CAPÍTULO 1

Introducción a los computadores

Un sistema operativo explota los recursos hardware de uno o más procesadores para proporcionar un conjunto de servicios a los usuarios del sistema. El sistema operativo también gestiona la memoria secundaria y los dispositivos de E/S (entrada/salida) para sus usuarios. Por tanto, es importante tener algunos conocimientos del hardware del computador subyacente antes de iniciar el estudio de los sistemas operativos.

Este capítulo proporciona una visión general del hardware del computador. En la mayoría de las áreas, el estudio es breve, asumiendo que el lector ya está familiarizado con este tema. Sin embargo, se estudiarán con cierto detalle varios aspectos por su repercusión en los temas tratados más adelante en el libro.

1.1. ELEMENTOS BÁSICOS

Al más alto nivel, un computador consta del procesador, la memoria y los componentes de E/S, incluyendo uno o más módulos de cada tipo. Estos componentes se interconectan de manera que se pueda lograr la función principal del computador, que es ejecutar programas. Por tanto, hay cuatro elementos estructurales principales:

- **Procesador.** Controla el funcionamiento del computador y realiza sus funciones de procesamiento de datos. Cuando sólo hay un procesador, se denomina usualmente **unidad central de proceso** (*Central Processing Unit*, CPU).
- **Memoria principal.** Almacena datos y programas. Esta memoria es habitualmente volátil; es decir, cuando se apaga el computador, se pierde su contenido. En contraste, el contenido de la memoria del disco se mantiene incluso cuando se apaga el computador. A la memoria principal se le denomina también *memoria real* o *memoria primaria*.
- **Módulos de E/S.** Transfieren los datos entre el computador y su entorno externo. El entorno externo está formado por diversos dispositivos, incluyendo dispositivos de memoria secundaria (por ejemplo, discos), equipos de comunicaciones y terminales.
- **Bus del sistema.** Proporciona comunicación entre los procesadores, la memoria principal y los módulos de E/S.

La Figura 1.1 muestra estos componentes de más alto nivel. Una de las funciones del procesador es el intercambio de datos con la memoria. Para este fin, se utilizan normalmente dos registros internos (al procesador): un registro de dirección de memoria (RDIM), que especifica la dirección de memoria de la siguiente lectura o escritura; y un registro de datos de memoria (RDAM), que contiene los datos que se van a escribir en la memoria o que recibe los datos leídos de la memoria. De manera similar, un registro de dirección de E/S (RDIE/S) especifica un determinado dispositivo de E/S, y un registro de datos de E/S (RDAE/S) permite el intercambio de datos entre un módulo de E/S y el procesador.

Un módulo de memoria consta de un conjunto de posiciones definidas mediante direcciones numeradas secuencialmente. Cada posición contiene un patrón de bits que se puede interpretar como una instrucción o como datos. Un módulo de E/S transfiere datos desde los dispositivos externos hacia el procesador y la memoria, y viceversa. Contiene *buffers* (es decir, zonas de almacenamiento internas) que mantienen temporalmente los datos hasta que se puedan enviar.

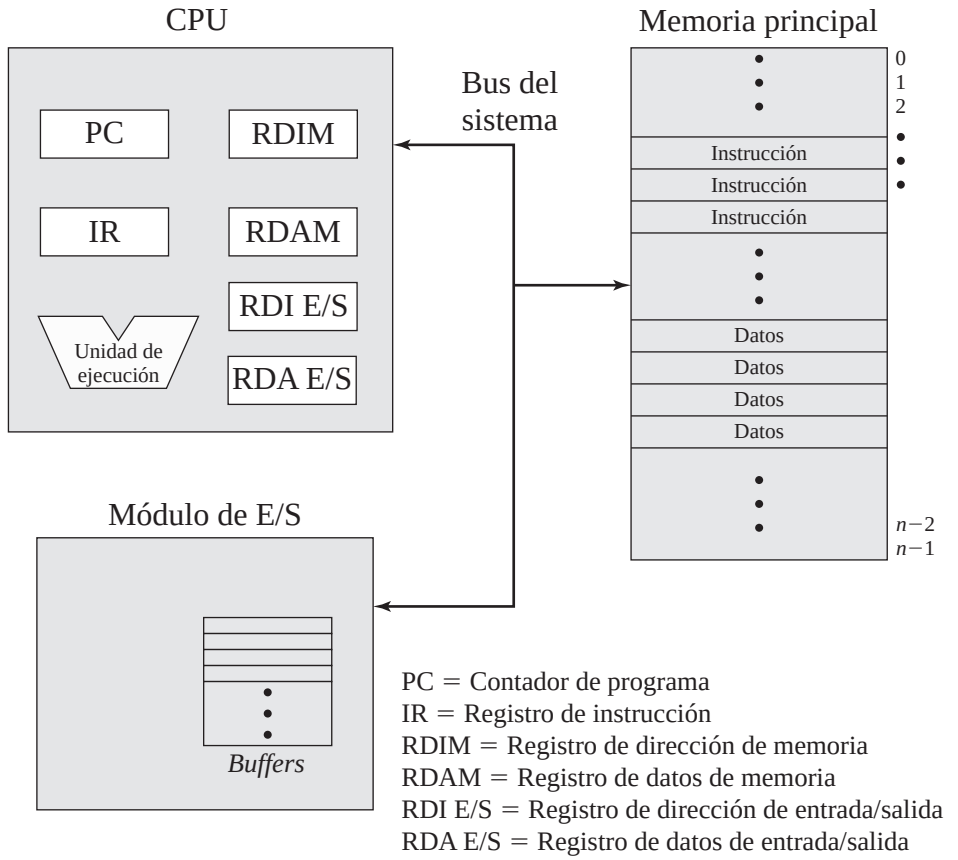


Figura 1.1. Componentes de un computador: visión al más alto nivel.

1.2. REGISTROS DEL PROCESADOR

Un procesador incluye un conjunto de registros que proporcionan un tipo de memoria que es más rápida y de menor capacidad que la memoria principal. Los registros del procesador sirven para dos funciones:

- **Registros visibles para el usuario.** Permiten al programador en lenguaje máquina o en ensamblador minimizar las referencias a memoria principal optimizando el uso de registros. Para lenguajes de alto nivel, un compilador que realice optimización intentará tomar decisiones inteligentes sobre qué variables se asignan a registros y cuáles a posiciones de memoria principal. Algunos lenguajes de alto nivel, tales como C, permiten al programador sugerir al compilador qué variables deberían almacenarse en registros.

- **Registros de control y estado.** Usados por el procesador para controlar su operación y por rutinas privilegiadas del sistema operativo para controlar la ejecución de programas.

No hay una clasificación nítida de los registros entre estas dos categorías. Por ejemplo, en algunas máquinas el contador de programa es visible para el usuario, pero en muchas otras no lo es. Sin embargo, para el estudio que se presenta a continuación, es conveniente utilizar estas categorías.

REGISTROS VISIBLES PARA EL USUARIO

A un registro visible para el usuario se puede acceder por medio del lenguaje de máquina ejecutado por el procesador que está generalmente disponible para todos los programas, incluyendo tanto programas de aplicación como programas de sistema. Los tipos de registros que están normalmente disponibles son registros de datos, de dirección y de códigos de condición.

El programador puede utilizar los **registros de datos** para diversas funciones. En algunos casos, son, en esencia, de propósito general y pueden usarse con cualquier instrucción de máquina que realice operaciones sobre datos. Sin embargo, frecuentemente, hay restricciones. Por ejemplo, puede haber registros dedicados a operaciones de coma flotante y otros a operaciones con enteros.

Los **registros de dirección** contienen direcciones de memoria principal de datos e instrucciones, o una parte de la dirección que se utiliza en el cálculo de la dirección efectiva o completa. Estos registros pueden ser en sí mismos de propósito general, o pueden estar dedicados a una forma, o modo, particular de direccionamiento de memoria. A continuación, se incluyen algunos ejemplos:

- **Registro índice.** El direccionamiento indexado es un modo común de direccionamiento que implica sumar un índice a un valor de base para obtener una dirección efectiva.
- **Puntero de segmento.** Con direccionamiento segmentado, la memoria se divide en segmentos, que son bloques de palabras[1] de longitud variable. Una referencia de memoria consta de una referencia a un determinado segmento y un desplazamiento dentro del segmento; este modo de direccionamiento es importante en el estudio de la gestión de memoria que se realizará en el Capítulo 7. En este modo de direccionamiento, se utiliza un registro para mantener la dirección base (posición de inicio) del segmento. Puede haber múltiples registros; por ejemplo, uno para el sistema operativo (es decir, cuando el código del sistema operativo se está ejecutando en el procesador) y otro para la aplicación que se está ejecutando actualmente.
- **Puntero de pila.** Si hay direccionamiento de pila[2] visible para el usuario, hay un registro dedicado que apunta a la cima de la pila. Esto permite el uso de instrucciones que no contienen campo de dirección, tales como las que permiten apilar (*push*) y extraer (*pop*).

En algunas máquinas, una llamada a una subrutina o a un procedimiento implica salvar automáticamente todos los registros visibles para el usuario, que se restaurarán al retornar. El procesador realiza estas operaciones de salvar y restaurar como parte de la ejecución de las instrucciones de llamada y de retorno. Esto permite que cada procedimiento use estos registros independientemente. En otras máquinas, el programador es el responsable de guardar el contenido de los registros visibles para el usuario antes de una llamada a un procedimiento, incluyendo instrucciones para ello en el programa. Por tanto, las funciones de salvar y restaurar se pueden realizar en hardware o en software, dependiendo del procesador.

---

[1] No hay una definición universal del término *palabra*. En general, una **palabra** es un conjunto ordenado de bytes o bits que es la unidad normal con la que se almacena, transmite, u opera la información dentro de un determinado computador. Normalmente, si un computador tiene un juego de instrucciones de longitud fija, la longitud de las instrucciones es igual a la de la palabra.

[2] Una pila se almacena en la memoria principal y es un conjunto secuencial de posiciones a las que se hace referencia de manera similar a como ocurre con una pila física de papeles, insertando y extrayendo elementos de la cima de la misma. *Véase* el Apéndice 1B donde se incluye una explicación sobre la gestión de la pila.

REGISTROS DE CONTROL Y ESTADO

Se emplean varios registros del procesador para controlar el funcionamiento del mismo. En la mayoría de las máquinas, muchos de ellos no son visibles para el usuario. A algunos de ellos se puede acceder mediante instrucciones de máquina ejecutadas en lo que se denomina modo de control o de sistema operativo.

Por supuesto, diferentes máquinas tendrán distintas organizaciones de registros y utilizarán diferente terminología. A continuación, se proporcionará una lista razonablemente completa de tipos de registros, con una breve descripción de cada uno de ellos. Además de los registros RDIRM, RDAM, RDIE/S y RDAE/S mencionados anteriormente (Figura 1.1), los siguientes son esenciales para la ejecución de instrucciones:

- **Contador de programa (*Program Counter*, PC).** Contiene la dirección de la próxima instrucción que se leerá de la memoria.
- **Registro de instrucción (*Instruction Register*, IR).** Contiene la última instrucción leída.

Todos los diseños de procesador incluyen también un registro, o conjunto de registros, conocido usualmente como la palabra de estado del programa (*Program Status Word*, PSW), que contiene información de estado. La PSW contiene normalmente códigos de condición, además de otra información de estado, tales como un bit para habilitar/inhabilitar las interrupciones y un bit de modo usuario/supervisor.

Los **códigos de condición** (también llamados *indicadores*) son bits cuyo valor lo asigna normalmente el hardware de procesador teniendo en cuenta el resultado de las operaciones. Por ejemplo, una operación aritmética puede producir un resultado positivo, negativo, cero o desbordamiento. Además de almacenarse el resultado en sí mismo en un registro o en la memoria, se fija también un código de condición en concordancia con el resultado de la ejecución de la instrucción aritmética. Posteriormente, se puede comprobar el código de condición como parte de una operación de salto condicional. Los bits de código de condición se agrupan en uno o más registros. Normalmente, forman parte de un registro de control. Generalmente, las instrucciones de máquina permiten que estos bits se lean mediante una referencia implícita, pero no pueden ser alterados por una referencia explícita debido a que están destinados a la realimentación del resultado de la ejecución de una instrucción.

En máquinas que utilizan múltiples tipos de interrupciones, se puede proporcionar un conjunto de registros de interrupciones, con un puntero a cada rutina de tratamiento de interrupción. Si se utiliza una pila para implementar ciertas funciones (por ejemplo llamadas a procedimientos), se necesita un puntero de pila de sistema (*véase* el Apéndice 1B). El hardware de gestión de memoria, estudiado en el Capítulo 7, requiere registros dedicados. Asimismo, se pueden utilizar registros en el control de las operaciones de E/S.

En el diseño de la organización del registro de control y estado influyen varios factores. Un aspecto fundamental es proporcionar apoyo al sistema operativo. Ciertos tipos de información de control son útiles específicamente para el sistema operativo. Si el diseñador del procesador tiene un conocimiento funcional del sistema operativo que se va a utilizar, se puede diseñar la organización de registros de manera que se proporcione soporte por parte del hardware de características particulares de ese sistema operativo, en aspectos tales como la protección de memoria y la multiplexación entre programas de usuario.

Otra decisión de diseño fundamental es el reparto de la información de control entre los registros y la memoria. Es habitual dedicar las primeras (las de direcciones más bajas) cientos o miles de palabras de memoria para propósitos de control. El diseñador debe decidir cuánta información de control

debería estar en registros, más rápidos y más caros, y cuánta en la memoria principal, menos rápida y más económica.

1.3. EJECUCIÓN DE INSTRUCCIONES

Un programa que va a ejecutarse en un procesador consta de un conjunto de instrucciones almacenado en memoria. En su forma más simple, el procesamiento de una instrucción consta de dos pasos: el procesador lee (*busca*) instrucciones de la memoria, una cada vez, y ejecuta cada una de ellas. La ejecución del programa consiste en repetir el proceso de búsqueda y ejecución de instrucciones. La ejecución de la instrucción puede involucrar varias operaciones dependiendo de la naturaleza de la misma.

Se denomina *ciclo de instrucción* al procesamiento requerido por una única instrucción. En la Figura 1.2 se describe el ciclo de instrucción utilizando la descripción simplificada de dos pasos. A estos dos pasos se les denomina *fase de búsqueda* y de *ejecución*. La ejecución del programa se detiene sólo si se apaga la máquina, se produce algún tipo de error irrecuperable o se ejecuta una instrucción del programa que para el procesador.

BÚSQUEDA Y EJECUCIÓN DE UNA INSTRUCCIÓN

Al principio de cada ciclo de instrucción, el procesador lee una instrucción de la memoria. En un procesador típico, el contador del programa (PC) almacena la dirección de la siguiente instrucción que se va a leer. A menos que se le indique otra cosa, el procesador siempre incrementa el PC después de cada instrucción ejecutada, de manera que se leerá la siguiente instrucción en orden secuencial (es decir, la instrucción situada en la siguiente dirección de memoria más alta). Considere, por ejemplo, un computador simplificado en el que cada instrucción ocupa una palabra de memoria de 16 bits. Suponga que el contador del programa está situado en la posición 300. El procesador leerá la siguiente instrucción de la posición 300. En sucesivos ciclos de instrucción completados satisfactoriamente, se leerán instrucciones de las posiciones 301, 302, 303, y así sucesivamente. Esta secuencia se puede alterar, como se explicará posteriormente.

La instrucción leída se carga dentro de un registro del procesador conocido como registro de instrucción (IR). La instrucción contiene bits que especifican la acción que debe realizar el procesador. El procesador interpreta la instrucción y lleva a cabo la acción requerida. En general, estas acciones se dividen en cuatro categorías:

- **Procesador-memoria.** Se pueden transferir datos desde el procesador a la memoria o viceversa.
- **Procesador-E/S.** Se pueden enviar datos a un dispositivo periférico o recibirlos desde el mismo, transfiriéndolos entre el procesador y un módulo de E/S.

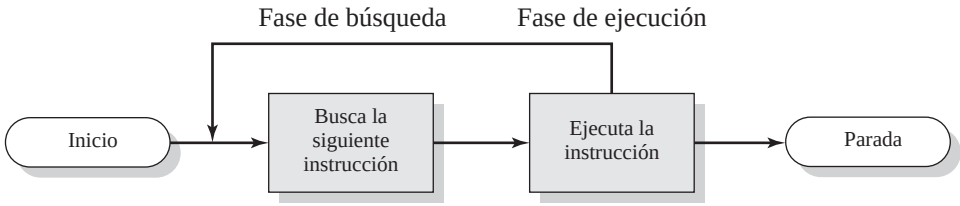


Figura 1.2. Ciclo de instrucción básico.

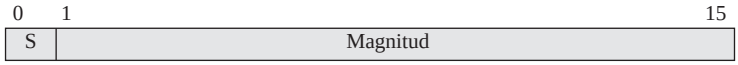
- **Procesamiento de datos.** El procesador puede realizar algunas operaciones aritméticas o lógicas sobre los datos.
- **Control.** Una instrucción puede especificar que se va a alterar la secuencia de ejecución. Por ejemplo, el procesador puede leer una instrucción de la posición 149, que especifica que la siguiente instrucción estará en la posición 182. El procesador almacenará en el contador del programa un valor de 182. Como consecuencia, en la siguiente fase de búsqueda, se leerá la instrucción de la posición 182 en vez de la 150.

Una ejecución de una instrucción puede involucrar una combinación de estas acciones.

Considere un ejemplo sencillo utilizando una máquina hipotética que incluye las características mostradas en la Figura 1.3. El procesador contiene un único registro de datos, llamado el acumulador (AC). Tanto las instrucciones como los datos tienen una longitud de 16 bits, estando la memoria organizada como una secuencia de palabras de 16 bits. El formato de la instrucción proporciona 4 bits para el código de operación, permitiendo hasta $2^4 = 16$ códigos de operación diferentes (representados por un único dígito hexadecimal[3]). Con los 12 bits restantes del formato de la instrucción, se pueden direccionar directamente hasta $2^{12} = 4.096$ (4K) palabras de memoria (denotadas por tres dígitos hexadecimales).

La Figura 1.4 ilustra una ejecución parcial de un programa, mostrando las partes relevantes de la memoria y de los registros del procesador. El fragmento de programa mostrado suma el contenido de la palabra de memoria en la dirección 940 al de la palabra de memoria en la dirección 941, almacenando el resultado en esta última posición. Se requieren tres instrucciones, que corresponden a tres fases de búsqueda y de ejecución, como se describe a continuación:

(a) Formato de instrucción



(b) Formato de un entero

Contador de programa (PC) = Dirección de la instrucción
Registro de instrucción (IR) = Instrucción que se está ejecutando
Acumulador (AC) = Almacenamiento temporal

(c) Registros internos de la CPU

0001 = Carga AC desde la memoria
0010 = Almacena AC en memoria
0101 = Suma a AC de la memoria

(d) Lista parcial de códigos-de-op

Figura 1.3. Características de una máquina hipotética.

[3] Puede encontrar un repaso básico de los sistema numéricos (decimal, binario y hexadecimal) en el *Computer Science Student Resource Site* en **WilliamStallings.com/StudentSupport.html**.

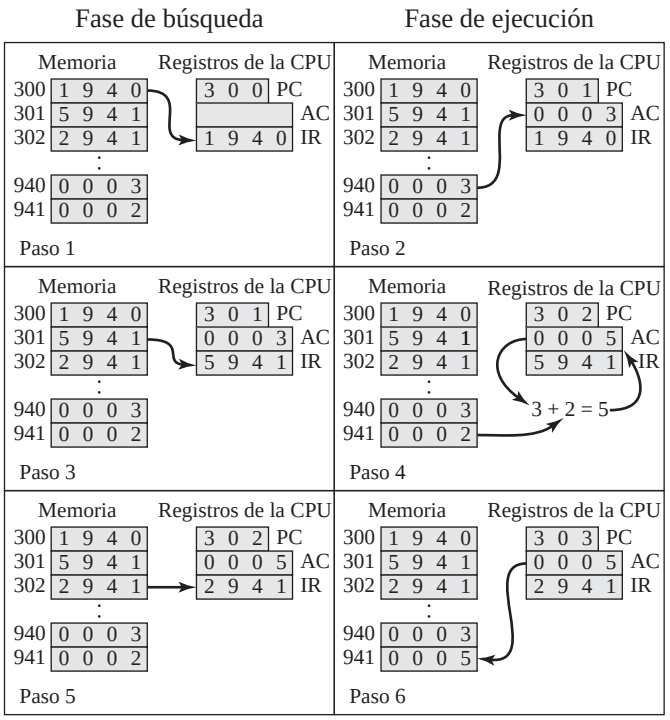


Figura 1.4. Ejemplo de ejecución de un programa (contenido de la memoria y los registros en hexadecimal).

1. El PC contiene el valor 300, la dirección de la primera instrucción. Esta instrucción (el valor 1940 en hexadecimal) se carga dentro del registro de instrucción IR y se incrementa el PC. Nótese que este proceso involucra el uso del registro de dirección de memoria (RDIM) y el registro de datos de memoria (RDAM). Para simplificar, no se muestran estos registros intermedios.
2. Los primeros 4 bits (primer dígito hexadecimal) en el IR indican que en el AC se va a cargar un valor leído de la memoria. Los restantes 12 bits (tres dígitos hexadecimales) especifican la dirección de memoria, que es 940.
3. Se lee la siguiente instrucción (5941) de la posición 301 y se incrementa el PC.
4. El contenido previo del AC y el contenido de la posición 941 se suman y el resultado se almacena en el AC.
5. Se lee la siguiente instrucción (2941) de la posición 302 y se incrementa el PC.
6. Se almacena el contenido del AC en la posición 941.

En este ejemplo, se necesitan tres ciclos de instrucción, de tal forma que cada uno consta de una fase de búsqueda y una fase de ejecución, para sumar el contenido de la posición 940 al contenido de la 941. Con un juego de instrucciones más complejo, se necesitarían menos ciclos de instrucción. La mayoría de los procesadores modernos incluyen instrucciones que contienen más de una dirección. Por tanto, la fase de ejecución de una determinada instrucción puede involucrar más de una referencia a memoria. Asimismo, en vez de referencias a memoria, una instrucción puede especificar una operación de E/S.

SISTEMA DE E/S

Se pueden intercambiar datos directamente entre un módulo de E/S (por ejemplo, un controlador de disco) y el procesador. Al igual que el procesador puede iniciar una lectura o una escritura en memoria, especificando la dirección de una posición de memoria, también puede leer o escribir datos en un módulo de E/S. En este caso, el procesador identifica un dispositivo específico que está controlado por un determinado módulo de E/S. Por tanto, podría producirse una secuencia de instrucciones similar a la de la Figura 1.4, con instrucciones de E/S en vez de instrucciones que hacen referencia a memoria.

En algunos casos, es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria para liberar al procesador de la tarea de E/S. En tales casos, el procesador concede a un módulo de E/S la autorización para leer o escribir de la memoria, de manera que la transferencia entre memoria y E/S puede llevarse a cabo sin implicar al procesador. Durante dicha transferencia, el módulo de E/S emite mandatos de lectura y escritura a la memoria, liberando al procesador de la responsabilidad del intercambio. Esta operación, conocida como acceso directo a memoria (*Direct Memory Access*, DMA) se examinará al final de este capítulo.

1.4. INTERRUPCIONES

Prácticamente todos los computadores proporcionan un mecanismo por el cual otros módulos (memoria y E/S) pueden interrumpir el secuenciamiento normal del procesador. La Tabla 1.1 detalla los tipos más comunes de interrupciones.

Básicamente, las interrupciones constituyen una manera de mejorar la utilización del procesador. Por ejemplo, la mayoría de los dispositivos de E/S son mucho más lentos que el procesador. Supóngase que el procesador está transfiriendo datos a una impresora utilizando el esquema de ciclo de instrucción de la Figura 1.2. Después de cada instrucción de escritura, el procesador debe parar y permanecer inactivo hasta que la impresora la lleve a cabo. La longitud de esta pausa puede ser del orden de muchos miles o incluso millones de ciclos de instrucción. Claramente, es un enorme desperdicio de la capacidad del procesador.

Tabla 1.1. Clases de interrupciones.

De programa	Generada por alguna condición que se produce como resultado de la ejecución de una instrucción, tales como un desbordamiento aritmético, una división por cero, un intento de ejecutar una instrucción de máquina ilegal, y las referencias fuera del espacio de la memoria permitido para un usuario.
Por temporizador	Generada por un temporizador del procesador. Permite al sistema operativo realizar ciertas funciones de forma regular.
De E/S	Generada por un controlador de E/S para señalar la conclusión normal de una operación o para indicar diversas condiciones de error.
Por fallo del hardware	Generada por un fallo, como un fallo en el suministro de energía o un error de paridad en la memoria.

Para dar un ejemplo concreto, considere un computador personal que operase a 1GHz, lo que le permitiría ejecutar aproximadamente 10^9 instrucciones por segundo[4]. Un típico disco duro tiene una

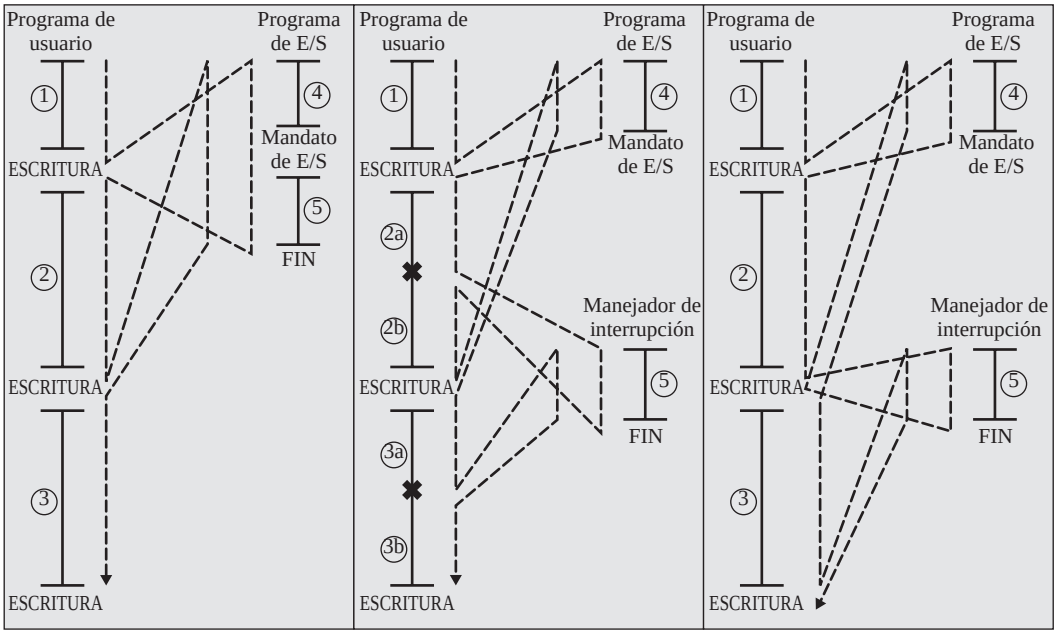
[4] Una discussion de los usos de prefijos numéricos, tales como giga y tera, está disponible en un documento de apoyo en el *Computer Science Student Resource Site* en **WilliamStallings.com/StudentSupport.html.**

velocidad de rotación de 7200 revoluciones por minuto, que corresponde con un tiempo de rotación de media pista de 4 ms., que es 4 millones de veces más lento que el procesador.

La Figura 1.5a muestra esta cuestión. El programa de usuario realiza una serie de llamadas de ESCRITURA intercaladas con el procesamiento. Los segmentos de código 1, 2 y 3 se refieren a secuencias de instrucciones que no involucran E/S. Las llamadas de ESCRITURA invocan a una rutina de E/S que es una utilidad del sistema que realizará la operación real de E/S. El programa de E/S consta de tres secciones:

- Una secuencia de instrucciones, etiquetada como 4 en la figura, para preparar la operación real de E/S. Esto puede incluir copiar los datos de salida en un *buffer* especial y preparar los parámetros de un mandato para el dispositivo.
- El mandato real de E/S. Sin el uso de interrupciones, una vez que se emite este mandato, el programa debe esperar a que el dispositivo de E/S realice la función solicitada (o comprobar periódicamente el estado, o muestrear, el dispositivo de E/S). El programa podría esperar simplemente realizando repetidamente una operación de comprobación para determinar si se ha realizado la operación de E/S.
- Una secuencia de instrucciones, etiquetada como 5 en la figura, para completar la operación. Esto puede incluir establecer un valor que indique el éxito o el fallo de la operación.

Debido a que la operación de E/S puede tardar un tiempo relativamente largo hasta que se completa, el programa de E/S se queda colgado esperando que se complete; por ello, el programa de usuario se detiene en el momento de la llamada de ESCRITURA durante un periodo de tiempo considerable.



(a) Sin interrupciones (b) Interrupciones; espera de E/S breve (c) Interrupciones; espera de E/S larga

Figura 1.5. Flujo de programa del control sin interrupciones y con ellas.

INTERRUPCIONES Y EL CICLO DE INSTRUCCIÓN

Gracias a las interrupciones, el procesador puede dedicarse a ejecutar otras instrucciones mientras que la operación de E/S se está llevando a cabo. Considere el flujo de control mostrado en la Figura 1.5b. Como anteriormente, el programa de usuario alcanza un punto en el que hace una llamada al sistema que consiste en una llamada de ESCRITURA. El programa de E/S que se invoca en este caso consta sólo del código de preparación y el mandato real de E/S. Después de que se ejecuten estas pocas instrucciones, se devuelve el control al programa de usuario. Mientras tanto, el dispositivo externo está ocupado aceptando datos de la memoria del computador e imprimiéndolos. La operación de E/S se lleva a cabo de forma concurrente con la ejecución de instrucciones en el programa de usuario.

Cuando el dispositivo externo está listo para ser atendido, es decir, cuando está preparado para aceptar más datos del procesador, el módulo de E/S de este dispositivo externo manda una señal de *petición de interrupción* al procesador. El procesador responde suspendiendo la ejecución del programa actual, saltando a la rutina de servicio específica de este dispositivo de E/S, conocida como manejador de interrupción, y reanudando la ejecución original después de haber atendido al dispositivo. En la Figura 1.5b se indican con una **X** los puntos en los que se produce cada interrupción. Téngase en cuenta que se puede producir una interrupción en cualquier punto de la ejecución del programa principal, no sólo en una determinada instrucción.

De cara al programa de usuario, una interrupción suspende la secuencia normal de ejecución. Cuando se completa el procesamiento de la interrupción, se reanuda la ejecución (Figura 1.6). Por tanto, el programa de usuario no tiene que contener ningún código especial para tratar las interrupciones; el procesador y el sistema operativo son responsables de suspender el programa de usuario y, posteriormente, reanudarlo en el mismo punto.

Para tratar las *interrupciones*, se añade una *fase de interrupción* al ciclo de instrucción, como se muestra en la Figura 1.7 (compárese con la Figura 1.2). En la fase de interrupción, el procesador comprueba si se ha producido cualquier interrupción, hecho indicado por la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador continúa con la fase de búsqueda y lee la siguiente instrucción del programa actual. Si está pendiente una interrupción, el procesador suspende la ejecución del programa actual y ejecuta la rutina del *manejador de interrupción*. La rutina del manejador de interrupción es generalmente parte del sistema operativo. Normalmente, esta rutina

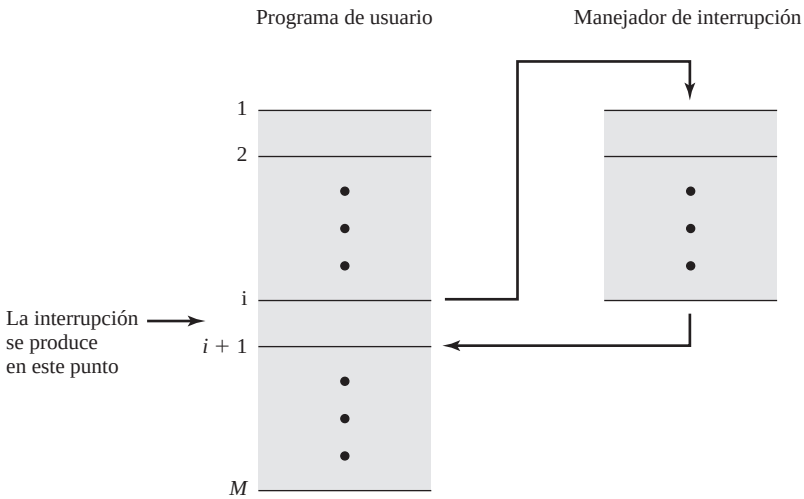


Figura 1.6. Transferencia de control mediante interrupciones.

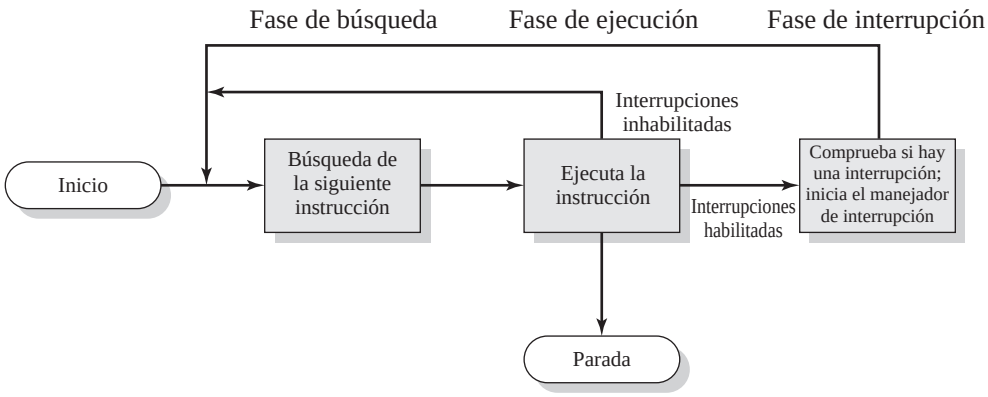


Figura 1.7. Ciclo de instrucción con interrupciones.

determina la naturaleza de la interrupción y realiza las acciones que se requieran. En el ejemplo que se está usando, el manejador determina qué módulo de E/S generó la interrupción y puede dar paso a un programa que escriba más datos en ese módulo de E/S. Cuando se completa la rutina del manejador de interrupción, el procesador puede reanudar la ejecución del programa de usuario en el punto de la interrupción.

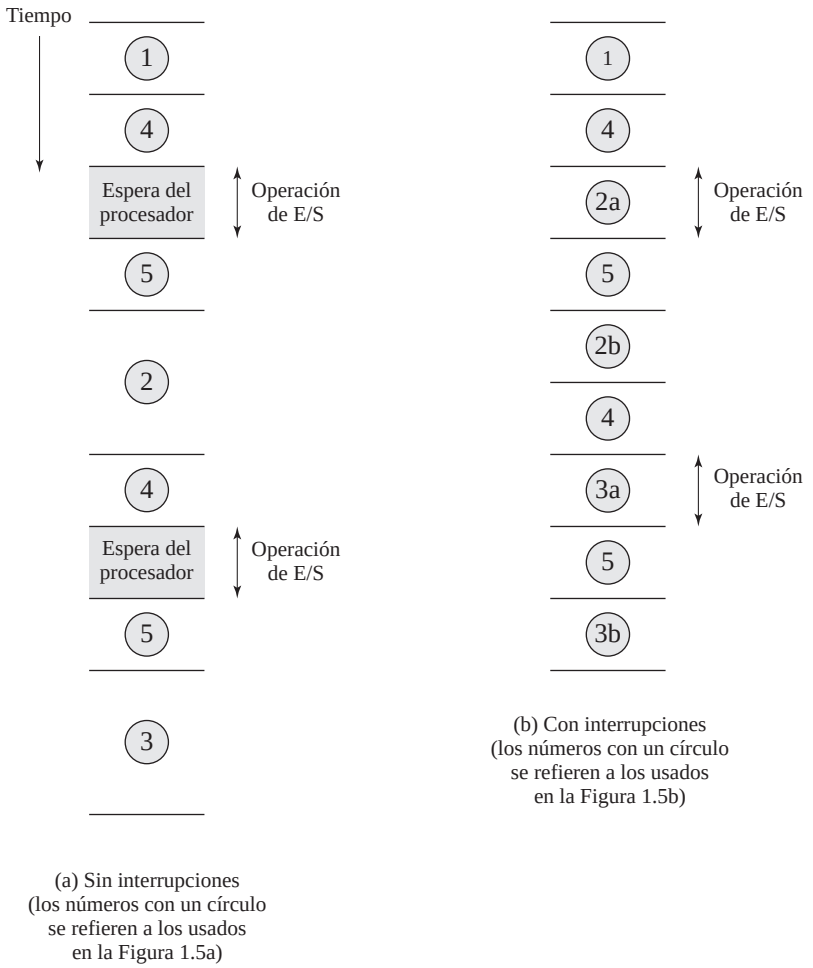
Es evidente que este proceso implica cierta sobrecarga. Deben ejecutarse instrucciones adicionales (en el manejador de interrupción) para determinar la naturaleza de la interrupción y decidir sobre la acción apropiada. Sin embargo, debido a la cantidad relativamente elevada de tiempo que se gastaría simplemente a la espera de una operación de E/S, el procesador se puede emplear mucho más eficientemente con el uso de interrupciones.

Para apreciar la ganancia en eficiencia, considere la Figura 1.8, que es un diagrama de tiempo basado en el flujo de control de las Figuras 1.5a y 1.5b. Las Figuras 1.5b y 1.8 asumen que el tiempo requerido para la operación de E/S es relativamente corto: inferior al tiempo que tarda en completarse la ejecución de instrucciones entre las operaciones de escritura del programa de usuario. El caso más típico, especialmente para un dispositivo lento como una impresora, es que la operación de E/S tarde mucho más tiempo que la ejecución de una secuencia de instrucciones de usuario. La Figura 1.5c ilustra este tipo de situación. En este caso, el programa de usuario alcanza la segunda llamada de ESCRITURA antes de que se complete la operación de E/S generada por la primera llamada. El resultado es que el programa de usuario se queda colgado en ese punto. Cuando se completa la operación de E/S precedente, se puede procesar la nueva llamada de ESCRITURA y se puede empezar una nueva operación de E/S. La Figura 1.9 muestra la temporización de esta situación con el uso de interrupciones o sin ellas. Se puede observar que hay una ganancia en eficiencia debido a que parte del tiempo durante el que se realiza la operación de E/S se solapa con la ejecución de las instrucciones del usuario.

PROCESAMIENTO DE INTERRUPCIONES

La aparición de una interrupción dispara varios eventos, tanto en el hardware del procesador como en el software. La Figura 1.10 muestra una secuencia típica. Cuando un dispositivo de E/S completa una operación de E/S, se produce la siguiente secuencia de eventos en el hardware:

1. El dispositivo genera una señal de interrupción hacia el procesador.
2. El procesador termina la ejecución de la instrucción actual antes de responder a la interrupción, como se indica en la Figura 1.7.



(a) Sin interrupciones (los números con un círculo se refieren a los usados en la Figura 1.5a)

(b) Con interrupciones (los números con un círculo se refieren a los usados en la Figura 1.5b)

Figura 1.8. Temporización del programa: espera breve de E/S.

3. El procesador comprueba si hay una petición de interrupción pendiente, determina que hay una y manda una señal de reconocimiento al dispositivo que produjo la interrupción. Este reconocimiento permite que el dispositivo elimine su señal de interrupción.
4. En ese momento, el procesador necesita prepararse para transferir el control a la rutina de interrupción. Para comenzar, necesita salvar la información requerida para reanudar el programa actual en el momento de la interrupción. La información mínima requerida es la palabra de estado del programa (PSW) y la posición de la siguiente instrucción que se va a ejecutar, que está contenida en el contador de programa. Esta información se puede apilar en la pila de control de sistema (*véase* el Apéndice 1B).
5. A continuación, el procesador carga el contador del programa con la posición del punto de entrada de la rutina de manejo de interrupción que responderá a esta interrupción. Dependiendo de la arquitectura de computador y del diseño del sistema operativo, puede haber un único programa, uno por cada tipo de interrupción o uno por cada dispositivo y tipo de interrupción. Si hay más de una rutina de manejo de interrupción, el procesador debe determinar cuál invocar. Esta información puede estar incluida en la señal de interrupción original o el procesador

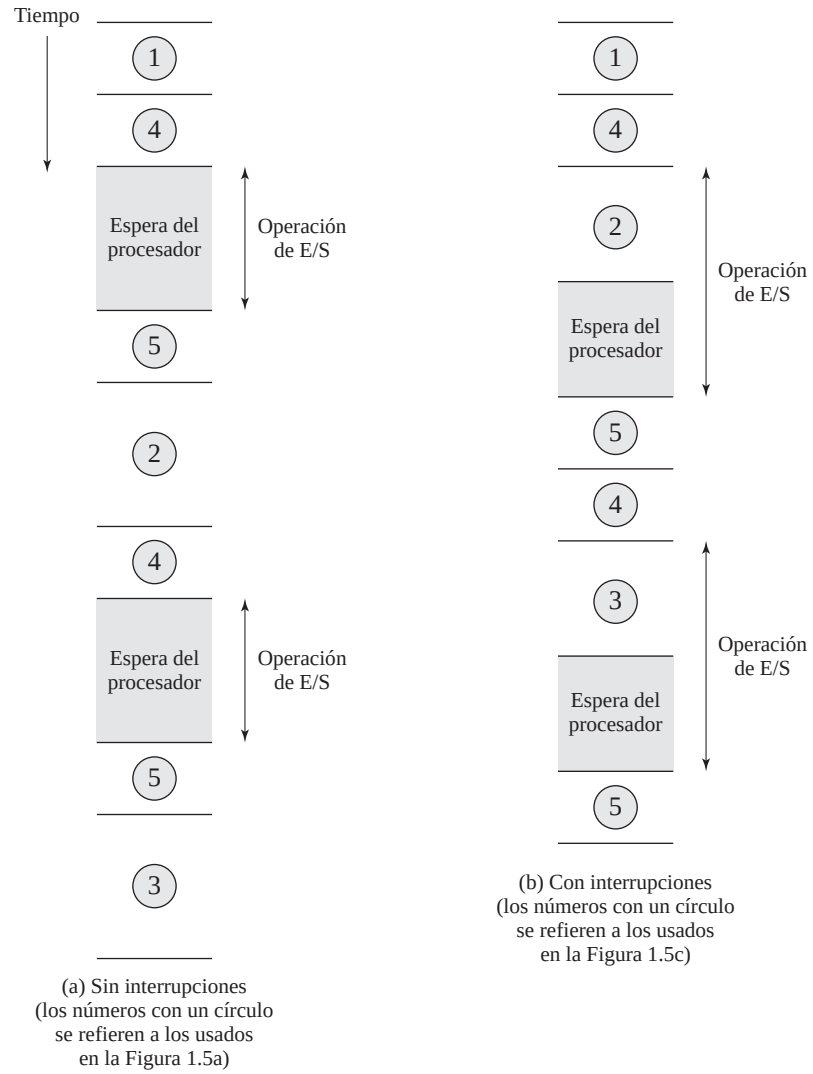


Figura 1.9. Temporización del programa: espera larga de E/S.

puede tener que realizar una petición al dispositivo que generó la interrupción para obtener una respuesta que contiene la información requerida.

Una vez que se ha cargado el contador del programa, el procesador continúa con el siguiente ciclo de instrucción, que comienza con una lectura de instrucción. Dado que la lectura de la instrucción está determinada por el contenido del contador del programa, el resultado es que se transfiere el control al programa manejador de interrupción. La ejecución de este programa conlleva las siguientes operaciones:

6. En este momento, el contador del programa y la PSW vinculados con el programa interrumpido se han almacenado en la pila del sistema. Sin embargo, hay otra información que se considera parte del estado del programa en ejecución. En concreto, se necesita salvar el contenido de los registros del procesador, puesto que estos registros los podría utilizar el manejador de interrup-

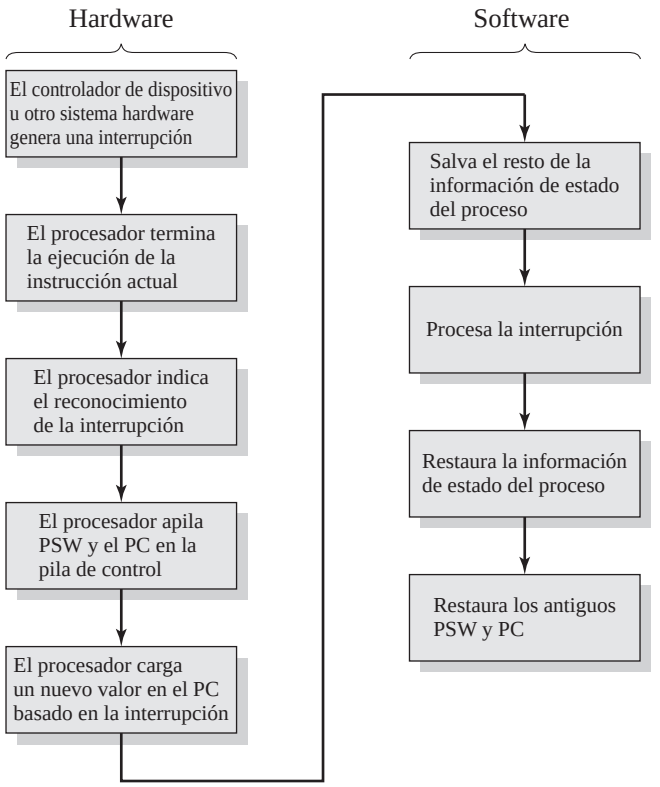


Figura 1.10. Procesamiento simple de interrupciones.

ciones. Por tanto, se deben salvar todos estos valores, así como cualquier otra información de estado. Generalmente, el manejador de interrupción comenzará salvando el contenido de todos los registros en la pila. En el Capítulo 3 se estudiará qué otra información de estado debe salvarse. La Figura 1.11a muestra un ejemplo sencillo. En este caso, un programa de usuario se interrumpe después de la instrucción en la posición *N*. El contenido de todos los registros, así como la dirección de la siguiente instrucción ($N + 1$), un total de *M* palabras, se apilan en la pila de control. El puntero de pila se actualiza para que señale a la nueva cima de la pila, mientras que el contador de programa quedará apuntando al principio de la rutina de servicio de interrupción.

7. El manejador de interrupción puede en este momento comenzar a procesar la interrupción. Esto incluirá un examen de la información de estado relacionada con la operación de E/S o con otro evento distinto que haya causado la interrupción. Asimismo, puede implicar el envío de mandatos adicionales o reconocimientos al dispositivo de E/S.
8. Cuando se completa el procesamiento de la interrupción, se recuperan los valores de los registros salvados en la pila y se restituyen en los registros (como ejemplo, *véase* la Figura 1.11b).
9. La última acción consiste en restituir de la pila los valores de la PSW y del contador del programa. Como resultado, la siguiente instrucción que se va ejecutar corresponderá al programa previamente interrumpido.

Es importante salvar toda la información de estado del programa interrumpido para su posterior reanudación. Esto se debe a que la interrupción no es una rutina llamada desde el programa. En su lu-

gar, la interrupción puede suceder en cualquier momento y, por tanto, en cualquier punto de la ejecución de un programa de usuario. Su aparición es imprevisible.

MÚLTIPLES INTERRUPCIONES

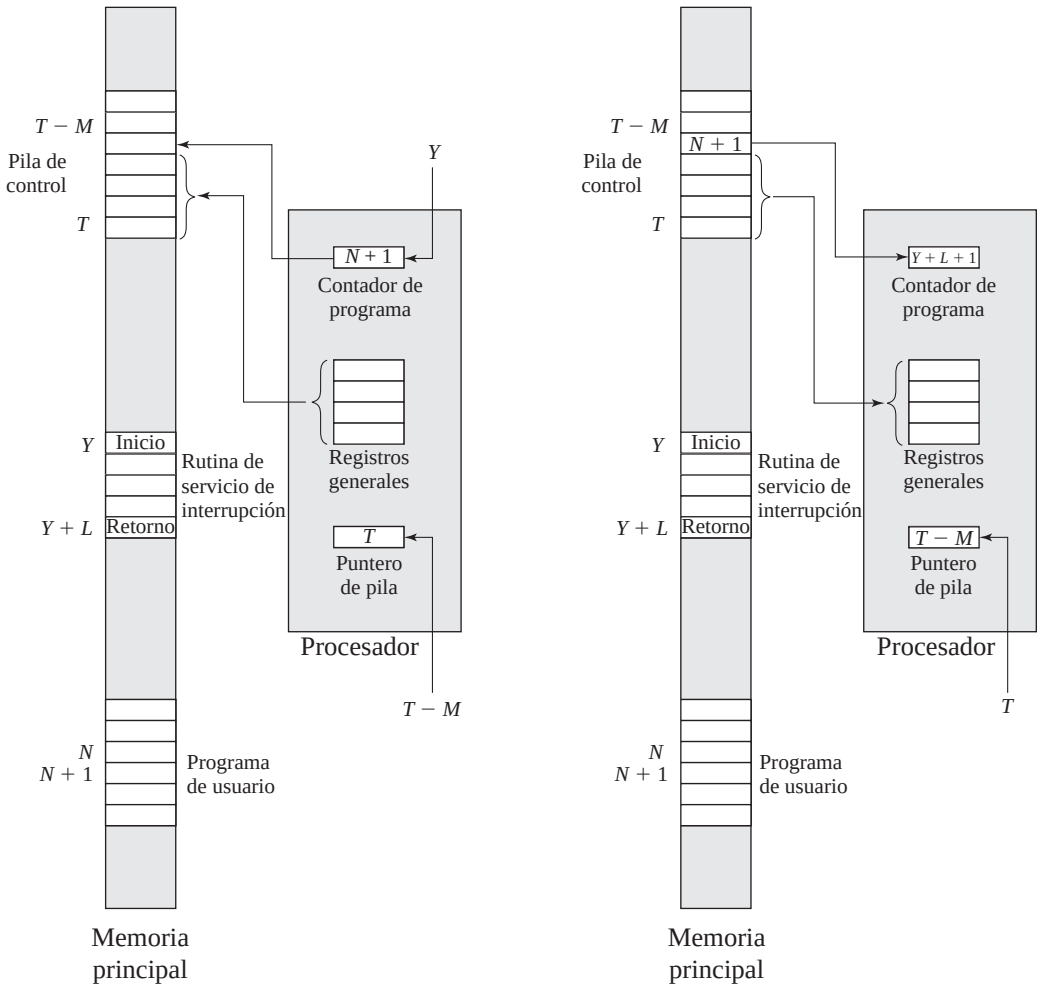
El estudio realizado hasta el momento ha tratado solamente el caso de que se produzca una única interrupción. Supóngase, sin embargo, que se producen múltiples interrupciones. Por ejemplo, un programa puede estar recibiendo datos de una línea de comunicación e imprimiendo resultados al mismo tiempo. La impresora generará una interrupción cada vez que completa una operación de impresión. El controlador de la línea de comunicación generará una interrupción cada vez que llega una unidad de datos. La unidad podría consistir en un único carácter o en un bloque, dependiendo de la naturaleza del protocolo de comunicaciones. En cualquier caso, es posible que se produzca una interrupción de comunicación mientras se está procesando una interrupción de la impresora.

Se pueden considerar dos alternativas a la hora de tratar con múltiples interrupciones. La primera es inhabilitar las interrupciones mientras que se está procesando una interrupción. Una *interrupción inhabilitada* significa simplemente que el procesador ignorará cualquier nueva señal de petición de interrupción. Si se produce una interrupción durante este tiempo, generalmente permanecerá pendiente de ser procesada, de manera que el procesador sólo la comprobará después de que se rehabiliten las interrupciones. Por tanto, cuando se ejecuta un programa de usuario y se produce una interrupción, se inhabilitan las interrupciones inmediatamente. Después de que se completa la rutina de manejo de la interrupción, se rehabilitan las interrupciones antes de reanudar el programa de usuario, y el procesador comprueba si se han producido interrupciones adicionales. Esta estrategia es válida y sencilla, puesto que las interrupciones se manejan en estricto orden secuencial (Figura 1.12a).

La desventaja de la estrategia anterior es que no tiene en cuenta la prioridad relativa o el grado de urgencia de las interrupciones. Por ejemplo, cuando llegan datos por la línea de comunicación, se puede necesitar que se procesen rápidamente de manera que se deje sitio para otros datos que pueden llegar. Si el primer lote de datos no se ha procesado antes de que llegue el segundo, los datos pueden perderse porque el *buffer* del dispositivo de E/S puede llenarse y desbordarse.

Una segunda estrategia es definir prioridades para las interrupciones y permitir que una interrupción de más prioridad cause que se interrumpa la ejecución de un manejador de una interrupción de menor prioridad (Figura 1.12b). Como ejemplo de esta segunda estrategia, considere un sistema con tres dispositivos de E/S: una impresora, un disco y una línea de comunicación, con prioridades crecientes de 2, 4 y 5, respectivamente. La Figura 1.13, basada en un ejemplo de [TANE97], muestra una posible secuencia. Un programa de usuario comienza en $t = 0$. En $t = 10$, se produce una interrupción de impresora; se almacena la información de usuario en la pila del sistema y la ejecución continúa en la rutina de servicio de interrupción (*Interrupt Service Routine*, ISR) de la impresora. Mientras todavía se está ejecutando esta rutina, en $t = 15$ se produce una interrupción del equipo de comunicaciones. Debido a que la línea de comunicación tiene una prioridad superior a la de la impresora, se sirve la petición de interrupción. Se interrumpe la ISR de la impresora, se almacena su estado en la pila y la ejecución continúa con la ISR del equipo de comunicaciones. Mientras se está ejecutando esta rutina, se produce una interrupción del disco ($t = 20$). Dado que esta interrupción es de menor prioridad, simplemente se queda en espera, y la ISR de la línea de comunicación se ejecuta hasta su conclusión.

Cuando se completa la ISR de la línea de comunicación ($t = 25$), se restituye el estado previo del proceso, que corresponde con la ejecución de la ISR de la impresora. Sin embargo, antes incluso de que pueda ejecutarse una sola instrucción de esta rutina, el procesador atiende la interrupción de disco de mayor prioridad y transfiere el control a la ISR del disco. Sólo cuando se completa esa rutina ($t = 35$), se reanuda la ISR de la impresora. Cuando esta última rutina se completa ($t = 40$), se devuelve finalmente el control al programa de usuario.



(a) La interrupción se produce después de la instrucción en la posición N

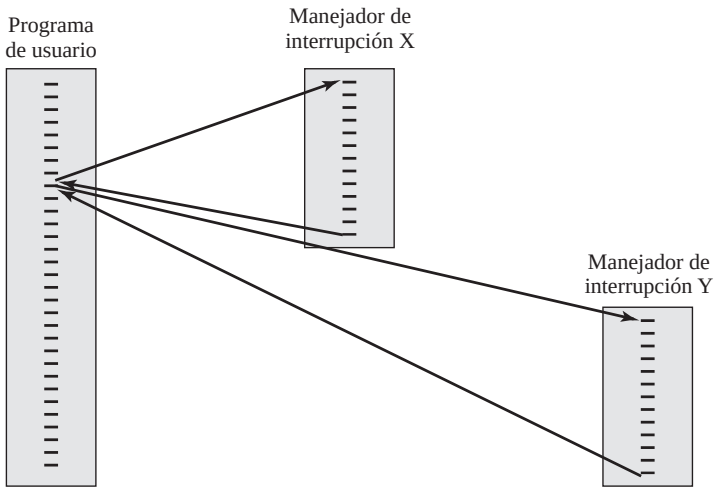
(b) Retorno de interrupción

Figura 1.11. Cambios en la memoria y en los registros durante una interrupción.

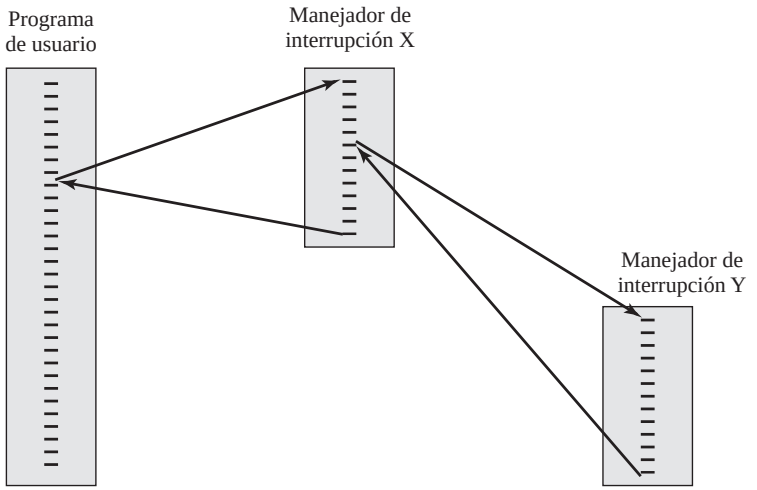
MULTIPROGRAMACIÓN

Incluso utilizando interrupciones, puede que el procesador siga sin utilizarse eficientemente. Por ejemplo, considérese la Figura 1.9b, que demuestra una mejor utilización del procesador. Si el tiempo requerido para completar una operación de E/S es mucho mayor que el código de usuario entre las llamadas de E/S (una situación habitual), el procesador estará parado la mayor parte del tiempo. Una solución a este problema es permitir que múltiples programas de usuario estén activos al mismo tiempo.

Supóngase, por ejemplo, que el procesador tiene que ejecutar dos programas. Uno de ellos simplemente se dedica a leer datos de la memoria y copiarlos a un dispositivo externo; el otro es algún tipo de aplicación que implica mucho cálculo. El procesador puede empezar con el programa que



(a) Procesamiento secuencial de interrupciones



(b) Procesamiento anidado de interrupciones

Figura 1.12. Transferencia de control con múltiples interrupciones.

genera salida, emitir un mandato de escritura al dispositivo externo y, a continuación, empezar la ejecución de la otra aplicación. Cuando el procesador trata con varios programas, la secuencia en la que se ejecutan los programas dependerá de su prioridad relativa, así como de si están esperando la finalización de una operación de E/S. Cuando se interrumpe un programa y se transfiere el control a un manejador de interrupción, una vez que se ha completado la rutina del manejador de interrupción, puede ocurrir que no se le devuelva inmediatamente el control al programa de usuario que estaba en ejecución en ese momento. En su lugar, el control puede pasar a algún otro programa pendiente de ejecutar que tenga una prioridad mayor. Posteriormente, se reanudará el programa de usuario interrumpido previamente, en el momento en que tenga la mayor prioridad. Este concepto de múltiples programas que ejecutan en turnos se denomina multiprogramación y se estudiará más adelante en el Capítulo 2.

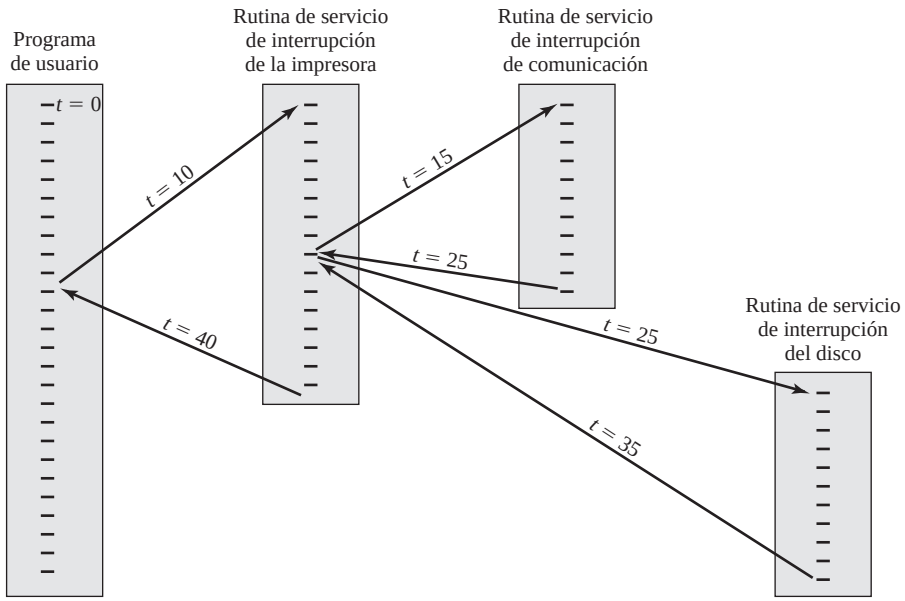


Figura 1.13. Ejemplo de secuencia de tiempo con múltiples interrupciones.

1.5. LA JERARQUÍA DE MEMORIA

Las restricciones de diseño en la memoria de un computador se pueden resumir en tres preguntas: ¿cuál es su capacidad? ¿Cuál es su velocidad? ¿Cuál es su coste?

La pregunta sobre cuánta debe ser su capacidad es algo que no tiene límite. Si se dispone de una determinada capacidad, probablemente se desarrollarán aplicaciones que la usarán. La cuestión acerca de la velocidad tiene, hasta cierto tiempo, una respuesta más fácil. Para alcanzar un rendimiento máximo, la memoria debe ser capaz de mantener el ritmo del procesador. Es decir, según el procesador va ejecutando instrucciones, no debería haber pausas esperando que estén disponibles las instrucciones o los operandos. Se debe considerar también la última pregunta. Para un sistema práctico, el coste de la memoria debe ser razonable en relación con los otros componentes.

Como se podría esperar, hay un compromiso entre las tres características fundamentales de la memoria: a saber, coste, capacidad y tiempo de acceso. En cualquier momento dado, se utilizan diversas tecnologías para implementar los sistemas de memoria. En todo este espectro de tecnologías, se cumplen las siguientes relaciones:

- Cuanto menor tiempo de acceso, mayor coste por bit.
- Cuanto mayor capacidad, menor coste por bit.
- Cuanto mayor capacidad, menor velocidad de acceso.

Queda claro el dilema al que se enfrenta el diseñador. A él le gustaría utilizar tecnologías que proporcionen una memoria de gran capacidad, tanto porque se necesita esa capacidad como porque su coste por bit es bajo. Sin embargo, para cumplir con los requisitos de rendimiento, el diseñador necesita utilizar memorias de capacidad relativamente baja con tiempos de acceso rápidos.

La solución a este dilema consiste en no basarse en un único componente de memoria o en una sola tecnología, sino emplear una **jerarquía de memoria**. En la Figura 1.14 se muestra una jerarquía típica. Según se desciende en la jerarquía, ocurre lo siguiente:

a) Disminución del coste por bit.

b) Aumento de la capacidad.

c) Aumento del tiempo de acceso.

d) Disminución de la frecuencia de acceso a la memoria por parte del procesador.

Por tanto, las memorias más rápidas, caras y pequeñas se complementan con memorias más lentas, baratas y grandes. La clave para el éxito de esta organización es el último aspecto: la disminución de la frecuencia de acceso. Este concepto se examinará con mayor detalle más adelante en este mismo capítulo, cuando se estudie la cache, y en posteriores capítulos del libro, cuando se presente la memoria virtual. De todos modos, se proporcionará una breve explicación en ese momento.

Supóngase que el procesador tiene acceso a dos niveles de memoria. El nivel 1 contiene 1.000 bytes y tiene un tiempo de acceso de 0.1 μs; el nivel 2 contiene 100.000 bytes y tiene un tiempo de acceso de 1μs. Asuma que si un byte que se va a acceder está en el nivel 1, el procesador lo hace di-

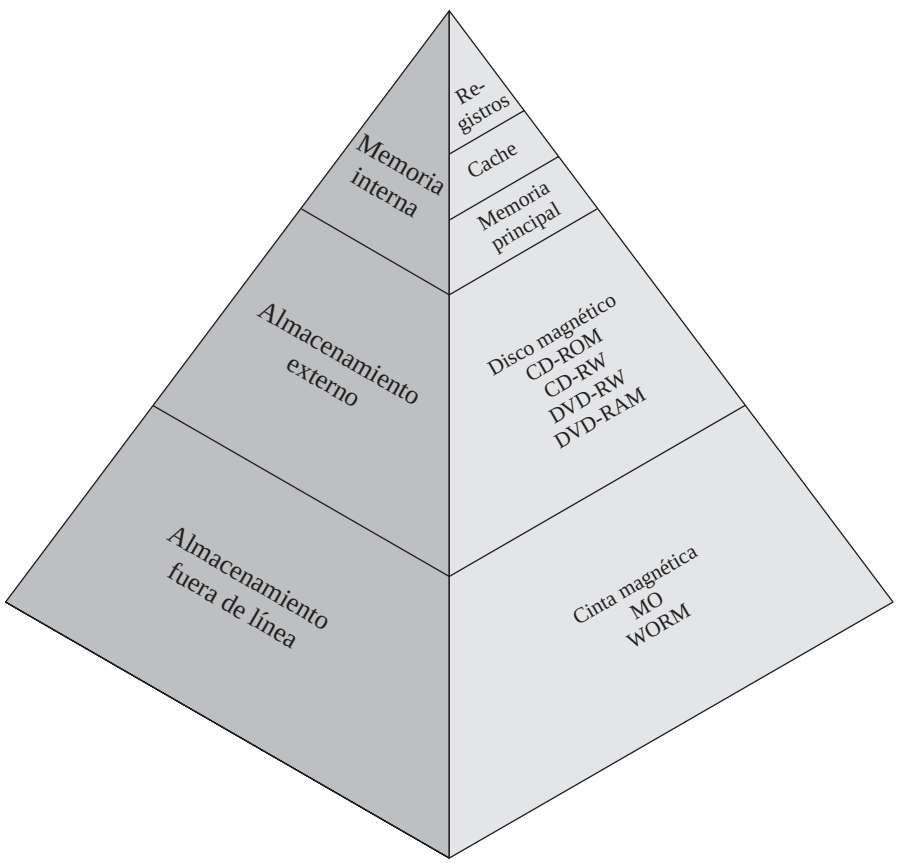


Figura 1.14. La jerarquía de memoria.

rectamente. Sin embargo, si está en el nivel 2, primero se transfiere el byte al nivel 1 y, a continuación, el procesador lo accede. Para simplificar, se ignorará el tiempo requerido por el procesador para determinar si el byte está en el nivel 1 o en el 2. La Figura 1.15 muestra la forma general de la curva que representa esta situación. La figura muestra el tiempo de acceso medio a una memoria de dos niveles como una función de la **tasa de aciertos** A, donde A se define como la fracción de todos los accesos a memoria que se encuentran en la memoria más rápida (por ejemplo, la cache), T_1 es el tiempo de acceso al nivel 1 y T_2 al nivel 2[5]. Como se puede observar, para porcentajes elevados de accesos al nivel 1, el tiempo medio total de acceso está mucho más próximo al correspondiente al nivel 1 que al del nivel 2.

En el ejemplo, se supone que el 95% de los accesos a memoria se encuentran en la cache (A = 0,95). Por tanto, el tiempo medio para acceder a un byte se puede expresar como:

$$(0{,}95)(0{,}1\ \mu s) + (0{,}05)(0{,}1\ \mu s + 1\ \mu s) = 0{,}095 + 0{,}055 = 0{,}15\ \mu s$$

El resultado está próximo al tiempo de acceso de la memoria más rápida. Por tanto, en principio, la estrategia de utilizar dos niveles de memoria funciona, pero sólo si se cumplen las condiciones de la (a) a la (d). Mediante el empleo de diversas tecnologías, existe un rango de sistemas de memoria que satisfacen las condiciones de la (a) a la (c). Afortunadamente, la condición (d) también es generalmente válida.

La validez de la condición (d) está basada en un principio conocido como la *proximidad de referencias* [DENN68]. Durante el curso de ejecución de un programa, las referencias de memoria del procesador, tanto a instrucciones como a datos, tienden a agruparse. Los programas contienen habitualmente diversos bucles iterativos y subrutinas. Una vez que se inicia un bucle o una subrutina, hay referencias repetidas a un pequeño conjunto de instrucciones. Del mismo modo, las operaciones con tablas y vectores involucran accesos a conjuntos agrupados de bytes de datos. En un periodo de tiem-

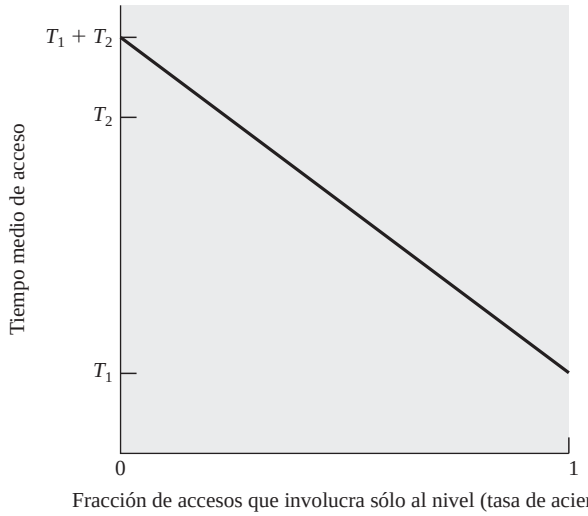


Figura 1.15. Rendimiento de una memoria simple de dos niveles.

[5] Si la palabra accedida se encuentra en la memoria más rápida, a esto se le define como un **acierto**, mientras que un **fallo** sucede cuando la palabra accedida no está en esta memoria más rápida.

po largo, las agrupaciones que se están usando van cambiando, pero en un periodo corto, el procesador está principalmente trabajando con grupos fijos de referencias a memoria.

Por consiguiente, es posible organizar los datos a través de la jerarquía de manera que el porcentaje de accesos a cada nivel sucesivamente más bajo es considerablemente menor que al nivel inferior. Considere el ejemplo de dos niveles presentado previamente. Supóngase que la memoria de nivel 2 contiene todos los datos e instrucciones del programa. Las agrupaciones actuales se pueden almacenar temporalmente en el nivel 1. De vez en cuando, una de las agrupaciones en el nivel 1 tendrá que ser expulsada al nivel 2 para hacer sitio a una nueva agrupación que llega al nivel 1. De media, sin embargo, la mayoría de las referencias corresponderá con instrucciones y datos contenidos en el nivel 1.

Este principio se puede aplicar a más de dos niveles de memoria. El tipo de memoria más rápida, pequeña y costosa consiste en los registros internos del procesador. Normalmente, un procesador contendrá unas pocas docenas de ese tipo de registros, aunque algunas máquinas contienen cientos de registros. Descendiendo dos niveles, está la memoria principal que es el sistema de memoria interna fundamental del computador. Cada posición de memoria principal tiene una única dirección. La mayoría de las instrucciones de máquina hacen referencia a una o más direcciones de memoria principal. La memoria principal se amplía usualmente con una cache, más pequeña y de mayor velocidad. La cache normalmente no es visible al programador o, incluso, al procesador. Se trata de un dispositivo que controla el movimiento de datos entre la memoria principal y los registros de procesador con objeto de mejorar el rendimiento.

Las tres formas de memoria descritas son, normalmente, volátiles y emplean tecnología de semiconductores. El uso de tres niveles explota el hecho de que la memoria de semiconductores se presenta en diversos tipos, que se diferencian en velocidad y coste. Los datos se almacenan de forma más permanente en los dispositivos de almacenamiento masivo externos, de los cuales los más comunes son los discos duros y los dispositivos extraíbles, tales como los discos extraíbles, las cintas y el almacenamiento óptico. La memoria no volátil externa se denomina también **memoria secundaria** o **memoria auxiliar**. Se usa para almacenar los ficheros de programas y datos, siendo usualmente visible al programador sólo en términos de ficheros y registros, en contraposición a bytes o palabras individuales. El disco se utiliza también para proporcionar una extensión de la memoria principal conocida como memoria virtual, que se estudiará en el Capítulo 8.

De hecho, se pueden añadir niveles adicionales a la jerarquía por software. Por ejemplo, se puede utilizar una parte de la memoria principal como una zona de almacenamiento intermedio para guardar temporalmente datos que van a ser leídos del disco. Esta técnica, denominada a veces cache de disco (que se estudia en detalle en el Capítulo 11), mejora el rendimiento de dos maneras:

- Las escrituras en el disco pueden agruparse. En vez de muchas transferencias de datos de pequeño tamaño, se producen unas pocas transferencias de gran tamaño. Esto mejora el rendimiento del disco y minimiza el grado de implicación del procesador.
- Un programa puede acceder a algunos datos destinados a ser escritos antes del siguiente volcado al disco. En ese caso, los datos se recuperan rápidamente de la cache software en vez de lentamente como ocurre cuando se accede al disco.

El Apéndice 1A estudia las implicaciones en el rendimiento de las estructuras de memoria de múltiples niveles.

1.6. MEMORIA CACHE

Aunque la memoria cache es invisible para el sistema operativo, interactúa con otros elementos del hardware de gestión de memoria. Además, muchos de los principios usados en los esquemas de memoria virtual (estudiados en el Capítulo 8) son aplicables también a la memoria cache.

MOTIVACIÓN

En todos los ciclos de instrucción, el procesador accede a memoria al menos una vez, para leer la instrucción y, con frecuencia, una o más veces adicionales, para leer y/o almacenar los resultados. La velocidad a la que el procesador puede ejecutar instrucciones está claramente limitada por el tiempo de ciclo de memoria (el tiempo que se tarda en leer o escribir una palabra de la memoria). Esta limitación ha sido de hecho un problema significativo debido a la persistente discrepancia entre la velocidad del procesador y la de la memoria principal; a lo largo de los años, la velocidad del procesador se ha incrementado constantemente de forma más rápida que la velocidad de acceso a la memoria. El diseñador se encuentra con un compromiso entre velocidad, coste y tamaño. Idealmente, se debería construir la memoria principal con la misma tecnología que la de los registros del procesador, consiguiendo tiempos de ciclo de memoria comparables a los tiempos de ciclo del procesador. Esta estrategia siempre ha resultado demasiado costosa. La solución consiste en aprovecharse del principio de la proximidad utilizando una memoria pequeña y rápida entre el procesador y la memoria principal, denominada cache.

FUNDAMENTOS DE LA CACHE

El propósito de la memoria cache es proporcionar un tiempo de acceso a memoria próximo al de las memorias más rápidas disponibles y, al mismo tiempo, ofrecer un tamaño de memoria grande que tenga el precio de los tipos de memorias de semiconductores menos costosas. El concepto se muestra en la Figura 1.16. Hay una memoria principal relativamente grande y lenta junto con una memoria cache más pequeña y rápida. La cache contiene una copia de una parte de la memoria principal. Cuando el procesador intenta leer un byte de la memoria, se hace una comprobación para determinar si el byte está en la cache. Si es así, se le entrega el byte al procesador. En caso contrario, se lee e introduce dentro de la cache un bloque de memoria principal, que consta de un cierto número fijo de bytes, y, a continuación, se le entrega el byte pedido al procesador. Debido al fenómeno de la proximidad de referencias, cuando se lee e introduce dentro de la cache un bloque de datos para satisfacer una única referencia de memoria, es probable que muchas de las referencias a memoria en el futuro próximo correspondan con otros bytes del bloque.

La Figura 1.17 representa la estructura de un sistema de memoria cache/principal. La memoria principal consta de hasta 2^n palabras direccionables, teniendo cada palabra una única dirección de n-bits. A efectos de correspondencia entre niveles, se considera que esta memoria consta de un número de **bloques** de longitud fija de K palabras cada uno. Es decir, hay $M = 2^n/K$ bloques. La cache consiste en C **huecos** (denominados también *líneas*) de K palabras cada uno, tal que el número de huecos es considerablemente menor que el número de bloques de la memoria principal ($C << M$)[6]. Algunos sub-

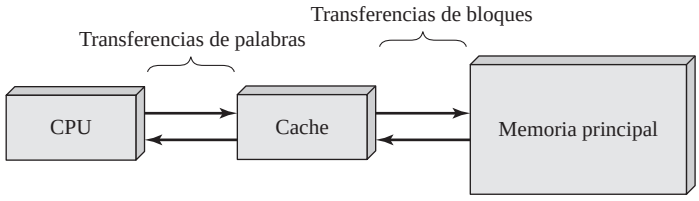


Figura 1.16. Cache y memoria principal.

[6] El símbolo << significa *mucho menor que*. De manera similar, el símbolo >> significa *mucho mayor que*.

conjuntos de los bloques de memoria principal residen en los huecos de la cache. Si se lee una palabra de un bloque de memoria que no está en la cache, se transfiere ese bloque a uno de los huecos de la cache. Dado que hay más bloques que huecos, no se puede dedicar un hueco individual de forma única y permanente a un determinado bloque. Por tanto, cada hueco incluye una etiqueta que identifica qué bloque en concreto se almacena actualmente en él. La etiqueta corresponde normalmente con varios bits de la parte de mayor peso de la dirección y hace referencia a todas las direcciones que comienzan con esa secuencia de bits.

Como un ejemplo sencillo, supóngase una dirección de seis bits y una etiqueta de 2 bits. La etiqueta 01 se refiere al bloque de posiciones con las siguientes direcciones: 010000, 010001, 010010, 010011, 010100, 010101, 010110, 010111, 011000, 011001, 011010, 011011, 011100, 011101, 011110 y 011111.

La Figura 1.18 muestra la operación de lectura. El procesador genera la dirección DL de la palabra que pretende leer. Si la cache contiene la palabra, se la entrega al procesador. En caso contrario, se carga en la cache el bloque que contiene esa palabra, proporcionando al procesador dicha palabra.

DISEÑO DE LA CACHE

Un estudio detallado del diseño de la cache queda fuera del alcance de este libro. A continuación, se resumen brevemente los elementos fundamentales. Se comprobará más adelante que hay que afrontar

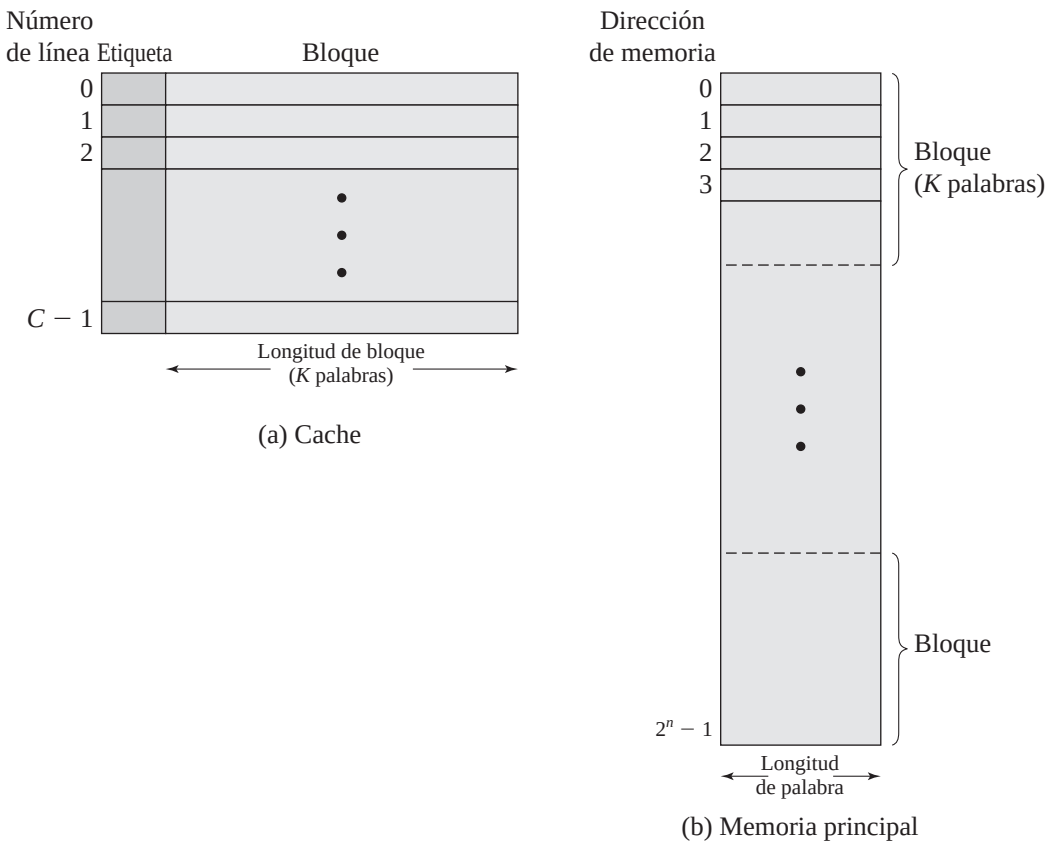


Figura 1.17. Estructura de cache/memoria principal.

aspectos de diseño similares al tratar el diseño de la memoria virtual y de la cache de disco. Se dividen en las siguientes categorías:

- Tamaño de la cache.
- Tamaño del bloque.
- Función de correspondencia.
- Algoritmo de remplazo.
- Política de escritura.

Se ha tratado ya el tema del **tamaño de la cache**, llegándose a la conclusión de que una cache de un tamaño razonablemente pequeño puede tener un impacto significativo en el rendimiento. Otro aspecto relacionado con la capacidad de la cache es el **tamaño del bloque**: la unidad de datos que se intercambia entre la cache y la memoria principal. Según el tamaño del bloque se incrementa desde muy pequeño a tamaños mayores, al principio la tasa de aciertos aumentará debido al principio de la proximidad: la alta probabilidad de que accedan en el futuro inmediato a los datos que están en la proximidad de una palabra a la que se ha hecho referencia. Según se incrementa el tamaño de bloque, se llevan a la cache más datos útiles. Sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño del bloque sigua creciendo, ya que la probabilidad de volver a usar los datos recientemente

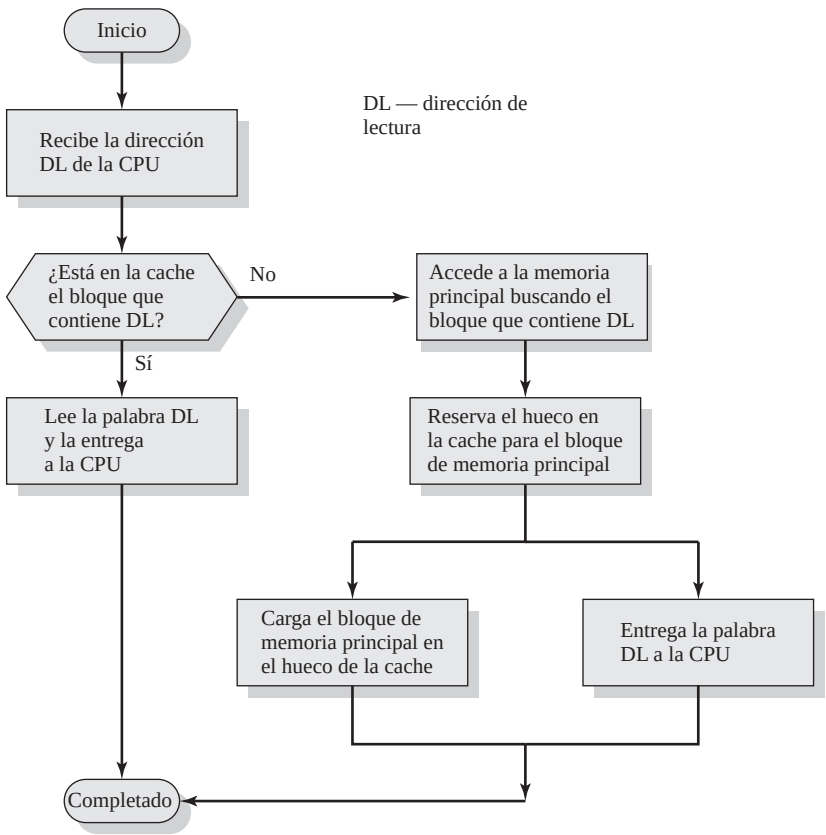


Figura 1.18. Operación de lectura de cache.

leídos se hace menor que la de utilizar nuevamente los datos que se van a expulsar de la cache para dejar sitio al nuevo bloque.

Cuando se lee e incluye un nuevo bloque de datos en la cache, la **función de correspondencia** determina qué posición de la cache ocupará el bloque. Existen dos restricciones que afectan al diseño de la función de correspondencia.

En primer lugar, cuando se introduce un bloque en la cache, se puede tener que remplazar otro. Sería deseable hacer esto de manera que se minimizara la probabilidad de que se remplazase un bloque que se necesitara en el futuro inmediato. Cuanto más flexible es la función de correspondencia, mayor grado de libertad a la hora de diseñar un algoritmo de remplazo que maximice la tasa de aciertos. En segundo lugar, cuanto más flexible es la función de correspondencia, más compleja es la circuitería requerida para buscar en la cache y determinar si un bloque dado está allí.

El **algoritmo de remplazo** selecciona, dentro de las restricciones de la función de correspondencia, qué bloque remplazar cuando un nuevo bloque va a cargarse en la cache y ésta tiene todos los huecos llenos con otros bloques. Sería deseable remplazar el bloque que menos probablemente se va a necesitar de nuevo en el futuro inmediato. Aunque es imposible identificar tal bloque, una estrategia razonablemente eficiente es remplazar el bloque que ha estado en la cache durante más tiempo sin haberse producido ninguna referencia a él. Esta política se denomina el algoritmo del *menos recientemente usado* (*Least Recently Used*, LRU). Se necesitan mecanismos hardware para identificar el bloque menos recientemente usado.

Si se altera el contenido de un bloque en la cache, es necesario volverlo a escribir en la memoria principal antes de remplazarlo. La **política de escritura** dicta cuando tiene lugar la operación de escritura en memoria. Una alternativa es que la escritura se produzca cada vez que se actualiza el bloque. Otra opción es que la escritura se realice sólo cuando se remplaza el bloque. La última estrategia minimiza las operaciones de escritura en memoria pero deja la memoria principal temporalmente en un estado obsoleto. Esto puede interferir con el modo de operación de un multiprocesador y con el acceso directo a memoria realizado por los módulos hardware de E/S.

1.7. TÉCNICAS DE COMUNICACIÓN DE E/S

Hay tres técnicas para llevar a cabo las operaciones de E/S:

- E/S programada.
- E/S dirigida de interrupciones.
- Acceso directo a memoria (*Direct Memory Access*, DMA).

E/S PROGRAMADA

Cuando el procesador ejecuta un programa y encuentra una instrucción relacionada con la E/S, ejecuta esa instrucción generando un mandato al módulo de E/S apropiado. En el caso de la E/S programada, el módulo de E/S realiza la acción solicitada y fija los bits correspondientes en el registro de estado de E/S, pero no realiza ninguna acción para avisar al procesador. En concreto, no interrumpe al procesador. Por tanto, después de que se invoca la instrucción de E/S, el procesador debe tomar un papel activo para determinar cuándo se completa la instrucción de E/S. Por este motivo, el procesador comprueba periódicamente el estado del módulo de E/S hasta que encuentra que se ha completado la operación.

Con esta técnica, el procesador es responsable de extraer los datos de la memoria principal en una operación de salida y de almacenarlos en ella en una operación de entrada. El sotfware de E/S se escribe de manera que el procesador ejecuta instrucciones que le dan control directo de la operación de E/S, incluyendo comprobar el estado del dispositivo, enviar un mandato de lectura o de escritura, y transferir los datos. Por tanto, el juego de instrucciones incluye instrucciones de E/S de las siguientes categorías:

- **Control.** Utilizadas para activar un dispositivo externo y especificarle qué debe hacer. Por ejemplo, se le puede indicar a una unidad de cinta magnética que se rebobine o avance un registro.
- **Estado.** Utilizadas para comprobar diversas condiciones de estado asociadas a un módulo de E/S y sus periféricos.
- **Transferencia.** Utilizadas para leer y/o escribir datos entre los registros del procesador y los dispositivos externos.

La Figura 1.19a proporciona un ejemplo del uso de E/S programada para leer un bloque de datos de un dispositivo externo (p. ej. un registro de cinta) y almacenarlo en memoria. Los datos se leen palabra a palabra (por ejemplo, 16 bits). Por cada palabra que se lee, el procesador debe permanecer en un bucle de comprobación del estado hasta que determina que la palabra está disponible en el registro de datos del módulo de E/S. Este diagrama de flujo subraya las desventajas principales de esta técnica: es un proceso que consume un tiempo apreciable que mantiene al procesador ocupado innecesariamente.

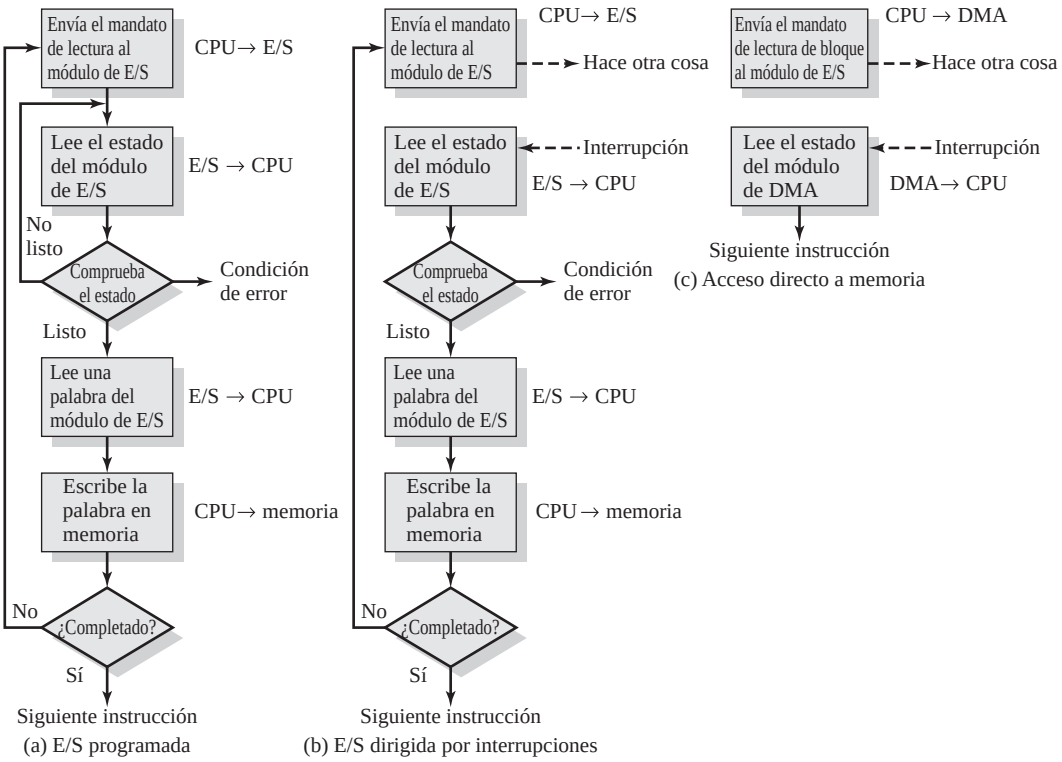


Figura 1.19. Tres técnicas para leer un bloque de datos.

E/S DIRIGIDA POR INTERRUPCIONES

El problema de la E/S programada es que el procesador tiene que esperar mucho tiempo hasta que el módulo de E/S correspondiente esté listo para la recepción o la transmisión de más datos. El procesador, mientras está esperando, debe comprobar repetidamente el estado del módulo de E/S. Como resultado, el nivel de rendimiento de todo el sistema se degrada gravemente.

Una alternativa es que el procesador genere un mandato de E/S para un módulo y, acto seguido, continúe realizando algún otro trabajo útil. El módulo de E/S interrumpirá más tarde al procesador para solicitar su servicio cuando esté listo para intercambiar datos con el mismo. El procesador ejecutará la transferencia de datos, como antes, y después reanudará el procesamiento previo.

Considere cómo funciona esta alternativa, primero desde el punto de vista del módulo de E/S. Para una operación de entrada, el módulo de E/S recibe un mandato de LECTURA del procesador. El módulo de E/S pasa entonces a leer los datos de un periférico asociado. Una vez que los datos están en el registro de datos del módulo, el módulo genera una interrupción al procesador a través de una línea de control. El módulo entonces espera hasta que el procesador pida sus datos. Cuando se hace la petición, el módulo sitúa sus datos en el bus de datos y ya está listo para otra operación de E/S.

Desde el punto de vista del procesador, las acciones correspondientes a una operación de lectura son las que se describen a continuación. El procesador genera un mandato de LECTURA. Salva el contexto (por ejemplo, el contador de programa y los registros del procesador) del programa actual y lo abandona, pasando a hacer otra cosa (por ejemplo, el procesador puede estar trabajando en varios programas diferentes a la vez). Al final de cada ciclo de instrucción, el procesador comprueba si hay interrupciones (Figura 1.7). Cuando se produce la interrupción del módulo de E/S, el procesador salva el contexto del programa que se está ejecutando actualmente y comienza a ejecutar un programa de manejo de interrupción que procesa la interrupción. En este caso, el procesador lee la palabra de datos del módulo de E/S y la almacena en memoria. A continuación, restaura el contexto del programa que había realizado el mandato de E/S (o de algún otro programa) y reanuda su ejecución.

La Figura 1.19b muestra el uso de la E/S dirigida por interrupciones para leer un bloque de datos. La E/S dirigida por interrupciones es más eficiente que la E/S programada ya que elimina la espera innecesaria. Sin embargo, la E/S dirigida por interrupciones todavía consume mucho tiempo de procesador, puesto que cada palabra de datos que va desde la memoria al módulo de E/S o desde el módulo de E/S hasta la memoria debe pasar a través del procesador.

Casi invariablemente, habrá múltiples módulos de E/S en un computador, por lo que se necesitan mecanismos para permitir que el procesador determine qué dispositivo causó la interrupción y para decidir, en caso de múltiples interrupciones, cuál debe manejar primero. En algunos sistemas, hay múltiples líneas de interrupción, de manera que cada módulo de E/S usa una línea diferente. Cada línea tendrá una prioridad diferente. Alternativamente, puede haber una única línea de interrupción, pero se utilizan líneas adicionales para guardar la dirección de un dispositivo. De nuevo, se le asignan diferentes prioridades a los distintos dispositivos.

ACCESO DIRECTO A MEMORIA

La E/S dirigida por interrupciones, aunque más eficiente que la E/S programada simple, todavía requiere la intervención activa del procesador para transferir datos entre la memoria y un módulo de E/S, ya que cualquier transferencia de datos debe atravesar un camino a través del procesador. Por tanto, ambas formas de E/S sufren dos inconvenientes inherentes:

1. La tasa de transferencia de E/S está limitada por la velocidad con la que el procesador puede comprobar el estado de un dispositivo y ofrecerle servicio.
2. El procesador está involucrado en la gestión de una transferencia de E/S; se deben ejecutar varias instrucciones por cada transferencia de E/S.

Cuando se van a transferir grandes volúmenes de datos, se requiere una técnica más eficiente: el acceso directo a memoria (*Direct Memory Access*, DMA). La función de DMA puede llevarla a cabo un módulo separado conectado en el bus del sistema o puede estar incluida en un módulo de E/S. En cualquier caso, la técnica funciona como se describe a continuación. Cuando el procesador desea leer o escribir un bloque de datos, genera un mandato al módulo de DMA, enviándole la siguiente información:

- Si se trata de una lectura o de una escritura.
- La dirección del dispositivo de E/S involucrado.
- La posición inicial de memoria en la que se desea leer los datos o donde se quieren escribir.
- El número de palabras que se pretende leer o escribir.

A continuación, el procesador continúa con otro trabajo. Ha delegado esta operación de E/S al módulo de DMA, que se ocupará de la misma. El módulo de DMA transferirá el bloque completo de datos, palabra a palabra, hacia la memoria o desde ella sin pasar a través del procesador. Por tanto, el procesador solamente está involucrado al principio y al final de la transferencia (Figura 1.19c).

El módulo de DMA necesita tomar el control del bus para transferir datos hacia la memoria o desde ella. Debido a esta competencia en el uso del bus, puede haber veces en las que el procesador necesita el bus y debe esperar al módulo de DMA. Nótese que esto no es una interrupción; el procesador no salva un contexto y pasa a hacer otra cosa. En su lugar, el procesador se detiene durante un ciclo de bus (el tiempo que se tarda en transferir una palabra a través del bus). El efecto global es causar que el procesador ejecute más lentamente durante una transferencia de DMA en el caso de que el procesador requiera acceso al bus. Sin embargo, para una transferencia de E/S de múltiples palabras, el DMA es mucho más eficiente que la E/S dirigida por interrupciones o la programada.

1.8. LECTURAS Y SITIOS WEB RECOMENDADOS

[STAL03] cubre en detalle los temas de este capítulo. Además, hay muchos otros libros sobre arquitectura y organización de computadores. Entre los textos más notables están los siguientes: [PATT98] es un estudio general; [HENN02], de los mismos autores, es un libro más avanzado que enfatiza sobre aspectos cuantitativos de diseño.

HENN02 Hennessy, J., y Patterson, D. *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 2002.

PATT98 Patterson, D., y Hennessy, J. *Computer Organization and Design: The Hardware/Software Interface*. San Mateo, CA: Morgan Kaufmann, 1998.

STAL03 Stallings, W. *Computer Organization and Architecture, 6th ed.* Upper Saddle River, NJ: Prentice Hall, 2003.

SITIOS WEB RECOMENDADOS

- **WWW Computer Architecture Home Page.** Un índice amplio de información relevante para los investigadores en arquitectura de computadores, incluyendo grupos y proyectos de arquitectura, organizaciones técnicas, bibliografía, empleo, e información comercial.
- **CPU Info Center.** Información sobre procesadores específicos, incluyendo artículos técnicos, información de productos y los últimos anuncios.

1.9. TÉRMINOS CLAVE, CUESTIONES DE REPASO Y PROBLEMAS

TÉRMINOS CLAVE

acceso directo a memoria (DMA)	marco de pila	proximidad temporal
bus del sistema	memoria cache	puntero de pila
ciclo de instrucción	memoria principal	puntero de segmento
código de condición	memoria secundaria	registro
contador de programa	módulo de E/S	registro de datos
E/S dirigida por interrupciones	multiprogramación	registro de dirección
E/S programada	pila	registro índice
entrada/salida (E/S)	procedimiento reentrante	registro de instrucción
hueco de cache	procesador	tasa de aciertos
Instrucción	proximidad	unidad central de proceso (CPU)
Interrupción	proximidad espacial	

CUESTIONES DE REPASO

1.1. Enumere y defina brevemente los cuatro elementos principales de un computador.

1.2. Defina las dos categorías principales de los registros del procesador.

1.3. En términos generales, ¿cuáles son las cuatro acciones distintas que puede especificar una instrucción de máquina?

1.4. ¿Qué es una interrupción?

1.5. ¿Cómo se tratan múltiples interrupciones?

1.6. ¿Qué características distinguen a los diversos elementos de una jerarquía de memoria?

1.7. ¿Qué es una memoria cache?

1.8. Enumere y defina brevemente las tres técnicas para las operaciones de E/S.

1.9. ¿Cuál es la diferencia entre la proximidad espacial y la temporal?

1.10. En general, ¿cuáles son las estrategias para aprovechar la proximidad espacial y la temporal?

PROBLEMAS

1.1. Suponga que la máquina hipotética de la Figura 1.3 tiene también dos instrucciones de E/S:

0011 = Carga el AC con un valor leído de un dispositivo de E/S
0111 = Almacena el AC en un dispositivo de E/S

En estos casos, la dirección de 12 bits identifica un determinado dispositivo externo. Muestre la ejecución del programa (utilizando el formato de la Figura 1.4) correspondiente al siguiente fragmento:

1. Carga el AC con un valor leído del dispositivo 5.
2. Suma al AC el contenido de la posición de memoria 940.
3. Almacena el AC en el dispositivo 6.
4. Asuma que el siguiente valor leído del dispositivo 5 es 3 y que la posición 940 contiene el valor 2.

1.2. La ejecución del programa de la Figura 1.4 se describe en el texto utilizando seis pasos. Extienda esta descripción para mostrar el uso del RDIM y del RDAM.

1.3. Considere un hipotético microprocesador de 32 bits que tiene instrucciones de 32 bits compuestas de dos campos: el primer byte contiene el código de operación y el resto un operando inmediato o la dirección de un operando.

a) ¿Cuál es la máxima capacidad de memoria directamente direccionable (en bytes)?
b) Estudie el impacto en la velocidad del sistema dependiendo de si el bus del microprocesador tiene:

1. un bus de direcciones local de 32 bits y un bus de datos local de 16 bits o
2. un bus de direcciones local de 16 bits y un bus de datos local de 16 bits.

c) ¿Cuántos bits se necesitan para el contador del programa y para el registro de instrucciones?

1.4. Considere un microprocesador hipotético que genera una dirección de 16 bits (por ejemplo, asuma que el contador de programa y los registros de dirección tienen un ancho de 16 bits) y que tiene un bus de datos de 16 bits.

a) ¿Cuál es el máximo espacio de direcciones de memoria al que el procesador puede acceder directamente si se conecta a una «memoria de 16 bits»?
b) ¿Cuál es el máximo espacio de direcciones de memoria al que el procesador puede acceder directamente si se conecta a una «memoria de 8 bits»?
c) ¿Qué características arquitectónicas permitirán a este microprocesador acceder a un «espacio de E/S» separado?
d) Si una instrucción de entrada/salida puede especificar un número de puerto de E/S de 8 bits, ¿cuántos puertos de E/S de 8 bits puede manejar el microprocesador? ¿Y cuántos de 16 bits? Razone la respuesta.

1.5. Considere un microprocesador de 32 bits, con un bus de datos externo de 16 bits, alimentado por un reloj de entrada de 8 MHz. Asuma que este microprocesador tiene un ciclo de bus cuya duración mínima es igual a cuatro ciclos del reloj de entrada. ¿Cuál es la tasa de transferencia de datos máxima en el bus que este microprocesador puede mantener medida en bytes/s? Para incrementar su rendimiento, ¿sería mejor hacer que su bus de datos externo tenga 32 bits o doblar la frecuencia del reloj externo suministrada al microprocesador?

Detalle cualquier otra suposición que se realice, razonando la misma. *Sugerencia*: determine el número de bytes que se pueden transferir por cada ciclo de bus.

1.6. Considere un computador que contiene un módulo de E/S que controla un sencillo teletipo con impresora y teclado. La CPU contiene los siguientes registros, que están conectados directamente con el bus del sistema:

RENT: Registro de entrada, 8 bits

RSAL: Registro de salida, 8 bits

INE: Indicador de entrada, 1 bit

INS: Indicador de salida, 1 bit

HAI: Habilitación de interrupción, 1 bit

El módulo de E/S controla la entrada de teclado del teletipo y la salida a la impresora. El teletipo es capaz de codificar un símbolo alfanumérico en palabras de 8 bits y descodificar una palabra de 8 bits en un símbolo alfanumérico. El indicador de entrada se activa cuando se introduce una palabra de 8 bits en el registro de entrada del teletipo. El indicador de salida se activa cuando se imprime una palabra.

a) Describa cómo la CPU, utilizando los cuatro primeros registros enumerados en este problema, puede realizar E/S con el teletipo.

b) Describa cómo se puede realizar más eficientemente la función empleando también HAI.

1.7. En prácticamente todos los sistemas que incluyen módulos de DMA, se otorga mayor prioridad a los accesos del módulo de DMA a la memoria principal que a los accesos del procesador. ¿Por qué?

1.8. Un módulo de DMA está transfiriendo caracteres a la memoria principal desde un dispositivo externo transmitiendo a 9600 bits por segundo (bps). El procesador puede leer instrucciones a una velocidad de 1 millón de instrucciones por segundo. ¿En cuánto se ralentizará el procesador debido a la actividad de DMA?

1.9. Un computador consta de una CPU y un dispositivo *D* de E/S conectado a la memoria principal *M* mediante un bus compartido con una anchura de bus de datos de una palabra. La CPU puede ejecutar un máximo de 10^6 instrucciones por segundo. Una instrucción media requiere cinco ciclos de máquina, tres de los cuales utilizan el bus de memoria. Una operación de lectura o escritura en memoria utiliza un ciclo de máquina. Supóngase que la CPU está ejecutando constantemente programas en segundo plano (*background*) que requieren el 95% de su tasa de ejecución de instrucciones pero ninguna instrucción de E/S. Asuma que un ciclo de procesador es igual a un ciclo de bus. Ahora suponga que se tienen que transferir bloques de datos muy grandes entre *M* y *D*.

a) Si se utiliza E/S programada y cada transferencia de E/S de una palabra requiere que la CPU ejecute dos instrucciones, estime la tasa máxima de transferencia de datos posible de E/S, en palabras por segundo, a través de *D*.

b) Estime la misma tasa si se utiliza una transferencia mediante DMA

1.10. Considere el siguiente código:

```
for (i = 0; i < 20; i++)
        for (j = 0; j < 10; j++)
                a[i] = a[i] *j
```

a) Proporcione un ejemplo de proximidad espacial en el código.
b) Proporcione un ejemplo de proximidad temporal en el código.

1.11. Generalice las ecuaciones (1.1) y (1.2) del Apéndice 1A para jerarquías de memoria de *n* niveles.

1.12. Considere un sistema de memoria con los siguientes parámetros:

T_c = 100 ns C_c = 0,01 céntimos/bit
T_m = 1.200 ns C_m = 0,001 céntimos/bit

a) ¿Cuál es el coste de 1MByte de memoria principal?
b) ¿Cuál es el coste de 1Mbyte de memoria principal utilizando tecnología de memoria cache?
c) Si el tiempo de acceso efectivo es un 10% mayor que el tiempo de acceso a la cache, ¿cuál es la tasa de aciertos *A*?

1.13. Un computador tiene una cache, una memoria principal y un disco usado para la memoria virtual. Si la palabra accedida está en la cache, se requieren 20 ns para accederla. Si está en la memoria principal pero no en la cache, se necesitan 60 ns para cargarla en la cache (esto incluye el tiempo para comprobar inicialmente si está en la cache) y, a continuación, la referencia comienza de nuevo. Si la palabra no está en memoria principal, se requieren 12 ms para buscar la palabra del disco, seguido de 60 ns para copiarla de la cache y luego se inicia nuevamente la referencia. La tasa de aciertos de la cache es 0,9 y la de la memoria principal es 0,6. ¿Cuál es el tiempo medio en ns requerido para acceder a una palabra en este sistema?

1.4. Suponga que el procesador utiliza una pila para gestionar las llamadas a procedimiento y los retornos de los mismos. ¿Puede eliminarse el contador de programa utilizando la cima de la pila como contador de programa?

APÉNDICE 1A CARACTERÍSTICAS DE RENDIMIENTO DE LAS MEMORIAS DE DOS NIVELES

En este capítulo, se hace referencia a la cache que actúa como un *buffer* entre la memoria principal y el procesador, creando una memoria interna de dos niveles. Esta arquitectura de dos niveles proporciona un rendimiento mejorado con respecto a una memoria de un nivel equiparable, explotando una propiedad conocida como proximidad, que se analizará en este apéndice.

Tabla 1.2. Características de las memorias de dos niveles.

	Cache de memoria principal	Memoria virtual (Paginación)	Cache de disco
Proporción típica entre tiempos de acceso	5:1	10^6:1	10^6:1
Sistema de gestión de memoria	Implementado por un hardware especial	Combinación de hardware y software de sistema	Software de sistema
Tamaño típico de bloque	4 a 128 bytes	64 a 4096 bytes	64 a 4096 bytes
Acceso del procesador al segundo nivel	Acceso directo	Acceso indirecto	Acceso indirecto

El mecanismo de cache de memoria principal es parte de la arquitectura del computador, implementado en hardware y habitualmente invisible al sistema operativo. Por tanto, en este libro no se trata este mecanismo. Sin embargo, hay otras dos instancias de la técnica de memoria de dos niveles que también explotan la propiedad de la proximidad y que son, al menos parcialmente, implementadas en el sistema operativo: la memoria virtual y la cache de disco (Tabla 1.2). Estos dos temas se analizarán en los Capítulos 8 y 11, respectivamente. En este apéndice, se revisarán algunas características de rendimiento de la memoria de dos niveles que son comunes a las tres técnicas.

PROXIMIDAD

La base de la ganancia en rendimiento de la memoria de dos niveles reside en el principio de la proximidad, comentado en la Sección 1.5. Este principio establece que las referencias a memoria tienden a agruparse. En un largo periodo de tiempo, los grupos que se están usando van cambiando, pero en un periodo corto, el procesador está primordialmente trabajando con grupos fijos de referencias a memoria.

La experiencia real muestra que el principio de la proximidad es válido. Para comprobarlo, considere la siguiente línea de razonamiento:

1. Excepto para las interrupciones de salto y llamada, que constituyen sólo una pequeña parte de todas las instrucciones del programa, la ejecución del programa es secuencial. Por tanto, en la mayoría de los casos, la próxima instrucción que se va a leer es la que sigue inmediatamente a la última instrucción leída.
2. No es frecuente que se produzca una larga secuencia ininterrumpida de llamadas a procedimiento seguida por la correspondiente secuencia de retornos. Lo habitual es que un programa permanezca confinado en una ventana de anidamiento de invocación de procedimientos bastante estrecha. Por tanto, durante un periodo corto de tiempo, las referencias a instrucciones tienden a localizarse en unos pocos procedimientos.
3. La mayoría de las construcciones iterativas consta de un número relativamente pequeño de instrucciones repetidas muchas veces. Mientras se ejecuta una iteración, el cálculo queda confinado, por tanto, en una pequeña parte contigua del programa.
4. En muchos programas, gran parte del cálculo implica procesar estructuras de datos, tales como vectores o secuencias de registros. En muchos casos, las sucesivas referencias a estas estructuras de datos corresponderán con elementos de datos situados próximamente.

Esta línea de razonamiento se ha confirmado en muchos estudios. Con respecto al primer punto, varios estudios han analizado el comportamiento de programas escritos en lenguajes de alto nivel. La Tabla 1.3 incluye los resultados fundamentales, midiendo la aparición de varios tipos de sentencias durante la ejecución, extraídos de los estudios que se detallan a continuación. El más antiguo estudio del comportamiento de un lenguaje de programación, realizado por Knuth [KNUT71], examinaba una colección de programas en FORTRAN usados como ejercicios para estudiantes. Tanenbaum [TANE78] publicó medidas recogidas de unos 300 procedimientos utilizados en programas del sistema operativo y escritos en un lenguaje que da soporte a la programación estructurada (SAL). Patterson y Sequin [PATT82] analizaron un conjunto de medidas obtenidas por compiladores, programas de composición de documentos, de diseño asistido por computador (*Computer-Aided Design*, CAD), de ordenamiento y de comparación de ficheros. Se estudiaron los lenguajes de programación C y Pascal. Huck [HUCK83] analizó cuatro programas seleccionados para representar una mezcla de cálculos científicos de propósito general, incluyendo la transformada rápida de Fourier y la integración de sistemas de ecuaciones diferenciales. Hay una coincidencia general en los resultados de esta mezcla

de lenguajes y aplicaciones en lo que se refiere a que las instrucciones de salto y de llamada representan sólo una fracción de las sentencias ejecutadas durante la vida de un programa. Por tanto, estos estudios confirman la primera afirmación de la lista precedente.

Tabla 1.3. Frecuencia dinámica relativa de las operaciones en lenguajes de alto nivel.

Estudio	[HUCK83]	[KNUT71]	[PATT82]		[TANE78]
Lenguaje	Pascal	FORTRAN	Pascal	C	SAL
Tipo de carga	Científica	Estudiantes	Sistema	Sistema	Sistema
Asignación	74	67	45	38	42
Bucle	4	3	5	3	4
Llamada	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
Otros	—	7	6	1	6

Con respecto a la segunda afirmación, los estudios presentados en [PATT85] proporcionan una confirmación. Esto se ilustra en la Figura 1.20, que muestra el comportamiento de llamadas y retornos. Cada llamada se representa por una línea descendente que se desplaza hacia la derecha, y cada retorno por una línea ascendente desplazándose a la derecha. En la figura, se define una *ventana* con una profundidad igual a 5. Sólo una secuencia de llamadas y retornos con un movimiento neto de 6 en cualquier dirección causa que la ventana se mueva. Como puede observarse, el programa en ejecución puede permanecer dentro de una ventana estacionaria durante largos periodos de tiempo. Un estudio de los mismos autores sobre programas en C y Pascal mostró que una ventana de profundidad 8 sólo necesitaría desplazarse en menos del 1% de las llamadas o retornos [TAMI83].

El principio de proximidad de referencias continúa siendo validado en los estudios más recientes. Por ejemplo, la Figura 1.21 muestra el resultado de un estudio de los patrones de acceso a las páginas web de un determinado sitio.

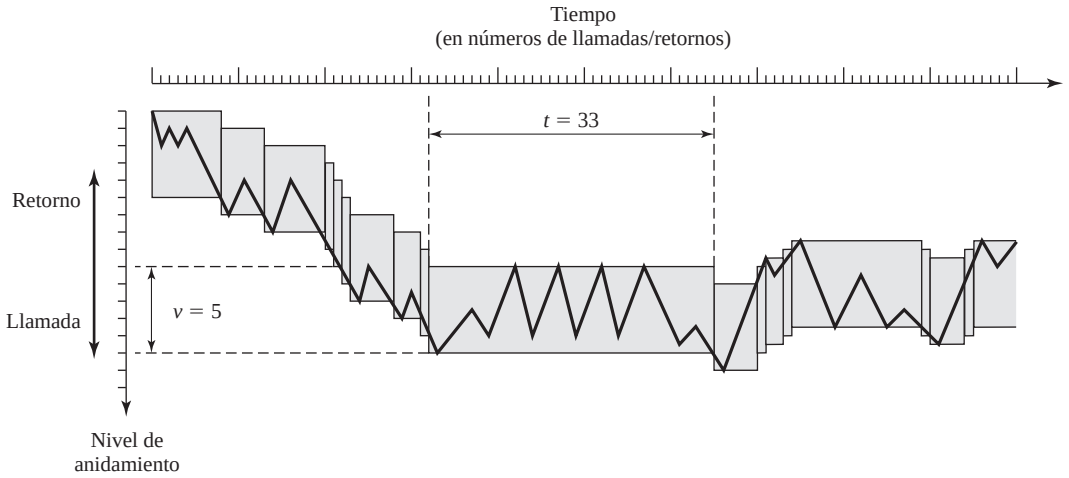


Figura 1.20. Ejemplo de comportamiento de llamadas y retornos de un programa.

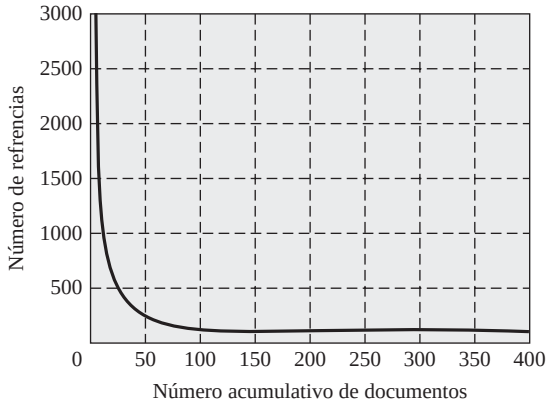


Figura 1.21. Proximidad de referencias para páginas web [BAEN97].

La bibliografía sobre el tema hace una distinción entre la proximidad espacial y la temporal. La **proximidad espacial** se refiere a la tendencia de una ejecución a involucrar posiciones de memoria que están agrupadas. Esto refleja la tendencia de un procesador a acceder secuencialmente a las instrucciones. La proximidad espacial también refleja la tendencia de un programa a acceder de forma secuencial a las posiciones de datos, como cuando se procesa una tabla de datos. La **proximidad temporal** hace referencia a la tendencia de un procesador a acceder a posiciones de memoria que se han utilizado recientemente. Por ejemplo, cuando se ejecuta un bucle, el procesador ejecuta el mismo juego de instrucciones repetidamente.

Tradicionalmente, la proximidad temporal se explota manteniendo en la memoria cache los valores de las instrucciones y los datos usados recientemente aprovechando una jerarquía de cache. La proximidad espacial se explota generalmente utilizando bloques de cache más grandes e incorporando mecanismos de lectura anticipada (se buscan elementos cuyo uso se prevé) en la lógica de control de la cache. Recientemente, ha habido investigaciones considerables para la mejora de estas técnicas con objeto de alcanzar un mayor rendimiento, pero las estrategias básicas siguen siendo las mismas.

MODO DE OPERACIÓN DE LA MEMORIA DE DOS NIVELES

La propiedad de la proximidad se puede explotar para la creación de una memoria de dos niveles. La memoria de nivel superior (M1) es más pequeña, más rápida y más cara (por bit) que la memoria de nivel inferior (M2). M1 se utiliza como un almacenamiento temporal para una parte del contenido de M2, que es más grande. Cuando se realiza una referencia a memoria, se hace un intento de acceder al elemento en M1. Si tiene éxito, se lleva a cabo un acceso rápido. En caso contrario, se copia un bloque de posiciones de memoria de M2 a M1 y, a continuación, el acceso tiene lugar en M1. Gracias a la proximidad, una vez que se trae un bloque a M1, debería haber varios accesos a las posiciones en ese bloque, dando como resultado un servicio global rápido.

Para expresar el tiempo medio de acceso a un elemento, no sólo se deberá considerar las velocidades de los dos niveles de memoria sino también la probabilidad de que una referencia dada puede encontrarse en M1. Se tiene:

$$\begin{aligned} T_S &= A \times T_1 + (1 - A) \times (T_1 + T_2) \\ &= T_1 + (1 - A) \times T_2 \end{aligned} \tag{1.1}$$

donde:

T_S = tiempo medio de acceso (del sistema).

T_1 = tiempo de acceso a M1 (por ejemplo, cache, cache de disco).

T_2 = tiempo de acceso a M2 (por ejemplo, memoria principal, disco).

A = tasa de aciertos (tasa de referencias encontradas en M1).

La Figura 1.15 muestra el tiempo medio de acceso como una función de la tasa de aciertos. Como puede observarse, para un porcentaje alto de aciertos, el tiempo medio de acceso total está mucho más cerca al de M1 que al de M2.

RENDIMIENTO

A continuación, se examinan algunos de los parámetros relevantes para la valoración de un mecanismo de memoria de dos niveles. En primer lugar, se considera el coste:

$$C_S = \frac{C_1 D_1 + C_2 D_2}{D_1 + D_2} \tag{1.2}$$

donde:

C_S = coste medio por bit de la memoria combinada de dos niveles

C_1 = coste medio por bit de la memoria M1 de nivel superior

C_2 = coste medio por bit de la memoria M2 de nivel inferior

D_1 = tamaño de M1

D_2 = tamaño de M2

Sería deseable que $C_S \approx C_2$. Dado que $C_1 >> C_2$, se requiere que $M_1 << M_2$. La Figura 1.22 muestra la relación[7].

Seguidamente, considere el tiempo de acceso. Para que una memoria de dos niveles proporcione una mejora significativa de rendimiento, se necesita tener T_S aproximadamente igual a T_1 ($T_S \approx T_1$). Dado que T_1 es mucho menor que T_2 ($T_1 << T_2$), se necesita una tasa de aciertos próxima a 1.

Por consiguiente, se pretende que M1 sea pequeña para mantener el coste bajo y grande para mejorar la tasa de acierto y, por tanto, el rendimiento. ¿Hay un tamaño de M1 que satisface ambos requisitos hasta un punto razonable? Se puede responder a esta cuestión mediante una serie de preguntas adicionales:

- ¿Qué valor de la tasa de aciertos se necesita para satisfacer el requisito de rendimiento planteado?
- ¿Qué tamaño de M1 asegurará la tasa de aciertos requerida?
- ¿Satisface este tamaño el requisito de coste?

---

[7] Nótese que los dos ejes usan una escala logarítmica. Una revisión básica de las escalas logarítmicas se encuentra en el documento de repaso de matemáticas en el *Computer Science Student Support Site* en **WilliamStallings.com/StudentSupport.html**..

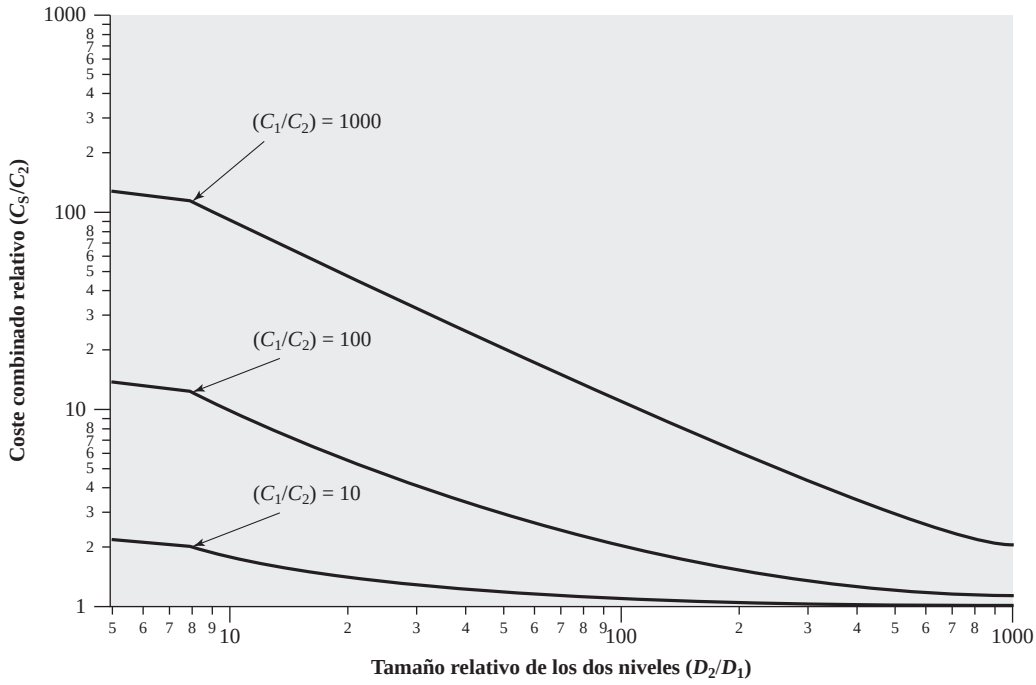


Figura 1.22. Relación del coste medio de la memoria en relación con su tamaño relativo para una memoria de dos niveles.

Para responder a estas preguntas, considere la cantidad T_1/T_S, que se conoce como *eficiencia de acceso*. Es una medida que refleja hasta qué punto el tiempo de acceso medio (T_S) está cerca del tiempo de acceso de M1 (T_1). A partir de la ecuación (1.1):

$$\frac{T_1}{T_S} = \frac{1}{1 + (1 - A)\dfrac{T_2}{T_1}} \tag{1.3}$$

En la Figura 1.23 se dibuja T_1/T_S como una función de la tasa de aciertos A, con la cantidad T_2/T_1 como parámetro. Parece necesitarse una tasa de aciertos en el intervalo de 0,8 a 0,9 para satisfacer el requisito de rendimiento.

En este momento se puede expresar más exactamente la pregunta sobre el tamaño relativo de la memoria. ¿Es razonable una tasa de aciertos mayor o igual a 0,8 para $D_1 << D_2$? Esto dependerá de varios factores, incluyendo las características del software que se está ejecutando y los detalles de diseño de la memoria de dos niveles. El determinante principal es, evidentemente, el grado de proximidad. La Figura 1.24 sugiere el efecto de la proximidad en la tasa de aciertos. Claramente, si M1 tiene el mismo tamaño que M2, la tasa de aciertos será igual a 1,0: todos los elementos en M2 también estarán siempre almacenados en M1. A continuación, supóngase que no hay proximidad, es decir, las referencias son completamente aleatorias. En ese caso, la tasa de aciertos será una función estrictamente lineal del tamaño relativo de la memoria. Por ejemplo, si M1 tiene la mitad de tamaño de M2, en cualquier momento la mitad de los elementos de M2 están también en M1 y la tasa de aciertos será de 0,5. Sin embargo, en la práctica, hay algún grado de proximidad en las referencias. En la figura se indican los efectos de una proximidad moderada y de una acusada.

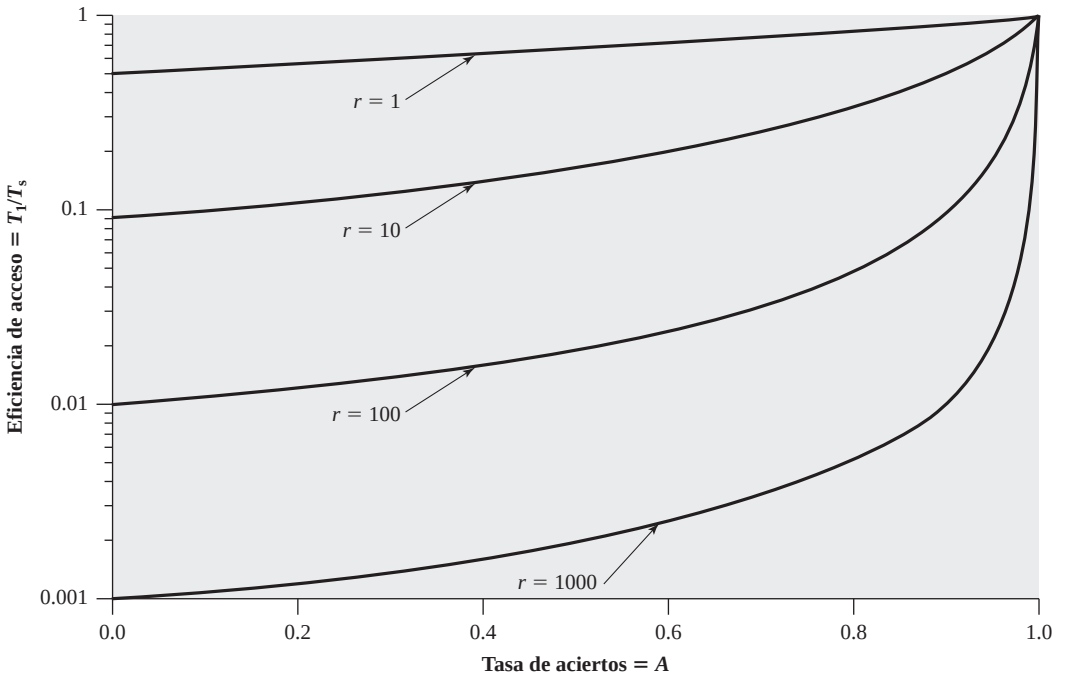


Figura 1.23. Eficiencia de acceso en función de la tasa de aciertos ($r = T_2/T_1$).

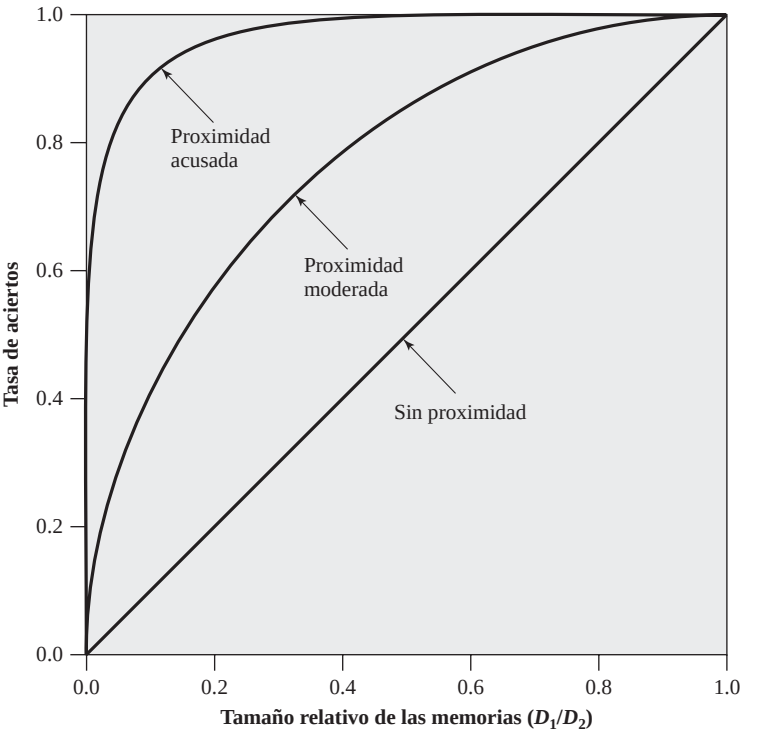


Figura 1.24. Tasa de aciertos en función del tamaño de la memoria.

En consecuencia, si hay una proximidad acusada, es posible lograr valores elevados en la tasa de aciertos incluso con un tamaño de memoria de nivel superior relativamente pequeño. Por ejemplo, numerosos estudios muestran que tamaños de cache bastante pequeños producirán una tasa de aciertos por encima del 0,75, *con independencia del tamaño de la memoria principal* (por ejemplo, [AGAR98], [PRZY88], [STRE83] y [SMIT82]). Una cache con un tamaño en el intervalo entre 1K y 128K palabras es generalmente adecuada, mientras que actualmente la memoria principal se presenta normalmente en un intervalo que se extiende entre cientos de megabytes hasta más de un gigabyte. Cuando se considera la memoria virtual y la cache de disco, se pueden citar otros estudios que confirman el mismo fenómeno, es decir, que una memoria M1 relativamente pequeña produce un valor elevado en la tasa de aciertos gracias a la proximidad.

Esto conduce a la última pregunta listada anteriormente: ¿satisface el tamaño relativo de las dos memorias el requisito de coste? La respuesta es claramente sí. Si se necesita sólo una memoria de nivel superior relativamente pequeña para alcanzar un buen rendimiento, el coste medio por bit de los dos niveles de memoria se aproximará al de la memoria de nivel inferior.

APÉNDICE 1B CONTROL DE PROCEDIMIENTOS

Una técnica habitual para controlar la ejecución de llamadas a procedimiento y los retornos de los mismos es utilizar una pila. Este apéndice resume las propiedades básicas de las pilas y revisa su uso para el control de procedimientos.

IMPLEMENTACIÓN DE LA PILA

Una pila es un conjunto ordenado de elementos, tal que en cada momento solamente se puede acceder a uno de ellos (el más recientemente añadido). El punto de acceso se denomina *cima* de la pila El número de elementos de la pila, o longitud de la pila, es variable. Por esta razón, se conoce también a una pila como una *lista de apilamiento* o *lista donde el último que entra es el primero que sale* (*Last-In First-Out*, LIFO).

La implementación de una pila requiere que haya un conjunto de posiciones dedicado a almacenar los elementos de la pila. En la Figura 1.25 se muestra una técnica habitual. Se reserva en memoria principal (o memoria virtual) un bloque contiguo de posiciones. La mayoría de las veces el bloque está parcialmente lleno con elementos de la pila y el resto está disponible para el crecimiento de la pila. Se necesitan tres direcciones para un funcionamiento adecuado, que habitualmente se almacenan en registros del procesador:

- **Puntero de pila.** Contiene la dirección de la cima de la pila. Si se añade un elemento (APILA) o se elimina (EXTRAE), el puntero se decrementa o se incrementa para contener la dirección de la nueva cima de la pila.
- **Base de la pila.** Contiene la dirección de la posición inferior en el bloque reservado. Se trata de la primera posición que se utiliza cuando se añade un elemento a una pila vacía. Si se hace un intento de extraer un elemento cuando la pila está vacía, se informa del error.
- **Límite de la pila.** Contiene la dirección del otro extremo, o cima, del bloque reservado. Si se hace un intento para apilar un elemento cuando la pila está llena, se indica el error.

Tradicionalmente, y en la mayoría de las máquinas actuales, la base de la pila está en el extremo con la dirección más alta del bloque de pila reservado, y el límite está en el extremo con la dirección más baja. Por tanto, la pila crece desde las direcciones más altas a las más bajas.

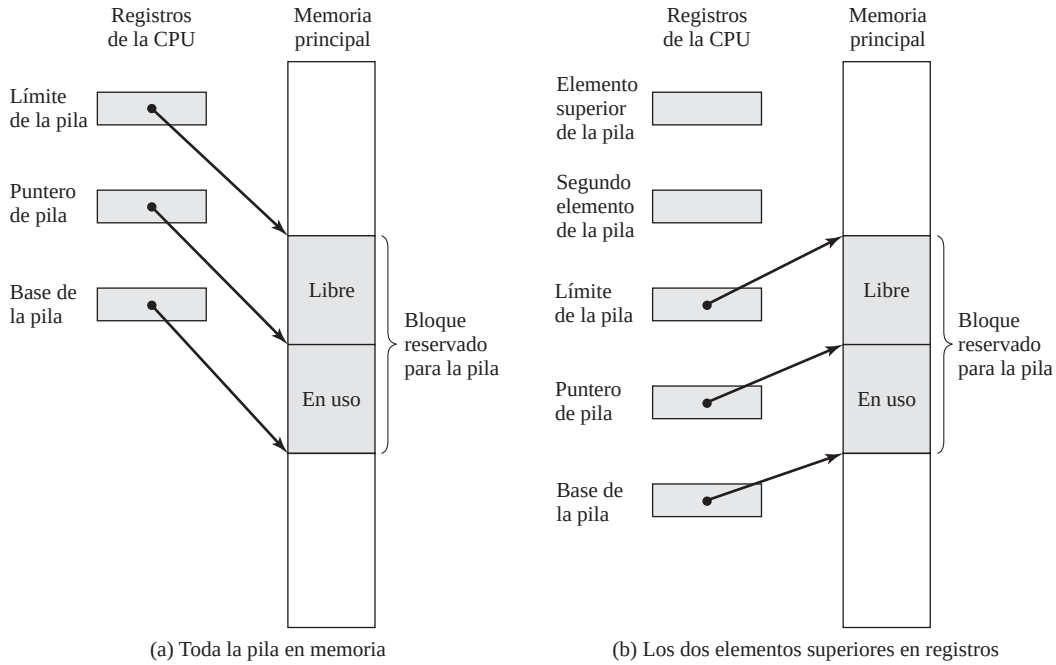


Figura 1.25. Organización habitual de la pila.

LLAMADAS Y RETORNOS DE PROCEDIMIENTOS

Una técnica habitual para gestionar las llamadas y los retornos de los procedimientos es utilizar una pila. Cuando el procesador ejecuta una llamada, se almacena (apila) la dirección de retorno en la pila. Cuando se ejecuta un retorno, se utiliza la dirección de la cima de la pila y se elimina (extrae) esa dirección de la pila. La Figura 1.27 muestra el uso de la pila para los procedimientos anidados presentados en la Figura 1.26.

Es también necesario con frecuencia pasar parámetros en una llamada a procedimiento. Estos se podrían pasar en los registros. Otra posibilidad es almacenar los parámetros en la memoria justo después de las instrucciones de llamada. En este caso, el retorno debe estar en la posición siguiente a los parámetros. Ambas técnicas tienen sus inconvenientes. Si se utilizan los registros, el programa llamado y el que realiza la llamada deben escribirse de manera que se asegure que los registros se utilizan apropiadamente. El almacenamiento de parámetros en memoria hace difícil intercambiar un número variable de parámetros.

Una estrategia más flexible para el paso de parámetros es la pila. Cuando el procesador ejecuta una llamada, no sólo apila la dirección de retorno, sino también los parámetros que se desean pasar al procedimiento llamado. El procedimiento invocado puede acceder a los parámetros en la pila. Al retornar, los parámetros de retorno se pueden almacenar también en la pila, *debajo* de la dirección de retorno. El conjunto completo de parámetros, incluyendo la dirección de retorno, que se almacena en una invocación de procedimiento se denomina **marco de pila**.

En la Figura 1.28 se muestra un ejemplo. El ejemplo se refiere a un procedimiento P en el que se declaran las variables locales $x1$ y $x2$, y el procedimiento Q, al cual puede llamar P y en el que se declaran las variables $y1$ y $y2$. El primer elemento almacenado en cada marco de pila es un puntero al principio del marco previo. Esta técnica se necesita si el número o la longitud de los parámetros que

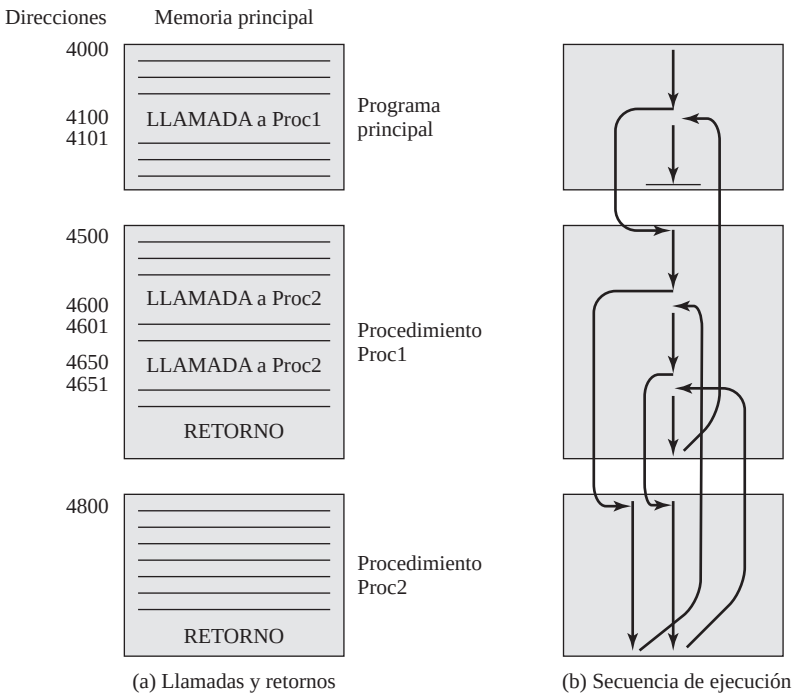


Figura 1.26. Procedimientos anidados.

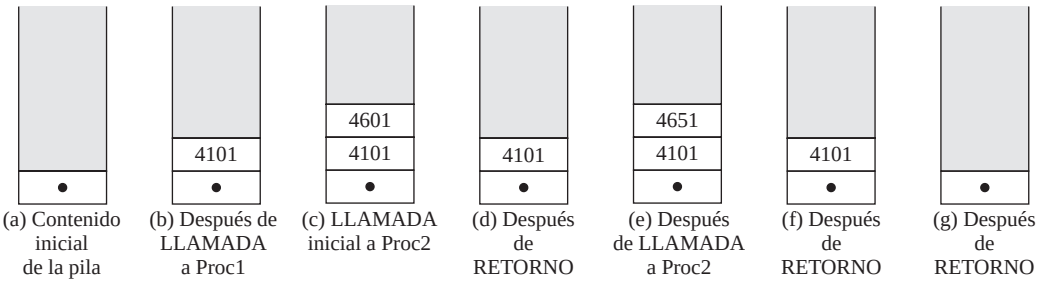


Figura 1.27. Uso de la pila para implementar los procedimientos anidados de la Figura 1.26.

se van a apilar es variable. A continuación, se almacena el punto de retorno del procedimiento que corresponde a este marco de pila. Finalmente, se reserva espacio en la cima del marco de pila para las variables locales. Estas variables locales se pueden utilizar para el paso de parámetros. Por ejemplo, supóngase que cuando P llama a Q le pasa un valor como parámetro. Este valor se podría almacenar en la variable *y*1. Por tanto, en un lenguaje de alto nivel, habría una instrucción en la rutina P similar a la siguiente:

LLAMADA Q(*y*1)

Cuando se ejecuta esta llamada, se crea un nuevo marco de pila para Q (Figura 1.28b), que incluye un puntero al marco de pila para P, la dirección de retorno de P, y dos variables locales para Q, una

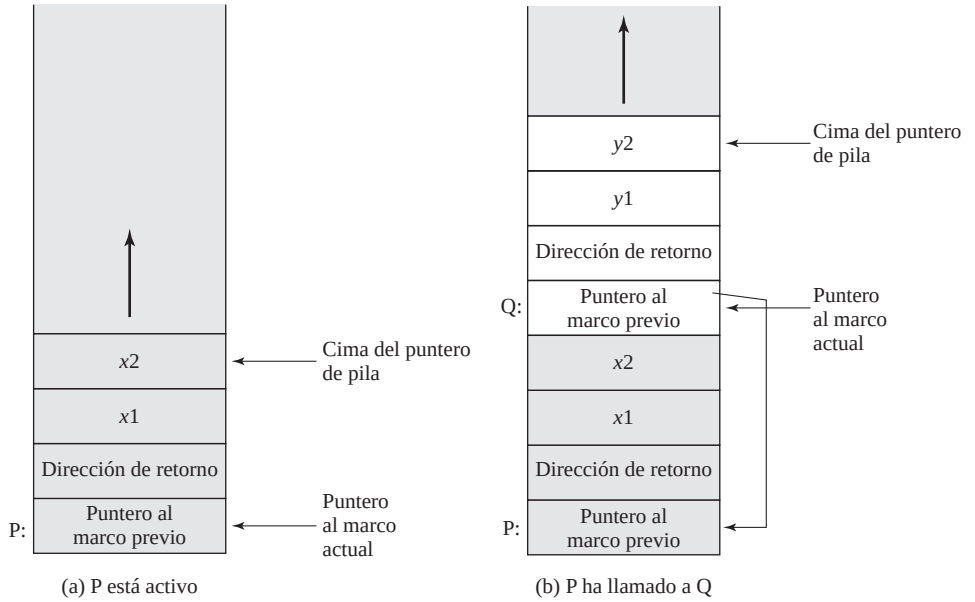


Figura 1.28. Crecimiento del marco de pila utilizando los procedimientos de ejemplo P y Q.

de las cuales se inicia con el valor pasado como parámetro desde P. La otra variable local, *y*2, es simplemente una variable local utilizada por Q en sus cálculos. En el siguiente apartado se analiza la necesidad de incluir las variables locales en el marco de pila.

PROCEDIMIENTOS REENTRANTES

Un concepto útil, particularmente en un sistema que da soporte a múltiples usuarios al mismo tiempo, es el de los procedimientos reentrantes. Un procedimiento reentrante es aquél en el que una única copia del código del programa se puede compartir por múltiples usuarios durante el mismo periodo de tiempo. El carácter reentrante de un procedimiento tiene dos aspectos fundamentales: el código del programa no puede modificarse a sí mismo y los datos locales de cada usuario deben almacenarse separadamente. Un procedimiento reentrante puede ser interrumpido e invocado por el programa que causó la interrupción y, a pesar de ello, ejecutarse correctamente al retornar al procedimiento. En un sistema compartido, el carácter reentrante permite un uso más eficiente de la memoria principal. Se mantiene una copia del código del programa en memoria principal, pero más de una aplicación puede llamar al procedimiento.

Por tanto, un procedimiento reentrante debe tener una parte permanente (las instrucciones que constituyen el procedimiento) y una parte temporal (un puntero de retorno al programa que realizó la llamada, así como espacio para las variables locales usadas por el programa). Cada instancia de la ejecución, llamada activación, de un procedimiento ejecutará el código en la parte permanente pero debe tener su propia copia de los parámetros y las variables locales. La parte temporal asociada con una activación en particular se denomina *registro de activación*.

La manera más conveniente de dar soporte a los procedimientos reentrantes es mediante una pila. Cuando se llama a un procedimiento reentrante, el registro de activación del procedimiento se puede almacenar en la pila. Por tanto, el registro de activación se convierte en parte del marco de pila que se crea en la llamada a procedimiento.