUMA.
un conjunto de computadora monoprocesador independientes, o de smp, pueden interconectarse para formar un cluster. la comunicación entre los computadores se realiza mediante conexiones fijas o mediante algún tipo de red.

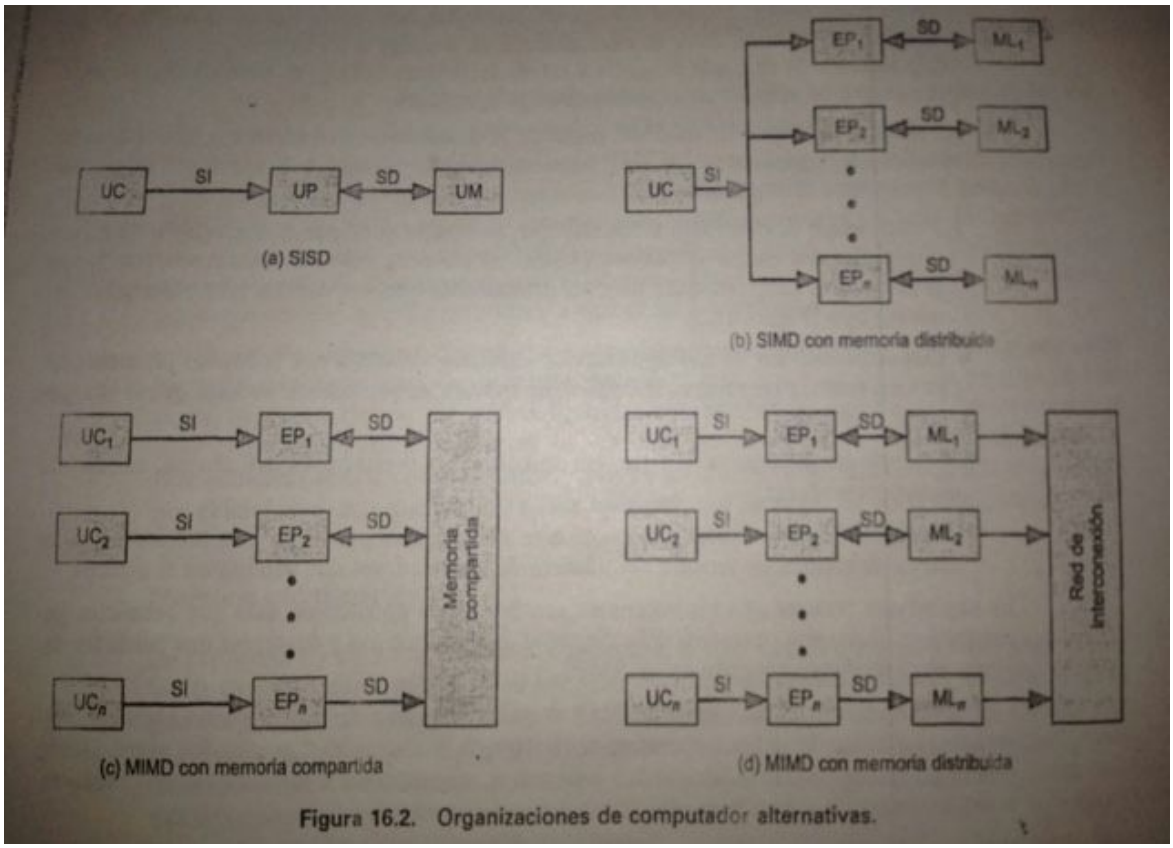


Figura 16.2. Organizaciones de computador alternativas.

los aspectos de diseño relacionados con los SMP, los clusters, y los NUMA, son complejos, implicando cuestiones relativas a la organización física, las estructuras de interconexión, el diseño de los sistemas operativos, y el software de las aplicaciones. nuestro interés se centra fundamentalmente en la organización, aunque se describirán brevemente aspectos del diseño de los sistemas operativos.

multiprocesador simétricos

a medida que aumenta la demanda de mayores prestaciones, y dado que el coste de los microprocesador continúa reduciéndose los fabricantes han introducido los sistemas SMP. el término SMP se refiere a la arquitectura hardware del computador, y también al comportamiento del sistema operativo que utiliza dicha arquitectura . un SMP puede definirse como un computador con las siguientes características:

1. hay dos o más procesador similares de capacidades comparables
2. estos procesadores comparten la memoria principal y la e/s, y están interconectados mediante un bus u otro tipo de sistema de interconexión, de forma que el tiempo de acceso a memoria es aproximadamente el mismo para todos los procesadores
3. todos los procesadores comparten los dispositivos de e/s, bien a través de los mismos canales, o bien mediante canales distintos que proporcionan caminos de acceso al mismo dispositivo
4. todos los procesadores pueden desempeñar las mismas funciones(de ahí el término simétrico
5. el sistema está controlado por un sistema operativo integrado, que proporciona la interacción entre los procesadore y sus programas en los niveles de trabajo, tarea, fichero y datos.

el punto 5 corresponde a una de las diferencias con los sistemas multiprocesadores debidamentes acoplados, tales como los clusters. en estos la unidad de interacción física es normalmente un mensaje o un fichero completo. en un SMP, la interacción se puede producir a través de elementos de datos individuales, y puede existir un elevado nivel de cooperación entre procesadores.

el sistema operativo de un SMP planifica la distribución de proceso o hilos entre todos los procesador. un SMP tiene las siguientes ventajas potenciales con respecto a la arquitectura monoprocesador:

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

estos son beneficios potenciales, y no garantizados. el sistema operativo debe disponer de herramientas y funciones que permitan explotar el paralelismo presente en un SMP.
una característica atractiva de un SMP es que la existencia de varios procesadores es transparente al usuario. el sistema operativo se encarga de la sincronización entre los procesadores y de la planificación de los hilo de los procesos, asignandolos a los distintos procesadores.

organización

- hay dos o más procesadores
- cada procesador es autónomo, incluyendo una unidad de control, una ALU, registros y posiblemente cache.
- cada procesador tiene acceso a una memoria principal compartida y a los dispositivos de e/s a través de alguna interconexión.
- los procesadores pueden comunicarse entre sí a través de la memoria compartida por mensajes e información de control. también pueden comunicarse entre sí.
- la memoria se organiza para que sea posible los accesos simultáneos a bloqueos de memoria separados
- en algunas configuraciones cada procesador puede tener su propia memoria principal privada y sus canales de e/s además de los recursos compartidos

la organización de un sistema multiprocesador puede clasificarse como:

1.bus de tiempo compartido:
el bus de tiempo compartido es el mecanismos más simple para construir un sistema multiprocesador. la estructura y las interfaces son básicamente las mismas que las de un sistema de un único procesador que utilice un bus para la interconexión. el bus

consta de líneas de control, dirección y datos. para facilitar las transferencias de DMA con los procesadores de e/s se proporcionan los elementos para el:

- direccionamiento:debe ser posible distinguir los módulos del bus para determinar la fuente y el destino de los datos.
- arbitraje: cualquier módulo de e/s puede funcionar temporalmente como maestro. se proporciona un mecanismo para arbitrar entre las peticiones que compiten por el control del bus, utilizando algún tipo de esquema de prioridad
- tiempo compartido: cuando un módulo está controlando el bus, los otros módulos no tiene acceso al mismo y deben suspender su operación hasta que dispongan del bus

estas características monoprocesador son utilizables directamente en una configuración de SMP. en esta caso, hay varias cpu además de varios procesadores de e/s que intentan tener acceso a uno o mas modulo de memoria a través del bus.

la organización del bus presenta diversas ventajas en comparación con otras propuestas:

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

la principal desventaja de la organización de bus son las prestaciones. todas las referencias a memoria pasan por el bus. en consecuencia la velocidad del sistema está limitada por el tiempo de ciclo. para mejorar las prestaciones es deseable equipar a cada procesador con una memoria caché. esta reduciría dramáticamente el número de accesos. típicamente los pc y las estaciones de trabajo de tipo SMP tienen dos niveles de cache: una caché L1 interna en el chip y una caché L2 externa o interna.
el uso de caches introduce algunas consideraciones de diseño nuevas . puesto que cada caché local contiene una imagen de una parte de la memoria si se altera una palabra en una caché. es concebible que eso podría invalidar una palabra en otra caché. para evitarlo se debe avisar a los otros procesador de uqe se ha producido una actualización de memoria. este problema se conoce como problema de coherencia de cache, que es resuelto típicamente por el hardware más que por el sistema operativo.

2.memoria multipuerto
la propuesta de memoria multipuerto permite el acceso directo e independiente a los módulos de memoria desde cada uno de los procesadores y los módulos de e/s. se necesita una cierta lógica asociada a la memoria para resolver los conflictos. el método que se utiliza a menudo para resolver conflictos consiste en asignar prioridades fijas a cada puerto de memoria. normalmente, la interfaz física y eléctrica en cada puerto es idéntica a la que aparece en un módulo de memoria de un solo puerto. así, se necesitan pocas o ninguna modificación en los procesadores o en los módulos de e/s para acomodar la memoria multipuerto.
la aproximación de la memoria multipuerto es más compleja que la aproximación de bus, precisandose añadir al sistema de memoria una buena cantidad de lógica. no obstante, se consiguen mejores prestaciones, puesto que cada procesador tiene un camino dedicado a cada módulo de memoria. otra ventaja del multipuerto es que

permite configurar partes de la memoria como privadas para uno o más procesadores y/o módulos de e/s. esta característica permite incrementar la seguridad frente a accesos no autorizados, y para el almacenamiento de rutinas de restablecimiento en zonas de memoria no susceptibles de ser modificadas por otros procesadores.
otra cuestión más:se debe utilizar una estrategia de escritura directa para controlar la caché, puesto que no hay forma de avisar a los otros procesadores de una actualización de memoria.

3.unidad de control central
la unidad de control central encauza las distintas secuencias de datos entre los distintos módulos independientes: procesador, memoria y e/s. el controlador puede almacenar temporalmente peticiones, y realizar las funciones de arbitraje y temporización. además puede transmitir mensajes de estado y control entre los procesadores y alertar sobre cambios en las caches.
puesto que toda la lógica de coordinación de la configuración de multiprocesador se concentra en la unidad central de control, las interfaces de e/s, memoria y procesador, no sufren cambios esenciales. esto proporciona la flexibilidad y la simplicidad de las interfaces en la aproximación de us. las desventajas clave de esta aproximación son que la unidad de control es bastante compleja, y que representa un cuello de botella potencial para las prestaciones.
la estructura de unidad de control central es bastante común en grandes computadoras(mainframes) de varios procesadores, tales como los miembros más grandes de la familia s/370 de ibm.hoy dia, esta alternativa es poco frecuente.

consideraciones de diseño de un sistema operativo de multiprocesador
un sistema operativo de SMP gestiona los procesadores y demás recursos del computador, para que el usuario perciba un solo sistema operativo controlando los recursos del sistema. e hecho, el computador debería parecer un sistema monoprocesador con multiprogramación. anto en un SMP como en un sistema monoprocesador, pueden estar activos varios trabajos o procesos al mismo tiempo, y es responsabilidad del sistema operativo planificar su ejecución y asignar los recursos.
un usuario puede desarrollar aplicaciones que utilizan varios procesado o varios hilos dentro de un proceso, sin tener en cuenta si se dispone de uno o varios procesadores.
así , un sistema operativo de multiprocesador, debe proporcionar toda la funcionalidad de un sistema operativo con multiprogramación, más las características adicionales que permitan utilizar varios procesadores. entre los puntos clave de diseño están:

- procesos concurrentes simultáneos: las rutinas del sistema operativo deben ser reentrantes, para permitir que varios procesadores pueden ejecutar simultáneamente el mismo código IS. con varios procesadores ejecutando la misma o distintas partes del sistema operativo, las tablas y las estructuras de gestión del sistema operativo deben manejarse aproximadamente, para evitar bloqueos u operaciones no válidas.
- planificación: la planificación puede realizarla cualquier procesador, por lo que deben evitarse los conflictos. el planificador debe asignar los procesos preparados a los procesadores disponibles.
- sincronización: puesto que hay varios procesos que pueden acceder a espacios de memoria y a recursos de e/s compartidos, debe proporcionarse una sincronización efectiva. la sincronización asegura la exclusión mutua y la ordenación de eventos.

- gestión de memoria: la gestión de memoria en un multiprocesador debe comprender todos los aspectos propios de los computadores monoprocesadores.además el sistema operativo debe explotar el paralelismo que proporciona el hardware(ejemplo las memorias multipuerto) para obtener mejores prestaciones. los mecanismos de paginación en procesadores distintos deben coordinarse para mantener las consistencia cuando varios procesadores comparten una pagina o un segmento y para decir sobre el reemplazo de páginas.
- fiabilidad y tolerancia ante los fallos: el sistema operativo deberia hacer posible una degradación gradual, cuando se produce un fallo en un procesador. el planificador y otros elementos del sistema operativo, deben reconocer la pérdida de un procesador y reestructurar las tablas de gestión en consecuencia.

clusters

constituyen la alternativa a los multiprocesadores simétricos (SMP) para disponer de prestaciones y disponibilidad elevadas y son particularmente atractivos en aplicaciones propias de un servidor. se puede definir un cluster como un grupo de computadores completos interconectados, que trabajan conjuntamente como un único recurso de computado, creándose la ilusión de que se trata de una sola máquina. el término computador completo hace referencia a un sistema que puede funcionar por sí solo, independientemente del cluster. usualmente en la literatura cada computador del cluster se denomina nodo.

se enumeran cuatro beneficios que pueden conseguirse con un cluster. también se contemplan como objetivos o requisitos de diseño:

1. escalabilidad absoluta: es posible configurar clusters grandes, que incluso superan las prestaciones de las computadoras independientes más potentes. un cluster puede tener decenas de máquinas, cada una de las cuales puede ser un multiprocesador.
2. escalabilidad incremental: un cluster se configura de forma que sea posible añadir nuevos sistemas al cluster en ampliaciones sucesivas. así un usuario puede comenzar con un sistema modesto y ampliarlo a medid que lo necesite, sin tener que sustituir el sistema de que dispone por uno nuevo que proporcione mayores prestaciones.
3. alta disponibilidad: puesto que cada nodo del cluster es un computador autónomo, el fallo de uno de los nodos no significa la pérdida del servicio. en muchos casos es el software el que proporciona automáticamente la tolerancia de fallos.
4. mejor relación precio/prestaciones: al utilizar elementos estandarizados, es posible configurar un cluster con mayor o igual potencia de cómputo que un computador independiente mayor a mucho menos costo.

configuraciones de clústeres

se considera si las computadoras comparten el acceso al mismo disco. interconexiones de un enlace de alta velocidad, que puede utilizarse para intercambiar mensajes que coordinan la actividad del cluster. el enlace puede ser una lan que se comparte con otros computadores no incluidos en el cluster, o puede tratarse de un medio de interconexión específico. en este último caso uno o varios computadores del cluster tendrán un enlace a una lan o una wan, de forma que sea posible la conexión

entre el cluster , actuando como servidor los clientes remotos.
otra alternativa es un cluster con disco compartido. en este caso generalmente también existe un enlace entre los nodos. además existe un subsistema de disco que se conecta directamente a los computadores del cluster. el subsistema de disco coun es un raid. el uso del raid o algún tipo de tecnología de disco redundantes es común en los cluster para que la elevada disponibilidad que se consigue con la presencia de varios computadores , no se vea comprometida por un disco compartido que pueda convertirse en un único punto de fallo.
existe un procedimiento conocido como espera pasiva, bastante antiguo y común que consiste simplemente en mantener toda la carga de trabajo en un computador mientras que otro permanece inactivo hasta tomar el relevo del primero, cuando se produce un fallo de este. para coordinar las máquinas, el computador activo o primario envía un mensaje de actividad al computador en espera. en el caso de que estos mensajes dejen de llegar, el computador en espera asume que el servidor ha fallado y se pone en marcha. esta alternativa aumenta la disponibilidad, pero no las prestaciones. es más, si los dos computadores sólo se intercambian el mensaje de actividad y si no tiene dicos comunes, entonces el computador en espa constituye un computador de reserva para ejecutar procesos, pero no tiene acceso a las bases de datos gestionadas por el primer computador.
la espera activa no suele aplicarse en un clúster. l término cluster hace referencia a varios computadores interconectados que se encuentran activos realizando algún tipo de procesamiento a la vez y que proporcionan una imagen de sistema único al exterior. el término secundario activo e utiliza a menudo para referirse a esta configuración. se pueden distinguir tres métodos para componer el cluster: con servidores separados, sin compartir nada y compartiendo memoria.

en el primero de los métodos, cada computador es un servidor independiente con su propio disco y no existen discos compartidos por los sistemas. esta organización proporciona, tanto disponibilidad como unas prestaciones elevadas. se precisa algún tipo de software de gestión o planificación para asignar a los servidores las peticiones que se van recibiendo de los clientes, de forma que se equilibre la carga de los mismos se consiga una utilización elevada. es deseable tener una cierta capacidad de fallos, lo que significa que , si un computador falla mientras está ejecutando una aplicación otro computador del cluster puede acceder a la aplicación y completarla. para que esto suceda, los datos se deben copiar constantemente en los distintos sitemas, de manera que cada procesador pueda acceder a los datos actuales de los demás. el coste que supone este intercambio de datos ocasiona una penalización en las prestaciones, pero es el precio por conseguir una disponibilidad elevada.
para reducir el coste que suponen las comunicaciones, la mayoría de los cluster están constituidos por servidores conectados a discos comunes. una variación a esta alternativa se designa como configuración sin compartir nada. en esta aproximación, los discos comunes se dividen en volúmenes diferentes y cada volumen pasa a pertenecer a un solo procesador. si el computador falla, el cluster se debe reconfigurar, de forma que los volúmenes que pertenecían al computador defectuoso pasen a los otros computadoras.
también es posible hacer que los computadores compartan el disco al mismo tiempo(aproximación de disco compartido), para que todos los computadores tengan acceso a todos los volúmenes de todos los discos. esta aproximación necesita utilizar algún tipo de procedimiento por el acceso exclusivo , para asegurar que, en un momento dado, solo un computador puede acceder a los datos.

gestión de fallos

la forma en que se actúa frente a un fallo en un cluster depende del tipo de configuración del mismo. en general, se pueden utilizar dos alternativas para enfrentarse a los fallos: clusters de alta disponibilidad y clusters tolerantes a fallos. un cluster de alta disponibilidad es le que ofrece una probabilidad elevada de que todos sus recursos estén en servicio. si se produce un fallo, tal como la caída del sistema o la perdida de un volumen de disco, se pierden las tareas en curso. si una de estas tareas se reinicia, puede ser un computador distinto el que le de servicio. no obstante, el sistema operativo el cluster no garantiza el estado de transacciones ejecutadas parcialmente. esto debería gestionarse en el nivel aplicación

un cluster tolerante a fallos garantiza que todos los recursos estén disponibles. esto se consigue utilizando discos compartidos redundantes, y mecanismo para salvar las transacciones no terminada y concluir las transacciones completadas.

la función de conmutar aplicaciones y datos en el cluster, desde un sistema defectuoso a otro alternativo, se denomina transferencia por fallo. una función adicional es la restauración de las aplicaciones y los datos por el sistema original, una vez superado el fallo, se denomina recuperación después de un fallo. la recuperación puede hacerse automáticamente, pero esto es deseable sólo si el fallo ha sido completamente reparado y es poco probable que vuelva a producirse. en caso contrario, la recuperación automática puede ocasionar transferencias continuas en un sentido y en otro, de programas y datos debido a la aparición de un fallo y se producirán problemas en cuanto a las prestaciones y a la recuperación.

equilibrado descarga

un cluster necesita una capacidad efectiva para equilibrar la carga entre los computadores disponibles. esta capacidad es necesaria para satisfacer el requisito de la escalabilidad incremental. cuando un computador se añade al cluster, las aplicaciones encargadas de la asignación de tareas deberían incluir automáticamente a dicho computador junto con los restantes, para distribuir la carga de forma equilibrada. los mecanismos de un nivel de software intermedio entre el sistema operativo y las aplicaciones (middleware)neceistan reconocer los servicios que pueden aparecer en los distintos miembros del cluster , y pueden migrar desde un miembro a otro.

clusters frente a SMP

tanto los cluster como los multiprocesadores simétricos constituyen configuración con varios procesadores pueden ejecutar aplicaciones con una alta demanda de recursos. amas soluciones disponibles comercialmente, aunque los smp lo están desde hace más tiempo.

la principal ventaja con un smp es que resulta más fácil de gestionar y configurar que un cluster. el smp esta mucho más cerca del modelo de computador de un solo procesador para el que están disponibles casi todas las aplicaciones. el principal cambio que se necesita para pasar de un computador monoprocesador a un smp se refiere al funcionamiento del planificador. otra ventaja de un smp es que necesita menos espacio físico y consume menos energía que un cluster comparable. una última e importante ventaja es que los smp son plataformas estables y bien establecidas.

con el tiempo no obstante las ventajas de los clusters serán las que probablemente harán que sean estos los que dominen el mercado de servidores de altas prestaciones. los cluster son superiores a los smp en términos de escalabilidad

absoluta e incremental y además también son superiores en términos de disponibilidad puesto que todos los componentes del sistema pueden hacerse altamente redundantes.

Procesadores Superescalares. Ejemplos. Clasificación de arquitecturas paralelo: taxonomía de Flynn. Ejemplos de aplicación.

el término superescalar hace referencia a una máquina diseñada para mejorar la velocidad de ejecución de las instrucciones escalares. el nombre contrasta el propósito de este esfuerzo frente a los procesadores vectoriales. en la mayoría de las aplicaciones la mayor parte de las operaciones se realizan con cantidades escalares. así pues la aproximación superescalar representa el siguiente paso en la evolución de los procesadore de uso general de altas prestaciones.
lo esencial del enfoque superescalar es su habilidad para ejecutar instrucciones de manera independiente en diferentes cauces. el concepto puede llevarse más lejos permitiendo que las instrucciones se ejecuten en un orden diferente al del programa. hay múltiples unidades funcionales cada una de las cuales está implementada como un cauce segmentado, que admiten la ejecución en paralelo de varias instrucciones.en este ejemplo dos operaciones enteras dos de coma flotante y nua de memoria(carga o almacenamiento) pueden estar ejecutándose al mismo tiempo.

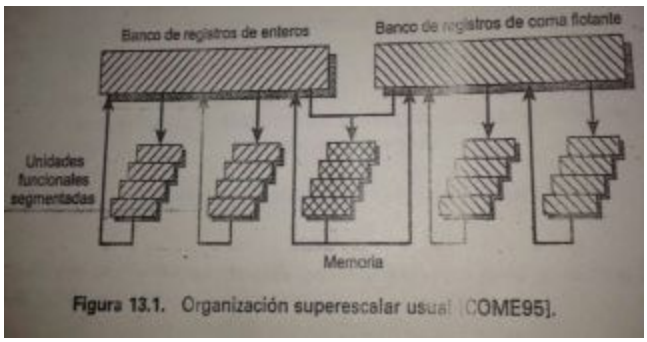


Figura 13.1. Organización superescalar usual [COME95].

superescalar frente a supersegmentado
una solución alternativa para alcanzar mayores prestaciones es la llamada supersegmentación, esta aprovecha el hecho de que muchas etapas del cauce realizan tareas que requieren menos de la mitad de un ciclo de reloj. de este modo, se dobla la velocidad de reloj interna, lo que permite la realización de dos tareas en un ciclo de reloj externo.
la figura 13.2 compara las dos aproximaciones. la parte superior del diagrama ilustra un cauce normal, usado como base de la comparación. el cauce base emite una instrucción por ciclo de reloj y puede ejecutar una etapa del cauce en cada ciclo. el cauce tiene cuatro etapas: captación de instrucción, decodificación e la operación, ejecución de la operación y escritura del resultado. la etapa de ejecución se ha destacado con una trama por motivos de claridad. observe que aunque se ejecuten varias instrucciones concurrentemente, solo hay una instrucción en la etapa de ejecución en un determinado instante.
la siguiente parte del diagrama muestra una implementación super segmentada que es capaz de ejecutar dos etapas del cauce por ciclo de reloj. una forma alternativa de enfocar consiste en que las funciones realizadas en cada etapa se pueden dividir en

dos partes no solapadas, y que cada una se ejecuta en medio ciclo de reloj. se dice que una implementación de un cauce supersegmentado que se comporta de esta forma es de grado 2. por último la parte inferior del diagrama muestra implementación superescalar capaz de ejecutar en paralelo dos instrucciones en cada etapa. naturalmente también son posibles implementaciones super segmentadas y superescalares de mayor grados.
las dos realizaciones, super segmentada y superescalar representadas en la figura 13.2 ejecutan el mismo número de instrucciones en el mismo tiempo cuando funcionan de forma ininterrumpida. el procesador supersegmentado se queda atrás con respecto al procesador superescalar al comienzo del programa y en cada destino de un salto

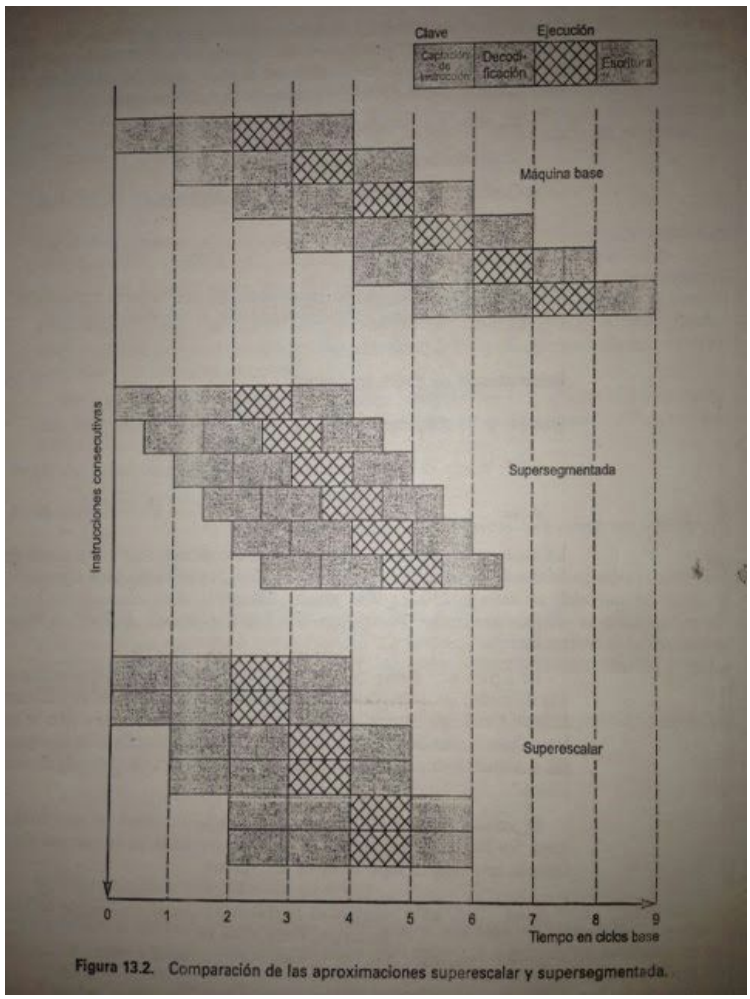


Figura 13.2. Comparación de las aproximaciones superescalar y supersegmentada.

limitaciones la aproximación superescalar depende de la habilidad para ejecutar múltiples instrucciones en paralelo. la expresión paralelismo a nivel de instrucciones se refiere al grado en el que en promedio las instrucciones de un programa se pueden ejecutar en paralelo. para maximizar el paralelismo a nivel de instrucciones, se puede usar una combinación de optimizaciones realizadas por el compilador y de técnicas hardware. antes de examinar las técnicas de diseño utilizadas en las máquinas superescalares para aumentar el paralelismo a nivel de instrucciones, debemos considerar las limitaciones fundamentales del paralelismo a las que el sistema tiene que enfrentarse:

- dependencia de datos verdadera.
- dependencia relativa al procedimiento
- conflictos en los recursos.
- dependencia de salida
- antidependencia.

cuestiones relacionadas con el diseño

paralelismo a nivel de instrucciones y paralelismo de la máquina
hace una importante distinción entre dos conceptos relacionados: el paralelismo a nivel de instrucciones y el paralelismo de la máquina. existe paralelismo a nivel de instrucciones cuando las instrucciones de una secuencia son independientes y por tanto pueden ejecutarse en paralelo solapandose.
como ejemplo del concepto de paralelismo a nivel de instrucciones consideremos los dos siguientes fragmentos de código

código 1:
load R1 <- R2
add R3 <- R3, "1"
add R4 <- R4, R2

código 2:
add R3 <- R3, "1"
add R4 <- R3, R2
store [R4] <- R0

las tres instrucciones el codigo 1 son independientes y en teoría las tres podrían ejecutarse en paralelo. por contraste las tres instrucciones del código 2 no pueden ejecutarse en paralelo, porque la segunda instrucción usa el resultado de la primera y la tercera instrucción usa el resultado de la segunda.
el paralelismo a nivel de instrucciones depende de la frecuencia de dependencias de datos verdaderas y dependencias relativas al procedimiento que haya en el código. estos factores dependen a su vez de la arquitectura del repertorio de instrucciones y de la aplicación. el paralelismo a nivel de instrucciones depende también de lo que llama espera de una operación: el tiempo que transcurre hasta que el resultado de una instrucción está disponible para ser usado como operando de una instrucción posterior. la espera determina cuánto retardo causara una dependencia de datos o relativa al procedimiento.

el paralelismo de la máquina es una medida de la capacidad del procesador para sacar partido al paralelismo a nivel de instrucciones. el paralelismo de la máquina depende del número de instrucciones que pueden captarse y ejecutarse al mismo tiempo, y de la velcoidad y sofisticacion del mecanismo que usa el procesador para localizar instrucciones independientes.
tanto el paralelismo a nivel de instrucciones como el de máquina son factores importantes para mejorar las prestaciones. un programa puede no tener el suficiente nivel de paralelismo a nivel de instrucciones como para sacar el máximo partido al de máquina. el empleo de una arquitectura con instrucciones de longitud fija, como en risc aumenta el paralelismo a nivel de instrucciones. por otra parte un escaso paralelismo de máquina limitará las prestaciones sin que importe la naturaleza del

programa

politicas de emisión de instrucciones

el procesador tiene que ser capaz de identificar el paralelismo a nivel de instrucciones, y organizar la captación, decodificación y ejecución de las instrucciones en paralelo. utiliza el término emisión de instrucciones para referirse al proceso de iniciar la ejecución de instrucciones en la unidades funcionales del procesador y el termino política de emisión de instrucciones para referirse al protocolo usado para emitir instrucciones.
esencialmente, el procesador intenta localizar instrucciones más allá del punto de ejecución en curso, que puedan introducirse en el cauce y ejecutarse. con respecto a esto, son importantes tres ordenaciones:

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

cuanto más sofisticado sea el procesador, menos limitado estará por la estrecha relación entre estas ordenaciones. para optimizar la utilización de los diversos elementos del cauce, el procesador tendrá que alterar uno o más de estos ordenes con respecto al orden que se encontraría en una ejecución secuencial estricta. la unica restriccion que tiene el procesador es que el resultado debe ser correcto. de este modo, el procesador debe acomodar las diversas dependencias y conflictos discutidos antes.
en términos generales, podemos agrupar las politicas de emisión de instrucciones de los procesador superescalares en las siguientes categorías:

- emision en orden y finalización en orden.
- emisión en orden y finalización desordenada.
- emisión desordenada y finalización desordenada.

emisión en orden y finalización en orden:
la política de emisión de instrucciones más sencilla es emitir instrucciones en el orden exacto en que lo haría una ejecución secuencial(emision en orden) y escribir los resultados en ese mismo orden(finalización en orden). ni siquiera los cauces escalares siguen una política tan ingenua. no obstante, es útil considerar esta política como base con la cual comparar otras aproximaciones más sofiticadas.
ejemplo:suponemos un cauce superescalar capaz de captar y decodificar dos instrucciones a la vez con tres unidades funcionales independientes(aritmética entera, y aritmética de coma flotante) y con dos copias de la etapa de escritura del cauce.
las instrucciones se captan de dos en dos y se pasan a la unidad de decodificación. como las instrucciones se captan por parejas, las dos siguientes instrucciones tienen que esperar hasta que la pareja de etapas de decodificación del cauce se encuentre vacía. para garantizar la finalización en orden, cuando hay una pugna por una unidad funcional o cuando una unidad funcional necesita más de un ciclo para generar un resultado, la emisión de instrucciones se detiene temporalmente.
en este ejemplo el tiempo transcurrido desde la decodificación e la primera instrucción hasta la escritura de los últimos resultados es de ocho ciclos.

emision en orden y finalización desordenada
se usa en los procesadores risc escalares para mejorar la velocidad de las

instrucciones que necesitan muchos ciclos.con la finalización desordenada , puede haber cualquier número de instrucciones en la etapa de ejecución en un momento dado, hasta alcanzar el máximo grado de paralelismo de la máquina ocupando todas las unidades funcionales. la emisión de instrucciones se para cuando hay una pugna por un recurso , una dependencia de datos o una dependencia relativa al procedimiento.
aparte de las limitaciones anteriores, surge una nueva dependencia a la cual nos referimos anteriormente como dependencia de salida. el siguiente fragmento de código ilustra la dependencia:
I1: R3 <- R3 op R5
I2: R4 <- R3 + 1
I3: R3 <- R5 + 1
I4: R7 <- R3 op R4

la instrucción I2 no puede ejecutarse antes que la instrucción I1 ya que necesita el resultado almacenado en el registro R3 por I1. de un modo parecido I4 debe esperar a I3 porque usa un resultado producido por esta. en la relación I3 y I1 no hay dependencia de datos , sin embargo si I3 se ejecuta hasta el final antes que I1 se captará un valor incorrecto del contenido de R3 para la ejecución de I4. por consiguiente I3 debe terminar después de I3 para producir el valor correcto de salida. para asegurar esto, la emisión de la tercera instrucción debe detenerse si su resultado puede ser sobrescrito más tarde por una instrucción anterior que tarda más en finalizar.
la finalización desordenada necesita una logica de emisión de instrucciones más compleja que la finalización en orden. además es más difícil ocuparse de las interrupciones y excepciones. cuando ocurre una interrupción, la ejecución de instrucciones se suspende en el punto actual, para reanudarse posteriormente. el procesador debe asegurar que la reanudación tiene en cuenta que en el momento de la interrupción alguna construcciones posteriores a la instrucción que provocó dicha interrupción puede haber finalizado ya.

emisión desordenada y finalización desordenada.
con la emisión en orden, el procesador solo decoficara instrucciones hasta el punto de dependencia o conflicto. no se decodifican más instrucciones hasta que el conflicto se resuelve. por consiguiente el procesador no puede buscar más allá del punto de conflicto, instrucciones que podrían ser independientes de las que hay en el cauce y que podrían introducirse provechosamente en este.
para permitir la emisión desordenada, es necesario desacoplar las etapas del cauce de decodificación y ejecución . esto se hace mediante un buffer llamado ventana de instrucciones. con esta organizacion, cuando un procesdaor trmina e ecodficiar una instruccion, coloca esta en la ventana de instrucciones. mientras el buffer no se llene, el procesador puede continuar captando y decodificando nuevas instrucciones. cuando una unidad funcional de la etapa de ejecución quedó disponible , se puede emitir una instrucción desde la ventana de instrucciones a la etapa de ejecución., cualquier instrucción puede emitirse siempre que necesite la unidad funcional particular que está disponible y ningún conflicto ni dependencia la bloquea.
el resultado de esta organización es que el procesador tiene capacidad de anticipación, lo que le permite identificar instrucciones independientes que pueden introducirse en la etapa de ejecución. las instrucciones se emiten desde la ventana de instrucciones, sin que se tenga muy en cuent a su orden original en el program. como

antes la unica restriccion es que el programa funcione correctamente.,

renombramientos de registros
hemos visto que permitir la emisión desordenada de instrucciones y/o la finalización desordenada puede dar origen a dependencias de salida y antidependencias. estas dependencias son distintas de las dependencias de datos verdaderas y de los conflictos en los recursos, que reflejan el flujo de datos a través de un programa y la secuencia de la ejecución. las dependencias de salida y las antidependencias por otra parte surgen porque los valores de los registros no pueden reflejar la ya la secuencia de valores dictada por el flujo del programa.
cuando las instrucciones se emiten y se completan secuencialmente, es posible especificar el contenido de cada registro en cada punto de la ejecución. cuando se usan técnica de desordenación, los valores de los registros no pueden conocerse completamente en cada instante temporal, considerando solo la secuencia de instrucciones dictada por el programa. en realidad los valores entran en conflicto por el uso de los registros, y el procesador debe resolver tale conflictos deteniendo ocasionalmente alguna etapa del cauce.
las antidependencias y las dependencias de salida son dos ejemplos de conflictos de almacenamiento. varias instrucciones compiten por el uso de los mismos registros, generando restricciones en el cauce que reducen las prestaciones. el problema se agudiza cuando se utilizan tecnicas de optimizacion de registros, ya que estas técnicas del compilador intentan maximizar el uso de registros, maximizando por tanto el número de conflictos de almacenamiento.

hay un método para hacer frente a este tipo de conflictos de almacenamiento que se basa en una solución tradicional para los conflictos en los recursos: la duplicación de recursos. en este contexto, la técnica se conoce como renombramiento de registros. básicamente el hardware del procesador asigna dinámicamente los registros, que están asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. cuando se crea un nuevo valor de registro(es decir cuando se ejecuta una instrucción que tiene un registro como operando destino) se asigna un nuevo registro par ese valor. las instrucciones posteriores que accedan a ese valor como operando fuente en ese registro, tienen que sufrir un proceso de renombramiento: las referencias a registros de esas instrucciones han de revisarse para referenciar el registro que contiene el valor que se necesita. de este modo, las referencias a un mismo registro original en diferentes instrucciones, pueden referirse a distintos registos reales, suponiendo diferentes valores.

implementación superescalar
elementos a tener en cuenta en el hardware:

- estrategias de captación de instrucciones que captan simultáneamente

múltiples instrucciones, a menudo prediciendo los resultado de las instrucciones de bifurcación condicional y captando más allá de ellas. estas funciones requieren la utilización de múltiples etapas de captación y decodificación y lógica de predicción de saltos.

- lógica para determinar dependencias verdaderas entre valores de registros y mecanismos para comunicar esos valores a donde sea necesario durante la ejecución.
- mecanismos para iniciar o emitir múltiples instrucciones en paralelo.
- recursos para la ejecución en paralelo de múltiples instrucciones, que incluyan múltiples unidades funcionales segmentadas y jerarquías de memoria capaces de atender múltiples referencias a memoria.
- mecanismo para entregar el estado del procesador en un orden correcto.

Arquitecturas Multiprocesador. Memoria compartida o distribuida. Análisis de prestaciones.

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