Módulo de E/S

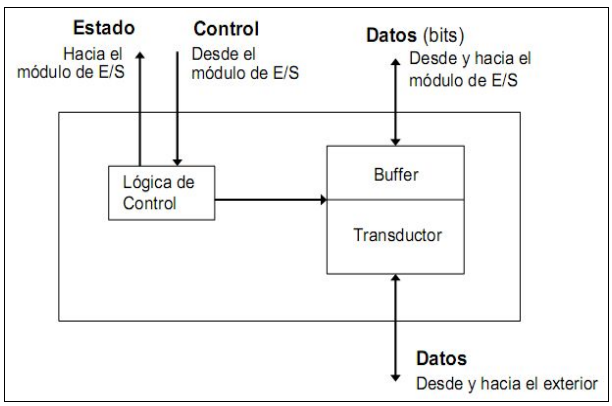


Ilustración 1: Esquema de un módulo de E/S

Un módulo de E/S que se encarga de la mayoría de los detalles del procesamiento, presentando al procesador una interfaz de alto nivel, se denomina generalmente canal de E/S o procesador de E/S. Un módulo que sea bastante simple y requiera un control detallado, normalmente se denomina controlador de E/S o Controlador de dispositivo. Los controladores de E/S aparecen usualmente en microcomputadores, mientras que los canales d e E/S se utilizan en grande computadores centrales (mainframes).

- Realiza la interfaz entre el procesador y la memoria (bus) y los periféricos.
- Pueden manejar uno o más periféricos.

Dispositivos Externos

- De interacción con humanos (Ej: pantalla, impresora, teclado)
- De interacción con máquinas (Ej: Control y supervisión (robótica))
- De comunicación con dispositivos remotos (Ej: Módem,Tarjeta de la interfaz de red (NIC))

Elementos de un dispositivo externo

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

Estructura o funciones de un Módulo de E/S

Las principales funciones y requisitos de un módulo de E/S se encuentran dentro de las siguientes categorías:

- **Control y temporización de uno o más dispositivos externos.**
- **Comunicación con el procesador (registros) y memoria.**
- **Comunicación con los dispositivos (periféricos).**
- **Almacenamiento temporal (buffering) de datos.**
- **Detección de errores.**

Control y temporización de uno o más dispositivos : Pasos de operación E/S

Ej: transferencia de datos de dispositivo a CPU

- La CPU comprueba el estado del dispositivo preguntando al módulo de E/S.
- El módulo devuelve el estado del dispositivo.
- Si el dispositivo está preparado, la CPU solicita la transferencia de datos al módulo.
- El módulo obtiene el dato del dispositivo externo.
- El módulo transfiere los datos a la CPU.

Comunicación con el procesador

- *Decodificación de comando* : el módulo de E/S acepta órdenes del procesador. Estas órdenes se envían generalmente utilizando líneas del bus de control.
- *Intercambio de datos* (entre el procesador y el módulo E/S a través del bus de datos).

- *Reporte de estado del dispositivo* (BUSY o READY) : puesto que los periféricos son lentos, es importante conocer el estado del módulo de E/S.
- *Reconocimiento de dirección del dispositivo que controla* : cada dispositivo de E/S tiene una dirección en memoria única.

Comunicación con el periférico

- Implica intercambiar órdenes, datos e información de estado.

Almacenamiento temporal (buffering) de datos.

Los datos provenientes de la memoria se envían al módulo de E/S y después se envía al periférico a la velocidad de éste. En el sentido contrario los datos se almacenan para no mantener a la memoria ocupada en una operación de transferencia lenta.

Detección de errores.

Un módulo de E/S es responsable de la detección de errores y de informar de estos errores al procesador (como defectos mecánicos, eléctricos o cambios accidentales en lo bits al transmitirse desde el dispositivo al módulo de E/S).

Técnicas de operaciones de E/S

- **Programada**

Cuando el procesador está ejecutando un programa y encuentra una instrucción relacionada con la 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.

  - Intercambio de datos entre la CPU y el módulo
  - La CPU tiene control directo sobre la operación de E/S
    - Comprobación del estado del dispositivo
    - Envío de comandos de lectura/escritura
    - Transferencia de datos
  - La CPU espera que el módulo E/S termine la operación
  - Por lo tanto la CPU permanece ociosa durante un período de tiempo (no deseable)

- **Mediante interrupciones (*ver aparte*)**

  - La CPU no tiene que esperar la finalización de la tarea de E/S, puede seguir procesando.
  - No se repite la comprobación de los estados de los módulos.
  - El módulo envía un pedido de interrupción a la CPU cuando está listo nuevamente.

- **Acceso directo a memoria (DMA) (*ver aparte*)**

INTERRUPCCIONES

Una interrupción es una alteración en el proceso de ejecución de un programa por efecto de un evento aleatorio que requiere atención de la CPU. Están pensadas para administrar operaciones de E/S sobre periféricos.

El uso de interrupciones implica una mayor eficiencia de la CPU. La mayor desventaja surge de la naturaleza aleatoria de una interrupción, ya que debemos modificar el estado de un programa en el momento en que ésta ocurre.

Es difícil verificar que un programa funcione correctamente con el uso de la administración de interrupciones. El modo de la interrupción se llama servicio de interrupción.

Tipos de interrupciones

- ***Por resultado de una ejecución de una instrucción (de programa).***
  - Ej: desbordamiento aritmético (“overflow”), división por cero
- ***Por un temporizador interno del procesador.***
  - Permite al S.O. realizar ciertas funciones de manera regular.
- ***Por una operación de E/S.***

- Ej: para indicar la finalización normal de una operación.
- ***Por un fallo de hardware.***
- Ej: error de paridad en la memoria, pérdida de energía.

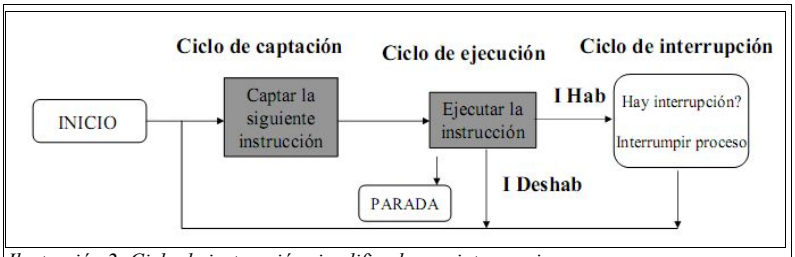


Ilustración 2: Ciclo de instrucción simplificado con interrupciones

Ciclo de Instrucción
•Captación.
•Ejecución.
•Gestión de interrupciones.

Ciclo de Interrupción

- Se comprueba si se ha generado alguna interrupción (que es indicada por alguna señal de interrupción)
- Si no hay señales se capta la siguiente instrucción.
- Si hay alguna interrupción pendiente:
  - Se suspende la ejecución del programa en curso.
  - Guarda su contexto (próxima instrucción a ejecutar y el estado del procesador)
  - Carga el PC con la dirección de comienzo de una rutina de gestión de interrupción. Se inhiben otras interrupciones.
  - Finaliza la rutina de gestión, el procesador retoma la ejecución del programa del usuario en el punto de interrupción.

Tratamiento de interrupciones Múltiples:

Existen dos formas diferentes de tratar las interrupciones múltiples:

Desactivar las interrupciones mientras se está procesando una interrupción:

- El procesador puede y debe ignorar la señal de petición de interrupción si se produce una interrupción en ese momento.
- Si se hubiera generado una interrupción se mantiene pendiente y se examinará luego una vez que se hayan habilitado nuevamente.
- Ocurre una interrupción, se inhabilitan, se gestiona la misma y luego se habilitan otra vez.
- Por lo tanto las interrupciones se manejan en un orden secuencial estricto.

Se definen prioridades para el tratamiento de interrupciones múltiple:

- Una interrupción de prioridad más alta puede interrumpir a un gestor de interrupción de prioridad menor.
- Cuando se ha gestionado la interrupción de prioridad más alta, el procesador vuelve a las interrupciones previas (de menor prioridad).
- Terminadas todas las rutinas de gestión de interrupciones se retoma el programa del usuario.

Jerarquía de interrupciones

Si hay múltiples fuentes se establece cuales son más importantes.

- ***No enmascarables***: se atienden siempre. Indican eventos peligrosos o de alta prioridad.
- ***Enmascarables***: pueden ser ignoradas. Con instrucciones inhibimos su ocurrencia.

Cuestiones de diseño

- ¿Cómo saber qué dispositivo ha provocado la interrupción?
- Con múltiples interrupciones ¿Cómo elegir la interrupción que se debe atender? ¿Establecemos prioridades?

Identificación del módulo que interrumpe

- **Diferentes líneas para cada módulo**
  - PC
  - Limita el número de dispositivos
- **Consulta software** (Poll o encuesta)
  - Ocurrido un pedido de interrupción la CPU consulta a cada módulo para determinar quien fue el demandante.
  - Resulta lento.
- **Conexión en cadena** (daisy chain) “hard poll”

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

Cuestiones de diseño en interrupciones múltiples

- Todas las líneas de interrupción tienen un orden de prioridad.
- Las líneas con más prioridad pueden interrumpir a las líneas con menor prioridad.
- Si existe un maestro del bus, sólo el maestro puede interrumpir.

Inconvenientes de la E/S programada y con interrupciones

- La velocidad de transferencia de E/S está limitada por la velocidad a la cual el procesador puede comprobar y dar servicio a un dispositivo.
- El procesador debe dedicarse a la gestión de la transferencias de E/S; se debe ejecutar cierto número de instrucciones por cada transferencia de E/S.

DMA

El controlador de DMA (*Direct Memory Access*) es un dispositivo capaz de controlar una transferencia de datos entre un periférico y memoria sin intervención de la CPU.

Muchos sistemas hardware utilizan DMA, incluyendo controladores de unidades de disco, tarjetas gráficas y tarjetas de sonido. DMA es una característica esencial en todos los ordenadores modernos, ya que permite a dispositivos de diferentes velocidades comunicarse sin someter a la CPU a una carga masiva de interrupciones.

Una transferencia DMA consiste principalmente en copiar un bloque de memoria de un dispositivo a otro. En lugar de que la CPU inicie la transferencia, la transferencia se lleva a cabo por el controlador DMA. Un ejemplo típico es mover un bloque de memoria desde una memoria externa a una interna más rápida. Tal operación no ocupa el procesador, por lo tanto, los ciclos que se irían a utilizar si no se implementase DMA se utilizan para planificar otras tareas. Las transferencias DMA son esenciales para aumentar el rendimiento de aplicaciones que requieran muchos recursos.

Cabe destacar que aunque no se necesite a la CPU para la transacción de datos, sí que se necesita el bus del sistema (tanto bus de datos como bus de direcciones), por lo que existen diferentes estrategias para regular su uso, permitiendo así que no quede totalmente acaparado por el controlador DMA.

Controlador de DMA

El Controlador de DMA (DMAC) debe actuar como maestro del bus durante la transferencia DMA y debe ser capaz de

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

Etapas de una transferencia DMA

Inicialización de la transferencia

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

Inicialización del interfaz (*Bus master: CPU-Bus slave: Interfaz*)

- Nº de bytes a transferir
- Tipo de transferencia (lectura/escritura)
- Otra información de control (pista, sector, etc.)

Inicialización controlador DMA(*Bus master: CPU-Bus slave: DMAC*)

- Nº de bytes o palabras a transferir
- Tipo de transferencia (lectura/escritura)
- Dirección de memoria inicial para la transferencia
- Nº de canal (para DMAs con varios canales)

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

Realización de la transferencia

- Cuando el periférico está listo para realizar la transferencia se lo indica al DMAC
- El DMAC pide el control del bus y se realiza la transferencia entre el periférico y la memoria
- Bus master: DMAC + Periférico - Bus slave: Memoria
  - Después de la transferencia de cada palabra se actualizan los registros del DMAC
    - Nº de bytes o palabras a transferir
    - Dirección de memoria

Finalización de la transferencia

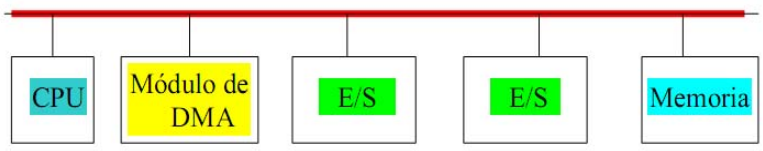
- El DMAC libera el bus y devuelve el control a la CPU
- El DMAC suele activar una señal de interrupción para indicar a la CPU la finalización de la operación de E/S solicitada

Estrategias de transferencia por DMA

A continuación se exponen diferentes técnicas para realizar la transferencia de datos. El uso de cada una de ellas dependerá de las características que se deseen primar en un sistema.

Estrategias de transferencia por DMA	Descripción
Por robo de ciclo	Se basa en usar uno o más ciclos de CPU por cada instrucción que se ejecuta (de ahí el nombre). De esta forma se consigue una alta disponibilidad del bus del sistema para la CPU, aunque, en consecuencia, la transferencia de los datos será considerablemente lenta. Este método es el que se usa habitualmente ya que la interferencia con la CPU es muy baja. VENTAJAS: No se degrada el rendimiento del sistema. DESVENTAJAS: La transferencia tarda más tiempo en llevarse a cabo.
Por ráfagas	Consiste en enviar el bloque de datos solicitado mediante una ráfaga, ocupando el bus del sistema hasta finalizar la transmisión. Así se consigue la máxima velocidad, sin embargo la CPU no podrá usar el bus durante todo ese tiempo, por lo que permanecería inactiva. VENTAJAS: La transferencia se realiza de forma rápida. DESVENTAJAS: Durante el tiempo que dura la transferencia la CPU no puede utilizar el bus con memoria, lo que puede degradar el rendimiento del sistema
transparente	Se trata de usar el bus del sistema cuando se tiene certeza de que la CPU no lo necesita, como por ejemplo en aquellas fases del proceso de ejecución de las instrucciones donde nunca se usa ya que la CPU realiza tareas internas (ej. fase de decodificación de la instrucción). De esta manera, como su nombre indica, la DMA permanecerá transparente para la CPU y la transferencia se hará sin obstaculizar la relación CPU-bus del sistema. Como desventaja, la velocidad de transferencia es la más baja posible.
Scatter-gather	Permite la transferencia de datos a varias áreas de memoria en una transacción DMA simple. Es equivalente al encadenamiento de múltiples peticiones DMA simples. De nuevo, el objetivo es liberar a la CPU de las tareas de copia de datos e interrupciones de entrada/salida múltiples.

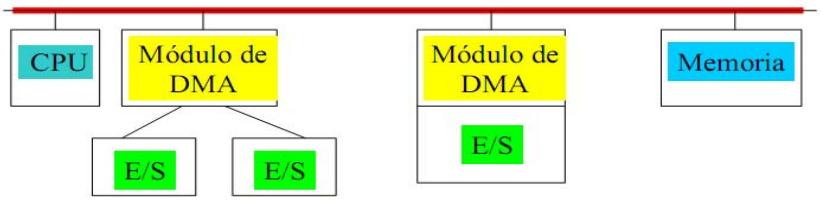
Configuraciones del DMA



Bus único, módulo de DMA independiente.

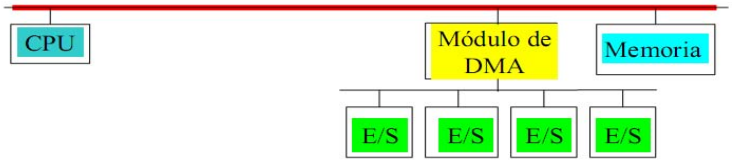
- El DMA utiliza E/S programadas para transferir entre la memoria y los módulos de E/S.
- Cada transferencia utiliza dos veces el bus.

- Desde el módulo de E/S al DMA y desde el DMA a la memoria.
- La CPU debe “esperar” dos veces.



Bus único, DMA integrado al módulo de E/S o conectado a varios módulos de E/S.

- El módulo de E/S puede controlar varios dispositivos.
- Cada transferencia usa el bus una vez.
- Del DMA a la memoria.
- La CPU se detiene sólo una vez
- comunicación de la finalización de la tarea.



Bus de E/S separado del bus del sistema.

- Derivado de la configuración anterior.
- Cada transferencia usa el bus del sistema una sola vez.
- Del DMA a la memoria.
- A la CPU se la interrumpe sólo una vez.
- Configuración fácilmente expandible.
- Como en el caso anterior la transferencia de datos entre el DMA y los módulos de E/S se realizan fuera del bus del sistema.

Canales de E/S

Características de los Canales de E/S

- Los canales representan una extensión al concepto de DMA.
- Tienen la habilidad de ejecutar instrucciones de E/S.
- Completo control de la transferencia de datos por lo tanto la CPU no ejecuta instrucciones de E/S.
- Instrucciones almacenadas en memoria principal que serán ejecutadas por un procesador de propósito especial en el canal.
- La CPU inicia una transferencia de E/S instruyendo al canal para ejecutar el programa que está en memoria.
- Este programa especifica dispositivos, áreas de memoria a usar, prioridades y acciones ante errores.
- El canal siguiendo las instrucciones controla la transferencia de datos.

Tipos de canales de E/S

Selector

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

Multiplexor

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

- Acepta y transmite caracteres.
- Multiplexor de bloques:
- Intercala bloques de datos desde distintos dispositivos.

La interfaz externa

- Provista para un periférico desde un módulo de E/S.
- ¿Serial o paralelo?
- Paralela: tape, disco (dispositivo de alta velocidad).
- Serial: terminales, mouse.
- Existe un módulo que dialoga con el dispositivo. Los pasos son:
- El módulo envía señal de control pidiendo permiso para enviar datos al dispositivo.
- Este confirma el pedido.
- El módulo transfiere los datos.
- El periférico confirma la recepción de los mismos.
- Uso de buffers internos en el módulo para compensación de velocidades.

Tipos de interfaces

- La conexión entre un módulo de E/S y los dispositivos externos puede ser:
- Punto a punto: línea dedicada.
- Ej: teclado, impresora, modem ...etc.
- Multipunto: ídem a un bus externo.
- Ej: dispositivos de almacenamiento en masa externos (discos SCSI).

Procesadores Superescalares

Un Procesador superescalar es aquél que usa múltiples cauces de instrucciones independientes. Cada cauce consta de múltiples etapas, de modo que pude tratar varias instrucciones a la vez.

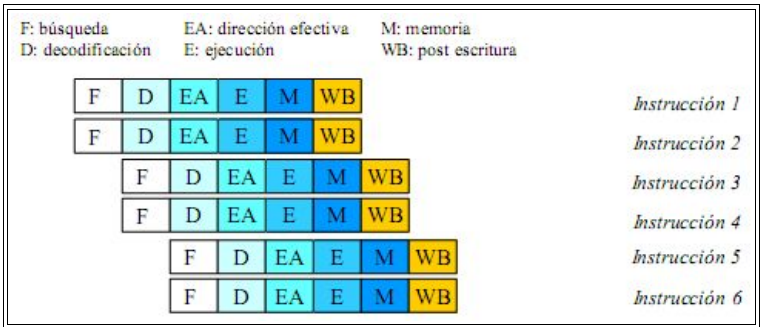


Ilustración 3: Ejecución superescalar de instrucciones

El hecho de que haya vario cauces introduce un nuevo nivel de paralelismo, permitiendo que varios flujos de instrucciones se procesen simultáneamente. Un procesador superescalar saca provecho de lo que se conoce como ***paralelismo a nivel de instrucciones***, que hace referencia al grado en que las instrucciones de un programa pueden ejecutarse en paralelo.

Se pueden llevar a cabo (completar) más de una instrucción simultáneamente.
Conlleva la duplicación de algunas o todas las partes de la CPU/ALU.

- Captar instrucciones múltiples al mismo tiempo.
- Ejecutar sumas y multiplicaciones simultáneamente.
- Ejecutar carga/almacenamiento, mientras se lleva a cabo una operación en ALU.

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

Gran responsabilidad del compilador en el aprovechamiento del procesamiento superescalar.

Típicamente, un procesador superescalar capta varias instrucciones a la vez y, a continuación, intenta encontrar instrucciones cercanas a que sean independientes entre sí puedan, por consiguiente, ejecutarse en paralelo. Si la entrada de una instrucción depende de la salida de una instrucción precedente, la segunda instrucción no puede completar su ejecución al mimo tiempo ni antes que la primera. Una vez que se han identificado tales dependencias, el procesador puede emitir y completar instrucciones en un orden diferente al del código máquina original.

El procesador puede eliminar algunas dependencias innecesarias mediante el uso de registros adicionales y el renombramiento de las referencias a registros en el código original.

Mientras que los procesadores RISC puros emplean con frecuencia saltos retardados para maximizar la utilización del cauce de instrucciones, este método es menos apropiado para las máquinas superescalares. En

lugar de eso, la mayoría de las máquinas superescalares emplean métodos tradicionales de predicción de saltos para aumentar su rendimiento.

Características de los procesadores superescalares

La estructura típica de un procesador superescalar consta de un pipeline con las siguientes etapas:

- lectura (fetch)
- decodificación (decode)
- lanzamiento (dispatch)
- ejecución (execute)
- escritura (writeback)
- finalización (retirement)

El número máximo de instrucciones en una etapa concreta del pipeline se denomina grado, así un procesador superescalar de grado 4 en lectura (fetch) es capaz de leer como máximo cuatro instrucciones por ciclo. El grado de la etapa de ejecución depende del número y del tipo de las unidades funcionales.

Un procesador superescalar es capaz de ejecutar más de una instrucción simultáneamente únicamente si las instrucciones no presentan algún tipo de dependencia (hazard). Los tipos de dependencia entre instrucciones son :

- **dependencia estructural**, esta ocurre cuando dos instrucciones requieren el mismo tipo unidad funcional y su número no es suficiente.
- **dependencia de datos**, esta ocurre cuando una instrucción necesita del resultado de otra instrucción para ejecutarse, por ejemplo R1<=R2+R3 y R4<=R1+5.
- **dependencia de escritura o falsa dependencia** , esta ocurre cuando dos instrucciones necesitan escribir en la misma memoria, por ejemplo R1<=R2+R3 y R1<=R1+5.

La detección y resolución de las dependencias entre instrucciones puede ser estática (durante la compilación) o dinámica, es decir, a medida que se ejecuta un programa, generalmente durante las etapas de codificación y lanzamiento de las instrucciones.

La detección y resolución dinámica de las dependencias entre instrucciones suele realizarse mediante alguna variante del algoritmo de ***Tomasulo*** que permite la ejecución de instrucciones en un orden distinto al del programa también llamada ejecución en desorden.

Las arquitecturas superescalares adolecen de una estructura compleja y llevan a un mal aprovechamiento de sus recursos debido en parte a la dificultad en encontrar suficientes instrucciones paralelizables. Una forma de obtener un mayor número de instrucciones paralelizables es aumentar la ventana de instrucciones, es decir el conjunto de instrucciones que la unidad de lanzamiento considera como candidatas a ser lanzadas en un momento dado. Desafortunadamente la complejidad del procesador superescalar aumenta desproporcionadamente con respecto al tamaño de dicha ventana lo que se traduce por un ralentizamiento general del circuito. Otra forma de obtener más instrucciones paralelizables es manipulando instrucciones de más de un programa a la vez, lo que se conoce bajo el nombre de multitarea simultánea o multithreading simultáneo.

La implementación de la arquitectura superescalar requiere lo siguiente:

- Estrategias de captación simultánea de múltiples instrucciones.
- Lógica para determinar dependencias verdaderas entre valores de registros y mecanismos para comunicar esos valores.
- Mecanismos para iniciar o emitir múltiples instrucciones en paralelo.
- Recursos para la ejecución en paralelo de múltiples instrucciones.
- Mecanismos para entregar el estado del procesador en un orden correcto.

Segmentación de Instrucciones
Segmentación de cauce

La segmentación de cauce, también denominada pipeline, es una técnica empleada en el diseño de procesadores, basada en la división de la ejecución de las instrucciones en etapas, consiguiendo así que una instrucción empiece a ejecutarse antes de que hayan terminado las anteriores y, por tanto, que haya varias instrucciones procesándose simultáneamente.

Etapas básicas de un cauce

- **Búsqueda o Captación** (F, Fetch)
  - Acceso a memoria (búsqueda de la instrucción)
  - Incremento del PC
- **Decodificación** (D, Decode)
  - Decodificación de la instrucción

- Obtención de los operandos
- **Ejecución** (E, Execute)
  - Si es procesamiento: ejecución en la ALU
  - Si es acceso a memoria: obtención de la dirección efectiva
  - Si es salto: cálculo del destino y decisión de salto (s/n)

Cada una de las etapas debe completar sus acciones en un ciclo de reloj, pasando sus resultados a la etapa siguiente y recibiéndolos de la anterior. Para eso es necesario almacenar los datos en registros intermedios. Cualquier valor que pueda ser necesario en una etapa posterior debe irse propagando a través de esos registros intermedios hasta que ya no sea necesario.

Para conseguir la segmentación es necesario que una instrucción utilice solamente una etapa en cada ciclo de ejecución.

Ya que todas las etapas deben de tardar lo mismo en su ejecución, el tiempo de ciclo será el de la etapa más lenta, más el del retardo provocado por la utilización de los registros intermedios. Comparando este esquema con el multiciclo, el tiempo de ciclo será más lento, pero el CPI (Ciclos Por Instrucción) será menor, lo que provoca un aumento del rendimiento. Ya que si no tenemos en cuenta los riesgos estructurales (que pueden provocar paradas en el pipeline), tendríamos que en cada ciclo de reloj, termina de ejecutarse una instrucción (CPI=1).

Características

- La segmentación es una técnica de mejora de prestaciones a nivel de diseño hardware.
- La segmentación es invisible al programador.
- Necesidad de uniformizar las etapas.
  - Al tiempo de la más lenta
  - Al número de etapas máximo
- El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones (RISC).

Cauce segmentado en 4 etapas

- **Búsqueda** (F)
- **Decodificación** (D)
- **Ejecución** (E)

Consideramos una nueva etapa

- **Actualización** (W)
  - Almacenamiento del resultado en memoria (si lo hay)

Análisis de la segmentación

Atascos de un cauce (stall)

Situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que le corresponde.

- **Estructurales**
  - Provocados por conflictos por los recursos
- **Por dependencia de datos**
  - Ocurren cuando dos instrucciones se comunican por medio de un dato (ej.: una lo produce y la otra lo usa)
- **Por dependencia de control**
  - Ocurren cuando la ejecución de una instrucción depende de cómo se ejecute otra (ej.: un salto y los 2 posibles caminos)

Riesgos estructurales

Dos instrucciones necesitan utilizar el mismo recurso hardware en el mismo ciclo.

- Dos accesos simultáneos a memoria, al banco de registros o a la ALU.

Posibles soluciones

- Replicación de unidades funcionales
- Caches diferenciadas de datos e instrucciones
- Más de un puerto de acceso a memoria o al banco de registros.

Riesgos por dependencias de datos

- Condición en la que los operandos fuente o destino de una instrucción no están disponibles en el momento en que se necesitan en una etapa determinada del cauce.
  - Alguna operación tiene que retardarse.

Se necesita realizar la detección de riesgos

→ Unidad de detección de riesgos y/o compilador mas complejo

Posibles soluciones

- **Hardware**:
  - Adelantamiento de operandos (forwarding)

Si el dato necesario está disponible a la salida de la ALU se lleva a la entrada de la etapa correspondiente sin esperar a la escritura.
Necesita registros extras (fuentes, en buffer y resultado).

- **Software**:
  - Introducción de ciclos de NOP por el compilador
    - Generamos Retardo
  - Reordenación de las instrucciones
    - ejecución “fuera de orden”

Tipos de dependencias de datos

- **Lectura después de Escritura (RAW, dependencia verdadera)**
  - vista hasta ahora: una instrucción genera un dato que lee otra posterior.
- **Escritura después de Escritura (WAW, dependencia en salida)**
  - una instrucción escribe un dato después que otra posterior sólo se da si se deja que las instrucciones se adelanten unas a otras.
- **Escritura después de Lectura (WAR, antidependencia)**
  - una instrucción modifica un valor antes de que otra anterior que lo tiene que leer, lo lea.
  - no se puede dar en nuestro cauce simple.

Riesgos de control (o instrucciones)

Necesidad de tomar una decisión basada en los resultados de una instrucción mientras las otras se están ejecutando.

- **Instrucciones de salto**
  - **Incondicional**. Penalización de salto.
    - La dirección de destino se debe determinar lo más pronto posible, dentro del cauce, para reducir la penalización.
  - **Condicional**
    - Introduce riesgo adicional por la dependencia entre la condición de salto y el resultado de una instrucción previa.

Posibles soluciones en riesgos de control

- Cola de instrucciones y prebúsqueda
  - Captar las instrucciones antes que sean necesarias y ponerlas en una cola ⇒ Unidad de Emisión.
  - La unidad de Emisión debe detectar las instrucciones de salto

Tratamiento de saltos

Opciones para tratar los saltos mas comunes:

- **Flujos múltiples**
  - Varios cauces (uno por cada opción de salto).
  - Precaptan cada salto en diferentes cauces.
  - Utilizar el cauce correcto.
  - Desventajas:
    - Provoca retardos en el acceso al bus y a los registros.
    - Si hay múltiples saltos, se necesita un mayor número de cauces.
- **Precaptar el destino del salto**
  - Se precapta la instrucción destino del salto, además de las instrucciones siguientes a la bifurcación.
  - La instrucción se guarda hasta que se ejecute la instrucción de bifurcación.
- **Buffer de bucles**

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

- **Salto retardado**
  - No hacer el salto hasta que sea necesario.
    - Siempre se ejecuta la siguiente instrucción secuencial, el salto se ejecuta después del retardo de una instrucción.
  - Las instrucciones en los huecos de retardo de salto (delay-slot) se captan siempre.
  - Se trata de situar instrucciones útiles (que no dependan del salto) en los huecos de retardo. Si no es posible, se utilizan instrucciones NOP.
  - Requiere reordenar las instrucciones.
- **Predicción de saltos**
  - **Técnicas estáticas**
    - Predecir que nunca se salta:
      - Asume que el salto no se producirá.
      - Siempre capta la siguiente instrucción.
    - Predecir que siempre se salta:
      - Asume que el salto se producirá.
      - Siempre capta la instrucción destino del salto.
    - Predecir según el código de operación:
      - Hay instrucciones con más probabilidades de saltar.
      - La tasa de acierto puede llegar a alcanzar un 75%.
  - **Técnicas dinámicas**
    - Conmutador saltar/no saltar:
      - Basado en la historia de las instrucciones.
      - Eficaz para los bucles.
    - Tabla de historia de saltos (branch-target buffer)
      - Pequeña cache asociada a la etapa de búsqueda (F)
      - Tres campos:
        - Dirección de una instrucción de bifurcación
        - Información de la instrucción destino
        - Dirección del destino o Instrucción destino
        - N bits de estado (historia de uso)

Procesadores Supersegmentados

La supersegmentación 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 e reloj interna, lo que permite la realización de dos tareas en un ciclo de reloj externo.

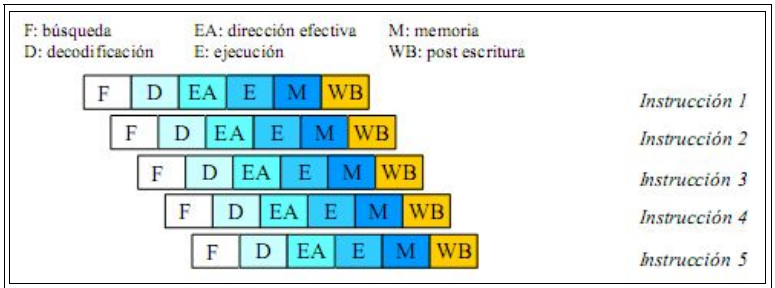


Ilustración 4: Ejecución supersegmentada de instrucciones

Muchas operaciones no necesitan todo un ciclo de reloj por lo que obtendremos mayores prestaciones subdividiendo el ciclo de reloj en sub-intervalos. En consecuencia:

- Resulta una mayor frecuencia del ciclo de reloj.

División de las etapas “macro” del cauce segmentado en sub-etapas más pequeñas (y, por tanto, más rápidas) y se transmiten los datos a la mayor velocidad del ciclo de reloj.

El tiempo para las instrucciones individuales no varía.

- Aumenta el grado del paralelismo.
- Incrementa la aceleración percibida.

Superescalar vs. Supersegmentado

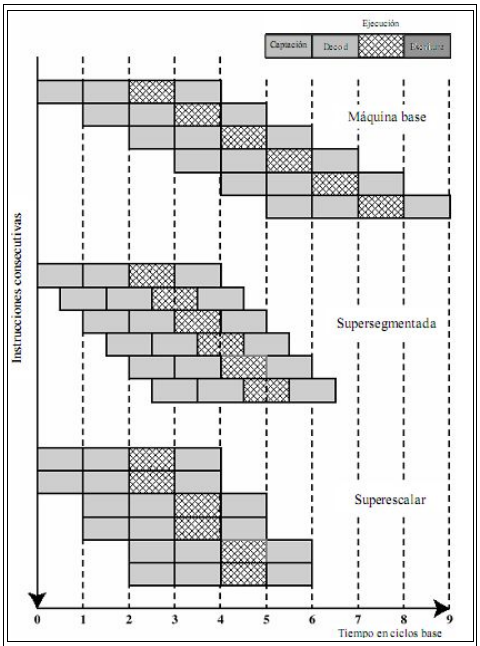


Ilustración 5: Comparación entre ejecuciones supersegmentadas, superescalares y maquinas base

RISC

Aunque los sistemas RISC *(Reduced Instruction Set Computer)* se han definido y diseñado de diversas formas por parte de diferentes grupos, los elementos clave compartidos por la mayoría (no todos) de los diseños son:

- Un gran numero de registros e 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.

El objetivo de diseñar máquinas con esta arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. Las máquinas RISC protagonizan la tendencia actual de construcción de microprocesadores. PowerPC, DEC Alpha, MIPS, ARM, ... son ejemplos de algunos de ellos.

La razón por la que se opta por los procesadores RISC, es para contrarrestar los inconvenientes del CISC, como el hecho de que el software resulta mucho más caro que el hardware, y que el nivel del lenguaje era cada vez más complicado. Debido a esto tenemos que hay un salto semántico (Diferencias entre operaciones HLL y operaciones de la Arquitectura). Todo esto conduce a que se tengan repertorios de instrucciones grandes, más modos de direccionamiento y varias sentencias de HLL implementadas en el Hardware (por ejemplo, el CASE del VAX).

En fin, los procesadores CISC se crearon con la finalidad de facilitar el trabajo del escritor de compiladores, mejorar la eficiencia de la ejecución(porque utilizan secuencias complejas de operaciones en microcódigo) y dar soporte a HLL más complejos.

Se han hecho estudios relacionados que identifican que instrucciones se utilizan más, la cantidad y el origen y el destino de las llamadas a procedimientos, con la intención de ofrecer mejor soporte para los HLL optimizando las prestaciones de las características más usadas y que más tiempo consumen. Es por ello que se llegó a la conclusión que es deseable

- Usar un gran número de registros (Optimizando las referencias a operandos). Esto se puede hacer de dos formas:
  - Aproximación por Software:
    - El compilador es necesario para asignar registros.
    - Asignación de registros a las variables que se usen mas en un período de tiempo dado.
    - Requiere el uso de sofisticados algoritmos de análisis de programas.
  - Aproximación por Hardware:
    - Utilización de más registros.
    - De esta manera, más variables pueden mantenerse en registros durante periodos de tiempo más largos.

  Entonces el hecho de tener muchos Registros tiende a reducir el acceso a memoria. Por estudios anteriores esto indicaría que deberíamos almacenar las variables escalares locales en registros.

Pero todo esto conlleva un problema: Cada llamada de procedimiento/función cambia la 'localidad', por lo que los parámetros deben ser pasados, los resultados tienen que ser devueltos, las variables de los programas de llamada tienen que ser restauradas.

- Prestar cuidadosa atención al diseño de los cauces de instrucciones (utilizando predicción de bifurcaciones, etc).
- Es recomendable un repertorio de instrucciones simplificado (reducido).

¿Por que CISC?

¿Simplificación del compilador?

• Ésta primera razón parece obvia
• Instrucciones de máquina complejas son difíciles de aprovechar
• La optimización es más difícil: tamaño, velocidad.

¿Programas más pequeños?

• El programa ocupa menos memoria, pero la memoria hoy día es muy barata.
• El número de bits de memoria que ocupa no tiene porqué ser más pequeño al tener menos instrucciones
• Más instrucciones necesitan códigos de operación más largos.
• Las referencias a registros necesitan menos bits.

¿Programas más rápidos?

• Propensión a usar las instrucciones más sencillas.
• Unidad de control más compleja.
• Memoria de control del microprograma más grande.
• Aumenta el tiempo de ejecución de las instrucciones simples.

No está nada claro que la tendencia hacia CISC sea apropiada. No existe una clara barrera diferenciadora entre RISC y CISC, y por lo tanto muchos diseños incluyen características de ambos criterios (Por ejemplo, PowerPC y Pentium II).

En fin, la controversia entre la elección del RISC o CISC se enfoca en los siguientes aspectos de la eficiencia de la ejecución de programas:

Cuantitativa:
• Comparación del tamaño de los programas y su velocidad de ejecución

Cualitativa:
• Revisión de soporte de lenguajes de alto nivel y uso óptimo de los recursos VLSI.

Pero el problema con las comparaciones son que:

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

Memoria Caché

El objetivo de la memoria caché es lograr que la velocidad de la memoria sea lo más rápida posible, consiguiendo al mismo tiempo un tamaño grande al precio de memorias semiconductoras menos costosas. 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 así. Se entrega dicha palabra al procesador. Si no, un bloque de memoria principal, consistente en un cierto número de palabras se transfiere a la caché 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 caché para satisfacer una referencia a memoria simple, es probable que se hagan referencias futuras a otras palabras del mismo bloque.

Ahora bien, hay dos tipos de principios de localidad:

Localidad Espacial: Los elementos cuyas direcciones están cercanas tienden a ser referenciados.

Localidad Temporal: Los elementos (datos o instrucciones) accedidos recientemente, también lo serán en un futuro próximo.

Diseño de las Memorias Caché

En el diseño de la memoria caché se deben considerar varios factores que influyen directamente en el rendimiento de la memoria y por lo tanto en su objetivo de aumentar la velocidad de respuesta de la jerarquía de memoria. Estos factores son las políticas de ubicación, extracción, reemplazo, escritura y el tamaño de la caché y de sus bloques.

Tipo de Política	Clasificación de Política	Descripción
Ubicación (Correspondencia)	**Directa**	Consiste en hacer corresponder un bloque de memoria principal a sólo una línea posible de cache. Su principal desventaja es que hay una posición concreta de cache para cada bloque dado. Por ello, si un programa referencia repetidas veces a palabras de 2 bloques diferentes asignados en la misma línea, dichos bloques se estarían intercambiando continuamente en la caché y la tasa de aciertos sería baja.
	Asociativa	Permite que cada bloque de memoria principal pueda cargarse en cualquier línea de memoria caché. La principal desventaja es la completa circuitería necesaria para examinar en paralelo las etiquetas de todas las lineas de caché.
	Asociativa por conjuntos	Un bloque puede almacenarse en un conjunto restringido de lugares en la caché. Un conjunto es un grupo de lineas de la caché. Un bloque de memoria podrá ubicarse en cualquier posición dentro del conjunto asociado de la memoria caché.
Extracción	**Por demanda**	Un bloque sólo se trae a memoria caché cuando ha sido referenciado y se produzca un fallo.
	Con prebúsqueda	Cuando se referencia el bloque i-ésimo de memoria principal, se trae además el bloque (i+1)-esimo. Esta política se basa en la propiedad de localidad espacial de los programas.
Reemplazo	**Aleatoria**	El bloque es reemplazado de forma aleatoria.
	FIFO	Se usa un algoritmo First In First Out (primero entrado primero salido) para determinar qué bloque debe abandonar la caché. Este algoritmo generalmente es poco eficiente.
	LRU	Se sustituye el bloque que hace más tiempo que no se ha utilizado.
	LFU	Se reemplaza el bloque que se ha usado con menos frecuencia.
Escritura	**Escritura inmediata (Escritura directa)**	Cuando se escribe en un bloque que se encuentra en memoria caché, la información se modifica también simultáneamente en memoria principal, manteniendo así la coherencia en todo momento. Suele combinarse con la técnica de "No carga en escritura" (No Write Allocation) que significa que, cuando haya que escribir en un bloque que no se encuentra en la caché, la modificación se realizará únicamente en memoria principal, sin traer dicho bloque a caché, y además sólo se actualizará la palabra concreta que haya cambiado. Con múltiples CPU, observar el tráfico a memoria principal para mantener actualizada cada cache local. Se genera mucho tráfico y retrasa la escritura.
	Post-Escritura (Aplazada)	Cuando se escribe en un bloque que se encuentra en memoria caché, queda marcado como basura usando un bit especial llamado normalmente dirty bit o bit de basura. Cuando el bloque sea desalojado de memoria caché (mediante la correspondiente política de reemplazo), se comprueba el bit de basura, y si está activado se escribe la información de dicho bloque en memoria principal. Esta política suele combinarse con la técnica de "Carga en escritura" (Write Allocation), que significa que, cuando haya que escribir en un bloque que no se encuentra en la caché, traeremos a caché el bloque en cuestión y lo modificaremos ahí. La memoria principal se actualiza en el reemplazo y puede contener información errónea en algún momento.

En cuanto al **tamaño de línea** que tiene que tener una memoria caché, podemos destacar situaciones claves:

- Bloques más grandes reducen el número de bloque 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 sobrescriba 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 necesario a corto plazo.

La relación entre tamaño de bloque y tasa de aciertos es compleja, dependiendo de las características de localidad de cada 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.

Numero de caches

Cuando se introdujeron originalmente las caches, un sistema tenía normalmente sólo una caché. Más recientemente, se ha convertido en una norma el uso de múltiples caché. Hay dos aspectos de diseño relacionados con ese tema, que son. El número de niveles de caché y el uso de caché unificada frente a caché partida.

Con el aumento de la densidad de integración, ha sido posible tener una caché en el mismo chip del procesador: caché “con chip”. Comparada con la accesible a través de un bus externo, la caché “on-chip” reduce la actividad el bus externo del procesador y, por tanto, reduce los tiempo de se ejecución e incrementa las prestaciones globales del sistema. Cuando al instrucción o dato requeridos se encentran en la caché “on-chip”, se elimina el acceso al bus. Debido a que los caminos de datos internos al procesador con muy cortos en comparación con la longitud de los buses, los accesos en la caché “on-chip” se efectúan apreciablemente más rápidos que los ciclos de bus, incluso en ausencia de estados de espera. Además, durante este periodo el bus está libre para realizar otras transferencias.

La inclusión de una caché “on-chip” deja abierta la cuestión de si es además deseable una caché externa u “off-chip”. Normalmente la respuesta es afirmativa, y los diseños más actuales incluyen tanto cacheo n chip como externa. La estructura resultante se conoce como caché de dos niveles, siendo la caché interna el nivel 1 (L1), y la externa el nivel 2 (L2). La razón por la que se incluye una caché L2 es la siguiente. Si no hay caché L2 y el procesador hace una petición de acceso a una posición de memoria que no está en la caché L1, entonces el procesador debe acceder a la DRAM o a la ROM a través del bus. Debido a la lentitud usual del bus y a los tiempos de acceso de las memorias, se obtienen bajas prestaciones. Por otra parte, si se utiliza una caché L2 SRAM, entonces, con frecuencia, la información que falta puede recuperarse rápidamente. Si la SRAM es suficientemente rápida para adecuarse a la velocidad del bus, los datos pueden accederse con cero estados de espera, el tipo de transferencia de bus más rápido.

La mejora potencial del uso de una caché L2 depende de las tasas de aciertos en ambas caché L1 y L2. Varios estudios han demostrado que, en general, el uso de un segundo nivel de caché mejora las prestaciones.

Cuando hicieron su aparición las caché “on-chip”, muchos de los diseños contenían una sola caché para almacenar las referencias, tanto a datos como a instrucciones. Más recientemente, se ha hecho normal separar la caché en dos: una dedicada a instrucciones y otra a datos.

Una caché unificada tiene varias ventajas potenciales:

Para un tamaño dado de caché, una unificada tiene una tasa de aciertos mayor que una partida, ya que nivela automáticamente la caga entre captación de instrucciones y de datos. Es decir, 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.
Sólo se necesita diseñar e implementar una caché.

A pesar de estas ventajas, la tendencia es hacia cachés partidas, particularmente para máquinas superescalares, tales como el Pentium II y el PowerPC, en las que se enfatiza la ejecución paralela de instrucciones y la pre-captación de instrucciones futuras previstas. La ventaja clave del diseño de caché partida es que elimina la competición por la caché entre el procesador de instrucción y la unidad de ejecución. Esto es importante en diseño que cuentan con segmentación de cauce (pipelining) de instrucciones. Normalmente, el procesador pactará instrucciones anticipadamente y llenará un buffer con las instrucciones que van a ejecutarse. Supongamos ahora u se tiene una caché unificada de instrucciones/datos. Cuando la unidad de ejecución realiza un acceso a memoria para cargar y almacenar datos., sen envía la petición a la caché unificada. Si, al mismo tiempo, el pre-captador de instrucciones emite una petición de lectura de una instrucción a la caché, dicha petición será temporalmente bloqueada para que la caché pueda servir primero a la unidad de ejecución, permitiéndole

completar la ejecución de la instrucción en curso. Esta disputa por la caché puede degradar las prestaciones, interfiriendo con el uso eficiente del cauce segmentación de instrucciones. La caché partida supera esta dificultad.

RAID

En informática, el acrónimo RAID (originalmente del inglés Redundant Array of Inexpensive Disks, 'conjunto redundante de discos baratos', en la actualidad también de Redundant Array of Independent Disks, 'conjunto redundante de discos independientes') hace referencia a un sistema de almacenamiento que usa múltiples discos duros entre los que distribuye o replica los datos.

Dependiendo de su configuración (a la que suele llamarse «nivel»), los beneficios de un RAID respecto a un único disco son uno o varios de los siguientes:

- mayor integridad.
- mayor tolerancia a fallos.
- mayor throughput (rendimiento).
- mayor capacidad.

En sus implementaciones originales, su ventaja clave era la habilidad de combinar varios dispositivos de bajo coste y tecnología más antigua en un conjunto que ofrecía mayor capacidad, fiabilidad, velocidad o una combinación de éstas que un solo dispositivo de última generación y coste más alto.

En el nivel más simple, un RAID combina varios discos duros en una sola unidad lógica. Así, en lugar de ver varios discos duros diferentes, el sistema operativo ve uno solo. Los RAIDs suelen usarse en servidores y normalmente (aunque no es necesario) se implementan con unidades de disco de la misma capacidad. Debido al decremento en el precio de los discos duros y la mayor disponibilidad de las opciones RAID incluidas en los chipsets de las placas base, los RAIDs se encuentran también como opción en los ordenadores personales más avanzados. Esto es especialmente frecuente en los computadores dedicados a tareas intensivas de almacenamiento, como edición de audio y vídeo.

La especificación RAID original sugería cierto número de «niveles RAID» o combinaciones diferentes de discos. Cada una tenía ventajas y desventajas teóricas. Con el paso de los años, han aparecido diferentes implementaciones del concepto RAID. La mayoría difieren sustancialmente de los niveles RAID idealizados originales, pero se ha conservado la costumbre de llamarlas con números. Esto puede resultar confuso, dado que una implementación RAID 5, por ejemplo, puede diferir sustancialmente de otra. Los niveles RAID 3 y RAID 4 son confundidos con frecuencia e incluso usados indistintamente.

La misma definición de RAID ha estado en disputa durante años. El uso de término «redundante» hace que muchos objeten sobre que el RAID 0 sea realmente un RAID. De igual forma, el cambio de «barato» a «independiente» confunde a muchos sobre el pretendido propósito del RAID. Incluso hay algunas implementaciones del concepto RAID que usan un solo disco. Pero en general, diremos que cualquier sistema que emplee los conceptos RAID básicos de combinar espacio físico en disco para los fines de mejorar la fiabilidad, capacidad o rendimiento es un sistema RAID.

Historia

A Norman Ken Ouchi de IBM le fue concedida en 1978 la Patente USPTO nº 4,092,732, titulada «Sistema para recuperar datos almacenados en una unidad de memoria averiada» (System for recovering data stored in failed memory unit), cuyas demandas describen los que más tarde sería denominado escritura totalmente dividida (full striping). Esta patente de 1978 también menciona la copia espejo (mirroring o duplexing), que más tarde sería denominada RAID 1, y la protección con cálculo de paridad dedicado, que más tarde sería denominada RAID 4, que eran ya arte previo en aquella época.

La tecnología RAID fue definida por primera vez en 1987 por un grupo de informáticos de la Universidad de California, Berkeley. Este grupo estudió la posibilidad de usar dos o más discos que aparecieran como un único dispositivo para el sistema.

En 1988, los niveles RAID 1 a 5 fueron definidos formalmente por David A. Patterson, Garth A. Gibson y Randy H. Katz en el ensayo «Un Caso para Conjuntos de Discos Redundantes Económicos (RAID)» —A Case for Redundant Arrays of Inexpensive Disks (RAID)—, publicado en la Conferencia SIGMOD de 1988 (págs. 109-116). El término «RAID» se usó por vez primera en este ensayo, que dio origen a toda la industria de los conjuntos de discos.

Implementaciones

La distribución de datos en varios discos puede ser gestionada por hardware dedicado o por software. Además, existen sistemas RAID híbridos basados en software y hardware específico.

Con la implementación por software, el sistema operativo gestiona los discos del conjunto a través de una controladora de disco normal (IDE/ATA, Serial ATA, SCSI, SAS o Fibre Channel). Considerada tradicionalmente una solución más lenta, con el rendimiento de las CPUs modernas puede llegar a ser más rápida que algunas implementaciones hardware, a expensas de dejar menos tiempo de proceso al resto de tareas del sistema.

Una implementación de RAID basada en hardware requiere al menos una controladora RAID específica, ya sea como una tarjeta de expansión independiente o integrada en la placa base, que gestione la administración de los discos y efectúe los cálculos de paridad (necesarios para algunos niveles RAID). Esta opción suele ofrecer un mejor rendimiento y hace que el soporte por parte del sistema operativo sea más sencillo (de hecho, puede ser totalmente transparente para éste). Las implementaciones basadas en hardware suelen soportar sustitución en caliente (hot swapping), permitiendo que los discos que fallen puedan reemplazarse sin necesidad de detener el sistema.

En los RAIDs mayores, la controladora y los discos suelen montarse en una caja externa específica, que a su vez se conecta al sistema principal mediante una o varias conexiones SCSI, Fibre Channel o iSCSI. A veces el sistema RAID es totalmente autónomo, conectándose al resto del sistema como un NAS.

Los RAIDs híbridos se han hecho muy populares con la introducción de controladoras RAID hardware baratas. En realidad, el hardware es una controladora de disco normal sin características RAID, pero el sistema incorpora una aplicación de bajo nivel que permite a los usuarios construir RAIDs controlados por la BIOS. Será necesario usar un controlador de dispositivo específico para que el sistema operativo reconozca la controladora como un único dispositivo RAID. Estos sistemas efectúan en realidad todos los cálculos por software (es decir, los realiza la CPU), con la consiguiente pérdida de rendimiento, y típicamente están restringidos a una única controladora de disco.

Una importante característica de los sistemas RAID por hardware es que pueden incorporar un caché de escritura no volátil (con alimentación de respaldo por batería) que permite aumentar el rendimiento del conjunto de discos sin comprometer la integridad de los datos en caso de fallo del sistema. Esta característica no está obviamente disponible en los sistemas RAID por software, que suelen presentar por tanto el problema de reconstruir el conjunto de discos cuando el sistema es reiniciado tras un fallo para asegurar la integridad de los datos. Por el contrario, los sistemas basados en software son mucho más flexibles (permitiendo, por ejemplo, construir RAIDs de particiones en lugar de discos completos y agrupar en un mismo RAID discos conectados en varias controladoras) y los basados en hardware añaden un punto de fallo más al sistema (la controladora RAID).

Todas las implementaciones pueden soportar el uso de uno o más discos de reserva (hot spare), unidades preinstaladas que pueden usarse inmediatamente (y casi siempre automáticamente) tras el fallo de un disco del RAID. Esto reduce el tiempo del período de reparación al acortar el tiempo de reconstrucción del RAID.

Niveles RAID estándar

Los niveles RAID más comúnmente usados son:

- RAID 0: Conjunto dividido
- RAID 1: Conjunto en espejo
- RAID 5: Conjunto dividido con paridad distribuida

RAID 0 (Data Striping)

Un RAID 0 (también llamado conjunto dividido o volumen dividido) segun SC, distribuye los datos equitativamente entre dos o más discos sin información de paridad que proporcione redundancia. Es importante señalar que el RAID 0 no era uno de los niveles RAID originales y que no es redundante. El RAID 0 se usa normalmente para incrementar el rendimiento, aunque también puede utilizarse como forma de crear un pequeño número de grandes discos virtuales a partir de un gran número de pequeños discos físicos. Un RAID 0 puede ser creado con discos de diferentes tamaños, pero el espacio de almacenamiento añadido al conjunto estará limitado al tamaño del disco más pequeño (por ejemplo, si un disco de 300 GB se divide con uno de 100 GB, el tamaño del conjunto resultante será 200 GB). Una buena implementación de un RAID 0 dividirá las operaciones de lectura y escritura en bloques de igual tamaño y los distribuirá equitativamente entre los dos discos. También es posible crear un RAID 0 con más de un disco, si bien la fiabilidad del conjunto será igual a la fiabilidad media de cada disco entre el número de discos del conjunto; es decir, la fiabilidad total —medida como MTTF o MTBF— es (aproximadamente) inversamente proporcional al número de discos del conjunto. Esto se debe a que el sistema de ficheros se distribuye entre todos los discos sin redundancia, por lo que cuando uno de ellos falla se pierde una parte muy importante de los datos.

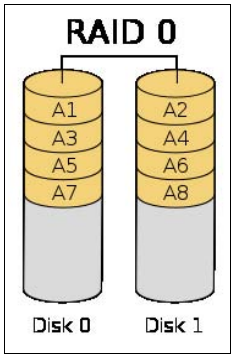


Ilustración 6: Diagrama de una configuración RAID 0

Con un RAID 0, si todos los sectores accedidos están en el mismo disco, entonces el tiempo de búsqueda será el de dicho disco. Si los sectores a acceder están distribuidos equitativamente entre los discos, entonces el tiempo de búsqueda aparente estará entre el más rápido y el más lento de los discos del conjunto, pues todos los discos necesitan acceder a su parte de los datos antes de que la operación pueda completarse. Esto podría llevar a tiempos de búsqueda cercanos al peor escenario para un único disco, salvo si los discos giran sincronizadamente, lo que daría tiempos de búsqueda sólo ligeramente superiores al de un único disco. La velocidad de transferencia del conjunto será la suma de la de todos los discos, limitada sólo por la velocidad de la controladora RAID.

El RAID 0 es útil para configuraciones tales como servidores NFS de solo lectura en las que montar muchos discos es un proceso costoso en tiempo y la redundancia es irrelevante. Otro uso es cuando el número de discos está limitado por el sistema operativo: por ejemplo, en Microsoft Windows el número de unidades lógicas (letras) está limitado a 24, por lo que el RAID 0 es una forma de usar más discos (en Windows 2000 Professional y posteriores es posible montar particiones en directorios, de forma parecida a Unix, eliminando así la necesidad de asignar una letra a cada unidad). El RAID 0 es también una opción popular para sistemas destinados a juegos en los que se desea un buen rendimiento y la integridad no es muy importante, si bien el costo es una preocupación para la mayoría de los usuarios.

JBOD

Aunque la concatenación de discos (también llamada JBOD, de Just a Bunch Of Drives, 'Sólo un Montón de Discos') no es uno de los niveles RAID numerados, sí es un método popular de combinar múltiples discos duros físicos en un solo disco virtual. Como su nombre indica, los discos son meramente concatenados entre sí, de forma que se comporten como un único disco.

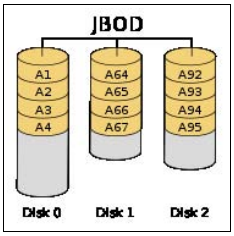


Ilustración 7: Diagrama de una configuración JBOD

En este sentido, la concatenación es como el proceso contrario al particionado: mientras éste toma un disco físico y crea dos o más unidades lógicas, JBOD usa dos o más discos físicos para crear una unidad lógica.

Al consistir en un conjunto de discos independientes (sin redundancia), puede ser visto como un primo lejano del RAID 0. JBOD es usado a veces para combinar varias unidades pequeñas (obsoletas) en una unidad mayor con un tamaño útil.

JBOD es parecido al ampliamente usado gestor de volúmenes lógicos LVM y LSM en los sistemas Unix. JBOD es útil para sistemas que no soportan LVM/LSM (como Microsoft Windows, si bien Windows 2003 Server, Windows XP Pro y Windows 2000 soportan JBOD vía software, llamado spanning de discos dinámicos). La diferencia entre JBOD y LVM/LSM es que la traducción de la dirección lógica del dispositivo concatenado a la dirección física del disco es realizada por el hardware RAID en el primer caso y por el núcleo en el segundo.

Una ventaja de JBOD sobre RAID 0 es que, en caso de fallo de un disco, en RAID 0 suele producirse la pérdida de todos los datos del conjunto, mientras en JBOD sólo se pierden los datos del disco afectado, conservándose los de los restantes discos. Sin embargo, JBOD no supone ninguna mejora de rendimiento.

RAID 1 (Data Mirroring)

Un RAID 1 crea una copia exacta (o espejo) de un conjunto de datos en dos o más discos. Esto resulta útil cuando el rendimiento en lectura es más importante que la capacidad. Un conjunto RAID 1 sólo puede ser tan grande como el más pequeño de sus discos. Un RAID 1 clásico consiste en dos discos en espejo, lo que incrementa exponencialmente la fiabilidad respecto a un solo disco; es decir, la probabilidad de fallo del conjunto es igual al producto de las probabilidades de fallo de cada uno de los discos (pues para que el conjunto falle es necesario que lo hagan todos sus discos).

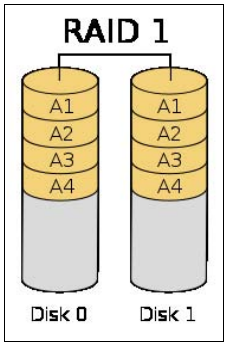


Ilustración 8: Diagrama de una configuración RAID 1

Adicionalmente, dado que todos los datos están en dos o más discos, con hardware habitualmente independiente, el rendimiento de lectura se incrementa aproximadamente como múltiplo linear del número del copias; es decir, un RAID 1 puede estar leyendo simultáneamente dos datos diferentes en dos discos diferentes, por lo que su rendimiento se duplica. Para maximizar los beneficios sobre el rendimiento del RAID 1 se recomienda el uso de controladoras de disco independientes, una para cada disco (práctica que algunos denominan splitting o duplexing).

Como en el RAID 0, el tiempo medio de lectura se reduce, ya que los sectores a buscar pueden dividirse entre los discos, bajando el tiempo de búsqueda y subiendo la tasa de transferencia, con el único límite de la velocidad soportada por la controladora RAID. Sin embargo, muchas tarjetas RAID 1 IDE antiguas leen sólo de un disco de la pareja, por lo que su rendimiento es igual al de un único disco. Algunas implementaciones RAID 1 antiguas

también leen de ambos discos simultáneamente y comparan los datos para detectar errores. La detección y corrección de errores en los discos duros modernos hacen esta práctica poco útil.

Al escribir, el conjunto se comporta como un único disco, dado que los datos deben ser escritos en todos los discos del RAID 1. Por tanto, el rendimiento no mejora.

El RAID 1 tiene muchas ventajas de administración. Por ejemplo, en algunos entornos 24/7, es posible «dividir el espejo»: marcar un disco como inactivo, hacer una copia de seguridad de dicho disco y luego «reconstruir» el espejo. Esto requiere que la aplicación de gestión del conjunto soporte la recuperación de los datos del disco en el momento de la división. Este procedimiento es menos crítico que la presencia de una característica de snapshot en algunos sistemas de ficheros, en la que se reserva algún espacio para los cambios, presentando una vista estática en un punto temporal dado del sistema de ficheros. Alternativamente, un conjunto de discos puede ser almacenado de forma parecida a como se hace con las tradicionales cintas.

RAID 2

Un RAID 2 divide los datos a nivel de bits en lugar de a nivel de bloques y usa un código de Hamming para la corrección de errores. Los discos son sincronizados por la controladora para funcionar al unísono. Éste es el único nivel RAID original que actualmente no se usa. Permite tasas de trasferencias extremadamente altas.

Teóricamente, un RAID 2 necesitaría 39 discos en un sistema informático moderno: 32 se usarían para almacenar los bits individuales que forman cada palabra y 7 se usarían para la corrección de errores.

RAID 3

Un RAID 3 usa división a nivel de bytes con un disco de paridad dedicado. El RAID 3 se usa rara vez en la práctica. Uno de sus efectos secundarios es que normalmente no puede atender varias peticiones simultáneas, debido a que por definición cualquier simple bloque de datos se dividirá por todos los miembros del conjunto, residiendo la misma dirección dentro de cada uno de ellos. Así, cualquier operación de lectura o escritura exige activar todos los discos del conjunto.

En el ejemplo del gráfico, una petición del bloque «A» formado por los bytes A1 a A6 requeriría que los tres discos de datos buscaran el comienzo (A1) y devolvieran su contenido. Una petición simultánea del bloque «B» tendría que esperar a que la anterior concluyese.

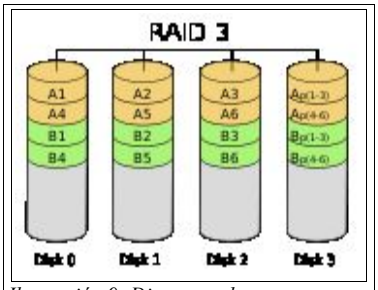


Ilustración 9: Diagrama de una configuración RAID 3. Cada número representa un byte de datos; cada columna, un disco.

RAID 4

Un RAID 4 usa división a nivel de bloques con un disco de paridad dedicado. Necesita un mínimo de 3 discos físicos. El RAID 4 es parecido al RAID 3 excepto porque divide a nivel de bloques en lugar de a nivel de bytes. Esto permite que cada miembro del conjunto funcione independientemente cuando se solicita un único bloque. Si la controladora de disco lo permite, un conjunto RAID 4 puede servir varias peticiones de lectura simultáneamente. En principio también sería posible servir varias peticiones de escritura simultáneamente, pero al estar toda la información de paridad en un solo disco, éste se convertiría en el cuello de botella del conjunto.

En el gráfico de ejemplo anterior, una petición del bloque «A1» sería servida por el disco 1. Una petición simultánea del bloque «B1» tendría que esperar, pero una petición de «B2» podría atenderse concurrentemente.

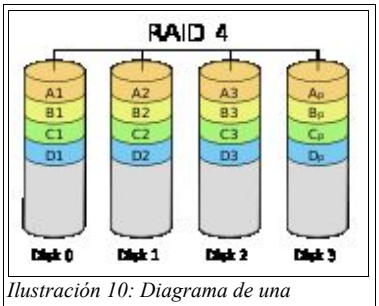


Ilustración 10: Diagrama de una configuración RAID 4. Cada número representa un bloque de datos; cada columna, un disco.

RAID 5

Un RAID 5 usa división de datos a nivel de bloques distribuyendo la información de paridad entre todos los discos miembros del conjunto. El RAID 5 ha logrado popularidad gracias a su bajo coste de redundancia. Generalmente, el RAID 5 se implementa con soporte hardware para el cálculo de la paridad.

En el gráfico de ejemplo anterior, una petición de lectura del bloque «A1» sería servida por el disco 0. Una petición de lectura simultánea del bloque «B1» tendría que esperar, pero una petición de lectura de «B2» podría atenderse concurrentemente ya que seria servida por el disco 1.

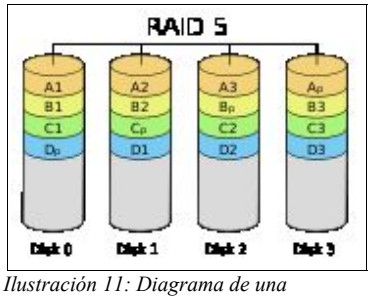


Ilustración 11: Diagrama de una configuración RAID 5

Cada vez que un bloque de datos se escribe en un RAID 5, se genera un bloque de paridad dentro de la misma división (stripe). Un bloque se compone a menudo de muchos sectores consecutivos de disco. Una serie de bloques (un bloque de cada uno de los discos del conjunto) recibe el nombre colectivo de división (stripe). Si otro bloque, o alguna porción de un bloque, es escrita en esa misma división, el bloque de paridad (o una parte del mismo) es recalculada y vuelta a escribir. El disco utilizado por el bloque de paridad está escalonado de una división a la siguiente, de ahí el término «bloques de paridad distribuidos». Las escrituras en un RAID 5 son costosas en términos de operaciones de disco y tráfico entre los discos y la controladora.

Los bloques de paridad no se leen en las operaciones de lectura de datos, ya que esto sería una sobrecarga innecesaria y disminuiría el rendimiento. Sin embargo, los bloques de paridad se leen cuando la lectura de un sector de datos provoca un error de CRC. En este caso, el sector en la misma posición relativa dentro de cada uno de los bloques de datos restantes en la división y dentro del bloque de paridad en la división se utilizan para reconstruir el sector erróneo. El error CRC se oculta así al resto del sistema. De la misma forma, si falla un disco del conjunto, los bloques de paridad de los restantes discos son combinados matemáticamente con los bloques de datos de los restantes discos para reconstruir los datos del disco que ha fallado «al vuelo».

Lo anterior se denomina a veces Modo Interino de Recuperación de Datos (Interim Data Recovery Mode). El sistema sabe que un disco ha fallado, pero sólo con el fin de que el sistema operativo pueda notificar al administrador que una unidad necesita ser reemplazada: las aplicaciones en ejecución siguen funcionando ajenas al fallo. Las lecturas y escrituras continúan normalmente en el conjunto de discos, aunque con alguna degradación de rendimiento. La diferencia entre el RAID 4 y el RAID 5 es que, en el Modo Interno de Recuperación de Datos, el RAID 5 puede ser ligeramente más rápido, debido a que, cuando el CRC y la paridad están en el disco que falló, los cálculos no tienen que realizarse, mientras que en el RAID 4, si uno de los discos de datos falla, los cálculos tienen que ser realizados en cada acceso.

El RAID 5 requiere al menos tres unidades de disco para ser implementado. El fallo de un segundo disco provoca la pérdida completa de los datos.

El número máximo de discos en un grupo de redundancia RAID 5 es teóricamente ilimitado, pero en la práctica es común limitar el número de unidades. Los inconvenientes de usar grupos de redundancia mayores son una mayor probabilidad de fallo simultáneo de dos discos, un mayor tiempo de reconstrucción y una mayor probabilidad de hallar un sector irrecuperable durante una reconstrucción. A medida que el número de discos en un conjunto RAID 5 crece, el MTBF (tiempo medio entre fallos) puede ser más bajo que el de un único disco. Esto sucede cuando la probabilidad de que falle un segundo disco en los N-1 discos restantes de un conjunto en el que ha fallado un disco en el tiempo necesario para detectar, reemplazar y recrear dicho disco es mayor que la probabilidad de fallo de un único disco. Una alternativa que proporciona una protección de paridad dual, permitiendo así mayor número de discos por grupo, es el RAID 6.

Algunos vendedores RAID evitan montar discos de los mismos lotes en un grupo de redundancia para minimizar la probabilidad de fallos simultáneos al principio y el final de su vida útil.

Las implementaciones RAID 5 presentan un rendimiento malo cuando se someten a cargas de trabajo que incluyen muchas escrituras más pequeñas que el tamaño de una división (stripe). Esto se debe a que la paridad debe ser actualizada para cada escritura, lo que exige realizar secuencias de lectura, modificación y escritura tanto para el bloque de datos como para el de paridad. Implementaciones más complejas incluyen a menudo cachés de escritura no volátiles para reducir este problema de rendimiento.

En el caso de un fallo del sistema cuando hay escrituras activas, la paridad de una división (stripe) puede quedar en un estado inconsistente con los datos. Si esto no se detecta y repara antes de que un disco o bloque falle, pueden perderse datos debido a que se usará una paridad incorrecta para reconstruir el bloque perdido en dicha división. Esta potencial vulnerabilidad se conoce a veces como «agujero de escritura». Son comunes el uso de caché no volátiles y otras técnicas para reducir la probabilidad de ocurrencia de esta vulnerabilidad.

RAID 6

Un RAID 6 amplía el nivel RAID 5 añadiendo otro bloque de paridad, por lo que divide los datos a nivel de bloques y distribuye los dos bloques de paridad entre todos los miembros del conjunto. El RAID 6 no era uno de los niveles RAID originales.

El RAID 6 puede ser considerado un caso especial de código Reed-Solomon.[1] El RAID 6, siendo un caso degenerado, exige sólo sumas en el campo de Galois.

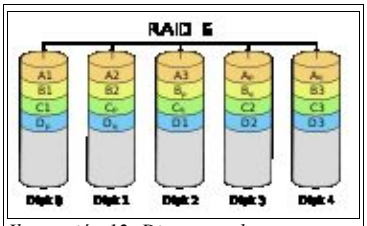


Ilustración 12: Diagrama de una configuración RAID 6. Cada número representa un bloque de datos; cada columna, un disco; p y q, códigos Reed-Solomon.

Dado que se está operando sobre bits, lo que se usa es un campo binario de Galois (). En las representaciones cíclicas de los campos binarios de Galois, la suma se calcula con un simple XOR.

Tras comprender el RAID 6 como caso especial de un código Reed-Solomon, se puede ver que es posible ampliar este enfoque para generar redundancia simplemente produciendo otro código, típicamente un polinomio en (m = 8 significa que estamos operando sobre bytes). Al añadir códigos adicionales es posible alcanzar cualquier número de discos redundantes, y recuperarse de un fallo de ese mismo número de discos en cualesquiera puntos del conjunto, pero en el nivel RAID 6 se usan dos únicos códigos.

Al igual que en el RAID 5, en el RAID 6 la paridad se distribuye en divisiones (stripes), con los bloques de paridad en un lugar diferente en cada división.

El RAID 6 es ineficiente cuando se usa un pequeño número de discos pero a medida que el conjunto crece y se dispone de más discos la pérdida en capacidad de almacenamiento se hace menos importante, creciendo al mismo tiempo la probabilidad de que dos discos fallen simultáneamente. El RAID 6 proporciona protección contra fallos dobles de discos y contra fallos cuando se está reconstruyendo un disco. En caso de que sólo tengamos un conjunto puede ser más adecuado que usar un RAID 5 con un disco de reserva (hot spare).

La capacidad de datos de un conjunto RAID 6 es n-2, siendo n el número total de discos del conjunto.

Un RAID 6 no penaliza el rendimiento de las operaciones de lectura, pero sí el de las de escritura debido al proceso que exigen los cálculos adicionales de paridad. Esta penalización puede minimizarse agrupando las escrituras en el menos número posible de divisiones (stripes), lo que puede lograrse mediante el uso de un sistema de ficheros WAFL.

RAID 5E y RAID 6E

Se suele llamar RAID 5E y RAID 6E a las variantes de RAID 5 y RAID 6 que incluyen discos de reserva. Estos discos pueden estar conectados y preparados (hot spare) o en espera (standby spare). En los RAID 5E y RAID 6E, los discos de reserva están disponibles para cualquiera de las unidades miembro. No suponen mejora alguna del rendimiento, pero sí se minimiza el tiempo de reconstrucción (en el caso de los discos hot spare) y las labores de administración cuando se producen fallos. Un disco de reserva no es realmente parte del conjunto hasta que un disco falla y el conjunto se reconstruye sobre el de reserva.

Niveles RAID anidados

Muchas controladoras permiten anidar niveles RAID, es decir, que un RAID pueda usarse como elemento básico de otro en lugar de discos físicos. Resulta instructivo pensar en estos conjuntos como capas dispuestas unas sobre otras, con los discos físicos en la inferior.

Los RAIDs anidados se indican normalmente uniendo en un solo número los correspondientes a los niveles RAID usados, añadiendo a veces un «+» entre ellos. Por ejemplo, el RAID 10 (o RAID 1+0) consiste conceptualmente en múltiples conjuntos de nivel 1 almacenados en discos físicos con un nivel 0 encima, agrupando los anteriores niveles 1. En el caso del RAID 0+1 se usa más esta forma que RAID 01 para evitar la confusión con el RAID 1. Sin embargo, cuando el conjunto de más alto nivel es un RAID 0 (como en el RAID 10 y en el RAID 50), la mayoría de los vendedores eligen omitir el «+», a pesar de que RAID 5+0 sea más informativo.

Al anidar niveles RAID, se suele combinar un nivel RAID que proporcione redundancia con un RAID 0 que aumenta el rendimiento. Con estas configuraciones es preferible tener el RAID 0 como nivel más alto y los conjuntos redundantes debajo, porque así será necesario reconstruir menos discos cuando uno falle. (Así, el RAID 10 es preferible al RAID 0+1 aunque las ventajas administrativas de «dividir el espejo» del RAID 1 se perderían.)

Los niveles RAID anidados más comúnmente usados son:

- RAID 0+1: Un espejo de divisiones
- RAID 1+0: Una división de espejos
- RAID 30: Una división de niveles RAID con paridad dedicada
- RAID 100: Una división de una división de espejos

Posibilidades de RAID

Lo que RAID puede hacer

- **RAID puede mejorar el uptime.**

Los niveles RAID 1, 0+1 o 10, 5 y 6 (sus variantes, como el 50) permiten que un disco falle mecánicamente y que aun así los datos del conjunto sigan siendo accesibles para los usuarios. En lugar de exigir que se realice una restauración costosa en tiempo desde una cinta, DVD o algún otro medio de respaldo lento, un RAID permite que los datos se recuperen en un disco de reemplazo a partir de los restantes discos del conjunto, mientras al mismo tiempo permanece disponible para los usuarios en un modo degradado. Esto es muy valorado por las empresas, ya que el tiempo de no disponibilidad suele tener graves repercusiones. Para usuarios domésticos, puede permitir el ahorro del tiempo de restauración de volúmenes grandes, que requerirían varios DVD o cintas para las copias de seguridad.

- **RAID puede mejorar el rendimiento de ciertas aplicaciones.**

Los niveles RAID 0, 5 y 6 usan variantes de división (striping) de datos, lo que permite que varios discos atiendan simultáneamente las operaciones de lectura lineales, aumentando la tasa de transferencia sostenida. Las aplicaciones de escritorio que trabajan con ficheros grandes, como la edición de vídeo e imágenes, se benefician de esta mejora. También es útil para las operaciones de copia de respaldo de disco a disco. Además, si se usa un RAID 1 o un RAID basado en división con un tamaño de bloque lo suficientemente grande se logran mejoras de rendimiento para patrones de acceso que implique múltiples lecturas simultáneas (por ejemplo, bases de datos multiusuario).

Lo que RAID NO puede hacer

- **RAID no protege los datos.**

Un conjunto RAID tiene un sistema de ficheros, lo que supone un punto único de fallo al ser vulnerable a una amplia variedad de riesgos aparte del fallo físico de disco, por lo que RAID no evita la pérdida de datos por estas causas. RAID no impedirá que un virus destruya los datos, que éstos se corrompan, que sufran la modificación o borrado accidental por parte del usuario ni que un fallo físico en otro componente del sistema afecten a los datos. RAID tampoco supone protección alguna frente a desastres naturales o provocados por el hombre como incendios o inundaciones. Para proteger los datos, deben realizarse copias de seguridad en medios tales como DVD, cintas o discos duros externos, y almacenarlas en lugares geográficos distantes.

- **RAID no simplifica la recuperación de un desastre.**

Cuando se trabaja con un solo disco, éste es accesible normalmente mediante un controlador ATA o SCSI incluido en la mayoría de los sistemas operativos. Sin embargo, las controladoras RAID necesitan controladores software específicos. Las herramientas de recuperación que trabajan con discos simples en controladoras genéricas necesitarán controladores especiales para acceder a los datos de los conjuntos RAID. Si estas herramientas no los soportan, los datos serán inaccesibles para ellas.

- **RAID no mejora el rendimiento de las aplicaciones.**

Esto resulta especialmente cierto en las configuraciones típicas de escritorio. La mayoría de aplicaciones de escritorio y videojuegos hacen énfasis en la estrategia de buffering y los tiempos de búsqueda de los discos. Una mayor tasa de transferencia sostenida supone poco beneficio para los usuarios de estas aplicaciones, al ser la mayoría de los ficheros a los que se accede muy pequeños. La división de discos de un RAID 0 mejora el rendimiento de transferencia lineal pero no lo demás, lo que hace que la mayoría de las aplicaciones de escritorio y juegos no muestren mejora alguna, salvo excepciones. Para estos usos, lo mejor es comprar un disco más grande, rápido y caro en lugar de dos discos más lentos y pequeños en una configuración RAID 0.

- **RAID no facilita el traslado a un sistema nuevo.**

Cuando se usa un solo disco, es relativamente fácil trasladar el disco a un sistema nuevo: basta con conectarlo, si cuenta con la misma interfaz. Con un RAID no es tan sencillo: la BIOS RAID debe ser capaz de leer los metadatos de los miembros del conjunto para reconocerlo adecuadamente y hacerlo disponible al sistema operativo. Dado que los distintos fabricantes de controladoras RAID usan diferentes formatos de metadatos (incluso controladoras de un mismo fabricante son incompatibles si corresponden a series diferentes) es virtualmente imposible mover un conjunto RAID a una controladora diferente, por lo que suele ser necesario mover también la controladora. Esto resulta imposible en aquellos sistemas donde está integrada en la placa base. Esta limitación puede obviarse con el uso de RAIDs por software, que a su vez añaden otras diferentes (especialmente relacionadas con el rendimiento).

Bus

Un Bus es un camino de comunicación entre dos o más dispositivos. Una característica clave de un bus es que se trata de un medio de transmisión compartido. Al bus se conectan varios dispositivo, y cualquier señal transmitida por uno de esos dispositivos está disponible para que los otros dispositivos conectados al bus puedan acceder a ella. Si dos dispositivos transmiten durante el mismo periodo de tiempo, sus señales pueden solaparse y distorsionarse. Consiguientemente, solo un dispositivo puede transmitir con éxito en un momento dado.

Usualmente, un bus está constituido por varios caminos de comunicación, o líneas. Cada línea es capaz de transmitir señales binarias representadas por 1 y por 0. En un intervalo de tiempo, se puede transmitir una secuencia de dígitos binarios a través de una única línea.

Tipo de Bus	Características
De Datos	A este nivel no existe la diferencia entre “datos” e “instrucciones”. El ancho del bus es un factor clave a la hora de determinar sus prestaciones.
De Dirección	Designa la fuente o destino del dato. El ancho del bus de direcciones determina la máxima capacidad de memoria posible en el sistema.
De Control	Brinda información sobre señales de control y temporización (Señal de escritura/lectura en memoria, petición de interrupción, señales de reloj, etc).

Jerarquía de Buses

Si se conecta un gran número de dispositivos las prestaciones pueden disminuir por dos causas:

- Retardo de propagación provocados por una gran cantidad de dispositivos conectados en el bus.
- Imposibilidad del bus de manejar una gran cantidad de peticiones de transferencias acumuladas.

Por consiguiente, la mayoría de los computadores utilizan varios buses, normalmente organizados jerarquicamente. La arquitectura de buses tradicional es razonablemente eficiente, pero muestra su debilidad a medida que los dispositivos de E/S ofrecen prestaciones cada vez mayores. La respuesta común de esta situación, por parte de la industria, ha sido proponer un bus de alta velocidad, que está estrechamente integrado con el resto del sistema, y requiere solo un adaptador entre el bus del procesador y el bus de alta velocidad. En algunas ocasiones, esta disposición es conocida como arquitectura de entreplana.

Elementos de diseño de un bus

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

Elemento	Subtipo	Descripción	Ventaja	Desventaja
Tipo	Dedicado	Se refiere al uso de múltiples buses, cada uno de los cuales conecta sólo a un subconjunto de módulos. Ejemplo: el uso de líneas separadas para direcciones y para datos.	Tiene un elevado rendimiento (hay menos disputas por el acceso al bus).	Incremento en el tamaño y costo del sistema.
	Multiplexado	Se refiere al uso de la menor cantidad de buses posibles, cada uno conectado al mayor subconjunto de módulos posible. Ejemplo: el uso de las mismas líneas para direcciones y datos.	Uso de menos líneas,	Circuitería compleja en cada módulo. Posible reducción de las prestaciones (los eventos que deben compartir las mismas líneas no pueden producirse en paralelo)
Método de arbitraje	Centralizado	Un único dispositivo de hardware, denominado controlador del bus, es responsable de asignar tiempos en el bus.		
	Distribuido	No existe controlador central. Cada		

Elemento	Subtipo	Descripción	Ventaja	Desventaja
		módulo dispone de lógica para controlar el acceso, y los módulos actúan conjuntamente para compartir el bus.		
Temporización	Síncrono	La presencia de un evento en el bus está determinada por un relog.	Fácil de implementar y comprobar	Es menos flexible que la temporización asíncrona.
	Asíncrono	La presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo. Ej: El procesador situá las señales de dirección y lectura en el bus. Después de un breve intervalo para que las señales se estabilicen, activa la señal MSYN, indicando la presencia de señales de dirección y control válidas. El módulo de memoria responde proporcionando el dato y una señal SSYN	Es mas flexible a la hora de utilizar la velocidad de dispositivos lentos y rápidos	Es mas complejo de controlar
Anchura del bus	De Datos	Mientras es mas ancho permite mandar mas bits paralelamente.		
	De Dirección	El ancho del bus de direcciones determina la máxima capacidad de memoria posible en el sistema.		
Transferencia de datos	Lectura			
	Escritura			
	Lectura Modificación Escritura			
	Lectura después de escritura			
	Bloque			

NOTA: VER CANALES DE E/S, SCSI, FIREWIRE Y USB PARA PONER EN EL APUNTE O NO