mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

TOMO 10

Editorial Delta, S.A.

mi COMPUTER

mi COMPUTER

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, F. Martín
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

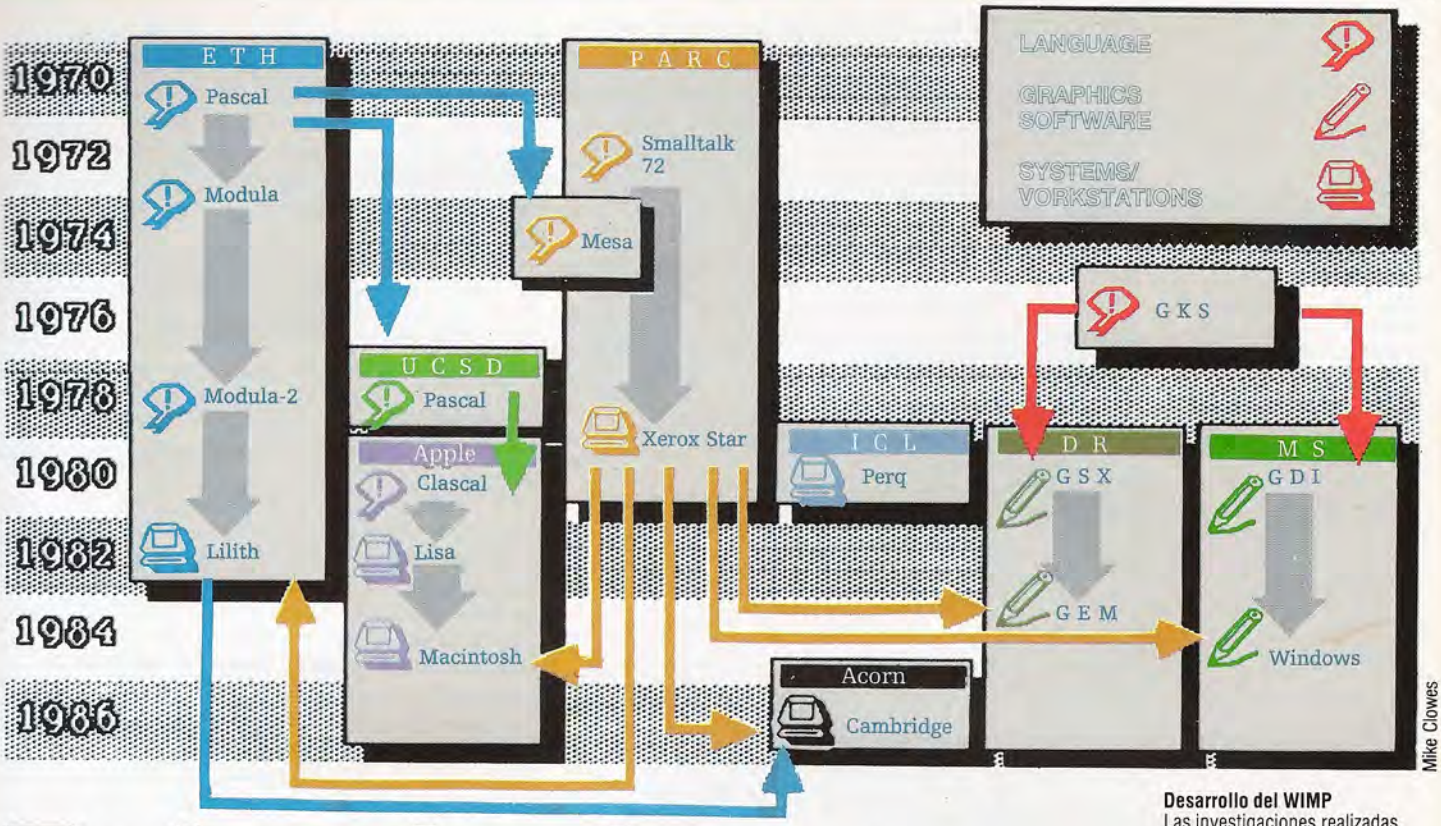
Realización gráfica: Luis F. Balaguer

© 1983 Orbis Publishing Ltd., London
© 1984 Editorial Delta, S.A., Barcelona
ISBN: 84-7596-183-6 (tomo 10) 84-85822-82-X (obra completa)
Depósito legal: B. 52/1984
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona)
Impreso en España - Printed in Spain - 1985

mi COMPUTER

VOLUMEN 10

Editorial Delta, S.A.



Mike Clowes

Desarrollo del WIMP
Las investigaciones realizadas en el PARC Xerox tuvieron influencia en muchas áreas diferentes del desarrollo de lenguajes y de hardware. Niklaus Wirth estuvo un año en el PARC y los resultados se pueden apreciar claramente en este caso en su posterior desarrollo del PASCAL y el MODULA-2. Muchos de los productos primeros están todavía disponibles. Por ejemplo, GKS (Graphics Kernel System), que proporciona un sistema de programación de gráficos independientes de la máquina, se ha convertido en una aplicación compleja (y onerosa), aunque continúa limitada a los usuarios y la investigación CAD-CAM. En el otro extremo de la escala, el GSX de Digital Research, una ampliación del sistema operativo, ha aparecido en el procesador de textos Amstrad PCW 8256, empaquetado con CP/M. Las abreviaturas utilizadas son las siguientes:

ETH: Elektro Technische Hochschule (profesor Niklaus Wirth)
PARC: Centro de investigación de Xerox en Palo Alto
UCSD: Universidad de California en San Diego
GKS: Graphics Kernel System (estandarizado en 1977)
DR: Digital Research Ltd
GSX: Graphics System Extension (DR)
GEM: Graphics Environment Manager (DR)
MS: Microsoft Corporation
GDI: Graphics Device Interface (MS)

Todo sobre los "blitters"

Proseguimos nuestro análisis de los antecedentes del desarrollo del GEM (Graphics Environment Manager) de Digital Research

El SMALLTALK y sus sucesores GEM, Macintosh y MS-Windows suponían grandes necesidades de hardware y, como vimos en el capítulo anterior, fueron los investigadores del PARC de Xerox quienes asumieron la tarea de desarrollar máquinas que fueran capaces de ejecutar los nuevos sistemas.

El Xerox Star se lanzó en 1980 y se servía en gran medida de la investigación de Xerox en el campo de la MMI (*man-machine interface*: interface hombre-máquina) y en el entorno SMALLTALK orientado hacia el objeto. El Star utilizaba un ratón de tres botones, con ventanas e iconos en una pantalla de mapa de bits en alta resolución. Este enfoque radical a la informática, junto con la poca disponibilidad de software y el costo de la máquina, determinaron que no fuera un éxito comercial.

No obstante, sí sirvió para lograr que la gente tomara conciencia de las posibilidades de los WIMP. Por ejemplo, en Gran Bretaña, ICL produjo la estación de trabajo Perq, tomando prestadas muchas de las características del Star. El Perq estaba dirigido al mundo científico y académico, y hacía hincapié en puntos tales como sus gráficos CAD y la capacidad de mezclar texto con gráficos. El sistema incluía una pantalla monocromática de alta definición, ventanas, menú y punteros de barras, y costaba alrededor de £10 000. Tuvo un moderado éxito comercial, vendiéndose en el mercado británico más de 2 500 unidades, si bien ello se debió en gran parte a su compatibilidad para la conexión en red con ordenadores centrales ICL. El Perq no fue tan radical como el Star, ¡de allí el éxito obtenido!

No fue hasta 1983, cuando se lanzó el Apple Lisa en el mercado mundial, que la tecnología WIMP atrajo realmente la atención de la industria informática. Comercializada con gran dinamismo y a un precio inferior a los 10 000 dólares (1 600 000 pts, aproximadamente), la máquina despertó un enorme interés. Sin embargo, se la criticó por su lento acceso a disco y su falta de software, y comercialmente se vio deslucida por el lanzamiento del IBM PC. Aunque rebautizada como Macintosh XL y experimentando una reducción de precio, su producción se suspendió en 1985.

Para implementar gráficos eficaces para iconos, menús y ventanas, es esencial una visualización por mapa de bits de gran velocidad. En el corazón del GEM, o de cualquier *kernel* de gráficos similar, debe haber un método muy potente para mover las imágenes a través de la pantalla. Aunque SMALLTALK era un sistema interpretado (no compilado),

los algoritmos gráficos debían ser sumamente eficaces. Incluso sin considerar ninguna aplicación basada en gráficos, la tarea de gestionar las ventanas superpuestas y mantener actualizados sus cambiantes contenidos era una labor formidable. En este punto, una lentitud de respuesta incidiría en la "amabilidad" del sistema en su totalidad.

Además, las exigencias de ejecutar varios programas o tareas de forma concurrente (si bien simuladas mediante división de tiempo en una única CPU) hacían que el tiempo dedicado a manipular la pantalla debiera reducirse a la mínima expresión. Los potentes procesadores modernos de 16/32 bits, como el Motorola 68000 (utilizado en el Macintosh, Atari y Amiga) pueden alcanzar velocidades de alrededor de 2 mips (millones de instrucciones por segundo), pero la cantidad de código necesario para cuidar de la pantalla impone grandes demandas aun en estos procesadores. Para satisfacer las demandas de los sistemas operativos se han tenido que desarrollar nuevas técnicas.

La única forma de alcanzar un rendimiento eficaz es utilizar algoritmos "tejidos" en el firmware o chips especiales diseñados para llevar a cabo la manipulación adecuada de bits al recibir una instrucción. La CPU principal queda, entonces, libre para continuar con el cálculo y no es necesario escatimarle tiempo para manejar la VDU. Quizá el mejor ejemplo de este enfoque sea el Commodore Amiga.

El Amiga posee un 68000 equipado con tres chips periféricos exclusivos que se ocupan de canales de E/S. Estos dispositivos (denominados Portia, Agnes y Daphne) son responsables de gran parte del impresionante rendimiento de la máquina. Una visualización de alta resolución, por ejemplo, se puede actualizar muchas veces por segundo (con sonido de acompañamiento) para producir un dibujo animado, mientras se utiliza apenas el 9% del tiempo del procesador principal.

Si bien las modernas técnicas de diseño de chips han contribuido enormemente a la facilidad con que se pueden implementar tales chips para periféricos, el diseño de los algoritmos propiamente dichos también tuvo sus orígenes en los primeros trabajos realizados por Xerox. Al algoritmo SMALLTALK se lo llamó BITBLT (BIT BL*ock* T*ransfer*: transferencia de bloques de bits). En ocasiones se suele aludir al mismo coloquialmente como *bit-blit* o *blitter*. Al manipular imágenes, el BITBLT no las visualiza una y otra vez en la pantalla utilizando algoritmos para alterar la memoria de pantalla. En cambio, opera sobre la memoria de pantalla de forma directa e independientemente del procesador principal, explorando las posiciones de memoria de forma muy similar a la de un haz de electrones al barrer la pantalla de una VDU.

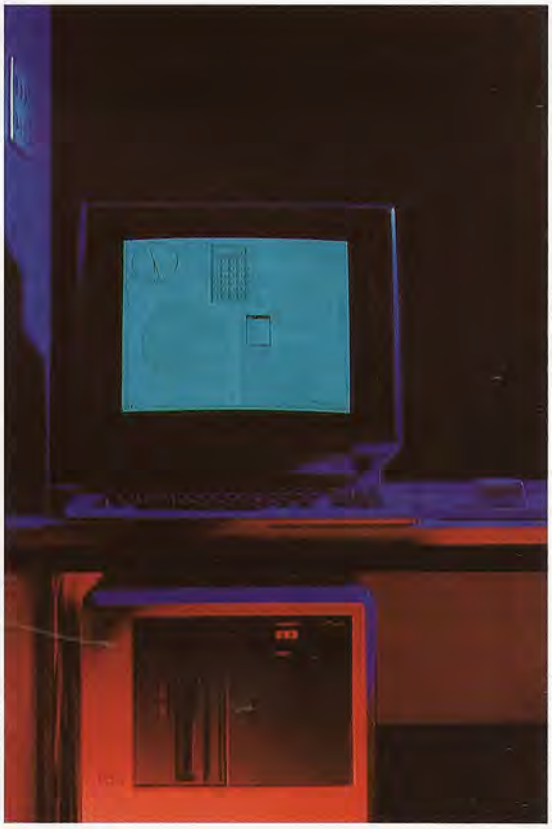
El BITBLT consigue su gran velocidad no sólo por explorar la memoria de este modo, sino también por el hecho de manipular la memoria de pantalla a nivel de bytes en lugar de bit a bit. Esto podría parecer una limitación seria; por ejemplo, ¿cómo alteramos los pixels de mapa de bits individuales si sólo podemos manipularlos en grupos de ocho? Y cuando deseamos desplazar una imagen a través de un fondo estático, ¿cómo restablecemos la imagen sobre la que previamente se ha escrito?

Los *blitters* resuelven este problema de un solo golpe utilizando operaciones lógicas binarias de gran velocidad: operando mediante AND, OR y XOR los bytes de la memoria de pantalla. El algoritmo puede generar una serie de máscaras de bytes que se superponen a la visualización existente. Estas operaciones lógicas son sumamente veloces y XOR, en particular, posee la ventaja de ser capaz de restaurar un fondo previamente sobreescrito.

La combinación de la tecnología de transferencia de bloques de bits, veloces procesadores de 16/32 bits como el Motorola 68000, y el moderno diseño de chips para dispositivos periféricos ha allanado el camino para los sistemas WIMP asequibles y eficaces.

Cortesía de Rank Xerox

La estrella del espectáculo
El Xerox Star ofrece numerosas facilidades desarrolladas a partir de los conceptos SMALLTALK originales. El sistema ofrece facilidades para gráficos y texto de fuentes múltiples y puntos múltiples, todo ello operando bajo un entorno WIMP



Marcus Wilson-Smith

Características de gráficos
El Perq, fabricado por ICL, fue el primer ordenador producido en Gran Bretaña que aplicó los conceptos pioneros de Xerox. La pantalla de alta resolución ofrecía un sobresaliente potencial de gráficos monocromáticos, y la máquina se vendió básicamente a establecimientos de investigación científica. Uno de los primeros usuarios fue la unidad de inteligencia artificial de la Universidad de Edimburgo, que utilizó las facilidades gráficas del Perq para (entre otras tareas) experimentos en el campo de la visión del robot

Apostar en el casino

Al finalizar esta serie examinaremos los diversos sistemas que se suelen aplicar para aumentar las posibilidades del jugador

Cuando alguien le habla de un sistema de apuestas infalible, es comprensible que usted se muestre escéptico. Sin embargo, existen unos pocos sistemas que funcionan bien en las condiciones para las cuales fueron diseñados. Uno de tales sistemas lo inventó el matemático y filósofo francés Jean Le Rond d'Alembert (1717-1783), que colaboró con Diderot en la redacción de la *Enciclopedia*. Su sistema exige que las posibilidades reales a favor de usted sean del 50 % o superiores. En general esto no es así, pero si el apostador posee un buen programa de predicción puede resultar valedero. El propio D'Alembert utilizó su sistema de apuestas para amasar una considerable suma en los días en que las ruedas de la ruleta eran imparciales y no había ventaja para la casa.

En la actualidad, la ruleta es peor que ineficaz (a menos que usted encuentre una bola que se incline a su favor), pero si su programa de predicción está dando algo más que un reembolso a la par, este método puede serle rentable. Funciona así:

- Empezar con una apuesta de 5 unidades.
- Tras cada apuesta ganada reducir la apuesta en una unidad.
- Tras cada apuesta perdida aumentar la apuesta en una unidad.
- Si llega a quedar en cero, recomenzar con 5 unidades.

La eficacia del método es muy compleja, pero se puede demostrar empíricamente con un ordenador. Nosotros hemos escrito un programa de demostración para darle una idea de lo que cabe esperar de tal sistema. Esto es bastante fácil de hacer, porque el BASIC posee un generador de números aleatorios incorporado (la función RND).

Con este programa usted puede variar la probabilidad de ganar una apuesta, y usarla para ver cómo incide ello en el rédito. Asimismo, puede adaptarlo para probar otras estrategias de apuestas ideadas por usted mismo, sin perder ni un céntimo. De hecho, ¡la gran ventaja de las simulaciones como ésta es que lo mantienen a uno al margen del local de apuestas! Proporcionan el escenario para probar sus teorías y descartarlas sin que su bolsillo corra ningún riesgo. Y ocasionalmente también podrían confirmar que usted ha descubierto un plan que vale la pena.

La esencia del método de D'Alembert es que magnifica las expectativas de ganancias. Sin embargo, si las posibilidades están en su contra, también magnifica sus pérdidas. En otro juego de casino (el *blackjack*) se puede utilizar una estrategia diferente para revertir la ventaja de la casa a su favor. Usted simplemente debe apostar fuerte en las manos buenas y apostar el mínimo permitido en las malas. Como es natural, para poder hacerlo tiene que distinguir entre manos buenas y malas, convirtiéndose en un *calculador* o *contador*. Es decir, debe memorizar los naipes que ya se han jugado.

Imagebank

La fiebre del oro
Los casinos atraen a muchos tipos de jugadores diferentes, pero los apostadores serios constituyen una minoría. A pesar del margen de beneficios de que disfruta la casa en todos los establecimientos de apuestas, se han desarrollado algunos sistemas de predicción para maximizar las ganancias del apostador. Lamentablemente, la mayoría de los sistemas requieren un patrón de apuestas que el personal del casino puede reconocer con facilidad, y un apostador sistemático podría encontrarse con que se le invitara amablemente a abandonar la mesa

Todos los sistemas contadores del *blackjack* provienen del que utilizara Edward Thorp a principios de los años sesenta. Este hombre ganó una fortuna en Las Vegas y escribió un libro sobre sus métodos, cuyo título es *Beat the dealer*. Si bien el empleo cuidadoso de su sistema inclina las posibilidades a su favor, posee dos inconvenientes principales:

1. El coeficiente esperado de rédito sobre el movimiento total es del orden del 0,5 %.
2. El personal de los casinos está entrenado para detectar a los contadores.

El primer problema proviene del segundo. Dado que el sistema se puede detectar fácilmente, las versiones sofisticadas del sistema exigen un comportamiento más sutil, lo que limita las ganancias potenciales. Esto significa que si apuesta 1 000 pts. en una hora sólo puede esperar ganar unas 40 pts.

El segundo problema significa que, tras manifestar durante cierto tiempo el perfil de apuestas de un contador eficiente, es probable que el gerente del casino lo invite a una cena por cuenta de la casa. Si usted se niega, es casi seguro que un robusto empleado del club lo "escolte" hacia la salida del local. En consecuencia, abandonemos la atmósfera saturada de humo del casino e introduzcámonos en un área que hasta ahora, y sorprendentemente, no ha sido tenida en cuenta: la aplicación de las técnicas de inteligencia artificial (AI) en las apuestas.

La rueda de la fortuna

El juego de la ruleta es un ejemplo obvio de la estimación de las probabilidades matemáticas a favor de la casa. En la versión norteamericana del juego, la mesa tiene el trazado que vemos aquí y los jugadores pueden colocar fichas en diversos lugares para apostar a un solo número, a grupos de números o a un color determinado. La auténtica probabilidad de que salga un número determinado es de 36 a 1, pero en realidad la casa da posibilidades de 35 a 1 concediéndose a sí misma, por término medio, un margen de beneficios del 2,7 %. En apuestas de 50 % (impar/par, rojo/negro, 1-18/19-36), la casa devuelve la mitad de la apuesta jugada si sale el cero, reduciendo su margen de beneficios al 1,35 %.

Kevin Jones

Uno de los desarrollos más interesantes a que ha dado lugar la AI en la última década es el sistema experto. Como hemos visto en series anteriores, un sistema experto comparte muchas de las características de un experto humano, incluyendo la capacidad de manejar inferencias inciertas en situaciones mal definidas mediante la lógica formal o el razonamiento basado en la probabilidad.

Cuando se está desarrollando un sistema experto para usar en prospecciones de minerales preciosos o diagnosticar enfermedades, lo primero que hacen los diseñadores es procurarse la ayuda de un especialista humano en la materia que pueda realizar la tarea con un elevado nivel de destreza. Luego se codifica el conocimiento del experto y se lo refina para que lo utilice el ordenador. El mismo enfoque se puede utilizar, por ejemplo, con las carreras de caballos.

Un investigador de la Carnegie-Mellon University de Pittsburgh, Stephen F. Smith, ha desarrollado un programa de aprendizaje llamado LS-1 que aprendió a jugar al póquer cerrado.

El póquer cerrado es un juego para dos o más jugadores. A cada jugador se le reparten cinco naipes. Los jugadores, por turno, tienen la opción de pedir a los demás que muestren su mano, apostar o abandonar. Cuando se realiza la primera vuelta, cada jugador puede sustituir hasta tres naipes de su mano con naipes nuevos (no vistos) de la baraja. Un abandono da por terminada la actual ronda de juego. Una apuesta consiste en colocar algo de dinero, al menos tanto como la apuesta anterior, en un "bote" central. Si se realiza una segunda vuelta, se muestran todas las manos y el jugador que tenga la mano de mayor jerarquía se queda con todo el dinero del bote.

El LS-1 es un sistema de aprendizaje que emplea un algoritmo de adaptación evolutiva o *genética*. Su bucle principal opera del siguiente modo:

1. Generar al azar unas reglas de partida.
2. Evaluar todas las reglas y si la puntuación media es suficientemente alta, parar y visualizarlas.
3. De lo contrario, calcular para cada regla su probabilidad de selección `p=e/E`, donde `e` es su puntuación individual y `E` la puntuación global de todas las reglas.
4. Generar la siguiente población mediante una selección basada en las probabilidades calculadas en 3 y aplicando ciertos operadores genéticos; luego repetir desde el paso 2.

Cada pasada por este bucle corresponde a una única generación.

Los operadores genéticos mencionados anteriormente son cruce, inversión y mutación. Intentan simular, de forma simplificada, lo que sucede en el proceso de la evolución. El cruce es un procedimiento de apareamiento en virtud del cual se combina la información de dos estructuras paternas para conformar un nuevo *descendiente*, es decir, una regla candidata a la comprobación. La inversión reordena la información de la regla. La mutación introduce cambios caprichosos.

Es evidente que esta competencia entre estructuras de reglas tiene mucho en común con la competencia entre organismos naturales. Las reglas que tienen las mejores posibilidades de dejar descendientes son aquellas que se desempeñan mejor en la tarea.

En el caso del póquer cerrado, los resultados fueron impresionantes. Se enfrentó al LS-1 con un programa de póquer hecho a mano, puesto que ningún oponente hubiera tenido paciencia suficiente para jugar más de 40 000 rondas. Para cada decisión de apuesta tuvo cuatro acciones alternativas: apuesta fuerte, apuesta baja, verlas o abandono. Comenzando a partir de nada, con un conjunto de reglas iniciales aleatorias, LS-1 demostró ser tan fuerte que hubo que reprogramar al ordenador oponente para convertirlo en un rival más preparado. El programa modificado resultó una prueba más dura, pero al cabo de varios miles de rondas el LS-1 le ganó nueve de cada diez manos. Cuando terminó, fue uno de los jugadores de póquer no humanos más fuertes del mundo.

Otro estudio piloto, también con regustos evolutivos, se llevó a cabo utilizando el sistema de aprendizaje BEAGLE (*Biological Evolutionary Algorithm Generating Logical Expressions*: algoritmo de evolución biológica generador de expresiones lógicas) sobre una muestra de datos de carreras de caballos celebradas en Gran Bretaña en 1982.

Los datos comprendían a 153 caballos a partir de 51 hándicaps de todas las edades con 10 corredores o menos. En el pronóstico de apuestas de cada carrera sólo se tuvieron en cuenta los tres mejores caballos. La base de datos se dividió en dos partes: se utilizaron 99 registros como conjunto de entrenamiento (para formar las reglas) y los 54 restantes se utilizaron como datos de prueba (para ver si las reglas se generalizaban en ejemplos nuevos). Cada caballo se midió sobre 18 variables, incluyendo:

VELOCIDAD	cifra de velocidad para el caballo
ESCALADO	Si la cifra de velocidad es la más alta, casi la más alta, etc.
FC	posición en el pronóstico
ÚLTIMA	lugar en última salida
PENÚLT.	lugar en penúltima salida
ANTEPE-NÚLTIMA	lugar en antepenúltima salida
DIAS	días desde la última carrera
SPOTFORM	evaluación según el *Daily Mirror*
BIN	mejor promedio de dist. reciente
GOING	estado de la pista
WT	peso del jockey

Además, se usó la variable GANA para registrar si el caballo ganó.

El programa BEAGLE utiliza un esquema de aprendizaje evolutivo similar al que describimos en la serie dedicada a la inteligencia artificial (p. 1802). Al proporcionársele los datos de las carreras de caballos, salió con reglas tales como (VELOCIDAD>60) y (PESO> (VELOCIDAD * 2.18)), que entre ambas ayudan a distinguir entre probables ganadores y perdedores. Estas reglas se utilizan conjuntamente para dar una "rúbrica" o "huella digital". Es decir, si las reglas 1 y 3 son verdaderas y las reglas 2 y 4 falsas, la rúbrica sería el número binario 0101 (porque el sistema trabaja hacia atrás) o 5 decimal. Este índice 5 señala la posición 5 de una tabla en la cual se acumula información sobre esa combinación de valores de reglas.

Cuando el módulo LEAF (*Logical Evaluator And Forecaster*: evaluador y pronosticador lógico) aplicó las reglas generadas por ordenador a los datos no vistos, las mismas resultaron correctas el 80 % de las veces. Por supuesto, usted puede acertar el 72 % de las veces, simplemente diciendo "no", porque la mayoría de los caballos pierden. Pero un examen de los resultados impresos demuestra que las reglas actuaron como un sistema de filtro muy eficaz. El BEAGLE sólo pronosticó cuatro ganadores: todos ellos ganaron. Ésta es una buena señal, porque la selectividad es la esencia de la apuesta científica. (Uno no puede esperar pronosticar todas las carreras; lo importante es esperar hasta hallar "un caballo de carreras entre burros" y entonces dar el golpe.) Además, sus 13 mejores pronósticos incluyeron 10 ganadores y sólo tres perdedores. Dicho en otras palabras, los nueve caballos del grupo de probabilidad 0,4854 lo hicieron mejor de lo que se esperaba, con seis ganadores.

Analizando esto desde otro ángulo, la cantidad de ganadores entre los 41 peores fue de sólo cinco. La categoría inferior, los "sin esperanzas", contuvo sólo un error, de modo que es eficiente para filtrar los perdedores. (Observe que el LEAF coloca signos de interrogación en el listado de pronósticos para señalar las predicciones efectuadas sobre la base de muestras pequeñas.)

En líneas generales, entonces, existe la evidencia de que las técnicas de aprendizaje desarrolladas por la investigación en AI pueden ser rentables en el hipódromo. Esto es apenas un estudio preliminar, pero los resultados son muy alentadores. En la literatura dedicada al estudio de la AI hay gran riqueza de técnicas a la espera de ser explotadas.

Simulador de apuestas

```
 10 REM ******************************
 15 REM ** METODO DE D'ALEMBERT: ******
 20 REM ******************************
 75 @%=4
100 PRINT"Simulador de apuestas:"
101 REPEAT
110  PRINT
120  PRINT "Por favor dar probabilidades de exito ";
122  INPUT PS
125  UNTIL PS > 0 AND PS < =1
130 INPUT "Cuantos ensayos ",T
140 N=0
150 GANANCIAS=0: PERDIDAS =0
155 ULTIMA=0:ST=-99
160 W=0
190 REM -- Bucle principal:
200 REPEAT
202  N=N+1
210  GOSUB 1000: REM calcular apuesta
212  PRINT N;" ";
220  IF RND(1) < = PS THEN S=1 ELSE S=0
230  IF S THEN GOSUB 1500 ELSE GOSUB 1600
240  ULTIMA=S
250  UNTIL N> = T
300 PRINT: PRINT "Total ganado = "; GANANCIAS-PERDIDAS
330 PRINT "Proporcion de exitos: ";W/T
360 END
999 :
1000 REM -- Rutina decision apuesta:
1010 REM se puede alterar para experimentar:
1020 IF ULTIMA THEN ST=ST-1 ELSE ST=ST+1
1030 IF ST < 1 THEN ST=5
1040 RETURN
1050 :
1500 REM -- Rutina exito:
1510 GANANCIAS=GANANCIAS+ST
1520 PRINT "Gano ";ST;TAB(15);"Ganancias= ";GANANCIAS-PERDIDAS
1525 W=W+1
1550 RETURN
1560 :
1600 REM -- Rutina fracaso:
1610 PERDIDAS=PERDIDAS+ST
1620 PRINT "Perdido ";ST;TAB(15);"Ganancias = ";GANANCIAS-PERDIDAS
1650 RETURN
1660 :
```

Hoja completa

Finalmente, crearemos rutinas que nos permitan guardar hojas completas de datos y fórmulas

La última adición a nuestro programa de hoja electrónica es la implementación de las rutinas para guardar y cargar, que nos permiten crear registros permanentes de datos o fórmulas en disco o cinta. Puesto que cada una de las cuatro máquinas para las que está diseñada la hoja electrónica trata los archivos de forma diferente, incluimos listados separados para la gama Amstrad, el Commodore 64, el BBC Micro y el Sinclair Spectrum.

En las cuatro versiones las rutinas cargar y guardar empiezan en la línea 7000, visualizando un corto submenú para las diversas opciones disponibles. Éstas son:

1: Cargar una hoja de DATOS
2: Cargar una matriz de FORMULAS
3: Guardar una hoja de DATOS
4: Cargar una hoja de FORMULAS
"X": Para salir

A partir del menú de opciones, usted puede ver que el programa le permite guardar o cargar independientemente datos o fórmulas. Esta característica es especialmente útil porque le permite crear una plantilla de fórmulas para una tarea determinada (supongamos, calcular las devoluciones del IVA), guardarla en cinta o disco o luego cargarla cada vez que la necesite, para poder trabajar cada vez con un nuevo conjunto de datos. Por el contrario, quizá tenga una tarea que requiera realizar varias operaciones diferentes sobre el mismo conjunto de datos, en cuyo caso el conjunto original de datos se puede almacenar y volver a cargar cuando así se necesite para cada nuevo cálculo.

Implementar esta facilidad para cargar y guardar fórmulas o datos individuales en realidad es muy fácil. Usted recordará que todos los valores de datos para la hoja están retenidos en la matriz numérica bidimensional M(,), y que las correspondientes fórmulas de celdas están retenidas en una matriz alfanumérica unidimensional, F$(). Todo cuanto habremos de hacer es guardar la matriz adecuada como un archivo secuencial. Para volver a cargarla, sólo debemos leer nuevamente el archivo en la matriz de la cual se tomó, perdiendo, como es lógico, el contenido que la matriz tenga en ese momento.

Comencemos nuestro análisis de cómo guarda el programa las matrices examinando las versiones para el Amstrad CPC y el BBC Micro. Estas máquinas emplean una sintaxis similar para abrir y cerrar archivos. OPENIN abre un archivo para una entrada y OPENOUT abre un archivo sobre el cual se ha de escribir, pero mientras que el sistema de tratamiento de archivos del BBC Micro destina un número de canal como parte de la operación de apertura del archivo, el Amstrad sólo permite utilizar un único canal (# 9) para sus operaciones PRINT e INPUT sobre archivos.

Las rutinas para guardar simplemente recorren la matriz, elemento a elemento, utilizando un bucle FOR...NEXT (o, en el caso de la matriz bidimensional M(,), un par de bucles anidados) para grabarlos en el archivo secuencial. La rutina para cargar no es más que el mismo proceso pero en sentido inverso.

Si bien el Spectrum no soporta archivos secuenciales, le permitirá simular el proceso guardando matrices numéricas o alfanuméricas en cinta empleando la sintaxis:

```
SAVE I$ DATA F$()
```

donde I$ es el nombre del archivo y F$(), la matriz a guardar.

Dado que todo cuanto necesitamos hacer es guardar matrices, este sistema resulta ideal para nuestras aplicaciones.

La versión Commodore

Por último, llegamos a la versión para el Commodore 64. Al desarrollar el programa, quizá le parezca que hay un error en el BASIC Commodore. Un archivo secuencial se puede preparar a partir de una matriz numérica y volver a leerlo sin ninguna dificultad.

El problema surge al escribir los elementos de una matriz alfanumérica en un archivo.

Cuando se DIMensiona una matriz alfanumérica, todos los elementos se establecen inicialmente en series nulas, representadas en el sistema Commodore mediante CHR$(0). Si imaginamos que utilizamos la hoja electrónica para programar algunas fórmulas, pero dejando algunas celdas sin fórmulas, los elementos correspondientes de la matriz de fórmulas continuarán nulos y cuando se guarde F$() se escribirán en el archivo como CHR$(0). El problema se plantea cuando se vuelve a leer el archivo mediante INPUT #.

Si bien esta instrucción leerá cualquier fórmula que se haya establecido antes de guardarla, no volverá a leer aquellos elementos de la matriz que fueran nulos y el sistema quedará colgado. Usted puede sortear esta "característica" comprobando cada elemento de la matriz alfanumérica antes de escribirlo en el archivo, y si el elemento es cero, puede escribir CHR$(1) en el lugar del CHR$(0) que normalmente el Commodore 64 escribiría de forma automática. Aunque CHR$(1) no representa ningún carácter, sortea el fracaso del INPUT # en reconocer CHR$(0) y, por lo tanto, todo lo que hemos de hacer es modificar la rutina que vuelve a leer el archivo en F$(). Esto significa que los elementos leídos como CHR$(1) se establecen como la serie nula (ver líneas 7425 y 7230 de la versión para el Commodore).

Las rutinas que aparecen en el listado para el Commodore 64 son para sistemas de disco. Para

sistemas de cinta, introduzca los siguientes cambios:

```
7110 GOSUB 7500:OPEN 1,1,0,I$
7219 GOSUB 7500:OPEN 1,1,0,I$
7310 GOSUB 7500:OPEN 1,1,1,I$
7410 GOSUB 7500:OPEN 1,1,1,I$
```

Y así llegamos al final de nuestra serie sobre hojas electrónicas. Al comenzarla habíamos esbozado algunas de las limitaciones de nuestro programa. Por ejemplo, el ancho de la celda se fija en cinco caracteres, en una celda sólo se pueden entrar datos numéricos y se ha simplificado la detección de errores en las entradas para acortar el listado. Como un proyecto para el futuro, quizá le interese añadir y ampliar algunas de las características que hemos presentado.

HELP!

La subrutina de la línea 6000 proporciona una pantalla de ayuda que indica las teclas que se han de pulsar para obtener las diversas funciones. Lo que no le dice es para qué sirve cada función, de modo que nos referiremos a cada una de las funciones de la hoja describiendo su funcionamiento.

ENTRAR FORMULA NUEVA EN CELDA: Se puede entrar en la celda de la posición actual del cursor. Las fórmulas realizan operaciones aritméticas con constantes y valores de celdas (a las que se alude por nombre, como A1, B12) y se utilizan para dirigir la función CALCULAR de la hoja electrónica.

ALMACENAR HOJA ACTUAL: Los datos se pueden almacenar temporalmente dentro del programa. Esta facilidad se utiliza para poner a prueba estrategias de cálculo diferentes en la misma hoja de datos, o usar como mecanismo de seguridad para que los datos no se corrompan a consecuencia de una dirección errónea de la función CALCULAR.

TOMAR HOJA ALMACENADA: Recupera una hoja de datos almacenada previamente.

CALCULAR: Utiliza las fórmulas y los datos entrados en la hoja para dirigir sus operaciones, colocando los resultados de los cálculos en celdas adecuadas.

REPRODUCIR: Es útil poder reproducir una fórmula de una celda determinada a través de una fila o a lo largo de una columna. La sintaxis requerida es N1 (N2-N3), donde N1 es el nombre de la celda cuya fórmula se ha de reproducir, y N2 y N3 son los límites entre los cuales ha de tener lugar la reproducción. La rutina selecciona automáticamente la reproducción en fila o en columna.

BORRAR HOJA: Restablece a cero los datos de todas las celdas.

TAB CURSOR: El cursor de la hoja electrónica se puede desplazar mediante el uso de las teclas del cursor, pero en virtud de esta función se pueden efectuar saltos a una celda determinada. Entre el nombre de la celda a la cual usted desea ir.

RUTINAS CARGAR/GUARDAR: Selecciona el submenú cargar/guardar, que a su vez le permite guardar los datos o fórmulas en curso en disco o cinta, y cargar los almacenados con anterioridad.

Pantallas HELP y tratamiento de archivos

Commodore 64:

```
6000 REM ***** PANTALLA DE AYUDA *****
6010 PRINT CHR$(147):PRINT
6020 PRINT:PRINT "                F1 - IMPRIME ESTA
     PANTALLA HELP "
6030 PRINT:PRINT "                F2 - ENTRAR NUEVA
     FORMULA EN CELDA "
6040 PRINT:PRINT "                F3 - ALMACENAR
     HOJA ACTUAL "
6050 PRINT:PRINT "                F4 - TOMAR HOJA
     ALMACENADA "
6060 PRINT:PRINT "                F5 - CALCULAR
     HOJA "
6070 PRINT:PRINT "                F6 - BORRAR
     HOJA "
6080 PRINT:PRINT "                F7 - REPRODUCIR
     FORMULA "
6090 PRINT:PRINT "                F8 - CARGAR/GUARDAR
     HOJAS "
6100 PRINT:PRINT "                'G' - PARA
     TABULAR CURSOR
6110 GET A$:IF A$="" THEN 6110
6120 GOSUB 1000:GOSUB 1700:RETURN
7000 REM ***** RUTINAS CARGAR/GUARDAR *****
7010 PRINT CHR$(147):PRINT:PRINT
7020 PRINT:PRINT "                F1 - CARGAR HOJA
     DATOS "
7030 PRINT:PRINT "                F3 - CARGAR
     MATRIZ FORMULAS "
7040 PRINT:PRINT "                F5 - GUARDAR HOJA
     DATOS "
7050 PRINT:PRINT "                F7 - GUARDAR
     MATRIZ FORMULAS "
7055 PRINT:PRINT "                'X' - PARA SALIR "
7056 PRINT:PRINT:PRINT
7060 GET A$:IF A$="" THEN 7060
7070 IF A$=CHR$(133) THEN 7100
7080 IF A$=CHR$(134) THEN 7200
7090 IF A$=CHR$(135) THEN 7300
7095 IF A$=CHR$(136) THEN 7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB 1700:
     RETURN
7100 REM **** CARGAR DATOS ****
7110 GOSUB 7500:I$="0:"+I$+",S,R":OPEN
     2,8,2,I$
7120 FOR I=1 TO 15:FOR J=1 TO 15
7130 INPUT #2,M(I,J)
7140 NEXT J,I
7150 CLOSE2
7160 GOTO 7010
7200 REM **** CARGAR MATRIZ FORMULAS ****
7210 GOSUB 7500:I$="0:"+I$+      ":OPEN
     ,S,R":OPEN 2,8,2,I$
7220 FOR I=1 TO 255
7230 INPUT # 2,F$(I):IF F$(1)=CHR$(1) THEN
     F$(I)=""
7240 NEXT 1
7250 CLOSE2
7260 GOTO 7010
7300 REM **** GUARDAR DATOS ****
7310 GOSUB 7500:I$="0:"+I$+",S,W":OPEN 2,8,2,I$
7320 FOR I=1 TO 15:FOR J=1 TO 15
7330 PRINT # 2,M(I,J)
7340 NEXT J,I
7350 CLOSE2
7360 GOTO 7010
7400 REM **** GUARDAR MATRIZ FORMULAS ****
7410 GOSUB 7500:I$="0:"+I$+ ",S,W":OPEN
     2,8,2,I$
7420 FOR I=1 TO 255
7425 IF F$(I)="" THEN PRINT # 2,CHR$(1):
     GOTO 7440
7430 PRINT#2,F$(I)
7440 NEXT I
7450 CLOSE2
7460 GOTO 7010
7500 INPUT "ENTRAR NOMBRE ARCHIVO:";I$
7510 RETURN
```

F1: Imprime pantalla HELP
F2: Entrar nueva fórmula en celda
F3: Almacenar hoja actual
F4: Tomar hoja almacenada
F5: Calcular hoja
F6: Borrar hoja
F7: Reproducir fórmula
F8: Rutinas cargar/guardar
"G": Tabular cursor

BBC Micro:

"H": Pantalla HELP
"F": Entrar nueva fórmula en celda
"S": Almacenar hoja actual
"G": Tomar hoja almacenada
"C": Calcular hoja
"Z": Borrar hoja
"R": Reproducir fórmula
"T": Tabular cursor
"D" Rutinas cargar/guardar

```
1189 IF A$="D" THEN GOSUB 7000:
     REM RUTINAS CARGAR
     GUARDAR
6000 REM **** PANTALLA DE AYUDA ****
6020 CLS:PRINT TAB(6,2)"H – IMPRIMIR ESTA
     PANTALLA HELP"
6030 PRINT TAB (6,4)"F – ENTRAR NUEVA
     FORMULA EN CELDA"
6040 PRINT TAB (6,6)"S – ALMACENAR HOJA
     ACTUAL "
6050 PRINT TAB (6,8)"G – TOMAR HOJA
     ALMACENADA"
6060 PRINT TAB (6,10)"C – CALCULAR HOJA"
6070 PRINT TAB (6,12)"Z – BORRAR HOJA"
6080 PRINT TAB (6,14)"R – REPRODUCIR
     FORMULA"
6090 PRINT TAB (6,16)"T – TABULAR A NUEVA
     POS/ CURSOR"
6100 PRINT TAB (6,18)"D – RUTINAS
     CARGAR/GUARDAR"
6110 A$=GET$:GOSUB 1000:GOSUB
     1700:RETURN
7000 REM *** RUTINAS CARGAR /
     GUARDAR *****
7010 CLS:PRINT TAB (6,4)"1 – CARGAR HOJA
     DATOS"
7030 PRINT TAB (6,6)"2 – CARGAR MATRIZ
     FORMULAS"
7040 PRINT TAB (6,8)"3 – GUARDAR HOJA
     DATOS"
7050 PRINT TAB (6,10)"4 – GUARDAR MATRIZ
     FORMULAS"
7055 PRINT TAB (6,12)"'X' – PARA
     SALIR"
7056 A$=GET$:A=VAL(A$)
7060 IF A > 0 AND A < 5 THEN ON A GOTO
     7100,7200,7300,7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB
     1700:RETURN
7100 REM ***** CARGAR DATOS *****
7110 GOSUB 7500:F=OPENIN I$
7120 FOR I=1 TO 15:FOR J=1
     TO 15:INPUT # F,M(I,J):
     NEXT J,I
7130 CLOSE # F: GOTO 7010
7200 REM **** CARGAR FORMULAS ****
7210 GOSUB 7500:F=OPENIN I$
7220 FOR I=1 TO 255:INPUT # F,F$(I):
     NEXT I
7230 CLOSE # F:GOTO 7010
7300 REM **** GUARDAR DATOS ****
7310 GOSUB 7500:F=OPENOUT I$
7320 FOR I=1 TO 15:FOR J=1
     TO 15:PRINT # F,M(I,J):
     NEXT J,I
7330 CLOSE # F:GOTO 7010
7400 REM **** GUARDAR FORMULAS ****
7410 GOSUB 7500:F=OPENOUT
     I$
7420 FOR I=1 TO 255:INPUT # F,F$(I):
     NEXT I
7430 CLOSE # F:GOTO 7010
7500 INPUT TAB(0,22)"NOMBRE
     ARCHIVO"I$
7510 RETURN
```

Amstrad CPC 464/664:

"H": Pantalla HELP
"F": Entrar nueva fórmula en celda
"S": Almacenar hoja actual
"G": Tomar hoja almacenada
"C": Calcular hoja
"Z": Borrar hoja
"R": Reproducir fórmula
"T": Tabular cursor
"D": Rutinas cargar/guardar

Nota: Pulsar CAPS LOCK antes de la ejecución

```
1175 IF A$="Z" THEN GOSUB 5000:REM
     BORRAR LA HOJA
1177 IF A$="S" THEN GOSUB 5150:REM
     ALMACENAR HOJA
1178 IF A$="H" THEN GOSUB 6000:REM
     PANTALLA AYUDA
1189 IF A$="D" THEN GOSUB 7000:REM
     RUTINAS CARGAR/GUARDAR
6000 REM **** pantalla de ayuda ****
6020 CLS:LOCATE 6,3:PRINT"H – Imprimir esta
     pantalla HELP"
6030 PRINT:PRINT TAB(6)"F – Entrar nueva
     FORMULA"
6040 PRINT:PRINT TAB (6)"S – ALMACENAR
     hoja actual"
6050 PRINT:PRINT TAB(6)"V – Tomar hoja
     ANTERIOR"
6060 PRINT:PRINT TAB(6)"C – CALCULAR hoja"
6070 PRINT:PRINT TAB(6)"Z – BORRAR hoja"
6080 PRINT:PRINT TAB(6)"R = REPRODUCIR
     formula"
6090 PRINT:PRINT TAB(6)"G – IR a una nueva
     celda"
6100 PRINT:PRINT TAB(6)"D – Rutinas
     GUARDAR/CARGAR"
6110 WHILE INKEY$="":WEND
6120 GOSUB 1000:GOSUB 1700:RETURN
7000 REM *** RUTINAS CARGAR/GUARDAR ***
7010 CLS:LOCATE 6,4:PRINT"1 – CARGAR HOJA
     DATOS"
7020 PRINT:PRINT TAB(6)"2 – CARGAR MATRIZ
     FORMULAS"
7030 PRINT:PRINT TAB(6)"3 – GUARDAR
     MATRIZ DATOS"
7040 PRINT:PRINT TAB(6)"4 – GUARDAR
     MATRIZ FORMULAS"
7050 PRINT:PRINT TAB(6)"'X' – PARA SALIR"
7055 A$="":WHILE A$="":A$=INKEY$:WEND
7056 A=VAL(A$)
7060 ON A GOTO 7100,7200,7300,7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB
     1700:RETURN
7100 REM **** CARGAR DATOS ****
7110 GOSUB 7500:OPENIN I$
7120 FOR I=1 TO 15:FOR J=1 TO 15
7130 INPUT # 9,M(I,J)
7140 NEXT J,I
7150 CLOSEIN:GOTO 7010
7200 REM **** CARGAR FORMULAS ****
7210 GOSUB 7500:OPENIN I$
7220 FOR I=1 TO 255
7230 INPUT # 9,F$(I)
7240 NEXT I
7250 CLOSEIN:GOTO 7010
7300 REM **** GUARDAR DATOS ****
7310 GOSUB 7500:OPENOUT I$
7320 FOR I=1 TO 15:FOR J=I TO 15
7330 PRINT # 9,M(I,J)
7340 NEXT J,I
7350 CLOSEOUT:GOTO 7010
7400 REM **** GUARDAR FORMULAS ****
7410 GOSUB 7500:OPENOUT I$
7420 FOR I=1 TO 255
7430 PRINT # 9,F$(I)
7440 NEXT I
7450 CLOSEOUT:GOTO 7010
7500 LOCATE 1,22:INPUT"NOMBRE ARCHIVO";I$
7510 RETURN
```

Sinclair Spectrum:

"H": Imprime pantalla HELP
"E": Entrar datos numéricos
"F": Entrar nueva fórmula
"C": Calcular hoja
"S": Almacenar hoja actual
"G": Tomar hoja almacenada
"Z": Borrar hoja
"R": Reproducir fórmula
"T": Tabular cursor
"D": Rutinas Cargar/guardar

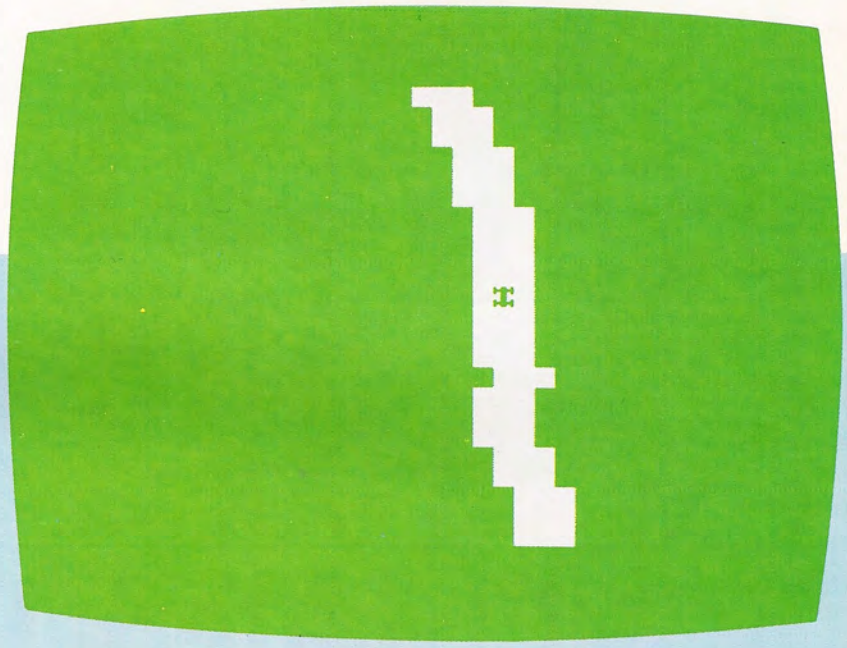
Nota: Pulsar CAPS LOCK antes de la ejecución

```
6000 > REM IMPRIMIR PANTALLA AYUDA
6010 CLS
6020 PRINT : PRINT " H – IMPRIME ESTA
     PANTALLA HELP"
6030 PRINT : PRINT " E – ENTRAR DATOS
     NUMERICOS"
6040 PRINT : PRINT " F – ENTRAR NUEVA
     FORMULA"
6050 PRINT : PRINT " C – CALCULAR
     HOJA"
6060 PRINT : PRINT " S – ALMACENAR HOJA
     ACTUAL"
6070 PRINT : PRINT " G – TOMAR HOJA
     ALMACENADA"
6080 PRINT : PRINT " Z – BORRAR
     HOJA"
6090 PRINT : PRINT " R – REPRODUCIR
     FORMULAS"
6100 PRINT : PRINT " T – TABULAR A NUEVA
     CELDA"
6110 PRINT : PRINT " D – OPCIONES
     CARGAR/GUARDAR"
6120 LET A$=INKEY$: IF A$="" THEN GO TO
     6120
6130 GO SUB 1000: GO SUB 1700:
     GOTO 1100
7000 REM OPCIONES CARGAR/
     GUARDAR
7010 CLS: PRINT
7020 PRINT " OPCIONES CARGAR Y
     GUARDAR"
7030 PRINT
7040 PRINT : PRINT "          1 – CARGAR
     HOJA DATOS"
7045 PRINT : PRINT "          2 – CARGAR
     MATRIZ FORMULAS"
7050 PRINT : PRINT "          3 – GUARDAR
     HOJA DATOS"
7055 PRINT : PRINT "          4 – GUARDAR
     MATRIZ FORMULAS"
7056 PRINT: PRINT "               X – PARA
     SALIR"
7060 LET A$=INKEY$: IF A$="" THEN GO TO
     7060
7070 IF A$="1" THEN GO TO 7100
7080 IF A$="2" THEN GO TO 7200
7090 IF A$="3" THEN GO TO 7300
7095 IF A$="4" THEN GO TO 7400
7096 IF A$="X" THEN GO SUB 1000:GO SUB
     1700: RETURN
7100 REM **** CARGAR DATOS ****
7110 GO SUB 7500
7120 LOAD I$ DATA M()
7130 GO TO 7010
7200 REM **** CARGAR FORMULAS ****
7210 GO SUB 7500
7220 LOAD I$ DATA F$()
7230 GO TO 7010
7300 REM ***** GUARDAR DATOS *****
7310 GO SUB 7500
7320 SAVE I$ DATA M()
7330 GO TO 7010
7400 GO SUB 7500
7420 SAVE I$ DATA F$()
7430 GO TO 7010
7500 PRINT AT 18,0
7510 INPUT "ENTRAR NOMBRE ARCHIVO";I$
7520 RETURN
```

Gran Premio

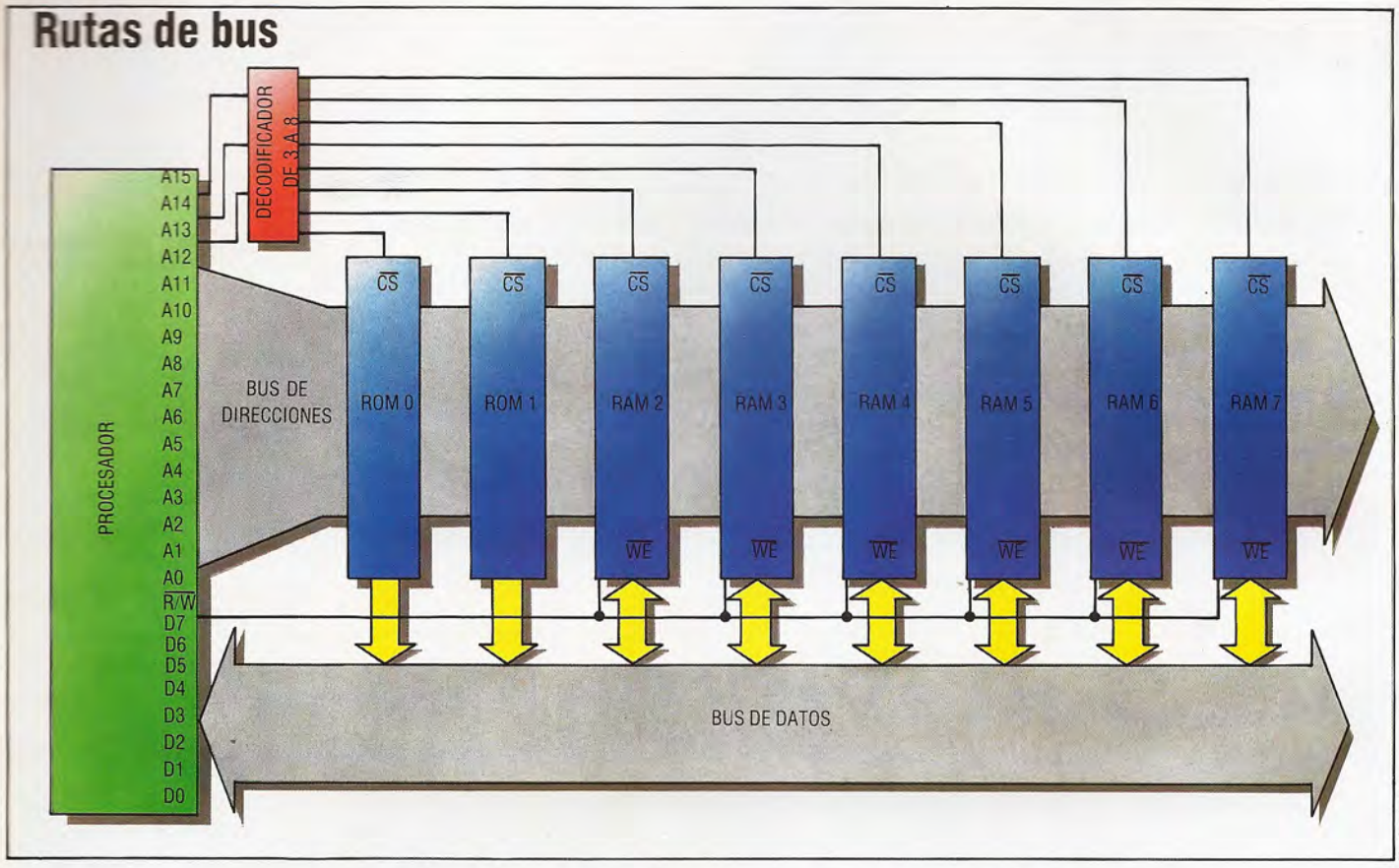
He aquí nuevamente este popular juego que siempre encabeza los "hit parades" de los programas recreativos. En esta ocasión presentamos una versión para el microordenador MSX

Al volante de su Fórmula 1 intente recorrer la mayor distancia posible. Su coche dispone de dos velocidades. Mantenga pulsada la barra espaciadora para correr en segunda velocidad. La dirección se controla mediante las teclas del cursor. En segunda, el vehículo marcha a doble velocidad. Pero ¡cuidado con los accidentes!



```
10  REM ***************
20  REM * GRAN PREMIO *
30  REM ***************
40  KEY OFF
50  WIDTH 39
60  GOSUB 450
70  H=STICK (0)
80  PX=PX+(H=7)-(H=3)
90  IF STRIG(0) THEN T=2 ELSE T=1
100 IF VPEEK (PX+40)<>CR THEN 240
110 RX=RX+(RND(1)<.5) - (RND (1)<.5)
120 IF RX<RN THEN RX=RN
130 IF RX>RM THEN RX=RM
140 VPOKE XP,CR
150 LOCATE RX,24,0
160 PRINT R$
170 VPOKE PX,V
180 K=K+T
190 DL=(2+T)*50
200 FOR I=1 TO DL
210 NEXT I
220 XP=PX
230 GOTO 70
240 VPOKE XP,CR
250 FOR I=1 TO 5
260 VPOKE PX+40,A
270 FOR J=1 TO 100
280 NEXT J
290 BEEP
300 VPOKE PX+40,V
310 FOR J=1 TO 100
320 NEXT J
330 NEXT I
340 LOCATE 10,10,0
350 PRINT "KMS RECORRIDOS :";K;
360 IF INKEY$<>"" THEN 360
370 LOCATE 14,16,0
380 PRINT "OTRA ?";
390 D$=INKEY$
400 IF D$="" THEN 390
410 IF D$<>"N" AND D$<>"n"
    THEN RUN
420 LOCATE 0,0,1
430 CLS
440 END
450 CLS
460 SCREEN 0,0
470 GOSUB 680
480 DEFINT A-Y
490 COLOR 14,12
500 A=221
510 R$=CHR$ (219) +CHR$ (219) +
    CHR$ (219)
520 CR=219
530 RX=18
540 V=128
550 RY=24
560 T=1
570 PX=420
580 XP=PX
590 LOCATE 0,0,0
600 FOR I=0 TO 24
610 LOCATE RX,I,0
620 PRINT R$
630 NEXT I
640 VPOKE PX,V
650 RM=34
660 RN=2
670 RETURN
680 FOR I=0 TO 7
690 READ A
700 VPOKE 3072+I,A*4
710 NEXT I
720 RETURN
730 DATA 18,0,18,51,51,18,0,18
```

© SYBEX



Líneas de escritura
He aquí una disposición típica de un procesador conectado a 64 Kbytes de RAM estática y ROM. La línea $\overline{R/W}$ (*read/write*) del procesador está conectada con la línea de permisión de escritura, $\overline{WE}$ *(write entable)*, de cada una de las RAM. Cuando el procesador envía *low* a la línea WE, se pueden escribir datos en la posición de RAM direccionada. Los tres bits superiores simplemente se decodifican en ocho líneas de selección de chip y se utilizan para habilitar una RAM o ROM determinada. Los 13 bits *low* se transportan a la totalidad de los ocho chips de memoria y especifican la posición en cuestión

El material de la memoria

En esta ocasión centraremos nuestra atención en los chips de memoria y veremos cómo se puede escribir o leer un byte

Un punto sobre el hardware del ordenador que la mayoría de las personas tienen claro es la diferencia entre memoria RAM y memoria ROM. La RAM es una memoria que se puede leer y en la que se puede escribir, y la ROM sólo se puede leer, habiéndose definido su contenido durante el proceso de fabricación. Sin embargo, hay dos clases principales de RAM, denominadas *estática* y *dinámica*.

En este capítulo veremos cómo el microprocesador selecciona un byte de memoria determinado para efectuar una operación de lectura o escritura.

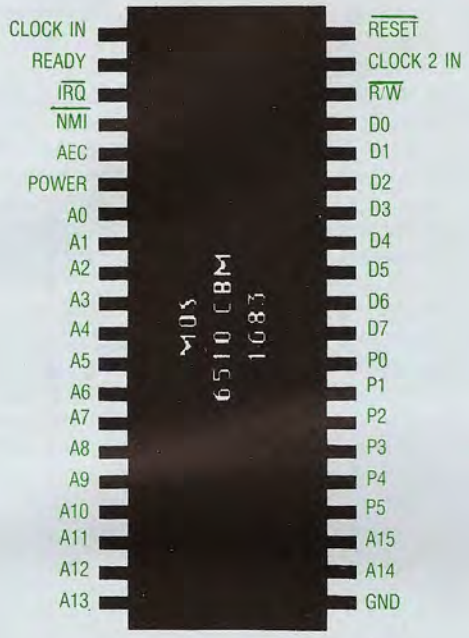
La RAM *estática* utiliza un pequeño circuito lógico secuencial que se conoce como flip-flop J-K para almacenar bits de datos individuales. Un flip-flop posee la capacidad de retener un estado determinado (ya sea cero o uno) en su salida hasta que un impulso lo haga bascular al estado opuesto. Se usan 8 flip-flops para formar cada byte de memoria y, en consecuencia, un chip de RAM estática tendrá empaquetados muchos de tales dispositivos.

La RAM *dinámica* almacena bits de datos como cargas eléctricas. Aunque almacenar datos de este modo ocupa mucho menos espacio (permitiendo que las RAM dinámicas tengan empaquetada más memoria por chip), las RAM se han de refrescar regularmente porque la carga tiende a debilitarse.

Los diagramas de la página contigua (arriba) muestran las conexiones de patillas para el microprocesador 6510 (un derivado del 6502) utilizado por el Commodore 64 y una típica ROM de 8 K. El procesador posee cuatro grupos de patillas principales: un bus de direcciones de 16 bits (A0 a A15), un bus de datos de ocho bits (D0 a D7), líneas de control de la CPU y del sistema. El 6510 posee la característica inusual de una puerta de seis bits en el chip que permite, entre otras cosas, conmutar bancos adicionales de ROM o RAM en su espacio de direcciones de 64 K.

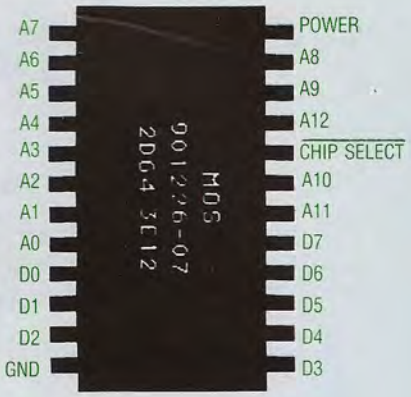
La ROM de 8 K reflejada posee 13 líneas de direcciones (A0 a A12), que le permiten seleccionar 8 192 posiciones individuales, ocho líneas de datos y una patilla de selección de chip. Esta disposición también es típica de un chip de RAM estática, con la excepción de que tal chip de RAM tendría una

Procesador 6510



Veinte pares de patillas
La mayoría de los procesadores de ocho bits están diseñados de acuerdo a un formato de 40 patillas, como el 6510 que vemos aquí. Observe las 16 patillas de direcciones, las ocho patillas de datos, las señales externas de reloj y control. En el control del sistema se incluyen entradas de interrupciones y una línea de permisión de lectura/escritura. Este procesador es una versión mejorada del 6502, siendo la principal adición un registro de datos en el chip utilizado para E/S de cassette y para controlar los bancos de memoria de los 84 K de RAM y ROM

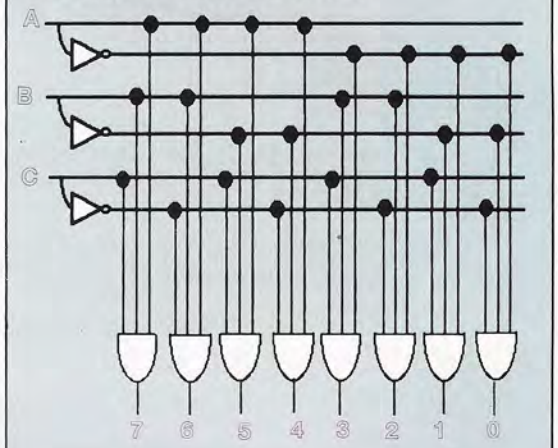
Chip de ROM



Caroline Clayton

Residencia selecta
El chip de ROM de 8 K que vemos aquí posee una disposición de patillas típicas entre esta clase de dispositivos. Se necesitan trece líneas de direcciones para direccionar individualmente cada una de las 8 192 posiciones, ocho patillas de datos permiten la conexión directa al bus de datos y se utiliza una línea de selección de chip junto con un decodificador externo para seleccionar esa ROM en lugar de otras ROM del sistema

Decodificador de 3 a 8



línea adicional de permisión de escritura. La clave para el direccionamiento de RAM o RAM estática es la línea de selección de chip.

Un procesador de ocho bits posee la capacidad de direccionar hasta 64 K de memoria utilizando su bus de direcciones de 16 bits (puesto que 2↑16=65 536=64 K). Por tanto, el procesador podría acceder a ocho chips de ROM o RAM de 8 K cada uno. Las líneas de dirección de A0 a A12 son necesarias para localizar el byte en el chip de memoria adecuado, pero ello nos deja a A13, A14 y A15, que se pueden utilizar para seleccionar el chip en cuestión. La disposición de, pongamos por caso, dos ROM de 8 K y seis RAM estáticas de 8 K podría ser la que se muestra aquí. Los tres bits superiores de la dirección se pasan por un decodificador de 3 a 8 para seleccionar el chip que contiene el byte direccionado, y los otros 13 bits de la dirección se cablean en común a los 8 chips, para seleccionar el byte concreto del chip seleccionado. Por consiguiente, todas las posiciones cuyas direcciones comiencen con 000 se pueden encontrar en el chip 0; aquellas cuyas direcciones empiecen con 001 se seleccionan en el chip 1, y así sucesivamente.

Mientras que las ROM sólo pueden ser leídas, las RAM se pueden leer o bien grabar. Para seleccionar la operación requerida, todas las RAM poseen una línea de permisión de escritura (WE: *write enable*) conectada a la línea de lectura/escritura ($\overline{R/W}$: *read/write*) del procesador. En el caso del 6510, la línea $\overline{R/W}$ siempre se retiene *high*, excepto cuando se ha de efectuar una operación de escritura, momento en que se pone *low*. Ahora se puede rastrear la acción de los ciclos de lectura o escritura y su incidencia en la memoria.

Cuando el procesador comienza a ejecutar una instrucción READ, como "cargar en el acumulador el contenido de la posición $C000", la dirección ($C000) se coloca en el bus de direcciones y la línea $\overline{R/W}$ se mantiene *high*. Los tres bits superiores de la dirección serán, en este ejemplo, 110, de modo que el sexto chip de RAM (que se denomina RAM 6) de nuestra disposición de memoria sería seleccionado a través del decodificador de 3 a 8. Los 13 bits restantes son todos cero, de modo que se seleccionará la primera posición de RAM 6 y se conectará al bus de datos de modo que su contenido se pueda volver a transmitir al procesador y colocar en el acumulador.

Durante una instrucción WRITE (como "almacenar en $C000 el contenido del acumulador") la posición se selecciona de la misma manera pero esta vez la $\overline{R/W}$ se mantiene *low*, haciendo que la patilla de permisión de lectura ($\overline{WE}$) se ponga *low*. En esta etapa de la instrucción el contenido del acumulador ya se habrá colocado en el bus de datos y estará escrito en la primera posición de RAM 6.

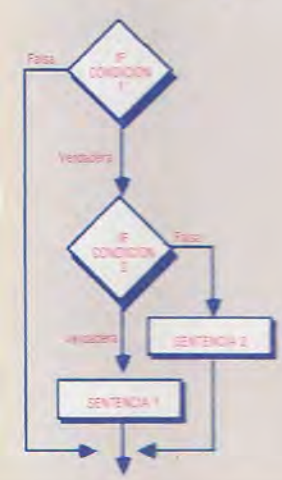
Descifrando el código
Los circuitos decodificadores tienen muchas aplicaciones en el hardware de ordenador. Su labor consiste en aceptar una entrada codificada en binario y producir señales individuales que correspondan a cada permutación del código. Por consiguiente, en ocho salidas separadas se puedcn decodificar tres entradas; la línea 0 corresponde a 000 en las entradas, la línea 1 corresponde a 001 y así sucesivamente hasta la línea 7, que es activada por 111 en las líneas de entrada

División operativa

Nos corresponde centrar nuestra atención en la división de procedimientos, o sea, la parte de un programa en COBOL que lleva a cabo operaciones y cálculos

La división de procedimientos (*procedure division*) de un programa en COBOL corresponde a la función básica del programa. El trazado de la división de procedimientos y el vocabulario utilizado pretenden conseguir que el programa se parezca en la mayor medida posible al inglés empresarial. Cada sección se compone de párrafos, que a su vez se componen de oraciones que deben terminar con un punto y aparte. Una oración correspondería a una sentencia en un lenguaje como el BASIC, pero una oración COBOL puede a su vez estar dividida en cláusulas (las comas son opcionales), cada una de las cuales puede ser una "sentencia" o calificar a otra. Se dice que la palabra clave de una cláusula o una oración que especifica la acción a emprender es el *verbo*. Con mucho, el verbo más utilizado es MOVE, que se limita a transferir datos de una zona a otra, tomando la forma:

```
MOVE identificador-1 TO identificador-2.
```



Problemas con los IF

El empleo del punto y aparte como terminador para la construcción IF en COBOL puede plantear problemas. Por ejemplo, la construcción:

```
IF condición1 (then)
   IF condición 2 (then)
      sentencia1
ELSE condición2.
```

se ejecutaría como podemos ver en el diagrama de flujo superior. No obstante, el papel del punto y aparte como terminador de AMBAS sentencias significa que el COBOL compila la alternativa no deseada que vemos abajo. La forma de sortear este problema consiste en colocar la selección interior en un párrafo separado y después llevar a cabo (PERFORM) ese párrafo. De modo que la construcción de arriba se codificaría mejor en COBOL como:

```
IF condición1 (then)
   PERFORM párrafo 1
   ELSE sentencia2

.....

párrafo1
   IF condición2
      sentencia1
```

En un lenguaje como el COBOL, incluso una idea tan simple como ésta posee considerables ramificaciones. En primer lugar, hay dos tipos de movimiento: el movimiento alfanumérico transfiere carácter por carácter desde la izquierda, truncando o añadiendo blancos en la medida de lo necesario; el movimiento numérico coloca el punto decimal adecuadamente y efectúa cualquier cambio de uso que sea necesario. El movimiento numérico también trunca o rellena con ceros ya sea antes o después del punto decimal. El tipo de movimiento depende de las PICTURE de los dos datos. En realidad la diversión comienza cuando uno considera los MOVE entre elementos alfanuméricos y numéricos; algunos MOVE no se permiten bajo ninguna circunstancia, pero éstos básicamente implican el uso de PICTURE especiales de edición y MOVE alfabético-numéricos. Si bien existe una especificación detallada de lo que sucede en cada MOVE legítimo, aun así es complicado, de modo que por el momento basta decir que una sentencia que permita fácilmente que usted trunque dígitos significativos sin ningún mensaje en tiempo de ejecución se debe utilizar con sumo cuidado.

La aritmética no es el punto fuerte del COBOL, si bien hay que decir que son pocos los lenguajes que ofrecen como facilidad estándar la precisión de 18 dígitos del COBOL. La aritmética se puede realizar de dos formas. Primero, están los verbos aritméticos ADD (sumar), SUBTRACT (restar), MULTIPLY (multiplicar) y DIVIDE (dividir). P. ej.:

```
ADD número-1 TO número-2.
```

suma el contenido de número-1 al contenido de número-2, y:

```
ADD número-1 TO número-2 GIVING número-3.
```

forma la suma de número-1 y número-2 en número-3. Existe otra opción que permite llevar a cabo la misma aritmética en varios lugares:

```
ADD 5 TO número-1,número-2
```

p. ej., le suma 5 a los valores actuales tanto de número-1 como de número-2.

Los otros verbos aritméticos responden al mismo formato general:

```
SUBTRACT número-1 FROM número-2 [GIVING
   número-3].
MULTIPLY número-1 BY número-2 [GIVING
   número-3].
DIVIDE número-1 INTO número-2
   [GIVING número-3]
   [REMAINDER número-4].
```

Observe, en cada caso, que el valor final se halla en el campo enumerado en último lugar.

Al igual que con los MOVE, se plantean problemas cuando el resultado de una operación aritmética es demasiado largo para el espacio que se ha preparado para almacenarlo. Si se pierden dígitos menos significativos, normalmente se truncarán, excepto cuando el nombre del dato receptor va seguido de la palabra ROUNDED (redondeado). Truncar los dígitos más significativos es, por supuesto, mucho más peligroso. El COBOL permitirá que esto suceda, pero da la opción de añadir una cláusula ON SIZE ERROR (en caso de error de tamaño) a cualquier sentencia aritmética. Esto no tiene ningún

efecto, a menos que se produzca un recorte de dígitos más significativos, en cuyo caso deja intacto el contenido previo y efectúa la acción especificada en la cláusula.

Como usted puede imaginar, en COBOL todo, excepto la aritmética más simple, se puede volver muy tedioso de escribir. Las cosas se han simplificado gracias a la introducción del verbo COMPUTE (calcular), que puede ir seguido por una sentencia aritmética de una forma similar a una sentencia de asignación de BASIC o FORTRAN, como:

COMPUTE número-1=(número-2 + 3)* número-3.

Algunos compiladores incluso permiten el uso de exponenciación en tales expresiones, pero ello no es estándar.

Existen varios problemas que surgen a raíz del uso de COMPUTE, uno de los cuales es el problema de SIZE ERRORS que se podrían producir en cualquier resultado intermedio así como en el resultado final. Se puede utilizar una cláusula ON SIZE ERROR, pero la misma no le dirá en qué etapa del cálculo se produjo el desbordamiento. Por este y otros motivos, muchos puristas del COBOL no emplean el verbo COMPUTE sino que se basan en los verbos aritméticos comunes, que pueden ser tediosos de escribir pero resultan mucho más claros y seguros.

La división de procedimientos se prepara en un cierto número de párrafos. Cada párrafo lleva un nombre; el nombre se escribe en la zona A que empieza en la columna 7 y las sentencias se escriben en la zona B, que empieza en la columna 12. Estos párrafos operan de forma muy similar a una subrutina de BASIC. En consecuencia, por lo general no hay módulos ni procedimientos separados como en PASCAL, y no hay ni parámetros ni variables locales. La ejecución normal del código es desde el comienzo hasta el final, ejecutándose cada párrafo de uno en uno a medida que van apareciendo.

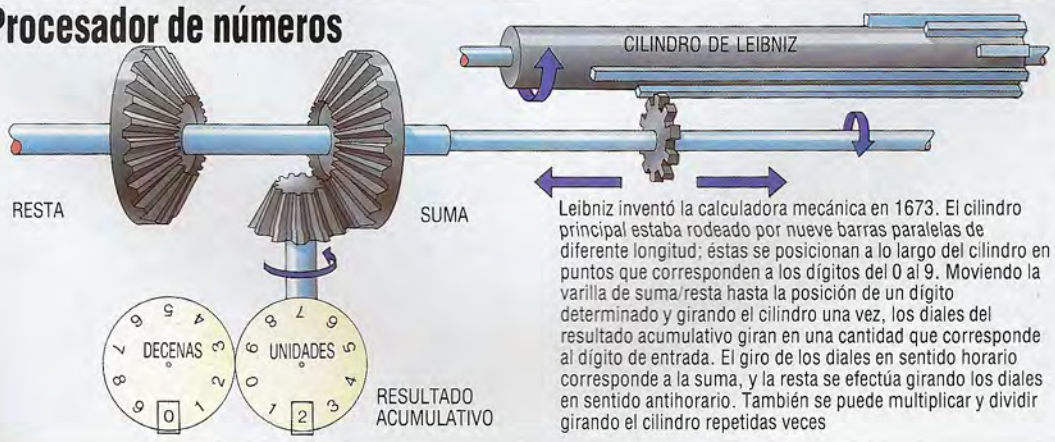
En COBOL la secuencia de control se puede modi-

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PI-CALC.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
 SPECIAL-NAMES. CONSOLE IS CRT.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
  01  SCREEN PIC X(1920).
*
  01  DI-1 REDEFINES SCREEN.
      02 FILLER PIC X(160).
      02 DI-TX1 PIC X(160).
      02 DI-TX2 PIC X(13).
      02 DI-TERM PIC X(15).
      02 FILLER PIC X(136).
      02 DI-TX3 PIC X(6).
      02 DI-PI PIC X(15).
      02 FILLER PIC X(1415).
*
  01  DI-2 REDEFINES SCREEN.
      02 FILLER PIC X(333).
      02 DI-TERM2 PIC X(15).
      02 FILLER PIC X(142).
      02 DI-PI2 PIC X(15).
      02 FILLER PIC X(1415).
*
  01  WORK-AREA.
      02 PI PIC S9V9(14).
      02 TERM PIC S9V9(14).
      02 W PIC S9V9(14).
      02 N PIC 9999.
      02 N1 PIC 9999.
      02 N2 PIC 9999.
      02 ED PIC -9.9(12).
*
  01  CONSTANTS.
      02 TX1 PIC X(17) VALUE "CALCULATION OF PI".
      02 TX2 PIC X(12) VALUE "NEXT TERM IS".
      02 TX3 PIC X(5) VALUE "PI IS".
*
 PROCEDURE DIVISION.
 LA-START.
      DISPLAY SPACE.
      MOVE SPACE TO SCREEN.
      MOVE TX1 TO DI-TX1.
      MOVE TX2 TO DI-TX2.
      MOVE TX3 TO DI-TX3.
      MOVE 0.5 TO ED.
      MOVE ED TO DI-TERM.
      MOVE 3 TO ED.
      MOVE ED TO DI-PI.
      DISPLAY DI-1.
      MOVE 0.5 TO PI.
      MOVE 0.5 TO TERM.
      MOVE 3 TO N.

 LOOP.
      MOVE N TO N2.
      SUBTRACT 2 FROM N2.
      MULTIPLY N2 BY N2.
      MULTIPLY N2 BY TERM.
      MOVE N TO N1.
      SUBTRACT 1 FROM N1.
      MULTIPLY N BY N1.
      MULTIPLY 4 BY N1.
      DIVIDE N1 INTO TERM.
      IF TERM < 0.0000000000001 THEN GO TO HALT.
      ADD TERM TO PI.
      MOVE PI TO W.
      MULTIPLY 6 BY W.
      MOVE W TO ED.
      MOVE ED TO DI-PI2.
      MOVE TERM TO ED.
      MOVE ED TO DI-TERM2.
      DISPLAY DI-2.
      ADD 2 TO N.
      IF N < 100 GO TO LOOP.
 HALT.
      STOP RUN.
```

Calculando "pi"
Este programa en COBOL ilustra el uso de los verbos aritméticos para calcular un valor para *pi* usando las series de Leibniz. Esta serie de expresiones algebraicas opera sobre el principio de calcular una serie infinita de términos, con cada término de la serie volviéndose sucesivamente más pequeño.

Gran parte del trabajo de Leibniz ha sido de suma importancia para los procedimientos informáticos modernos y, aun en su propio tiempo, condujo al desarrollo de "máquinas de calcular", una de las cuales se ilustra abajo. El programa en COBOL, ejecutado en una máquina bastante más moderna, se escribió bajo CP/M utilizando MicroFocus CIS-COBOL

Procesador de números



Leibniz inventó la calculadora mecánica en 1673. El cilindro principal estaba rodeado por nueve barras paralelas de diferente longitud; éstas se posicionan a lo largo del cilindro en puntos que corresponden a los dígitos del 0 al 9. Moviendo la varilla de suma/resta hasta la posición de un dígito determinado y girando el cilindro una vez, los diales del resultado acumulativo giran en una cantidad que corresponde al dígito de entrada. El giro de los diales en sentido horario corresponde a la suma, y la resta se efectúa girando los diales en sentido antihorario. También se puede multiplicar y dividir girando el cilindro repetidas veces

Kevin Jones

ficar de numerosas maneras, básicamente mediante el verbo PERFORM (efectuar), que posee muchas variantes. Dos de éstas son:

```
PERFORM párrafo-1.
```

que actúa como una llamada a procedimiento o como un GOSUB, ejecutando las sentencias del párrafo y luego devolviendo el control a la sentencia que sigue al PERFORM, y

```
PERFORM párrafo-1 THRU párrafo-n.
```

que ejecutará varios párrafos consecutivos antes de devolver el control.

Otra forma de incidir en el flujo de control es utilizar:

```
GOTO párrafo-1.
```

que funciona del modo habitual, y que en COBOL se debe evitar en la misma medida que en otros lenguajes.

Como lenguaje destinado prioritariamente a aplicaciones de gestión, el COBOL proporciona un amplio conjunto de opciones de lectura y escritura para archivos en disco o cinta. Sin embargo, el COBOL estándar tiene facilidades mínimas para entrada/salida interactivas utilizando una pantalla y un teclado. Los dos verbos para manipular esto son:

```
ACCEPT nombre-dato.
```

para entrada desde el teclado, y:

```
DISPLAY nombre-dato.
```

Éstas operan en la modalidad de teletipo normal, de modo muy similar al INPUT y PRINT del BASIC. La mayoría de las versiones de COBOL para micros incluyen verbos ACCEPT (aceptar) y DISPLAY (visualizar) mejorados con el fin de permitir el trabajo en pantalla.

Una de las principales facilidades que se espera de cualquier lenguaje de programación moderno es la gama de estructuras de control disponibles, en especial las facilidades para selección e iteración. El COBOL proporciona una construcción IF...THEN...ELSE para la selección, si bien es limitado en comparación a la que ofrece, por ejemplo, el PASCAL. El formato es:

```
IF condición
  sentencia(s)
ELSE
  sentencia(s).
```

Observe que no hay ningún THEN y también que la sentencia IF acaba con un punto y aparte como cualquier otra sentencia, lo que significa que no puede haber ningún punto y aparte entre las sentencias ni tampoco en la parte IF o en la parte ELSE. Normalmente esto no tiene importancia, porque las sentencias pueden ir una detrás de la otra en una misma oración, pero establece una diferencia si alguna de las sentencias ofrece alternativas, como puede ser otra sentencia IF o READ que compruebe una marca de final de archivo. En estos casos, el punto y aparte que termina la selección interior terminará asimismo la exterior.

Las condiciones booleanas que comprueba el IF se pueden formar de numerosas maneras. Primero, se pueden establecer (como en la mayoría de los otros lenguajes) usando los operadores relacionales <, > y =, así como usando NOT para crear los otros, como en NOT <, que significa lo mismo que > =. Los operadores se pueden escribir como símbolos o bien escribir al completo, como en número-1 IS EQUAL TO 5. Cualquier número de estas condiciones se puede unir con AND y OR para producir una condición compuesta, como en (número-1 = 5) OR (número-2 NOT = 6).

El COBOL es uno de los pocos lenguajes que permiten abreviar estas condiciones compuestas cuando en una condición compuesta se está utilizando nuevamente un nombre de dato o el mismo operador. Las siguientes son dos condiciones de COBOL legales:

```
número-1 < 10 AND > 6
letra-1="A" OR "B" OR "C"
```

El COBOL no posee un tipo de datos booleano, pero proporciona una entrada de datos especial de nivel 88 que se puede asociar con cualquier dato elemental para especificar un valor, escala o conjunto de valores y un nombre de condición booleana que se puede utilizar allí donde se emplearía una condición normal.

Por ejemplo:

```
77 número-1 PIC 99.
88 el-número-es-válido VALORES 1,3,24 THRU 67.
```

(Observe que algunos COBOL sólo permiten ya sea una escala o bien una lista de valores, pero no ambas.)

El nombre de condición el-número-es-válido se asocia con número-1 sólo por estar escrito inmediatamente después de él en la división de datos. Siempre que se almacene un valor en número-1, el valor de el-número-es-válido se ajusta como verdadero o falso adecuadamente y se puede comprobar de forma directa, como en:

```
IF el-número-es-válido
  PERFORM rutina-válido
ELSE
  PERFORM rutina-no-válido.
```

En COBOL, la iteración (efectuar un bucle) se puede realizar de diversas maneras, todas las cuales implican variantes de la sentencia PERFORM. La forma básica es:

```
PERFORM nombre-párrafo UNTIL condición.
```

donde la condición puede ser cualquier combinación de relaciones o elementos de nivel 88, como con IF. Actúa como un bucle WHILE de BASIC y otros lenguajes, comprobándose la condición cada vez antes de llevar a cabo el párrafo. Si la condición comienza siendo verdadera, el párrafo no se efectúa nunca.

En COBOL, el equivalente más próximo a un bucle FOR...NEXT de BASIC es una variante de este uso del PERFORM, que permite incrementar uno o más datos numéricos a partir de un valor inicial y en función de un valor de paso. La siguiente sentencia llevará a cabo el código de párrafo-1 cinco veces:

```
PERFORM párrafo-1 VARYING número-1 FROM
  1 BY 1
UNTIL número-1>5.
```

Observe que la condición terminadora puede ser cualquiera: no ha de implicar necesariamente al dato numérico.

TRABAJAR CON PALABRAS

En esta nueva serie repasaremos los conceptos fundamentales del tratamiento de textos y analizaremos algunos de los paquetes más populares

El tratamiento de textos es, esencialmente, una combinación de distintas operaciones, algunas de las cuales dependen del usuario y otras las realiza completamente el ordenador. Estas operaciones son la entrada de texto en el ordenador, la creación de un archivo en RAM o en disco para almacenar el texto (por lo general, en formato ASCII), la visualización de todo el archivo o parte del mismo en una VDU de forma legible, la manipulación de los datos de ese archivo y, por último, el tratamiento de los diversos archivos creados.

Es necesario comprender todo esto de forma clara, porque todo procesador de textos se ha de juzgar en función del éxito con el que lleva a cabo, o permite que el usuario lleve a cabo, estas operaciones. Algunos programas de tratamiento de textos son, por ejemplo, especialmente buenos para visualizar texto (*MacWrite*, p. ej., muestra en pantalla toda una variedad de exóticas fuentes), mientras que otros son soberbios para el tratamiento de archivos. Además del rendimiento de estas tareas de nivel relativamente bajo, el software para tratamiento de textos también se debe juzgar en función de la potencia y las facilidades de alto nivel que ofrezca.

Chris Stevens

Verdaderamente especializado
El Amstrad PCW 8256 es el último de la gama de productos de precio reducido de la firma Amstrad. Empaquetado como procesador de textos especializado y basado en el chip Z80 con 256 Kbytes de RAM disponible, el PCW proporciona un ordenador, pantalla, unidad de disco integrada, impresora y software para tratamiento de textos empaquetado

Cortesía de Microwriter Ltd.

Lenguaje de dedos
El Microwriter ofrece un concepto completamente diferente en tratamiento de textos al de cualquier otra máquina disponible en el mercado. Prescindiendo por completo del teclado QWERTY, los caracteres individuales se entran pulsando distintas combinaciones de teclas. Una vez entrado el texto de un documento, el Microwriter se conecta, entonces, a una impresora para obtener una copia impresa

En la actualidad el tratamiento de textos se ha vuelto tan familiar que la compleja programación que requiere se suele dar por sentada. Además, es un área de programación cuyo origen es bastante reciente. Ello se debe a diversas razones. En primer lugar, es esencial una VDU para visualizar el texto, y éstas sólo se han vuelto dispositivos de salida estándares para ordenadores en los diez últimos años. En segundo lugar, los ordenadores están diseñados para tratar números, no texto. La introducción del código ASCII evidentemente soluciona este problema, pero el código retiene 255 caracteres diferentes y, como tal, no es en absoluto eficiente en cuanto a su uso de memoria. Un teclado de máquina de escribir estándar contiene poco más de 100 caracteres (contando las teclas con 2 caracteres), de modo que es fácil ver que un código de 255 caracteres posee una considerable redundancia incorporada. Por este motivo, el tratamiento de textos no sólo es complejo, sino que también es muy exigente desde el punto de vista de la memoria.

Esta complejidad conduce al problema de tener que abordar el tratamiento de textos de muchas maneras diferentes. Los procesadores de textos vienen en muchas formas, tamaños y formatos diferentes.

Los procesadores de textos especializados, como los que fabrican Wang e IBM, son muy populares en el mercado de gestión. Consisten en un terminal de visualización, un teclado con varias teclas diseñadas específicamente para funciones de tratamiento de textos (una tecla Paste [pegar], p. ej.) y un medio para almacenar archivos de texto en disco rígido o flexible. Con la excepción del Amstrad PCW 8256, tales máquinas son caras. No obstante, ofrecen ventajas derivadas de la especialización: teclados construidos especialmente, facilidades para redes y visualizaciones de diseño ergonómico. El tratamiento de textos supone mucho tiempo contemplando la pantalla, de modo que es esencial una visualización ancha (más de 80 columnas) y nítida.

En compañía del ordenador
Los ordenadores portátiles para regazo están ganando terreno gradualmente en el mercado imformático. La ventaja de estas máquinas es que el usuario puede llevar a cabo una "informática sobre la marcha". La mayoría de los ordenadores para regazo, como el Olivetti M10 que vemos fotografiado aquí, tienen retenida en ROM la capacidad para tratamiento de textos. Si bien en estas máquinas las facilidades de tratamiento de textos y la cantidad de memoria libre son restringidas debido a la limitada potencia disponible, son herramientas útiles para cartas, memorándums, etc.

Por el contrario, el software para tratamiento de textos se puede clasificar en varias categorías diferentes. En el nivel inferior, se puede conseguir cierta forma de tratamiento de textos utilizando un sencillo editor de pantalla como el que proporcionan la mayoría de los sistemas operativos. Usted puede emplear esta facilidad para imprimir párrafos cortos en modalidad directa, pero obviamente esto no será suficiente para nada, con la excepción del memorándum más breve.

Ascendiendo en la clasificación está el "programa para tratamiento de textos", que permite que el usuario prepare un archivo de texto y lo edite directamente en pantalla. Éste es un concepto que se acerca mucho más al de un auténtico procesador de textos, y algunos manipuladores de textos incluso ofrecen rutinas de búsqueda (para localizar una serie de caracteres determinada) y funciones de recortar/pegar, que permiten que el usuario manipule pequeños bloques de textos dentro de un archivo. Un buen ejemplo de tales programas es la facilidad TEXT que se ofrece en la gama de ordenadores portátiles Kyocera, comercializados bajo el logotipo del Olivetti M10 y el Tandy Modelo 100. El programa ofrece funciones de recortar, pegar, copiar, buscar e imprimir, almacenando el texto como un archivo en RAM CMOS, con una batería de apoyo, de modo que seguirá estando allí la próxima vez que se use la máquina.

Por último tenemos los programas que pueden reclamar con toda justicia el título de procesadores de textos. Éstos se pueden clasificar en dos categorías: basados en RAM y basados en disco. La primera categoría se introdujo en realidad para beneficio de los usuarios de ordenadores personales que carecían de almacenamiento en disco y, por lo tanto, requerían un programa que se cargara enteramente en RAM y, debido a que el almacenamiento en cassette es tan lento, almacenara también en RAM todo el documento. Sólo cuando éste se ha editado en una versión final es guardado en cinta. Los programas basados en RAM son limitados en cuanto a la gama de facilidades que ofrecen, por razones obvias. En particular, el tamaño de los documentos está severamente limitado, y también suele estar limitado el tamaño de los bloques de texto que se puedan desplazar.

Sin embargo, los programas basados en RAM ofrecen una gran ventaja respecto a sus parientes basados en disco: tienden a ser de operación mucho más rápida, dado que las distintas facilidades no dependen de numerosos accesos a disco para funcionar. Un buen ejemplo de procesador de textos basado en RAM es el *Tasword*, que permite que el usuario cree un documento de hasta 13 000 caracteres, o de algo más de 2 000 palabras.

Los programas basados en disco suelen permitir el trabajo con documentos muchísimo más grandes y, en teoría, las dimensiones del documento sólo están limitadas por la zona de almacenamiento disponible en el disco de datos. El programa carga porciones del archivo en cuestión en la medida en que así se requiera, si bien a menudo esto puede conducir a considerables demoras, especialmente si usted se halla al final de un documento largo y desea retornar al principio. Ejemplos típicos de sistemas basados en disco son el *WordStar* y el *PerfectWriter*. Tales programas con frecuencia vienen empaquetados en sistemas de gestión, pero de lo contrario tienden a ser muy caros. Una ventaja de los programas basados en disco es que suelen hacer un uso exhaustivo del sistema operativo de disco sobre el que operan (CP/M, MS-DOS, etc.) y, en consecuencia, tienden a gozar de una mayor compatibilidad de archivos que sus equivalentes para ordenadores personales.

En esta serie veremos algunos tipos diferentes de procesadores de textos y evaluaremos sus puntos fuertes y sus puntos débiles. Empezando por el *WordStar*, un programa que ha conseguido establecerse como un estándar de la industria en función del cual se juzgan todos los otros programas, continuaremos con el *MacWrite* como ejemplo de un programa que sobresale por su visualización. Asimismo, analizaremos diversos programas basados en RAM, como el *Tasword* y el *Mini-Office*.

Una obra maestra

Su gran capacidad de direccionamiento hace que el microprocesador 68000 pueda competir en potencia hasta con los ordenadores centrales

El microprocesador 68000 tiene su origen en la serie 6800 de microordenadores de Motorola, tan bien acogida en el mercado, cuyo punto final fue el microprocesador 6809. Motorola reanudó este cabo con su serie 68000 de alto rendimiento, que ha sido posible gracias a los adelantos en técnicas de VLSI (*very large scale integration*: integración a muy grande escala), que permiten albergar más de 100 000 elementos electrónicos lógicos (tales como puertas AND/OR, flip-flops, etc.) en un solo chip de 64 patillas y de apenas 5 cm de longitud. Con esto se consigue una capacidad de direccionamiento muy similar a la del miniordenador DEC PDP-11, con todas las ventajas y la potencia del juego de instrucciones de un ordenador central: todo en un solo chip.

Estudiaremos más adelante esta capacidad de direccionamiento y de instrucciones. De momento, nos vamos a ceñir a la arquitectura global del 68000 para disponer de un modelo útil donde aplicar nuestros programas. La idea de un modelo es importante porque necesitamos limitar el campo de nuestros conocimientos del microordenador a aquellos aspectos que harán que nuestros programas funcionen. Por ejemplo, no necesitamos comprender cómo trabajan los elementos flip-flop, para poder sumar dos números y establecer el resultado en un registro. Pero sí que nos es necesario saber detalles sobre la longitud de un registro y las instrucciones ADD si queremos resolver ese problema. Nuestro modelo queda, por ello, limitado a las funciones imprescindibles para programar un micro.

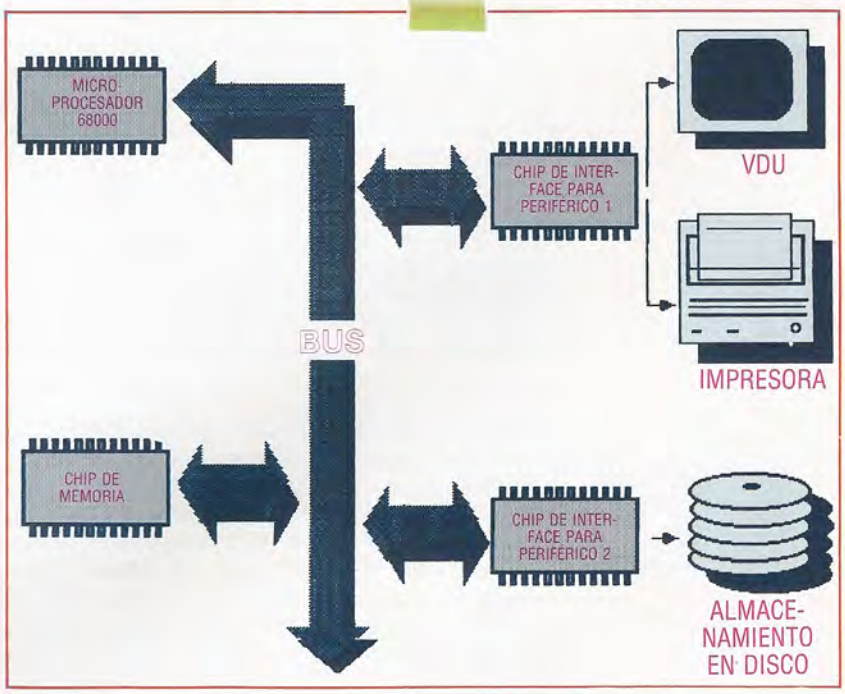
La arquitectura del hardware

Si examinamos las funciones eléctricas de las patillas del microprocesador 68000 veremos que las señales están relacionadas con el control de los buses digitales. Estos buses son los medios de que se vale el microordenador para comunicarse con el mundo electrónico de su entorno y con los cuales se puede construir un sistema de microordenadores. Sobre una única placa es posible construir un típico sistema como el que se esquematiza en el dibujo. Aquí el procesador puede tomar la instrucción siguiente desde la memoria a través del bus, ejecutar la instrucción internamente dentro del procesador y, si fuera necesario, enviar un carácter a un chip de interface para visualizarlo ante el usuario.

Esta arquitectura del hardware constituye la base de nuestro modelo. Todavía necesitamos pensar

Un chip para el futuro
La serie 68000 otorga a los micros de hoy la potencia de proceso de un mini, y los fabricantes se han apresurado a sacar partido de ello. Las máquinas que aquí se muestran emplean todas el 68000 y se han desarrollado potentes chips periféricos para tratar gráficos y sonido que se ejecutan en conexión con el chip

El modelo básico
El esquema muestra, simplificado, un sistema de ordenador típico de una sola placa. Las instrucciones y los datos se toman de la memoria a través del bus y se procesan. Los resultados se depositan de nuevo en el bus para ser transmitidos a la pantalla, a otros periféricos o a la RAM

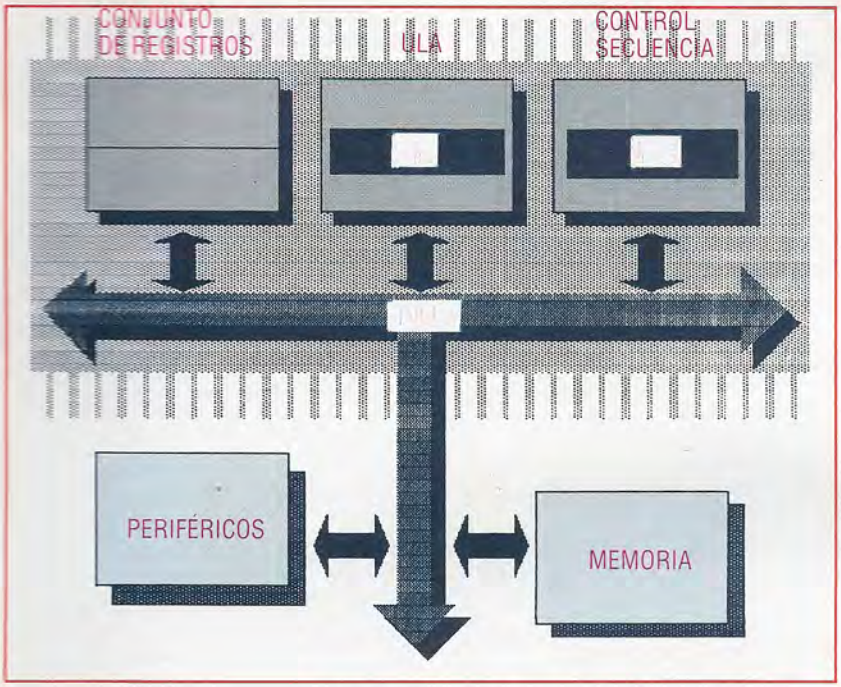


Mike Clowes

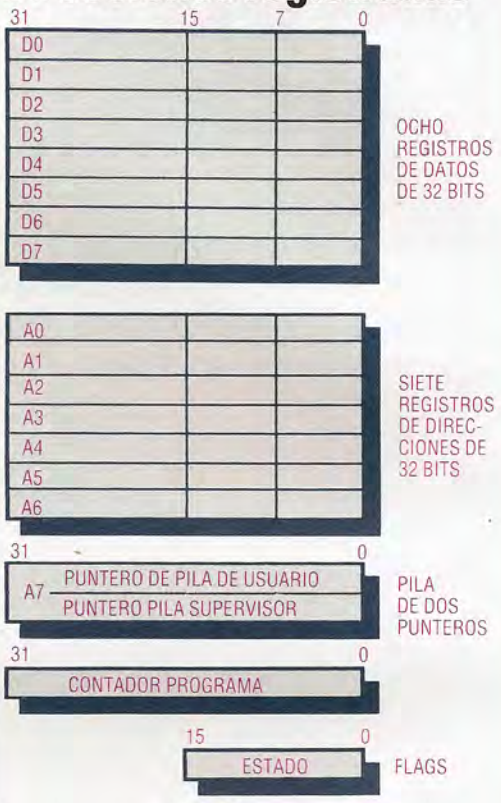
Modelo del programador
El diseño interno del 68000 puede reducirse esencialmente a tres unidades principales: los registros de datos y direcciones, la unidad aritmético-lógica y la unidad de control de secuencia. Cada unidad comunica con las restantes y con los dispositivos periféricos a través del bus de datos digital

como si el microprocesador sólo contara con un bus de comunicación central, pero debemos tener en cuenta también los componentes de nivel de registro interno.

El segundo dibujo muestra el modelo del programador donde podemos ver que la única diferencia en la arquitectura del hardware reside en que el chip del 68000 se ha escindido en tres componentes lógicos. Analicémoslos uno a uno.



Direcciones registradas



Los quince registros del 68000 parece que ofrecen casi un incomparable panorama. Sin embargo, es obligatoria una cierta disciplina a causa de la subdivisión de los registros internos en categorías de direcciones y categorías de datos

Estos registros ya no son, claro está, más que una matriz de rápidas posiciones de memoria empleadas de la misma manera que cualquiera de los registros de otros ordenadores para el almacenamiento de datos o resultados intermedios. La diferencia que caracteriza al 68000 es que cuenta con ocho registros de datos y ocho registros de direcciones formados por 32 bits (dicho de otra manera, son registros de 32 bits de ancho).

No importan las razones de Motorola para escindir los registros en dos subconjuntos, lo cierto es que se han simplificado las operaciones que debe efectuar la máquina y su programación se ha hecho más fácil.

Por ejemplo, si deseamos cargar el contenido de la memoria cuya dirección se encuentra en un registro, listo para una operación aritmética, sabemos con esta arquitectura que el puntero, o la dirección, debe ser cargado en un registro de dirección y que los datos deben ser almacenados en un registro de datos. Traducido al lenguaje assembler, sería así:

```
MOVE        (A2),D4
```

donde MOVE es la instrucción de trasladar (o copiar) los datos, A2 significa la dirección de origen como contenido del registro 2 de direcciones, y D4 significa que el destino es el registro 4 de datos. Pero si la dirección de los datos de origen estuviera en D4, no podríamos usar

```
MOVE        (D4),A2
```

dado que con MOVE tanto los modos de direccionamiento usados en origen y en destino son ilegales.

Otro detalle a tener en cuenta cuando se consideran los conjuntos de registros, y en particular los registros de datos, es que los datos a los que se debe acceder pueden ser de cinco tipos:

- Bit – un solo dígito binario
- Dígito BCD (o cuarteto)
- Byte – carácter de ocho bits
- Palabra – palabra de 16 bits
- Palabra larga – palabra de 32 bits.

Trataremos los dígitos BCD y las palabras largas en capítulos venideros; de momento los consideraremos como unidades cada vez más grandes de datos.

Gran parte de las instrucciones nos permiten especificar el tipo de dato junto con la instrucción, especificándolo como un atributo de la propia instrucción. Por ejemplo, si sólo deseáramos copiar los bits del 0 al 7 desde D1 a D2 nos serviremos de:

```
MOVE.B      D1,D2
```

pero si deseáramos los 32 bits enteros, la instrucción se transformaría en:

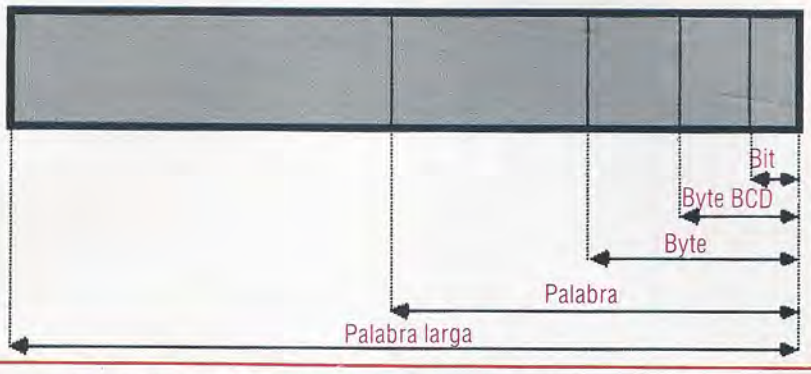
```
MOVE.L      D1,D2
```

El atributo (o código de tamaño de los datos) para una palabra de 26 bits es .W, que es el atributo por defecto cuando no se especifica ninguno.

Un último punto a notar antes de abandonar el conjunto de registros es que uno de los registros de direcciones, el A7, se emplea como puntero de pila del sistema, por lo que debemos tener cuidado de no cambiar alegremente este registro. Más tarde examinaremos las pilas.

Longitud de las palabras

El 68000 puede, mediante instrucciones ampliadas, operar con palabras de cinco longitudes distintas, procesando datos en grupos cuyo tamaño va de 1 a 32 bits. Lo cual implica las ventajas de la flexibilidad y de una mayor potencia de proceso respecto de los viejos chips de ocho bits



El siguiente elemento por considerar dentro del modelo del programador es la unidad aritmética lógica (ULA). Es la parte del micro encargada de realizar tareas aritméticas o lógicas dejando el resultado en el operando de destino.

Por ejemplo, es la ULA quien suma D1 y D2 y coloca el resultado en D2 cuando se ejecuta la siguiente instrucción:

```
ADD     D1,D2
```

Repitamos que es posible especificar el atributo longitud de los datos con la instrucción:

```
ADD.B   D1,D2
```

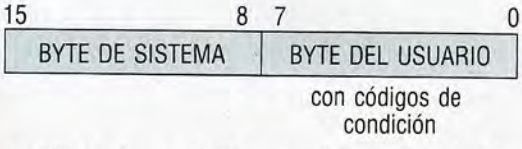
se limitará a sumar los bytes menos significativos.

Volviendo al diagrama del modelo del programador, existe un registro con etiqueta SR, o registro de estado, asociado a la ULA. Este registro está presente de tal modo que podemos recordar el resultado del paso de instrucción anterior. Esto se utiliza para poder efectuar una bifurcación condicional en el programa, según sea el resultado de la ejecución de la instrucción previa. Por ejemplo:

```
ADD     D1,D2    suma D1 a D2
BVS     OFLOW    comprueba desbordamiento
MOVE    D1,D3    copia D1 sobre D3
```

donde una de dos, o bien bifurcamos hasta la etiqueta OFLOW si el bit de desbordamiento está activado en SR, o bien se efectúa MOVE D1, D2 si no lo está. La instrucción BVS (*Branch if oVerflow Set*: *bifurcar si desbordamiento activado*) comprueba el desbordamiento o el bit V en el registro de estado, que puede ponerse a uno como resultado de la instrucción ADD (como ocurre en este ejemplo concreto). La condición de desbordamiento surge, como es evidente, a causa de que el resultado de una operación aritmética no se ajusta al tamaño de la palabra empleada en el operando. Si no llegáramos a detectar esta condición obtendríamos respuestas erróneas.

Una última observación sobre el registro de estado: su longitud es de 16 bits, y partes de cada byte son empleadas por el sistema de la siguiente manera:

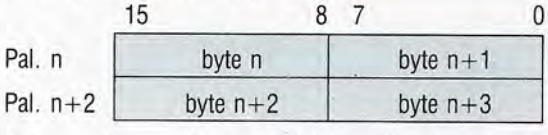


Los códigos de condición son el conjunto de bits individuales para indicar el resultado de la ejecución de la instrucción anterior: el bit V es un ejemplo de uno de estos códigos, que se estudiarán profusamente más adelante.

La sección de control de secuencias de nuestro modelo de microprocesador contiene el contador del programa, el registro que determina la dirección de la siguiente instrucción que ha de ser tomada de la memoria. Una vez tomada la instrucción, es decodificada por el control de secuencia para determinar el tipo de instrucción que la ULA ha de ejecutar y dónde están los operandos de origen y destino.

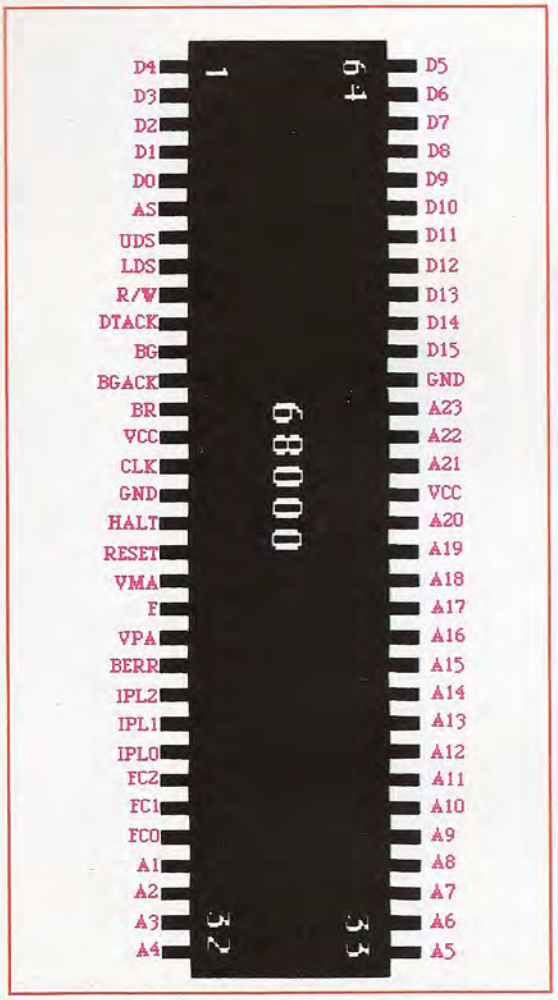
El contador del programa tiene 32 bits de largo pero las patillas a través de las cuales se conecta con el bus sólo permite el uso de 24 bits. Aun así, esto permite una gama amplísima de direcciones de memoria que llega hasta FFFFFF bytes. Cada dígito hexa corresponde a cuatro bits binarios, por lo que los 24 bits corresponden a un rango de direcciones de byte de 16 777 216. No obstante, se ha de notar que todas las instrucciones deben comenzar en una dirección impar (o frontera de palabra), lo que significa que será más comprensible pensar que hay un máximo de 8 388 608 palabras en ese rango de direcciones.

Ahora que estamos hablando de acceso a la memoria, será conveniente considerar cómo se organizan los datos en ella. Esto es necesario porque los bytes individuales son direccionables en esta máquina, y el direccionamiento de bytes en esta máquina es diferente, por ejemplo, del direccionamiento del PDP-11. El siguiente esquema ilustra el direccionamiento de la memoria del 68000 mostrando cómo la dirección impar proporciona el byte más significativo de la palabra.



Algunas instrucciones cambiarán directamente el contador del programa a fin de provocar una bifurcación, bien sea incondicional como BRA BACKHERE (donde BRA quiere decir *branch always* [bifurcar siempre] a una dirección representada simbólicamente por la etiqueta BACKHERE), bien condicional como en el ejemplo anterior, BVS OFLOW, en el que la bifurcación depende de si el bit V está a uno en la palabra del PS. En ambos casos, cuando ocurre una bifurcación o cambio en el flujo secuencial de ejecución normal, el contador del programa se modifica para apuntar a la siguiente instrucción a ejecutar.

Patillas a granel
Aquí están las patillas de comunicación del 68000. Obsérvense las 24 líneas de direcciones y las 15 de datos. A diferencia de su pariente más próximo, el 68008 (que posee un bus de direcciones de sólo 8 bits), el 68000 puede direccionar en modo directo dentro del ámbito del 000000 al FFFFFF hexa, es decir, del 0 al 16 777 216. Su primo mayor, el 68020, tiene un bus de direcciones de 32 bits completo. El 68000 es un perfecto ejemplo de microprocesador moderno: más de 100 000 circuitos lógicos se combinan para obtener la potencia de proceso de un miniordenador en un único chip



El conjunto de instrucciones

Los fabricantes de ordenadores suelen jactarse del número de instrucciones independientes que pueden soportar sus máquinas. Esto puede confundir a más de uno puesto que a menudo es necesaria una instrucción distinta para cada tipo de datos de operando o modo de direccionamiento. Esto es del todo verdadero en los antiguos ordenadores de ocho bits en los que, por ejemplo con el 8085, hay disponibles 63 instrucciones distintas de MOV, y cada una se sirve de los registros en una combinación diferente.

Más importante que el número absoluto de códigos de instrucción es la gama de instrucciones, sus objetos de datos y la flexibilidad de los modos de direccionamiento que pueden ser empleados con las instrucciones.

Motorola ha previsto un conjunto de instrucciones poderosísimo en lo que respecta a objetos de datos, que nos permitirá direccionar bytes, palabras y palabras largas con la mayoría de las instrucciones. Podemos realizar la multiplicación, división binaria y la aritmética BCD, así como la gama habitual de instrucciones de operaciones lógicas, control de programa y copia de datos.

No obstante, aun cuando se dispone de una gama muy útil de modos de direccionamiento, no podemos usarlos sin estudiar antes la instrucción. Por ejemplo, si el operando de destino para la instrucción MOVE es un registro de dirección, debemos emplear la instrucción MOVEA o bien LEA. En otras palabras, no podemos escribir MOVE D3,A6 pero sí que podemos MOVEA D3,A6. La razón es que al proporcionar un conjunto de instrucciones tan poderoso con respecto a los objetos de datos, se ha sacrificado una pequeña parte de la capacidad de direccionamiento.

En su totalidad el conjunto de instrucciones es muy potente en comparación con los micros de ocho bits (incluidos algunos miniordenadores y ordenadores centrales) con un excelente abanico de objetos de datos, pero con una flexibilidad decepcionante en los modos de direccionamiento utilizables con las instrucciones. En algo se compensa esto gracias a la densidad de algunas instrucciones (conocidas como instrucciones *rápidas*), donde la instrucción entera se codifica en una palabra de máquina.

Por ejemplo, para establecer a 25 un registro de datos escribiríamos:

```
MOVEQ#25,D3
```

que llevará 25 a D3 y la instrucción entera sólo ocupará una palabra, incluyendo el dato constante 25.

En el próximo capítulo estudiaremos con mayor detalle los modos de direccionamiento disponibles y cómo los podemos usar para acceder a los datos. Incluiremos algunos ejemplos de programas.

Por delante de su época

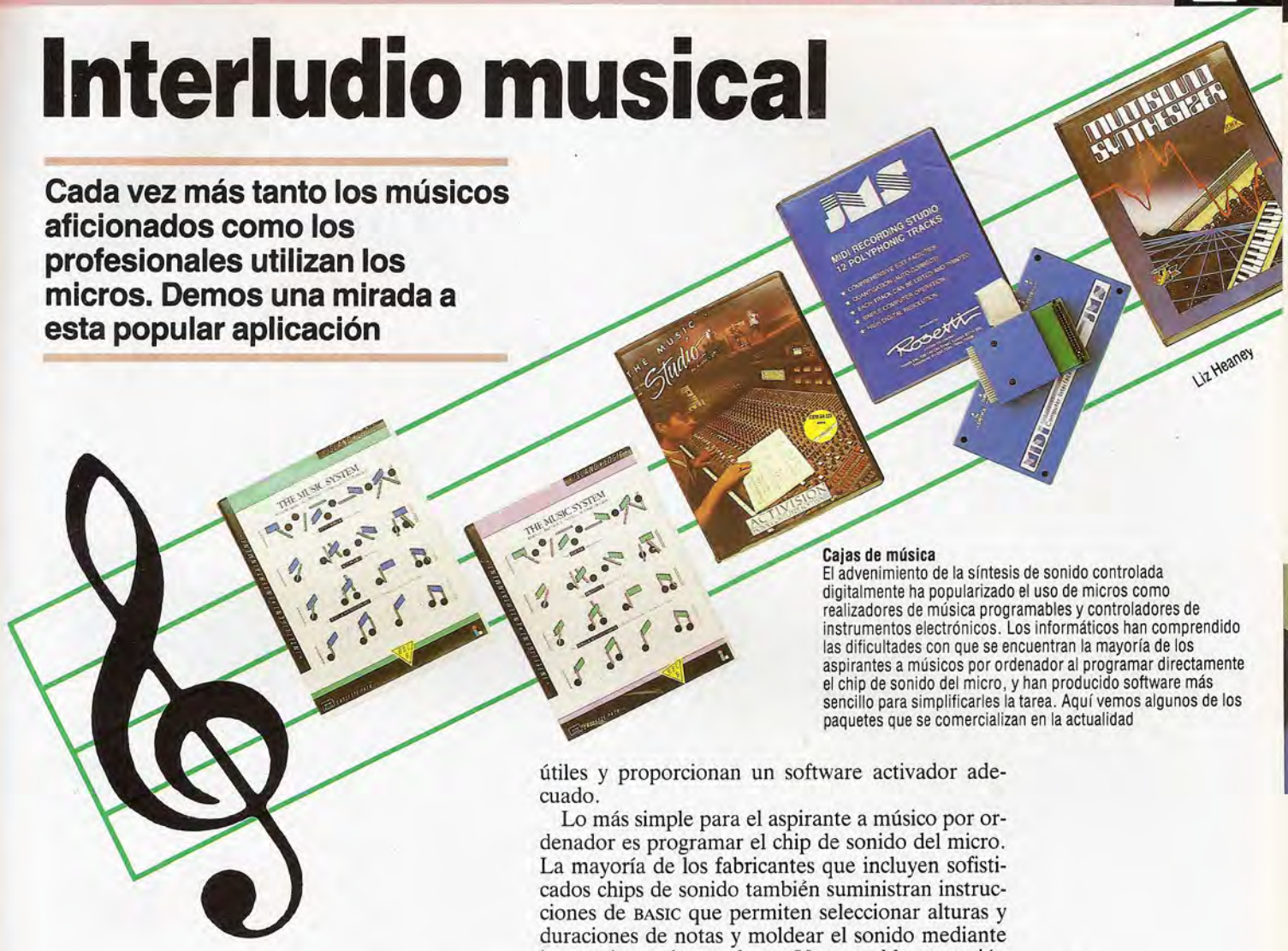
El Motorola 6809 llegó demasiado tarde como para causar impacto en el *boom* de los micros domésticos a finales de los años setenta y comienzos de los ochenta. El 68000 de 16 bits fue lanzado cuando muchos fabricantes estaban diseñando una nueva generación de máquinas, por lo que se ha incorporado a ordenadores como el QL de Sinclair, el Macintosh de Apple y el ST de Atari.

Desde el punto de vista del programador, es una delicia trabajar con este chip. El procesador tiene 17 registros de 32 bits, un bus de datos de 16 bits y un bus de direcciones de 24 bits, además del juego de instrucciones.

No obstante, y a pesar del impresionante acabado de muchos de los ordenadores que hasta hoy han incorporado el 68000, existen todavía problemas por resolver antes de que se logre utilizar su completo potencial. El problema básico hasta el momento ha sido el que el 68000 es tan adelantado en su tecnología respecto a otros chips que los fabricantes de ordenadores lo están empleando muy por debajo de sus posibilidades. Un ejemplo extremo es el QL de Sinclair. Este ordenador está basado en la versión 68008 del chip, que sólo tiene un bus de datos de ocho bits. Con lo cual, el QL no es, para muchas aplicaciones, mucho más rápido que sus predecesores de ocho bits. Sin embargo, el futuro del procesador está en el chip 68010, que presenta un bus de direcciones de 16 bits pero que ofrece el soporte de una memoria virtual. Esto permite que parte de la memoria resida en un dispositivo de almacenamiento de fondo más barato, permitiendo así el empleo de mucha más memoria de la que se podría disponer de otro modo. Otro chip de Motorola —el 68020— ha sido saludado como un ordenador de 32 bits en un chip que ofrece ¡el 97 % de la potencia de un ordenador central!

Interludio musical

Cada vez más tanto los músicos aficionados como los profesionales utilizan los micros. Demos una mirada a esta popular aplicación

Liz Heaney

Cajas de música
El advenimiento de la síntesis de sonido controlada digitalmente ha popularizado el uso de micros como realizadores de música programables y controladores de instrumentos electrónicos. Los informáticos han comprendido las dificultades con que se encuentran la mayoría de los aspirantes a músicos por ordenador al programar directamente el chip de sonido del micro, y han producido software más sencillo para simplificarles la tarea. Aquí vemos algunos de los paquetes que se comercializan en la actualidad

Los paquetes de música para microordenadores han estado disponibles desde la introducción de los primeros modelos PET y Apple a finales de los años setenta. Estos primeros ejemplares eran algo primarios y de utilidad limitada; pero ese mismo período también fue testigo de la introducción de sintetizadores de música electrónicos y portátiles igualmente elementales diseñados para ser usados en casa y en el escenario. Debido a que la música occidental está estructurada según reglas matemáticas precisas, basándose en el exacto equilibrio en la generación de sonidos en combinación con silencios, muchos músicos apreciaron rápidamente las posibilidades.

Inicialmente, la principal razón que motivó la inclusión de facilidades de sonido en los micros personales fue el proporcionar ruidos adecuados para los juegos. Pero algunos fabricantes, como Commodore, Acorn y Amstrad, comprendieron que existía una demanda de capacidades para hacer música, e incluyeron en sus máquinas chips bastante sofisticados para generar y moldear sonidos. Otros fabricantes, como Sinclair, continuaron ciñéndose a los generadores de tonos de tipo *beep*. Ésta es la razón por la cual hay tan pocos paquetes de música disponibles para el Spectrum, aunque existen algunos accesorios de hardware que añaden sistemas con circuitos para generación de tonos más útiles y proporcionan un software activador adecuado.

Lo más simple para el aspirante a músico por ordenador es programar el chip de sonido del micro. La mayoría de los fabricantes que incluyen sofisticados chips de sonido también suministran instrucciones de BASIC que permiten seleccionar alturas y duraciones de notas y moldear el sonido mediante instrucciones de envoltura. Una notable excepción la constituye Commodore, que no permite acceder a su chip de sonido (presumiblemente el mejor que existe en un micro personal de costo reducido) desde BASIC: este chip se debe programar utilizando una complicada serie de PEEKs y POKEs directamente en los registros internos del chip de sonido.

La mayoría de los chips de sonido disponen de tres voces, lo que permite tocar hasta tres notas al mismo tiempo. Por lo tanto, se pueden producir acordes y piezas a tres voces con facilidad. El control de envoltura de volumen afecta a la calidad de una nota y por lo general se define mediante una larga serie de números, que a su vez definen la altura y el tamaño de paso de cada sección de envoltura. El moldeo de las envolturas de tono permite la adición de efectos más sofisticados, tales como vibrato. Además, suele haber disponibles distintas formas de onda, como triangulares y cuadradas.

El principal problema de la programación de música desde BASIC reside en que es bastante lenta en comparación con el grado de precisión del control de tiempo necesario para producir una música que suene natural. Incluso el sistema de cola de sonido activado por interrupciones de Amstrad sólo consigue atenuar ligeramente este problema. La mejor solución (desde el punto de vista de la programación) es utilizar código máquina; pero, por supuesto, ello hace que las cosas se vuelvan sumamente difíciles excepto para los programadores más meti-

culosos. En consecuencia, numerosas empresas que han detectado el creciente interés por la programación de música han producido interfaces sencillas para personas que deseen tocar música.

Estos paquetes se dividen en dos tipos principales: los que incorporan las facilidades para generación de sonido del ordenador, y los que utilizan al ordenador para controlar un equipo externo de generación de sonido. El ejemplo más evidente de este último tipo es una red de teclados MIDI controlados por un micro. A la primera categoría la podemos subdividir en los paquetes que requieren un hardware adicional, como un teclado, y aquellos que se basan exclusivamente en software.

Existen a la venta muchos paquetes que convierten al ordenador en un instrumento que se puede tocar en tiempo real; es decir, que las pulsaciones del teclado se convierten inmediatamente en los sonidos audibles correspondientes. Los paquetes más económicos utilizan el teclado del ordenador, pero accesorios caros, como el Echo, proporcionan un medio más tradicional para tocar. Tales paquetes suelen incluir métodos para inicializar el chip de sonido para poder alterar la calidad del sonido.

Es aquí donde los microordenadores pueden contribuir enormemente al bagaje del músico, porque es posible escribir una pieza musical paso a paso utilizando la notación estándar o un lenguaje musical especializado de la misma forma en que uno utilizaría un procesador de textos. Esto ofrece la ventaja adicional de permitir que uno escuche y edite la pieza que está componiendo. Además, es posible que el ordenador analice y convierta una pieza tocada en tiempo real al sistema de notación que se esté utilizando. Una vez que el compositor está satisfecho, la pieza ya acabada se puede almacenar para futuras referencias e imprimir en notación estándar mediante una impresora matricial común.

En este caso, no existe ningún paralelismo con el sintetizador especializado, aparte del empleo de compositores o secuenciadores dedicados o el uso de un micro para el mismo fin a través de una MIDI, como la CX5M de Yamaha.

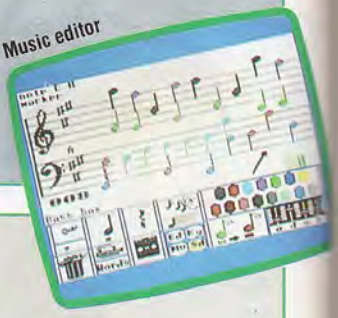
La mayoría de las personas a quienes interesa la programación se sentirán tentadas a programar música y efectos sonoros. El principal atractivo de la mayoría de los paquetes de música tiene una doble vertiente: la programación paso a paso de música permite que aun el más amateur de los usuarios produzca piezas que presenten un cierto grado de dificultad. El auténtico músico también se siente atraído por tales paquetes, ya que permiten la experimentación a través del familiar medio de la notación musical estándar de composición y forma. Sistemas como el CX5M, que combina facilidades de composición con un medio para controlar instrumentos musicales electrónicos externos, maximiza el potencial de este tipo de sistemas.

Para los usuarios de instrumentos MIDI, la inversión adicional, relativamente pequeña, que representan un micro personal y una unidad de disco les proporciona acceso a un método de control, grabación y reproducción simultáneos de hasta 16 instrumentos MIDI. Aunque la mayor parte de los instrumentos MIDI se basan en teclados y máquinas de ritmos, también están adquiriendo notable difusión y aceptación los controladores de guitarras e instrumentos de viento.

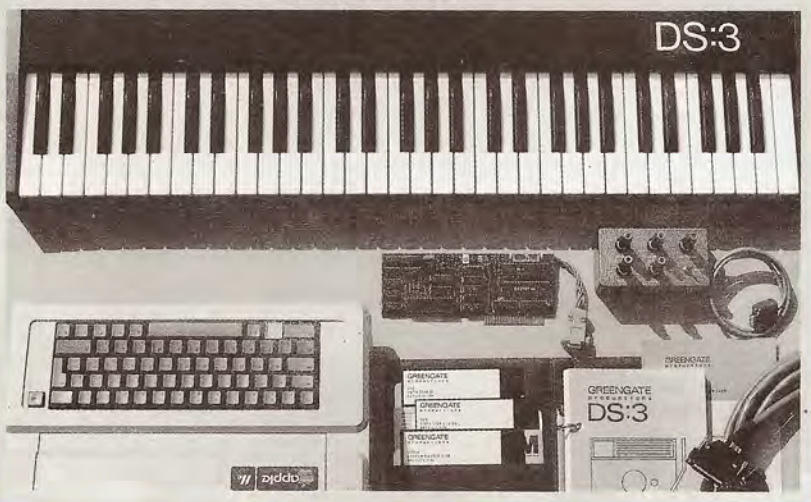
Componer música más fácilmente

La programación paso a paso es una facilidad que permite que aun el recién iniciado absoluto escriba sus propias melodías, que el ordenador puede volver a reproducir. Aunque es probable que haya tantas versiones de programación paso a paso como programas para composición musical, las características esenciales son las mismas.

Las notas musicales se entran en el ordenador a través de su propio teclado, y algunas veces las notas van sonando a medida que son introducidas. Las notas se visualizan luego en la pantalla, a menudo en el pentagrama musical estándar de cinco líneas. Una vez que se ha completado la pieza, el usuario puede editar la composición volviendo hacia atrás y alterando una nota, cambiando la clave de tiempo, etc. Este método de composición ofrece una gran ventaja: permite a los usuarios oír lo que están componiendo a medida que van entrando las notas, y pueden disponer de una reproducción instantánea. Además brinda a los aspirantes a compositores la posibilidad de oír sus melodías, las que quizá sean incapaces de ejecutar en un instrumento musical convencional

Un programa notable
El *Music studio* de Activision, activado por iconos, permite programar el chip de sonido, además de componer melodías que se pueden reproducir. El *Sound engineer* posee formato tipo sintetizador, ofreciéndole la posibilidad de tocar un instrumento o programar el suyo propio. El programa *Music editor* permite programar melodías paso a paso. Al igual que *Sound engineer*, las facilidades se seleccionan por medio de un cursor "batuta", y las composiciones se escriben en la notación musical estándar

Muestreo

El muestreo suele englobarse en la etiqueta general de síntesis de música, si bien en realidad es la conversión y almacenamiento de un sonido (una señal analógica) en una serie de valores digitales. Éstos pueden ser convertidos de nuevo a casi su valor analógico inicial a voluntad, para tocar, componer o secuenciar. Hasta hace poco tal "grabación" digital estaba limitada a sistemas exclusivos muy caros. Ahora se encuentra disponible en el mercado, a un precio asequible, el sistema DS3 para el Apple II, que permite el muestreo polifónico de cuatro notas. Se han desarrollado, a precios también muy convenientes, algunos sistemas monofónicos para el Commodore 64 (Autographics Microsound 64) y para el Spectrum (Ricoll Sound Sampler y Datel DSS)



Sonido sintetizado

Al igual que los ordenadores, los primeros sintetizadores eran máquinas enormes. Se basaban en circuitos de osciladores para producir uno o dos sonidos derivados de formas de onda de impulso simple, triangulares o aserradas, individualmente o en combinación. Otro sistema de circuitos moldeaba la envoltura de volumen de tales sonidos y proporcionaba diferentes tipos de filtros de señal para alterar el carácter de los sonidos, al eliminar de la señal frecuencias seleccionadas.

Esto dotaba a los sintetizadores de un cierto grado de versatilidad, pero su gran tamaño significaba que sólo podían ser instalados en estudios profesionales. Asimismo, su utilidad como instrumento musical era un tanto limitada, porque los sonidos que producían tendían a ser "chillones", tenues y descoloridos. Fue en estas máquinas donde nació la idea de sintetizadores como instrumentos de teclado tipo piano, en los que cada nota se pudiera marcar como "encendida" o "apagada" mediante la operación de un único interruptor debajo de la tecla correspondiente.

Estos primeros instrumentos portátiles eran monofónicos y operaban de acuerdo a los mismos principios que los voluminosos modelos de estudio, pero economizaban en peso, costo y tamaño, al emplear circuitos integrados; asimismo, seguían siendo enteramente analógicos.

Las técnicas digitales se introdujeron a comienzos de los años ochenta para estabilizar las frecuencias generadas por los osciladores analógicos, dado que éstos tendían a "desviarse" de la melodía. A ello le siguió enseguida la adición de circuitos de memoria para almacenar los ajustes del potenciómetro y los interruptores, permitiendo recuperar e implementar los parámetros para un sonido determinado en cualquier momento con sólo tocar una tecla. Esto es esencial para cambios rápidos de sonido durante una actuación en vivo.

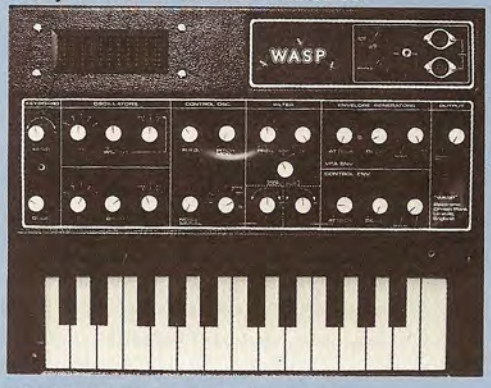
Inicialmente, debido al elevado costo de los chips de memoria, sólo se podían memorizar entre 4 y 12 sonidos. Los sintetizadores modernos tienen capacidad para recordar centenares de disposiciones distintas. Asimismo, la producción masiva de chips de sonido ha significado una drástica reducción de precios.

La digitalización de los valores de parámetros para cada circuito modelador de sonido permitió que muchos fabricantes recortaran aún más los costos eliminando los potenciómetros, mandos e interruptores convencionales en favor de una cantidad mínima de interruptores de membrana. Ello significó que se pudiera "llamar" un determinado parámetro y visualizar su valor actual como un número LED con teclas "+" y "−" para modificar el valor. De modo que hacia fines de 1982 los principales fabricantes tuvieron que acordar una interface y sistema de transferencia de datos estándar para que los sintetizadores se pudieran conectar entre sí y entre ordenadores con fines de control.

En agosto de 1983 se estableció un sistema estándar que se dio a conocer como MIDI. Éste en realidad es un estándar de software en virtud del cual las transmisiones MIDI poseen ciertos

Minisistema
El MiniMoog, que vemos en la fotografía, fue el primero de una serie de sintetizadores analógicos portátiles económicos basados en la tecnología del circuito integrado

significados estandarizados. El vínculo entre dos instrumentos especificados por la MIDI no es más que un sistema en serie asíncrono que ya se venía implementando con total éxito desde hacía varios años en sistemas de ordenador, permitiendo que los mismos se comunicaran con periféricos. Uno de los primeros modelos que apareció equipado con MIDI fue, asimismo, el primer sintetizador portátil totalmente digital, el Yamaha DX7, un sistema de ordenador especializado en la generación y control de sonidos musicales.



Las velocidades de transmisión que se especifican en la MIDI son tales que un micro personal estándar puede actuar como receptor, emisor o procesador intermedio de bytes MIDI. Por último, se ha sellado el vínculo entre la tecnología del ordenador y el equipo de sintetizador profesional. Existen a la venta numerosos paquetes que permiten conectar un micro con un sistema MIDI para permitirle actuar como la unidad de control maestra en un sofisticado sistema de composición, reproducción y grabación

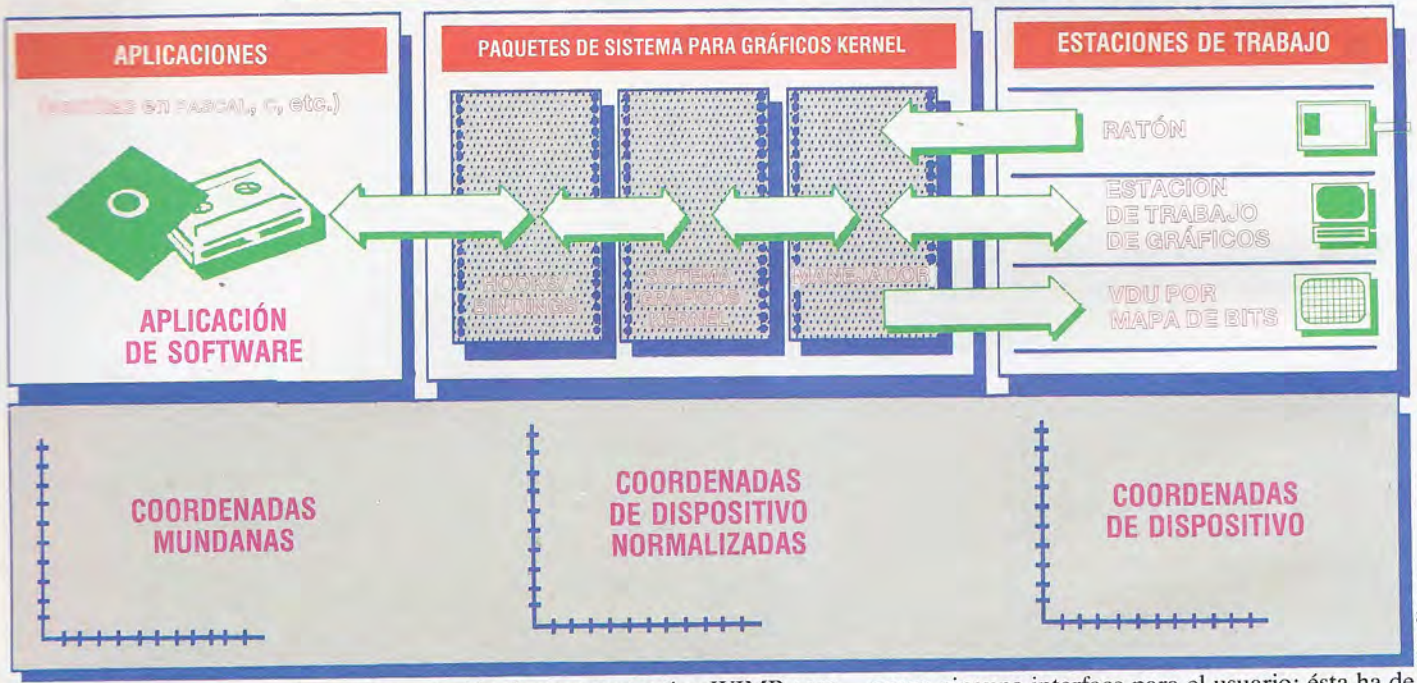
Yamaha DX7

Un sintetizador a su alcance
El desarrollo de sintetizadores de música y ordenadores personales ofrece algunos paralelismos. El sintetizador Wasp, que vemos aquí, puso el costo de los sintetizadores electrónicos al alcance de la mayoría de las personas, al prescindir del oneroso teclado mecánico y reemplazarlo por una económica membrana plástica. El Wasp apareció casi al mismo tiempo que los históricos ordenadores personales ZX80 y ZX81 de Sinclair, que también incorporaban teclados de membrana

Marcus Wilson-Smith

Saliendo de su concha

El sistema de gráficos «kernel» es una forma de abordar el problema de la portabilidad de las implementaciones de entornos WIMP



Mike Clowes

Bien coordinado
El GKS y otros sistemas de interface para gráficos de bajo nivel obtienen la portabilidad traduciendo las coordenadas del mundo real entradas por el usuario a coordenadas de dispositivo aceptables para el hardware. Mediante este enfoque, sólo necesitan depender de la máquina los manejadores de dispositivo

La implementación de un entorno tipo WIMP para un ordenador es una operación costosa, con un mercado limitado para el producto final. Además, no resuelve el problema clave de hacer que las aplicaciones gráficas sean totalmente portables. La portabilidad implica que el sistema operativo normal tenga un "caparazón" a su alrededor, controlando las ventanas de la pantalla de gráficos, el dispositivo apuntador y también la ejecución por separado de los programas seleccionados. Cuando termina cada operación, el control se vuelve a pasar al caparazón WIMP supervisor en vez de al sistema operativo normal.

En este sentido, el programa controlador en realidad es un sistema operativo, pero aun así puede llamar al sistema operativo nativo para núcleos de tareas de mantenimiento tales como tratamiento de archivos, etc. La diversidad del hardware de gráficos disponible no armoniza con un entorno coherente de esta naturaleza, pero en la actualidad hay varios esquemas para proporcionar caparazones de gráficos para una amplia gama de máquinas que están luchando por hacerse un sitio en el mercado.

En 1977 se publicaron las primeras especificaciones para un *sistema kernel para gráficos* (GKS) que permitiría la programación de gráficos de una forma independiente de la máquina. Ahora el GKS está disponible a través de varias fuentes, pero su complejidad y elevado precio limitan su utilización a los usuarios de universidades y de aplicaciones de CAD/CAM serias. Digital Research fue la primera empresa especializada en microordenadores que siguió con su caparazón GSX (*graphics system extension*), que se basaba en GKS pero no era compatible con el mismo.

El GSX es una ampliación al sistema operativo (el CP/M-86 de la propia DR), que se construye a medida para cada máquina. Una vez implementado, las aplicaciones pueden llamar a rutinas GSX sin preocuparse ni tener que saber nada acerca de las verdaderas capacidades de hardware, software y firmware utilizadas para llevar a cabo cualquier operación. El GSX descansa sobre el hardware tal como lo hace el OS, pero lamentablemente no posee ninguna interface para el usuario; ésta ha de ser realizada (¡con suma dificultad!) por el programador de aplicaciones.

Una empresa de software británica proporciona una biblioteca de rutinas para acceder a las facilidades GSX sin los problemas de escala, mapa de coordenadas y desbordamiento de capacidad, que constituyen las pesadillas del programador de GSX. Prospero Software suministra tanto compiladores de ISO PASCAL como de FORTRAN 77, y su biblioteca Prospect se puede enlazar con programas escritos en estos lenguajes y ejecutar sin ninguna alteración en cualquier máquina que posea una implementación GSX. Ello proporcionará activadores para una amplia variedad de dispositivos tales como entradas por ratón, tablilla y teclado, así como dispositivos de salida para impresora, plotter y VDU. La biblioteca Prospect accede a las facilidades GSX incluyendo trazado de puntos y líneas, sombreado y, si el dispositivo lo permite, escala y rotación de textos, anchura de línea y color.

Incluso con una interface para gráficos independiente de la máquina y una biblioteca de programas relativamente amable como el Prospect de Prospèro, la tarea de implementar gráficos exige un enorme esfuerzo de programación. Por cuanto concierne al usuario medio, la única interacción que se

puede dar es con una aplicación, tanto si utiliza gráficos como si no, y tanto si éstos son activados o no por GKS, GSX, Prospect, FORTRAN, PASCAL, etc.

Para poder implementar un sistema WIMP completo, la totalidad del caparazón debe proporcionar acceso a todos los recursos de un ordenador, no tan sólo a los gráficos y no sólo a través de aplicaciones individuales o programas de sistema. Tanto Digital Research como Microsoft están siguiendo este camino con GEM y MS-Windows, respectivamente. Aparte de IBM, otros fabricantes de hardware que están utilizando GEM y MS-Windows en sus máquinas son Apricot y RML (en el Nimbus).

El *sistema kernel de gráficos*, en el cual se basa el GEM, surgió del deseo de desarrollar un software para gráficos que fuera portable. Digital Research implementó el GSX como un subconjunto del GKS no estándar pero asequible y desde entonces ha llevado la idea adelante con el *administrador del entorno gráfico* (*graphics environment manager*: GEM). El GEM es, en consecuencia, producto de dos corrientes principales de desarrollo: el estándar GKS y la metáfora del escritorio SMALLTALK popularizada por el Lisa y el Macintosh de Apple.

Al igual que el CP/M proporciona un OS estándar a través del cual el software de aplicaciones puede controlar una configuración de hardware variable, un *kernel* para gráficos como el GEM debe envolver con un caparazón blando bien definido los dispositivos físicos de un sistema WIMP. Uno de los conceptos fundamentales es el de la estación de trabajo. Ésta puede ser casi cualquier dispositivo, desde un ratón o trazador/plotter hasta un sistema de E/S completo como un terminal de gráficos.

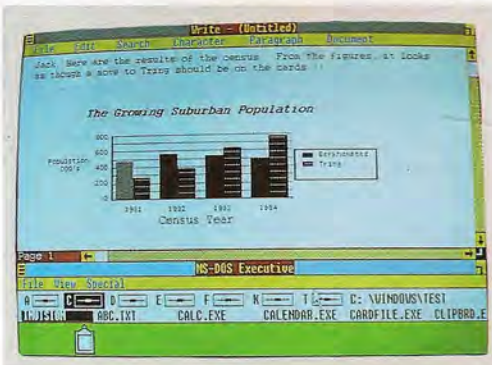
Por cuanto concierne al GEM, a cada dispositivo del sistema se accede exactamente de la misma forma, y las características y limitaciones de hardware quedan ocultas en el "activador de dispositivo" para esa estación de trabajo determinada. Por consiguiente, a un dispositivo de salida se le puede ordenar que dibuje una determinada forma, tanto si se trata de un plotter como de una pantalla de rayos catódicos. Debido a que la construcción física de cada dispositivo hardware afecta las capacidades y la resolución de los gráficos, se necesita un sistema de coordenadas independiente del dispositivo para aislar las aplicaciones gráficas del hardware. El método del GEM se basa en GKS, que define tres conjuntos de coordenadas.

Cada dispositivo, por supuesto, debe trabajar en función de su propio juego de *coordenadas de dispositivo*. Mediante analogía, una pantalla en color, por ejemplo, tendrá distintas escalas cartesianas en los ejes X e Y de acuerdo a la resolución de la VDU. Las coordenadas y atributos verdaderos del dispositivo son los únicos parámetros de un sistema que opera bajo GEM que varían en función del hardware disponible.

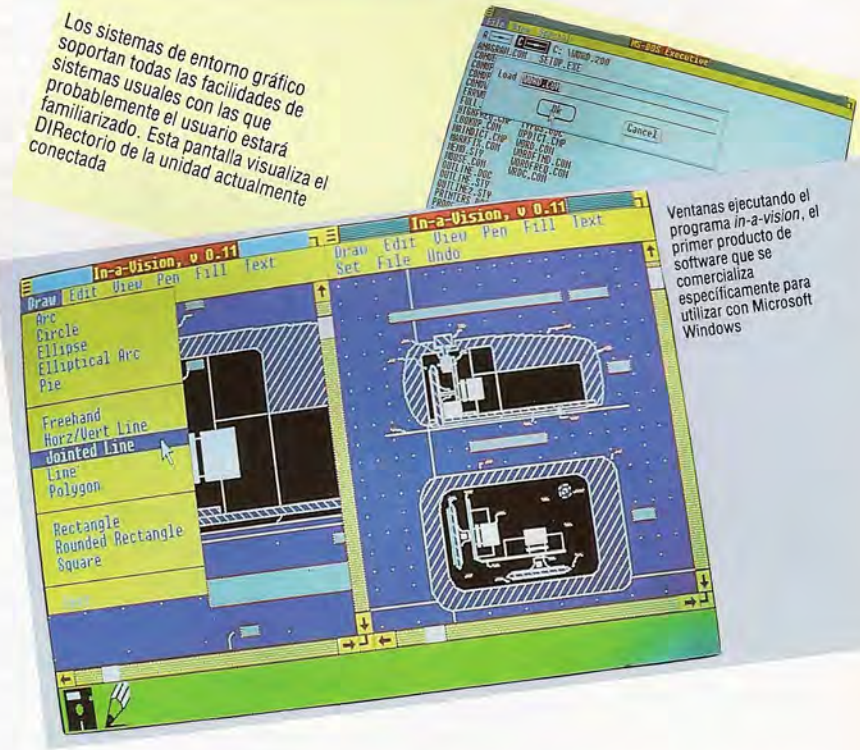
Las coordenadas del dispositivo se transforman a partir de un conjunto de coordenadas "normalizadas" que utiliza el GKS para todas las operaciones de gráficos. Este segundo conjunto de *coordenadas de dispositivo normalizadas* (NDCs: *normalised device co-ordinates*) a su vez se asocia con las coordenadas utilizadas por todo software de aplicaciones en el mundo real. Éstas se conocen como *coordenadas mundanas* (WCs: *world co-ordinates*) y el programador de aplicaciones las puede definir en cualquier escala conveniente. El nivel NDC del sistema GKS de tres hileras está normalizado a una escala de números reales entre cero y uno, de modo que la única limitación en cuanto a resolución es la aritmética de números reales.

En los grandes ordenadores centrales, la potencia y la velocidad disponibles palian la mayor parte de los problemas, pero con los microordenadores las cosas no son tan fáciles. La aritmética de números reales puede consumir mucho tiempo en comparación con la de números enteros, de modo que, en este sentido, el GEM se distancia del sistema GKS. La NDC del GEM responde a una escala entre cero y 32 767, permitiendo almacenar y manipular cada valor en una palabra máquina (16 bits), y permitiendo la minimización del consumo de tiempo introducido por las transformaciones.

El GEM paga un precio pequeño por la velocidad, debido en gran parte a las coordenadas normalizadas de enteros que utiliza, y ello se ve compensado con creces mediante el ahorro que se obtiene en el tiempo de desarrollo del software. Por ejemplo, una vez escrita la biblioteca inicial de rutinas gráficas para el GEM, cualquier aplicación podría incorporarlas y hacer un uso cabal de la capacidad incorporada para manejar los iconos, ventanas, etcétera.



El color puede suponer una gran diferencia en el tratamiento de imágenes y el resaltado de mensajes en la pantalla, aun cuando la aplicación sea para fines monocromáticos, como ésta, en la que una gráfica desarrollada con un programa se transfiere a un documento de tratamiento de textos



Los sistemas de entorno gráfico soportan todas las facilidades de sistemas usuales con las que probablemente el usuario estará familiarizado. Esta pantalla visualiza el DIRectorio de la unidad actualmente conectada

Ventanas ejecutando el programa *in-a-vision*, el primer producto de software que se comercializa específicamente para utilizar con Microsoft Windows

Visualización por ventana
MS-Windows es un sistema de entorno gráfico producido por Microsoft y confeccionado especialmente para alcanzar la compatibilidad con otras aplicaciones y sistemas Microsoft. Las pantallas que vemos aquí muestran el sistema en pleno funcionamiento. Tanto GEM como MS-Windows hacen un uso cabal del color, a diferencia del Apple Macintosh, aunque esto conlleva una disminución de la resolución de pantalla

Lenguaje comercial

Finalmente consideraremos las facilidades del COBOL para tratar archivos y tablas

Entre una tabla y una matriz existe una diferencia: el acceso a los elementos de una tabla se realiza mediante una clave KEY (parte de los datos de cada elemento de la tabla) y no directamente mediante un subíndice o índice. Además, normalmente cada elemento de la tabla es una estructura de registro, conteniendo una cantidad de campos.

Esta clase de estructura se puede manipular en algunos otros lenguajes (el PASCAL, p. ej.) mediante el uso de matrices de registros. El COBOL posee una facilidad para proporcionar tablas, que también se puede utilizar para proporcionar matrices simples. La característica esencial es la idea de que cualquier dato, excepto en nivel 01 o 77, se puede definir mediante repetición añadiendo la cláusula OCCURS nn TIMES (se produce nn veces), donde nn puede ser cualquier constante entera positiva, y TIMES se puede omitir. Por ejemplo, la siguiente define una matriz simple de 20 enteros:

```
01 matriz-simple.
   02 elemento-matriz PIC 9(5) OCCURS 20 TIMES.
```

Los elementos de la matriz son referenciados del modo normal en sentencias, como en elemento-matriz (5) o elemento-matriz (número-1), donde número-1 es un dato entero positivo. Tenga presente, no obstante, que la matriz en su totalidad también tiene un nombre y que, por lo tanto, las operaciones tales como MOVE se pueden llevar a cabo sobre la totalidad de la matriz al mismo tiempo.

Se puede definir una matriz bidimensional (o más grande) dividiendo aún más el campo repetido en un componente que se repita a sí mismo:

```
01 matriz-bidimensional.
   02 fila-matriz OCCURS 20 TIMES.
      03 elemento-matriz OCCURS 30 TIMES.
```

Los elementos de esta matriz son referenciados como elemento-matriz (3,4), por ejemplo, o como elemento-matriz (número-1,número-2); pero se puede referenciar cada fila, como en fila-matriz(6), o la matriz en su totalidad.

Se pueden definir estructuras más complejas (que hagan esta facilidad para tablas en vez de para meras matrices) combinando elementos repetidos de varios tipos con la facilidad del COBOL para subdividir datos. Por ejemplo, la siguiente definición es para una tabla de precios de diversos artículos del stock de una zapatería, que vienen en 20 números distintos:

```
01 tablas-stock.
   02 descripciones-números OCCURS 20 TIMES.
      03 número-inglés PIC 9V9.
      03 número-métrico PIC 99V99.
   02 cantidad-de-artículos-en-existencia PIC 999.
   02 artículos-en-existencia.
      03 artículo-del-stock OCCURS 500 TIMES,
                            ASCENDING KEY IS
                            número-stock
         04 número-stock PIC X(6).
         04 descripción-stock PIC X(20).
         04 precios-stock PIC 999V99 OCCURS
            20 TIMES.
         04 indicador-stock PIC X.
         88 en-existencia VALUE "S".
```

Observe cómo toda la información relevante se mantiene junta en una gran tabla. Cuando se repite un elemento del grupo, como artículo-del-stock en esta definición, también se repiten todos sus subcampos, de modo que podemos referirnos a número-stock (6) y a precios-stock (100,3). Asimismo, el nombre de la condición de nivel 88 puede llevar subíndice, de modo que podemos utilizar IF en-existencia (120).... por ejemplo. La cláusula ASCENDING (o DESCENDING) KEY es opcional; sólo se

utiliza cuando se ha de mantener la tabla ordenada y, entonces, sólo si se ha de utilizar el verbo SEARCH ALL. Es responsabilidad del programador mantener los elementos ordenados, dado que el COBOL no maneja esto de forma automática.

Los datos usados para indexar la tabla pueden ser cualquier elemento numérico, siempre que el valor que contenga sea un entero positivo. Pero por razones de conveniencia y eficacia, el COBOL proporciona un tipo de datos especial, INDEX, y con cada cláusula OCCURS podemos añadir INDEXED BY nombre-índice. Esto define un elemento-dato numérico que sólo se puede utilizar para almacenar valores enteros positivos, y para indexar la matriz con la cual se define. Se puede definir más de un índice para una tabla y se puede declarar cualquier elemento numérico como USAGE INDEX, en cuyo caso puede ser usado para indexar una matriz en uso o en aritmética de índices.

Los elementos de datos índice no se pueden utilizar en las sentencias aritméticas comunes, sino que poseen su propio verbo aritmético, SET:

```
SET nombre-índice TO valor-numérico.
SET nombre-índice UP BY valor-numérico.
SET nombre-índice DOWN BY valor-numérico.
```

donde el valor numérico puede ser una constante o cualquier elemento numérico ordinario.

Una de las principales funciones que es necesario llevar a cabo con una tabla (al contrario que en una matriz) es la búsqueda. El hecho es que, puesto que el acceso se determina por medio de un valor clave, es necesario buscar a través de la tabla con el fin de encontrar la entrada con una clave determinada. El COBOL proporciona esta facilidad directamente mediante el verbo SEARCH (buscar). Éste posee dos formas:

```
SEARCH nombre-tabla VARYING nombre-índice (o
  elemento-numérico)
  AT END sentencia-imperativa
  WHEN condición-1 sentencia-imperativa.
```

donde la cláusula VARYING y la cláusula AT END son opcionales con un número cualquiera de cláusulas WHEN. Esto provoca una búsqueda lineal de la tabla (observando a cada elemento de uno en uno, empezando por el primero), permitiendo que usted especifique la acción a emprender cuando se satisfaga una determinada condición o cuando la búsqueda haya concluido. Una sentencia imperativa es toda sentencia que hace que se tome una acción directa, sin ninguna alternativa. De modo que, por ejemplo, un IF no es una sentencia imperativa, pero un MOVE sí lo es.

La forma alternativa del verbo SEARCH es:

```
SEARCH ALL nombre-tabla
  AT END sentencia-imperativa
  WHEN condición-1 sentencia imperativa.
```

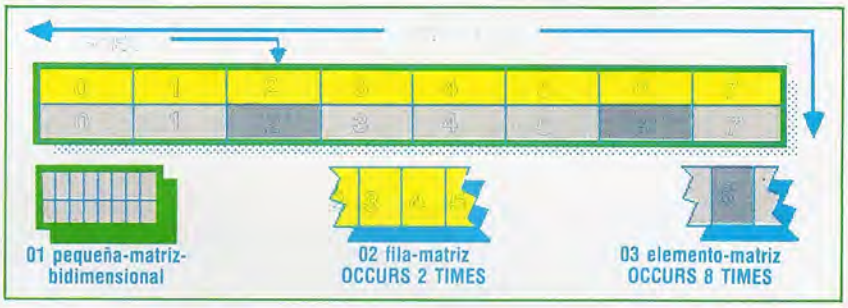
En este caso, sólo puede haber una cláusula WHEN y la comprobación debe tener la forma:

```
elemento-dato=valor
```

o un número de ellas conectadas mediante AND. La diferencia entre las dos formas del verbo es que en este caso se realiza una búsqueda binaria en la tabla, dividiéndola repetidamente por la mitad y decidiendo en qué mitad está el elemento requerido. Para que esto funcione, los valores de la tabla deben estar ordenados y en la definición de la tabla debe haberse especificado una cláusula ASCENDING o DESCENDING KEY.

Una de las principales características del COBOL que lo hacen tan adecuado para aplicaciones comerciales es la riqueza de facilidades para tratamiento de archivos. Todas las versiones de COBOL pueden manejar métodos de acceso secuencial, directo o indexado de una forma coherente y relativamente directa. Los archivos en dispositivos externos inicialmente se declaran en la sección de E/S (input-output section), el párrafo de control de ficheros (file-control) de la división de entorno (environment division). A cada archivo se le asigna un nombre de una sentencia SELECT independiente, que toma la forma:

Acceso indexado
La capacidad del COBOL de subdividir tipos de datos dentro de la división de datos (*data division*) permite al usuario construir complejos registros, que se pueden INDEXar y ser objeto de búsquedas (SEARCH). Además, el hecho de que la propia matriz y cada fila se declaren por separado significa que se pueden llevar a cabo MOVEs en bloque sobre los datos que encuadrarían a los diversos elementos constitutivos




```
SELECT nombre-archivo ASSIGN TO nombre-
dispositivo.
```

donde el nombre-archivo es un identificador del COBOL y el nombre-dispositivo es un identificador dependiente del sistema, que puede ser simplemente DSK o LPT, o un verdadero nombre de archivo del sistema. Si la versión de COBOL dada no requiere el nombre del archivo del sistema en este punto, en algún punto necesitará una cláusula adicional en la que poder especificar este nombre. Si en este punto no se da ninguna otra información, se asume que se trata de un archivo secuencial.

Existen numerosas cláusulas opcionales que permiten un control virtualmente completo (donde sea posible) de la forma en que realmente se almacena el archivo. Las dos más ampliamente utilizadas definen la organización (ORGANISATION) y el acceso (ACCESS). Hay tres opciones para ORGANISATION: SEQUENTIAL (secuencial, por defecto), RELATIVE (relativa, el nombre en COBOL para un archivo organizado por acceso directo mediante un número de registro) e INDEXED (indexada). Cuando la organización de un archivo es SEQUENTIAL, el único tipo de ACCESS permitido es SEQUENTIAL. Sin embargo, los otros dos tipos de organización también permiten el acceso RANDOM (aleatorio, directamente a un registro determinado) o DYNAMIC (dinámico), que es una combinación de RANDOM y SEQUENTIAL.

En el caso de archivos INDEXED, se debe declarar una RECORD KEY (clave de registro), que será uno de los campos del registro de datos utilizado como índice. En el caso de archivos RELATIVE, se debe declarar una RELATIVE KEY, que es un elemento de datos numérico empleado para almacenar el número de registro.

Cada nombre de archivo que se mencione en una sentencia SELECT se debe entonces definir en la sección de archivos (FILE SECTION) de la división de datos (DATA DIVISION), donde se da el nombre en

una declaración FD (*file definition*: definición de archivo) junto con otra información dependiente del sistema, como el tamaño del buffer, seguido por una definición de datos normal para la estructura del registro.

Aunque parezca un poco complicado declarar un archivo, las sentencias que se deben colocar en las divisiones de entorno (*environment*) y de datos (*data*) son estándares para la inmensa mayoría de las aplicaciones.

En la división de procedimientos (*procedure division*) hay varios verbos que se emplean específicamente para tratamiento de archivos. Se debe abrir cada archivo antes de utilizarlo:

```
OPEN nombre-archivo FOR modalidad-acceso.
```

donde la modalidad-acceso puede ser INPUT, OUTPUT o INPUT-OUTPUT; y, cuando se ha terminado con el archivo, debe ser cerrado:

```
CLOSE nombre-archivo.
```

La lectura y la escritura se realizan mediante los verbos READ (leer) y WRITE (escribir). Al igual que muchos verbos del COBOL, éstos tienen muchas opciones, incluyendo opciones para usar protección de registros y archivos en aplicaciones multiusuario. Sin embargo, en la mayoría de los casos sus usos son bastante directos; las formas básicas son:

```
READ nombre-archivo.
WRITE nombre-archivo.
```

La diferencia entre ambos reside en que READ requiere que el nombre del archivo y la ejecución de la sentencia llenen el registro de datos especificado para ese archivo en la sección de archivos (*file section*) de la división de datos (*data division*). La sentencia WRITE requiere el nombre de ese registro de datos y coloca su contenido en el dispositivo especificado en la sentencia SELECT.

Si el archivo se ha especificado con acceso secuencial, el READ leerá el siguiente registro del archivo y avanzará hacia el próximo. Se ha de leer la marca de fin del archivo para determinar si se ha llegado o no al final. La sentencia READ debe incluir la cláusula AT END, que especifica la acción a emprender al leer esta marca. La forma normal es utilizando un flag:

```
77 f-d-a PIC X VALUE 'N'.
88 fin-del-archivo VALUE 'S'.
.....
PROCEDURE DIVISION.
PÁRRAFOS-PRINCIPALES-DE-CONTROL.
   OPEN INPUT in-archivo, OUTPUT out-archivo.
   READ in-archivo AT END MOVE 'S' TO f-d-a.
   PERFORM párrafo-procesar-registro
      UNTIL fin-del archivo.
PERFORM cerrar.
   STOP RUN.
PARRAFO-PROCESAR-REGISTRO.
.....
   WRITE out-registro.
   READ in-archivo AT END MOVE 'S' TO f-d-a.
```

El WRITE escribirá el registro nuevo al final del archivo en cada ocasión.

Cuando el acceso al archivo es RANDOM o DYNAMIC, y la organización del archivo es INDEXED o RELATIVE, la lectura o escritura del archivo es un proceso con dos etapas. Primero, se debe colocar un valor adecuado en el campo de clave, y en el caso de un archivo RELATIVE, el número de registro requerido se coloca en la RELATIVE KEY antes de ejecutar la instrucción READ.

Los archivos INDEXED deben tener colocado un valor adecuado en el campo RECORD KEY. En lugar de la cláusula AT END, ha de haber una cláusula INVALID KEY que se ejecutará si no hay ningún registro que corresponda al valor clave dado.

En estos dos casos, el verbo WRITE sólo se utiliza para escribir registros nuevos. Un registro que se ha actualizado se escribirá utilizando el verbo REWRITE (reescribir), y se puede eliminar un registro utilizando DELETE (suprimir). Estos dos verbos requieren una cláusula INVALID KEY.

El acceso secuencial a un archivo siempre es posible utilizando:

```
READ nombre-archivo NEXT RECORD.
```

Ésta ha sido una introducción muy breve al tema del tratamiento de archivos en COBOL que, por ser una de sus facilidades más importantes, requeriría se le dedicara una serie completa. De hecho, aquí sólo hemos analizado brevemente la mayoría de las características del lenguaje.

Liz Heaney

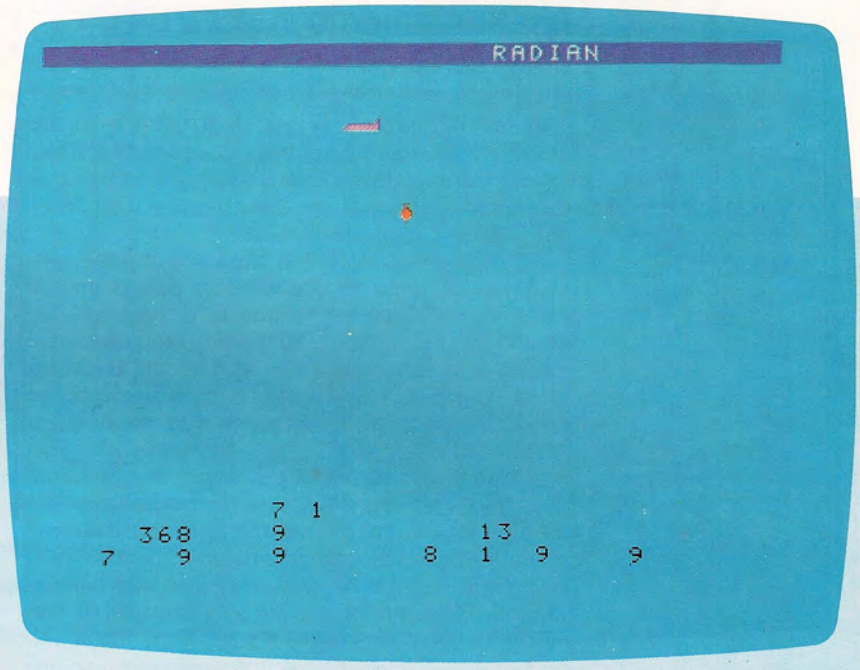
La gama COBOL

La popularidad de que disfruta el COBOL en la comunidad empresarial trae aparejada la existencia de gran número de implementaciones del mismo para micros, aunque (debido a las limitaciones de memoria) muy pocas para micros personales. Es esencial, por supuesto, un sistema basado en disco. Algunas de las versiones más conocidas son CIS-COBOL de Microfocus, Microsoft COBOL y RM/COBOL (que vemos aquí). Todas ellas operan bajo CP/M y MS-DOS. Los usuarios de máquinas personales que ejecuten CP/M (Memotech, Amstrad y Commodore, p.ej.) podrían, por supuesto, utilizar cualquiera de estos paquetes (haciendo concesiones de memoria), pero podría resultarles caro. Al igual que con el FORTRAN, una alternativa más económica es Nevada COBOL, de NewStar Software

Numerix

Para los usuarios de un microordenador EXL 100, he aquí un atractivo programa de juegos escrito por Pierre Monsaut, en el estilo del conocido "Bombardeo aéreo"

En esta ocasión debe bombardear con ayuda de su avión las cifras que se encuentran en la parte inferior de la pantalla, a fin de añadir sus valores al total de puntos. Para soltar una bomba, pulse una tecla cualquiera. Cada cifra alcanzada aumenta el número de bombas disponibles.




```
100 REM **********
110 REM * NUMERIX *
120 REM **********
130 DIM TB(40,22)
140 R=0
150 GOSUB 850
160 GOSUB 560
170 LOCATE (AY,AX)
180 CALL COLOR("1MC")
190 PRINT A$;
200 CALL COLOR ("1RC")
210 CALL KEY1(D3,D4)
220 IF D3<>255 AND BY=0 THEN BX=AX:BY=AY
    +1:NM=NM-1
230 IF BY<>0 THEN BY=BY+1
240 IF BY>22 THEN LOCATE (BY-1,BX):PRINT N$;:BY=0:IF
    NM<1 THEN 410
250 IF BY<>0 AND TB(BX,BY)<>0 THEN GOSUB 310
260 IF BY<>0 THEN LOCATE (BY-1,BX):PRINT N$;:LOCATE
    (BY,BX):PRINT B$;
270 IF BY=0 THEN FOR I=1 TO 5:NEXT I
280 AX=AX-1
290 IF AX<1 THEN AX=38:LOCATE (AY,1):PRINT M$;
300 GOTO 170
310 LOCATE (BY-1,BX)
320 PRINT N$;
330 LOCATE (BY,BX)
340 PRINT N$;
350 S=S+TB(BX,BY)*10
360 TB(BX,BY)=0
370 BY=0
380 NM=NM+.5
390 GOSUB 720
400 RETURN
410 CALL COLOR("1BC")
420 LOCATE (10,15)
430 PRINT "PUNTOS:";S;
440 IF S>R THEN LET R=S
450 LOCATE (13,15)
460 PRINT "RECORD:";R;
470 LOCATE (16,15)
480 PRINT "OTRA?";
490 CALL KEY1(D3,D4)
500 IF D3<>255 THEN 490
510 CALL KEY1(D3,D4)
520 IF D3=255 THEN 510
530 IF D3<>78 THEN 150
540 CLS
550 END
560 CLS ("BCC")
570 N$=CHR$(32)
580 A$=CHR$(100)&CHR$(101)&N$
590 AX=38
600 AY=3
610 B$=CHR$(102)
620 M$=N$&N$&N$
630 BX=0
640 BY=0
650 GOSUB 810
660 FOR I=1 TO 15
670 GOSUB 720
680 NEXT I
690 NM=20
700 S=0
710 RETURN
720 J=INTRND(9)
730 X=INTRND(37)+1
740 Y=INTRND(3)+19
750 IF TB(X,Y)<>0 THEN 730
760 CALL COLOR("1BC")
770 LOCATE (Y,X)
780 PRINT CHR$(J+48);
790 TB(X,Y)=J
800 RETURN
810 CALL CHAR (100,"0000000000003F7FFF00")
820 CALL CHAR (101,"000000000103FFFFFF00")
830 CALL CHAR (102,"002810387C7C7C381000")
840 RETURN
850 FOR I=1 TO 40
860 FOR J=18 TO 22
870 TB(I,J)=0
880 NEXT J
890 NEXT I
900 RETURN
```

© SYBEX

Rendimiento dinámico

Analizaremos la RAM dinámica e indagaremos en el funcionamiento interno de un chip de RAM

Ya sabemos que hay dos tipos de RAM principales: estática y dinámica. Mientras que la RAM estática se basa en un pequeño dispositivo lógico denominado flip-flop, la RAM dinámica retiene sus bits de datos como cargas electrónicas.

Ambas formas de RAM tienen ventajas y desventajas. El transistor que se utiliza para retener la carga de un único bit en una RAM dinámica es mucho más pequeño que el flip-flop empleado para retener la misma unidad de datos en una RAM estática. Las RAM dinámicas, por consiguiente, se pueden empaquetar de forma más densa en la superficie del chip. Sin embargo, las cargas que retienen los datos en una RAM dinámica se esfumarán cada pocos milisegundos y, por tanto, se requiere un sistema de circuitos adicional para "refrescarlas" periódicamente. Este problema no se plantea en el caso de la RAM estática y, en consecuencia, si el sistema sólo requiere una pequeña cantidad de memoria, la RAM estática es la opción más económica. Cuando se requieren memorias mayores, se justifica el gasto adicional que representa el sistema de circuitos de refresco y es más probable que se utilice la RAM dinámica.

Anteriormente hemos visto que la RAM estática normalmente se organiza en registros de ocho bits, teniendo cada chip ocho patillas de datos unidas a las ocho líneas del bus de datos. Las RAM dinámicas tienden a construirse en líneas diferentes; cada chip de RAM dinámica suele representar uno de los ocho bits de datos que componen una posición de la memoria, y ocho de tales chips, cableados en paralelo, constituyen los bytes de memoria.

En el primer diagrama vemos el 4116, con las conexiones de patillas típicas de un chip de 16 Kbits. En la actualidad muchos micros de ocho bits utilizan RAM dinámicas 4164 (64 Kbits) de este tipo para conseguir una memoria RAM de 64 Kbytes. Aunque podemos imaginarnos que el chip tiene un bit de "ancho" y 16 por 1 024 bits de "largo", en realidad está dispuesto en 128 filas por 128 columnas.

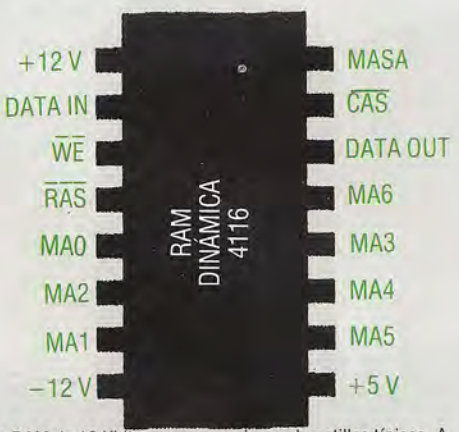
Si comparamos las patillas de salida de este chip con las de la ROM estática, podemos apreciar varias diferencias notables. En primer lugar, en lugar de ocho patillas de datos hay sólo dos, denominadas *data in* y *data out*. Cabría esperar que un chip de 16 Kbits requiriera 14 bits de direcciones para seleccionar cada bit de modo exclusivo, pero en realidad hay sólo siete. También hay presentes otras dos líneas hasta ahora desconocidas: $\overline{RAS}$ y $\overline{CAS}$, que son la *"strobe" de dirección de fila* y la *"strobe" de dirección de columna*, respectivamente. Estas dos señales de temporización permiten presentar la dirección en dos mitades (de allí las siete patillas de dirección en lugar de las 14 esperadas). La línea $\overline{RAS}$ también sirve para refrescar la RAM dinámica.

El proceso de refresco consiste en leer los datos de la RAM dinámica y escribirlos de nuevo para restablecer la carga. En nuestra RAM de ejemplo, esto se puede hacer de fila en fila. Por lo tanto, sólo se necesitan 128 operaciones para refrescar todo el chip. Si bien el refresco suele reducir la velocidad del procesador al demorar los accesos a la memoria durante los ciclos de refresco, es probable que la demora sea apenas del 5%.

Anteriormente hemos mencionado ya que los 14 bits de direcciones necesarios para una memoria de 16 Kbytes se pueden presentar en nuestro ejemplo de RAM dinámica en dos partes de siete bits. Esto se consigue a través de un chip lógico externo que junta las líneas de dirección de orden *low* y *high* con la señal de temporización $\overline{CAS}$. El segundo diagrama muestra una disposición simplificada en donde las líneas de direcciones de orden *low* (A0 a A6) están conectadas a las patillas de direcciones del chip cuando $\overline{CAS}$ está *high* y las líneas de direcciones de orden *high* están conectadas cuando $\overline{CAS}$ está *low*. De modo que en el chip de RAM propiamente dicho hay dos cerrojos (*latches*) que tienen la capacidad de "congelar" los datos que entran en ellos y, por lo tanto, la dirección de fila se toma cuando $\overline{CAS}$ está *high* y la dirección de columna se toma cuando $\overline{CAS}$ está *low*. La colocación de estos valores tratados con *latches* a través de decodificadores de 7 a 128 permite acceder al bit direccionado.

El último diagrama muestra cómo se conecta una memoria RAM dinámica de 16 Kbytes a las líneas de direcciones, de datos y de lectura/escritura del procesador. La línea $\overline{CAS}$ se conecta en paralelo a cada una de las ocho RAM y al chip lógico de direccionamiento. $\overline{RAS}$ y $\overline{WE}$ también están conectadas a las ocho RAM. Las patillas *data in* y *data out de* cada RAM están cableadas entre sí y conectan con una de las ocho líneas del bus de datos.

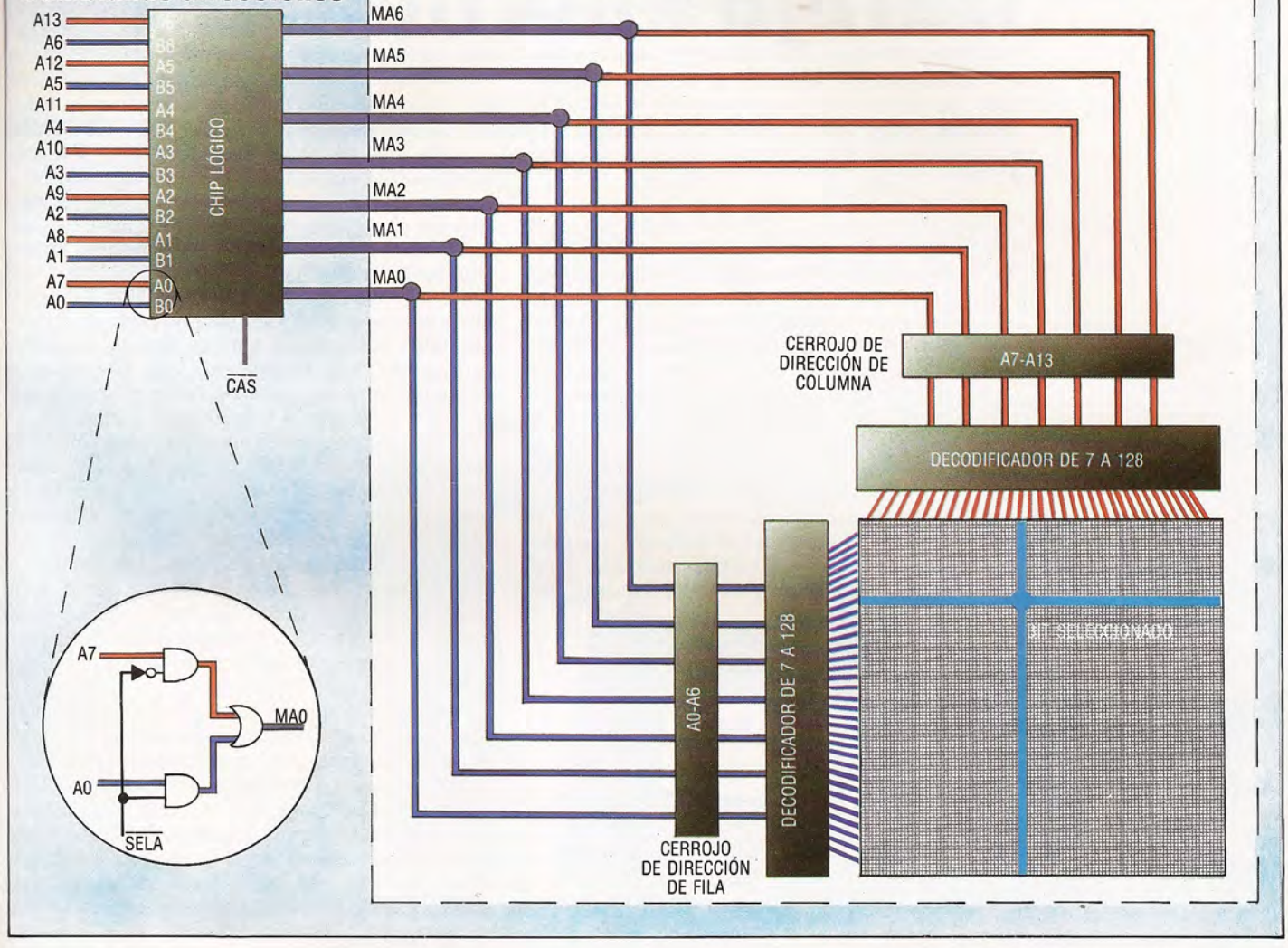
RAM dinámica



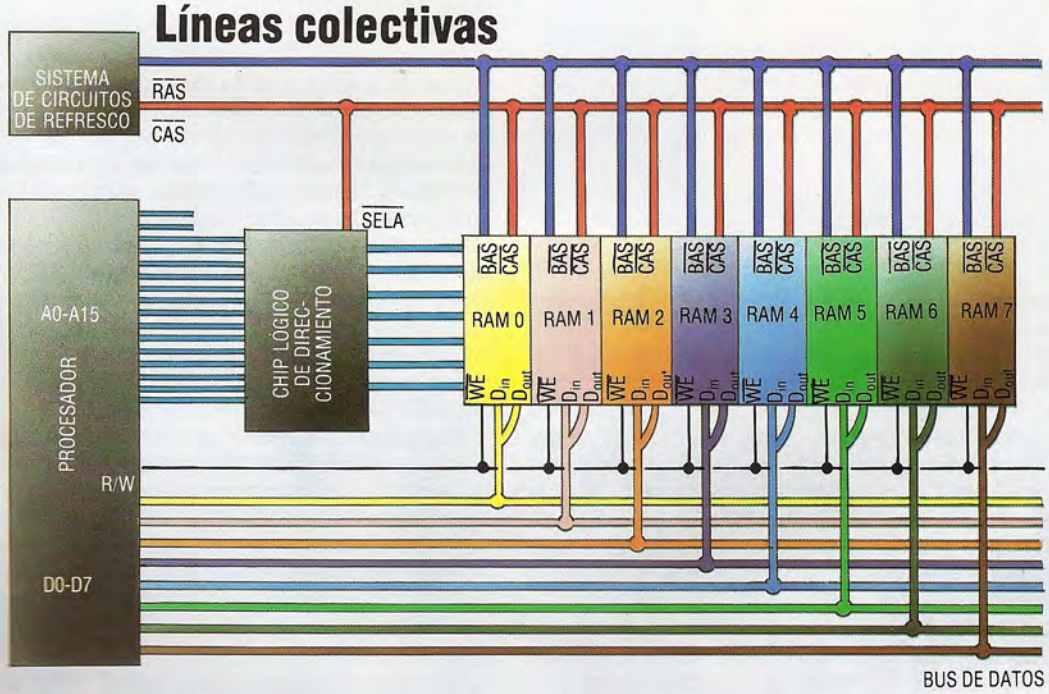
Esta RAM de 16 Kbits posee conexiones de patillas típicas. A diferencia de una RAM estática, una RAM dinámica normalmente corresponde a un único bit de la palabra de datos, de ahí las líneas *data in* y *data out* individuales. Sólo está presente la mitad de las líneas de dirección que cabría esperar, simplificando considerablemente el cableado de tales RAM

Mike Clowes

Cambiando direcciones



Líneas colectivas



Líneas compartidas
Las 14 líneas de dirección necesarias para seleccionar cada bit en una RAM de 16 Kbits se utilizan para producir direcciones de columna y fila de siete bits. Para simplificar el sistema de circuitos, el chip propiamente dicho posee sólo siete patillas de dirección y la conmutación de las partes de orden *low* y *high* de la dirección de entrada normalmente las manipula un chip lógico externo (utilizando CAS como señal de sincronización). El diagrama muestra la sencilla disposición de lógica combinatoria necesaria para conmutar las dos mitades de la dirección

Figuras de ocho
Se puede construir una memoria de 16 Kbytes cableando ocho chips 4116 en paralelo de modo que compartan las líneas de dirección, R/W, RAS y CAS. En dicha disposición, cada RAM dinámica estaría conectada a un único bit del bus de datos

Baraja informatizada

Iniciamos un nuevo proyecto en que crearemos un programa para jugar a un conocido juego de cartas: el veintiuno

Los juegos de naipes son ideales para implementarlos en un ordenador. Sus reglas de juego y estrategias, rígidamente definidas, con frecuencia se basan en gran medida en el análisis matemático. El veintiuno, por ejemplo, es especialmente fácil de programar y las reglas son muy directas, lo que hace que codificarlo para el ordenador sea una tarea sencilla. Sin embargo, los principios de la creación de una baraja de naipes, la visualización de éstos y la evaluación de las manos, ilustran ampliamente los tipos de problemas que suelen plantearse.

En primer lugar, encarguémonos de preparar la baraja y visualizar los naipes individuales, incorporando listados separados para el Commodore 64, el Spectrum, el BBC Micro y la gama Amstrad. Sin embargo, por el momento sólo ofreceremos la versión Commodore.

Kevin Jones

El método más sencillo de representar una baraja de naipes es retenerla como una matriz bidimensional. En nuestro juego se DIMensiona la matriz DK(,) de modo que tenga 52 elementos de largo y dos elementos de ancho. Dado que en una baraja normal hay 52 naipes (sin contar los comodines), es evidente por qué necesitamos 52 elementos en una dirección. Tomando un naipe individual, vemos que posee dos propiedades que lo hacen exclusivo dentro de la baraja: su valor (as, 2, 3, etc.) y su palo (corazones, diamantes, tréboles y picas). Por consiguiente, el valor y el palo de cada carta se retienen separadamente dentro de la matriz, utilizando la segunda dimensión.

Para que la matriz pueda ser numérica, hemos empleado los números del 1 al 13 para retener el valor del naipe, de modo que 1 es el as y 13 es el rey, y un número del 1 al 4 para representar el palo. La matriz de la baraja y otras matrices utilizadas para representar los naipes y sus posiciones se deben inicializar al comienzo del programa, lo que se realiza mediante la subrutina de la línea 500.

El método que hemos empleado para simular la barajada de naipes utiliza un par de bucles anidados. El bucle exterior va contando a través de los cuatro palos y el bucle interior va contando a través de los 13 naipes de cada palo. Por tanto, las dos variables contadoras de los bucles representan el palo del naipe y el valor del naipe a entrar en la matriz de la baraja dentro del bucle interno. Todo cuanto necesitamos hacer es seleccionar el punto en el cual deseamos colocar el naipe actual dentro de la baraja.

Puesto que se supone que el mazo está barajado, nuestra rutina selecciona el punto de entrada a éste de forma aleatoria; pero aquí hay un pequeño obstáculo: es posible que la posición seleccionada en la matriz de la baraja ya esté ocupada. Para sortear este problema, una pequeña rutina dentro del bucle interno comprueba el punto de entrada seleccionado al azar: si no está vacío, incrementa el punto de entrada (dando toda la vuelta hasta la parte de arriba de la baraja si fuera necesario) hasta hallar un espacio vacío. Al salir de nuestra estructura de bucles anidados, nuestra baraja contendrá 52 naipes distintos dispuestos por un orden aleatorio.

Para visualizar un naipe, todo cuanto necesitamos es su valor y su palo. Sin embargo, hemos de interpretar estos dos datos para producir el patrón del naipe y las etiquetas de los ángulos del mismo. Si bien para la mayoría de los naipes la etiqueta de los ángulos corresponderá directamente al valor de cada carta, el As, el Valet, la Dama y el Rey necesitarán etiquetas A, J, Q y K. Asimismo, hemos de hacer una pequeña trampa representando el 10 como T, para que la etiqueta se pueda visualizar como una única columna en el naipe. La forma más sencilla de manejar todo esto es preparando una matriz en serie de números de tarjeta, CN$(), para retener las etiquetas de los 13 naipes.

El patrón del naipe es un poco más difícil. Por razones de simplicidad, representaremos todas las imágenes de las cartas como si fueran ases, es decir, visualizando un solo símbolo del palo en el centro del naipe. Si usted observa una baraja de naipes, verá que todos los patrones son formas geométricas regulares y, de hecho, es bastante fácil diseñar una plantilla en la cual tengan cabida todos los patrones.

Hay tres columnas y siete filas que pueden retener símbolos de palos, lo que nos da 21 posiciones

de símbolo en total, y hay varios métodos de retener patrones. Aquí hemos optado por retener cada patrón de naipe como una serie binaria de 21 elementos, usando 1 para representar un símbolo visualizado y 0 cuando no haya presente ningún símbolo. Las definiciones para los 13 naipes están retenidas en sentencias de datos en la línea 2100 y se leen en la matriz CD$(). Ahora podemos analizar la rutina de visualización de naipes en sí misma.

En la versión para el Commodore 64 que ofrecemos aquí no hay ningún método de situar directamente el cursor en un punto dado de la pantalla. Por lo tanto, el programa utiliza una subrutina en la línea 900 para posicionar el cursor en un punto definido por TX y TY.

La subrutina de la línea 1050 imprime la silueta del naipe usando caracteres predefinidos. Luego la subrutina de la línea 1100 añade los detalles del naipe empleando los números del valor y palo del mismo, que se han pasado como CN y SU. La primera tarea es seleccionar el símbolo del palo de una serie de los cuatro símbolos definida en la rutina de inicialización. El color del naipe también se puede hallar fácilmente comprobando el número del palo. Puesto que los palos están dispuestos por orden de corazones, diamantes, tréboles y picas, simplemente necesitamos establecer el color del naipe en negro si el número de palo es mayor que dos y, de lo contrario, establecerlo en rojo.

Ahora se puede imprimir el patrón a partir de la descripción binaria retenida en CD(CN). Se utilizan un par de bucles anidados para recorrer cada una de las siete filas, tomando la serie binaria en grupos de tres y ensamblando una serie de espacios y símbolos de palo que constituyen la fila actual con la que se está trabajando. Tras imprimir el patrón del naipe, podemos añadir las etiquetas de los ángulos posicionando el cursor en el ángulo e imprimiendo CN$(CN).

En esta etapa veremos cómo se posicionan los naipes. Durante el juego, se les repartirán cartas al jugador o al ordenador y se las colocará de forma tal que queden superpuestas pero desplazadas en dos unidades horizontalmente y una unidad verticalmente. Para llevar el registro de la posición en la cual se ha de imprimir el naipe siguiente, se utilizan las matrices X() e Y(). El primer elemento retiene la posición del siguiente naipe para la mano del jugador y el segundo la posición del siguiente naipe para el ordenador. De este modo, al comenzar la rutina de impresión de naipes el cursor se desplaza hasta X(PL), Y(PL), donde PL es el número de jugador (1 o 2). Al final de la rutina, X(PL) e Y(PL) se incrementan en 2 y 1, respectivamente, listas para repartir el siguiente naipe a ese jugador.

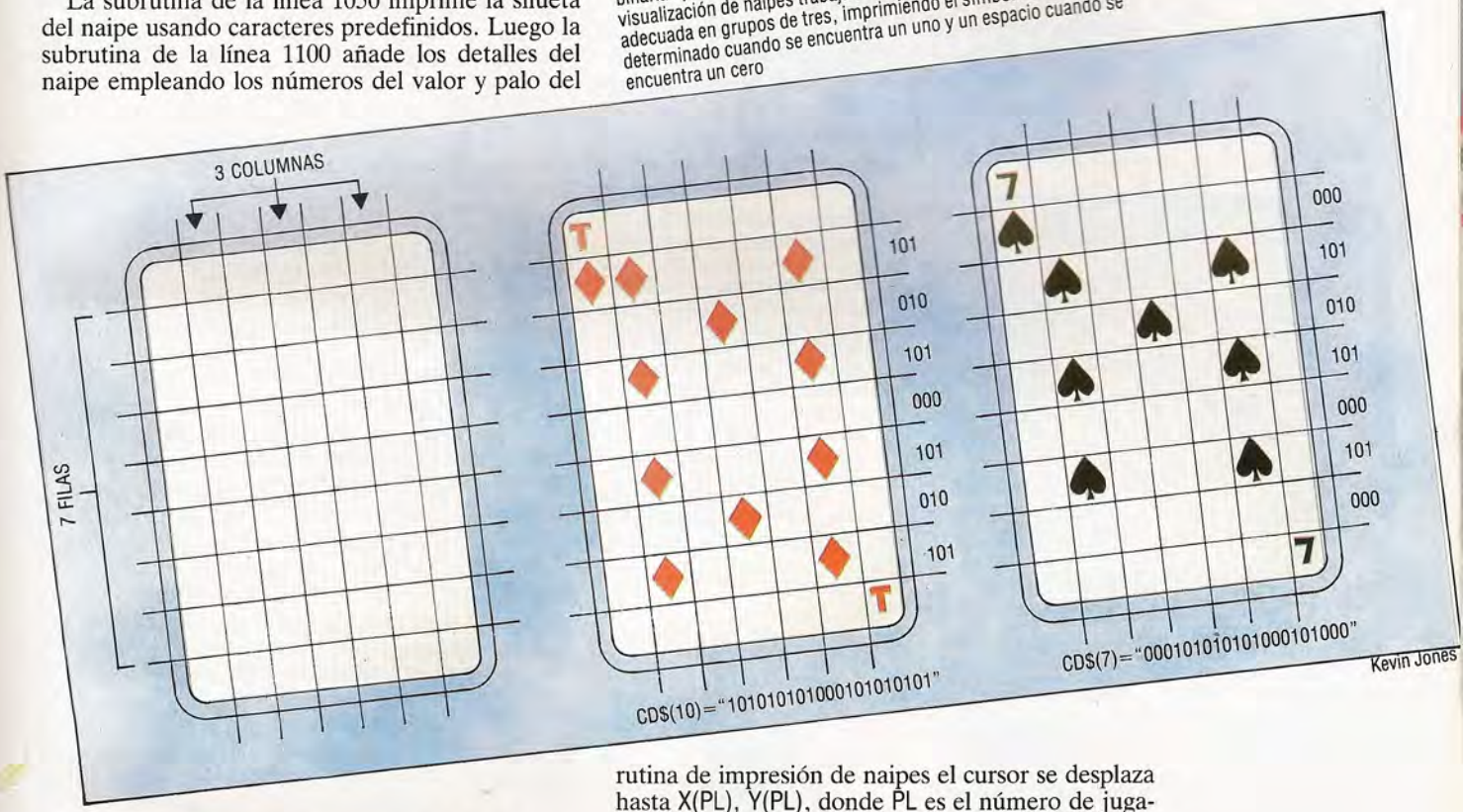
En la versión del juego que hemos programado, el ordenador juega como si fuera la banca, de modo que el primer naipe se le reparte boca abajo. Por consiguiente, necesitamos una pequeña rutina para visualizar el reverso de esta carta. La subrutina de la línea 1200 maneja esto llamando primero a la rutina de silueta de naipes y rellenando luego el interior utilizando un carácter de tablero.

La última rutina de esta sección nos permite repartir naipes de la parte de arriba de la baraja y visualizarlos. La breve subrutina de la línea 1300 es la encargada de esto. Toma los elementos de la matriz de la baraja que corresponden a la parte de arriba del mazo y los coloca en CN y SU, lista para llamar a la rutina de visualización de naipes.

La forma más obvia de manipular el reparto sería tomar los primeros elementos de la matriz de la baraja, desplazar todos los otros elementos un lugar hacia adelante y colocar los elementos que se acaban de quitar en DK(52,1) y DK(52,2). En realidad,

Creación de una carta

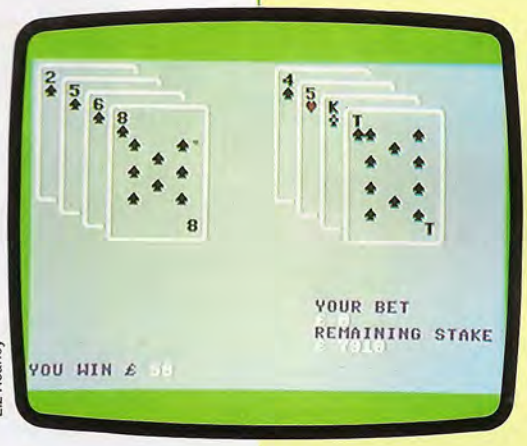
Los patrones para los 13 tipos de naipe se almacenan como series binarias. Utilizando una plantilla de tres columnas por siete filas, nuestro diagrama ilustra cómo se han creado las series binarias para un patrón de "diez" y "siete". La rutina de visualización de naipes trabaja a través de la serie binaria adecuada en grupos de tres, imprimiendo el símbolo del palo determinado cuando se encuentra un uno y un espacio cuando se encuentra un cero



todo este movimiento de elementos de la matriz lleva mucho tiempo y, en última instancia, es innecesario. En cambio, podemos utilizar una variable, DP, que señale hacia el "tope" del mazo y se incremente cada vez que se reparte un naipe, pasando de la posición 52 a la 1 si fuera necesario. Para repartir un naipe, por lo tanto, tomamos el elemento DK(DP,1) como el número del naipe y DK(DP,2) como el número de palo.

Al objeto de ver el efecto de nuestro trabajo hasta este momento, se pueden añadir las siguientes líneas para llamar a las diversas rutinas de modo que tanto al jugador como al ordenador se les repartan cinco cartas:

```
60 PL=1:FOR C=1 TO 10
70 PL=3-PL:REM NUMERO DEL JUGADOR
80 FL=0:GOSUB 1300:REM REPARTIR NAIPE
90 INPUT'PULSAR RETURN PARA
   SIGUIENTE NAIPE';RESP$
100 NEXT C
```



Liz Heaney

Necesitas manos...
Nuestro programa de veintiuno visualiza los naipes de las manos del jugador y de la banca en las mitades izquierda y derecha de la pantalla. El desplazamiento hacia un lado y hacia abajo de cada naipe recién repartido permite ver todos los valores de los naipes de cada mano. En esta etapa del proyecto tan sólo se podrá visualizar los naipes. Los avisos, mensajes y visualizaciones de apuestas serán el tema de futuros capítulos

Visualización de naipes y barajada

Programa principal:

```
10 REM **** VEINTIUNO COMMODORE 64 ****
20 GOSUB 500:REM INIC MATRICES ETC
50 REM **** BUCLE DEL JUEGO ****
55 GOSUB 600:REM INIC JUEGO
```

Inicialización de matrices:

```
500 REM ***** INIC MATRICES ETC ****
510 SP$="":DW$="":FOR I= 1 TO 25:DW$=DW$+
    CHR$(17):SP$=SP$+ "":NEXT I
511 SP$=SP$+LEFT$(SP$,14)
512 BK$="":LI$="":FOR I=1 TO 7:LI$=LI$+
    CHR$(195):BK$=BK$+ CHR$(166):NEXT I
513 BR$=CHR$(194)
515 DIM X(2),Y(2):REM POSICIONES SIGUIENTE NAIPE
520 SU$=CHR$(211)+ CHR$(218)+ CHR$(216)+
    CHR$(193):REM TIPOS DE PALO
530 DIM CN$(13):FOR I=1 TO 13:READ CN$(I):NEXT:REM
    LEER DATOS NUMEROS
540 DIM CD$(13):FOR I=1 TO 13:READ CD$(I):NEXT:REM
    LEER DATOS PATRONES
560 DIM DK(52,2):REM PREPARAR MATRIZ BARAJA NAIPES
570 GOSUB 3000:REM BARAJAR MAZO
580 POKE 53280,5:POKE 53281,15:REM COLORES PANTALLA
590 RETURN
600 REM **** INIC JUEGO ****
605 PRINT CHR$(147):REM BORRAR PANTALLA
620 X(1)=0:Y(1)=0:X(2)=20:Y(2)=20
630 RETURN
```

Rutina de visualización de naipes:

```
900  REM **** PRINT AT ****
910  PRINT CHR$(19);:PRINT
     LEFT$(DW$,TY);TAB(TX);:RETURN
1000 REM **** VISUALIZAR NAIPE ****
1010 GOSUB 1050:GOSUB 1100:RETURN
1050 REM **** NAIPE EN BLANCO ****
1055 TX=X(PL):TY=Y(PL):GOSUB 900:REM
     POSICION
1060 PRINT TAB(TX);CHR$(5);CHR$(213);LI$:CHR$
     (201)
1070 FOR I=1 TO 9:PRINT TAB(TX);BR$;"";BR$:
     NEXT I
1080 PRINT TAB(TX);CHR$(202);LI$;CHR$(203)
1090 RETURN
1100 REM **** DETALLES NAIPE ****
1120 TX=X(PL)+ 2:TY=Y(PL)+ 2:GOSUB 900:REM
     POSICION
1125 CT$=MID$(SU$,SU,1):REM SELECCIONAR TIPO
     PALO
1127 CO$=CHR$(28):IF SU>2 THEN CO$=CHR$(144):REM
     SELECCIONAR COLOR
1128 PRINT CO$;
1130 FOR I=1 TO 19 STEP 3
1140 CC$=MID$(CD$(CN),I,3):CL$=""
1142 FOR J=1 TO 3:C$=CHR$(29)+ CHR$(29)
1144 IF MID$(CC$,J,1)="1" THEN C$=CT$+
     CHR$(29)
1146 CL$=CL$+C$:NEXT J
1150 PRINT TAB(X(PL)+ 2);CL$:NEXT I
1160 REM **** AÑADIR ETIQUETAS ANGULOS ****
1170 TX=X(PL)+1:TY=Y(PL)+ 1:GOSUB 900:PRINT
     CN$(CN):REM NUMERO
1180 TY=TY+ 1:GOSUB 900:PRINT CT$:REM PALO
1190 TX=X(PL)+ 7:TY=Y(PL)+9:GOSUB 900:PRINT
     CN$(CN):REM NUMERO DE ABAJO
1192 X(PL)=X(PL)+ 2:Y(PL)=Y(PL)+ 1
1195 RETURN
1200 REM **** VISUALIZAR REVERSO NAIPE ****
1210 GOSUB 1050:GOSUB 1250:RETURN
1250 REM **** REVERSO NAIPE ****
1255 TX=X(PL):TY=Y(PL)+ 1:GOSUB 900:REM
     POSICION
1260 FOR I=1 TO 9:PRINT TAB(TX);BR$;CHR$(156);BK$ ;
     CHR$(5);BR$:NEXT I
1270 X(PL)=X(PL)+2:Y(PL)=Y(PL)+ 1
1280 RETURN
1300 REM **** REPARTIR UN NAIPE ****
1310 CN=DK(DP,1):SU=(DP,2)
1320 DP=D+ DP+ 1:IF DP> 52 THEN DP=1:REM BARAJA
     CIRCULAR
1330 IF FL=1 THEN GOSUB 1200:RETURN:REM VISUALIZAR
     REVERSO NAIPE
1335 GOSUB 1000:RETURN:REM VISUALIZAR NAIPE
2000 :REM **** DATOS NUMEROS NAIPES ****
2010 DATA A,2,3,4,5,6,7,8,9,T,J,Q,K
2100 REM **** DATOS VISUALIZACION NAIPES ****
2110 DATA"00000000001000000000":REM A
2120 DATA"00001000000000010000":REM 2
2130 DATA"00001000001000010000":REM 3
2140 DATA"00010100000000101000":REM 4
2150 DATA"00010100001000101000":REM 5
2160 DATA"00010100101000101000":REM 6
2170 DATA"00010101010100101000":REM 7
2180 DATA"00010101010101010000":REM 8
2190 DATA"10100010101010000101":REM 9
2200 DATA"10101010100010101010":REM 10
2210 DATA"00000000001000000000":REM J
2220 DATA"00000000001000000000":REM Q
2230 DATA"00000000001000000000":REM K
```

Mezclando la baraja:

```
3000 REM **** BARAJAR EL MAZO ****
3005 R=RND(-TI):DP=1
3007 FOR I=1 TO 52:DK(I,1)=0:NEXT I
3010 FOR I=1 TO4:FOR J=1 TO 13
3020 EP=INT(RND(1)* 52)+ 1:REM SELECCIONAR PUNTO
     ENTRADA
3030 IF DK(EP,1)=0 THEN 3050
3040 EP=EP+ 1:IF EP> 52 THEN EP=1
3045 GOTO 3030
3050 DK(EP,1)=J:DK(EP,2)=I:NEXT J,I
3060 RETURN
```

La dirección exacta, por favor

Investigaremos los modos de direccionamiento del 68000. Esto nos ayudará a la hora de programar el chip con registros y tablas

En el capítulo de introducción de esta serie hablamos de la capacidad de direccionamiento del 68000 a propósito del juego de instrucciones. En particular advertimos que a pesar de tener un amplio abanico de modos de direccionamiento con los que es fácil referenciar bytes, palabras y palabras largas, debemos ser muy cautos en el empleo de tales modos con determinadas instrucciones.

Sea la siguiente *instrucción generalizada*:

```
OPCODE fuente, destino
```

Ésta es una manera semiformal de describir lo que realiza una clase de instrucciones con los operandos fuente y destino. La secuencia puede que exija tomar el operando fuente (dondequiera que se encuentre, y a través del cálculo que sea preciso para llegar hasta él), realizar la operación definida por el opcode sobre el operando y depositar el resultado en el operando destino (aquí también puede que sea preciso realizar algún cálculo para obtener la dirección de este operando). Por ejemplo, la instrucción MOVEA D3,A6 hace que el contenido de D3 (fuente) se lleve a A6 (destino). El opcode, MOVEA, indica que ha de moverse una dirección.

Se trata de un ejemplo muy sencillo de direccionamiento, donde no es preciso cálculo alguno para obtener la dirección de los operandos. En el otro extremo podríamos encontrarnos con que uno de los operandos esté direccionado mediante la suma del contenido de un registro de direcciones y un desplazamiento entero más el contenido de un registro índice (similar a la instrucción LD r, (IX+d) del Z80). Ya tendremos tiempo de comentar esto más adelante; de momento, baste con advertir que puede haber un buen puñado de operaciones aritméticas por realizar hasta dar con la dirección de los operandos.

Volviendo a nuestro modelo generalizado de instrucciones, también es posible que el opcode precise tan sólo un operando, como en este caso:

```
OPCODE fuente
```

Por ejemplo, la instrucción de bifurcación (como BRA BACKHERE) sólo necesita un operando (es decir, la dirección para bifurcar hacia BACKHERE). Finalmente, podemos también encontrarnos sólo con:

```
OPCODE
```

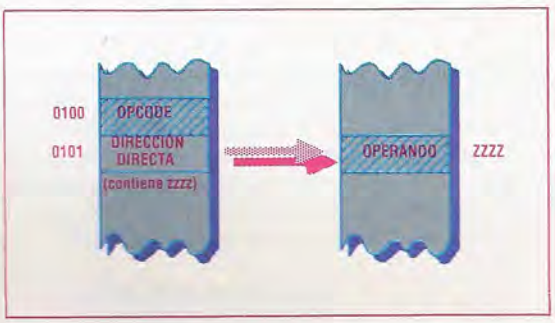
sin operando alguno. Un ejemplo típico de esta forma es la instrucción NOP, que sirve para indicar que no se realice ninguna operación. En efecto, se trata de una instrucción ficticia, y su utilidad es evidente en los parcheos manuales o en el cambio de programa en la memoria también manual.

Al emplear este modelo generalizado de la gama de instrucciones lo importante es tener en cuenta que siempre que haya operandos habrá algún tipo de cálculo de direcciones. Este cálculo es el que ha de ser especificado por el programador entre el conjunto de los cálculos o los modos de direccionamiento disponibles en el 68000.

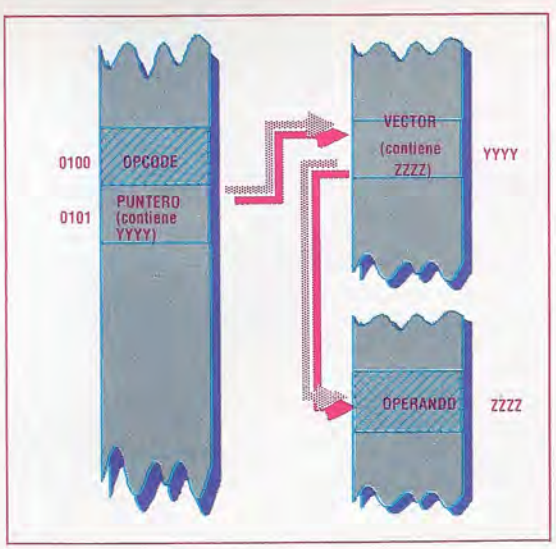
Muchos ordenadores disponen de al menos cinco *modos de direccionamiento* o maneras diferentes de direccionar los operandos. Nuestros diagramas ilustran la diferencia operativa de dos de estos modos:

- *Direccionamiento directo (o absoluto)*: En este modo la dirección de memoria del operando se almacena en la propia instrucción.
- *Direccionamiento indirecto (o puntero)*: En la instrucción se da una dirección de memoria que contiene a su vez la dirección del operando.

Como muestra el dibujo, en el direccionamiento directo el opcode opera sobre el operando que está en la dirección XXXX, mientras que en el direccionamiento indirecto, el operando será hallado en la po-



Direccionamiento directo
En este modo de direccionamiento el opcode (código de operación) de la instrucción es seguido inmediatamente por la dirección de una posición de memoria que contiene los datos del operando



Direccionamiento indirecto
El direccionamiento indirecto requiere que la operación especificada por el opcode acceda a su operando mediante una dirección "vector". La dirección del vector sigue inmediatamente al opcode, y el vector contiene la dirección del operando requerido. Este modo es muy útil cuando se calcula la dirección de un operando en tiempo de ejecución, dado que sólo se necesita calcular de nuevo el contenido del vector

sición ZZZZ, a la cual apunta un *puntero* que en este caso se encuentra en la posición YYYY de la memoria. Los otros tres modos principales de direccionamiento son:

- *Modo inmediato*: Cuando uno de los operandos de la instrucción es una constante. Por ejemplo, en la instrucción MOVEQ #25,D3 la constante 25 está direccionada en modo inmediato.
- *Modo de registros*: En este modo, el operando es uno de los registros disponibles y es especificado en el mismo código de la instrucción. Los operandos en la siguiente instrucción MOVE D2,D4 son los registros de datos D2 y D4.
- *Modo implícito*: Aquí los operandos están implícitos en la misma instrucción. Por ejemplo, en el caso de RTS (*ReTurn from Subroutine*: retorno de subrutina) son operandos implícitos el puntero de la pila y el contador del programa.

Observemos con mayor atención la manera en que el 68000 direcciona sus operandos. Además de los modos generales de direccionamiento que acabamos de ver, el 68000 tiene un *modo relativo de contador de programa* (también llamado *PC relativo*). Examinemos estos modos uno por uno:

- *Direccionamiento absoluto*: Empleando este modo podemos acceder a cualquier posición de la memoria. La dirección del operando aparece después de la instrucción. Por ejemplo:

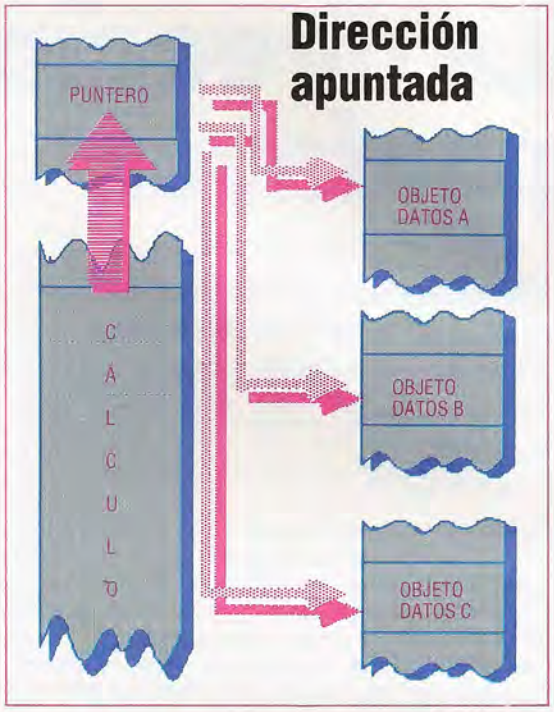
```
Direc.:   Código:  Etiq.:   Instrucción:
1000      D678             ADD     DATA,D3
1002      2000
.
.
.
2000      0001     DATA    DC.W    1
```

En este ejemplo podemos ver que el nombre simbólico DATA ha recibido la dirección $2000, la instrucción "suma (ADD) la fuente absoluta (DATA) al destino D3" ha sido codificada como D678, y que DATA de dirección absoluta se alberga en la posición $1002 (llamada *extensión de la palabra*). Otro ejemplo donde sólo encontramos un operando es:

```
Dir.:     Código:  Etiq.:   Instrucción:
1000      4278             CLR     COUNT
1002      3000
.
.
.
3000      0        COUNT   DS      1
```

Aquí el contenido de la posición $3000 (COUNT) queda limpio (puesto a cero) tras la ejecución de la instrucción CLR (*clear*: limpiar).

En el capítulo anterior dijimos que el PC era un registro de 32 bits (aunque sólo 24 de estos bits son significativos). Esto significa que la dirección absoluta que especifica el operando puede constar de más de una palabra como en los dos ejemplos anteriores. ¿Cómo hace el ensamblador para saber cuánto espacio ha de reservar para la dirección absoluta? Sin duda sería un dispendio injustificado el tener una extensión de palabra larga por cada referencia de dirección absoluta, por ello lo que ocurre es que se emplea la extensión adecuada siempre que se conoce la dirección del operando (en el caso de una referencia hacia atrás). En otras circunstancias habremos de especificar al ensamblador el empleo de las extensiones de palabra corta o palabra larga.



Volvamos al ejemplo de ADD para mostrar los efectos de una extensión de palabra larga:

```
Dir.:     Código:  Etiq.:   Instrucción:
1000      D679     ADD     HIDATA,D3
1002      0020
1004      0000
```

En este ejemplo la dirección absoluta de HIDATA es $200000. Nótese que el código para la parte ADD de la instrucción sigue siendo D6 pero que la parte de la dirección del operando ha cambiado de $78 hasta $79. En próximos capítulos veremos las formas para lograr esta extensión de palabra larga para el direccionamiento absoluto.

- *Direccionamiento por registro*: Es la manera más sencilla de direccionamiento en el 68000; en este caso el operando es uno de los registros del microprocesador. Por ejemplo, ADD D0,D3, donde la palabra D0 se añade al contenido de D3.

Hay algunas limitaciones en el empleo de este modo. Por ejemplo, no es posible tener un registro de dirección como destino de la instrucción ADD: así, no se acepta ADD D0,A4. Esto tiene arreglo si nos valemos de una instrucción diferente, ADDA D0,A4 donde ADDA es la instrucción de dirección de suma.

Si deseamos emplear palabras largas como objetos de datos en los ejemplos anteriores, deberemos incluir el atributo .L con la instrucción: ADD.L D0, D3 empleará las palabras de 32 bits enteras como objetos de datos.

- *Direccionamiento indirecto por registro*: Este modo es probablemente el más importante del 68000, dado que proporciona el *puntero* mencionado anteriormente y también los medios con los que se ejecutan las operaciones sobre la pila. Analicemos primero el empleo del puntero.

En la tarea de la programación necesitamos con frecuencia apuntar a un objeto de datos, ya sea un byte, una palabra larga o un objeto estructurado de datos como un registro o una tabla. Puede que, entonces, deseemos repetir el cálculo o la operación en otro miembro del mismo tipo de objeto de datos: es aquí donde el puntero resulta útil. El dibujo del puntero que adjuntamos muestra cómo puede ser empleado para dirigirse a diferentes elementos de un registro. Inicialmente el puntero dirige el cálculo que se ha de efectuar sobre el objeto de datos A; el puntero puede entonces ser restaurado para dirigir el cálculo que ha de efectuarse sobre cualquiera de los restantes objetos de datos.

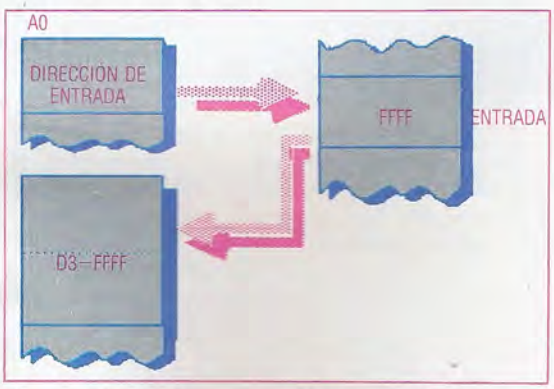
El elemento del 68000 a utilizar para lograr esto es un puntero de registro de direcciones, en vez del puntero almacenado junto a la instrucción, como en nuestro ejemplo general de modos de direccionamiento.

Esto puede comportar ligeros inconvenientes en algunas ocasiones, pero disponemos de ocho registros de direcciones.

Veamos ahora un ejemplo sencillo de modo de direccionamiento indirecto:

```
LEA      INPUT,A0
MOVE.W   (A0),D3
```

La instrucción LEA carga A0 con la dirección del objeto de datos de entrada llamado INPUT, el cual es seguidamente copiado en D3, como se muestra en el siguiente esquema:



Veamos ahora cómo se amplía este modo para operar sobre listas de datos. Ante todo, tenemos la *extensión de postincremento*. Se trata de que, una vez que se ha accedido al objeto de datos mediante el puntero, éste se incrementa para que apunte al contenido siguiente de la lista.

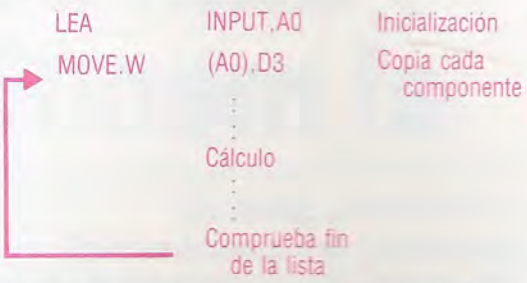
Examinemos el empleo de la extensión de postincremento en un ejemplo donde los objetos de datos son palabras:

```
MOVE.W   (A0)+,D3
```

Aquí el puntero en A0 es incrementado en dos unidades después que se ha accedido a la palabra apuntada por A0 y copiada en D3. Así, cuando apuntamos a la dirección $2000 originalmente, después de la operación MOVE.W veremos que A0 contiene $2002.

Es claro que si hemos empleado objetos de bytes entonces una operación MOVE.B sólo incrementará A0 en una unidad; y en cuatro con la operación MOVE.L.

La potencia de este modo de direccionamiento es obvia si se emplea en un bucle de programa para realizar algún cálculo en cada componente de una lista, por ejemplo. Así:



Cada vez que se realiza el cálculo se apunta automáticamente al componente siguiente de la lista tras cada ejecución de la instrucción MOVE.

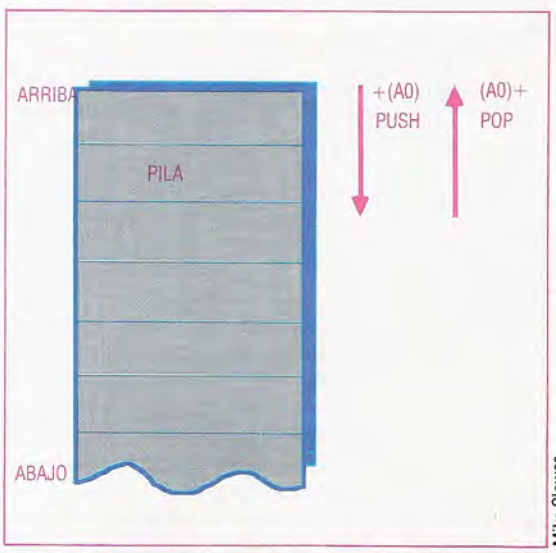
La otra extensión es denominada *predecremento*. En este caso el puntero de direcciones se decrementa antes de acceder al objeto de datos. P. ej.:

```
MOVE.W   -(A0),D3
```

En este caso A0 se decrementará en dos unidades antes de ser empleado para copiar en D3 el objeto de datos.

El direccionamiento indirecto predecremental es el modo complementario del direccionamiento indirecto incremental. El posdecremento progresa a través de una lista a medida que se incrementan las direcciones, mientras que el predecremento funciona al revés. Sin embargo, no podemos separar las operaciones de *pre* y *post* según nos convenga. Por ejemplo no son admisibles ni (A0)- ni +(A0). Debemos, en cambio, respetar el predecremento -(A0) y el posdecremento (A0)+ especificados.

Habrá notado que en cierto sentido los modos de direccionamiento *pre* y *post* son operaciones de pila. Nos servimos de estas operaciones para "poner en" o "sacar de" una pila, y una pila es, como se define convencionalmente, una serie de direcciones de arriba hacia abajo. De esta manera, la instrucción MOVE D0,-(A0) "pone" (*push*), y MOVE (A0)+,D0 es una operación que "saca" (*pop*), tal como se aprecia en el siguiente dibujo.



Mike Clowes

Dirección apilada
Las instrucciones PUSH y POP, tan conocidas de los programadores del Z80, pueden simularse mediante las potentes extensiones predecremento y posdecremento del 68000, que pueden usarse para ejecutar un PUSH y un POP respectivamente

Más adelante estudiaremos las pilas y cómo se emplean en el 68000. De momento, baste con observar que disponemos de un modo de direccionamiento muy cómodo para acceder a listas de datos.

¿Cuál es el menú?

La inmensa gama de posibilidades que ofrece el "WordStar" lo convierte en un programa extraordinariamente eficaz

En el curso de la última década se han escrito literalmente centenares de programas para tratamiento de textos, y se ha dicho que cada uno de ellos representaba una mejora respecto a los demás. Pero el líder del mercado continúa siendo uno de los procesadores de textos más antiguos. El *WordStar*, de MicroPro, se escribió originalmente para máquinas CP/M e hizo su aparición en 1978, poco después de la creación de la compañía. Desde entonces se lo ha reescrito para operar bajo PC-DOS/MS-DOS, y fue elegido para el IBM PC y, por tanto, para sus compatibles, que ahora dominan el mercado de gestión. La fuerza que ha hecho resistir al *WordStar* son sus casi inigualadas facilidades para tratamiento de textos. Ello es así a pesar del hecho de que el sistema no es especialmente amable con el usuario, si bien fue pionero en el uso de menús de "ayuda" en pantalla.

Como acabamos de mencionar, las raíces del *WordStar* se hallan en el sistema operativo CP/M y, como tal, conlleva muchas de las mejores y peores características de ese sistema. En primer lugar, cuando se carga el programa se lo añade a la lista de programas "transitorios" del CP/M, lo que le permite sacar partido de las facilidades de operación de disco del CP/M. Y, al igual que el CP/M, el *WordStar* hace uso de una amplia gama de caracteres de control.

Tras cargar y ejecutar el programa, los usuarios del *WordStar* se encuentran con el Menú de apertura, que visualiza todas las opciones disponibles, así como el contenido de la unidad de disco que esté conectada en ese momento. Si usted no ha cambiado el disco, éste consistirá en los programas que componen el *WordStar*. Es interesante destacar aquí que los nombres de archivo del *WordStar* utilizan el mismo formato que los nombres de archivo CP/M: un nombre de ocho caracteres o menos seguido por un punto y una extensión de tres letras opcional.

En el Menú de apertura hay una lista de 13 instrucciones, que están divididas en cinco secciones. Bajo la denominación Instrucciones preliminares, usted tiene la opción de cambiar la unidad conectada, activar y desactivar el Directorio de archivos y establecer el Nivel de ayuda. El Nivel de ayuda por defecto tras el arranque es el nivel 3, y visualiza todas las instrucciones disponibles en la parte superior de la pantalla. Descendiendo hasta el nivel 0, la pantalla estará limpia a excepción de la línea de estado.

Capacidades esenciales

A continuación relacionamos 11 facilidades disponibles en muchos programas de tratamiento de textos. En este capítulo y en los siguientes veremos si los paquetes examinados disponen de estas facilidades y cómo están implementadas.

Desplazamiento de palabras
La capacidad de un programa para desplazar una palabra de una línea a la siguiente si no hay suficiente espacio.

Movimiento de bloques
Permite que el usuario defina un "bloque" de texto que se puede manipular independientemente del resto del documento.

Ayuda en pantalla
Ofrece asistencia, en forma de mensajes, informando al usuario sobre cómo acceder a las facilidades disponibles.

Pantalla de 80 columnas
Un paquete de tratamiento de textos debe otorgar al usuario la mayor visión posible de un documento. Para esto se considera que el mínimo es una pantalla de 80 columnas.

Contador de palabras
Indica al usuario cuánto ha escrito hasta entonces.

Buscar/reemplazar
Busca letras, palabras o frases y las cambia.

En esta etapa usted tiene la opción de abrir o editar un archivo de documento o uno normal. El primero es texto que se puede editar con el *WordStar*, mientras que el segundo comprende programas que se pueden ejecutar mediante su ordenador. A continuación sigue una lista de Instrucciones de archivo que incluye opciones para imprimir, cambiar el nombre, copiar y borrar.

Las dos últimas secciones del Menú de apertura son Instrucciones del sistema y Opciones WordStar. La primera le proporciona una interface con CP/M, permitiéndole ejecutar uno de los programas en disco o bien salir al sistema operativo. La segunda le ofrece la opción de usar ya sea *MailMerge* o bien *SpellStar*. El primero es un programa para correspondencia automática, ideal para imprimir cartas personalizadas, direcciones para sobres, y otras actividades de correspondencia propias de empresas. *SpellStar* es un verificador de ortografía que compara cada palabra entrada en un documento con un diccionario retenido en la memoria. Se destacarán todas las palabras que no posean correspondencia en el diccionario, para que el usuario las confirme.

En este punto, el empleo de la opción D hace que se abra un archivo de documento (o uno que se esté creando, si se trata de un archivo nuevo), y la pantalla volverá a pasar al Menú principal. En la línea de arriba aparece indicada la unidad conectada y el nombre del archivo de documento a editar. A ello le sigue la posición actual del cursor y un aviso que informa si la facilidad Insertar está activada o no.

Debajo de esto, suponiendo que el Nivel de ayuda aún esté establecido en 3, hay otra lista de instruc-

WYSIWYG
Éstas son las siglas de "*what you see is what you get*" (lo que ves es lo que tienes) y alude al proceso que permite contemplar el texto en la VDU en la forma en que aparecerá en la página impresa.

Facilidad para correspondencia
Instalados ya sea integralmente o bien como programas asociados, estos programas pueden "elaborar a medida" cartas o documentos estándares insertando nombres y direcciones específicos en los puntos requeridos e imprimiendo las etiquetas de las direcciones.

Verificador de ortografía
Proporcionados en formato similar al de los programas de correspondencia, los verificadores de ortografía leen cada palabra de un documento y la comparan con un diccionario retenido en la memoria. Se le señalan al usuario los vocablos que no concuerden con los del diccionario.

Tipos de letra disponibles
Aunque en gran parte depende de la impresora, muchos procesadores de texto soportan distintos tipos, tales como negrita y cursiva.

Unión de archivos
Los documentos están necesariamente limitados por la cantidad de memoria disponible. Por tanto, cuando se imprimen documentos extensos es útil poder unir los archivos para las funciones de impresión o Buscar/reemplazar.

Una galaxia de estrellas

Puesto que el *WordStar* se ha hecho tan popular, no es sorprendente descubrir que su fabricante, MicroPro, haya lanzado varias versiones del programa CP/M original. En los últimos años, una de las versiones más populares del *WordStar* ha sido *WordStar Professional*, que ha sido traducido para operar bajo el sistema MS-DOS utilizado en la gama Apricot y en el IBM PC. Como incentivo adicional, MicroPro ha incluido *SpellStar* y *MailMerge*. Más recientemente, la empresa ha lanzado el *WordStar 2000*, también para el IBM PC. Este programa, aunque en la superficie es similar a su antecesor, se ha descrito como "el Rolls Royce de los procesadores de textos". Proporcionado en cinco discos, los programas y archivos que constituyen el *WordStar 2000* añaden ¡hasta dos Mbytes! Pero esta potencia es cara. Por último, a modo de indicio sobre el futuro del *WordStar* CP/M, MicroPro ha escrito una versión del programa denominada *Pocket WordStar* destinada a usuarios del Amstrad CPC 464 y 664. Dado que estos ordenadores mantienen una visualización en pantalla de alta resolución, de los 64 Kbytes disponibles no queda memoria suficiente para poder ejecutar el *WordStar* adecuadamente. En consecuencia, MicroPro ha desarrollado para estas máquinas una versión más pequeña, que será comercializada por Cumana

ciones disponibles, nuevamente divididas en secciones. La primera visualiza un resumen de los movimientos del cursor, ilustrando ampliamente los lazos que unen al *WordStar* con el CP/M. Las primeras máquinas CP/M carecían de teclas para control del cursor y, por lo tanto, los movimientos se llevaban a cabo pulsando la tecla Control con otra tecla simultáneamente. Estas teclas están concentradas en el lado izquierdo del teclado, siendo su núcleo las teclas S, E, D y X, que corresponden respectivamente a cursor izquierda, arriba, derecha y abajo.

Entre los diversos elementos, usted observará B (CTRL B), que está listado como Reformar. Esto volverá a ajustar un párrafo en pantalla después de que la edición y las sucesivas correcciones hayan convertido al original en un tremendo lío. (Dicho sea de paso, el formato que aparece en la pantalla corresponde exactamente a cómo aparecerá en la copia, siempre y cuando, por supuesto, su impresora soporte los márgenes que se hayan establecido en el *WordStar*.)

A la derecha de la pantalla de ayuda Menú principal hay una serie de otros títulos de menú que le proporcionan acceso a otras funciones adicionales. El Menú de ayuda, por ejemplo, proporciona descripciones detalladas de cómo trabaja cada instrucción del *WordStar*. El Menú rápido, por el contrario, contiene diversas instrucciones "rápidas", incluyendo las numerosas opciones Hallar y reemplazar.

El Menú en pantalla proporciona los medios para formatear la pantalla a cualquier tamaño y forma, permitiendo establecer tabulaciones, márgenes, espaciado de líneas, etc. Una función muy útil permite establecer la longitud de la página, que aparece en el texto como una línea de puntos. Con ello puede verse dónde se producirán los cambios de página durante la impresión, y por lo tanto, puede distribuir consiguientemente el documento.

En el Menú en pantalla también está disponible el activador Wordwrap. Éste, una vez establecido, toma aquella palabra que no cabe al final de una línea y la coloca al comienzo de la siguiente, eliminando de este modo la molestia de tener que preocuparse por separar las palabras en sílabas mediante guiones.

El tercero de los menús adicionales contiene algunas de las instrucciones más útiles y potentes de que dispone el *WordStar*. Aunque el Menú de bloques puede llevar a cabo numerosas operaciones, su función primordial es tratar bloques de texto dentro de un documento. La gama de operaciones está listada bajo la sección Operaciones de bloques.

Mediante instrucciones CTRL al comienzo y al final del texto requerido se pueden crear bloques, que se visualizarán en "video invertido" (el texto del bloque aparecerá con un brillo considerablemente menor que el resto del texto, para su fácil reconocimiento). Una vez creado el bloque, se puede trasladar o copiar el texto en otra zona del documento, suprimirlo o incluso guardarlo en un archivo separado en disco. Pero usted no se ve limitado a desplazar párrafos solamente. Con el *WordStar* también se pueden desplazar columnas, lo que resulta muy cómodo cuando se está trabajando con tablas o cartas con dos o más columnas de texto. En este caso, todo es cuestión de demarcar columnas de bloques en lugar de líneas.

En el Menú de bloques se incluyen tres instruccio-

Desplazamiento de palabras

★★★★

Si bien el *WordStar* desplaza las palabras cuando se está escribiendo el texto, cuando se cambian los márgenes no vuelve a formatear el texto de forma automática.

Movimiento de bloques

★★★★★

El *WordStar* permite la definición de bloques de cualquier tamaño y la totalidad de la gama de manipulación de bloques.

Ayuda en pantalla

★★★★★

Los menús de "Ayuda en pantalla" son característicos del *WordStar* y contribuyeron a acrecentar su popularidad.

Pantalla de 80 columnas

★★★★★

El programa no sólo soporta una pantalla de 80 columnas, sino que también permite márgenes de hasta 255 caracteres.

Contador de palabras

El *WordStar* no posee facilidad para contar las palabras.

Buscar/reemplazar

★★★

El *WordStar* soporta varias versiones de esta facilidad, permitiendo la búsqueda de series de hasta 30 caracteres.

WYSIWYG

★★★★★

Más que ningún otro paquete para tratamiento de textos, el *WordStar* es famoso por su capacidad para formatear el texto en pantalla.

Facilidad correspondencia

★★★★

El *WordStar* fue uno de los primeros que incluyó un programa para correspondencia, y el *MailMerge* sigue siendo el estándar que sirve de referencia a los demás.

Verificador de ortografía

★★★★

Las versiones avanzadas del *SpellStar* poseen un diccionario de alrededor de 20 000 palabras para la verificación ortográfica.

Tipos de letra disponibles

★★★

Los tipos de letra adicionales incluyen negrita y cursiva.

Union de archivos

★★

Se pueden imprimir de modo continuo archivos separados utilizando el *MailMerge*, pero no se soportan facilidades tales como "Buscar/reemplazar" a través de varios archivos al mismo tiempo.

À la carte

El *WordStar* se caracteriza por los menús de ayuda, que se visualizan en la parte superior de la pantalla. Aunque los usuarios ya experimentados tienden a prescindir de ellos, son sumamente útiles para los principiantes, porque proporcionan una fácil referencia a las instrucciones disponibles.

Cada una de las tres pantallas que vemos aquí proporciona una gama de instrucciones diferente. La pantalla de apertura, que es la que se presenta al usuario cuando se carga el *WordStar*, contiene mayormente las instrucciones DOS que le permiten acceder a los archivos. El menú principal contiene instrucciones que con toda probabilidad se usarán cuando se digite el texto, mientras que el menú de bloques contiene varias instrucciones de edición

Le dice al usuario cuál es la unidad conectada y cuál el archivo que se está editando

Éste es el menú principal bajo el nivel de ayuda tres, dando explicaciones sobre las instrucciones disponibles

Números de página, línea y columna indican la posición del cursor

Este aviso recuerda que está activada la mod. INSERTAR

Las versiones IBM del *WordStar* permiten entrar numerosas instrucciones a través de las teclas de función. Esta línea muestra la configuración actual de las teclas de función

Esta línea muestra la cantidad de columnas establecidas, si bien la misma se puede modificar. El signo ! indica los ajustes TAB por defecto, que también se pueden alterar

nes para guardar, ofreciendo las opciones de retornar al texto, ir al menú principal o salir del *WordStar*. Usted debe tener presente que cuando se utilizan instrucciones incluidas en los menús separados no es necesario volver a llamar el menú a la pantalla. Para guardar y salir del sistema, por ejemplo, sólo debe digitar CTRL KX, que ejecutará la instrucción automáticamente.

Vamos a concluir esta breve reseña del *WordStar* con una análisis de las *instrucciones de punto*. Éstas están incorporadas en el texto en líneas separadas, empezando cada una de ellas, como parece natural, con un punto, y están relacionadas con el formateo de la impresión y la adición de características a la salida impresa, como anchuras de los márgenes superior e inferior y números de página. La instrucción .pl (seguida por un número), por ejemplo, establece el número de líneas a imprimir en una página. Debido a que el *WordStar* no actúa sobre estas instrucciones hasta que el software esté procesando el documento para su impresión, en realidad actúan como caracteres de control para la impresora.

La potencia del *WordStar* ha conseguido mantener este programa a la cabeza de su especialidad durante más de una década. Dado que el CP/M se "extiende hacia abajo" del mercado, hacia máquinas personales tales como la gama Amstrad, parece probable que el *WordStar* se convierta también en el líder del mercado de ordenadores personales.

Un paquete popular

El *WordStar* constituye un interesante ejemplo de cómo la reconocida popularidad de un programa genera su propio impulso para hacerse aún más popular. Una vez que el *WordStar* se estableció como líder del software para tratamiento de textos para micros CP/M, muchos fabricantes comenzaron a empaquetarlo en sus máquinas, lo que condujo a una base de usuarios aún mayor para el paquete. Un temprano ejemplo de este tipo de comercialización fue el Osborne 1, máquina que no sólo tenía empaquetado el *WordStar*, sino también la hoja electrónica *SuperCalc*. El hecho de que ambos programas operaran bajo CP/M significaba que podían compartir archivos comunes. Esta idea demostró ser tan popular que otros programadores comenzaron a incorporar el concepto de pasar y compartir archivos, lo que finalmente condujo al desarrollo de paquetes de software integrado, como el *Lotus 1-2-3*. En la actualidad, casi todos los micros vienen con software empaquetado, en especial, procesadores de textos. Los mismos van desde el sofisticado *Perfect Software* de Thorn EMI (para el Advance y el Wren) hasta el económico Tasword II, que se empaquetó en el *six-pack* que se entregó junto con las primeras versiones del Spectrum+

Modelo personal

"Expert-Ease" es un programa que permite al usuario novel crear un sistema experto a su medida

Por definición los sistemas expertos tienden a aplicarse en campos determinados, tales como diagnóstico médico o reconocimiento geológico. Pero el generador de sistemas expertos ofrece al usuario la posibilidad de aplicar esta tecnología *soft* a problemas más inmediatos que, de lo contrario, no garantizarían el costo de preparar un sistema a medida. Un buen ejemplo de este enfoque es el que proporciona *Expert-Ease*, de Thorn-EMI Software, que permite crear un sistema experto propio (denominado *modelo*) y ejecutarlo en un IBM PC o un ACT Sirius, con un mínimo de 128 K de RAM.

Tal como mostramos en la página 1781, la parte de un sistema experto que realiza todo el trabajo es el motor de inferencias: el módulo de proceso lógico. Es la programación de esta sección lo que normalmente pone el enfoque de un sistema experto fuera del alcance de la mayoría de los usuarios, a menos que el gasto de programar la aplicación pretendida se considere absolutamente imperativo. Pero en realidad *Expert-Ease* crea un motor de inferencias para el usuario, de modo que todo el trabajo pesado lo realiza el ordenador en lugar de la persona que está creando el modelo.

Existen cuatro enfoques principales para diseñar el motor de inferencias de un sistema experto: el formalismo basado en reglas, empleando una estructura IF...THEN; las redes semánticas, basadas en conjuntos y subconjuntos ("batería descargada", por ejemplo, es un miembro del conjunto "fallo eléctrico"); ventanas o marcos, que se preparan como una base de datos tradicional con la excepción de que los valores especificados de los campos especificados hacen que el sistema saque conclusiones sobre el registro; y cláusulas cuerno, la lógica de predicado en la que se basa el PROLOG. *Expert-Ease* utiliza un quinto método automatizado: el *sistema analógico de aprendizaje de conceptos* (*Analogue Concept Learning System*: ACLS).

El ACLS se desarrolló como módulo independiente en el laboratorio de inteligencia artificial de la Universidad de Edimburgo por el personal y los consultores de Intelligent Terminals Ltd, una empresa propiedad del profesor Donald Mitchie, a quien se considera una de las primeras autoridades del mundo en materia de inteligencia artificial. El estudio del funcionamiento del ACLS está fuera del alcance de este capítulo, pero para quienes estén interesados en el mismo, se desarrolló a partir del *Iterative Dichotomiser 3* (dicotomizador iterativo: ID3), un programa de dominio público escrito en PASCAL.

Chris Corr

Un problema sencillo

Los modelos de *Expert-Ease* en realidad se crean hacia atrás, en el sentido de que usted primero debe decidir cuáles son todas las posibles conclusiones y luego decidir cuáles son las preguntas que es necesario que responda el usuario, y por último decidir qué respuestas conducirán a qué decisiones. A modo de ejemplo de la estructura del *Expert-Ease*, veamos una aplicación muy simple: resolver el problema de si debemos ir o no al partido de fútbol de esta tarde. A pesar de sus evidentes limitaciones, al abordar este problema demostramos el potencial del sistema para aplicaciones en áreas más exigentes de diagnóstico y toma de decisiones.

Las dos conclusiones posibles para este problema son, obviamente, "ir" y "no ir", de modo que lo siguiente que hemos de hacer es decidir qué preguntas necesitamos formular o, por decirlo de otro modo, qué factores son importantes para tomar la decisión. Nuevamente, para simplificar, vamos a suponer que sólo estamos interesados en el buen tiempo y en tener suficiente tiempo. Comencemos por elaborar una tabla de decisión:

Ejemplo/ Pregunta	¿Buen tiempo?	¿Tiempo suficiente?	Decisión
1.	S	S	IR
2.	S	N	NO
3.	N	*	NO
4.	N	*	NO

Guía de experto
Expert-Ease, un generador de sistemas expertos producido por Thorn-EMI, permite elaborar sistemas expertos a la medida de forma rápida y económica para distintas aplicaciones. Particularmente útil en diagnóstico y análisis de problemas, la potencia del sistema experto (aunque previamente restringido a áreas con un elevado perfil financiero) ya se puede aplicar a situaciones más domésticas

Es claro que si las respuestas son negativas, la decisión es "no". Podemos alterar la tabla así:

Ejemplo/ Pregunta	¿Buen tiempo?	¿Tiempo suficiente?	Decisión
1.	S	S	IR
2.	S	N	NO
3.	N	S	NO
4.	N	N	NO

donde el asterisco significa "no preocuparse". De manera que si la respuesta a la primera pregunta es negativa, es innecesario formular la segunda.

Así es, en realidad, cómo requiere el *Expert-Ease* que entremos la información. En primer lugar, crea una tabla marco que contiene las preguntas y los posibles resultados (conocida como *pantalla de atributos*) y después rellena esta pantalla con respuestas ejemplo (conocida como *pantalla de ejemplos*). La única diferencia es que el *Expert-Ease* no nos limita a respuestas de sí o no: podemos tener tantas respuestas como queramos. Las preguntas se diseñan en un formato de múltiple opción, y rellenamos los números en la pantalla de ejemplos. Una vez creada la pantalla de ejemplos, el *Expert-Ease* utiliza el ACLS para inducir la lógica de la tabla de decisión para crear una "regla". Ésta se aplica cuando se ejecuta el módulo.

El ACLS (que en el paquete se denomina *rutina de inducción*) comprueba nuestra tabla en busca de conflictos y nos los señala (tales como: formadas 11 reglas nudo, ejemplos 1 5 6 contradicen la regla).

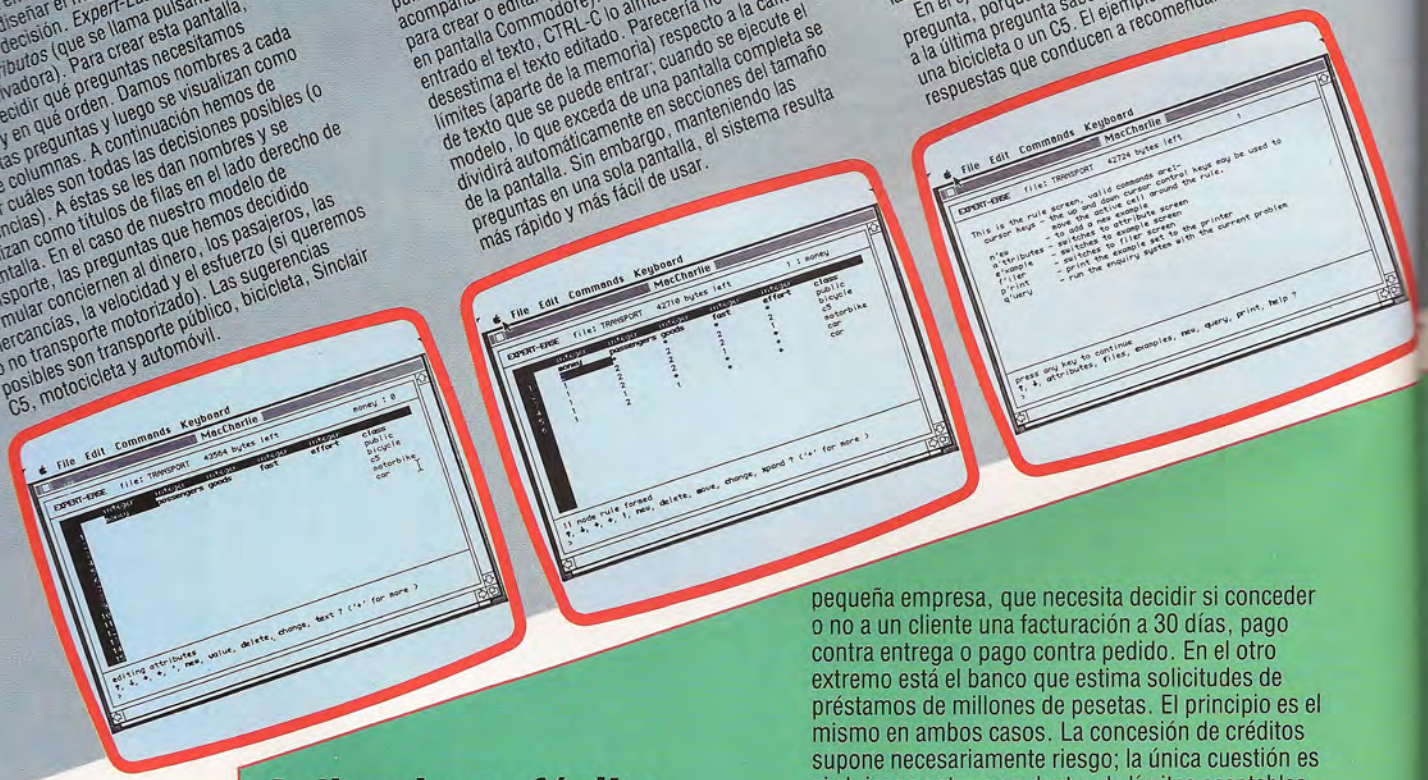
Expert-Ease es indudablemente un programa sumamente potente y sofisticado. Para la mayoría de las aplicaciones comunes, los modelos que crea responden a cualquier sistema experto especializado. Además, se pueden encadenar modelos entre sí, de

ación de un modelo

mera etapa en la creación de un modelo rt-Ease es diseñar el marco en el cual se basará squema de decisión. *Expert-Ease* lo denomina antalla de *atributos* (que se llama pulsando A de la antalla archivadora). Para crear esta pantalla, emos de decidir qué preguntas necesitamos esponder y en qué orden. Damos nombres a cada una de estas preguntas y luego se visualizan como títulos de columnas. A continuación hemos de elaborar cuáles son todas las decisiones posibles (o sugerencias). A éstas se les dan nombres y se visualizan como títulos de filas en el lado derecho de la pantalla. En el caso de nuestro modelo de transporte, las preguntas que hemos decidido formular conciernen al dinero, los pasajeros, las mercancías, la velocidad y el esfuerzo (si queremos o no transporte motorizado). Las sugerencias posibles son transporte público, bicicleta, Sinclair C5, motocicleta y automóvil.

Una vez que hemos creado estos títulos, podemos entrar el texto para las preguntas y sugerencias pulsando T (para visualizar el texto de acompañamiento), seguida de ' (una sola comilla) para crear o editar (el editor está al nivel del editor en pantalla Commodore). Una vez que hemos entrado el texto, CTRL-C lo almacena y Escape desestima el texto editado. Parecería no haber límites (aparte de la memoria) respecto a la cantidad de texto que se puede entrar; cuando se ejecute el modelo, lo que exceda de una pantalla completa se dividirá automáticamente en secciones del tamaño de la pantalla. Sin embargo, manteniendo las preguntas en una sola pantalla, el sistema resulta más rápido y más fácil de usar.

La pantalla de ejemplos es el verdadero esquema de decisión. Se lo llama así porque el usuario muestra al *Expert-Ease*, mediante ejemplos, cómo se han de tomar las decisiones. Por consiguiente, el ejemplo 1 es un usuario que responde 2 (no) a la pregunta del dinero, indicando que carece de capital para gastar. Puesto que las respuestas a las otras preguntas son irrelevantes, colocamos un asterisco (el símbolo de "no preocuparse") bajo cada una de las otras columnas. El *Expert-Ease* se saltará entonces todas las preguntas que lleven debajo un asterisco. En el ejemplo 2 necesitamos formular cada pregunta, porque sólo cuando tenemos la respuesta a la última pregunta sabemos si el usuario requiere una bicicleta o un C5. El ejemplo 3 muestra las respuestas que conducen a recomendar un C5.



Aplicaciones fáciles

El *Expert-Ease* es ideal para crear modelos simples, en especial aquellos que quizá se hayan de modificar para adecuarse al futuro desarrollo.

Aplicaciones de crédito

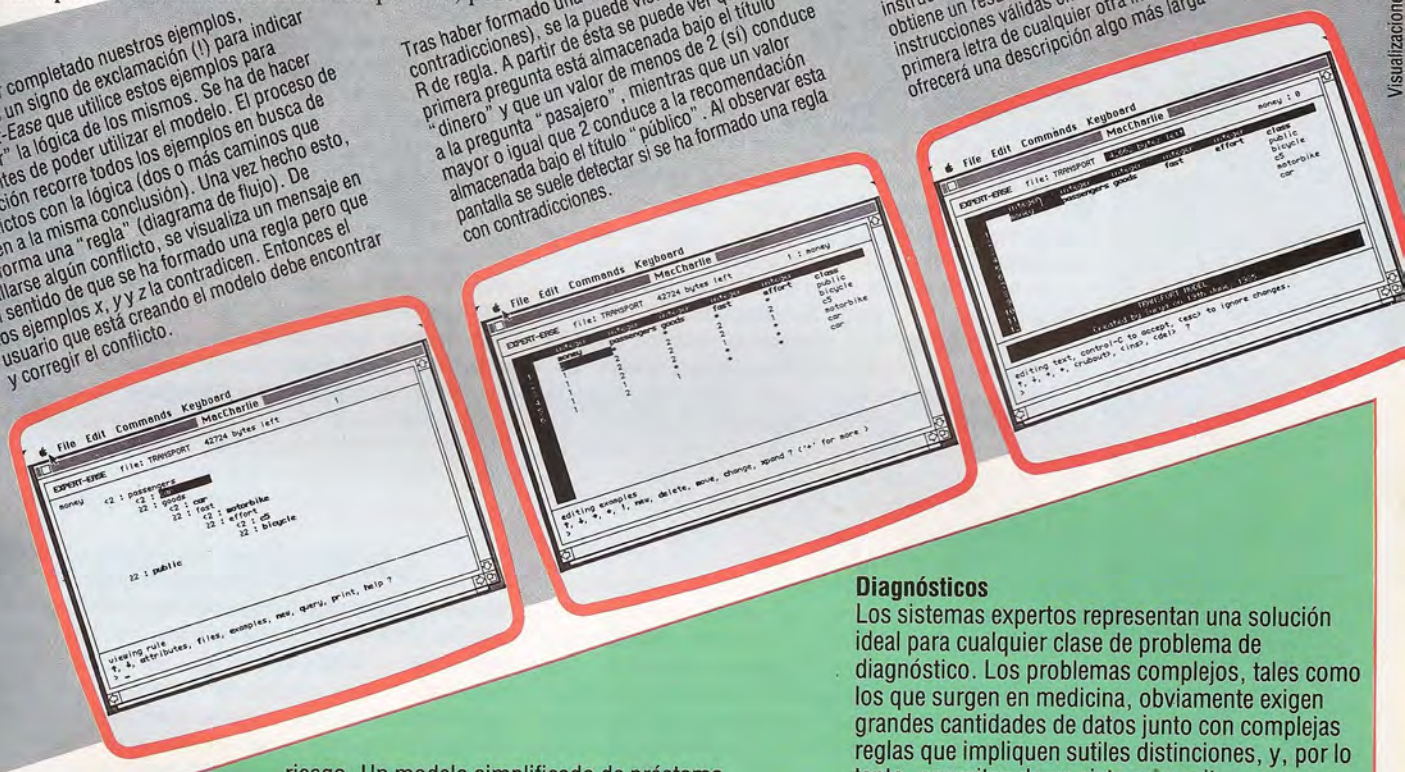
Los sistemas expertos pueden ser de enorme utilidad en el área de proceso de solicitudes de crédito. En uno de los extremos del espectro está la pequeña empresa, que necesita decidir si conceder o no a un cliente una facturación a 30 días, pago contra entrega o pago contra pedido. En el otro extremo está el banco que estima solicitudes de préstamos de millones de pesetas. El principio es el mismo en ambos casos. La concesión de créditos supone necesariamente riesgo; la única cuestión es si el riesgo entra o no dentro de límites aceptables. La tarea del controlador de créditos o gestor de préstamos es inicialmente evaluar el riesgo y, en segundo lugar, decidir si éste entra dentro de los límites aceptables para la empresa o el banco. Crear un modelo *Expert-Ease* para llevar a cabo la misma función es simplemente cuestión de categorizar los criterios utilizados para evaluar la "conveniencia" del crédito, idear preguntas para cada uno y utilizar las respuestas para evaluar el

modo que es factible descomponer problemas complejos en piezas manejables. La principal diferencia entre un modelo *Expert-Ease* y un sistema experto escrito a medida es que el primero no explica su propio razonamiento. Usted, por supuesto, puede crear sus propias explicaciones en el texto de la conclusión, pero al usuario no se le presenta el verdadero camino de decisión del modelo. Esto no constituye desventaja en sistemas diseñados para que los utilicen no expertos, puesto que el razonamiento podría ser excesivamente complicado, pero sí es significativo en modelos diseñados para ayudar a los expertos, quienes quizá deseen cuestionar una parte o la totalidad del razonamiento.

Tras haber completado nuestros ejemplos, entramos un signo de exclamación (!) para indicar al *Expert-Ease* que utilice estos ejemplos para "inducir" la lógica de los mismos. Se ha de hacer esto antes de poder utilizar el modelo. El proceso de inducción recorre todos los ejemplos en busca de conflictos con la lógica (dos o más caminos que lleven a la misma conclusión). Una vez hecho esto, se forma una "regla" (diagrama de flujo). De hallarse algún conflicto, se visualiza un mensaje en el sentido de que se ha formado una regla pero que los ejemplos *x*, *y* y *z* la contradicen. Entonces el usuario que está creando el modelo debe encontrar y corregir el conflicto.

Tras haber formado una regla (con o sin contradicciones), se la puede visualizar pulsando la R de regla. A partir de ésta se puede ver que la primera pregunta está almacenada bajo el título "dinero" y que un valor de menos de 2 (sí) conduce a la pregunta "pasajero", mientras que un valor mayor o igual que 2 conduce a la recomendación almacenada bajo el título "público". Al observar esta pantalla se suele detectar si se ha formado una regla con contradicciones.

Usted puede disponer de breves descripciones de las instrucciones del *Expert-Ease* en cualquier momento dado pulsando H o ?. La excepción a esto se produce cuando usted se halla en mitad de la tarea de rellenar un ejemplo en la pantalla de muestra; en este caso primero es necesario completar el ejemplo. Se le preguntará con qué instrucción le agradaría una ayuda. Pulsando H obtiene un resumen de una línea de todas las instrucciones válidas en la pantalla. Pulsando la primera letra de cualquier otra instrucción se le ofrecerá una descripción algo más larga



riesgo. Un modelo simplificado de préstamo bancario podría ser más o menos así:

Seguridad	Deuda irrecup.	Girado en descub.	Ingresos regulares	Resultados
S	*	*	*	OK
N	S	*	*	NO
N	N	S	S	SOMETER
N	N	S	N	NO
N	N	N	N	SOMETER
N	N	N	S	OK

Una ejecución típica podría ser como ésta:

¿Ofrece seguridad el cliente? No
¿Tiene el cliente algún antecedente de deuda irrecuperable? No
¿El cliente ha girado alguna vez en descubierto sin autorización? Sí
¿Tiene éste ingresos regulares comprobables? Sí
Por favor someter esta solicitud a un gestor de préstamos.

De este modo, las solicitudes directas las puede manejar un empleado subalterno utilizando el sistema experto, mientras que las solicitudes que requieran una consideración más detallada se someterán a un gestor de préstamos o gerente. Si el banco modificara sus criterios generales para la concesión de préstamos, el modelo se podría modificar rápidamente.

Diagnósticos

Los sistemas expertos representan una solución ideal para cualquier clase de problema de diagnóstico. Los problemas complejos, tales como los que surgen en medicina, obviamente exigen grandes cantidades de datos junto con complejas reglas que impliquen sutiles distinciones, y, por lo tanto, necesitan de un sistema escrito especialmente. Sin embargo, *Expert-Ease* puede tratar fácilmente tareas menos complicadas.

Por ejemplo, una empresa de helicópteros que lleva y trae a su trabajo a los empleados de una instalación petrolífera utiliza el *Expert-Ease* como ayuda en el servicio de mantenimiento rutinario de su flota de aparatos. Normalmente, cuando se trae un helicóptero para un servicio de rutina menor, se requiere un ingeniero superior para inspeccionarlo y determinar el tipo de servicio requerido. Una vez hecho esto, los mecánicos pueden llevar a cabo el trabajo. Sin embargo, utilizando el *Expert-Ease* un mecánico puede determinar el trabajo que es necesario llevar a cabo sin consultar al ingeniero superior. Éste se limita a realizar su inspección normal una vez que se ha efectuado el trabajo.

El mismo principio se puede ampliar fácilmente al mantenimiento y reparación de todo tipo de equipos mecánicos, eléctricos y electrónicos, incluyendo, por supuesto, ordenadores. De hecho, el *Expert-Ease* se puede utilizar para detectar fallos de software o hardware. Por ejemplo, cuando un cliente solicita servicio técnico, los ingenieros han de pasar un valioso tiempo detectando el fallo de la máquina. No obstante, con el *Expert-Ease* esta tarea se podría dejar en manos de personal menos cualificado, planteando a los ingenieros sólo los problemas graves

Visualizaciones de Macintosh con MacCharlie

Lenguaje puente

Iniciamos una serie dedicada al C, un lenguaje que llena el vacío existente entre los lenguajes de alto y bajo nivel

Recientemente, gran parte del desarrollo de lenguajes de alto nivel se ha dirigido hacia la creación de lenguajes que reflejen mejor los procesos del pensamiento humano y alejen del hardware al programador en la mayor medida posible. Queda, sin embargo, muchísima programación en la cual el informático está implicado de forma directa en manipular el hardware y en obtener la máxima eficacia de máquinas pequeñas y relativamente lentas.

Tradicionalmente esto se ha hecho en lenguaje ensamblador o, esporádicamente, utilizando un lenguaje como el FORTRAN en un hardware que haya sido diseñado para un rendimiento óptimo con el lenguaje. No obstante, el problema de utilizar lenguaje ensamblador es que si bien proporciona una máxima eficacia (en manos de un buen programador), es específico de una máquina. Por consiguiente, durante mucho tiempo ha existido la necesidad de un lenguaje sencillo a un nivel relativamente bajo que fuera fácilmente implementable en una gran variedad de máquinas, combinando las estructuras y la conveniencia de un lenguaje de alto nivel con un control directo y eficaz sobre el hardware.

El primer lenguaje del cual se pudo afirmar que reunía estos requisitos fue el BCPL, desarrollado por Martin Richards en la Universidad de Cambridge. Ken Thompson y otros colaboradores de Bell Labs aplicaron muchas de las ideas del BCPL para un lenguaje denominado B. En aquella época estaban desarrollando el sistema operativo Unix y para ello necesitaron un lenguaje de alto nivel. Kernighan y Ritchie convirtieron al B en el C y en 1978 publicaron el libro que lo define, *The C Programming Language*, que sirve como estándar para las implementaciones de C (hasta que se llegue a un acuerdo para un nuevo estándar internacional).

También se estaba desarrollando el Unix y desde entonces el lenguaje y el sistema operativo han estado íntimamente relacionados. Las 13 000 líneas de código que componen el *kernel* o núcleo del Unix contienen apenas 800 líneas de ensamblador, siendo de C el resto de las líneas. Otro punto a favor del C es que se trata de un lenguaje relativamente pequeño, lo que no sólo lo vuelve fácil de aprender y utilizar, sino que también conlleva que los compiladores sean pequeños y fáciles de escribir, de modo que el lenguaje se puede implementar en una amplia variedad de máquinas. Además, dado que el lenguaje ya es de bajo nivel, el código producido es compacto y rápido. El resultado de esto es que no sólo el Unix y las aplicaciones Unix se escriben en C, sino también el grueso de los sistemas operativos MS-DOS y CP/M.

Existen compiladores de C para casi todos los micros, así como para ordenadores centrales, y su uso ha permitido que las casas de software produzcan programas que se puedan trasladar con facilidad entre máquinas tan disímiles como el Apple Macintosh y el IBM PC. A lo largo de esta serie nos ceñiremos a la versión estricta del lenguaje de Kernighan y Ritchie, y allí donde sea conveniente, como cuando consideremos bibliotecas de funciones, daremos por sentado el empleo del Unix, si bien la mayoría de las otras versiones de C seguirán esta versión lo más estrechamente posible.

Un programa en C se construye mediante funciones de definición, cada una de las cuales toma varios argumentos, valores o punteros de valores, que la función utiliza para producir un valor o puntero que se devuelve como resultado. Cada función se invoca simplemente escribiendo su nombre. El valor devuelto se puede utilizar, aunque en algunos casos se descarta.

Todos los programas en C empiezan ejecutando una función llamada main, de modo que el nivel más exterior de un programa debe ser una definición de esta función. He aquí un programa muy simple pero muy completo para efectuar la primera prueba de cualquier nuevo lenguaje o sistema de ordenador:

```
main ()
{
  printf("hola mundo\n");
}
```

Incluso este sencillo programa nos pone de relieve algunas importantes características del lenguaje. La definición de una función consiste en el nombre de la función seguido por una lista de sus argumentos encerrados entre paréntesis. En este caso, sucede que main no tiene argumentos pero, aun así, se han de colocar los paréntesis.

El cuerpo de la definición de función consiste en una secuencia de sentencias (sólo una, en este caso) encerrada entre corchetes ({}) y cada sentencia termina con un punto y coma. Los corchetes actúan de modo similar al Begin...End del PASCAL, en tanto y en cuanto se los puede utilizar para encerrar cualquier secuencia de sentencias y declaraciones para conformar una sentencia compuesta, que a su vez se puede utilizar en cualquier posición en la que se pueda usar una sentencia simple. El punto y coma se utiliza de manera diferente al PASCAL, pero la regla es mucho más simple: cada sentencia completa debe ir seguida de un punto y coma. La única sentencia de este programa es una llamada a una función de biblioteca, printf.

La entrada/salida no está definida estrictamente en C, porque varía mucho de una máquina a otra. En cambio, queda en manos de cada una de las implementaciones el proporcionar las E/S que se requieran en una biblioteca, si bien la mayoría son fieles al estándar. El valor devuelto por la función printf no sirve, de modo que se lo ignora. Puede haber un número cualquiera de argumentos para printf, ya sea numéricos, variables de tipo carácter o series (como en este caso). Asimismo, printf no genera automáticamente un retorno, pero el C proporciona una cantidad de símbolos especiales para caracteres de control, todos los cuales comienzan con una barra invertida (\). El símbolo de retorno (o línea) es \n (véase la tabla *Secuencias Escape*).

Marcus Wilson-Smith

C de calidad
El Hisoft-c se ejecuta en los ordenadores Amstrad y Spectrum. La implementación es moderadamente estándar, con la grave excepción de que no se incluyen números de punto flotante. No obstante, Hisoft tiene entre sus planes producir en un futuro cercano una versión con punto flotante. Otro punto objetable lo constituye el manual, que, si bien es exhaustivo, está muy mal dispuesto y no es adecuado para principiantes. Siempre y cuando el usuario tenga ya alguna experiencia previa o una documentación adicional, éste es un paquete excelente

El siguiente programa daría exactamente el mismo resultado que el primero:

```
main ()
{
   printf('hola ');
   printf('mundo');
   printf('\n');
}
```

El c es un lenguaje con variables tipificadas, es decir, cada variable se debe declarar como de un tipo determinado. Siempre están disponibles los siguientes tipos:

char	—caracteres, ocupando un único byte.
short	—enteros cortos.
int	—enteros normales.
long	—enteros largos.
float	—números (reales) con punto flotante.
double	—números con punto flotante de doble precisión.

Las declaraciones de variables se efectúan dando el nombre del tipo seguido por una lista de nombres de las variables que usted quiere que sean de ese tipo. Se deben declarar todas las variables, si bien, como ya veremos, las declaraciones pueden producirse casi en cualquier punto del programa. Además, los nombres de las variables pueden ser de cualquier longitud, pero por lo general sólo son significativos los ocho primeros caracteres.

Asignación y aritmética

Como es habitual, no se admiten espacios en nombres de variables, pero se puede utilizar el carácter de subrayado. Los nombres no pueden empezar con un dígito ni tampoco con un subrayado, pero esto sólo obedece a un posible conflicto con funciones de biblioteca. El tipo de letra (mayúscula o minúscula) es significativo, de modo que, por ejemplo, A es una variable distinta de a. No está especificada la cantidad total de bytes para cada tipo numérico, de modo que pueden variar para cada implementación. Típicamente, sin embargo, un entero short será de ocho bits, int será de 16 bits y long será de 32 bits.

La asignación y la aritmética siguen convenciones bastante normales, utilizándose el signo de igualdad (=) como operador de asignación. Los operadores aritméticos usuales +, −, * y / se utilizan junto con %, que da el módulo. De modo que:

```
int x,y,z

z=x%y
```

Las expresiones y las asignaciones pueden incluir cualquier variedad de tipos numéricos, y char, que a estos fines se considera como un entero de un byte que contiene el código ASCII del carácter. El c lleva a cabo de forma automática todas las conversiones de tipos necesarias convirtiendo los tipos *inferiores* a *superiores*. Es interesante destacar aquí que todas las operaciones de punto flotante normalmente se efectúan en doble precisión, aun cuando todos los operandos sean meramente float. También es posible forzar un cambio de tipo mediante un "matiz" en el cual el nombre de variable va precedido por el nuevo tipo entre paréntesis. Por ejemplo:

```
int n;
float f;
. . . .
f=sqrt((double)n);
```

toma una copia de doble precisión del valor de n, dado que la función raíz cuadrada espera que su argumento sea de tipo double, pero el verdadero valor de n queda inalterado.

El c proporciona dos operadores adicionales muy útiles, que son los operadores de incremento y decremento ++ y −−. De modo que, por ejemplo, ++a significa incrementar y −−a significa decrementar el valor de a *antes* de que se ejecute la sentencia actual; a++ significa incrementar y a−− significa decrementar el valor de a *después* de que se ejecute la sentencia actual. Por consiguiente:

```
int a,b
. . .
b=1;
a=b++;/* deja 1 en a y 2 en b*/
```

Mientras que:

```
b=1
a=++b;/* deja 2 tanto en a como en b*/
```

Otro detalle es que la asignación, al igual que todas las otras operaciones del c, se trata como una función y devuelve un valor, a saber, el valor que se está asignando. Éste es un valor perfectamente legítimo para utilizar en cualquier otra expresión; de modo que, por ejemplo, podemos escribir:

```
x=y=z;
```

que, efectivamente, asigna el valor de z tanto a x como a y.

Ya hemos mencionado que la entrada/salida es manejada por funciones de biblioteca y que puede variar de una implementación a otra. No obstante, evidentemente es difícil escribir muchos programas sin algún tipo de E/S, de modo que terminaremos este primer capítulo considerando de modo sucinto las dos funciones principales para E/S de terminal. Una de ellas ya la hemos visto: printf, que formatea y produce valores de diversos tipos en el dispositivo de salida estándar (normalmente la pantalla). Su sintaxis completa es:

```
printf(serie_de_control, arg1,arg2,.....);
```

donde los argumentos pueden ser series o variables de cualquiera de los tipos de datos básicos.

La serie_de_control puede contener secuencias comunes de caracteres, que simplemente se producen tal como son, pero también contiene un especificador de formato para cada uno de los argumentos que, dicho sea de paso, también proporciona la cantidad de argumentos. Un especificador de formato empieza con un signo % y termina con uno de los caracteres de conversión, como vemos en la tabla *Conversión de formatos*. Entre los argumentos puede aparecer −, para especificar el ajuste a la izquierda en el campo de salida, y un número para especificar la anchura del campo en el cual se ha de imprimir el valor. Opcionalmente, éste puede ir seguido por un punto y otro número que especifique la cantidad de caracteres a imprimir realmente (en el caso de una serie) o la cantidad de posiciones tras el punto decimal (en el caso de un número float o double precision).

Secuencias Escape

\n	Nueva línea
\t	Tab horizontal
\b	Retroceso
\r	Retorno de carro
\f	Nueva página
\'	Produce el car. apóstrofo (de lo contrario, utilizado para encerrar un car.)
\"	Produce el car. del signo de comillas (de lo contrario, usado para encerrar series)
\d	Donde d es un dígito octal. Produce el car. con el cód. ASCII correspondiente
\xh	Donde h es un dígito hexa. Produce el car. con el cód. ASCII correspondiente
\e	Escape
\v	Tab vertical

Conversión de formatos

Carácter	Tipo de argumento	Acción
d	int	formato decimal
o	int	formato octal
x	int	formato hexa
u	int	formato decimal sin signo
c	char/int	salida como car. simple.
s	*char	salida como serie
e	float/double	formato mantisa-exponente
f	float/double	formato decimal normal
g	float/double	cualquiera de d, e o f que sea más corto
%		produce el carácter %

Trabajar con ventanas

Finalmente estudiaremos algunas de las facilidades operativas y las estructuras de programa del GEM

Rutas diferentes
La imagen "final" de fácil uso del GEM contradice su innata complejidad. La VDI (interface de dispositivo virtual) permite escribir código independiente de la máquina y se divide en dos partes. Una, el GDOS, cumple un papel de OS casi tradicional procesando las entradas del usuario, y la otra (los manejadores de dispositivo) se adapta a la determinada configuración de hardware que se esté utilizando. El módulo AES (*applications environment services*: servicios de entorno de aplicaciones) contiene diversas subrutinas de alto nivel para tratamiento de pantalla y periféricos

Si bien es probable que Windows, de Microsoft, llegue a ser un producto importante para sistemas de gestión, existen muchas más probabilidades de que el usuario personal encuentre el GEM más asequible. El GEM ya viene empaquetado con el nuevo Atari 520ST basado en el Motorola 68000 y también está apareciendo en máquinas tales como el Apricot y el RML Nimbus.

Así como los programadores de aplicaciones convencionales pueden escribir para un sistema operativo en vez de para una determinada máquina, los WIMP basados en GKS proporcionan una forma estandarizada de direccionar los dispositivos gráficos subyacentes. Un programa escrito para GEM, que se basa en GKS, es intencionadamente portable para todo hardware que posea el kernel necesario para software de gráficos. Los sistemas de kernel para gráficos tales como el GEM, GSX y GKS actúan a modo de un caparazón de software alrededor tanto del sistema existente como del hardware para gráficos. La entrada desde el teclado y el tratamiento de archivos se controlan mediante el sistema operativo anfitrión, como antes, pero toda manipulación de gráficos es interceptada por el "sistema operativo" de gráficos.

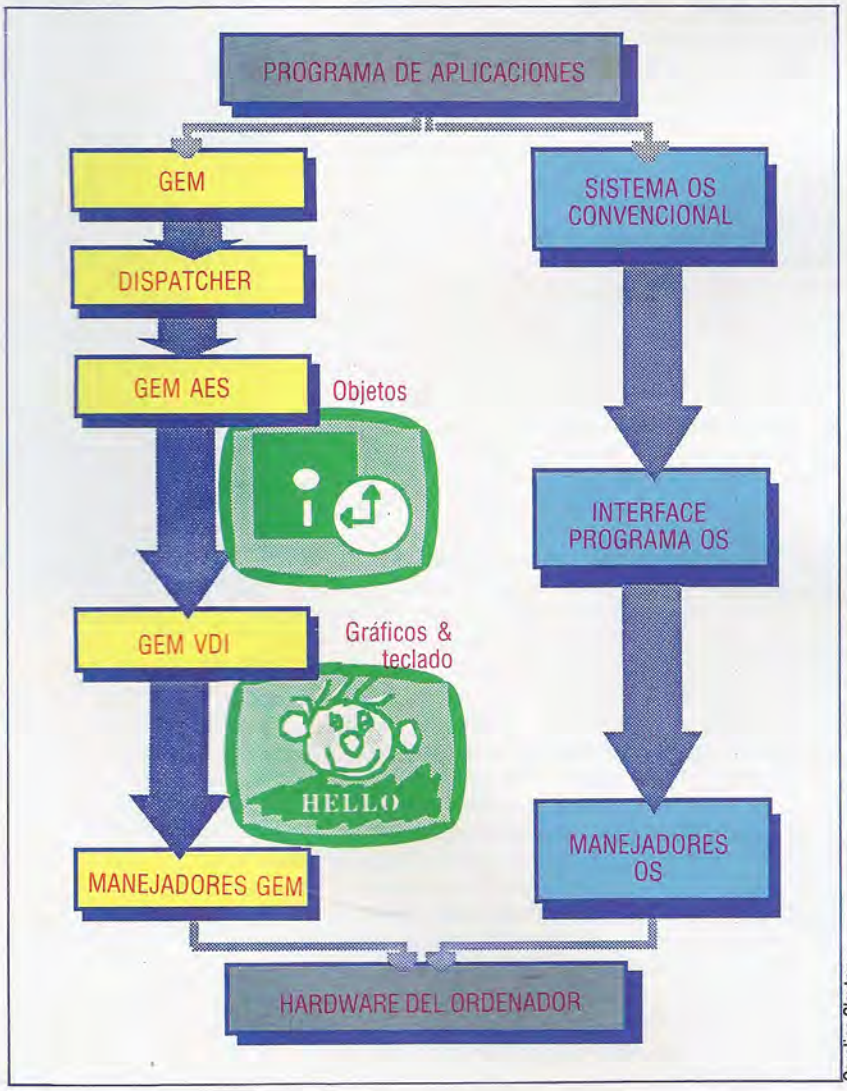
Digital Research alude a su producto como GEM DOS, pero el GEM propiamente dicho en realidad no lleva a cabo ninguna de las funciones DOS normales. Las instrucciones provenientes de dispositivos que no sean el teclado (un ratón, p. ej.) se traducen a llamadas equivalentes a las rutinas del sistema existentes. La actualización de la visualización en pantalla u otros dispositivos gráficos la maneja el sistema GSX subyacente: el corazón del GEM. Este proceso es adicional al intérprete de líneas de instrucciones del sistema convencional.

Siempre que una operación del sistema requiera una visualización de datos, la habitual visualización de carácter a carácter se sustituye por un flujo de datos gráficos por mapa de bits hacia el manejador del dispositivo apropiado. Cada dispositivo de hardware posee su propio manejador con unas características adecuadas a la función de ese dispositivo (entrada, salida, resolución, etc.). Se pueden instalar manejadores de dispositivos en la medida que sea necesario. Este plano de comunicación software entre el sistema y dispositivos de hardware específicos es lo que se denomina *interface de dispositivo virtual (virtual device interface*: VDI).

Programar para el GEM propiamente dicho es un proceso complejo. En ausencia de una poderosa biblioteca de procedimientos para tratar eventos, activar y desactivar ventanas (restableciendo los contenidos anteriores), etc., el programador de GEM debe construir cualquier rutina requerida llamando a las primitivas VDI GEM/GSX. Por ejemplo, la trivial tarea de crear una ventana y escribir algún texto en ella, como en PRINT 'Hola!' en BASIC, en GEM entraña un considerable esfuerzo de programación. En primer lugar, se deben inicializar la posición y dimensiones de la ventana a unos valores por defecto. Éstos deben ser modificables, en caso de que la ventana se "arrastre" o se cambien sus dimensiones, y se deben establecer otros atributos tales como los colores de texto y de fondo (intensidades, si fuera monocromática), caracteres fuente, tamaño, etc.

La activación de otra ventana se detecta como un *evento*, en cuyo caso esta ventana recientemente activada queda superpuesta en la aplicación actual. Puede suspender temporalmente el proceso hasta que se la reactive, pero también puede resultar necesario (con MS-Windows, Concurrent DOS y la nueva versión multitarea de GEM) llevar a cabo su proceso como una tarea de fondo. Cuando se lo vuelva a la vida, el proceso se debe visualizar a sí mismo como la ventana situada en el extremo superior de la pantalla y, significativamente, es responsable de volver a dibujarse a sí misma encima de otras ventanas visualizadas en la pantalla.

Todos los programas, independientemente de lo



Caroline Clayton

triviales que sean, deben contener todo el código para su propio tratamiento de ventanas, incluyendo el registro del contenido actual de la ventana y su posición en relación a cualquier otro dato no visualizado.

Esto a su vez significa una adición de unos 10 Kbytes de código fuente antes de que un programa empiece a hacer algo.

No existe ninguna biblioteca de procedimientos GEM o GSX general a la cual llamar para manejar estas tareas domésticas WIMP. Las casas de software que escriben para GEM están programando en MODULA-2, PASCAL, C, BCPL e incluso ensamblador. Pero a pesar de su pericia esto supone problemas, de modo que los usuarios personales deben tenerlo presente cuando piensen en escribir aplicaciones sencillas GEM en unos pocos cientos de líneas de BASIC. No obstante, si usted sabe programar en cualquiera de los lenguajes que acabamos de mencionar, están apareciendo numerosos productos que prometen proporcionar una interface de programación adecuada a las complejidades del sistema WIMP subyacente.

Juego de herramientas WIMP

La firma TDI, con sede en Bristol, fue una de las primeras que comercializó un juego de herramientas útil y asequible para la programación WIMP. Denominado *MODULA-2/ST*, les ofrece a los usuarios del Atari 520ST la posibilidad de programar seriamente para el GEM en el lenguaje que diseñara Niklaus Wirth para Lilith.

Como ya hemos visto, la empresa británica de software Prospero suministra una biblioteca de rutinas para usar con sus compiladores de gran calidad de PASCAL y FORTRAN, denominada *Prospect*. Ésta proporciona una interface de alto nivel para GSX, la primera implementación de subconjuntos de GKS de Digital Research. El sistema GSX se incorpora en el GEM, actuando como una interface de dispositivo virtual de bajo nivel (el corazón del sistema) y el *Prospect* también estará disponible para GEM.

Sin embargo, si usted está interesado en utilizar el GEM, existen varios sistemas para el usuario serio, incluyendo los siguientes:

- Un Atari 520ST (empaquetado con GEM), MODULA-2 y el Toolkit TDI.
- Un Apricot (o, mejor aun, un RML Nimbus), GEM, un compilador de buena calidad de MODULA-2, PASCAL, BCPL o C.
- Un sistema CP/M-86 o MS-DOS con GSX, PROPASCAL y la biblioteca de interface para gráficos Prospero Prospect.
- Un IBM PC, AT o 100% compatible, un ensamblador 8086 o c, y el *GEM programmer's toolkit*, de Digital Research. Éste es útil para personas muy experimentadas que posean un conocimiento profundo de programación de sistemas y muchísimo dinero y tiempo libre.

Para quienes no disponen de tiempo para programar, Digital Research posee un juego de aplicaciones que operan bajo el entorno GEM. *Desktop* es un "administrador de escritorio" para ejecutar otras aplicaciones (incluyendo multitareas, en la nueva versión de GEM), tratamiento de archivos, impresión y trazado con accesorios de escritorio tales como un reloj y una calculadora. *GEM Write* es un procesador de textos similar al WIMP, y *GEM Paint* es un paquete de dibujo para aplicaciones artísticas. Los mismos se pueden adquirir individualmente o bien en forma de paquete incluyendo a los tres, bajo el nombre de GEM Collection. Lamentablemente, los productos DR disponibles sólo se ejecutan en ordenadores IBM PC de 256 Kbytes (o máquinas totalmente compatibles) y los usuarios de otras máquinas habrán de esperar a que sus respectivos fabricantes produzcan sus propias versiones.

Las mil caras del GEM

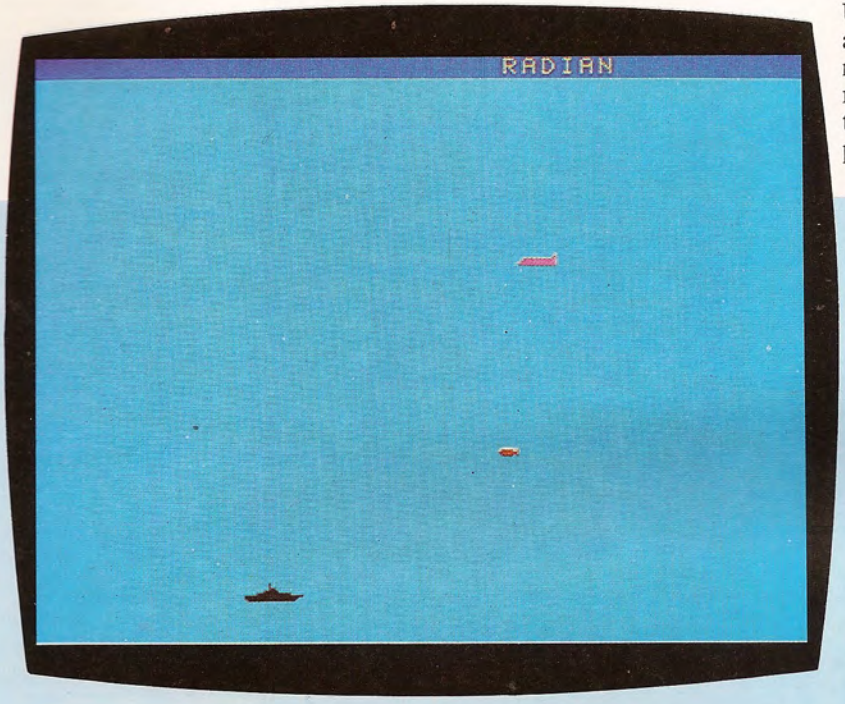
El GEM contiene dos partes funcionales principales: los *servicios de entorno de aplicaciones* (AES) y la *interface de dispositivo virtual* (VDI). Los componentes del GEM AES incluyen las bibliotecas de subrutinas (que incluyen los caparazones para ventanas, seguimiento del ratón, visualización de mensajes y dibujo de objetos) y un *dispatcher* de multitareas, actualmente limitado a tres pero con una capacidad para realizar hasta 12 tareas en la nueva versión de GEM. El AES también incorpora el buffer de artículos de escritorio (para el reloj, la calculadora, etc.), así como el buffer menú/alerta (el detector de eventos). La GEM VDI es esencialmente el núcleo de "ampliación del sistema gráfico" (GSX) derivado del GKS, pero mejorado con RASTER-OPS y FACES. El primero son las operaciones de bit AND, OR y XOR en bloques fuente y objeto. El segundo almacena los caracteres fuente en archivos que se cargan de forma dinámica. Estas facilidades de nivel más bajo están separadas en dos partes. La primera es el GDOS (*graphics device operating system*: sistema operativo de dispositivo gráfico), que es análogo a una interface OS normal. La segunda parte, los manejadores de dispositivos instalables, dependen de la configuración de hardware y dispositivos periféricos de cada sistema. Todas las funciones gráficas independientes de dispositivos y del anfitrión están contenidas en el GDOS, lo que le confiere independencia del sistema (los detalles específicos para cada dispositivo se encuentran en la sección de manejo separada). El GDOS proporciona un sistema para gráficos estándar y portable que permanece constante, independientemente de los atributos de cualquier estación de trabajo específica. El principal punto de entrada en la VDI es una única subrutina con cinco argumentos:

- Matriz de control
- Matriz de parámetros de entrada
- Matriz de coordenadas de punto de entrada
- Matriz de parámetros de salida
- Matriz de coordenadas de punto de salida

Los *bindings* del lenguaje c poseen identificadores que actúan como nombres mnemotécnicos debido a la complejidad y enorme cantidad de llamadas disponibles. De este modo, las llamadas a funciones que, por ejemplo, empiecen con v__ son todas llamadas a rutinas VDI

Exocet en el EXL 100

He aquí de nuevo el mortífero misil Exocet. La versión que presentamos ha sido escrita por Pierre Monsaut para el microordenador EXL 100 de Exelvision



Un portaaviones enemigo se ha aventurado por sus aguas territoriales y hace caso omiso a los requerimientos de identificación. A los mandos de su Mirage 2000, usted debe destruirlo antes de que constituya una amenaza para su base. Para disparar pulse una tecla cualquiera.

```
100 REM **********
110 REM * EXOCET *
120 REM **********
130 R=0
140 GOSUB 970
150 GOSUB 790
160 CALL COLOR("1MC")
170 LOCATE (AY,AX):PRINT A$;
180 IF BX>36 THEN LOCATE (BY,37):PRINT M$;:BB=
    1:GOTO 210
190 CALL COLOR("1BC")
200 LOCATE (BY,BX):PRINT B$;
210 AX=AX-1
220 IF AX<1 THEN LOCATE (AY,1):PRINT M$;:AX=38
230 BB=BB+.2
240 BX=INT (BB)
250 CALL KEY1(D3,D4)
260 IF D3<>255 AND EY=0 THEN
    EX=AX:EY=AY+1:NX=NX-1
270 IF EY<>0 THEN 310
280 FOR I=1 TO 10
290 NEXT I
300 GOTO 160
310 EX=EX-1
320 EY=EY+1
330 LOCATE (EY-1,EX+1):PRINT N$;
340 IF EX<1 THEN EX=38
350 IF EY=23 THEN 400
360 IF EY=22 AND ABS(EX-2-BX)<2 THEN GOSUB
    620
370 CALL COLOR("1RC")
380 LOCATE (EY,EX):PRINT E$;
390 GOTO 160
400 IF EX=40 THEN EX=1
410 LOCATE (EY-1,EX+1):PRINT N$;
420 EY=0
430 EX=0
440 IF NX=0 THEN 460
450 GOTO 160
460 CLS
470 IF S>R THEN R=S
480 CALL KEY1(D3,D4)
490 IF D3<>255 THEN 480
500 CALL COLOR("1BC")
510 LOCATE (10,11)
520 PRINT "PUNTOS:";S;
530 LOCATE (13,11)
540 PRINT "RECORD:";R;
550 LOCATE (16,11)
560 PRINT "OTRA?";
570 CALL KEY1(D3,D4)
580 IF D3=255 THEN 570
590 IF D3<>78 THEN 150
600 CLS
610 END
620 LOCATE (EY-1,EX+1)
630 PRINT N$;
640 S=S+10
650 LOCATE (EY,EX)
660 CALL COLOR("1bC")
670 PRINT F$;
680 FOR I=1 TO 30
690 X=INTRND(4)-2
700 Y=INTRND(6)-1
710 LOCATE (EY-Y,EX-X)
720 PRINT F$;
730 NEXT I
740 FOR I=1 TO 200
750 NEXT I
760 NX=NX+1
770 CLS
780 GOTO 160
790 CLS
800 B$=CHR$(32)&CHR$(100)&CHR$(101)&CHR$(102)
810 AX=38
820 S=0
830 BB=1
840 BX=1
850 A$=CHR$(103)&CHR$(104)&CHR$(32)
860 N$=CHR$(32)
870 MS=N$&N$&N$
880 E$=CHR$(105)
890 F$=CHR$(106)
900 EX=0
910 EY=0
920 XC=2
930 NX=20
940 BY=22
950 AY=B
960 RETURN
970 CLS ("BCb")
980 CALL CHAR(100,"000000000007FFFF7F00")
990 CALL CHAR(101,"00202038FCFFFFFFFF00")
1000 CALL CHAR(102,"0000000000E0FFFCF800")
1010 CALL CHAR(103,"0000000000003F7FFF00")
1020 CALL CHAR(104,"000000000103FFFFFF00")
1030 CALL CHAR(105,"00000000007DFF7D0000")
1040 CALL CHAR(106,"000B21800A0028001000")
1050 RETURN
```

© SYBEX

¡Cartas arriba!

Proseguimos nuestro proyecto dedicado al veintiuno ocupándonos de las rutinas de visualización de naipes y barajada del mazo para Amstrad, Spectrum y BBC. Necesitaremos definir caracteres para los palos de las cartas y los bordes, además de resolver un problema que se presenta con la función TAB en estos micros. La tabulación a una cierta posición de una línea borra todos los datos a la izquierda de esa posición TAB. Esto exige adaptar ligeramente la rutina de visualización.

Visualización de naipes y barajada

Gama Amstrad CPC

Programa principal:

```
10 REM **** Veintiuno Amstrad ****
20 GOSUB 500:REM inic matriz etc
50 REM **** Bucle del juego ****
55 GOSUB 600: REM inic juego
```

Inicialización de matrices:

```
500 REM **** inic matrices etc ****
505 INK 0,16:rosa=0:INK 1,3:rojo=1:INK
    2,0:negro=2:INK 3,26:blanco=3
512 bk$=STRING$(7,207)
513 br$=CHR$(149):li$=STRING$(7,154)
515 DIM x(2),y(2):REM siguientes posicion naipes
520 su$=CHR$(228)+CHR$(227)+CHR$(226)+CHR$(229):REM
    simbolos palos
530 DIM cn$(13):FOR i=1 TO 13:READ cn$(i):NEXT i:REM leer
    datos numero
540 DIM cd$(13):FOR i=1 TO 13:READ cd$(i):NEXT i:REM leer
    datos de patron
560 DIM dk(52,2):REM baraja naipes
570 GOSUB 3000:REM barajar mazo
580 BORDER 9:PAPER rosa:REM colores pantalla
590 RETURN
600 REM **** inic juego ****
605 CLS
610 hp(1)=1:hp(2)=1
620 x(1)=0:y(1)=0:x(2)=20:y(2)=0
625 bt=0:sb=0:GOSUB 4200:REM imprimir apuestas
630 RETURN
```

Rutina de visualización de naipes:

```
900 REM **** impresión en ****
910 LOCATE tx+1,ty+1:RETURN
1000 REM **** visualizar naipe ****
1010 GOSUB 1050:GOSUB 1100:RETURN
1050 REM **** naipe en blanco ****
1055 tx=x(pl):ty=y(pl):GOSUB 900: REM posicion
1060 PEN blanco:PRINT CHR$(150);li$:CHR$(156)
1070 FOR i=1 TO 9:tx=x(pl):ty=ty+1:GOSUB 900:PRINT
     br$;SPACE$(7);br$:NEXT i
1080 tx=x(pl):ty=ty+1:GOSUB 900:PRINT
     CHR$(147);li$;CHR$(153)
1090 RETURN
1100 REM **** detalles naipe ****
1120 tx=x(pl)+2:ty=y(pl)+1:GOSUB 900:REM posicion
1125 ct$=MID$(su$,su,1):REM seleccionar tipo de palo
1127 co=rojo:IF su> 2 THEN co=negro
1128 PEN co
1130 FOR i=1 TO 19 STEP 3
1140 cc$=MID$(cd$(cn),i,3):cl$=""
1142 FOR j=1 TO 3:c$=SPACE$(2)
1144 IF MID$(cc$,j,1)="1" THEN c$=ct$+SPACE$(1)
1146 cl$=cl$+c$:NEXT j
1150 tx=x(pl)+2:ty=ty+1:GOSUB 900:PRINT
     cl$:NEXT i
1160 REM **** añadir etiquetas angulos ****
1170 tx=x(pl)+1:ty=y(pl)+1:GOSUB 900: PRINT
     cn$(cn):REM numero
1180 ty=ty+1:GOSUB 900:PRINT ct$:REM palo
1190 tx=x(pl)+7:ty=y(pl)+9:GOSUB 900:PRINT cn$(cn):REM
     numero de abajo
1192 x(pl)=x(pl)+2:y(pl)= y (pl)+1
1195 RETURN
1200 REM **** visualizar reverso naipe ****
1210 GOSUB 1050:GOSUB 1250:RETURN
1250 REM **** reverso naipe ****
1255 tx=x(pl):ty=y(pl):GOSUB 900:REM posicion
1260 FOR i=1 TO 9:tx=x(pl):ty=ty+1:GOSUB 900:PRINT
     br$;:PEN rojo:PRINT bk$;:PEN blanco:PRINT br$:NEXT i
1270 x(pl)=x(pl)+2:y(pl)=y(pl)+1
1280 RETURN
1300 REM **** repartir un naipe ****
1310 cn=dk(dp,1):su=dk(dp,2)
1320 dp=dp+1:IF dp> 52 THEN dp=1
1330 IF fl=1 THEN GOSUB 1200:RETURN:REM visualizar reservo
     naipe
1335 GOSUB 1000:RETURN:REM visualizar naipe
2000 REM **** datos numeros naipe ****
2010 DATA A,2,3,4,5,6,7,8,9,T,J,Q,K
2100 REM **** datos visualizacion naipes ****
2110 DATA "00000000001000000000"
2120 DATA "00001000000000010000"
2130 DATA "00001000001000010000"
2140 DATA "00010100000000101000"
2150 DATA "00010100001000101000"
2160 DATA "00010100101000101000"
2170 DATA "00010101010000101000"
2180 DATA "00010101010101010000"
2190 DATA "10100010101010000101"
2200 DATA "10101010000101010101"
2210 DATA "00000000001000000000"
2220 DATA "00000000001000000000"
2230 DATA "00000000001000000000"
```

Mezclando la baraja:

```
3000 REM **** barajar el mazo ****
3005 RANDOMIZE TIME:dp=1
3007 FOR i=1 TO 52:dk(i,1)=0:NEXT i
3010 FOR i=1 TO 4:FOR j=1 TO 13
3020 ep=INT(RND(1)* 52)+1:REM seleccionar punto entrada
3030 IF dk(ep,1)=0 THEN 3050
3040 ep=ep+1:IF ep>52 THEN ep=1
3050 GOTO 3030
3050 dk(ep,1)=j:dk(ep,2)=i:NEXT j,i
3060 RETURN
```

BBC MICRO

Nota: Digite PAGE=$0E00 antes de entrar el programa o cargarlo desde disco o cinta

Programa principal:

```
10 REM VEINTIUNO BBC
15 MODE 1
20 GOSUB 500
50 REM
55 GOSUB 600
```

Inicialización de matrices:

```
500 REM
510 SP$=" ":FOR I=1 TO 39:SP$=SP$+" ":NEXT I
512 BK$=" ":LI$=" ":FOR I=1 TO
    7:LI$=LI$+CHR$(195):BK$=BK$+CHR$(166): NEXT I
```

```
513 BR$=CHR$(194)
515 DIM X(2),Y(2)
520 SU$=CHR$(211)+CHR$(218)+CHR$(216)+
    CHR$(193)
530 DIM CN$(13):FOR I=1 TO 13:READ CN$(I):
    NEXT
535 CN$(10)=CHR$(128)
540 DIM CD$(13):FOR I=1 TO 13:READ CD$(I):
    NEXT
560 DIM DK(52,2)
570 GOSUB 3000
571 COLOUR 131:COLOUR 0
572 VDU 23,166,165,66,90,36,90,165,90,66
573 VDU 23,195,0,0,0,255,255,0,0,0
574 VDU 23,194,24,24,24,24,24,24,24,24
575 VDU 23,193,16,56,124,254,254,238,84,56
576 VDU 23,216,24,60,90,255,255,90,24,60
577 VDU 23,218,16,56,124,254,254,124,56,16
578 VDU 23,211,36,126,255,255,255,126,60,24
579 VDU 23,213,0,0,0,15,31,28,24,24
580 VDU 23,201,0,0,0,240,248,56,24,24
581 VDU 23,202,24,24,28,31,15,0,0,0
582 VDU 23,203,24,24,56,248,240,0,0,0
583 VDU 23,128,206,219,219,219,219,219,206,0
590 RETURN
600 REM
601 DB=0:CS=0
605 CLS
620 X(1)=0:Y(1)=0:X(2)=20:Y(2)=0
630 RETURN
```

Rutina de visualización de naipes:

```
900 REM **** IMPRIMIR EN ****
910 PRINT TAB(TX,TY);:RETURN
1000 REM
1010 GOSUB 1050:GOSUB 1100:RETURN
1050 REM
1055 TX=X(PL):TY=Y(PL):GOSUB 900:REM POSICION
1060 COLOUR 0:PRINT TAB(TX,TY);CHR$(213);LI$;
     CHR$(201)
1070 FOR I=1 TO 9:PRINT TAB(TX,TY+I);BR$" ";BR$:
     NEXT I
1080 PRINT TAB(TX,TY+10);CHR$(202);LI$;CHR$(203)
1090 RETURN
1100 REM
1120 TX=X(PL)+2:TY=Y(PL)+1:GOSUB 900
1125 CT$=MID$(SU$,SU,1)
1127 COLOUR 1:IF SU>2 THEN COLOUR 0
1130 FOR I=1 TO 19 STEP 3
1140 CC$=MID$(CD$(CN),I,3):CL$=""
1142 FOR J=1 TO 3:C$=CHR$(9)+CHR$(9)
1144 IF MID$(CC$,J,I)="1" THEN C$=CT$+CHR$(9)
1146 CL$=CL$+C$:NEXT J
1150 PRINT TAB(X(PL)+2,TY+((I+3)/3));CL$:NEXT I
1170 TX=X(PL)+1:TY=Y(PL)+1:GOSUB 900:PRINT
     CN$(CN)
1180 TY=TY+1:GOSUB 900:PRINT CT$
1190 TX=X(PL)+7:TY=Y(PL)+9:GOSUB 900:PRINT
     CN$(CN)
1192 X(PL)=X(PL)+2:Y(PL)=Y(PL)+1
1195 RETURN
1200 REM
1210 GOSUB 1050:GOSUB 1250:RETURN
1250 REM
1255 TX=X(PL):TY=Y(PL)+1:GOSUB 900
1260 FOR I=1 TO 9:PRINT TAB(TX,I);BR$;BK$;BR$:
     NEXT I
1270 X(PL)=X(PL)+2:Y(PL)=Y(PL)+1
1280 RETURN
1300 REM
1310 CN=DK(DP,1):SU=DK(DP,2)
1320 DP=DP+1:IF DP>52 THEN DP=1
1330 IF FL=1 THEN GOSUB 1200:RETURN
1335 GOSUB 1000:RETURN
2010 DATA A,2,3,4,5,6,7,8,9,T,J,Q,K
2110 DATA"00000000001000000000"
2120 DATA"00001000000000010000"
2130 DATA"00001000001000010000"
2140 DATA"00010100000000101000"
2150 DATA"00010100001000101000"
2160 DATA"00010100010100010100"
2170 DATA"00010101010100010100"
2180 DATA"00010101010101010100"
2190 DATA"10100010101010100101"
2200 DATA"10101010100010101010"
2210 DATA"00000000001000000000":REM J
2220 DATA"00000000001000000000":REM Q
2230 DATA"00000000001000000000":REM K
```

Mezclando la baraja:

```
3000 REM
3005 R=RND(-TIME):DP=1
3007 FOR I=1 TO 52:DK(I,1)=0:NEXT I
3010 FOR I=1 TO 4:FOR J=1 TO 13
3020 EP=INT(RND(1)*52)+1
3030 IF DK(EP,1)=0 THEN 3050
3040 EP=EP+1:IF EP>52 THEN EP=1
3045 GOTO 3030
3050 DK(EP,1)=J:DK(EP,2)=I:NEXT J,I
3060 RETURN
```

Sinclair Spectrum:

Programa principal:

```
10>REM **** VEINTIUNO SPECTRUM ****
15 PRINT "ESPERE POR FAVOR..."
16 POKE 23658,8
20 GO SUB 500: REM INIC MATRICES ETC
50 REM **** BUCLE DEL JUEGO ****
55 GO SUB 600:REM INIC JUEGO
```

Inicialización de matrices:

```
500>REM **** INIC MATRICES ETC ****
510 LET S$="":FOR I=1 TO 25: LET S$=S$+"":
    NEXT I
511 LET S$=S$+S$(TO 14)
512 LET B$="": LET L$="": FOR I=1 TO 7: LET L$=L$+
    CHR$ 145: LET B$= B$+CHR$ 144:NEXT I
513 DET D$=CHR$ 146
515 DIM X(2): DIM Y (2): REM SIGUIENTES POSICIONES
    NAIPES
520 LET U$=CHR$ 147+CHR$ 148+CHR$ 149+CHR$ 150:
    REM TIPOS DE PALO
530 DIM N$(13): FOR I=1 TO 13: READ N$(I):NEXT I:REM LEER
    DATOS NUMEROS
540 DIM C$(13,21): FOR I=1 TO 13: READ C$(I): NEXT I: REM
    LEER DATOS PATRON
560 DIM K(52,2): REM PREPARAR MATRIZ BARAJA
    NAIPES
570 GO SUB 3000: REM BARAJAR MAZO
580 BORDER 4: PAPER 7: INK 9: CLS
581 REM
582 FOR L=USR "A" TO USR "L"+7
583 READ US: POKE L,US: NEXT L
590 RETURN
600 REM **** INIC JUEGO ****
602 LET DB=0: LET CS=0
605 CLS
620 LET X(1)=0: LET Y(1)=0: LET X(2)=14: LET Y(2)=0
630 RETURN
```

Rutina de visualización de naipes:

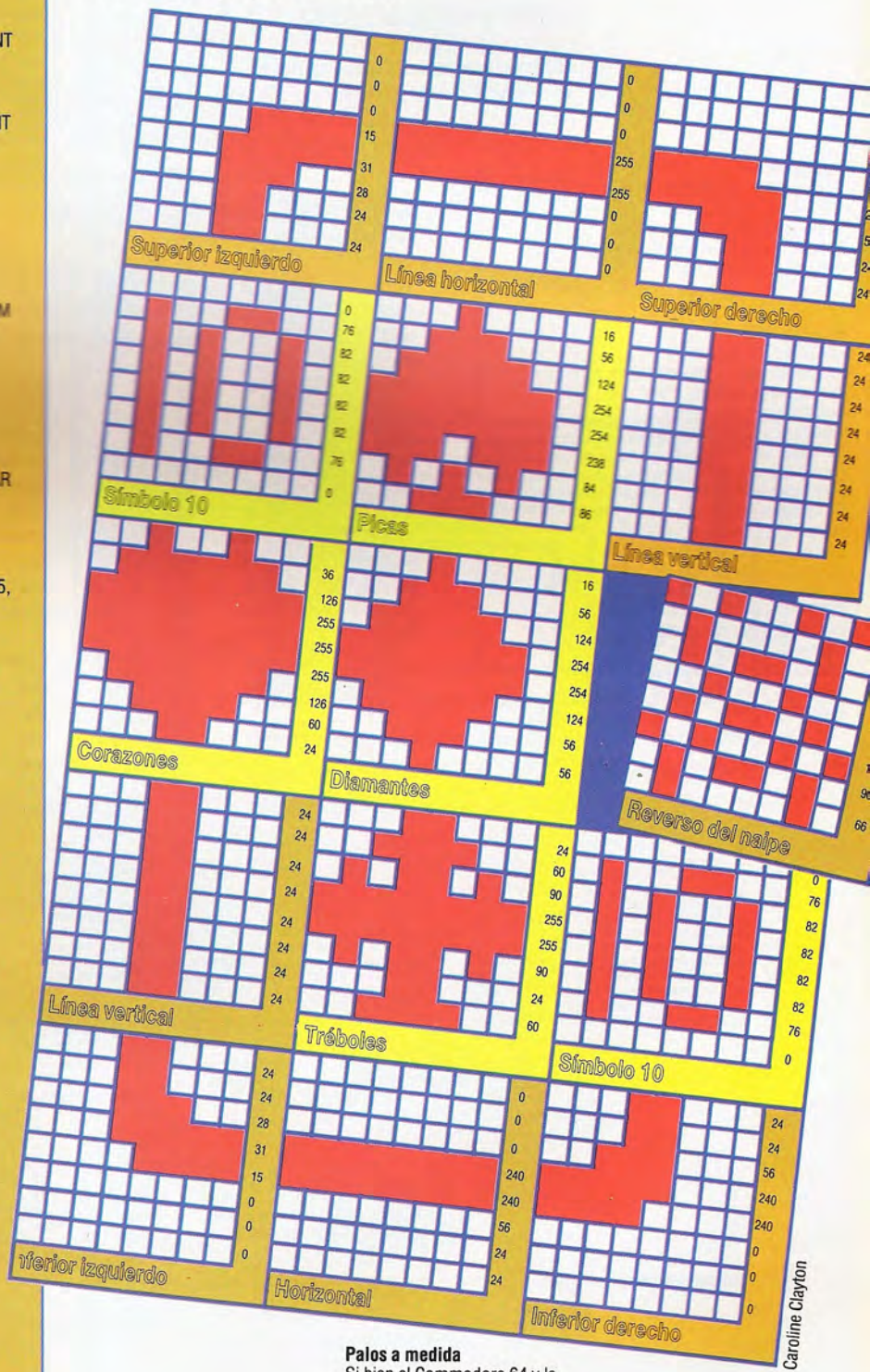
```
900 >REM **** IMPRIMIR EN ****
910  PRINT AT TY,TX;:RETURN
1000 REM **** VISUALIZAR NAIPE ****
1010 GO SUB 1050: GO SUB 1100: RETURN
1050 REM **** NAIPE EN BLANCO ****
1055 LET TX=X(PL)+1:LET TY=Y(PL): GO SUB 900: REM POSI-
     CION
1060 PRINT AT Y(PL),TX;CHR$ 151;L$;CHR$ 152
1070 FOR I=1 TO 9: PRINT AT Y(PL)+I,TX:CHR$ 146;" ";CHR$
     146: NEXT I
1080 PRINT AT 10+Y(PL),TX;CHR$ 153;L$;CHR$ 154
1090 RETURN
1100 REM **** DETALLES NAIPE ****
1120 LET TX=X(PL)+2: LET TY=Y(PL)+2:GO SUB 900: REM
     POSICION
1125 LET T$=U$(SU): REM SELECCIONAR TIPO DE
     PALO
```

```
1127 INK 0: IF SU>2 THEN INK 2:REM SELECCIONAR
     COLOR
1130 FOR I=1 TO 19 STEP 3
1140 LET Z$=C$(CN)(I TO I+2)
1142 FOR J=1 TO 3: LET W$=" "
1143 PRINT " ";
1144 IF Z$(J)="1" THEN PRINT T$;
1145 IF Z$(J)="0" THEN PRINT " ";
1146 NEXT J
1150 PRINT AT ((I+3)/3)+TY,X(PL)+2;:NEXT I
1160 REM **** AÑADIR ETIQUETAS ANGULOS ****
1170 LET TX=X(PL)+2: LET TY=Y(PL)+1: GO SUB 900: PRINT
     N$(CN): REM POSICION
1180 LET TY=TY+1: GO SUB 900: PRINT T$: REM
     PALO
1190 LET TX=X(PL)+8: LET TY=Y(PL)+9: GO SUB 900: PRINT
     N$(CN): REM POSICION
1192 LET X(PL)=X(PL)+2: LET Y(PL)=Y(PL)+1
1193 INK 0
1195 RETURN
1200 REM **** VISUALIZAR REVERSO NAIPE ****
1210 GO SUB 1050: INK 1: GOSUB 1250: INK 0:
     RETURN
1250 REM **** REVERSO NAIPE ****
1255 LET TX=X(PL)+2: LET TY=Y(PL)+1: GO SUB 900: REM
     POSICION
1260 FOR I=1 TO 8: PRINT AT I,TX;BS: NEXT I
1270 LET X(PL)=X(PL) +2: LET Y (PL)=Y(PL)+1
1280 RETURN
1300 REM **** REPARTIR UN NAIPE ****
1310 LET CN=K(DP,1): LET SU=K(DP,2)
1320 LET DP=DP+1: IF DP>52 THEN LET DP=1
1330 IF FL=1 THEN GO SUB 1200: RETURN: REM VISUALIZAR
     REVERSO NAIPE
1335 GO SUB 1000: RETURN: REM VISUALIZAR
     NAIPE
2000 REM **** DATOS NUMEROS DE NAIPES ****
2010 DATA "A","2","3","4","5","6","7","8","9",CHR$ 155,
     "J","Q","K"
2100 REM **** DATOS VISUALIZACION NAIPES ****
2110 DATA "0000000000100000000000": REM A
2120 DATA "0000100000000000010000": REM 2
2130 DATA "0000100000100000010000": REM 3
2140 DATA "0001010000000000101000": REM 4
2150 DATA "0001010000100000101000": REM 5
2160 DATA "0001010001010001010000": REM 6
2170 DATA "0001010101010001010000": REM 7
2180 DATA "0001010101010101010000": REM 8
2190 DATA "1010001010101010000101": REM 9
2200 DATA "1010101010001010101010": REM 10
2210 DATA "0000000000100000000000": REM J
2220 DATA "0000000000100000000000": REM Q
2230 DATA "0000000000100000000000": REM K
2240 REM UDG
2250 DATA 165,66,90,36,90,165,90,66
2251 DATA 0,0,0,255,255,0,0,0
2252 DATA 24,24,24,24,24,24,24,24,
2253 DATA 16,56,124,254,254,238,84,56
2254 DATA 24,60,90,255,255,90,24,60
2255 DATA 16,56,124,254,254,124,56,16
2256 DATA 36,126,255,255,255,126,60,24
2257 DATA 0,0,0,15,31,28,24,24
2258 DATA 0,0,0,240,248,56,24,24
2259 DATA 24,24,28,31,15,0,0,0
2260 DATA 24,24,56,248,240,0,0,0
2261 DATA 0,76,82,82,82,82,76,0
```

Mezclando la baraja:

```
3000 >REM **** BARAJANDO EL MAZO ****
3005 RANDOMIZE: LET DP=1
3007 FOR I=1 TO 52: LET K(I,1)=0: NEXT I
3010 FOR I=1 TO 4: FOR J=1 TO 13
3020 LET EP= INT (RND*52)+: REM SELECCIONAR PUNTO
     ENTRADA
3030 IF K(EP,1)=0 THEN GO TO 3050
3040 LET EP=EP+1: IF EP>52 THEN LET EP=1
3045 GO TO 3030
3050 LET K(EP,1)=J: LET K(EP,2)=I: NEXT J:
     NEXT I
3060 RETURN
```

En el borde
Todas estas versiones del veintiuno utilizan gráficos de caracteres para dibujar el borde de los naipes. En las versiones para el BBC Micro y el Spectrum hemos de definir nosotros los caracteres de los bordes. En la versión para el BBC Micro, todas las definiciones de palo y borde se realizan utilizando la instrucción VDU 23 entre 572-583. En la versión para el Spectrum, las definiciones de caracteres se retienen como DATA entre 2240-2261 y el READ forma parte de la subrutina de inicialización entre 581-583



Palos a medida
Si bien el Commodore 64 y la gama de micros Amstrad CPC poseen caracteres de palos de baraja como parte de sus juegos de caracteres estándares, en la versiones para el BBC Micro y el Spectrum necesitamos definir nuestros caracteres de palos

Ir y venir

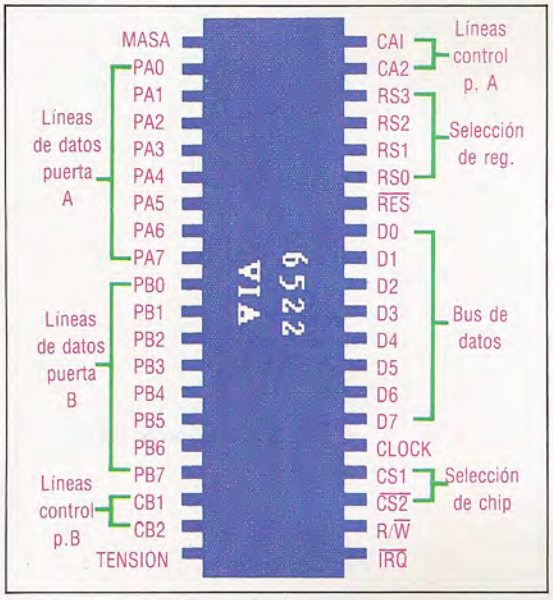
Centraremos nuestra atención en las facilidades de E/S en serie y en paralelo

Hasta ahora hemos analizado el procesador y cómo se comunica con la memoria ROM y RAM. El sistema que hemos descrito es autocontenido, lo que, por supuesto, limita su uso. El procesador/memoria básico se comunica con otros dispositivos de muchas maneras. Las entradas se pueden obtener utilizando un teclado, una palanca de mando o a través de algún dispositivo externo unido al sistema a través de una puerta, y la salida normalmente se efectúa a través de una pantalla o una impresora.

No podemos conectar directamente dispositivos de E/S al bus de datos principal porque los datos de salida normalmente no están presentes en el mismo el tiempo suficiente como para que un dispositivo externo los utilice. Por el contrario, los datos que entran se deben retener en el bus de datos el tiempo suficiente para que el procesador se ocupe de ellos. En consecuencia, a nivel general la tarea de una interface de E/S consiste en congelar los datos el tiempo suficiente para que se realice la entrada o salida y para sincronizar las operaciones E/S.

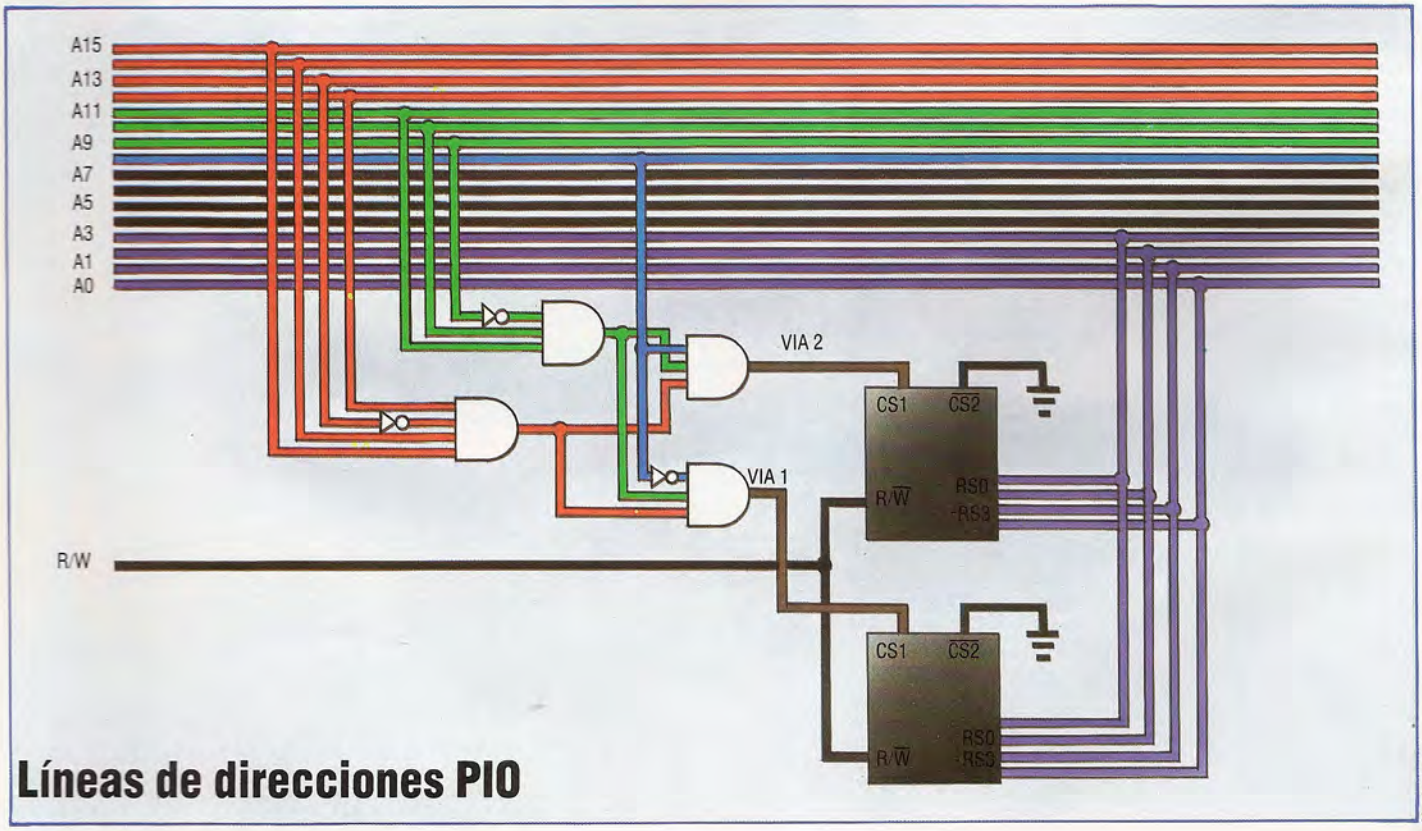
Por supuesto, la mayoría de los dispositivos de E/S hacen muchas más cosas además de proporcionar un par de *latches* y líneas de control. Pero antes de seguir adelante, vale la pena hacer la distinción entre los dos tipos principales de dispositivo de E/S con que cuentan la mayoría de los micros. Si ignoramos el problema de la salida a la pantalla, que por lo general se maneja a través de sistemas de circuitos especializados (debido, entre otras cosas, a la velocidad de transferencia de datos requerida para actualizar una pantalla), el tipo de interface E/S utilizada suele depender del método en que estén unidos los dispositivos, que será un enlace ya sea en serie o bien en paralelo. Por lo tanto, los dispositivos de E/S se dividen en dos categorías principales: dispositivos PIO y dispositivos SIO.

Ya hemos visto de forma detallada métodos de comunicaciones en serie. En particular, nuestra serie sobre la interface MIDI en el apartado *Bricolaje* proporciona un ejemplo clásico de cómo imple-



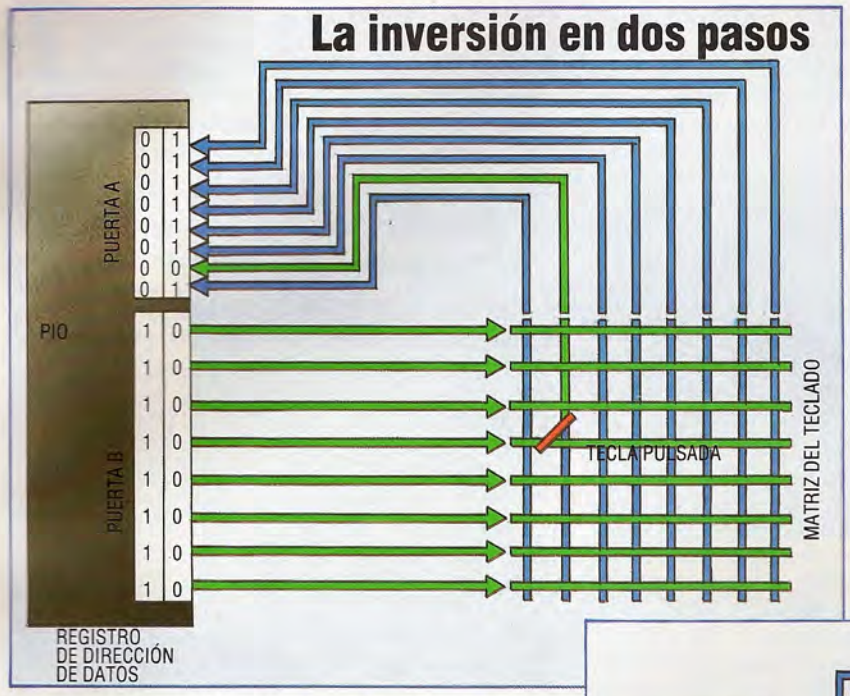
Patillas in/out
El chip VIA 6522 utilizado en el BBC Micro constituye un ejemplo típico de chip de E/S en paralelo: posee dos puertas de ocho bits, y líneas al bus de datos

El código PIO
Los chips PIO se pueden programar colocando valores en sus registros internos. Para permitir el acceso a estos registros, el procesador ha de ser capaz de "verlos" en su propio espacio de direcciones. Este ejemplo muestra dos chips PIO preparados para que su contenido aparezca en $DC00 y $DD00. Se decodifican los siete bits superiores del bus de direcciones y A8 selecciona $DC00 o $DD00. Puesto que cada PIO posee 16 registros internos, los cuatro bits inferiores del bus de direcciones se utilizan para direccionar un registro individual

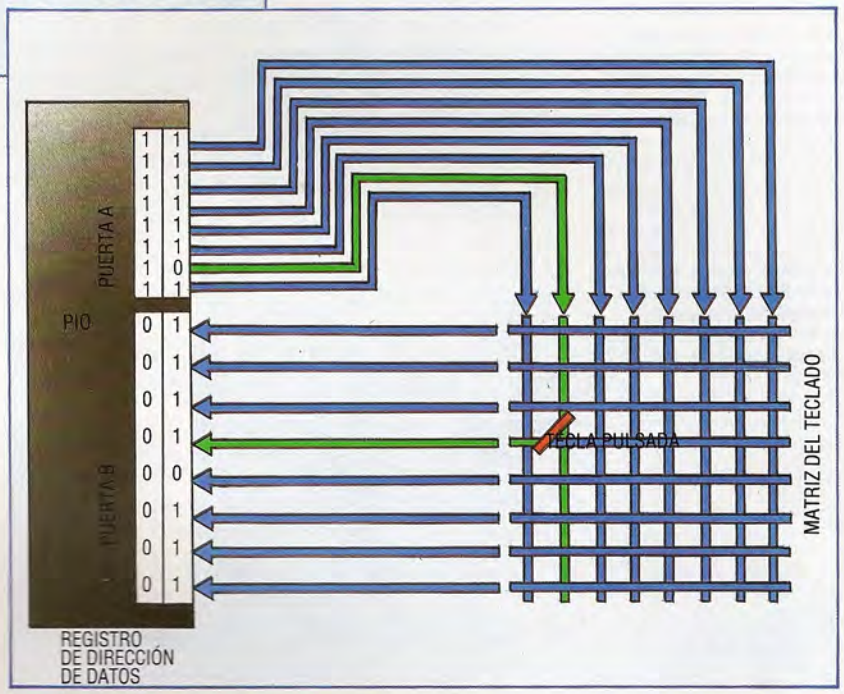


Líneas de direcciones PIO

Kevin Jones



La clave del problema
La técnica de inversión de líneas es uno de los diversos métodos que se utilizan para identificar pulsaciones de teclas en el teclado, y emplea la programabilidad direccional de las dos puertas PIO de ocho bits para producir ceros a lo largo de las filas de la matriz del teclado y producir luego el resultado recibido en la otra puerta a lo largo de las columnas. El código de 16 bits resultante retenido en los registros de datos de las dos puertas identifica inequívocamente la tecla pulsada



mentar un dispositivo de comunicaciones asíncronas. Los componentes esenciales de una interface de ese tipo son registros de desplazamiento, que permiten convertir los bits de datos en paralelo provenientes del bus de datos en un flujo en serie, y la conversión de un flujo de entrada a un formato en paralelo. Este dispositivo suele manejar la generación y comprobación de paridad automática.

Para todo sistema de ordenador personal, el chip PIO reviste una importancia más inmediata. Sin embargo, el término PIO no se utiliza con carácter universal, y términos tales como VIA (*versatile interface adaptor*: adaptador versátil de interface), PIA (*peripheral interface adaptor*: adaptador de interface para periféricos) y CIA (*complex interface adaptor*: adaptador de interface compleja) se refieren esencialmente al mismo dispositivo.

Los micros personales con facilidades completas para conexión en interface, tales como el BBC Micro y el Commodore 64, están equipados con dos chips PIO, uno de los cuales normalmente está dedicado a los dispositivos de E/S incorporados, como el teclado y las puertas para palanca de mando. El segundo PIO se puede emplear conjuntamente con una impresora en paralelo y una puerta para el usuario, dando al usuario o a los fabricantes un acceso independiente al sistema de E/S.

En el diagrama (página contigua, arriba) vemos el chip VIA 6522 utilizado en el BBC Micro. Disponiendo de 40 patillas, este paquete proporciona ocho enlaces de bus de datos y dos puertas de ocho bits que se pueden utilizar para entrada o salida. Otras líneas incluyen controles de interrupciones y de lectura/escritura, bits de selección de registro y líneas de comunicación para las dos puertas E/S.

Aunque un chip PIO es, desde el punto de vista electrónico, un dispositivo sumamente complejo, resulta bastante sencillo comprenderlo en su aspecto de funcionamiento. Típicamente, un chip PIO incluido en un micro de ocho bits tiene dos puertas de E/S (A y B) con control de dirección de datos individual en cada bit de puerta proporcionado por registros de dirección de datos y líneas de control de comunicación separadas. Muchos PIO incluyen también un registro de desplazamiento que se puede utilizar para transmitir o recibir datos a lo largo de una de las líneas de puerta de comunicación. Éstas suelen estar bajo el control de temporización interna de uno de los dos temporizadores, o bajo una fuente temporizadora externa entrada a través de otra línea de comunicación.

Un registro de estado de interrupciones de ocho bits utiliza cada bit para indicar una solicitud de interrupción desde un cierto número de fuentes: las líneas de control de comunicación, los temporizadores o los registros de desplazamiento. Un registro de permisión de interrupciones programable permite que el programador seleccione cuál (si la hubiera) de las fuentes de interrupciones PIO necesita realmente interrumpir al procesador.

Concentrémonos ahora en cómo un PIO se puede conectar al sistema procesador/memoria. La mayoría de los sistemas con facilidades de E/S com-

pletas emplean dos chips PIO. Supongamos que quisiéramos que los registros internos de estos dos dispositivos aparecieran en $DC00 y $DD00. Para pasar datos a los PIO y recibir datos provenientes de ellos, el procesador debe ser capaz de tratar los registros internos del PIO como si fueran posiciones de memoria y direccionarlos de la misma forma. Para hacer que los registros PIO aparezcan en el espacio de direcciones, debemos, por lo tanto, manipular algunas líneas del bus de direcciones.

El diagrama ilustra cómo se tratan los ocho bits superiores del bus de direcciones para seleccionar cualquiera de los PIO. Los cuatro bits superiores se operan para producir un uno cuando 1101 ($D) está presente. Los tres bits siguientes se operan para producir un uno cuando 110 está presente. La selección entre $DC0 y $DD0 se realiza realmente mediante el bit de dirección A8. Los cuatro bits inferiores conectan directamente con las cuatro patillas de selección de registro del PIO. Por tanto, los registros internos del PIO 1 se pueden direccionar como las posiciones de $DC00 a $DC0F, y los del PIO 2 se pueden direccionar como de $DD00 a $DD0F.

Normalmente un PIO es exclusivo del sistema, para la conexión de palancas de mando y el teclado del ordenador principal. Existen varios métodos de conectar un teclado complejo de 64 teclas, el más simple de los cuales consiste en construir el teclado de modo que tenga por debajo una matriz de cables, dispuestos en ocho filas y ocho columnas. Entonces se conectan las líneas de fila a una puerta PIO y las líneas de columna a otra, como se indica en el diagrama. Cuando se pulsa una tecla, se conectan los cables de la fila y columna correspondientes a la intersección.

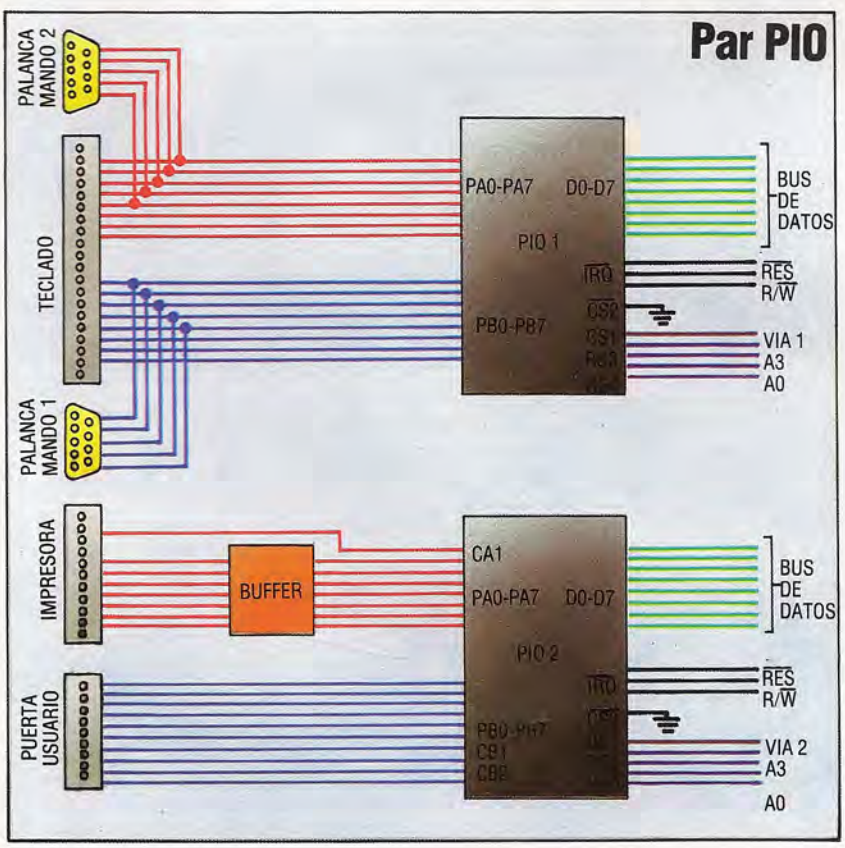
El método de inversión de línea utilizado para detectar qué tecla se ha pulsado se basa fundamentalmente en software, dependiendo de la capacidad que posee la puerta del PIO para programarse para entrada o salida. El paso 1 supone establecer la puerta B en salida y colocar cero en su registro de datos. La puerta A se establece para entrada y, dado que las líneas de columna están todas retenidas *high*, el bit del registro de datos de la puerta B correspondiente a la tecla pulsada estará *low*, mientras que todos los otros estarán *high*.

El paso 2 implica invertir la dirección de datos de las dos puertas. Ahora la puerta B actúa como la puerta de entrada y cada bit se establecerá en uno, a excepción del bit que corresponde a la fila de teclas que se esté pulsando. El código de 16 bits retenido ahora en los registros de las dos puertas se puede utilizar para identificar la posición de la ROM de caracteres que contiene el código de ocho bits del carácter cuya tecla se ha pulsado.

Otro de los métodos para identificación de tecla es la exploración de filas. En este caso cada columna se establece *high*, buscando a la vez un correspondiente nivel *high* en una de las líneas de fila y chips exclusivos que decodifican la matriz del teclado para generar códigos directamente.

Ya hemos visto la decodificación de direcciones PIO. El diagrama final (abajo) muestra cómo se puede utilizar un par de chips PIO para acceder al teclado y proporcionar puertas para el usuario e impresora en paralelo. Mientras que el primero puede depender completamente de la conexión al teclado, el segundo PIO dispone de dos puertas de ocho bits. La primera de ellas se puede utilizar como una puerta de salida para una impresora, a través de un buffer adecuado para proteger al PIO contra conexiones incorrectas en la puerta para impresora. La segunda queda, entonces, disponible como puerta para el usuario.

Centinelas gemelos
Este diagrama muestra cómo se pueden utilizar dos chips PIO para conectar en interface un teclado, impresora en paralelo, palanca de mando y puerta para el usuario al bus de datos. Una de las principales tareas del firmware o las rutinas kernel que se proporcionan con el sistema es programar los PIO de modo que manipulen los datos entrantes y salientes. Por consiguiente, para la mayoría de las aplicaciones el programador de lenguaje máquina puede llamar a las rutinas kernel en vez de diseñar las suyas propias. No obstante, en algunas ocasiones resulta útil programar el PIO directamente, como para emplear los temporizadores PIO o utilizar la puerta para el usuario



Programando el PIO

El chip PIO es programable, es decir, posee registros internos a los que el procesador principal puede acceder a través de su sistema de direccionamiento normal. Colocando ciertos valores en estos registros se puede hacer que el PIO se comporte de ciertas maneras. Por ejemplo, se pueden programar independientemente cada una de las ocho líneas de datos que componen cada puerta E/S, de modo que actúen como líneas de entrada o salida mediante un registro de dirección de datos. En consecuencia, se puede hacer que una misma puerta sea para entrada y salida simultáneamente utilizando diferentes bits. De este modo, diseñando un software manejador apropiado, un PIO se puede aplicar a la mayoría de los problemas de conexión en interface. Además, debido a que las operaciones de E/S se han de realizar a intervalos rígidamente definidos, los PIO incorporan temporizadores de cuenta atrás de 16 bits que se pueden cargar con un valor y reducir con cada impulso del reloj del sistema, generando una señal de interrupción cuando se produce un desbordamiento negativo (el contador intenta reducir más atrás de cero) del temporizador. En consecuencia, a través de los temporizadores PIO se pueden controlar las comunicaciones de datos síncronas, en virtud de las cuales se transmiten o reciben datos a intervalos regulares

Tres grandes

Analizaremos tres paquetes que han tenido gran aceptación entre los usuarios personales

El tratamiento de textos es, indudablemente, una de las aplicaciones para microordenadores personales más conocidas. Los programas van desde sencillos editores de textos de precio muy reducido hasta sofisticados y costosos paquetes de tratamiento de textos basados en ROM o en disco. Sin embargo, la bondad de cualquier programa está en relación directa con la del hardware para el cual ha sido diseñado, y los paquetes escritos para la mayoría de los micros personales carecen de muchas de las facilidades más complejas que ofrecen los programas para ordenadores de gestión.

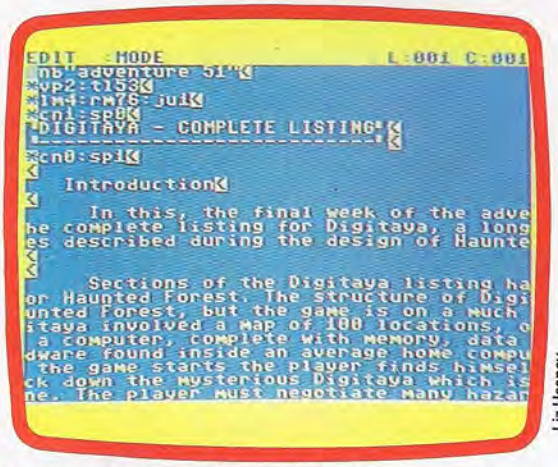
La restricción fundamental de un procesador de textos para un ordenador de ocho bits viene dada por las limitaciones de memoria. Toda facilidad adicional incluida en el software a menudo ocupa un espacio que, en otras circunstancias, estaría disponible para almacenar texto. Esto ha demostrado ser un obstáculo importantísimo en la implementación del *WordStar* para el Amstrad CPC 464 y 664. Si bien estas máquinas pueden usar el sistema operativo CP/M, la memoria disponible es insuficiente para implementar adecuadamente el programa *WordStar* completo. No obstante, se ha desarrollado una versión a pequeña escala del programa, denominada *Pocket WordStar* (WordStar de bolsillo).

Otro problema con el que tropiezan los programadores de procesadores de textos para micros personales es el método de almacenamiento de apoyo. La mayoría de los micros personales continúan basándose en cassettes para el almacenamiento de apoyo; por lo tanto, tanto el programa como el texto se han de retener enteramente en la memoria del ordenador, imponiendo demandas aún mayores en la memoria disponible.

"Easy script"

Para el Commodore 64 se han escrito muchos paquetes para tratamiento de textos, pero posiblemente el más difundido de ellos sea el *Easy script* (debido fundamentalmente a que Commodore lo suministra empaquetado con la unidad de disco).

El principal problema para quien desee escribir un paquete de tratamiento de textos para el Commodore 64 es la visualización en pantalla, que sólo puede producir una resolución de caracteres máxima de 40 por 25. En consecuencia, el programador se ve obligado a sacar el máximo provecho de las 40 columnas, lo que significa que el *Easy script* no cuenta con el lujo que suponen las pantallas de ayuda, que ocuparían demasiado espacio en la pantalla. Además, las limitaciones de pantalla impiden que el *Easy script* posea alguna facilidad para desplazamiento de palabras, aunque es posible con-



Liz Heaney

templar el texto en formato definitivo antes de enviarlo a la impresora.

Para superar este problema, los programadores del *Easy script* han optado por utilizar la pantalla como una "ventana" del documento. Los márgenes se pueden fijar en 240 columnas a lo ancho, y cuando una línea llega al límite de las 40 columnas la pantalla se desplaza lateralmente hasta llegar a la limitación del margen. En este punto, el retorno a la siguiente línea resultará un tanto abrupto. Si bien este método no es el ideal, el usuario suele acostumbrarse fácilmente al mismo o, de lo contrario, puede escribir en 40 columnas y después reformatear el texto antes de enviarlo a la impresora.

A las diversas instrucciones de formateo y edición de texto para impresión que son esenciales para un programa de tratamiento de textos se accede a través de las teclas de función del Commodore 64. Pulsando una de ellas usted entrará en una modalidad determinada y, si a continuación efectúa una pulsación de tecla, se seleccionará la opción adecuada. Por tanto, a las instrucciones de edición se accede pulsando F1, lo que se indica mediante la palabra Edit, que aparece intermitentemente en la parte superior de la pantalla. El trazado del texto y la página se seleccionan pulsando F3, que produce un asterisco invertido en la posición del cursor. A continuación se pueden determinar los códigos para establecer los márgenes, ajustar el texto, etc. *Easy script* hace un uso intensivo de estas y otras instrucciones incorporadas en el texto, las cuales sólo actúan cuando se envía el texto a la impresora. Esto significa que muchas de las funciones no se verán en la pantalla, reduciendo la capacidad de ver el aspecto que tendrá el documento final impreso.

Easy script contiene casi todas las facilidades que cabría esperar de un paquete de tratamiento de textos, tales como manipulación de bloques enteros, lo que incluye la copia, transferencia y archivado separado de los bloques, si bien hay que destacar que algunas de las facilidades funcionan mejor que otras. La función Buscar/Reemplazar, por ejemplo, es notablemente lenta, en especial cuando se utilizan archivos unidos que se han cargado por separado y que luego se han buscado.

"Tasword II"

Un programa para tratamiento de textos que se utiliza ampliamente en la gama Amstrad y en el Spectrum+ es el *Tasword II*. Al cargarlo, este programa guarda un sorprendente parecido con el *WordStar*. En la parte superior de la pantalla hay un menú de Ayuda que contiene una lista de las instrucciones (y sus caracteres de control), dividida en secciones lógicas. Debajo de este menú está la zona de 16 por 80 caracteres reservada para escribir el texto.

El *Tasword II* es un potente paquete para tratamiento de textos. El programa contiene una gran cantidad de instrucciones que permiten que el usuario formatee el texto en la pantalla. Éste se puede desplazar hacia la derecha, la izquierda o centrar, y se pueden establecer los márgenes hasta un máximo de 128 columnas. Si se vuelven a establecer los márgenes, se puede reajustar el texto de modo que quepa en el nuevo formato. El *Tasword* tiene implementado el desplazamiento de palabras, reajustando la línea de forma automática si se lleva una palabra hasta la línea siguiente. El programa es inusual en cuanto que los espacios adicionales sólo se incluyen en la segunda mitad de la línea.

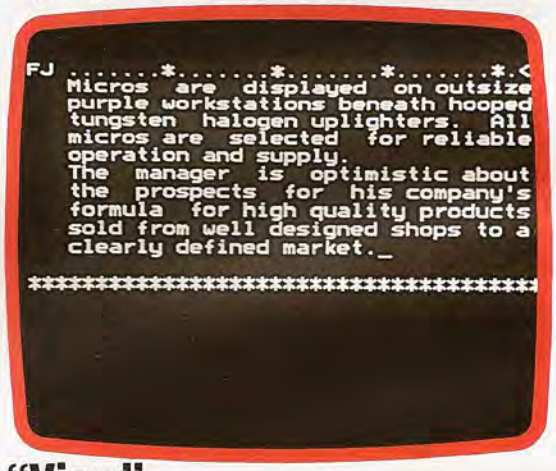
El programa también facilita instrucciones de bloques, si bien la lista de ellas no es muy amplia. Una vez definidos, los bloques se pueden desplazar, copiar o borrar, pero las instrucciones adolecen de un problema que se plantea en varias características del *Tasword II*: el programa delimita los bloques en conjuntos de líneas. De este modo, cuando el usuario define un bloque que comience a medio camino de una línea, el programa suele desplazar la línea entera, incluyendo las palabras que supuestamente habrían de quedar fuera del área del bloque.

Cuando se utiliza la modalidad insertar se plantea un problema similar. Muchos procesadores de textos insertan automáticamente el texto en la posición del cursor y desplazan el resto del texto hacia la derecha. Lamentablemente, el *Tasword* permite ya sea insertar un único carácter, o bien insertar una línea entera. Además, esta línea extra suele aparecer debajo de la actual, ignorando nuevamente el hecho de que el cursor pueda hallarse en mitad de una línea y no al final de la misma.

Donde el *Tasword* se destaca verdaderamente es en la gran cantidad de facilidades para control de impresora que ofrece. En primer lugar, los usuarios pueden adaptar a su medida, o "instalar", su propia versión de Tasword.

El *Tasword* posee una amplia gama de caracteres de control para la impresora basados en los códigos Epson estándares. A ellos se accede pulsando CTRL y la barra Space al mismo tiempo, y a continuación la letra adecuada, con las mayúsculas activando la opción y las minúsculas desactivándola. Hay 20 códigos disponibles en el programa, que van desde espaciado entre líneas hasta diversas formas de impresión resaltada, como subrayado, negritas y cursiva.

Los caracteres de control de la impresora Tasword proveen caracteres fuente adicionales, como Lectura Light y Compacta, pero los mismos no están disponibles en el programa propiamente dicho. Para poder acceder a ellos, Tasman, el fabricante, ha introducido un segundo programa que se puede ejecutar en el *Tasword* y se denomina *Tasprint*, que se puede configurar para una amplia gama de impresoras matriciales. Una vez configurado, el *Tasprint* se puede cargar antes que el *Tasword* y se asigna a sí mismo una zona de la memoria sobre la cual no escribirá el *Tasword*, tras lo cual el usuario puede acceder a las cinco fuentes adicionales. Como es natural, al hacer uso de estas fuentes adicionales se reduce la zona de textos; *Tasprint* ocupa alrededor de 5 de los 13 Kbytes que estarían disponibles en caso contrario.



"View"

El problema a superar en el caso del BBC Micro es ligeramente distinto al del Commodore 64. Su visualización en pantalla utiliza una gran cantidad de memoria que, en el Modelo B estándar, es escasa. Al igual que numerosas otras empresas, AcornSoft ha optado por suministrar su paquete para tratamiento de textos, *View*, en ROM para instalar en uno de los conectores laterales del BBC Micro. Éste no ocupa nada del espacio de memoria disponible y, al mismo tiempo, evita el problema con el que se enfrentan algunos de los sistemas grandes de tratamiento de textos, que han de leer continuamente programas extras desde disco.

Ésta es una consideración importante. Si usted decidiera contar con una visualización de 80 columnas, lo mejor que se podría esperar son algo menos de 10 000 caracteres en memoria. Si en ella se incluyera un programa de ocho Kbytes, el área de textos se reduciría considerablemente, pero con el *View* no es necesario utilizar la modalidad de 80 columnas. La pantalla se puede emplear como una ventana sobre el documento, como con el *Easy script* y la mayoría de los otros paquetes.

El *View* es sin duda uno de los mejores programas para tratamiento de textos para micros. En lugar de tener la pantalla de ayuda, se proporciona al usuario una plantilla que se coloca sobre las teclas de función, dado que a casi todas las instrucciones de tratamiento de textos se accede a través de estas teclas, ya sea individualmente o conjuntamente con las teclas Shift o Control.

Entre las facilidades adicionales que incluye el *View* está la facilidad que permite al usuario definir sus propias *macros*. Éstas son programas cortos (creados utilizando instrucciones del *View* y texto) que se pueden emplear para crear instrucciones adicionales mediante un código de dos letras definido por el usuario.

Asimismo el programa incluye instrucciones que llevan a cabo funciones tales como cuenta de palabras, formateo, proceso continuo (que permite editar un documento desde disco o cinta y después pasarlo directamente a otro archivo) y un juego completo de instrucciones Buscar/Reemplazar. Éstas no sólo realizan las funciones habituales, sino que permiten caracteres.

WordStar de bolsillo

Una muestra de la seriedad con la que en la actualidad se está considerando a la gama de ordenadores Amstrad está en el hecho de que varias empresas de software que previamente sólo fabricaban programas para aplicaciones de gestión, ahora están "adaptando" sus programas CP/M para ejecutarlos en máquinas Amstrad. Una de las jugadas maestras de la empresa de Alan Sugar es que MicroPro ha desarrollado una versión del *WordStar*, el líder de los paquetes profesionales para tratamiento de textos, para ejecutar en los ordenadores CPC 464 y 664. Aunque el *Pocket WordStar* opera con menos memoria que el paquete original, esto no significa que haya perdido muchas de las facilidades del programa madre. Por el contrario, en el *Pocket WordStar* se han implementado también casi todas las facilidades que proporciona el paquete *WordStar* completo. Las únicas que le faltan a la versión hecha a medida para Amstrad son aquellas que requieren más memoria de la que hay disponible en el ordenador; p.ej., la facilidad que permite al usuario imprimir (Print) un archivo (File) y editar (Edit), al mismo tiempo. MicroPro ha logrado conseguir esto al retener en disco varios submenús, que se cargan cuando se necesitan. Esto, por supuesto, significa que el *Pocket WordStar* es algo más lento que su predecesor, pero ésta es una pequeña contrapartida a cambio de un paquete de tratamiento de textos excelente para una máquina de costo tan económico

EASY SCRIPT

Desplazamiento de palabras

★

Easy script sólo lo soporta cuando el documento se está enviando a la impresora.

Movimiento de bloques

★★★★

Admite transferencia, copiado y archivado separado.

Ayuda en pantalla

No tiene.

Pantalla de 80 columnas

Easy script posee una pantalla de 40 columnas.

Contador de palabras

Easy script no posee facilidad para contar las palabras.

Buscar/Reemplazar

★★

Se pueden buscar series de hasta 32 caracteres, si bien la facilidad es lentísima.

WYSIWYG

★★

Aunque el *Easy script* permite examinar el trazado de la página, la baja resolución de pantalla limita su utilidad.

Facilidad para correspondencia

★★

Incorpora una facilidad Mail Merge simple, pero no permite imprimir etiquetas de direcciones.

Verificador de ortografía

★★★

Los mismos fabricantes ofrecen un paquete hermano: *Easy spell*.

Fuentes disponibles

★★★

Easy script soporta impresión agrandada, condensada, invertida y en negritas.

Unión de archivos

★★★★★

El programa soporta instrucciones para unir archivos, ya sea desde cassette o disco, para editar o imprimir.

TASWORD II

Desplazamiento de palabras

★★★★

Completo y con ajuste automático.

Movimiento de bloques

★★★

Permite funciones para transferir, copiar y suprimir, pero no archivado separado.

Ayuda en pantalla

★★★★★

Se proporciona mediante una ventana en la parte superior de la pantalla, que se puede desplazar para visualizar todas las instrucciones disponibles.

Pantalla de 80 columnas

★★★★★

El programa visualiza una pantalla completa de 80 columnas, con un valor máximo para el margen derecho de 128.

Contador de palabras

★★★★

Disponible constantemente.

Buscar/Reemplazar

★★

Tasword II sólo permite la búsqueda de una única palabra.

WYSIWYG

★★

Hay varias opciones de formato de impresión que sólo la impresora reconoce como caracteres de control; no se pueden visualizar en la pantalla.

Facilidad correspondencia

No tiene.

Verificador de ortografía

No tiene.

Fuentes disponibles

★★★★★

Tasword II implementa varias fuentes diferentes; el *Tasprint* proporciona estilos adicionales.

Unión de archivos

★

El programa sólo permite añadir un archivo en la cola de otro.

VIEW

Desplazamiento de palabras

★★★★

Soporta desplazamiento de palabras completo, aunque no hay instrucciones para desactivar esta función.

Movimiento de bloques

★★★★

Las operaciones de bloques que permite incluyen supresión, transferencia y copia.

Ayuda en pantalla

★

No ofrece ayuda en pantalla, pero proporciona una plantilla para cubrir las teclas de función.

Pantalla de 80 columnas

★★★

View soporta una pantalla de 76 columnas. Sin embargo, ésta reduce drásticamente la cantidad de memoria que queda para el texto.

Contador de palabras

★★★★

Si bien posee una facilidad para contar las palabras (que en realidad cuenta los espacios), no es una visualización constante.

Buscar/Reemplazar

★★★½

View sólo puede hallar y reemplazar individuales, si bien soporta caracteres *wildcard*

WYSIWYG

★★★★

El programa permite la visualización del documento tal como aparecerá en la página, exceptuando las facilidades resaltadas (subrayando la impresión en negrita).

Facilidad correspondencia

No tiene.

Verificador de ortografía

No tiene, pero AcornSoft ha prometido que en el futuro proporcionará uno.

Fuentes disponibles

★

El único énfasis adicional disponible son las negritas.

Unión de archivos

★★★★★

Las instrucciones Macro permiten una amplia gama de facilidades para unir archivos.

Mudar de sitio

Nos quedan por examinar cuatro modos de direccionamiento y mostrar sus diferencias con algunos ejemplos en código máquina

Para empezar hagamos un repaso de los modos que ya conocemos:

- *Direccionamiento absoluto*: En este modo el operando fuente o destino es una dirección de memoria.
- *Direccionamiento con registro*: La fuente o destino es un registro.
- *Direccionamiento indirecto con registro (y puntero)*: Apuntamos al objeto fuente o destino.
- *Direccionamiento indirecto con registro (y postincremento o predecremento)*: Recorremos una lista de datos de arriba abajo o viceversa.

Consideremos ahora el modo *inmediato*, en el que almacenamos una constante próxima a la instrucción.

- *Direccionamiento inmediato*: En este modo el operando fuente es una constante. Por ejemplo, MOVE.W #$43,D0 ordena que la constante (especificada mediante el símbolo #) $43 ($ significa hexadecimal) es llevada a D0. Se dice que es un direccionamiento inmediato porque la palabra constante se almacena en la propia instrucción MOVE. Es obvio que para operandos bytes la extensión toma un byte, y para palabras largas toma cuatro bytes.

Este modo inmediato es muy útil para establecer una constante, por ejemplo para un bucle con cuatro reiteraciones. Pero es de notar que pueden darse ocasiones en que sea mejor definir una constante en una posición determinada y emplear el direccionamiento absoluto hacia esta posición cada vez que se necesite. Por ejemplo:

```
MOVE    COUNT,D0
MOVE    COUNT,D5
COUNT   DC.W 4
```

Esto puede que se prefiera al direccionamiento en modo inmediato cada vez que se prevea que el valor COUNT ha de cambiar. Con el método absoluto cambiamos el contenido de COUNT; de otra manera tendríamos que investigar todo el código para todas las referencias en modo inmediato, empleando, por ejemplo, 4, lo cual, además de ser aburrido, puede comportar errores.

Es posible un modo inmediato *rápido* con ciertas instrucciones, donde la constante está contenida dentro de las instrucciones en una palabra. Tanto uno como el otro de estos dos ejemplos:

```
ADDQ    #3,D0
SUBQ    #1,D3
```

se codificarían en una palabra (y, por tanto, se ejecutarán más rápidamente ya que tienen la constante junto con la instrucción). Obsérvese, sin embargo, que las constantes con ADDQ y SUBQ sólo pueden estar en el intervalo #1 a #8.

La instrucción MOVE tiene también una versión rápida en la que el ámbito de las constantes se mueve entre −128 y +127. Así, MOVEQ #98,D4 pone en D4 el decimal 98.

Para reforzar su memoria, he aquí un ejemplo donde se emplean todos los modos de direccionamiento analizados hasta aquí:

```
1                LEA    OUTPUT,A1    Absoluto y registro
2                MOVE   A1,A2        Registro
3                MOVE   -(A1),D3     Predec. y registro
4                ADDQ   #3,D3        Inmediato rápido y registro
5                MOVE   D3,(A2)+     Registro y postinc.
6                ADD    #6,D3        Inmediato y registro
7                MOVE   D3,(A2)      Registro e indirecto
8
9       INPUT    DC.W   #6           Constante 6
10      OUTPUT   DS.W   2            Espacio para dos palabras
```

La posición INPUT contiene una constante de 6 y la dirección OUTPUT tiene un espacio para dos palabras.

Interesa tener en cuenta lo que está contenido en OUTPUT y OUTPUT+2 tras la ejecución de este fragmento en código máquina. Las líneas 1 y 2 hacen que A1 y 2 apunten a OUTPUT. La línea 3 hace que A1 apunte a INPUT antes de direccionar su operando, por lo que en D3 se cargará 6. La línea 4 añadirá 3 a D3, o sea el contenido de D3 es ahora 9.

En la línea 5, D3 se cargará en OUTPUT, ya que A2 apunta a OUTPUT. Una vez hecho esto, A2 se incrementa en dos para apuntar a la segunda palabra OUTPUT. La línea 6 suma 6 al contenido de D3 (contenido total, 15), que será cargado en la segunda palabra OUTPUT.

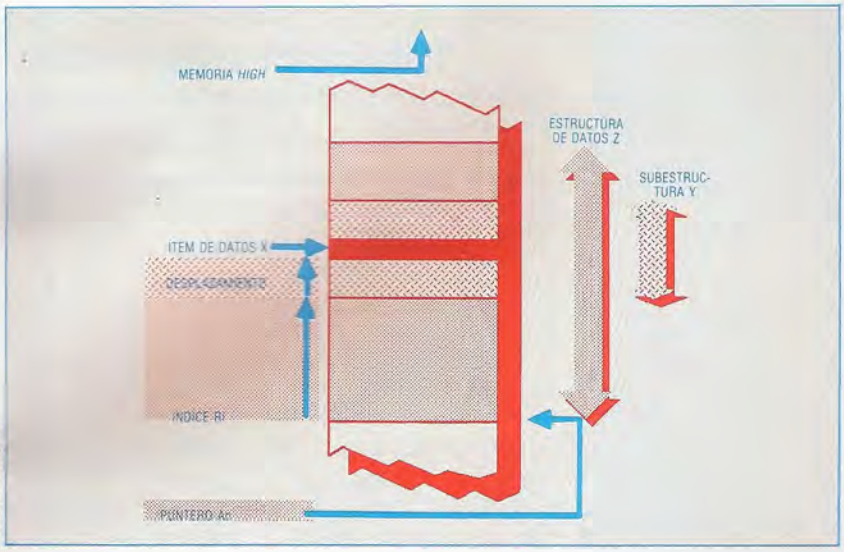
- *Direccionamiento indirecto (con desplazamiento e índice)*: Debemos considerar primero el significado que aquí se da a los términos *desplazamiento* (*displacement*) e *índice* (*index*). Con el 68000 por *desplazamiento* se entiende una "distancia" u *offset* fijo desde un punto de referencia básico; por su lado, *índice* significa un desplazamiento variable a través de un registro.

La diferencia entre estos dos términos queda ilustrada en el dibujo (página contigua). En él se muestran:

1) un componente de datos estructurados llamado Z, al que apunta el registro de direcciones An;
2) una subestructura interna llamada Y, indexada mediante un registro índice llamado Ri (registro de direcciones o de datos);
3) y un elemento de Y llamado X, al que se puede acceder con un desplazamiento fijo.

Ejemplos de estas estructuras de datos:

Saberse el camino
La *Guía del usuario* del 68000 contiene breves detalles de las instrucciones y modos del 68000, pero donde resulta más útil es en las sugerencias sobre cómo sacar provecho de las facilidades de éste en numerosas aplicaciones de ejemplo. El libro ha sido publicado por la editorial británica Sigma Press



Caroline Clayton

1) Z: tabla de registros, o tabla de dimensiones;
2) Y: el registro mismo, o bien una fila de una tabla;
3) X: elemento de un registro, o bien el entero de una tabla.

Se trata de tres componentes de datos estructurados que son conocidos en los modernos lenguajes de alto nivel tipo PASCAL y ADA, y, por tanto, es importante poder referirnos fácilmente a tales componentes. Naturalmente, no hay razón alguna por la que el programa ensamblador no pueda estructurar los datos de modo similar: ¡sin duda disponemos de medios para hacer esto con el 68000!

Volvamos una vez más a nuestro ejemplo de datos estructurados. Habrá notado que la dirección de X está formada por la suma An+Ri+desplazamiento, y que podemos alterar An y Ri conforme se ejecuta el programa. Podemos mirar ahora algunos ejemplos de direccionamiento, y comenzaremos con uno que lo ilustra cabalmente en su forma más sencilla (direccionamiento indirecto con desplazamiento):

```
MOVE.W DISP (A0),D1
```

donde DISP habrá sido definido previamente como nombre simbólico, supongamos, de 6. Esta instrucción emplea el direccionamiento indirecto con un desplazamiento de 6 como modo de direccionamiento fuente. Si, por ejemplo, A0 estuviera apuntando a la dirección $1000 (1000 hexa), entonces el contenido de la dirección $1006 se cargaría en D1. Nótese que al aceptarse una extensión de palabra entera con la instrucción para el desplazamiento, podemos tener un entero con signo de 16 bits completos para desplazamiento (valores entre +32767 y −32768).

Veamos ahora la modalidad de direccionamiento más poderosa que tiene el programador del 68000 (el desplazamiento), considerada en este ejemplo:

```
MOVE.W DISP (A0,D0.W),D1
```

En este caso, la dirección fuente está formada añadiendo juntamente el registro base, A0, con el registro índice, D0, y el desplazamiento fijo, DISP. Es de notar, sin embargo, que en este caso DISP sólo puede ser un entero con signo de ocho bits, aunque el registro índice puede ser una palabra larga de 32 bits completos.

Obsérvese que cuando usted compara estos dos modos de direccionamiento (el indirecto y el desplazamiento, con o sin índice), el desplazamiento siempre es fijo en tiempo de ensamblaje. Sin embargo, si usted necesita alterar dinámicamente un desplazamiento, entonces podrá optar siempre por usar:

```
MOVE.W 0(A0,D1),D2
```

donde D1 se convierte ahora en el desplazamiento de hecho (que podemos alterar en tiempo de ejecución del programa), y el desplazamiento fijo es cero.

Finalmente es oportuno señalar que el conjunto de instrucciones del 68000 ofrece estos dos potentes modos de direccionamiento para la mayoría de las instrucciones más comunes. Veamos un ejemplo de direccionamiento que emplea desplazamientos e índices, con bytes como atributos de datos.

```
1        LEA      LIST,A1
2        MOVEQ    #4,D1          D1:=4
3        MOVE.B   2(A1),D6       D6:=3
4        ADD.B    0(A1,D1),D6    D6:=8
5        MOVE.B   D6,4(A1,D1)    LIST+8:=8
6
7  LIST  DC.B     1,2,3,4,5,6,7,8,0
```

A1 es puesto de modo que apunte a LIST en la línea 1 y seguidamente D1 es puesto a 4. En la línea 3, al puntero que hay en A1 se añade un desplazamiento de 2, de forma que el tercer elemento de LIST es cargado en D6. Después, con un índice de 4 en D1, se añade a D6 el contenido de elemento 5, siendo copiado en LIST+8.

Obsérvese que en la línea 4 hemos empleado el desplazamiento de 0: esto permite emplear D1 como registro índice, que podemos alterar cuando se ejecuta el programa, si fuera el caso.

• *Direccionamiento relativo al PC*: Antes de pasar a analizar en detalle los modos de direccionamiento relativo al PC debemos echar un vistazo a algunas directivas del ensamblador. Las directivas son instrucciones dadas al ensamblador que no producen directamente código ejecutable, pero que influyen en factores tales como el formato del listado del programa fuente o la definición de símbolos. Una de estas directivas se refiere específicamente a la dirección de inicio del programa y el tipo de código producido. Es la directiva ORG, o sea origen.

Normalmente se especificará la dirección de inicio, por ejemplo así:

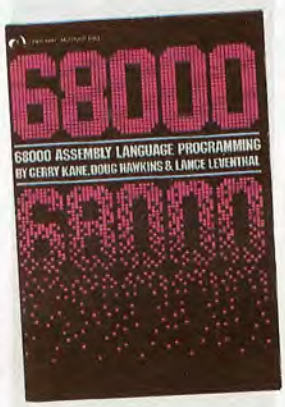
```
ORG    $1000
```

Con ello establecemos la dirección de inicio en $1000 (en realidad es la dirección de carga para el cargador binario), y todo el código que sigue es cargado en direcciones secuencialmente crecientes. Si aparece una nueva directiva ORG, establecerá entonces una nueva dirección de carga en ese punto. Una variante de esto sería:

```
ORG.L    $2000
```

donde todas las referencias posteriores se tomarán como palabras largas completas, y por ende cualquiera de estas referencias ocupará dos palabras enteras.

La directiva RORG define una dirección de carga



Enseñanza general
Este libro sobre programación en lenguaje assembly del 68000 es una guía completa sobre el tema publicado por la editorial británica Osborne/McGraw-Hill. Los programadores experimentados encontrarán algo superfluas las secciones sobre teoría de la programación y el estilo del libro es enfáticamente magistral. Pero si por un lado le falta informalidad por otro incluye toda la información que usted pueda necesitar. Un buen detalle es el empleo de encabezamientos de párrafo en mayúsculas, lo que permite buscar rápidamente un tema concreto a través de las secciones

desde la cual todas las referencias de memoria se consideran relativas al contador del programa (PC), dejando de ser absolutas. El principio de este tipo de código se basa en que, independientemente de donde se cargue el código, las referencias de memoria corresponden a esa dirección de carga. Pongamos por ejemplo el siguiente listado:

```
                        RORG    $2000
2000  D47A      START   ADD     CONST1,D2
2002  000A
2004  3202              MOVE    D2,D1
2006  6000              BRA     START
2008  FFFB
200A  4E40              TRAP    #0
200C  0026      CONST1  DC.W    38
```

Aquí, CONST1 está referenciado con un desplazamiento de 16 bits de A hexa (10 decimal) en la posición 2002, que se añade al PC (contador del programa). Para cuando se cargue el desplazamiento en el ejemplo, el PC será 2002 de modo que la dirección de datos fuente será 200C. Obsérvese que no hay instrucciones específicas o diferencias en la sintaxis de direccionamiento para obtener el código relativo al PC; todo lo que se requiere es la directiva RORG del ensamblador.

Hay unos cuantos puntos más sobre este código dignos de nota. Primero, que la instrucción BRA (*branch always*: bifurcar siempre) tiene un desplazamiento asociado con ella en la dirección 2008, el cual, una vez añadido al PC, dará la dirección de operando START. Esto significa que la instrucción BRA dará siempre el código relativo al PC. El segundo punto a notar es la instrucción TRAP. Se usa en este contexto como una instrucción de parada, con el añadido de que la entrada se hace a una pantalla que imprime los registros y permite al usuario el examen de la memoria. Esta instrucción, si su pantalla se lo permite, puede resultar muy útil.

Veamos otro ejemplo, donde el direccionamiento de modo relativo al PC da un desplazamiento negativo:

```
                        RORG    $2000
2000  0026      CONST1  DC.W    38
2002  DA7A      START   ADD     CONST1,D2
2004  FFFC
2006  3202              MOVE    D2,D1
2008  6000              BRA     START
200A  FFF8
200C  4E40              TRAP    #0
```

Aquí, la referencia a CONST1 es un desplazamiento negativo en la dirección #2004. (Naturalmente, usted conocerá que FFFC es negativo porque el bit de signo estará activado. La magnitud, o tamaño, de los números pueden encontrarse invirtiéndolos y añadiendo uno; por tanto, FFFC es el decimal−6).

Una limitación a este direccionamiento es que solamente se permite como operando fuente. Por ejemplo:

```
                RORG    $2000
TOTAL           DC.W    0
START           ADD     D2,TOTAL
```

generará un error de ensamblador. Esto significa que este tipo de direccionamiento es muy conveniente para el código que ha de ponerse en una ROM (dado que no podemos escribir en las ROM) en una dirección fija, pero todavía puede ser comprobada en cualquier posición de RAM conveniente (donde, como sabemos, podemos escribir y leer).

Puede que esté pensando en que es una restricción innecesaria el no poder escribir en la memoria con este modo de direccionamiento. Sin embargo, observe que usted puede escribir en cualquier posición absoluta refiriéndose sencillamente a esa posición. Por ejemplo:

```
                ORG     $1000
TOTAL           DS.W    0
                RORG    $2000
CONST1          DC.W    38
START           ADD     CONST1,D2
                MOVE    D2,TOTAL
```

donde la referencia a TOTAL se acepta porque está en un área de dirección absoluta.

El modo relativo al PC puede también empliarse de modo que sea posible el uso de un índice con el desplazamiento.

Por ejemplo:

```
                RORG    $3000
INDEX           DC.W    10
START           MOVEA   INDEX,A5
                ADD     6(A5),D2
```

En este caso tenemos que usar la instrucción MOVEA, que carga A5 con el contenido de la posición de memoria INDEX. No podríamos usar LEA porque tomaría la dirección de INDEX (de ningún modo permisible en el modo relativo al PC). Es claro que una instrucción MOVE directa sería ilegal, como ya hemos visto (un registro de dirección no es legal como operando destino).

Volvamos ahora al ejemplo donde el operando fuente de la instrucción ADD será el valor de PC después de la instrucción ADD más el registro índice (aquí, A5) con un desplazamiento de 6. En este ejemplo, la dirección fuente será 16 bytes más allá de la dirección que contiene la palabra de extensión 6.

Una limitación de este modo es, sin embargo, el que el desplazamiento esté formado por ocho bits dentro del opcode, limitando nuestro desplazamiento a un número de bytes desde +127 a −128.

La importancia de este modo relativo al PC es que el código puede ejecutarse en cualquier sitio de la memoria, como hemos visto en el caso de escribir código para la ROM. Obsérvese, sin embargo, que esta forma de direccionamiento es extremadamente útil, por lo general, para escribir código independiente de la posición en memoria. Esta forma de código puede ser necesaria para extensos programas de muchos módulos donde la posición de un módulo en la memoria no es fijada hasta que el módulo se carga en la memoria. Naturalmente, para largos sistemas de multiprogramación, puede que necesitemos una unidad de gestión de la memoria con facilidades adicionales (que proporciona Motorola), pero para cualquier esquema de memoria más sencillo el modo relativo al PC es muy importante.

- *Direccionamiento implícito*: Este modo de direccionamiento no debería ser dificultoso, dado que el opcode especifica los registros que hay que usar. Por ejemplo, la instrucción RTS afectará al PC y al puntero de la pila; BRA afecta al PC.

Transmisión visual

Los sistemas de videotex envían imágenes de alta calidad a los ordenadores a través de las líneas telefónicas o las ondas

La mayoría de los usuarios de ordenadores personales se han acostumbrado a un nivel de definición en los gráficos muy inferior al de la calidad fotográfica. No obstante, un estándar de imagen más elevado es sólo una más de las características técnicas de lo que ha llegado a conocerse como *videotex*. Suscita confusión el hecho de que la palabra haya adquirido dos significados relacionados entre sí pero diferentes, según se la utilice en Gran Bretaña o en Estados Unidos. El primero cubre información visual y textual distribuida ya sea por un canal de televisión o por cable. En Estados Unidos, el término *videotex* se aplica a una combinación de texto y video transmitida exclusivamente por cable.

El inconveniente de todo sistema de videotex de gran calidad es la velocidad a la cual aparecen las imágenes en la pantalla. Mientras que una página de videotex típica del servicio Prestel de la British Telecom suele demorar menos de un segundo en llenar la pantalla, en otros sistemas puede tardar 10 segundos o más. La razón estriba en la cantidad de datos que es necesario transmitir para que aparezca una única pantalla. Sería muy ilustrativo comparar los sistemas Prestel y Photo Videotex de la British Telecom.

Las definiciones de pantalla se cuantifican en pixels. Una página de Prestel suele ser de 234 pixels por 22 líneas. Por el contrario, una pantalla de Photo Videotex tiene una definición de 270 pixels por 240 líneas, pero en contrapartida por esta mayor definición es preciso satisfacer unas exigencias de datos muchísimo mayores para almacenar tal imagen. (El Photo Videotex requiere 128 Kbytes, mientras que una página de Prestel necesita 5 Kbytes como máximo.)

La compresión de datos está jugando un papel cada vez más importante en la tarea de rebajar las exigencias de almacenamiento de datos. El Photo Videotex, por ejemplo, es capaz de comprimir unas exigencias teóricas de 128 Kbytes en 64 Kbytes reales. Con el tiempo, los sistemas "inteligentes" podrán muestrear la pantalla y enviar datos sólo a las partes de la imagen que hayan cambiado. Así se reducirá drásticamente la cantidad resultante de datos que sea necesario enviar.

El Videotex se puede utilizar para cualquier aplicación en la que se necesite combinar texto con ilustraciones en color. Por ejemplo, en los sistemas de seguridad, que son los principales usuarios, el videotex ayuda a emparejar rostros con nombres. Con esta disposición, un guarda puede consultar los registros de monitor retenidos en una base de datos central. Al digitar un nombre, aparecerá un registro conteniendo una foto en color de la persona más cierta cantidad de información de carácter personal. En el caso de que el guarda no estuviera seguro de que la persona corresponda a la fotografía, es una tarea sencilla efectuar una doble comprobación y pedir a la persona en cuestión la fecha de nacimiento, el número del documento nacional de identidad, etc.

Asimismo, el videotex es un importante medio publicitario. Los agentes de viajes y los agentes inmobiliarios ya se valen de sistemas de videotex para poner en contacto a compradores y vendedores. Futuros desarrollos de este enfoque podrían conducir a que se colocaran pantallas en los escaparates de las tiendas y se los hiciera pasar por fotogramas retenidos en una base de datos. Los viandantes, por lo tanto, podrían ver detalles de las últimas ofertas o de las fincas que se encuentran en venta. Puesto que el videotex es interactivo, también se podrían transmitir las decisiones de compra. Los agentes de viajes, por ejemplo, podrían realizar reservas con las agencias. Si se estuviera utilizando una línea telefónica o una red privada, las dos firmas no necesitarían siquiera hallarse en el mismo país.

En Gran Bretaña, por intermedio de un servicio independiente llamado Armchair Grocer (tendero de sillón) y a través de un cable de televisión, se

Crispin Thomas

Un universo ante sus ojos La tecnología del videotex permite que los usuarios transmitan imágenes fotográficas, textos y gráficos por ordenador a terminales remotos, generando, en consecuencia, muchas aplicaciones diferentes en el hogar y en el comercio. Las noticias locales, la publicidad clasificada y el punto de información de ventas se pueden procesar y enlazar a sistemas interactivos para facilitar la realimentación del usuario. Por ejemplo, podrían llegar a ser de uso corriente gestiones tan disímiles como realizar las reservas para unas vacaciones en el extranjero y —en el futuro— comentar y contemplar las propuestas de planificación local

pueden hacer compras con el Prestel (Club 403). El videotex, incluso, permite a los usuarios curiosear en los catálogos comerciales en la intimidad de sus hogares. Las compras se pueden efectuar dando un número de tarjeta de crédito o bien mediante transferencia de fondos electrónica directa.

Las fronteras del videotex ya se están ensanchando para incluir las "videoconferencias". Una empresa norteamericana, Datapoint, ha producido un dispositivo denominado Minx, que contiene una cámara de video, un teléfono con altavoz, una pantalla en color y otra de referencia mucho más pequeña. Utilizando una red de área local (Arcnet) o línea de datos, las dos partes pueden llamarse y ver a la persona con quien están hablando. El Minx posee facilidades añadidas, puesto que también es compatible con el IBM PC. La línea que está portando la llamada de voz tomará simultáneamente las señales de video y de datos. En consecuencia, ambas partes pueden ver visualizados los mismos datos durante la conversación, contemplándose aun a sí mismos en las pantallas de referencia más pequeñas. De la misma manera se pueden intercambiar archivos. Tal sistema exige un enlace de datos de dos megabytes por segundo.

Inicialmente, los servicios de videotex requerían un ordenador central o un miniordenador para almacenar los datos y gestionar las llamadas. Esta situación está experimentando un acelerado cambio gracias a los micros. Por ejemplo, a pequeña escala, en Gran Bretaña —país pionero en el campo de la informática—, la firma Soft Machinery está utilizando el BBC Micro para operar un servicio denominado The Gnome at Home (El gnomo en casa). En efecto, al unir varios micros entre sí mediante una red Econet, se hace posible que hasta ocho usuarios llamen simultáneamente. El software es una versión modificada de Communitel, un paquete desarrollado por Notting Dale ITEC como anfitrión de videotex para el BBC Micro. En muchos sentidos, Communitel es similar a un tablón de anuncios, si bien es compatible con el Prestel.

Apricot ha producido el Apricot Viewdata, que proporciona un sistema anfitrión de videotex privado para entre 8 y 16 usuarios simultáneos. Utilizando puertas RS232, se accede a la opción a través de un conjunto de modems o una red de área local. Al ser compatible con Prestel, se lo puede conectar al sistema Prestel propiamente dicho a través de un *gateway*. (*Gateway* es el término genérico para designar un enlace entre dos ordenadores en un sistema de videotex.)

El sistema Photo Videotex de British Telecom también utiliza un microordenador (el IBM PC) que combina una entrada de imágenes en color o en blanco y negro proveniente de una cámara PAL o RGB con texto estándar. La calidad de la imagen es comparable a la de una fotografía en color.

El videotex puede aportar soluciones a una gran variedad de problemas. En la medida en que se vaya extendiendo el uso de las redes de área local y los enlaces telefónicos digitales —por fibra óptica y por satélite—, los terminales de videotex se convertirán en una presencia familiar en la industria, en el comercio e incluso en el hogar.

Videotex y teletexto

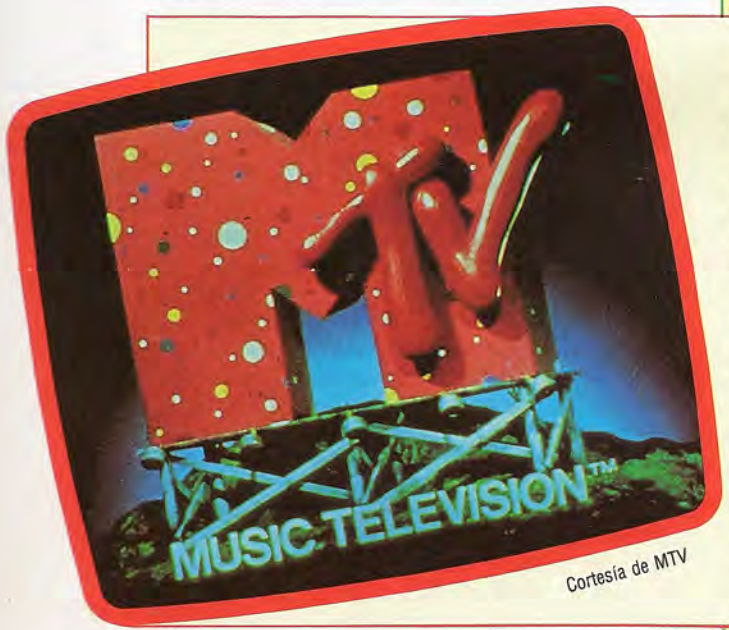
Para distinguir los dos tipos de videotex que por lo general se utilizan en Gran Bretaña se han acuñado los términos *videotex* y *teletexto*. El videotex se basa en el trabajo que realizó Sam Fedida en lo que se ha convertido en el Centro de Investigación de British Telecom en Martlesham Heath. Como resultado de sus propuestas, en 1979 nació el primer sistema de videotex del mundo disponible comercialmente, en la forma del Prestel. La forma de videotex del Prestel ha obtenido un gran éxito comercial, permitiéndole penetrar en un mercado tan marcadamente competitivo como el de Estados Unidos. Sin embargo, el Prestel no es de ningún modo el único servicio de videotex en Gran Bretaña. Existen servicios de videotex privados; ADP e Istel (subsidiaria de British Leyland) poseen sus propios sistemas accesibles por teléfono.

Si bien el videotex se basa fundamentalmente en las líneas telefónicas, las transmisiones de teletexto se adaptan muy bien a la señal de televisión estándar. Nuevamente, el resultado es una combinación de texto y gráficos. Para poder

Cortesía de British Telecom

Imágenes cableadas

Si bien la televisión por cable es un medio ideal para las transmisiones de videotex, en Gran Bretaña es un competidor de poca altura. Los informes de 1985 señalaban que sólo 127 000 familias recibían televisión por cable. Cuando las jóvenes empresas de cable como Westminster Cable estén totalmente en actividad, esta cifra se aproximará a 300 000. En 1984 se llevó a cabo un experimento en transmisión de software por cable. Llamado Proyecto Gamestar, fue una empresa conjunta entre British Telecom y Micronet 800. Los usuarios pudieron alquilar un Spectrum y un modem Prism modificado para cargar desde una línea software de juegos. Fue secundado por algunas empresas de cable incluyendo a Rediffusion, pero el proyecto fue abandonado debido a dificultades técnicas. Hoy British Telecom se está concentrando en el Cabletext, un derivado de Prestel para empresas de cable



Cortesía de MTV

Pase privado
El sistema Photo Videotex de la British Telecom utiliza un IBM PC y proporciona amplias facilidades de edición, permitiendo cortar las imágenes, desplazarlas y aplicarles *zoom*, y combinarlas con superposiciones, texto y gráficos mejorados. Entre las aplicaciones se incluyen noticias locales, publicidad, promociones de puntos de venta, registros de personal y sistemas caseros para grandes empresas. El sistema enlaza terminales de usuarios, terminales de edición y un sistema de ordenador central, permitiendo la comunicación en dos sentidos y una respuesta directa. La calidad de la imagen es comparable a la de un televisor doméstico y cada imagen en color de tamaño completo requiere 64 Kbytes de almacenamiento

recibir teletexto se necesitan aparatos de televisión modificados especialmente. El teletexto emplea páginas tal como su equivalente de videotex, pero, a diferencia de éste, no es del todo interactivo y no puede transmitir las respuestas del usuario.

En algunas ocasiones se alude al teletexto como "videotex emitido" (no se debe confundir el término con "teletex", que es una forma mejorada de telex). La cooperación técnica entre la British Telecom y la BBC/IBA ha dado lugar a un grado de compatibilidad entre los dos sistemas. En consecuencia, se ha aplicado el término "*world system*" (sistema mundial) a todo cuanto se ajuste a los estándares británicos Prestel u Oracle/Ceefax.

Tradicionalmente, los fabricantes norteamericanos han preferido un grado de resolución de imagen mayor que el empleado por el Prestel. Es así como sus sistemas de videotex mezclan imágenes de video generadas por una cámara con el texto de un ordenador. Norteamérica no cuenta con un equivalente de teletexto. Por razones de claridad, aquí se alude a los sistemas que utilizan televisión como *teletexto*, mientras que *videotex* se empleará para cualquier tipo de sistema que emplee cable o líneas telefónicas

El videotex británico

El tipo de cable utilizado por un sistema de videotex puede ser de muy variadas formas. Un sistema de videotex podría formar parte de una red de área local, por ejemplo. El cable que realiza la conexión física puede ser un par arrollado de hilos (apenas dos cables), un cable coaxial (como una antena de televisión) o un cable de varios hilos (como en el cable RS232 utilizado para impresoras y modems).

Alternativamente, se puede utilizar a modo de cable una línea de la British Telecom. La línea telefónica normal que se utiliza para una llamada de voz cotidiana forma parte de la Public Switched Telephone Network. Una línea PSTN puede unir el extremo receptor directamente con la fuente de videotex, como en el caso del servicio Prestel de la British Telecom.

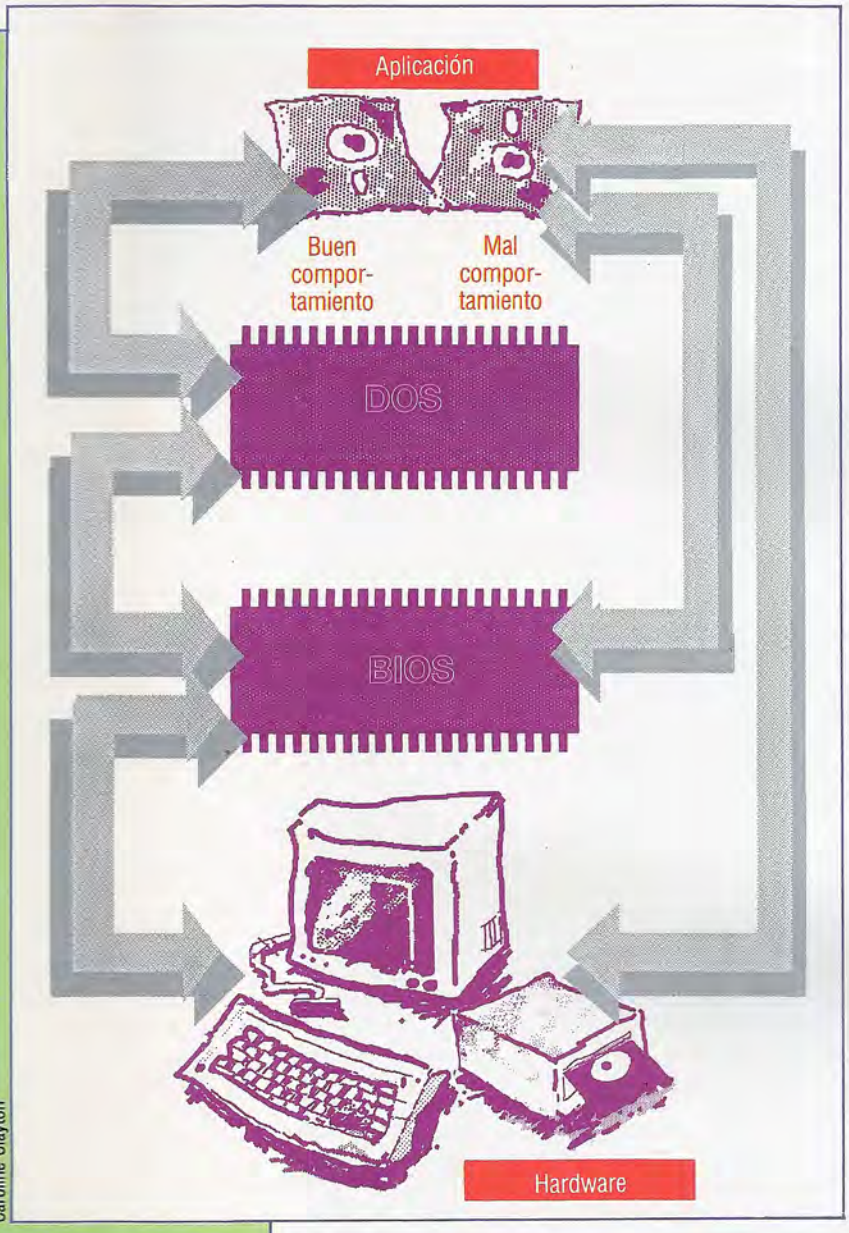
El usuario del micro puede acceder al Packet SwitchStream (PSS) a través de la PSTN. Utilizando un modem se efectúa una llamada local a lo que se conoce como un "nudo". El nudo toma los datos y los convierte en *packets* (paquetes; en realidad, lotes de datos autocontenidos) bajo un protocolo denominado X25. El nudo posee sus propios enlaces de datos directos con otros nudos diseminados a través del país. El servicio cruza incluso las fronteras y está enlazado con servicios similares de al menos otros 20 países. El servicio se denomina International Packet SwitchStream (IPSS). La conmutación de paquetes puede hacer frente a velocidades de transmisión de datos de hasta 48 Kbits por segundo, y es tan bueno como el enlace con el nudo local: si la línea es ruidosa, se necesita comprobación de errores y disminuyen las velocidades de transmisión.

Por este motivo es preferible utilizar una línea privada, exclusiva, que transmitirá los datos con mucha más limpieza. Las líneas contratadas privadas conforman los servicios "*X-stream*" de la British Telecom, tales como MultiStream, KiloStream, MegaStream e incluso SatStream (un enlace por satélite). El rival de British Telecom, Mercury, también puede proporcionar líneas de datos exclusivas.

Tales servicios son en realidad medidas provisionales hasta que la totalidad del país se convierta en la red de intercambio telefónico digital de la British Telecom, el System X, para finales de esta década. La capacidad del System X para conmutar datos además de llamadas de voz hará del videotex una fácil opción.

En 1985, un sistema de videotex que produjera un enlace de calidad fotográfica requería una velocidad de transmisión de datos de entre 64 Kbits y dos Mbytes por segundo

Portador estándar



Caroline Clayton

Buena conducta

Se dice que el software que hace uso de llamadas MS-DOS estándares es de "buen comportamiento". En este caso, el DOS llama a la rutina pertinente BIOS indirectamente a través de una "tabla de salto". En el diagrama se ilustra esta ruta de "buen comportamiento". Se considera que los programas que presuponen el conocimiento de direcciones absolutas en BIOS, o que se saltan al MS-DOS completamente al "hablar" directamente al hardware subyacente, tienen un "mal comportamiento". En el IBM PC en particular, el BIOS es un chip de ROM patentado y la tabla de salto comienza en la dirección 400H. Ésta está situada 512 bytes (dos páginas hexadecimales) por debajo de la base de tablas MS-DOS estándar, en 600H. Por consiguiente, un programa que llame al BIOS o al hardware directamente, sólo se ejecutará en una máquina IBM o un clono 100% compatible

Luego de estudiar el CP/M de Digital Research, estándar de la industria de 8 bits, examinemos el MS-DOS de Microsoft, que ha surgido como el estándar de 16 bits

En los primeros días de los microordenadores de ocho bits, la industria estaba dominada por máquinas que utilizaban el procesador Zilog Z80 y CP/M. Este sistema operativo de disco lo había escrito originalmente Gary Kildall para el chip Intel 8080; posteriormente Kildall crearía Digital Research.

Para cuando IBM decidió entrar en el mercado del micro, los sistemas de ocho bits se estaban acercando al fin de su vida útil; pero aún no había surgido una combinación definitiva de CPU y sistema operativo como alternativa de 16 bits para el Z80 y el CP/M. Por consiguiente, IBM tuvo la opción de introducir una configuración de hardware y software completamente nueva para el mercado de 16 bits. Decidieron no hacerlo por diversos motivos:

1. A pesar de dominar el mundo de los ordenadores centrales, IBM no tenía experiencia en el mercado de los micros, que era radicalmente diferente.
2. El éxito de DEC (Digital Equipment Corporation) en el campo del miniordenador le había demostrado a IBM que los usuarios ya no estaban predispuestos a adquirir productos de IBM sin una evaluación seria del producto.
3. La cantidad de tiempo requerida para desarrollar una base de software adecuada para cualquier sistema nuevo hubiera sido inaceptable en el mundo del micro, en el cual los cambios se suceden con gran rapidez, a menos que IBM obtuviera el apoyo de casas de software independientes.
4. Los costos de desarrollo tanto del software como del hardware habrían significado unos precios de venta al público inaceptablemente elevados.

En consecuencia, se tomó la decisión de adoptar lo "mejor" de lo que hubiera disponible entonces (es decir, todo lo que predominara en el mercado) y adaptarlo en la medida de lo necesario para sistemas de 16 bits.

Este enfoque tan poco innovador dio como resultado el IBM PC, máquina conservadora en cuanto a diseño, lenta en ejecución y que no abre ninguna brecha, especialmente si se la compara con sus contemporáneas, el Apple Lisa, por ejemplo. Incluso la elección del procesador Intel 8088 y un compromiso de futuros desarrollos centrados en las series de chips 8086/186/286/386 (conocidos generalmente como la familia 8086) fue una decisión pragmática. Si bien el 8088 tenía una arquitectura interna de 16 bits, el bus de datos externo era sólo de ocho bits de anchura. Esto supone que cada "palabra" de 16 bits se busca y trae en dos operaciones, reduciendo, por tanto, la velocidad de proceso, pero permitiendo el uso de chips periféricos de ocho bits existentes ya probados (y baratos).

Para el sistema operativo y el software de aplicaciones IBM acudió a Microsoft, empresa que había ganado prestigio en la microinformática fundamentalmente por su difundida versión de BASIC. En aquel entonces, Microsoft estaba en contacto con una firma denominada Seattle Computer Products, que estaba trabajando con sistemas de 16 bits basados en el Intel. En ausencia de un CP/M para la familia 8086, un programador llamado Tim Patterson había escrito un DOS en lenguaje máquina (denominado SCP-DOS) que había modelado siguiendo muy de cerca el CP/M. Su sistema también se conoció como QDOS (no confundir con el OS de Sinclair para el QL), y estas siglas respondían a la frase "*quick and dirty operating system*" (sistema operativo rápido y sucio). Esa descripción se le adecuaba a la perfección; el OS de Patterson era utilizable, pero no estaba totalmente desarrollado

ni tampoco depurado. Microsoft adquirió los derechos para el sistema y les concedió licencia de uso a los fabricantes de equipos originales (*original equipment manufacturers*: OEMs).

PC-DOS

IBM produjo su propia versión (versión 1) del MS-DOS para el IBM PC, denominada PC-DOS, y desde entonces ha estado al día con los principales desarrollos en ulteriores versiones del MS-DOS; a partir de este momento, siempre que en esta serie mencionemos una versión MS-DOS, la referencia será igualmente aplicable al PC-DOS. Pero veamos primero algunas de las diferencias entre ambos. Por cuanto concierne al usuario, esencialmente no existe ninguna. Algunos detalles internos difieren, y varían algunos detalles irrelevantes tales como la numeración de las unidades de disco; pero eso es todo, al menos en teoría. Lamentablemente, sin embargo, se han escrito muchos programas de aplicaciones para el IBM PC que direccionan el hardware directamente, en lugar de utilizar las llamadas al sistema operativo proporcionadas. Fundamentalmente por este motivo surgen problemas de compatibilidad (no es un problema inherente a las diferentes versiones de MS-DOS y PC-DOS). A los programas que llaman al DOS correctamente se los suele describir como "de buen comportamiento", y tales programas se suelen ejecutar sin modificación alguna en cualquier máquina que posea ya sea MS-DOS o bien PC-DOS de la misma versión o de versiones *recientes*. Naturalmente, no se puede esperar que las operaciones que hacen uso de las flamantes mejoras al DOS en, por ejemplo, la versión 3.1, funcionen a la perfección cuando se las ejecute en una versión anterior que no soporte esas determinadas facilidades.

La primera versión del MS-DOS era esencialmente una versión de CP/M recodificada para la familia de 16 bits de Intel. Si usted se remite a nuestra serie dedicada al CP/M, reconocerá muchas de las instrucciones y observará las semejanzas de estructura interna. No obstante, Tim Patterson sí cambió varios detalles y facilitó la labor del usuario en muchas instrucciones.

Cuando Microsoft comenzó a trabajar en las mejoras, tuvo en mente el futuro uso del Unix (el sistema operativo multiusuario de los Laboratorios Bell) y, en particular, la versión Microsoft, Xenix. Se pretendía un cierto grado de compatibilidad entre los dos, con la implementación de llamadas comunes al sistema. Pero, con mucho, el adelanto más importante en DOS 2 desde el punto de vista del usuario fue la introducción de la "redirección" de E/S, *pipelines* y "directorios jerárquicos", todos los cuales fueron ideas tomadas del Unix. En realidad, podemos definir al DOS 2 como "el DOS 1 más un poco de Unix".

DOS 3

La versión más reciente de MS-DOS (DOS 3) incorpora todas las facilidades del DOS 2 con la adición de soporte multiusuario. De las dos versiones subsidiarias principales (3.0 y 3.1), los OEMs que proporcionan sistemas para conexión en red, como Apricot y Research Machines, están utilizando la versión 3.1 con Microsoft Networks incorporado.

Clónico

Los primeros problemas de compatibilidad surgieron al desplazar IBM una "tabla de salto" dos páginas (hexadecimales) hacia abajo en la memoria y colocarla en una ROM en 400H (la tabla MS-DOS empieza en 600H). Esta parte del BIOS (sistema básico de entrada/salida) está sujeta al *copyright* de IBM, y plantea problemas para otros fabricantes que desean construir máquinas 100% compatibles con IBM ("clonos"). Sin embargo, si todo el acceso al hardware subyacente se realiza a través del MS-DOS por medio de las llamadas al sistema correctas, no existe ningún problema para producir software que se pueda ejecutar en cualquier ordenador MS-DOS.

Hay, no obstante, un pequeño precio a pagar por ello. Las llamadas al sistema le añaden una ligera penalización en tiempo a la velocidad de ejecución y, lo que quizá resulte más significativo, puede ser que no se comprendan las ventajas resultantes de unas especificaciones de hardware más avanzadas. Por estos motivos, varias casas de software han escrito programas dependientes de hardware (o firmware) que necesitan un entorno de hardware que se corresponda más íntimamente con el del IBM PC. Tampoco es un hecho desconocido el que los vendedores de software hagan deliberadamente que sus productos no sean portables. Digital Research, por ejemplo, altera unos pocos bytes de código para hacer que sea necesario comprar versiones diferentes (de aplicaciones GEM, p. ej.) para cada máquina, aun para el más compatible de los clonos

Esto, sin embargo, puede hacer que el sistema operativo sea innecesariamente grande, de modo que se dispone de la DOS 3.0 para otros entornos, privada de la capacidad para conexión en red de la 3.1, pero conservando las facilidades para compartir archivos y proteger registros. La solución por software proporciona una respuesta práctica y eficaz a los problemas de ejecutar aplicaciones que necesitan permitir el acceso a bases de datos comunes.

El último procesador de Intel (80286), utilizado en el IBM PC-AT, posee tanto una capacidad de direccionamiento de memoria de tres Mbytes como la facilidad, incorporada en el hardware, de proteger regiones de archivos. El DOS 3.0 permite compartir archivos en el 8086 y el 80186 mediante llamadas adicionales al sistema que pueden prohibir temporalmente el acceso a una región. Esto impide la "colisión de acceso", pero a expensas de la velocidad de proceso (por el tiempo extra que requieren las llamadas al sistema). Se han introducido otras mejoras y, si bien aún pueden hallarse uno o dos vestigios del DOS "rápido y sucio" de Seattle, el MS-DOS se puede considerar como un producto depurado con un futuro por delante.

Entre las mejoras más importantes que hay disponibles en la actualidad está Microsoft Windows. Al igual que el GEM de DR, proporciona un entorno tipo WIMP junto con facilidades para multitareas y operaciones de "cortar y pegar" entre aplicaciones. Se está implementando como una ampliación al sistema operativo MS-DOS/PC-DOS básico, y está diseñado para proclamar una nueva era en la sencillez de uso de ordenadores convencionales.

Chris Stevens

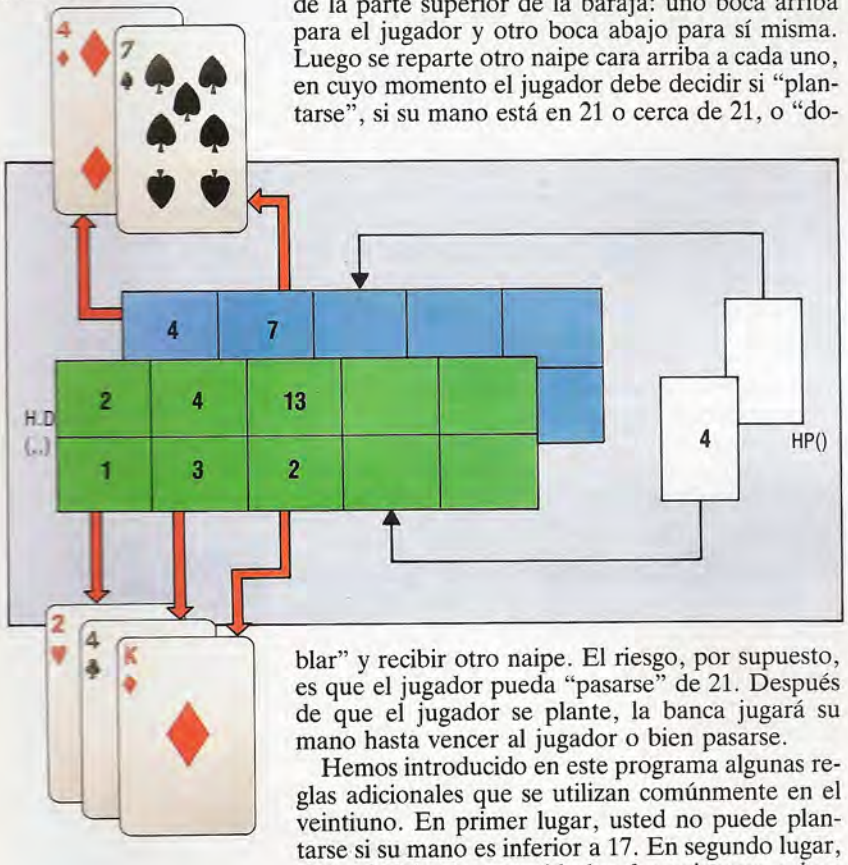
Ostentoso dominio
El dominio del IBM PC y su adopción del MS-DOS (bajo el nombre de PC-DOS) le ha asegurado al MS-DOS el primer puesto entre los OS de gestión de 16 bits. El sistema nació como una versión de CP/M hecha a medida, rudimentaria pero efectiva, por un ex empleado de Digital Research. Luego fue adquirida por Microsoft, que le dio la licencia a IBM

Un juego en regla

Nos ocuparemos de las rutinas encargadas del almacenamiento y de la evaluación de cada mano

Esbozaremos las reglas por las que se regirá nuestro programa; existen tantas versiones del juego como nombres del mismo. No obstante, todas tienen varios aspectos fundamentales en común, siendo el más importante el que el jugador intenta acercarse lo más posible a la puntuación de 21 sin pasarse. Las cartas de nuestro programa llevan los siguientes valores: los naipes numerados valen en función de los valores de sus caras, los naipes de figura (Valet, Dama y Rey) valen 10, y los ases valen uno u 11, a discreción del jugador.

En su forma más simple, en el veintiuno juega un jugador contra la "banca", que reparte dos naipes de la parte superior de la baraja: uno boca arriba para el jugador y otro boca abajo para sí misma. Luego se reparte otro naipe cara arriba a cada uno, en cuyo momento el jugador debe decidir si "plantarse", si su mano está en 21 o cerca de 21, o "doblar" y recibir otro naipe. El riesgo, por supuesto, es que el jugador pueda "pasarse" de 21. Después de que el jugador se plante, la banca jugará su mano hasta vencer al jugador o bien pasarse.



Dándonos una mano
Se utiliza una matriz tridimensional, HD(,,), para retener las manos del jugador y la banca a medida que se reparten los naipes de la baraja. Una segunda matriz, HP(), retiene punteros al siguiente espacio libre de cada sección de la matriz de manos. Al final de cada juego el programa no necesita limpiar la matriz; tan sólo debe restablecer los punteros a uno

Hemos introducido en este programa algunas reglas adicionales que se utilizan comúnmente en el veintiuno. En primer lugar, usted no puede plantarse si su mano es inferior a 17. En segundo lugar, si tras habérsele repartido los dos primeros naipes su marcador es de 12, 13 o 14, tiene la opción de "quemar" sus naipes, y hacer que se le repartan otros dos. En caso de empate entre usted y la banca, ésta siempre gana.

Aunque hemos visto cómo se reparten los naipes de la baraja y cómo se visualizan, no le hemos mostrado cómo el programa podría "recordar" las cartas repartidas. Para retener las manos del jugador y de la banca, prepararemos una matriz tridimensional HD(,,). Dado que ninguno de los bandos requerirá nunca más de cinco naipes, esta matriz se dimensiona en la línea 550 para ser de dos elementos (para los dos jugadores) por cinco (para los naipes) por dos (para retener los números y palos por separado). Se utiliza una segunda matriz, HP(), para retener un puntero hacia el siguiente espacio libre en la matriz de manos para cada jugador.

En esta etapa necesitamos añadir la línea 1325 a la rutina para repartir desarrollada previamente, de modo que el número de naipe, CN, y el número de palo, SU, se añadan en la matriz de manos y se incremente el puntero de mano. Observe que si PL=1 al entrar esta rutina, el naipe repartido se entra en la mitad de la matriz de manos perteneciente al jugador; si PL=2, los detalles del naipe se entran en la mitad de la banca.

Ahora se pueden repartir los cuatro primeros naipes del juego estableciendo PL en 1 o 2 y llamando a la rutina para repartir. Puesto que la primera carta a repartir a la banca ha de estar boca abajo, la rutina para repartir está escrita de modo tal que si una bandera (FL) está en 1, se llama a la rutina para visualizar el reverso del naipe. Los detalles de la carta oculta se entran en la matriz de manos.

La subrutina de la línea 800 está diseñada como una rutina de evaluación con fines generales que cumple dos funciones. Primero, calcula el valor total retenido en la mano que se está examinando, y en segundo lugar establece la variable EF en un valor entre 1 y 5, correspondiente a los cinco estados posibles en los que puede estar una mano. El cálculo del estado de una mano es relativamente directo, pero lo que es más difícil es calcular el valor total de ésta. Si bien a primera vista parece que todo cuanto necesitamos hacer es totalizar los números de naipes retenidos en la matriz de manos, los naipes de figura valen 10 y los ases valen poco o mucho. El primer problema se puede resolver fácilmente comprobando el número de naipe; si éste es superior a 10, entonces se le sumará 10 al valor total.

El problema de los ases, sin embargo, requiere un poco más de atención, dado que una mano puede contener hasta cuatro ases, creando hasta 16 posibles permutaciones de valor con las que tratar. En realidad, no vale la pena considerar la mayoría de estas permutaciones, porque hacen que la mano se pase inmediatamente. En consecuencia, podemos seguir una sencilla regla para manejar los ases: el primer as de la mano puede valer uno u 11, pero todos los ases siguientes se cuentan en su valor bajo o la mano se pasará.

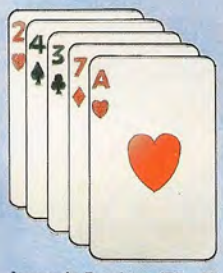
Para codificar esta regla, la forma más simple es tener dos opciones de valor: TT(PL,1), en la cual el primer as se cuenta bajo, y TT(PL,2), en la cual el primer as se cuenta alto. Utilizamos un único bucle para sumar los valores de las manos, tratando a todos los ases encontrados como bajos, y contando la cantidad de ases. Al salir del bucle, comprobamos la cuenta de ases y, si no es cero, le sumamos 10 a TT(PL,2). El resto de la rutina de evaluación comprueba los totales de TT(PL,1) y TT(PL,2) o la matriz de manos para determinar en qué estado se halla la mano y establece la bandera de estado de evaluación, EF, en consecuencia.

Si a usted se le han repartido dos naipes que totalizan 12, 13 o 14, tiene la opción de quemarlos y se le repartirán otros dos. Esta rutina llama a la rutina de evaluación analizada anteriormente y utiliza los totales de TT(PL,1) y TT(PL,2) para determinar si usted puede quemarlos. Si opta por hacerlo así, se borra la mitad izquierda de la visualización en pantalla, se restablece su puntero de mano, HP(PL), y se le reparten dos nuevos naipes.

Esquemas de valores

En el veintiuno *(pontoon)*, una mano puede estar durante el juego en cinco estados diferentes. La siguiente lista muestra los diversos estados que reconoce nuestro programa por orden jerárquico. La rutina de evaluación establece una variable, EF, según el estado en el que esté la mano

Royal pontoon (EF=2)
Un marcador de 21 compuesto por un as y una figura. Un as y un 10 no se consideran *royal pontoon*



Juego de 5 naipes (EF=5)
Un marcador de 21 o menos con cinco naipes

Pontoon (EF=3)
Un marcador de exactamente 21

Menos de 21 (EF=1)
Un marcador de menos de 21. Éste puede ser el estado de una mano al final de un turno o en alguna etapa intermedia

Pasarse, o *bust* (EF=4)
Un marcador de más de 21

Kevin Jones

Rutinas de almacenamiento y evaluación de mano

Sinclair Spectrum

```
 60 > LET FL=0: LET PL=1: GO SUB 1300: REM REPARTIR
    NAIPE A APOSTADOR
 70 LET FL=1: LET PL=2: GO SUB 1300: REM REPARTIR
    NAIPE A BANCA
 85 LET FL=0: LET PL=1: GO SUB 1300: REM REPARTIR
    NAIPE A APOSTADOR
 90 LET FL=0: LET PL=2: GO SUB 1300: REM REPARTIR
    NAIPE A BANCA
 95 REM **** TURNO DEL APOSTADOR ****
100 LET PL=1
102 GO SUB 2300: REM OPCION QUEMAR
```

Dimensionar matrices de manos

```
550 > DIM D(2,5,2):DIM P(2): REM MANOS DEL APOSTADOR
    Y DE LA BANCA
555 DIM T(2,2): REM TOTALES PUNTUACION
670 REM **** BORRAR VISUALIZACION NAIPE ****
675 LET X(PL)=EP: LET Y(PL)=0
680 PRINT AT 0,0;: FOR I=1 TO 12: PRINT AT I,EP,S$(TO 7):
    NEXT I: RETURN
700 REM **** PREPARAR PARA ENTRADA ****
710 LET TX=0: LET TY=17: GO SUB 900: PRINT S$( TO 31)
720 LET TX=0: LET TY=17: GO SUB 900: RETURN
```

Rutina de evaluación de mano

```
800 REM **** EVALUAR MANO ****
810 LET AV=1: FOR J=1 TO 2
812 LET T(PL,J)=0
815 FOR I=1 TO P(PL)-1
820 IF D(PL,I,1)=1 THEN LET T(PL,J)=T(PL,J)+AV-1
825 IF D(PL,I,1)>10 THEN LET
    T(PL,J)=T(PL,J)+10-D(PL,I,1)
830 LET T(PL,J)=T(PL,J)+ D(PL,I,1)
840 NEXT I: LET AV=11: NEXT J
852 IF (T(PL,1)<=21) OR T(PL,2)<=21) AND P(PL)> 5
    THEN LET EF=5: RETURN
854 IF D(PL,1,1)=1 AND D(PL,2,1)> 10 THEN LET EF=2:
    RETURN
855 IF D(PL,2,1)=1 AND D(PL,1,1)> 10 THEN LET EF=2:
    RETURN
856 IF T(PL,1)=21 OR T(PL,2)=21 THEN LET EF=3: RETURN
858 IF T(PL,1)<21 OR T(PL,2)<21 THEN LET EF=1: RETURN
860 IF T(PL,1)> 21 AND T(PL,2)>21 THEN LET EF=4:
    RETURN
2300 > REM **** OPCION QUEMAR ****
2305 GO SUB 800: REM EVALUAR
2310 IF T(PL,1)<>T(PL,2) OR T(PL,1)<12 OR T(PL,1)> 14
    THEN RETURN
2340 GO SUB 700: PRINT "QUEMAR (S/N) ";
2345 LET A$=INKEY$: IF A$="" THEN GO TO 2345
2347 IF A$<> CHR$ 13 THEN PRINT A$
2350 IF A$<> "S" THEN RETURN
2360 LET P(1)=1: REM RESTABLECER PUNTERO MANO
2370 LET EP=0: GO SUB 670: REM BORRAR NAIPES
2380 LET FL=0: LET PL=1: GO SUB 1300: GO SUB 800: REM
    REPARTIR NAIPE A APOSTADOR
2390 LET FL=0: LET PL=1: GO SUB 1300: GO SUB 800: REM
    REPARTIR NAIPE A APOSTADOR
2400 GO TO 2300: REM QUEMAR OTRA VEZ?
```

BBC Micro

```
 60 FL=0: PL=1: GOSUB 1300
 70 FL=1: PL=2: GOSUB 1300
 85 FL=0: PL=1: GOSUB 1300
 90 FL=0: PL=2: GOSUB 1300
 95 REM
100 PL=1
102 GOSUB 2300
```

Dimensionar matrices de manos

```
550 DIM HD(2,5,2),HP(2)
555 DIM TT(2,2)
670 REM **** BORRAR VISUALIZACION NAIPE ****
675 X(PL)=EP:Y(PL)=0
```

```
680 PRINT TAB(0,0);:FOR I=1 TO 12:PRINT
    TAB(EP,I);LEFT$(SP$,19):NEXT I: RETURN
700 REM
710 COLOUR 1:TX=0:TY=23:GOSUB 900:PRINT SP$
720 TX=0:TY=23:GOSUB 900:RETURN
```

Rutina de evaluación de mano

```
800 REM
810 AC=0:AV=0:TT(PL,1)=0
815 FOR I=1 TO HP(PL)-1
820 IF HD(PL,I,1)=1 THEN AC=AC+1
825 IF HD(PL,I,1)>10 THEN
    TT(PL,1)=TT(PL,1)+10-HD(PL,I,1)
830 TT(PL,1)=TT(PL,1)+HD(PL,I,1)
840 NEXT I
845 IF AC> 0 THEN AV=10
850 TT(PL,2)=TT(PL,1)+AV
852 IF(TT(PL,1)<=21 OR TT(PL,2)<=21) AND HP(PL)>5
    THEN EF=5:RETURN
854 IF HD(PL,1,1)=1 AND HD(PL,2,1)> 10 THEN
    EF=2:RETURN
855 IF HD(PL,2,1)=1 AND HD(PL,1,1)> 10 THEN
    EF=2:RETURN
856 IF TT(PL,1)=21 OR TT(PL,2)=21 THEN EF=3:RETURN
858 IF TT(PL,1)<21 OR TT(PL,2)<21 THEN EF=1:RETURN
860 IF TT(PL,1)>21 AND TT(PL,2)>21 THEN EF=4:RETURN
2300 REM
2305 GOSUB 800
2310 IF TT(PL,1)<>TT(PL,2) OR TT(PL,1)<12 OR
    TT(PL,1)>14 THEN RETURN
2340 GOSUB 700:PRINT"QUEMAR (S/N)";
2345 RESP$=GET$
2347 IF RESP$<>CHR$(13) THEN PRINT RESP$
2350 IF RESP$<>"S" THEN RETURN
2360 HP(1)=1
2370 EP=0:GOSUB 670
2380 FL=0:PL=1:GOSUB 1300:GOSUB 800
2390 FL=0:PL=1:GOSUB 1300:GOSUB 800
2400 GOTO 2300
```

Gama Amstrad CPC

```
 60 fl=0:pl=1:GOSUB 1300:REM repartir naipe a apostador
 70 fl=1:pl=2:GOSUB 1300:REM repartir naipe a la banca
 85 fl=0:pl=1:GOSUB 1300:REM repartir naipe a apostador
 90 fl=0:pl=3:GOSUB 1300:REM repartir naipe a la banca
 95 REM **** turno del apostador ****
100 pl=1
102 GOSUB 2300:REM opcion quemar
```

Dimensionar matrices de manos

```
550 DIM hd(2,5,2),hp(2):REM manos apostador y banca
555 DIM tt(2,2):REM totales marcadores
670 REM **** borrar naipe ****
675 x(pl)=ep:y(pl)=0:tx=ep:ty=0
680 FOR i=1 TO 12:GOSUB 900:PRINT SPACE$(19)
690 ty=ty+1:NEXT i:RETURN
700 REM **** preparar para entrada ****
710 PEN blanco:tx=0:ty=23:GOSUB 900:PRINT SPACE$(39)
720 tx=0:ty=23:GOSUB 900:RETURN
```

Rutina de evaluación de mano

```
800 REM **** evaluar mano ****
810 ac=0:av=0:tt(pl,1)=0
815 FOR i=1 TO hp(pl)-1
820 IF hd(pl,i,1)=1 THEN ac=ac+1
825 IF hd(pl,i,1)> 10 THEN tt(pl,1)=tt(pl,1)+10-hd(pl,i,1)
830 tt(pl,1)=tt(pl,1)+hd(pl,i,1)
840 NEXT i
845 IF ac> 0 THEN av=10
850 tt(pl,2)=tt(pl,1)+av
852 IF (tt(pl,1)<=21 OR tt(pl,2)<=21)AND hp(pl)> 5 THEN
    ef=5:RETURN
854 IF hd(pl,1,1)=1 AND hd(pl,2,1)> 10 THEN ef=2:RETURN
855 IF hd(pl,2,1)=1 AND hd(pl,1,1)>10 THEN ef=2:RETURN
856 IF tt(pl,1)=21 OR tt(pl,2)=21 THEN ef=3:RETURN
858 IF tt(pl,1)< 21 OR tt(pl,2)<21 THEN ef=1:RETURN
860 IF tt(pl,1)> 21 AND tt(pl,2)>21 THEN ef=4:RETURN
2300 REM **** opcion quemar ****
2305 GOSUB 800:REM evaluar
2310 IF tt(pl,1)<>tt(pl,2) OR tt(pl,1)< 12 OR tt(pl,2)> 14
    THEN RETURN
2340 GOSUB 700:PRINT "Quemar (s/n)";
2345 resp$="":WHILE resp$="":resp$=INKEY$:WEND
2347 IF resp$<>CHR$(13) THEN PRINT resp$
2350 IF resp$<> "s" THEN RETURN
2360 hp(pl)=1:REM restablecer puntero mano
2370 ep=0:GOSUB 670:REM borrar naipes
2380 fl=0:pl=1:GOSUB 1300:GOSUB 800:REM repartir naipe a
    apostador
2390 fl=0:pl=1:GOSUB 1300:GOSUB 800:REM repartir naipe a
    apostador
2400 GOTO 2300:REM quemar otra vez?
```

Commodore 64

```
 60 FL=0:PL=1:GOSUB 1300:REM REPARTIR NAIPE A
    APOSTADOR
 70 FL=1:PL=2:GOSUB 1300:REM REPARTIR NAIPE A LA
    BANCA
 85 FL=0:PL=1:GOSUB 1300:REM REPARTIR NAIPE A
    APOSTADOR
 90 FL=0:PL=2:GOSUB 1300:REM REPARTIR NAIPE A LA
    BANCA
 95 REM **** TURNO DEL APOSTADOR ****
100 PL=1
102 GOSUB 2300:REM OPCION QUEMAR
120 GOSUB 2600:REM DOBLAR ETC
550 DIM HD(2,5,2),HP(2):REM MANOS DEL APOSTADOR Y DE
    LA BANCA
555 DIM TT(2,2):REM TOTALES MARCADORES
```

Dimensionar matrices de manos

```
670 REM **** BORRAR VISUALIZACION NAIPE ****
675 X(PL)=EP:Y(PL)=0
680 PRINT CHR$(19);:FOR I=1 TO 12:PRINT
    TAB(EP);LEFT$(SP$,19):NEXT I: RETURN
700 REM **** PREPARAR PARA ENTRADA ****
710 PRINT CHR$(5);:TX=0:TY=23:GOSUB 900:PRINT SP$
720 TX=0:TY=23:GOSUB 900:RETURN
```

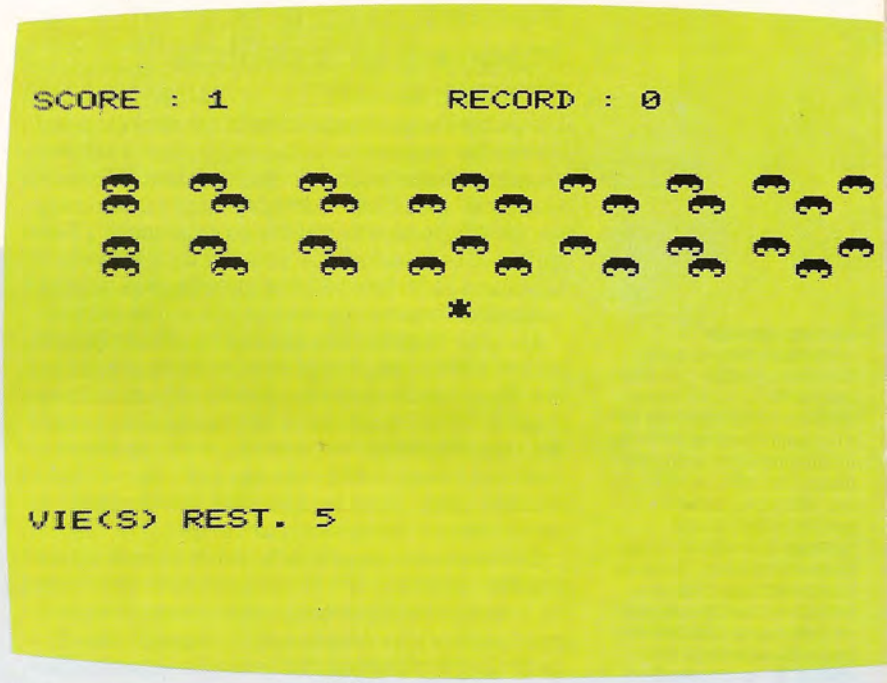
Rutina de evaluación de mano

```
800 REM **** EVALUAR MANO ****
810 AC=0:AV=0:TT(PL,1)=0
815 FOR I=1 TO HP(PL)-1
820 IF HD(PL,I,1)=1 THEN AC=AC+1
825 IF HD(PL,I,1)> 10 THEN TT(PL,1)=TT(PL,1)+
    10-HD(PL,I,1)
830 TT(PL,1)=TT(PL,1)+HD(PL,I,1)
840 NEXT I
845 IF AC> 0 THEN AV=10
850 TT(PL,2)=TT(PL,1)+AV
852 IF (TT(PL,1)< =21 OR TT(PL,2)<=21) AND HP(PL)>5
    THEN EF=5:RETURN
854 IF HD(PL,1,1)=1 AND HD(PL,2,1)> 10 THEN
    EF=2:RETURN
855 IF HD(PL,2,1)=1 AND HD(PL,1,1)> 10 THEN
    EF=2:RETURN
856 IF TT(PL,1)=21 OR TT (PL,2)=21 THEN EF=3:RETURN
858 IF TT(PL,1)< 21 OR TT(PL,2)<21 THEN EF=1:RETURN
860 IF TT(PL,1)> 21 AND TT(PL,2)> 21 THEN EF=4:RETURN
2300 REM **** OPCION QUEMAR ****
2305 GOSUB 800:REM EVALUAR
2310 IF TT(PL,1)<>TT(PL,2) OR TT(PL,1)< 12 OR TT(PL,1)>
    14 THEN RETURN
2340 GOSUB 700:PRINT"QUEMAR (S/N)";
2345 GET RESP$:IF RESP$="" THEN 2345
2347 IF RESP$<> CHR$(13) THEN PRINT RESP$
2350 IF RESP$<> "S" THEN RETURN
2360 HP(1)=1:REM RESTABLECER PUNTERO MANO
2370 EP=0:GOSUB 670:REM BORRAR NAIPES
2380 FL=0:PL=1:GOSUB 1300:GOSUB 800:REM REPARTIR
    NAIPE A APOSTADOR
2390 FL=0:PL=1:GOSUB 1300:GOSUB 800:REM REPARTIR
    NAIPE A APOSTADOR
2400 GOTO 2300:REM QUEMAR OTRA VEZ?
```

Cangrejos

Las leyes de la selección natural y de la supervivencia de los más aptos enmendadas por la informática. Esta versión de este original juego ha sido escrita para los ordenadores MSX

Usted debe ayudar a una pobre tortuga a volver al mar, evitando a los voraces cangrejos que deambulan por la playa. Cada tortuga que alcance su propósito le proporciona 1 punto. Dispone de cinco itinerarios para intentar marcar su puntuación máxima. Emplee las teclas de control del cursor para avanzar y para retroceder.




```
10 REM *************
20 REM * CANGREJOS *
30 REM *************
40 SCREEN 0,0
50 DEFINT A-Z
60 KEY OFF
70 WIDTH 39
80 GOSUB 990
90 GOSUB 1130
100 GOSUB 840
110 LOCATE 0,20,0
120 PRINT "VIDAS RESTANTES.";NP;
130 A$=RIGHT$(A$,1)+LEFT$ (A$,38)
140 B$=RIGHT$(B$,38)+LEFT$(B$,1)
150 LOCATE 0,X1,0
160 PRINT A$;
170 LOCATE 0,X2,0
180 PRINT B$;
190 LOCATE 0,X3,0
200 PRINT A$;
210 LOCATE 0,X4,0
220 PRINT B$;
230 D$=INKEY$
240 PY=PY+(STICK(0)=1)-(STICK(0)=5)
250 IF PY>10 THEN PY=10
260 IF PY=2 THEN 370
270 C=VPEEK(PX+PY*40+1)
280 IF C<>32 AND C<>128 THEN 550
290 LOCATE PX,YP,0
300 PRINT N$;
310 LOCATE PX,PY,0
320 PRINT P$;
330 YP=PY
340 T=T+1
350 IF T>500 THEN 660
360 GOTO 110
370 LOCATE PX,YP,0
380 PRINT N$;
390 LOCATE PX,PY,0
400 PRINT P$;
410 BEEP
420 FOR I=1 TO 200
430 NEXT I
440 LOCATE PX,PY,0
450 PRINT N$;
460 PY=10
470 YP=PY
480 S=S+1
490 LOCATE 0,0,0
500 PRINT "PUNTUACION:";S:
510 LOCATE 19,0
520 PRINT "RECORD :";R·
530 GOSUB 1040
540 GOTO 110
550 NP=NP-1
560 LOCATE PX,YP,0
570 PRINT N$;
580 LOCATE PX,PY,0
590 PRINT CHR$(128);
600 GOSUB 1090
610 IF NP=0 THEN 660
620 PY=10
630 YP=PY
640 GOSUB 1040
650 GOTO 110
660 CLS
670 IF S>R THEN R=S
680 IF T<500 THEN 710
690 LOCATE 10,8,0
700 PRINT "** TIEMPO TRANSCURRIDO**";
710 LOCATE 10,12,0
720 PRINT "PUNTUACION:";S;
730 LOCATE 10,16,0
740 PRINT "RECORD :";R;
750 LOCATE 10,20
760 PRINT "OTRA ?";
770 IF INKEY$<>"" THEN 770
780 D$=INKEY$
790 IF D$="" THEN 780
800 IF D$<>"N" AND D$<>"n" THEN 100
810 CLS
820 LOCATE Q,0,1
830 END
840 CLS
850 COLOR 1,11
860 P$=CHR$(128)
870 N$=CHR$(32)
880 S=0
890 NP=5
900 PX=19
910 PY=10
920 YP=PY
930 X1=4
940 X2=5
950 X3=7
960 X4=8
970 T=0
980 RETURN
990 FOR I=1 TO 39
1000 READ A
1010 A$=A$+CHR$(A)
1020 NEXT I
1030 B$=A$
1040 X=RND(1)*35+2
1050 A$=RIGHT$(A$,X)+LEFT$(A$,39-X)
1060 RETURN
1070 DATA 32, 129, 130, 32, 32, 129, 130, 32, 32,
     32, 129, 130, 32, 32, 32, 32, 32, 129, 130, 32
1080 DATA 32, 32, 129, 130, 32, 32, 32, 129, 130,
     32, 32, 129, 130, 32, 32, 129, 130, 32, 32
1090 PLAY "T100O3G16O4C2O3G8O4C16E2T150C
     8E16G8F+16f8D+16E8C16O3A8G16O4C4C32E16
1100 FOR I=1 TO 4000
1110 NEXT I
1120 RETURN
1130 FOR I=0 TO 23
1140 READ A
1150 VPOKE 3072+I,A*4
1160 NEXT I
1170 RETURN
1180 DATA 12,45,63,30,30,63,45,0
1190 DATA 7,15,31,31,18,16,12,0
1200 DATA 56,60,62,62,18,2,12,0
```

© SYBEX

El chip de video

Veamos ahora de qué manera los patrones binarios de la memoria se convierten en imágenes en la pantalla

Un punto de gran importancia comercial en todo ordenador personal es la calidad y diversidad de su visualización de video. ¿Cuántos colores se pueden visualizar? ¿Cuál es la cantidad máxima de caracteres que se pueden visualizar en una línea? ¿Tiene sprites el ordenador? La mayoría de estas preguntas tienen su respuesta en el tipo de chip de video que utilice la máquina.

Un chip de video (en algunas ocasiones denominado CRTC: *cathode-ray tube controller*: controlador de tubo de rayos catódicos) desempeña una función básica: toma datos de visualización, retenidos como bytes en la memoria, y los convierte en formas y colores en la pantalla. Los chips de video difieren entre sí en la forma en que los datos se convierten en información de imagen.

Debido a que los chips de video necesitan tener vínculos estrechos con la memoria de la cual toman su información de visualización, normalmente se incorpora en ellos un sistema de circuitos de refresco de RAM dinámica. Al igual que los chips PIO, de los que hablamos en el capítulo anterior, los chips de video poseen registros internos programables que controlan sus funciones. La programación directa del chip la suele realizar el sistema operativo del ordenador, pero se pueden conseguir efectos de video especiales si el mismo usuario programa el chip de video.

El primer tipo de chip de video que analizaremos retiene todos sus datos de visualización y de color juntos en la memoria, haciendo que la tarea de conversión a una imagen resulte mucho más sencilla. Éste es el tipo de visualización que utilizan el BBC Micro, el Acorn Electron y la gama Amstrad. Tales chips normalmente soportan una cantidad de modalidades de visualización diferentes en las que varían el tamaño de los caracteres y el número de colores disponibles. El diagrama ilustra cómo se interpreta un único byte de la memoria en distintas modalidades de pantalla. Si bien este tipo de disposición simplifica el trabajo del chip de video, y es fácil mezclar texto y gráficos en alta resolución, la desventaja es la gran cantidad de memoria requerida para retener una pantalla de información. Una visualización de 160 por 256 pixels con 16 colores, por ejemplo, requiere 20 Kbytes de memoria.

Cuando el costo de la memoria era elevado, tales métodos de visualización no eran factibles y se empleaba un método de visualización alternativo, que aún está presente en el Commodore 64. En vez de retener cada pixel de información de visualización y color de forma explícita en la memoria, este segundo tipo de chip de video utiliza un sistema simplificado para visualizar gráficos de caracteres solamente. En tal visualización, cada posición de carácter corresponde a un byte de la memoria. Cada byte retiene tan sólo un código de carácter que el chip de video utiliza para acceder a una tabla de ROM de las verdaderas definiciones de caracteres de ocho por ocho pixels. De este modo, una visualización de caracteres de 40 por 25 requiere menos de un Kbyte.

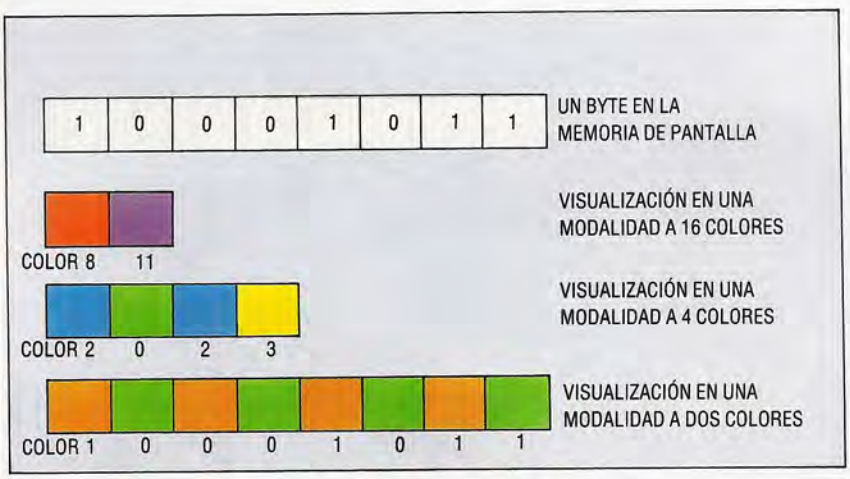
Las implementaciones en color de este tipo de sistema emplean otra zona de 1 000 bytes para retener los datos de color; cada byte retiene un código de color para la posición del carácter correspondiente.

El sistema de visualización de video empleado por el BBC Micro (y otros) aún ha de utilizar definiciones de caracteres de ROM, pero el sistema operativo accede a las mismas cuando ha de escribir un carácter en la memoria de pantalla. El proceso de buscar la definición y realizar las manipulaciones necesarias para codificar la información integral de color se lleva a cabo en la etapa de escribir en la memoria de pantalla, y no en la etapa de conversión de memoria a imagen.

Las visualizaciones de video basadas en este segundo método normalmente poseen una segunda modalidad en alta resolución en la cual los bits retenidos en la memoria guardan una correspondencia directa con los pixels de la visualización final. Este tipo de visualización de gráficos se conoce como *bit mapping* (mapa de bits). La información de color no se codifica en los bits de la visualización (como en el primer sistema descrito), sino que se retiene del mismo modo que para la visualización de caracteres solamente. Es decir, una zona de RAM de 1 000 bytes en la que cada byte retiene información de color para una zona de la pantalla de ocho por ocho pixels.

En el Commodore 64, esto permite gráficos en alta resolución, pero no se puede mezclar texto directamente a menos que el usuario escriba sus propias rutinas para escribir en el mapa de bits las verdaderas definiciones de caracteres. Resulta interesante que sea éste el enfoque adoptado para el Spectrum, que almacena su pantalla como un simple mapa de bits y utiliza una RAM de color separada para controlar cada sección de ocho por ocho pixels de la pantalla. En el Spectrum se pueden mezclar caracteres y gráficos en alta resolución por-

Color por números
Los controladores de video como los utilizados en la gama Amstrad CPC y el BBC Micro codifican la información de color y pixel *on/off* junta en memoria. En estos dos ordenadores hay disponibles varias modalidades de pantalla que interpretan los patrones retenidos en la memoria de formas diferentes. En una modalidad a 16 colores, se necesitan cuatro bits para codificar el color de cada pixel; de modo que en cada byte sólo se pueden representar dos colores. En las modalidades a cuatro y dos colores, sólo se necesitan dos y un bit, respectivamente, para el código de color de un pixel. En estas modalidades un byte puede representar cuatro u ocho pixels



que el sistema operativo escribe las verdaderas definiciones de caracteres en el mapa de bits.

El chip de video controla la visualización de gráficos sprite (en caso de que exista dicha facilidad). Por lo general, los sprites se definen mediante imágenes de bits en RAM, y sus posiciones en la pantalla, así como otros atributos, tales como color y tamaño, los controlan registros internos del chip. Dado que se utiliza un único chip para manipular gráficos normales y sprites, los chips de video como el VIC del Commodore 64 se pueden programar para generar interrupciones cuando los sprites chocan con otros sprites o con datos del fondo.

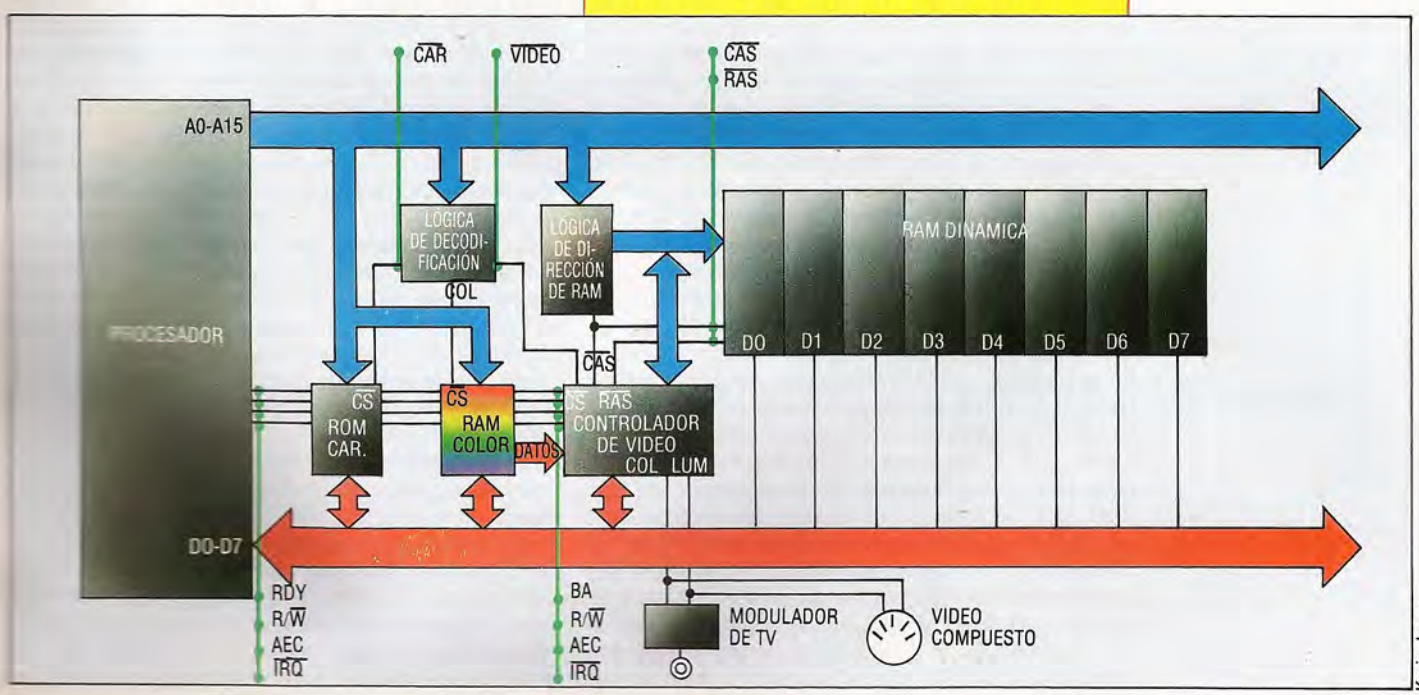
Los dos tipos de chip de video que estamos analizando aquí soportan el desplazamiento fuera de la pantalla de formas diferentes. El segundo tipo realiza el desplazamiento a través de software. Puesto que el número de bytes que componen la visualización en pantalla es reducido, el traslado de los datos de pantalla a posiciones nuevas se puede efectuar con rapidez en código máquina. El alto número de bytes necesarios para retener la pantalla en el primer método lleva a que el desplazamiento por software sea demasiado lento. Por consiguiente, los chips de video basados en este sistema incorporan un registro de "comienzo de pantalla", que retiene la dirección del principio de la memoria de pantalla. Por lo tanto, el desplazamiento se puede conseguir con sólo cambiar la dirección de este registro, y con una manipulación cuidadosa se puede utilizar este método para desplazar la pantalla hacia arriba, hacia abajo, a izquierda o a derecha.

El tipo de chip de video empleado en el BBC Micro tiene mucho menos trabajo que hacer que, por poner un ejemplo, el chip VIC del Commodore 64. En consecuencia, puede realizar sus accesos a la memoria durante la fase del ciclo de reloj, cuando el procesador propiamente dicho no accede a la memoria. Este chip de video, por lo tanto, no reduce la velocidad del procesador. Además de utilizar el chip de video, el BBC incorpora una ULA diseñada a medida que proporciona la temporización para la totalidad del sistema, calcula las relaciones entre colores lógicos y físicos y proporciona salida de video RGB.

El chip VIC necesita efectuar varios accesos a memoria. Algunos, como el refresco de la RAM y las búsquedas de datos de caracteres (en el BBC Micro), son invisibles para el procesador, porque se pueden realizar en la fase alternada del reloj del sistema. Algunas funciones necesitan datos a una velocidad mayor que ésta y las operaciones del procesador han de ser suspendidas temporalmente mientras el chip de video realiza sus accesos a la memoria.

Tomando la imagen

El diagrama muestra cómo se conecta con el sistema el chip VIC del Commodore 64. El VIC se conecta directamente al bus de datos y utiliza las mismas líneas de dirección multiplexadas utilizadas por la RAM dinámica, que también se refresca utilizando las líneas de control CAS y RAS. Algunas de las operaciones del chip de video se pueden llevar a cabo sin perturbar al procesador. Asimismo, el control de habilitación del bus de direcciones (AEC) asegura que los manejadores del bus de direcciones del procesador estén desconectados durante la fase de reloj en la que el chip realiza sus accesos a memoria. Los accesos a las zonas de 1000 bytes de visualización de RAM y datos de sprites se debe realizar más rápido que cada fase alterna, y, por tanto, es necesario inhabilitar el procesador mientras el chip de video "roba" ciclos de memoria para acceder a estas zonas. Esto se consigue uniendo la línea de bus disponible (*bus available*: BA) del VIC con la patilla READY del procesador. Se conceden tres ciclos más de reloj para que el procesador complete cualquier operación de acceso a la memoria. Sin embargo, en la fase de acceso del procesador del cuarto ciclo de reloj, AEC permanecerá *low* para permitirle al VIC el acceso a la memoria durante esta fase. El mantenimiento de la visualización de video disminuye la velocidad del procesador



Kevin Jones

Programa atípico

Sobre él ya se ha escrito todo
Apple se ha enfrentado con una enorme reticencia por parte de los destinatarios a la hora de querer imponer el Macintosh como máquina de gestión seria. Parte del problema ha residido, sin duda, en la ausencia de un exhaustivo paquete para tratamiento de textos, que es un prerrequisito para las aplicaciones de gestión. No obstante, han aparecido varios programas para tratamiento de textos, si bien sólo son un puñado en comparación con los centenares que existen a la venta para el IBM PC

Profusión de iconos
Al igual que todos los paquetes escritos para el Macintosh, el *MacWrite* hace un uso intensivo de los iconos para ejecutar instrucciones posicionadas a lo largo del perímetro de la pantalla. Muchos de éstos (como la barra de desplazamiento) se utilizan comúnmente en numerosos programas para el Macintosh, mientras que otros son exclusivos del *MacWrite*

Conozcamos el "MacWrite", el paquete de tratamiento de textos producido por Apple para el Macintosh

Contrastando con los paquetes que hemos visto en anteriores capítulos, que se basan en menús y caracteres de control, *MacWrite*, el programa de tratamiento de textos empaquetado con el Macintosh, se basa en gran medida en los iconos y las ventanas controladas por ratón, que ahora ya se esperan en cualquier programa para el Macintosh.

Al cargar *MacWrite*, al usuario se le presenta una pantalla de apertura, compuesta por una gran ventana en la cual se visualizará el documento. Arriba hay una serie de iconos y una regla, y arriba de todo hay una barra de menú compuesta por seis títulos. Debajo de ésta se halla la barra de títulos, donde aparece el nombre del documento actual.

A diferencia de la mayoría de los otros paquetes, en los que el cursor se desplaza con la ayuda de teclas de control, *MacWrite* lo manipula con el controlador de ratón. En otros procesadores de textos, el cursor se utiliza para señalar una posición en el texto que indica dónde se ha de llevar a cabo una determinada opción. Se puede posicionar, por ejemplo, para insertar texto o crear un bloque. En *MacWrite* el cursor se emplea de forma idéntica, pero también se utiliza para mover los iconos, que alteran los controles de tabulación y de márgenes, así como para bajar y seleccionar las entradas de los menús.

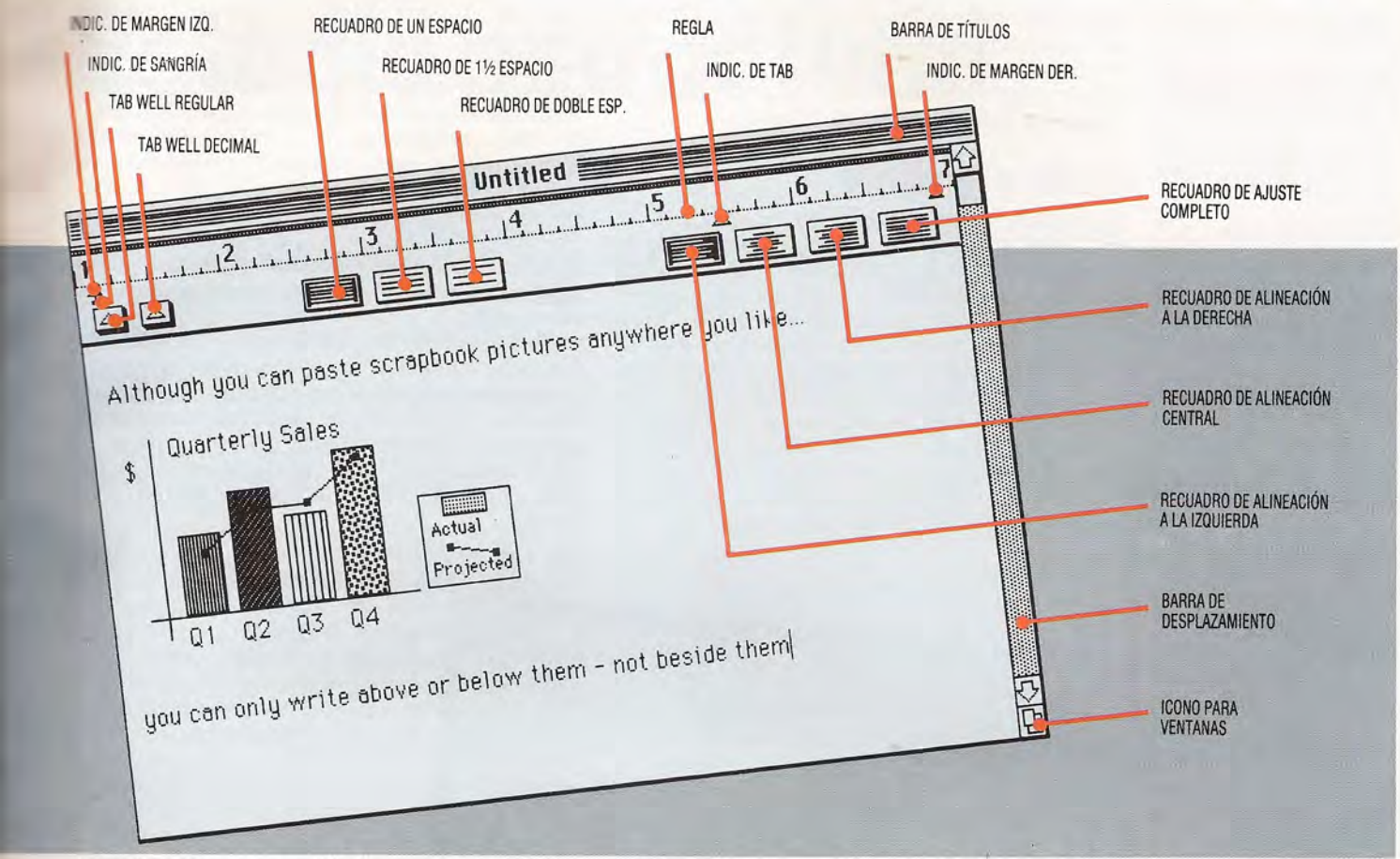
Al bajar un menú se visualizan varias opciones diferentes, pero no todas ellas son utilizables de inmediato. Las que sí están disponibles se indican en letras negras, mientras que las otras están sombreadas en gris. Con frecuencia el menú consiste en un cierto número de opciones preestablecidas, habiendo una marca junto a la seleccionada actualmente. La selección de otra opción preestablecida trasladará la marca hasta esa selección, indicando que ésa es la opción actual por defecto.

El menú File (de archivos) contiene las instrucciones DOS que permiten abrir y cerrar archivos, además de guardarlos bajo el nombre de archivo actual o uno distinto. También se incluye en este menú la instrucción Print (imprimir).

El segundo de los menús listados es Edit (editar), que proporciona instrucciones para manipular el texto.

Quizá para el principiante la más importante de éstas sea Undo (deshacer), que borrará todas las acciones llevadas a cabo desde que se pulsó el botón del ratón. Tal vez esto parezca drástico, pero muchos errores se producirán a resultas de selecciones erróneas con el ratón. De modo que, utilizando Undo, se perderá muy poco del trabajo valioso. En este menú también se incluyen las instrucciones para movimiento de bloques.

Al igual que con *WordStar*, los bloques se crean delimitando el texto seleccionado. Tras colocar el cursor ya sea al comienzo o al final del texto y pulsando el ratón, el cursor cambiará a un icono conocido como el "punto de inserción", que permitirá insertar texto a partir de esa posición. Si usted desplaza el cursor hacia arriba o hacia abajo de este punto, el texto correspondiente aparecerá en video



invertido, lo que en realidad marca el bloque. Una segunda pulsación del ratón establecerá entonces el otro extremo del bloque.

Una vez marcado el bloque, se pueden realizar con él varias operaciones diferentes. Retornando al menú Edit, por ejemplo, se puede seleccionar la opción Cut (cortar), que coloca al bloque en un "tablero de chinchetas". Éste es un buffer que almacena el bloque antes de efectuar operaciones en él. Para contemplar el contenido de este buffer, se selecciona la opción Show Clipboard (mostrar "tablero de chinchetas") del menú Edit. Estableciendo otro punto de inserción y retornando al menú Edit, usted puede seleccionar las opciones Copy (copiar) o Paste (pegar), que le permiten ya sea copiar o transferir bloques en esa posición. Observe que todos los bloques implicados en estas operaciones se reformatearán automáticamente.

El menú Search (buscar) contiene la instrucciones Find/Replace (hallar/reemplazar) y es idéntico al implementado en la mayoría de los paquetes de tratamiento de textos.

Muchos procesadores de textos permiten utilizar diferentes tipos de letra en un documento dado, pero la mayoría de ellos se limitan a los que utilizan para dar énfasis, tales como cursiva y negrita. Además, dado que la mayoría de los paquetes se limitan a utilizar las matrices de caracteres que ya están retenidas en la ROM del sistema operativo del ordenador, por lo general es imposible ver exactamente cómo aparecerá el documento hasta que se lo ha impreso.

MacWrite, sin embargo, contiene muchos tipos de letra diferentes, añadiéndosele más a medida que aparece cada nueva versión de software. Asimismo, debido a que la pantalla de textos de Macintosh es por mapa de bits en lugar de tener la disposición usual de celdas de caracteres, estos tipos de letra se pueden visualizar en la pantalla. No obstante, a pesar de esta facilidad, el sistema no es perfecto. Muchos de los tipos de letra, como el London (de estilo gótico), son casi ilegibles.

Esta facilidad de mapa de bits se utiliza con resultados más satisfactorios en el menú Style (estilo), una serie de opciones que proporcionan las formas de énfasis y efecto más usuales, tales como letras subrayadas y en negrita. Éstas se pueden imprimir en pantalla para poder apreciar su efecto antes de enviar el documento a la impresora.

La mayor parte de las funciones de trazado de página del *MacWrite* están retenidas en el menú Format (formatear), si bien los verdaderos formatos de tamaño de página están retenidos en el menú File. Se pueden establecer cabeceras y pies de página (títulos estándares que aparecerán arriba y abajo de cada página impresa), así como el número de página, hora y fecha. Los tres últimos se visualizan como iconos y se los puede "arrancar" de su propia barra de menú y colocar en cualquier sitio en la zona de cabecera o pie.

Debajo de la regla, en la parte superior de la pantalla, hay una serie de pequeños iconos de flecha que se pueden arrastrar a lo largo para establecer los márgenes, sangría de párrafos y ajustes Tab. El documento se restablecerá automáticamente cuando se modifique algún parámetro. Dos recuadros situados debajo y a la derecha de la regla son los Tab wells, a partir de los cuales usted puede obtener otros iconos y ponerlos en la regla. A la derecha de éstos hay varios iconos de página, cada uno

MACWRITE

Desplazamiento de palabras

★★★★★

El desplazamiento de palabras y la alineación automática están totalmente implementados.

Movimiento de bloques

★★★★★

En un documento, el texto y los gráficos se pueden desplazar y posicionar en cualquier punto.

Ayuda en pantalla

No tiene.

Pantalla de 80 columnas

★★★

Sólo posee pant. de 60 columnas en mod. por defecto; el texto con caracteres de matriz de 9 puntos (el tamaño de letra más pequeño) permite 80.

Contador de palabras

En el *MacWrite* no se proporciona ninguna facilidad para contar las palabras.

Buscar/reemplazar

★★★

Permite buscar y alterar hasta 44 caracteres en la ventana actual.

WYSIWYG

★★★★★

Mostrar varios tipos de letra y diagramas en pantalla es una de sus características fundamentales.

Facil. para correspondencia

No tiene.

Verificador de ortografía

No tiene, aunque hay a la venta utilidades producidas por fabricantes independientes.

Tipos de letra disponibles

★★★★★

Implementa una amplia variedad de tipos de letra, si bien el número de éstos varía de una versión a otra.

Unión de archivos

★

A pesar de que algunas facilidades (como Cortar y Pegar) se pueden transportar entre documentos, no existe una facilidad de unión de archivos como tal.

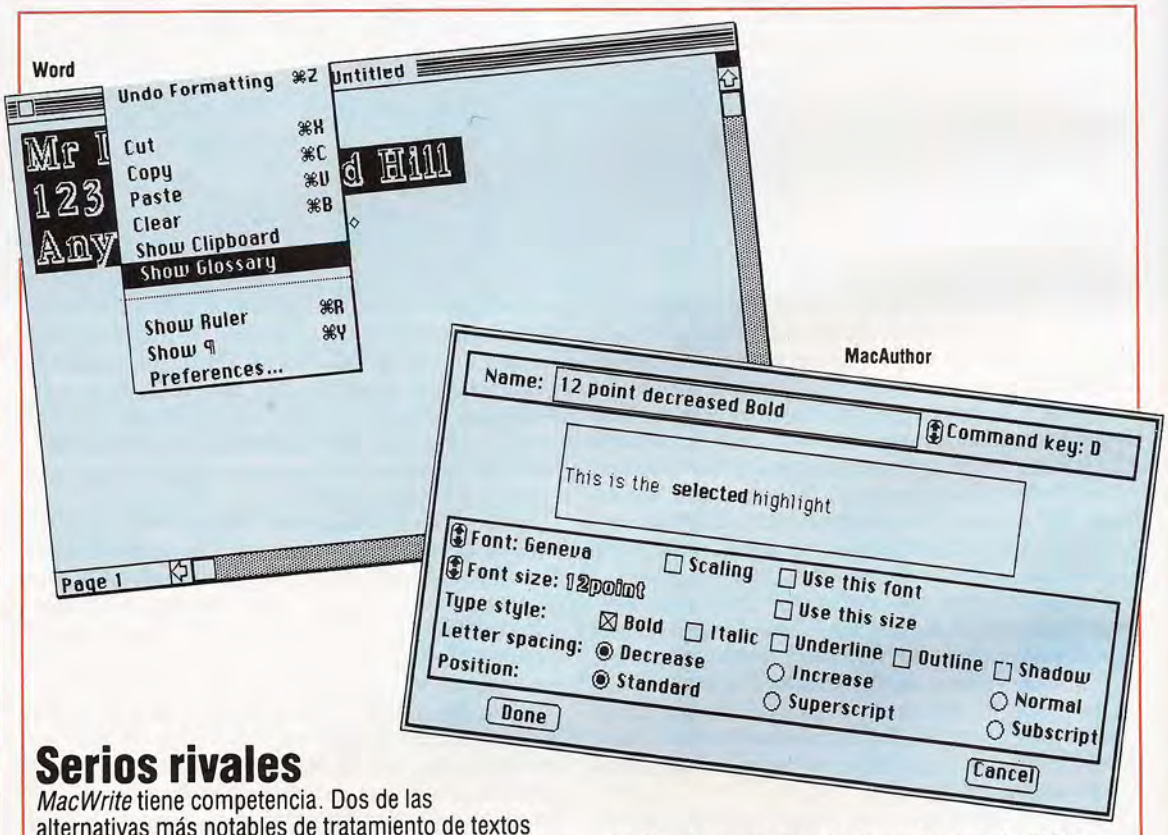
de los cuales posee líneas que indican sus funciones. Los primeros tres corresponden a separación de una línea, de una línea y media y de dos líneas, mientras que los cuatro restantes al otro extremo de la barra indican el ajuste de línea.

Texto y gráficos

El Macintosh permite combinar textos y gráficos en la página y visualizarlos en la pantalla. *MacWrite* soporta esta facilidad, si bien las imágenes se han de dibujar con *MacPaint* y *MacDraw*, los paquetes artísticos empaquetados en el Macintosh. A las ilustraciones se puede acceder mediante la facilidad Scrapbook (álbum de recortes), que forma parte del escritorio incorporado del Macintosh. Scrapbook es un área de la memoria reservada para almacenar y transferir tanto texto como gráficos a cualquier lugar de un documento. Además, debido a la flexibilidad de la máquina, las ilustraciones se pueden ajustar a cualquier tamaño que se requiera.

Si bien el *MacWrite* es un paquete sumamente sencillo de aprender y utilizar, así y todo posee sus inconvenientes. En primer lugar, el programa deja poco espacio de memoria disponible para texto en el Macintosh estándar. Esto podría resultar difícil de creer considerando que la máquina tiene 128 Kbytes, pero el sistema operativo y la pantalla por mapa de bits ocupan muchísimo espacio. Asimismo, el hecho de que el texto del *MacWrite* resida enteramente en la memoria del ordenador (al contrario que en algunos procesadores de textos avanzados, que guardan continuamente el texto en disco), significa que en la máquina sólo se pueden escribir alrededor de cinco páginas antes de quedarse sin memoria.

Así y todo, el *MacWrite* es un paquete inusual basado en un OS atípico. La forma en que se ha adaptado el formato WIMP para una aplicación que previamente se había basado por entero en entrada por teclado es a la vez interesante e ingeniosa.



Serios rivales

MacWrite tiene competencia. Dos de las alternativas más notables de tratamiento de textos para el Macintosh son *Word*, de Microsoft, y *MacAuthor*, de Icon Technology. Ambos pretenden solucionar el problema central del *MacWrite*: su incapacidad para tratar más de unas pocas páginas de texto. A primera vista, ambos paquetes parecen similares al *MacWrite*, con instrucciones activadas por iconos y una pantalla rodeada por las barras de desplazamiento y títulos. No obstante, contienen facilidades adicionales que hacen del Macintosh una proposición mucho más seria para intentar el tratamiento de textos profesional.

Word contiene varias facilidades destinadas a permitir mayor flexibilidad y potencia, permitiendo manipular hasta cuatro documentos en pantalla al mismo tiempo. También ofrece un tipo mejorado de Glosario, en lugar de depender de la facilidad Scrapbook, que tiene sus limitaciones.

Mientras que *Word* está pensado como sistema de tratamiento de textos de gestión para el Macintosh, *MacAuthor*, como su nombre sugiere, está dirigido a los escritores. Se ha desarrollado para permitir la mayor flexibilidad posible en el uso de caracteres. El usuario puede crear caracteres nuevos, por ejemplo, combinando los ya existentes en el juego de caracteres y alterando el "estilo" del texto a su voluntad. Debido a esta mayor flexibilidad, familiarizarse con *MacAuthor* es un poco más difícil que hacerlo con *Word* o *MacWrite*. A pesar de esto, a muchos escritores este esfuerzo adicional les resultará rentable

Estudio estructural

Llegados a este punto, nos corresponde examinar algunas de las estructuras de control propias del C

Los operadores relacionales ordinarios del C son los mismos que los de la mayoría de los otros lenguajes, es decir, <, <= –>, >= –, pero observe que el símbolo de igualdad es == y el de no igualdad es !=. Los conectores lógicos son && para AND e || para OR. La precedencia de operadores es la habitual, de modo que se pueden construir expresiones lógicas complejas de forma muy parecida al BASIC. Estas expresiones asumen un significado mayor cuando uno recuerda que una asignación posee un valor y, por consiguiente, se puede comprobar en una condición lógica. Por ejemplo, mediante el empleo de la función de biblioteca estándar getchar(), que busca el siguiente carácter del dispositivo de entrada estándar, podemos construir una condición:

```
contador_car(long_línea_máx – 1&&
    (c=getchar())!=/n'&&c!=EOF
```

Esta condición comprobará primero una cantidad máxima de caracteres y luego, de no haberse alcanzado esta cantidad, buscará el siguiente carácter y comprobará si se trata de un final de línea o un final de archivo, todo ello en una expresión lógica.

Otra característica es que las expresiones lógicas también poseen valores; falso es cero y verdadero es uno (de hecho, todo valor que no sea cero se tomará como verdadero). De modo que, si es necesario, se pueden utilizar expresiones lógicas en los cálculos. Por el contrario, los valores numéricos simples se pueden probar directamente sin utilizar ningún operador. El operador de negación unario ! se puede colocar delante de cualquier valor entero y lo cambiará por cero si no es cero, o por uno si es cero. De modo, entonces, que la comprobación:

```
if(!valor_ent)
```

equivale a:

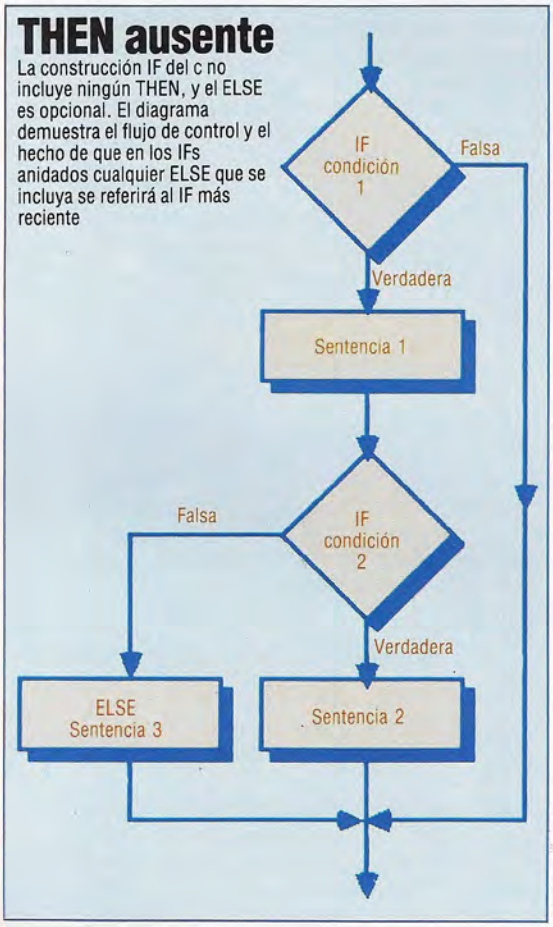
```
if (valor_ent==0)
```

Además del conjunto normal de operadores lógicos, el C posee cierto número de operadores lógicos aplicables a nivel de bit que se pueden utilizar sólo en tipos integer o char cualesquiera para proporcionar la manipulación de bits que ofrecen la mayoría de los ensambladores. Éstos son:

&	AND
|	OR inclusivo
^	OR exclusivo
<<n	donde n es un entero, desplaza n bits hacia la izquierda
>>n	donde n es un entero, desplaza n bits hacia la derecha
~	complemento a uno

THEN ausente

La construcción IF del C no incluye ningún THEN, y el ELSE es opcional. El diagrama demuestra el flujo de control y el hecho de que en los IFs anidados cualquier ELSE que se incluya se referirá al IF más reciente



Caroline Clayton

A modo de ejemplo, a menudo se emplea el operador & para enmascarar ciertos bits de modo que se puedan comprobar uno o varios en una palabra. Para comprobar si el bit 3 está establecido en uno en un único byte, c, podemos usar:

```
if(c & 0x08)
```

Observe la notación 0x, que precede a una constante hexa; un número que comienza con un cero se considera que está en octal.

En C la selección se maneja mediante la clase usual de sentencia IF, que toma la forma:

```
if(expresión)
  sentencia_1
else
  sentencia_2
```

donde la expresión normalmente es una expresión lógica, pero, como ya hemos visto, también puede ser cualquier expresión que dé un valor entero. Observe que no hay ningún then y, como siempre, la parte else es opcional.

Las sentencias pueden ser simples o compuestas encerradas entre llaves {}. Los ifs anidados pueden ser un problema, tal como sucede en otros lenguajes, pero se aplica la regla general, de modo que cada else se empareja con el if más reciente. Si usted desea cambiar esta situación, puede utilizar llaves.

Las selecciones más complejas, a partir de más de dos alternativas, se pueden tratar mediante la construcción switch, que desempeña la misma función que la sentencia case del PASCAL. El formato de switch es:

```
switch (expresión_entera)
{
case valor_1:sentencia_1;
case valor_2:sentencia_2;
.....
default:sentencia_defecto;
}
```

Los valores de case deben ser constantes y todos diferentes; la sentencia por defecto es opcional. Los valores de case sirven como etiquetas de modo que la ejecución no se transfiera automáticamente al final de la sentencia de case cuando ya se haya cumplido uno de los casos. En cambio, "baja" hasta la siguiente etiqueta. Para impedir esto se puede utilizar la sentencia break para transferir el control a la sentencia que sigue tras el switch. Veamos un ejemplo:

La iteración, o *looping*, se puede tratar de tres formas.

La construcción más utilizada es el bucle while, que asume la forma:

```
while(condición)sentencia;
```

Como es habitual, la sentencia puede ser simple o compuesta (entre llaves).

También es común, aunque no constituye una buena práctica, colocar la mayor parte del cuerpo del bucle en la condición, recordando que la condición puede ser cualquier expresión de enteros. Los siguientes ejemplos representan dos maneras de escribir el mismo bucle para entrar una secuencia de dígitos como caracteres y convertirlos en un número entero, terminando con el primer no dígito:

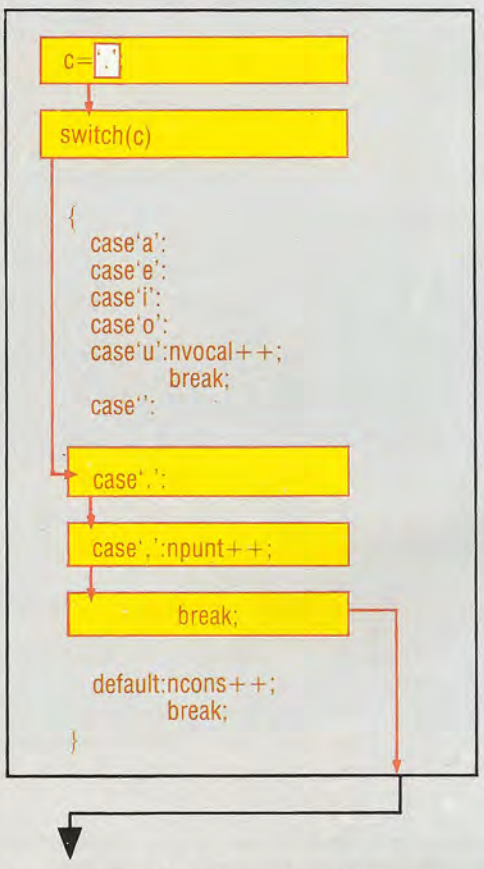
```
int num;
char c;
num=0;
c=getchar();
while(c)='0'&&c(='9')
{
  num=num*10+c-'0';
  c=getchar();
}
```

Observe el modo en que se pueden utilizar los caracteres c y '0' en aritmética. En estos casos se toma automáticamente el código ASCII y los valores se convierten en el tipo int para el cálculo. El segundo procedimiento coloca a la sentencia c en la condición del bucle:

```
num=0;
while((c=getchar())>='0'&& c<='9')
  num=num*10+c-'0';
```

Observe aquí que sólo hay una única sentencia básica a repetir, de modo que no hay necesidad de utilizar las llaves.

La segunda forma de bucle del c es utilizar la sentencia for. Ésta es similar, en algunos sentidos, al bucle FOR...NEXT del BASIC, en cuanto a que hay una inicialización, una condición terminadora (o continuadora) y una "función de paso" que se repite a cada pasada del bucle. En BASIC, como en la mayoría de los otros lenguajes, la inicialización es un valor entero en una variable entera; la condición terminadora es un valor final que debe alcanzar esta variable, y la función de paso tan sólo añade un valor entero a la variable. En c, esta sentencia es mucho más general:



Control de "switch"

La construcción SWITCH actúa de modo muy similar a una construcción ON...GOTO de BASIC. Se evalúa el parámetro de SWITCH y se pasa el control al CASE adecuado. El ejemplo que vemos suma 1 a la variable contadora adecuada (nvocal, npunt o ncons) según el valor de la variable c. Esto proporciona un medio para contar cuántas veces se producen vocales, consonantes y signos de puntuación. Normalmente la variable c sería un valor de entrada, pero aquí, para mostrar el flujo de control en un caso determinado, se le ha asignado el valor '.'. Observe que la "bajada" desde un CASE al siguiente proporciona una forma conveniente de dar cabida a múltiples casos con la misma acción. El "break" final no es estrictamente necesario, pero constituye una buena práctica en caso de que en el futuro usted decida incluir un CASE extra al final

```
for(exp1;exp2;exp3)
   sentencia;
```

donde exp1 representa la inicialización y se lleva a cabo antes de que la sentencia se ejecute por primera vez. La condición de continuación, exp2, y el bucle se repetirán mientras esta expresión permanezca verdadera (o no cero); exp3 se lleva a cabo al final de cada repetición de la sentencia. En BASIC:

```
FOR I=1 TO N
sentencia(s)
NEXT I
```

tiene su equivalente en c con:

```
for(i=1;i<=n;i++)
sentencia;
```

Para ilustrar la flexibilidad de la construcción for y la potencia del lenguaje, podríamos codificar el mismo bucle del ejemplo del bucle WHILE para convertir una serie de caracteres de entrada en un número entero del siguiente modo:

```
for(num=0;(c=getchar())>='0'&& c<='9';
   num=num*10+c-'0');
```

Observe que todo el cuerpo del bucle está incorporado en la condición. Esto suele ser posible en c, pero es discutible si constituye o no un buen hábito. El código producido es, ciertamente, compacto y eficiente, pero también es muy críptico.

En c existe una tercera forma de bucle, que es el equivalente del bucle repeat...until del PASCAL. A diferencia de las otras dos construcciones, la comprobación se realiza al final del bucle en vez de al principio, y toma esta forma:

```
do
   sentencia
while(condición);
```

Esta construcción, sin embargo, se utiliza muy raramente.

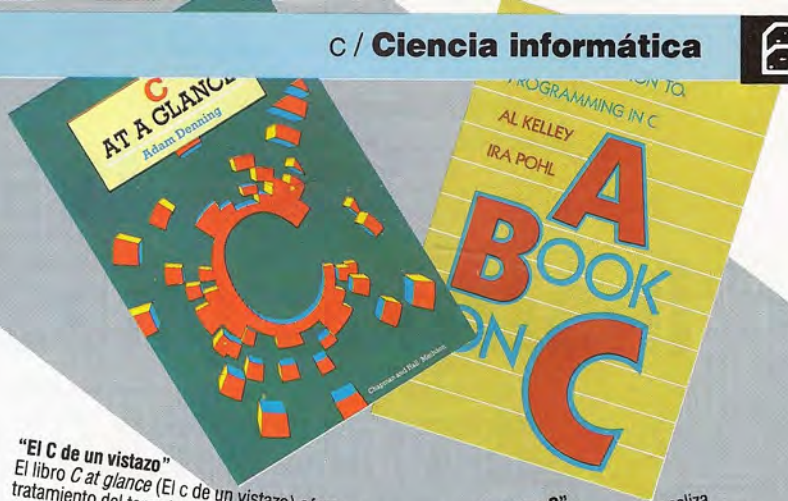
"Break" y "continue"

Existen dos sentencias adicionales que hacen que los bucles resulten bastante más fáciles de usar que en otros lenguajes. Ya nos hemos encontrado antes con la sentencia break. Si se encuentra un break en medio de un bucle, la ejecución se transferirá automáticamente a la sentencia que sigue al final del bucle. El siguiente ejemplo leerá una sucesión de caracteres terminando con un Return, pero también saldrá del bucle si se entra un Control C (código 3 ASCII):

```
while((c=getchar())!='\n')
{
  if c==3
     break
  else
        /*seguir procesando c*/;
}
```

La sentencia trabaja de modo similar, pero simplemente pasa a la siguiente ejecución del bucle en vez de interrumpirse por completo. No es utilizada muy comúnmente.

La existencia de break y continue elimina casi totalmente la necesidad de una sentencia goto. En c hay un goto, junto con un mecanismo para etiquetar sentencias, pero nunca es necesario emplearlo, y hacerlo es una práctica desaconsejable. Después de haber dado fe de su existencia, ya no volveremos a mencionarlo. Si realmente usted desea escribir un programa en c utilizando gotos, le bastará cualquier libro de texto sobre este lenguaje.



"El C de un vistazo"
El libro C at glance (El c de un vistazo) ofrece un tratamiento del tema bastante más en profundidad de lo que sugiere el título. La obra presupone el conocimiento del BASIC por parte del lector y enfoca el c desde este supuesto, haciéndolo fácilmente accesible a la mayoría de programadores de ordenadores personales. Su autor es Adam Denning y ha sido publicado por la editorial británica Chapman and Hall

"Un libro sobre C"
A book on C (Un libro sobre c) analiza exhaustivamente el lenguaje y explora temas relacionados con el mismo, en particular el Unix, con el cual el c está estrechamente relacionado. Sus autores son Ira Pohl y Al Kelley

Imperativo primordial

```
/* Programa para imprimir todos los números
primos menores que 1000*/
main()
{
   int contador_primos=0,límite=1000,divisor,
   número_a_comprobar;
/* Observe la inicialización de las dos variables,
contador_primos y límite, en la sentencia de
declaración*/
   for(número_a_comprobar=2,número_a_
   comprobar<límite,++núm._a_comprobar)
/* bucle principal para comprobar todos los
números entre 2 y 1000*/
   {
      divisor=1;
/* buscar divisores utilizando el operador%(mod)*/
   while(núm._a_comprobar%++divisor !=0);
/* el incr. se realiza antes de la comprobación*/
/* el número es primo si el divisor alcanza el valor
del número_a_comprobar*/
   if (divisor==número_a_comprobar)
/* sumar 1 al contador*/
   {
      ++contador_primos;
/* imprimir número primo, diez en una línea*/
         printf ("%6d",número_a_comprobar);
         if(contador_primos % 10==0)
            printf("/n");
   }
}
printf("\n\nhay %d números primos menores que
,"n/bo°
   contador_primos,límite);
}
```

Un ejemplo de lo mejor
Nuestro programa de muestra cuenta e imprime todos los números primos entre 2 y 1000. Observe el compacto código producido utilizando la instrucción de preincrementado ++ y el formato del bucle FOR

Siga las instrucciones

Iniciamos aquí el estudio detallado de las instrucciones del 68000 de Motorola. Empezaremos por las más sencillas

En los dos capítulos anteriores hemos examinado la forma en que el 68000 direcciona sus operandos y hemos mostrado el empleo de algunas instrucciones (tales como MOVE, ADD y la generalización 'OPCODE'). Demos un paso más para estudiar el conjunto de instrucciones con más detalle, comenzando por las instrucciones de copia de datos, para pasar después a las instrucciones de cálculo que conciernen a la aritmética en sistema de numeración binario.

Vamos a demostrar la utilidad de instrucciones más importantes, y a resaltar todo detalle relevante o posible trampa en su empleo. Si usted pretende programar el 68000 con frecuencia, habrá de considerar, sin embargo, la adquisición o bien del *Manual del usuario del 68000 de Motorola* o bien alguna de las restantes publicaciones que mencionamos en este curso.

Antes de estudiar las instrucciones hemos de examinar con algún detalle el contenido del registro de estado (SR: *status register*). La mitad de este registro contiene los códigos de condición que indican el resultado de la última instrucción ejecutada. Cada código de condición puede ser considerado como memoria de un bit asignada a una condición aritmética particular. Examinémoslos uno por uno:

- *BIT 0: bit de arrastre o bit C:* Este bit se pone a uno cuando la operación aritmética produce un arrastre en el bit más significativo del operando de datos. Por ejemplo:

la suma de	0110 0000
y	1110 0000
da	1 0100 0000

En este caso, se arrastra un 1 del bit más significativo de la suma, que tiene un byte de longitud.

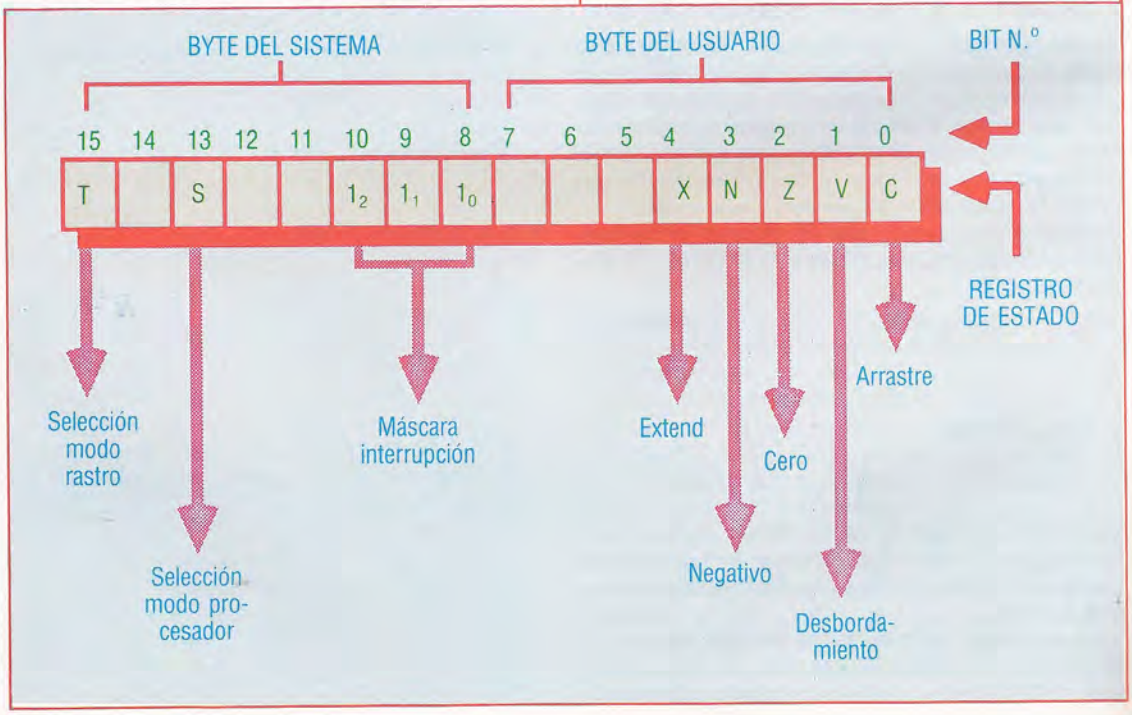
En este ejemplo el arrastre es llevado al bit C del SR, y no al bit menos significativo de la siguiente unidad de datos (en este caso, una palabra). Es de notar también que el arrastre puede ser significativo o no, según lo que se esté haciendo. Si, por ejemplo, se está calculando algún resultado de precisión múltiple (sobre otras unidades del operando de datos), es claro que el bit C debe ser significativo.

- *BIT 1: bit de desbordamiento o bit V:* Se activa cuando el resultado del cálculo no se ajusta al inter-

Estado mayor

El registro de estado de 32 bits se divide en las secciones del sistema y del usuario, cuyo significado de bits se muestra aquí. Obsérvense los bits de estado empleados para señalar mediante flag los modos del procesador: el 68000 puede operar en dos modos, el supervisor y el usuario. La razón de esto es que en un entorno de multiproceso es necesario ser capaz de controlar las diferentes tareas que se ejecutan. Por lo general, las tareas se ejecutan en el modo usuario (que posee su propio puntero de pila y no permite la alteración de los bits de estado del sistema), y el software de sistema (como sería la supervisión de las tareas) se ejecuta en el modo supervisor.

Para comenzar, es frecuente que usted entienda más conveniente el empleo del modo supervisor del 68000, en el cual son legales todos los opcodes, y desentenderse del problema de evitar las instrucciones privilegiadas en el modo usuario. Además, el 68000 posee un modo *trace* (rastro), que es una facilidad muy eficaz a la hora de depurar errores y que permite al usuario rastrear los pasos de las instrucciones, mientras que el 68000 llama automáticamente a la rutina de depuración para el usuario tras cada instrucción



valo de bits de los operandos de datos. Por ejemplo, si se suma un 1 a 32767 (el número entero positivo máximo para una palabra de 16 bits), se obtiene un desbordamiento en los operandos para datos de palabras, y el resultado binario carecerá de significado.

- *BIT 2: bit cero o bit 2:* Se activa cuando el resultado de un cálculo previo es cero.
- *BIT 3: bit negativo o bit N:* Se activa en los resultados negativos.
- *BIT 4: bit de extensión o bit X:* Se emplea en operaciones de multiprecisión, pero en general equivale al bit C (aunque no viene afectado por las instrucciones MOVE).

La instrucción del 68000 para copiar datos es MOVE, que copia de una fuente a un destino. Es una instrucción versátil: se puede emplear con ella cualquier modo de direccionamiento como fuente y casi la mayoría de los modos de direccionamiento como destino (a excepción de los registros de direcciones, los modos relativos al PC y el modo inmediato). Este grupo de modos de direccionamiento se conoce como *modos de datos alterables,* y existe un subgrupo dentro de éste denominado *modos de memoria alterable* (datos alterables menos registros de datos). De ambos grupos nos ocuparemos más adelante.

Volviendo a cómo se ha de emplear la instrucción MOVE, se advertirá que ninguna de las instrucciones MOVE siguientes es lícita:

```
RORG $1000
MOVE D2,MIMI   el PC relativo a MIMI no es válido
MOVE D2,A2     no es válido el registro de
               direcciones como destino
```

Nótese que la instrucción MOVE afecta sólo a los bits N y Z del SR, y que los bits V y C serán puestos a cero.

Para solventar el problema de los registros de direcciones como operandos de destino, se dispone de dos opciones:

- Emplear MOVEA, que toma el contenido del operando fuente y lo copia en el registro destino de direcciones.
- Emplear LEA, que toma la dirección fuente (por lo general, absoluta) y la copia en el registro de direcciones.

Ninguna de estas dos instrucciones afectará a los códigos de condición. Del mismo modo, hay instrucciones especiales para llevar los datos desde y hacia el registro de estado y el puntero de pila del usuario, pero son instrucciones que tienen que ver sobre todo con la programación de sistemas.

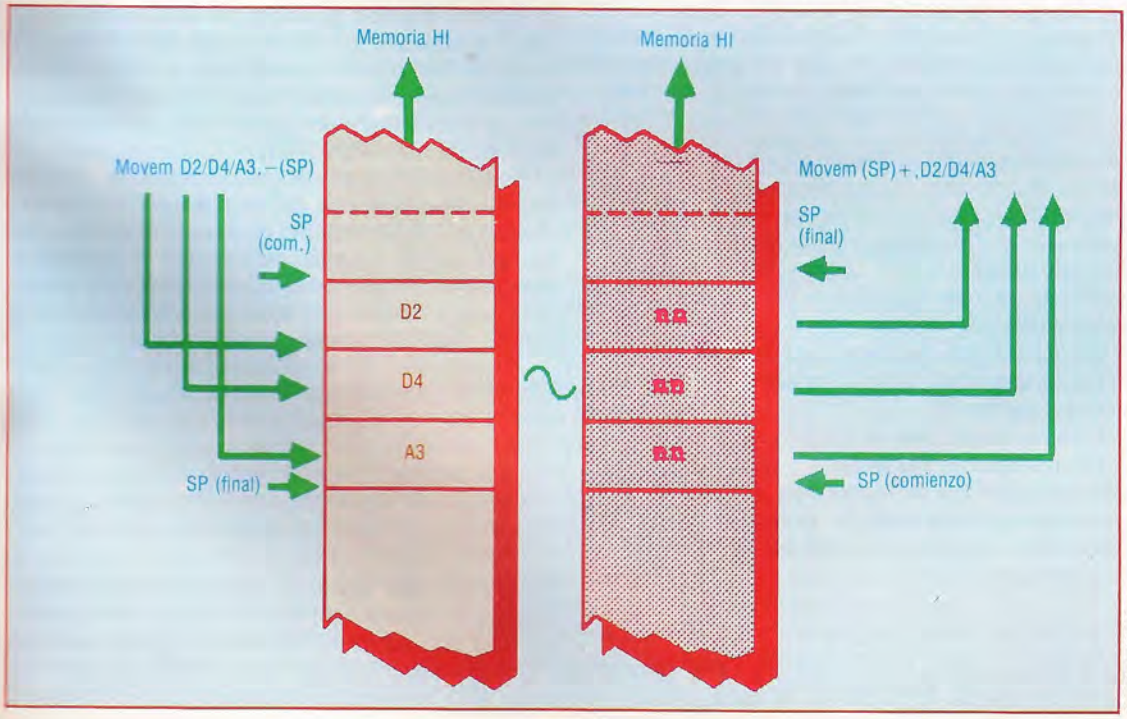
Otra instrucción para copiar datos extraordinariamente poderosa es MOVEM, que nos permite guardar o recuperar cualquier registro declarado (de direcciones o de datos) desde o hacia posiciones de memoria consecutivas. Esto significa que al entrar en una subrutina todos los registros que de otra manera perderían su validez en ella pueden preservarse y a la salida recuperarse. Por ejemplo:

```
entrada MOVE    D2/D4/A3, PAD
                (el código de la subrutina
                emplea D2, D4 y A3)
salida MOVEM    PAD, D2/D4/A3
```

Con ello se guardarán D2, D4 y A3 en PAD a la entrada, y los tres registros se restaurarán a la salida. Los registros se pueden guardar también en la pila. Así, por ejemplo, podríamos tener:

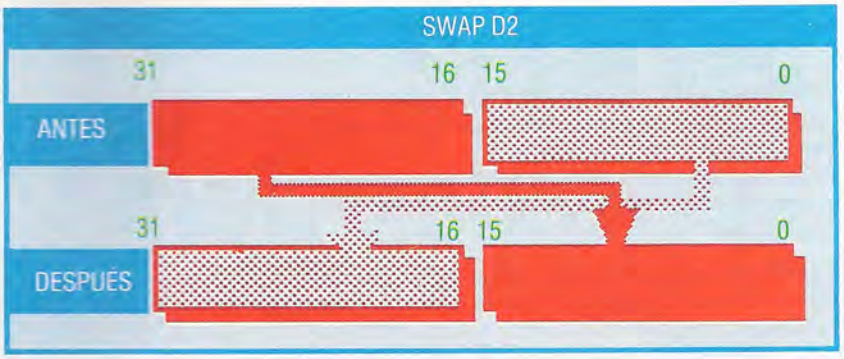
```
entrada MOVEM   D2/D4/A3,–(SP)
                código de la subrutina
salida MOVEM    (SP)+,D2/D4/A3
```

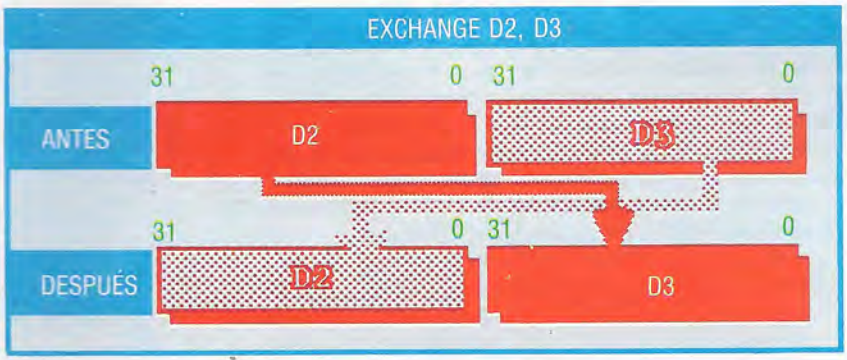
Otra variante de la instrucción MOVE es la instrucción rápida *(quick)* MOVE (o MOVEQ). Es útil cuando se establecen constantes de ocho bits con signo (desde +127 hasta –128) en un registro de datos dentro de una palabra de memoria. Entre los em-



Caroline Clayton

Vayan pasando, por favor
La poderosa instrucción MOVEM nos permite manipular los "listados de registros" en secuencia dentro de una instrucción. Mostramos aquí su uso junto con las instrucciones predecremento y postincremento para poner y hacer salir tres registros hacia y desde la pila. La posición del puntero de la pila es la que se muestra antes y después de cada instrucción





Dos tipos de intercambio
El empleo del almacenamiento de palabras largas puede resultar más bien pesado cuando son necesarios operandos con longitud de un byte o una palabra. SWAP nos ayuda en este problema intercambiando los valores contenidos en los bits del 0 al 15 de un registro con los contenidos en los bits del 16 al 32. Además, EXG (la instrucción de intercambio) permite el intercambio de palabras largas, como se muestra en el esquema inferior

pleos más comunes estará el establecer contadores de bucles dentro de un registro de datos. Por ejemplo:

```
MOVEQ #34,D2
```

establecería 34 en D2 en una palabra de memoria. Nótese que si se quita la Q algunos ensambladores no asumen el modo rápido. Por esto, la instrucción MOVE #34,D2 se codificaría en dos palabras.

Otra instrucción para copiar datos de posible utilidad para operaciones de la pila es PEA *(push effective address:* poner la dirección efectiva... en la pila). Por ejemplo, PEA BETTY llevará la dirección de BETTY a la pila: hará –(SP).

Examinemos finalmente las instrucciones de intercambio. Como indica su significado inglés, SWAP intercambia la posición de palabras enteras dentro de un registro de datos. Así:

```
SWAP  D2
```

hará que la palabra contenida en los bits del 0 al 15 cambie su posición con la palabra contenida en los bits del 16 al 31. Esto puede ser útil si por un supuesto deseamos repetir algún cálculo sobre palabras almacenadas en palabras largas completas dentro de un registro de datos. En este caso el procedimiento sería el siguiente:

- Cargar en D2 la palabra larga completa
- Hacer el cálculo sobre la palabra
- Intercambiar D2
- Hacer cálculo sobre la palabra
- Intercambiar la palabra

Tenemos también la instrucción de cambio EXG, que cambia palabras completas de 32 bits según un determinado modelo. Algunos ejemplos:

```
EXG D2,D3   Intercambio entre registros de datos
EXG A3,A4   Intercambio entre registros de dirs.
EXG D2,A5   Intercambio de datos y direcciones
```

Esta instrucción no es más que una SWAP para palabras completas de 32 bits de tamaño.

Aritmética con enteros

Estas instrucciones forman la base de todos los cálculos aritméticos (ya sean con fracciones, valores reales o de doble precisión) que, básicamente, consistan en sumar números binarios. Aun cuando la aplicación que usted desea no implique cálculo numérico alguno, necesita saber cómo realizar las operaciones aritméticas más sencillas a fin de que, por ejemplo, pueda convertir códigos de caracteres o formar índices de tablas.

La instrucción ADD se limita a sumar la fuente con el destino y almacenar el resultado en el destino. Los objetos de datos pueden ser de cualesquiera tamaños de atributos de datos y son afectados todos los códigos de condición. Por ejemplo, ADD.W D2,D3 sumará el contenido de palabra de D2 con D3 y almacenará el resultado en D3.

Para los datos fuente se pueden emplear todos los modos de direccionamiento, pero es necesario el empleo de una instrucción especial, ADDA, cuando el destino es un registro de direcciones. Así, ADDA D2,A4 sumará el contenido de D2 en A4.

Para el caso de datos inmediatos, está también la instrucción ADDI. Esta instrucción suma el dato inmediato (se permiten todos los atributos) almacenado como una palabra de extensión en el destino (se permiten sólo los modos alterables de datos). Así, ADDI #3423, BETTY sumará 3423 al contenido de BETTY.

A semejanza de MOVE, también para ADDI disponemos de una forma rápida, ADDQ. Pero el empleo de esta última instrucción sólo admite datos dentro del intervalo del 1 al 8. Así, ADDQ 5,D2 sumará 5 al contenido de D2, y la instrucción entera ocupa una palabra.

Hay que hacer una puntualización importante sobre las instrucciones ADD que hemos examinado hasta aquí: y es que no incluyen arrastre alguno en la suma destino. Si queremos esto (en especial para operaciones aritméticas en doble precisión) deberemos emplear la instrucción ADDX. Así ADDX D2,D4 sumará los contenidos de D2 y D4, poniendo el bit de extensión en SR y almacenando el resultado en D4. Supongamos que D2 contenga 0000 0000 y D4 sea 0000 0001 con X=1; pues bien, tras la ejecución de ADDX tendremos en D4 0000 0010. Por el contrario, si hubiéramos empleado ADD, el contenido final de D4 sería 0000 0001.

El siguiente grupo de instrucciones aritméticas que es necesario examinar es el grupo SUB (restar, o *subtract*). Este conjunto es complementario al de las ADD, con las mismas restricciones en el direccionamiento y repercusiones sobre el código de condición. Los ejemplos que damos a continuación son todos válidos al tiempo que típicos:

```
SUB  D2,D3    Restar D2 de D3
              y poner el resultado en D3
SUBA #4,A3    Restar 4 de A3
SUBI #200,D2  Restar 2 de D2
SUBQ #1,D2    Resta rápida D2 menos 1
SUBX D2,D4    Da a D4 el resultado D4-D2-X
```

Es de notar que Motorola no proporciona una instrucción independiente de incremento o decremento. Para realizar estas funciones es preciso recurrir a las versiones rápidas de ADD y SUB (¡después de todo, sólo ocupan una palabra!).

Tony Sleep

Una educación integral

La introducción de los ordenadores en los planes de estudio escolares en los años setenta y ochenta ha sido tema de intenso debate tanto entre los maestros como entre los padres. Aquí analizaremos el cambiante contenido de los cursos de informática durante la pasada década y demostraremos que las tendencias actuales indican que tales cursos tienen un futuro limitado.

Al introducirse la informática en los planes de enseñanza de los países europeos (a finales de los años sesenta) no se consideró necesaria la presencia de los ordenadores en las aulas. Por este motivo no fue sorprendente que los primeros exámenes de la nueva asignatura tuvieran un marcado acento en lo teórico, dejando en segundo término las aplicaciones prácticas. Por supuesto, los alumnos de entonces no tenían acceso a los potentes micros y periféricos de hoy en día, que les hubieran permitido crear sofisticados sistemas "de la vida real".

Durante los años setenta, la informática adquirió notable importancia, llegando a situarse junto a las asignaturas más tradicionales.

Los planes de estudio relativos a informática varían de un país a otro, pero los maestros más "inspirados" desde el punto de vista educativo suelen incluir materias tales como "crear una conciencia del impacto de los ordenadores..." y "desarrollar aptitudes para las comunicaciones y la resolución de problemas..." En los años ochenta, se le restó trascendencia a la enseñanza de programación (especialmente BASIC) como parte fundamental del curso, pasando a utilizar paquetes existentes para resolver problemas. Irónicamente, se podría argumentar que un conocimiento de un lenguaje de programación quizá constituya el aspecto más vocacional de tal curso, si bien obviamente esto dependería de la elección del lenguaje. Además, los aspectos de "informática" de la asignatura (puertas lógicas, sumadores, etc.) desaparecieron casi por completo y la dirección de los cursos en su conjunto se orientó hacia el usuario y se apartó de los aspectos técnicos del hardware.

Conciencia vocacional
La popularidad del estudio de la informática como tal no está creciendo tan rápidamente como pudiera pensarse. Cada vez más los alumnos están abandonando su estudio a medida que alcanzan la edad de dejar la escuela, prefiriendo combinar un conocimiento elemental de la informática con otros conocimientos, tales como contabilidad, diseño y administración

En Gran Bretaña, en 1985, se intentó unificar los diversos planes de estudio existentes (véase recuadro de p. 2242). Una parte esencial del curso implicaba la resolución de problemas: el proyecto práctico que se requiere que complete cada candidato. Como cabía esperar, el formato de este proyecto se ha alterado considerablemente con el correr de los años, tendiendo a una situación en la que se solicita al alumno que plantee y resuelva un problema utilizando un ordenador. Esta solución puede suponer la utilización de paquetes existentes, la escritura de programas nuevos o una combinación de ambos. Un proyecto simple podría ser el de crear un sistema sencillo para conservar los archivos de videos y prestatarios de una biblioteca ambulante.

Se supone que un problema como éste tiene un equivalente en el mundo real y que, por lo tanto, posee un significado para el alumno.

En la práctica, por supuesto, el grado de éxito que alcancen estos objetivos dependerá de la disponibilidad de hardware y software de la escuela. Aun así, el proyecto, como ejercicio para la resolución de problemas de la vida real, es (desde el punto de vista educativo) uno de los argumentos de mayor peso para la inclusión de un curso de informática en las escuelas secundarias.

El plan de estudios típico de informática a nivel avanzado amplía el del nivel básico en la profundización de la comprensión requerida más que en el contenido del curso. En cierta medida, sin embargo, es más difícil justificar el examen a este nivel.

En general las universidades no requieren informática a nivel avanzado como prerrequisito para otorgar un título en esta disciplina, lo que da lugar a la suposición de que tal curso

empieza desde un grado elemental. De modo similar, muchos cursos de ordenador que se imparten en establecimientos de educación superior no dan por sentado ningún conocimiento previo. Es difícil, entonces, discernir con exactitud dónde queda situado el nivel del curso avanzado. Una posibilidad es que las escuelas de educación superior eleven sus expectativas y, como en el caso de otras asignaturas, exijan a los estudiantes algunas cualificaciones previas para acceder a los cursos de informática. A la larga, ello seguramente elevaría los niveles finales de educación.

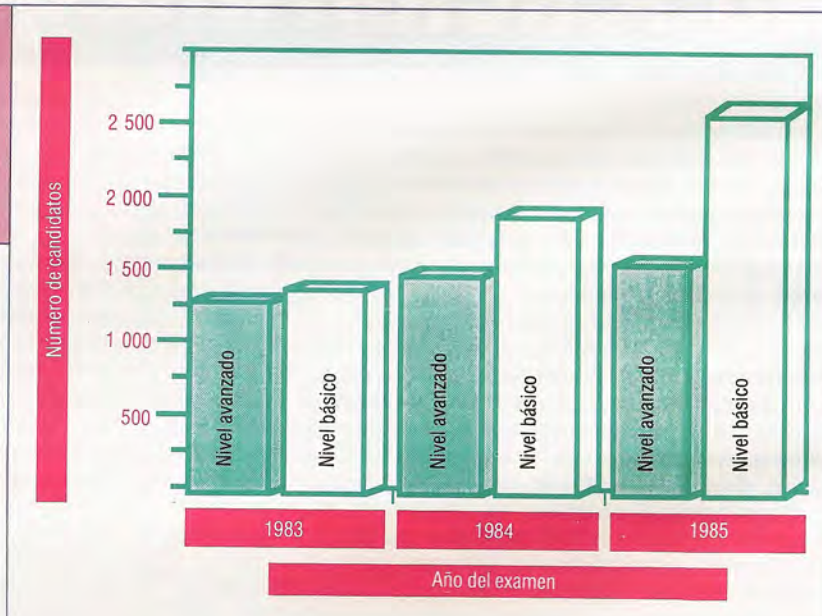
Aparte de los cursos para exámenes estándares, muchas escuelas británicas han introducido cursos de lo que se ha dado en llamar "apreciación de ordenadores" o "tecnología de la informática" (TI). Estos cursos se han creado atendiendo a diversos motivos. En muchos casos, las escuelas comprenden que existe un "vacío en el plan de estudios" entre el creciente uso de los ordenadores por parte de los alumnos de escuelas primarias-superiores y los cursos de las escuelas secundarias. Otros centros de enseñanza reaccionan ante las presiones de los padres para incluir cursos sobre temas que los padres consideran como "el conocimiento del futuro". Además, los cursos desarrollados recientemente CPVE y 16-Plus para alumnos de nivel no avanzado también exigen un componente de "tecnología de la información". El grado con el cual se abordan estas innovaciones (incluso donde se las reconoce) varía enormemente en la práctica. La razón de ser de un curso de esta naturaleza a menudo es bastante confusa, con el resultado de que los alumnos acaban "jugando con el ordenador".

Tecnología de la información

Idealmente, los objetivos de un curso de TI son proporcionar una visión profunda de métodos de base tecnológica para reunir, almacenar, procesar y distribuir información en todas sus formas (vocal, textual, iconográfica, numérica, etc.) Centrándose en métodos en los que se pueda utilizar un ordenador, se persigue que todos los alumnos tengan confianza en el uso del ordenador para reunir, almacenar y acceder a la información. En el proceso didáctico se suelen emplear, por ejemplo, paquetes de recuperación de información, procesadores de textos o sistemas de videotex.

Es evidente que las aptitudes requeridas son importantes para todo el aprendizaje; ya es opinión generalizada que un curso de tecnología de la información en los primeros años de la escuela secundaria es tan esencial como estudiar lenguaje y literatura o matemáticas. A medida que las escuelas dispongan de paquetes de software eficaces, no sólo aumentará la gama de las actividades posibles, sino que también se hará menos hincapié en la enseñanza de programación, arquitectura del ordenador, etc. A consecuencia de estos factores, bien puede ser que desaparezca la informática como materia de examen. Esto podría abrir las puertas a la posibilidad de cursos más específicamente vocacionales, como tratamiento de textos, etcétera. Por último, quizá veamos a los ordenadores en las escuelas como un "departamento de servicios", utilizado indistintamente por la administración, los maestros y los alumnos, lo que posiblemente redundaría en ventajas para todos.

Estudios de informática en Gran Bretaña
Estas cifras, dadas a conocer por el London School Examination Board, muestra el sorprendente crecimiento del número de alumnos que cursan informática en nivel básico. Por el contrario, el aumento en la cantidad de estudiantes que se presentan a exámenes del nivel avanzado ha sido mucho menos acusado. Con esto cobra fuerza la hipótesis de que, al terminar la escuela, los estudiantes ven el conocimiento básico de la informática como un valioso complemento de otros estudios, pero no necesariamente deseable como objeto de una especialización



Caroline Clayton

Intento unificador

Plan unificado de estudios de informática en un establecimiento educacional de Londres:

Aptitudes para la comunicación
— Representación e interpretación de algoritmos
— Necesidad y contenido de una documentación adecuada

Informática básica
— Hardware (CPU, memoria, características funcionales de dispositivos de entrada y salida y almacenamiento de apoyo)
— Arquitectura del ordenador (registros, ciclo buscar-ejecutar)
— Representación para la máquina (enteros, caracteres, instrucciones)
— Software (lenguajes de alto y bajo nivel, traductores, errores y ayudas para el diagnóstico, sistemas operativos, paquetes de aplicaciones)

Proceso de la información
— Principales aspectos del análisis de sistemas
— Captura de datos, validación, transmisión y validación
— Métodos para reducir los errores
— Especificación y diseño de archivos
— Especificación de documentos de E/S
— Descripción de aplicaciones utilizando sistemas de diagramas de flujo
— Métodos para la recuperación y seguridad de archivos
— Métodos para el proceso (por lotes, tiempo real, distribuido, etc.)
— Limitaciones e idoneidad del uso del ordenador en aplicaciones dadas
— Efectos sociales tales como intimidad y acceso a los datos

Resolución de problemas mediante el uso de un ordenador
— Capacidad para describir un problema, captar los datos, usar un ordenador para procesar estos datos y presentar los resultados

Función formal

Examinaremos la estructura y el papel de las funciones en c y analizaremos un programa de ejemplo

El lenguaje c se basa fundamentalmente en el concepto de *función*. No es un lenguaje funcional como tal, porque la forma en que opera cada función se define de modo procedural. No obstante, en c virtualmente todo se define desde el punto de vista del concepto *función*.

Lo que queremos significar por *función* en este contexto es un proceso al que se le pasan ciertos valores (los argumentos o parámetros) y que utiliza esos valores para producir un único valor que es devuelto. Los valores no han de ser necesariamente tipos básicos, como enteros o reales, sino que pueden ser tipos estructurados tales como matrices.

Hay otros varios conceptos esenciales que debemos considerar: los bloques, por ejemplo. En c, un *bloque* es una sección autocontenida de código, indicada mediante llaves ({ }); en PASCAL, a fines de comparación, un bloque se encierra entre sentencias begin...end. Asimismo, necesitamos valorar la idea del *ámbito*, es decir, la región en la cual está disponible para utilizarse. Son pocas las versiones de BASIC que admiten variables locales, de modo que es posible que muchos programadores de BASIC no estén habituados a la idea de tener que restringir el uso de ciertas variables a ciertas porciones de un programa.

En c, las variables se pueden declarar a lo largo de un programa y, en consecuencia, es especialmente importante saber cuándo y dónde se puede aludir a ellas.

Las reglas básicas para la declaración de variables en c son:

- Cuando se declara una variable al comienzo de una definición de función, su ámbito es la totalidad del cuerpo de función, incluyendo cualquier otra función que pudiera definirse dentro del cuerpo de la función original. En particular, dado que el programa se compone de la función main(), las variables declaradas al comienzo tendrán como ámbito la totalidad del programa, siendo, por tanto, variables *globales*.

  En el diseño de programas, es un aspecto de creciente importancia el que tales módulos de un programa sean totalmente autocontenidos y utilicen sólo variables declaradas localmente y parámetros formales. Si la ejecución de una función implica otros datos aparte de los de las variables locales, entonces se dice que esto es un *efecto lateral*. Por ejemplo, la operación de un dispositivo de entrada/salida por lo general es un efecto lateral, como lo es el uso de toda variable global. Algunos efectos laterales son beneficiosos, es decir, no tienen efecto alguno sobre la ejecución de otras partes del programa; pero otros son nocivos y su uso se debe evitar siempre que sea posible. Esto se aplica especialmente al uso de variables globales.
- Cuando se declara una variable dentro de un bloque, su ámbito se extiende solamente al resto de ese bloque.

La declaración de variables es especialmente importante porque, como veremos más adelante, forma parte de la "filosofía" del c el que un programa se haya de construir en módulos. Cada uno de estos módulos suele existir en un archivo separado, y las reglas que gobiernan el hecho de que una variable pueda tener un ámbito que incluya funciones definidas en un archivo diferente son algo complejas. Más adelante analizaremos estas reglas.

Hay otras dos formas de pasar argumentos o parámetros a una función:

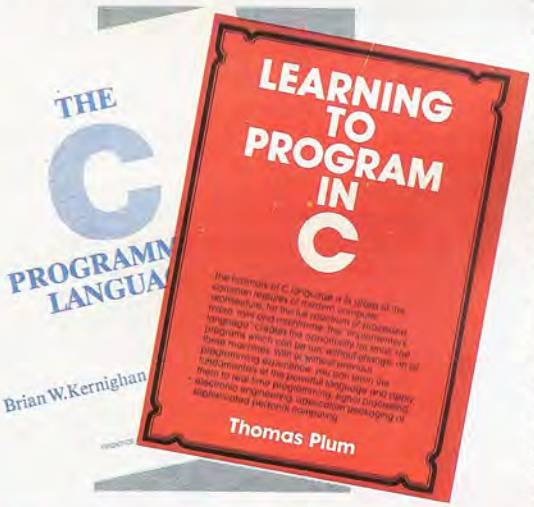
- *Valor:* Donde se crean variables enteramente nuevas. Estas variables son locales a la función y se inicializan en los valores pasados por la rutina de llamada.
- *Referencia:* Donde se pasa la dirección de la variable que contiene al parámetro. De este modo, se utiliza la misma variable en la función y la rutina de llamada.

En c, todos los parámetros se pasan por valor; pero, como veremos más adelante, el lenguaje posee amplias facilidades para manipular direcciones o punteros de variables y, por tanto, se puede obtener la llamada por referencia.

La sintaxis formal para una definición de función es:

```
tipo_función nombre_función (lista
   _argumentos_formales);
declaraciones_variables;
{
      cuerpo_de_la_función;
}
```

El tipo_función se puede omitir si el valor devuelto es int. La lista_argumentos_formales es una lista de variables de los tipos adecuados en las que se colocarán los valores cuando se utilice la función. A la función se la llama utilizando:




```
nombre_función(lista_argumentos_actuales)
```

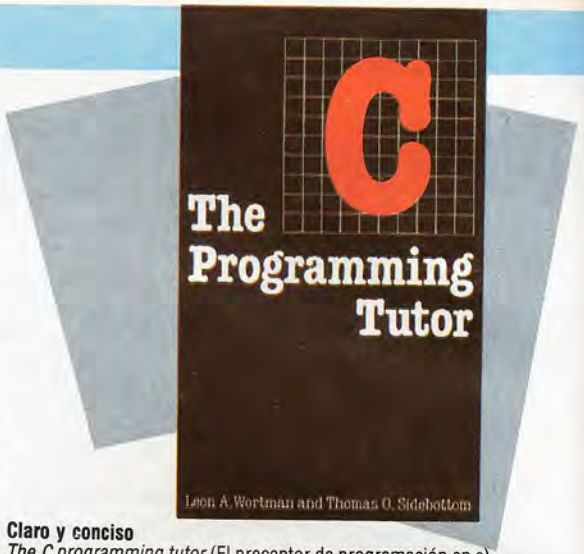
en cualquier punto donde sea adecuado utilizar un valor del tipo que devuelva la función. No obstante, si el tipo del valor devuelto es distinto de int, entonces el nombre de la función debe haberse declarado como una variable del tipo adecuado antes de poder llamarla. La lista de argumentos actuales es la lista de aquellos valores, o variables que contienen aquellos valores, que se han de pasar a la función y colocar en los argumentos formales. Los argumentos reales y formales, por supuesto, deben coincidir en tipo y posición en las dos listas.

El último detalle que debemos considerar antes de escribir una función es el uso de la sentencia return para dar el valor que se ha de devolver a la rutina de llamada. Puede haber una cantidad cualquiera de estas sentencias, pero se debe ejecutar una antes de salir de la función.

El siguiente programa entrará un conjunto de números reales e imprimirá los miembros mayor y menor del conjunto. El programa utiliza tres funciones. Una de ellas, imprimircabecera, no tiene argumentos ni sentencia return, lo que significa que el valor devuelto por la función no está determinado. Puesto que la rutina de llamada no requiere el valor devuelto, éste simplemente se "descarta". Esta función sólo tiene efectos laterales, pero todos ellos son beneficiosos y, por lo tanto, no hemos de preocuparnos por ellos.

El programa también utiliza la función de biblioteca scanf, que realiza la entrada formateada de un modo muy similar a la empleada por printf para salida. El primer parámetro para la función es una serie que contiene información sobre el formato de los valores a entrar, utilizando la misma técnica que printf. Los otros argumentos de la función scanf son las variables en las cuales se han de colocar los valores. No obstante, dado que los parámetros sólo se pueden pasar por valor, no se los puede utilizar para devolver valores a la rutina de llamada. Por consiguiente, es necesario que estos parámetros sean direcciones de variables en vez de las variables propiamente dichas. Cada variable nombrada en la llamada a scanf debe ir precedida por el operador de dirección, &. Más adelante veremos otros usos de este operador.

```
# include<stdio.h>
main ( )
{
   int contador,tamaño_de_la_entrada,
   double máspequeño,másgrande,número,mín( ),
   máx( );
/*observe la declaración de las funciones mín y máx*/
/*observe también que la declaración dentro del
bloque hace que las variables no estén disponibles
fuera del bloque*/
   imprimircabecera( );
/*función llamada, se supone que el valor devuelto es
int y se lo "descarta", es decir, no es usado en ningún
sitio*/
   printf("\ntamaño de la entrada:");
   scanf("%d",&tamaño_de_la_entrada);
/*  averiguar cuántos números se van a entrar*/
   printf("Ahora entre %dnúmeros reales:\n",
   tamaño_de_la_entrada);
```



Claro y conciso
The C programming tutor (El preceptor de programación en c), de Wortman y Sidebottom, constituye un excelente texto para principiantes. Escrito en un estilo claro y conciso, introduce y explica temas complejos sin confundir al lector y es especialmente recomendable para aquellos lectores que carezcan de experiencia en lenguajes de programación, exceptuando el BASIC. Este libro ha sido publicado por la editorial británica Prentice-Hall

Biblioteca c
La editorial Prentice-Hall ha tenido una notable influencia en la creciente popularidad del c al mantener una amplia lista de publicaciones dedicadas al tema, incluyendo el texto *The C programming language* (El lenguaje de programacion c), original de Kernighan y Ritchie. Por su alto precio, este escueto volumen no está al alcance de muchos usuarios; no obstante, hasta que se autorice un estándar internacional, seguirá siendo la especificación definitiva del lenguaje. Otro texto importante editado por Prentice-Hall es *Learning to program in C* (Aprendiendo a programar en c), de Thomas Plum

```
  /* tomar el primer número que será el másgrande y
máspequeño actual*/
    scanf("%lf",&número);
    másgrande=máspequeño=número;
  /* ahora tomar los otros números*/
    for(contador=2;contador<=tamaño_de_la_
      entrada;
      ++contador)
      {
        scanf("%lf",&número);
        máspequeño=mín(máspequeño,número);
        másgrande=máx(másgrande,número);
      }
      printf("\nmáspequeño es &f\nmásgrande es
      %f\n",máspequeño,másgrande);
}
/* ahora vienen las definiciones de funciones*/
imprimircabecera ( )
/* no se necesita tipo, se supone que es int*/
{
  printf("\n%s\n%s\n%s\n",
    "Entre primero el tamaño del conjunto de
        números",
    "después entre esa cantidad de números
        reales.»,
    "Se visualizarán el más grande y el más
        pequeño");
}
double mín(x,y);
double x,y;
/* observe que se declaran todos los parámetros*/
{
  if(x<y)
    return(x);
  else
    return(y);
}
double máx(x,y);
double x,y;
{
  if(x>y)
    return(x);
  else
    return(y);
}
```

El preprocesador c

Muchos compiladores, en diversos lenguajes, configuran *directivas de compilación* incorporadas en un programa. Éstas son ciertas líneas que se reconocen como instrucciones dirigidas al compilador y no como sentencias del lenguaje. El c lleva esta idea un paso más hacia adelante que los otros lenguajes, y todos los compiladores de c incorporan un *preprocesador* que reconoce ciertas *líneas de control* como instrucciones para alterar consiguientemente el programa antes de presentárselo al propio compilador. Las líneas de control se distinguen mediante un signo # al comienzo. Asimismo, las directivas del preprocesador no necesitan ir seguidas de punto y coma, puesto que no se consideran como parte del programa.

La directiva del preprocesador # include <nombrearchivo> o # include "nombrearchivo" indica al preprocesador que incluya en este punto el contenido del archivo mencionado. Puede que en algunos sistemas no exista diferencia entre el uso de paréntesis en ángulo, <>, y comillas; pero otros sistemas pueden hacer una diferencia en los lugares que se buscan con el fin de hallar los archivos. La convención habitual es que los paréntesis en ángulo se emplean para archivos del sistema, mientras que las comillas se utilizan para los propios archivos del usuario. Las funciones printf y scanf que hemos estado empleando están incluidas en un archivo estándar del sistema que se denomina stdio.h, de modo que todos los programas que las utilicen deben incluir la directiva # include<stdio.h>, como al comienzo de nuestro programa.

Se pueden anidar estas llamadas de modo que uno de los archivos que se incluya posea también un # include para incorporar otro archivo.

La directiva # define permite una forma limitada de macrosustitución. En su forma más simple, permite la definición de una constante; por ejemplo, # define EOF–1 haría que el identificador EOF se reemplazara por –1 cada vez que apareciera en el archivo del programa. Esta facilidad hace que resulte mucho más sencillo realizar cambios en valores "constantes" en el caso de que surja la necesidad. Otros ejemplos son:

```
# define PI3.14159
# define EQ==
```

La convención generalmente aceptada es que los identificadores definidos de este modo se escriben en mayúsculas, reservando las minúsculas para las variables normales. Se pueden incluir parámetros en una macrodefinición, como en

```
# define SUMSQ(x,y)((x * x)+(y * y))
```

que tiene un efecto similar a las definiciones de función en BASIC. Por ejemplo, la ocurrencia de un SUMSQ(3,4) se reemplazaría por ((3 * 3)+(4 * 4)). Observe la inclusión de paréntesis como medida de seguridad; la macro se podría utilizar en una situación en la que los paréntesis fueran necesarios, lo que significa que siempre es mejor incluirlos.

Existe una directiva # undef que hace que el preprocesador no realice más sustituciones para la macro mencionada después de haberla encontrado.

Otra de las directivas comúnmente utilizadas es la construcción # if... # else... # endif, que permite la compilación condicionada. Si la expresión de constante que sigue a la sentencia # if se evalúa como verdadera (no cero), entonces se compilan las líneas de código que siguen tras la sentencia. Si la expresión de constante se evalúa como falsa (cero), entonces se compilan las líneas que siguen al # else.

Un ejemplo de su utilización se puede observar durante el desarrollo de un programa, cuando puede ser de ayuda incluir muchísimas sentencias printf adicionales al objeto de llevar el registro de lo que está sucediendo. Lo que podemos hacer es colocar directivas de la forma # define DEBUG 1, y siempre que utilicemos un printf adicional colocamos:

```
# if DEBUG
    printf(valores...)
# endif
```

Luego, cuando hemos acabado la depuración, tan sólo necesitamos cambiar las directivas por # define DEBUG 0, y al volver a compilar nuestro programa éste ya no incluirá esas sentencias.

Oro macizo

SAMNA WORD III

Desplazamiento de palabras

★★★★★

Proporciona desplazamiento automático y ajuste de línea completa.

Movimiento de bloques

★★★★★

Tiene todas las instrucciones estándares, incluyendo el movimiento de columnas.

Ayuda en pantalla

★★★★★

El programa es uno de los más amables que hay disponibles desde este punto de vista.

Pantalla de 80 columnas

★★★★★

La pantalla por defecto establece los márgenes en 72 columnas, pero se puede alterar para usar las 80 columnas completas.

Contador de palabras

No tiene.

Buscar/reemplazar

★★★★★

Se pueden utilizar hasta 30 caracteres, con numerosas opciones de búsqueda.

WYSIWYG

★★★★

Aunque el *Word III* posee facilidades muy útiles, tales como el Zoom, le faltan otras corrientes (como visualizar el subrayado en pantalla).

Facil. para correspondencia

★★★★★

Permite almacenamiento de bases de datos y correspondencia automática.

Verificador de ortografía

★★★★

La utilidad Proof es rápida y amable, aunque quizá los usuarios británicos se impacienten a causa del diccionario norteamericanizado.

Tipos de letra disponibles

★★★

Si bien el *Word III* soporta subíndices y negritas, no dispone de tipos alternativos.

Unión de archivos

No tiene.

La última palabra
El *Samna Word III* es un sofisticado programa para tratamiento de textos que soporta facilidades de las que carecen los sistemas más económicos. Aparte de las funciones usuales de inserción y manipulación del texto que ofrecen la mayoría de los programas, el *Word III* posee complejas (aunque amables) facilidades para verificación de ortografía, correspondencia automática y formateo del texto. *Word III* posee una facilidad denominada Zoom, que da al usuario una idea del aspecto final de la página

Examinemos de cerca el que se considera el "Rolls Royce" del tratamiento de textos

Como casi todos los paquetes para tratamiento de textos muy caros, el *Samna Word III* se ejecuta en la gama de ordenadores IBM PC, siempre que operen bajo la versión 2.0 de MS-DOS o superior. Ello se debe a que el *Word III* implementa directorios de archivos y "vías" que sólo se introdujeron en el DOS 2.0. El paquete se compone de un manual de introducción, trazados del teclado Samna para varios lenguajes diferentes, un manual más detallado que explica cada una de las instrucciones, y cinco discos flexibles que contienen el programa propiamente dicho.

Los discos contienen varias utilidades diferentes. El programa principal está retenido en dos de los discos y otros discos retienen las rutinas de la impresora y el diccionario. El quinto, que en realidad no forma parte del programa principal, es el disco de entrenamiento, que contiene numerosos programas de enseñanza cortos que guían al usuario a través del programa.

Una vez instalado en varios discos, el programa total ocupa más de 700 Kbytes de espacio de disco en 45 archivos del usuario: una cantidad considerable tratándose de un solo programa. La ejecución del programa presenta al usuario una pantalla Scratchpad, que informa que actualmente no hay cargado ningún archivo con nombre. A los lados de la pantalla hay barras coloreadas indicando las posiciones de los márgenes, y arriba de éstas otra barra que muestra los ajustes de tabulación y márgenes por defecto. Arriba de la pantalla hay indicadores que muestran la posición actual del cursor, el nombre del archivo y un aviso READY!.

Una de las grandes ventajas del IBM PC y sus compatibles es la gran cantidad de teclas programables que se han puesto a disposición de quienes desarrollan software. Los programadores del *Word III* han sacado de ello el máximo partido y no sólo han empleado las 10 teclas de función para proporcionar una serie de utilidades para tratamiento de textos, sino que también han reprogramado algunas de las otras teclas con la misma finalidad.

El teclado numérico, por ejemplo, se emplea como un grupo de cursor (una función utilizada normalmente por el teclado IBM), pero algunas de las teclas, como 7, 9 y 1, se han utilizado para desplazar el cursor a lo largo de una palabra, frase y párrafo, respectivamente. Por otra parte, la tecla Scroll Lock se ha reprogramado como la tecla Mark, que se utiliza para definir bloques que se pueden manipular dentro del texto. Las teclas de función se han programado para realizar las funciones más comúnmente utilizadas en tratamiento de textos, tales como el subrayado y el ajuste.

El *Word III,* sin embargo, es mucho más amplio que los otros programas que hemos analizado en esta serie. La llamada a una función, por ejemplo, a menudo conduce a otra pregunta que, a su vez, solicita al usuario exactamente sobre qué zona del texto ha de operar dicha función, como la totalidad del documento, el párrafo o todo cuanto haya después de la posición actual del cursor.

Los programadores de Samna han intentado conseguir los objetivos, con frecuencia incompatibles, de máximas flexibilidad y sencillez de uso. A modo de ejemplo, el *Word III* no posee facilidad de inserción automática. Cuando el usuario desee insertar texto en medio de un documento, pulsando

la tecla de inserción se visualizará un espacio adecuado. En el *Tasword II*, esta técnica requiere que el usuario vuelva a alinear el párrafo después de insertar el texto. En el *Word III*, sin embargo, con tan sólo volver a pulsar la tecla de inserción el cursor se libera de su posición y vuelve a alinear automáticamente el texto, con lo que se obtiene la mejor combinación.

El *Word III* posee una facilidad de ayuda sumamente útil. El usuario puede llamarla pulsando la tecla Help situada en el ángulo superior izquierdo del teclado de máquina de escribir, que visualiza la gama de opciones que hay disponibles. Por el contrario, si el usuario duda antes de llamar a una función, la pantalla de ayuda se visualizará a sí misma de forma automática.

Varias de las teclas de función están programadas para llevar a cabo una amplia gama de operaciones diversas. Entre las más útiles está la tecla Do. Entre las utilidades que se pueden entrar a partir de ella figuran muchas de las funciones de archivo normales, como borrar y copiar, pero la tecla también se utiliza para acceder a varias operaciones de las que carecen los procesadores de textos más económicos.

Cuando se escribe un documento largo, la pantalla sólo es capaz de visualizar una ventana. No obstante, si usted quisiera ver qué aspecto tendrá el documento entero en una página, la selección de la función Zoom visualizará el trazado de una página con barras mostrando dónde están posicionados los títulos y los párrafos. Esto le permite producir un trazado más atractivo sin tener que imprimirlo varias veces.

Otra de las funciones de la tecla Do supera otra limitación del efecto de ventana. Cuando usted está digitando numerosas columnas en un documento muy ancho, por ejemplo, quizá sea necesario aludir a una columna que no esté visualizada en la pantalla. La instrucción Fold "plegará" el documento automáticamente de modo que en la misma pantalla se puedan visualizar las dos caras de la página.

En la actualidad, uno de los requisitos de los paquetes de gestión es que sean compatibles no sólo con el IBM PC, sino también con otro software, en especial con el tan utilizado *Lotus 1-2-3*. Con el objeto de mantener la compatibilidad con este último, el *Word III* puede traducir archivos ASCII desde disco y visualizarlos en formato Samna. De este modo, dentro del *Word III* se pueden visualizar y manipular documentos del *Lotus*.

Todo procesador de textos que aspire a un lugar de privilegio en el mercado de gestión debe contar con facilidad para correspondencia automática y verificación de ortografía, estando ambas implementadas en el *Word III*. El verificador de ortografía, conocido como la instrucción Proof, es uno de los más amplios existentes actualmente en todos los paquetes.

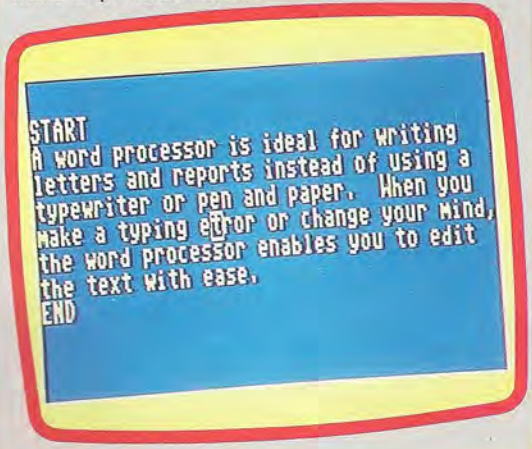
Al igual que algunas utilidades del programa, el verificador de ortografía permite escoger entre numerosas opciones, especificando qué es lo que se desea "corregir". Una vez decidido esto, el ordenador revisa entonces cada palabra, cotejándola con las que tiene retenidas en su propio diccionario. Cuando encuentra una palabra que no reconoce, el ordenador visualiza una serie de opciones, incluyendo numerosas ortografías alternativas.

La utilidad de correspondencia automática implementada en el *Word III* se denomina Automatic Merge, y consiste en una pequeña base de datos en la que usted puede almacenar un cierto número de campos, tales como nombre, apellido y dirección. Éstos se pueden luego disponer en el sitio adecuado en un documento estándar e imprimir. Asimismo, el programa admite *máscaras* para buscar un determinado registro.

El *Word III* de Samna es un paquete para tratamiento de textos muy potente que resiste muy bien la comparación con numerosas máquinas exclusivas para tratamiento de textos. No obstante, sigue vigente la pregunta de si el paquete vale su alto precio. Si su empresa requiere un paquete para tratamiento de textos tan amplio, con facilidades que no encontrará en ningún otro, indudablemente la respuesta es afirmativa.

La letra pequeña

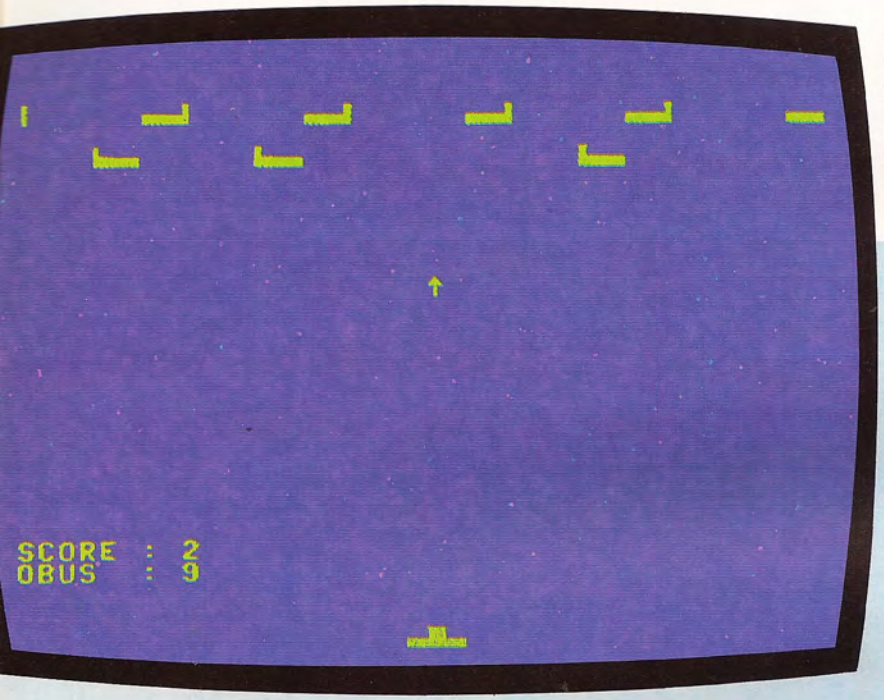
En el extremo opuesto de la escala que ocupa *Samna Word III* está *Mini Office*, de Database Publications. Este último no sólo incluye un programa para tratamiento de textos sino también utilidades de base de datos, gráficos y hoja electrónica. Se ejecuta en el BBC Micro, Electron, Commodore 64 y gama Amstrad. Por supuesto, el paquete para tratamiento de textos posee muy pocas de las sofisticadas funciones del *Word III* y, así y todo, el programa contiene numerosas facilidades de las que carecen programas más caros. Aunque la facilidad para tratamiento de textos en el *Mini Office* es básica, muchos usuarios que no requieran complejas funciones para formateo de la impresión encontrarán que el programa contiene todo cuanto necesitan para escribir cartas y otros documentos. El programa es activado por menú, que permite que el usuario seleccione varias opciones, a través de las teclas de función, para manipular el texto. Entre éstas se incluyen facilidades tales como cargar, guardar e imprimir documentos. La pantalla de textos soporta inserción/supresión y desplazamiento automáticos, si bien el programa no permite el ajuste de líneas. La parte superior de la pantalla proporciona información útil, incluyendo un contador de palabras, la cantidad de caracteres disponibles restantes y un reloj que le indica cuánto tiempo lleva digitando. Asimismo, el programa permite transcribir texto de una a otra parte de un documento con sólo copiar caracteres a partir de una posición del cursor previamente definida y hasta la posición actual



Liz Heaney

Defensa antiaérea

Este juego, también conocido como "D.C.A. (Defensa contra aviones)", invierte los papeles asignados en "Bombardeo aéreo". Ha sido escrito para el Commodore 64



Su misión es manejar las defensas contra aviones e intentar abatir los aviones que vuelan sobre su posición. Para disparar hay que usar una tecla cualquiera. Inicialmente dispone de quince obuses. Si consigue abatir diez aviones, obtendrá una bonificación de diez puntos y diez nuevos obuses adicionales.

```
  5 REM ***********************************
 10 REM *              D.C.A              *
 15 REM ***********************************
 20 GOSUB 1000
 30 GOTO 720
100 A$=RIGHT$(A$,39)+LEFT$(A$,1)
110 B$=RIGHT$(B$,1)+LEFT$(B$,39)
120 PRINT A$
140 PRINT B$;HH$;
200 GET X$
210 IF X$<>" " AND M=MI THEN
    M=MI-80:NM=NM-1:GOTO 230
220 IF M<>MI THEN M=M-80
230 IF M<>M2+80 THEN 250
240 IF PEEK(M2-1)<>32 THEN 500
250 IF M<>M1+80 THEN 270
260 IF PEEK(M1+1)<>32 THEN 600
270 IF M=MI THEN 310
280 POKE M,CM
290 POKE M+N,MC
300 IF M+80<>MI AND M<>M1 THEN POKE
    M+80,CR
310 IF M<1064 THEN M=MI:GOTO 720
320 IF NM<1 AND M=MI THEN 4000
330 GOTO 100
500 B$=LEFT$(B$,17)+01$+RIGHT$(B$,16)
550 GOTO 650
600 A$=LEFT$(A$,17)+01$+RIGHT$(A$,16)
650 POKE M-80,160
660 POKE (M-80)+N,2
670 POKE M+80,CR
680 S=S+1
700 IF S>1 AND INT(S/10)=S/10 THEN GOSUB
    800
710 M=MI
720 FOR I=1 TO 20
730 PRINT CHR$(17);
740 NEXT I
750 PRINT "PUNTOS[1SPC]:";S
760 PRINT "OBUS[2SPC]:";STR$(NM);
    "[1SPC]"
770 PRINT HH$;
780 GOTO 310
800 A$=A1$:B$=B1$:NM=NM+10
830 FOR I=1 TO 500:NEXT I
850 S=S+10
860 RETURN
1000 A$=" ":B$=" ":HH$=CHR$(19)
1030 M1=1044:M2=1124:MI=2004
1060 M=MI:CM=30:MC=5:CR=32:
     NM=15
1110 01$=" ":N=54272:X$=" ":S=0
1200 FOR I=1 TO 7
1210 01$=01$+"[1SPC]"
1220 NEXT I
1230 FOR I=1 TO 40
1240 READ A,B
1250 A$=A$+CHR$(A)
1260 B$=B$+CHR$(B)
1270 IF I/8=INT(I/8) THEN RESTORE
1280 NEXT I
1290 A1$=A$:B1$=B$
1500 PRINT CHR$(147);
2000 FOR I=M-1 TO M+1
2010 POKE I,98:POKE I+N,5
2030 NEXT I
2040 POKE M,160
2050 POKE 53280,0
2060 POKE 53281,6
2070 PRINT CHR$(158);
2080 RETURN
4000 PRINT CHR$(147)
4010 IF S>RE THEN RE=S
4020 FOR I=1 TO 4
4030 PRINT
4040 NEXT I
4050 PRINT TAB(13)"PUNTOS[1SPC]:";S
4060 FOR I=1 TO 4
4070 PRINT
4080 NEXT I
4090 PRINT TAB(11)"PUNTUACION[1SPC]
     MAXIMA[1SPC]:";RE
4100 FOR I=1 TO 4
4110 GET X$
4120 PRINT
4130 NEXT I
4140 PRINT TAB(13)"OTRA[1SPC]?"
4150 GET X$
4160 IF X$=" " THEN 4150
4170 IF X$<>"N" THEN 20
4180 END
10000 DATA 32,182,162,162,162,162,181
10010 DATA 32,32,32,32,32,32,32,32,32
```

© SYBEX

Interiores

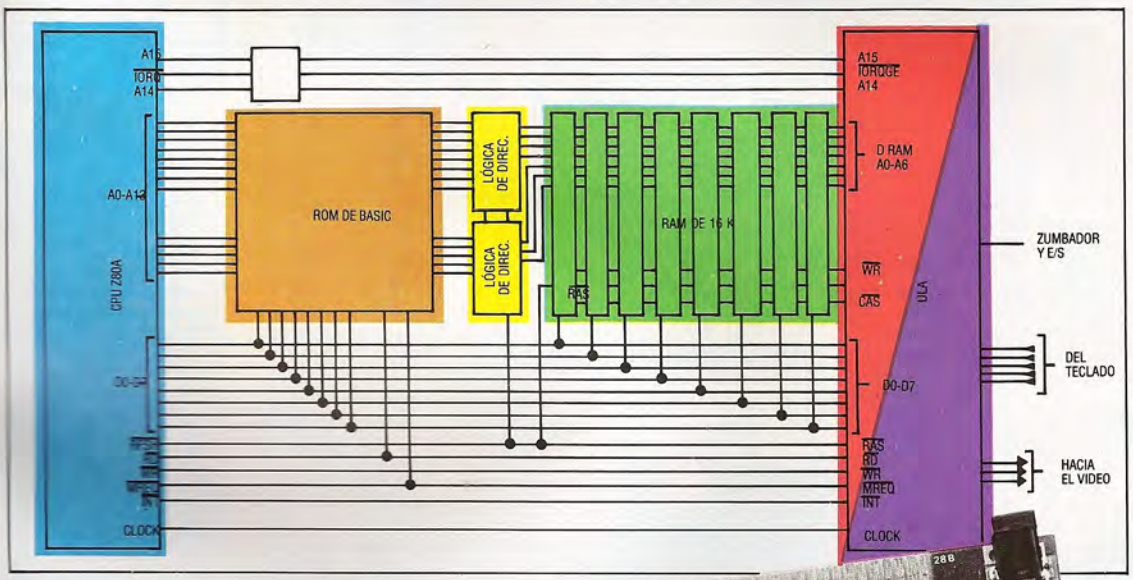
Ponemos punto final a esta serie dedicada a analizar el hardware de los ordenadores examinando el interior de tres populares micros: Sinclair Spectrum, Commodore 64 y BBC Modelo B. De esta manera podremos identificar en cada máquina los componentes que hemos estudiado. Asimismo, proporcionamos diagramas de lógica simplificada para demostrar cómo se concretan los circuitos teóricos.

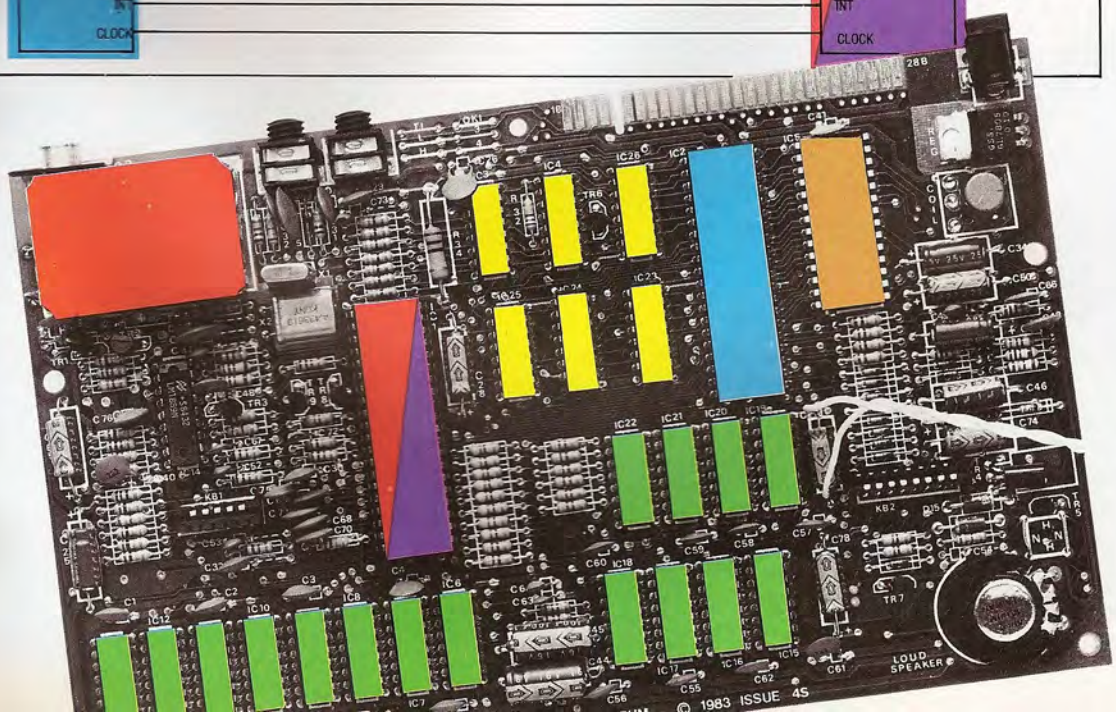
Sinclair Spectrum

Desde el punto de vista de su arquitectura, el ZX Spectrum es bastante distinto de las otras máquinas que vemos aquí. Basado en el procesador Z80, la mayoría de las funciones del Spectrum las lleva a cabo una ULA especial. Este chip diseñado a medida se ocupa del proceso de video, refresco de memoria y E/S mínima desde teclado, grabadora de cassette y zumbador. El diagrama (abajo) corresponde a uno de los primeros Spectrum de 16 K. Las versiones de 48 K poseen chips de RAM extras y decodificación de direcciones. Al igual que en todos los sistemas de visualización de video para micros, la ULA necesita realizar accesos rápidos y regulares a la memoria. En el Spectrum de 48 K, el primer bloque de 16 K, que contiene los datos de video, está conectado directamente a la ULA. En circunstancias normales, ésta puede acceder a esta área de 16 K independientemente, mientras la CPU accede a la ROM de BASIC o al área de RAM extra de 32 K. El problema se plantea cuando la CPU desea acceder al área de video de 16 K (cosa que puede hacer, porque allí están almacenadas las variables del sistema de BASIC). Para evitar el conflicto entre la CPU y la ULA, la ULA monitoriza las líneas de dirección A14 y A15 y detiene temporalmente el reloj de la CPU si se produce un conflicto de acceso

Clave

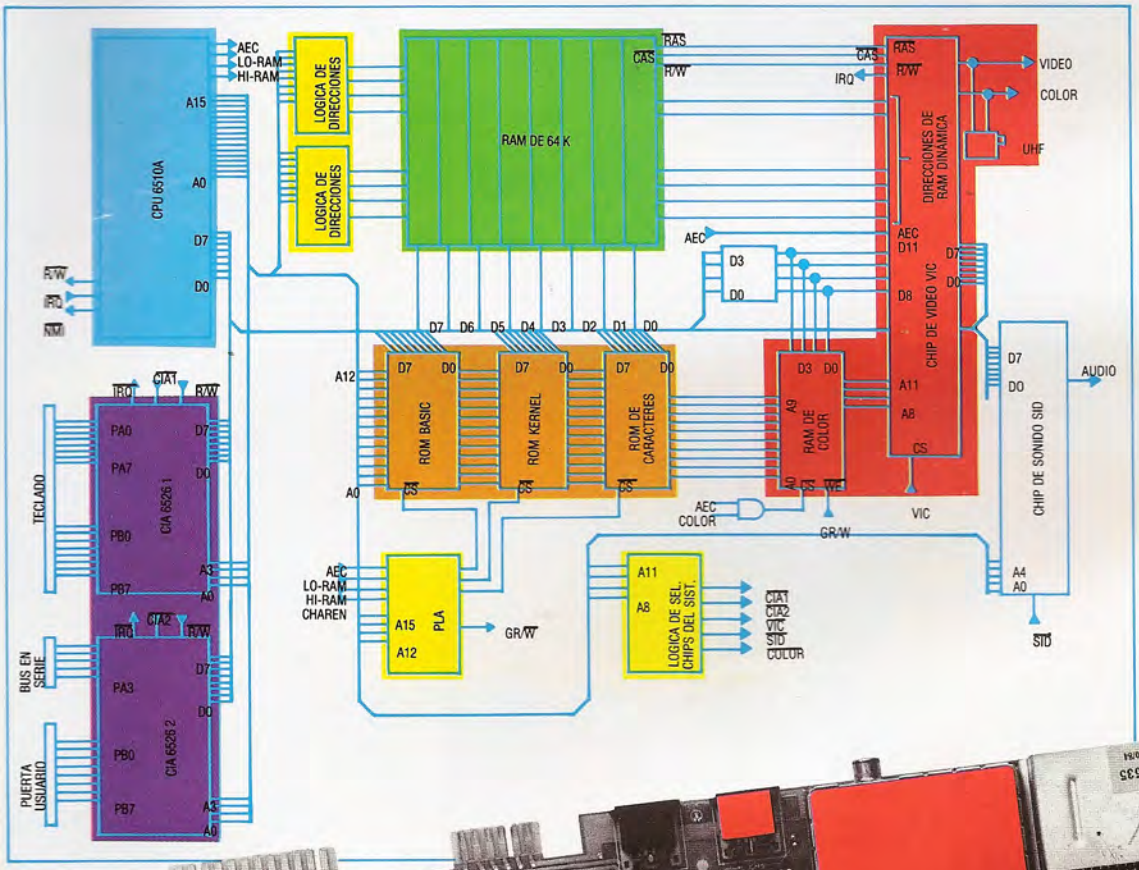
- Procesador
- RAM dinámica
- ROM
- Subsistema PIO
- Subsistema de visualización de video
- Lógica del sistema o de direccionamiento

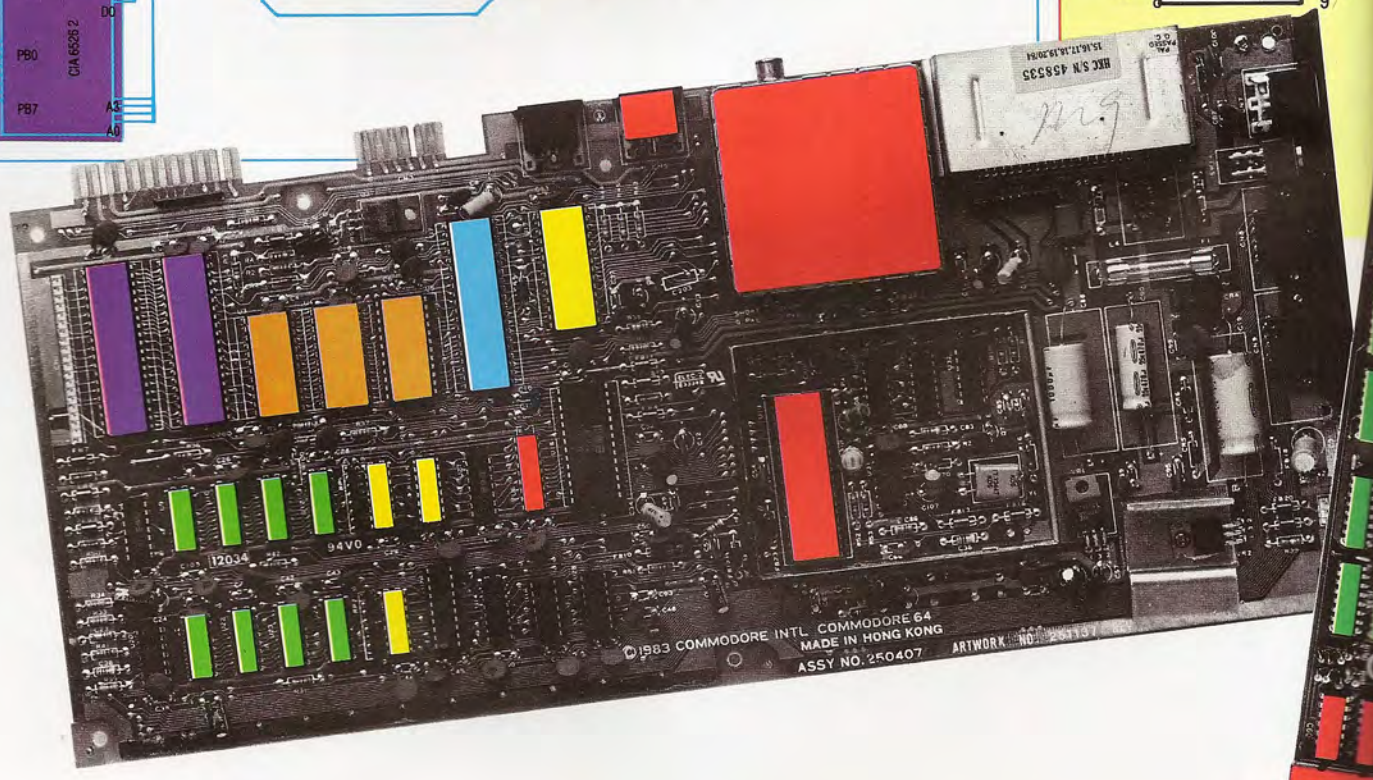




Commodore 64

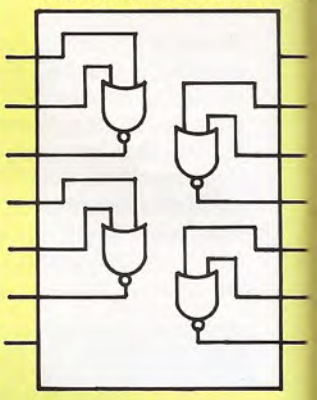
Desde el punto de vista del hardware, el Commodore 64 es una máquina bien equipada. Posee 64 K de RAM dinámica, proporcionada por ocho chips de RAM de 64 Kbits, y tres ROM de ocho K que contienen el BASIC, las rutinas kernel de E/S y las definiciones de caracteres. El procesador 6510 es una versión mejorada del 6502 que permite conmutar las ROM en el espacio de direcciones del procesador a través de un registro y una PLA especial del chip. El C64 tiene dos chips PIO, uno de los cuales está dedicado al teclado. El segundo PIO utiliza una puerta para controlar el propio sistema de bus en serie de Commodore, al cual se pueden conectar impresoras, unidades de disco y otros dispositivos periféricos; la otra puerta queda libre como puerta para el usuario. El chip VIC direcciona la RAM dinámica directamente para obtener información de pantalla. Puesto que necesita más tiempo de comunicación con la memoria que el que se puede conseguir realizando accesos durante la fase del reloj del sistema en que la CPU no está utilizando los buses de datos y direcciones, el VIC detiene temporalmente el procesador de cuando en cuando para poder acceder a la RAM



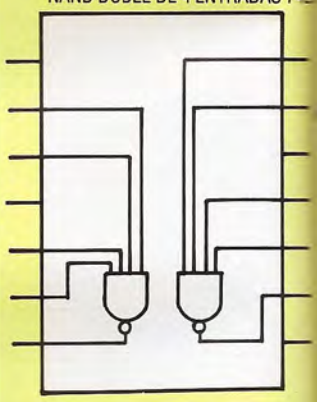


Deducciones lóg

NAND CUAD DE 2 ENTRADAS 74LS



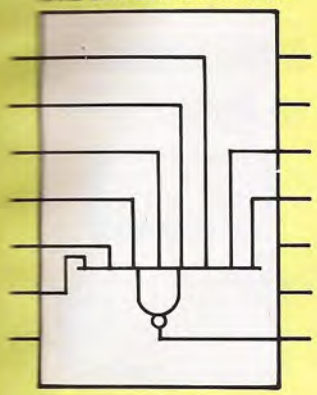
NAND DOBLE DE 4 ENTRADAS 74L



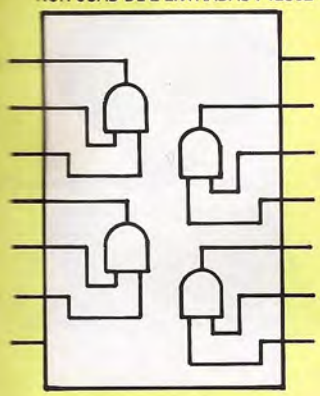
DECODIFICADOR DE 3 A 8 74LS138

Chris Stevens

NAND DE 8 ENTRADAS 74LS30



NOR CUAD DE 2 ENTRADAS 74LS02

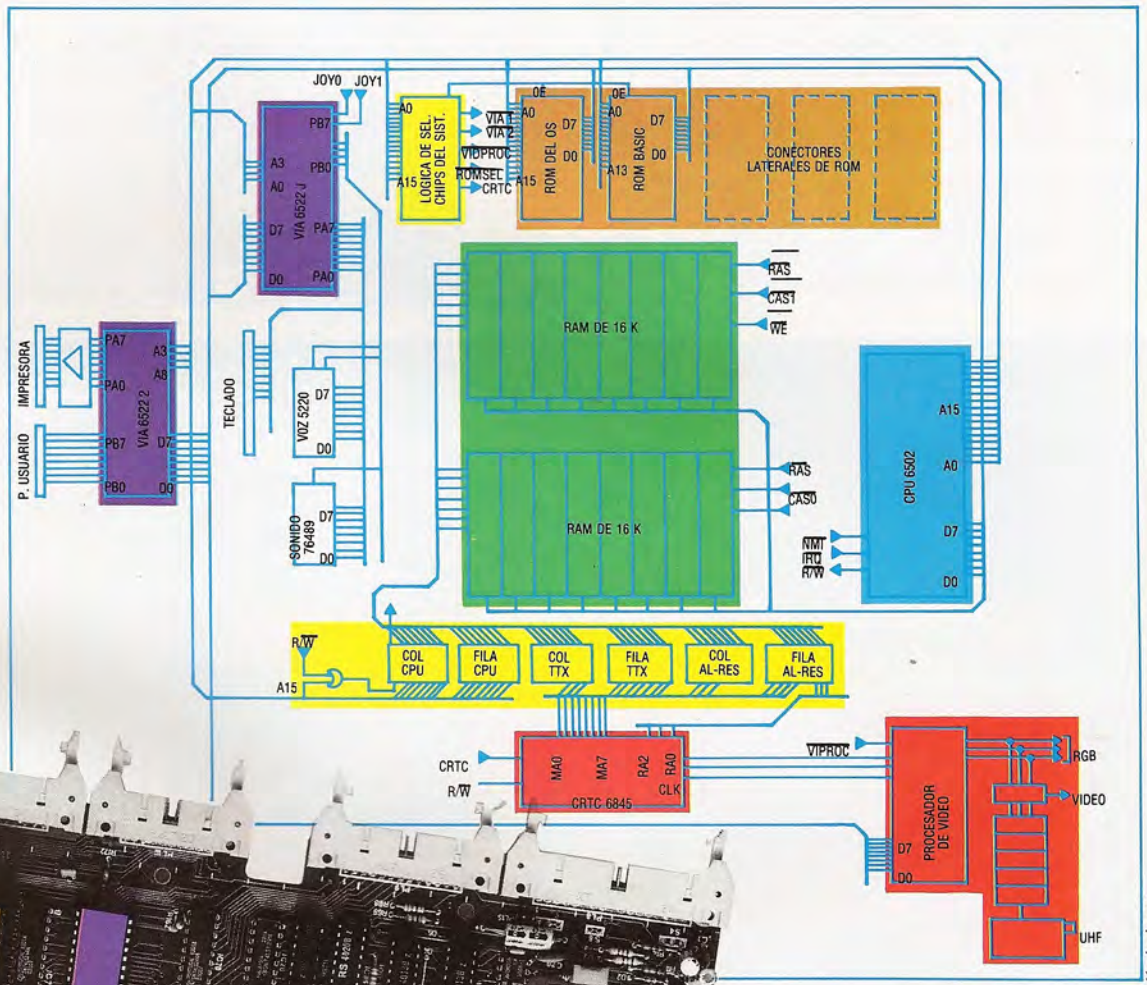


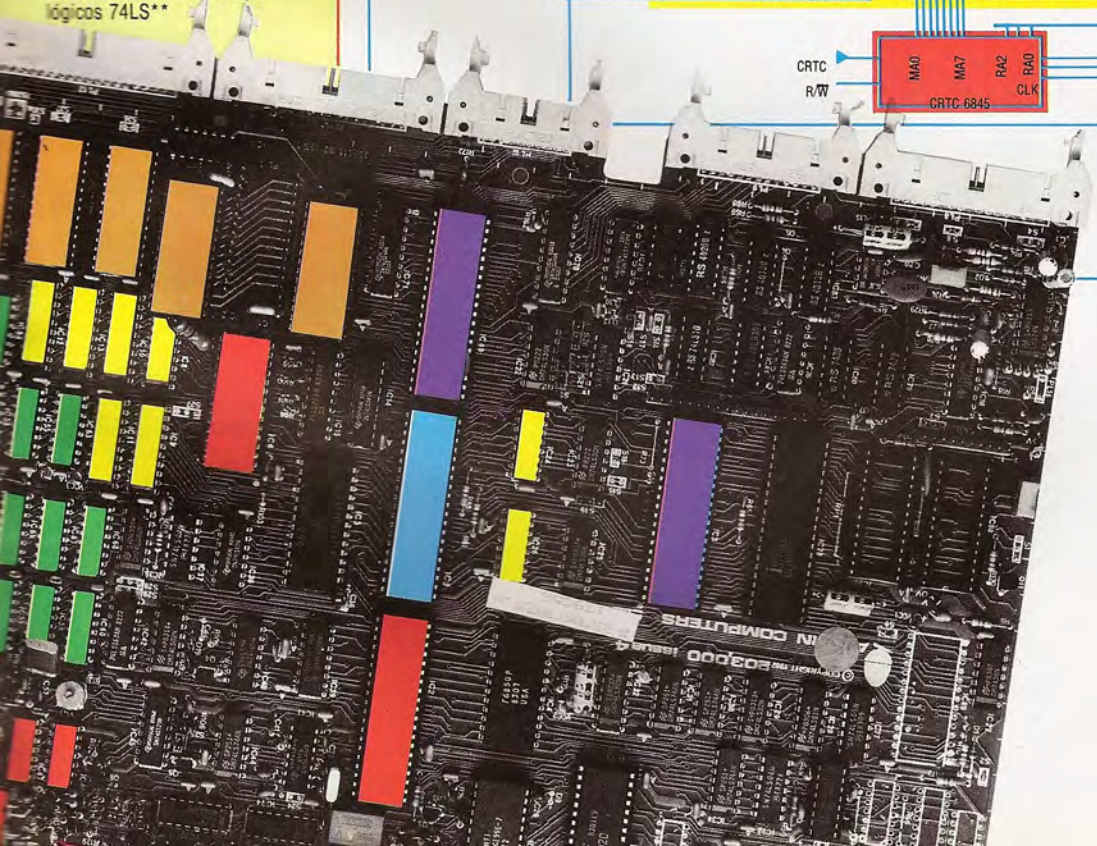
Lógica aplicada
Muchos de los chips más pequeños que se encuentran en la placa impresa de un micro pertenecen a la familia 74** o 74LS**. Estos chips contienen diversas combinaciones de puertas lógicas y se utilizan para *gating* o decodificar señales de dirección o control. Aquí vemos los detalles de varios chips lógicos 74LS**

BBC Modelo B Micro

Diseñado originalmente en 1981, el BBC Modelo B todavía utiliza 16 chips de RAM dinámica de 16 K para proveerse de 32 K de memoria. Se utilizan seis chips de lógica de direccionamiento para permitir que la CPU y los chips de video y teletexto compartan la RAM. En la configuración básica se proporcionan dos ROM de 8 K así como tres ranuras laterales de ROM para ROMs adicionales.

Se proporcionan dos PIO, así como un chip de E/S en serie (que no se incluye en el diagrama). El primer PIO es exclusivo para el sistema y controla los chips del teclado, de sonido y de voz. Este PIO también monitoriza los botones de disparo de la palanca de mando y controla algunas de las funciones de video tales como de desplazamiento de pantalla. La CPU no se comunica directamente con los sistemas de teclado y sonido, sino que utiliza la puerta A del PIO del sistema como si fuera en realidad un bus de datos lento. El segundo PIO permite la conexión de una impresora en paralelo y a la puerta PIO que queda libre se puede acceder a través de la puerta para el usuario





Kevin Jones

Mano a mano

Prosiguiendo nuestro proyecto, completaremos la programación correspondiente al jugador y analizaremos la mano en preparación para el turno del ordenador

Ya hemos desarrollado una rutina con fines generales que sumará el total de una mano y establecerá una variable, EF, en diversos valores correspondientes a los posibles estados de la mano. Ahora podemos usar esta rutina para poder terminar la mano del jugador.

Recuerde que en esta etapa del juego se le habrán repartido al jugador y a la banca dos naipes a cada uno. El jugador puede plantarse o bien pedir carta para acercarse más a 21 sin pasarse. Para inclinar un poco más el juego a favor de la banca hemos incluido la regla que impide que usted se plante si su mano totaliza menos de 17. En un próximo capítulo incluiremos una tercera opción de apuesta que le permitirá doblar su apuesta y se le reparta un naipe más.

La línea 120 llama a la rutina pedir/plantarse del bucle principal del juego después de haber comprobado la opción quemar. La subrutina de la línea 2600 en realidad no realiza el trabajo de pedir carta o plantarse sino que, en cambio, llama a otra subrutina de la línea 2700 para que lleve a cabo la tarea. Al retornar de esta subrutina, usted habrá completado su mano y EF estará establecida en uno de los cinco estados posibles de la mano, como *royal pontoon* ("veintiuno real") o *bust* ("pasarse"). Median-

La mano del jugador

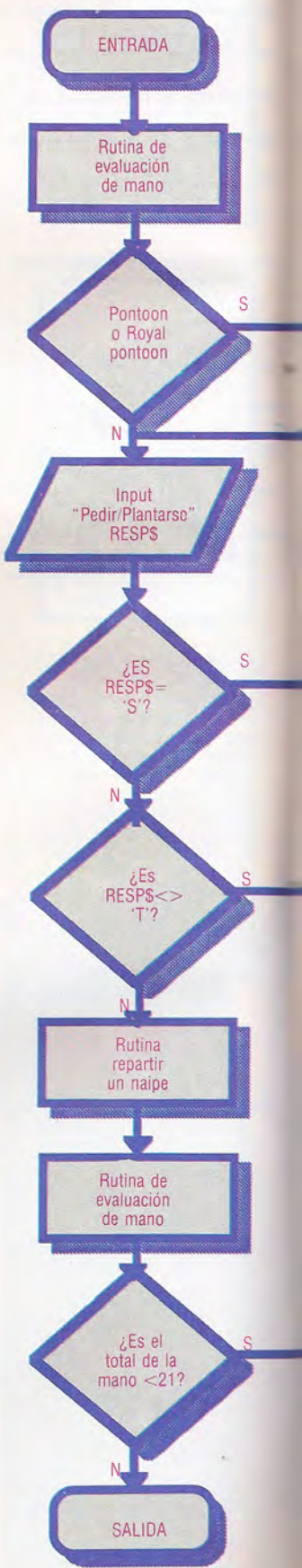
BBC Micro

```
 120 GOSUB 2600
2600 REM
2610 GOSUB 2700
2620 ON EF GOSUB 3500,3600,3700,3800,3900
2630 RETURN
2700 REM
2710 GOSUB 800:IF EF=2 OR EF=3 THEN
     RETURN
2720 GOSUB 700:PRINT"PEDIR/PLANTARSE/
     DOBLAR";
2725 RESP$=GET$
2727 IF RESP$="S" THEN GOSUB 2800:IF CS=0 THEN
     RETURN
2730 IF RESP$="S" AND CS=1 THEN 2700
2750 IF RESP$<>"T" THEN 2700
2760 FL=0:PL=1:GOSUB 1300
2770 GOSUB 800:IF EF=1 THEN 2700
2780 RETURN
2800 REM
2810 CS=1:GOSUB 800
2820 IF (TT(PL,2)> 17 AND TT(PL,2)< 22) OR TT(PL,1)>=17
     THEN CS=0: RETURN
2825 GOSUB 700:PRINT"NO PUEDES PLANTARTE CON MENOS
     DE 17"
2830 FOR DL=1 TO 5000:NEXT DL
2840 RETURN
3500 REM **** MENOS DE 21 ****
3505 PV=1:PS=TT(1,2):IF PS> 21 THEN PS=TT(1,1)
3510 GOSUB 700:PRINT"MENOS DE 21":FOR DL=1 TO
     500:NEXT DL:RETURN
3600 REM **** ROYAL PONTOON ****
3605 PV=4
3610 GOSUB 700:PRINT"ROYAL PONTOON":RETURN
3700 REM
3705 PV=2
3710 GOSUB 700:PRINT"PONTOON":RETURN
3800 REM
3805 PV=0
3810 GOSUB 700:PRINT"BUST":RETURN
3900 REM
3905 PV=3
3910 GOSUB 700:PRINT"JUEGO DE CINCO NAIPES":
     RETURN
```

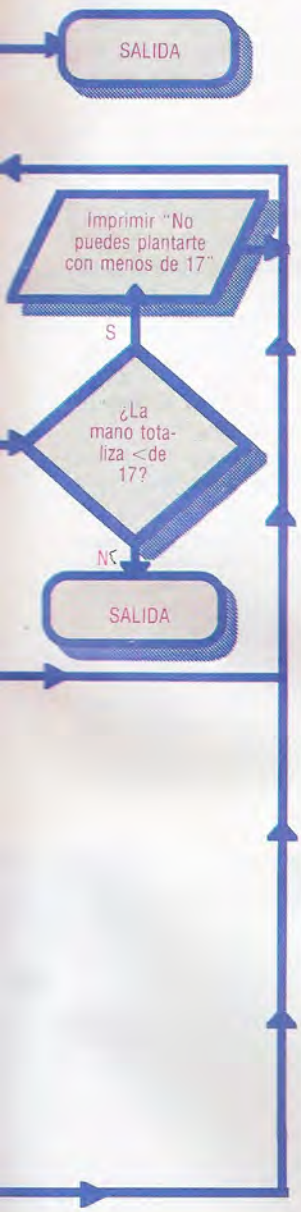
Gama Amstrad CPC

```
 120 GOSUB 2600:REM PEDIR etc.
2600 REM **** PEDIR apostador etc. ****
2610 GOSUB 2700:REM hacerlo!
2620 ON ef GOSUB 3500,3600,3700,3800,3900
2630 RETURN
2700 REM **** pedir/plantarse/doblar ****
2710 GOSUB 800:IF ef=2 OR ef=3 THEN RETURN:REM
     comprobar pontoon/royal pontoon
2720 GOSUB 700:PRINT" pedir/plantarse/doblar";
2725 resp$="":WHILE resp$="":resp$=INKEY$:
     WEND
2727 IF resp$<>CHR$(13) THEN PRINT resp$
2730 IF resp$="s" THEN GOSUB 2800:IF cs=0 THEN
     RETURN
2733 IF resp$="s" AND cs=1 THEN 2700:REM no se puede
     plantar
2750 IF resp$ <> "t" THEN 2700:REM error de
     entrada
2760 fl=0:pl=1:GOSUB 1300:REM repartir
2770 GOSUB 800:IF ef=1 THEN 2700:REM otra
     vez?
2780 RETURN
2800 REM **** plantarse ****
2810 cs=1:GOSUB 800:REM evaluar
2820 IF (tt(pl,2)>=17 AND tt(pl,2)<22) OR tt(pl,1)>=17 THEN
     cs=0:RETURN
2825 GOSUB 700:PEN rojo:PRINT "No te puedes plantar con
     menos de 17"
2830 FOR dl=1 TO 100:NEXT dl
2840 RETURN
3500 REM **** menos de 21 ****
3505 pv=1:ps=tt(1,2):IF ps>21 THEN ps=tt(1,1)
3510 GOSUB 700:PRINT "Menos de 21":RETURN
3600 REM **** royal pontoon ****
3605 pv=4
3610 GOSUB 700:PRINT "Royal pontoon":
     RETURN
3700 REM **** pontoon ****
3705 pv=2
3710 GOSUB 700:PRINT "Pontoon":RETURN
3800 REM **** bust ****
3805 pv=0
3810 GOSUB 700:PRINT "Bust": RETURN
3900 REM **** juego de cinco naipes ****
3905 pv=3
3910 GOSUB 700:PRINT "Juego de cinco naipes":
     RETURN
```

Muéstrame tu mano



El diagrama de flujo muestra cómo se controla la parte del juego correspondiente al jugador. La salida de la estructura de control sólo se puede conseguir bajo tres condiciones: si los dos naipes que se le repartieron originalmente al jugador dan *pontoon* (un as y un diez) o *royal pontoon* (un as y una figura), si el jugador se planta legalmente o si el jugador se pasa



Caroline Clayton

te el uso de la instrucción ON...GOSUB (exceptuando la versión para el Spectrum, que saca partido de la facilidad de números de línea calculados del BASIC Spectrum), el valor de EF dirige al programa hasta otra subrutina. Ésta imprime un mensaje en el cual se le informa sobre el estado de la mano completada, y establece una variable PV. Esta variable se utilizará más adelante, cuando le toque el turno al ordenador, para retener el estado de la mano suya.

La auténtica rutina pedir/plantarse empieza en la línea 2700. Inmediatamente después de entrar en ella se llama a la rutina de evaluación. Se comprueban tanto el *royal pontoon* (un as y una figura) como el *pontoon* de dos naipes (un as y un diez) en los dos naipes que ya se han repartido al jugador. La rutina continúa pidiendo al jugador que entre T para pedir o S para plantarse y después comprueba el teclado en busca de una respuesta.

Utilizando GET/INKEY$/GET$ en lugar de INPUT, usted simplemente puede pulsar las teclas T o S sin tener también que pulsar la tecla Return (tener que pulsar ésta una y otra vez suele resultar molesto). La desventaja de usar GET/INKEY$/GET$ es que las pulsaciones de tecla no generan automáticamente las adecuadas visualizaciones en pantalla, como lo hacen con INPUT. Por consiguiente, necesitamos añadir líneas a nuestro programa para visualizarlas.

Si en este punto usted decide plantarse, el programa debe comprobar su mano para determinar si es o no menor que 17. La rutina de la línea 2800 comprueba esto mirando los marcadores de mano para el jugador, que ya habrán sido calculados por la rutina de evaluación de manos. Si su mano totaliza menos que 17, se imprime un mensaje. Además, se establece una bandera, CS, en 1. Al volver de la rutina pedir/plantarse, CS determinará si usted está o no en condiciones de plantarse, en cuyo caso se puede salir de la rutina. No obstante, si el programa le pide que vuelva a seleccionar, volverá hacia atrás en el bucle hasta el comienzo de la rutina.

Si selecciona la opción pedir, se le repartirá un naipe y se volverá a evaluar su mano. Si su mano suma aún menos de 21 (lo que se indicaría con EF=1) después de repartirse el nuevo naipe, la rutina vuelve hacia atrás en el bucle para una nueva entrada.

Con esto completamos la programación para la mano del jugador. En el próximo capítulo veremos cómo se puede programar el ordenador para que sea capaz de replicar "inteligentemente" a la mano del jugador.

Sinclair Spectrum

```
 120 GO SUB 2600: REM PEDIR ETC.
2600 >REM **** PEDIR APOSTADOR ETC EVALUAR ****
2610 GO SUB 2700: REM HACERLO!
2620 GO SUB (EF*100)+3400
2630 RETURN
2700 REM **** PEDIR/PLANTARSE/DOBLAR ****
2710 GO SUB 800: IF EF=2 OR EF=3 THEN RETURN: REM
     COMPROBAR PONTOON/ROYAL PONTOON
2720 GO SUB 700: PRINT " PEDIR/PLANTARSE/DOBLAR ";
2725 LET A$=INKEY$: IF A$="" THEN GO TO 2725
2727 IF A$<>CHR$ 13 THEN PRINT A$
2730 IF A$="S" THEN GO SUB 2800: IF CS=0 THEN
     RETURN
2733 IF A$="S" AND CS=1 THEN GO TO 2700:REM NO SE
     PUEDE PLANTAR
2750 IF A$<>"T" THEN GO TO 2700: REM ERROR DE
     ENTRADA
2760 LET FL=0:LET PL=1:GO SUB 1300: REM
     REPARTIR
2770 GO SUB 800: IF EF=1 THEN GO TO 2700: REM OTRA
     VEZ?
2780 RETURN
2800 REM **** PLANTARSE ****
2810 LET CS=1: GO SUB 800: REM EVALUAR
2820 IF (T(PL,2)>=17 AND T(PL,2)<22) OR T(PL,1)>=17
     THEN LET CS=0: RETURN
2825 GO SUB 700: PRINT FLASH 1;"NO PUEDES PLANTARTE
     CON MENOS DE 17!"
2830 FOR L=1 TO 300: NEXT L
2840 RETURN
3500 REM MENOS DE 21
3505 LET PV=1: LET PS=T(1,2): IF PS>21THEN LET
     PS=T(1,1)
3510 GO SUB 700: PRINT "MENOS DE 21":
     RETURN
3600 REM **** ROYAL PONTOON ****
3605 LET PV=4
3610 GO SUB 700: PRINT "ROYAL PONTOON":
     RETURN
3700 REM **** PONTOON ****
3705 LET PV=2
3710 GO SUB 700: PRINT "PONTOON": RETURN
3800 REM **** BUST ****
3805 LET PV=0
3810 GO SUB 700: PRINT "BUST": RETURN
3900 REM **** JUEGO DE CINCO NAIPES ****
3905 LET PV=3
3910 GO SUB 700: PRINT "JUEGO DE CINCO NAIPES":
     RETURN
```

Commodore 64

```
 120 GOSUB 2600:REM PEDIR ETC
2600 REM **** PEDIR APOSTADOR ETC EVALUAR ****
2610 GOSUB 2700:REM HACERLO!
2620 ON EF GOSUB 3500,3600,3700,3800,
     3900
2630 RETURN
2700 REM **** PEDIR/PLANTARSE/DOBLAR ****
2710 GOSUB 800:IF EF=2 OR EF=3 THEN RETURN: REM
     COMPROBAR PONTOON/ ROYAL PONTOON
2720 GOSUB 700:PRINT" PEDIR/PLANTEARSE/
     DOBLAR ";
2725 GET RESP$:IF RESP$="" THEN 2725
2727 IF RESP$<>CHR$(13) THEN PRINT RESP$
2730 IF RESP$="S" THEN GOSUB 2800:IF CS=0 THEN
     RETURN
2733 IF RESP$×"S" AND CS=1 THEN 2700:REM NO SE
     PUEDE PLANTAR
2750 IF RESP$<>"T"THEN 2700:REM ERROR DE
     ENTRADA
2760 FL=0:PL=1:GOSUB 1300:REM REPARTIR
2770 GOSUB 800:IF EF=1 THEN 2700:REM OTRA
     VEZ?
2780 RETURN
2800 REM **** PLANTARSE ****
2810 CS=1:GOSUB 800:REM EVALUAR
2820 IF (TT(PL,2)>=17 AND TT(PL,2)<22) OR
     TT(PL,1)>=17 THEN CS=0:RETURN
2825 GOSUB 700:PRINTCHR$(28);"NO PUEDES PLANTARTE
     CON MENOS DE 17"
2830 FOR DL=1 TO 1000:NEXT DL
2840 RETURN
3500 REM **** MENOS DE 21 ****
3505 PV=1:PS=TT(1,2):IF PS>21 THEN PS=
     TT(1,1)
3510 GOSUB 700:PRINT"MENOS DE 21":RETURN
3600 REM **** ROYAL PONTOON ****
3605 PV=4
3610 GOSUB 700:PRINT"ROYAL PONTOON":
     RETURN
3700 REM **** PONTOON ****
3705 PV=2
3710 GOSUB 700:PRINT"PONTOON":RETURN
3800 REM **** BUST ****
3805 PV=0
3810 GOSUB 700:PRINT"BUST":RETURN
3900 REM **** JUEGO DE CINCO NAIPES ****
3905 PV=3
3910 GOSUB 700:PRINT"JUEGO DE CINCO NAIPES":
     RETURN
```

Preguntas al directorio

Desde buscar en un directorio hasta imprimir un archivo, las instrucciones residentes del MS-DOS se ocupan de casi todas las eventualidades

Los sistemas MS-DOS normalmente se cargan tan sólo con el encendido (desde un disco rígido) o insertando un disco de sistema flexible. Muchas máquinas poseen teclados tipo IBM y en éstas se puede efectuar una reinicialización o arranque en caliente en cualquier otro momento mediante el método "Control-Alt-Delete" tradicional del IBM PC (se mantienen pulsadas dos teclas del lado izquierdo del teclado, etiquetadas CTRL y ALT, mientras simultáneamente se pulsa la tecla DEL [*delete*] del lado derecho). La última operación, después de que se ha cargado el MS-DOS y llevado a cabo la inicialización del sistema, es para consultar la unidad por defecto en busca de un archivo *batch* denominado AUTOEXEC.BAT. Si está presente, las instrucciones que contiene se ejecutarán automáticamente como si se las hubiera entrado desde el teclado.

Podríamos, por ejemplo, escribir un archivo AUTOEXEC.BAT simple que contuviera la instrucción:

```
MENU
```

donde MENU.COM (o MENU.EXE) fuera un programa de aplicación adecuado (un archivo EXE [de *executable*: ejecutable] o COM [de *command*: instrucción]). Este programa se ejecutaría automáticamente cada vez que se cargara el sistema con este disco en particular; esto se conoce como sistema *turnkey* (giro de llave).

El MS-DOS (y el PC-DOS) posee más potencia y mayor flexibilidad, y todas las utilidades "transitorias" se proporcionan como archivos EXE o COM separados. En este capítulo nos concentraremos, sin embargo, en el núcleo de instrucciones "residentes" comunes a todas las versiones de MS-DOS y PC-DOS. Éstas están disponibles a través del intérprete de línea de comando DOS, o "caparazón", denominado COMMAND.COM (la única parte del DOS que queda "visible" para el usuario).

De todas las utilidades del sistema, quizá la que se utiliza con mayor frecuencia sea la instrucción de directorio. Ésta lista todos los archivos de un disco o, a partir del DOS 2 en adelante, de un *directorio* (una porción con nombre y restringida de todo el espacio del disco). La forma básica de la instrucción no es más que:

```
dir
```

Al igual que en CP/M, sobre el cual se modeló íntimamente el DOS 1, esta forma simple lista todos los archivos del disco conectado actualmente (por "defecto"). Si se requiriera un listado del directorio de otra unidad, esto se especificaría como un parámetro de línea de comando. Nuevamente, esto es idéntico al uso del CP/M y también sigue las mismas especificaciones para referencias ambiguas a archivos: ? representa cualquier carácter individual, y se puede utilizar * ya sea para un grupo de caracteres o bien, posiblemente, para ningún carácter en absoluto.

En el nombre de archivo primario se admiten entre uno y ocho caracteres, y hasta tres caracteres (o ninguno) para la extensión o nombre secundario; nuevamente, tal como en CP/M. No obstante, cuando el listado aparece en la pantalla se observa una diferencia radical tanto en el formato utilizado como en la cantidad de información visualizada. El MS-DOS lista no sólo los nombres de los archivos, sino también el tamaño de éstos (en bytes) y la fecha y hora en que se escribió cada archivo en el disco.

He aquí un listado de directorio típico:

```
A>dir

Volume in drive A has no label
Directory of A:

COMMAND  COM   22672   9-03-85  11:36a
AUTOEXEC BAT     128  22-07-85   5:15p
WP       EXE   73156  12-10-84  12:07p
WPMSG    OVR   38269  27-09-84  12:02p
WPINST   EXE   56385   3-10-84  12:04p
WCOUNT   PAS    4986   1-08-85   3:11p
WCOUNT   OBJ    3874   1-08-85   3:12p
WCOUNT   EXE    8385   1-08-85   3:14p
WCOUNT   BAK    4732  31-07-85  11:23p
MARY1    TXT     634   7-08-85  11:05a
INVOICE1 885     477  21-08-85   9:36a
MARY2    TXT     938  15-08-85  12:47p
```

Hil Rudge, cortesía de "Microscope"

En muchos sistemas quizá la fecha aparezca en formato norteamericano (mes-día-año), pero las versiones de DOS recientes se ciñen a la convención europea, además de admitir juegos de caracteres especiales para las lenguas europeas y escandinavas. Si bien los tamaños se dan en bytes, los archivos suelen ocupar más espacio en el disco. Cada archivo utiliza una cierta cantidad de sectores completos, generalmente 512 bytes cada uno.

El DOS también visualiza el número de archivos y el espacio restante en el disco. Esto combina muchas de las características de la utilidad CP/M externa ("transitoria") STAT.COM con las de la instrucción dir residente y, junto con el sello de hora y fecha de la creación del archivo, se convierte en una instrucción mucho más útil. El motivo fundamental de esta funcionalidad adicional es la supresión de la barrera de 65 Kbytes de memoria.

Los sistemas CP/M estaban casi invariablemente restringidos a este espacio máximo de direccionamiento, a menos que implementaran sofisticados sistemas de paginación de memoria. Cada puñado de bytes que empleaba el OS le "robaba" al usuario igual cantidad de RAM libre, de modo que en el código del sistema sólo se incorporaba lo esencial como instrucciones residentes. Se obtenía información detallada (como el tamaño del archivo y atributos) mediante la ejecución de un programa separado (como STAT) a modo de utilidad transitoria.

El MS-DOS también posee una gama amplia y versátil de estos programas, pero merced al tan acrecentado espacio de memoria de los sistemas de 16 bits (lo normal es 256 Kbytes o más), el sistema residente puede ser mucho más grande y, por tanto, contener muchas facilidades más potentes. La versión 3 del DOS fácilmente puede requerir más de 60 Kbytes de RAM; la cantidad exacta depende en parte del distribuidor que suministre el sistema. El MS-DOS da por sentada la existencia de un reloj del sistema e, incluso aunque no haya un verdadero dispositivo de hardware (con apoyo de batería) para seguir funcionando cuando se apague el ordenador, habrá disponible una simulación por software, por lo general operando con un método de interrupción continua.

En este caso, el reloj de software se inicializará en, por ejemplo, 1-01-85 12:00:00 y se debe ajustar manualmente cuando se carga el sistema a través de las instrucciones DOS date y time. Éstas visualizan la hora (o fecha) actual e invitan a entrar nuevos valores desde el teclado, que deben responder a los mismos formatos consignados con anterioridad, aunque no necesariamente completos. La interacción también se puede abreviar entrando nuevos valores en la línea de instrucción. De modo que, por ejemplo:

```
time 14
```

establecería al reloj del sistema en 14:00:00 (los dos últimos dígitos representan centésimas de segundo, si bien a menudo la resolución de tiempo no será tan alta como esto). Todo archivo que se cree poco después se estampará con los valores actuales del sistema y podría aparecer en el listado del directorio de esta forma:

```
NUEVOARCHIVO     TXT      512   21-08-85   2:05p
```

A todos los programas MS-DOS se les da la extensión EXE (*executable*: ejecutable) o COM (*command*: instrucción), siendo la única diferencia las restricciones en cuanto a las posiciones de memoria en las cuales se pueden volver a localizar y ejecutar. En términos generales, los archivos EXE son más flexibles y se pueden cargar en los límites de un párrafo

Instrucciones residentes del MS-DOS

Instrucción		Función	Interrs.	Uso	
	copy	copiar	/a/b/v	copy	{unidad:} nombre-con máscara {destino}
11 10 85	date	visualizar /fecha est.		date date	{dd-mm-aa} {mm/dd/aa} (versión USA)
	del	borrar arch.		del	{unidad:} {n.-con-máscara}
	dir	listar arch.	/p/w	dir	{unidad:} {n.-con-máscara}
	erase	ver del			
	ren	cambiar nombre archivos		ren	{unidad:} nombre1 nombre 2
	rename	ver ren			
12 20 52	time	visualizar /hora est.		time	{hh:mm:ss}
	type	listar archivos texto		type	{unidad:} nombre-archivo

Clave:
{ } los elementos que aparecen entre llaves son opcionales

| una barra vertical separa alternativas

carácter ::= cualquier carácter ASCII excepto",.?*<>:;[]/\|
nombreprimario::=carácter{carácter} (máximo ocho)
nombresecundario::={carácter} (máximo tres)
nombrearchivo::=nombreprimario{.nombresecundario}
nombre-con-máscara::=nombrearchivo conteniendo caracteres "de máscara" (*y?)
dispositivo::=CON | LST | PRN | AUX | LPT1 etc.
destino::=nombrearchivo | dispositivo
unidad::=A | B | etc. (generalmente hasta P)

Caroline Clayton

Un hombre de suerte

La historia de cómo Bill Gates, de Microsoft, se aseguró el contrato para suministrar a IBM un OS de 16 bits (MS-DOS) es un tanto inusual. Habiendo sido contactada por IBM, inicialmente Microsoft no estaba segura de si podría aceptar el contrato para producir el OS. Por consiguiente, IBM decidió hacer una visita a Digital Research para hablar del asunto con Gary Kildall, quien era conocido por haber escrito el CP/M. El día de la visita, sin embargo, Kildall no estaba en su despacho. Tras esperarlo durante dos horas, la delegación de IBM se marchó y posteriormente se le concedió el contrato a Microsoft. De este modo, un contrato que conduciría a la creación de un estándar industrial se le concedió a una empresa a la cual sólo se conocía por una versión de BASIC y que contaba con poca experiencia en tecnología de sistemas operativos

(palabra del ordenador), mientras que los archivos COM deben residir dentro de los límites de un *segmento* (al comienzo de una zona o segmento de 64 Kbytes, como las convenciones de la familia Intel 8086). Se utilizan otros nombres de archivo secundarios con significados especiales, en ocasiones por razones de mera conveniencia, de modo que con frecuencia a los documentos, cartas y otros archivos de texto se les suele dar la extensión .TXT (o .DOC) a modo de ayuda mnemotécnica. Los archivos objeto en código máquina con formato Intel estándar (reubicable), tendrán la extensión .OBJ, designándose al código fuente con las extensiones .PAS, .FOR o .C usuales para programas en PASCAL, FORTRAN y C.

Estudiando un poco más el listado del directorio, podemos deducir que el programa WCOUNT.EXE se originó como un archivo fuente en PASCAL (WCOUNT.PAS se guardó a las quince horas y 11 minutos de la tarde del 1.º de agosto). Éste se compli-có como un archivo .OBJ de formato Intel estándar alrededor de un minuto después, y dos minutos más tarde se montó para producir el programa ejecutable (EXE) final. Todavía se detectan indicios de la versión previa (WCOUNT.BAK) archivada como copia de seguridad (BAK) por el procesador de textos (WP.EXE), que obviamente causó bastantes problemas, a juzgar por el hecho de que el programador parece haber estado trabajando hasta altas horas de la noche la semana anterior (el 31 de julio). Quizá, a juzgar por su nombre, podríamos aventurar que este programa cuenta las palabras de un archivo de textos; ¡y toda esta información la hemos obtenido con sólo un rápido estudio de la información que ofrece dir!

La instrucción tiene un par de ases bajo la manga que la diferencian más aún de su equivalente CP/M. Éstos asumen la forma de *interruptores*, u opciones añadidas en la línea de instrucción. El interruptor P da una página (o pantalla llena) de archivos por vez, haciendo una pausa hasta que se pulsa una tecla antes de proseguir con el listado. Esto es especialmente útil en los modernos discos de gran capacidad con los que se podrían almacenar centenares de archivos en un dispositivo o directorio. El otro interruptor, W, permite la supresión de la información detallada, y hace que sólo los nombres de archivo se listen en formato *Wide* (ancho), de cinco por línea. Muchas instrucciones DOS, internas y externas, poseen tales interruptores y su condición se indica precediendo a *cada uno* con una barra inclinada:

```
dir c:*exe/p/w
```

Esto podría abarcar hasta 115 (5×23) archivos .EXE por "página" de pantalla.

Otra instrucción que se comporta como su equivalente de CP/M es la que se utiliza con mayor frecuencia para visualizar el contenido de un archivo de textos en la pantalla (o impresora). La sintaxis es la siguiente:

```
type {dispositivo:} nombrearchivo
```

De modo que, por ejemplo, para listar un programa en MODULA-2 llamado RATON.MOD en el dispositivo de disco B, entraríamos:

```
type b:raton.mod
```

Al igual que en los sistemas CP/M, generar un Control-P antes del retorno de carro activará la impresora, en caso de que deseemos un listado por impresora.

Existe un método alternativo al del CP/M, que en cierto modo es preferible y, por cierto, más flexible. Mientras que el CP/M utiliza PIP.COM como programa de sistema transitorio para transferencia de datos, el MS-DOS tiene una utilidad COPY muy potente incorporada como instrucción residente. En su forma más simple, se la invoca mediante:

```
copy {dispositivo:} nombrefuente {destino}
```

donde el destino puede ser otro nombre de archivo o un dispositivo del sistema como CON (la consola), LIST o PRN (el dispositivo de listado estándar o impresora), respectivamente. De modo que, para obtener una salida impresa:

```
copy b:raton.mod prn
```

conseguiría lo mismo que la instrucción type de arriba, pero sin tener que utilizar Control-P para evitar que en el listado aparezca ninguna otra cosa a excepción del contenido del archivo. Con la instrucción type se añadiría un aviso del sistema (como, por ejemplo, A>).

La instrucción copy puede tomar tres interruptores opcionales: /a copia archivos ASCII utilizando una marca de final del archivo (Control-Z), /b emplea el final físico del archivo, ignorando cualquier Control-Z, y /v verifica cada transferencia de datos (conmutable permanentemente con verify activado). Esto es posible porque el MS-DOS lleva un registro del tiempo que permanece cada archivo en el directorio. Si no se especifica el destino, copy da por sentado que usted se refiere a un archivo del mismo nombre que la(s) fuente(s), pero en la unidad por defecto (como copy c:*.txt/a/v).

Esta potente instrucción residente tiene muchas otras posibilidades. Entre las más interesantes está la capacidad de concatenar archivos. Si decimos:

```
copy b:*.txt grandoc.txt
```

todos los archivos de texto del dispositivo B se concatenarán en el archivo grandoc.txt en la unidad por defecto (por el orden del directorio). Si deseamos controlar el orden o especificar archivos con extensiones diferentes, se puede usar el signo +:

```
copy c:x1.wrk+x2.dat+b:x3.frm xxx.new
```

Omitiendo el destino se obtiene el añadido a un archivo existente:

```
copy page1.txt+page2.txt+page3.txt
```

con el contenido de page2.txt y luego de page3.txt añadiéndose al final de page1.txt, conservando este nombre para el archivo resultante.

Hay un editor del sistema (denominado EDLIN.EXE), del cual nos ocuparemos en el próximo capítulo, junto con algunas de las otras «instrucciones» transitorias. Mientras tanto, se puede crear un archivo de texto sin la ayuda de ninguna clase de procesador de textos con tan sólo impartir la instrucción:

```
copy con nuevoarchivo.txt
```

Tras entrar el texto deseado, se cierra el archivo y se lo escribe en disco entrando Control-Z (la marca de final de archivo). A diferencia de pip (CP/M), se lo debe enviar seguido de un ENTER<CR>).

Al pie de la letra

Prosiguiendo nuestro estudio de las instrucciones del 68000 de Motorola, consideraremos la instrucción de comparación

Estrechamente relacionada con la instrucción SUB tenemos la instrucción CMP (comparación), cuya importancia estriba en que, sea cual sea el área de programa en que estemos trabajando, siempre habrá elementos de decisión en el diseño de ese programa. Consideremos el siguiente diseño en alto nivel, que incluye un elemento *pseudo-code* de decisión:

```
If inputchar='N' then
   cleararray
end
```

Aquí la decisión If se codificará así en la fase de implementación del proyecto:

```
CMP.B  #'N',D0     Compara el byte de entrada
                   con N
BNE    NOTSAME     Va a NOTSAME si es
                   diferente
```

Cuando se ejecuta la instrucción CMP, la fuente es restada del destino y se cambian sólo los códigos de condición, sin que sea afectado el destino (como ocurre, por ejemplo, con la instrucción SUB). La bifurcación condicional, BNE, que sigue provocará una bifurcación a la etiqueta NOTSAME si el resultado de la recta de D0-'N' no es cero.

Veamos otros dos ejemplos que emplean atributos diferentes de datos:

```
CMP.W SPEED,D3     Compara contenido palabra de
                   posición SPEED con D3
CMP.L D1,D2        Compara pals. largas completas
                   contenidas en D1 y D2
```

Se notará que los campos de destino en estos ejemplos son registros de datos, y que de hecho puede utilizarse cualquier modo de direccionamiento como modo de direccionamiento fuente. Si es necesario direccionar cualquier otra forma de destino, se usan diferentes instrucciones CMP. Por ejemplo:

```
CMPA   BETTY,A3    Compara el contenido de la
                   posición BETTY con A3
```

En realidad es posible el uso de cualquier modo de direccionamiento fuente. Sin embargo, la forma inmediata que sigue sólo permite modos alterables de datos.

```
CMPI   #3,(A4)     Compara cualquier A4 que se
                   apunte con 3
```

Una última variante de CMP digna de mención es la instrucción CMPM. Por ejemplo, CMPM (A2)+,(A3)2 comparará el contenido de las posiciones de memoria apuntadas por A2 y A3 y después posincrementará los punteros. Por ello, esta instrucción puede ser utilizada para comparar, por ejemplo, palabras clave almacenadas con caracteres que están en un buffer de entrada, y todo eso también en una palabra. He aquí un ejemplo:

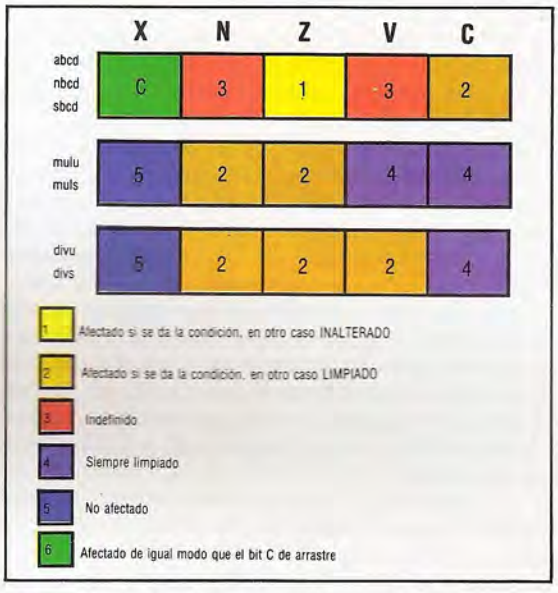
```
LEA    KEYS,A2     Establece el primer puntero
                   hacia las palabras claves
LEA    BUFFER,A3   Y el puntero del buffer
CHECK  CMPM.B      Compara las dos posiciones de
       (A2)+,      memoria
       (A3)+
BEQ    CHECK       Va comparando hasta que
                   sean diferentes
```

El programa en total ocupa sólo siete palabras.

Otro aspecto ulterior es que la instrucción CMPM sólo se permite con el modo postincremento de direccionamiento; así, si se desea cualquier otra forma, entonces un operando ha de ser cargado en el registro de datos en primer lugar y emplear la instrucción CMP.

Dos sencillas instrucciones aritméticas son NEG y EXT. La instrucción NEG (negación) resta de 0 el registro de datos del operando, es decir, forma el valor negativo en complemento a dos del contenido del registro de datos (no se permite ningún otro modo de direccionamiento). Así, por ejemplo, si D0=1111 1010 (−6, en decimal) y ejecutamos NEG.B D0, ocurrirá que D0 contendrá 0000 0110 (6, en decimal). Empleos típicos de esta instrucción incluirán la obtención del valor absoluto de un operando de datos comprobando primeramente el valor negativo y después dándole esa forma negativa. Existe también una forma ampliada de la instrucción, que incluye el bit X, llamada NEGX.

La instrucción EXT se emplea para ampliar con el bit signo del operando de datos el tamaño del operando más grande. Así, si se ejecuta EXT.W D0



Cambio de estado
Los efectos de las instrucciones DIV, MUL y BCD sobre el registro de estado son los que aquí se ilustran. Es importante observar que en algunos casos la ausencia de una condición no se traduce en la limpieza del flag correspondiente, que quedará inalterado y, si se emplea de testio, puede dar una falsa lectura (la que corresponde a una operación previa)

donde D0=1111 0101 (FA, en hexa) dará D0=FFFA (en hexa). Esta instrucción es útil cuando se trabaja con números de precisión múltiple, en especial cuando se incluyen operandos más grandes de datos, como pueden ser las instrucciones de multiplicación y división.

Interpretación de patrones de bits

Antes de seguir con el examen de instrucciones más complicadas, debemos considerar el empleo de los modelos binarios de bits en un operando de datos de un byte. Por ejemplo, cuando D0=1001 0110 podemos suponer que se trata de un entero con un valor de –106 (recuerde que el valor se obtiene haciendo una inversión y sumando la unidad). Sin embargo, si el número no tiene signo y se asume el intervalo global binario con enteros positivos, el valor sería 96 (en hexa), o sea 150 en decimal (9×16+6×1). Los números sin signo son útiles si sabemos que sólo vamos a necesitar un intervalo grande positivo. Por ejemplo, el intervalo de direcciones de un ordenador puede considerarse como un intervalo de números positivos sin signo, y a condición de que no operemos con esos números mediante operadores con complemento a dos, así pueden tomarse.

Pero hay todavía una tercera interpretación del modelo de bits dado; es el sistema llamado BCD (*binary coded decimal*: decimal codificado en binario). Se trata de una codificación muy adecuada en la que todo grupo de cuatro dígitos binarios es considerado como el código de un dígito decimal. En este caso, nuestro ejemplo (D0=1001 0110) tendría en BCD el valor de 96 (1001=9 y 0110=6).

Debemos hacer tres importantes observaciones sobre esta codificación. Primera, lo fácil que es convertir números incluso muy grandes de decimal a BCD, o viceversa. Por ejemplo, el decimal 9 631 en BCD se escribe 1001 0110 0011 0001. La conversión, como se ve, es muy sencilla.

La segunda observación es que también resulta fácil definir la precisión de los números codificados de esta manera. La tercera se refiere a que codificamos dígitos del 0 al 9, por lo que los códigos sobrantes no tienen validez (es decir, los códigos que van desde el 1010, que es 10 en decimal, al 1111, que es 15 en decimal).

La importancia de estas observaciones se hará evidente cuando estudiemos las distintas instrucciones aritméticas del 68000. Veámosla por un momento en la operación de multiplicar. Si multiplicamos dos operandos de 16 bits, el modelo de bits resultante puede tener 32 bits de longitud. Usted mismo puede comprobarlo, pero entienda que para una palabra de n bits de longitud, el producto del número más grande, $2^{(n-1)}$, será $2^{2(n-1)}$, o sea, dos veces la longitud de la palabra original.

Para la instrucción de multiplicar en binario, disponemos por ello de dos operandos de 16 bits que dan como resultado 32 bits. Puesto que podemos operar con números con y sin signo, tenemos dos instrucciones independientes, la MULU (*multiply unsigned*: multiplicación sin signo) y la MULS (*multiply signed*: multiplicación con signo). Ambas instrucciones multiplican los dos operandos y ponen un resultado de 32 bits en el registro de datos de destino. De nuevo se observará que sólo se permiten modos de direccionamiento de datos para el operando fuente.

Así, por ejemplo, MULU #20,D0 cuando D0=XXXX 0003 (la X significa aquí 0 o 1) dará D0=0000 003C. Se notará que se emplea el registro entero de datos de 32 bits aun cuando no sea necesario en este ejemplo particular.

Del mismo modo, cuando D0=XXXX FFFF, MULU #$10,D0 dará como resultado D0=000F FFF0 pero con MULS se obtendría D0=FFFF FFF0. Esto es así porque el resultado es de signo ampliado a todos los 32 bits. Se observará que es posible comprobar fácilmente estos resultados dado que el multiplicador de 10 hexa equivale a un desplazamiento a la izquierda de 4, o a una multiplicación por 16.

Una consideración final sobre la instrucción para multiplicar se refiere a los códigos de condición. Los flags N y Z se activan según sea el resultado, pero los bits V y C son puestos a 0. Aunque no es posible que con operandos de palabras y resultados de una palabra de longitud obtengamos un desbordamiento, es interesante conocer si el resultado de palabra se desbordaría, porque entonces se sabrá fácilmente si el resultado puede ser truncado a una palabra en caso de necesidad.

Veamos un fragmento típico de programa que emplea la instrucción MULS. Supongamos que se desea convertir caracteres ASCII representativos de números decimales en palabras binarias. La primera operación será convertir el carácter de entrada en binario y sumar seguidamente este patrón de bits en el acumulador binario. Antes de efectuar la suma tenemos que asegurarnos de que las entradas anteriores fueron multiplicadas por 10 (pues se trata de entradas decimales). Así, por ejemplo, un posible programa sería éste:

```
SUB.B  #'0',D0   Forma el binario del carácter decimal
MULS   #10,D5    Multiplica la suma previa por 10 decimal
ADD.W  D0,D5     Suma el nuevo valor
```

En este ejemplo el carácter de entrada está en D0 y la nueva palabra binaria que representa en binario los caracteres de entrada está en D5.

Examinemos ahora la instrucción de dividir, DIVU (para números sin signo) y DIVS (para números con signo). Ambas instrucciones toman el entero de 32 bits que está en el registro de datos de destino y lo dividen por el operando fuente de 16 bits. El resultado se pone en el registro de datos de destino y en los 16 bits inferiores colocando cualquier resto en los 16 bits más significativos. Por ejemplo, si D0=0000 0005 (en hexa):

```
DIVU   #3,D0
```

dará D0=0002 0001 (1 con resto 2). Observe que el operando destino es un operando completo de 32 bits, lo que significa que primero puede necesitarse una palabra larga simple, o una larga EXT.L ampliada con signo. Dado que es posible el desbordamiento (p. ej., dividiendo por la unidad una palabra entera de 32 bits obtendremos como resultado algo más de 16 bits), se tendrá en cuenta oportunamente el bit de desbordamiento.

Un sencillo algoritmo que nos permita contar todos los números de una lista de palabras terminados en cero podría ser el siguiente:

```
                ORG $1000
        LEA     LIST1,A0
PUNTERO                         establece
                                  dirección
        CLR.L   D0              borrado total
        CLR.L   D1
LOOP    ADDQ    #1,D1           contador bucle
                                cuenta núm.
        ADD.W   (A0)+,D0          palabras
        TST     (A0)            y suma
                                comprueba
        BNE     LOOP              fin de lista
        DIVU    D1,D0
                                divide suma
        TRAP    0                 por contador
LIST1   DC.W    1,2,3,4,0
```

Se notará que se han empleado borrados de largas palabras en la parte de inicialización de este programa, ya que la suma será tratada como operando completo de 32 bits. En D1 se coloca un contador y la palabra apuntada A0 se añade a D0 con el direccionamiento de postincremento. Seguidamente se comprueba por medio de la instrucción TST la siguiente palabra que está en LIST1. Esta instrucción se limita a colocar códigos de condición listos para la instrucción BNE, y no altera operando alguno.

La instrucción TST es necesaria porque la instrucción anterior ADD afectará los códigos de condición según sea el resultado de sumar los operandos de datos en D0, y no según el valor de los datos cargados por el puntero A0. La instrucción BNE (*Branch if Not Equal to zero:* bifurcar si no es igual a cero) provocará la vuelta a LOOP mientras no sea cero el siguiente elemento de la lista. Cuando ocurra que es igual a cero, D0 contendrá la suma y D1 el número de elementos no cero (el contador de datos).

Cuando se haya ejecutado el bucle (LOOP) cuatro veces, los valores contenidos en los registros de datos serán:

```
D0=0000 000A (en decimal,10)
D1=0000 0004
```

Por tanto, una vez ejecutada la instrucción DIVU, entonces D0 contendrá 0002 0002 (10 ÷4=2 y me llevo 2).

Una última observación sobre las instrucciones para dividir es que una división por cero provocará un *trap* (una interrupción software en el monitor del sistema), ya que un número infinito ciertamente no es representable en 16 bits. Si no queremos caer en esta "trampa", hemos de establecer un testigo cuando el divisor sea cero.

Por ejemplo:

```
TST     D1
BEQ     ERROR
DIVU    D1,D0
```

Ya hemos visto la conveniencia del BCD para la representación de números decimales. Pero se ha de notar que un dígito BCD válido corresponde a un dígito en hexa (4 = 0100 binario = 4 hexa = 4 BCD), por lo que las constantes BCD equivalen a constantes hexa. Por ejemplo:

```
MOVE.B      #$54,D0   pone 54 BCD en D0
```

Pero la analogía acaba aquí. Al sumar dos dígitos BCD en aritmética binaria, por ejemplo, obtendremos una respuesta incorrecta:

```
0100 1001   (49 en BCD) suma
0000 0001   binaria
            (1 en BCD)
-----------
0100 1010
```

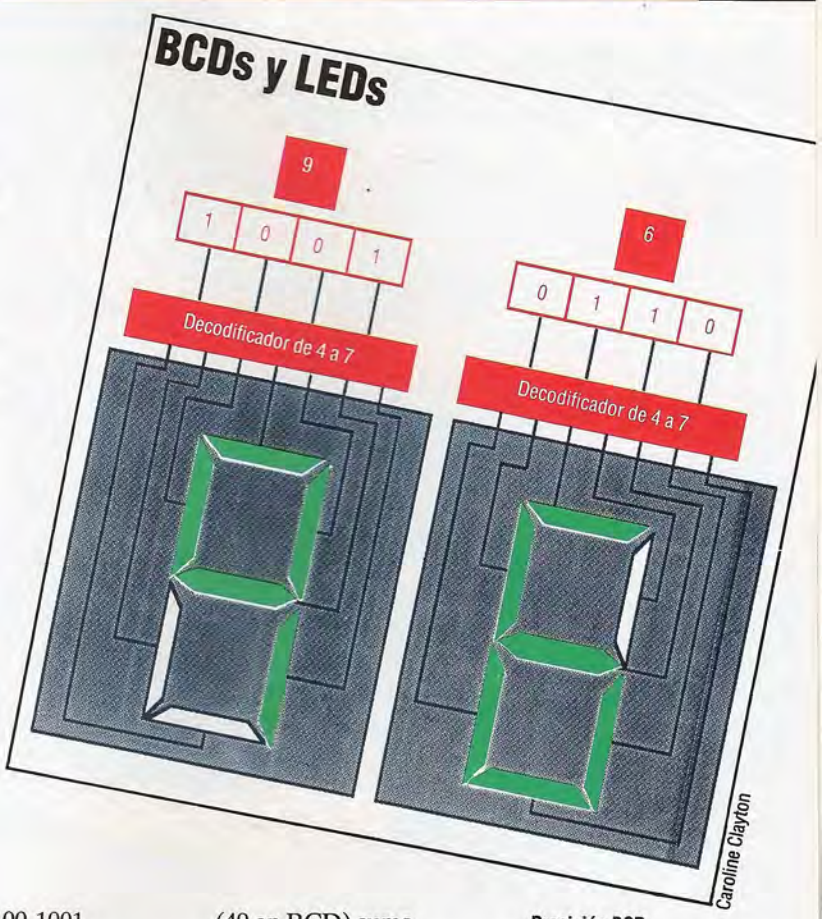
Obtenemos un dígito BCD ilegal como dígito BCD menos significativo. Esto no debe preocuparnos, sin embargo, dado que el 68000 posee un grupo de instrucciones BCD para sumar (ABCD), restar (SBCD) y negar (NBCD).

Nuestro estudio de las instrucciones BCD se limitará a la instrucción ABCD, que suma el byte fuente (dos dígitos BCD) al byte destino, con el bit X, poniendo la suma en el destino. Los únicos modos de direccionamiento permitidos son los pares de registros de datos y los pares de registros de direcciones predecrementadas. Así, por ejemplo, si D0=44 BCD y D1=01 BCD, entonces tras ejecutar

```
ABCD    D0,D1
```

D1 contendrá 45 BCD. Si el bit X que está en el SR estaba activado (equivalente al arrastre para BCD), el resultado sería 46. Igualmente, si sumamos 1 a 99 en D0, el resultado sería 00 con el bit activado en el registro de estado. Con esto se puede colegir que si bien estamos limitados a operandos de un byte, podemos ampliar fácilmente la precisión de nuestros cálculos empleando el bit X para llevar el arrastre a componentes del byte más significativo.

Los códigos de condición X, C y Z se activan para todas las instrucciones BCD. Sin embargo, es de notar que la instrucción NBCD (Negar en BCD) permite modos de destino alterables de datos preferentemente a pares predecrementados o simples datos.



Precisión BCD
Lo mismo que para operar con enteros con y sin signo, el 68000 posee un juego de instrucciones para la manipulación de datos tomados en forma BCD (decimal codificado en binario). Esto es especialmente útil cuando se necesita una precisión aritmética total. Pero también ofrece otras ventajas en distintas aplicaciones. Por ejemplo, en el manejo de visualizaciones LED de siete segmentos. El hardware que decodifica los datos en BCD emplea un decodificador de líneas de 4 por 7, como en el dibujo, resultando más sencillo que la decodificación de números binarios

Pantallas mil

Tesoro enterrado
Quo vadis, de Softek, es una aventura en gráficos con un premio real. Trasladándose por los escenarios de la aventura y resolviendo tres acertijos, el jugador puede recoger pistas que lo orientarán en la búsqueda de un cetro de oro y plata que se halla enterrado

"Quo vadis" introduce al jugador en un misterioso y fascinante ambiente situado en un mítico medievo, ¡y posee 1 024 pantallas!

El concepto "aventura recreativa", que combina elementos de los programas de aventuras y recreativos tradicionales, ha dado un nuevo giro desde que algunas empresas de software han centrado su interés en otra tradición que ha alcanzado creciente difusión desde la publicación del libro *Masquerade* (Mascarada), de Kit Williams. En esta obra se proporcionaban a los lectores numerosas pistas visuales y escritas que conducían a la ubicación real de una liebre de oro, enterrada en algún lugar de la campiña inglesa. La primera empresa que introdujo esta idea en el mercado de software para ordenadores fue Automata con su juego *Pimania*, que ofrecía como premio un objeto de oro. Después vino *Hare raiser* (Criador de liebres), un juego por ordenador similar en concepto al *Masquerade* y que incluso ofrecía un premio idéntico, que se le compró a su descubridor original y luego se volvió a ocultar.

Softek continuó esta tradición de ofrecer premios en su juego de aventuras *Quo vadis*, que ofrecía un cetro de oro y plata para la primera persona que resolviera los enigmas y escapara del juego. La esfera de acción del programa constituía una sensación adicional, dado que ofrecía una superficie de juego que cubría 1 024 pantallas diferentes.

El juego es una aventura recreativa en la que el usuario asume el papel de un caballero que busca un cetro que se halla escondido en algún punto de un complejo sistema de cavernas subterráneas. Es este laberinto subterráneo lo que ocupa las 1 024 pantallas; está constituido, además, por 118 cuevas, cada una de las cuales es más grande que la superficie total de juego de muchas de las aventuras recreativas normales.

Para desplazarse a través de las cavernas es preciso saltar de una saliente rocosa a otra o, a veces, ascender o descender trepando por unas cuerdas. Además hay que salvar diversos peligros, como las simas de lava de la parte inferior de las cuevas, ya que caer en ellas puede acarrear fatales consecuencias. Con estas características de la plataforma del juego se combinan diversos elementos del juego de acción tradicional. Las cavernas están habitadas por 38 clases diferentes de monstruos, quienes atacan al caballero cuando éste penetra en una cueva, y se utiliza la palanca de mando para disparar en una de ocho direcciones.

Si el caballero resulta herido por los monstruos pierde puntos de energía, y cuando éstos hayan disminuido de 100 a 0 el juego termina. El caballero, no obstante, puede reaprovisionarse de energía mediante cofres de tesoros que encuentra en ciertos escenarios.

En algunas cavernas, los enigmas escritos sobre las paredes proporcionan pistas que pueden ser útiles para hallar el cetro.

Quo vadis es impresionante no sólo en virtud de su tamaño. El juego no se compone de pantallas separadas que se vayan superponiendo unas sobre otras, sino que la imagen se va desplazando con gran uniformidad para revelar un poco más del paisaje circundante. De este modo, da la impresión de penetrar lentamente en un mundo realista y misterioso.

Dada la inmensidad de *Quo vadis*, era inevitable que hubiera de resentirse algún aspecto del juego; no obstante, es justo afirmar que los gráficos y el sonido responden por completo a la media. Pero cuanto más se practica el juego los gráficos van proporcionando más atmósfera, y las brillantes antorchas, las velas mortecinas, las oscuras escaleras y las húmedas paredes contribuyen en gran medida a crear un convincente ambiente subterráneo.

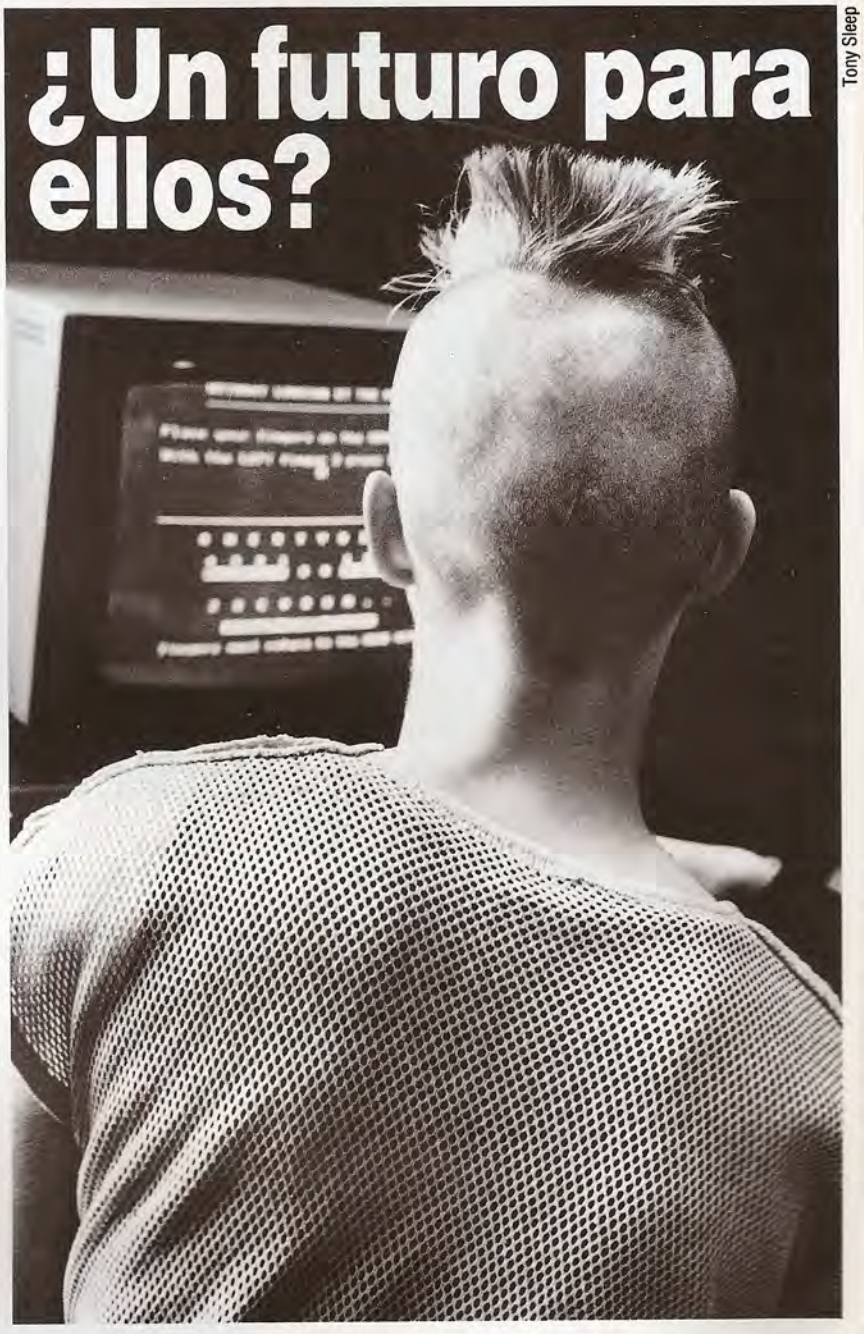
El hecho de que un juego tan grande pueda retenerse en la memoria del Commodore 64 se debe atribuir a las potentísimas técnicas de compactación de código utilizadas en la programación. Sólo se emplean seis bytes para cada pantalla, dejando, por lo tanto, espacio suficiente para las rutinas del juego.

Poco después de la aparición del juego, Softek sacó en disco un *Quo vadis generator* para quienes considerasen insuficientes las 1 024 pantallas ya incluidas. La empresa afirma que mediante el uso de este disco ahora hay disponibles millones de pantallas diferentes.

Quo vadis: Para el Commodore 64
Editado por: Softek International, 12/13 Henrietta St., Covent Garden, London WC2E 8LH, Gran Bretaña
Formato: Cassette o disco
Palanca de mando: Necesaria

¿Un futuro para ellos?

En este primer capítulo de esta nueva serie, analizaremos las aptitudes que se requieren para desenvolverse en el campo laboral de la informática



Tony Sleep

La primera decisión que ha de adoptar el postulante a un puesto de trabajo en el medio informático es en qué parte de esta compleja industria desea trabajar, teniendo presente que los buenos trabajos escasean en todas partes. En esta serie consideraremos carreras que difieren tanto entre sí como el día de la noche, ofreciendo cada una de ellas una gratificación laboral diferente: sueldos, perspectivas de futuro, compromiso técnico y satisfacción personal.

Un punto a considerar es que normalmente una carrera de informática sigue un rumbo fijo. Si usted decide avanzar en una dirección, entonces ése es el camino en el cual seguirá durante cierto tiempo. Esto es particularmente cierto si la firma que lo ha contratado realiza una fuerte inversión en su formación profesional o si usted obtiene conocimientos y destreza especializados.

En informática la especialización laboral es muy amplia, aunque hay algunas categorías comunes. Cada trabajo plantea exigencias diferentes de personal y a su vez ofrece diferentes incentivos y perspectivas. Los fabricantes de ordenadores y electrónica, por ejemplo, requieren tres tipos de personal, todos ellos trabajando en hardware, y lo mismo puede decirse de las telecomunicaciones, un área de la industria que cada vez adquiere mayor relevancia. Existen trabajadores de línea de producción como los de cualquier otra fábrica, con salarios algo superiores pero similares perspectivas; ingenieros de producción, cuya tarea es asegurar que el producto final funcione, y personal de investigación y diseño.

Diseño de ordenadores

El diseño de ordenadores, ya sea de microprocesadores o de sistemas completos, es una especialidad destinada a quienes poseen una relativa experiencia (o para auténticos genios), pero la *ingeniería* también está abierta a quienes poseen un talento medio. Ambos trabajos requieren personal entrenado. Los proveedores que han obtenido éxito con sus productos tienden a emplear más personal para desarrollar la siguiente gama de éstos.

Estos trabajos de diseño son territorio exclusivo para graduados, quienes exigen salarios elevados, seguridad y prestigio. En la industria los graduados son más numerosos que los no graduados en una proporción de nueve a uno, según el organismo británico COSIT (*Computing services industry training council*: consejo de adiestramiento de la industria de servicios informáticos). Un curso académico que se aconseja seguir es uno de ingeniería electrónica o informática en una universidad o instituto politécnico. Exigiendo niveles básico y avanzado, en tres años se enseña al alumno teoría básica y práctica avanzada.

Los niveles avanzados exclusivamente en muy raras ocasiònes son suficientes: los patrones esperan que uno sea capaz de arreglárselas con una placa de circuito impreso, además de conocer los principios sobre los que opera un microordenador.

Es probable que ni siquiera los programadores más brillantes ni los aficionados a la electrónica empiecen diseñando un PC o un chip. Hay otras aptitudes que son esenciales antes de que se le permita diseñar siquiera una pieza pequeña. Un proyecto de diseño por lo general se comparte entre un equipo formado por una docena o incluso varios centenares de trabajadores, desarrollando y completando subsecciones del proyecto completo.

Los patrones tienden a pensar que la experiencia académica es una ventaja en informática y electrónica porque permite que los graduados se adapten a la velocidad a la cual las nuevas tecnologías se aplican a los productos de hoy en día. Existe, sin embargo, la desafortunada creencia de que las mujeres son incapaces de adecuarse a los cambios:

¿Perspectivas más brillantes? La industria informática ofrece una gran cantidad de posibilidades para quienes buscan trabajo, pero es poco probable que usted consiga un puesto de trabajo contando sólo con experiencia en ordenadores personales. Una formación profesional en un centro educativo cualificado representa un buen comienzo para una carrera que, aunque inicialmente sea variada, a menudo implicará una especialización en un campo determinado. En casos extremos, esto puede conducir a perspectivas de empleo más reducidas en el futuro, a menos que se haga un serio esfuerzo para mantenerse al día en cuanto a innovaciones de hardware y software.

Personal de ventas al detalle

Un número creciente de jóvenes se inicia en el mundo laboral trabajando en una tienda de ordenadores o en una casa de sistemas con puertas a la calle a través de la que se venden al público pequeños sistemas, periféricos, suministros y software. Las ventas en las tiendas no suelen ser frecuentes, y su relativa importancia respecto a otras vías de comercialización las sitúa en el extremo inferior del mercado.

El trabajo permite que el recién llegado aprenda sobre ordenadores personales, pero pocas veces le permite familiarizarse con una amplia gama de sistemas o técnicas de programación. No obstante, para algunas personas es un punto de entrada válido hacia una carrera en el campo de las ventas y marketing con una firma importante. Tales trabajos no suelen estar bien remunerados y exigen cumplir largas jornadas

la proporción de hombres y mujeres en la industria es de cuatro a uno a favor de los primeros (cifras del COSIT). Afortunadamente, este errado concepto está siendo cuestionado con mayor fuerza cada día.

Sin embargo, los entusiastas pueden convertirse en personal de ingeniería con poco o nada de entrenamiento formal. Siempre se solicita personal que sepa instalar una pieza de equipo electrónico, pero el entrenamiento y el salario son limitados y las perspectivas para el futuro son poco halagüeñas. Las posibilidades por lo general se limitan a las firmas de microordenadores más pequeñas.

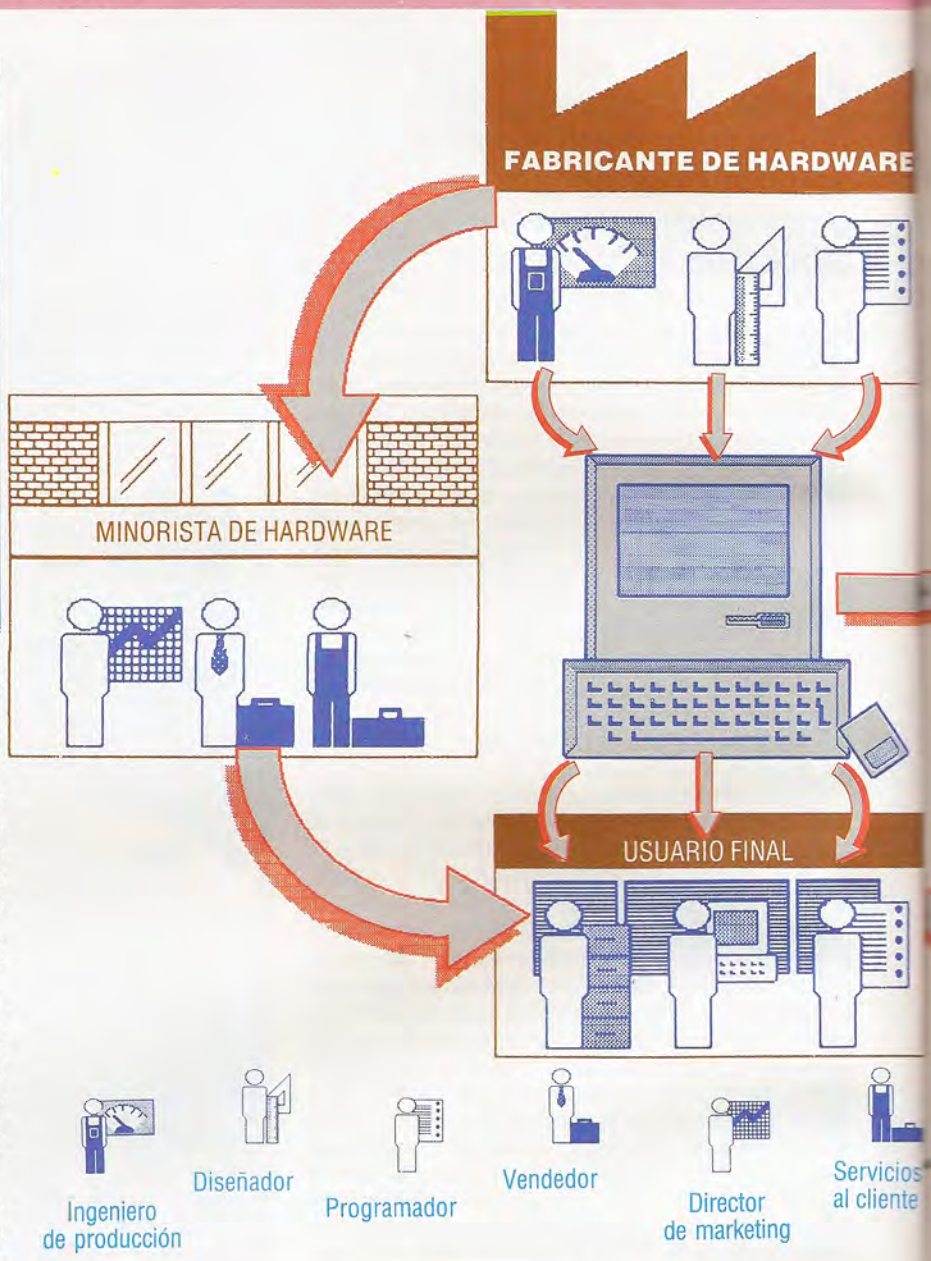
En la industria del ordenador, el software y el mantenimiento son las principales fuentes de empleo. Los programadores (definidos de forma genérica de modo de incluir a otro sector de personal de apoyo) representan más de la mitad del personal; el personal de ventas y marketing constituye la tercera parte.

La mayoría de los programadores continúan trabajando ya sea en operaciones de proceso de datos (*data processing*: DP) —que ahora tienden a desarrollarse en servicios de información de administración (*management information services*: MIS)— o bien con proveedores de ordenadores, casas de software o agencias de informática.

El DP constituye la vía más utilizada para iniciar una carrera en informática, pero está disminuyendo en importancia a medida que la operatoria de los ordenadores se vuelve más fácil y más automática.

Ciertas categorías laborales de bajo nivel, como operatoria, programación o análisis de sistemas básicos, quizá se puedan considerar como un paso en la carrera, no como un fin en sí mismas.

En la actualidad la programación es un tema sumamente especificado. Es poco probable que un experto en un lenguaje de microordenador como el FORTH o el PASCAL pueda captar de inmediato las particularidades más sutiles del COBOL o el FORTRAN en un ordenador central, y la situación se complica por el predominio del PASCAL y el COBOL en las áreas de gestión, mientras que el FORTH y el FORTRAN son más populares en las áreas técnicas. El C también está adquiriendo cada vez mayor importancia, en particular en lo que se refiere a la programación de sistemas.



Los sueldos de los informáticos

Categoría	Septiembre 1984	Marzo 1985
Director de informática	6.6	8.9
Director de DP	9.6	8.6
Analista de sist. senior	9.0	10.5
Analista senior	7.3	7.9
Programador senior	4.8	8.6
Programador analista	6.5	7.8
Programador	5.7	1.6
Ejecutivos	**6.6**	**7.6**

Estructura salarial
En esta tabla se refleja la cambiante tabla salarial para una gama de personal de ordenadores en Gran Bretaña. Las cifras, que representan algunos cambios notables en poco más de seis meses, están sacadas del Regional Reward Survey, informe dado a conocer en septiembre de 1985. Las cifras para los aumentos salariales anuales están promediadas a través de todo el país, y se deben comparar con la inflación, que durante el mismo período fue de varios puntos

Una proporción significativa de las personas de la industria del ordenador (quizá una tercera parte del personal) trabaja para suministradores de ordenadores. Además de la fabricación, el diseño y el apoyo de mantenimiento hay muchísimo personal de ventas. Éste puede trabajar directamente tanto para un proveedor de ordenadores como para un

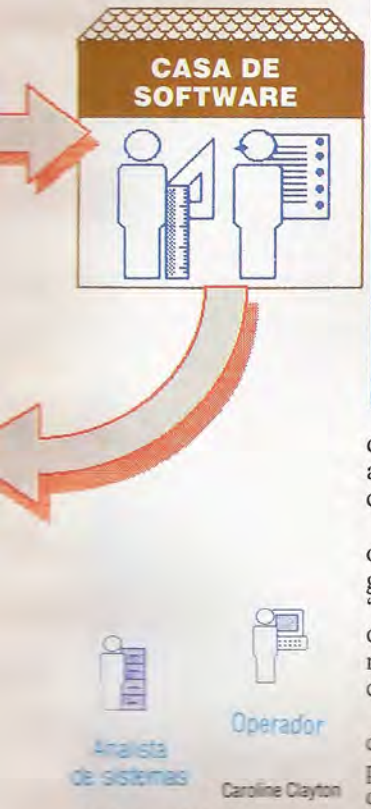
Ingenieros de mantenimiento

Un ingeniero de mantenimiento requiere conocimientos generales de ingeniería eléctrica y un conocimiento básico de programación. El sueldo y las condiciones son buenos, en compensación por el sacrificio que supone el tener que trabajar a través de una vasta zona geográfica y tener que estar disponible las 24 horas del día. Los ingenieros con base en la fábrica tienen una tarea más sencilla, descubriendo y arreglando fallos en las máquinas que les traen o que se hallan en proceso de fabricación. La tarea que realiza el ingeniero de mantenimiento exige, sobre todo, una gran paciencia: hallar un chip defectuoso en una habitación llena de ordenadores centrales es agotador. Un conocimiento básico de programación ayuda al ingeniero a rastrear un fallo en un conjunto de placas de circuito impreso o incluso en una placa individual. Este tipo de trabajo tenderá cada vez más a desaparecer en la industria a medida que los ordenadores vayan siendo más fiables y los sistemas de diagnóstico en línea se encarguen de buscar los errores. En el futuro cercano el ingeniero de mantenimiento de campo podría ver limitado su trabajo a entregar una pieza de recambio, siendo el propio ordenador el que localice su avería y solicite el recambio adecuado

Escribir juegos para ordenador

La era del adolescente millonario escritor de juegos para ordenador ha terminado (si es que había llegado a iniciarse). La idea de que cualquiera podía escribir un juego *best-seller* que, como un disco de música *pop*, vendiera millones de copias de la noche a la mañana, no estuvo sustentada más que por un puñado de ejemplos cuestionables.

Los juegos los escriben una cantidad significativa de programadores independientes, muchos de los cuales, siendo muy jóvenes, se introdujeron en el negocio al aparecer ordenadores personales baratos y potentes y haber una gran demanda de software empaquetado, incluyendo juegos. Son pocos los programadores que trabajan de forma aislada. La mayoría de ellos trabajan para empresas pequeñas bajo contrato o venden a una empresa los derechos para fabricar, comercializar y distribuir sus programas. Una carrera de características tan independientes plantea problemas importantes. En primer lugar, es preciso trabajar todos los días, tanto si se tienen ganas como si no, para prosperar. Muchos programadores independientes tienen otro empleo en relación de dependencia y trabajan como independientes sólo a tiempo parcial. En segundo lugar, ha habido un alarmante número de quiebras entre pequeñas casas de software dedicadas a los mercados de juegos y ordenadores personales. Por último, escribir juegos no es, en realidad, una carrera. No hay nadie que enseñe a programar. Existe una tendencia a estancarse y retornar a lo que uno hace mejor y, por tanto, a no desarrollar aptitudes nuevas



Caroline Clayton

Relaciones industriales
El hardware continúa siendo el centro de la industria del ordenador, pero quien busque empleo quizá encuentre una fuente más segura especializándose en una de las actividades (marketing, p. ej.) que giran alrededor del mismo. El diagrama ilustra las diversas relaciones existentes entre el fabricante, el minorista, el fabricante independiente y el usuario final

distribuidor, una casa de sistemas o de software, o alguno de los muchos comercios minoristas de microordenadores.

Al igual que en otros sectores, la industria del ordenador divide al personal de ventas en tres categorías. En primer lugar está el personal de venta "directa", si bien los que "viajan" recogiendo pedidos directos constituyen una novedad relativa. A la mayoría de ellos se les encarga el manejo de varias cuentas.

En segundo lugar está el personal de marketing, cuya labor consiste en generar contactos con compradores de ordenadores ya existentes (en el caso de grandes sistemas) o, más raramente, con compradores noveles. Una empresa de ordenadores centrales como IBM entrena a estos empleados de marketing para que sean el verdadero personal de ventas, hablando directamente con el gerente de DP/MIS, el director gerente o la junta del potencial cliente. Un proveedor de microordenadores pequeño contratará un director de marketing para asegurar que los distribuidores y minoristas tengan existencias de sus productos.

El tercer trabajo de ventas implica unir entre sí los diversos elementos del aprovisionamiento y apoyo de hardware, software y sistemas. En la industria del ordenador, los papeles de administración como éste están todavía en manos de ex programadores o ingenieros.

El adiestramiento se encuentra inmerso en un laberinto de siglas; por este motivo es esencial, antes de iniciar un curso, averiguar lo que proporcionará cada conjunto de iniciales. En Gran Bretaña, por ejemplo, país muy avanzado en este campo, es la Manpower Services Commission (MSC: comisión de servicios de recursos humanos) el organismo que se encarga de la formación profesional y de impartir cursos básicamente sobre programación y técnicas de software bajo el auspicio del Training Opportunities Program (TOPS: programa de oportunidades de adiestramiento), diseñado en gran parte para el reciclaje profesional.

El TOPS ha sido criticado por su lentitud para actualizar sus cursos de modo de incluir métodos modernos, pero al estar sobre la mesa las propuestas del comité Alvey del Department of Trade and Industry (Departamento de Comercio e Industria) para corregir este defecto, parece ser que seguirá siendo la principal fuente de personal para programación de ordenadores después de las universidades y los politécnicos.

Los ITEC (Information Technology Education Centres: centros de educación en tecnología de la información) constituyen otro plan de captación, habiéndose creado más de 150 centros desde 1982 a lo largo y ancho de las islas británicas para proporcionar adiestramiento a los jóvenes que se encuentran en la encrucijada de terminar la escuela y conseguir un primer empleo. Bajo el YTS (Youth Training Scheme: plan de adiestramiento juvenil) se remunera a las personas de entre 16 y 19 años de edad que asistan a las clases de formación.

Mientras tanto, los fabricantes de ordenadores siguen estando entre quienes ofrecen los más altos salarios, y se les considera los mejores patrones en cuanto a condiciones generales, formación y perspectivas para el futuro que ofrecen. Por lo general esto es aplicable a los distribuidores y a las casas de software, pero es exactamente lo contrario en el caso de los comercios minoristas de ordenadores.

Lo pequeño es bello

Iniciamos esta serie dedicada al estudio del sistema operativo Unix considerando su historia y la filosofía de su diseño

Además de funciones básicas de E/S y facilidades para gestión de archivos y memoria, los sistemas operativos proporcionan un juego de *herramientas* o *utilidades* que crean un entorno de trabajo adecuado para el usuario. Estas utilidades pueden ir desde compiladores para diversos lenguajes hasta editores de texto y formateadores de impresión. De hecho, los principales puntos fuertes del Unix, además del hecho de ser un sistema operativo simple y a la vez potente y elegante, son la gran variedad de herramientas que proporciona y el modo en que permite utilizar la salida de un programa como entrada para otro. Al hacerlo, permite conectar entre sí varias herramientas simples para llevar a cabo las tareas más complicadas. La filosofía del Unix se puede enunciar con las siguientes máximas:

1. Escribir programas pequeños y simples que realicen bien una tarea.
2. Esperar utilizar la salida de cualquier programa como la entrada para otro, aun cuando el segundo programa no se haya escrito todavía.
3. Escribir siempre programas nuevos para realizar tareas nuevas, en vez de modificar los antiguos; de ese modo usted acrecienta la biblioteca de herramientas disponibles.

Hay, sin embargo, una dificultad. La idea de contar con una amplia variedad de herramientas que se pueden interconectar para llevar a cabo tareas complejas es lógica para los programadores y otros profesionales de la informática, pero no hace que el sistema sea muy "amable" con los usuarios inexpertos. Si usted desea hacer un uso exhaustivo de un sistema Unix, son muchas las cosas que ha de recordar.

La única forma de superar esto es proporcionar un sistema que puedan utilizar las personas sin experiencia: una "fachada" amable con el usuario que le esconda la complejidad y potencia completa del Unix. No obstante, es virtualmente imposible hacer esto de modo tal que la potencia permanezca disponible en su totalidad; por este motivo, sistemas Unix se encuentran fundamentalmente en instituciones académicas antes que en sistemas de gestión o personales.

El Unix lo desarrolló casi por accidente el programador Ken Thompson cuando trabajaba en los Laboratorios Bell, que constituye el núcleo de investigación de la gigantesca AT&T Corporation norteamericana. Thompson se encontraba realizando una simulación del movimiento planetario en un ordenador GE645, que estaba ejecutando uno de los primeros sistemas operativos multiusuarios denominado Multics. Los problemas propios de desarrollar software utilizando un entorno hostil en un gran ordenador lo llevaron a transferir sus esfuerzos a un pequeño ordenador DEC, un PDP-7. Para hacerlo, escribió un OS que daba la mayoría de las facilidades del Multics en la máquina más pequeña. Tuvo tanto éxito que atrajo la atención de Dennis Ritchie y otros colaboradores, quienes en 1971, en los mismos laboratorios, desarrollaron el sistema convirtiéndolo en la primera versión totalmente operativa de Unix.

Al igual que la mayoría de los sistemas operativos de la época, el Unix se escribió primero en ensamblador, y operó adecuadamente sólo en el miniordenador DEC, fundamentalmente en la serie PDP-11. Sin embargo, Dennis Ritchie también estaba trabajando en un lenguaje llamado B, un desarrollo del BCPL, el lenguaje de alto nivel que ofrecía casi el mismo control directo sobre el hardware que el ensamblador. Enseguida éste se convirtió en el C y el Unix se reescribió en este nuevo lenguaje. Sólo una pequeña fracción del Unix continúa estando escrita en ensamblador, lo que facilita la transferencia del código a máquinas diferentes: todo cuanto se necesita es un compilador de C. En la actualidad el Unix se ejecuta virtualmente en cualquier tipo de máquina, desde los mayores ordenadores centrales hasta los micros de 16 bits, y algunas versiones limitadas de Unix incluso se pueden ejecutar en micros de ocho bits.

Hasta hace poco tiempo era bastante difícil y costoso adquirir el Unix para uso comercial, pero AT&T tiene la política de conceder licencias casi gratuitas para las instituciones académicas, de modo que el uso del Unix se ha generalizado en facultades y universidades. En el mundo comercial se ha abierto camino especialmente en microorde-

PC PAM
Aunque el Unix por lo general se suele encontrar en miniordenadores, Hewlett-Packard ha implementado una versión, llamada HP-UX, para ejecutar en su Integral PC. Este potente portátil multitareas incluye una pantalla de plasma, impresora integral de chorro de tinta y 768 Kbytes de memoria, y opera bajo un sistema WIMP denominado PAM

Liz Heaney

La historia del Unix

Al haber sido desarrollado por un pequeño equipo, el Unix continúa siendo sensible y compacto. Muchas de sus excelentes facilidades se han abierto camino en sistemas operativos relativamente nuevos, tales como el MS-DOS

1969 Dennis Ritchie y Ken Thompson, de los Laboratorios Bell de Estados Unidos, desarrollan un OS a su propia conveniencia para operar en un ordenador PDP-7. Lo bautizan "Unix" haciendo un juego de palabras con el gran OS Multics en uso en aquel entonces

1971 Se transfiere a ordenadores PDP-11

1973 Las versiones originales estaban escritas en lenguaje ensamblador. Casi en su totalidad se recodificó en c, haciéndolo fácilmente portable a otros ordenadores, para lo que sólo se requiere un compilador de c

1974-75 Se introduce el PWB/Unix, que, en esencia, es una versión grande del Unix

1977 Se desarrolló la versión 7 para liberar al Unix de características dependientes de la máquina. Ésta es la versión que se utiliza actualmente

nadores como Xenix, Cromix, Onyx, Idris, Coherent y OS-9, todos los cuales tienen muchas características en común con el Unix. Ahora, sistemas operativos como el CP/M y el MS-DOS están adoptando características estilo Unix.

El Unix puro tal vez no llegue nunca a ser un éxito comercial, pero ha causado un impacto profundo en el desarrollo de sistemas operativos y seguirá siendo importante durante mucho tiempo. Las versiones actuales incluyen la versión 7 de Laboratorios Bell, que pronto se sustituirá por la versión 5 y el derivado desarrollado en la Universidad de California en Berkeley, que se ha extendido en la versión 4.2. Todos los ejemplos citados aquí están tomados de la versión Berkeley 4.2, pero se aplican sin dificultad a la mayoría de las demás versiones.

Una característica fundamental del Unix es el caparazón, la parte del OS con la que se comunica el usuario. Es mucho más que el simple procesador de comandos de muchos sistemas, porque tanto el proveedor del sistema como el usuario individual pueden confeccionarlo a medida para incorporar nuevas instrucciones y una interface para el usuario distinta si así se requiriera. También proporciona muchas de las facilidades de un lenguaje de programación como el c, y junto con la gran variedad de herramientas disponibles se pueden escribir programas largos y complejos sin utilizar un lenguaje de programación en absoluto.

Inicialmente veremos algunos de los comandos estándares y luego veremos las formas en que se crean comandos y nombres nuevos para comandos ya existentes. El formato de la mayoría de los comandos Unix es el mismo:

```
nombre_instrucción opciones archivo_o_
nombre_directorio
```

donde cada opción empieza con un signo menos. Un hecho importante a recordar sobre el Unix es que es sensible a los tipos de letra, es decir, los nombres "JUAN", "Juan" y "juan" se considerarán todos diferentes. Es un error común utilizar letras mayúsculas para un comando y encontrarse con que éste no funciona (la mayoría de los comandos Unix estándares utilizan letras en minúsculas).

La primera etapa en la utilización de un sistema Unix (o, para el caso, de cualquier otro sistema multiusuario), es *conectarse* (hacer *log in*). Esto se realiza automáticamente siempre que se conecta el sistema, pero a partir de entonces se puede hacer en cualquier momento digitando la instrucción login. El sistema le proporcionará el siguiente aviso:

```
login:
```

En cuyo momento usted debe digitar su nombre de identificación de usuario exclusivo. El sistema responde con:

```
password:
```

en cuyo punto usted digita su propia contraseña, que no se refleja en la pantalla. Las contraseñas normalmente tienen al menos seis caracteres de longitud. Si comete un error al entrar su contraseña, la debe repetir, pero para impedir que personas no autorizadas descubran contraseñas por el método de ensayo y error el sistema podría negarse a dejarlo probar otra vez al cabo de un cierto número de errores. Una vez completado el procedimiento, por lo general el sistema emitirá un mensaje de bienvenida y, finalmente, el aviso del OS.

En esta etapa el Unix ejecutará el propio programa de caparazón hecho a medida por usted y contenido en un archivo especial, .login, que entre otras cosas podría definir su propio aviso personal. El aviso Unix estándar es %. Ahora a usted se lo reconocerá como un usuario autorizado y el Unix le habrá conectado automáticamente a su propia área del disco donde se conservan todos sus archivos. Es posible acceder a archivos de otras áreas, en particular aquellos designados como públicos, pero, si se requiriera, usted podría proteger archivos y áreas determinadas para impedir el acceso no autorizado. Si desea cambiar su contraseña, lo que es aconsejable de vez en cuando, podría hacerlo digitando la instrucción passwd, que le solicitará que digite primero su contraseña antigua y después su nueva contraseña dos veces.

La mejor forma de conocer a un nuevo sistema de ordenador es probándolo y el Unix proporciona dos útiles facilidades para ayudarle a conocer el sistema. El Unix es un caso casi único en sistemas operativos por el hecho de que incluye un manual en línea con entradas para virtualmente todas las instrucciones o temas que usted pudiera querer utilizar. La instrucción que permite al usuario ver una entrada del manual es:

```
man nombre_tema
```

Para los usuarios novatos, el Unix incorpora un programa de autoformación que se inicializa mediante la instrucción learn, que ofrece lecciones sobre diversos temas.

Para dar por terminada una sesión con el Unix, se debe impartir la instrucción logout; ésta le desconectará del sistema hasta su próximo login.

Juega la banca

Nos dedicaremos a considerar las rutinas que permiten que el ordenador replique inteligentemente al jugador

Después de que el jugador ha completado su mano, le llega su turno a la banca. Ésta disfruta de varias ventajas en la versión de las reglas del veintiuno que hemos adoptado para nuestro programa. En primer lugar, la banca conoce los marcadores que han obtenido los jugadores y, por tanto, no tiene que aventurar peticiones de cartas arriesgadas que pudieran hacer que la mano se pasara. Segunda ventaja: sólo necesita empatar al jugador para ganar la vuelta y quedarse con su dinero. Para atenuar un poco estas ventajas, el programa no permite que la banca queme una mano de 12, 13 o 14.

En esta etapa del juego la pantalla visualizará los naipes del jugador y los dos naipes que se repartieron inicialmente a la banca. El primero de éstos se repartió cara abajo, de modo que la siguiente tarea del programa es dar vuelta el naipe. La forma más simple de hacerlo es borrar los naipes de la banca, utilizando la rutina de borrado general de la línea 670, y después reimprimirlos boca arriba estableciendo CN y SU (los números de naipe y palo) en los valores retenidos en la sección de la banca de la matriz de manos, HD(,,), y por último llamar a la rutina de visualización de naipes que desarrollamos anteriormente en esta serie. Todo esto se lleva a cabo entre las líneas 150 y 170 del bucle principal del programa. Ahora el programa está en condiciones de jugar automáticamente la mano de la banca.

Si el jugador ha pasado, no es preciso emprender

El turno del ordenador

Gama Amstrad CPC

```
130 REM **** turno del ordenador ****
140 pl=2
150 ep=20:GOSUB 670:REM borrar naipes
160 cn=hd(pl,1,1):su=hd(pl,1,2):GOSUB 1000:REM
    reimprimir
170 cn=hd(pl,2,1):su=hd(pl,2,2):GOSUB 1000:REM
    reimprimir
180 GOSUB 2500:REM la banca pide etc
190 REM **** ganar o perder ****
200 IF bw=1 THEN GOSUB 700:PEN negro:PRINT"lo siento
    pero pierdes":GOTO 300
210 GOSUB 700:PEN negro:PRINT"ganas tu"
300 a$="":WHILE a$="";a$=INKEY$:WEND
305 REM ** si shift/s entonces barajar **
310 IF a$="S" THEN GOSUB 700:PRINT"barajando...por favor
    espera":GOSUB 3000
320 GOTO 50
2500 REM **** turno de la banca ****
2520 ON pv+1 GOSUB 2540, 2550, 2560, 2570, 2580
2530 RETURN
2540 bw=1:RETURN:REM apostador se pasa
2550 ts=ps:GOSUB 5000:RETURN:REM pedir hasta vencer
    apostador o pasarse
2560 ts=21:GOSUB 5000:RETURN:REM pedir hasta pontoon o
    pasarse
2570 GOSUB 5200:RETURN:REM pedir hasta juego de 5 naipes
    o pasarse
2580 GOSUB 800:IF ef=2 THEN bw=1:RETURN:REM royal
    pontoon de la banca
2585 bw=0:RETURN:REM royal pontoon del apostador
5000 REM **** la banca pide hasta alcanzar objetivo o pasarse
    ****
5010 GOSUB 800:REM evaluar
5020 IF ef=4 THEN bw=0:RETURN:REM se pasa
5025 IF ef=2 OR ef=5 THEN bw=1:RETURN:REM royal
    pontoon/juego de cinco naipes de la banca
5040 IF tt(2,1)> =ts OR (tt(2,2)<=21 AND tt(2,2)> =ts)
    THEN bw=1:RETURN
5045 IF hp(2)>5 THEN bw=0:RETURN:REM repartidos cinco
    naipes
5050 GOSUB 1300:GOTO 5000:REM repartir y volver a
    evaluar
5200 REM **** pedir hasta juego de 5 naipes o pasarse ****
5220 GOSUB 800:REM evaluar
5225 IF ef=4 THEN bw=0:RETURN:REM se pasa
5227 IF ef=2 OR ef=5 THEN bw=1:RETURN:REM royal
    pontoon/juego de cinco naipes de la banca
5228 IF hp(2)> 5 THEN bw=0:RETURN:REM repartidos cinco
    naipes
5230 GOSUB 1300:GOTO 5200:REM repartir naipe y volver a
    evaluar
```

Sinclair Spectrum

```
130 >REM **** TURNO DEL ORDENADOR ****
140 LET PL=2
150 LET EP=14: GO SUB 670: REM BORRAR NAIPES
160 LET CN=D(PL,1,1): LET SU=D(PL,1,2): GO SUB 100:
    REM REIMPRIMIR
170 LET CN=D(PL,2,1): LET SU=D(PL,2,2):GO SUB 1000:
    REM REIMPRIMIR
180 GO SUB 2500: REM LA BANCA PIDE ETC
190 REM **** GANAR O PERDER ****
200 IF BW=1 THEN GO SUB 700: PRINT "LO SIENTO PIERDES
    TU": GO TO 300
210 GO SUB 700: PRINT " GANAS TU#";BT
300 LET A$=INKEY$: IF A$="" THEN GO TO 300
305 REM ** SI SYM/S ENTONCES BARAJAR **
310 IF A$=CHR$ 195 THEN GO SUB 700: PRINT
    "BARAJANDO... ESPERA POR FAVOR": GO SUB
    3000
320 GO TO 50
2500 > REM **** TURNO DE LA BANCA ****
2520 GO SUB (PV*10)+ 2540
2530 RETURN
2540 LET BW=1: RETURN: REM APOSTADOR SE PASA
2550 LET TS=PS: GO SUB 5000: RETURN: REM PEDIR HASTA
    VENCER AL APOSTADOR O PASARSE
2560 LET TS=21: GO SUB 5000: RETURN: REM PEDIR HASTA
    PONTOON O PASARSE
2570 GO SUB 5200: RETURN: REM PEDIR HASTA JUEGO DE
    CINCO NAIPES O PASARSE
2580 GO SUB 800/ IF EF=2 THEN LET BW=1: RETURN: REM
    ROYAL PONTOON
2585 LET BW=0: RETURN: REM ROYAL PONTOON DEL
    APOSTADOR
5000 >REM **** LA BANCA PIDE HASTA ALCANZAR
    OBJETIVO O PASARSE
5010 GO SUB 800: REM EVALUAR
5020 IF EF=4 THEN LET BW=0: RETURN: REM SE PASA
5025 IF EF=2 OR EF=5 THEN LET BW=1: RETURN
5040 IF T(2,1)>=TS OR (T(2,2)<=21 AND T(2,2)>=TS)
    THEN LET BW=1: RETURN
5050 GO SUB 1300: GO TO 5000: REM REPARTIR Y VOLVER A
    EVALUAR
5200 REM **** PEDIR HASTA JUEGO DE 5 NAIPES O PASARSE
    ****
5220 GO SUB 800: REM EVALUAR
5225 IF EF=4 THEN LET BW=0: RETURN: REM SE PASA
5227 IF EF=2 OR EF=5 THEN LET BW=1: RETURN
5228 IF P(2)> 5 THEN LET BW=0: RETURN
5230 GO SUB 1300: GO TO 5200
```

ninguna acción. La banca ha ganado y, obviamente, no hay necesidad alguna de naipes extras.

Si el jugador posee un marcador de menos de 21, la banca debe pedir repetidamente hasta igualar o vencer el marcador o hasta pasarse.

Si el jugador tiene *pontoon*, la banca debe pedir hasta obtener 21 o pasarse.

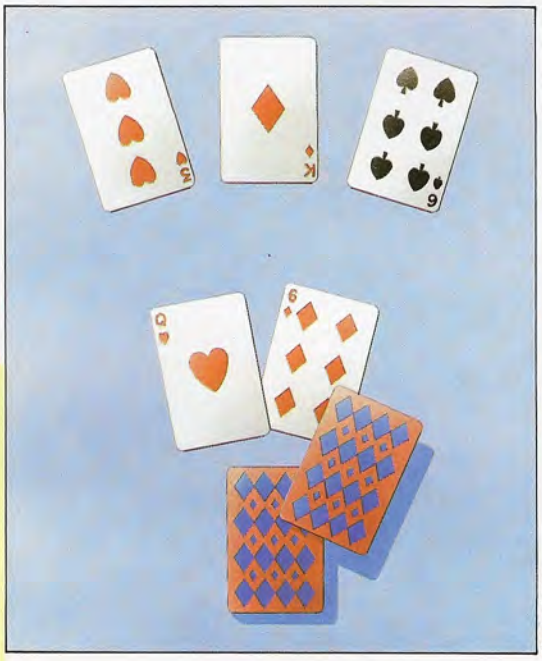
Si el jugador posee un juego de cinco naipes, debido a que el juego de cinco naipes vence al *pontoon* común, la banca debe pedir en un intento por obtener los cinco naipes, pero, por supuesto, puede pasarse en el intento.

Si el jugador tiene *royal pontoon*, la banca sólo puede ganar obteniendo un *royal pontoon* con las dos cartas ya repartidas.

Todas estas acciones, por supuesto, son innecesarias si la banca tiene *royal pontoon*, estado que no se puede superar.

La subrutina de la línea 2500 selecciona el curso de acción, utilizando una instrucción ON...GOSUB junto con la variable PV. En el caso de pedir para superar el marcador del jugador, se establece un marcador meta TS y se llama a otra subrutina, en la línea 5000. Si la banca ha de pedir para obtener un juego de cinco naipes, se necesita una rutina ligeramente diferente y ésta se halla en la línea 5200. En todos los casos el turno de la banca terminará cuando ésta gane o pierda, lo que se indica mediante el establecimiento en 1 o 0, según el caso, de una bandera BW.

El valor de esta bandera determina el mensaje de final del juego a imprimir.



Kevin Jones

Estado de cuenta de la banca
En esta versión del veintiuno o *pontoon*, la banca disfruta de la ventaja de saber cuál es la mano que debe superar. En este juego de ejemplo, el apostador se ha plantado en 19. La banca sabe que se debe repartir a sí misma al menos un naipe más en un intento por ganar la partida

Commodore 64

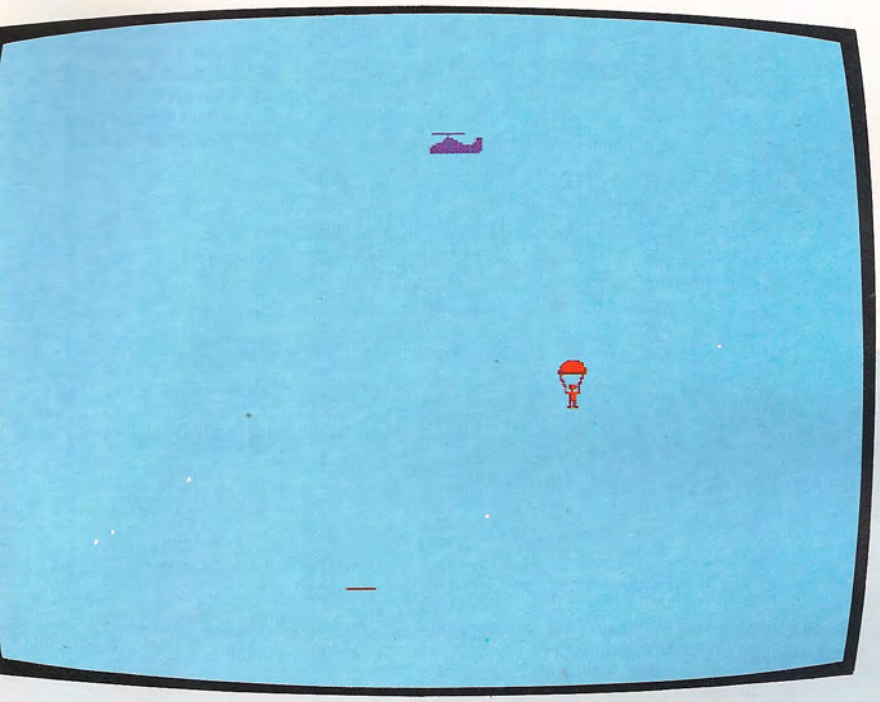
```
130 REM **** TURNO DEL ORDENADOR ****
140 PL=2
150 EP=20:GOSUB 670:REM BORRAR NAIPES
160 CN=HD(PL,1,1):SU=HD(PL,1,2):GOSUB 1000:REM
    REIMPRIMIR
170 CN=HD(PL,2,1):SU=HD(PL,2,2):GOSUB 1000:REM
    REIMPRIMIR
180 GOSUB 2500:REM LA BANCA PIDE ETC
190 REM **** GANAR O PERDER ****
200 IF BW=1 THEN GOSUB 700:PRINT CHR$(156);"LO
    SIENTO TU PIERDES":GOTO 300
210 GOSUB 700:PRINT CHR$(156);"GANAS TU"
300 GET A$:IF A$="" THEN 300
305 REM ** SI SHIFT/S ENTONCES BARAJAR **
310 IF A$=CHR$(211) THEN GOSUB 700:PRINT"BARAJANDO...
    ESPERA POR FAVOR": GOSUB 3000
320 GOTO 50
2500 REM **** TURNO DE LA BANCA ****
2520 ON PV+1 GOSUB 2540, 2550, 2560, 2570, 2580
2530 RETURN
2540 BW=1:RETURN:REM EL APOSTADOR SE PASA
2550 TS=PS:GOSUB 5000:RETURN:REM PEDIR HASTA
    SUPERAR AL APOSTADOR O PASARSE
2560 TS=21:GOSUB 5000:RETURN:REM PEDIR HASTA
    PONTOON O PASARSE
2570 GOSUB 5200:RETURN:REM PEDIR HASTA JUEGO DE 5
    NAIPES O PASARSE
2580 GOSUB 800:IF EF=2 THEN BW=1:RETURN:REM ROYAL
    PONTOON
2585 BW=0:RETURN:REM ROYAL PONTOON DEL
    APOSTADOR
5000 REM **** LA BANCA PIDE HASTA ALCANZAR OBJETIVO
    O PASARSE ****
5010 GOSUB 800:REM EVALUAR
5020 IF EF=4 THEN BW=0: RETURN:REM SE PASA
5025 IF EF=2 OR EF=5 THEN BW=1:RETURN
5040 IF TT(2,1)> =TS OR(TT(2,2)<=21 AND TT(2,2)>=TS)
    THEN BW=1:RETURN
5045 IF HP(2)> 5 THEN BW=0:RETURN:REM REPARTIDOS
    CINCO NAIPES
5050 GOSUB 1300:GOTO 5000:REM REPARTIR Y VOLVER A
    EVALUAR
5200 REM **** PEDIR HASTA 5 NAIPES O PASARSE ****
5220 GOSUB 800:REM EVALUAR
5225 IF EF=4 THEN BW=0:RETURN:REM SE PASA
5227 IF EF=2 OR EF=5 THEN BW=1:RETURN
5228 IF HP(2)> 5 THEN BW=0:RETURN
5230 GOSUB 1300:GOTO 5200:REM REPARTIR
    NAIPE
```

BBC Micro

```
130 REM
140 PL=2
150 EP=20:GOSUB 670
160 CN=HD(PL,1,1):SU=HD(PL,1,2):GOSUB 1000
170 CN=HD(PL,2,1):SU=HD(PL,2,2):GOSUB 1000
180 GOSUB 2500
190 REM
200 IF BW=1 THEN GOSUB 700:PRINT"LO SIENTO, PIERDES
    TU!":GOTO 300
210 GOSUB 700:PRINT"GANAS TU "
300 A$=GET$
305 REM
310 IF A$="S" THEN GOSUB 700:PRINT"BARAJANDO...
    ESPERA POR FAVOR":GOSUB 3000
320 GOTO 50
2500 REM
2520 ON PV+1 GOSUB 2540, 2550, 2560, 2570, 2580
2530 RETURN
2540 BW=1:RETURN
2550 TS=PS:GOSUB 5000:RETURN
2560 TS=21:GOSUB 5000:RETURN
2570 GOSUB 5200:RETURN
2580 GOSUB 800:IF EF=2 THEN BW=1
2585 BW=0:RETURN
5000 FOR DL=1 TO 100:NEXT
5010 GOSUB 800
5020 IF EF=4 THEN BW=0:RETURN
5025 IF EF=2 THEN BW=1:RETURN
5040 IF TT(2,1)> =TS OR (TT(2,2)<=21 AND TT(2,2)>=TS)
    THEN BW=1:RETURN
5050 GOSUB 1300:GOTO 5000
5200 REM
5210 GOSUB 1300
5220 GOSUB 800
5225 IF EF=4 THEN BW=0
5230 IF EF<>5 THEN 5210
5240 BW=1:RETURN
```

Paracaídas

Se han escrito versiones de este juego de acción para numerosos modelos de microordenadores. En esta ocasión presentamos el listado destinado al M05 de Thomson



Saltando de un helicóptero en vuelo, intente alcanzar el blanco situado en el suelo. Una primera pulsación sobre una tecla le permite descender verticalmente en caída libre. Una segunda pulsación produce la abertura del paracaídas. El descenso continúa más lentamente y en un ángulo de 45°, puesto que el viento le empuja. Cuanto mayor sea el tiempo que espere para abrir el paracaídas, tanto menor será la desviación. Pero no aguarde excesivamente, ya que, por debajo de 100 m, el paracaídas no se abrirá...

```
10 REM **************
20 REM * PARACAIDAS *
30 REM **************
40 GOSUB 470
50 HH=HH-4
60 IF HH=0 THEN PUT (0,1)-(27,9),R
70 IF HH=0 THEN HH=288
80 PUT (HH,1)-(HH+27,9),H
90 D$=INKEY$
100 IF D$="" THEN 140
110 IF PV>100 THEN 140
120 IF SP=1 THEN OP=1 ELSE SP=1
130 IF OP=0 THEN PV=10:PH=HH
140 IF SP=0 THEN 230
150 IF OP=0 THEN PV=PV+8
160 IF OP=1 THEN PV=PV+1:PH=PH-1
170 IF PV>167 OR PH<1 THEN 260
180 IF OP=1 THEN 210
190 PUT (PH,PV)-(PH+14,PV+23),PF
200 GOTO 50
210 PUT (PH,PV)-(PH+14,PV+23),PO
220 GOTO 50
230 FOR I=1 TO 50
240 NEXT I
250 GOTO 50
260 IF ABS(PH-A)>4 THEN 320
270 FOR I=1 TO 1000
280 NEXT I
290 S=S+10
300 GOSUB 670
310 GOTO 50
320 CLS
330 SCREEN 1,2,2
340 LOCATE 10,10
350 ATTRB 1,1
360 PRINT "SCORE :";S;
370 LOCATE 10,16
380 PRINT "OTRA ?";
390 ATTRB 0,0
400 IF INKEY$<>"" THEN 400
410 D$=INKEY$
420 IF D$="" THEN 410
430 IF D$<>"N" THEN RUN
440 CLS
450 SCREEN 4,6,6
460 END
470 LOCATE 0,0,0
480 SCREEN 4,6,6
490 DEFINT A-Z
500 DIM H(27,8)
510 DIM PF(14,23)
520 DIM PO(14,23)
530 DIM R(27,8)
540 CLS
550 DRAW "C4BM51,50R14L6G4L2G1L1G1D1F1R2
    1E1U5L2D1G2L5H1L1H1L1H1L1"
560 PAINT (58,56),4
570 GET (50,50)-(77,58),H
580 GET (100,50)-(127,58),R
590 CLS
600 DRAW "C1BM54,50G1L1G1D1G1D1R12U1H1U1H1L1H1L4"
610 PAINT (56,54),1
620 DRAW "C1BM51,56D1F1D1F1D3R2D6L1R1U3R
    2D3R1L1U4L1U4L1U1R2D1L1D4R1U2R2U3E1U1E1U1"
630 GET (50,46)-(64,69),PO
640 CLS
650 DRAW "C1BM54,61R1D6L2D2U2R3D2L1D1R2U
    1L1U2R3D2U2L3U3D3R1U6R1"
660 GET (50,46)-(64,69),PF
670 CLS
680 HH=288
690 HV=1
700 A=INT(RND*191)+10
710 DRAW "COBM"+STR$(A)+",191R12"
720 SP=0
730 OP=0
740 PV=0
750 PH=0
760 RETURN
```

© SYBEX

El Amstrad PCW 8256, conocido familiarmente como "Joyce", constituye un completo sistema de tratamiento de textos

Una cita con Joyce

Amstrad Consumer Electronics tiene la tradición de empaquetar tecnología ya existente en sistemas "todo en uno". El Amstrad PCW 8256, si bien es una máquina muy distinta a sus predecesoras, se ha construido en gran parte siguiendo la misma línea. Diseñado como paquete integrado para tratamiento de textos, el sistema incluye en su precio una unidad de disco incorporada, una impresora matricial y el software para tratamiento de textos.

Aun teniendo en cuenta la filosofía de fabricación y marketing de la empresa, el lanzamiento del PCW 8256, apodado "Joyce", nombre de la secretaria de Alan Sugar, causó sorpresa en la industria. Mientras que los ordenadores Amstrad anteriores estaban diseñados para atraer al mercado más amplio posible, la última máquina es la primera de la empresa dirigida a un sector específico, si bien más grande. De modo, entonces, que aunque el Amstrad CPW 8256 se puede utilizar como ordenador estándar, la empresa lo está dirigiendo básicamente a las pequeñas empresas.

Al igual que anteriores ordenadores Amstrad, la pantalla está alojada en una carcasa con la fuente de alimentación al costado de una unidad de disco incorporada y un conector que le proporciona potencia a la impresora independiente. Un cable enrollado que sale del teclado se enchufa en un conector lateral.

Las 82 teclas incluyen el grupo QWERTY estándar más algunas adicionales, tales como Alt y Extra, para proporcionar juegos de caracteres alternativos. Éstos incluyen caracteres griegos y otros, tales como ü, que permiten que el ordenador procese toda una variedad de lenguas. Inmediatamente a la derecha de éstas hay cuatro teclas de función programables, que poseen numerosos usos y proporcionan hasta ocho funciones diferentes. Hay un grupo de teclas de cursor en la esquina inferior derecha del teclado y arriba hay varias teclas dedicadas a diversas funciones de tratamiento de textos. Si bien representa una mejora respecto al teclado estándar de Amstrad, las teclas resuenan un poco cuando se entra texto a cualquier velocidad.

Pantalla verde

La pantalla es un modelo de fósforo verde y, según Amstrad, será el único tipo disponible. Con una resolución de 92 por 32 caracteres, la pantalla es más grande que la que se proporciona en la mayoría de los ordenadores. En pantalla, los caracteres son del mismo tipo de letra, pero más grandes que los de otros modelos Amstrad, con lo que el texto resulta más fácil de leer. La unidad de disco se halla en la esquina superior derecha de la carcasa de la pantalla, debajo de la cual hay otra ranura para instalar una segunda unidad de disco, detrás de la placa donde está inscrita la marca.

La impresora matricial de cinco por ocho agujas que se suministra junto con el PCW 8256 es capaz de dar cabida a hojas A4 individuales o al formulario continuo estándar de 11 pulgadas. La primera de las dos modalidades en las que puede operar se conoce como "modalidad de borrador" y, tal como sugiere su nombre, es ideal para producir copias en borrador, memorándums sencillos, etc. Esta modalidad, no obstante, ilustra claramente las limitaciones de la impresora. Una observación atenta revela que los puntos individuales son incapaces de formar una línea continua.

Amstrad no fabrica su propia impresora, sino que "emula modelos" de otras empresas, política que ha suscitado algunas quejas de parte de los usuarios. Lo que es más importante, sin embargo, es que una impresora de buena calidad es esencial para un sistema de tratamiento de textos, de modo que sería lícito cuestionar la decisión de Amstrad de utilizar este modelo en particular con el PCW 8256. Aparentemente la empresa ha reconocido esta limitación y ha anunciado que proveerá una interface RS232 y Centronics que permitirá la instalación en la máquina básica de impresoras de más alta calidad.

Usted, sin embargo, se enfrentará con varios problemas si decide instalar otra impresora. Debido a que *LocoScript*, el software para tratamiento de textos empaquetado con la máquina, se basa en gran medida en el hardware existente, posee dificultades para comunicarse con otras impresoras. La solución de Amstrad es modificar el sistema opera-

Amstrad amplía su horizonte
El PCW 8256, apodado "Joyce", es el primer micro de Amstrad pensado principalmente para el mercado de gestión. Si bien está siendo comercializado como procesador de textos exclusivo (con pantalla, impresora, unidad de disco, y software para tratamiento de textos empaquetado), el PCW 8256 también es un ordenador de ocho bits muy potente

tivo CP/M 3.0 para poder leer los archivos en formato ASCII y enviarlos a la impresora. Esto significa, por supuesto, que usted tendrá que salir del *LocoScript* para poder imprimir el archivo. Además, muchas de las facilidades para formateo de la impresión más atractivas del *LocoScript* no se pueden incluir en tal sistema.

Sin embargo, la modalidad de impresión de "alta calidad" supera estas limitaciones de hardware. La impresora para dos veces por cada línea, rellenando los agujeros dejados por los puntos, de modo muy similar al sistema usado para imprimir en negrita. Aunque esto mejora enormemente la calidad de la impresión, reduce de manera drástica la velocidad de la impresora, que pasa de 90 caracteres por segundo (cps) en modalidad de borrador, a 20 cps. Existen algunas dudas respecto a la fiabilidad de la impresora. La fila central de agujas del sistema que revisamos falló tras apenas dos días de utilización.

Al igual que los anteriores ordenadores de Amstrad, el PCW 8256 se basa en el conocido procesador Z80 equipado con 256 Kbytes de RAM. La máquina incorpora las ahora comunes técnicas de *conmutación de bancos*, ya que de lo contrario sería incapaz de manipular tanta memoria. Se han organizado alrededor de 110 Kbytes como un disco de silicio, que el procesador trata como un disco flexible y, por tanto, accede a la información consiguientemente. Dado que la información está retenida en chips de RAM, se puede acceder a la misma casi instantáneamente, lo que, por supuesto, es más rápido que desde un disco.

El software empaquetado

El software empaquetado con el PCW 8256 incluye CP/M 3.0, una versión mejorada del OS de anteriores máquinas Amstrad. También se incluye el sistema operativo de disco AMSDOS de Amstrad, junto con el paquete para gráficos GSX de Digital Research. Éste permite que el CP/M visualice gráficos (facilidad que originalmente no se había incorporado en su diseño) y ofrece la posibilidad de que varios paquetes de gestión disponibles bajo CP/M hagan uso de la facilidad de gráficos.

Asimismo, Amstrad ha empaquetado con la máquina el Mallard BASIC, el Dr LOGO y el *LocoScript*. Amstrad afirma que el *LocoScript* se ha diseñado para que sea potente y a la vez tan fácil de usar como una máquina de escribir eléctrica.

Tras el encendido, la pantalla *LocoScript* se llena de marcas de TAB, pensadas para ayudar a formatear el texto. No obstante, a muchos usuarios les parecerá que esto produce una visualización más bien confusa, que se interpone en medio del texto. Pero los programadores han permitido que el usuario elimine todas estas marcas, con lo que se obtiene una visualización más clara.

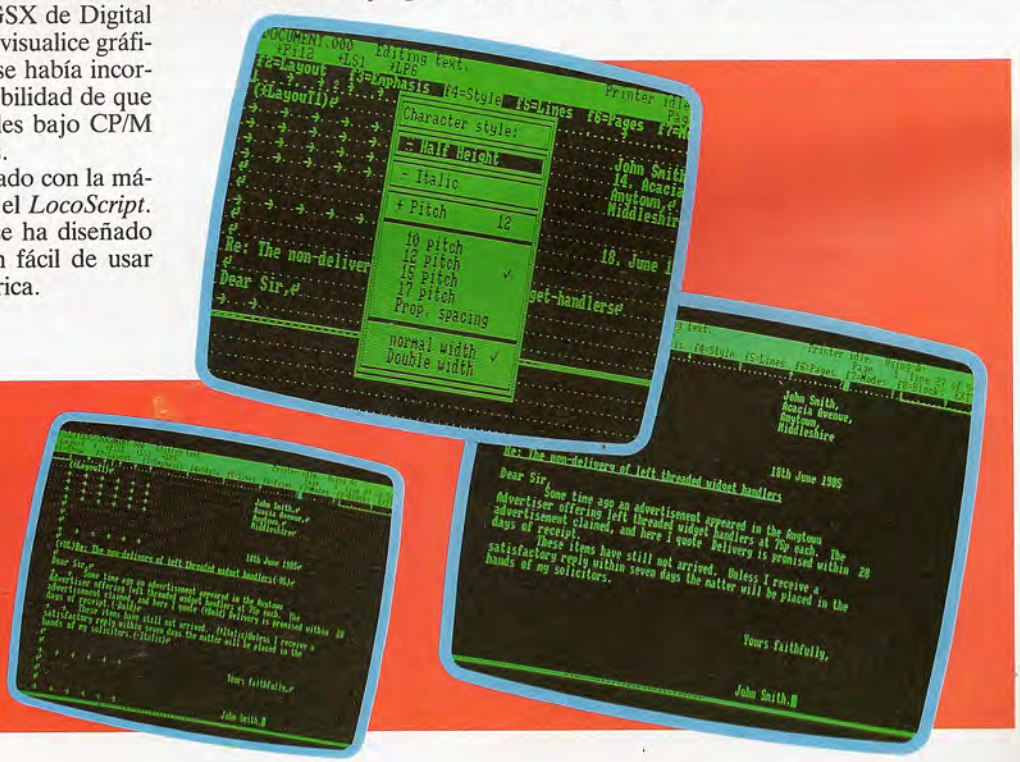
El sistema, por cierto, parece ser tan potente y flexible como muchos sistemas para tratamiento de textos diseñados para máquinas de gestión de gran calidad. *LocoScript* posee facilidades que permiten desplazar el texto dentro de un documento, trasladar el cursor por caracteres, palabras o párrafos, y localizar palabras y frases, tareas todas estas que se llevan a cabo mediante las teclas especializadas en tratamiento de textos incorporadas en el teclado. Para copiar un área de texto de una parte de un documento en otra parte, por ejemplo, es necesario desplazar el cursor hasta el comienzo del área en cuestión y pulsar la tecla COPY. El proceso se repite luego al final de esa sección. La transferencia se efectúa desplazando el cursor hasta la posición correspondiente y pulsando la tecla Paste.

A las facilidades tales como alineación, enfatización de caracteres y formatos de impresión se accede a través de las teclas de función. Aparecerá un menú con una serie de opciones que se pueden seleccionar utilizando las teclas del cursor y pulsando luego la tecla Enter. Este sistema de menús y submenús se ha diseñado para aumentar las facilidades para tratamiento de textos.

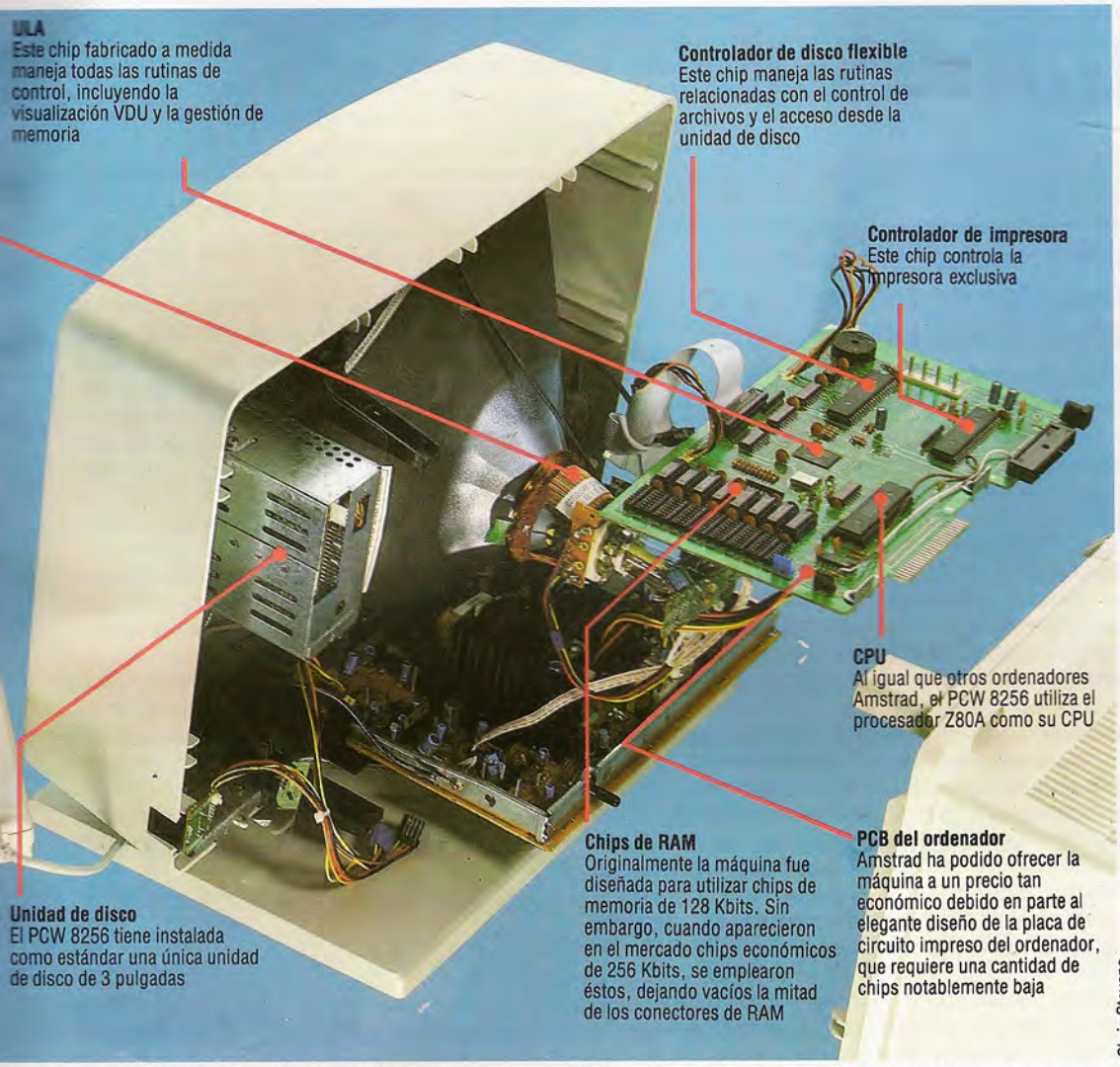
Como hemos visto en nuestra serie dedicada a los paquetes para tratamiento de textos, la mayoría de éstos hacen un uso intensivo de caracteres Control y menús de pantalla separados para llevar a cabo la amplia variedad de funciones disponibles. Es comprensible que los programadores del PCW 8256 hayan optado por utilizar este sistema de teclas de función programables, menús y ventanas

Pantalla
Esta pantalla incluye la placa de circuito impreso de decodificación de señales

Teclado
Al igual que muchos otros procesadores de textos especializados, el PCW 8256 contiene algunas teclas adicionales inexistentes en ordenadores estándares, que se utilizan en tratamiento de textos



Facilidades en pantalla
El *LocoScript*, el software para tratamiento de textos diseñado a medida para el PCW 8256, soporta una amplia gama de facilidades. Si bien muchas de ellas están disponibles a través de una serie de ventanas situadas en la parte superior de la pantalla, muchas otras están incorporadas en el texto. Aunque es útil examinar estas instrucciones en pantalla, a muchos usuarios les parecerá que las mismas confunden el texto que se está escribiendo. Sin embargo, se las puede eliminar de la visualización en pantalla



Chris Stevens

AMSTRAD PCW 8256

DIMENSIONES
375 × 326 × 309 mm

MEMORIA
RAM de 256 K, de los cuales 128 K están configurados como un disco de RAM

CPU
Z80

VISUALIZACION DE VIDEO
92 × 32 caracteres; 24 líneas × 90 caracteres, zona de texto en *LocoScript*

INTERFACES
Conector para segunda unidad de disco, interface para impresora, conector para teclado

SOFTWARE SUMINISTRADO
Versión 3.0 de CP/M, programa de tratamiento de textos *LocoScript*, Mallard BASIC, Dr LOGO, utilidad para gráficos GSX

DOCUMENTACION
El manual es exhaustivo, tal como nos tiene acostumbrados Amstrad con sus manuales

VENTAJAS
El PCW 8256 constituye una oferta muy interesante al incluir en su precio un ordenador de 256 K, pantalla, unidad de disco e impresora

DESVENTAJAS
Parte del hardware muestra signos de la reducción de costos. En especial, la impresora parece ser lenta y de dudosa fiabilidad. Además, parece ser que incluso con una interface especial una impresora diferente tampoco podría sacar partido de muchas de las facilidades de *LocoScript*

como un medio de simplificar el sistema. Aunque usted ya no tendrá que aprender los numerosos caracteres Control que utilizan la mayoría de los procesadores de textos, cada una de las teclas de función de la máquina Amstrad conduce a su propio menú, y algunas de las etiquetas tanto de las teclas como de los menús no dan una idea cabal de su finalidad. Esto significa que usted deberá aprender algunos pasos y pulsaciones de teclas para obtener el efecto deseado. (Observe que si bien el PCW 8256 no posee ninguna tecla Control, se sustituye la tecla Alt para gran número de programas basados en CP/M).

Entrada de comandos

LocoScript proporciona un método más rápido de entrar comandos. A los lados de la barra Space hay teclas marcadas ± y =. Para alinear una línea por la derecha, pulsando la tecla ± seguida por RJ se ejecutará la instrucción, mientras que = la dará por terminada. Esto, así y todo, lleva más tiempo que utilizar un único carácter de control; habría sido mucho más simple emplear un mecanismo *toggle*.

Se suma a la confusión parte de la edición en pantalla. Una vez que se ha modificado un párrafo, algunas de las líneas podrán parecer desiguales porque el software no habrá ajustado el párrafo, obligando al propio usuario a hacerlo pulsando la tecla Relay. Este proceso de ajuste o alineamiento manual es común en procesadores de textos más antiguos, en particular en el *WordStar*, pero cabría esperar que un paquete desarrollado en 1985 hubiera incorporado el ajuste automático.

Otra característica curiosa del *LocoScript* es que parecería que el OS del ordenador fuera incapaz de leer muchos de los archivos. Esto es consecuencia de la capacidad del CP/M 3.0 para soportar numerosos usuarios diferentes. Cuando se carga el OS, pasa automáticamente por defecto al usuario 0. Al buscar en los archivos listados bajo diferentes números de usuario, el operador se encontrará con los documentos *LocoScript* "desaparecidos".

A pesar de estos problemas, el PCW 8256 representa una buena oferta y contiene numerosas facilidades excelentes. La gama y la variedad de las instrucciones para la impresora y formateo de páginas son similares a las que hay disponibles en paquetes incluidos en micros que cuestan varias veces el precio de la máquina de Amstrad. Parece ser, sin embargo, que esta vez la política de Amstrad de ofrecer la máxima cantidad de facilidades por el mínimo precio se ha llevado a un límite extremo. Los usuarios quizá opinen que un diseño un poco mejor (en especial en el caso de la impresora), aun al costo de algunas de las utilidades, hubiera hecho del PCW 8256 una propuesta muchísimo más interesante.

Guardabosque

Los directorios jerárquicos representan un método muy práctico de manejar grandes cantidades de archivos

Todos los comandos residentes que analizamos en el capítulo anterior están disponibles en todas las versiones de MS-DOS. Sin embargo, desde la versión 2.0 en adelante se puede contar con una nueva gama de facilidades, la mayoría de las cuales han sido sacadas del Unix. Tuvo vital importancia la introducción de una estructura de *directorio jerárquico*. La misma resulta particularmente útil con los modernos discos flexibles de elevada densidad, y es absolutamente esencial para los sistemas de disco rígido.

Cuando se carga por primera vez un sistema DOS 2, el usuario "ve" una zona de la unidad de disco por defecto que contiene varios archivos. En la mayoría de los casos se trata de programas (.EXE o .COM) o bien archivos de datos. Es normal tener tanto como 720 Kbytes de almacenamiento en un microflexible moderno de 3½ pulgadas, y varias decenas de megabytes en un disco rígido. El gran número de archivos significa que un simple listado del directorio podría suponer la visualización de una enorme cantidad de información, la mayor parte de la cual es irrelevante para el campo de interés actual del usuario. Está claro que se requiere un método para descomponer la zona de disco en *subdirectorios* manejables.

El directorio por defecto se llama *raíz* y se puede crear un subdirectorio dentro de la raíz mediante la instrucción md (de mkdir, o MaKe DIRectory: hacer directorio). De modo que:

```
md trabajo
```

crearía un "archivo" (denominado trabajo) en el directorio raíz que "contuviera" otros archivos de datos directos.

Quizá deseáramos crear algunos documentos o programas fuente dentro de este nuevo directorio, de modo que cambiamos el directorio (CHange DIRectory) impartiendo la instrucción chdir, o su alternativa:

```
cd work
```

Si ahora digitamos dir (para listar el contenido de nuestro "directorio actual"), podemos esperar que esté vacío. Sin embargo, el MS-DOS ya ha creado dos subdirectorios dentro de trabajo. Éstos se llaman . y .. (un tanto enigmáticamente) y se refieren, respectivamente, al directorio actual (trabajo) y su "padre". Por tanto, si entráramos el comando:

```
dir..
```

se ofrecería un listado de todos los archivos del directorio raíz. Ahora éste incluiría la entrada:

```
TRABAJO <DIR> 17-11-85  11.21 a
```

Si ahora damos los comandos:

```
mkdir hoy
chdir hoy
```

nos encontraríamos "en" un directorio que tendría como directorio padre a trabajo y a la raíz como su "abuelo".

Al especificar el nombre de cualquier archivo del sistema, debemos dar un *nombre de paso* completo si el archivo no está en el directorio actual. De modo que, por ejemplo, un programa del directorio raíz se podría especificar ya sea como ..\..\ prog o simplemente \ prog (la barra invertida se utiliza como un separador en nombres de paso o, si es el primer símbolo de un nombre de paso, como una abreviatura para la raíz). El movimiento un nivel "hacia arriba" (hasta el subdirectorio trabajo) se puede conseguir tanto con chdir.. o, en este caso, con cd/trabajo.

Si ahora entráramos md fuente, conseguiríamos crear un directorio "hermano" de hoy, y ambos se considerarían como "hijos" de nuestro directorio actual, trabajo.

Como puede verse, esta estructura empieza a parecerse a un árbol genealógico y de hecho se denomina *árbol* de directorios. La adición de algunas ramas más podría producir la estructura en la siguiente página.

Por consiguiente, un nombre de paso completo para un archivo del directorio texto denominado MEMO.DOC podría ser:

```
trabajo\fuente\texto\memo.doc
```

Éste localizaría el archivo en cualquier parte del árbol, dado que se especifica un nombre de paso completo comenzando desde la raíz (\); no obstante, si actualmente estuviéramos en el subdirectorio hoy podríamos utilizar:

```
..\fuente\texto\memo. doc
```

En cualquier momento, un comando dir dará un listado del directorio actual, pero se puede especificar un camino alternativo, como en:

```
dir b:\pascal\fuente\prognuevos
```

Observe que en este caso el nombre final puede ser el de un directorio, no de un archivo, y que se podría dar una unidad de disco diferente como el primer elemento del camino (en este caso, la unidad b). Mientras no se quite un disco de una unidad, el disco "recordará" todo subdirectorio actualmente seleccionado en esa unidad.

Esto significa que:

```
copy\*.*b:
```

copiará todos los archivos del directorio raíz del disco actual en el directorio actual, cualquiera sea, de la unidad b. A cada disco o *volumen* se le puede dar un nombre, ya sea cuando se formatea originalmente el disco o mediante la utilidad transitoria LABEL.EXE.

LABEL.EXE y FORMAT.EXE son dos de los programas que se suministran como utilidades *transitorias*. Éstos son programas separados (por lo general llevan la ampliación .EXE, pero en algunas ocasiones .COM) que se pueden ejecutar entrando sus nombres primarios. Con el fin de formatear un

disco, por ejemplo, se puede ejecutar el programa FORMAT.EXE entrando:

```
format b:
```

La instrucción FORMAT toma dos "interruptores" opcionales delimitados por una barra. Las opciones son:

\s Para copiar el sistema MS-DOS tras el formateo.
\p Para preguntar el nombre de un volumen o "etiqueta" a dar al disco. Sólo se permite un archivo (raíz) que tenga este atributo.

Ello da por sentado que el directorio operativo actual contiene FORMAT.EXE; pero supongamos que no lo contenga. El DOS 2 proporciona un comando residente sumamente útil para posibilitar la definición de los nombres de camino por defecto:

```
path=A:\sistema;A:\sistema\utils
```

La entrada de un comando no reconocido hará que el sistema siga el camino (o serie de caminos alternativos separados mediante punto y coma) con el fin de encontrar el comando. Usted puede averiguar cuál es el camino de búsqueda actual entrando:

```
path
```

y si no se ha definido ninguno se imprimirá el mensaje No path.

Muchas versiones de MS-DOS a menudo se proporcionan con comandos externos adecuados para el sistema OEM, pero hay algunos tan útiles que se encuentran en todas las versiones. PRINT.EXE permite la impresión como tarea de fondo de un archivo de texto mientras el usuario continúa trabajando en el ordenador. EDLIN.EXE es un editor de líneas que permite la creación de archivos de texto sin necesidad de un procesador de textos adicional. DISKCOPY.COM (o DCOPY) copia el contenido completo de un disco, y DISKCOMP (o COMP) se puede utilizar para verificar una transferencia.

Todas las utilidades MS-DOS que suponen el uso conceptual de dos unidades de disco son suficientemente "inteligentes" como para comprender la necesidad del intercambio de discos cuando se ejecuta un entorno de disco único. Por ejemplo, DISKCOPY carga datos de un disco fuente en la memoria y luego imparte un aviso para cambiar discos. Una vez completa la operación de copiado, se da otro aviso para insertar el disco fuente; y así sucesivamente, hasta haber transferido el disco entero. Se detectan los sectores malos y el disco de destino se puede formatear a la vez que es copiado.

La utilidad más útil para comprobar discos es CHKDSK.EXE, que proporciona información completa sobre el estado de un disco, incluyendo la señalización de la existencia de "archivos ocultos". Empleado sin argumentos, el comando:

```
chkdsk
```

comprueba la unidad "conectada" actualmente y produce un informe con la siguiente forma:

```
720666 bytes espacio total de disco
 47528 bytes en dos archivos ocultos
  2048 bytes en cuatro directorios
526386 bytes en 39 archivos del usuario
144704 bytes disponibles en el disco

262144 bytes memoria total
198726 bytes libres
```

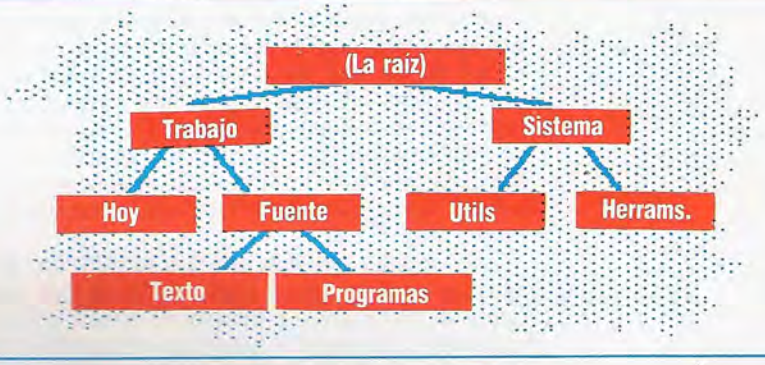
Es de gran utilidad que CHKDSK informe sobre el estado de la memoria del sistema (los dos últimos ítems) y que también compruebe el uso general del disco y la File Allocation Table, FAT (tabla de ubicación de archivos) retenida para cada dispositivo. El MS-DOS puede percibir cuándo se están cambiando los medios y reconstruir una FAT para un nuevo disco leyendo la información del sistema relativa al mismo.

Una palabra de advertencia: jamás use la utilidad CHKDSK del DOS suministrada con una versión de MS-DOS para comprobar discos DOS de otra versión importante. Los sectores del disco y la información sobre el directorio pueden estar dispuestos en un formato muy diferente, y esto puede ser especialmente peligroso si el DOS 1.x y el DOS 2.x están "vivos" en el mismo sitio. La mezcolanza resultante de enlaces entretejidos podría significar horas de trabajo infructuoso tratando de recuperar, pongamos por caso, un documento en el cual se hubieran intercalado "páginas" de código máquina de otro archivo. Si por descuido usted utiliza CHKDSK del DOS 1 en un sistema de disco rígido DOS 2 con 10 o 20 Mbytes en juego, las consecuencias pueden ser desastrosas.

Si usted tuviera motivos para mejorar un antiguo IBM PC o Sanyo pasando de DOS 1 a DOS 2, la moraleja es obvia: hacer copias de seguridad de todos los datos existentes en discos formateados no sistematizados, y después destruir todos los sistemas actuales (mediante reformateo si fuera necesario) antes de instalar DOS 2. Los OEM que suministran las mejores implementaciones de MS-DOS se aseguran de que las utilidades transitorias incorporen código para comprobar que el sistema en el que se estén ejecutando sean de una versión compatible, pero así y todo se debe ser cuidadoso.

División del trabajo

El sistema de directorio de archivos jerárquico implementado en el MS-DOS permite que los usuarios de sistemas de disco de gran capacidad dividan los archivos en subcategorías, dado que los directorios no sólo pueden contener nombres de archivos sino nombres de otros directorios. En este ejemplo el archivo "texto" es un miembro del directorio "fuente", que por su parte es miembro del directorio "trabajo". El directorio raíz contiene las principales categorías de grupos de archivos dentro de la estructura arborescente



Caroline Clayton

Conciencia de clase

El compilador del C reconoce a las variables como pertenecientes a una de cuatro diferentes clases

Además de ser de un tipo determinado, todas las variables de C poseen una *clase de almacenamiento*, que determina la forma en que el compilador las reconoce y asigna memoria para ellas. Hay cuatro clases: automática, externa, registro y estática. Las variables pueden llevar especificada su clase colocando las palabras clave auto, extern, register o static delante del tipo en la declaración.

Por ejemplo:

```
extern double x,y;
```

Casi todas las variables son *automáticas* por defecto. Toda variable declarada dentro del cuerpo de una función será automática por defecto y local de esa función. Por consiguiente, cada vez que se entre la función se asignará nuevo espacio de almacenamiento para esa variable y el mismo se perderá tras salir de la función. No existe modo alguno por el cual tal variable pueda conservar un valor entre dos llamadas a la función. Lo mismo se aplica a toda variable definida dentro de un bloque de código, indicado por estar encerrado entre llaves {}. Recuerde que main() es una función, de modo que incluso las variables declaradas aquí normalmente serán automáticas.

Las variables *externas*, por el contrario, son globales, es decir, se puede aludir a ellas en cualquier punto del programa, e incluso en ciertos casos a través de funciones que aparezcan en un archivo de código fuente distinto al de la declaración. Una variable declarada fuera del cuerpo de una función se considerará externa por defecto. Asimismo, una variable externa se puede declarar en cualquier otro punto dentro del cuerpo de una función o bloque. Continuará existiendo aún después de que acabe la ejecución de una función y se puede aludir a ella en cualquier punto del programa que sea posterior a la declaración.

Si se declaran variables externas del mismo nombre en dos o más archivos de código fuente, se considerarán como la misma variable cuando los archivos se unan entre sí. Si se declara una variable local con el mismo nombre que una variable global, "enmascarará" a la variable global dentro de su ámbito, de modo que las alusiones a ese nombre de variable serán a la variable local dentro de su ámbito y a la variable local fuera de él.

Las variables *de registro* se comportan exactamente como las variables automáticas, y de hecho en algunos casos se tratarán del mismo modo. Sin embargo, de ser posible, el compilador asignará para ellas espacio en almacenamiento de alta velocidad. Esto evidentemente dependerá mucho del procesador que se esté utilizando para ejecutar el programa, y en particular del tamaño y cantidad de sus registros. Un 68000, por ejemplo, con sus 16 registros de 32 bits para fines generales, posee alcance suficiente para utilizar los registros para una cantidad limitada de variables de registro, mientras que un Z80 probablemente no tendrá espacio. Las variables de registro sólo deben utilizarse de cuando en cuando y declararse lo más tarde posible para restringir su ámbito y dejar el espacio libre lo más pronto posible. Las variables de control de bucle son candidatas comunes para esta clase.

Las variables *estáticas* por lo general son locales a una función o bloque, pero difieren de las variables automáticas en que el almacenamiento asignado y el valor allí almacenado se conservan durante la ejecución de la función o el bloque. Un posible uso de una variable estática podría ser contar la cantidad de veces que se llama a una función. Asimismo, pueden proporcionar "ocultación de datos" por cuanto los valores almacenados en tales variables no son accesibles externamente, a diferencia de las variables externas. Una variable estática que se declara externamente a un juego de funciones está disponible globalmente para esas funciones, pero no está disponible para las funciones ajenas a ese archivo determinado.

La utilización de matrices

Una matriz se declara del mismo modo que otras variables, con su tamaño (es decir, la cantidad de elementos) entre corchetes a continuación del nombre de variable.

Por ejemplo:

```
int matrizent[100];
```

reserva espacio de almacenamiento para los elementos de matriz matrizent[0], matrizent[1], etc., hasta matrizent[99]. En C los subíndices siempre parten desde cero, y la declaración se refiere a la cantidad total de elementos, de modo que, en este caso, no hay ningún elemento matrizent[100]. Las matrices estáticas o externas se pueden inicializar mediante la adición a la declaración de una lista de valores encerrados entre llaves.

Por ejemplo:

```
static int días_del_mes[12]=
        {31,28,31,30,31,30,31,31,30,31,30,31};
```

Si la lista no está completa, los elementos restantes se establecen en cero. De no efectuarse ninguna inicialización para una matriz estática o externa, todos los elementos se inicializarán a cero. Las matrices automáticas no se pueden inicializar y el espacio se creará con valores de desecho, de modo que no se puede confiar en que los elementos se inicialicen en cero.

Obviamente, no tiene sentido la idea de una matriz de registro.

Si se está inicializando una matriz, el C no exige que se mencione el tamaño; éste se tomará automáticamente como la cantidad de valores proporcionados, de modo que la declaración anterior muy bien podría haberse escrito como:

```
static int días_del_mes[]=
        {31,28,31,30,31,30,31,31,30,31,30,31};
```

Esta facilidad es de especial utilidad para series, o matrices de tipo char, donde la serie inicializadora simplemente se puede encerrar entre comillas. De modo que las dos declaraciones siguientes son equivalentes:

```
static char st[]="hola";
```

y

```
static char st[]={'h','o','1','a'};
```

Observe, sin embargo, que estas series no son dinámicas en el mismo sentido que las series del BASIC: su longitud no puede variar respecto a aquella declarada.

En un ulterior capítulo veremos con mayor detalle la manipulación de series.

El c puede abordar matrices con virtualmente cualquier cantidad de dimensiones. El único punto a recordar respecto a las matrices de dos o más dimensiones es que cada subíndice debe tener su propio par de corchetes, de modo que la declaración para una matriz bidimensional de cuatro por cinco elementos se entraría como:

```
int matrizbid[4][5];
```

Las matrices pueden ser pasadas a las funciones como parámetros, pero en este caso se las pasa por referencia; es decir, se pasa a la función la dirección del primer elemento y cualquier cambio en la matriz realizado dentro de la función permanecerá vigente cuando se vuelva a transferir el control a la rutina de llamada. No es necesario declarar el tamaño de una matriz dentro de la función, porque de todos modos éste se conoce, de modo que las funciones se pueden escribir con un carácter bastante general para que operen sobre matrices de cualquier tamaño.

Los listados que ofrecemos como ejemplo emplean muchos de los conceptos analizados aquí. Estamos suponiendo que se requiere un generador de números aleatorios y las funciones para manipularlo están escritas en un archivo, el cual se puede unir a cualquier programa que necesite números aleatorios.

En un archivo separado hay un breve programa de prueba que requiere un conjunto grande de números aleatorios y cuenta las frecuencias de números en intervalos dados para comprobar una distribución uniforme.

Listado 1, almacenado en archivo 1

```
#define MULT 25173
#define MODULUS 65536
#define INCREMENT 13849
static int semilla;

/*esta declaración es externa por cuanto concierne a
este archivo de modo que la variable semilla está
disponible para todas las funciones de este archivo,
pero no se puede aludir a ella fuera del archivo; de
hecho el nombre "semilla" se puede usar libremente
en otros archivos*/
randomise(s)

int s
{
    semilla=s;
}
random(n)
int n

/*devuelve un número aleatorio entre 0 y n*/

{
    semilla=(MULT* semilla+INCREMENT)%
  MODULUS;
/*el método de congruencia lineal*/
    return((int)((double)semilla/(double)MODULUS*
  (double)n+0.5));

/*observe la conversión de los valores enteros a punto
  flotante para la aritmética y otra vez a int, el +0.5 es
  para redondear al entero más próximo*/

}
double rnd()

/*devuelve un número aleatorio entre 0 y 1*/

{
    semilla=(MULT* semilla *INCREMENT)%
    MODULUS:
    return ((double)semilla/(double)MODULUS);
}
```

Listado 2, almacenado en archivo 2

```
#define SEMILLA 17
#define TAMAÑO 100
#define LIMITE 10000
#define NUM__DE__GRUPOS 10
int grupos []={10,20,30,40,50,60,70,80,90,100};

/*matriz externa se puede inicializar; ésta contiene
límites de grupo para contar frecuencias*/

int frecuencias [10];
/*esta matriz externa se inicializará en ceros*/
main()
int r;
{
    randomise (SEMILLA);
    register int i;

/*utilizando clase de registro para variables
de bucle para proceso más eficiente, y
guardando la declaración lo más tarde
posible*/

    for (i=0;i< LIMITE;++i)
    {
/*tomar 10000 números aleatorios entre 0 y 100*/
        r=random (TAMAÑO);
        register int j;
/*comprobar a qué grupo pertenecen*/
        for (j=0;j< NUM__DE__GRUPOS;
        ++j)
        {
        if (r < grupos [j])
        {
/*contar el número aleatorio en el grupo
correspondiente y salir del bucle */
            ++frecuencias [j];
            romper;
        }
      }
    }
/*imprimir una tabla de frecuencias para comprobar
distribución uniforme*/
    for (j=0;j< NUM__DE__GRUPOS;++j)
    {printf("%d__%d=%d\n",grupos [j]__
10,grupos[j],frecuencias[j])}
}
```

Rendimiento de los comandos

Nos corresponde analizar las instrucciones lógicas junto con las operaciones de rotación y desplazamiento y las instrucciones de bifurcación

Analicemos primeramente las instrucciones lógicas de que está provisto el 68000. La instrucción AND opera con un "Y" lógico el fuente con el destino, dejando el resultado en el destino y afectando en consonancia a los bits N y Z del registro de estado. Existen varios modos de direccionamiento posibles, pero el fuente o el destino por lo menos deben ser un registro de datos. Además se permiten los atributos de datos; así, por ejemplo, si D0=1010 1010 y D1=1111 0000, y ejecutamos la instrucción:

```
AND D1,D0
```

entonces resulta D0=1010 0000.

Éste es un sencillo ejemplo del empleo de una instrucción AND. En este caso, hemos tomado los cuatro bits menos significativos poniendo a cero los bits de máscara que están en D1, y hemos hecho que los MSB (bits más significativos) aparezcan en su estado original mediante una máscara de bits puestos a 1. Una versión más explícita de esta operación incluiría quizá la instrucción ANDI. Ésta acepta el modo de direccionamiento inmediato como operando fuente, y los modos alterables de datos como destino. Si empleamos ANDI en el ejemplo anterior tendríamos:

```
ANDI #$F0,D0
```

Las restantes instrucciones lógicas son OR y ORI para la operación OR lógica, EOR y EORI para la OR exclusiva, y la instrucción NOT. Estas instrucciones, respecto a los modos de direccionamiento, son parecidas a la instrucción AND, siendo idéntica la forma en que es afectado el código de condición. La manipulación de los bits con instrucciones lógicas es de gran importancia para controlar, por ejemplo, los bits que hacen de *flags*, o bien una palabra o las líneas de entrada/salida digitales para periféricos o equipos externos.

Cuando el operando de datos es un bit único, el 68000 tiene un conjunto de instrucciones para manipulación de cuatro bits que pueden ser utilizadas en lugar de las instrucciones lógicas. Estas instrucciones comprueban el estado de un bit específico (numerado de 0 a 31, en un registro de datos, sólo para palabras largas) o un byte de la memoria. Afectan también al bit Z del SR según sea el estado de ese bit. Por tanto, el bit Z se convierte en una memoria invertida de un bit respecto al bit especificado. La instrucción BTST puede usarse del modo siguiente. Si, por ejemplo, D0=XXXX XXXX XXX1 0000 (en binario, siendo X un 0 o un 1), entonces:

```
BTST #4,D0
```

comprobará el bit cuatro y pondrá Z a cero (el bit del ejemplo no es cero). Ahora bien:

```
BTST #3,D0
```

pondría Z a uno. Todos los códigos de condición permanecen inalterados.

Las restantes instrucciones afectan al bit que se comprueba. Y son:

```
BSET    comprueba y pone el bit a 1
BCLR    comprueba y pone el bit a 0
BCHG    comprueba y cambia el valor del bit
```

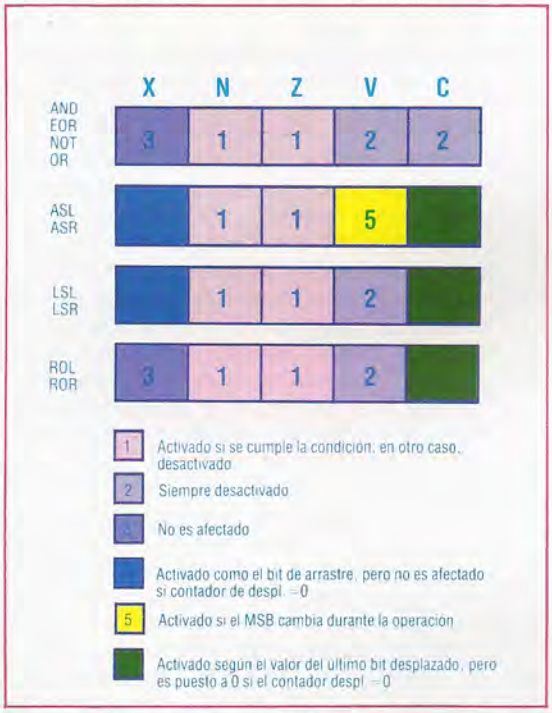
Así si ejecutamos:

```
BCHG    #4,D0
```

en el ejemplo anterior (donde D0 está puesto a 1) entonces D0 se convertiría en D0=XXXX XXXX XXX0 0000 y Z se pone a 0, que indica el estado antes del cambio.

Es de notar que todas estas instrucciones realizan las operaciones de comprobación y alteración de bits en una sola instrucción. Esto puede ser importante en un entorno de multiprogramación, donde la posibilidad de ser interrumpido entre dos instrucciones puede conducir a resultados impredecibles. Una observación final sobre las instrucciones lógicas es que aún si se aplica la comprobación, no es necesario realizar acción alguna a partir de esa información. Se puede emplear la instrucción como manipulador de bits.

Ante todo, veamos lo que sucede cuando un patrón de bits es desplazado a la izquierda o a la derecha. Si D0 contiene 0000 0000 0000 1000, el desplazamiento de este modelo tres lugares a la derecha dará 0000 0000 0000 0001. El contenido de D0 ha sido dividido por ocho (o sea, 2^3), en otras palabras, un desplazamiento hacia la derecha corresponde a una división por dos por cada lugar que se desplace a la derecha. Además, se sigue de esto



Estado al día
Las instrucciones lógicas, de-desplazamiento y de rotación afectan el contenido del registro de estado de la manera que aquí se ilustra. Las instrucciones de desplazamiento aritmético difieren de los desplazamientos lógicos sólo en el empleo que hacen del flag V para indicar cambios posibles en el bit de signo del operando

que un desplazamiento a la izquierda equivale a una multiplicación por idéntica cantidad.

Obsérvese que, en nuestro ejemplo, hemos supuesto que se rellenan con ceros los nuevos dígitos binarios (por la izquierda en los desplazamientos a la derecha, y viceversa). Esto se deberá cambiar si deseamos conservar el signo de nuestro número cuando hacemos un desplazamiento a la derecha. Así, por ejemplo, si D0=1111 1111 1111 0000 (−16, en decimal) un desplazamiento a la derecha de tres lugares daría D0=1111 1111 1111 1110, que es −2 en decimal. Aquí hemos puesto unos por la derecha para poder mantener el signo negativo del número. En general, podemos decir que para desplazamientos a la derecha el signo se conserva introduciendo bits del mismo signo que el bit que indica el signo (en el operando de datos es el bit más significativo). Para desplazamientos a la izquierda, introduciremos siempre ceros en los bits menos significativos de la palabra. Si deseamos conservar el signo del operando de esta manera entonces la operación de desplazamiento se denomina *desplazamiento aritmético*.

En el 68000 estas instrucciones de desplazamiento aritmético son ASL para el desplazamiento aritmético a la izquierda (*Arithmetic Shift Left*) y ASR para el desplazamiento aritmético a la derecha (*Arithmetic Shift Right*). Si el destino es un registro de datos, entonces podemos desplazar hasta ocho bits empleando el modo inmediato para el número de desplazamientos, o podemos servirnos del contenido de otro registro de datos como contador de desplazamientos hasta 31 bits en una palabra larga. Así, por ejemplo:

```
ASR     #8,D0
```

será el desplazamiento máximo en el modo inmediato, y:

```
ASL     D1,D0
```

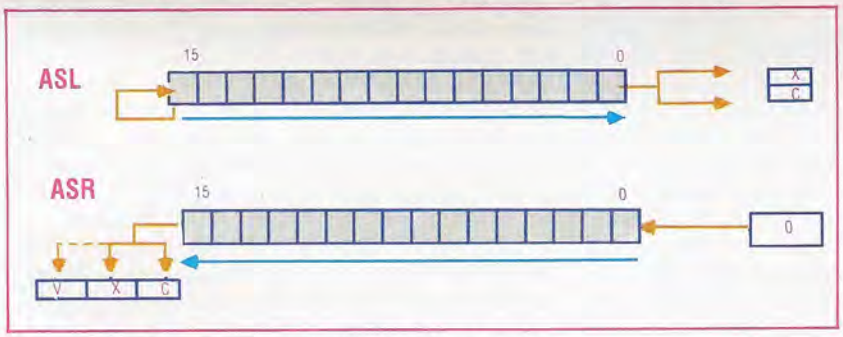
nos permitirá desplazar D0 a la izquierda un número de lugares indicado por el contenido de D1 (esto es, hasta 31 lugares). Si, no obstante, deseamos desplazar una posición de memoria, entonces la única opción posible será desplazar un lugar cada vez. Por ejemplo, ASR JUAN desplazará la posición de memoria indicada por JUAN un lugar a la derecha.

Se observará, por el dibujo del control de estado (página contigua), que se afectan todos los códigos de condición del SR (los bits C y X señalan los bits descartados por el desplazamiento, y el bit V indica cualquier cambio del bit de signo en desplazamientos a la izquierda). Los restantes bits, el N y el Z, reciben el valor acorde con el resultado.

Puede que usted se pregunte por qué se atribuye tanta importancia a estos desplazamientos aritméticos, considerando sobre todo que el 68000 está provisto de instrucciones para multiplicar y dividir. La razón está en que el tiempo de ejecución de estos dos tipos de operaciones es diferente. Una operación MULT tarda 70 ciclos de reloj y una ASL o ASR tan sólo tarda 6+2n ciclos, donde "n" es el número de desplazamientos ejecutados. Así, el tiempo de ejecución de un desplazamiento aritmético está entre los ocho ciclos para un solo desplazamiento y los 68 ciclos para un desplazamiento completo de 31 bits. De esta manera, para un reducido número de desplazamientos, las instrucciones de desplazamiento serán considerablemente más eficientes que las instrucciones de multiplicar o dividir. En el caso extremo de una instrucción de dividir por dos ¡se conseguiría una rapidez 19 veces mayor!

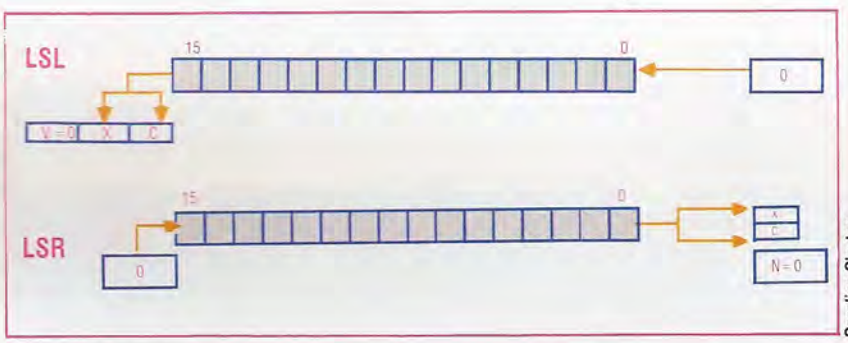
Obsérvese que también disponemos de instrucciones de *desplazamiento lógico*, LSL y LSR, que no conservan el signo del operando de datos e introducen siempre ceros en los nuevos dígitos binarios. Las restricciones de direccionamiento de los desplazamientos aritméticos son también válidas en los desplazamientos lógicos, y el bit V siempre se pone a cero. El desplazamiento lógico se utiliza provechosamente para establecer o comprobar operandos de datos que contengan subgrupos o campos más pequeños.

Finalmente en este conjunto de instrucciones de desplazamiento toca el turno al grupo de instrucciones rotatorias. Son sin duda una reliquia histórica,

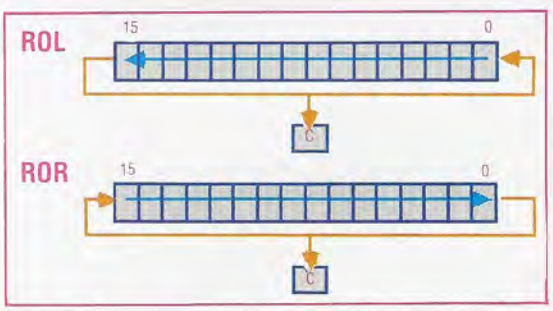


Desplazamiento aritmético
Cuando se desplazan los bits a la derecha (ASR), la operación aritmética copia el bit 15 en el bit 14, pero el contenido del bit 15 permanece inalterado. El bit de signo se respeta. El bit 0 se copia tanto en el bit C como en el X del registro de estado (SR). Obsérvese que cuando se efectúa un desplazamiento a la izquierda, los ceros se copian en el bit 0, y el bit V se pone a 1 si el cont. del bit 15 ha cambiado

Desplazamiento lógico
Las operaciones de desplazamiento lógico se limitan a introducir ceros bien sea en el bit 15 bien en el bit 0, según que el desplazamiento sea a la izquierda o a la derecha. Pero debe tenerse en cuenta que V se pone a cero durante la LSL, y N se pone a cero durante la LSR. El último bit que se desplaza fuera se almacena en C y en X en ambas operaciones



Caroline Clayton



Instrucciones de rotación
Las instrucciones ROTATE sólo hacen rotar a los operandos, poniendo el bit 0 en el bit 15 durante ROR y viceversa durante ROL. Obsérvese que el bit "intercambiado" se copia en el bit C del registro de estado durante la operación

dado que su principal finalidad es la comprobación de bits singulares de operandos de datos mediante la rotación y comprobación del estado resultante en el flag de condición. Naturalmente, en el 68000 disponemos de un conjunto completo de instrucciones para comprobar y modificar bits, y tardan casi el mismo tiempo que el conjunto de las rotatorias. Quizá pueda bastar aquí un sencillo ejemplo de rotación. ROR #3,D0 rotará el contenido de D0 tres lugares a la derecha, y dará al bit C un valor acorde con el último bit que ha dado la vuelta, yendo del menos significativo al más significativo.

Avancemos algo más para examinar las instrucciones de control de programas. Se trata de un importantísimo conjunto de instrucciones que controlan la secuencia de ejecución. Un grupo, denominado *bifurcaciones condicionales*, alterará el flujo secuencial normal de las instrucciones en función de una condición que se comprueba. Las *bifurcaciones incondicionales*, por su parte, siempre producirán una bifurcación o cambio del flujo normal secuencial de las instrucciones. Examinaremos primero estas últimas.

El modo habitual de ejecutar una bifurcación incondicional es:

```
BRA      NUEVAETIQ
```

donde el flujo de instrucciones continuará a partir de la posición denominada NUEVAETIQ una vez ejecutada la instrucción BRA. Si el desplazamiento de un byte puede guardarse en una palabra de ocho bits con signo, la instrucción resultante se codificará en una palabra. La mitad de la palabra contendrá el opcode BRA (60 en hexa) y el otro byte contendrá el desplazamiento de 16 bits con signo. Si el desplazamiento no es posible contenerlo de esta manera, o no se conoce todavía la dirección de bifurcación (es decir, el assembler no puede calcularla porque implica una referencia ulterior), entonces el byte de desplazamiento contendrá valor cero y la palabra siguiente contendrá el desplazamiento de 16 bits completo con signo.

Por lo general, la instrucción BRA será la adecuada; no obstante, en algunas ocasiones desearemos hacer algo más refinado cuando se calcula la dirección de la bifurcación. Por ejemplo, cuando se quiere bifurcar a una dirección contenida en una tabla con un índice establecido en un registro (recuérdese que esta manera de direccionar se denomina *indirecta con índice y desplazamiento*). Por desgracia, la instrucción BRA no acepta esta forma de dirección calculada, y así Motorola ha proporcionado la instrucción JMP (*jump*: saltar), donde es posible el cálculo de la dirección de bifurcación si es preciso.

Veamos un ejemplo de programa que ilustra las diferentes formas de bifurcación incondicional. La lista adjunta (*De un lugar a otro*) ilustra los principios en que se basa. Las instrucciones de salto en este ejemplo muestran un direccionamiento absoluto (hacia adelante y hacia atrás) e indirecto con indexación y desplazamiento. Otros modos de direccionamiento permitidos son el *indirecto simple* —por ejemplo, (A2)— y el *relativo al PC*.

Las instrucciones BRA muestran primero un ejemplo donde el desplazamiento de la dirección se contiene en la palabra de instrucción, y después dos ejemplos donde se utilizan palabras de extensión completa. En el primero de ellos (BRA FINISH) se ha utilizado una extensión completa de palabra aunque el desplazamiento puede ser contenido en un byte. Esto se debe a que el assembler no conoce la dirección de FINISH todavía, y por ello debe proveer una extensión completa de palabra para el desplazamiento. Si se sabe que el desplazamiento hacia adelante se ajusta a un byte con signo, entonces se puede obligar al assembler a que utilice la forma abreviada de la instrucción mediante el sufijo .S (*short*: breve). Así, nuestro ejemplo podría ser más adecuadamente escrito como BRA.S FINISH.

Examinemos ahora el segundo grupo de instrucciones de bifurcación: las *bifurcaciones condicionales*. Este grupo se subdivide en tres subgrupos:

- Bifurcaciones de complemento a dos
- Bifurcaciones sin signo
- Control de bucles

Los dos primeros subgrupos tienen un formato común en las instrucciones:

```
Bcc ETIQ
```

donde cc se refiere al código de condición que se comprueba. Si esta condición es verdadera, entonces tiene lugar una bifurcación a ETIQ; en caso contrario se ejecuta la siguiente instrucción de la secuencia. La condición comprobada se muestra en el cuadro *Condiciones para bifurcar* (derecha).

La columna "Verdadero si" del cuadro es la condición aritmética resultante de la comparación por medio de CMP o la instrucción SUB (que, obviamente, se ejecuta inmediatamente antes de aplicarse la bifurcación condicional). En el caso del primer subgrupo de condiciones mostrado en el cuadro (las bifurcaciones del complemento a dos), el bit V, o flag de desbordamiento, se incluye en todo caso en la comprobación lógica y éste es el factor que determina la pertenencia a este subgrupo. Implica además la necesidad de una comprobación adicional del bit de desbordamiento para una corrección completa, por lo que es preciso examinar cuidadosamente las condiciones lógicas para las bifurcaciones indicadas en el *Manual del usuario*. Por ejemplo, una bifurcación BGE comprueba si N=V, y habrá una bifurcación si se cumple NOT N AND NOT

De un lugar a otro

```
                    WAYOFF EQU     $3FFF
                           ORG     $1000
100     3200       START  MOVE    D0,D1
1002    4EF8 1000         JMP     START     *Extensión pal. completa
1006    4EF8 101C         JMP     FINISH    *De nuevo pero hacia adel.
100A    4EE8 0005         JMP     5(A0)     *Indirecto con desplaz.
100E    4EF2 000C         JMP     12(A2,D0  *Indirecto con índice y
                                             desplazamiento

*Versiones que siempre bifurcan

*Desplaz. en la palabra del opcode si está en un byte

1012    60EC              BRA START         *Desplazamiento negativo
1014    6000 0004         BRA FINISH        *Hacia adelante, sin saberse
                                             cuánto hacia adelante
1018    6000 2FE5         BRA WAYOFF        *Da una pal. de desplaz.
                                             de 16 bytes completa
101C    3200       FINISH MOVE D0,D1
```

V (es decir, si no hay desbordamiento y no es negativo), o cuando ambos N y V son verdaderos (desbordamiento y negativo).

Veamos un ejemplo. Supongamos que se desea comparar los dos números con signo contenidos en D1 y D2 y bifurcar hacia MAYOR si D2 es más grande que D1. Si la condición se comprueba antes con una instrucción comparativa, se emplea la bifurcación condicional BGT como sigue:

```
CMP   D1,D2        *forma D2-D1
BGT   D2MAYOR      *bifurca a D2MAYOR si no es
                   *cero ni negativo y no hay
                    desbordamiento
```

El segundo subgrupo de bifurcaciones condicionales ilustrado en el cuadro trata los números sin signo y la comparación de los códigos de condición es relativamente más sencilla. Por ejemplo, para comparar el contenido de la posición LUISA con D1, con independencia de cualquier condición de desbordamiento, se puede ejecutar esta secuencia:

```
CMP   LUISA,D1     *forma D1 - LUISA
BEQ   IGUAL        *bifurca a IGUAL si z=1
```

Un segundo ejemplo de estas bifurcaciones condicionales se da cuando se codifica en ensamblador para la realización de un bucle. Por ejemplo, el ensamblador equivalente de:

```
FOR I:=1 TO 5 hacer
(...parte principal del programa a ejecutar 5 veces...)
NEXT I
```

sería:

```
      MOVEQ  #5,D7   *establece contador bucle
LOOP  (...parte princ.   prog. a ejecutar 5 veces...)

      SUBQ   #1,D7   *decrementa el contador
      BNE    LOOP    *repite hasta que D7=0
```

quedando codificado eficazmente en sólo tres palabras (dado que se han usado las instr. rápidas).

El tercer subgrupo de bifurcaciones condicionales está formado por una única instrucción, DBcc (*Decrement and Branch*: decrementar y bifurcar al cumplirse la condición cc). Esta instrucción es una ampliación del programa de control de bucles dado más arriba, pero codificando tanto el decremento como la bifurcación condicional en una sola instrucción. De hecho la instrucción imita el pseudo-code típico del PASCAL para el bucle reiterativo:

```
REPEAT
  (...cuerpo del bucle...)
UNTIL
  'c'=verdadero o Dn=-1
```

donde cc es una de las condiciones descritas en el segundo subgrupo ilustrado en el cuadro y Dn es un registro de datos empleado para retener el contador del bucle. Veamos cómo puede codificarse con la instrucción DBcc:

```
      MOVEQ #5,D1 *establece contador bucle
LOOP  (...cuerpo del programa...)
      DEBQ D1,LOOP *salida si el último 'cc' es 0
                   *o bien D1=1 (6 iteraciones)
```

Obsérvense con cuidado las condiciones de salida explicadas en los comentarios anteriores, y cómo además el sentido de la bifurcación es diferente respecto de la instrucción BEQ normal. Si la comprobación condicional no es necesaria, entonces puede emplearse una cc de F (de "falso") para dar el equivalente DBcc de un simple bucle con FOR. Por ejemplo:

```
      MOVEQ #4,D3
LOOP  (...cuerpo del bucle FOR...)
      DBF   D3,LOOP
```

Lo cual equivale a nuestro bucle FOR original de cinco iteraciones, y hasta ocupa la misma memoria (todas las instrucciones DBcc ocupan dos palabras). Algunos programadores opinarán, sin embargo, que nuestra codificación primera era más explícita.

Condiciones para bifurcar

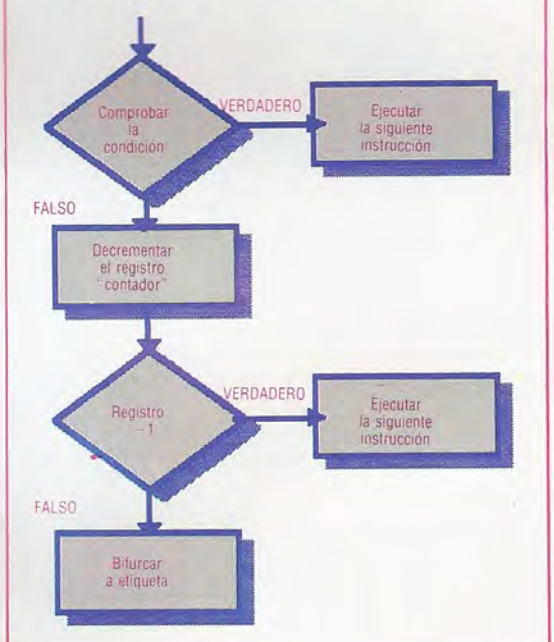
Las siguientes condiciones pueden comprobarse antes de una bifurc. hacia una dir. calculada por medio de la aritmética del complemento a dos

Condición cc	Verdadero si
GT mayor que	destino>fuente
LT menor que	destino<fuente
GE mayor o igual que	destino>=fuente
LE menor o igual que	destino<=fuente
VS desbordamiento	V=1
VC no hay desbord.	V=0

Las siguientes condiciones pueden comprobarse antes de realizar una bifurcación hacia una dirección calculada mediante enteros sin signo

EQ igual a cero	Z=1
NE no igual a cero	Z=0
MI menos	N=1
PL más	N=0
HI más alto que	C=Z=0
LS más bajo que	C o Z=1
CS arrastre activado	C=1
CC arrastre limpio	C=0

A condición de que...



Tiempo para comprobar
La DBcc proporciona al programador del 68000 varias y poderosas oportunidades de programación a alto nivel. El diagrama de flujo de la instrucción es el que aquí se muestra. Obsérvese que no sólo se limita a decrementar el registro contador (compararla con la instrucción DJNZ del Z80), sino que también comprueba la condición. A veces, sin embargo, podemos necesitar la ejecución de un bloque entero de código reiteradamente (como en el bucle FOR...NEXT del BASIC), pudiéndose emplear para este fin la instrucción condicionada FALSE (DBF)

Somos el mundo

Visualizaciones cortesía de "Your Spectrum"

Show benéfico
Vemos aquí una selección del software correspondiente a la versión de SoftAid para el Spectrum. Los juegos escogidos para la compilación eran todos clásicos de estilo recreativo. Los royalties y los beneficios generados por las ventas de la cassette se han donado al Band-Aid Trust

SoftAid es una organización creada por un grupo de firmas de software con el fin de recaudar fondos destinados a paliar el hambre en África

Es así como, unificando sus recursos, SoftAid ha producido una cassette de compilación que ha logrado reunir miles de libras que se entregarán a la institución benéfica Band-Aid. Los esfuerzos de esta institución, dirigida por Bob Geldof, tuvieron eco en el mundo. Tras el éxito del disco single *Do they know it's Christmas?* a fines de 1984, y los conciertos Live Aid en Londres y Filadelfia (Estados Unidos) en julio de 1985, numerosas organizaciones de otros campos del mundo del entretenimiento han ofrecido sus servicios gratuitamente para recaudar fondos para dicha causa. Una de las iniciativas que tuvieron mayor aceptación provino de las casas de software de juegos, a las que a menudo les agrada compararse con la industria de la música *pop*. El resultado fue SoftAid, una serie de cintas para los micros Commodore y Spectrum que incluían algunos de los juegos de mayor éxito en el mercado.

La fuerza motriz de SoftAid ha sido el Guild of Software Houses (GOSH), asociación gremial de la industria de juegos por ordenador cuya función habitual es intentar desterrar la piratería de software. La directiva del gremio preguntó a cada uno de sus asociados si estaban dispuestos a contribuir con un juego al proyecto. La idea original era que las casas de software escribieran juegos nuevos para la compilación, pero luego se decidió que era más factible recopilar juegos cuya popularidad entre el público ya hubiera sido demostrada. De modo que estos juegos se incorporaron en un único paquete al precio especial de £4,99.

Las casas de software mostraron una predisposición tan favorable a colaborar con el proyecto, que GOSH se encontró con más juegos de los que podía incluir en una única cassette para cada una de las dos máquinas para el que se había preparado originalmente la compilación SofAid. En consecuencia, hubieron de rechazarse ofertas de juegos de empresas tan conocidas como Llamasoft y Software Projects. El software seleccionado incluía algunos de los juegos más populares de los últimos años, entre ellos el clásico *Ant attack*. Los juegos que se incluyeron en la cinta para el Spectrum son: *Spellbound*, de Beyond; *Starbike*, de The Edge; *Kokotoni Wilf*, de Elite; *The pyramid*, de Fantasy; *Horace goes ski-ing*, de Melbourne House; *Gilligan's gold*, de Ocean; *Ant attack*, de Quicksilva; *3D tank duel*, de Real Time; *Jack and the Beanstalk*, de Thor, y *Sorcery*, de Virgin. Para el Commodore 64 los juegos incluidos son: *Gumshoe*, de A'n'F; *Beam rider*, de Activision; *Star trader*, de Bug-Byte; *Gyropod*, de Taskset; *China miner*, de Interceptor; *Kokotoni Wilf*, de Elite; *Gilligan's gold*, de Ocean; *Fred*, de Quicksilva; *Falcon patrol*, de Virgin, y *Flak*, de US Gold. Durante el desarrollo del proyecto, GOSH trabajó en estrecha colaboración con el Band-Aid Trust y, como atractivo adicional, se añadió al principio de cada cassette el single de Band-Aid *Do they know it's Christmas?*, proporcionando así otro incentivo para adquirir la interesante compilación.

Asimismo, GOSH respetó la filosofía general de Band-Aid de solicitar a todos los participantes en la producción y distribución de la cassette que prestaran sus servicios de forma gratuita. No sólo los juegos se cedieron gratuitamente, sino que todas las personas implicadas renunciaron a cobrar los gastos de impresión, publicidad, reproducción de cintas, distribución y venta al público, teniéndose que pagar solamente el costo de los materiales. Las cintas se pusieron a la venta en la primavera de 1985; inmediatamente se colocaron en los primeros puestos de las listas de *best-sellers* para las dos máquinas en cuestión, y consiguieron recaudar muchos miles de libras para el Band-Aid Trust. Y todo ello a pesar de que muchos de los clientes que adquirieron las cintas probablemente ya tenían en sus colecciones particulares algunos de los juegos incluidos en la compilación.

SoftAid: Para el Sinclair Spectrum y el Commodore 64. (Se están preparando versiones para el BBC Micro y la gama Amstrad.)
Editado por: The Guild Of Software Houses, 12-13 Henrietta Street, London WC2, Gran Bretaña
Autores: Varios
Palancas de mando: Opcionales
Formato: Cassette

Intuición e imaginación

Se asegura que la aparición del Commodore Amiga constituirá un acontecimiento debido a su innovador diseño y excepcional rendimiento

El Amiga fue desarrollado originalmente por la firma norteamericana Amiga Corporation y posteriormente sus derechos fueron adquiridos por Commodore. Si bien la máquina ha aparecido en Estados Unidos bajo el patrocinio de Commodore, ninguno de sus distintivos menciona el nombre de la compañía. Debido a que el ordenador está dirigido al mercado de gestión o al usuario serio, y a que en dicho país Commodore está considerada como un pequeño fabricante de máquinas de juegos (en gran parte a consecuencia del éxito del Commodore 64), la empresa decidió comercializar la máquina con una imagen completamente nueva.

Alojado en una única carcasa con una unidad de disco incorporada, el Commodore Amiga tiene un aspecto muy de gestión. El teclado, que está separado del ordenador y conectado mediante un generoso trozo de cable, tiene patas en la parte posterior, que se pueden levantar para elevar en ángulo las teclas y obtener mayor comodidad al teclear. Si bien el tacto del teclado no es el mejor de los existentes, ciertamente es adecuado para la mayoría de las aplicaciones. Abajo y a la derecha de las teclas de máquina de escribir hay un grupo de teclas de cursor que se puede utilizar para reproducir las funciones del ratón de mano.

El Amiga viene con una unidad de disco de 3½ pulgadas de doble cara, en la que los fabricantes han conseguido insertar hasta 880 Kbytes de datos. Esto permite al Amiga acceder a más datos en una sola unidad que muchos ordenadores con unidades gemelas.

El panel frontal a la izquierda de la unidad revela, al quitarlo, una ranura para ampliación. Dotado con 256 Kbytes de memoria como estándar, el Amiga puede acomodar un módulo de memoria adicional que simplemente se introduce en el panel frontal, con lo que la máquina alcanza los 512 Kbytes. Para ampliar el Amiga hacia la configuración máxima de ocho Mbytes, en el lado derecho de la carcasa del ordenador se encuentra una segunda ranura a través de la cual se puede conectar en interface la memoria adicional. En este lado de la máquina también se proporciona un par de conectores D de nueve patillas para palancas de mando o, lo más importante, para el controlador de ratón.

La parte posterior aloja las conexiones para interface de periféricos. Desde la izquierda, éstas son la puerta para teclado, la puerta para impresora Centronics y una interface para una segunda unidad de disco. Asimismo, el Amiga posee una conexión RS-232C para modem y otros periféricos en serie y un par de conectores de tipo *phono*. Éstos se pueden enchufar directamente en la pantalla especializada que se proporciona para el Amiga o, lo que es más conveniente, en un sistema de *hi-fi* en el cual se podrá apreciar cabalmente la calidad del sonido del ordenador.

Marcus Wilson-Smith

Un salto evolutivo
El Commodore Amiga representa un notabilísimo desarrollo en la evolución del microordenador. El uso intensivo de chips construidos a medida, aplicando tecnología *blitter*, mejora enormemente las capacidades de gráficos y sonido de la máquina, mientras la CPU queda libre para manejar otros procesos

Las tres puertas restantes están reservadas para funciones de video, incluyendo un conector de 23 vías para monitores RGB y un conector para video compuesto. También hay disponible un conector *video in* que permite que el Amiga entre imágenes de video desde un VCR y visualice encima de esas imágenes otras generadas por ordenador, algo así como el sistema Pioneer PX-7 (véase p. 2069). Es probable que entre los futuros accesorios de hardware para el Amiga se incluya un *frame-grabber* (manipulador de ventanas) que digitaliza un fotograma de una fuente de video, que después se puede reflejar, rotar o tratar de otro modo bajo control de software.

En el centro del Amiga está el procesador 68000 de Motorola (como en el Apple Macintosh y el Atari 520ST), pero lo que en realidad le proporciona al Amiga su sobresaliente rendimiento es la influencia que ejercen, entre sí y sobre el 68000, tres chips hechos a medida. Denominados Agnus, Portia y Daphne, los chips pueden controlar visualización de video, sprites y E/S de disco de forma más o menos autónoma, dejando libre al 68000 para que lleve a cabo las aplicaciones de proceso a toda velo-

Todo en las teclas
El teclado que viene con el Amiga es uno de los mejores que hay actualmente en el mercado. Siguiendo las tendencias modernas, incluye un teclado numérico, 10 teclas de función programables y otras numerosas teclas que se pueden utilizar en funciones de control

La firma Metacomco

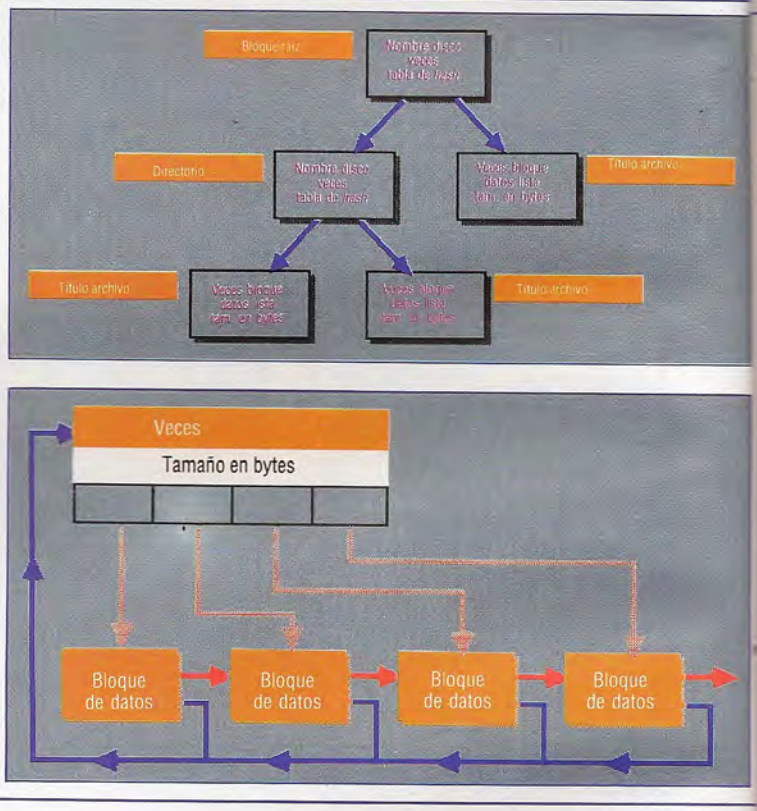
La casa de software Metacomco, con sede en Bristol, se creó en 1981 para desarrollar software de sistemas para ordenadores de 16 y 32 bits. Su Personal BASIC, desarrollado para Digital Research, se ha convertido en un estándar en ordenadores que ejecutan CP/M-86. Más recientemente, Metacomco ha creado software 68000 y producido versiones de PASCAL, LISP y C para el Sinclair QL y el Atari 520ST, así como ensambladores 68000 y paquetes para desarrollo de software. Metacomco fue contactada para escribir el AmigaDOS para el bisoño Amiga cuando otra casa de sistemas, contratada inicialmente para desarrollar un OS, falló en entregar el producto. El AmigaDOS se basa en un sistema de red multitareas conocido como Tripos, creado hacía algunos años en la Universidad de Cambridge. Al doctor Tim King, jefe del departamento de I&D de Metacomco, se le

cidad. Agnus, por ejemplo, posee su propio coprocesador y «manipulador de imagen de bits» (*blitter*) que le permite mover un millón de pixels por segundo. La lógica para dibujo de líneas y relleno de formas también forma parte del sistema de circuitos de Agnus. Todo esto significa que el Amiga posee la capacidad de mover, alterar y rellenar formas tan rápidamente que crea la ilusión de hacerlo de forma instantánea. Además, el blitter se utiliza para transferir datos de disco entre los *buffers* de disco y la memoria.

Hay disponibles dos tipos de sprites. Los primeros, conocidos como *Vsprites*, se controlan desde hardware, permitiendo su movimiento alrededor de la pantalla a gran velocidad. El segundo tipo de sprites se controla desde software utilizando el blitter. Estos "objetos de blitter" (*bobs*) permiten formas y coloreados más complejos de los que se pueden conseguir con los Vsprites. La suma de todas estas características hace que el Amiga pueda producir gráficos a una velocidad extraordinaria y de una calidad sin precedentes, que antes sólo se podían obtener en máquinas recreativas muy especiales.

Las capacidades de sonido también son notables, de una calidad equiparable a la de muchos sintetizadores comerciales. El Amiga puede reproducir digitalmente sonidos reales muestreados que se pueden manipular en estéreo en cualquier altura dada. También se incluye como estándar la síntesis de voz, que puede hablar con voces masculinas o femeninas, añadir inflexiones y procesar texto escrito. Dos de las aplicaciones que siempre saltan a la imaginación son procesadores de textos que posean la facultad de leerle texto al usuario o de leerle los mensajes de su buzón de correo electrónico. Pues bien, el BASIC que se proporciona con el sistema (AmigaBASIC) soporta todas estas facilidades y permite producir gráficos excelentes a partir de unas pocas líneas de programa. La síntesis de voz la soportan instrucciones de BASIC que traducen un texto a una serie de fonemas, que después son "hablados".

El Amiga presenta al usuario un entorno tipo WIMP amable, llamado Workbench, que sigue el estilo del Macintosh y las máquinas basadas en GEM. Los archivos se pueden cargar apuntando a los iconos adecuados y pulsando un botón del ratón. Debajo de éste se halla el sistema operativo propio, conocido como AmigaDOS, que hace que la máquina sea auténticamente multitarea. Amiga-



Cuadros de una exposición

La calidad de los gráficos del Commodore Amiga lo colocan muy por encima de otras máquinas de su misma escala de precios. La compleja animación y sus gráficos en alta resolución con sutil coloreado hacen del Amiga una máquina recreativa por excelencia. Es probable que su entorno operativo WIMP estilo Macintosh atraiga a muchos usuarios no especializados, tales como diseñadores gráficos y artistas

El Amiga opera a través de una interface icónica, conocida como Workbench. Están presentes las conocidas ventanas, iconos de disco y de cubo de desperdicios

asignó la tarea de adaptar el Tripos y convertirlo en un sistema operativo funcional para el Amiga en el plazo de un mes. De este modo, Metacomco se comprometió en el proyecto Amiga, produciendo el AmigaBASIC (el intérprete de BASIC empaquetado con la máquina), PASCAL y LISP. Asimismo, ha escrito un macroensamblador y sistemas de desarrollo que se ejecutan bajo Unix y MS-DOS. Si bien el Amiga DOS no soporta conexión en red, el Tripos es fundamentalmente un OS de red; los planes apuntan hacia futuras versiones del AmigaDOS que permitan la conexión en red de hasta 255 Amigas. Para Tim King, este tipo de sistema es una alternativa a las facilidades de la informática de ordenadores de unidad central. La red es suficientemente flexible para aceptar máquinas extras cuando sea necesario, cada una de ellas capaz de utilizar los periféricos de otra estación o de acceder a un "servidor" de archivos centralizado equipado con un disco rígido

Sistema de archivos

El sistema de archivos que utiliza el AmigaDOS es inusual. Se basa en una estructura arborescente con un complejo conjunto de punteros hacia adelante y hacia atrás que unen los bloques del disco. No existe ninguna pista de directorio como tal, sino un bloque "raíz" con punteros a títulos de archivos u otros directorios.
El título de archivo contiene una serie de punteros a cada bloque de datos del archivo y otra información sobre el archivo (como su tamaño expresado en bytes y el momento en que se accedió al mismo por última vez). Los bloques de datos también están encadenados entre sí mediante una serie de punteros hacia adelante y cada bloque de datos también apunta hacia atrás, al título de archivo al cual pertenece. Este complejo sistema de punteros tiene una implicación muy importante. Si un disco se corrompe, a partir de apenas un único bloque "bueno" del disco se puede recuperar la totalidad de la estructura de archivos del disco, siguiendo y rehaciendo los enlaces de punteros entre los bloques. La integridad de los datos es de gran importancia, en particular para los usuarios de gestión, y la capacidad del Amiga para recuperar discos estropeados probablemente contribuirá a aumentar su atractivo

DOS, a su vez, hace llamadas a Intuition, la parte del firmware que se encarga del control de ventanas y ratón.

En la práctica, las capacidades para multitarea del Amiga permiten ejecutar varias aplicaciones de forma simultánea y a la vez independiente. Es probable que éste sea uno de los principales argumentos de venta para el sector de gestión, porque se trata del primer micro de su escala de precio que proporciona tales facilidades. Debido a que las capacidades de proceso de la máquina están compartidas sobre una base de "repartición de tiempo" (siendo objeto cada aplicación de cortos intervalos de atención por turno), se produce una proporcional pérdida de velocidad mientras más aplicaciones se ejecuten juntas en multitarea.

En Estados Unidos, el dominio que mantiene IBM en el mercado de gestión y el rechazo de los usuarios de ordenadores de oficina a la idea de probar máquinas nuevas (a ello se debe la abundancia de clonos IBM existente en el mercado), plantean un grave problema a los fabricantes de ordenadores que desean introducir sus productos en dicho mercado. Aunque el Amiga supera fácilmente al IBM PC por un precio cercano a la mitad, Commodore ha intentado también asegurarse el éxito del Amiga produciendo una unidad de disco de 5½ pulgadas y una opción de software de emulación que hacen que la máquina sea compatible con el IBM. Commodore sostiene que el mismo permite que el Amiga ejecute paquetes tales como el *Lotus 1-2-3*. La técnica de emulación, en esencia, traduce opcodes 8088 a opcodes 68000. Evidentemente, este proceso reduce la velocidad del Amiga, pero existen planes para producir una placa de emulación de hardware que solucionaría este problema.

El Amiga sienta nuevos estándares desde el punto de vista de velocidad, gráficos y sonido, y quizá esté ampliamente justificada la afirmación de su fabricante en el sentido de que su innovador enfoque le hará ganar nuevos mercados.

El Amiga podría catalogarse como una máquina de juegos excelente pero cara, o como una máquina de gestión potente y económica. De modo que, en algún sentido, el ordenador podría experimentar una crisis de identidad, cuya resolución dependería de la respuesta que obtuviera en el mercado.

Commodore Amiga

DIMENSIONES
444 × 300 × 120 mm

CPU
Motorola 68000, operando a 8 MHz

MEMORIA
256 Kbytes, ampliables a 8 Mbytes

PANTALLA
Su resolución máxima para textos es de 80 × 25 caracteres. Las cuatro modalidades para gráficos disponibles van desde una modalidad de 320 × 200 pixels a 32 colores, hasta una modalidad de 640 × 400 pixels a 16 colores

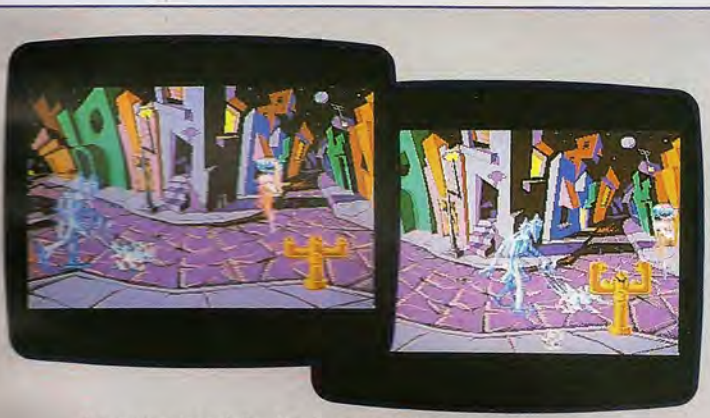
INTERFACES
Dos puertas para ratón, bus de ampliación, interface RS232, interface Centronics, puerta para segunda unidad de disco, puertas para video compuesto y RGB

TECLADO
82 teclas, incluyendo 10 teclas de función y teclado numérico

VENTAJAS
Permite la auténtica consecución de multitareas, y las capacidades de sonido y gráficos son sobresalientes

DESVENTAJAS
Gran parte del software prometido para la máquina aún no se ha materializado. Además, Commodore aún no ha decidido a qué mercado dirigirá la máquina. La conquista del público, y, en consecuencia, el éxito del Amiga, dependerán de la política de comercialización

Marcus Wilson-Smith



Los personajes móviles de esta escena animada se crean utilizando objetos *blitter*, "sprites" de software que permiten complejas definiciones de forma y color. Los objetos *blitter* incluyen detección de proximidad. Aquí, la figura de la boca de riego tensa sus músculos para mantener a raya al perro que se acerca

Como se demuestra aquí, el Amiga es un micro multitarea. Podemos ver ejecutándose juntos al Workbench y una demostración de una pelota rebotando

Si bien el AmigaBASIC no es una implementación excelente del lenguaje, sí soporta la mayoría de las facilidades de la máquina. Fue suficiente un programa en BASIC de cinco líneas para producir esta visualización garabateada con el ratón

Hagan sus apuestas

Finalizamos nuestro proyecto añadiéndole al juego las rutinas de apuestas e incluyendo una pantalla de títulos

Apostar al dar vuelta los naipes es una parte integral del juego del veintiuno (o *pontoon*). El programa le concede una cantidad inicial de £10 000. El restante capital del jugador se retiene a lo largo del juego en la variable SK. Existen varios métodos de apuesta. Adoptaremos el siguiente sistema:

- El jugador debe apostar £50 al comienzo de cada ronda para entrar en el juego.
- Esta apuesta inicial la realiza automáticamente el programa.

Habiéndosele repartido un naipe, el jugador puede "comprar" un segundo naipe haciendo una apuesta adicional de hasta £1 000. Un jugador perspicaz reconocerá que algunos naipes (un as, p. ej.) son más prometedores que otros y apostará en consecuencia.

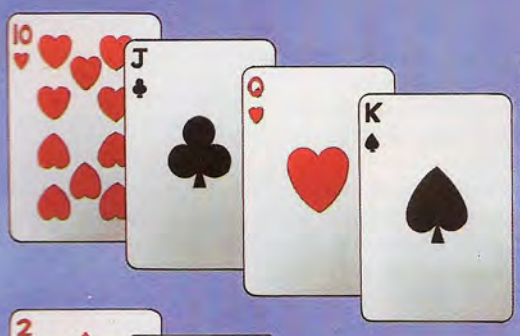
Al jugador se le pueden repartir naipes adicionales hasta que se plante o se pase. Si el jugador cree que posee una mano sumamente favorable, o si apuesta demasiado bajo en su primer naipe, entonces puede optar por doblar su apuesta y se le repartirá otro naipe más.

El jugador gana la ronda si, después de que el ordenador (la banca) juega su mano, su marcador resulta superior al de la banca. En este caso su apuesta se le devuelve y se le suma a su capital una cantidad adicional igual a su apuesta. De lo contrario, pierde el dinero apostado.

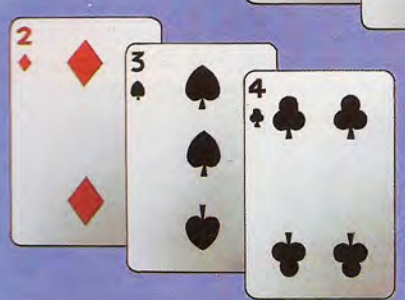
Estrategias de apuesta

Tal como sucede con todas las apuestas, las estrategias de éxito se basan en el principio de "ganar mucho y perder poco". La fase crucial de la apuesta en nuestra versión del veintiuno se identifica con el momento en que el jugador adquiere un segundo naipe. Abajo esbozamos algunas sugerencias sencillas para mejorar su rendimiento

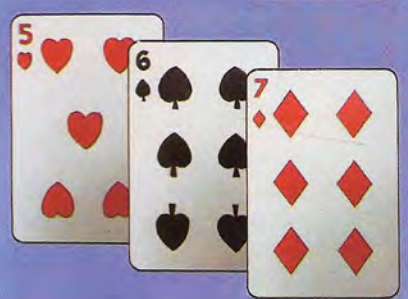
Vale un premio muy bueno
- Hasta 16 naipes en el mazo que den *royal pontoon/ pontoon*
- El as se cuenta como 1 u 11, valor flexible a medida que se desarrolla la mano
- Posible juego de cinco naipes



Vale un buen premio
- Subsiguientes 2, 3 o 4 permiten quemar los naipes
- Un segundo naipe alto (9, 10, J, Q, K) permite plantarse
- Un ulterior as da *pontoon/royal pontoon*
- El inconveniente está en que una ulterior carta de "orden mediano" (5 o 6) puede hacer que el tercer naipe haga pasarse la mano



Vale un premio mediano
- Los naipes bajos dan posibilidades de juego de cinco naipes
- Posibilidad de pasarse si el segundo naipe es alto



Sólo un premio bajo
- Los naipes de orden mediano pueden plantear problemas si se reparte un 9, 10, J, Q o K como segundo naipe

Kevin Jones

El pote de las apuestas

A lo largo del juego se visualizan en la pantalla la apuesta actual y los niveles de capital inicial restantes. Los títulos para este "pote" los imprime la rutina de la línea 4200, que es llamada por la rutina "inicializar el juego" de la línea 625. Esta línea también restablece una variable, BT, que se utiliza para llevar el registro de la apuesta efectuada durante la ronda actual.

La rutina "imprimir premio", que empieza en la línea 4300, es una rutina de propósito general a la que se puede llamar desde varios puntos del programa principal. Se realizan apuestas adicionales estableciendo la variable SB antes de llamar a la rutina. En circunstancias normales, SB se suma a la apuesta acumulativa, BT, y se resta del paquete inicial restante, SK. No obstante, como esta rutina es general, necesita realizar varias comprobaciones sobre la validez de la apuesta a efectuar. Si al jugador le quedan menos de £50, y se efectúa la apuesta automática inicial (indicada porque la bandera BG=1), la rutina imprimirá un mensaje anunciando que los fondos que quedan son insuficientes y da al jugador la opción de recomenzar el juego desde el principio.

Si el jugador desea doblar su apuesta, pero no dispone de capital suficiente, entonces se debe imprimir un mensaje y desautorizar la apuesta. Esta comprobación se efectúa restando la apuesta proyectada y viendo si queda un paquete inicial restante negativo, SK. De ser así, entonces se vuelve a sumar la apuesta y se sale de la rutina.

La tercera circunstancia inusual se produce si el jugador intenta comprar un segundo naipe apostando más dinero del que posee. En este caso, la rutina imprimirá un mensaje apropiado y reducirá la apuesta adicional a la cantidad de capital que le quede disponible al jugador.

La rutina pasa luego a borrar la apuesta impresa previamente y los valores de paquete inicial restante sobreimprimiendo algunos espacios antes de imprimir los nuevos valores.

La apuesta inicial de £50 se realiza automáticamente al comienzo de cada ronda mediante la línea 72. Las líneas 73-80 permiten que el jugador compre un segundo naipe, comprobando la cantidad entrada para asegurar que no supere el límite de £1 000.

Cuando el jugador pide para recibir más naipes en la rutina que comienza en la línea 2700, puede optar por doblar en su último naipe. La rutina para doblar propiamente dicha se halla en la línea 2900 y llama a la rutina "imprimir apuesta". Si el jugador no puede permitirse doblar (lo que se indicaría si CA=1), entonces se sale de la rutina y el programa revierte a la rutina pedir/plantarse normal. De lo contrario, se reparte al jugador un último naipe, se evalúa la mano y termina la rutina.

En el capítulo anterior de este proyecto desarrollamos las rutinas que deciden qué mano ha ganado y luego imprimen un mensaje "ganar o perder", según convenga. Sólo resta añadir una línea que sume la cantidad apostada al total de paquete inicial restante en el caso de que el jugador gane la

Rutinas de apuestas

BBC Micro

```
 30 GOSUB 4000
 72 BG=1:SB=IS:GOSUB 4300:BG=0
 73 GOSUB 700:PRINT"COMPRAS UN NAIPE (S/N)";
 74 RESP$=GET$
 75 IF RESP$<> CHR$(13) THEN PRINT RESP$
 77 IF RESP$<> "S" THEN 85
 78 RESP$="":GOSUB 700:INPUT "TU APUESTA (MAX
    £1000)"RESP$
 79 IF VAL(RESP$)>1000 THEN 78
 80 SB=VAL(RESP$):GOSUB 4300
210 GOSUB 700:PRINT"GANAS £";BT
220 SB=-2* BT:BT=-SB:GOSUB 4300
565 SK=10000:IS=50
610 HP(1)=1:HP(2)=1
625 BT=0:GOSUB 4200
2740 IF RESP$="D" THEN GOSUB 2900:IF CA=0 THEN
     RETURN
2900 REM **** DOBLAR ****
2910 DB=1:SB=BT:GOSUB 4300
2915 IF CA=1 THEN DB=0:RETURN
2920 FL=0:PL=1:GOSUB 1300
2930 GOSUB 800
2940 DB=0:RETURN
4000 REM
4010 CLS
4020 TX=13:TY=3:GOSUB 900:PRINT"BBC/ELECTRON"
4030 PRINT TAB(TX+3);"PONTOON"
4040 PRINT TAB(TX+5);"POR"
4050 PRINT TAB(TX-7);"PETE SHAW & STEVE COLWILL"
4060 TX=9:TY=8:GOSUB 900:PRINT"TE QUEDAN £";SK
4070 TX=10:TY=12:GOSUB 900:PRINT"PULSA CUALQUIER
     TECLA PARA JUGAR"
4080 A$=GET$
4090 RETURN
4200 REM
4210 COLOUR 4:TX=24:TY=18:GOSUB 900:PRINT"TU
     APUESTA"
4220 TX=24:TY=20:GOSUB 900:PRINT"INICIAL RESTANTE"
4230 RETURN
4300 REM
4302 CA=0:REM NO SE PUEDE PERMITIR DOBLAR
4305 IF SK>=50 OR BG=0 THEN 4310
4306 RESP$="":GOSUB 700:INPUT"TE HAS QUEDADO SIN
     DINERO!" VUELVES A JUGAR (S/N)";RESP$
4307 IF RESP$="S" THEN RUN
4308 END
4310 SK=SK-SB:BT=BT+SB
4315 IF DB=1 AND SK< 0 THEN GOSUB 700:PRINT"NO TE LO
     PUEDES PERMITIR!"
4317 IF DB=1 AND SK< 0 THEN BT=BT-SB:CA=1:RETURN
4320 IF SK< 0 THEN BT=SK+SB:SK=0:GOSUB
     700:PRINT"SOLO PUEDES PERMITIRTE £";BT
4340 COLOUR 1:TX=24:TY=19:GOSUB 900:PRINT
     LEFT$(SP$,15)
4345 TX=24:TY=19:GOSUB 900:PRINT "£";BT
4350 TX=24:TY=21:GOSUB 900:PRINT LEFT$(SP$,15)
4355 TX=24:TY=21:GOSUB 900:PRINT "£";SK
4360 RETURN
```

Gama Amstrad CPC

```
 30 GOSUB 400:REM pantalla de titulos
 72 bg=1:sb=is:GOSUB 4300:bg=0:REM imprimir apuesta
 73 GOSUB 700:PRINT "compras un naipe (s/n)";
 74 resp$="":While resp$="":resp$=INKEY$:WEND
 75 IF resp$<>CHR$(13) THEN PRINT resp$
 77 IF resp$<>"s" THEN 85
 78 resp$="":GOSUB 700:INPUT"tu apuesta (max
    £1000)";resp$
 79 IF VAL(resp$)>1000 THEN 78
 80 sb=VAL(resp$):GOSUB 4300:REM imprimir apuesta
210 GOSUB 700:PEN negro:PRINT"ganas £";:PEN
    blanco:PRINT bt
220 sb=-2* bt:bt=-sb:GOSUB 4300:REM imprimir apuesta
565 sk=10000:is=50:REM paquetes
610 hp(1)=1:hp(2)=1
625 bt=0:sb=0:GOSUB 4200:REM imprimir pote apuestas
2740 IF resp$="d" THEN GOSUB 2900:IF ca=0 THEN RETURN
2900 REM **** doblar ****
2910 db=1:sb=bt:GOSUB 4300:REM imprimir apuesta
2915 IF ca=1 THEN db=0:RETURN
2920 fl=0:pl=0:GOSUB 1300:REM repartir
2930 GOSUB 800:REM evaluar
2940 db=0:RETURN
4000 REM **** titulo etc ****
4010 CLS
4020 tx=11:ty=3:GOSUB 900:PEN rojo:PRINT"Gama Amstrad
     CPC"
4030 PRINT TAB(tx+6)"Pontoon"
4040 PRINT TAB(tx+8)"Por"
4050 PRINT TAB(tx+3)"Steve Colwill"
4060 tx=9:ty=8:GOSUB 900:PEN negro:PRINT"Tu paquete
     inicial es de £";sk
4070 tx=10:ty=12:GOSUB 900:PEN blanco:PRINT"Pulsa una
     tecla para jugar"
4080 WHILE INKEY$="":WEND
4090 RETURN
4200 REM **** pote de apuestas ****
4210 tx=24:ty=18:GOSUB 900:PEN rojo:PRINT"Tu apuesta"
4220 tx=24:ty=20:GOSUB 900:PRINT"Paquete restante"
4230 RETURN
4300 REM **** imprimir paquete ****
4302 ca=0:REM no se puede permitir doblar bandera
4305 IF sk> =50 OR bg=0 THEN 4310
4306 resp$="":GOSUB 700:PEN rojo:INPUT"Te has quedadc
     sin dinero! Vuelves a jugar (s/n)";resp$
4307 IF resp$="s" THEN RUN
4308 END
4310 sk=sk-sb:bt=bt+sb
4315 IF db=1 AND sk<0 THEN GOSUB 700:PEN
     rojo:PRINT"No puedes permitirtelo!":bt=bt-sb:sk=
     sk+sb:ca=1:RETURN
4320 IF sk<0 THEN bt=sk+sb:sk=0:GOSUB 700:PEN
     rojo:PRINT"Solo puedes permitirte £";bt
4340 tx=24:ty=19:GOSUB 900:PRINT SPACE$(15)
4345 tx=24:ty=19:GOSUB 900:PEN blanco:PRINT"£";bt
4350 tx=24:ty=21:GOSUB 900:PRINT SPACE$(15)
4355 tx=24:ty=21:GOSUB 900:PRINT"£";sk
4360 RETURN
```

mano. Esto se realiza insertando la línea 220 en el bucle principal.

Ahora hemos completado los listados para cada una de las cuatro máquinas y el jugador podrá jugar al juego completo contra el ordenador. El programa barajará el mazo automáticamente al comienzo del juego y pulsando SHIFT/S (SYMBOL SHIFT/S en el Spectrum) se puede conseguir que lo vuelva a barajar automáticamente después de cada ronda.

En un proyecto futuro usted podría mejorar la visualización de naipes y las rutinas de reparto dadas y crear un juego del veintiuno para más de un jugador.

Sinclair Spectrum

```
 30 GO SUB 4000: REM PANTALLA DE TITULOS
 72 LET BG=1: LET SB=IS:GO SUB 4300: LET BG=0: REM
    IMPRIMIR APUESTA
 73 LET A$="": GO SUB 700: PRINT INK 2;"COMPRAS UN
    NAIPE (S/N) ";
 74 LET A$=INKEY$: IF $="" THEN GO TO 74
 75 IF AS<> CHR$ 13 THEN PRINT A$
 77 IF A$<> "S" THEN GO TO 85
 78 LET A$="": GO SUB 700: INPUT "TU APUESTA (MAX
    £1000)";LINE A$
 79 IF VAL A$> 1000 THEN GO TO 78
 80 LET SB=VAL A$: GO SUB 4300: REM IMPRIMIR
    APUESTA
210 GO SUB 700: PRINT "GANAS £";BT
220 LET SB=-2*BT: LET BT=-SB: GO SUB 4300: REM
    IMPRIMIR APUESTA
565 LET SK=10000: LET IS=50: REM PAQUETES
610 LET P(1)=1:LET P(2)=1
625 LET BT=0:GO SUB 4200:REM IMPRIMIR POTE
    APUESTAS
2740 IF A$="D" THEN GO SUB 2900: IF CA=0 THEN RETURN:
    REM DOBLAR
2900 REM **** DOBLAR ****
2910 LET DB=1: LET SB=BT: GO SUB 4300: REM IMPRIMIR
    APUESTA
2915 IF CA=1 THEN LET DB=0: RETURN
2920 LET FL=0: LET PL=1: GO SUB 1300:REM REPARTIR
2930 GO SUB 800:REM EVALUAR
2940 LET DB=0: RETURN
4000 REM **** TITULO ETC ****
4010 CLS
4020 LET TX=10: LET TY=3: GO SUB 900: PRINT " ZX
    SPECTRUM"
4030 PRINT TAB(TX+3);"PONTOON"
4040 PRINT TAB (TX+5);"POR"
4050 PRINT TAB(TX-7);"PETE SHAW & STEVE COLWILL"
4060 LET TX=5:LET TY=8: GO SUB 900: PRINT "TU PAQUETE
    ES DE £";SK
4070 LET TX=5: LET TY=12:GO SUB 900: PRINT FLASH
    1;"PULSA CUALQUIER TECLA PARA JUGAR"
4080 LET A$=INKEY$: IF A$="" THEN GO TO 4080
4090 RETURN
4200 REM **** IMPRIMIR PAQUETE ****
4210 PRINT AT 18,15;"TU APUESTA"
4220 PRINT AT 20,15;"PAQUETE RESTANTE"
4230 RETURN
4300 REM **** IMPRIMIR PAQUETE ****
4302 LET CA=0: REM NO SE PUEDE PERMITIR DOBLAR
    BANDERA
4305 IF SK> =50 OR BG=0 THEN GO TO 4310
4306 LET A$="": GO SUB 700: INPUT "TE HAS QUEDADO SIN
    DINERO! VUELVES A JUGAR (S/N)";A$
4307 IF A$="S" THEN RUN
4308 STOP
4310 LET SK=SK-SB: LET BT=BT+SB
4315 IF DB=1 AND SK<0 THEN GO SUB 700: PRINT FLASH
    1;"NO PUEDES PERMITIRTELO!"
4317 IF DB=1 AND SK<0 THEN LET BT=BT-SB: LET
    SK=SK+SB: LET CA=1: RETURN
4320 IF SK<0 THEN LET BT=SK+SB: LET SK=0: GOSUB 700:
    PRINT FLASH 1; "SOLO PUEDES PERMITIRTE £";BT
4340 LET TX=24: LET TY=18: GO SUB 900: PRINT S$(TO 15)
4345 LET TX=24: LET TY=18: GO SUB 900: PRINT "£";BT
4350 LET TX=24: LET TY=20: GO SUB 900:PRINT S$(TO 15)
4355 LET TX=24: LET TY=20: GO SUB 900: PRINT "£";SK
4360 RETURN
```

Commodore 64

```
 72 BG=1:SB=IS:GOSUB 4300:BG=0:REM IMPRIMIR
    APUESTA
 73 RESP$="":GOSUB 700:PRINT"COMPRAS UN NAIPE
    (S/N)";
 74 GET RESP$:IF RESP$="" THEN 74
 75 IF RESP$<>CHR$(13) THEN PRINT RESP$
 77 IF RESP$<>"S" THEN 85
 78 RESP$="":GOSUB 700:INPUT"TU APUESTA (MAX
    £1000)";RESP$
 79 IF VAL(RESP$)>1000 THEN 78
 80 SB=VAL(RESP$):GOSUB 4300:REM IMPRIMIR APUESTA
210 GOSUB 700:PRINT CHR$(156);"GANAS £";CHR$(5);BT
220 SB=-2* BT:BT=-SB:GOSUB 4300:REM IMPRIMIR
    APUESTA
565 SK=10000 : IS=50:REM PAQUETES
610 HP(1)=1:HP(2)=1
625 BT=0:GOSUB 4200:REM IMPRIMIR POTE DE APUESTAS
2740 IF RESP$="D" THEN GOSUB 2900: IF CA=0 THEN
    RETURN
2900 REM **** DOBLAR ****
2910 DB=1:SB=BT:GOSUB 4300:REM IMPRIMIR APUESTA
2915 IF CA=1 THEN DB=0:RETURN
2920 FL=0:PL=1:GOSUB 1300:REM REPARTIR
2930 GOSUB 800:REM EVALUAR
2940 DB=0:RETURN
4000 REM **** TITULO ETC ****
4010 PRINT CHR$(147):REM LIMPIAR PANTALLA
4020 TX=13:TY=3:GOSUB 900:PRINT
    CHR$(156);"COMMODORE 64"
4030 PRINTTAB(TX+3);"PONTOON"
4040 PRINTTAB(TX+5);"POR"
4050 PRINTTAB(TX);"STEVE COLWILL"
4060 TX=9:TY=8:GOSUB 900:PRINT CHR$(28);"TU PAQUETE
    INICIAL ES DE £";SK
4070 TX=10:TY=12:GOSUB 900:PRINT CHR$(5);"PULSA UNA
    TECLA PARA JUGAR"
4080 GET A$:IF A$="" THEN 4080
4090 RETURN
4200 REM **** POTE DE APUESTAS ****
4210 TX=24:TY=18:GOSUB 900:PRINT CHR$(156);"TU
    APUESTA"
4220 TX=24:TY=20:GOSUB 900:PRINT"PAQUETE
    RESTANTE"
4230 RETURN
4300 REM **** IMPRIMIR PAQUETE ****
4302 CA=0:REM NO SE PUEDE PERMITIR DOBLAR BANDERA
4305 IF SK> =50 OR BG=0 THEN 4310
4306 RESP$="":GOSUB 700:PRINT CHR$(28);:INPUT"TE HAS
    QUEDADO SIN DINERO! VUELVES A JUGAR (S/N)";RESP$
4307 IF RESP$="S" THEN RUN
4308 END
4310 SK=SK-SB:BT=BT+SB
4315 IF DB=1 AND SK< 0 THEN GOSUB 700:PRINT
    CHR$(28);"NO PUEDES PERMITIRTELO!"
4317 IF DB=1 AND SK< 0 THEN
    BT=BT-SB:SK=SK+SB:CA=1:RETURN
4320 IF SK< 0 THEN BT=SK+SB:SK=0:GOSUB 700:PRINT
    CHR$(28);"SOLO PUEDES PERMITIRTE £";BT
4340 TX=24:TY=19:GOSUB 900:PRINT LEFT$(SP$,15)
4345 TX=24:TY=19:GOSUB 900:PRINT CHR$(5);"£";BT
4350 TX=24:TY=21:GOSUB 900:PRINT LEFT$(SP$,15)
4355 TX=24:TY=21:GOSUB 900:PRINT CHR$(5);"£";SK
4360 RETURN
```

Programa pionero

"Shadow of the unicorn" (La sombra del unicornio) es el primer programa de juegos que incorpora un dispositivo que impide la acción de los piratas

Combatir la piratería de las cintas se ha convertido en una obsesión para las casas de software. Las empresas reconocen estar perdiendo millones de pesetas al año a causa de los piratas. El obstáculo que impide que la cruzada en contra de los piratas llegue a buen puerto reside en que el soporte más popular para los juegos por ordenador sigue siendo la cinta de cassette, que, lamentablemente para las casas de software, es la más fácil de copiar.

La idea que subyace en el *dongle*, un dispositivo de software que se enchufa en el ordenador, ha estado rondando en la mente de los fabricantes de software durante un tiempo. El *dongle* contiene parte de un programa, y sin el mismo el resto de éste (retenido en cassette o disco) no se ejecutará. Pero se presenta un grave inconveniente: los costos de fabricación de los *dongles* son más altos que los de las cassettes estándares. Esto significa que las casas de software que adopten el sistema deben convencer a los usuarios de que los *dongles* son para proteger mejor los intereses de éstos y no sólo para beneficiar exclusivamente al fabricante.

Mikro-Gen es la primera casa de juegos que adopta el *dongle* como método para impedir la piratería de software. El primer juego que lanza bajo el nuevo formato es *Shadow of the unicorn* (La sombra del unicornio), que está dividido entre un programa retenido en una cinta de cassette normal y un dispositivo *dongle* que se enchufa en la puerta para ampliación del Spectrum.

El *dongle* propiamente dicho consta de una EPROM de 16 Kbytes, una puerta para palanca de mando y un chip decodificador de palanca de mando. La EPROM contiene las facilidades de carga para la cassette y varias de las rutinas gráficas utilizadas en el juego. Tras el encendido, la EPROM se asocia con las direcciones de las zonas de memoria normalmente ocupadas por la ROM de BASIC, permitiendo espacio adicional para el resto del programa.

Aventura de estilo recreativo (con reminiscencias de *Lords of midnight*), el juego se desarrolla en los reinos de Oronfal y Falforn, y la trama consiste en derrotar a las fuerzas del mal que ahora habitan en el territorio. Inicialmente, el jugador controla a tres personajes: Mithulin (el rey de Oronfal), Ulin-Gail (un sátiro) y Avarath (un hechicero). Sin embargo, a lo largo del juego hay otros personajes que el usuario puede controlar una vez que los encuentra. El primero de éstos es Holdin, el capitán de Falforn, que comienza el juego en la misma posición que el hechicero y, por tanto, queda inmediatamente bajo el control del jugador.

Como en otras aventuras recreativas, no sólo hay varios escenarios a explorar, sino también enemigos a vencer y objetos a recoger que pueden ayudar al jugador a medida que el juego avance. En las primeras etapas, los enemigos más comunes son amenazadoras y agresivas criaturas con aspecto de duendes, quienes pueden ser destruidas fácilmente por algunos de los personajes, como el hechicero. Otros personajes sólo pueden eliminarlos con gran dificultad, mientras que el resto, que no dispone de armas, sólo puede huir ante el ataque de los duendes.

Mientras se controla a un personaje determinado, los factores de "energía" y "daño del jugador" se visualizan en forma de barras en la parte superior de la pantalla. Evidentemente, el ataque de un duende reducirá la energía del personaje y, si se permite que el duende alcance al personaje, aumentará el factor de daño. La energía se puede reponer alimentándose con uno de los arbustos distribuidos por el reino. Los daños, sin embargo, por lo general sólo se repararán con el tiempo.

A medida que los personajes se desplazan por el reino, se encuentran con numerosos edificios en los que experimentarán dificultades para introducirse. Intentar moverse en la dirección de la puerta sólo hará que la escena cambie, dejando al personaje al otro lado del edificio. Como cabe esperar, existen formas y medios de entrar en los edificios, pero sólo si el acercamiento se realiza del modo correcto.

Al igual que otros juegos de aventuras, *Shadow of the unicorn* viene acompañado de un libro que ofrece detalles sobre el ambiente del juego y proporciona al jugador variadas pistas acerca de cómo completar la aventura.

Shadow of the unicorn es, sin duda alguna, una jugada audaz por parte de Mikro-Gen. Pero es evidente que la empresa ha hecho un esfuerzo por presentar el juego al usuario de la manera más atractiva posible. Queda por ver si conseguirá redefinir el mercado y asestar el golpe de gracia a los piratas.

Shadow of the unicorn (La sombra del unicornio):
Para el Sinclair Spectrum
Editado por: Mikro-Gen, Unit 15, Western Centre, Bracknell, Berks, Gran Bretaña
Autor: Dale McLoughlin
Palanca de mando: Opcional
Formato: Cassette y EPROM

El West Bridge

Cerca de Olindel

Siga por aquí
Si bien *Shadow of the unicorn* tiene incluidos en su diseño numerosos elementos recreativos, la base del juego es la de una aventura tradicional. El jugador controla a algunos de los personajes, quienes buscan en los diversos escenarios las pistas y ayudas que les permitirán vencer a las fuerzas del mal

Paquetes con valor añadido
El *dongle* de Mikro-Gen, que se suministra con la aventura *Shadow of the unicorn*, le proporciona al jugador 16 Kbytes adicionales de memoria utilizable, mejorando de este modo la cantidad y calidad del juego. Con el Mikro-Plus también se suministra una interface para palanca de mando, puesto que el dispositivo no se puede instalar con la Interface 2

Influencia paterna

Examinaremos la estructura arborescente en la cual se basa el directorio Unix así como algunas de las instrucciones más importantes

En el capítulo anterior vimos cómo cada usuario de un sistema operativo multiusuario como el Unix posee su propia área especial para almacenamiento de disco denominada *directorio*, en donde pueden conservar sus propios programas y datos protegidos de la interferencia de otros usuarios mediante un sistema de contraseñas. Además, también es necesario tener: "directorios públicos" que contengan los programas del sistema disponibles para todos; "directorios del sistema", que conservan, entre otras cosas, información sobre los usuarios y sus directorios; y la capacidad de presentar más de un directorio de modo que los usuarios puedan mantener separados diferentes aspectos de su trabajo.

Esto significa que pueden coexistir una gran cantidad de directorios, teniendo los usuarios diversos niveles de acceso a los mismos. Por ejemplo, todos los usuarios querrán moverse libremente entre sus propios directorios pudiendo, al mismo tiempo, ejecutar programas de los directorios públicos, pero sólo algunos de ellos gozarán de libertad para introducir cambios en estos programas.

Es bueno saber qué otros directorios hay, e incluso qué archivos hay en ellos. Asimismo, debe ser posible trasladar o copiar archivos entre directorios, aunque no necesariamente poder introducir ningún cambio en ellos. El Unix posee una estructura de directorios que permite que los usuarios creen directorios nuevos y supriman aquellos que se han creado. Además, incluye instrucciones para desplazarse a distintos directorios y transferir archivos.

La estructura de directorios que emplea el Unix es la estructura *arborescente*, utilizada en informática para organizar conjuntos de datos con complejas interrelaciones. Cada directorio puede contener varios archivos y subdirectorios, a todos los cuales se les confieren nombres de modo que puedan ser referenciados. Aunque en muchas situaciones el Unix tratará los subdirectorios y los archivos indistintamente, sin embargo lleva el registro de cuál es cuál. Tal subdirectorio es un directorio *descendiente*, o *hijo*, del directorio *padre* que retiene su nombre. Cada directorio, por consiguiente, tendrá un padre y una cantidad no especificada de descendientes.

Si ascendemos en la cadena consultando cada vez al directorio padre, hemos de llegar finalmente a una interrupción en un directorio sin padre. Se dice que éste es el directorio *raíz* y se alude a él mediante el nombre /. A cualquier otro directorio se puede aludir por su *nombre de camino* (*pathname*), que es una lista de directorios comenzando desde la raíz, separando al nombre de cada directorio con /.

El diagrama ilustra un sistema de directorios Unix simplificado. El nombre completo para el directorio del usuario Juan es /usr/dept1/Juan. Un archivo también tiene un nombre completo, que se compone del nombre de camino hacia el directorio que lo contiene, seguido de /nombrearchivo. Normalmente no es necesario utilizar un nombre de camino completo para especificar un archivo dado; con frecuencia no sólo se lo puede abreviar, sino también omitir por completo cuando se está trabajando exclusivamente con el propio directorio del usuario.

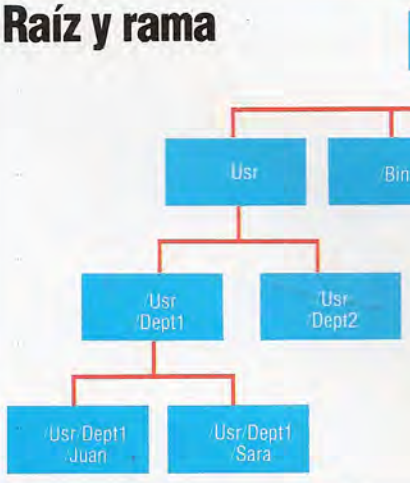
Si el usuario necesita averiguar el nombre de camino completo del directorio en el que se halla, la instrucción pwd (*print working directory*: imprimir el directorio con el cual se está trabajando) se lo proporcionará.

El Unix posee más libertad que casi todos los otros sistemas operativos en cuanto a los nombres que se pueden utilizar para archivos y directorios. Se pueden emplear hasta 14 caracteres, cualesquiera sean, incluyendo espacios; pero es aconsejable evitar ciertos caracteres especiales que el Unix reserva para fines particulares. Éstos son:

\/ " ' *; – ? [] () ~ ! ${}<> :

Cuando se alude a nombres de archivos y directorios, el Unix tiene un sistema de caracteres de más-

Raíz y rama



La estructura de directorios del Unix permite que los usuarios se concentren en sus actividades dentro de subárboles limitados sin ocuparse de otros archivos no pertinentes. Se alude a un archivo mediante su nombre de camino a través del árbol, desde el nudo raíz. De este modo, se alude al directorio del usuario Sara como as/usr/dept1/Sara

Caroline Clayton

Opciones de lista

Junto con el comando ls se puede utilizar lo siguiente:

–a	Lista todas las entradas, incl. arch. del sist.
–c	Lista por tiempo de creación de archivo
–l	Listado completo
–m	Salida en flujo separada por comas
–r	Orden inverso
–s	Dando el tamaño en bloques
–F	Marca los directorios con un / y los programas ejecutables con *
–R	Lista el cont. de todos los subdirectorios

Las opciones se pueden combinar (–a y –r, p. ej., se pueden unir para obtener –ar)

Diálogo con el Unix

```
Berkeley 4.2 Vox/Unix (infsc3)
Type <Ctrl-D> to disconnect

login:com—mcc
Password:
{observe que la contraseña no se reproduce en la pantalla}

You are a Normal user (Class 3)
Jobs: 19 Superiors: 2 Maximum: 21
Last login: Fri Oct 18 11:45:37 on ttyn05

     Welcome to the Information Sciences VAX/UNIX System.

%pwd
/mnt/com/com-mcc          {éste es mi directorio base}
%cd fred                  {pasando a un subdirectorio}
%ls                       {listar todos los archivos de este directorio}
rec.c  receive  rx.p  transmit  tx.p
%ls —a
      rec.c  receive  rx.p  transmit  tx.p

{observe la presencia de dos nombres extras '.', que es este directorio, y '..', que es el
padre inmediato del mismo}

%ls –l
total 42
–rw–rw–r– –   1 com–mcc      502  Sep 17 12:07 rec.c
–rwxr–xr–x    1 com–mcc    18432  Oct 21 11:02 receive
–rw–r– –r– –  1 com–mcc     1068  Oct 18 14:44 rx.p
–rwxr–xr–x    1 com–mcc    19456  Oct 21 11:03 transmit
–rw–r– –r– –  1 com–mcc     1244  Oct 21 11:01 tx.p

{los tres grupos de 'rwx' indican derechos de acceso para l. el propietario de este
directorio, 2. otros del grupo del propietario, y 3. todas las otras personas. r significa
lectura permitida, w significa escritura permitida y x significa ejecución permitida. Un
guión significa que no está permitido el acceso de este tipo}

%ls ?x.p                  {utilizando caracteres de máscara}
rx.p  tx.p
%ls r*
rec.c  receive  rx.p
%ls  *.[pc]
rec.c  rx.p  tx.p

%mkdir mike               {haciendo un directorio nuevo}
%ls –l
total 43
drwxr–xr–x    2 com–mcc      24  Oct 21 11:10 mike
–rw–rw–r– –   1 com–mcc     502  Sep 17 12:07 rec.c
–rwxr–xr–x    1 com–mcc   19456  Oct 21 11:03 transmit
–rw–r– –r– –  1 com–mcc    1244  Oct 21 11:01 tx.p

{observe que un subdirectorio nuevo se indica mediante una 'd'.}

%cd mike                  {cambiar directorio de trabajo a uno nuevo}
%pwd
/mnt/com/com–mcc/fred/mike
%cd ..                    {de vuelta al directorio padre}
–rwxr–xr–x    1 com–mcc   18432  Oct 21 11:02 receive
–rw–r– –r– –  1 com–mcc    1068  Oct 18 14:44 rx.p
%pwd
/mnt/com/com–mcc/fred
%cp rx.p mike             {copiar archivo 'rx.p' en el nuevo directorio}
%mv tx.p mike             {trasladar archivo 'tx.p' al nuevo directorio}
%ls                       {ahora 'tx.p' se ha ido de este directorio}
mike  rec.c  receive  rx.p  transmit
%cd  mike
%ls
rx.p  tx.p
%who                      {averiguar quién más está utilizando el sistema}
root                 console Oct 21 08:07
com–rgd              ttyn00  Oct 21 09:45
ccs–klf              ttyn01  Oct 21 09:45
cs4bc                ttyn04  Oct 21 10:57
com-mcc              ttyn05  Oct 21 10:58
ccs-mdb              ttyn06  Oct 21 10:06
cs4cy                ttyn07  Oct 21 11:06
com–ah               ttyn08  Oct 21 11:00
com–jh1              ttyn10  Oct 21 11:05
cs4bg                ttyn11  Oct 21 11:05
%finger com-mcc           {detalles sobre otro usuario}
Login name: com–mcc       Real name: curtis
Department: Not known
Directory: /mt/com/com–mcc
Shell: /bin/csh
Logged in at 10:58 on Mon Oct 21 on ttyn05 36 secs idle No Plan.
%logout                   {terminar esta sesión}
Host sent disconnect
```

cara para abreviar conjuntos de nombres. Se pueden utilizar los siguientes caracteres de máscara:

- ? se utiliza como comodín de caracteres individuales.
- * se utiliza como comodín para cualquier grupo de caracteres, excluyendo un punto como el primer carácter de un nombre. Esto es para impedir que archivos del sistema tales como .login o .cshrc se borren accidentalmente.
- [] (con una lista o gama de caracteres dentro de los corchetes) sirve como comodín para cualquier carácter individual con uno de los caracteres encerrados entre los corchetes, como [a,e,f], [A-Z], etc.

Para ver cómo funcionan, consideremos algunos de los comandos disponibles para tratamiento de archivos.

La instrucción ls se emplea para listar los archivos y subdirectorios de un directorio, y, al igual que muchas instrucciones Unix, ls puede tomar varias opciones (el conjunto completo se indica en el recuadro). Hay dos instrucciones que le permiten listar el contenido de un archivo: cat, que también se puede utilizar para concatenar archivos, y more, que le permite contemplar sólo una pantalla de información por vez.

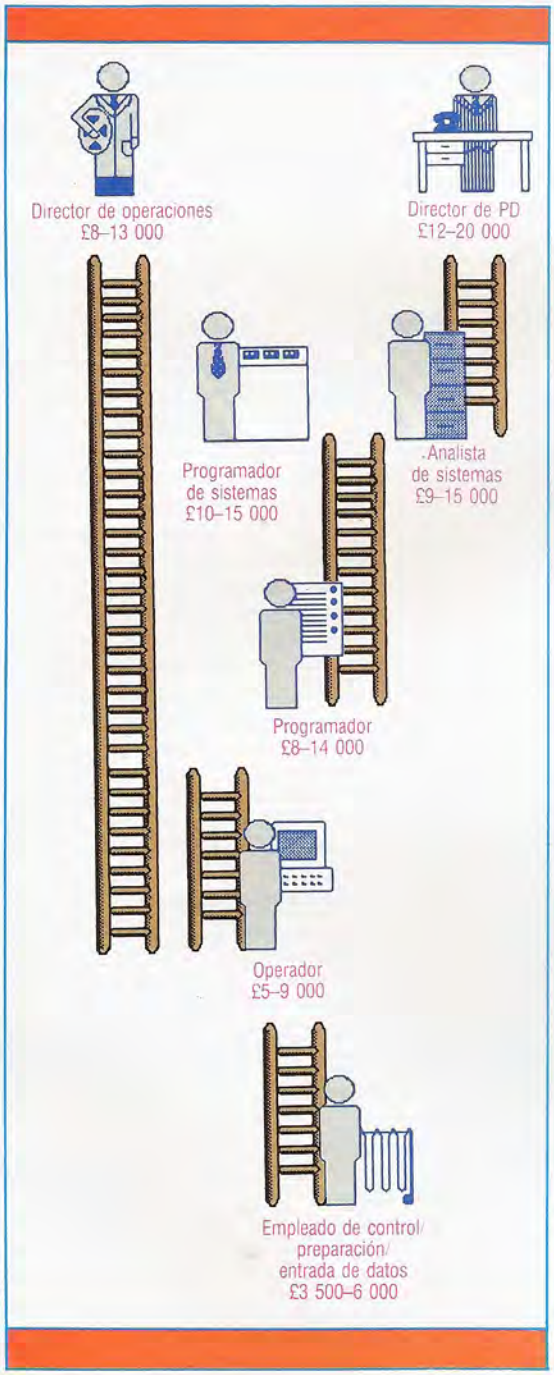
Los usuarios pueden crear y suprimir sus propios subdirectorios a su entera voluntad. La instrucción mkdir creará un nuevo subdirectorio del directorio actual, y rmdir lo eliminará, siempre que no se hayan dejado en él archivos o subdirectorios.

El directorio de trabajo actual se puede cambiar utilizando la instrucción cd. Si el nuevo archivo de trabajo es un subdirectorio del antiguo, se da el nombre del directorio; pero si el nuevo directorio está en algún otro sitio, se ha de dar un nombre de camino completo. El uso de cd sin nombre de directorio siempre le remite de nuevo al directorio base (aquel con el cual usted se conectó originalmente).

Ofrecemos una sesión con el Unix a modo de ejemplo utilizando algunos de estos comandos. Se han insertado observaciones (encerradas entre llaves {}) para explicar lo que está sucediendo; las mismas no se producen en un diálogo real.

Proceso de datos

Proceso de datos (PD) es un sector de la industria informática con una demanda aparentemente inagotable de nuevos trabajadores



Escalera al éxito
El PD ofrece numerosas oportunidades laborales, que van desde el papel ejecutivo del gerente de departamento hasta el desafío intelectual de la programación de sistemas. Nuestro diagrama refleja los distintos caminos para hacer una carrera, junto con una indicación de la escala salarial vigente en Gran Bretaña para cada empleo

Caroline Clayton

Al ser Gran Bretaña uno de los países más avanzados en el campo de la informática, lo que allí suceda en este ámbito tiende a tener resonancia, en mayor o menor medida, en los restantes países europeos. Por este motivo es interesante dar una mirada a la actualidad educacional y laboral en este país en lo que respecta a la informática.

Cada semana, en las revistas *Computing* y *Computing Weekly* aparecen más de 100 páginas de anuncios de trabajo ofreciendo puestos en proceso de datos con salarios que parten de un mínimo de £10 000 a £15 000 (una libra esterlina equivale a unas 215 pts.). Veamos las oportunidades que ofrece este campo, cuáles son los caminos que conducen a tales carreras y cómo puede introducirse el novato en la industria.

Primero hemos de preguntarnos qué es realmente el proceso de datos. En esencia, el PD supone el tratamiento, recuperación y almacenamiento de información relacionada con una aplicación. Gran parte del PD gira alrededor de registros de rutina: llevar las cuentas o la nómina de una empresa, por ejemplo.

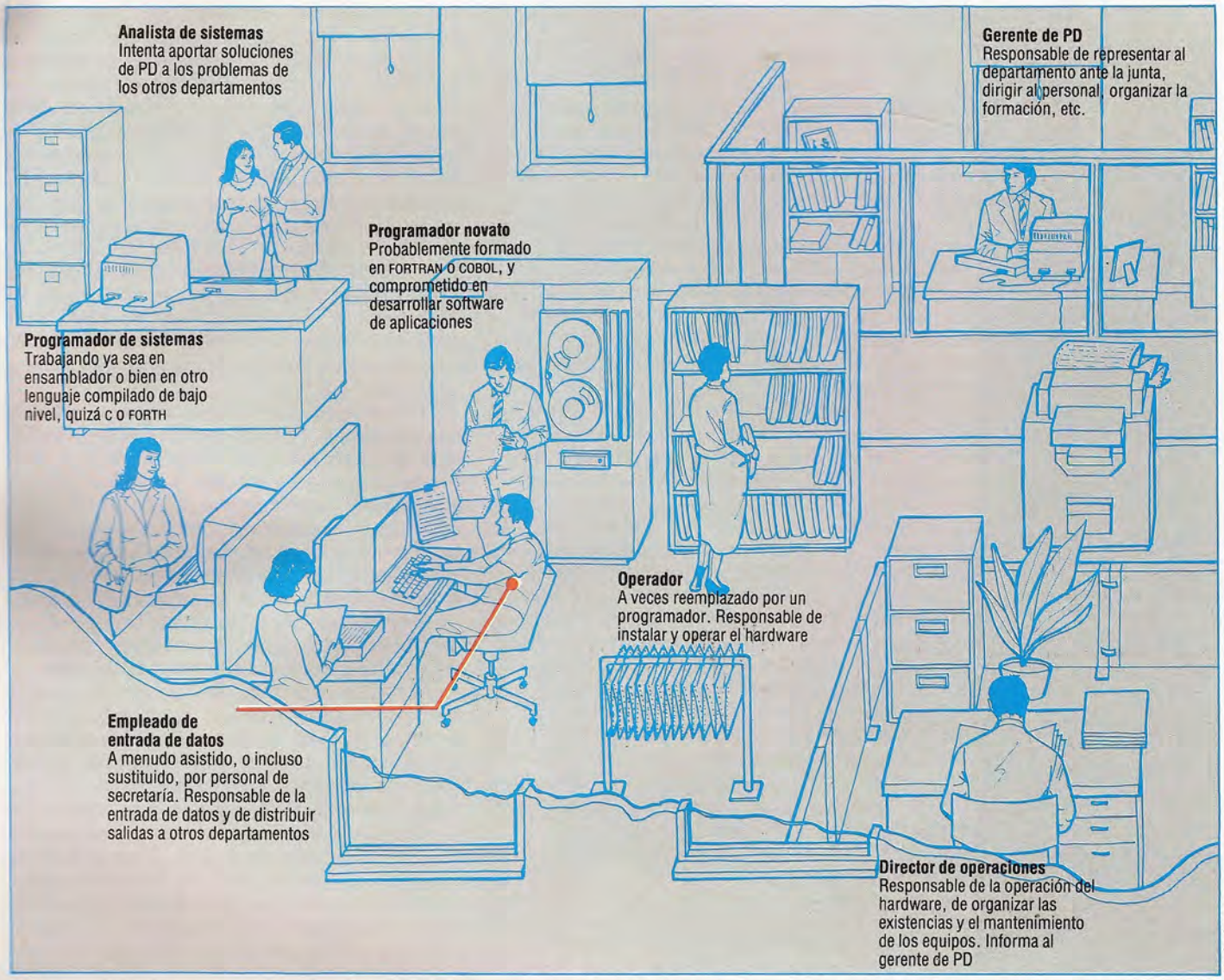
Una carrera en PD no significa que automáticamente el postulante tendrá acceso al último equipo en materia de hardware de microordenador. En la mayoría de los sitios de PD, el micro se sigue viendo como un juguete y el 90% del trabajo se lleva a cabo en miniordenadores u ordenadores centrales, suministrados por fabricantes como IBM, ICL y Burroughs.

El dibujo (izquierda) muestra las principales áreas de empleo y los "peldaños" en el sector de PD. Las mismas, no obstante, no son categorías rígidas ni expeditas. En algunos sitios, por ejemplo, una persona combinará el papel de analista y programador, en otros el de operador y programador. En general, sin embargo, éstas son las categorías laborales en las que se podría incluir el 90% de las personas de la industria.

En la parte inferior de la escalera están los empleados de *preparación de datos*, *entrada de datos* o *control de datos*. Estos trabajos implican un escaso conocimiento real, si no ninguno en absoluto, sobre informática, y tal como revelan los niveles de salarios, están en manos de empleados no cualificados. De hecho, el trabajo supone "digitar" datos en una máquina a través del teclado almacenándolos sobre un disco o cinta. También implica preparar los impresos (cortarlos, etc.) y enviarlos. En los lugares más pequeños, este aspecto del trabajo lo puede realizar el personal de secretaría con la ayuda de operadores y programadores. Es discutible si un empleado de preparación de datos se puede realmente considerar como miembro del personal de PD. En muchos lugares, la preparación de datos se considera como una labor no cualificada y sin perspectivas, que no abre las puertas a un puesto como programador u operador.

El trabajo del operador

El *operador*, la siguiente etapa hacia arriba partiendo del empleado de preparación, es responsable de operar el ordenador propiamente dicho. En consecuencia, la labor del operador se puede comparar con la del conductor de un automóvil, mientras que se puede considerar al programador como un oficial de navegación. El operador es responsable de



Kevin Jones

Estaciones de acción
La ilustración muestra el interior de un típico departamento de PD de una gran compañía. La posición de tales departamentos se ha erosionado de manera significativa en los últimos años a medida que más jefes de departamentos han ido ganando acceso a sus propios micros de escritorio. Sin embargo, el PD sigue siendo una de las mayores fuentes de empleo y ofrece importantes oportunidades para hacer una carrera

inicializar físicamente las tareas en las máquinas, cargando cintas y discos, cargando papel en las impresoras y poniendo las máquinas en funcionamiento y desconectándolas.

Tradicionalmente, la transferencia desde el lado del operador al de la programación era relativamente sencilla, pero esta tradición está desapareciendo rápidamente a consecuencia de la "descualificación" del papel del operador y el creciente número de programadores formados y con titulación. De hecho, en el ámbito de los operadores hay muchísimas personas tratando de apartarse del campo. La facilidad con que se pueda pasar de operador al campo de la programación depende mucho de la actitud de la gerencia. Si bien no se debe descartar de buenas a primeras ser operador durante un breve período, es importante evaluar las posibilidades de promoción antes de aceptar el puesto.

La mayoría de las personas que parecen tener una carrera a largo plazo en PD intentarán ascender por la escalera hasta el siguiente peldaño: *el programador*. En comparación con la programación en un micro, la programación en un entorno PD tiende, sin embargo, a ser una actividad mucho más especializada.

Como programador novato es probable que usted aprenda COBOL o bien FORTRAN (o posiblemente PASCAL), lo que le permitiría escribir software de aplicaciones. Sin embargo, en muchos lugares, y según el hardware utilizado, hay otro estrato de programación: el del programador de sistemas. El software de sistemas se sitúa entre el sistema operativo del ordenador y el software de aplicaciones que se esté ejecutando, y casi siempre se escribe en un lenguaje de bajo nivel específico de la máquina, casi con toda seguridad ensamblador. No obstante, el C está adquiriendo creciente difusión en este campo debido a su velocidad y portabilidad.

Muchos programadores seguirán ascendiendo hasta convertirse en analistas de sistemas, el más reciente de los principales papeles laborales en PD. Esencialmente, el analista de sistemas se ocupa de decidir con los usuarios la clase de información que requieren o que podría serles útil. La tarea requiere capacidad para hablar con gerentes que quizá no posean conocimientos sobre informática, y para explicarles lo que es y no es posible. Muchas personas desconfían y temen a los ordenadores, viéndolos como una amenaza para su futuro laboral. El analista de sistemas ha de ser capaz de vencer esta desconfianza y hacerse una idea exacta de lo que supone una tarea determinada. Por este motivo, el análisis requiere gran aptitud para la comunicación (y mucha paciencia y diplomacia).

Rutas diferentes

Como programador, usted puede elegir entre dos caminos alternativos para hacer una carrera: convertirse en especialista de software o en analista de sistemas. La ruta que elija dependerá mucho de su carácter y su actitud ante la vida. Si es un solitario que no necesita mucho contacto humano en su vida laboral, entonces se sentirá feliz de permanecer en la programación. De hecho, si usted encuentra el área adecuada en la cual especializarse, la programación puede ser sumamente lucrativa y muy gratificante desde el punto de vista intelectual. Si, por el contrario, le agrada trabajar con otras personas y no se resigna a la idea de contemplar una pantalla por el resto de sus días, el análisis de sistemas podría ser la respuesta.

El gerente de PD lleva a cabo tres funciones. Con la ayuda del analista de sistemas y los programadores senior, el gerente elige y evalúa el hardware y software necesarios para la instalación. En segundo lugar, es responsable de dirigir a quienes trabajan en el departamento de PD y asegurarse de que reciban el adiestramiento adecuado. Por último, representa al departamento ante el mundo exterior, informando ya sea a la junta o al director de servicios de administración.

Hay cuatro formas básicas de convertirse en programador: usted puede progresar desde operador (no es una ruta que aconsejamos especialmente), puede introducirse en el campo a raíz de un curso de especialización, como graduado en informática, o como un graduado en prácticas con una titulación que no sea especialmente relevante (o como alguien que posea vasta experiencia comercial pero ninguna experiencia previa en informática).

Es un hecho incuestionable que la programación (y el PD en general) se está convirtiendo rápidamente en una profesión para graduados. Los graduados en informática pueden realmente seleccionar y elegir; incluso los graduados en inglés o en alguna ciencia social pueden ser capaces de obtener un adiestramiento siempre que superen la prueba fundamental. Los graduados en matemáticas o ciencias con alguna experiencia en un ordenador central de una universidad tienen ante sí un brillante comienzo.

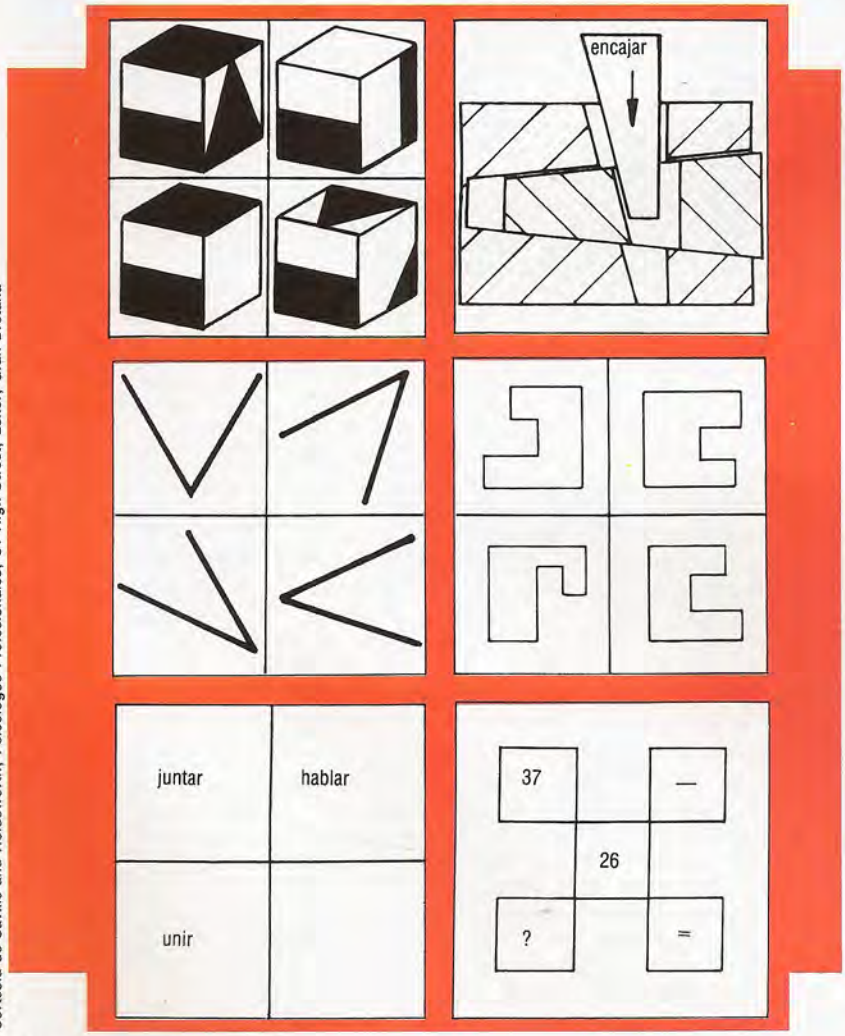
El PD no es un coto cerrado a quienes sean un poco mayores y quienes posean una experiencia útil en la industria. La firma Ford, y otras numerosas empresas automovilísticas, han adiestrado en programación a directores de líneas con un buen conocimiento de la industria del motor. Numerosas empresas agrupan a programadores experimentados con gerentes con la esperanza de obtener una mejor identificación de las necesidades del usuario.

El progreso en una carrera de PD depende en gran medida del nivel de adiestramiento que usted pueda recibir. El adiestramiento en alguna rama secreta del software de sistemas, por ejemplo, podría proporcionarle aptitudes que en Estados Unidos representan la posibilidad de acceder a un puesto con una remuneración de $4 000 o $7 000 al mes (un dólar norteamericano equivale a unas 150 pts). Pruebe de averiguar cuáles son los antecedentes de su potencial patrón respecto al adiestramiento antes de compromoterse con él. Algunas compañías le ofrecen a su personal una formación profesional mínima, o esperan que ellos aprendan en el trabajo, en la creencia generalizada de que un empleado bien adiestrado permanecerá poco tiempo. En general, cuanto mayor sea la empresa, mejor será el adiestramiento.

Aparte de la formación, el otro factor prioritario que incide en los niveles de salario es la experiencia con el ordenador. En general, es bastante difícil saltar de la máquina de un fabricante a la de otro: si usted ha estado trabajando con máquinas ICL es difícil conseguir empleo en un punto IBM. Esto es reflejo del hecho de que cada fabricante ha desarrollado software de sistemas diferentes y que éstos difieren considerablemente entre sí. Sería arriesgado obtener un empleo para trabajar con una máquina de cierta antigüedad producida por un fabricante de poco éxito, puesto que obviamente en otros lugares no habría mayor demanda para las aptitudes que usted pudiera desarrollar. Sería preferible, entonces, trabajar en un punto IBM (más del 80% de todos los ordenadores centrales del mundo son IBM).

Pero Well y Burroughs también cuentan con grandes bases de usuarios. Posiblemente la firma ICL esté en peligro de perder su dominio en el mercado británico de ordenadores centrales y no sería una buena decisión si usted tuviera intención de trabajar en el extranjero. Ha de pensárselo detenidamente antes de iniciar una carrera que suponga especialización en hardware Univac, NAS o Kienzle, por ejemplo.

Más allá del coeficiente intelectual
Con frecuencia los jefes de personal se muestran reacios a confiar exclusivamente en la entrevista con el interesado para la selección de postulantes a un empleo, y exigen que éstos realicen uno o más tests para determinar su idoneidad. Estos tests, por lo general diseñados por psicólogos profesionales, en cierto sentido son similares a los tests de coeficiente intelectual, pero van más lejos en su evaluación de aptitudes relacionadas con el puesto de trabajo. La ilustración muestra una representación simplificada de algunos de los problemas con que se puede encontrar un postulante a aprendiz técnico. De izquierda a derecha y de arriba abajo: razonamiento espacial, comprensión mecánica, estimación visual, conocimiento espacial, razonamiento verbal y cálculo numérico



Cortesía de Saville and Holdsworth, Psicólogos Profesionales, 81 High Street, Esher, Gran Bretaña

Abrir un camino

En MS-DOS se pueden organizar lógicamente software y datos colocando todo tipo de archivos en un árbol jerárquico de subdirectorios

Las utilidades del Unix pwd (*print working directory*: imprimir directorio de trabajo) y tree (visualizar la estructura del directorio) no siempre las proporcionan los OEM que suministran MS-DOS, pero hay otras alternativas disponibles. Incluyendo los caracteres $p en el comando prompt, se visualizará el nombre de camino completo (desde la raíz de la unidad actual) como parte del aviso DOS después de que termine la ejecución de cada comando.

Supongamos, no obstante, que estamos trabajando en un subdirectorio llamado, pongamos por caso, TT (de tratamiento de textos) y queremos formatear un disco y hacer una copia de un nuevo documento. Dar la instrucción format b: no sirve de nada porque la utilidad transitoria FORMAT.EXE se halla en un subdirectorio llamado sistema. Debemos decir \sistema\format b:, utilizando el nombre de camino completo. Si otro programa, llamado WCOUNT.EXE (*word count*: contar palabras) estuviera en la unidad C en un subdirectorio varios niveles más abajo, nuevamente tendríamos que dar el nombre de camino completo para que se encontrara y ejecutara. De modo que contar la cantidad de palabras de un documento llamado informe17.doc podría suponer tener que teclear:

```
C:\texto\herramientas\wcount informe17.doc
```

Afortunadamente, hay una forma muy simple de sortear este problema. El comando residente path visualizará o creará una lista de caminos por defecto en los que el MS-DOS buscará siempre que un comando no reconocido no esté situado en el directorio actual. Solo, path visualizará el mensaje No path antes de que se hayan especificado caminos por defecto. Si todos los programas y utilidades de más frecuente uso estuvieran incluidos en los directorios que acabamos de mencionar, podríamos impartir la instrucción:

```
path=A:\sistema;C:\texto\herramientas
```

Ahora, cualquiera sea el disco o directorio actual, el procesador de instrucciones encontrará cualquier programa en el directorio actual o en alguno de los otros dos especificados. La búsqueda se dirige por el mismo orden en el que se especifican los caminos; de modo que, por ejemplo, si se hubiera de reproducir el nombre de un programa, se ejecutaría la primera pareja.

Una palabra de atención: *no* se deben insertar espacios en la lista de argumentos, porque son considerados como terminadores. El comando:

```
path C:\sistema;A:\milib;C:\texto\herramientas
```

no conseguiría, por tanto, encontrar nada que no estuviera ya sea en el directorio actual o bien en la unidad C del directorio del sistema.

Las versiones de MS-DOS inspiradas en el Unix (de la versión 2.0 en adelante) poseen la capacidad de *redirigir* la E/S desde y hacia tanto dispositivos del sistema como archivos de disco. Los dispositivos estándares del sistema tales como CON (la consola) y PRN (la impresora) se manejan exactamente del mismo modo que como se trataría un archivo de texto en disco. Utilizando la instrucción residente type, normalmente se visualizaría el contenido de un archivo de texto en la pantalla, pero la salida se podría redirigir, con el símbolo *chevron* (>), a cualquier dispositivo de archivo que quisiéramos:

```
type informe17.doc>prn
```

iniciaría la impresión y produciría una salida impresa del archivo. El comando:

Más comandos residentes

He aquí otro listado de algunos de los comandos residentes del MS-DOS. Todos los comandos que vienen a continuación están incorporados en COMMAND.COM, el equivalente del MS-DOS al intérprete de línea de comando del CP/M, o CCP.

	Función
cls	Limpiar la pantalla
prompt	Cambiar el aviso del sistema
pwd	Imprimir directorio de trabajo
re (o rmdir)	Suprimir un direct. (debe estar vacío)
ver	Imprimir el núm. de versión MS-DOS
vol	Imprimir el ID de volumen

No todos los sistemas, sin embargo, tendrán todos estos comandos. Por ejemplo, ped (tomado del Unix) no es esencial, dado que se puede utilizar el comando prompt para visualizar el camino actual, como en:

```
prompt $p
```

Aludiendo la "p", en este caso, a *path* (camino). Otros caracteres que vayan tras el símbolo $ en un argumento prompt pueden tener significados especiales, y algunos de los mismos son:

d	Imprimir la fecha
t	Imprimir la hora
_	Enviar un CR/LF para crear una nueva línea
e	Enviar un carácter escape

La secuencia de escape, tal como la utiliza un terminal ANSI estándar, se puede utilizar para cambiar los atributos de la VDU, de modo que:

```
prompt $t on $d $_$e[7m $p $e[m
```

imprimirá la hora y fecha en una línea, y el camino actual en video invertido en la siguiente. De manera que:

```
15:42:36 on 17-11-85
A:\SYSTEM\UTILS
```

```
dir a:> c:nov22.dir
```

listaría el contenido del directorio, no a la VDU como siempre, sino a un archivo de la unidad C llamado nov22.dir. Utilizando de esta manera la redirección de salida, se puede lograr que cualquier programa o utilidad envíe su salida a cualquier archivo o dispositivo del sistema.

El doble *chevron*, >>, produce la misma redirección de salida pero sin sobreescribir el contenido previo de un archivo. De modo que después de:

```
dir b:>> c:nov22.dir
```

el archivo nov22.dir contendrá una lista de los archivos y subdirectorios de los directorios actuales de las unidades A y B.

Utilizado de forma aislada esto es apenas sólo conveniente, pero en combinación con unas pocas "herramientas de software" el empleo de la redirección de E/S puede casi transformar el MS-DOS en un eficaz lenguaje de programación interactivo.

Algunas de las herramientas de software más eficaces son programas simples que introducen algún cambio en los datos leídos a partir de la entrada estándar antes de pasárselos a la salida estándar. Éstos se denominan *filtros*, por la evidente analogía con la electrónica. Es muy fácil escribir filtros, incluso para los programadores recién iniciados, con la salvedad de que han de estar escritos en un lenguaje compilado de modo que el código máquina producido sea "independiente" y se pueda ejecutar tal como lo hace cualquier utilidad del sistema sin que esté presente un intérprete o el cód. fuente.

El Unix y las versiones 2/3 de MS-DOS poseen la capacidad de conectar archivos con cualquiera de las rutinas de manipulación de caracteres para los dispositivos de entrada *y* salida estándares del sistema. Para redirigir la entrada se utiliza <. En este caso, se le puede indicar a todo programa que espera datos de entrada desde el teclado que los obtenga de un archivo. Esto incluye instrucciones tanto transitorias como residentes. Por ejemplo:

```
fecha<fecha.hoy
```

aceptará una representación en serie de una fecha de un archivo llamado fecha.hoy, apareciendo en la pantalla, como siempre, el mensaje de salida normal producido por la instrucción. Esto puede ser muy útil si sólo se dispone de un reloj de software, que se restablece cada vez que se vuelve a cargar el sistema. La inclusión de la línea anterior en el archivo autoexec.bat evita el tener que volver a digitar continuamente la fecha.

Hay algunos filtros útiles que se suministran como utilidades transitorias del MS-DOS. Uno de ellos (MORE.COM) toma cualquier texto entrado y simplemente lo copia en la salida estándar. No obstante, cada 23 líneas MORE visualiza el mensaje:

```
--More--
```

y aguarda a que se pulse una tecla, permitiendo de este modo contemplar una pantalla de texto cada vez al ritmo deseado. Por ejemplo:

```
more<informe.doc
```

Naturalmente, la redirección de salida no es especialmente útil con MORE.

Una de las utilidades transitorias del MS-DOS más útiles es SORT.EXE. Éste es un filtro que lee

líneas de texto y, antes de copiarlas sin modificación en la salida estándar, las clasifica por orden alfabético. Si deseáramos un listado de directorio clasificado de todos los archivos de texto, por ejemplo, se podría conseguir con:

```
dir*.txt> temp.dir
sort < temp.dir
del temp.dir
```

El MS-DOS, al igual que el Unix, permite llevar a cabo estos procesos en una operación mediante un *tubo* o *tubería*. Se utiliza el símbolo | para indicar que la salida de un proceso se debe redirigir a un canal o corriente que ha de actuar como la fuente de entrada para un ulterior proceso. En el caso que

Planta de filtración

Un "filtro" simple puede ser un programa que simplemente tome una entrada estándar y altere, añada o suprima ciertos caracteres antes de pasar el resultado (filtrado) a la salida estándar. Consideremos un filtro simple (llamado pipz) para impedir que lleguen a la salida caracteres con códigos ASCII de 128 en adelante. Los archivos de texto producidos con el *WordStar* y con ciertos otros programas a menudo contienen tales caracteres, puesto que el bit más significativo se puede establecer con fines de formateado. Se puede realizar fácilmente una copia "en limpio" de un archivo de texto que originalmente contuviera tales códigos, mediante una instrucción para redirigir los canales tanto de entrada como de salida:

```
pipz<archmalo.txt>archbueno.txt
```

Este sencillo programa (en versión en PASCAL) "pliega" todos los valores de caracteres a la gama de códigos ASCII normal (hasta 127) y, por lo tanto, simula el empleo de la opción [z] con la utilidad pip del CP/M, poniendo a cero el bit *high* de

Plano de distribución
La estructura de archivos jerárquica del MS-DOS se puede considerar como una serie de habitaciones conectadas entre sí. La entrada a ciertas habitaciones sólo se puede efectuar a través de otras. Las habitaciones "muertas" representan archivos reales de programas, en vez de directorios (en color azul). Supongamos que estamos trabajando dentro del subdirectorio llamado "milib" (en color rojo) en un archivo de texto llamado "informe17.doc" Los programas de utilidades tales como contadores de palabras (wcount) y los formateadores de disco residen en otras "habitaciones". Para no tener que especificar el camino completo desde la raíz cada vez que queramos utilizar una utilidad, el MS-DOS proporciona un comando path que define caminos por defecto. Para disfrutar del acceso directo a los directorios de utilidades podríamos impartir el comando: path\sistema;\texto\herramientas, que especificaría dos caminos alternativos para la utilidad requerida

cada byte de carácter. Esto es particularmente útil porque la instrucción copy del MS-DOS no dispone de esta opción. El archivo se podría listar, asimismo, directamente a la impresora mediante:

```
pipz<archmalo.txt>prn
```

Un ejemplo de filtro en PASCAL:

```
PROGRAM PipZ (input,output);
{plegar cars a gama ASCII 0..127}
VAR
  c:char;
BEGIN
  WHILE NOT EoF (input) DO
    BEGIN
      WHILE NOT EoLn (input) DO
        BEGIN
          read (input,c);
          write (output,
            chr (ord(c) MOD 128))
        END;

      ReadLn (input);
      WriteLn (output)
    END
END.
```

acabamos de ilustrar, por lo tanto, sólo necesitamos decir:

```
dir*.txt|sort
```

El MS-DOS manipula de forma invisible todo archivo temporal que se requiera, y lo redirige automáticamente cuando termina una tubería.

La instrucción sort puede tomar opciones, incluyendo /r (para clasificar por orden inverso) y /+n (donde n es cualquier número), indicando la posición de carácter de cada línea en la cual comenzarían las comparaciones para clasificación. Los detalles del directorio del MS-DOS incluyen los tamaños de los archivos expresados en bytes (empezando por la posición del 15.º carácter) y también la hora y fecha de creación. Podríamos, en consecuencia, producir un listado en salida impresa del directorio clasificado con los archivos más grandes listados primero:

```
dir|sort/r/+ 15 >prn
```

Otro potente filtro del DOS (FIND.EXE) se puede utilizar, por ejemplo, para hallar series en cualquier archivo de texto dado, simplemente mediante:

```
find"sort "C:\artículos\MSDOS4.txt
```

En la salida se lista cada línea del archivo que contenga la serie "sort " (una sola vez, incluso cuando en la misma línea apareciera en varias ocasiones). Observe que el espacio que sigue a "sort significa que no se hallarán sorted ni sorting. Ni tampoco se localizaría Sort, porque find es sensible a las mayúsculas o minúsculas.

Esta deficiencia se puede superar fácilmente escribiendo nuestro propio filtro simple (LOWER.EXE) para convertir todas las entradas a minúsculas. Entubándola "en serie" antes de otras instrucciones entubadas se suprime la sensibilidad a los tipos de letra de todas las otras herramientas:

```
lower<meeting.doc|find"metal"|more
```

listaría todas las líneas del documento (meeting.doc) que hicieran referencia a metal, metálico o metalurgia (con o sin mayúsculas) y haría una pausa tras visualizar cada 23 de tales líneas.

La potencia de las tuberías y la redirección queda bien ilustrada con instrucciones tales como:

```
dir*.pas|find"-08-85"|sort>ptn
```

que imprime todos los archivos fuente en PASCAL (.pas) creados en agosto por orden alfabético. La salida de find se podría listar con los números de línea antepuestos (la opción /n) o se podría suprimir la visualización de líneas y obtener con /c una cuenta total del número de ocurrencias. La opción /v halla todas las líneas que *no* contengan la serie.

Supongamos que tenemos un archivo de texto (compañía.dat) que contiene algunos nombres y otros detalles, incluyendo salarios, que comienza en la columna 25:

```
lower<compañía.dat|find"smith"/v|sort/+25|more
```

produciría una lista de todos los que *no* se llamaran Smith o Smithers, etc., clasificada por orden descendente de salarios.

Mediante el empleo de tubos y redirección hemos creado un nuevo programa en la línea de comando construyéndolo en el acto a partir de simples herramientas de software de filtro.

Punteros

Nos corresponde examinar el procedimiento de declaración y el empleo de los punteros

Diseñado como sustituto del lenguaje ensamblador, el C posee una gama de facilidades inexistentes en otros lenguajes de alto nivel. Una de éstas es la facilidad para aludir directamente a direcciones máquina. El PASCAL posee tipos *puntero*, pero la diferencia entre éstos y las direcciones máquina no está clara en absoluto. En C, el tipo puntero alude a direcciones reales, si bien hay que recordar que en una máquina multitarea, particularmente aquella que utilice memoria virtual, el verdadero conocimiento de la dirección numérica de una variable normalmente no es de especial utilidad.

No obstante, el C posee la capacidad de aludir a direcciones absolutas y, por tanto, se puede utilizar para activar dispositivos de hardware directamente. Toda variable, del tipo que sea, tiene una dirección que se puede asignar a un puntero. A menudo es más conveniente, en especial con las series (matrices de caracteres), emplear aritmética de puntero para acceder a elementos de la serie en lugar de utilizar los habituales subíndices de matriz.

El carácter * se utiliza para declarar una variable puntero, de modo que:

```
int*p;
```

declararía una variable puntero p. Observe la declaración int: los punteros sólo pueden apuntar a un tipo de variable determinado (el tipo "básico"), de modo que si quisiéramos que un puntero accediera a variables de tipo char, deberíamos entrar:

```
char*p;
```

y así sucesivamente. El carácter *, cuando es utilizado de este modo, significa simplemente "crear un puntero", de modo que en esta etapa el puntero propiamente dicho no apunta hacia ningún lugar.

Para inicializar un puntero en un valor requerido, usamos el carácter &, el cual, colocado delante de una variable, tiene el efecto de devolver la dirección de ésta en vez de su valor. De modo que:

```
p=&i;
```

establecería p como un puntero de i. En este punto, usted debe observar que el carácter * también se puede utilizar para indicar el valor al que apunta una variable puntero.

En este caso:

```
i = *p;
```

almacenaría el *valor* apuntado por p en i, mientras que:

```
i = p;
```

almacenaría la *dirección* apuntada por p. Usted, por supuesto, necesitará un *cast* para evitar mensajes de error del compilador al manipular direcciones absolutas, como en:

```
p = (int*) 1000;
```

Los operadores de incremento y decremento ++ y -- se pueden utilizar con variables puntero, y aumentan o disminuyen en la cantidad adecuada para el tipo de variable que se esté apuntando. En un sistema que utilice enteros de cuatro bytes, por lo tanto, si p es un puntero al primer elemento de una matriz de enteros, p++ aumentará la dirección de p en cuatro para apuntar al siguiente elemento. Lo mismo sucedería si utilizáramos p=p+1: sumaría el tamaño del elemento de dato adecuado en lugar de apenas 1. Esto pone de relieve el hecho de que los punteros no son lo mismo que enteros, aun cuando los valores en ellos almacenados sean verdaderamente números enteros.

Los punteros se pueden pasar como parámetros en llamadas a funciones. Ello ofrece la facilidad de pasar por referencia, es decir, el valor del puntero se le pasará a una variable local de la función, pero este valor seguirá apuntando al mismo elemento que apuntaba en el programa de llamada. En consecuencia, los cambios que se introduzcan en el valor almacenado continuarán allí cuando se devuelva el control.

Existe una correspondencia muy íntima entre punteros y matrices. Cuando se declara una matriz, el nombre de la matriz sin subindexar es, en realidad, un puntero al primer elemento de la matriz. Dicho en otras palabras, dado int a [100], *p, luego p=a; y p=&a[0]; son totalmente equivalentes. Esto significa que con frecuencia el acceso a elementos de la matriz (o de submatrices) se puede realizar más eficazmente mediante aritmética de punteros que mediante subíndices.

Utilización de series

Las series son, simplemente, matrices unidimensionales de variables de tipo char. Dado que su uso es fundamental para muchas aplicaciones, el C, no obstante, proporciona unas pocas facilidades especiales y funciones de biblioteca para llevar a cabo los tipos normales de proceso de series. En C, una serie se compone de una matriz de char en la que los caracteres reales que componen la serie van seguidos inmediatamente por el carácter nulo \0 (ASCII cero). Esto podría ocurrir en cualquier punto de la matriz, de modo que tenemos al menos la ilusión de series dinámicas de longitudes variables aun cuando las matrices subyacentes hayan de ser de longitud fija. Recuerde que el carácter \0 ocupará una posición de carácter en la matriz, de modo que ésta debe ser definida con al menos un elemento más que la cantidad máxima de caracteres que vaya a albergar.

Se pueden utilizar constantes en serie, si están encerradas entre comillas dobles, y se las puede asignar a una matriz de char de longitud adecuada; el terminador \0 se añadirá automáticamente. La variable a la cual se asignan puede ser ya sea una matriz o bien un puntero a char:

```
char *s;
*s="abc";
```

hace que los cuatro caracteres a, b, c y \0 se alma-

Funciones de manipulación de series

strcmp (s1,s2)—s1 y s2 son punteros a char (series), se devuelve un valor entero que es menor que cero si s1<s2, cero si s1=s2 y mayor que cero si s1>s2
strncmp(s1,s2,n)—Similar a strcmp excepto que, como máximo, se comparan n caracteres
strlen(s)—Donde s es un puntero a char, devuelve la longitud (en enteros) de la serie
index(s,c)—Donde s es un puntero a char y c es un char, devuelve un puntero a la primera aparición de c en s, o NULL si no aparece ninguna vez
rindex(s,c)—Similar a index, excepto que la búsqueda se lleva a cabo desde el carácter situado más a la derecha
strcat(s1,s2)—Añade una copia de s2 al final de s1, devolviendo s1. Se da por sentado que s1 es suficientemente larga como para contener todos los caracteres
strncat(s1,s2)—Añade como máximo n caracteres no nulos al final de s1, devolviendo s1
strcpy(s1,s2)—Copia el contenido de s2, hasta la '\', en s1. Se da por sentado que s1 es suficientemente larga. El cont. previo de s1 se pierde. Se devuelve el valor de s1.
strncpy(s1,s2,n)—Copia como máximo n caracteres de s2 en s1, añadiendo una '\' a menos que n> = la longitud de s2

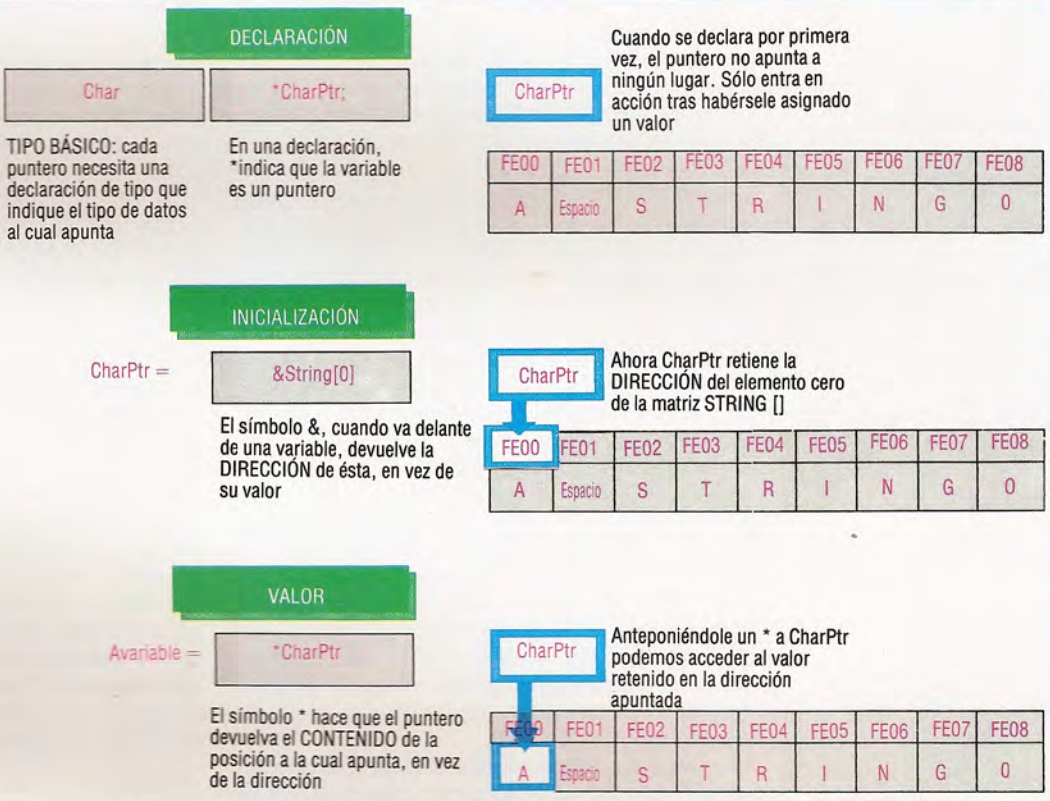
Burbujeante

```
burbuja(a,n)
int a [],ponerenorden();
/* a es una matriz de n enteros a clasificar*/
{
int i,j;
   for(i=0;i<n – 1;++i)
      for (j=n – 1;i<j; – –j)
         ponerenorden(&a[j – 1],&a[j]);
/*paso de las direcciones de los elementos
de la matriz como parámetros*/
}
ponerenorden(x,y)
int*x,*y,swap();
/*x e y son punteros a enteros*/
{
      if(*x>*y)
         swap(x,y);
}
swap(x,y)
int*x,*y;
{
      int temp;
      temp= *p;
      *p=*q;
      *q=temp;
/*observe que las direcciones de los dos enteros se
pasan por valor pero la función puede aludir a los
valores reales almacenados allí aun cuando estén
estrictamente fuera del ámbito*/
}
```

Apuntando por números
Nuestro listado muestra la estrecha relación existente entre los punteros y las matrices en c, implementando el tradicional algoritmo de clasificación por el método de la burbuja utilizando una matriz de enteros. Con frecuencia los punteros del c pueden ser más eficaces que los subíndices para acceder a los elementos de las matrices

En tres actos

Aquí vemos tres etapas en el uso de un puntero teórico CharPtr (utilizado para hacer referencia a distintos elementos de la matriz en serie STRING []). Observe que tras la inicialización, CharPtr devuelve una dirección, mientras que *CharPtr devuelve el valor retenido en esa dirección. Los punteros se pueden incrementar y reducir para apuntar a distintos elementos de una matriz: el valor del puntero se ajustará de acuerdo al "tipo de dato básico" dado en la sentencia de declaración. Por ejemplo, un tipo básico de char dará por resultado ajustes de pasos de un solo byte, e int (en la mayoría de los sistemas) producirá ajustes de dos bytes



cenen en cuatro posiciones consecutivas empezando en la dirección actual de s, que parte apuntando al carácter a. Después de ++s, s apuntaría entonces al b, y así sucesivamente.

Los punteros son variables como cualquier otra, de modo que ocupan un espacio de almacenamiento que tiene una dirección. En consecuencia, es bastante posible tener punteros a punteros, y así sucesivamente. Esto puede dificultar bastante las cosas, pero son muy pocos los usos de matrices de punteros. Uno de ellos, en particular, es en el reconocimiento de argumentos de línea de comando, es decir, caracteres que se digitan en la misma línea que la llamada al programa.

Hay dos parámetros especiales que se le pueden pasar a la función principal, que proporciona acceso a los mismos. Se trata de argc, que da una cuenta de la cantidad de argumentos de línea de comando (estando los argumentos separados por al menos un espacio), y argv, que es una matriz de punteros a series con elementos argc. Cada elemento de argv apuntará al primer carácter en esa serie de números de la línea de comando. El siguiente y breve programa ilustra esto simplemente imprimiendo los argumentos de línea de comando, uno en cada línea:

```
main(argc,argv)
int argc;
char * argv[];
/* observe que esta declaración da una matriz
de punteros*/
{
   int i;
   for (i=1;i< argc;i++)
      printf("%s\n",argv[i]);
/* observe que el formato '%s' requiere un puntero a
una serie*/
}
```

Caer en una subrutina

Llegados a este punto, examinaremos la instrucción JSR ("Jump to subroutine": salto a la subrutina) y su utilidad práctica

Para mostrar cómo actúa esta instrucción, comencemos con un sencillo programa que transforma una tabla de 10 palabras de longitud mediante la fórmula:

```
TARRAY[I]:=ARRAY[I]^ 2+20*ARRAY[I]
```

y obtiene la suma de todos los elementos que componen TARRAY colocándola en la posición SUM. La fórmula parece un tanto complicada, pero lo único que expresa es "toma cada elemento de la tabla, elévalo al cuadrado, suma 20 veces dicho elemento, y después almacena el resultado en la casilla correspondiente de la nueva tabla, TARRAY".

La implementación del 68000 para esta transformación se muestra en el "listado uno". Para comprobar la corrección del programa debemos examinar los valores de TARRAY para los datos de comprobación (del 1 al 10) almacenados en ARRAY. Si usted ejecuta el programa, los valores almacenados serán, naturalmente, 21, 44, 69, 96, etc. Observe que el problema no especifica el intervalo de números que se han de usar en ARRAY. Si nuestras respuestas han de ser correctas, el resultado de la transformación (y la suma) se ajustarán a la longitud de una palabra, es decir, 16 bits.

Pero ésta no es, ni mucho menos, la única solución del problema, pues existen otros muchos caminos para llegar al mismo resultado. Esto ya depende del diseño de programa, y es en esta fase del proceso de codificación cuando cobra importancia el empleo de la subrutina, especialmente en el lenguaje assembly.

Hay muchos métodos para diseñar programas, pero una manera de alcanzar lo que pudiéramos llamar un programa "aseado" (al tiempo que resulta fácilmente comprensible cuando se vuelve a revisar) es el llamado *desglose funcional*. Se trata de un método que tiene por finalidad la elaboración de secciones de programas bien definidas que realicen ciertas funciones con los datos. Las funciones se describen mediante verbos tales como "calcular", "trasladar" o "comprobar", y estas secciones de código, en un lenguaje de alto nivel, se denominan *procedimientos*, con los datos facilitados a través de los *parámetros*.

Al nivel de la codificación en ensamblador, la única construcción modular adecuada disponible es la subrutina, que en el 68000 se llama por medio de:

```
JSR     SUBR
```

Esto altera el flujo del programa para realizar un salto a la sección de código etiquetada como SUBR. El código que está en SUBR se ejecutará hasta encontrar RTS, en cuyo punto el control del flujo del programa se devuelve a la instrucción siguiente a JSR (el mnemónico JSR significa *Jump to SubRoutine* [salto a la subrutina] y RTS quiere decir *ReTurn from Subroutine* [retorno de subrutina]).

Volvamos a nuestro programa ejemplo. Supongamos que en vez de realizar el cálculo de elevar al cuadrado, multiplicar y sumar "en una línea", deseamos "descomponer" esto con una llamada a la subrutina para conseguir un mejor diseño. Acabaremos con un programa algo así como el mostrado en el "listado dos". Todo lo que ha cambiado ha sido que en la línea 6 tenemos una llamada de subrutina a CALC, y lo que antes aparecía en las líneas de la 6 a la 9 se contiene ahora en las líneas 12 a la 15, con una RTS colocada al final.

La ventaja está en que ahora poseemos una sección de código bien definida (las líneas de la 12 a la 16), que realiza una operación específica sobre los datos pasados en D1 (un parámetro). No sólo es un buen diseño sino que podemos llamar a este código, CALC, ¡tantas veces como precisemos para que repita el mismo cálculo sobre otros datos siempre contenidos en D1 (y depositados en D2)!

Otro detalle a tener en cuenta cuando se implementan subrutinas, en especial en entornos de multiusuario, es que se tiene por una buena práctica el disponer de una sola entrada y una sola salida para las subrutinas. Esto quiere decir que debemos evitar las llamadas a otras subrutinas desde dentro de una subrutina. Tampoco es recomendable implementar un gran número de *returns* condicionados, que hagan la depuración una tarea difícil.

Ventajas de las subrutinas

La subrutina no sólo favorece el buen diseño de un programa, sino que proporciona programas económicos en lo que a ocupación de memoria se refiere. Esto se debe a que ahora no será necesario repetir el código de CALC cada vez que debamos emplearlo. Es adaptable tanto en el sentido de poderla llamar desde cualquier punto del programa como el que si deseamos introducir alguna modificación sólo tendremos que mirar un fragmento determinado del programa para realizarla.

Por ejemplo, supongamos que deseamos esta transformación:

```
D2:=D1^2-20*D1
```

la nueva subrutina CALC cambiará su línea 15 así:

```
SUB     D1,D2
```

y asunto concluido. Es más, el programa, además de ser adaptable, resulta fácil de depurar. Por

Listado uno

```
* Transformación de ARRAY en TARRAY mediante
* TARRAY[I]:=ARRAY[I]^2+20*ARRAY[I]
* Ambas tablas constan de 10 elementos
* Los registros empleados son:
*   A0 apunta a ARRAY
*   A1 apunta a TARRAY
*   D0 es un contador de bucle
*   D1 es un almacenamiento temporal
*   D2 es una copia de ARRAY[I]

          ORG     $1000
1 START   MOVEQ   #10,D0      * establece contador bucle
2         LEA     ARRAY,A0    * establece el primer puntero
3         LEA     TARRAY,A1   * y el segundo
4         CLR     SUM         * prueba si la suma es cero
5 LOOP    MOVE    (A0)+,D1    * toma ARRAY[I]
6         MOVE    D1,D2       * y la copia
7         MULS    D2,D2       * hace el cuadrado de ARRAY[I]
8         MULS    #20,D1      * D1 llega a ser 20 veces ARRAY[I]
9         ADD     D1,D2       * suma elementos
10        MOVE    D2,(A1)+    * almacena en TARRAY[I]
11        ADD     D2,SUM      * y guarda el total dinámico
12        SUBQ    #1,D0       * decrementa contador bucle
13        BNE     LOOP        * itera si no es el fin
14        TRAP    #0          * salida a monitor
15 ARRAY  DC.W    1,2,3,4,5,6,7,8,9,10
16 TARRAY DS.W    10
17 SUM    DS.W    1
18        END
```

Notas al programa:
Entre las líneas 1 y 4 se inicializan las variables empleadas por el programa; después, en la línea 5, cada elemento de ARRAY es cargado en D1 mediante el direccionamiento indirecto con posdecremento. En la línea 6 se hace una copia en D2, de manera que no tenemos que acceder a la tabla de nuevo. Las líneas 7 y 8 forman dos componentes de la transformación en D1 y D2, mediante MULS, y la línea 9 suma ambas. La línea 10 almacena el resultado en el elemento correspondiente de TARRAY, y la línea 11 suma el resultado en la suma dinámica, SUM. Finalmente, las líneas 12 y 13 son sentencias de control del bucle que permiten transformar sólo los 10 elementos (entre 5 y 12/13 se establece, en efecto, un bucle FOR con el contador inicializado en la línea 1)

Listado dos

```
          ORG     $1000
1 START   MOVEQ   #10,D0
          LEA     ARRAY,A0
          LEA     TARRAY,A1
          CLR     SUM
5 LOOP    MOVE    (A0)+,D1
6         JSR     CALC        * llamada a subrut. para calcular
7         MOVE    D2,(A1)+    * nuevo valor de la tabla
8         ADD     D2,SUM
9         SUBQ    #1,D0
10        BNE     LOOP
11        TRAP    #0
12 CALC   MOVE    D1,D2       * subrut. para calc. D1^2+20*D1
13        MULS    D2,D2       * y poner el valor en D2
14        MULS    #20,D1
15        ADD     D1,D2
16        RTS                 *retorno al programa que llamó
```

ejemplo, podemos probar la validez de CALC mediante todo tipo de valores que daremos a D1 y examinar los resultados en D2 sin mayores complicaciones con otros trozos del programa o con datos irrelevantes. Sólo debe haber una entrada a la subrutina (en la etiqueta de dirección de la subrutina) y sólo una salida (en la instrucción RTS). Así podemos afirmar que la estructuración de nuestros programas con subrutinas facilitan las pruebas y la depuración.

Parece, pues, que así se satisfacen los criterios de un "buen programa", por lo menos desde el punto de vista de la economía, adaptabilidad y fiabilidad. Pero el empleo de subrutinas comporta también ciertas desventajas. Para percatarnos de ellas (y examinar la transmisión de datos por medio de parámetros) es necesario echar un vistazo al mecanismo de la pila en el 68000.

El mecanismo de la pila

Cuando ejecutamos la instrucción JSR es preciso recordar la dirección de la instrucción siguiente a aquella que llama a la rutina (el enlace con la subrutina). La instrucción JSR hace que se coloque en la pila la dirección de esa instrucción siguiente (se necesitan cuatro bytes para la dirección de una palabra larga completa) y que el contador del programa (PC) se cargue con la dirección de la subrutina. Cuando se ha ejecutado RTS en la subrutina, el PC se cargará con dicha dirección que será sacada de la pila, y la ejecución continuará después de la llamada a la subrutina. Toda esta manipulación de la dirección se hace al margen del usuario, pues el 68000 se encarga de todo. Pero hay efectos colaterales que el programador debe conocer.

Ante todo, hay que cerciorarse de que el puntero de la pila, SP (o bien, A7 en el 68000), es activado correctamente, de lo contrario podemos sobreescribir el código o los datos, ¡o incluso violar las direcciones del hardware! El modo habitual de hacer esto es:

```
STACK   EQU  $1000       *pone la pila a 1000 hexa
BEGIN   LEA  STACK,SP    *inicializa el puntero
                         de la pila
```

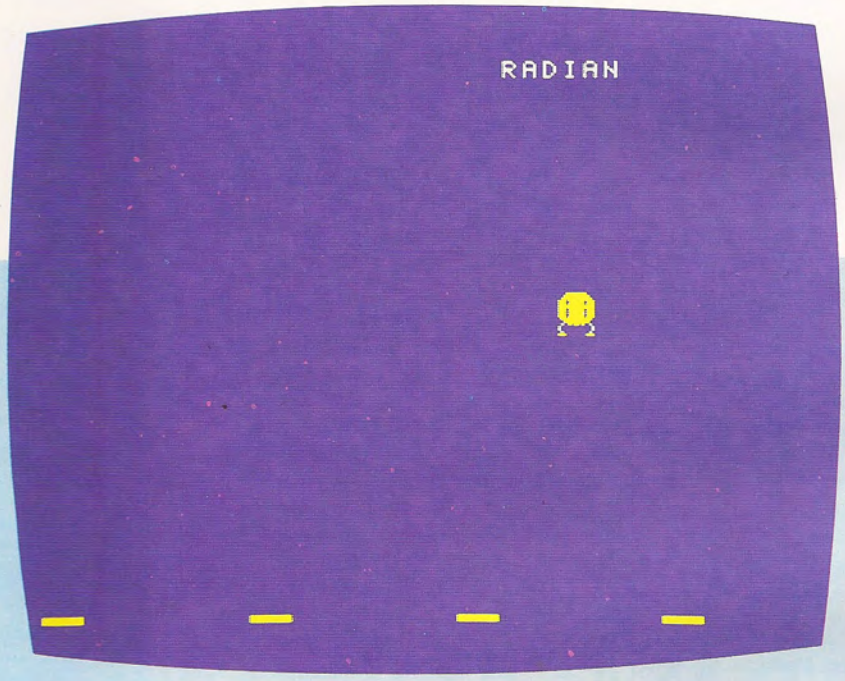
y la pila irá pasando del 1000 al cero (porque cada *push* o envío a la pila es un predecremento).

En segundo lugar, ¡debemos estar seguros de que la pila no haya crecido demasiado! Debemos calcular la cantidad de pila que necesitamos usar, lo cual es difícil. Para programas relativamente sencillos con sencillas llamadas a subrutinas y empleos de la pila, es posible tener una respuesta exacta mediante el examen del código. Para programas *anidados* (es decir, con subrutinas que llaman a otras subrutinas) o *recursivos* (una subrutina se llama a sí misma), el problema de dar un tamaño a la pila puede ser casi irresoluble, y se han de emplear técnicas de ensayo y error.

En el próximo capítulo daremos ejemplos pormenorizados de llamadas a subrutinas y, en particular, nos ceñiremos a los métodos para pasar los datos a y desde la subrutina, empleando en ciertos casos algunas instrucciones únicas del 68000.

Aterrizaje

He aquí un juego de acción que está siempre de actualidad. Esta versión, escrita por Pierre Monsaut, está destinada al microordenador EXL 100



Después de un largo viaje sin gravedad, no resulta nada fácil aterrizar suavemente con una nave espacial; pero gracias a su ordenador está en condiciones de efectuar un entrenamiento sin riesgos. Debe posar su nave sobre una de las cuatro zonas destinadas al efecto. Usted puede dirigirse a la derecha y a la izquierda con ayuda de las teclas de control del cursor.

```
100 REM **************
110 REM * ATERRIZAJE *
120 REM **************
130 GOSUB 590
140 FOR I=1 TO 100
150 NEXT I
160 IF DL<0 THEN DL=0
170 GOSUB 690
180 FOR Q=2 TO 20
190 FOR I=0 TO DL*10
200 NEXT I
210 CALL KEY1(D3,D4)
220 NH=NX
230 NX=NX+(D3=131)-(D3=129)
240 IF NX<2 TTHEN NX=2
250 IF NX>38 THEN NX=38
260 LOCATE (Q-1,NH)
270 PRINT C$;
280 LOCATE (Q,NH)
290 PRINT C$;
300 LOCATE (Q,NX)
310 PRINT N$;
320 LOCATE (Q+1,NX)
330 PRINT M$;
340 NEXT Q
350 IF INT ( (NX-3)/10)=(NX-3)/10 THEN DL=DL-1:GOTO 140
360 CALL COLOR("1Yb")
370 LOCATE (Q,NX)
380 PRINT C$;
390 LOCATE (Q-1,NX)
400 PRINT C$;
410 LOCATE (Q+1,NX-1)
420 PRINT H$;
430 CALL COLOR("1RG")
440 LOCATE (7,6)
450 PRINT "SU NAVE SE HA ESTRELLADO";
460 LOCATE (12,12)
470 PRINT "SCORE :";S-1;
480 LOCATE (15,11)
490 PRINT "OTRA ?";
500 FOR I=1 TO 100
510 NEXT I
520 CALL KEY1(D3,D4)
530 IF D3<>255 THEN 520
540 CALL KEY1(D3,D4)
550 IF D3=255 THEN 540
560 IF D3<>78 THEN RUN
570 CLS
580 END
590 CLS ("Ybb")
600 GOSUB 800
610 B$=CHR$(32)
620 N$=CHR$(100)&CHR$(101)
630 C$=B$&B$
640 M$=CHR$(102)&CHR$(103)
650 A$=CHR$(104)&CHR$(104)
660 H$=CHR$(103)&CHR$(101)&CHR$(100)&CHR$(102)
670 S=0
680 RETURN
690 CALL COLOR("1Yb")
700 FOR I=1 TO 20
710 NEXT I
720 CLS
730 FOR I=0 TO 3
740 LOCATE (22,3+I*10)
750 PRINT A$;
760 NEXT I
770 S=S+1
780 NX=INTRND(35)+2
790 RETURN
800 CALL CHAR(100,"0000001F3F7FEFEFFFEF")
810 CALL CHAR(101,"000000E0F0F8DCDCFCDC")
820 CALL CHAR(102,"EFEFEF7F3F4BB08040E0")
830 CALL CHAR(103,"DCDCDCF8F0480404081C")
840 CALL CHAR(104,"FFFFFF00000000000000")
850 DL=20
860 RETURN
```

© SYBEX

Marcus Wilson-Sm

Punto de partida
Para quienes no tengan experiencia y deseen hacer una carrera en el ámbito de las ventas o el marketing, probablemente el lugar más adecuado para iniciarse sea una tienda minorista como la que vemos en la fotografía, donde se venden equipos a usuarios finales. No obstante, las perspectivas continúan siendo escasas en un entorno sometido a demasiadas presiones como para permitir que el empleado adquiera una conciencia más cabal y profunda de la nueva tecnología y las técnicas de venta

La senda del éxito

En un mercado tan complejo como el informático, los departamentos de ventas y marketing tienen gran relevancia

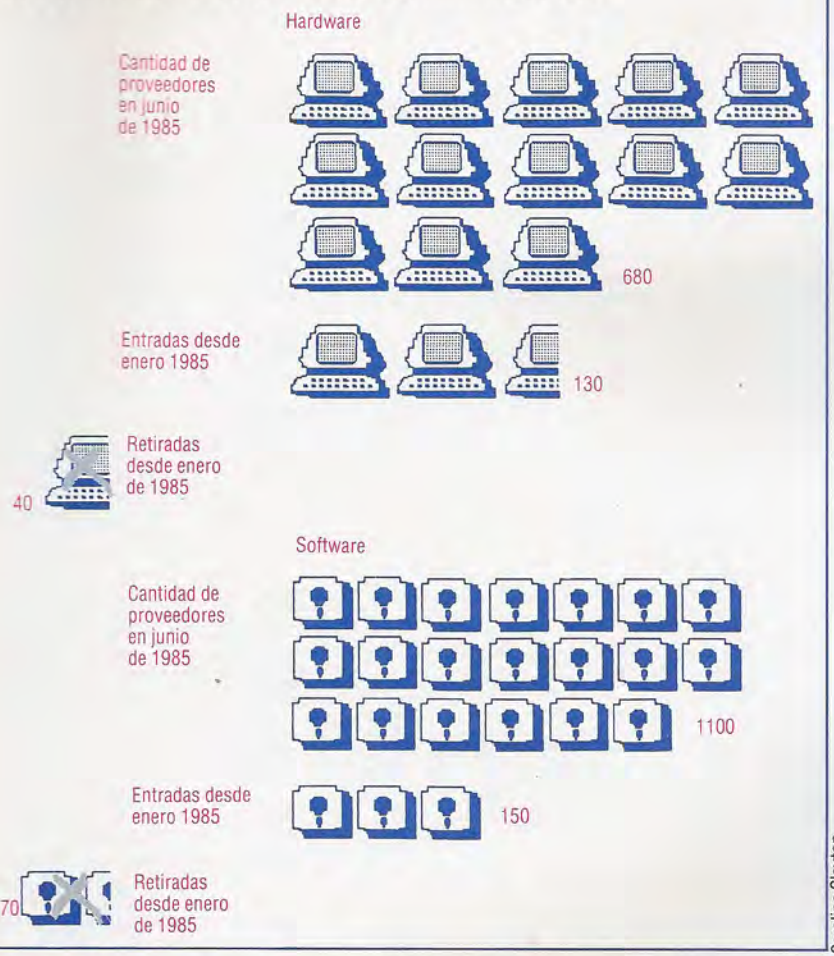
El gran crecimiento que experimentó la industria del microordenador a comienzos de los años ochenta comenzó a menguar y estabilizarse hacia mediados de 1984 y, por consiguiente, las expectativas de creación de puestos de trabajo también se redujeron. Se cuentan por centenares las firmas que se han retirado del negocio, pero se han creado casi el triple de ellas, lo que indica que en el campo de la informática se está creando un alto número de colocaciones.

En Gran Bretaña los puestos de trabajo en ventas y marketing figuran entre los mejor pagados en la industria del ordenador. En el extremo superior, el vendedor experimentado que gestiona valiosos contratos de hardware o software con grandes clientes corporativos puede esperar hacerse con £80 000 (17 millones de pesetas, aproximadamente) o más al año, principalmente a base de comisiones. Los ejecutivos de marketing (que no cobran sobre la base de comisiones) tienen pocas posibilidades de ganar tanto dinero, aunque cuando un empleado llega al puesto de director de marketing de una empresa entre mediana o grande son bastante corrientes los sueldos de más de £30 000. Tanto ventas como marketing constituyen sendas convencionales a través de las cuales los empleados más emprendedores y ambiciosos pueden acceder a la cúpula de una empresa. Hasta hace poco tiempo, la totalidad de los principales ejecutivos de IBM habían ascendido a través del escalafón de ventas.

Algunas agencias de colocaciones recomiendan que los jóvenes aspirantes a vendedores empiecen por seguir un curso de informática, con el objeto de obtener una comprensión elemental del producto. Luego aceptar un empleo en una firma minorista que venda equipos de oficina u ordenadores personales. Esto se suele pagar bastante mal, pero les proporcionará alguna experiencia básica. Después de ello, estarán en una posición más ventajosa para convencer a una compañía de fabricación o a un distribuidor de ordenadores para que les dé empleo. El primer año tampoco ganarán mucho, y se esperará que acompañen a un vendedor senior y aprendan a hacer demostraciones de ordenadores y paquetes de software.

Después de esto, si el aspirante demuestra aptitud, sus perspectivas mejorarán rápidamente. Avalado por un año de experiencia en cualquiera de los micros y paquetes de software principales, estará capacitado para un puesto ejecutivo de ventas, cuyo sueldo básico es posible duplicar con las comisiones. Es muy probable que la empresa también ponga a su disposición un automóvil. Cuando se anuncia un trabajo de "£20 000 OTE" (*on-target earnings*), significa que las £20 000 son las "ganancias sobre cuota" que el vendedor percibirá sólo si alcanza el objetivo de ventas que ha establecido para él el director de ventas. (Según los especialis-

Crecimiento de los suministros



Caroline Clayton

Suministros informáticos
A pesar de las pesimistas previsiones de algunos observadores, la cantidad de proveedores dentro de la industria del ordenador ha seguido creciendo a ritmo constante. El crecimiento estable, más que unos beneficios elevados, a menudo será una tentación para que se creen nuevos negocios en un mercado que, de no existir esta circunstancia, sería evitado (cifras del NCC Microsystems Centre, Gran Bretaña)

tas en gestión de empleo, la mayoría de las empresas muestran preferencia por aspirantes de edades entre veinte y veinticinco años.)

Al cabo de tres o cuatro años, el aspirante puede llegar a ser ejecutivo de ventas senior o vendedor de sistemas. En una empresa que venda ordenadores tanto individuales como multiusuario, el personal senior, evidentemente, manejará los sistemas multiusuario más costosos, que tienen las comisiones netas más elevadas. Para cuando cumpla los treinta años de edad, sus expectativas estarán en ocupar un puesto de director de ventas, a quien corresponderá determinar las cuotas a cubrir por su personal de ventas. A este nivel quizá descubra que sus atribuciones y su campo de acción coinciden en parte con los del director de marketing, y de hecho muchos ejecutivos en este momento deciden pasarse a esta última actividad. Tenga presente que las ventas pueden ser un asunto comprometido, especialmente en la industria del micro. Muchos vendedores se han encontrado con que su director de ventas les asignaba objetivos inalcanzables. Al no conseguir acceder a la meta fijada durante dos o tres trimestres, fueron despedidos. ¡Éste no es un trabajo para aquellos que busquen seguridad!

Si descubre que no le agradan las ventas, o que no es especialmente idóneo para vender, el mejor consejo es abandonar el negocio mientras aún sea joven. Si mediada la treintena aún no ha accedido a un puesto de director de ventas, quizá le resulte difícil conseguir otro empleo en este campo.

El juego de vender

En una empresa las funciones de ventas y marketing con frecuencia se confunden. Un vendedor trata directamente con el cliente. El ejecutivo de marketing, por el contrario, está más implicado en la estrategia: cómo empaquetar el producto, qué nombre ponerle, cómo anunciarlo, etc. En la industria del microordenador hay dos ramas fundamentales: la fabricación y la reventa. La gente de ventas que trabaja para fabricantes de hardware y software típicamente vende sus productos a minoristas o distribuidores. Los vendedores para revendedores, tales como casas de sistemas, detallistas o cadenas minoristas, tratan directamente con el usuario final. En Estados Unidos es corriente que los clientes entren en una tienda de ordenadores y adquieran un ordenador de gestión. Por el contrario, en Gran Bretaña casi todas las ventas de micros implican el envío de correspondencia a los potenciales clientes (una función de marketing) o bien una "llamada en frío" (el vendedor establece un contacto directo por teléfono o personalmente). En los primeros días de la microinformática, la industria no le prestaba gran atención al marketing. Las nuevas empresas fabricaban novedosos y atractivos productos y los vendedores los vendían. En algunas ocasiones tenían éxito (el Spectrum) y en otras no (el Coleco Adam). Ahora, sin embargo, la industria es más sofisticada y en 1985 el fabricante de ordenadores personales de mayor éxito fue Amstrad, con su serie CPC. No había nada nuevo en la tecnología; simplemente fue cuestión de empaquetarla de la forma que querían los clientes. Y eso es, en esencia, lo que se entiende por marketing

El marketing es un ámbito muy competitivo. Las firmas buscan graduados universitarios, preferentemente con titulación en matemáticas o física.

Un punto importante que hay que entender en relación al marketing es que rigen los mismos principios independientemente del producto que venda la compañía comercial. De modo que la experiencia obtenida en el departamento de marketing de una organización que fabrique jabón en polvo coloca al aspirante en una buena posición cuando llegue al mercado de ordenadores. De hecho, muchas agencias de empleo recomiendan que los graduados jóvenes trabajen durante dos o tres años en una gran corporación, preferiblemente alguna que los haga asistir a un curso de marketing o que les permita obtener un título en administración de empresas. Las firmas de ordenadores, al ser más pequeñas, suelen contar con menos recursos para formar profesionalmente a sus empleados.

El conocimiento en materia de ordenadores no es vital para obtener un puesto de marketing en una firma de microordenadores, pero es muy conveniente interesarse en el tema. Un empleado de marketing con dos o tres años de experiencia ganará unas £10 000 al año. Las tareas que deberá realizar para la firma incluirán escribir folletos y ejemplares publicitarios, enviar información a la red de ventas, organizar seminarios y conferencias de prensa, y atender a los periodistas (normalmente en esta última actividad será sustituido por alguien más especializado en este campo específico).

Reacción en cadena

La firma británica Granada Business Centres vende micros individuales y multiusuario a pequeñas y medianas empresas. La empresa declara que el requisito primordial de sus aspirantes a un puesto de vendedor es que posean experiencia en ventas, preferiblemente de equipos de oficina o, aun mejor, de sistemas de ordenador. Granada imparte su propia formación en productos y aptitudes para las ventas, estando comprendidas las edades del personal de ventas que contrata entre los 23 y 30 años. Cada tienda posee un ejecutivo de ventas senior que se ocupa de las grandes "cuentas nacionales", y un gerente de ventas que supervisa la actividad de ventas de su tienda. Después hay un gerente de distrito que se ocupa de un grupo de cuatro tiendas incluyendo al personal de mantenimiento y apoyo que trabaja para cada centro. El camino para hacer carrera es éste:

vendedor→ejecutivo de ventas senior→ gerente de ventas→gerente de distrito

El personal de marketing es mucho más reducido: cuatro personas, frente a 60 en el campo de ventas. Cada gerente de marketing suele cumplir una función específica. Una persona se encarga de las ferias, otra de la prensa y relaciones públicas, otra de la presentación de las tiendas, etc.

El siguiente paso conduce al cargo de gerente de marketing, donde se asume un papel más directo en establecer enlaces con las agencias de publicidad, decidir a qué ferias presentarse, planificar lanzamientos de productos y la estrategia de la empresa. El sueldo base sería de unas £14 000, con un coche de la empresa y otras prerrogativas. En una empresa pequeña (como son casi todas las firmas de microordenadores) sería razonable esperar convertirse pronto en director de marketing con un asiento propio en la mesa de juntas.

Pequeño ejemplo

Psion es una firma británica que produce software y ordenadores de mano. Tiene contratadas a 70 personas y espera que sus aspirantes a un puesto de ventas posean algo de experiencia en cuanto a la venta de equipos técnicos o de oficina, pero no hace especial hincapié en el conocimiento sobre ordenadores. Ofrece formación sobre los productos de la compañía. El vendedor tratará directamente con los comerciantes que llevan los productos de Psion y los grandes compradores corporativos. Un camino típico para hacer carrera sería:

vendedor→gerente de cuentas→gerente de ventas→director de ventas

Un gerente de cuentas es responsable de manejar varias cuentas y supervisar la capacidad de ventas. En el campo del marketing, Psion sólo contrata graduados y ejecutivos de marketing. Su tarea consiste en escribir los textos publicitarios, organizar los stands de las ferias, atender a la prensa y proporcionar apoyo al cuerpo de ventas. Aquí el camino sería:

ejecutivo de marketing→gerente de marketing→director de marketing

El gerente y el director de marketing son responsables de la planificación a largo plazo, de tratar con las agencias de publicidad, de encargar estudios de mercado, etc.
Las pequeñas empresas pueden ofrecer ventajas a un empleado que no esté seguro del camino a seguir, dado que su estructura es más flexible que la de las grandes firmas. En 1983, Psion contrató como programador a un graduado que luego pasó al apoyo técnico de ventas (respondiendo a cuestiones que el equipo de ventas no podía manejar solo) antes de ascender a gerente de cuentas de ventas. Hacia finales de 1985 había pasado a la división de exportaciones

Acierto publicitario
El marketing representa muchísimo más que vender un producto. Un ejecutivo de marketing estará comprometido con un producto desde la etapa de concepción y diseño hasta su lanzamiento, asegurándose de que durante el desarrollo del producto se tengan muy presentes las exigencias y preferencias del usuario final. El Amstrad PCW 8256 constituye un buen ejemplo de producto bien comercializado. Si bien utiliza la tecnología de ayer, responde a las necesidades del consumidor de hoy en cuanto a tratamiento de textos a bajo costo. (El slogan publicitario reza: "Más que un procesador de textos, por menos que una máquina de escribir".) Compare su éxito con, por ejemplo, el vehículo eléctrico Sinclair C5, que, a pesar de la gran campaña publicitaria que acompañó su lanzamiento, no logró venderse bien, lo que demuestra que la publicidad, por intensiva que sea, no puede vender un producto para el cual no existe un mercado

Cortesía de Amstrad

¡Utilidades Unix!

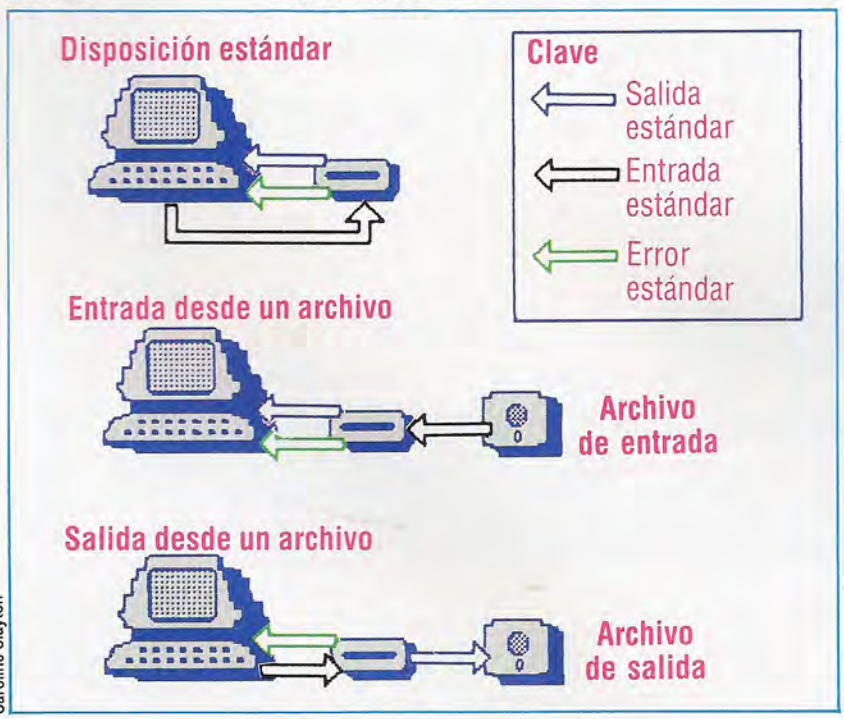
Unix ofrece una biblioteca de llamadas y comandos para simplificar la labor del usuario

Debido a que las utilidades y herramientas Unix son programas en c directamente, es fácil combinar con ellas los programas de uno mismo. Para simplificar aún más las cosas, el Unix tiene un enfoque unificado fundamental de los archivos y de dispositivos de E/S. Se considera que cada archivo es una corriente o secuencia de bytes. El Unix lee o escribe secuencialmente los bytes en el archivo y permite posicionar el puntero de archivo en cualquier desplazamiento de byte legítimo del archivo y comenzar a leer o escribir en este punto.

Los dispositivos de E/S se tratan exactamente igual que cualquier otro archivo. De hecho, en el directorio /dev siempre habrá una lista de dispositivos de E/S. Las operaciones que se pueden efectuar se observan mejor utilizando las llamadas a la biblioteca c de nivel más bajo:

- creat(nombrearchivo,modalidadprotección): Si ya existe un archivo con el nombre dado, entonces se lo trunca a longitud cero; de lo contrario, se crea un archivo con ese nombre. La modalidadprotección es un número de nueve bits (comúnmente en forma de tres dígitos octales) que especifican la permisión de lectura, escritura y ejecución (un bit cada uno) para el usuario, el grupo del usuario y de quien esté en el sistema. La función devuelve un descriptorarchivo entero.

Rutas de redirección
En circunstancias normales, la entrada para un programa que se esté ejecutando en un sistema Unix se produce a través del teclado. La salida y los avisos de error normalmente se dirigen hacia la pantalla. En el Unix, los operadores < y > permiten redirigir la entrada a un programa o bien la salida del mismo, de modo que el programa se comunique con otros dispositivos o archivos. Esta potente facilidad permite crear con toda sencillez archivos de datos, o bien que el "caparazón" ejecute una serie de comandos Unix retenidos en un archivo



Caroline Clayton

- open(nombrearchivo,modalidadlecturaescritura): Conecta el archivo mencionado con el programa, especificando (en modalidadlecturaescritura) 0 para lectura, 1 para escritura y 2 para acceso tanto de lectura como de escritura. El archivo debe existir ya y la función devuelve un descriptorarchivo entero.
- close(nombrearchivo): Desconecta el archivo del programa.
- unlink(nombrearchivo): Suprime el archivo del sistema.
- read(descriptorarchivo,buffer,cantidaddebytes) y write(descriptorarchivo,buffer,cantidaddebytes): Transfieren el número especificado de bytes entre el archivo, del cual se da el descriptor, y una zona de almacenamiento de buffer mencionada.

Tres archivos

Hay tres archivos abiertos automáticamente para cualquier programa, que se denominan standardinput, standardoutput y standarderror. Normalmente, standardinput está conectado al teclado y standardoutput y standarderror están conectados a la pantalla. Los dos primeros se pueden redirigir hacia o desde cualquier otro archivo o dispositivo mediante los operadores < y >.

Para cualquier comando o programa Unix:

```
nombreinstruccion<nombrearchivoodispositivo
```

hará que la entrada desde standardinput se tome realmente del archivo mencionado en vez de desde el teclado.

Del mismo modo:

```
nombreinstruccion>nombrearchivoodispositivo
```

hará que la salida de standardoutput se envíe al archivo mencionado en vez de a la pantalla. Una variación permitirá añadir la salida a un archivo en lugar de crear un archivo nuevo, si bien se creará un archivo nuevo si éste no existe:

```
nombreinstruccion>>nombrearchivoodispositivo
```

Las acciones de estas dos últimas se pueden alterar si se establece noclobber (una opción especial que más adelante veremos en mayor profundidad). Ésta no permitirá que se supriman archivos sin confirmación, pero se puede invalidar con la forma imperativa de estos dos operadores:

```
nombreinstruccion>!nombrearchivoodispositivo
nombreinstruccion>>!nombrearchivoodispositivo
```

A modo de ejemplo del empleo de esta facilidad, podemos crear un archivo que contenga la lista de archivos de un directorio utilizando:

```
ls>lsarchivo
```

otro uso es para proporcionar una facilidad de archivo de comandos, de modo que toda una secuencia de comandos se pueda ejecutar a la vez. Los comandos los ejecuta el "caparazón", el cual se puede invocar utilizando la instrucción sh. De modo que:

```
sh<nombrearchivo
```

hará que se ejecuten los comandos del archivo.

Una característica fundamental del Unix, que le confiere tanta potencia, es la facilidad de tubería que da el operador |.

Responde al formato:

```
nombrecomandoprograma|nombrecomandoprograma
```

que conecta standardoutput del primer comando con standardinput para el segundo. De modo que la salida del primer comando se toma automáticamente como la entrada para el segundo.

Esto no significa que se produzca un archivo intermedio, sino que los dos programas se ejecutarán de forma concurrente. Siempre que el segundo programa necesite entrada, suspenderá la operación hasta que el primer programa haya producido alguna salida. Por ejemplo, la instrucción ls no cuenta la cantidad de archivos del directorio mientras los va listando.

Existe una utilidad, wc, que cuenta la cantidad de palabras de un archivo utilizando la opción –w; el comando:

```
ls | wc–w
```

hace que la salida de ls (una lista de nombres de archivo) sea la entrada para wc, que cuenta el número de palabras, de modo que la salida será la cantidad de archivos del directorio.

El operador de tubería tiene un uso especial en conjunción con el comando tee, que enviará salida tanto a un archivo como a la pantalla. Así:

```
ls | tee lsarchivo
```

hará que el listado del directorio aparezca en la pantalla así como en lsarchivo.

El Unix es un sistema operativo multitarea y, en consecuencia, puede ejecutar dos o más tareas de forma concurrente. Pero también permite que cada usuario ejecute más de un programa a la vez. Si se coloca el operador & al final de la línea de comando, el comando o programa se ejecutará como tarea de fondo, dejando al usuario en libertad de ejecutar otro comando o programa. Por supuesto, si este proceso de fondo utiliza una entrada o salida de terminal, entonces es probable que ocurran cosas extrañas.

También se puede detener una tarea en medio de su ejecución pulsando Ctrl-Z, que permite reiniciar la tarea después, sin perderla. Si el usuario está ejecutando simultáneamente varias tareas y algunas se interrumpen, puede resultar difícil saber lo que está sucediendo. No obstante, el comando ps da una lista de los estados de todos los procesos que se están ejecutando para el usuario, y el comando jobs lista todas las tareas de fondo y las que están detenidas.

Cuando se detiene un proceso o se lo coloca como tarea de fondo, el Unix le otorga dos números: un número de tarea personal, como [1], [2], etcétera, y un número de proceso que está en relación con todos los procesos que se estén ejecutando actualmente en la máquina. El número de tarea se puede utilizar en unión con los comandos fg y bg para intercambiar procesos. El comando fg%númerodetarea, por ejemplo, reiniciará un proceso detenido en el primer plano, o pasará un proceso de fondo a primer plano. El comando bg%númerodetarea reiniciará un proceso de fondo, y mediante stop%númerodetarea se puede detener un proceso que se esté ejecutando en el fondo. Los procesos que ya no son necesarios se pueden detener permanentemente mediante el comando kill númerodeproceso.

Dirigiendo el tráfico

```
%ls –l > lsfile
(no aparece el listado del directorio en la
pantalla)

%cat lsfile          (mirar en lsarchivo)
total 41
–rw–r—r—      1 com–mcc      0    Oct 28  11:55 lsfile
drwxr–xr–x    2 com–mcc    512    Oct 21  11:11 mike
–rw–rw–r—     1 com–mcc    502    Sep 17  12:07 rec.c
–rwxr–xr–x    1 com–mcc  18432    Oct 21  11:02 receive
–rw–r—r—      1 com–mcc   1068    Oct 18  14:44 rx.p
–rwxr–xr–x    1 com–mcc  19456    Oct 21  11:03 transmit

%ls | sort –r > lsfile
(esta vez la lista del directorio se ha entubado a la utilidad sort con la opción -r,
que clasifica por orden inverso)

%cat lsfile          (listar archivo clasificado)
transmit
rx.p
receive
rec.c
mike
lsfile

%pc rx.p &           (ejecutar compilacion pascal como tarea de fondo)
[1] 1217             (se le da a la compilación el n.° de tarea [1]. y al proceso el
                     n.° 1217)

%jobs
[1] + Running pc rx.p

%ps
PID     TT      STAT TIME COMMAND
1217  n09 I       0:00   pc rx.p
1218  n09 R       0:03   pc0 -o/tmp/p0001217 rx.p
(this last process was set up by the pascal compiler)
1220  n09 R       0:01   ps

%stop 1217           (detenida compilación por citar el n.° de proceso)
[1] +Stopped(signal) pc rx.p

%bg %1               (compilación reiniciada en el fondo utilizando el n.° de job)
[1] pc rx.p &

%fg %1               (pasar compilación al primer plano)
pc rx.p

Stopped              (esto se hizo utilizando Ctrl Z)

%jobs
[1] +Stopped pc rx.p

%kill %1             (librarse por completo del proceso)
[1] Exit 1 pc rx.p

%ps
PID   TT   STAT TIME   COMMAND
1266  n09  R      0:01   ps
(ningún proceso en ejecución, a excepción de ps)

%logout
```

El Unix, como todos los sistemas operativos buenos, mantiene una hora y fecha que se pueden utilizar de diversas maneras. La instrucción date simplemente proporciona la fecha del día y cal un calendario, que puede presentarse con diversos formatos. La instrucción times nombrecomando le dirá la hora de ejecución para ese comando en particular, y at seguida por una hora y un nombre de archivo ejecutará una lista de comandos retenidos en el archivo en la hora especificada.

La hora del modem

Si bien el modem fue creado hace varios años, sólo ahora se ha generalizado su utilización

En Estados Unidos los modems constituyen un gran negocio. En Europa, sin embargo, su impacto ha sido menos pronunciado. Esto se debe a diversos motivos, que pueden reducirse a dos factores principales. En primer lugar, los recargos por conexión a la red telefónica, en particular durante las horas punta, pueden resultar prohibitivamente altos. Las personas que usan sus ordenadores en la red durante varias horas por semana con frecuencia se quedan atónitas ante su factura telefónica, que puede ascender a muchos miles de pesetas.

En segundo lugar, las compañías telefónicas han tardado bastante en comprender el potencial de las redes de comunicaciones y han desarrollado su sistema de un modo fragmentario. Esto significa que la base de datos Prestel, por ejemplo, utiliza distintos protocolos de tablones de anuncios privados, etcétera.

Otro factor que ha obstaculizado el desarrollo en los países europeos es el hecho de que antes de que se pueda utilizar un modem legalmente en la red telefónica, normalmente debe ser homologado por la compañía telefónica de turno, y la lentitud de estas empresas para conceder esta aprobación siempre ha sido notable. Ello ha significado que en muchos casos transcurrieran años antes de que los avances en la tecnología del modem llegaran hasta el usuario. En el caso de Gran Bretaña, desde la privatización de la BT (British Telecom), la autorización de los modems se ha transferido al Departamento de Comercio e Industria, lo que habría de volver más eficaz el proceso.

En Estados Unidos, muchas llamadas locales son gratuitas y casi todas las bases de datos y sistemas de comunicaciones se basan en el estándar Hayes. Éste es un sistema de protocolos que ha sido adoptado por otros numerosos fabricantes de modems. Esta normalización no existe para el usuario británico, aunque el desarrollo del sistema Hayes en este país es inminente.

En consecuencia, para ser totalmente adaptable en Gran Bretaña, es imprescindible que el modem tenga incorporadas estas y otras numerosas características para permitir al usuario la máxima flexibilidad en la a menudo espesa maraña de las comunicaciones.

A pesar de estos problemas, la venta de modems continúa aumentando. Veamos ahora las facilidades que ofrecen algunos de los modems disponibles en el mercado británico.

Crispin Thomas

PACE NIGHTINGALE

PRECIO
£137, excluyendo software e interface

PARA USAR CON
BBC Micro, Commodore 64 y gama Amstrad CPC

VELOCIDAD DE TRANSMISION
Envío y Recepción se pueden ajustar en una de 7 velocidades que van de 75 a 9800 baudios

Nightingale

El Pace Nightingale está dirigido específicamente al BBC Micro, si bien puede ser adoptado por cualquier micro que posea una puerta RS232. Al igual que el paquete Modem 1000, proporciona el software en una EPROM que se inserta en uno de los conectores de ROM laterales del BBC Micro. Si bien varias de las facilidades del modem están contenidas en el software activado por menú, el Nightingale permite establecer la velocidad de transmisión, a través de botones situados en la parte frontal del dispositivo, según los estándares del Prestel y los tablones de anuncios de 300 baudios más comunes. El software adapta el ordenador para la operación en Prestel o en modalidades de emulación de terminal. Desde cualquiera de estos menús se puede transferir información y manipularla dentro del buffer de la máquina. Asimismo, el Nightingale viene con un manual que es el más exhaustivo de los que acompañan a todos los modems que analizamos en este capítulo. Debido a la vasta cantidad de BBC Micros instalados en establecimientos educativos, probablemente los fabricantes hayan diseñado el manual teniendo presente el mercado educativo. Pero, sea cual fuere la razón, le dice al usuario mucho más de la escasa información que proporcionan otros muchos modems pensados para micros personales.

VTX 5000

A pesar de su historia azarosa, el VTX 5000, fabricado por OE Ltd, se ha convertido en el modem estándar para usuarios del Spectrum. Se desarrolló originalmente para Prism, que estaba interesada en ampliar la base de usuarios para Micronet, en la que la empresa tenía grandes participaciones. Aunque cuando Prism fue a la quiebra, a comienzos de 1985, el modem fue asumido por Modem House, siguió vendiéndose bien.

En la parte frontal del modem hay una luz de potencia, una luz Line (que indica si el modem está "en línea") junto con un interruptor. Otro interruptor permite seleccionar la modalidad requerida para el dispositivo, que puede ser para Micronet, TX (transmitir) o RX (recibir). En la parte posterior del dispositivo hay un conector para enchufe telefónico y un cable, que se pueden enchufar directamente a un conector telefónico de pared.

El software del VTX 5000 está contenido en ROM. En el interior del VTX 5000 hay dos placas de circuito impreso, una que se ocupa de la transmisión en serie y sistema de circuitos de decodificación, mientras que la otra contiene la lógica específica del Spectrum, tales como el chip ACIA de alimentación de potencia y el software basado en ROM. Este método de proporcionar el software del modem presenta un problema al usuario del Spectrum. La ROM trabaja paginando la ROM de BASIC y ocupando esa zona de memoria. El problema se plantea cuando los usuarios desean utilizar la Interface 1 junto con el VTX 5000, para, por ejemplo, cargar programas en un microdrive. Debido a

que la Interface 1 aplica la misma técnica para paginar su propia ROM, los dos dispositivos no se pueden utilizar juntos.

El software del modem es activado por menú y tras el encendido presenta un menú de opciones, incluyendo conexión con Micronet o envío de mensajes a través de la facilidad de buzón de correo. Sin embargo, el software no permite conectar con ninguno de la multitud de tablones de anuncios privados que están surgiendo a lo ancho y a lo largo de Europa, si bien es posible escribir software para hacerlo.

VTX 5000
PRECIO
£69,95
PARA USAR CON
Sinclair Spectrum
VELOCIDAD DE TRANSMISION
75/1200 baudios

Modem 1000

Con el mismo origen que el VTX 5000, el Modem 1000 tiene muchas facilidades en común con la máquina exclusiva para el Spectrum. Sin embargo, el Modem 1000 está diseñado para operar con una amplia variedad de ordenadores y, en consecuencia, no lleva ninguna ROM específica en su placa.

Los controles externos son idénticos a los suministrados en el VTX 5000. De modo que hay un interruptor Line y un LED, y un interruptor de modalidad de tres vías. La parte trasera del dispositivo contiene una puerta Data In que proporciona la interface para el ordenador, así como un enchufe telefónico. Puesto que el Modem 1000 no se alimenta desde el ordenador, se ha dotado al dispositivo de su propio cable y transformador de potencia.

El sistema de circuitos de transmisión en serie y chip ACIA es casi idéntico al utilizado en el VTX 5000. Sin embargo, para que el usuario se comunique a través del sistema se requiere algo de software. Para el BBC Micro, éste se proporciona en ROM, que se puede instalar en uno de los conectores de ROM laterales libres dentro del ordenador. Esto proporciona facilidades muy similares a las del Spectrum. Programas activados por menú permiten cargar programas, conectar con Micronet y recibir y transmitir mensajes.

Por la propia naturaleza del Commodore 64, la interface requerida para el Modem 1000 es más compleja que aquella para el VTX 5000. La misma se proporciona mediante una placa de interface co-

nectable que se instala en la puerta para cartuchos del ordenador. Esta interface no sólo contiene el software que se paginará en la memoria en el lugar de la ROM de BASIC del ordenador, sino también un sistema de circuitos adaptador especial que programa el conjunto ASCII Commodore según el estándar de comunicaciones utilizado generalmente. Otra parte de la placa produce una señal RS232 estándar que se le puede enviar a la puerta de datos del modem.

MODEM 1000
PRECIO
£69,95, excluyendo software e interface
PARA USAR CON
Todos los micros personales
VELOCIDAD DE TRANSMISION
75/1200 baudios

Communicator 104

Fabricado por KDS Electronics, este modem está diseñado para usar sólo con la gama de máquinas Amstrad CPC. Al igual que VTX 5000, se enchufa directamente en el bus de ampliación de la parte posterior del ordenador. Pero en este caso la potencia se suministra a través de un transformador externo instalado en el conector de potencia. Asimismo, al igual que los otros modems que hemos analizado, éste se conecta en serie con el teléfono. En consecuencia, para ponerse en línea con un tablón de anuncios, se debe primero marcar el número en el teléfono y después encender el modem.

El Communicator se diferencia de los otros modems de que hablamos aquí en que posee en el frente dos visualizaciones en siete segmentos que le proporcionan información tal como el número que se está discando actualmente, y si la ROM del modem está conectada en el sistema Amstrad.

El software para el Communicator está instalado en ROM dentro del modem propiamente dicho. Al igual que la gama OEL, los programas son activados por menú, si bien la cantidad de opciones disponibles es muchísimo mayor. El menú Mode Select (selección de modalidad) le permite seleccionar el tipo de sistema que desea utilizar: Prestel o tablón de anuncios, por ejemplo. Una vez seleccionada la modalidad, el modem ajustará automáticamente la velocidad de transmisión, si bien usted puede elegirla por sí mismo. El software activado por menú también soporta facilidades para llamada y respuesta automática y detección de portadora.

El Communicator soporta una vasta cantidad de útiles comandos residentes de ampliación del sistema. Éstos incluyen, entre otras cosas, comandos para entrar la ROM del modem, establecer la velocidad de transmisión y opciones de palabra de datos, y configurar la impresora para que proporcione una salida impresa de la sesión. El principal inconveniente del modem es que no se puede conectar el teléfono normal mientras se está utilizando. Por tanto, es necesario enchufar el modem o teléfono en el conector de pared con el fin de seleccionar la pieza del equipo que desea utilizar.

Crispin Thomas

WS3000

Comercializado por Miracle Technology, esta gama de modems se halla en el extremo superior del espectro de microordenadores, conteniendo la gama completa de facilidades que requieren la mayoría de los usuarios.

Hay tres modelos de la gama WS3000 que cubren una amplia variedad de velocidades de transmisión y protocolos, desde 75/1200 bps (bits por segundo) half duplex (para acceder al Prestel) hasta 1200/1200 bps transmisión full duplex para transmisiones de usuario a usuario. En el frente del modem hay varias luces indicadoras, que informan al usuario del estado actual del dispositivo. Éstas incluyen la operación de las patillas en la puerta RS232 y si el terminal está o no en línea. Cada uno de los tres modelos contiene utilidades que soportan llamada automática, respuesta automática y una puerta RS232 con buffer.

Además, la gama WS3000 soporta el sistema de protocolos de estándar Hayes. Éste es particularmente útil para los usuarios de gestión, dado que es creciente la cantidad de paquetes de gestión que incluyen facilidades que permiten enviar datos a través de un modem directamente desde el programa (la mayor parte de estos paquetes emplean protocolos Hayes).

Alien

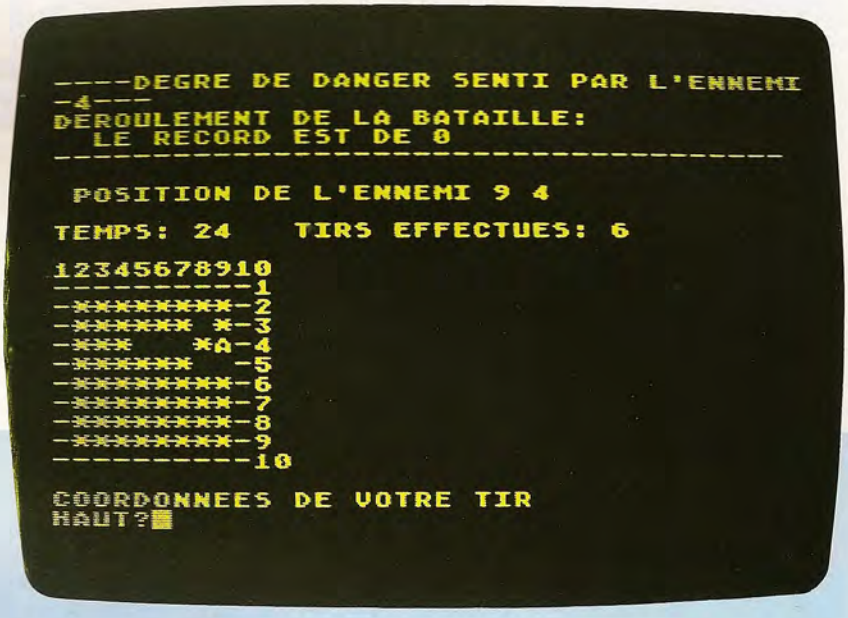
Usted es el comandante de una nave de combate y se interna en una galaxia desconocida... Es el comienzo de una gran aventura a los mandos de su ordenador Atari

Su misión es destruir a los enemigos que le rodean y le impiden aterrizar. La única manera de neutralizar al enemigo es destruir todos los espacios que lo rodean.

Usted solamente puede evolucionar dentro del campo de batalla. Los enclaves prohibidos están representados por signos "–". Si aterriza directamente sobre el enemigo (representado por la letra A), usted será destruido. El adversario se puede desplazar una o dos casillas a la vez, pero puede también permanecer inmóvil. Usted debe impartir sus instrucciones en la forma de dos coordenadas. No olvide pulsar la tecla Return después de cada una de ellas.

Cada posición destruida aparece como una casilla en blanco; las otras están representadas por los signos "*".

El enemigo observa el desarrollo de la batalla y le informa del grado de peligro que percibe. La mejor manera de combatirlo es lograr bloquearlo en un rincón, porque esto limita considerablemente sus posibilidades de desplazamiento. El adversario recela de esta estrategia y, utilizando la "inteligencia" en las líneas 6145 y 6161, intentará evitar los cuatro rincones del campo de batalla. Trate de rodear al enemigo de la forma más rápida posible con el fin de limitar su campo de acción. ¡Hay un récord que usted puede batir!




```
  1 GRAPHICS 0:SETCOLOR 4,8,0:SETCOLOR 2,8,0
  3 DIM A(10,10)
  5 HISCORE=0
 10 REM ALIEN
 20 REM (C)HARTNELL 1982
 30 GOSUB 9000:REM INICIALIZACION
 40 GOSUB 8000:REM LETREROS
 50 GOSUB 7000:REM MOVIMIENTO JUGADOR
 60 GOSUB 6000:REM MODIF ENEMIGO
 70 TEMPS=TEMPS-1
 75 TIRS=TIRS+1
 80 IF TEMPS=0 THEN 6570
 85 GOSUB 8000
 90 GOTO 50
910 TEMPS=30
5000 REM COLISION
5010 PRINT "HA TOCADO AL CAPITAN ENEMIGO"
5020 PRINT "Y LO HA DESTRUIDO"
5030 GOTO 6570
6000 REM MODIF ENNEMI
6010 REM TEST SI ENCERCLE
6020 H=0
6030 IF A(M-1,N)=2 THEN H=H+1
6040 IF A(M-1,N-1)=2 THEN H=H+1
6050 IF A(M,N-1)=2 THEN H=H+1
6060 IF A(M,N+1)=2 THEN H=H+1
6070 IF A(M-1,N+1)=2 THEN H=H+1
6080 IF A(M+1,N+1)=2 THEN H=H+1
6090 IF A(M+1,N-1)=2 THEN H=H+1
6100 IF A(M+1,N)=2 THEN H=H+1
6110 IF H=8 THEN 6500:REM CERCO
6120 REM DESPLAZAMIENTO DEL ENEMIGO
6125 E=M:F=N
6130 M=M-INT(RND(1)*2)+INT(RND)1)*2)
6140 IF M<2 OR M>9 THEN 6130
6145 IF (M<4 OR M>7) AND RND(1)>0.7 THEN 6130
6150 N=N-INT(RND(1)*2)+INT(RND(1)*2)
6160 IF N<2 OR N>9 THEN 6150
6161 IF (N<4 OR N>7) AND RND(1)>0.7 THEN 6150
6162 IF A(M,N)=2 THEN 6130
6165 A(E,F)=0
6170 A(M,N)=1
6300 RETURN
6500 REM CERCO
6505 GOSUB 8000
6510 PRINT "LO HA CONSEGUIDO! BRAVO"
6520 PRINT "HA FALLADO "; TIRS; "TIRS"
6530 PRINT "Y LO HA HECHO EN"; TEMPS, "MINUTOS", "LEFT"
6540 Q=TEMPS*125.67
6550 PRINT "SU SCORE      ";Q
6560 IF Q>RECORD THEN RECORD=Q
6570 PRINT "EL RECORD ES ";RECORD
6580 PRINT
6590 PRINT "PULSE 1 PARA VOLVER A JUGAR, 2 PARA
     DETENERSE"
6600 INPUT A
6610 IF A=1 THEN 10
6620 PRINT "A LA PROXIMA CAPITAN"
6630 END
7000 REM MOVIMIENTO JUGADOR
7010 PRINT "COORDENADAS DE SU DISPARO"
7020 PRINT "VERTICAL.";
7030 TRAP 7030:INPUT S:TRAP 40000
7035 IF S<2 OR S>9 THEN 7030
7040 PRINT "HORIZONTAL.";
7050 TRAP 7050:INPUT R:TRAP 40000
7055 IF R<2 OR R>9 THEN 7050
7066 IF A(R,S)=1 THEN 5000:REM DESTRUCCION DEL ENEMIGO,
     FIN DE LA PARTIDA
7070 IF A(R,S)=2 THEN PRINT "SECTOR YA DESTRUIDO":FOR
     P=200 TO 255:SOUND 0,P,10,15:NEXT P:SOUND 0,0,0,0
7090 IF A(R,S)=2 THEN RETURN
7100 A(R,S)=2
7110 RETURN
8000 REM LETREROS
8005 PRINT CHR$(125)
8020 PRINT
8030 "PRINT GRADO DE PELIGRO DETECTADO POR EL
     ENEMIGO-";H;"----"
8050 PRINT "DESARROLLO DE LA BATALLA:",, "EL RECORD ES DE
     ": RECORD
8060 PRINT " ----------------------------------------
     ----------"
8080 PRINT
8090 PRINT "POSICION DEL ENEMIGO ";N;" ";M
8100 PRINT
8110 PRINT "TEMPS: ";TEMPS;" DISPAROS EFECTUADOS: ";TIRS
8120 PRINT
8125 PRINT "12345678910"
8130 FOR K=1 TO 10
8140 FOR J=1 TO 10
8145 IF K<2 OR K>9 OR J<2 OR J>9 THEN PRINT "-";
8146 IF K<2 OR K>9 OR J<2 OR J>9 THEN 8180
8150 IF A(K,J)=0 THEN PRINT " ";:SOUND 0,*8+J*4,10,15
8160 IF A(K,J)=1 THEN PRINT "A";:FOR P=20 TO 0 STEP -1:SOUND
     0,P,12,15:NEXT P
8170 IF A(K,J)=2 THEN PRINT " ";:FOR P=4 0 TO 0 STEP -2:SOUND
     0,P,8,15:NEXT P
8180 SOUND 0,0,0,0:NEXT J
8190 PRINT K
8200 NEXT K
8210 PRINT
8990 RETURN
9000 REM INICIALIZACION
9010 TEMPS=30
9020 TIRS=0
9030 H=0
9050 FOR B=1 TO 10
9055 FOR C=1 TO 10
9060 A(B,C)=0
9065 IF B<2 OR B>9 OR C<2 OR C>9 THEN A(B,C)=2
9067 NEXT C
9070 NEXT B
9080 M=INT(RND(1)*7)+2
9090 N=INT(RND(1)*7)+2
9100 A(M,N)=1
9900 RETURN
```

© SYBEX

Frédéric Voisin

Dibujo generado por ordenador
Esta salida impresa está tomada del ImageWriter. El dibujo fue encargado por la casa discográfica C.S.A. para ilustrar la cubierta de un disco. Revela algunas de las gamas de texturas y la gran resolución que se pueden obtener con el *MacPaint*

El potencial artístico del Macintosh se ha plasmado brillantemente en la obra del pintor francés Frédéric Voisin

Magia y color

El *MacPaint*, como todo el software para el Macintosh, está basado en WIMP. Una vez cargado el disco, el escritorio ofrece diversos estilos de dibujo. El usuario puede elegir un pincel de espesores variables, un lápiz, un bote de pintura para rellenar superficies definidas, y un serógrafo para crear un efecto de punteado. Además de estas facilidades de mano alzada, *MacPaint* ofrece la opción de generar líneas rectas, cuadrados, círculos y otras formas geométricas. Cualquier área cerrada se puede rellenar con uno de 38 patrones y sombreados, incluyendo puntos, líneas oblicuas, trabajo de reticulado y cuadros. Los errores se pueden borrar escogiendo el icono de la goma, y una parte de la imagen se puede "enlazar" y trasladar. El menú Goodies (golosinas) ofrece el potencial para otros refinamientos; Fat Bits, por ejemplo, aumenta una superficie en pixels individuales, permitiendo que el usuario suprima o añada diminutos detalles.

Al igual que todo el software Macintosh, *MacPaint* se controla mediante el cursor activado por ratón, que hace innecesario el teclado. Junto con el *MacPaint* también se pueden utilizar accesorios tales como MacTablet, una pequeña tablilla para gráficos con un lápiz óptico.

El artista francés Frédéric Voisin se ha convertido en un entusiasta del *MacPaint*. Hace veinte meses sólo empleaba los útiles tradicionales de su actividad: lápices, lapicero, tinta y papel; ahora utiliza al Macintosh. Su primer contacto con un ordenador lo tuvo con el Apple Lisa de un amigo suyo. Con sólo dibujar una línea con el Lisa comprendió su potencial para generar gráficos. Eso también lo hizo pensar en producir pinturas a gran escala en lugar de ilustraciones. Esta idea la concibió a partir de dos características del Lisa: primero, el hecho de que la imagen en pantalla fuera en blanco y negro y que la forma obvia de añadirle color fuera pintar sobre la salida impresa; segundo, aunque la imagen era pequeña, la gran resolución de la pantalla le sugirió de inmediato las posibilidades de ampliación.

Voisin no perdió tiempo y obtuvo de Apple un Macintosh con el fin de seguir experimentando. Irónicamente, fue un ordenador lo que llevó a Voisin a volcar su interés en la tradición clásica de la pintura. Es decir, hacer un bosquejo sobre el lienzo y pintar sobre el mismo. Pero en vez de utilizar un lápiz para dibujar el boceto, Voisin empieza con

Claudia Zeff

una salida impresa de su ImageWriter, la fotocopia y la amplía hasta tres metros de altura utilizando una máquina copiadora de arquitecto. La siguiente parte del proceso es igualmente tradicional. Voisin pega la copia sobre el lienzo empleandó una cola elaborada con huesos de conejo, mezcla que vienen empleando los artistas desde que comenzaron a pintar sobre lienzo. La cola posee tal adaptabilidad que permite estirar uniformemente un gran trozo de papel sobre un lienzo. Después pinta sobre la copia con pinturas acrílicas fluorescentes para crear vibrantes pinturas posmodernistas de robots, guerreros zulúes y bailarines de *break-dance*. Le gusta exhibir sus lienzos bajo luz ultravioleta, que se combina con la pintura fluorescente para crear un efecto centelleante y pulsátil similar al de una pantalla VDU en color.

En el transcurso del último año Voisin produjo 30 pinturas en su Macintosh de 128 Kbytes utilizando el *MacPaint*, aunque afirma que aún le queda por descubrir el verdadero potencial del programa. Siente un enorme respeto por Bill Atkinson, su diseñador, quien, según cree Voisin, "se ha puesto dentro de la piel de los ilustradores", anticipándose a sus necesidades. La versión del *MacPaint* que emplea Voisin en realidad es bastante antigua, careciendo de muchos de los refinamientos de las versiones más recientes.

Las líneas de la portada
Ésta es la ilustración original utilizada en la portada de este número de *Mi Computer*. La combinación de la línea compacta y el negro sólido de los rostros con los tonos más suaves del humo ejemplifica la gama de estilos que ofrece *MacPaint*

Arte generado por ordenador

A diferencia de muchos de sus compañeros de profesión, a Voisin no lo intimida la idea de usar un ordenador; lo ve como una herramienta a explorar. Para él, el ratón es como un lápiz: un instrumento que uno utiliza con el cerebro y con la mano. Voisin piensa que finalmente el arte generado por ordenador alcanzará el mismo estatus que el trabajo realizado en un medio más tradicional, como los grabados; la única diferencia residiría en que el original está almacenado en un disco flexible en lugar de estar grabado en una placa de cobre. Voisin cree que esto llevará a una nueva generación de "artistas de masas", que producirán arte asequible a un gran público en lugar de a unos pocos coleccionistas de arte. El Ministerio de Cultura francés ha puesto a su disposición un Radiance, un ordenador de gran capacidad con una impresora en color. Con el mismo ha producido ediciones limitadas de salidas impresas en color. Voisin ansía probar el Macintosh en color, si bien sólo lo utilizará para ilustraciones y no para producir grandes lienzos.

Voisin aún ha de experimentar con diversos accesorios y software, como el *MacDraw*, disponibles para el Macintosh. Por otra parte, afirma que no podría trabajar sin su Macintosh. "Lo necesito como si se tratara del teléfono o un pincel y pintura. El Macintosh es grandioso. Es una revolución."

Ediciones "radiantes"
Vemos aquí instantáneas de creaciones de Voisin tomadas del ordenador Radiance. La alta resolución y la facilidad multicolor de la máquina la convierten en un instrumento ideal para crear ilustraciones de calidad. El Ministerio de Cultura francés ha puesto el ordenador a disposición de Voisin para que pueda producir ediciones de salidas impresas en color de su obra

Frédéric Voisin

Cantidad y calidad

En c es posible mejorar y aumentar los tipos de datos definidos por el usuario. Veamos de qué manera

Como lenguaje extensible, el c se puede ampliar para que incluya las propias bibliotecas de definiciones de usted mismo. Ya hemos visto cómo se puede hacer esto con las funciones definidas por el usuario, que simplemente se mantienen en su propio archivo y se accede a ellas utilizando un include en el programa. Asimismo, usted puede aumentar los tipos de datos estándares definidos por el usuario, como int, char, etc. La primera forma en que puede hacerlo es empleando typedef, que permite dar nuevos nombres a tipos estándares. De modo que:

```
typedef int metros;
typedef double vector [10];
typedef char *serie;
```

permitiría declarar variables más adelante en el programa.

Parecería que esto no tiene mayor sentido, pero puede servir para dos cosas. Primero, puede hacer más comprensibles los programas, lo cual, tratándose del c, siempre resulta positivo. En segundo lugar, puede ayudar a superar dificultades para el traslado de programas entre distintas máquinas. La cantidad de bytes que ocupa un int, por ejemplo, no está estandarizada. Quizá diera muchísimo trabajo transportar un programa que utilizara el hecho de que int tuviera cuatro bytes de longitud a una máquina que sólo usara dos bytes.

Es mucho más fácil cambiar un typedef al comienzo del programa en lugar de buscarlo a través de todas las ocurrencias en las que la diferencia fuera significativa. Sería normal querer conservar todos los typedef juntos en un archivo include separado.

El c posee características semejantes al PASCAL en cuanto a la provisión de una estructura de totalización para combinar elementos de datos relacionados de tipos distintos. La instrucción del c struct ofrece una estructura similar a la de RECORD en PASCAL. He aquí un ejemplo de su uso:

```
struct naipe
{
  int número;
  char palo;
}
```

Aquí se define una estructura compuesta por dos partes (un número entero y un palo de caracteres) que van juntas para conformar un naipe de la baraja. Las variables de este tipo se pueden declarar en este y ulteriores puntos del programa. Así:

```
struct naipe baraja [52];
```

formaría una baraja de naipes. La variable se puede declarar junto con la definición de estructura, como en:

```
struct naipe
{
  int número;
  char palo;
}baraja[52];
```

El nombre del rótulo (naipe, en este caso) se puede omitir si se han de declarar todas las variables en un lugar, como en:

```
struct
{
  int número;
  char palo;
}baraja [52], manos [4] [13],* jugadas;
```

pero esto se suele considerar una mala práctica.

Se pueden asignar estructuras completas, pasarlas como argumentos a una función o devolverlas como valores por una función, del mismo modo que cualquiera de las funciones estándares, tales como:

```
naipe [1] [5]=baraja [27];
```

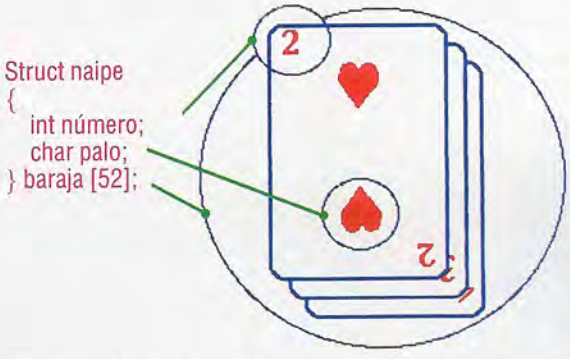
o se puede aludir a los elementos individuales utilizando un punto para una variable normal, y –> para una variable puntero de este tipo, como en:

```
jugadas->número=baraja [34].número;
```

Estos elementos individuales se comportan como variables comunes del tipo adecuado. Una estructura se puede inicializar del mismo modo que una matriz con el nombre de la variable seguido por una lista de valores para los elementos de la estructura encerrados entre llaves.

Las estructuras reservan palabras de almacenamiento consecutivas para sus elementos; no obstante, unions utiliza el mismo almacenamiento para sus elementos, lo que permite ver la memoria de más de una manera. Por ejemplo, si una máquina usa cuatro bytes para un int, si quisiéramos aludir a estos bytes individualmente, podríamos definir un union que utilizara los mismos cuatro bytes de almacenamiento para un int o cuatro chars:

```
union int_o_char
{
  int parte_int;
  char parte_char [4];
}
```



Caroline Clayton

Estructura flexible
El tipo struct del c permite que el programador combine distintos tipos de datos dentro de una estructura. Por ejemplo, en una única estructura declarada como podemos apreciar en la ilustración se puede representar una baraja de naipes por palo y por número de naipe

A los elementos de un union se alude del mismo modo que a los elementos de una estructura.

En lenguaje normal, se podría aludir a los elementos individuales de una estructura como *campos*; sin embargo, el c utiliza este término para referenciar un tipo determinado de elemento que incluye un tamaño de bit. Ya hemos visto que el c posee operadores a nivel de bit, y los campos nos confieren la facilidad adicional de aludir por nombre a cualquier grupo de uno o más bits en una palabra. Esto se realiza colocando un signo de dos puntos tras el nombre del elemento, seguido por el tamaño de bits. Por ejemplo, en una máquina con una palabra de 16 bits y un tipo int de 16 bits, podríamos definir:

```
structpalabra_en_bytes
{
   unsigned byte0:8,byte1:8;
}
structpalabra_en_bits
{
   unsigned bit0:1, bit1:1, bit2:1, bit3:1, bit4:1,
   bit5:1, bit6:1, bit7:1, bit8:1, bit9:1, bit10:1,
   bit11:1, bit12:1, bit13:1, bit14:1, bit15:1;
}
unión palabra
{
   int w;
   struct palabra_en_bytes w8;
   struct palabra_en_bits w1;
}
```

Esto nos permite aludir a la misma palabra de almacenamiento como un todo, como dos bytes o como 16 bits. Observe el empleo del tipo de datos unsigned (sin signo), que equivale a int con la excepción de que sólo permite valores positivos. Por supuesto, es importante recordar que cuando se especifica de este modo el tamaño, sólo se deben asignar valores en un rango adecuado. A un elemento con un tamaño de un solo bit se le pueden asignar legítimamente los dos valores 0 y 1.

Una mano de bridge

El siguiente ejemplo ilustra algunas funciones que se pueden utilizar para un programa que juegue una mano de bridge. La idea básica del juego (al menos por cuanto nos concierne aquí) es que se mezcle la baraja completa y se reparta entre cuatro jugadores. Una característica de la mano que permite al jugador decidir cómo jugar es el contador de puntos. Éste cuenta cuatro puntos para un As, tres para un Rey, dos para una Dama y uno para un Valet. Aquí las funciones simulan la baraja, el reparto de naipes y los puntos de cuenta. La estructura naipe y la matriz de naipes que componen la baraja se definen e inicializan de modo externo, de manera de poder inicializar la baraja.

```
struct naipe
{
   int número;
   char palo;
}
/* utilizando un espacio mínimo para cada
naipe*/
/* declarar e inicializar baraja*/
struct naipe mano [4] [13], baraja [52]=
{{1,'T'}, {2,'T'}, {3,'T'}, {4,'T'}, {5,'T'},
{6,'T'}, {7,'T'}, {8,'T'}, {9,'T'}, {10,'T'},
{11,'T'}, {12,'T'}, {13,'T'},
{1,'D'}, {2,'D'}, {3,'D'}, {4,'D'}, {5,'D'},
{6,'D'}, {7,'D'}, {8,'D'}, {9,'D'}, {10,'D'},
{11,'D'}, {12,'D'}, {13,'D'},
{1,'C'}, {2,'C'}, {3,'C'}, {4,'C'}, {5,'C'},
{6,'C'}, {7,'C'}, {8,'C'}, {9,'C'}, {10,'C'},
{11,'C'}, {12,'C'}, {13,'C'},
{1,'P'}, {2,'P'}, {3,'P'}, {4,'P'}, {5,'P'},
{6,'P'}, {7,'P'}, {8,'P'}, {9,'P'}, {10,'P'},
{11,'P'}, {12,'P'}, {13,'P'}}
barajar (baraja)
struct naipe baraja [];
/*Asumimos la existencia de un generador de
números aleatorios (random (n)) para devolver
un número aleatorio entre 0 y n-1. Se puede
utilizar la función del sistema rand () que
devuelve un int aleatorio*/
/*Los naipes se barajan intercambiando cada
naipe con algún otro seleccionado al azar*/
{
   int i,j;
   for (i=0;i<52;++i)
   {
      j=random(52);
      swap(&baraja[i],&baraja[j]);
   }
}
swap(p,q)
struct naipe*p, *q;
{
   struct naipe temp;
   temp = *p;
   *p = *q;
   *q= temp;
}
repartir(baraja,mano)
struct naipe baraja[], mano[][];
{
   int i,j,k=0;
   for (i=0;i<4;++i)
      for (j=0;j<13;++j)
         mano [i][j]=baraja[k++];
}
contar_puntos(una_mano)
struct naipe *una_mano;
/* una_mano es un puntero al primer naipe
de un conjunto de 13 en una mano, como
&mano[2][0]*/
{
   int i, contador_puntos=0;
   for (i=0;i<13;++i)
   {
      if(una_mano—> número=1)
         contador_puntos+=4;
      else if (una_mano—> número > 10)
         contador_puntos+=una_
         mano—>número — 10;
         ++una_mano;
/* recuerde que éste ahora apuntará al
siguiente naipe de la baraja*/
   }
   return(contador_puntos);
}
```

Ahorrar espacio

Iniciamos una serie dedicada a analizar las ventajas de las técnicas de compresión de datos

Las técnicas de compresión de textos se pueden utilizar para muchas aplicaciones diferentes, el ejemplo más evidente de las cuales es el de los usuarios de micros personales que deseen programar grandes juegos de aventuras. Pero piense en el ahorro de tiempo y la reducción de las facturas del teléfono que podrían obtener las empresas si comprimieran sus documentos antes de transmitirlos de una a otra oficina. El efecto de las técnicas de compresión eficaces sobre discos flexibles o cartuchos microdrive puede ser igualmente drástico. Es posible conseguir reducciones de volumen de entre el 50 y 60% en archivos de texto moderadamente grandes.

Existen tres técnicas básicas para comprimir archivos. Aquí las describiremos por separado, pero en la práctica suelen utilizarse conjuntamente para obtener la máxima compresión. El primer método, y el más simple, consiste en emplear un juego de caracteres reducido. En la mayoría de los micros, un carácter se almacena en un byte porque éste es el tamaño que le resulta más fácil de manipular al procesador. Esto da ocho bits y, por consiguiente, 256 caracteres posibles. Cuando usted considera que sólo se necesitan 96 caracteres para almacenar los alfabetos incluyendo mayúsculas y minúsculas, números y todos los diversos signos de puntuación del juego ASCII, 256 parece exagerado.

La cantidad mínima de bits enteros necesarios para almacenar 95 caracteres es siete, de modo que codificar la información en siete bits en vez de en ocho reduciría el volumen total en un 12,5%. La solución obvia es emplear menos caracteres. Algunos, como +, *, < y >, por lo general no se utilizan en archivos de documentos y se pueden descartar con razonable seguridad. No obstante, aun el hacer esto no supondrá una diferencia considerable a menos que la aplicación involucrada pueda arreglárselas sin la totalidad del juego de minúsculas o soportar alguna medida igualmente drástica. En consecuencia, el método del juego de caracteres reducido tiene un atractivo limitado. No obstante, se lo emplea para comunicaciones de telex en donde se introduce una dimensión extra. Es decir, utilizar dos caracteres como caracteres de cambio, como la tecla shift de un teclado, para cambiar entre un juego de caracteres alfabéticos en mayúscula y un juego de numerales y signos de puntuación. Ello significa que los caracteres requieren sólo cinco bits cada uno, dando una interesante reducción del 37,5%. En la práctica este porcentaje se reduce por la presencia de caracteres de cambio.

Podemos comprimir el texto agrandando el juego de caracteres. Éste es el segundo procedimiento e implica utilizar caracteres de recambio no empleados por el juego ASCII a modo de "distintivos". Siempre que se encuentre uno de estos valores en el archivo comprimido, se lo utiliza para buscar un valor en una tabla, que contiene una lista de palabras comunes, partes de palabras o frases, tales como "los" o "Uds", o, en documentos financieros, "fiscal", etc. Si estos distintivos se eligen con cuidado, se pueden producir grandes reducciones, si bien al costo de tener que almacenar una tabla de distintivos relativamente larga. La mayoría de los micros, por ejemplo, almacenan programas en BASIC en forma distintivada, con todas las palabras clave almacenadas como bytes individuales.

Se pueden apreciar las ventajas de la entrada distintivada si su máquina le permite colocar una versión no distintivada de un programa en disco o cinta, ya sea directamente (como el Amstrad) o bien LISTando un archivo (como en el Spectrum); la diferencia en tamaño puede resultar sorprendente. Aunque los distintivos para programas en BASIC son, evidentemente, fijos, de hecho los mejores programas distintivadores para grandes documentos buscarán en el archivo a comprimir, hallarán los distintivos óptimos, y almacenarán la tabla de distintivos junto con el texto comprimido.

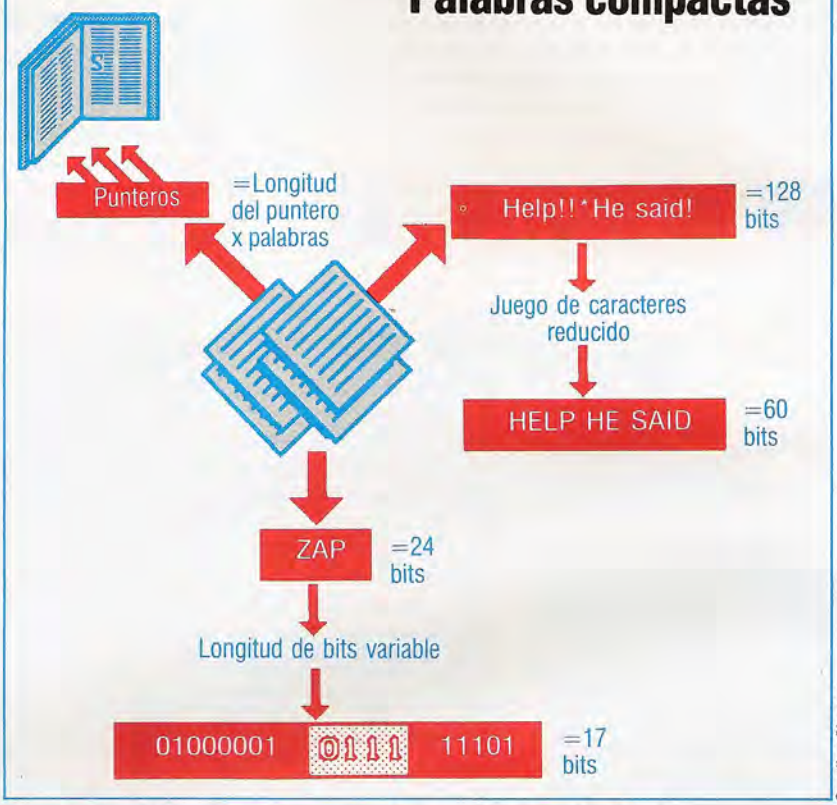
Este método funciona solamente porque algunas palabras y frases se utilizan con mucha frecuencia, y se puede ampliar para que comprenda caracteres

Tabla de frecuencias

Muchos programas de compresión de textos se basan en las frecuencias de caracteres para una operación eficaz. En inglés, el orden de frecuencia de los caracteres alfabéticos es éste:

ETAONRISHDLFCMUGYPWBVKXJQZ

Palabras compactas



Caroline Clayton

Encaje justo

La redundancia incorporada del juego de caracteres ASCII con frecuencia requiere alguna forma de compactación de datos, ya sea para maximizar espacio de almacenamiento o bien para reducir los tiempos de transmisión de datos. Existen tres métodos principales de compresión, que normalmente se combinan para conseguir los mejores resultados. Los diccionarios almacenan palabras de uso común, permitiendo construir un archivo de textos como una serie de punteros. Un juego de caracteres reducido elimina los caracteres redundantes (y con frecuencia también las mayúsculas o bien las minúsculas). Por último, la codificación de "bits variables" (o codificación Hoffman) asigna patrones de bits de longitud reducida a las letras que se producen con mayor frecuencia.

En castellano la frecuencia de caracteres difiere del inglés, ordenándose éstos de la siguiente manera: E, A, I, S, U, O, R, N, T, D, L, C, M, Q, P, B, F, G, H, V, Y, J, X, Z, Ñ, K, W

Compresión ingeniosa

Con el fin de reducir la cantidad de espacio que ocupan los caracteres, podemos codificarlos utilizando una técnica que se conoce como *codificación Hoffman*. El primer paso consiste en calcular la frecuencia relativa de los caracteres implicados; de modo que si quisiéramos comprimir la serie "ASSB", las frecuencias relativas de las tres letras involucradas serían A:1, S:2 y B:1. El siguiente paso es codificar cada carácter, utilizando menos bits para los caracteres más comunes. Luego los códigos se colocan en serie y cuando se requiere codificación el ordenador comienza en el principio del archivo y lo va leyendo, sustituyendo las letras correctas para cada código. Pero se presenta un problema. ¿Cómo sabemos cuándo termina un código y empieza otro? Por ejemplo, si A está codificada como 1, B como 0 y C como 10, ¿cómo podemos decir, cuando detectamos una secuencia de 10, si la misma significa C o AB?

La respuesta reside en codificar cada letra de modo que, leyéndola de izquierda a derecha, podamos estar seguros de que el primer código completo detectado (sea cual fuere su longitud) es el que deseamos. Retomando nuevamente el ejemplo de A, B y C, necesitaríamos codificarlas como 1, 00 y 01 respectivamente.

El programa, que se ejecutará en la mayoría de los micros con ligeras modificaciones o bien ninguna (los usuarios del Spectrum deberán alterar las sentencias de dimensionado de series), hace todo el trabajo pesado por usted. Puede verlo en acción mientras va leyendo los datos proporcionados por defecto, o bien puede proporcionarle los propios caracteres y frecuencias de usted.

Experimente con distintos valores: verá que las reducciones de bits más importantes se producirán cuando las frecuencias relativas muestren grandes fluctuaciones (como en la sentencia de datos proporcionada)

```
10 INPUT "Quieres proporcionar tus propios datos?
   (s/n) ",i$
20 IF i$<> "s" AND i$<>"n" THEN GOTO 10
30 IF i$="n" GOTO 130
40 INPUT "Numero de caracteres a codificar ",n
50 IF N> 255 THEN PRINT "Son demasiados caracteres... 255
   como maximo": GOTO 40
60 DIM c$(n), f(n),r (2*n-1), t(2*n-1)
70 FOR i=1 TO n
80 PRINT "Entra numero de caracter ";i: INPUT s$
90 IF LEN(s$)> 1 THEN PRINT "Solo un caracter por
   vez...":GOTO 80
100 PRINT "Entra la frecuencia de numero de caracter ";i:INPUT
   f(i)
110 a$=a$+s$
120 NEXT i:GOTO 150
130 n=26: DIM c$(n), f(n), r(2*n-1), t(2*n-1)
140 FOR i=1 TO n:READ s$,f(i):a$=a$+s$:NEXT i
150 FOR i=1 TO n:r(i)=f(i): NEXT i
160 FOR i=n+1 TO 2* n-1
170 z=9999:v=9999:k=0:g=0
180 FOR q=1 TO i-1
190 IF t(q)<>0 THEN GOTO 220
200 IF r(q)< z THEN g=k:v=z:k=q:z=r(q):GOTO
   220
210 IF r(q)< v THEN g=q:v=r(q)
220 NEXT q
230 p=i
240 r(p)=z+v: t(k)=-p: t(g)=p
250 NEXT i
260 FOR i=1 TO n
270 c$(i)=""
280 p=i
290 IF t(p)=0 THEN GOTO 340
300 IF t(p)> 0 THEN c$(i)="0"+c$(i): GOTO 320
310 c$(i)="1"+c$(i)
320 p=ABS(t(p))
330 GOTO 290
340 NEXT i
350 FOR i=1 TO n
360 PRINT MID$(a$,i,l),r(i),c$(i)
370 NEXT i
380 END
390 DATA e, 250, t, 220, a, 24, o, 23, n, 22, r, 21
400 DATA i, 20, s, 19, h, 18, d, 17, l, 16, f, 15
410 DATA c, 14, m, 13, u, 12, g, 11, y, 10, p, 9
420 DATA w, 8, b, 7, v, 6, k, 5, x, 4, j, 3, q, 2, z, 1
```

individuales. Este enfoque constituye la base de la tercera técnica de compresión. En los dos procedimientos anteriores, la cantidad de bits utilizada para cada carácter o distintivo era fija. No obstante, la compresión por longitud de bits variables (también conocida como *codificación Hoffman*), como su nombre implica requiere codificar los caracteres más comunes en menos bits que los caracteres menos comunes.

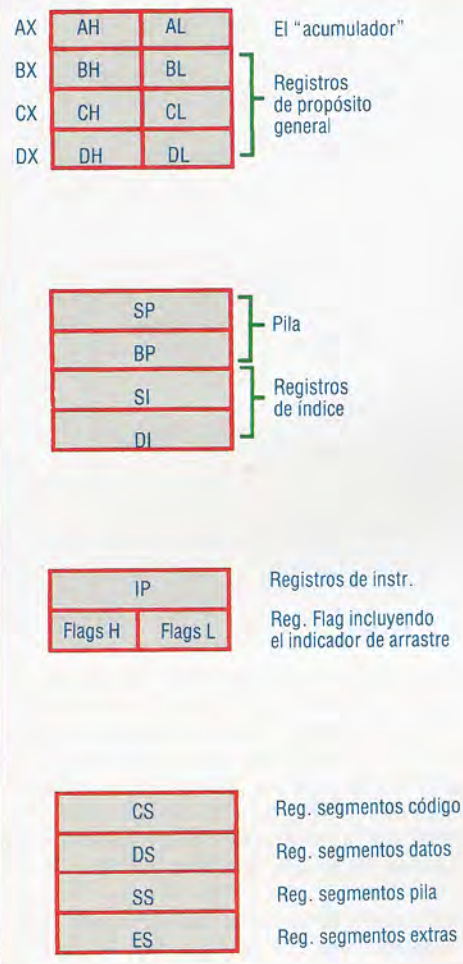
Este enfoque no es nuevo: Samuel Morse lo utilizó, por ejemplo, para su código telegráfico de "puntos y rayas". Sin embargo, Morse tuvo la ventaja de contar con "caracteres" extras en la forma de vacíos en la transmisión, que utilizó para delimitar caracteres y palabras. Aun sin disfrutar de esta ventaja, Hoffman aportó un sistema que se basaba en el hecho de que ninguno de sus caracteres se podía componer colocando accidentalmente otros dos caracteres distintos consecutivos.

Sin embargo, esta restricción significa que la mayoría de los caracteres poco comunes pueden requerir el almacenamiento de 17 bits o más. Con el fin de reducir la cantidad máxima de bits, y para añadir la capacidad de incluir también algunos distintivos, se ha desarrollado un método refinado que (aunque de longitud variable) utiliza solamente secuencias de palabras de cuatro bits. Una palabra de cuatro bits se puede usar para almacenar 16 patrones diferentes, de los cuales tres se emplean como patrones de señal para dar información sobre los cuatro elementos de bits siguientes. Los otros 13 se utilizan para los 11 caracteres más usados, junto con el carácter de espacio y el carácter de salto de línea.

Los tres valores de señal se emplean para denotar ya sea que la siguiente palabra de cuatro bits se ha de utilizar como un valor de carácter de una de las 16 palabras más comunes, o bien que los ocho bits siguientes se deben utilizar para denotar uno de los caracteres menos usados o una palabra menos común. Este procedimiento permite el uso de los 96 caracteres completos, más un juego de 205 distintivos para palabras comunes. Se podrían incluir otros refinamientos en el supuesto de que todas las palabras del diccionario comenzaran con un espacio (a menos que se las coloque al comienzo de una línea), ahorrando de este modo cuatro bits cada vez que se utilice una de ellas. Estas características en combinación dan excelentes resultados, con una reducción que suele superar el 50%.

Llamadas a función

Concluiremos esta breve serie dedicada a estudiar el MS-DOS analizando este sistema operativo desde el punto de vista del programador



El formato de desplazamiento

El procesador 8086/8088 puede direccionar hasta un megabyte de memoria. Esto requiere direcciones de 20 bits, y puesto que el 8088 se basa en registros de 16 bits, por consiguiente se necesitan bits extras para conformar una dirección. Los diseñadores del 8086/8088 sortearon este problema adoptando un segmento: formato de desplazamiento. El registro de segmentos direcciona un bloque de memoria de 64 Kbytes y el registro de desplazamiento identifica al byte dentro del bloque. Por ejemplo, la dirección de la instrucción actual está retenida en CS:IP. Los registros más importantes son A, B, C, D y el registro Flag. Se puede aludir a estos registros como una palabra de 16 bits completa utilizando el sufijo 'X' (por ejemplo, AX) o como bytes *low* y *high* utilizando los sufijos 'L' y 'H'. El byte *low* del registro D es, en consecuencia, DL

Caroline Clayton

Las muchas llamadas al sistema disponibles en MS-DOS por lo general se clasifican en seis categorías diferentes:

- E/S de caracteres desde y hacia los dispositivos estándares del sistema.
- Tratamiento de archivos (incluyendo la gestión del directorio).
- Gestión de memoria.
- Gestión de procesos.
- Llamadas misceláneas a "funciones" del sistema.
- Llamadas especiales a Microsoft Networks.

Todos estos servicios del sistema se pueden llamar desde cualquier aplicación, y proporcionan una interface unificada y exhaustiva que asegura la compatibilidad entre distintas versiones del OS. Las llamadas a direcciones absolutas o directas a dispositivos de hardware no serán necesariamente portables a otros sistemas MS-DOS o, incluso, compatibles con versiones DOS diferentes, ni siquiera siendo del mismo fabricante.

El empleo de la palabra *función* para describir todas las llamadas al sistema es terminología MS-DOS estándar, derivada del BCPL y del C a través del Unix. *No* significa que el servicio no sea más que una función que devuelva algún dato. Asimismo, incluye rutinas que verdaderamente hacen algo útil (como abrir archivos, etc.). Éstos, por supuesto, son procedimientos, pero en toda la literatura son funciones denominadas de una forma vaga.

Las funciones de fines generales más útiles son las llamadas *misceláneas*. Cada una se ve afectada por una interrupción de software y siempre implica la consecución de los siguientes pasos:

1. Trasladar los datos requeridos a los registros adecuados (en DL, p. ej., se podría requerir un número de unidad).
2. Colocar el número de función en AH (el byte *high* de AX).
3. Generar la interrupción de software 21H.

Todo dato devuelto por la función se hallará en los registros 8086 al retornar de la llamada, ya sea directamente o a través de un puntero o dirección. Por lo tanto, la función DOS 19H, por ejemplo (todos los números, de función son hexadecimales), devuelve la unidad de disco seleccionada actualmente como un código en el registro AL. No requiere ningún código de acción, de modo que se podría codificar como:

```
mov   ah,19H    ;tomar unidad seleccionada
int   21H       ;llamar a la función
```

El resultado se devuelve en el registro AL, y es un número que representa a la unidad (A=0, B=1, y así sucesivamente). Sumando a este resultado el código ASCII para el carácter 'A' y llamando a la función DOS 05H (imprimir el código de DL como un carácter ASCII), podríamos visualizar la unidad conectada como un carácter en mayúsculas:

```
add al, 'A'          ;convertir 0 en 'A', 1 en 'B', etc.
mov dl, al           ;trasladarlo al registro DL
mov ah,05H           ;imprimir carácter
int 21H              ;llamar a la función
```

Los códigos de instrucción empleados en el ejemplo anterior son para el propio ensamblador de Microsoft, pero no siempre el OEM se lo facilita al usuario final. Afortunadamente, no es esencial tener un ensamblador; como veremos, por lo general es muy fácil llamar al MS-DOS desde lenguajes de alto nivel. Sin embargo, si usted tiene intenciones de hacer muchísima programación en código máquina, asegúrese de que adquiere un ensamblador que genere archivos en código máquina reubicables en formato Intel (.OBJ) estándar. Éste es el estándar *de facto* para máquinas de la familia 8086, y la mayoría de los montadores (incluyendo aquellos para PASCAL, FORTRAN, C, etc.) de proveedores afamados le permitirán enlazar módulos de edición escritos en cualquiera de estos lenguajes (con bibliotecas de código máquina ensambladas si así lo deseara).

Una función completa

Finalmente, he aquí una llamada a función de alto nivel completa. Muy a menudo es necesario averiguar cuánto espacio de disco hay disponible antes de escribir datos en un archivo. Llamando a la función del DOS 36H (decimal 54) podemos comprobar el disco antes de escribir en él y, de ser necesario, darnos la opción de cerrar archivos o intercambiar discos si así se requiriera. Para ejecutar esta llamada al sistema, se coloca en AH el número de función del modo usual, y DL debe contener el número de unidad; la unidad A es 1, la B es 2, etc. Un código de 0 indica la unidad por defecto. Luego es preciso inicializar el campo DL (asociado al byte *low* del registro D) en el valor requerido, colocar 36H en AH y realizar la llamada al sistema. Al retornar, AX contendrá el código de error OFFFFH si algo no marcha bien (que se haya dado un código de unidad no válido, p. ej.); de lo contrario, se devuelven los siguientes datos (mostrando lo que retiene cada registro):

AX	Cantidad de sectores por grupo (*cluster*)
BX	Grupos disponibles
CX	Cantidad de bytes por sector
DX	Cantidad total de grupos

De modo que el espacio total de disco disponible (en bytes) es el producto de los valores:

grupos × sectores por grupo × bytes por sector

Observe que un grupo es una unidad de asignación de espacio de disco, y corresponde a una cantidad de sectores completos.

```
FUNCTION EspacioDisco (unidad:integer):
integer;
  {devuelve la cantidad de bytes libres en la
  unidad}
  VAR
     registros: SysReg;{ver manual}
  BEGIN
    WITH registros DO
      BEGIN
        DL:=unidad;{0=defecto,1=A,etc.}
        AH:=36H; {número de función}
        sistema (registros);{llamarlo}
        IF AX=0FFFFH
          THEN {hubo un error, de modo que:}
            BEGIN
            WriteLn ('EspacioDisco: ERROR
            (código no válido?)');
            EspacioDisco:=0{por lo que
            sabemos!}
          END
        ELSE
             EspacioDisco:=AX*BX*CX
      END {sectores*grupos*bytes}
END: {EspacioDisco}
```

Programación de alto nivel

Una alternativa al empleo de un ensamblador es el sustituto del BASIC de colocar (POKE) instrucciones (ensambladas manualmente) en un bloque conveniente de memoria libre. Esto, no obstante, sólo se recomienda para tareas bastante triviales tales como en las cortas rutinas que acabamos de ilustrar. Usted necesitará una lista de *opcodes* Intel, un detallado mapa de memoria de su máquina, conocimiento sobre el espacio para programas transitorios, ¡y una buena dosis de paciencia! Especialmente desde el punto de vista de la legibilidad, lo mejor es evitar esta opción. Afortunadamente, muchas implementaciones de lenguajes compilados para el MS-DOS incluyen las ampliaciones necesarias para escribir código máquina externo y enlazarlo con el programa principal.

El propio PASCAL-86 de Microsoft posee un procedimiento denominado DOSXQQ, que da solicitudes de "función" directas. El MT+86 de Digital Research posee una rutina similar. Quizá el mejor enfoque (y, por cierto, el más sencillo) sea el adoptado tanto por el compilador homologado por ISO de Prospero Software (Pro PASCAL), muy bien considerado, como el económico Turbo PASCAL. Éstos ofrecen un vehículo excelente para programación del DOS sin necesidad de utilizar lenguaje ensamblador.

Tanto el Pro PASCAL como el Turbo PASCAL poseen un descriptor TYPE que mapea todos los registros 8086 esenciales en un RECORD (registro), cuyos campos se pueden inicializar antes de una llamada y acceder luego a ellos para devolver datos desde la función DOS. La sintaxis es similar en cada caso (para detalles completos, remitirse al manual de PASCAL).

Prospero posee un procedimiento llamado, como es lógico, system, que toma un parámetro de variable del tipo mencionado (SysReg) y realiza la auténtica llamada a la función. De modo que el procedimiento completo "imprimir la unidad actualmente seleccionada" que acabamos de dar en ensamblador se convertiría en:

```
WITH registros DO
  BEGIN {imprimir la unidad seleccionada:}
    AH:=25; {código de función 19H}
    system (registros);{llamarla}
    WriteLn ('La unidad seleccionada actualmente
       es', chr (AL+ord ('A')),':')
  END
```

Una pila de ideas

Nuestro examen se centra en las relaciones entre la subrutina y la pila, como paso previo al estudio de los parámetros

Comenzaremos analizando un ejemplo que realiza dos llamadas a la subrutina CALC, ya presentada anteriormente.

```
                    ....          *líneas de código
2000 4EB8           JSR CALC      *llamada a la
                                   subrutina
                    (...líneas de código...)
2100 4EB8           JSR CALC      *otra vez
2102 4000
                    (...líneas de código...)
4000 3200     CALC  MOVE D0,D7    *entrada subrutina
                    (...líneas de código...)
4100 4E75           RTS           *retorno a llamada
```

Cada vez que se ejecuta la instrucción JSR, el 68000 coloca el contenido del contador de programa (la dirección de la instrucción siguiente a la instrucción JSR) en la pila y carga la dirección de la rutina especificada por JSR en el PC. La subrutina se ejecuta seguidamente y cuando encuentra RTS el 68000 saca la dirección de retorno (conocida como *dirección de enlace*) de la pila y la pone en el PC. Es de observar que la dirección de enlace (*linkage*) se guarda en formato de palabra larga completa, por lo que en el ejemplo, tras la primera ejecución de CALC, la pila y el PC contendrán los valores que se ilustran en el dibujo (página contigua).

Después de la ejecución de RTS, al final de la subrutina CALC, se restaura la dirección de enlace en el PC y la pila queda como se ilustra en la segunda parte del dibujo. Es de observar que la dirección de enlace continúa escrita en la pila, y será sobreescrita la siguiente vez que ésta sea usada.

Este mecanismo de enlace es obra del hardware del 68000. Pero su significado es que si anidamos subrutinas, el mecanismo de enlace también se cuidará automáticamente de esta situación.

Es claro que este método de apilar las direcciones de enlace automáticamente se cuida de las subrutinas anidadas mediante la ampliación de la pila. Supongamos que CALC se llama a sí misma (llamada recursiva). De nuevo el mecanismo de enlace se encargará de la situación apilando todas las direcciones de enlace una encima de la otra hasta que acaben las llamadas recursivas, o hasta que no se venga abajo el funcionamiento por culpa de alguna violación de dirección, es decir, ¡porque la pila ha crecido en exceso!

Volvamos ahora al problema de cómo se pasan los parámetros, tanto de entrada como de salida. En el capítulo anterior pasamos datos a la subrutina CALC mediante D1 y recuperamos datos de ella con D2. Esta sencilla solución del problema basta para pequeños programas, pero precisamos una solución más general para programas mayores, sobre todo pensando en que sólo se dispone de unos cuantos registros.

Un método utilizado para superar esta dificultad, en especial con compiladores, es el empleo de la pila como instrumento para pasar parámetros. Así, escribiríamos esto para pasar parámetros a CALC:

```
2000  MOVE PARAM1,-(SP)  *pone el primer
                          parámetro en la pila
2004  MOVE PARAM2,-(SP)  *y el segundo
2008  JSR CALC           *llamada a CALC
200A  .....
```

donde la pila contendría, al entrar en CALC:

```
                PARAM1
                PARAM2
                enlace ls
SP en CALC→     enlace ms
```

Para acceder a PARAM1 tendremos que utilizar un desplazamiento en el SP de modo que salte por encima de la dirección de enlace:

```
ADDQ  #6,SP     *ajuste puntero
MOVE  (SP),D4   *toma el
                 parámetro
```

o, más sencillamente:

```
MOVE  6(SP),D4
```

Naturalmente, si alteramos el puntero de la pila, del modo que sea, debemos cerciorarnos de que antes de ejecutar la instrucción RTS apunte a la dirección de enlace, de otro modo pueden sobrevenir resultados desastrosos e impredecibles. Este principio es también válido cuando CALC devuelve parámetros mediante la pila, a menos que no sobreescriba uno o los dos parámetros de entrada, como es obvio.

No obstante, un método más sencillo será el empleo de una pila independiente para los parámetros, por ejemplo, la A6, dejando tranquilo al puntero de la pila (que se reserva para las funciones del hardware). En este caso, la llamada sería:

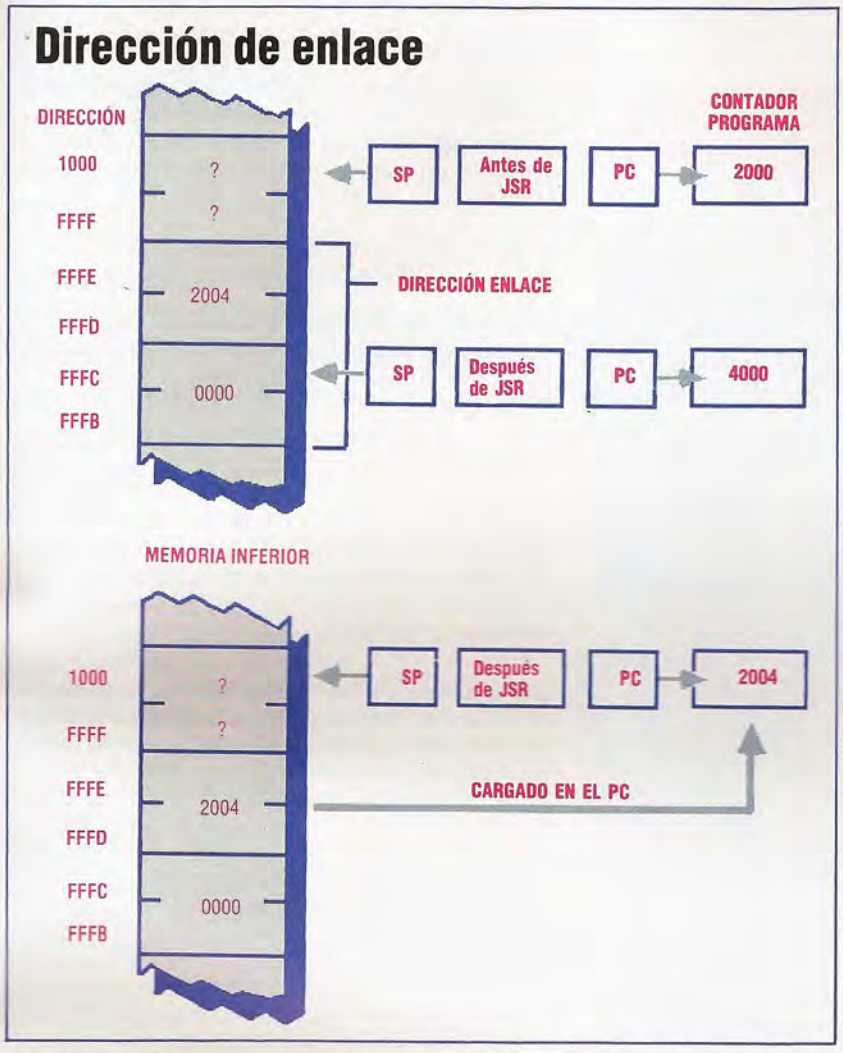
```
MOVE  PARAM1,-(A6)     *pone primero
                        el parámetro en la
                        pila A6
MOVE  PARAM2,-(A6)     *y el segundo
JSR   CALC             *guarda el SP
                        para el enlace
```

y dentro de la subrutina podemos fácilmente tomar los parámetros empleando, por ejemplo:

```
MOVE  (A6)+,D2         *toma el segundo
                        parámetro
MOVE  (A6)+,D1         *y el primero
```

ya que no existe el peligro de "pisar encima" de algún enlace de llamada a rutina. Es de notar que se toman los parámetros en el orden inverso respecto a la llamada.

Las pilas se emplean en los modernos lenguajes de alto nivel estructurados en bloques, como el PASCAL, en una forma muy ordenada y estructurada.



Vínculo permanente
El 68000 guarda una dirección de retorno (la *dirección de enlace*) en la pila antes de pasar el control del programa a la subrutina. Esta dirección se guarda en formato de palabra larga en la pila y se restaura en el contador de programa (*program counter*: PC) cuando se encuentra una instrucción RTS

La estructura de pila típica permite parámetros y variables locales de un procedimiento (que se implementará como subrutina en el código a ejecutar) que ha de permitir espacio en la pila dentro de un área definida. Así, por ejemplo:

```
Procedure Calcular (xval, yval:int);
var
        store:int;
begin
        store:=2*xval+yval^2;
```

como un fragmento de PASCAL tendría los parámetros de entrada xval e yval, y la variable local store, espacio reservado en la pila. Además, todo almacenamiento temporal sin nombre que necesite el compilador (pongamos por caso, para retener el componente yval^2 de store mientras se evalúa 2*xval) empleará la pila en su manera normal de poner y sacar.

Este arreglo está bien estructurado desde la perspectiva del compilador, pero en algunos ordenadores puede llevar a un nada despreciable dispendio en la manipulación de los datos de pila. En el 68000, la situación está facilitada considerablemente por el uso de dos instrucciones: LNK (*link*: enlazar) y UNLNK (*unlink*: desenlazar).

Estas instrucciones se emplean juntas y permiten la fácil manipulación de datos mediante la reserva de bloques de memoria dentro de la pila para uso de la subrutina. Después de una entrada a la subrutina, LNK establece un registro de direcciones definido (denominado *frame pointer*, FP, puntero marco) en un área de datos de la pila, y baja el SP de la pila un cierto número de posiciones. Por ejemplo, si la pila era:

```
        param1
        param2
        enlace ls
SP→     enlace ms
```

después de ejecutar JSR, el estado, tras la ejecución de la instrucción LNK, sería:

```
                param1
                param2
                enlace ls
                enlace ms
FP →            FP antiguo
|               denominado
desplazamiento local
|               variables
SP
```

El espacio de la pila crecería "hacia abajo", como indica el dibujo, dado que la subrutina necesita mayor espacio de trabajo.

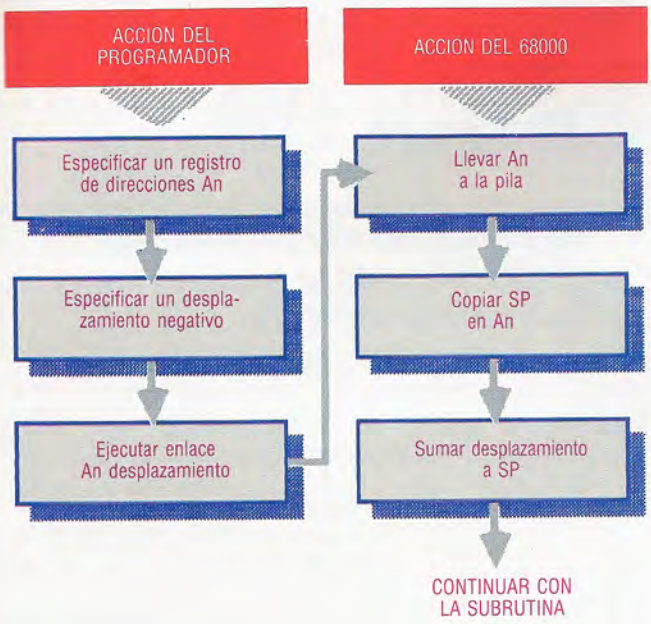
Una vez ejecutada la instrucción UNLNK al final de la subrutina, inmediatamente antes de la instrucción RTS, los punteros volverán a su estado previo a la entrada en la subrutina.

Hasta aquí hemos visto los dos extremos respecto del pase de parámetros en el 68000. En un extremo tenemos el empleo sencillo de los registros en el que el parámetro se carga en el registro de los datos (en realidad, un registro de direcciones si deseamos pasar una referencia a un parámetro). En el otro extremo tenemos el empleo altamente estructurado de la pila con el fin de implementar el pase de parámetros en un lenguaje de alto nivel. Veamos ahora un término medio, un estado en que podemos hacer un empleo algo más sofisticado que la utilización del registro sencillo pero que no implica el uso de la pila.

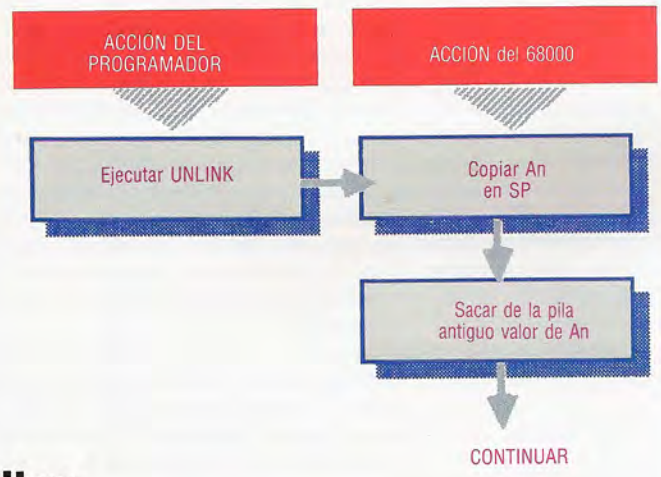
El primer método consiste en emplear un área definida de datos dentro de un grupo de subrutinas (p. ej., llamado módulo) que contiene todos los parámetros de entrada/salida asociados con ese módulo. Esta área de datos no es global, pero sólo es empleada por un conjunto definido de subrutinas para el propósito específico de pasar parámetros. Cada subrutina sabrá exactamente dónde tomar los parámetros y dónde dejar su parámetro de salida al final. Así, por ejemplo, podríamos escribir:

```
CALC    LEA     INPUTS,A4       *apunta al
                                 área parám. entrada
        MOVE    (A4),D0         *toma el primer
                                 parámetro
        MOVE    (A4)+,D1        *y también el
                                 segundo
        ....                    *instrucciones
                                 diversas
        MOVE    D5,OUTPUT       *entra el
                                 parám. salida
        RTS
```

Naturalmente, podemos pasar un puntero al área de parámetros en un registro de direcciones definido. Esto significaría que la primera línea en el ejemplo anterior resulta innecesaria. En efecto,

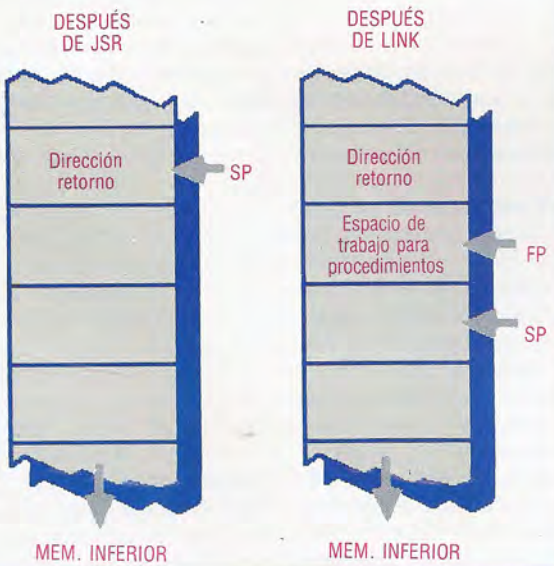


UNLINK



Pista libre

LINK y UNLINK proporcionan al programa unos medios rápidos de reserva de espacio de pila para la subrutina. Esto es importante en especial en un entorno de multiproceso, donde una subrutina puede ser interrumpida por otro programa. LINK toma un desplazamiento especificado por el usuario y hace descender el puntero de la pila en la memoria. Un registro, el An, se emplea como *puntero marco* por la rutina para acceder a los datos almacenados en el bloque reservado. Finalmente, SP y An pueden ser restaurados con sus valores previos mediante UNLINK. Este esquema muestra primero las acciones que se realizan y, en la parte inferior, el efecto en la pila



Caroline Clayton

este método se basa en la creación de áreas independientes de pilas para los parámetros de ENTRADAS y SALIDA.

El segundo método vuelve al procedimiento más sencillo de todos, el del uso directo de los registros, pero se aprovecha de la instrucción MOVEM. Examinemos primero esta instrucción. Sea la siguiente formulación:

```
MOVEM lista registro,AREARESERVA
```

Esto guardaría los registros definidos en la lista de registros en la dirección absoluta AREARESERVA y siguientes posiciones hacia arriba.

Por ejemplo:

```
MOVE D3/D5/A2, AREARESERVA
```

cargará D3 en AREARESERVA, D5 en AREARESERVA+2, y A2 en AREARESERVA+4. Podríamos guardar, asimismo, los registros en la pila con:

```
MOVEM D3/D5/A2,–(SP)
```

que apilará el registro en direcciones consecutivas decrecientes. Si los registros consecutivos se han de guardar, entonces el ensamblador acepta la versión taquigráfica:

```
MOVEM D1–D5/A4,–(SP)
```

donde se apilarán desde D1 hasta D5 y A4.

Podemos también almacenar los registros empleando la lista de registros como destino en la instrucción MOVEM. Por ejemplo:

```
MOVEM–(SP),D1–D5/A4
```

restaurará los registros almacenados en el ejemplo anterior.

Debemos preguntarnos para qué nos sirve esto en el pase de parámetros. La respuesta está en la conveniencia de almacenar registros que se han distribuido para un objetivo específico, por ejemplo para uso del sistema y después quedar libres para usar los registros en lo que queramos.

Por ejemplo, si reservamos D0 y D1 como registros de parámetros, entonces con tal de que almacenemos los restantes registros en la entrada a la subrutina, podríamos utilizar los restantes registros según se exija en la subrutina.

Por ejemplo:

```
CALC    MOVEM D2–D7,–(SP)   *guarda antiguos
                             D2-D7
        MOVE   D0,D2        *y carga el
                             primer parámetro
        ....                *instrucciones
                             que emplean
                             D2-D7
        MOVEM –(SP),D2–D7   *y restaura
                             valores antiguos
        RTS
```

De esto se puede colegir que el 68000 nos permite el empleo de varios métodos para pasar parámetros a las subrutinas y ¡nos proporciona además algunas herramientas útiles para hacerlo!

Ingeniería

Ahora nos referiremos a la ingeniería en hardware y en software, carreras relacionadas con la instalación y mantenimiento de nuevos sistemas

El concepto de ingeniero de hardware, la persona que diseña y construye sistemas de ordenador, aparece rodeado de un aura mítica; no obstante, se trata de una actividad de características perfectamente delimitadas y definibles.

La etapa inicial es la de la especificación: decidir cuáles son las funciones que ha de llevar a cabo la nueva máquina. Esta etapa tiende a ser competencia del ingeniero más experimentado, en unión con el departamento de marketing. En el campo militar y de defensa (en Gran Bretaña, p. ej., hay más ingenieros de hardware trabajando en proyectos de carácter militar que en proyectos civiles), la especificación es privativa del Ministerio de Defensa.

La siguiente etapa para el ingeniero, tras recibir las especificaciones, consiste en decidir la viabilidad del proyecto y cuánto tiempo llevaría su realización. (Los plazos representan la esencia de todos los proyectos de ingeniería, tanto militares como civiles.) Habiendo calculado el costo del proyecto y evaluado las limitaciones de tiempo, el ingeniero debe diseñar el sistema. Para adoptar una decisión acerca de los componentes a utilizar y el modo de conectarlos, los ingenieros han de estar al tanto de los últimos avances tecnológicos a nivel de componentes y, en consecuencia, habrán de ser asiduos lectores de la prensa especializada. Tras el diseño inicial, habrán de ensamblarse y probarse varios prototipos antes de que se pueda comenzar la producción final.

Diseño de prototipos

Toda ingeniería inevitablemente es un compromiso para aprovechar al máximo unos recursos limitados. Si sólo se dispone de seis meses, es imposible construir un sistema ideal desde cero. Si usted tiene que construir un producto que sea compatible con otros ya existentes, no podrá conferirle una arquitectura revolucionaria. Si un producto ha de venderse a un precio determinado, habrá límites para los precios de los componentes a utilizar. Por consiguiente, es tarea del ingeniero diseñar el mejor sistema posible dentro de esas limitaciones.

La ingeniería de hardware se ha constituido en una carrera privativa de los jóvenes. La "crisis de la edad", que suele afectar en la mayoría de las industrias a los trabajadores que tienen entre 40 y 45 años, por lo general se presenta antes entre los ingenieros informáticos. En efecto, muchos de ellos, al acercarse a los treinta años de edad, se ven invadidos por una sensación de fracaso si hasta ese momento no han logrado introducirse en la gerencia o en el departamento de ventas. Con frecuencia el ingeniero de 35 años de edad se suele ver como alguien "venido a menos" que carece de las aptitudes personales y para la comunicación necesarias para la gerencia y cuyos conocimientos tecnológicos probablemente están desfasados.

El acceso a la carrera de ingeniería de hardware se produce a dos niveles. Muchas grandes empresas (en especial aquellas pertenecientes al campo de la defensa) suelen contratar a jóvenes de entre 16 y 18

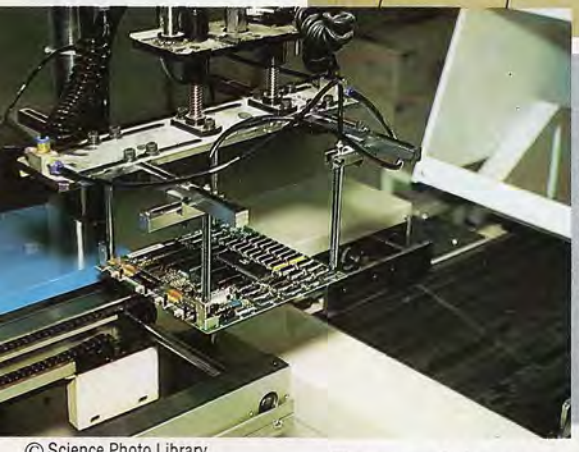
© Science Photo Library

Diseño de placas
La mayoría de los miniordenadores y ordenadores centrales se diseñan de forma modular, lo que facilita la tarea del técnico. Se dedican diferentes placas a las distintas funciones, y se suele disponer de equipos de comprobación para controlar su rendimiento

años de edad con altas calificaciones y proporcionarles formación para que se desempeñen como aprendices. Tal como sucede en la mayoría de las profesiones moderadamente bien remuneradas, la ingeniería de hardware se está volcando cada vez más hacia los graduados. Si usted desea introducirse a nivel de graduado, necesitaría un título en física o bien en ingeniería electrónica; la ingeniería de hardware no es un campo que se preste a las matemáticas, la informática, ni otras titulaciones de cálculo.

© Science Photo Library

La ayuda de la alta tecnología
A los técnicos se les está permitiendo disponer de una creciente cantidad de equipos inteligentes para facilitar el diagnóstico y la corrección de fallos. Aquí vemos a un técnico explorando un gran ordenador central Cray con un detector infrarrojo, en busca de "puntos calientes" que podrían indicar un defecto de funcionamiento. Además, ahora muchos sistemas se diseñan con módulos incorporados de comprobación de fallos fácilmente sustituibles, lo que simplifica el papel del técnico y, en algunos casos, hace casi prescindible su aportación



© Science Photo Library

Diseño automatizado
El papel del ingeniero de diseño suele incluir, en las primeras etapas del desarrollo del producto, la consideración cabal de los medios de construcción. Un producto nuevo destinado a una venta masiva se ha de diseñar de modo de obtener el máximo partido de las técnicas de producción automatizada. La placa del Apple Macintosh que vemos en la fotografía la está ensamblando un robot

En general, las empresas dedicadas a desarrollar proyectos de carácter militar no suelen pagar tan bien como el sector comercial. En Gran Bretaña, es probable que los graduados o aprendices que hayan concluido su preparación sean contratados con sueldos que oscilen entre £8 000 y £10 000. Contrariamente a otras especializaciones informáticas, en ingeniería de hardware es necesario tener una experiencia superior a un año para obtener un aumento salarial de importancia; durante los primeros dos o tres años, los incrementos tienden a ser del orden de apenas un 10 %. El ingeniero de hardware experimentado suele percibir unos emolumentos de entre £10 000 y £13 000 en el campo de la defensa y de entre £13 000 y £16 000 en el sector comercial.

A la edad de entre 28 y 30 años, el ingeniero tenderá a dejar el campo para introducirse en la dirección o las ventas, valiéndose de sus cualificaciones técnicas e industriales. Por supuesto, los salarios pueden ser especialmente elevados en el campo de ventas, percibiendo los vendedores hasta £40 000. La dirección de proyectos es el primer peldaño en la escalera de la gerencia. Tras él, sin embargo, el ingeniero se convertirá en gerente y sus aptitudes técnicas ya no tendrán una importancia tan preponderante.

La ingeniería de software marcha paso a paso con la ingeniería de hardware. Esencialmente, la tarea del ingeniero de software consiste en proporcionar el entorno de software (el sistema operativo) que acompañará al hardware. Con frecuencia, el papel del ingeniero de software es el de adaptar un sistema operativo ya existente aportado por una empresa independiente en lugar de producir un sistema completamente nuevo.

Debido a la escasez de personal que cuente con la suficiente preparación, los ingenieros de software tienden a percibir sueldos más altos que los ingenieros de hardware. En Gran Bretaña, éstos parten de alrededor de £10 000, subiendo hasta £20 000. El acceso se concede casi exclusivamente a los graduados, y, a diferencia de la ingeniería de hardware, se contrata a titulados en carreras de cálculo y se les proporciona adiestramiento.

La crisis que suele afectar a los ingenieros de hardware al acercarse a los treinta años de edad también alcanza a los ingenieros de software, si bien de manera menos acusada. Nuevamente, el ingeniero de software tiende a pasar a las ventas, la gerencia o al asesoramiento.

En general, los ingenieros de hardware pueden pasarse al campo del software, mientras que los ingenieros de software no pueden pasarse al hardware. Los dos papeles laborales tienden a estar separados, sin embargo, colaborando en un proyecto trabajadores de cada campo.

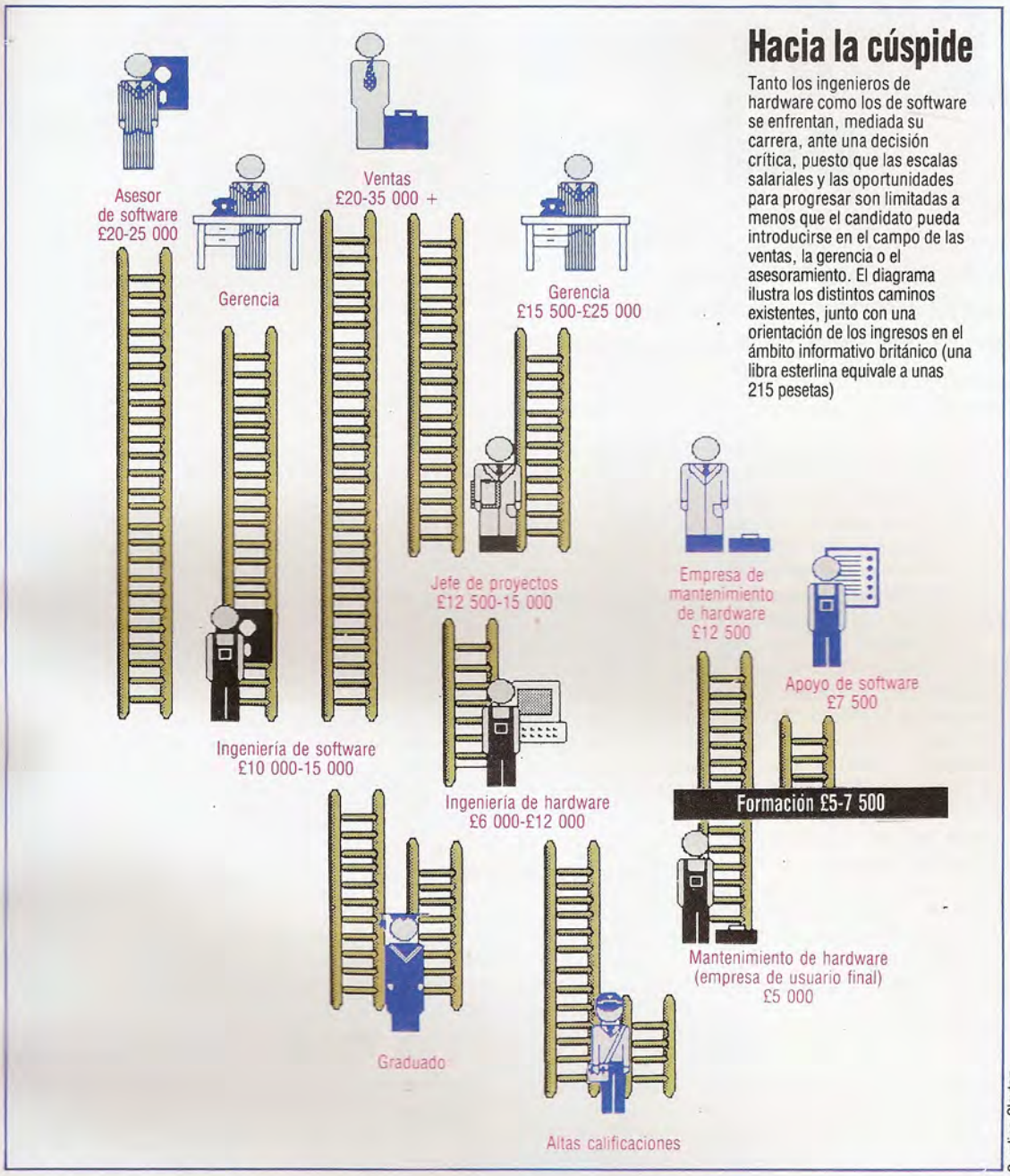
Apoyo técnico

El campo del apoyo técnico proporciona numerosas oportunidades para quienes deseen introducirse en la industria del ordenador y posean alguna experiencia en cuanto a máquinas personales. Al igual que en la ingeniería, existe una clara distinción entre mantenimiento y apoyo de hardware y software. De las dos áreas, el hardware tiende a ser "el pariente pobre", con niveles salariales inferiores a los del área de software.

El apoyo de hardware cubre la instalación de productos en el local del cliente y la comprobación y el diagnóstico de fallos de hardware. El personal de apoyo de hardware tenderá a trabajar para empresas de suministros o bien para empresas de mantenimiento independientes, como las que se contratan para mantener y "detectar problemas" en los ordenadores de otras empresas.

Casi todos los comerciantes de micros de gestión suelen contratar personal de apoyo técnico de hardware y a menudo también a aficionados jóvenes y perspicaces. Los salarios iniciales son bastante bajos, pero con el paso del tiempo pueden alcanzar un nivel bastante elevado.

En el contexto de un negocio pequeño, la persona de apoyo tenderá a adquirir muchísima expe-



riencia en materia de software, en la demostración de productos y en el asesoramiento al personal de ventas. Una persona joven de apoyo de hardware en un comercio de IBM PC o en una casa de sistemas de miniordenador puede esperar encontrarse frente a varios caminos para hacer carrera: el negocio de apoyo de software algo mejor remunerado, trabajar para una de las casas de mantenimiento independientes o pasar al campo de las ventas.

No obstante, antes de introducirse en el campo del apoyo de hardware, el aspirante ha de tener presente algunos puntos. Si bien en algunas ocasiones no parece así, en realidad los ordenadores se están volviendo cada vez más fiables y se los está diseñando para ser más fáciles de apoyar. En algunos ordenadores, el ingeniero simplemente ha de sustituir un módulo por otro, y los programas de autodiagnóstico significan que se requieren menos conocimientos para averiguar lo que marcha mal en un ordenador. Teniendo presente esto, en un futuro cercano el personal de apoyo de hardware bien podría encontrarse con que su aportación tecnológica sea cada vez más prescindible.

El apoyo de software supone averiguar por qué un software determinado no funciona bien en una máquina determinada y remediarlo. El personal de apoyo de software también habrá de transferir programas de una máquina a otra y podría verse involucrado en áreas especializadas tales como las comunicaciones.

Al igual que en el apoyo de hardware, en este campo se pueden introducir personas con experiencia relativamente escasa. Esencialmente, hay tres niveles de acceso: como aficionado de entre 16 y 17 años de edad que desee aprender y esté dispuesto a percibir un bajo salario durante un par de años; como usuario final con algo de experiencia de programación y utilizando ordenadores de gestión, o como graduado con una titulación en una carrera de cálculo.

Biblioteca pública

Llegados a este punto, centraremos nuestra atención en la biblioteca de instrucciones estándares de E/S

Al igual que muchos otros lenguajes modernos, el C no especifica detalles de entrada/salida (E/S) dentro del lenguaje propiamente dicho. Una especificación exhaustiva de E/S para un lenguaje que se ha de utilizar en una amplia variedad de aplicaciones y en varias máquinas diferentes sería muy restrictiva, dificultando muchas operaciones directas, como es el caso del COBOL y el FORTRAN.

Por ejemplo, en COBOL es imposible aceptar un único carácter directamente desde el teclado sin emplear extensiones no estándares al lenguaje. No obstante, si la E/S no está especificada en absoluto, entonces resulta difícil lograr la portabilidad de programas entre máquinas y sistemas operativos. La entrada por teclado en PASCAL constituye un buen ejemplo de una operación que se puede comportar de modo bastante diferente en distintas versiones del lenguaje.

En C se proporciona un grado de estandarización mediante la biblioteca estándar, stdio.h, que define numerosas funciones de E/S de alto y bajo nivel. Los detalles y las facilidades exactas de la misma pueden variar de una implementación a otra, pero es mejor ceñirse a las funciones de alto nivel. Por otra parte, el programador de una versión específica del lenguaje está en libertad de proporcionar funciones que saquen partido de determinados aspectos de una máquina.

Ya hemos analizado la función "caballo de tiro" para la salida en pantalla (printf) y su correspondiente función para entrada por teclado (scanf). Hay muchas otras funciones que permiten un control más preciso del teclado y la pantalla, así como para E/S a archivos en disco y otros dispositivos. El C ha simplificado la idea de la E/S a archivos y dispositivos tratándolos exactamente de la misma manera. Para el C, un archivo es un flujo de bytes; las funciones de entrada simplemente toman un byte o grupo de bytes de una corriente de entrada, y las funciones de salida envían una corriente. Si se conecta una corriente al disco, es posible desplazarse a lo largo de la corriente hasta una posición especificada, proporcionando de este modo un acceso aleatorio. Cuando los archivos se tratan de esta manera, desaparece la diferencia lógica entre un archivo y un dispositivo físico.

Normalmente los archivos se deben abrir antes de utilizarlos, para especificar el dispositivo o área del disco hacia donde se dirige el flujo de bytes. No obstante, en todo programa en C hay tres archivos que siempre están abiertos. Éstos son: stdin, que normalmente está conectado al teclado; stdout y stderr, que suelen estar conectados a la pantalla. Muchos sistemas operativos, como el Unix, permiten redirigir la entrada y salida, de modo que estas asignaciones se pueden modificar. Las funciones printf y scanf operan sobre stdout y stdin, pero no constituyen más que casos especiales de las funciones más generales fprintf y fscanf.

De modo que, por ejemplo, las sentencias:

```
printf (.....);
fprintf(stdout,.....);
```

son idénticas en operación. Existen otras dos funciones similares, sprintf y sscanf, que realizan E/S hacia y desde series en la memoria principal.

Desde el punto de vista del C, stdout en realidad es un puntero a un dato del tipo FILE, que normalmente se define mediante una macro (# define) en el archivo stdio.h. Para abrir un archivo en disco o dispositivo, se utiliza la función fopen(nombrearchivo,modalidadarchivo). Ésta toma una serie para el nombrearchivo que puede aludir a un dispositivo o archivo según las convenciones del OS. La modalidadarchivo puede ser "r" para leer el archivo, "w" para escribir en él, o "a" para añadirle datos. Si se abre el archivo con "a" o "w" y el mismo no existe, será creado; un archivo que sí exista y se abra con "w" se sobreescribirá. El puntero de archivo (un entero largo que indica la posición actual a lo largo del flujo de bytes) se posicionará al comienzo del archivo con "r" o "w". El valor devuelto por la fun-

Prueba de fuego

Existen muchas funciones y macros útiles incluidas en diversas bibliotecas que nos es imposible analizar. Pero en el archivo ctype.h hay numerosas funciones útiles para manipulación de series. Las mismas se pueden incluir con la línea:

```
# include(ctype.h)
```

El primer grupo proporciona un medio de comprobar determinados atributos en un carácter, devolviendo un valor no cero (verdadero) si el carácter comprobado posee el atributo, y cero en caso contrario. Normalmente están implementadas con gran eficacia.

Nombre	Verdadero si
isalpha(c)	c es una letra
isupper(c)	c está en mayúscula
islower(c)	c está en minúscula
isdigit(c)	c es un dígito
isxdigit(c)	c es un dígito hexa
isspace(c)	c es un carácter espacio
isalnum(c)	c es una letra o un dígito
ispunct(c)	c es un carácter de punt.
isprint(c)	c es un carácter imprimible
iscntrl(c)	c es un carácter de control
isascii(c)	c es un código ASCII

El segundo grupo de funciones proporciona tres convenientes conversiones:

Nombre	Efecto
toupper(c)	convierte c a mayúscula
tolower(c)	convierte c a minúscula
toascii(c)	convierte c a código ASCII

Estudios de biblioteca

Las siguientes son otras funciones de E/S de la biblioteca estándar del c:

getc(puntero_a_archivo) toma el siguiente byte del archivo o dispositivo que se ha abierto con "r" El valor se devuelve como un int. Se devuelve un determinado valor EOF si se encuentra el final del archivo. Puede estar implementada como una macro

getchar() Equivale a getc(stdin)

fgetc(puntero_a_archivo) Vuelve a colocar el carácter c en el archivo desde el cual presumiblemente se había leído. Devuelve el valor int de c. Se debe haber leído al menos un carácter del archivo antes de que se pueda volver a despositar uno, y no es fiable si se coloca más de un carácter por vez

putc(c,puntero_a_archivo) Produce el carácter c en el archivo de salida, devolviendo el valor int de c. Puede ser una macro

putchar(c) Equivale a putc(c,stdout)

fputc(c,puntero_a_archivo) Equivale a putc, pero siempre está implementado como función

gets(s) Donde s es una serie (puntero a char), lee caracteres de stdin hasta encontrar una nueva línea. Esta última no se coloca en la serie, que se terminará correctamente con un "\0". Se devuelve el valor de s

fgets(s,n,puntero_a_archivo) Lee caracteres del archivo en la serie s hasta que se hayan leído n-1 caracteres o hasta hallar una nueva línea. Ésta se colocará en s, que se terminará con "\0". Se devuelve el valor de s

puts(s) Produce la serie s en stdout añadiendo una nueva línea

fputs(s,puntero_a_archivo) Produce la serie s en el archivo sin ninguna nueva línea

fseek(puntero_a_archivo, desplazamiento, lugar) Desplaza el puntero de archivo a lo largo del flujo de bytes hasta el desplazamiento desde el lugar especificado; el lugar puede ser 0 para el comienzo del archivo, 1 para la posición actual o 2 para el final del archivo.
El desplazamiento debe ser de tipo long int

rewind(puntero_a_archivo) Equivale a: **fseek(puntero_a_archivo),0L,0)**

ftell(puntero_a_archivo) Devuelve el desplazamiento actual (un long int) desde el comienzo del archivo

unlink(nombrearchivo) Suprime del directorio el archivo mencionado. Devuelve -1 si el archivo no existe, y 0 en caso contrario

exit(estado) Termina un programa devolviendo el valor int de estado al OS o proceso de llamada. Se utiliza 0 para una terminación normal

Caroline Clayton

ción fopen es un puntero al tipo FILE que se puede utilizar en otras funciones; será NULL si por algún motivo no se puede acceder al archivo.

Existe una función fclose(puntero_al_archivo) que cierra un archivo abierto. Todos los archivos se cerrarán automáticamente al terminar un programa, pero esta función podría ser necesaria en razón de un límite del OS sobre la cantidad de archivos que se puedan tener abiertos de forma simultánea.

Tasación de caracteres

```
/* este programa contará el número de palabras, y
el número de caracteres en un archivo, cuyo
nombre se da en la línea de comando */
# include(stdio.h)
# include(ctype.h)
main(argc,argv)
int argc;
char * argv;
   {
   int cont_car=0,cont_pal=0,c,
   inword   =0;
   FILE*en_archivo, * fopen();
/* observe el nombre del archivo y la función
fopen, declarados como punteros al tipo FILE */
/*comprobación de la cantidad correcta de
argumentos*/
   If(argc !=2)
   {
      fprintf(stderr,"\nusage es %s
      nombrearchivo\n",
      *argv);
/*recuerde que la primera entrada en una matriz
argv es el verdadero nombre del programa*/
      exit(1);
   }
/*abrir el archivo y comprobar si existe, ++ argv
apunta al nombre del archivo*/
   if((en_archivo=fopen(*++argv,"r"))
   ==NULL)
{
      fprintf(stderr,"\nno puede abrir %s\n",
      *argv);
/*recuerde que ahora argv está apuntando al
nombre del archivo*/
   exist(1);
   }
   while((c=getc(en_archivo)   !=EOF)
   {
      ++cont_car;
      if(enpalabra)
      {
         if(isalnum(c))
         ;/*una sentencia vacía*/
         else
         {
            enpalabra=0;
            ++cont_pal;
         }
      }
      else
         if(isalnum(c))
            enpalabra=1;
   }
   if(enpalabra)
      ++cont_pal;
   printf("\nnúmero de caracteres=%d",
   cont_car);
   printf("\nnúmero de palabras=%d\n",
   cont_pal);
   fclose(en_archivo);
}
```

Opciones abiertas

Analizaremos las herramientas del Unix relacionadas con la gestión de archivos y la preparación de textos

La forma general de un comando Unix es:

nombre_comando opciones argumentos

Cada porción del comando debe estar separada del siguiente por al menos un espacio. Los argumentos de un comando por lo general son nombres de archivo o directorios; si se omite el argumento, se utilizan por defecto los archivos standardinput y standardoutput (el teclado y la pantalla) para la entrada y salida.

Las opciones asumen la forma de letras individuales, precedidas por un guión, y se utilizan en unión con el comando para llevar a cabo una serie de tareas. Allí donde las opciones no requieren otra información, como el nombre de un archivo, tras el guión puede ir más de una opción. Por ejemplo, el comando de listado del directorio ls posee varias opciones, incluyendo l, que produce un listado completo, y a, que lista entradas del archivo. Los argumentos para ls pueden ser ya sea una especificación de archivo o bien un directorio, y de no incluirse ningún argumento se tomará por defecto el directorio actual.

Para listar el contenido del directorio /usr, con ambas opciones seleccionadas, el comando podría ser uno de los siguientes:

```
ls -l-a/usr
ls -la/usr
```

Los comandos impartidos incorrectamente generan un mensaje de error, que indica al usuario que no se reconoce el comando o bien, de ser posible, le detalla la utilización correcta. Observe que se pueden escribir en una misma línea dos o más comandos, separándolos mediante un punto y coma.

El comando wc cuenta el número de caracteres, palabras y líneas en un archivo de texto. Las opciones son:

- **l** para contar sólo líneas
- **w** para contar sólo palabras
- **c** para contar sólo caracteres
- **p** para contar páginas (una página tiene 66 líneas)

Los argumentos pueden ser uno o más nombres de archivo. Si se proporciona más de un nombre de archivo, se cuenta cada archivo y se da un total para todos los archivos especificados. Si no se da como argumento ningún archivo, ws da por sentado que la entrada proviene del teclado.

En standardoutput (normalmente la pantalla), head visualiza las primeras líneas de un archivo. La única opción es una que determina la cantidad de líneas a visualizar. Por ejemplo, −15 visualiza las primeras 15 líneas. Los argumentos deben incluir un nombre de archivo o más.

El comando tail da las últimas líneas de un archivo. Las opciones son:

- **+n** una excepción a la regla de que las opciones siempre comienzan con −; se visualiza el resto del archivo a partir de un punto de 10 líneas desde el comienzo
- **−n** visualiza las n líneas finales del archivo; el valor por defecto es 10
- **l** cuenta en líneas (por defecto)
- **b** cambia la unidad a bloques de almacenamiento de disco
- **c** cambia la unidad a caracteres
- **r** visualiza el archivo por orden inverso

La instrucción sort clasifica un archivo por orden de clave, o clasifica y mezcla varios. Las opciones son:

- **b** para ignorar los espacios delanteros
- **d** orden de diccionario, utilizando sólo letras, dígitos y espacios en blanco
- **f** insensible a mayúsculas o minúsculas
- **n** clasificar números por valor aritmético en vez de por dígitos
- **o** dirige la salida a un archivo en lugar de a standardoutput
- **r** clasificar por orden inverso

Los argumentos son uno o más nombres de archivo; si sólo se da un nombre de archivo, se clasifica el contenido de ese archivo. La especificación de múltiples nombres de archivo hace que los archivos se clasifiquen y mezclen entre sí.

El comando cmp compara el contenido de dos archivos para determinar cualquier diferencia entre ambos. Cuando se opera en su modalidad por defecto, el comando devuelve el byte y número de línea en donde se detectó la primera diferencia. La única opción, l, informa sobre todas las diferencias entre los dos archivos. Si se omite un nombre de archivo, se da por sentado standardinput.

La instrucción comm considera dos archivos que se han de clasificar por orden de código ASCII y visualiza tres columnas: aquellas líneas existentes en el primer archivo; aquellas líneas existentes sólo en el segundo archivo; y aquellas líneas comunes a ambos. Las opciones son, simplemente, 1, 2 y 3 para omitir una de las tres columnas. Se deben especificar como argumentos los dos nombres de archivo.

La instrucción diff también encuentra las diferencias entre dos archivos. Indica los cambios que se deben efectuar en el primer archivo para hacerlo idéntico al segundo, utilizando a para añadir, c para cambiar, d para suprimir, < para una línea del primer archivo, y > para una línea del segundo archivo. Las opciones son:

- **b** para ignorar espacios en blanco finales e igualar las series de espacios en blanco independientemente de su longitud
- **e** para producir salida como instrucciones para el editor
- **r** sólo se emplea con directorios, y permite que diff se aplique a sí misma de forma recursiva para cualquier subdirectorio

Los argumentos pueden ser tanto un par de nombres de archivo como de nombres de directorio. Si los argumentos especifican dos directorios, diff lista todos los archivos exclusivos de cada directorio, y

luego lista en la tercera columna aquellos archivos comunes a ambos.

El comando uniq compara líneas adyacentes de un archivo de texto y suprime las entradas repetidas. Las opciones son:

- **u** para visualizar sólo aquellas líneas que no se repiten
- **d** para visualizar sólo las líneas que se repiten
- **c** para acompañar cada línea de salida con el número de veces que aparece dentro del archivo

Los argumentos pueden ser uno o dos nombres de archivo. Si se especifican dos archivos, la salida pasa al segundo archivo mencionado, en lugar de a la pantalla (standardoutput).

El comando lpr envía uno o más archivos a la impresora del sistema. Dado que la salida desde trabajos multitarea no se puede realizar directamente a la impresora, se coloca en una cola. El Unix explora la cola continuamente e imprime la salida de archivo en la cabeza de la cola, que se conoce como *spooling*. Los argumentos son uno o más nombres de archivo. No hay ningún juego de opciones estándares, puesto que las facilidades de impresión varían entre distintas instalaciones.

El comando lpq visualiza detalles del estado actual de la cola de impresión, que permiten comprobar si se ha imprimido un archivo determinado, o durante cuánto tiempo permanecerá en la cola a la espera de su turno de impresión. A cada tarea de impresión de la cola se le asigna un número, y no hay argumentos ni opciones estándares.

El comando lprm permite sacar un archivo de la cola de impresión antes de que se lo imprima. El argumento puede ser ya sea un nombre de archivo, el número de tarea de impresión obtenido mediante el uso de lpq, o un nombre login, en cuyo caso se suprimirán todos los archivos que pertenezcan a ese propietario.

El comando pr visualiza el contenido del archivo en standardoutput, formateado para impresión. El texto se organiza en páginas, con cinco líneas de margen inferior y un encabezamiento, compuesto por la fecha, el nombre del archivo y el número de página seguido por dos líneas en blanco. Por lo general, la salida se entuba en lpr. Las opciones son:

- **n** para organizar el texto en un número dado de columnas
- **m** para visualizar dos o más archivos lado a lado
- **t** para suprimir el encabezamiento y el margen inferior

Los argumentos son uno o más nombres de archivo. A continuación vemos un ejemplo de la potencia y eficacia del Unix ante la gestión de tres archivos de texto.

Gestión de archivos

```
%cat file1
The cat sat on the mat.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%cat file2
The cat sat on the dog.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%cat file3
The cat sat on the mat.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
This file has an extra line.
The owl and the pussy cat went to sea.

%wc file1      {contar el número de líneas, palabras y caracteres}
       4      29     132 file1

%wc -w file1   {contar sólo palabras}
      29 file1

%head -2 file1 {visualiza sólo las dos primeras líneas}
The cat sat on the mat.
Mary had a little lamb.

%tail +2 file1 {visualiza desde la segunda línea}
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%tail -2 file1 {visualizar las dos últimas líneas}
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%sort file1    {clasificar archivo por orden}
Mary had a little lamb.
The cat sat on the mat.
The owl and the pussy cat went to sea.
The quick brown fox jumps over the lazy dog.

%pr file3      {formatear un archivo para impresión}
The cat sat on the mat.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
This file has an extra line.
The owl and the pussy cat went to sea.

%sort file1 > file 4   {crear versiones clasificadas usando redirección}
%sort file2 > file5
%cmp file1 file2       {comparar un par de archivos}
file1 file2 differ char 20, line 1

%comm file4 file5
                                        Mary had a little lamb.
                        The cat sat on the dog.
The cat sat on the mat.
                                        The owl and the pussy cat went to sea.
                                        The quick brown fox jumps over the lazy dog.
{lines en file4/lines en file5/lines en ambos archivos}

%diff file1 file2      {visualizar diferencias entre archivos}
1c1                    {"c" implica "efectuar un cambio"}
< The cat sat on the mat.
---
> The cat sat on the dog.

%sort file2 file3 > file7   {mezclar dos archivos}
%cat file7
Mary had a little lamb.
Mary had a little lamb.
The cat sat on the dog.
The cat sat on the mat.
The owl and the pussy cat went to sea.
The owl and the pussy cat went to sea.
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
This file has an extra line.

%uniq file7            {visualizar archivo ignorando repeticiones}
Mary had a little lamb.
The cat sat on the dog.
The cat sat on the mat.
The owl and the pussy cat went to sea.
The quick brown fox jumps over the lazy dog.
The file has an extra line.
```

Cacería de patos

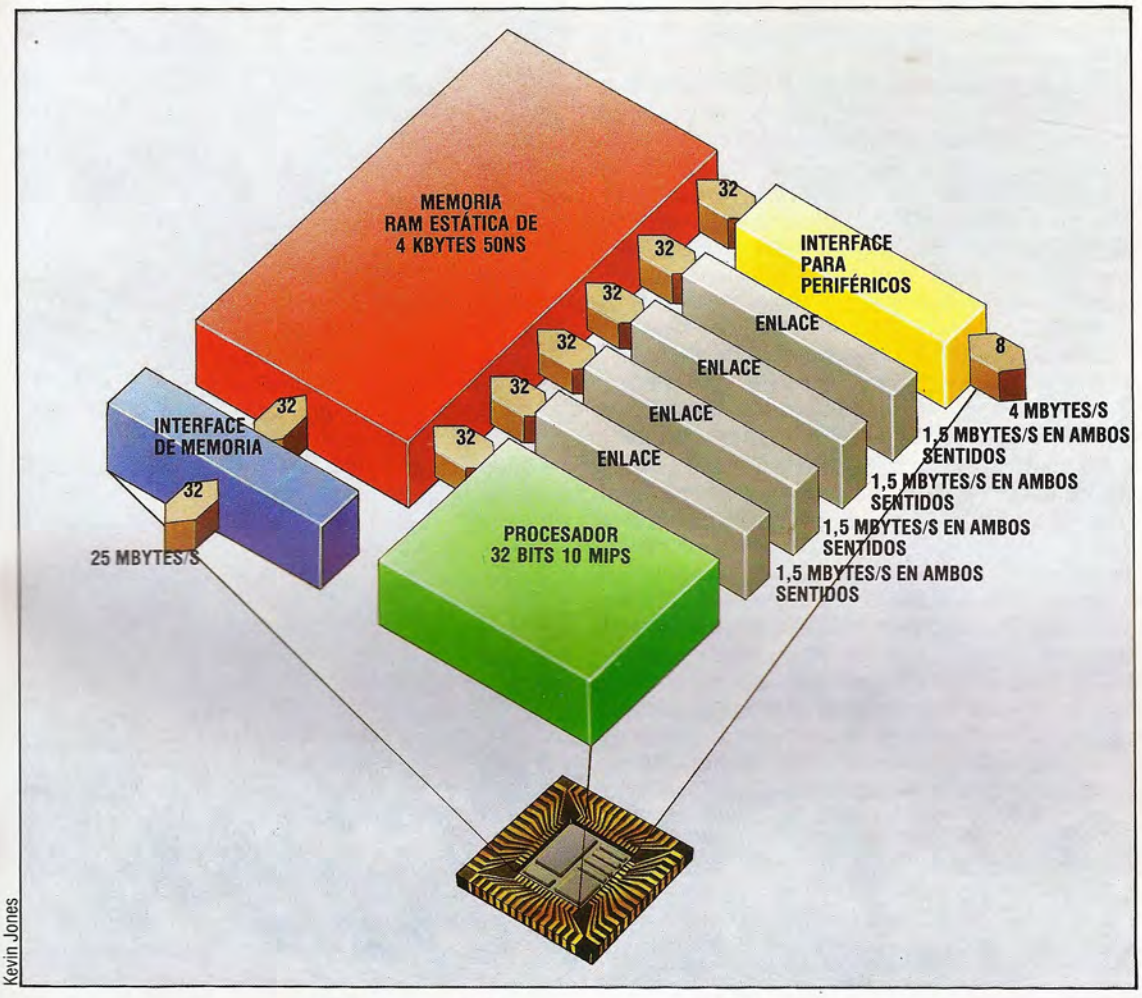
Su microordenador le invita a participar en una cacería de patos. Déjese tentar por este juego escrito por Pierre Monsaut para el Commodore 64

Los patos vuelan de derecha a izquierda por la parte alta de la pantalla. Usted se desplaza utilizando las teclas de control del cursor. Puede efectuar tantos disparos como quiera, pero tan sólo pasarán 20 patos y su objetivo es abatir el mayor número posible. Para disparar pulse la barra espaciadora. Si alcanza un pato, conseguirá un punto y lo verá caer batiendo sus alas. En esta versión el juego resulta bastante sencillo. Si desea aumentar su dificultad, sustituya la línea 4000 por:

```
4000 IF T-C = 1025 THEN 5000
```

```
  5 REM ********************
 10 REM * LA CAZA DEL PATO *
 15 REM ********************
 20 GOSUB 1000
100 X=-X
110 Y=X+0.5
120 C=C-1
130 IF C<0 THEN GOSUB 3000
140 IF NC=0 THEN 4500
150 PRINT CL$;TAB(C);CC$(INT(Y))
160 PJ=PJ+D1
170 IF PJ<PN THEN PJ=PN
180 IF PJ>PM THEN PJ=PM
190 POKE P1,CR
200 POKE PJ,CJ
210 POKE PJ+M,JC
220 P1=PJ
230 IF ABS(T-PJ)>2 THEN T=T-80:GOTO 250
240 T=PJ
250 IF T<1064 THEN 4000
260 POKE T,CJ
270 POKE T+M,CJ
280 IF T<>PJ THEN POKE T+80,CR
290 GET X$
300 D1=2*((X$=G$)-(X$=D$))
310 IF X$<>"[1SPC]" OR T<>PJ THEN 100
330 T=PJ-80
340 GOTO 100
1000 DIM CC$(1)
1010 CJ=30
1020 JC=5
1030 CC$(0)=CHR$(173)+CHR$(113)+CHR$(189)
     +"[1SPC]"
1040 CC$(1)=CHR$(176)+CHR$(113)+CHR$(174)
     +"[1SPC]"
1050 PJ=2004
1060 NC=20
1070 T=PJ
1080 X=0.5
1090 C=37
1100 M=54272
1110 CL$=CHR$(19)
1120 PM=2022
1130 PN=1985
1140 CR=32
1150 P1=PJ
1160 D$=CHR$(29)
1170 G$=CHR$(17)
2000 PRINT CHR$(147);
2010 POKE 53280,0
2020 POKE 53281,0
2030 PRINT CHR$(30);
2040 RETURN
3000 PRINT CL$;"[3SPC]";
3020 FOR I=1 TO 200
3030 NEXT I
3040 C=37
3050 NC=NC-1
3060 RETURN
4000 IF ABS((T-1024)-(C+1))<=1 THEN 5000
4010 POKE T+80,CR
4020 T=PJ
4030 IF NC<>0 THEN 260
4500 PRINT CHR$(147)
4510 FOR I=1 TO 10
4520 PRINT
4530 GET X$
4540 NEXT I
4550 PRINT TAB(13)"PUNTOS[1SPC]:";S
4560 FOR I=1 TO 10
4570 PRINT
4580 NEXT I
4590 PRINT TAB(16)"OTRA[1SPC]?"
4600 GET X$
4610 IF X$="" THEN 4600
4620 IF X$<>"N" THEN RUN
4630 END
5000 S=S+1
5010 D=C
5020 C=37
5030 PRINT CL$;
5040 FOR I=1 TO 22
5050 X=-X
5060 Y=X+0.5
5070 PRINT TAB(D);"[3SPC]"
5080 PRINT TAB(D);CC$(INT(Y))
5090 PRINT CHR$(145);
5100 FOR J=1 TO 50
5110 NEXT J
5120 NEXT I
5130 NC=NC-1
5140 FOR I=1 TO 500
5150 NEXT I
5160 PRINT CHR$(147);
5170 IF NC=0 THEN 4500
5180 GOTO 100
```

© SYBEX

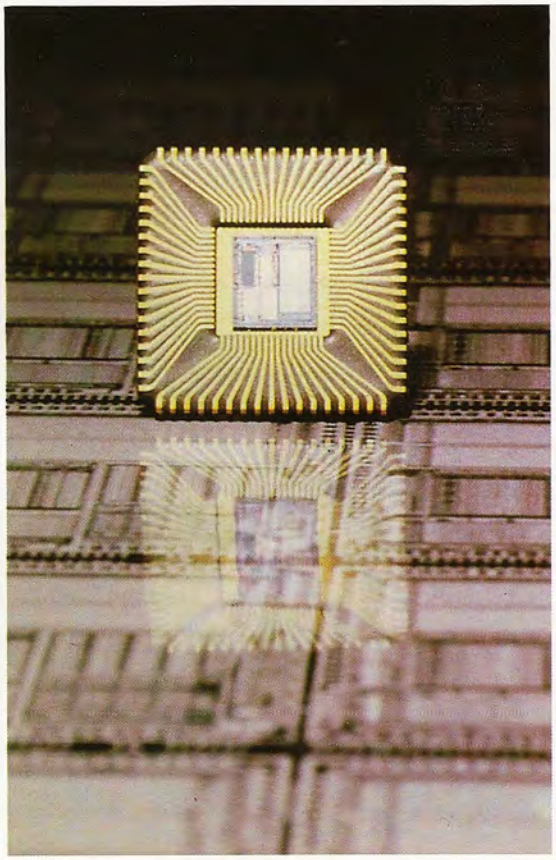


Kevin Jones

Cálculo trascendental
El Transputer, un nuevo desarrollo de Inmos, es un ordenador en un chip, equipado con un procesador de 32 bits, RAM y cuatro enlaces E/S en serie. El Transputer se considera como un componente a utilizar dentro de la arquitectura de ordenadores en paralelo mucho mayores. Los enlaces en serie permiten conectar entre sí en red varios Transputers, la topología de la cual vendrá determinada por la tarea para la que se haya diseñado el sistema

Ordenador en un chip

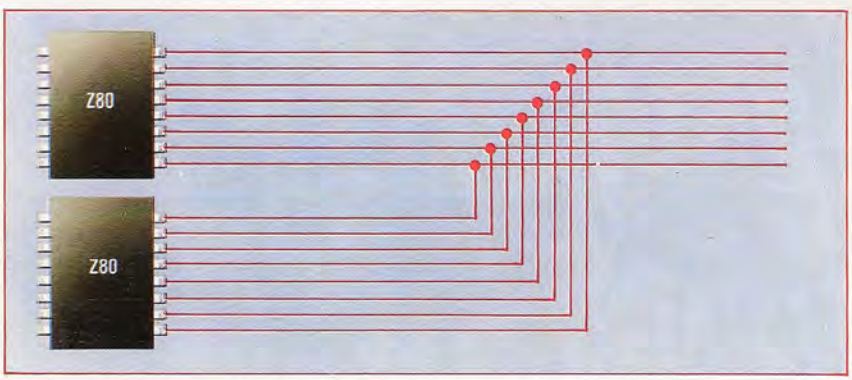
El lanzamiento del Inmos Transputer probablemente igualará o superará el impacto que supuso la irrupción del transistor en los años cincuenta



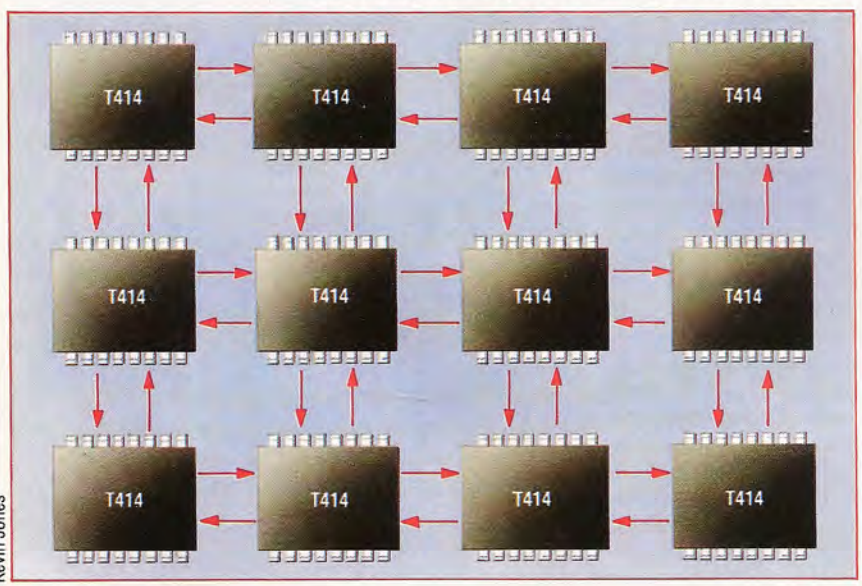
La invención del transistor en los años cincuenta abrió el camino para la fabricación masiva de ordenadores. Los microprocesadores de hoy en día se construyen a partir de centenares de miles de interruptores de transistor idénticos en el mismo trozo de silicio. El Inmos Transputer es un nuevo y revolucionario microprocesador destinado a utilizarse de forma similar al transistor: como un componente o bloque de construcción para sistemas más grandes (el nombre sugiere un híbrido de *trans*istor y com*puter*). Se pueden construir ordenadores en paralelo mediante la combinación de grandes cantidades de Transputers, que son tratados como "com-

ponentes programables" en vez de como CPU individuales y omnipotentes, ya que el Transputer incluye en un único chip todos los componentes de un sistema de ordenador. Cada Transputer posee una CPU, memoria y enlaces para comunicaciones en serie, todo en el mismo trocito de silicio. Ello significa que un único Transputer puede ejecutar un programa por sí mismo, sin necesidad de recursos externos (a excepción de potencia eléctrica y un reloj). Además, los Transputers pueden hablar entre sí mucho más fácilmente de lo que pueden hacerlo las CPU tradicionales, a través de sus enlaces en serie.

Un microprocesador tradicional, como el Z80, es el "jefe" en cualquier sistema en el cual se utilice. La comunicación con el mundo exterior se realiza a través de un bus en paralelo, siendo la propia CPU la que controla el acceso al mismo. Es difícil lograr que varios Z80 se hablen entre sí por dos razones. En primer lugar, físicamente es difícil diseñar una placa de circuitos en la cual muchos chips compartan un bus en paralelo; se necesita unir ocho cables cada vez que se realiza una conexión. Este problema se agrava, en lugar de remitir, con los chips más modernos de 16 y 32 bits. En segundo lugar, cada Z80 desea controlar el bus, de modo que se debe diseñar software para asegurar que dos procesadores no intenten hablar al mismo tiempo. Mientras se está comunicando a través del bus, el Z80 no puede llevar a cabo ninguna otra tarea.



Como cada Transputer posee cuatro enlaces en serie incorporados, puede hablar a otros cuatro chips.



Kevin Jones

Arquitectura Von Neumann

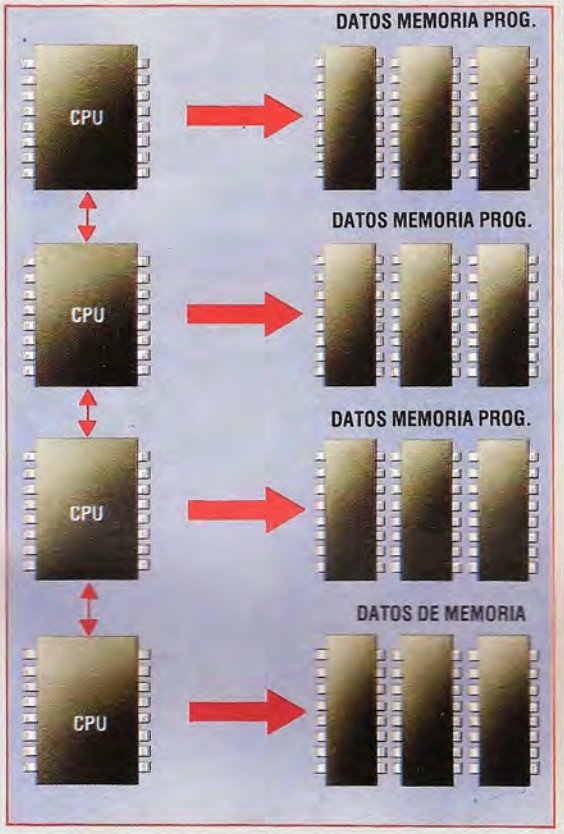
Todos los ordenadores que utilizamos en la actualidad se basan en los principios enunciados por John von Neumann en 1945. Un ordenador Von Neumann consta de una unidad central de proceso (CPU), conectada a alguna memoria que retiene tanto las instrucciones del programa que especifican lo que se ha de hacer como los datos sobre los cuales se hará. La CPU busca las instrucciones una después de la otra, o secuencialmente, y las ejecuta.

La arquitectura Von Neumann hizo posible que los ordenadores para fines generales se convirtieran en una propuesta práctica. La idea clave, de que los números en la memoria pueden representar tanto un programa como datos, liberó a los diseñadores que habían construido ordenadores exclusivos para tareas determinadas: descifrado de códigos o control de maquinarias. El problema radica en la forma secuencial en que una máquina Von Neumann busca y ejecuta sus instrucciones. Por más rápido que hagamos operar la CPU, no puede trabajar con mayor celeridad de lo que tarde en buscar sus instrucciones y datos en la memoria, y existen límites físicos en cuanto a la rapidez con que puede realizarlo. Así, la velocidad de comunicación entre un único procesador secuencial y su memoria se ha convertido en un cuello de botella. Se puede evitar diseñando ordenadores con más de una CPU y haciendo que todos trabajen al mismo tiempo, lo que se conoce como *proceso en paralelo*. Aun cuando la velocidad de comunicación con la memoria sea limitada, cien CPU que hablen con cien memorias pueden realizar en un tiempo dado cien veces el trabajo de que es capaz un solo procesador. Los ordenadores en paralelo ganan aún más velocidad porque cada CPU necesita hablar a menos memoria, y esto incrementa la velocidad máxima de comunicación, como vemos en el diagrama. El proceso en paralelo presenta dificultades que han impedido que se generalice su adopción. Para ejecutar un único programa en varias CPU, a cada procesador se le debe dar una parte de la tarea a cumplir, y debe realizar su parte sin entrar en colisión ni en competencia con sus vecinos. Si quisiéramos utilizar dos procesadores para calcular el valor de la expresión:

Esto permite construir matrices de dos o tres dimensiones de Transputers con un cableado mínimo.

Lo más importante, sin embargo, es que los enlaces del Transputer están diseñados teniendo en cuenta las comunicaciones, y no se requiere ningún software especial para evitar las colisiones. Un único Transputer es capaz de realizar muchas tareas al mismo tiempo; en particular, puede recibir mensajes desde sus cuatro enlaces simultáneamente sin tener que interrumpir lo que esté haciendo en ese momento.

Estas características de diseño permiten descomponer en partes un único programa y ejecutarlo en varios Transputers. Cada chip puede ejecutar su propia parte del programa en su propia memoria e informar a sus vecinos sobre los resultados enviándoles mensajes a través de los enlaces. Si necesita ejecutar más rápido una parte determinada del programa, sólo necesita añadir más Transputers.



(3+4)*(9 – 7+3)

cada uno de ellos tomaría una de las expresiones entre paréntesis y la calcularía, y luego uno de ellos realizaría la multiplicación final.

Es esencial prever cuál de las CPU llevará a cabo el paso final. Asimismo, la CPU 1 no debe intentar realizar el último paso hasta que la CPU 2 haya terminado con su parte del cálculo.

Es necesario que las CPU se hablen y se pasen resultados parciales y señales. Esta clase de problemas no se pueden tratar con facilidad mediante los lenguajes tradicionales, que dan por sentado que todo sucederá secuencialmente. Los intentos del pasado en cuestión de ordenadores en paralelo con frecuencia han resultado decepcionantes en rendimiento

Sin precedentes

Tecnología: El T414 se fabrica en tecnología CMOS de 1,5 micrones. Ello lo convierte en uno de los más densos y más complejos que se hayan construido nunca

Procesador: Un microprocesador de 32 bits capaz de ejecutar 10 millones de instrucciones por segundo (aproximadamente 10 veces más rápido que el Motorola 68000, p. ej.). Utiliza sólo 70 instrucciones que soportan el lenguaje OCCAM

Memoria en chip: RAM estática con un tiempo de acceso de 50 nanosegundos. Se utiliza para ejecutar pequeños procesos OCCAM; sustituye a los registros de un microprocesador convencional

Memoria fuera del chip: Se pueden direccionar hasta cuatro gigabytes de memoria externa, permitiendo utilizar el Transputer como chip CPU independiente en ordenadores personales. El procesador no ve diferencias entre memoria en chip y ajena al chip, salvo la velocidad de acceso

Enlaces en serie: Cada enlace es capaz de transferir 10 megabits por segundo en ambas direcciones; los cuatro enlaces pueden operar simultáneamente, dando una velocidad de datos total potencial de 80 Mbits por segundo. La comunicación es asincrónica, con una sincronización por hardware, de modo que los Transputers a cada extremo de un enlace no necesitan compartir la misma señal de reloj

Concurrencia: Un único T414 puede ejecutar múltiples procesos al mismo tiempo, controlado por un planificador de procesos incorporado. Todas las partes del chip operan concurrentemente entre sí, de modo que la parte del procesador puede realizar cálculos o acceder a la RAM mientras se envían mensajes en los enlaces

Otros productos: El T414 es el primero de una familia de productos Transputer compatibles en los que diversas funciones para fines especiales reemplazarán parte de la memoria o enlaces. Dos de ellos ya diseñados son el G213 Graphics Processor y el M212 Disk Controller

La empresa: Inmos se creó en 1978 con la ayuda del gobierno británico y ahora forma parte del grupo Thorn EMI. El diseño y desarrollo del Transputer y el OCCAM se realizó en Bristol, y el Transputer se fabrica en Newport (Gales)

La clave que permite la utilización del Transputer de este modo reside en el lenguaje utilizado para programarlo. El Transputer no posee un lenguaje ensamblador como los microprocesadores tradicionales, sino que ejecuta un lenguaje diseñado *ad hoc* denominado OCCAM, que está confeccionado especialmente para escribir programas paralelos. Un programa en OCCAM se divide en partes llamadas *procesos*, que son algo así como subrutinas o procedimientos en BASIC o en PASCAL. La gran diferencia es que los procesos se pueden ejecutar al mismo tiempo, así como en serie.

Además de almacenar valores como variables, como un lenguaje "convencional", el OCCAM puede comunicar valores a través de canales. De modo que diferentes procesos que se están ejecutando al mismo tiempo pueden comunicarse los resultados entre sí a través de un canal. Además, los canales OCCAM se sincronizan automáticamente, de modo que es imposible que se produzca la comunicación hasta que ambas partes no estén preparadas. Asimismo, el OCCAM posee la propiedad vital de no discernir si los procesos que componen un programa se están ejecutando en el mismo Transputer o en Transputers distintos; un canal puede no ser más que un "buzón" de la memoria en el primer caso, o un trozo de cable en el segundo.

El Transputer ofrece un enfoque nuevo y radical al diseño de sistemas de ordenador de gran rendimiento. Hace borrosa la distinción entre diseño de hardware y diseño de software, porque todo lo que se puede describir mediante un proceso en OCCAM se puede implementar en un chip. Alcanzar objetivos de rendimiento se convierte en una cuestión de elegir el tamaño y la topología (o "forma") adecuados de la red de Transputers para abordar el problema. Quizá el futuro ofrezca anillos, *donuts*, cubos y otras configuraciones.

Para superar la crisis

Los lenguajes "funcionales" como el HOPE se presentan como una posible solución a la crisis del software

Muchos científicos de ordenadores están experimentando con lenguajes de programación *funcionales* como una salida para la "crisis de software" (véase la página contigua). Los programas escritos en lenguajes funcionales tienen la propiedad de que se puede comprender cualquier parte del programa sin remitirse al resto de éste; no dependen de la historia u orden de ejecución de sus partes.

Esta "transparencia" se obtiene mediante la supresión del uso de asignación a variables. Algunos lenguajes funcionales prohíben directamente las variables y emplean "funciones", que devuelven valores para uso inmediato. El principio se puede ilustrar incluso en BASIC mediante estos dos programas:

```
10 X=57
20 A=SIN(X)
30 B=LOG(A)
40 C=SQRT(B)
50 PRINT C
```

```
10 PRINT SQRT(LOG(SIN(57)))
```

En el segundo programa no se utilizan variables, pero el resultado es el mismo que en el primero. El programa se compone exclusivamente de la aplicación de funciones a los resultados de otras funciones.

El único lenguaje funcional conocido es la forma "pura" original del LISP, pero la mayoría de los dialectos modernos han añadido características no funcionales, tales como la asignación con SETQ y bucles para acelerar la ejecución en ordenadores convencionales.

Un buen ejemplo de un lenguaje moderno puramente funcional es HOPE, creado en la Universidad de Edimburgo. Los programas en HOPE se escriben definiendo funciones, como en LISP. Cada función consta de una serie de ecuaciones, que indican a la función qué valor devolver para cada forma posible de sus argumentos.

En los programas en HOPE se admiten variables, pero sus valores no se pueden cambiar por asignación; la única forma de otorgar un valor a una variable es utilizándola como un *patrón* que concuerde con los argumentos de la función. Por ejemplo, una función para calcular el cuadrado de un número se podría escribir así:

```
dec cuadrado: num—> num;
---cuadrado(x)<=x*x;
```

donde dec alude a *declarar* y da comienzo a la definición; num—>num dice que la función cuadrado toma un número como su argumento, y devuelve uno con su valor (en HOPE, como en PASCAL, los valores poseen un tipo). La ecuación (iniciada mediante ---) dice que el cuadrado de cualquier número es ese número multiplicado por sí mismo; el símbolo <= significa "se define como" o "podría sustituirse por". El patrón x del lado izquierdo se empareja con cualquier número dado como argumento. El lado derecho de una ecuación debe ser una expresión, y puede utilizar solamente variables tomadas del patrón.

Nosotros utilizamos la función entrando, pongamos por caso, cuadrado(4); y HOPE responde con 16: num, dando el tipo así como el valor del resultado. Por supuesto, esta función se puede emplear para definir otras funciones. Un ejemplo algo más complejo es una función en HOPE para calcular el factorial de un número:

```
dec fact:num—>num;
--- fact(0)          <=1;
--- fact (succ(n))   <=(succ(n)*fact(n));
```

Las dos ecuaciones definen el valor de la función para todos los casos posibles (el tipo num representa enteros positivos, de modo que no se plantea el caso negativo). En el caso de que su argumento sea 0, entonces devuelve el valor 1. En cualquier otro caso, el factorial de "uno más que n" es "una más que n" veces el factorial de n. Ésta es una definición recursiva, porque fact está definida desde el punto de vista de ella misma; los lenguajes funcionales utilizan la recursión en lugar de la iteración.

El orden de las ecuaciones es irrelevante, y el siguiente funciona igualmente bien:

```
dec fact:num—> num;
--- fact(succ(n))<=(succ(n)*fact(n));
--- fact(0)<=1;
```

La función succ (*successor*: sucesor), que devuelve un número de uno más que su argumento, está incorporada en el HOPE y se la conoce como una función *constructora*.

En HOPE, todos los tipos de datos se construyen a través de su función constructora. Cuando escribimos una constante como 3, estamos evaluando una función llamada 3 cuyo valor es 3, pero que en realidad constituye una versión abreviada de expresión succ(succ(succ(0))).

Para representar múltiples objetos del mismo tipo, el HOPE utiliza *listas* en lugar de matrices. Las listas se escriben entre corchetes, de modo que [1,2,3,4] es una lista de cuatro números. Una lista se puede emparejar con un patrón, como x::y, donde x se empareja con el primer elemento de la lista e y se empareja con todo el resto. Utilizando la lista [1,2,3,4], x sería 1 e y la lista [2,3,4]. El símbolo ::, pronunciado "cons", es la función constructora para listas.

Las series de texto se representan como listas de caracteres y alternativamente se pueden escribir

entre comillas, de modo que "juan" significa lo mismo que ['j','u','a','n']. Una función para contar la cantidad de letras de una palabra podría ser:

```
dec contadorletras:list char—>num;
--- contadorletras(nil)<=0;
--- contadorletras(x::y)<=contadorletras(y)+1:
```

donde nil significa una lista vacía. Se utiliza así:

```
contador letras("aardvaark");
9:num
```

En HOPE los tipos son mucho más flexibles que en PASCAL, y es posible escribir funciones que puedan trabajar sobre cualquier tipo. Por ejemplo:

```
typevar:alpha

dec contadorlista:list(alpha)—>num;
--- contadorlista(nil) <=0;
--- contadorlista(x::y) <=contadorlista(y)+1;
```

contará los elementos de una lista de cualquier tipo y se podría utilizar en lugar de contadorletras.

Los programadores pueden definir sus propios tipos de datos, de complejidad cualquiera, y éstos se pueden pasar como argumentos o devolver como valores desde funciones. Incluso es posible pasar funciones como argumentos y devolverlas como resultados, lo que permite escribir programas sumamente potentes.

La crisis de software

La "crisis de software" es una forma drástica de describir el hecho de que, mientras que los ordenadores se vuelven cada vez más veloces y baratos, el costo de escribir software para ellos continúa aumentando y, lo que es aun peor, también aumenta el costo del mantenimiento del software existente.

Las raíces de la crisis de software están en la naturaleza de los lenguajes de programación convencionales. Si bien lenguajes modernos estructurados como el PASCAL han representado alguna mejora, todavía resulta difícil comprender lo que hace un programa simplemente leyendo su código fuente. En el caso del lenguaje ensamblador y lenguajes no estructurados, como el BASIC o el FORTRAN, el problema reviste aún mayor gravedad. Incluso puede ser que el propio autor de un programa tenga problemas para leer su trabajo más adelante, mientras que las otras personas que han de mantener tales programas tienen ante sí una tarea colosal. Respecto a los grandes sistemas de software, como los utilizados en la exploración espacial o en las instalaciones de defensa, es dudoso que alguien comprenda el programa en su totalidad. Gran parte del problema de la lectura de los programas surge del hecho de ser "dependientes de la historia". A diferencia de una fórmula matemática, el texto de un programa para ordenador no siempre transmite toda la información requerida para entender lo que hace. En un programa convencional, el valor de una variable depende de la "historia" previa de ejecución del programa. Tomemos a modo de ejemplo este programa en BASIC BBC:

```
 10 FLAG%=0
 15 REM
 20 DEF FNx(A%)
 30    FLAG%=1
 40    =2*A%
 45 REM
 50 DEF FNy(A%)
 60    LOCAL B%
 70    IF FLAG% THEN B%=3 ELSE B%=4
 80    =B%*A%
 85 REM
 90 PRINT FNy(2)+FNx(1)
100 PRINT FNy(2)+FNx(1)
```

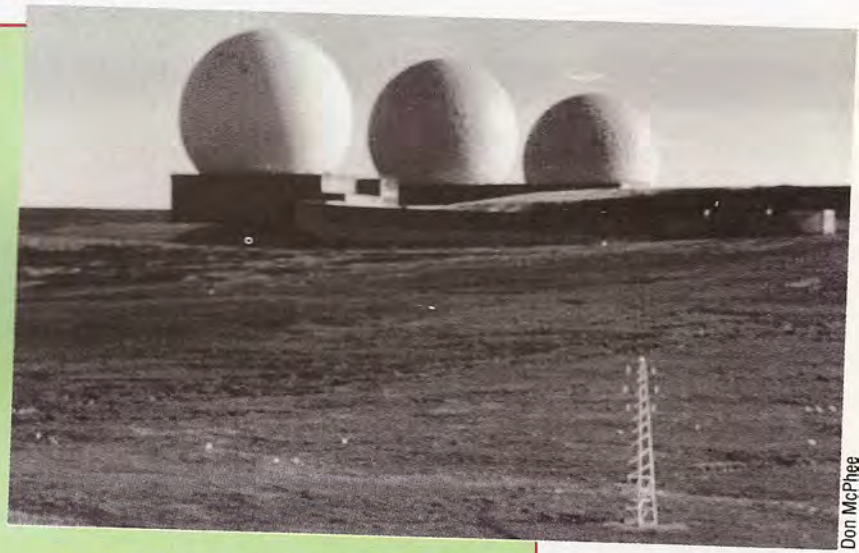
Las dos sentencias PRINT imprimirán los valores 8 y 10, respectivamente; el valor de FNy(2) y FNx(1) no es el mismo en ambos casos. Cambiando el orden de los términos también se altera el valor de la expresión, de modo que FNy(2)+FNx(1) no es lo mismo que FNx(1)+FNy(2). Para comprender la mayoría de los programas, hemos de "ejecutarlos" mentalmente utilizando un lápiz y un trozo de papel. Por el contrario, las sentencias matemáticas son independientes de la historia. Sabemos que las expresiones:

```
(3+4)*(6-2)=(4+3) * (6-2)=7*4=28
```

tienen todas el mismo valor, porque las "leyes" matemáticas nos dicen que 3+4 es lo mismo que 4+3, que 7 se puede sustituir legalmente por cualquiera de las dos expresiones, y que el valor de una expresión es el valor de sus componentes. Los programas escritos en lenguajes para ordenador convencionales no poseen ninguna de estas propiedades.

Si se pudiera lograr que los programas poseyeran estas propiedades matemáticas, se obtendrían varios beneficios inmensos y tendríamos una posible solución para la crisis.

- Los programas serían legibles. El texto del programa expresaría su significado sin ninguna necesidad de rastrear la historia de su ejecución.
- Los programas se podrían demostrar (como los teoremas) como correctos para todas las entradas posibles, en lugar de comprobarse meramente para un puñado de las posibles entradas. Esta prueba sería automatizada y la llevaría a cabo el ordenador.
- Programas ineficaces pero correctos se podrían transformar en otros más eficaces utilizando leyes como aquellas que rigen para las expresiones matemáticas. Este proceso podría automatizarse



Don McPhee

A la espera de una solución Producir software para sistemas basados en grandes ordenadores, tales como el sistema de alerta rápida de Fylingdale, en North Yorkshire (Gran Bretaña), constituye un problema. Intentar conectar y racionalizar el trabajo de muchos programadores en un único gran trozo de código utilizando las técnicas de programación tradicionales es muy difícil, y las ulteriores modificaciones pueden conducir a una situación de difícil salida. Se espera que los lenguajes funcionales proporcionen la solución definitiva a este problema

En un lugar de Escocia...
El lenguaje funcional HOPE lo desarrollaron R. M. Burstall y D. B. MacQueen, del Departamento de Informática de la Universidad de Edimburgo (Escocia), con la colaboración de D. T. Sannella, de Bell Labs, de Estados Unidos. El nombre del lenguaje hace referencia a Hope Square, en Edimburgo, donde está situado el Departamento de Informática

Por las mismas líneas

Uno de los grandes atractivos del estilo de programación funcional es que se presta a la ejecución en paralelo. Ello se debe a que las partes de un programa funcional son independientes entre sí, gracias a la falta de variables.

En lenguajes convencionales, el empleo de variables compartidas por distintas partes del programa supone enormes problemas para un ordenador en paralelo. Supongamos que hemos inventado una forma de BASIC en paralelo y que estos dos programas se están ejecutando simultáneamente:

```
10   A=0                    10   B=0
20   FOR X=1 TO K           .
.                           .
.                           .
100  NEXT X                 .
2000 A=B+56                 2000 PRINT A
.                           .
.                           .
```

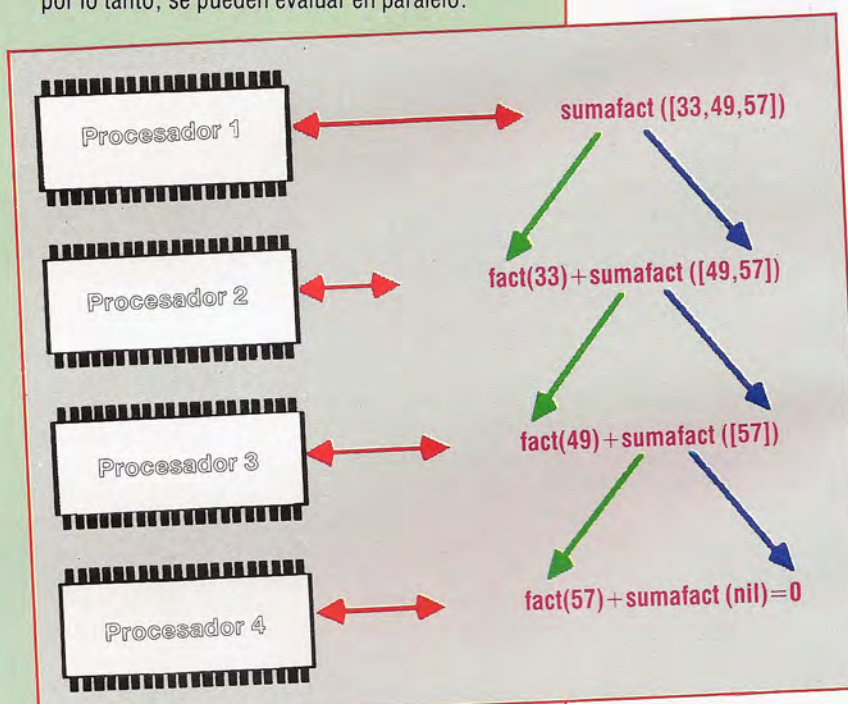
El valor que se imprima para A dependerá de que el primer programa haya llegado o no para entonces a su línea 2000. Si no ha llegado, A todavía es 0, de lo contrario habrá cambiado. Pero el tiempo de ejecución del primer programa varía con K y, por tanto, no podemos estar seguros del efecto que tendrá el programa. En un lenguaje como el HOPE, sin asignación a variables, no se presenta este problema. Tomemos a modo de ejemplo un programa para calcular la suma de los factoriales de una lista de números. Ya hemos visto la función factorial:

```
    dec fact:num—> num;
--- fact(0)       <=1;
--- fact(succ(n))<=(succ(n)*fact(n));
```

Ahora podemos escribir sumafact utilizando fact:

```
    dec sumafact:list(num)—>num;
--- sumafact(nil)<=0;
--- sumafact(x::y)<=fact(x)+sumafact(y);
```

Aquí decimos que la sumafact de una lista vacía es cero, de lo contrario se forma mediante la adición del factorial del primer elemento a la sumafact del resto. Los dos términos del lado derecho de la segunda ecuación son independientes entre sí y, por lo tanto, se pueden evaluar en paralelo.



Caroline Clayton

En cada etapa se puede traer un nuevo procesador para calcular el término fact, y el primer procesador suma todos los resultados parciales cuando la recursión se "desenrolla" hasta llegar a 0. El ordenador en paralelo Alice que se está desarrollando en Imperial College ejecuta el HOPE de esta manera, utilizando un anillo de Transputers. Cualquier procesador de anillo que esté ocioso puede asumir uno de los cálculos parciales, que se ponen en "oferta" haciendo circular sus descripciones alrededor del anillo

Tamaño económico

Desarrollaremos una de las técnicas de compresión de textos mediante un programa de propósito general

El método de compresión que empleamos en nuestro programa es una implementación reducida del algoritmo de compresión de cuatro bits que describimos en el capítulo anterior. Trabaja traduciendo el texto a bloques de cuatro bits, cada uno de los cuales es o un carácter utilizado comúnmente, o bien un código que proporciona información relativa a los cuatro bits siguientes. El significado de los distintos códigos de cuatro bits se refleja en el diagrama *Valores de nibbles*.

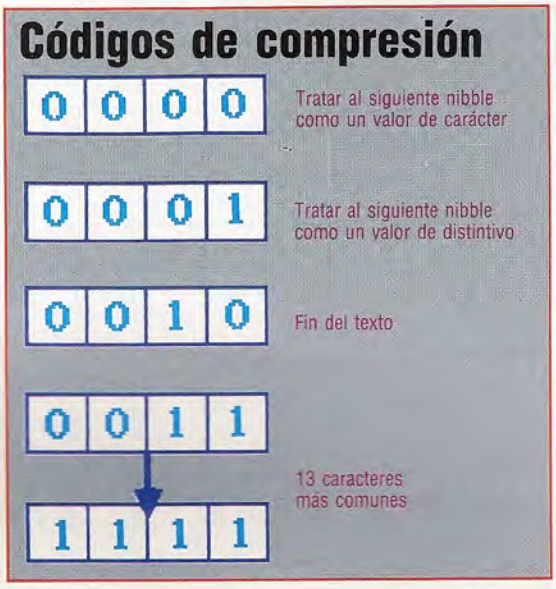
Tal como está, el programa sólo comprimirá texto compuesto en letras mayúsculas, espacios, comas y puntos. Asimismo, soportará distintivos, y 16 de éstos se han incluido en el programa para ilustrar esta facilidad.

El programa exige que el usuario especifique la dirección de la serie de entrada y la dirección en donde se ha de colocar la serie de salida. La serie se almacena como un byte de cuenta, seguido por los bytes de la serie. La longitud de la serie se limita a 255 bytes.

El programa es relativamente sencillo de modificar, de modo que los programadores de lenguaje máquina no tendrán problemas para convertirlo en una implementación completa. La modificación más eficaz sería incluir una facilidad para hacer referencia a una tabla de distintivos de 255 elementos. La misma le permitiría incluir tanto caracteres en mayúsculas como en minúsculas, así como más distintivos para palabras comunes. Esta facilidad se podría introducir fácilmente destinando un valor nibble de tres para significar "tratar a los ocho bits siguientes como un desplazamiento en una tabla de 255 elementos". Se insertaría antes de la comprobación de distintivos de ocho bits y esencialmente sería igual que la rutina existente, exceptuando la tabla que direcciona.

Si decide introducir esta modificación, debe reacomodar las tablas de modo que los valores de caracteres de cuatro bits representen a las letras *minúsculas* más comunes (más caracteres de espacio y salto de línea). Las siguientes letras minúsculas más comunes van en la tabla de ocho bits, y las letras menos comunes (más mayúsculas y distintivos) se colocan en la tabla nueva.

La dirección ORG del programa se puede cambiar para adecuarla a diferentes micros. El programa se ha de ensamblar, y el código objeto almacenar en cinta o disco como preparación para el próximo capítulo, en el que ofreceremos el programa en BASIC para activar la utilidad.



Valores de nibbles
Nuestro programa codifica el texto en secuencias de cuatro bits (nibbles), cuyo significado se refleja en el diagrama. Observe que sólo se utilizan como "señales" los valores del 0 al 2. El programador, sin embargo, podría incluir el valor 3 como una señal de que los dos nibbles siguientes se han de tomar para indicar un desplazamiento en una tabla de 255 elementos, permitiendo, en consecuencia, la inclusión de caracteres en minúsculas y más distintivos

Programa compresor de textos

Nuestro programa de compresión de textos se ofrece en dos partes. Aquí presentamos un listado ensamblador para máquinas Z80, y en el próximo capítulo proporcionaremos una versión 6502. Asimismo, para ambas implementaciones se proporcionarán cargadores de BASIC, para aquellos lectores que no posean ensambladores

```
        org     30000

        jr      start           ;saltar el espacio de datos

string: dw      0       ;cargar con dirección serie de entrada
output: dw      0       ;entrar dirección para serie de salida
status: db      0       ;retorno estado, señales 0 OK
mask:   db      0       ;
len:    db      0               ;espacio de almacenamiento para longitud entrada

start:  ld      hl,(string)     ;tomar dirección de la serie a comprimir
        ld      a,(hl)          ;tomar longitud serie de entrada
        ld      (len),a         ;almacenarla
        inc     hl              ;apuntar al comienzo de la serie
        ld      de,(output)     ;apuntar al espacio de salida
        push    de              ;guardar comienzo de salida
        inc     de              ;dejar 1 espacio para cuenta
        ld      (output),de     ;guardar dirección de salida hasta que
                                ;tengamos algo para almacenar
        ld      a,255
        ld      (mask),a        ;tratar primero nibble izquierdo

nchar:  ld      a,(hl)          ;tomar byte con el que trabajar
        call    check           ;asegurarse de que esté en la gama
        jr      nz,badchar      ;jr si carácter no permitido
        call    token           ;comprobar si comienzo de un distintivo
        jr      z,gottoken      ;hallado distintivo, procesarlo
        call    fourbit         ;comprobar caracter de 4 bits
        jr      z,got4bit       ;el carácter es de 4 bits, saltar
        call    eightbit        ;tomar valor carácter 8 bits
        push    af              ;no hay necesidad de comprobar-guardar valor
        ld      a,0             ;mientras indicador de 'car de 8 bits'
```

```
        call    writenib        ;..salida
        pop     af              ;restaurar 2ºnibble de val.8 bit
got4bit: call   writenib        ;salida valor del 2ºnibble de 8 bits,o
                                ;del 1er nibble de 4 bits
rejoin: inc     hl
        ld      a,(1en)         ;volver a tomar longitud actual
        dec     a               ;menos uno para car. recién procesado
        ld      (len),a         ;volver a poner nueva longitud
        and     a
        jr      naz,nchar       ;jr para proc. car. siguiente si existe
        ld      (status),a      ;señal terminado ok, a será=0
        ld      a,2             ;2=fin del texto comprimido
        call    writenib        ;escribirlo
        ex      de,hl           ;dirección de salida final en hl
        pop     de              ;tomar dirección serie de salida
        ld      a,(mask)        ;tomar valor máscara
        and     a
        jr      z,nodec         ;si máscara cero entonces valor final ok
        dec     hl              ;ignorar byte final, es nulo
nodec:  and     a               ;limpiar acarreo para resta
        sbc     hl,de           ;tomar longitud
        ld      a,l             ;tomar longitud - no más de 255!
        ld      (de),a          ;almacenar cuenta al comienzo de salida
        ret                     ;retornar

badchar: pop    de              ;limpiar pila
        ld      a,255
        ld      (status),a      ;fallo de señal
        ret
gottoken:                       ;
        push    af              ;guardar nibble distintivo
        ld      a,1             ;viene distintivo señal
        call    writenib
        pop     af              ;restaurar distintivo
        call    writenib        ;salida distintivo
        jr      rejoin

        ;subrutina para comprobar comienzo de nuevo distintivo
token:  push    hl              ;guardar dirección de entrada
        ex      de,hl           ;poner dirección de entrada en de
        ld      hl,tktable      ;que hl apunte a tabla de distintivos
        ld      a,(len)         ;poner longitud restante en a
        ld      c,a             ;y transferirla a c
        ld      b,0fh           ;inicializar número distintivo
tok1:   ld      a,(hl)          ;tomar longitud de distintivo actual
        and     a
        jr      z,notfound      ;si longitud=0 luego findelatabla
        push    de              ;guardar estos punteros por si
        push    hl              ;son necesarios
        inc     hl              ;apuntar a primer car.de distintivo
        push    bc              ;guardar número de distintivo
        ld      b,a             ;cuenta en b
tok2:   ld      a,(de)          ;tomar carácter de entrada
        cp      (hl)            ;comparar con carácter distintivo
        jr      nz,nxttoken     ;si no concuerda olvidar este distintivo
        inc     hl              ;apuntar al siguiente carácter
        inc     de
        dec     b               ;reducir cuenta caracteres restantes
        jr      nz,chkchars     ;si b no es cero entonces saltar
        ld      a,c
        ld      (len),a         ;almacenarlo
        pop     bc              ;hallado distintivo-restaurar n.º distintivo
        pop     hl              ;limpiar pila
        pop     hl              ;
        pop     hl              ;
        ex      de,hl           ;poner nueva dirección de entrada en hl
        dec     hl              ;y dec para mantener ok el bucle principal
        xor     a               ;establecer flag a cero para indicar éxito
        ld      a,b             ;poner número distintivo en a
        ret
chkchars:
        dec     c               ;reducir cuenta caracteres restantes
        jr      nz,tok2         ;si aún hay caracteres para comprobar volver
nxttoken:
        pop     bc              ;restaurar caracteres restantes+n.º distintivo
        dec     b               ;siguiente número de distintivo
        pop     hl              ;volver a poner puntero en número distintivo
                                actual
        ld      a,(hl)
        ld      e,a             ;longitud en de
        ld      d,0
        inc     hl              ;saltar cuenta
        add     hl,de           ;ahora hl apunta a cuenta siguientes
                                distintivos
        pop     de              ;restaurar dirección de entrada
        jr      tokl            ;saltar para comprobar siguiente distintivo
notfound:
        pop     hl              ;restaurar dirección de entrada original
        ld      a,(hl)          ;restaurar carácter
        or      a               ;restaurar flag a cero para indicar fracaso
        ret

        ;subrutina para comprobar caracteres de 4 bits

fourbit: push   hl              ;guardar dirección de entrada
        ld      hl,tab4bit      ;apuntar hl a tabla de caracteres
        call    tabscan         ;explorar tabla para valor
        pop     hl              ;restaurar dirección de entrada
        ret                     ;volver

        ;subrutina para comprobar caracteres de 8 bits
eightbit:

        push    hl              ;guardar dirección de entrada
        ld      hl,tab8bit      ;apuntar hacia tabla caracteres de 8 bits
        call    tabscan         ;explorar tabla
        pop     hl              ;restaurar dirección
        ret                     ;return

        ;rutina de exploración de tabla de propósito general

tabscan: ld     b,0fh           ;16 caracteres en cada tabla
                                ;(uno se numera 0)
tabsc2: cp      (hl)            ;comprobar carácter
        jr      z,tabsc3        ;retornar con 0 establecido si concordancia
        inc     hl              ;comprobar siguiente elemento de la tabla
        djnz    tabsc2          ;bucle hasta que ningún carácter en la tabla
        or      a               ;restablecer bandera a 0 para indicar ninguna
                                pareja
                                ;a contiene código car <>0
        ret                     ;volver
tabsc3: ld      a,b             ;poner nibble en a
        ret

        ;rutina para comprobar caracteres válidos
        ;sólo se permiten espacio , . ; : y letras mayúsculas
check:  cp      ' '             comprobar espacio primero
        ret     z
        cp      ','             ;y coma
        ret     z
        cp      '.'             ;por último punto
        ret     z
        cp      'A'             ;asumir abecedario contiguo
        jr      c,nogood        ;saltar si 'menor que' A
        cp      'Z'+1
        jr      nc,nogood
        ld      c,a
        xor     a               ;establecer cero para indicar carácter ok
        ld      a,c
        ret
nogood: ld      a,255
        and     a               ;restablecer cero para indicar fracaso
        ret

writenib:
        ld      c,a             ;guardar nibble
        ld      a,(mask)        ;
        ld      de,(output)     ;tomar dirección de byte para escribir
        and     a               ;comprobar valor máscara
        jr      nz,left         ;saltar si el nibble requiere desplazamiento
        ld      a,(de)          ;tomar nibble antiguo
        or      c               ;insertar nibble nuevo
        ld      (de),a          ;y volver a almacenarlo
        inc     de              ;apuntar al siguiente byte
        ld      (output),de     ;guardar dirección
        ld      a,255
        ld      (mask),a        ;apuntar nibble izquierdo próxima vez
        ret
left:   ld      a,c             ;poner valor en a
        sla     a               ;desplazarlo
        sla     a
```

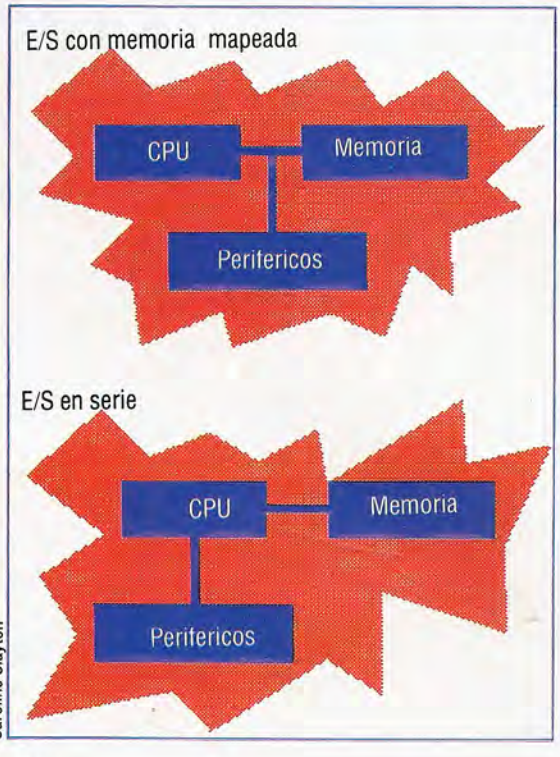
```
        sla     a
        sla     a
        ld      (de),a          ;y almacenarlo
        xor     a
        ld      (mask),a        ;señalar otro nibble la próxima vez
        ret

;programa de ampliación de texto 4 bits para Z80, independiente de la máquina

expand: ld      hl,(string)     ;string contiene la dirección de la serie a
                                ;expandir
        inc     hl              ;apuntar hl al comienzo de la serie
        ld      (string),hl     ;volver a poner comienzo nueva serie
        ld      de,(output)     ;apuntar al comienzo espacio de salida
        push    de              ;guardar para final
        ld      a,255
        ld      (mask),a        ;tratar primer nibble izquierdo

nextnib: call   getnib          ;tomar siguiente nibble de entrada
        cp      2               ;es este el final
        jp      z,fin           ;de ser así saltar para terminar
        jp      nc,exp4bit      ;si nibble>2 procesar 4 bits
        and     a
        jp      z,exp8bit       ;si nibble=1 procesar car.8 bits
        call    getnib          ;tomar nibble para ampliar a distintivo
        call    tokscan         ;hallar el que deseamos
tokloop: ld     b,a             ;cuenta en b
tklp1:  ld      a,(hl)          ;tomar carácter distintivo
        inc     hl              ;apuntar al siguiente carácter
        call    outchar         ;salida a serie nueva
        djnz    tklp1           ;bucle para todos los cars. distintivo
        jr      nextnib         ;saltar hacia atrás para siguiente nibble
                                entrada
tokscan: ld     hl,tktable
        ld      b,a             ;inicializar b
        ld      a,0fh
        sub     b
        ld      b,a             ;cuenta en b
        inc     b
tokscl: ld      a,(hl)          ;tomar longitud de este distintivo
        inc     hl              ;apuntar al primer carácter de distintivo
        dec     b               ;reducir cuenta
        ret     z               ;retornar si es el que queremos
        ld      e,a             ;saltar este distintivo
        ld      d,0
        add     hl,de
        jr      toksc1          ;saltar hacia atrás

fin:    ld      hl,(output)     ;poner fin de serie de salida en hl
        pop     de              ;comienzo de la serie de salida
        and     a               ;limpiar acarreo
        sbc     hl,de           ;longitud en hl
        ld      a,l             ;la longitud no debe ser > 255
        ld      (de),a          ;poner cuenta en lugar correcto
        ret                     ;retornar al sistema

exp8bit: call   getnib          ;tomar siguiente nibble para car.8 bits
        ld      hl,tab8bit      ;que hl apunte a la tabla correcta
        call    index           ;tomar el carácter derecho
        call    outchar         ;ponerlo en la serie de salida
        jr      nextnib         ;volver por más

exp4bit: ld     hl,tab4bit      ;apuntar a tabla 4 bits
        call    index           ;hallar carácter
        call    outchar         ;hacerlo salir
        jr      nextnib         ;volver a pasar

index:  ld      e,a             ;inicializar e
        ld      a,0fh
        sub     e               ;poner desplazamiento en a
        ld      e,a             ;poner en e
        ld      d,0
        add     hl,de           ;índice en tabla
        ld      a,(hl)          ;tomar carácter
        ret

outchar: ld     de,(output)     ;tomar última dirección producida para
        inc     de              ;apuntar a nueva dirección
        ld      (de),a          ;almacenar carácter
        ld      (output),de     ;volver a poner nueva dirección
        ret

getnib: ld      a,(mask)        ;tomar valor máscara
        ld      hl,(string)     ;tomar dirección de byte de entrada
        and     a
        ld      a,(hl)          ;tomar byte
        jr      nz,shfta        ;saltar si requerido nibble izquierdo
        and     0fh             ;enmascarar nibble derecho
        inc     hl              ;apuntar a nuevo byte
        ld      (string),hl     ;almacenar para próxima vez
        ld      c,a
        ld      a,255           ;indica nibble izquierdo próxima vez
        ld      (mask),a
        ld      a,c
        ret
shfta:  sra     a               ;desplazar nibble derecho por arriba
        sra     a
        sra     a
        sra     a
        and     0fh             ;desenmascararlo
        ld      c,a
        xor     a               ;indica nibble derecho próxima vez
        ld      (mask),a
        ld      a,c
        ret
        ;tabla de valores
        ;tabla de cuatro bits

tab4bit: db     ''              ;carácter espacio
        db      'E'
        db      'T'
        db      'A'
        db      'O'
        db      'N'
        db      'R'
        db      'I'
        db      'S'
        db      'H'
        db      'D'
        db      'L'
        db      'F'
        db      0               ;valores ficticios para llenar tabla
        db      0
        db      0

        ;caracteres de 8 bits
tab8bit: db     'C'
        db      'M'
        db      'U'
        db      'G'
        db      'Y'
        db      'P'
        db      'W'
        db      'B'
        db      'V'
        db      'K'
        db      'X'
        db      'J'
        db      'Q'
        db      'Z'
        db      ','
        db      '.'

        ;tabla de distintivos
tktable: db     3,'THE'
        db      4,'THIS'
        db      4,'THAT'
        db      2,'IF'
        db      3,'YOU'
        db      2,'ME'
        db      3,'WAS'
        db      2,'HE'
        db      3,'SHE'
        db      4,'THEY'
        db      2,'OF'
        db      2,'IT'
        db      2,'IS'
        db      3,'FOR'
        db      2,'ON'
        db      2,'TO'
        db      0               ;marca de fin de tabla
        end
```

Bit... bit... bit

¿Cuál es la información necesaria para que la CPU saque el mayor partido del mapa de memoria en las comunicaciones de E/S? Veámoslo

Hay dos métodos básicos para seleccionar y comunicarse con dispositivos de E/S:

- *Memoria mapeada*: Este método conecta el bus central de comunicaciones (al que están conectadas la CPU y la memoria) con los dispositivos de E/S. Mediante la adecuada combinación electrónica, los periféricos son sencillamente agregados a este bus. Lo que significa que un periférico se convierta en memoria mapeada, dado que el procesador puede leer y escribir ahora en el dispositivo como si se tratara de una memoria normal. Este procedimiento presenta la desventaja de ralentizar la velocidad en el bus, pues el periférico entra en competencia para obtener sus servicios.
- *E/S aisladas*: En este método se dispone de un bus especial para los dispositivos de E/S, con lo que se incrementa la velocidad de transferencia de los datos respecto al procedimiento anterior. El inconveniente está en que hay que conocer las nuevas instrucciones de E/S que rigen los dispositivos.



Caroline Clayton

Líneas de los buses
Este dibujo muestra la diferencia entre una E/S en serie, que emplea un bus de periférico especial para intercambiar información, y una E/S de memoria mapeada, donde los periféricos toman los datos del mismo bus con el que se accede a la memoria. En este último caso ciertas direcciones habrán de ser reservadas para los periféricos, a ellas puede acceder el ordenador de igual modo que a la RAM o a la ROM

Llegados a este punto no puede sorprendernos saber que el 68000 opta por el método de memoria mapeada. Ya estudiamos en otro lugar la estructura del microordenador. Vimos entonces que algunos dispositivos se conectaban al mismo bus que la memoria y que la CPU, lo que equivalía a decir unas E/S mapeadas.

Veamos ahora los tres tipos principales de información que el procesador exige, y que es mapeado en la memoria.

- *Estado*: Dado que los periféricos tardarán algún tiempo para realizar la operación deseada (imprimir un carácter, leer el teclado, p. ej.), es necesaria una información de su estado para impedir que el periférico reciba órdenes antes de haber acabado su última operación. También es necesaria otra información no directamente referida a la operación pero que puede interesar, supongamos, al modo de operación o a cualquier condición de fallo como quedarse sin papel en la impresora.
- *Control*: Necesitamos un registro que nos permita configurar o activar el periférico que estamos usando (p. ej., ordenar a una unidad de disco que lea un bloque de datos, o configurar un canal de comunicación a una velocidad específica de transmisión).
- *Datos*: Necesitamos, por último, poder leer o escribir datos en el periférico y retener la información allí hasta que se concluya la operación y quede listo para recibir nueva información.

Chips de E/S

En el campo de las señales en el bus principal del ordenador, la operación detallada de los chips de interface puede resultar bastante complicada. Para evitar esta complicación (desde un punto de vista de la programación) se utilizan chips de interface de usos específicos. Motorola proporciona dos chips, uno para el control de dispositivos en serie (ACIA) y otro para el control en paralelo (PIA).

Los dispositivos en serie se conectarán al chip mediante dos hilos solamente para envío y dos para recepción de información; los bits que constituyen los bytes de datos llegan en serie uno detrás de otro. En el caso de dispositivos en paralelo los bits de cada byte van juntos y al mismo tiempo. El periférico en sí se conecta con el chip a través de un número de un byte (o incluso de una palabra) de líneas.

Una de las ventajas del empleo de estos chips de interface está en que el programa cobra así un alto grado de flexibilidad en el establecimiento de detalles relativos a la configuración del hardware. No obstante, esto significa que el número de bits en la palabra de control, especialmente, puede ser bastante grande. Por ejemplo, una asignación de bit de control ACIA es la siguiente:

Bits 0 y 1	frecuencia de división del reloj e inicialización (velocidad bit serial)
Bits 2 al 4	formato de carácter en serie (del tipo paridad o no, número de bits de stop)
Bits 5 y 6	bits de control del transmisor (como la activación de interrupciones)
Bit 7	bit de control del receptor se pone a uno para activar interrupciones

El orden en que se establecen estos bits debe primeramente asegurar que tenga lugar una comunicación aceptable entre el ordenador y el periférico. Si esto falla se pueden obtener datos sin sentido en el dispositivo receptor.

Se precisan dos instrucciones para una E/S en serie concreta. Una de ellas configura el ACIA para el periférico serial. Por ejemplo:

```
MOVE.B   #$3,ACIACON   inicializa el hardware
MOVE.B   #$15,ACIACON  configura el hardware
```

En este caso, ACIACON ha de estar previamente definido.

Tendremos igualmente bits de estado que se leerán para una correcta transferencia de datos. Por ejemplo:

Bit 0	registro receptor datos preparado (un nuevo carácter está disponible)
Bit 1	registro transmisión vacío (se puede enviar un nuevo carácter)
Bits 2 y 3	señales control modem (CTS y DCD)
Bits 4 a 6	indicaciones de error sobre los datos recibidos (p.e., error de paridad)
Bit 7	petición de interrupción

Las interrupciones serán estudiadas en el próximo capítulo, pero los modems caen fuera del ámbito de esta serie.

Dado que sólo leemos un registro de estado y escribimos en un registro de control, podríamos usar sólo una dirección para lograr el mapeo de la memoria. Esto es igualmente aplicable al registro de datos, ya que sólo leemos de un registro receptor de datos y escribimos en un registro transmisor. Como ejemplo supongamos que queremos entrar un carácter,

```
INCH BTST    #0,ACIASTAT ¿hay carácter
                         disponible?
     BEQ     INCH        si no es así, seguir
                         probando
     MOVE.B  ACIADATA,D0 coloca el carácter
                         como parámetro
     RTS                 retorno subrutina
```

En este caso, esperamos hasta que el bit 0 se active en el registro de control antes de leer un registro de datos. ACIASTAT es igual que ACIACON en este ejemplo.

De modo similar podemos escribir una subrutina para salida de caracteres:

```
OUTCH BTST    #1ACIASTAT espera hasta
                         que el reg. de
                         datos esté
                         vacío
      BEQ     OUTCH      si no es así,
                         bifurcar y
                         volver a
                         comparar
      MOVE.B  D0,ACIADATA salida carácter

      RTS
```

Si es necesario repetir cualquier carácter recibido llamaríamos simplemente a una subrutina después de la otra:

```
ECHO  JSR     INCH       lee un carácter
      JSR     OUTCH      salida carácter
      BRA     ECHO       iteración
```

Es preciso subrayar aquí que la ejecución de una E/S de esta manera es poco eficaz dado que el ordenador trabaja a la velocidad de la transmisión de los caracteres. Aun así no es útil para ilustrar el mecanismo básico de una entrada/salida programada. En el próximo capítulo veremos cómo contribuyen las interrupciones a solventar este problema y cómo se ejecutan las transferencias de datos en E/S en paralelo.

Resumen de instrucciones del 68000

Antes de pasar a examinar las interrupciones en el 68000, vamos a proporcionar un resumen de todas las instrucciones del 68000 que hemos estudiado. La primera lista es el grupo que sirve para copiar datos y gira en torno a la instrucción MOVE:

Instrucción	Operación
MOVE	Mueve la fuente al destino
MOVEM	Mueve los registros de dirección y datos a y de la memoria (útil en entradas y salidas de subrutinas)
MOVEA	Mueve el contenido de la dirección efectiva al registro de direcciones
LEA	Mueve la dirección fuente al registro de direcciones
MOVEQ	Mueve con rapidez pequeñas constantes al registro de datos
PEA	Coloca la dirección efectiva en la pila
SWAP	Intercambia palabras superiores e inferiores en un registro de datos
EXG	Intercambia palabras largas entre determinados registros
LINK/UNLK	Empleado para establecer parámetros para llamadas a subrutinas (especializadas)

Este conjunto de instrucciones proporciona algunas facilidades bastante amplias, pero hay que estar atento en el empleo de instrucciones tan exóticas como LINK/UNLK. Veamos ahora las instrucciones encargadas de la aritmética con números enteros:

Instrucción	Operación
ADD	Suma fuente a destino
ADDA	Suma registro direcciones fuente a destino
ADDI	Suma datos inmediatos a destino de datos alterable
ADDQ	Suma rápida inmediata para constantes pequeñas (del 1 al 8)
ADDX	Suma con arrastre
SUB(A/I/Q/X)	Resta fuente de destino en cualquiera de las variantes de direccionamiento admitidas por ADD
CMP(A/I)	Compara fuente con destino y establece códigos de condición
CMPM	Compara posiciones de memoria y punteros de posincremento

NEG(X)	Hace negativo el registro de datos de operando
EXT	Amplía el signo en el registro de datos de operando
MULU	Multiplica datos sin signo en el registro de datos por dirección efectiva y pone el resultado en el registro de datos
MULS	Como MULU, para datos con signo
DIVU(S)	Divide el registro destino de datos por el operando fuente. Cociente y resto quedan en el reg. destino
TST	Activa los códigos de condición según sea el operando
TAS	Comprueba y activa el bit más significativo del operando
CLR	Limpia la dirección efectiva

Un buen racimo de instrucciones, ¡pero es necesario tener cuidado al seleccionar la instrucción adecuada para los modos de direccionamiento del operando! Sigue ahora una lista de instrucciones relativas a la aritmética en BCD:

Instrucción	Operación
ABCD	Suma operandos BCD colocando el resultado en el destino
SBCD	Resta los operandos
NBCD	Pone negativo el operando BCD

Es de notar que no existe ninguna instrucción BCD para multiplicar. Sin embargo, las instrucciones lógicas que ahora siguen colman estas necesidades:

Instrucción	Operación
AND	Opera con AND el fuente y el destino, siendo uno de ellos al menos un registro de datos
ANDI	AND lógico con datos inmediatos como fuente
OR	Opera con OR de modo semejante a la AND anterior
ORI	OR lógico con datos inmediatos
EOR	OR exclusivo
EORI	OR exclusivo con datos inmediatos
NOT	Inversión lógica del operando

Es el turno de las instrucciones con comprobación y manipulación de bits:

GRUPO MANIPULADOR DE BITS

Instrucción	Operación
BTST	Comprueba el bit especificado en el operando fuente
BSET	Comprueba y activa el operando destino
BCLR	Comprueba y limpia
BCHG	Comprueba y cambia el operando

GRUPO PARA DESPLAZAMIENTO Y ROTACION

Instrucción	Operación
ASI	Desplazamiento aritmético a izq..
ASR	Desplazamiento aritmético a der.
LSL	Desplazamiento lógico a izq.
LSR	Desplazamiento lógico a der.
ROL	Rotación a izq. del operando, activando adecuadamente el arrastre
ROR	Rotación a der. del operando, activando el bit de arrastre

Por último revisaremos las instrucciones de subrutinas y control de programas:

INSTRUCCIONES DE CONTROL DE PROGRAMAS

Instrucción	Operación
BRA	Bifurca siempre (forma eficaz de bifurcación incondicional)
JMP	Salta siempre (útil cuando se desea hacer algún cálculo en la dirección del salto)
Bcc	Bifurcación condicionada según el código de condición 'cc' que se comprueba. Es decir:

Para operandos con signo:

GT	mayor que
LT	menor que
GE	mayor o igual que
LE	menor o igual que
VS	desbordamiento activado
VC	desbordamiento limpio

Para operandos sin signo:

EQ	igual a
NE	no igual a
MI	menos
PI	más
HI	superior a
LS	inferior o igual que
CS	arrastre activado
CC	arrastre limpio

DBcc	Contador de bucle en decremento y bifurcación según condición de 'cc', como antes. ¡Recordar que la instrucción tiene un sentido diferente!

INSTRUCCIONES DE CONTROL DE SUBRUTINAS

Instrucción	Operación
JSR	Salto a subrutina
BSR	Forma eficiente de JSR
RTS	Retorno de subrutina

Este resumen no incluye todas las instrucciones del 68000. Se ha omitido deliberadamente un grupo especial relativo al control del sistema que emplea instrucciones privilegiadas. Estas instrucciones son empleadas en general por los ingenieros de sistemas operativos, por lo que están fuera de la finalidad de este estudio. Las instrucciones se refieren a operaciones sobre el registro de estado y los punteros de la pila, y las instrucciones de "interrupciones software" llamadas *trampas*. Naturalmente, pueden verse en detalle en el manual del usuario del 68000

Potencia inigualada

El miniordenador DEC VAX 11/780 es un excelente representante de las potentes máquinas de bases de datos y desarrollo de hoy en día

Dadas la potencia y velocidad de los micros actuales y el advenimiento de la conexión en red de software y hardware, sería lícito preguntarse por qué la gente continúa comprando miniordenadores muy caros de fabricantes como Prime y DEC. ¿Acaso estos mismos clientes no podrían reemplazar estas máquinas por ordenadores IBM PC y un par de discos para cada uno? La razón fundamental para adquirir un miniordenador, sin embargo, reside en su potencia bruta. La potencia de un ordenador se puede juzgar considerando su velocidad de cálculo, dimensiones de memoria y la gama de software y firmware disponible. En este sentido, ninguno de los micros actuales se aproxima a la potencia de los minis de tamaño mediano, e incluso en una red la potencia de cálculo está localizada: cada usuario está limitado a la potencia de su propia máquina.

Además, si un usuario está leyendo un listado, entonces su CPU estará ociosa, impidiendo que alguien de la red pueda acceder a ella. Frente a esta situación, la potencia completa de un miniordenador está disponible para todos los usuarios, si bien sólo parte del tiempo, y si un usuario no necesita su "parte de tiempo", entonces la misma será aprovechada por los otros usuarios del sistema.

Uno de los miniordenadores que ha tenido mayor aceptación es el VAX, de Digital Equipment Corporation (DEC). El VAX, de hecho, viene en varios modelos, que van desde el MicroVax, que al menos en tamaño es comparable a un micro de gestión estándar, hasta el superior de la gama, el 11/785. Nosotros hemos utilizado el 11/780, aunque la arquitectura es la misma para toda la gama, exceptuando el 11/782 y el 11/785.

Físicamente, el VAX es bastante diferente a los micros de gestión o personales estándares. En primer lugar, es más grande: el mueble básico mide 150 cm de altura, 100 cm de anchura y 75 cm de profundidad. Asimismo, mientras que la mayoría de los micros personales están incorporados en una única placa, utilizándose placas extras sólo con fines de ampliación, un 11/780 puede acomodar 16 placas que contengan solamente chips de memoria, si bien siempre contendrá otras varias placas que incluyan hardware para interface y la CPU.

Una CPU VAX no se halla en un único chip, ni siquiera en una única placa. Está diseñada para ser multitareas y para permitir una manipulación eficaz de la "memoria virtual". El VAX es una máquina de 32 bits, lo que permite direccionar más de cuatro gigabytes (cuatro billones de bytes) de memoria, aunque, incluso con el reducido precio de la RAM, toda esa memoria sería aún muy costosa. El concepto de memoria virtual se desarrolló para sacar partido de las capacidades de direccionamiento de la máquina. Sólo una fracción de la memoria disponible es verdaderamente RAM, y el hardware de la máquina se utiliza para comprobar si la dirección requerida se halla actualmente en la memoria; de no ser así, se la busca automáticamente en disco.

Este proceso se conoce como *paginación*. La memoria se manipula en páginas de 512 bytes, y cuando la memoria está llena, las páginas que han experimentado modificaciones mientras se hallan en la memoria se vuelven a escribir en disco a medida que se van requiriendo nuevas páginas. Todo esto, es completamente transparente para el usuario, dado que todos los programas "creen" estar hablándoles a cuatro gigabytes completos. En realidad, el hardware hace todo el trabajo de forma automática.

El sistema operativo estándar de DEC es VMS (aunque está disponible el Unix), que posee un potente lenguaje de comandos y un amplio juego de utilidades, incluyendo, entre otras, editores de línea y de pantalla completa, rutinas de clasificación y facilidades para correo electrónico. El único lenguaje proporcionado con la máquina es un macroensamblador, pero tanto DEC como otras fuen-

Marcus Wilson-Smith

Central de fuerza de proceso Por su alto precio, el miniordenador DEC VAX 11/780 es una opción viable solamente para empresas medianas y grandes. No obstante, por lo general se considera que los beneficios derivados del almacenamiento en línea masivo, la elevada velocidad de proceso de datos y la flexibilidad de esta facilidad de cálculo integrada bien valen los cuantiosos gastos de adquisición y mantenimiento

tes ofrecen compiladores para casi cualquier lenguaje por un costo adicional. El assembly VAX es una especie de revelación para los programadores habituados al código Z80 o 6502. Hay, por ejemplo, 16 registros, cada uno de los cuales es de 32 bits en tamaño. Cuatro de los mismos son especiales, utilizándose como punteros de pila y contador del programa, pero hay 12 para uso general. Y mientras los micros por lo general sólo pueden manipular en hardware aritmética de enteros, el VAX posee instrucciones para punto flotante (hasta números de punto flotante de 128 bits) y manipulación de caracteres. Asimismo, hay una única instrucción que corresponde a la estructura de bucle FOR...NEXT del BASIC. El juego total de 248 instrucciones incluye las operaciones básicas de mover y comparar, y un conjunto de instrucciones especiales para control de la CPU.

Para usar un VAX usted necesita un nombre de usuario y contraseña, asignados por el director del sistema, que le permite entrar en el sistema (*login*). Una vez en la máquina, se tiene acceso a las facilidades, cualesquiera que sean, que el director del sistema haya decidido que necesita el usuario. Cada nombre de usuario tiene privilegios y derechos asociados que limitan los archivos a los cuales se puede acceder, y también impide que usuarios no autorizados lleguen a ciertas partes del sistema operativo. Además, es posible proteger los propios archivos de modo que sólo ciertas personas puedan escribir en ellos, leerlos, suprimirlos o modificarlos.

El lenguaje de comandos se denomina DCL (Digital Command Language) e incluye muchas facilidades que normalmente sólo se encuentran en lenguajes de alto nivel. Éstas incluyen variables con nombre, sentencias GOTO e IF similares a sus equivalentes de BASIC, y un conjunto de funciones *lexicográficas* que se utilizan para devolver información del sistema tal como la hora y los nombres de los usuarios que están usando el sistema. La mayoría de estas funciones en realidad sólo se utilizan en *procedimientos de comandos*, que básicamente son listas de comandos DCL almacenados en un archivo y ejecutados como un programa. Para uso normal, no obstante, se emplea un único comando: por ejemplo, Show Users imprimiría una lista de usuarios. Una función lexicográfica se podría utilizar dentro de un procedimiento de comando, por ejemplo, para enviar un mensaje a cada usuario.

La mayoría de los comandos poseen varios cualificadores que se utilizan para modificar la acción. A modo de ejemplo, DELETE*.* suprimirá todos los archivos de un directorio (sólo aquellos que no estén protegidos) y DELETE/CONFIRM*.* llevará a cabo la misma acción, pero al usuario se le avisará en cada archivo para su confirmación. La lista de comandos completa es exhaustiva, e incluye todos los comandos usuales para tratamiento de archivos, así como algunos para modificar el entorno de trabajo y para permitir comunicarse con otros usuarios.

La comunicación con la máquina se efectúa a través de dos interfaces especiales. El Unibus se utiliza para terminales, discos lentos, unidades de cinta, impresoras y diversos equipos para fines especiales, como trazadores y dispositivos de control. La otra interface es Massbus, que es mucho más veloz y se utiliza para unidades de disco de acceso rápido.

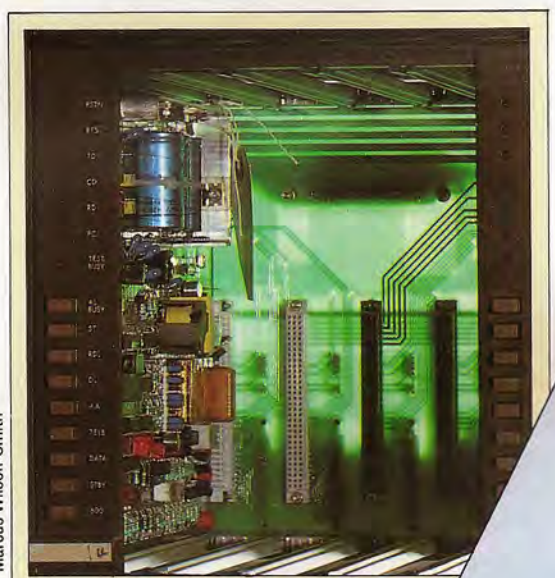
La configuración más pequeña de un 11/780 posee dos megabytes de memoria y ocho líneas para terminales. Adquiera al menos dos unidades de disco, cada una de las cuales debe ser capaz de almacenar al menos dos megabytes en un pequeño sistema operable. Se pueden instalar hasta ocho unidades de disco, y ampliar la memoria a ocho megabytes. Es posible conectar hasta 64 terminales con sólo instalar placas de interface adicionales.

Como entorno de desarrollo el VMS es excelente: el lenguaje de comandos es lógico, y hay archivos de texto de ayuda en línea para todos los comandos y cualificadores de comandos. También se proporcionan bibliotecas de útiles rutinas para usar con sus propios programas. Ello significa que el desarrollo de tareas tales como manipulación de pantalla y rutinas matemáticas con fines especiales se elimina, lo que da por resultado una gran reducción del tiempo de programación. Los archivos en disco compartidos por un equipo de desarrollo eliminan los programas derivados de los sistemas de micros independientes cuando alguien cambia un archivo sin decir a los otros usuarios que cambien los suyos.

El miniordenador proporciona un entorno de programación muy amable, y la potencia de la máquina permite que los programas de aplicaciones complejas se ejecuten más rápidamente que en un micro; pero la potencia es cara. Asimismo, los miniordenadores generan elevados costos de mantenimiento, de electricidad (alrededor de 6 kW como mínimo) y precisan aire acondicionado. Sin embargo, a medida que los micros vayan adquiriendo mayor potencia, reemplazarán paulatinamente a los miniordenadores; no obstante, éstos proporcionan a los usuarios un rendimiento que sólo es superado por el de los ordenadores centrales.

La consola norteamericana
El director del sistema del VAX se comunica con la máquina a través de la consola del operador (derecha), utilizándola casi como si se tratara de una telemáquina de escribir. El ordenador también puede enviar respuestas y mensajes de estado a la consola, donde se imprimen. De este modo se produce una salida impresa completa de cualquier diálogo entre el director y la máquina

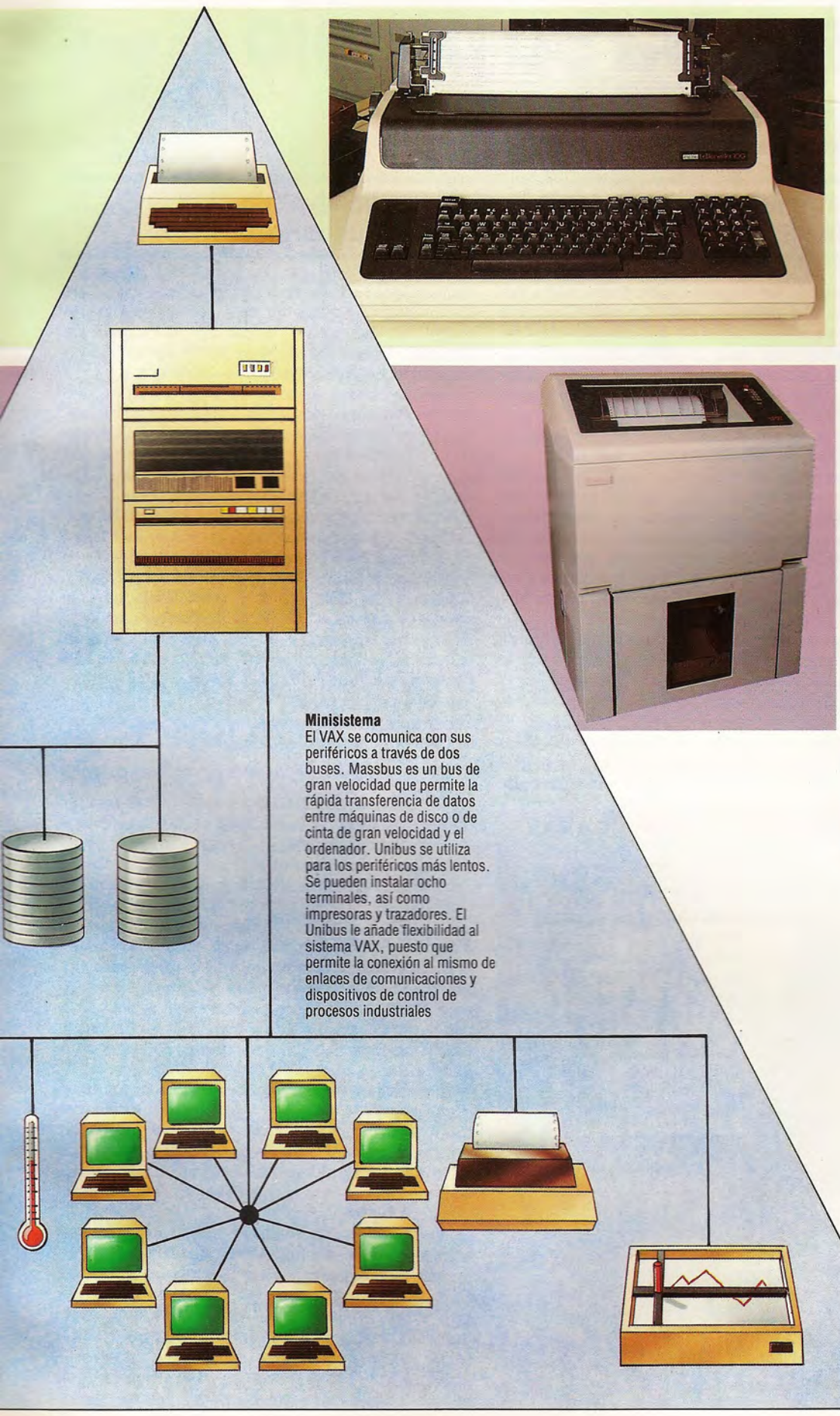
Impresora de línea
A través del Unibus se puede conectar al VAX una impresora de línea de gran velocidad (derecha)



Marcus Wilson-Smith

Modems M.U.D.
Multi-user dungeon, de la British Telecom, es un juego de aventuras interactivo con más de 1000 escenarios, para participar en él los usuarios deben llamar desde micros equipados con modems. El juego se ejecuta durante las horas no laborables en un miniordenador VAX, que por el día actúa como una base de datos europea. El VAX está equipado con numerosos modems (como vemos arriba) conectados a su red Unibus, permitiendo que hasta 100 personas jueguen simultáneamente, cada una a través de su propio enlace telefónico

Kevin Jones



Minisistema
El VAX se comunica con sus periféricos a través de dos buses. Massbus es un bus de gran velocidad que permite la rápida transferencia de datos entre máquinas de disco o de cinta de gran velocidad y el ordenador. Unibus se utiliza para los periféricos más lentos. Se pueden instalar ocho terminales, así como impresoras y trazadores. El Unibus le añade flexibilidad al sistema VAX, puesto que permite la conexión al mismo de enlaces de comunicaciones y dispositivos de control de procesos industriales

DEC VAX 11/780

DIMENSIONES
1 181 × 762 × 1 537 mm

CPU
VAX CPU

MEMORIA
RAM de entre 2 y 8 megabytes. La gestión de memoria virtual permite acceder a 4 gigabytes

PANTALLA
Visualización de textos de al menos 80 × 25, y resolución de hasta 1 024 × 1 024 pixels, con 16,5 millones de colores (según el tipo de terminal)

INTERFACES
Massbus para discos y cintas rápidos, Unibus para todos los otros dispositivos, incluyendo impresoras, trazadores y terminales, a través de un enlace RS232

LENGUAJES DISPONIBLES
Proporcionado con kit ensamblador, pero a un costo adicional hay disponibles compiladores para la mayoría de los lenguajes de alto nivel

DOCUMENTACION
El conjunto básico de documentación para el lenguaje de comandos se eleva a 30 volúmenes, ocupando 1,5 m de espacio de estantería. Éste incluye una concisa sección de referencia y guías de formación

VENTAJAS
El VAX proporciona un entorno de programación sumamente potente para desarrollo de software y administración de bases de datos. También es posible la conexión en red entre máquinas VAX

DESVENTAJAS
El sistema es bastante caro e implica elevados costos de operación, puesto que requiere una atmósfera libre de polvo y mantenimiento y apoyo a tiempo completo

Escriba su propio guión

Presentamos un ejemplo que ilustra significativamente la adaptabilidad del Unix a las exigencias del usuario

Textos sobre Unix
Es posible encontrar algunos interesantes libros que tratan del Unix. Entre éstos se incluyen tanto obras de "referencia" rápida para el programador experto como sencillas guías de introducción para el usuario ocasional. *The Unix environment* (El entorno Unix), de A. N. Walker, está escrito en un estilo desenfadado y es indicado para aquellos lectores que se inician en el Unix. Cubre las principales facilidades del sistema operativo, y también analiza la filosofía del Unix, sus facilidades típicas de hardware y la administración del sistema.

Real world Unix (El mundo real del Unix), de John D. Halamka, constituye un excelente texto práctico para quienes deseen aprender el Unix en el menor tiempo posible. El texto contiene numerosos ejemplos de las tareas más comunes y una guía de referencia rápida "el Unix en un minuto".

The Unix environment, de A. N. Walker, editado por John Wiley, Gran Bretaña
Real world Unix, de John D. Halamka, editado por SYBEX, Gran Bretaña

La mayoría de los sistemas operativos incorporan facilidades que permiten que los usuarios adapten el sistema para satisfacer sus necesidades específicas. Generalmente incluyen las opciones de ejecutar un lote de comandos a la vez (los archivos SUBMIT de CP/M y .BAT del MS-DOS, p. ej.) y ejecutar una determinada secuencia de comandos o programas cuando el usuario conecta por primera vez el sistema. El Unix, sin embargo, va más allá.

El Unix posee un "guión de caparazón", un lenguaje de programación por derecho propio, que posee facilidades para entrada/salida, selecciones e iteraciones. En combinación con los operadores de redirección y entubamiento del Unix, permite que los usuarios adiestrados construyan programas para la mayoría de las tareas sin necesidad de un lenguaje de programación "convencional".

El caparazón es el equivalente del procesador de comandos del CP/M y el usuario puede reescribirlo por completo. Existe un ligero inconveniente por el hecho de que distintos sistemas Unix puedan presentar al usuario distintos "rostros", aunque el sistema subyacente sea idéntico. Hay dos caparazones comunes para el sistema Unix Berkeley, en el cual se basa esta serie. El "caparazón c", como su nombre sugiere, se basa en el lenguaje c y es el que hemos utilizado para nuestros ejemplos. Sin embargo, si hubiéramos empleado el "caparazón Bourne", habría habido muy pocas diferencias en lo que hemos considerado hasta ahora.

Veamos ahora algunos de los comandos que afectan el entorno del usuario. Ya hemos visto brevemente cómo cada archivo o directorio puede acceder a privilegios para tres grupos de personas: los usuarios propiamente dichos (u), el grupo del usuario (g) y otros (o). (Esta idea de grupos es muy útil, especialmente en grandes organizaciones en las que es vital compartir los recursos y evaluar cada área de trabajo.) A cada una de estas tres categorías se le otorgan derechos de acceso para cada archivo y directorio del sistema, denotados mediante tres juegos de rwx en el listado de directorio completo dado por el comando ls–1. La r denota acceso para leer, la w para escribir y la x para ejecutar. Si una categoría no goza del acceso apropiado, la letra se sustituye por un guión.

Estos derechos de acceso se pueden cambiar utilizando la instrucción chmod, que toma la forma:

```
chmod categoría [+,–] acceso nombrearchivoodirectorio
```

donde + se utiliza para dar un derecho de acceso, y –, para suprimirlo. Así que, para otorgar al usuario y a su grupo derecho de escritura a un archivo llamado trabajoparcial, sería aceptable lo siguiente:

```
chmod ug+w trabajoparcial
```

Vea que se puede dar más de un cambio de permiso, separado por comas (pero no por espacios).

Otra opción permite cambiar los parámetros para el determinado tipo de terminal que se esté utilizando. La mayoría de los sistemas Unix vienen con tablas incorporadas para toda una variedad de tipos de terminal, uno de los cuales se puede seleccionar mediante el empleo de:

```
setenv TERM tipotermina!
```

Esto significa que no es necesario instalar paquetes de software para una pantalla y teclados determinados; en teoría, una instalación funcionará para todos ellos.

Para los tipos de terminal no estándares y para determinados fines, se pueden cambiar las teclas utilizadas para varias funciones con la instrucción stty:

```
stty everything
```

que dará una lista de los ajustes disponibles, y:

```
stty all
```

que dará una lista restringida de los más comunes.

Tarea a medida

```
Berkeley 4.2 Vax/Unix (infsc3)
Type (Ctrl-D) to disconnect

login: com–mcc
Password:
You are a Normal user (class 3)
Jobs : 15 Superiors : 4 Maximum : 21
Last Login: Wed Nov 13 15:05:36 on ttyn08

Hello nice to see you!          {éste es mi mensaje login de usuario}

%cat .login                     {consultar en archivo login}
setenv TERM tvi910+             {establecer mi tipo de terminal}
stty erase                      {Ctrl-H me permite usar el retroceso en mi PC en
                                 lugar de la tecla 'del'}
echo Hello nice to see you!     {éste es mi mensaje de sign-on}

%cat .cshrc
setenv EDITOR /usr/local/emacs
setenv NAME 'Mike Curtis'
set paht=(./usr/local/m2/usr/local/usr/ucb/usr/bin/bin/usr/games)
{el camino establece los directorios que se buscarán (por el orden dado)
para comandos, de modo que, si fuera necesario, se podría acceder directamente
a programas de otros directorios. El primer '.' es el directorio del propio
usuario}

set history=25                  {el caparazón recordará los últimos
                                 25 comandos}
set ignnoreeof                  {detiene desconexión accid. con Ctrl-D}
alias h history                 {adaptar comandos Unix a medida}
alias ty more
alias dir ls                    {usar dir en lugar de ls para listar
                                 el directorio}

%dir                            {demostrar instrucción dir}
lsfile mike rec.c receive rx.p transmit

%cat > ! temp                   {cat sin un archivo de entrada tomará la entrada
                                 desde el teclado hasta un Ctrl-D}

echo this is a list of mu files
/bin/ls                         {el n. completo del comando ls}
%mv temp ls                     {elaborar el propio comando ls a la medida}

%chmod u+x ls                   {usar chmod para hacerlo ejecutable. De lo
                                 contrario, ejecutado mediante sh < ls}
%ls                             {probar versión hecha a medida}
this is a list of my files
ls lsfile mike rec.c receive rx.p transmit

%history                        {repasar comandos impartidos}
    1 dir
    2 cat < ! temp
    3 mv temp ls
    4 chmod u+x ls
    5 ls
    6 history

% !5                            {repetir comando 5}
ls                              {reproduce primero el comando repetido}
this is a list of my files
ls lsfile mike rec.c rewceive rx.p transmit

%stty everything                {ver todos los ajustes de terminal}
new tty, spped 1200 baud
even odd –raw –nl echo –lcase –tandem –tabs –cbreak ff1
–crtbs –crterase –crtkill –ctlecho –prterase –tostop
–tilde –flusho –mdmbuf –litout – nohang
–pendin –decctlq –noflsh
erase kill werase rprnt flush lnext susp intr quit stop eof
^H  ^U  ^W   ^R   ^O  ^V  ^Z/^Y  ^C  ^   ^S/^Q  ^D

%stty all                       {algunos ajustes de teclas comunes}
new tty, speed 1200 baud; –tabs ff1

erase kill werase rprnt flush lnext susp intr quit stop eof
^H  ^U  ^W   ^R   ^O  ^V  ^Z/^Y ^C  ^   ^S/^Q  ^D

%logout
```

Cada función posee un nombre y se puede cambiar:

```
stty nombrefunción nuevatecla
```

Otra facilidad útil impide el uso sin autorización del terminal mediante el empleo de lock. Este comando solicitará una contraseña (distinta de la contraseña de login) y luego pedirá que se la repita. Entonces el terminal estará bloqueado y el Unix ignorará todas las pulsaciones de tecla hasta que se entre la contraseña una tercera vez.

Las dos facilidades siguientes constituyen una peculiaridad del caparazón c, pero son a la vez interesantes e instructivas. El comando history dará una lista de todos los comandos impartidos previamente, hasta una cantidad predefinida. Además, cualquiera de los comandos se puede repetir utilizando el operador !, por lo general seguido por el número de la lista del comando a repetir. De modo que:

```
!4
```

haría que se repitiera el cuarto comando de la lista.

El comando alias se puede utilizar para asignar otro nombre ya sea a cualquier comando o a cualquier serie de caracteres. Cada vez que se utilice el nuevo nombre, el Unix sustituirá la serie en su sitio. Para los usuarios de CP/M o MS-DOS que quieran utilizar dir para obtener un listado de directorio en lugar del comando ls del Unix:

```
alias dir ls
```

aceptará dir como ls, que puede entonces ir seguido por cualquiera de las opciones usuales. Si la serie contiene un punto y coma, que el Unix utiliza para separar comandos en una línea, se puede encerrar entre comillas:

```
alias dir 'fechas;ls -a'
```

El uso juicioso de alias puede aliviar muchas de las características hostiles del Unix.

En la mayoría de los directorios de los usuarios existen numerosos archivos especiales del sistema que se distinguen por tener nombres que comienzan con un punto. Éstos se protegen contra la supresión accidental mediante caracteres de máscara que no posean un punto delante. Dos de estos archivos son de especial interés, dado que preparan el entorno para un usuario: (.login y .cshrc [en el caparazón Bourne, sus funciones las cubre un único archivo, .profile]) y cada uno contiene un número de comandos de caparazón. El archivo .login se ejecuta cada vez que el usuario entra en el sistema, y se remite a .cshrc cada vez que se activa el programa de caparazón (estos archivos por lo general contienen fundamentalmente comandos alias, setenv y set). Las posibles opciones para setenv y set dependen de la instalación individual y son demasiado numerosas como para detallarlas aquí. Pero basta decir que se puede establecer virtualmente cualquier aspecto del sistema y los dispositivos periféricos según las propias especificaciones del usuario.

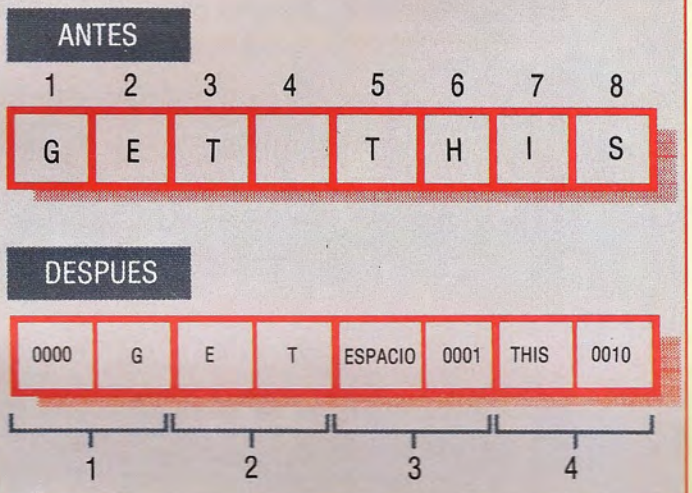
Los toques finales

En este penúltimo capítulo de nuestra serie presentamos la rutina de compresión en assembly 6502 y un programa Cargador en BASIC para la versión Z80 que ofrecimos en el capítulo anterior. La segunda mitad del programa y los programas Cargador y Activador en BASIC los incluiremos en el capítulo final

Rutina de compresión 6502

```
;++++ COMPRESION DE TEXTOS 6502 ++++
;
ZPTR1=$8B                ;PUNTERO ENTRADA PAGINA 0
ZPTR2=$8D                ;PUNTERO SALIDA PAGINA 0
ZPTR3=$FB                ;PUNTERO TABLA UTILIDADES
ZPTR4=$FE                ;PUNTERO TABLA UTILIDADES
*=$C000
OUTPUT   *=*+2           ;COMIENZO DE SERIE O/P
INPUT    *=*+2           ;COMIENZO DE SERIE I/P
STATUS   *=*+1           ;BYTE DE ESTADO
MASK     *=*+1           ;MASCARA NIBBLE
LEN      *=*+1           ;ENTRADA ALMACENAMIENTO LONG.
                         SERIE
OUTOFF   *=*+1           ;SALIDA ALMACENAMIENTO DESPL.
TOKCNT   *=*+1           ;CONTADOR DE DISTINTIVOS
TABCNT   *=*+1           ;CONTADOR TABLA
TEMP     *=*+1           ;GUARDAR COMIENZO DE SERIE O/P
;
START    LDA  INPUT
         STA  ZPTR1      ;ESTABLECER PUNTERO I/P
         LDA  INPUT+1
         STA  ZPTR1+1
         LDA  OUTPUT
         STA  ZPTR2      ;ESTABLECER PUNTERO O/P
         LDA  OUTPUT+1
         STA  ZTR2+1
         LDA  #1
         STA  OUTOFF     ;INIC. DESPL. O/P
         LDY  #0         ;INIC. DESPL. I/P
         LDA  (ZPTR1), Y
         STA  LEN        ;ALMACENAR LONGITUD SERIE I/P
         LDA  #255
         STA  MASK       ;ESTABLECER MASCARA NIBBLE
NCHAR    INY
         LDA  (ZPTR1), Y ;TOMAR SIGUIENTE CAR I/P
         JSR  CHECK      ;VALIDO?
         BNE  BADCHR
         JSR  TOKEN      ;DISTINTIVO?
         BEQ  GOTTOK
         JSR  FOURBT     ;CODIGO CUATRO BITS?
         BEQ  GOTFOR
         JSR  EIGHTB     ;CODIGO OCHO BITS?
         PHA             ;GUARDAR VALOR
         LDA  #0
         JSR  WRTNIB     ;ENVIAR CODIGO
         PLA
GOTFOR   JSR  WRTNIB     ;ENVIAR CODIGO 8 BITS
;
REJOIN   CPY  LEN        ;MAS CARACTERES I/P?
         BCC  NCHAR      ;IR POR ELLOS
         LDA  #0
         STA  STATUS     ;ESTADO 0=OK
         LDA  #2
         JSR  WRTNIB     ;FIN DE MARCA TEXTO
         LDA  MASK
         BEQ  NODEC
         DEC  OUTOFF
NODEC    LDY  #0
         LDA  OUTOFF
         STA  (ZPTR2),Y  ;ESCRIBIR LONGITUD O/P
         RTS
;
BADCHR   LDA  #255
         STA  STATUS     ;ENVIAR ESTADO MALO
         RTS

GOTTOK   PHA
         LDA  #1
         JSR  WRTNIB     ;ENVIAR CODIGO 0001
         PLA
         JSR  WRTNIB     ;Y CODIGO DISTINTIVO
         JMP  REJOIN
;
;++++  SUBRUTINAS+++++
TOKEN    PHA             ;GUARDAR CAR. ACTUAL
         TYA
         PHA             ;Y DESPL. I/P
         STY  TEMP
         LDA  ZPTR1
         CLC
         ADC  TEMP       ;CALC ZPTR3 PARA
         STA  ZPTR3
         LDA  ZPTR1+1    ;APUNTAR AL ACTUAL
         ADC  #0         ;CAR. I/P
         STA  ZPTR3+1
         LDA  #<TOKTAB
         STA  ZPTR4      ;ESTABLECER ZPTR4
         LDA  #>TOKTAB   ;PARA APUNTAR A
         STA  ZPTR4+1    ;TABLA DISTINTIVOS
         LDA  #0
         STA  TOKCNT     ;INIC. CONTADOR DISTINTIVOS
TOK1     LDY  #0
         LDA  (ZPTR4),Y  ;TOMAR LONGITUD SIGUIENTE
                         DISTINTIVO
         PHA             ;ALMACENARLA
         TAX
         BEQ  NOTFND     ;FIN DE LA TABLA
         LDA  ZPTR4
         CLC
         ADC  #1         ;INC ZPTR4
         STA  ZPTR4      ;ZPTR3 & ZPTR4
         LDA  ZPTR4+1    ;AMBOS AHORA EN
         ADC  #0         ;COMIENZOS TEXTO
         STA  ZPTR4+1
TOK2     LDA  (ZPTR3),Y
         CMP  (ZPTR4),Y  ;COMPARAR
         BNE  NEXTOK     ;NINGUNA PAREJA TOMAR SIGUIENTE
                         DISTINTIVO
         INY
         DEX             ;CONTHASTA
         BNE  TOK2       ;FIN PALABRA DISTINTIVO
;++ DISTINTIVO EMPAREJADO ++
         PLA             ;LIMPIAR PILA
         PLA
         DEY
         STY  TEMP
         CLC
         ADC  TEMP       ;CALC NUEVO DESPL. I/P
         TAY             ;PONERLO EN Y
         PLA
         LDA  TOKCNT
         LDX  #0         ;ESTABLECER Z=1
         RTS
;
NEXTOK   PLA             ;TOMAR LONGITUD DISTINTIVO
         STA  TEMP
         LDA  ZPTR4
         CLC
         ADC  TEMP
         STA  ZPTR4      ;AJUSTAR ZPTR4
         LDA  ZPTR4+1    ;PARA QUE APUNTE AL
         ADC  #0         ;SIGUIENTE DISTINTIVO
         STA  ZPTR4+1
         INC  TOKCNT
```

Compresión veloz
El uso de distintivos, más la representación de caracteres comunes en un código de cuatro bits, permite que nuestro algoritmo de compresión de textos consiga velocidades de compresión que alcanzan el 50 %. Aquí, una serie de ocho letras (incluyendo el espacio) se reduce a cuatro bytes. Observe los códigos de control de 4 bits que indican una palabra distintivada, un segundo juego de caracteres y el fin del mensaje




```
        JMP   TOK1
NOTFND  PLA                   ;DESCARTAR
        PLA
        TAY                   ;RESTAURAR PUNTERO I/P
        PLA                   ;RESTAURAR CAR. I/P
        LDX   #255            :Z=0=FALLO
        RTS
;
FOURBT  PHA                   ;GUARDAR CAR. I/P
        STY   TEMP            ;GUARDAR DESPL. I/P
        LDA   #<TAB4BT
        STA   ZPTR3
        LDA   #>TAB4BT
        STA   ZPTR3+1
        PLA                   ;VOLVER A TOMAR CAR I/P
        JMP   TBSCAN
EIGHTB  PHA                   ;GUARDAR CAR I/P
        STY   TEMP            ;GUARDAR DESPL. I/P
        LDA   #<TAB8BT
        STA   ZPTR3
        LDA   #>TAB8BT
        STA   ZPTR3+1
        PLA                   ;VOLVER A TOMAR CAR I/P
;
TBSCAN  LDY   #0
TBSCN2  CMP   (ZPTR3),Y
        BEQ   TBSCN1          ;HALLADO!
        INY
        CPY   #16             ;BUSCAR FIN TAB
        BNE   TBSCN2
        LDY   TEMP            ;RESTAURAR DESPL. I/P
        LDX   #255            ;Z=0=FALLO
        RTS
TBSCN1  TYA                   ;PONER VALOR TAB. EN A
        LDY   TEMP
        LDX   #0              ;Z=1=EXITO
        RTS
;
CHECK   CMP   #' '
        BEQ   RETURN
        CMP   #','
        BEQ   RETURN
        CMP   #'.'
        BEQ   RETURN
        CMP   #'A'
        BCC   NOGOOD
        CMP   #$5B
        BCS   NOGOOD
RETURN  LDX   #0
        RTS
NOGOOD  LDX   #255
        RTS
;
WRTNIB  PHA                   ;GUARDAR NIBBLE
        STY   TEMP            ;GUARDAR DESPL. I/P
        LDY   OUTOFF
        LDA   MASK
        BNE   LEFT            ;<> 0 SIGNIFICA LADO IZQUIERDO
        PLA
        ORA   (ZPTR2),Y       ;AÑADIR EL NUEVO AL ANTIGUO
        STA   (ZPTR2),Y       ;SUSTITUIRLO
        INC   OUTOFF
        LDY   TEMP            ;RESTAURAR DESPL. I/P
        LDA   #255
        STA   MASK            ;RESTABLECER MASCARA PARA
                              SIGUIENTE
        RTS
LEFT    PLA
        ASL   A
        ASL   A
        ASL   A
        ASL   A
        STA   (ZPTR2),Y
        LDY   TEMP
        LDA   #0
        STA   MASK
        RTS
;
;++++ LETRAS MAS COMUNES ++++
TAB4BT  .BYT  0
        .BYT  0
        .BYT  0
        .BYT  'F'
        .BYT  'L'
        .BYT  'D'
        .BYT  'H'
        .BYT  'S'
        .BYT  'I'
        .BYT  'R'
        .BYT  'N'
        .BYT  'O'
        .BYT  'A'
        .BYT  'T'
        .BYT  'E'
        .BYT  ' '
;++++ LETRAS MENOS COMUNES ++++
TAB8BT  .BYT  'C'
        .BYT  'M'
        .BYT  'U'
        .BYT  'G'
        .BYT  'Y'
        .BYT  'P'
        .BYT  'W'
        .BYT  'B'
        .BYT  'V'
        .BYT  'K'
        .BYT  'X'
        .BYT  'J'
        .BYT  'Q'
        .BYT  'Z'
        .BYT  ','
        .BYT  '.'
;++++ TABLA DE DISTINTIVOS ++++
TOKTAB  .BYT 3, 'THE'

        .BYT 4, 'THIS'

        .BYT 4, 'THAT'

        .BYT 2, 'IF'
```

```
.BYT 3, 'YOU'
.BYT 2, 'ME'
.BYT 3, 'WAS'
.BYT 2, 'HE'
.BYT 3, 'SHE'
.BYT 4, 'THEY'
.BYT 2, 'OF'
.BYT 2, 'IT'
.BYT 2, 'IS'
.BYT 3, 'FOR'
.BYT 2, 'ON'
.BYT 2, 'TO'
.BYT 0
.END
```

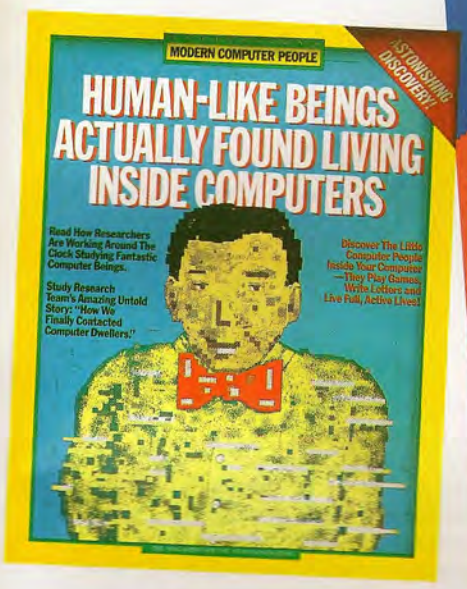
Cargador en BASIC Z80

```
10 MEMORY 29999
20 FOR n=30000 TO 30640
30 READ p:POKE n,p:c=(c+p)
40 NEXT n
50 IF c< > 64282 THEN PRINT" Error en suma de
   control..."
60 STOP
100 DATA 24,7,70,160,102,158,0,255,0,42,50,
    117,126,50,56,177
110 DATA 35,237,91,52,117,213,19,237,83,52,
    117,62,255,50,55,117
120 DATA 126,205,253,117,32,57,205,162,117,
    48,59,205,223,117,40
130 DATA 10,205,232,117,245,62,0,205,22,
    118,241,205,22,118,35
140 DATA 58,56,117,61,50,56,117,167,32,216,
    50,54,117,62,2,205
150 DATA 22,118,235,209,58,55,117,167,40,1,
    43,167,237,82,125,18
160 DATA 201,209,62,255,50,54,117,201,245,
    62,1,205,22,118,241
170 DATA 205,22,118,24,203,229,235,33,2,
    119,58,56,117,79,6,15
180 DATA 126,167,40,42,213,229,35,197,71,
    26,190,32,21,35,19,5
190 DATA 32,13,121,50,56,117,193,225,225,
    225,235,43,175,120,201
200 DATA 13,32,231,193,5,225,126,95,22,0,
    35,25,209,24,210,225
210 DATA 126,183,201,229,33,226,118,205
    241,117,225,201,229,33
220 DATA 242,118,205,241,117,225,201,6,15,
    190,40,5,35,16,250,183
230 DATA 201,120,201,254,32,200,254,44,200,
    254,46,200,254,65,56
240 DATA 8,254,91,48,4,79,175,121,201,62,
    255,167,201,79,58,55
250 DATA 117,237,91,52,117,167,32,14,26,
    177,18,19,237,83,52,117
260 DATA 62,255,50,55,117,201,121,203,39,
    203,39,203,39,203,39
270 DATA 18,175,50,55,117,201,42,50,117,35,
    34,50,117,237,91,52
280 DATA 117,213,62,255,50,55,117,205,185,
    118,254,2,202,129,118
290 DATA 210,153,118,167,202,139,118,205,
    185,118,205,110,118,71
300 DATA 126,35,205,174,118,16,249,24,255,
    32,2,119,71,62,15,144
310 DATA 71,4,126,35,5,200,95,22,0,25,24,
    246,42,52,117,209,167
320 DATA 237,82,125,18,201,205,185,118,33,
    242,118,205,164,118,205
330 DATA 174,118,24,182,33,226,118,205,164,
    118,205,174,118,24,171
340 DATA 95,62,15,147,95,22,0,25,126,201,
    237,91,52,117,19,18
350 DATA 237,83,52,117,201,58,55,117,42,50,
    117,167,126,32,14,230
360 DATA 15,35,34,50,117,79,62,255,50,55,
    117,121,201,203,47,203
370 DATA 47,203,47,203,47,230,15,79,175,50,55,
    117,121,201,32,69
380 DATA 84,65,79,78,82,73,83,72,68,76,70,
    0,0,0,67,77,85,71
390 DATA 89,80,87,66,86,75,88,74,81,90,44,
    46,3,84,72,69,4,84
400 DATA 72,73,83,4,84,72,65,84,2,73,70,3,
    89,79,85,2,77,69
410 DATA 3,87,65,83,2,72,69,3,83,72,69,4,
    84,72,69,89,2,79,70
420 DATA 2,73,84,2,73,83,3,70,79,82,2,79,
    78,2,84,79,0,205,167
430 DATA 126,32,14,230,15,35,34,50,117,79,
    62,255,50,55,117,121
440 DATA 201,203,47,203,47,203,47,203,47,
    230,15,79,175,50,55,117
450 DATA 121,201,32,69,84,65,79,78,82,73,
    83,72,68,76,70,0,0
460 DATA 0,67,77,85,71,89,80,87,66,86,75,
    88,74,81,90,44,46,3
470 DATA 84,72,69,4,84,72,73,83,4,84,72,65,
    84,2,73,70,3,89
480 DATA 79,85,2,77,69,3,87,65,83,2,72,69,
    3,83,72,69,4,84,72
490 DATA 69,89,2,79,70,2,73,0,0
```

Complementos al BASIC

Los usuarios del Spectrum necesitarán introducir una ligera modificación en el programa Cargador. La rutina de compresión también debe trabajar con otras máquinas Z80, incluyendo el Memotech, el Amstrad y el Einstein. Quizá los usuarios de algunas de estas máquinas también deban alterar la línea 10 del programa Cargador (consulte su manual para obtener detalles):

```
10 CLEAR 29999
```



Dimension Graphics

Personas queridas

He aquí un insólito programa, en el que el jugador debe tratar de hacer feliz a una diminuta persona que vive dentro del ordenador

A pesar del hecho de que el software de juegos por ordenador se ha desarrollado considerablemente desde los primeros días de los micros personales, la mayoría de los juegos aún tienen un objetivo predeterminado. Ya sean recreativos, de aventuras o estratégicos, el usuario tendrá como misión salir airoso de los desafíos ideados por el programador.

El programa se ha escrito de modo tal que la persona que hay en cada copia es, en cierto grado, exclusiva y única. Todas ellas tienen nombres diferentes, se visten de modo distinto y tienen prioridades exclusivas en cuanto a su forma de "vivir".

Obviamente se han dedicado enormes esfuerzos en proporcionarle a *Little computer people* historias en las cuales uno pueda sentirse comprometido. El software viene empaquetado con una gran carpeta en la que se detalla el "descubrimiento" de esta pequeña gente, costumbres, preferencias y hábitat.

Cuando se carga el programa por primera vez, la pantalla visualiza el interior de una casa, que cuenta con todas las habitaciones que cabría esperar en una casa normal de tres plantas. También hay un armario empotrado en el dormitorio y a menudo las personas desaparecen en el interior del mismo durante unos minutos, si bien no se sabe muy bien qué es lo que hacen realmente allí dentro. Unos minutos después de cargado el juego, su protegido entrará en la casa e inspeccionará sus dependencias. Si las aprueba, desaparecerá durante unos instantes antes de reaparecer con todas sus pertenencias, incluyendo su perrito.

Para mantener feliz a esta persona, usted ha de proporcionarle comida, agua y, sobre todo, muchísima diversión. El éxito que obtenga en la satisfacción de estas necesidades podrá evaluarlo por la expresión del rostro de la personilla, que puede ser desde desdichada hasta muy feliz. Por ejemplo, si ésta luce un aire desdichado y verde, es porque no se está alimentando muy bien. El jugador debe tener presente que también es necesario dar de comer al perro.

Activision insiste en que el idioma que hablan estas personas no se ha descifrado todavía, de modo que toda la comunicación con ellas se realiza a través del teclado. De este modo, es posible asegurarse de que su protegido sea feliz sugiriendo que tiene comida, puede encender fuego o tocar el piano. Es posible obtener un barómetro exacto del estado de ánimo de la persona sugiriendo que le escriba una carta a usted.

Se ha prestado mucha atención al detalle en la programación de los gráficos y el sonido, que es de muy buena calidad. La música producida cuando la personilla toca el piano es una de las mejores que han emanado nunca del Commodore 64. Lo que es más, se han sincronizado las manos de la misma con las notas y los acordes.

Little computer people: Para el Commodore 64
Editado por: Activision Inc, Box 7287 Mountain View, California, Estados Unidos
Autores: Richard Gold, David Crane y Sam Nelson
Formato: Cassette o disco
Palancas de mando: No son necesarias

Un día en la vida
Quienes desarrollaron *Little computer people* han dedicado todos sus esfuerzos a que las criaturas que habitan en el interior del ordenador parezcan lo más "reales" posible. La documentación con formato de revista que se entrega con el programa y que suele acompañar a todos los juegos es una de las más cuidadas que se hayan realizado. La casa en la cual se desarrolla la acción cuenta con todas las comodidades que comúnmente hacen más grata la vida. Aunque, todo hay que decirlo, en el transcurso del programa no suceden demasiadas cosas, el dedicarse a cuidar a esta "personita" tiene un atractivo emocional que muy pocos juegos pueden proporcionar

Escribir juegos

Estrella del software casero
El programador británico Jeft Minter es muy conocido entre los aficionados a los juegos recreativos. Sus programas han tenido un enorme éxito en un mercado que es sumamente competitivo, y todos ellos los ha desarrollado en su casa. La situación de programadores como Minter, sin embargo, a la larga puede ser inestable. Los programadores comerciales pueden beneficiarse del intercambio de ideas y la formación derivada de trabajar en una gran empresa, ventajas que les son negadas a los programadores por cuenta propia

A la conquista de usuarios
Virgin's Games Centre, en la Oxford Street londinense, es el ejemplo típico de una nueva generación de comercios alentados por el aumento en popularidad de los juegos de mesa y el software para ordenador. Sin embargo, la mayor parte de los programas se continúa vendiendo a través de las grandes cadenas minoristas, tales como W. H. Smiths y Boots. Convencer a una cadena minorista de que acepte existencias de su producto suele ser difícil, en especial en las temporadas en que hay mayor demanda, pero es preceptivo para alcanzar buenas cifras de ventas

La programación de juegos es una especialidad que suele comportar un margen de riesgo y gran competitividad

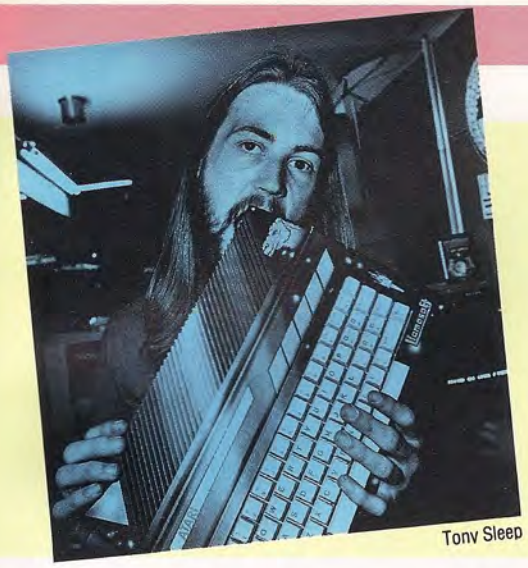
La programación por cuenta propia es un negocio arriesgado. En primer lugar, es probable que la retribución sea escasa, a menos que el juego sea un gran éxito. Los nombres más acreditados pueden recibir un anticipo y un porcentaje por los *royalties* correspondientes, pero es improbable que un programador desconocido reciba algo por adelantado y éstos, por lo general, son ínfimos. En segundo lugar, transcurrirá bastante tiempo antes de que se reciba algún pago. Preparar un juego para su lanzamiento puede llevar varios meses, de modo que cabe esperar que transcurra un año entre terminar el programa y recibir el primer pago. Y, por supuesto, no existe ninguna garantía de que le acepten el juego, aun teniendo un contrato. Todo contrato suele incluir una cláusula que afirma que el juego se debe completar a entera satisfacción de la casa de software, y la triste realidad es que algunos tipos de juegos pasan de moda con mucha rapidez.

La mayoría de las casas de software de juegos operan fundamentalmente con programadores por cuenta propia, pero existen oportunidades de empleo. Si usted posee talento e ideas, la mayor parte de las casas de software aprovecharán la posibilidad de proporcionarle un puesto de trabajo. Las ventajas que ofrece el trabajo por cuenta ajena sobre el trabajo por cuenta propia son, en teoría, seguridad, posibilidades de promoción, formación y la sensación de formar parte de un equipo. En la práctica, la seguridad, las perspectivas de promoción y la formación distan mucho de estar garantizado.

A la vista de la regularidad con que aparecen y desaparecen las casas de software, es discutible que exista un empleo seguro en la programación de juegos. Obtener estadísticas fiables es difícil, porque muchas "casas de software" no son más que un nombre comercial que encubre una actividad desarrollada en un garaje o en un cobertizo de jardín. En Gran Bretaña, por ejemplo, hay probablemente entre 300 y 350 casas de software de juegos en existencia en cualquier momento dado. Cada año se crean alrededor de 200 o más, y en el mismo período desaparece aproximadamente la misma cantidad. Este proceso de equilibrio crea una falsa impresión de estabilidad: aunque la cantidad total de empresas permanece moderadamente constante, ¡la proporción de fracasos probablemente supere el 50 % del volumen de la industria!

La formación y las perspectivas de promoción dependen en gran medida de la empresa de que se trate. Un puñado de las compañías más importantes proporcionan adiestramiento formal, pero la inmensa mayoría no lo hace. La casa de software típica organiza a los programadores en equipos, con un programador senior como jefe de equipo. Los programadores junior pueden consultar al jefe de equipo para recabar su ayuda y su consejo. Una o dos empresas les pagan a los programadores junior para acudir a clases nocturnas en facultades locales, pero normalmente sólo abonan la matrícula del curso, de modo que en realidad el alumno lo cursa fuera del horario de trabajo.

En las empresas más pequeñas, las perspectivas de promoción son escasas o inexistentes. El director gerente lleva la empresa y los programadores escriben el software. Quizá haya un gerente de ventas/marketing para comercializar los juegos, pero normalmente el personal queda reducido a eso. En las empresas más grandes, la escalera de promoción habitual es de programador junior a jefe de equipo, y tal vez de allí a la gerencia si la persona posee aptitudes e inclinación para ello. La promoción depende casi exclusivamente de la capacidad; si el candidato es eficiente, será promocionado a jefe de equipo en un plazo de alrededor de seis meses. Si no evidencia aptitudes para ocupar el cargo, probablemente no le servirá de nada, por ejemplo, tener



Tony Sleep

cinco años de antigüedad en la empresa. Las cualificaciones, e incluso la experiencia, cuentan muy poco. Lo que los patrones desean ver es una prueba de su capacidad (es decir, un juego terminado), ideas renovadoras y conciencia del mercado. La mayoría de las casas de software afirman que prefieren cualificaciones formales, pero todas ellas coinciden en que la falta de las mismas no consituiría un obstáculo para un programador con talento. Y muchas prefieren un joven de 17 años con ideas a un hombre de 40 con experiencia.

El escritor de juegos, por supuesto, habrá de programar en lenguaje máquina. Por lo general se exige Z80 o 6502, y el 68000 está adquiriendo también importancia. El BASIC, lamentablemente, no lo llevará a ninguna parte. Asimismo, necesitará un conocimiento profundo de uno de los micros personales que tengan mayor aceptación en la actualidad, por ejemplo, Spectrum, Commodore 64, MSX o la gama Amstrad CPC.

Programador de juegos: hechos concretos

Cualificaciones exigidas
Las cualificaciones formales en informática son de ayuda, pero no suelen ser esenciales.

Lenguajes requeridos
Lenguaje máquina. Normalmente Z80, 6502 o 68000. Asimismo, se espera que demuestre un profundo conocimiento de los micros personales más populares.

Escala salarial
Sumamente variable. El sueldo de un no titulado suele estar de acuerdo con su capacidad y el tamaño de la empresa. Los titulados pueden esperar una remuneración equiparable a las tasas normales de la industria.

Formación proporcionada
La formación regular es muy rara. La mayoría de las empresas proporcionan un adiestramiento informal, a través del propio trabajo realizado bajo la supervisión de un programador senior.

Perspectivas de promoción
Ninguna en las firmas pequeñas. En las grandes, cabe esperar progresar a programador senior y posiblemente a la gerencia, a tenor de los méritos.

Vendiendo su juego

Supongamos que usted acaba de escribir un juego destinado a hacerse más famoso que *Pacman, Defender* o incluso *Space invaders*. ¿Qué hacer para venderlo? Esta lista de comprobación está diseñada para encaminarlo por la senda correcta.

Protéjase
La mayoría de las casas de software son de una escrupulosa honestidad. No obstante, esto no es así en el caso de unas pocas. Usted puede proteger su juego tomando una sencillas precauciones:

- Incluir una nota de *copyright* en la primera línea del código. De ser posible, valerse de los trucos de su máquina para asegurarse de que no se la pueda eliminar; de lo contrario, empotrar más notas de *copyright* a lo largo del juego.
- Incluir líneas redundantes, es decir, líneas que parezcan formar parte del juego pero que no tengan ningún efecto. Si el juego aparece sin su consentimiento, puede usar estas líneas redundantes como ayuda para establecer su *copyright*.
- Depositar una copia del programa en su banco, solicitando un recibo con la fecha.
- Conservar todo lo que escribió. Las primeras versiones del programa, los diagramas de flujo, trozos de rutinas, notas garabateadas.
- Llevar el programa a la casa de software y obtener un recibo firmado y fechado. Si utiliza el correo, enviarlo con acuse de recibo.

Acabarlo con todos los detalles
Las casas de software cuidarán con todo gusto de la presentación del programa. Quizá añadan su propia pantalla de títulos, escriban el folleto de instrucciones, añadan un cargador más veloz, etc. No obstante, esperan recibir el juego completo en todos los sentidos.

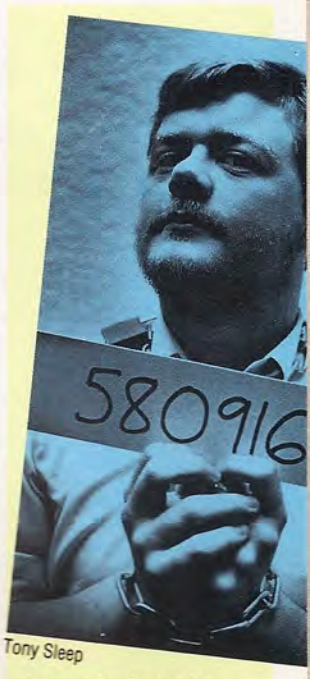
Preste atención a la presentación
Las casas de software reciben una gran cantidad de entregas no solicitadas. Todo cuanto pueda incluir para que la suya se destaque será de ayuda, como:

- Un resumen breve de 100 o 200 palabras.
- Instrucciones claras para cargarlo y jugar.
- Varias copias del juego por si se presentan dificultades de carga (disco antes que cinta).
- Un diagrama de flujo, si ayuda a esclarecer la estructura del juego.
- Detalles relativos a la máquina básica. Por ejemplo, 48 Kbytes de RAM, palanca de mando, unidad de disco, etc.

Marque todo con su nombre, dirección y número de teléfono en horas de trabajo.

No se comprometa de inmediato
Envíe su juego a tantas casas de software como sea posible, y no acepte la primera oferta en el instante en que se produzca. Si recibe una oferta, telefonee a algunas de las otras casas y hágales saber que le han hecho una oferta, pero sin ofrecer mayores detalles. Quizás ello los apure a evaluar su juego.

Nunca firme un contrato sin consejo profesional. En Gran Bretaña muchos abogados ofrecen consultas de tarifa fija, de alrededor de 20 minutos, por entre £5 y £10. Éste es un dinero bien invertido.



Tony Sleep

La experiencia cuenta
Dave Nicholls trabaja a jornada completa como programador para la Ram-Jam Corporation. El grueso de su trabajo consiste en convertir programas de un sistema a otro. Los antecedentes de Nicholls como programador de miniordenadores VAX y su experiencia comercial le facilitan la labor de pasar de un sistema a otro y de mantenerse al tanto de los nuevos desarrollos

Asignación de recursos

Finalmente analizaremos cómo interactúa el lenguaje con los sistemas operativos

Al C se lo utiliza principalmente, junto a otros fines, para escribir software de sistemas, sistemas operativos y utilidades, así como eficaz software de aplicaciones. Esto significa, entre otras cosas, que ha de existir una buena interface entre el lenguaje y el OS bajo el cual se está ejecutando, y que se han de poder programar funciones del mismo.

Una función particularmente útil cuando se escribe código que se ha de transferir entre sistemas diferentes en sizeof(objeto). Operador más que función, se le puede proporcionar tanto un nombre de variable como un nombre de tipo, y devuelve la cantidad de bytes utilizados para el almacenamiento de ese objeto.

Un ejemplo de su uso podría ser escribir código que funcione si int es de 16 o 32 bits. Con frecuencia se emplea junto con cuatro funciones (calloc(), malloc(), realloc() y free()), todas las cuales se ocupan de la asignación de memoria.

- **calloc ():**

```
char* calloc(número, tamaño)
int número,tamaño
```

La instrucción calloc() asigna, e inicializa en cero, una zona de memoria suficiente para retener un número de elementos, cada uno de ellos de la cantidad de bytes tamaño, devolviendo un puntero al primer elemento (o NULL si la memoria disponible es insuficiente). Por ejemplo, para reservar espacio para 50 enteros, podríamos usar:

```
p=calloc(50,tamañode(in));
```

- **malloc():**

```
char* malloc(bytes)
unsigned bytes;
```

Realiza una función similar, pero simplemente reserva la cantidad de bytes bytes sin inicializar. Con estas dos funciones, el valor devuelto es tan sólo un puntero a char, puesto que char siempre es un byte. Para utilizarlo realmente como un puntero a un objeto de un tipo diferente se lo debe *cast*.

- **realloc():**

```
char *realloc(p,bytes)
char *p;
int bytes;
```

El comando realloc() cambia el tamaño de la zona a la cual apunta p, a la cantidad de bytes bytes. Copiará el contenido del área antigua en el área nueva, pero puede o no reasignar una nueva sección de memoria, de modo que tras la misma los punteros de la zona ya no serán válidos.

- **free():**

```
free(p)
char *p;
```

Esta función libera un área de memoria que ha sido asignada por calloc, malloc o realloc para su reutilización. Podría ser un error desastroso utilizarla con cualquier otro valor de puntero.

Otra clase de funciones del C trata de la manipulación de errores. La mayor parte de los errores que pueden surgir en un programa en C se producen a raíz de la manipulación de entrada/salida o en las llamadas a la biblioteca. La biblioteca incluye una variable int estándar denominada errno y una lista de mensajes de error del sistema sys_errlist[], a la que se puede aludir directamente desde dentro del programa. Si se produce un error durante una llamada a la biblioteca, entonces se colocará un valor en errno para indicar esta circunstancia. Una llamada a:

```
perror(s)
char *s="mi mensaje de error";
```

visualizará en stderr tanto el mensaje dado como el de error del sistema correspondiente al valor actual de errno.

Cuando ocurren errores en entrada/salida en algunas ocasiones producirán un error del sistema normal, pero no siempre se puede confiar en ello. La función int ferror(puntero_archivo) comprobará si se ha producido un error al leer o al escribir en el archivo dado, devolviendo cero si se ha producido un error y no cero en caso contrario. La condición de error puede impedir el acceso al archivo, de modo que se proporciona la función clearerr (puntero_archivo) para restaurar la condición de error.

También se proporcionan varias funciones para la administración general de archivos en disco y directorios. Los detalles relativos a las mismas se deben incluir en su compilador, pero casi con toda seguridad incluirán access(), que comprueba la modalidad de acceso de un archivo o directorio mencionados; chmod(), que cambia la modalidad de acceso; y chdir() para cambiar el directorio de trabajo actual.

El programa de ejemplo es relativamente largo e implica casi todas las facilidades del C que hemos visto hasta ahora, incluyendo algunas (como llamadas recursivas a funciones) de las que no hemos hablado explícitamente, así como malloc, para obtener espacio de almacenamiento para mantener una lista enlazada. El programa tiene por objeto crear un índice para un archivo de texto grande; en otras palabras, una lista de todas las palabras utilizadas y los números de página en las que aparecen.

Las palabras se conservan en una lista enlazada que crece a medida que se va encontrando cada palabra. Cada palabra lleva asociado un puntero a otra lista enlazada de números de página para cada palabra.

El C es un lenguaje muy interesante de emplear y de amplia difusión. Es uno de los pocos lenguajes que ofrece la posibilidad de escribir un programa en un micro pequeño que también funcione en micros grandes, en minis e incluso en ordenadores centrales. Si usted no ha intentado aún utilizar el C, decididamente bien vale el esfuerzo: sus cualidades son muchas y su difusión es mayor cada vez.

Compilador de índice

```
/*En archivo index.h*/
#define NULL 0
# define TAMAÑOMAXIMOPALABRA 20
typedef char ENTRY[TAMAÑOMAXIMOPALABRA];
/*Cada elemento del índice estará compuesto
por la entrada real, un puntero a la lista de
números de página y un puntero al siguiente
elemento*/
struct número_página
{
  int pn;
  struct número_página*psiguiente;
}
typedef struct número_página página;
typedef página*penlace;
struct elemento_índice
{
  ENTRY entrada;
  penlace páginas;
  struct elemento_índice * salir
}
typedef struct elemento_índice elemento;
typedef elemento * enlace;
/* ahora podemos aludir a un artículo de la
lista como un "elemento" y un puntero a un
elemento se denomina un "enlace"*/
/* en archivo index.c*/
# include stdio.h
# include string.h
# include index.h
# define LPP=66;/* líneas por página */
main(argc,argv)
int argc;
char*argv[];
enlace head;
{
  FILE enarchivo;
  int lc=0,pc=1,enpalabra=0;
  ENTRY siguientepalabra;
  char*nw;
/* inicializar la lista con una primera entrada
ficticia (facilita las cosas!)*/
  head=nueva_entrada ("",NULL,0);
/* poner nombre archivo a índice */
  if(argc!=2)
  {
    fprinft(stderr,"\nusage es %s
nombrearchivo\n", *argv);
    salir(l);
  }
  if (enarchivo=
    fopen(*++argv,"r")==NULL)
    fprintf(stderr,"n\archivo no hallado
%s\n",
    * argav);
    salir(l);
}
  nw=siguientepalabra;
  while(c=getc(enarchivo)!=EOF)
  {
    if(c=='\n')
/* sumar uno al contador de líneas y
comprobar si final de página */
  {
    lc+=1;
    if(lc>LPP)
    {
    pc+=1;
    lc=0;
  }
}
else if (enpalabra)
{
  if(isalpha(c))
    *nw++=c;
  else
  {
    enpalabra=0;
    *nw='\o';
    insert(siguientepalabra,head,pc);
    nw=siguientepalabra;
  }
}
else
{
    if(isalpha(c))
    {
      enpalabra=1;
      *nw++=c;
    }
  }
}
if(enpalabra)
{
  *nw='\o';
  insert(siguientepalabra,head,
  pc);
}
  visualizar_índice;
}
insert(e)
ENTRY e;
{
insert(e,|,pnum)
ENTRY e;
enlace |;
int pnum;
static enlace último|;
{
  último|=head;
  if(|==NULL)
    {
      último|—>siguiente=siguiente_
entrada(e,|,pnum);
      return;
    }
  else
  {
    int s;
    s=strcmp(e,|—>entrada);
    if(s==0)
/* palabra ya presente de modo que añadir
número página */
    {
      añadir_número_página(pnum,
      |—> páginas);
      return;
    }
      else if(s>0)
/* se ha ido demasiado lejos de modo que
insertar nuevo nudo después del último de la
lista*/
    {
      último|—> siguiente=nueva_
entrada(e,|,pnum);
      return;
    }
    else
    {
/* no hallado aún de modo que recorrer la lista
utilizando una llamada recursiva a insert*/

      último|=|;
      insert(e,|—> siguiente,pnum);
      return;
    }
  }
}
enlace nueva_entrada(e,|,pnum)
ENTRY e;
enlace |;
int pnum;
{
/* tomar espacio suficiente para nueva entrada
  utilizando malloc */
  enlace nueva|;
  nueva|=(enlace)malloc(tamañode(elemento));
/* observe cast para convertir puntero char
  devuelto por malloc */
  nueva|—>entrada=e;
  nueva|—>siguiente=|;
  nueva|—>páginas= (penlace)malloc(tamañode(página)
  nueva|—>páginas—>pn=pnum;
  nueva|—>páginas—>psiguiente=NULL;
  return(nueva|);
}
añadir_número_página(pnum,pl)
int pnum;
plink pl;
{
/* hallar final de lista números de página */
  while (pl—> pnext)!=NULL)
    pl=pl—> pnext;
  pl—> psiguiente=(plink)malloc(tamañode(página));
  pl—> psiguiente—> psiguiente=NULL;
  pl—> psiguiente—> pn=pnum;
  return;
}
visualizar_índice;
enlace|;
penlace pl;
{
  |=head—> siguiente;
  while(|!=NULL)
  {
    prinft("%s\t",|—> entrada);
    pl=|—> páginas;
    while(pl—> siguiente!=NULL)
    {
      prinft("%d4,",pl—> pn);
      pl=pl—> siguiente;
    }
    prinft("%d\n",pl—> pn);
  }
  return;
}
```

Material fuente

Consideremos cuáles son las características que debe reunir un buen ensamblador y examinemos tres conocidos paquetes para ordenadores personales

Si bien un ensamblador es esencial para el programador serio, es sólo una de las muchas herramientas que se requieren. El ensamblador convierte el *código fuente* (escrito con los mnemotécnicos y símbolos del lenguaje assembly) original de la máquina a *código objeto* (el auténtico lenguaje máquina). También es necesario un programa monitor o depurador para verificar la corrección del programa y localizar sus fallos.

La mayoría de los ensambladores siguen el lenguaje assembly definido por el fabricante de la CPU, pero los ensambladores pequeños o sencillos a menudo utilizan una sintaxis no estándar para simplificar sus tareas. Para cargar un valor inmediato en el registro A de un 6502, por ejemplo, el lenguaje assembly correcto es LDA # $20, mientras que un ensamblador sencillo podría utilizar LDAIM $20. No obstante, se deben evitar los ensambladores no estándares, porque pueden ser difíciles de aprender y habrá de convertir los archivos fuente existentes antes de que se ensamblen.

Los ensambladores más sencillos están incorporados en programas depuradores. Denominados *ensambladores en línea*, permiten entrar y ensam-

Hisoft Devpac

Para el Amstrad CPC 464, 664 y 6128, Spectrum 48 K, micros CP/M y MSX

El conocido Devpac de Hisoft es una herramienta de desarrollo completa, con ensamblador, editor y depurador de código máquina. Los programas se desarrollan utilizando líneas numeradas similares al BASIC, pero también posee facilidades completas de edición: renumerar, suprimir y desplazar líneas, hallar y reemplazar, etc. El Devpac es del tipo de dos pasos, con una amplia gama de directivas y exhaustivas capacidades aritméticas. Una amplia gama de opciones de ensamblaje proporcionan al programador un control flexible del modo en que se produce el código. Tanto el programa ensamblador/editor como el depurador son reubicables, lo que permite cargarlos en la memoria en un punto adecuado para el programa que esté desarrollando. Si se cargan ambos programas al mismo tiempo, se enlazarán, permitiéndole a usted saltar fácilmente desde el ensamblador al depurador cuando ello sea necesario. El depurador imita el "panel frontal" de los ordenadores centrales produciendo una visualización de pantalla casi completa, mostrando el contenido de un bloque de memoria, los registros de la CPU y la próxima instrucción a ejecutarse. Además de las facilidades habituales, Hisoft proporciona facilidades de puntos de parada, ejecución paso a paso y seguimiento, convirtiéndolo en una herramienta ideal para depurar programas. Su completa visualización en pantalla es muy adecuada para los principiantes. Devpac viene con un detallado manual de 32 páginas. Aunque ni éste ni las instrucciones Devpac propiamente dichas son todo lo claras que deberían ser, las omisiones de información relativa a todo cuanto debe saber el programador son escasas. El Devpac es un paquete exhaustivo y bien producido

Tipo de ensamblador

Dos pasos, mnemotécnicos estándares

Límites

Código fuente: Limitado por la RAM, pero hay disponible enlace de archivos
Tamaño de la tabla de símbolos: especificado por el usuario

Directivas de ensamblador

ORG	Establecer el origen del programa
EQU	Asignar un valor a un símbolo
DEFB	Almacenar valor(es) de 8 bits
DEFW	Almacenar valor(es) de 16 bits
DEFS	Reservar espacio de almacenamiento
DEFM	Almacenar serie ASCII
ENT	Marca la dirección de comienzo del programa para la instrucción RUN del ensamblador
END	Marca el final del programa fuente
IF...ELSE	Ensamblaje condicionado
*E	Salto de página en impresora
*H	Define encabezamiento en listado
*S	Hacer una pausa en listado
*L	Activar y desactivar listado
*D	Dar direcciones de listado en decimal
*F	Incluir archivo fuente preescrito en este punto
*T	Escribir código objeto en cinta en vez de en la memoria

Opciones de ensamblaje

Producir tabla de símbolos o no
Generar código objeto o no
Producir listado assembly o no
Colocar código objeto tras tabla de símbolos o en dirección ORG
Comprobar que el código objeto no se sobreescriba en Devpac o no comprobarlo

Aritmética

+, −, *, /, MOD, AND, OR, NOT y XOR. Decimal, hexa y binario

blar directamente en la memoria una línea de lenguaje assembly. Pero no permiten que el programador asigne etiquetas y símbolos a las direcciones y valores utilizados en el programa, lo que los hace adecuados sólo para realizar cortas correcciones en programas que ya se hallen en la memoria.

Con frecuencia los depuradores incluyen facilidades tales como inserción de puntos de parada, ejecución paso a paso y seguimiento de la ejecución de un programa. La inserción de un punto de parada en el programa a depurar genera en él una instrucción de salto que dará el control al depurador cuando sea ejecutada y se suprimirá ella misma después de su ejecución. De este modo, se puede comprobar en cualquier punto el estado de los registros y las posiciones de memoria.

Los ensambladores completos pueden ser, como se ha dado en llamarlos, de *un solo paso* o de *dos pasos*. El ensamblador de un solo paso más sencillo lee el código fuente sólo una vez, de modo que si el programa salta hacia adelante hasta una instrucción etiquetada, el ensamblador no sabrá a qué dirección se refiere la etiqueta. En este caso, el ensamblador retrocederá y colocará la dirección una vez que la haya descubierto; sin embargo, la dirección real no aparecerá en el listado assembly. Es más común el uso de un ensamblador de dos pasos; éste lee el programa fuente una vez, comprobando su sintaxis y construyendo un listado completo de los símbolos utilizados en él antes de volver a leerlo y producir un listado assembly ya completo.

Todos los ensambladores poseen su propio juego de *directivas* o *pseudo-ops*. Son instrucciones que no forman parte del juego de instrucciones de la CPU y que no se traducen a código máquina. En cambio, indican al ensamblador que debe llevar a cabo alguna tarea específica, como conectar la impresora o reservar un bloque de espacio de memoria dentro del código para almacenamiento de datos. Las directivas varían de un paquete a otro, tanto en lo que hacen como en sus nombres y sintaxis. La mayoría de las directivas están bastante estandarizadas, pero las inusuales pueden representar un problema si está intentando lograr que un programa escrito para un ensamblador trabaje con otro.

La primera directiva con la cual se encontrará es ORG, que indica al ensamblador dónde se empezará a cargar en la memoria el programa a ensamblar. Asimismo, todos los ensambladores poseen una forma de definir símbolos: COUNT EQU 5, establecería el símbolo COUNTen 5. Además, es muy útil un buen juego de *directivas de datos*, que son instrucciones que le permiten establecer las áreas de almacenamiento de sus programas. Por lo general hay instrucciones para almacenar valores determinados (DEFB, DEFW) o series (DEFM), y para dejar libres algunos bytes para el programa (DEFS).

Operaciones aritméticas

La mayoría de los ensambladores pueden realizar aritmética, y es útil que proporcionen una gama completa de operadores: +, −, /, *, AND, OR, NOT y MOD. Este último, MOD (o un equivalente), es virtualmente indispensable, puesto que con frecuencia los programadores necesitan el ensamblador para dividir un valor de 16 bits en dos valores de ocho bits. Los buenos ensambladores también son capaces de manipular números en decimal, hexa y binario, así como de evaluar un carácter ASCII (de

Picturesque Editor/Assembler

Para el Spectrum de 16 K y 48 K

El Editor/Assembler de Picturesque viene siendo desde hace ya tiempo el favorito de los programadores del Spectrum, profesionales o aficionados indistintamente. Constituye un ensamblador y editor Z80 de dos pasos en el Spectrum, y opcionalmente se lo puede utilizar con el paquete Monitor de Picturesque. Opera tanto en máquinas de 16 K como de 48 K, si bien en la versión de 16 K sus facilidades son más restringidas. Para atenuar el problema que supone mostrar listados anchos en assembly en la visualización de 32 columnas del Spectrum, el programa simula una pantalla de 40 columnas.

La versión para 48 K posee la muy importante capacidad de ensamblar desde archivos de código fuente almacenados en cinta o en microdrive. En efecto, ello dispone de lugar para programas fuente de hasta 95 Kbytes de longitud y, por consiguiente, la creación de programas objeto grandes, como los necesarios para juegos comerciales. Ésta es una facilidad estándar en los ensambladores para micros de gestión. Muchos ensambladores para ordenadores personales se limitan a programas suficientemente cortos para que tanto su código fuente como su código objeto se hallen en la memoria al mismo tiempo. Picturesque proporciona un excelente manual de 56 páginas para el Editor/Assembler, que contiene información clara y detallada sobre todos los aspectos concernientes al paquete. A pesar de que existen alternativas más recientes que ofrecen más facilidades de ensamblaje, Picturesque es un excelente patrón y un paquete muy capaz por derecho propio

Tipo de ensamblador

Dos pasos, mnemotécnicos estándares

Límites

Código fuente: hasta 95 K utilizando microdrive
Tamaño de la tabla de símbolos: limitado por la memoria libre

Directivas de ensamblador

ORG	Establecer origen del programa
EQU,	Asignar valor a símbolo
DEFL	Como EQU, pero el símbolo se puede reasignar varias veces
DEFB	Almacenar valor(es) de 8 bits
DEFW	Almacenar valor(es) de 16 bits
DEFS	Reservar área de almacenamiento
DEFM	Almacenar serie ASCII
PRNT	Conectar y desconectar impresora
END	Marca el final del programa fuente

Opciones de ensamblaje

Ensamblaje desde memoria, cinta o microdrive
Listado assembly en impresora o en pantalla

Aritmética

+, −, < y > (bytes *high* y *low* de valores de 16 bits).
Hexa y decimal

modo que, por ejemplo, usted podría utilizar el símbolo A en lugar de tener que consultar por sí mismo su valor en una tabla ASCII).

Cuando usted solicita que se ensamble un programa, por lo general se le ofrece la posibilidad de especificar algunas opciones en cuanto al modo de llevar a cabo el ensamblaje. Éstas pueden incluir que se produzca o no un listado y tabla de símbolos, y que ésta sea visualizada o no en la pantalla, sea enviada a la impresora o se escriba en un archivo en disco o cinta. Una facilidad muy útil es un ensamblaje experimental, en virtud del cual el programa se ensambla pero no se carga en la memoria ningún código máquina (¡ésta es una forma rápida de comprobar que el programa se ensamble pero sin ensamblarlo realmente!).

Los mejores paquetes ensambladores permiten el ensamblaje tanto condicional como el uso de macros. Las directivas condicionales permiten ensamblar secciones de listado sólo si determinadas condiciones son verdaderas. Esto significa que un listado fuente puede generar varias versiones del código objeto según las exigencias. Una macro permite definir un nuevo mnemónico a partir de algunos de los ya existentes. Cuando se ensambla la macro, el ensamblador sustituye automáticamente las instrucciones correctas. Mediante la construcción de una biblioteca de macros para efectuar tareas determinadas (como imprimir mensajes, abrir archivos de datos, etc.), se pueden escribir programas con suma rapidez. No obstante, no son tan eficaces como los programas codificados a mano en cuanto a velocidad de ejecución y tamaño.

Aparte de las importantes directivas, la otra característica fundamental de los ensambladores es cómo están organizados. Muchos almacenan en la memoria tanto el código fuente como el código objeto, lo que limita el tamaño máximo del programa que se puede escribir. Una solución implica poder ensamblar a partir de un programa fuente en almacenamiento, lo que por lo general es práctico sólo en máquinas con discos o cintas rápidas, tales como la Sinclair Microdrive. Pero ésta es la única forma de crear programas grandes.

Además de las especificaciones del ensamblador, se debe comprobar su grado de rapidez. Un programa se ensamblará muchas veces a lo largo de su desarrollo, de modo que con frecuencia es preferible un pequeño ensamblador rápido a uno más lento con muchas facilidades. También es importante la calidad del editor suministrado. Lo ideal es un editor de pantalla completa similar a un procesador de textos, si bien muchos programas se basan en editores de línea.

Watford ROMAS

BBC Modelo B

ROMAS es un "ensamblador cruzado" profesional para el BBC B. Está diseñado para permitir el desarrollo de programas no sólo para el BBC sino para una amplia gama de micros. Se utiliza la directiva PROC para indicar al ensamblador para qué CPU está ensamblando. Actualmente el ROMAS soporta procesadores 6502, 6511, 8085, Z80, Z8, 6809, 65C02, 8041 y 8048, por lo cual es adecuado para casi todos los ordenadores de ocho bits y dispositivos controlados por microprocesador. ROMAS incluye utilidades que permiten la transmisión de programas objeto a otros micros y programadores EPROM. Además de ser un ensamblador cruzado, el ROMAS soporta ensamblaje condicional y posee una gama de directivas diseñadas para ayudar a mantener grandes programas y producir versiones de los mismos para ejecutar en diferentes máquinas. Por ejemplo, las directivas TYPE e INPUT permiten que el ensamblador solicite al usuario valores para símbolos durante el ensamblaje. En unión con las condicionales, es posible que al ensamblar un programa se pregunte "¿qué máquina? 1 = Spectrum, 2 = Amstrad, 3 = MSX" y luego se produzca la versión correcta del programa. ROMAS ensambla desde un archivo en disco y vuelve a escribir el código objeto otra vez en archivo en disco, que posteriormente será comprobado en el BBC, un segundo procesador BBC o el micro para el cual fue diseñado. El paquete incluye un editor en pantalla denominado EDT, que soporta macroinstrucciones; se pueden construir operaciones complejas a partir de instrucciones ya existentes. Indudablemente, ROMAS es uno de los editores más sofisticados que existen en la actualidad para un micro pequeño, incluyendo un manual de 156 páginas y un programa fuente de muestra (un desensamblador BBC). Si bien la mayor parte de los usuarios se sentirán satisfechos con el ensamblador incorporado en el BBC BASIC con la adición de un programa monitor, el ROMAS es una alternativa profesional para quienes desarrollan software seriamente

Liz Heaney

Tipo de ensamblaje

Ensamblador cruzado de dos pasos, mnemotécnicos estándares

Límites

Código fuente: limitado por la capacidad del disco
Tamaño de la tabla de símbolos: 7 K, pero se puede ampliar

Directivas de ensamblaje

ORG	Establecer origen programa
EQU	Asignar valor a símbolo
DB,DW,DS	Almacenar valores de 8 y 16 bits, series ASCII y reservar espacio
IF,ELSE,ENDIF	Ensamblaje condicional
END	Marca el final de un programa
EXTEND	Seguir ensamblando desde archivo fuente mencionado
HIMEM,LOMEN,MODE	Como en el BBC BASIC
TYPE,INPUT,GET	Interactuar con el usuario durante el ensamblaje
TITLE,PAGE,EJECT, LIST,TAB,WIDTH	Controles de listado
PROC	Seleccionar procesador para el cual ensamblar

Opciones de ensamblaje

Producir listado assembly en pantalla o en impresora

Aritmética

*, /, REM(modulo), +, −, AND, OR, NOT, XOR, desplaz. izq., desplaz. der. Decimal, hexa, octal y binario

Breve pausa

En el capítulo anterior estudiamos los mecanismos básicos de E/S en serie. Esta vez analizaremos las E/S a través de interfaces paralelas

Comenzaremos con una mirada a un ejemplo de salida en serie con una subrutina llamada mensaje. Esta rutina tiene la dirección del mensaje a ella pasado mediante el registro de dirección A3. Así, por ejemplo:

```
LEA TEXT,A3     establece un puntero hacia el
                mensaje TEX
JSR MENSAJE     salida del mensaje
```

dará salida al mensaje TEXT almacenado así:

```
TEXT:DC.B       'Mi Computer',$00
```

donde DC.B es una directiva del ensamblador para declarar el espacio de memoria que albergará el texto 'Mi Computer'. El byte nulo $00 representa el fin del mensaje, e informa a la subrutina mensaje que ya no hay más texto.

La subrutina para dar salida a todos los bytes en la tabla TEXT usará la subrutina OUTCH, que ya conocemos de un capítulo anterior. En el ejemplo, OUTCH empleaba el bit "preparado" para dar salida a los caracteres uno a uno tan pronto como estuviera lista la ACIA para enviar el carácter siguiente. La subrutina mensaje tendrá, pues, el siguiente aspecto:

```
MENSAJE: MOVE.B  (A3)+D0   toma el siguiente
                           byte de mensaje
         BEQ     DONE      comprueba el final
                           del mensaje
         JSR     OUTCH     salida de
                           caracteres
         BRA     MENSAJE   bucle hasta llegar
                           al final del
                           mensaje
DONE:    RTS
```

Cada byte del mensaje se copia en D0 (el registro de datos empleado como parámetro de valor para OUTCH), empleando A3 como puntero con direccionamiento indirecto posincremental. Si el byte es cero, se abandona la rutina mediante la etiqueta DONE; de lo contrario se da salida a un carácter a través de OUTCH.

Si precisamos dar salida a datos en el modo paralelo, en el que todos los bits de los datos aparecerán simultáneamente, el chip PIA nos proporciona las facilidades. Sin embargo, dado que éste es un chip interface para uso general, debemos configurarlo en la forma adecuada a la configuración particular de nuestro hardware. Esto se parece al chip ACIA, donde debemos establecer la velocidad de transmisión y los formatos de los bytes.

Para el PIA, habremos de configurar las ocho líneas paralelas de *ambas* mitades del chip (A y B) para que sean o bien líneas de salida o bien de entrada (es decir, debemos configurar la dirección de los datos). Para ello escribiremos en el bit 2 del registro de control (CRA o CRB) para indicar que deseamos establecer la dirección de los datos en un registro de dirección de datos (DDRA o DDRB). Por ejemplo:

```
CONFIGPIA: CLR.B   PIACRB       pone a cero
                                el bit de
                                registro de
                                control
           MOVE.B  #$FF,PIADDRB establece la
                                dirección de
                                los datos en
                                salida
           BSET    #2,PIACRB    retorna al
                                registro
                                normal de
                                datos
           CLR.B   PIACRA       da para A
                                la dirección
                                del sentido
                                de los datos
           CLR.B   PIADDRA      establece el
                                sentido de los
                                datos en
                                salida
           BSET    #2,PIACRA    vuelve el
                                registro
                                normal de
                                datos
```

Este fragmento establecerá todo el lado A para entrada, y el lado B para salida. La transmisión de los datos a los dispositivos paralelos se ejecutará con:

```
MOVE.B  D0,PIADRA   salida del contenido de D0
```

y para la entrada tendremos lo siguiente:

```
MOVE.B  PIADRA,D1   lee la entrada en D1
```

Obsérvese que todas las direcciones PIA a las que nos referimos deben establecerse inicialmente, como por ejemplo:

```
PIADRA  EQU  $30051
PIACRA  EQU  $30053
```

Obsérvese también que el significado de los datos leídos o escritos hacia el PIA dependen del tipo de dispositivo que se conecta eléctricamente a los canales digitales del chip (p. ej., podríamos tener una visualización de siete segmentos conectada, como se muestra en el diagrama donde puede verse que todo dígito decimal puede construirse mediante el segmento adecuado). Para visualizar el número 3, pongamos por caso, deberemos encender los segmentos 1, 4, 2, 5 y 3.

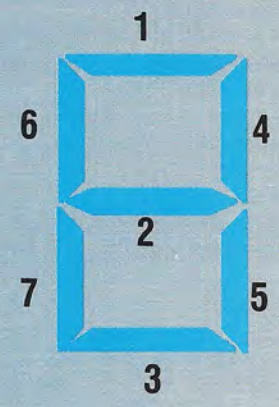
No sólo necesitaremos transmitir datos al dispositivo periférico sino que se puede necesitar el control de su función eléctrica. En el caso propuesto de una visualización con siete segmentos, puede que necesitemos fijar los datos en el dispositivo mediante algunos de los bits de reserva escritos en modo salida dentro de la palabra de datos. A menudo estas señales de control simulan el pulso de un "reloj" eléctrico, pudiendo traducirse en una activación y desactivación del bit de control. Así:

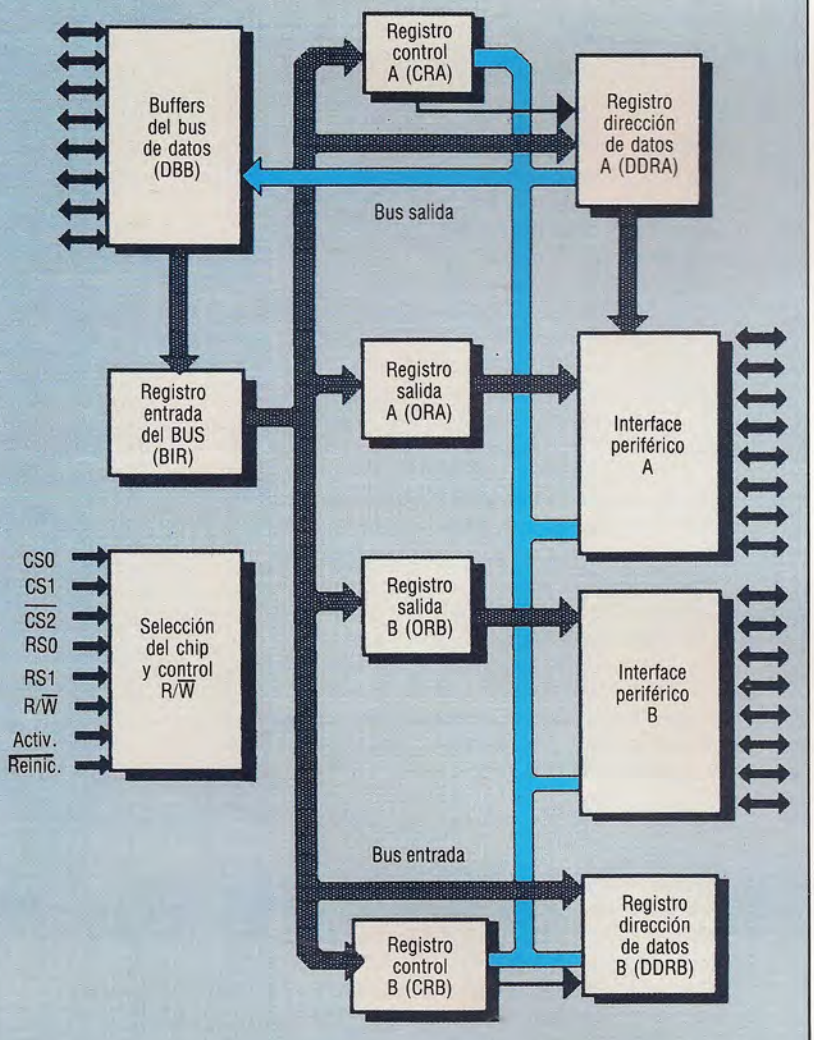
```
BSET    #CONBITNO,PIADRA
                    activa el bit de control
JSR     DELAY       espera un pequeño instante
BCLR    #CONBITNO,
        PIADRA      y lo reinicializa después
```

proporciona el "reloj" a cualquier canal digital que se asigne a CONBITNO en la dirección PIADRA.

Movimiento paralelo

El dibujo inferior muestra la estructura de un típico chip PIA, utilizado para proporcionar facilidades de entrada/salida al 68000. El chip proporciona dos puertas, asignadas convencionalmente: una para la entrada y otra para la salida. Cada puerta tiene sus propios registros de control, datos y dirección de datos. Una aplicación típica puede ser manejar una visualización de siete segmentos LED como el que se muestra en el dibujo (izquierda)





Caroline Clayton

Interrupciones

Aunque el tema de las interrupciones no siempre es bien entendido, la finalidad del uso de interrupciones en un sistema de ordenador tiene que ver con el uso eficiente de la CPU y el poder responder a eventos externos. Por ejemplo, puede que no deseemos que la CPU pierda el tiempo mientras se imprimen los caracteres, como en el ejemplo de la subrutina OUTCH. También necesitamos, para poder hacer otra cosa, saber cuándo se ha terminado de imprimir un carácter para enviar el siguiente disponible.

La situación es aún peor cuando el programa está a la espera de una entrada; digamos, por ejemplo, que del teclado. La eficiencia del sistema depende, como es obvio, de la velocidad de escritura y también de otras tareas asumidas por la CPU, tales como dar salida hacia la impresora en paralelo al tiempo que espera una entrada del teclado.

Para obtener esta operación en paralelo, debemos organizar la secuencia lógica de eventos dentro de la máquina para asegurarnos de que no perdemos el control del programa o algún dato. Veamos lo que se necesita.

- *Guardar el estado del ordenador.* Necesitamos hacer esto para poder volver al programa que hemos interrumpido sin ningún efecto notable o pérdida de datos (lo que sí perderemos inevitablemente es tiempo). Pero primero definiremos qué es lo que constituye el "estado" del ordenador.

Podemos considerarlo como el área total para programas y datos del programa del usuario con su estado de registro y contador del programa. En la práctica, y puesto que en general no se esperarán cambios en el programa del usuario desde la fuente de interrupciones, basta con guardar tan sólo el contador del programa (PC) y el registro de estado (SR). Esto nos dará una idea cabal del estado del programa interrumpido.

- *Localizar la fuente de interrupción.* Esto es necesario porque probablemente tendremos conectados al ordenador más de un dispositivo y necesitamos saber qué rutina de "servicio" se ha de ejecutar.

- *Inhibir interrupciones de otras fuentes.* Necesitamos hacerlo porque si llega otra interrupción mientras atendemos a una ya producida se pueden perder datos. Esto, sin embargo, sólo puede suceder si el tiempo invertido en atender la segunda interrupción ha sido mayor que el tiempo entre la llegada de los datos en la primera interrupción.

• *Entrar la rutina de servicio de la interrupción.* Necesitamos un mecanismo que haga esto, ya sea automáticamente o ejecutando una instrucción específica.

• *Volver al programa interrumpido.* Finalmente, después de haber atendido la interrupción, deberemos volver al estado exacto del programa inmediatamente anterior a la interrupción.

Veamos cómo se consigue esta organización lógica en el 68000. Ante todo, el estado de la máquina se guarda mediante la pila donde se almacena primero el PC del usuario (como palabra larga completa) y después el SR. Con este mecanismo, las interrupciones pueden anidarse de tal modo que se pueda interrumpir una rutina de tratamiento de interrupción.

Si la rutina de tratamiento de interrupción fuera a emplear algunos de los registros que ya hemos señalado para un propósito específico, la rutina puede salvar estos registros a la entrada y restaurarlos a la salida. La instrucción MOVEM hará esto en el 68000 con la mayor sencillez para nosotros:

```
MOVEM   D1,D3,-(SP)     apila los registros
                        de datos
                        código de servicio
                        de interrupción
MOVEM   (SP)+,D1,D3     y los restaura
```

El problema de localizar la fuente de interrupción es también fácil en el 68000, dado que a cada interrupción se le asigna generalmente una sola posición de memoria llamada *vector*, que puede considerarse que contiene un puntero hacia la dirección de la rutina de servicio. Naturalmente, pueden existir varios dispositivos en un determinado vector, en cuyo caso es necesario un *pooling* para establecer cuál es el dispositivo interruptor. Pero éste no suele ser el caso del 68000, por lo general.

La necesidad de arbitrar entre dispositivos interruptores también es atendida por usted. Y esto porque el 68000 asume la prioridad hardware de la fuente interruptora, es decir, *sólo* obtendrán la atención de la CPU las interrupciones de mayor prioridad. Esto significa que las fuentes de datos de alta velocidad tendrán una relativa alta prioridad en un sistema de multiinterrupciones, de modo que no se pierda dato alguno.

Una vez que la interrupción ha atraído la atención de la CPU, se carga en el PC el contenido del vector de interrupción, y se entra en la rutina de servicio de interrupciones. El dispositivo es atendido cargando los datos de entrada en un registro y de allí a algún buffer, o evacuando los datos de un buffer llevándolos a un dispositivo de salida.

La única acción que resta por hacer ahora es volver de la rutina de servicio al programa interrumpido. Esto se consigue casi de idéntica manera que una vuelta de subrutina, empleando esta vez una instrucción RTE y no una RTS. La instrucción RTE vuelve a cargar el PC y el SR automáticamente desde la pila de sistema, por lo que todas las rutinas de servicio deben concluir con esta instrucción. Significa también que en ese momento A7 (el puntero de la pila de sistema) debe estar apuntando a los registros del programa interrumpido. Entonces si usted coloca datos en la pila de sistema, ¡asegúrese de que son sacados de allí antes de ejecutar la instrucción RTE! O quizá más apropiadamente debería usar otra pila, dado que con el 68000 es posible emplear como pila cualquiera de los registros de direcciones.

Rutina de servicio

Veamos ahora una rutina de servicio de interrupciones típica donde tendremos también que hacer *pooling* de los dispositivos para encontrar la fuente interruptora. Dentro de nuestro sistema teórico, tenemos dos posibles fuentes de interrupción en lo que se llama nivel 4 (una línea de interrupción hardware de prioridad 4). Los dos dispositivos son un teclado externo en solitario y un reloj de tiempo real. El reloj se emplea para contar segundos, que sirven a otras partes del sistema (software).

Para establecer correctamente el vector de interrupciones para el nivel 4 (en la dirección $70), necesitamos inicializarlo cuando se inicie por primera vez el programa, como en:

```
MOVE.L  #EXTDEVS,$70    establ. vector nivel 4
```

Es obvio que puede haber otros registros por inicializar, en particular la pila del usuario y la pila de sistema. Por ejemplo, podemos usar:

```
MOVE.L  #STACK,SP       establ. pila sistema
MOVE.L  #USERTACK,A0    y pila usuario
```

donde establecemos A7 con la dirección STACK y A0 con la dirección USERSTACK.

La rutina de servicio de interrupciones EXTDEVS sería, por ejemplo:

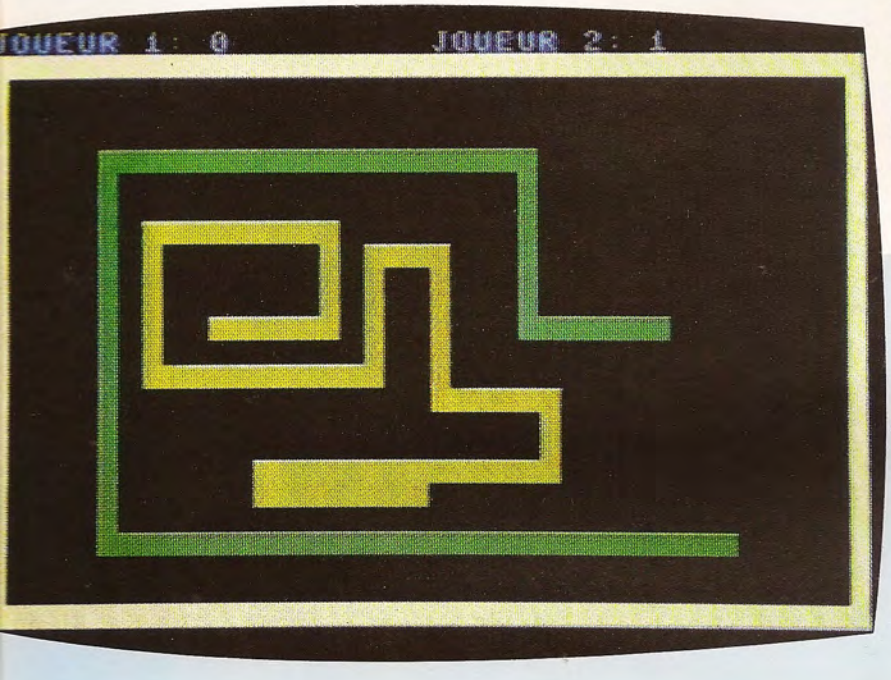
```
EXTDEVS MOVEM.L  D0-D7,(A0)  guarda los
                             regs. de datos
        BTST     #7,PIACRA   mira si está
                             el teclado
                             interrumpido
        BNE      CHAR
CKCLK   BTST     #6,PIACRB   si no, lo estará
                             el reloj
        BNE      CLOCK
        BRA      WILD        si no, entonces
                             espurio
CHAR    JSR      KEYBOARD    entra el
                             servicio
                             del teclado
        BRA      CKCLK       pero el reloj
                             puede estar
                             también
                             interrumpido!
CLOCK   JSR      SECS        entra el
                             servicio del reloj
WILD
        MOVEM.L  (SP)+,D0-   restaura los
                 D7          registros
                             de datos
        RTE
```

En esta rutina son guardados todos los registros de datos porque tanto CLOCK como CHAR los usarán. Nótese también que hacemos *pooling* de los dos dispositivos mirando los bits de estado de los registros en los registros de control del PIA. Comprobar el estado del reloj después de haber atendido al teclado, ya que puede tener a la vez dos interrupciones.

En el capítulo final de esta serie examinaremos el ensamblador y las facilidades que ofrece. Si se utilizan bien, quizá resulte la herramienta de desarrollo más poderosa de que dispone el programador.

Trazos en el C64

Como es sabido, este juego propone una auténtica "guerra de líneas". Pero los usuarios de un Commodore 64 ya pueden reemplazar la hoja de papel por la pantalla de su ordenador



Dos jugadores se enfrentan para dividirse el espacio vital. Cada uno de ellos debe esforzarse, al irse desplazando, para no cortar jamás su trazado o el de su adversario, y sin salirse del rectángulo dibujado en la pantalla. Los mandos a utilizar son:

Jugador de la derecha: <P>,<L>,<:> y <.>
Jugador de la izquierda: <W>,<A>,<S> y <Z>.

```
  5 REM  ************
 10 REM *  TRAZOS  *
 15 REM  ************
 20 GOSUB 1000
100 GET X$
110 C1=(X$="A")-(X$="S")+40*((X$="W")
    -(X$="Z"))
120 C2=(X$="L")-(X$=":")+40*((X$="P")
    -(X$="."))
130 IF C1<>0 THEN D1=C1
140 IF C2<>0 THEN D2=C2
150 P1=P1+D1
160 IF PEEK <P1><>32 THEN 3000
170 POKE P1,C
180 POKE P1+M,K1
190 P2=P2+D2
200 IF PEEK<P2><>32 THEN 4000
210 POKE P2,C
220 POKE P2+M,K2
230 FOR I=1 TO 50
240 NEXT I
250 GOTO 100
1000 P1=1514
1010 P2=1534
1020 K1=7
1030 K2=5
1040 C=160
1050 M=54272
1060 D1=1
1070 D2=-1
2000 PRINT CHR$<147>;
2010 POKE 53280,0
2020 POKE 53281,0
2030 FOR I=0 TO 39
2040 POKE 1064+I,C
2050 POKE 1064+I+M,1
2060 POKE 1984+I,C
2070 POKE 1984+I+M,1
2080 NEXT I
2090 FOR I=1 TO 22
2100 POKE 1064+I*40,C
2110 POKE 1064+I*40+M,1
2120 POKE 1103+I*40,C
2130 POKE 1103+I*40+M,1
2140 NEXT I
2160 POKE P1,C
2170 POKE P1+M,K1
2180 POKE P2,C
2190 POKE P2+M,K2
2200 PRINT "JUGADOR(1SPC)1:";J1,"JUGADOR
    (1SPC)2:";J2;
2210 RETURN
3000 J2=J2+1
3010 FOR I=1 TO 500
3020 GET X$
3030 NEXT I
3040 IF J2=10 THEN 5000
3050 GOTO 20
4000 J1=J1+1
4010 FOR I=1 TO 500
4020 GET X$
4030 NEXT I
4040 IF J1=10 THEN 5000
4050 GOTO 20
5000 PRINT CHR$(147);
5010 FOR I=1 TO 18
5020 PRINT
5030 NEXT I
5040 PRINT TAB(10)"EL(1SPC)JUGADOR
    (1SPC)";
5050 IF J1>J2 THEN PRINT "1";
5060 IF J2>J1 THEN PRINT "2";
5070 PRINT "(1SPC)GANA(1SPC)!"
5080 PRINT
5090 PRINT TAB(13)J1;"A";J2
5100 FOR I=1 TO 4
5110 PRINT
5120 GET X$
5130 NEXT I
5140 PRINT TAB(13)"OTRA(1SPC)?"
5150 GET X$
5160 IF X$="" THEN 5150
5170 IF X$<>"N" THEN RUN
5180 END
```

©SYBEX

Kevin Jones Associates

Puente informático

Concebido originalmente por Alan Kay, de Xerox, el futurista Dynabook se podría construir aplicando la tecnología de hoy día, aunque su precio lo convertiría en una propuesta poco práctica. En el futuro, sin embargo, un dispositivo como el que vemos podría convertirse en una ayuda esencial para todos, especialmente para aquellos grupos (la tercera edad, p. ej.) que están quedando cada vez más aislados en una sociedad tecnológica de cambios rápidos. Una pantalla redefinible sensible al tacto para la entrada, una visualización LCD de gran resolución, entrada de voz y una red de radio celular permitirían que el usuario controlara e interactuara con sistemas que irían desde informes meteorológicos locales hasta servicios bancarios personales. Puesto que el servicio se autofinanciaría, la unidad estándar podría suministrarse gratuitamente o bien bajo alquiler nominal del gobierno central

Un ordenador universal

Se conoce por Dynabook un conjunto de especificaciones ideales para un ordenador de ámbito universal formuladas en 1969

En 1969, un joven estudiante norteamericano de informática llamado Alan Kay presentó una tesis doctoral en la cual se imaginaba la invención de un ordenador portátil útil a nivel universal denominado Dynabook. El Dynabook habría de satisfacer todas las necesidades de proceso de información del usuario, incluyendo comunicaciones de audio y visuales y el acceso a bibliotecas de información pública. En 1969, el microprocesador no se había inventado aún y el tamaño de los ordenadores iba desde el de una gran nevera hasta el de varios armarios; lo que es más, las etiquetas de sus precios en dólares terminaban con al menos cinco ceros. El Dynabook de Kay era un auténtico sueño del futuro.

En 1971 Kay comenzó a trabajar en el Centro de Investigación de Xerox de Palo Alto (Palo Alto Research Centre: PARC) y ejerció su influencia en el grupo de investigaciones sobre el lenguaje (Language Research Group) que ideó el SMALLTALK, concebido originalmente como el lenguaje de programación del Dynabook y su sistema operativo. El equipo de Xerox nunca construyó el Dynabook: incluso tras la invención del microprocesador, en 1971, la tecnología existente sencillamente no era lo bastante potente. (Sin embargo, sí construyeron un ordenador portátil del tamaño de un maletín, unos cinco años antes de que Osborne hiciera renacer la idea.) No obstante, las concepciones que encerraba el SMALLTALK, tales como ventanas, iconos y ratones, poco a poco fueron filtrándose a la industria del ordenador, y finalmente condujeron al Lisa y el Macintosh de Apple y al Atari 520ST. Por su parte, Alan Kay trabaja actualmente como investigador para la Apple Corporation.

Una historia tan "antigua" como ésta es instructiva, porque el momento de construir el Dynabook está ya muy próximo: los componentes necesarios ya están disponibles o lo estarán muy pronto. Hoy, la visión de Alan Kay empieza a parecer viable.

El concepto original del Dynabook era un dispositivo del tamaño aproximado al de un libro encuadernado (*dyna[mic] book*: libro dinámico), y totalmente portátil gracias a su funcionamiento a pilas.

Requisitos tecnológicos

Demos una mirada a algunas de las tecnologías ya existentes o que están surgiendo y que podrían hacer realidad este ordenador del futuro.

- **Potencia de proceso:** Es necesario que en el centro de la máquina haya muchísima potencia de proceso. Ahora se pueden obtener en las tiendas microprocesadores de 32 bits con la potencia de un ordenador central en forma de chips individuales. La lista incluye al Motorola 68020, el Inmos T414 Transputer, el Intel 80386 y el Acorn ARM. Cada uno de ellos tiene capacidad de ejecutar entre tres y 10 millones de instrucciones por segundo.
- **Memoria:** Los fabricantes están produciendo chips de RAM de 256 Kbits, y son muchos los ordenadores personales que los incorporan. Pero todos están trabajando en la próxima generación de chips de un megabit, y algunos (incluyendo a la AT&T) en la actualidad hacen publicidad de componentes de muestra.
- **Visualización:** Varios fabricantes están produciendo visualizaciones de cristal líquido monocromáticas y planas que pueden simular la pantalla para gráficos de 640 × 200 del IBM PC. Ahora se están utilizando paneles electroluminosos para iluminar por detrás tales visualizaciones y salvar su principal inconveniente, es decir, su bajo contraste y sus limitados ángulos de visión. Además, la tecnología de la visualización plana se está desarrollando más rápidamente que casi todos los otros sectores de la industria, anunciándose nuevos desarrollos con una frecuencia casi semanal. Fabricantes japoneses, entre los que se incluye Epson, ya han mostrado prototipos de LED a color, mientras que Toshiba y otros están trabajando en tubos de plasma autoiluminados. El punto clave de la mayor parte de este trabajo de investigación es la búsqueda de una televisión de pantalla plana viable, que es lo que necesita el Dynabook. También habrá visualizaciones en pantalla plana sensibles al tacto, que probablemente eliminarán la necesidad del teclado. Una gran visualización sobre la superficie del ordenador incluiría una "imagen" sensible al tacto de un teclado, adaptable a medida bajo control de software. En cada etapa de un programa, se vería un conjunto diferente de "teclas" con etiquetas especiales, y se podrían dibujar imágenes directamente con un punzón en vez de con un ratón.
- **Fuente de alimentación:** El suministro de corriente eléctrica constituye el mayor problema. Las exigencias de las nuevas visualizaciones y las grandes memorias serían excesivas para las pilas de litio que se utilizan en los portátiles de regazo actuales, y la tecnología de la pila no está progresando necesariamente al mismo paso que el desarrollo de semiconductores. Sin embargo, poco a poco van surgiendo nuevos tipos de pilas con densidades de energía más elevadas. Y, al mismo tiempo, están declinando los requerimientos de potencia de los chips a medida que los fabricantes deciden aplicar la tecnología CMOS de bajo consumo.
- **Almacenamiento masivo:** Se cree que el disco láser compacto reemplazará con el tiempo a los diferentes tipos de almacenamiento magnético, tales como la cinta y los discos flexibles y Winchester. El Dynabook podría utilizar discos compactos de lectura como medio de distribución para poner en redes públicas "libros" de un interés muy especializado, música y publicaciones de video. Otra forma de almacenamiento masivo es la *tarjeta astuta*. Se trata de un dispositivo de memoria del tamaño de una tarjeta de crédito que se puede utilizar para todo, desde distribución de software hasta para llevar las cuentas bancarias; la empresa británica Cumana ahora distribuye software para ordenadores personales con su tarjeta Astron. En un futuro cercano, tales tarjetas podrían utilizarse para abonar mercancías y servicios a través del Dynabook
- **Sonido:** Durante un tiempo ha sido posible colocar todos los componentes de un sistema de audio en un único chip, un buen ejemplo de lo cual es el chip que contiene el Walkman Sony. Los receptores de radio y televisión se ven igualmente afectados (de allí la existencia de los televisores de bolsillo), y los chips de sintetizadores de música están progresando rápidamente. La voz, sin embargo, es un asunto diferente. Si bien la síntesis de voz es bastante fácil de obtener y los chips sintetizadores que se utilizan en la música moderna pueden realizarla a la perfección, la comprensión del lenguaje natural se encuentra aún en una etapa de desarrollo muy temprana.
- **Comunicaciones:** Las restantes piezas del rompecabezas del Dynabook pertenecen al área de las comunicaciones. Los modems telefónicos en un chip son ahora de uso común, pero no

Incluiría una visualización de gráficos/televisión en color y sería capaz de presentar y procesar texto, imágenes y sonido. Sobre todo, sería un poderoso medio de comunicación.

Un ordenador auténticamente revolucionario no ha de ser sólo portátil, sino también capaz de intercambiar toda clase de datos con otros usuarios, proporcionándonos un medio preponderante de comunicación a larga distancia, como lo es ahora el teléfono. En resumen, podríamos transmitir una carta completa, con imágenes y sonido, y recibir una respuesta similar.

Pero esto no es más que el comienzo. Tendríamos acceso a diversos servicios públicos, desde bibliotecas, que nos proporcionarían libros y programas para ordenador que se cargarían en el sistema, así como noticias, información meteorológica, entretenimiento y facilidades educativas. También dispondríamos de informes sobre el estado de cuentas bancarias, comprobación de guías de calles, pedido de mercancías y reserva de billetes de viaje; todo ello desde cualquier punto e independientemente de las líneas telefónicas y los enchufes de la red eléctrica.

Si bien gran parte de la tecnología que se describe bajo el título *Requisitos tecnológicos* se halla en la vanguardia del desarrollo y, por lo tanto, es costosa, la financiación del Dynabook no constituye verdaderamente un problema; si se concretara de modo racional, sería el artículo de fabricación masiva definitivo (en cuanto a que todos y cada uno de los habitantes del mundo podría tener uno). Con tales volúmenes de producción, el costo de hardware por unidad sería prácticamente insignificante. Los problemas de marketing que tanto preocupan a los fabricantes de ordenadores personales de hoy en día se volverían bastante triviales si los ordenadores fueran realmente *útiles* para la gente: y el Dynabook, más que útil, sería indispensable.

Tampoco puede decirse que el mayor problema radique en la tecnología. Radica, más bien, en encontrar el incentivo para dar una forma concreta al sistema. En última instancia será un problema político, porque sólo los gobiernos poseen los recursos necesarios para tal cometido. Y, sin embargo, todo lo que haría el Dynabook ya se hace, con frecuencia a cargo de los estados, pero con sistemas distintos, ineficaces y caros. No es demasiado inverosímil imaginar una situación en la que el estado pudiera permitirse *regalar* Dynabooks, tal como está haciendo actualmente el gobierno francés con los terminales telefónicos de microordenador (son más baratos que imprimir y distribuir guías telefónicas de papel). Se pagaría más por los servicios que por el hardware, tal como ahora pagamos facturas de teléfono.

Liz Heaney

En el aire
La introducción de una red de radio celular ha estimulado el desarrollo de modems de radio celular. El modelo que vemos, el Transam M1, es un modem inteligente que ofrece facilidades de respuesta automática y llamada automática si se lo utiliza con un teléfono estándar. No obstante, también configura un conector de tipo D de 15 vías para acoplar a un transceptor, posibilitando el envío y la recepción de datos a través de la red de radio celular.

libertad de movimientos, el enlace se debe obtener por emisión radiofónica, no por cable. Ahora están comenzando a aparecer los primeros chips de modem de radio celular. El Dynabook se basa en una red de comunicaciones internacional de capas múltiples. Grandes ordenadores centrales contienen la información básica, y el primer paso para alcanzarlos es un enlace de radio celular con una estación local de ordenadores. Este nudo de red se comunica con los centros regionales mediante cables de fibra óptica con base en tierra, y éstos, a su vez, se comunican mediante más fibras ópticas y enlaces por satélite con los siguientes niveles de la jerarquía. Muchos de estos componentes ya se están instalando; lo que impide cualquier desarrollo ulterior es la inexistencia de una estandarización.

Punto final

Finalmente proporcionaremos el listado de ampliación de textos 6502 y un programa cargador en BASIC

Los usuarios del 6502 que no dispongan de ensamblador, o que deseen economizar tiempo de entrada, pueden colocar (POKE) las rutinas de compresión/ampliación de textos en la memoria utilizando el programa Cargador en BASIC 6502. Una vez ejecutado este programa, se puede borrar y sustituir por el programa Activador en BASIC. Éste es adecuado para usar tanto con las rutinas 6502 como con las versiones Z80 que hemos ofrecido previamente.

El programa Activador en BASIC le pide que entre la serie a comprimir y que especifique una dirección para los datos comprimidos a almacenar. Usted, por supuesto, deberá anotar esta dirección por si en una etapa ulterior quisiera recuperar los datos. Esto se realiza tan sólo especificando la dirección de los datos comprimidos de modo que resulten accesibles mediante la rutina de descompresión.

La rutina de compresión/descompresión Z80 se halla en la posición 30000, que resultará idónea para la mayoría de los micros. Si posee un ensamblador, puede reubicar el código simplemente cambiando la sentencia ORG. El programa Activador en BASIC se puede modificar para incluir sus propios programas, quizá para proporcionar facilidades para construir archivos más grandes de texto comprimido compuestos por registros de 255 bytes.

Ampliación de textos 6502

La siguiente rutina en lenguaje assembly se debe añadir a la rutina de compresión 6502 que ofrecíamos en la página 2346, insertándola antes de las tablas de datos finales. Esta rutina se ha escrito en el Commodore 64. Los usuarios del BBC Micro pueden convertirla introduciendo estos cambios:

- Cambiar las asignaciones de punteros de página cero, ZPTR1 a ZPTR4, a las direcciones &80 a &87.
- El formato de instrucción LABEL *=*+2 se debe cambiar por .LABEL EQUW
- El formato de instrucción LABEL *=*+1 se debe cambiar por .LABEL EQUB
- El formato de instrucción .BYT se debe cambiar por EQUS
- Las direcciones de llamada dependerán de dónde se ensamble la rutina. Las etiqs. START y EXPAND denotan las direcciones de llamada de las rutinas de compresión y ampliación y se acceden desde BASIC

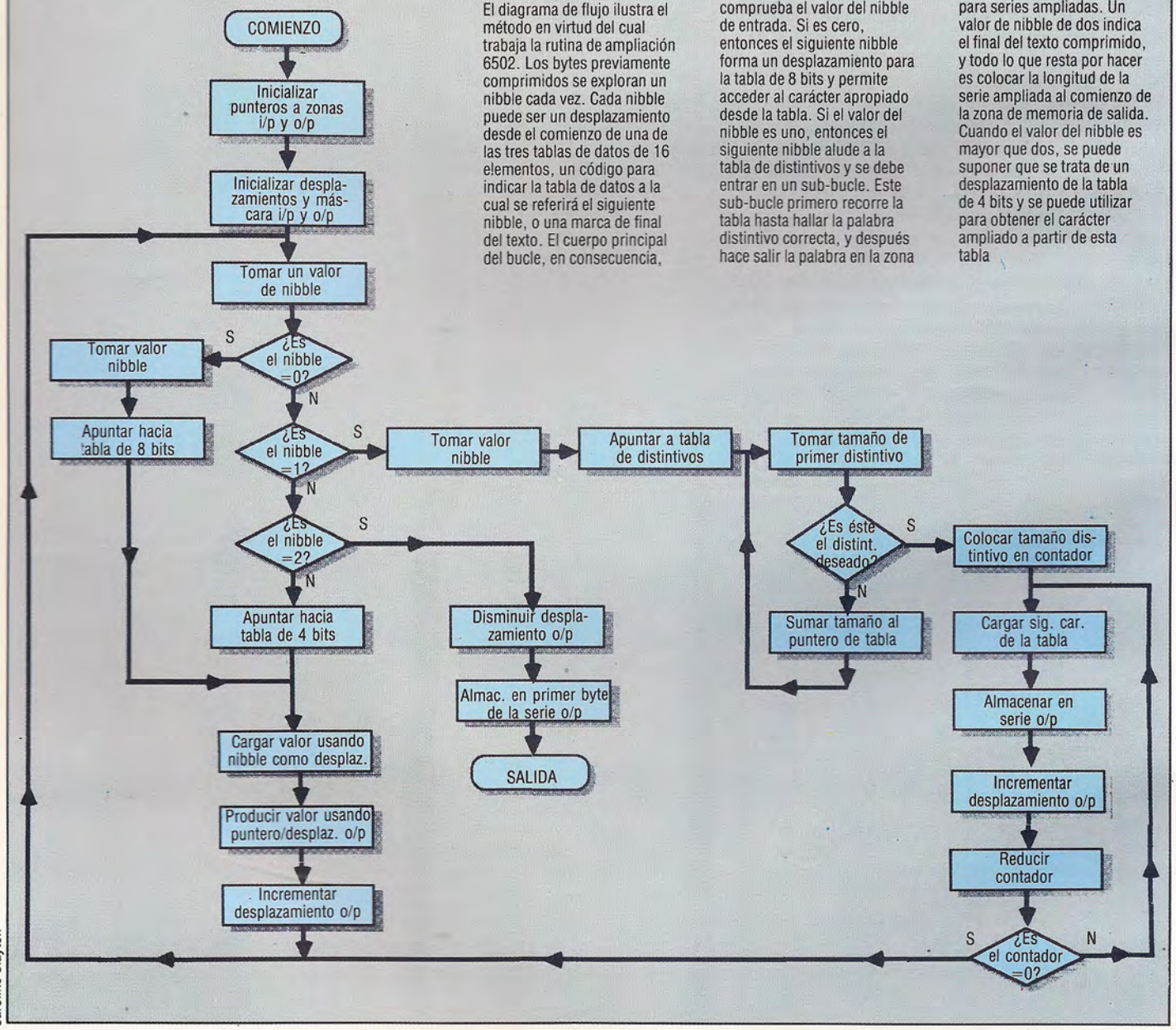
```
;
;++++RUTINA AMPLIACION 6502++++
EXPAND  LDA   INPUT
        STA   ZPTR1
        LDA   INPUT+1           ;ESTABLECER PAGINA 0
        STA   ZPTR1+1           ;PUNTEROS PARA APUNTAR
        LDA   OUTPUT            ;HACIA ZONAS DE
        STA   ZPTR2             ;ENTRADA Y SALIDA
        LDA   OUTPUT+1
        STA   ZPTR2+1

        LDA   #255              ;INICIALIZAR
        STA   MASK              ;DESPLAZAMIENTOS
        LDA   #1                ;Y MASCARA
        STA   LEN               ;INIC DESPL. I/P
        STA   OUTOFF            ;INIC DESPL. O/P
;++AQUI EMPIEZA EL BUCLE PRINCIPAL++
NEXNIB  JSR   GETNIB
        BEQ   OUT8BT            ;NIBBLE 0 SIGNIFICA CODIGO 8BITS
        CMP   #2
        BEQ   EXIT              ;ES LA MARCA DE FINAL?
        BCC   OUTTOK            ;ES UN DISTINTIVO?
;++TRATAR CODIGO 4BITS++
        TAY                     ;VALOR NIBBLE EN Y
        LDA   #<TAB4BT
        STA   ZPTR3             ;ESTABLECER ZPTR3 EN
        LDA   #>TAB4BT          ;COMIENZO DE TABLA
        STA   ZPTR3+1           ;DE 4 BITS
OUTCHR  LDA   (ZPTR3),Y         ;TOMAR CAR. DE LA TABLA
        LDY   OUTOFF            ;ESTABLECER Y PARA O/P
        STA   (ZPTR2),Y         ;ALMACENAR EN SERIE O/P
        INC   OUTOFF            ;BUMP PUNTERO DESPL. O/P
        JMP   NEXNIB            ;IR POR SIGUIENTE NIBBLE
;++TRATAR CODIGO 8 BITS++
OUT8BT  JSR   GETNIB            ;TOMAR NIBBLE CODIGO
        TAY                     ;PONERLO EN Y
        LDA   #<TAB8BT          ;ESTABLECER ZPTR3 PARA
        STA   ZPTR3             ;APUNTAR AL COMIENZO
        LDA   #>TAB8BT          ;DE TABLA DE 8 BITS
        STA   ZPTR3+1
        JMP   OUTCHR            ;SACARLO
;++TRATAR DISTINTIVO++
OUTTOK  JSR   GETNIB            ;TOMAR PUNTERO DISTINTIVO
        STA   TABCNT            ;ALMACENARLO
        LDA   #<TOKTAB          ;ESTABLECER ZPTR3 PARA
        STA   ZPTR3             ;QUE MIRE AL COMIENZO
        LDA   #>TOKTAB          ;DE TABLA DE DISTINTIVOS
        STA   ZPTR3+1
        LDX   #0
        LDY   #0
TOKA    LDA   (ZPTR3),Y         ;TOMAR LONGITUD DE DISTINTIVO
        CPX   TABCNT            ;SI TENEMOS EL QUE QUEREMOS
        BEQ   FOUND             ;BIFURCACION
        CLC
        ADC   #1
        ADC   ZPTR3             ;ESTABLECER ZPTR3 EN COMIENZO
        STA   ZPTR3             ;DE SIGUIENTE DISTINTIVO
        LDA   ZPTR3+1
        ADC   #0
        STA   ZPTR3+1
        INX
        BNE   TOKA              ;ESTE SIEMPRE SE DEBE BIFURCAR!
FOUND   TAX                     ;COLOCAR LONGITUD DISTINTIVO EN X
        LDY   #1                ;INICIALIZAR Y
FOUND1  LDA   (ZPTR3),Y         ;TOMAR CAR. DISTINTIVO
        STY   TEMP              ;ALMACENAR TEMP Y
        LDY   OUTOFF            ;TOMAR DESPL. O/P
        STA   (ZPTR2),Y         ;ALMACENAR CAR. EN SERIE O/P
        INC   OUTOFF            ;BUMP DESPL. O/P
        LDY   TEMP              ;VOLVER A TOMAR DESPL. DISTINTIVO
        INY
        DEX
        BNE   FOUND1            ;SI NO FINAL, TOMAR SIGUIENTE CAR.
        JMP   NEXNIB            ;RETOMAR BUCLE PRINCIPAL
;++TRATAR SALIDA++
EXIT    LDA   OUTOFF            ;TOMAR DESPL. O/P
        SEC
        SBC   #1                ;REDUCIR EN UNO
        LDY   #0                ;Y ALMACENAR AL PRINCIPIO
```

```
                STA   (ZPTR2),Y       ;DE SERIE O/P
                RTS                   ;RETORNAR AL SISTEMA
;++SUBRUTINA TOMAR UN NIBBLE++
GETNIB          LDA   MASK
                BNE   RIGHT           ;ES MASCARA 255?
                LDA   #255            ;PREPARAR MASCARA
                STA   MASK            ;PARA LA PROXIMA VEZ
                LDY   LEN             ;TOMAR DESPL. I/P
                LDA   (ZPTR1),Y       ;TOMAR BYTE I/P
                AND   #15             ;ENMASCARARLO
                INC   LEN             ;BUMP DESPL. I/P
                RTS
RIGHT           LDA   #0              ;PREPARAR MASCARA
                STA   MASK            ;PARA LA PROXIMA VEZ
                LDY   LEN             ;TOMAR DESPL. I/P
                LDA   (ZPTR1),4       ;TOMAR BYTE I/P
                LSR   A               ;DESPLAZAR NIBBLE IZQUIERDO
                LSR   A               ;HACIA LA MITAD DERECHA
                LSR   A               ;DEL BYTE
                LSR   A
                AND   #15             ;ENMASCARARLO
                RTS
```

Activador en BASIC Z80/6502

```
100 REM ***********************************
110 REM ** Programa para comprimir/expandir **
120 REM **                               **
130 REM * una serie de 255 bytes (solo mayusc.) *
140 REM ***********************************
150 CLS
160 INPUT " Version 6502 (S/N)";a$
170 IF a$="s" OR a$="S" THEN cf=1
180 INPUT "Expandir o Comprimir (E/C)?";a$
190 IF a$="" THEN COTO 180
200 IF a$="e" OR a$="E" THEN ec=1:GOTO 260
```

Ampliación



El diagrama de flujo ilustra el método en virtud del cual trabaja la rutina de ampliación 6502. Los bytes previamente comprimidos se exploran un nibble cada vez. Cada nibble puede ser un desplazamiento desde el comienzo de una de las tres tablas de datos de 16 elementos, un código para indicar la tabla de datos a la cual se referirá el siguiente nibble, o una marca de final del texto. El cuerpo principal del bucle, en consecuencia, comprueba el valor del nibble de entrada. Si es cero, entonces el siguiente nibble forma un desplazamiento para la tabla de 8 bits y permite acceder al carácter apropiado desde la tabla. Si el valor del nibble es uno, entonces el siguiente nibble alude a la tabla de distintivos y se debe entrar en un sub-bucle. Este sub-bucle primero recorre la tabla hasta hallar la palabra distintivo correcta, y después hace salir la palabra en la zona para series ampliadas. Un valor de nibble de dos indica el final del texto comprimido, y todo lo que resta por hacer es colocar la longitud de la serie ampliada al comienzo de la zona de memoria de salida. Cuando el valor del nibble es mayor que dos, se puede suponer que se trata de un desplazamiento de la tabla de 4 bits y se puede utilizar para obtener el carácter ampliado a partir de esta tabla

Caroline Clayton

```
210 IF a$="c" OR a$="C" THEN ec=0:GOTO 430
220 GOTO 180
230 REM
240 REM *** Expandir una serie comprimida ***
250 REM
260 PRINT "Donde esta la serie a ampliar?"
270 INPUT i
280 PRINT "Donde quiere la salida?"
290 REM Tenga cuidado de no machacar lenguaje
    maquina!
300 INPUT o
310 IF cf=0 THEN GOSUB 660:REM version Z80
320 IF cf=1 THEN GOSUB 700:REM version 6502
330 FOR x=1 TO PEEK(o)
340 PRINT CHR$(PEEK(o+x));
350 NEXT x
360 PRINT; "Continuar (s/n)? ":INPUT a$
370 IF a$ <> "S" AND a$ <> "s" THEN STOP
380 PRINT: GOTO 180
390 STOP
400 REM
410 REM ****** Comprimir una serie ******
420 REM
430 PRINT "Que he de comprimir? Recuerde que"
440 PRINT "La rutina solo acepta letras
    mayusculas"
450 PRINT
460 INPUT "Serie a comprimir: ";a$
470 PRINT "Donde debe ponerla?"
480 REM Ver advertencia de la linea 290!
490 INPUT i
500 PRINT "Donde quiere la salida?"
510 INPUT o
520 POKE i,LEN(a$)
530 FOR x=1 TO LEN(a$)
540 POKE i+x,ASC(MID$(a$,x,1))
550 REM poke i+x, code a$(x to x) en el
    Spectrum
560 NEXT x
570 GOSUB 660
580 PRINT "Longitud de la salida ";PEEK (o)
590 PRINT "Compresion conseguida: ";100-
    PEEK(o)/LEN(a$)*100;"%"
600 PRINT: "Continuar (s/n)? ":INPUT a$
610 IF a$<> "S" AND a$ <> "s" THEN STOP
620 PRINT: GOTO 180
630 POKE 30003,INT(i/256)
640 POKE 30002,i-PEEK(30003)*256
650 POKE 30005,INT(o/256)
660 POKE 30004,o-PEEK(30005)*256
670 RETURN
680 IF ec=1 THEN CALL 30270:RETURN
690 CALL 30000: IF PEEK (30006) <> 0 THEN PRINT
    "Compresion fallida": STOP
700 REM Version Commodore 6502
710 POKE 49153, INT (0/256)
720 POKE 49152, i-PEEK
730 POKE 49155, INT (i/256)
740 POKE 49154, i-PEEK(49155)*256
750 IF ec=1 THEN sys 49163
760 sys 49518: IF PEEK(49156)<>0 THEN
    PRINT"compresion fallida":STOP
770 RETURN
```

Cargador en BASIC 6502

```
 10 REM ***********************************
 15 REM ** CARGADOR EN BASIC COMPRESION **
 20 REM ** Y EXPANSION COMMODORE 64      **
 25 REM ***********************************
 30 FOR I=49163 TO 49807:READ A:POKEI,A
 35 CC=CC+A:NEXT I
 40 READ CS:IF CS<> CC THEN PRINT "ERROR":STOP
100 DATA173,2,192,133,139,173,3,192
110 DATA133,140,173,0,192,133,141,173
120 DATA1,192,133,142,169,1,141,7,192
130 DATA160,0,177,139,141,6,192,169
140 DATA255,141,5,192,200,177,139,32
150 DATA39,193,208,54,32,129,192,240
160 DATA55,32,242,192,240,10,32,2,193
170 DATA72,169,0,32,65,193,104,32,65
180 DATA193,204,6,192,144,220,169,0
190 DATA141,4,192,169,2,32,65,193,173
200 DATA5,192,240,3,206,7,192,160,0
210 DATA173,7,192,145,141,96,169,255
220 DATA141,4,192,96,72,169,1,32,65
230 DATA193,104,32,65,193,76,79,192,72
240 DATA152,72,140,10,192,165,139,24
250 DATA109,10,192,133,251,165,140,105
260 DATA0,133,252,169,83,133,254,169
270 DATA194,133,255,169,0,141,8,192
280 DATA160,0,177,254,72,170,240,65
290 DATA165,254,24,105,1,133,254,165
300 DATA255,105,0,133,255,177,251,209
310 DATA254,208,22,200,202,208,246,104
320 DATA104,136,140,10,192,24,109,10
330 DATA192,168,104,173,8,192,162,0,96
340 DATA104,141,10,192,165,254,24,109
350 DATA10,192,133,254,165,255,105,0
360 DATA133,255,238,8,192,76,162,192
370 DATA104,104,168,104,162,255,96,72
380 DATA140,10,192,169,51,133,251,169
390 DATA194,133,252,104,76,15,193,72
400 DATA140,10,192,169,67,133,251,169
410 DATA194,133,252,104,160,0,209,251
420 DATA240,11,200,192,16,208,247,172
430 DATA10,192,162,255,96,152,172,10
440 DATA192,162,0,96,201,32,240,16,201
450 DATA44,240,12,201,46,240,8,201,65
460 DATA144,7,201,91,176,3,162,0,96
470 DATA162,255,96,72,140,10,192,172,7
480 DATA192,173,5,192,208,17,104,17
490 DATA141,145,141,238,7,192,172,10
500 DATA192,169,255,141,5,192,96,104
510 DATA10,10,10,10,145,141,172,10,192
520 DATA169,0,141,5,192,96,173,2,192
530 DATA133,139,173,3,192,133,140,173
540 DATA0,192,133,141,173,1,192,133
550 DATA142,169,255,141,5,192,169,1
560 DATA141,6,192,141,7,192,32,13,194
570 DATA240,28,201,2,240,106,144,37
580 DATA168,169,51,133,251,169,194,133
590 DATA252,177,251,172,7,192,145,141
600 DATA238,7,192,76,143,193,32,13,194
610 DATA168,169,67,133,251,169,194,133
620 DATA252,76,163,193,32,13,194,141,9
630 DATA192,169,83,133,251,169,194,133
640 DATA252,162,0,160,0,177,251,236,9
650 DATA192,240,16,24,105,1,101,251
660 DATA133,251,165,252,105,0,133,252
670 DATA232,208,233,170,160,1,177,251
680 DATA140,10,192,172,7,192,145,141
690 DATA238,7,192,172,10,192,200,202
700 DATA208,236,76,143,193,173,7,192
710 DATA56,233,1,160,0,145,141,0,173,5
720 DATA192,208,16,169,255,141,5,192
730 DATA172,6,192,177,139,41,15,238,6
740 DATA192,96,169,0,141,5,192,172,6
750 DATA192,177,139,74,74,74,74,41,15
760 DATA96,0,0,0,70,76,68,72,83,73,82
770 DATA78,79,65,84,69,32,67,77,85,71
780 DATA89,80,87,66,86,75,88,74,81,90
790 DATA44,46,3,84,72,69,4,84,72,73,83
800 DATA4,84,72,65,84,2,73,70,3,89,79
810 DATA85,2,77,69,3,87,65,83,2,72,69
820 DATA3,83,72,69,4,84,72,69,89,2,79
830 DATA70,2,73,84,2,73,83,3,70,79,82
840 DATA2,79,78,2,84,79,0,7
850 DATA76822:REM*SUMA DE CONTROL*
```

Una entrada

Por último, centramos nuestra atención en las facilidades a mayor escala que ofrece este sistema operativo

La tecnología del correo electrónico ya ha sido desarrollada desde hace un tiempo, si bien su implantación generalizada se ha visto condicionada en cierta medida por su relativa dificultad de aplicación, poca velocidad y falta de fiabilidad. El sistema de correo electrónico del Unix, el primer sistema operativo que lo incluyó como estándar, permite la comunicación entre usuarios de la misma máquina, y se puede ampliar para cubrir las comunicaciones entre dos ordenadores cualesquiera basadas en Unix, a través de un enlace adecuado. Veremos cómo funciona el sistema entre usuarios de la misma máquina; puesto que la operatoria es idéntica para usuarios remotos, resultará una introducción idónea para ambos casos.

El sistema se activa mediante la instrucción mail, que abre su archivo de correspondencia personal y proporciona acceso a toda la correspondencia que le haya sido enviada. Todos los mensajes entrantes se identifican por un número y el "ID" del remitente. Si se enviaran copias a otros usuarios, también se indicaría este hecho. Los mensajes nuevos y sin leer llevan un indicador (cada uno normalmente va acompañado por un título de una línea para identificar el tema de que se trate) y se pueden leer ya sea individualmente o bien por el orden en que llegaron con sólo pulsar la tecla Return.

El sistema se identifica mediante el aviso &, que posee una variedad de comandos relativos a la manipulación del correo y las funciones del sistema, como cambiar el directorio, y usted puede salir con el comando x o q. El primero deja en el buzón toda la correspondencia que no se haya eliminado, de modo que volverá a aparecer la próxima vez que se active el sistema; el segundo extraerá los mensajes que el usuario ya ha leído y los colocará en su directorio bajo un archivo especial llamado mbox. En este punto están disponibles para la edición y las

Diálogo con la base de datos

```
Berkeley 4.2 Vax/Unix (infc3)
Type (Ctrl-D) to disconnect

login: com–mcc
Password:
You are a normal user (class 3)
Jobs: 6 Superiors: 2 Maximum: 21
Last login: Thu Nov 14 13:47:42 on ttyn04

You have mail.                    {se le informa si tiene corresp. aguardando}

%mail
Mail version 2.18 5/19/83. Type ? for help.
"/usr/spool/mail/com–mcc": 1 message 1 new
>N 1 com–vjp Fri Nov 15 09:52 11/271 "Specimen mailing"
&                                 {pulse sólo RETURN para obtener la nueva
                                   corresp.}
Message 1:
From com–vjp Fri Nov 15 09:52:35 1985
From: com–vjp (Vicki)
To: com–daf, com–mcc
Subject: Specimen mailing         {título del mensaje}
Hello                             {contenido del mensaje}

&?                                {da una lista de comandos}
    Mail          Commands
t   <message list>          types messages
n                           goto and type next message
e   <message list>          edit messages
f   <message list>          give head lines of messages
d   <message list>          delete messages
s   <message list> file     appends messages to files
u   <message list>          undelete messages
r   <message list>          reply to messages
pre <message list>          make messages go back to/usr/mail
m   <user list>             mail to specific users
q                           quit, saving unresolved messages in mbox
x                           quit, do not remove system mailbox
h                           print out active message headers
!                           shell escape
c   [directory]             chdir to directory or home if none given
A   <message list>          consists of integers, ranges of same, or user names
                            separated by spaces. If omitted, Mail uses the last
                            message typed.
A   <user list>             consists of user names or distribution names separated
                            by spaces.
Distribution names are defined in .sendrc in your home directory.

% ingres demo                    {ahora ejecute la base de datos "ingres" utilizando datos
                                  de demo}
INGRES version 7.10 (10/27/81) login
Fri Nov 15 10:28:42 1985
go
* print parts                    {comandos entrados en el espacio de trabajo y activadas
                                  mediante \g}
*\g
Executing...

parts relation
|pnum |pname          |colour |weight |qoh |
|-----|---------------|-------|-------|----|
|  10 |byte-soap      |clear  |    0  | 143|
|   1 |central processor|pink |   10  |   1|
|  11 |card reader    |grey   |  327  |   0|
|   2 |memory         |grey   |   20  |  32|
|  12 |card punch     |grey   |  427  |   0|
|   3 |disk drive     |black  |  685  |   2|
|  13 |paper tape reader|black|  107  |   0|
|   4 |tape drive     |black  |  450  |   4|
|  14 |paper tape punch|black |  147  |   0|
|   5 |tapes          |grey   |    1  | 250|
|   6 |line printer   |yellow |  578  |   3|
continue
*\p                               {muestra el contenido actual del espacio de trabajo}
print parts
```

otras funciones normalmente relacionadas con el tratamiento de archivos de texto, pero el sistema de correspondencia no puede acceder directamente a ellas.

Usted puede enviar correspondencia desde el sistema mediante el comando m, o desde fuera del mismo impartiendo el comando mail seguido del nombre (o nombres) del destinatario. Se utilizan los ID de *login* normales y la lista de nombres puede ser tan larga como desee el usuario; cada uno recibirá una copia de su mensaje.

Cuando la correspondencia se distribuye regularmente entre varios usuarios, usted puede ahorrarse el tener que escribir una lista de nombres cada vez merced a la facilidad alias. Para estos alias se utiliza un archivo especial, así como para establecer otros parámetros de correspondencia, como pedir que se le informe cuándo ha llegado algún mensaje (utilizando ya sea .mailrc o bien .sendrc).

La base de datos "ingres"

Con la mayoría de los sistemas Unix se suministra, o al menos está disponible, la base de datos *ingres*. Si bien su interface para el usuario es menos fácil de aplicar que la mayoría, es una base de datos relacional muy potente, muchísimo más, de hecho, que la mayoría de los sistemas más conocidos, como el dBase II. Pero así como la mayoría de los aspectos del Unix, se la puede adaptar a la medida para satisfacer sus propias necesidades si así fuera necesario, un ejemplo de lo cual podemos apreciar abajo.

Ningún sistema está completo sin alguna forma de editor de textos: el Unix proporciona al menos tres como estándar y opcionalmente se pueden añadir otros. El más básico es ed, un editor de línea similar tanto al ed del CP/M como al edline del MS-DOS. Al igual que todos los editores de línea, es más bien difícil de utilizar, pero posee la ventaja de tener todas las funciones de edición como comandos. Por consiguiente, se puede preparar un archivo que contenga una secuencia de comandos, permitiendo que el Unix lleve a cabo automáticamente la edición de un documento. Los otros dos editores son ded y vi, que se pueden utilizar como editores tanto de pantalla como de línea.

Por último, por razón de espacio nos hemos visto limitados a cubrir sólo una fracción de las facilidades del Unix, pero las que se han omitido están relacionadas mayormente con la programación y el desarrollo de programas, que interesan más a los usuarios avanzados que a los recién iniciados. El Unix es, indudablemente, uno de los sistemas operativos más importantes desde el punto de vista del desarrollo de la informática moderna, y aunque quizá jamás consiga un éxito comercial aplicado en los micros, su influencia se puede apreciar con toda claridad en casi todos los nuevos sistemas.

```
continue
*\r                          {limpia el espacio de trabajo}
go
* range of p is parts        {especifica una parte determinada de la
                              base de datos a utilizar, a la que se alude
                              como p}
* retrieve (p.pname)         {buscar y visualizar el campo dado}
*\g
Executing

|pname
|----------------------------|
|byte-soap                   |
|central processor           |
|card reader                 |
|memory                      |
|card punch                  |
|disk drive                  |
|paper tape reader           |
|tape drive                  |
|paper tape punch            |
|tapes                       |
|line printer                |
|----------------------------|
(11 tuples)

continue
* retrieve (p.pname,p.coclour)  {más de un campo}
* where p.colour = "grey"       {especificar criterio de búsqueda}
*\g
Executing...

2100: line 1, Attribute 'coclour' not in relation parts
{este mensaje de error no es muy útil: hemos escrito mal "color"}
continue
* retrieve (p.pname,p.colour)
* where p.colour = "grey"
*\g
Executing...

|pname                |colour  |
|---------------------|--------|
|card reader          |grey    |
|memory               |grey    |
|card punch           |grey    |
|tapes                |grey    |
|---------------------|--------|
(4 tuples)

continue
* retrieve (p.pname,p.pnum,total=p.qoh * p.weight)
{los valores visualizados se pueden calcular a partir de campos y un título dado}
*\g
Executing...

|pname                |pnum  |  total|
|---------------------|------|-------|
|byte-soap            |  10  |    0  |
|central processor    |   1  |   10  |
|card reader          |  11  |    0  |
|memory               |   2  |  640  |
|card punch           |  12  |    0  |
|disk drive           |   3  | 1370  |
|paper tape reader    |  13  |    0  |
|tape drive           |   4  | 1800  |
|paper tape punch     |  14  |    0  |
|tapes                |   5  |  250  |
|line printer         |   6  | 1734  |
|---------------------|------|-------|
(11 tuples)

continue
*\q                          {salir de la "ingres"}
INGRES version 7.10 (10/27/81) logout
Fri Nov 15 10:41:03 1985
goodbye com-mcc — come again
% logout
```

Suave transición

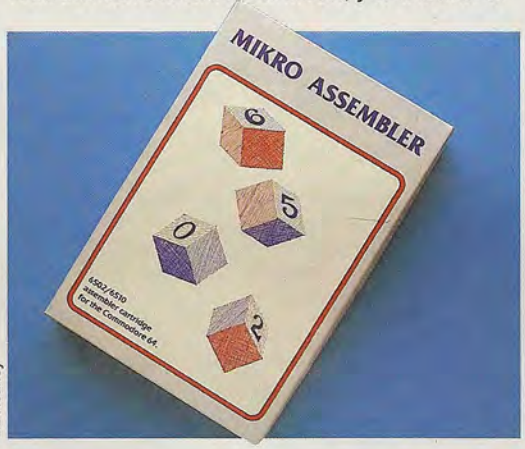
Finalmente examinaremos el Mikro de Supersoft, un ensamblador/monitor basado en cartucho para el Commodore 64

Entre los problemas que usted deberá afrontar cuando escriba por primera vez programas en código máquina se incluyen la falta de "protección" de la máquina inherente a programación en lenguaje máquina y el nada familiar método de preparar programas para su ejecución. Lo primero es el precio que el usuario ha de pagar por la potencia que le confiere la programación en lenguaje máquina.

El cartucho ensamblador Mikro de Supersoft permite entrar texto de programas fuente utilizando un editor de pantalla completa y números de línea, como los que emplea el BASIC del Commodore 64. Puesto que la mayoría de las personas que aprendan lenguaje máquina en su micro habrán primero aprendido BASIC, este método de entrada les resultará familiar y les ayudará a suavizar la transición de la programación en BASIC al trabajo en lenguaje máquina.

Supersoft Mikro

Mikro, de Supersoft, es desde hace tiempo el estándar para el Commodore 64. Cuenta con la ventaja de estar alojado en un cartucho enchufable, proporcionando un ensamblador y monitor completos integrados con el BASIC Commodore. Los programas fuente se entran como líneas normales de un programa en BASIC; el usuario no puede ejecutar el programa (RUN), pero sí utilizar las instrucciones LOAD y SAVE normales y el editor en pantalla de BASIC para almacenar y corregir su código fuente. Aunque ésta es una solución discutible, el Mikro añade una instrucción de números de línea AUTOmáticos, una instrucción DELETE para suprimir líneas y una serie FIND para facilitar la edición. Falta la instrucción esencial RENUMBER, y si usted ha de



Liz Heaney

renumerar su programa habrá de cargar una utilidad separada. Durante el ensamblaje, el Mikro puede producir un listado assembly completo en una impresora, pero no en pantalla, y también una lista clasificada de los símbolos utilizados en los programas. Las directivas de ensamblador son adecuadas, y la más útil es LNK, que permite escribir un programa en secciones y luego ensamblarlas desde cassette o disco como un programa. Ello facilita el desarrollo de programas de hasta unos 12 Kbytes de código objeto.

El Mikro posee un monitor de lenguaje máquina similar a muchos existentes para el C64 y a aquellos incorporados en la gama Commodore PET. Esto es práctico, puesto que no incluye puntos de parada ni seguimiento. Se entrega, además, un breve manual de 16 páginas, que, aunque describe los comandos, no es todo lo detallado que debería ser. En resumidas cuentas, el Mikro es adecuado y fiable, pero puede resultar incómodo para desarrollar programas largos.

Tipo de ensamblador

Dos pasos, mnemotécnicos estándar

Límites

Código fuente: 30 K, pero puede enlazar archivos
Tamaño de la tabla de símbolos: 11 K

Directivas de ensamblador

*	Se utiliza en lugar de ORG y DEFS. Alude a la posición actual, y una instrucción como *=$6000 indicará al ensamblador que empiece el progra[illegible] la posición $6000. Una instrucción como =* +256 reservaría un espacio de 256 bytes para datos en el programa
=	Asignar un valor a un símbolo
WOR	Almacenar valor(es) de 16 bits
BYT	Almacenar valor(es) de 8 bits
TXT	Almacenar serie ASCII
LNK	Continuar ensamblando desde el arch. fuente mencionado
OUT	Sacar listado assembly por impresora
OFF	Desactivar listado

Opciones de ensamblaje

Ninguna

Aritmética

+, =, < y > (bytes *high* y *low* de valores de 16 bits).
Decimal, hexa y binario

En los años noventa

Trasladémonos a principios de la próxima década e imaginemos un ordenador que reúna las características que se intuye incluirán las máquinas de ese cercano futuro

Los primeros años de la década de los noventa han sido testigos de cómo numerosas características que antes sólo estaban disponibles en micros de gestión se incorporaban en los micros personales. Máquinas de interface amables con el usuario acopladas con una creciente cantidad de facilidades a disposición del aficionado al ordenador personal han hecho que el ordenador moderno sea fácil de usar y le han conferido facilidades en la propia placa para una amplia gama de aplicaciones.

El Chestnut ("castaña"), de la firma Conquer, es la última producción de una línea de "puestos de trabajo personales" fabricada para el inmenso mercado del micro doméstico. Ahora que, al fin, se está comprendiendo el verdadero potencial del ordenador personal (recientes encuestas han indicado que la base de usuarios es ahora equivalente a la de los sistemas de alta fidelidad), el mercado está siendo "activado por la demanda" en vez de por la "oferta", es decir, los fabricantes están atendiendo las necesidades de los usuarios en lugar de producir artilugios de alta tecnología con los que confían en hallar un mercado.

A alrededor de £500 (o sea, poco más de 100 000 ptas), el Chestnut incluye las facilidades más populares de los últimos cinco años. Basado en el conocido procesador 68000, el paquete incluye un ordenador con pantalla en color integral, una única unidad de disco de 3½ pulgadas, doble densidad y doble cara, un minidisco rígido de 10 Mbytes y un megabyte de RAM. Midiendo apenas 300 × 240 × 120 mm, la carcasa del ordenador se hace eco de la reciente tendencia a crear máquinas planas cada vez más pequeñas y más en armonía con la superficie del escritorio.

El teclado y el mando de bola/ratón están conectados al ordenador a través de cables de fibra óptica. El teclado responde al formato QWERTY estándar con 10 teclas de función y un teclado numérico que también actúa como grupo de cursor. El teclado de máquina de escribir contiene varias teclas de función, como Ctrl y Alt, que permiten programar en la máquina funciones adicionales.

Las dimensiones del Chestnut le deben mucho a la tecnología de "pantalla plana" que se ha perfeccionado de manera notable en el curso de los últimos años. La pantalla no sólo mide la mitad que las pantallas de tubos de rayos catódicos de los años setenta y ochenta, sino que en los bordes ya no se produce distorsión de imagen.

Puerta en paralelo

La CD-ROM se considera como un periférico esencial. Al objeto de ser competitivo, el Chestnut se proporciona con una adecuada puerta en paralelo. Esta primera unidad de CD-ROM de Hitachi retenía 552 Mbytes de datos y su velocidad de transferencia de información era de 176 Kbytes por segundo

Liz Heaney

Debajo de la pantalla se halla la unidad de disco, que proporciona un disco individual de doble cara con capacidad de un megabyte. Opcionalmente se puede instalar una segunda unidad en un espacio ahora tapado por el logotipo del Chestnut, justo debajo de la primera unidad (Conquer pretende comercializar un modelo de unidades gemelas para aplicaciones de gestión).

Sobre el lado derecho de la carcasa principal del ordenador está la puerta controladora del ratón. Asimismo, hay un bus de ampliación en donde se pueden instalar segundos procesadores, procesadores de fotogramas y otros módulos de ampliación de memoria de un megabyte.

El panel posterior de la máquina alberga las puertas estándares para periféricos y comunicaciones. En el extremo derecho hay un par de conectores de *hi-fi* en los que se puede enchufar un sistema estéreo. Ello permite sacar el máximo partido del soberbio sonido del Chestnut, proporcionado por el chip sintetizador Y/S2416 que posee ocho osciladores en la placa y una gama de cinco octavas. De

este modo se puede producir un sonido polifónico con un teclado de piano electrónico.

La puerta para el usuario, que da cabida a teclados musicales o dispositivos robóticos, se halla a continuación, seguida por los conectores en serie MIDI. Aunque actualmente la mayoría de los músicos están utilizando el formato (en paralelo) MIDI-P, Conquer ha decidido incluir en el Chestnut las puertas en serie MIDI antiguas, para sacar partido de la gran cantidad de paquetes LAN producidos para el estándar MIDI-S.

Por supuesto, ya no hay necesidad de programar el chip sintetizador directamente. Puesto que ahora el software MIDI se proporciona en placa, la programación del chip es simplemente cuestión de configurar el sintetizador como un dispositivo MIDI y enviarle los códigos adecuados.

Al igual que la mayoría de los micros modernos, todas las conexiones en serie para la MIDI y el modem en placa se hallan en un único chip. El acceso a la red telefónica lo proporciona un par de enchufes telefónicos estándar, que permiten al usuario acceder a toda la gama de comunicaciones disponibles. Éstas incluyen Micronet, Final Frontier (el juego recreativo para múltiples usuarios) y Telemarket, un servicio en virtud del cual se pueden adquirir programas de juegos y aplicaciones a través de la red telefónica y cargarlos en el disco rígido del usuario.

La nueva generación de micros personales incorpora una facilidad que, comprensiblemente, ha entusiasmado al público: un fiable reconocimiento y síntesis de voz. Esta característica no ha derivado de innovación tecnológica alguna, sino de la drástica caída del precio de la memoria en los últimos años y de la creciente disponibilidad de procesadores de elevado rendimiento. Ello ha vuelto disponible memoria suficiente para almacenar las plantillas de voz que se requieren para el proceso.

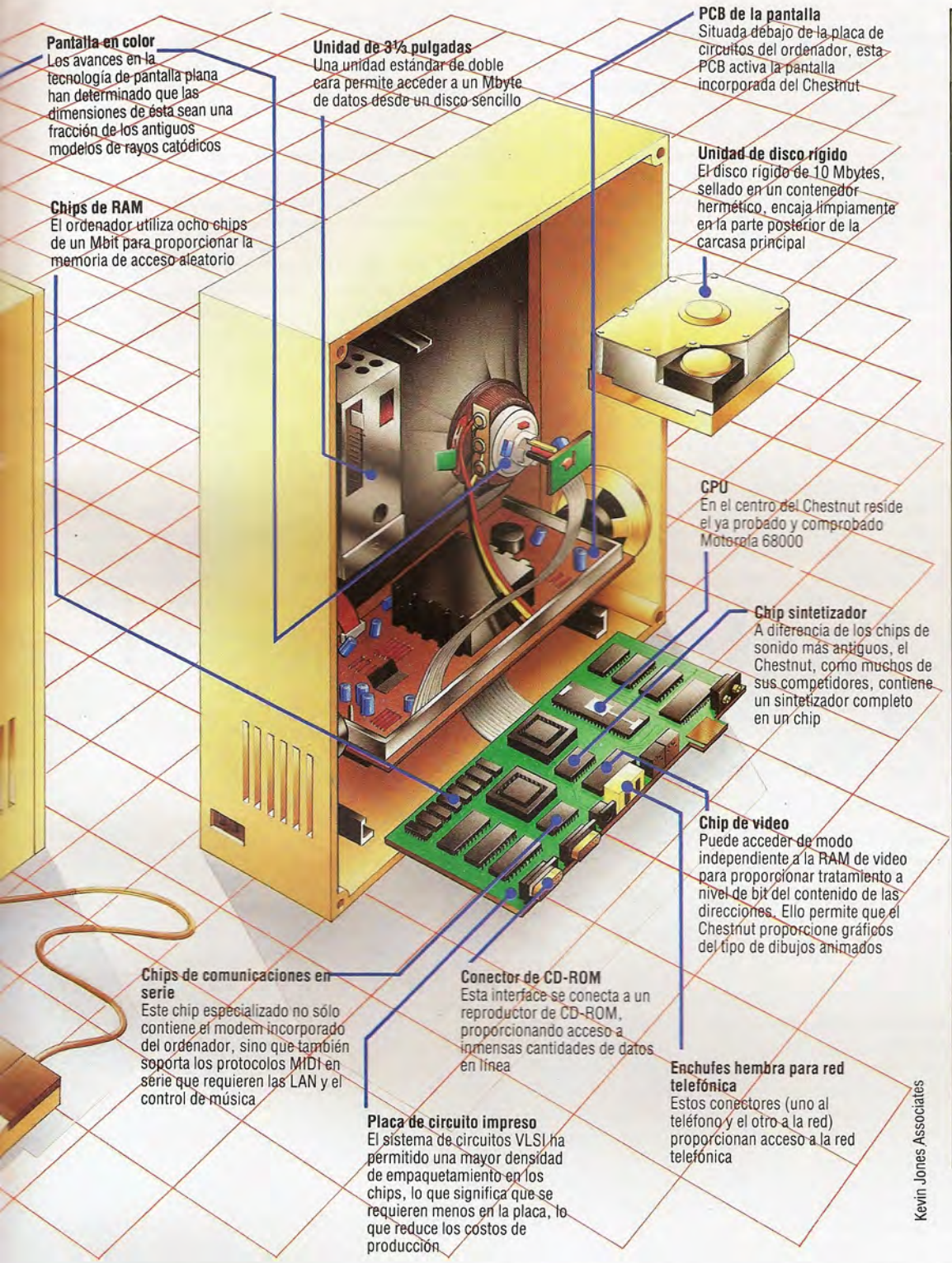
Un micrófono enchufado en el conector hembra D/A de la parte posterior del Chestnut le transmitirá instrucciones habladas al ordenador, el cual buscará las plantillas para hallar una pareja y responder, una vez encontrada ésta. Mediante el uso de estas mismas plantillas y el chip sintetizador, el Chestnut alcanza un elevado nivel de síntesis de voz y una limitada capacidad para conversar.

Junto a la interface D/A se halla la puerta para impresora Centronics habitual, y sobre el extremo izquierdo hay una interface en paralelo para una unidad de CD-ROM. Conquer cree que éste es uno de los más importantes aciertos de la máquina, dado que los usuarios podrán acceder a la gran cantidad de material de bases de datos, tales como la *Enciclopedia británica*, que recientemente ha aparecido en CD-ROM. Los ávidos aventureros podrán jugar a la popular *Odyssey*, de 500 Mbytes, que saca máximo partido de esta tecnología: el acceso rápido al almacenamiento masivo ha permitido a los programadores crear una aventura completamente activada por la voz y con detallados gráficos animados.

La pantalla de gran resolución que acompaña al Chestnut puede visualizar más de mil colores a la vez. El chip *blitter* de video proporciona una visualización uniforme y realista en modalidad Animation.

El WIMPOS-3, sistema operativo activado por menú, es similar a anteriores versiones desarrolladas por Conquer. La única diferencia parece residir en su mayor alcance para la manipulación de ventanas e iconos a través de instrucciones habladas, si bien el modelo que probamos en ocasiones no resultó fiable, especialmente cuando se le hablaba pronunciando a una velocidad más cercana a la del habla normal. Al menos por ahora, parecería que el uso del ratón, aunque anticuado, es preferible a correr el riesgo de perder archivos a raíz de una mala interpretación por parte de la máquina de las instrucciones vocalizadas.

El software empaquetado en el disco rígido del



Conquer Chestnut

DIMENSIONES
300 × 240 × 120 mm

CPU
68000 operando a 8 MHz

MEMORIA
1 Mbyte de RAM; 128 Kbytes de ROM

PANTALLA
Resolución de textos de 80 × 32; resolución de gráficos de 640 × 256 pixels, permitiendo visualizar simultáneamente hasta mil colores

INTERFACES
Bus de ampliación, conector para ratón/mando de bola, conectores *hi-fi* estéreo, puerta para el usuario, puertas MIDI gemelas, conectores red telefónica gemelos, conector A/D, puerta CD-ROM en paralelo

LENGUAJES DISPONIBLES
BASIC, compilador de C

UNIDAD DE DISCO
Individual, de 3½ pulgadas, doble cara y doble densidad, de un Mbyte, con una segunda unidad opcional

SISTEMA OPERATIVO
WIMPOS-3

DOCUMENTACION
Es completa y ofrece información pormenorizada acerca del sistema. Como detalle original, Conquer ofrece los manuales en CD-ROM, así como en la versión impresa normal

PUNTOS FUERTES
El Chestnut ofrece numerosas facilidades que hasta muy recientemente sólo estaban disponibles en máquinas más caras

PUNTOS DEBILES
La máquina no ofrece nada radicalmente nuevo y quizá experimente problemas para hacer oír su voz entre los productos de "tecnología punta" existentes en el mercado

ordenador incluye el programa de tratamiento de textos *ChestWrite*, el completísimo *ChestPaint*, el *ChestTalk*, para desarrollar plantillas de voz definidas por el propio usuario, así como un sistema experto, ChestComplaint, que permite que el mismo usuario diagnostique su estado de salud. Asimismo, Conquer suministra con la máquina el sistema WIMPOS, con el software de modem y MIDI. Los lenguajes de programación empaquetados con el Chestnut son BASIC y un compilador de C.

A pesar de las reiteradas afirmaciones de Conquer durante los seis meses que precedieron al lanzamiento de la máquina, el Chestnut ofrece pocas novedades. Su principal atractivo comercial probablemente residirá en su bajo precio y sus facilidades empaquetadas, las cuales, en las anteriores máquinas de la empresa, se vendían como extras opcionales. Pero a la vista de que los críticos informáticos han alcanzado tal talla que se han puesto a la altura de sus homólogos en el mundo del cine, el teatro y la literatura, resulta demasiado sencillo juzgar con severidad las nuevas máquinas de hoy en día, que hace apenas siete u ocho años habrían representado la cima de los logros de la informática.

Factores decisivos

Al elegir un lenguaje de programación es preciso armonizar las necesidades del usuario con las facilidades de aquél

Hasta hace poco tiempo, los usuarios de micros pequeños sólo disponían de BASIC o de lenguaje assembly. Ahora incluso las máquinas más pequeñas disponen de la mayoría de los lenguajes comunes, y usted sólo tiene que pasar apenas a una máquina compatible con el IBM PC o a cualquier otra máquina MS-DOS para disponer virtualmente de una gama completa. Dado que los compiladores e intérpretes pueden ser bastante caros y que, por lo tanto, pocas personas desearán tener más de tres o cuatro (si acaso), el problema de la elección subsiste a pesar de la mayor disponibilidad.

La eficacia, desde el punto de vista de la memoria utilizada y los tiempos de ejecución, puede convertirse en un factor importante, particularmente con programas que estén interactuando con el entorno externo. En BASIC, por ejemplo, algunos programas llevarían tanto tiempo de escribir y depurar, que sería más rápido aprender un lenguaje más adecuado y escribirlo en el mismo. Un programa para interrogar y actualizar un archivo de existencias de artículos en línea, por ejemplo, sería una tarea sencilla en COBOL empleando archivos indexados. Escrito en BASIC, habría de utilizar archivos se-

Lo bueno y lo malo

Lenguaje: BASIC
Puntos positivos: Fácil de aprender y utilizar, ampliamente disponible. Barato. Buena provisión de funciones aritméticas. Buen tratamiento de series
Puntos negativos: La mayoría de las versiones no poseen una buena facilidad de módulos ni estructuras de control. No estandarizado. La ejecución a menudo es ineficaz y lenta. Tratamiento de archivos pobre. Gama restringida de tipos de datos y estructuras. Fácil de escribir mal, código no claro
Uso: Programas cortos, de usar y tirar si se trata de problemas que impliquen aritmética o tratamiento de series

Lenguaje: LOGO
Puntos positivos: Sólida base matemática, fácil de aprender a un nivel básico. Gráficos de tortuga. Buenas facilidades para proceso de listas, buena provisión de módulos, tipos de datos y estructuras. Disponible amplia y económicamente
Puntos negativos: La programación más allá de la etapa inicial puede volverse complicada y críptica. Muchas versiones incompatibles. Ejecución ineficaz a veces
Uso: Gráficos, proceso de listas, aprendizaje sobre matemáticas y conceptos de programación avanzada

Lenguaje: PASCAL
Puntos positivos: Bien estructurado. Buena gama de tipos de datos. Relativamente estandarizado. Es fácil producir código de gran calidad
Puntos negativos: Entrada/salida no definidas con claridad, el tratamiento de archivos es deficiente
Uso: Aprendizaje de buenas técnicas de programación. Programación general semimatemática a pequeña escala

cuenciales o un algoritmo de *hashing* y archivos de acceso directo que lo volverían muchísimo más complicado de lo que sería necesario.

Las características de diseño que hacen que un lenguaje sea fácil de aprender y utilizar también tienden a crear problemas cuando aumenta el tamaño del programa, circunstancia en la cual uno de los requisitos fundamentales debe ser la facilidad para descomponer el programa en módulos moderadamente autocontenidos que se puedan programar independientemente.

Algunos lenguajes, como el COBOL y el FORTRAN, están definidos rígidamente por comités internacionales, lo que convierte en un lento procedimiento la introducción de cualquier cambio para reflejar el hardware y los nuevos tipos de proceso. Sin embargo, un programa escrito en una máquina sólo requerirá una recompilación, o, en el mejor de los casos, unos pocos cambios menores, para ejecutarse en una máquina diferente.

Los lenguajes como el PASCAL y el C poseen un estándar *de facto* definido por su(s) inventor(es). La mayoría de las versiones de estos lenguajes son bastante coherentes con el estándar, pero, especialmente en lenguajes como el PASCAL, en el cual la entrada/salida no está definida claramente, cada implementador tiene la libertad de introducir modificaciones y adiciones, aunque los programas que utilicen estas facilidades no serán muy portables. Por último, tenemos lenguajes como el BASIC, para el cual no existe estándar ninguno.

Otro factor es el tipo de programa traductor utilizado. Los intérpretes tienden a ser fáciles de emplear y mejores para el desarrollo de programas, pero más lentos y menos eficaces en ejecución. Los compiladores son más complicados de utilizar, pero producen un producto final más eficaz. Esto ahora se está reduciendo por la introducción de compiladores incrementales y depuradores animados.

Algunos lenguajes han suscitado desarrollos de hardware para permitir una ejecución más eficaz. Por ejemplo, se ha dedicado muchísimo esfuerzo en producir un procesador qué ejecute FORTH directamente. Del mismo modo, existe un procesador que ejecuta P-code PASCAL. En el futuro cercano quizá sea posible producir un compilador basado en hardware, que simplificará mucho el proceso de utilizar un lenguaje compilado.

No obstante, el factor primordial continúa siendo el tipo de aplicación para la cual se esté utilizando el lenguaje. Veamos algunos lenguajes y comparemos sus puntos fuertes y sus puntos débiles. Luego podemos ver algunas de las aplicaciones típicas y considerar qué lenguaje sería el más adecuado.

Lenguaje: FORTH
Puntos positivos: Ejecución eficaz, ampliable por el usuario. Buenas estructuras
Puntos negativos: Nivel demasiado bajo para gran parte de trabajo serio. Difícil de comprender. Muchas versiones diferentes
Uso: Trabajo de bajo nivel antes que el ensamblador, especialmente hardware controlador

Lenguaje: FORTRAN
Puntos positivos: Muy estandarizado. Grandes bibliotecas de software disponibles. Excepcional gama de funciones matemáticas y científicas
Puntos negativos: Anticuado. Estructuras pobres. Es fácil escribir código críptico. Entrada/salida diseñada en base a tarjetas perforadas. Trazado de programas estricto
Uso: Programación general científica, de ingeniería y matemáticas

Lenguaje: COBOL
Puntos positivos: Sumamente estandarizado, disponible en una gama de máquinas muy amplia. Fácil de aprender. Excelente tratamiento de archivos
Puntos negativos: Verborreico, difícil de aprender Los compiladores son grandes y caros. Mal estructurado, con una pobre gama de tipos de datos
Uso: Proceso de datos en gestión

Lenguaje: C
Puntos positivos: Buena gama de tipos de datos y estructuras. Buena estructura modular. Proporciona acceso al hardware. Produce código veloz y eficiente
Puntos negativos: Resulta fácil producir código críptico. Nivel demasiado bajo para muchos trabajos serios. Estandarizado inadecuadamente, basándose demasiado en las bibliotecas
Uso: Software de sistemas como recambio para el ensamblador. Allí donde se requiera máxima eficacia

Lenguaje: LISP
Puntos positivos: Proceso de listas matemáticamente sólido. Mucho software y apoyo disponibles. Muy utilizado en las máquinas grandes
Puntos negativos: Difícil de aprender, comprender y utilizar
Uso: Inteligencia artificial y proceso de listas

Lenguaje: PROLOG
Puntos positivos: Sólida base matemática. Fácil y entretenido de utilizar. Supuestamente más próximo al razonamiento humano
Puntos negativos: No es un lenguaje relacional "puro": posee algunos aspectos procedurales y necesita algunas facilidades extrañas, como el corte, para operar eficientemente
Uso: Inteligencia artificial, aplicaciones de bases de datos

Lenguaje: Ensamblador
Puntos positivos: Control absoluto de todos los aspectos. Potencialmente el más eficiente
Puntos negativos: Completamente no estándar. Difícil de aprender y utilizar
Uso: Máxima eficiencia, pero sólo si fallan los demás

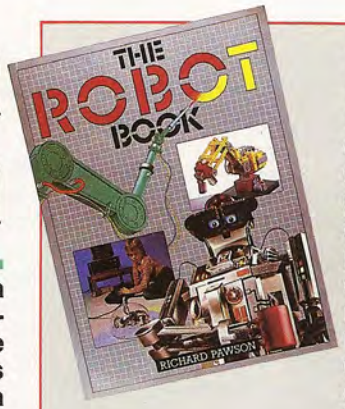
Lingüística aplicada

La tabla muestra una lista de aplicaciones, y los lenguajes más adecuados, por orden de preferencia:

Aplicación	Lenguajes
Análisis estadístico	FORTRAN, BASIC, PASCAL
Control de stocks	COBOL, PASCAL, BASIC
Contro de brazo-robot	FORTH, C, ensamblador
Juego de aventuras	PROLOG, C, BASIC
Sistema experto	LISP, PROLOG, LOGO
Educación	LOGO, PROLOG, PASCAL

Copia impresa

En los últimos años la literatura informática ha constituido un auténtico "boom" en la industria editorial. En estos momentos es posible hallar libros que versen sobre las áreas más desconocidas de la informática. He aquí una selección de títulos que informa de la variada gama de temas que abarca esta disciplina. Algunas obras abordan en profundidad materias que hemos cubierto a lo largo de este curso



El libro de los robots

The robot book, libro con una presentación de gran colorido, cubre los diversos aspectos del control por microordenador y la robótica. La historia de la robótica, las aplicaciones industrial, educativa y doméstica de los robots, su funcionamiento y proyectos de construcción prácticos constituyen las cuatro partes principales del libro. Las tres primeras proporcionan un tratamiento teórico del tema, mientras que la última sección incorpora este conocimiento en planes para que el usuario pueda construir sus propias máquinas. Se utilizan piezas de Lego y Fischertechnik y se dan instrucciones completas para conectar los proyectos terminados en interface con su micro. La última sección amplía algunos de los proyectos emprendidos en nuestro apartado de *Bricolaje*, detallando la construcción, entre otras cosas, de robots para jugar a las damas y repartir la baraja, dispositivos que caminan y brazos estáticos. Todos están bien presentados, con diagramas fáciles de seguir y explicaciones completas.

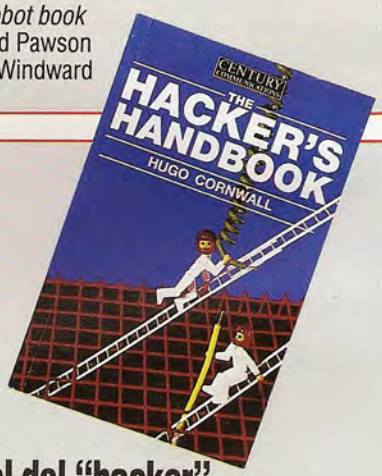
Título: *The robot book*
Autor: Richard Pawson
Editado por: Windward

Guía de la inteligencia artificial para el autostopista

The hitch-hiker's guide to artificial intelligence es una introducción amable e informativa a la inteligencia artificial y desarrolla muchos de los temas tratados en nuestra serie dedicada a ella. Solución de problemas, lenguaje natural y creatividad por ordenador son algunos de los temas que se tratan en los diez capítulos, profusamente ilustrados con didácticos programas en BASIC. Éste es uno de los aciertos del libro, dado que usted no necesitará poseer conocimiento alguno sobre los lenguajes específicos de la AI.

Esta "Guía de la inteligencia artificial para el autoestopista" se encuentra disponible para usuarios del BBC Micro, Apple II, IBM PC y la gama de micros Amstrad.

Título: *The hitch-hiker's guide to artificial intelligence*
Autores: Richard Forsyth y Chris Naylor
Editado por: Chapman and Hall/Methuen



El manual del "hacker"

The hacker's handbook llegó a los titulares de la prensa británica cuando la policía, temiendo la revelación de información confidencial, insistió en leer el libro antes de que fuera publicado. (Recordemos que se denomina *hacker* [asaltante] al usuario que accede ilícitamente a los ordenadores centrales utilizando ordenadores personales o modems.) Pero al *hacker* experimentado le dice poca cosa que ya no sepa, y al "novato" sólo le proporciona indicaciones vagas. El libro ofrece una detallada reseña de los dispositivos de comunicación, cómo están implementados en diversos sistemas y qué equipo se necesita para iniciarse como "asaltante electrónico". Contemplado bajo este prisma, *The hacker's handbook* es una valiosa guía para redes de datos, desde tablones de anuncios hasta las grandes bases de datos públicas, y, en consecuencia, es recomendable para quien se interese por utilizar un modem... aun cuando no sea para introducirse en los ordenadores de la OTAN.

Título: *The hacker's handbook*
Autor: Hugo Cornwall
Editado por: Century

De los chips a los sistemas

From chips to systems reúne en sus páginas todos los aspectos fundamentales del hardware de ordenador, disipando gran parte del misterio que nos impide aventurarnos en el tema. Dando por sentado sólo un conocimiento limitado de informática y electrónica, Rodnay Zaks esboza el funcionamiento de los componentes individuales y detalla cómo se interconectan para conformar sistemas completos. Abordando desde los principios del diseño de microprocesadores hasta la conexión en interface, Zaks analiza luego aplicaciones de microprocesador sencillas, tales como un controlador de motor de cassette, un convertidor analógico/digital y un controlador para horno de microondas.

Título: *From chips to systems*
Autor: Rodnay Zaks
Editado por: SYBEX

La realización del micro

Al igual que *The soul of a new machine*, de Tracey Kidder, *The making of the micro* proporciona una apasionante información sobre el desarrollo de los ordenadores, en esta ocasión desde la perspectiva histórica. La obra de figuras como Babbage, Von Neumann y Pascal se analizan con todo detalle, y algunas de ellas cobran más vida aún en virtud de la narración de sus asuntos personales.
Un punto en el cual este libro hace hincapié implícitamente es cómo son realmente los ordenadores antiguos, y cómo hemos llegado a darlos por sentado sin rendir verdaderos honores a sus pioneros. Pocas personas sabrán, por ejemplo, que Vannevar Bush realizó una mejora trascendental al reemplazar los elementos mecánicos de su "analizador diferencial" por tubos termiónicos. Esta obra de Christopher Evans constituye una interesante, aunque breve, incursión en los antecedentes de la nueva tecnología.

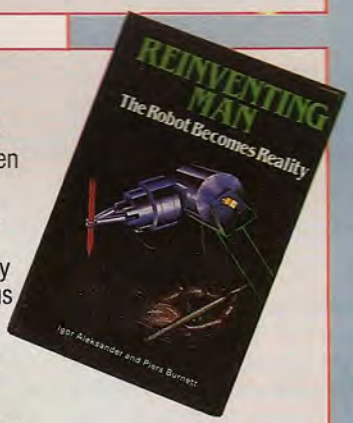
Título: *The making of the micro*
Autor: Christopher Evans
Editado por: Oxford University Press

Gane dinero escribiendo software

Se han escrito innumerables libros dirigidos a aspirantes a programadores que sueñan con escribir un programa que les haga ganar una fortuna. *Writing software for profit*, sin embargo, pretende señalar los escollos ocultos de esta búsqueda, sugiriendo algunos puntos a tener en consideración al escribir la primera línea de código. En su obra, A. J. Harding da una idea general sobre la industria del software, ilustrando la clase de programación y la presentación esperadas: un aspecto del negocio que con frecuencia soslayan otras publicaciones que tratan este tema.

Título: *Writing software for profit*
Autor: A. J. Harding
Editado por: Virgin Books

Reinventando al hombre

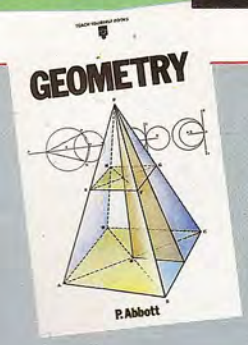


Reinventing man se ha convertido en una suerte de obra de referencia obligada para quienes se interesen por la robótica. Sin embargo, no proporciona demasiados detalles técnicos acerca de los robots. Intenta más bien repasar la historia del "hombre mecánico", comparándola con el desarrollo actual y probable y abatiendo varias falacias a lo largo de sus páginas.

Los autores explican de forma detallada cómo la tecnología actual puede conducir al hardware y el software que probablemente se necesitará para desarrollar los robots tan mitificados por los amantes de la ciencia-ficción, y predicen las limitaciones que es probable que tenga esta tecnología. Quien esté interesado en la inteligencia artificial y la robótica encontrará este libro absorbente, tanto por su riqueza de información como por el apasionante tema que aborda.

Título: *Reinventing man*
Autores: Igor Aleksander y Piers Burnett
Editado por: Kogan Page

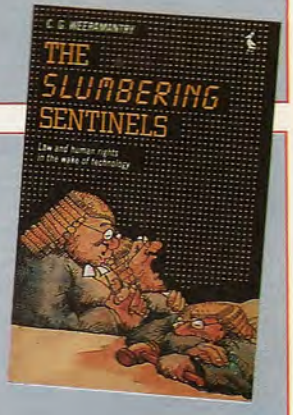
Aprenda geometría por sí mismo



Una de las dificultades con que tropiezan muchos programadores cuando empiezan a abordar proyectos más ambiciosos es la falta de teoría matemática o algebraica, el conocimiento de la cual puede ser especialmente útil para crear métodos de codificación nuevos y más compactos. *Teach yourself geometry*, al igual que otros libros de la serie, ofrece una base sólida en su tema, complementada con varios ejercicios, cuyas respuestas se incluyen al final. Es especialmente útil si usted desea hacer mucha programación de gráficos y constituye un significativo ejemplo de un tipo de literatura que, aunque no dirigida específicamente a los usuarios de ordenadores, puede ser de suma utilidad.

Título: *Teach yourself geometry*
Autor: P. Abbott
Editado por: Teach Yourself Books/Hodder and Stoughton

Los centinelas dormidos



The slumbering sentinels aborda un aspecto esencial de la nueva tecnología que está surgiendo: cómo la enfoca, la encuadra y regula la ley. Asimismo, se centra en dilemas morales planteados por los desarrollos tecnológicos y su efecto sobre los derechos del individuo. El libro no versa exclusivamente sobre ordenadores, pero aborda el tema con inquietante frecuencia, ilustrando la insuficiencia del marco constitucional actual de los diferentes países para hacer frente a situaciones tales como el almacenamiento y la revelación de información confidencial, evidenciando su renuencia a abordarlos, o al menos eso es lo que parecería. Del aspecto legal de la informática se habla raramente, y es su incursión en el campo del derecho lo que convierte a estos "centinelas dormidos" en un estudio interesante para quien esté comprometido en la aplicación (especialmente comercial) de la nueva tecnología.

Título: *The slumbering sentinels*
Autor: C. G. Weeramantry
Editado por: Penguin (Pelican Originals)

Tradúzcamelo

Niveles de compromiso
Desde el punto de vista del programador, se puede considerar el ordenador como tres "máquinas virtuales". A un nivel 3, el ordenador del ejemplo funciona en PASCAL. A nivel 2, funciona en assembler, y a nivel 1 el código máquina directamente se enfrenta con el hardware

Este capítulo final sobre programación del 68000 está dedicado al estudio de la operativa del ensamblador

A lo largo de esta serie hemos empleado con frecuencia las instrucciones a nivel ensamblador para ilustrar las características del microprocesador 68000. En nuestros análisis supusimos que el ordenador obedece estas instrucciones según van apareciendo en su forma ensamblador. La máquina interpreta de hecho un código binario que representa las instrucciones ensamblador de nivel más alto, siendo tarea del ensamblador el *traducir* estas instrucciones a código máquina.

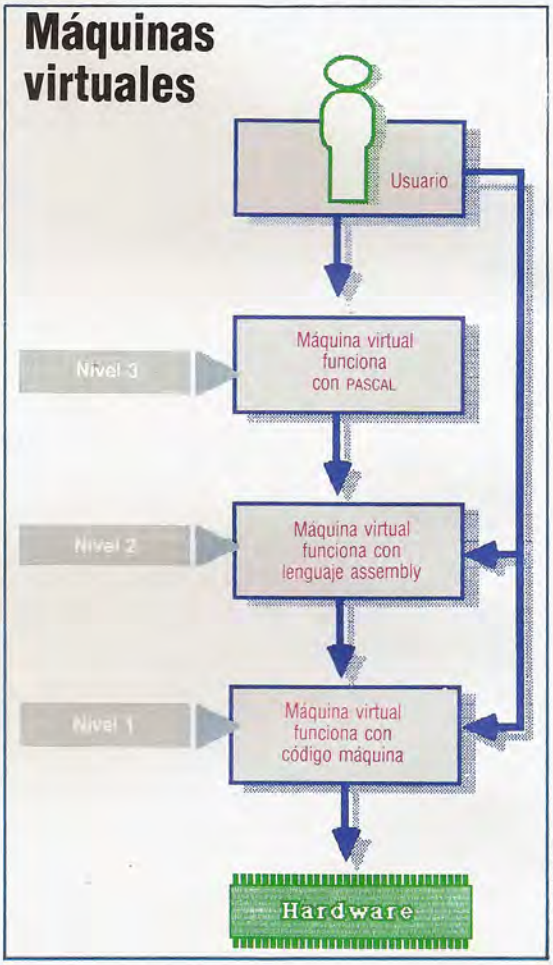
Sin embargo, es conveniente considerar la ejecución de nuestros programas como si la máquina los ejecutara a nivel fuente. Este nivel puede ser muy alto (p. ej., los paquetes de aplicación como el *WordStar*) o ligeramente inferior (como los programas en PASCAL) o muy bajo, casi equiparable al nivel de los programas de ensamblador. Esto explica el enfoque a varios niveles de nuestra visión de la máquina, que se compone de estratos discretos o niveles que se convierten en máquinas *virtuales*, ejecutoras de un determinado lenguaje de programación, como se ilustra en el diagrama (derecha).

El modo como un programa funciona a cualquier nivel es uno de éstos:

- *Traducción*: Un programa que está a un nivel se traduce al nivel inferior de la máquina. Por ejemplo, podemos traducir un programa PASCAL de nivel 3 en ensamblador para ser tratado a nivel 2 por un compilador.
- *Interpretación*: Un programa en un nivel es interpretado por un programa (denominado *intérprete*, claro está) que funciona en una máquina de nivel inferior. Por ejemplo, la máquina interpreta los modelos binarios de bits que representan las instrucciones para operaciones de máquina (o nivel de registro). Igualmente, el nivel 2 puede interpretar también programas en PASCAL desde el nivel 3.

Lo que más nos interesa aquí es el proceso de traducción. Hay muchas formas de traducción que van desde los compiladores hasta los ensambladores, pero todas tienen una cosa en común: todas toman instrucciones de alto nivel y proporcionan las correspondientes instrucciones a nivel inferior. Las instrucciones fuente originales no se precisan entonces, y en cierto sentido son "tiradas a la papelera" por el traductor (sin embargo, un intérprete todavía tendría necesidad del programa fuente original).

El objetivo del ensamblador es traducir las instrucciones de ensamblador en alto nivel a código binario ejecutable por la máquina. Por ejemplo:




```
MOVE.W D3,D5
```

se traduciría así:

```
0011 101 000 000 011
```

El 0011 corresponde a "mover una palabra" (MOVE.W); 101 000 corresponde a "al D5"; y 000 011 corresponde a "desde el D3".

Pero el trabajo a nivel de codificación de bits es extraordinariamente tedioso y propenso a errores, por lo que como mínimo necesitamos expresiones mnemotécnicas que nos permitan recordar las instrucciones y los objetos de datos. Por ejemplo, MOVE.W TOTAL,D4 significa que nos podemos referir a posiciones de memoria mediante nombres simbólicos que nos indiquen el significado del contenido de esa posición (TOTAL en este ejemplo).

Los tipos de error que podríamos cometer si codificáramos los bits a mano serían, entre otros:

- Equivocar los códigos de instrucción (opcodes).
- Direccionamiento absoluto incorrecto.
- Asignar un número inexacto de bytes por instrucción.

Esto quiere decir que no hablaremos siquiera del caso en que haya que hacer una traducción manual que supere media página de instrucciones. De aquí la importancia desempeñada por el ensamblador en la traducción del código fuente al código objeto.

El proceso assembly

El proceso assembly se inicia cuando el ensamblador lee las sentencias fuente en el archivo de texto fuente (texto escrito en ASCII) y produce un archivo de listado (o lo imprime) más un archivo objeto (o lo carga en la máquina). Nuestro archivo Formato Listado Assembly, que ilustra los elementos de una traducción, es un programa que ejecuta un cálculo aritmético en los elementos de una tabla.

Varios son los aspectos dignos de nota en el listado. Primero, usted establece una lista de contenidos de las posiciones y después el resto de la línea continúa con el código fuente original que sigue a un número insertado de sentencia. La información de errores se referirá al número de sentencia. Si la línea 14 contiene un error, se imprimirá E en esa línea y se hará una referencia a esa línea en la sección de información de errores del listado.

El formato del archivo binario es interesante por varias razones. Ante todo, el archivo ha de contener la información binaria en forma codificada, así como información del cargador para poder cargar correctamente el código binario en la memoria. La manera de conseguir esto es almacenando la información de direcciones y contenidos en binario codificado en hexa, por lo que el cargador debe conocer el formato binario. El formato exacto depende del ensamblador, pero la compatibilidad del cargador es esencial.

La última parte del listado es la impresión de la tabla de símbolos. Esta tabla proporciona los valores numéricos asignados a los niveles declarados en el programa. Por ejemplo, INPUT (definido en la sentencia 25) tiene valor 1024 (hexa) y está referenciado en la sentencia 10. Son todos detalles casi triviales en este pequeño programa de ejemplo, pero en un programa algo más extenso se convierten en información esencial para la depuración de errores.

El destino de la salida depende de donde esté funcionando el ensamblador (naturalmente, el ensamblador es un programa como otro cualquiera). Si, por ejemplo, estamos usando un método de *ensamblador cruzado*, el ensamblador funciona en un sistema huésped como el Unix, y el listado y la información binaria estarán en archivos. El archivo binario ha de ser cargado en el 68000 de destino antes de poder ejecutar el programa.

Otro modo de hacer esto es ejecutar el ensamblador en la máquina destino, si ésta tiene al menos algún tipo de sistema de gestión de archivos, aunque sea rudimentario. A veces los códigos binarios se cargan directamente en la memoria con el assembly destino y los listados pueden ser impresos directamente si está activada la impresora.

Los mecanismos del ensamblador

Los mecanismos que emplea un ensamblador nos dan idea de los problemas que pueden presentarse (¡que inevitablemente se presentarán!). Por lo general, el proceso de assembly es una organización de *dos pasos*. En el primer paso, el ensamblador:

- Decodifica los mnemotécnicos assembly (MOVE, ADD, MULS y demás).
- Cuenta los bytes de dirección (para poder calcular las direcciones).
- Construye una tabla de símbolos (que nos dice los valores asignados a cada símbolo).
- Descarta cualquier error de sintaxis (p. ej., ADDQQ es una instrucción ilegal).

Cuando el ensamblador lee la sentencia 9 en el primer paso por el programa *Formato listado assembly*, el contenido de etiqueta de la posición será 41F8. Que corresponde a un "movimiento rápido" a D0 en el modo inmediato con una constante de 12 (desde LENGTH en la tabla de símbolos).

Para la sentencia 10, el ensamblador puede establecer la posición 1002 con 41F8 para la instrucción LEA, pero no puede establecer todavía la dirección de INPUT. Así, la posición 1004 se deja vacía hasta el segundo paso cuando se conoce la dirección de INPUT.

En el segundo paso, el ensamblador puede sustituir las direcciones ahora conocidas dondequiera que se empleen dentro del programa, y dar salida al código binario y listado del programa. Pueden también darse errores que depurar, pero por lo general se atienden en el primer paso.

Formato listado assembly

```
POS     OBJETO                          SENT                    SENTENCIA FUENTE
                                        1   *Esta sentencia toma cada elemento de una tabla
                                        2   *y lo convierte en
                                        3   *                   INPUT[I]:=2*INPUT[I]+3
                                        4
                                        5
001000                                  6           ORG $1000
        =000C                           7   length: equ 12
                                        8
001000  700C                            9   start:  moveq   #length,d0
001002  41F8  1024                      10          lea     input,a0
001006  4244                            11          clr     d4
001008  4243                            12          clr     d3
                                        13
00100A  3610                            14  loop:   move.w  (a0),d3
00100C  C7FC  0002                      15          muls    #2,d3
001010  D67C  0003                      16          add.w   #3,d3
001014  30C3                            17          move    d3,(a0)+
001016  D843                            18          add     d3,d4
                                        19
001018  5340                            20          subq    #1,d0
00101A  6600  FFEE                      21          bne     loop
00101E  31C4  103C                      22          move    d4,sum
001022  4F40                            23          trap    #0
                                        24
001024  0001  0002  0003  0004          25  input:  dc.w    1,2,3,4,5
        0005
00102F  0006  0007  0008  0009          26          dc.w    6,7,8,9,10
        000A
001038  000B  000C                      27          dc.w    11,12
                                        28
00103C  0000                            29  sum:    dc.w    0
                                        30
                                        31          end
No se encontraron errores en este ensamblaje

        SÍMBOLOS                DEFN        VALOR       REFERENCIAS
INPUT                           25          1024        10
LENGTH                          7           000C        9
LOOP                            14          100A        21
START                           9U          1000
SUM                             29          103C        22
```

Podemos emplear asimismo el ensamblador como una útil calculadora, creando las expresiones simbólicas que calculan el valor de un operando. Por ejemplo, estas líneas introductorias de un listado assembly:

```
LENGTH   DC.W     (START-ARRAY)*4
ARRAY    DC.W     1,2,3,4,5
START    MOVE.W   LENGTH,D3
```

Aquí necesitamos calcular la longitud de la tabla en bytes (START-ARRAY multiplicada por 4), para emplearlo en el programa (para poner 3 en el registro de datos en este caso). Si más tarde cambiamos en algún momento la longitud de la tabla dentro del programa en ensamblador, LENGTH se restablecerá automáticamente.

Ayudas de documentación

Otra ayuda que proporciona el ensamblador está en las áreas de documentación. Es posible agregar comentarios a nuestros programas que orienten a los lectores sobre el significado de las instrucciones, y también podemos incluir detalles sobre registros convenidos y diseño del programa, por ejemplo. Véase el siguiente listado:

```
*Programa para entrar una cadena de caracteres
 y verificarla
*El texto es entrado a través de ACIA y verificado
*A. Gómez - Dic 85
*
*Registros convenidos
*
*Registros de direcciones
*       A1 puntero para entrar cadena
*       A2 puntero para almacenar teclado
*       A3 puntero para carácter actual
*Registros de datos
*       D0 entrada carácter
*       D1 salida carácter
*
START JSR  INIT          inicializa la E/S
      JSR  READSTRING    lee cadena entrada
      JSR  VALIDATE      y la comprueba
      JSR  SIGNAL        señal si es correcta
      BRA  START         bucle indefinido
```

Leyendo estos comentarios nos informamos sobre el programa, su estructura, los registros convenidos. Probablemente sería pedir demasiado si se exigieran aún más detalles a este nivel, pero es claro que el lector puede descender a otro nivel y mirar los comentarios contenidos en el código de la subrutina si necesita más información.

Lo importante es que el programa quede razonablemente bien documentado con el empleo de comentarios.

Directivas del ensamblador

También el assembler nos permite otras ayudas a la programación en el área general de las *directivas del ensamblador*. Pueden ser meras ayudas documentales como TTL o "directiva del título", que ordena la impresión de un determinado título en cada página del listado, o un medio para pasar la información al ensamblador. Por ejemplo, puede que deseemos señalar el final de nuestro texto de entrada para que el ensamblador comience su ensamblaje. Nuestro módulo del programa contendrá las siguientes líneas:

```
*Los comentarios de documentación
*Con nuestro nombre y fecha por lo menos
        TTL   Éste es mi programa
        ORG   $1000   *una instrucción de origen

START MOVE D1,D2      *y algunas instrucciones
                      *y otras y otras más
        END           *y la directiva de final
```

Las más importantes directivas del ensamblador se resumen en la tabla de más abajo.

Formato del ensamblador

No es de extrañar que los ensambladores construidos por distintos fabricantes difieran en detalles en cuanto a su operativa y formato. No obstante, la mayoría de los ensambladores se ajustan a una estructuración fuente semejante a la que sigue. Primero, los comentarios pueden ponerse empleando el signo * como primer carácter de la línea, o después de una instrucción si entre ésta y el comentario media un espacio como mínimo. Así se permiten estas dos líneas:

```
*Esto sólo es un comentario
START  ADD  D1,D2 *y esto, otro
```

Lo que constituye una instrucción debe naturalmente definirse, y consta de tres campos opcionales:

- *Campo de etiqueta*: Cada nombre empleado como una etiqueta debe comenzar con un carácter alfabético y tendrá de longitud total menos de treinta caracteres alfanuméricos. Se notará que si se omite este campo se ha de insertar al menos un espacio.
- *Campo del opcode*: Se empleará una del conjunto de instrucciones del 68000.
- *Campo del operando*: Los operandos de instrucción legales deben usarse después de un espacio al menos tras el campo del opcode. Pero en el campo del operando no deben existir espacios aun en el caso de emplear dos operandos.

He aquí algunos ejemplos de sentencias no permitidas en ensamblador:

Tabla de directivas del ensamblador

ORG	Abreviatura de "origen". Especifica la dirección de inicio del código
RORG	Vale lo dicho para ORG, pero con código relativo del PC
TTL	Pone un título en cada página del listado
END	Fin del texto para ensamblar
EQU	"Igual a" (*equate*): asigna un valor al símbolo especificado (p. ej., LENGTH EQU 10+50*ARRAY1, donde LENGTH vale 610 suponiendo que ARRAY1 vale 12)
DC	"Declarar Constante": asigna un valor constante (byte, palabra o longitud palabra larga). P. ej., PARAM DC.W 12,23 hace a PARAM una palabra que contiene 12 y PARAM+2 es una palabra que contiene 23
DS	"Declarar eSpacio": reserva un área de datos no inicializados (p. ej., STACK DS.B 100 establece un espacio de 100 bytes llamado STACK)
LLEN	Establece la longitud de línea
PAGE	Envía un salto de página durante el listado

```
Esto no es un comentario del todo pues falta el *
MOVE     D1, D2        *no debe haber espacios
                       *entre los operandos
MOVE     8TOIT,D5      *los nombres han de
                        comenzar
                       *con una letra y ADEMÁS
                        mediará
                       *al menos un espacio entre
                       *instrucción y operando
RTS                    *correcto
RTS      D1            *esto ya no (pues D1 aquí
                       *no tiene sentido)
*fin de ejemplo erróneo con comentarios verdaderos!
```

Digamos, para acabar, algo sobre la representación alfanumérica. Los números se representan en el ensamblador como valores decimales si no van precedidos del signo $. Así, 1234 es decimal pero $1234 es hexadecimal. Para las letras ASCII los caracteres se encierran entre comillas sencillas. Por ejemplo:

'Que pena, este curso se acaba!'

Acabamos de examinar el empleo del ensamblador como una herramienta de programación, ya no sólo como un medio de introducir codificación en el 68000. Las facilidades ofrecidas son bastante buenas especialmente con la facilidad macro y las bastante buenas directivas del ensamblador, así como los mensajes de error. Las ayudas de documentación también han sido dignas de resaltar, ¡dado que nunca han de olvidarse en favor de otras personas que sin duda habrán de leer los programas que usted escriba!

Perfecto en ralentí
El QL posee un microprocesador 68008, idéntico al 68000 salvo su bus de datos reducido a 8 bits. Ambos chips son compatibles por completo en cuanto a las instrucciones, por lo que nuestro curso de programación del 68000 es aplicable para programar el 68008. ¡La única diferencia está en la velocidad!

Facilidad MACRO

Justamente la facilidad MACRO es un método para sustituir textos en un programa fuente que se ha revelado como una herramienta poderosa de programación en grandes programas. Observemos que donde se especifica una "macro" se sustituirá el texto macro y el anteriormente definido se rechazará. Se puede, por ejemplo, definir así una macro ERROR:

```
ERROR    MACRO
         MOVE.B    #3,ERRORLOC
         JSR       SIGNAL
         ENDMACRO
```

Finalmente, la palabra ERROR producirá, en tiempo de ensamblaje, la siguiente serie de instrucciones:

```
MOVE.B   #3,ERROR.LOC
JSR      SIGNAL
```

Ahora podemos ampliar esta facilidad para incluir el empleo de parámetros a cada llamada si lo precisamos.

Mire este ejemplo:

```
         TTL       Macro ejemplo
         ORG       $1000
ADDON    MACRO
         ADD       \1,D2
         ADD       D2,\2
         ENDM

         ADDON     AVAL,SUM
         ADDON     BVAL,SUM
         ADDON     SUM,TOTAL
AVAL     DC.W      0
BVAL     DC.W      0
SUM      DC.W      0
TOTAL    DC.W      0
         END
```

Observará que\1 y\2 se refieren a los parámetros primero y segundo. En tiempo de ensamblaje, esto producirá:

```
3                ORG       $1000
4
5    ADDON       MACRO
6                ADD       \1,D2
7                ADD       D2,\2
8                ENDM
9
10               ADDON     AVAL.SUM
11+              ADD       AVAL,D2
12+              ADD       D2,SUM
13               ADDON     BVAL,SUM
14+              ADD       BVAL,D2
15+              ADD       D2,SUM
16               ADDON     SUM,TOTAL
17+              ADD       SUM,D2
18+              ADD       D2,TOTAL
19
20   AVAL        DC.W      0
21   BVAL        DC.W      0
22   SUM         DC.W      0
23   TOTAL       DC.W      0
24               END
```

Recuerde que los signos + después de los números de sentencia indican las líneas insertadas por el macroprocesador.

INDICE

Editorial Delta, S.A.

El índice ha sido realizado atendiendo a las especiales características de la obra, entre las cuales destaca haber proporcionado al lector un conocimiento enciclopédico de la informática y el trasfondo de su desarrollo y haber incluido apartados dedicados a temas tan disímiles como el bricolaje, el lenguaje máquina y la programación en BASIC.

Es así cómo, a cada una de las secciones individuales del curso, se ha asignado un recuadro en que se resumen los temas principales tratados bajo ese encabezamiento. Por ejemplo, en el recuadro *Programación* se da una relación de los temas tratados en esa sección del curso: desde los principios básicos de la programación hasta los proyectos que se han desarrollado.

El índice general está confeccionado con referencias cruzadas que remiten a los temas citados en esos recuadros y a entradas del mismo índice. El término *véase* seguido del nombre del apartado en letra cursiva remite al recuadro presidido por dicha denominación. Así, por ejemplo, "Música: *véase Aplicaciones*" remite al recuadro de igual título, donde se informa que el tema ha sido tratado en una serie dedicada a esa aplicación específica. A su vez, el término *véase* seguido de una palabra en letra redonda remite a la entrada de ésta en el índice general.

Al confeccionar el índice se ha intentado evitar crear entradas que incluyan una larga lista de números de página, basándonos en el criterio de remitir al lector a los puntos fundamentales del curso en donde se ha tratado un determinado tema y no a todas aquellas páginas donde se hace referencia a él.

Del mismo modo, las entradas correspondientes a los micros personales remiten por lo general a sus capítulos de hardware.

En el extremo inferior de cada página se incluye una pequeña tabla que indica los números de página correspondientes a cada volumen.

A

Abacus: 1204-1205
Absorción: 526
Aceleración de los programas: 1076-1077
ACIA: 1851
Acoplador acústico: 216-217, 596
Acorn: 540
Acorn DFS: 564-566, 570
Acorn Electron: 370-371, 1170
Acorn Electron Plus 3: 1549-1551
Acorn Plus 1: 929-931
Acorn Plus 3: 1549-1551
ACT Apricot: 729-731
Acumulador (registro del microprocesador): 596, 615-617
Administrador de bases de datos: 1578, 1704-1705
Advance 86: 829-831
Ahorcado, juego del: 984-985
AIM 65: 589-591
Ajedrez: 781-783, 1748-1751, 1896-1897
Álgebra booleana: 488-489, 526-528
ALGOL: 2067
Algoritmo: 814, 866-867
Alien (Atari): 2309
Altavoz: 595
ALU: *véase* Unidad aritmética lógica
Alunizaje (Commodore 64): 1548
Amsoft Speech Synthesiser: 1949-1951
Amstrad CPC 464: 909-912, 1170; sistema operativo, *véase Lenguaje máquina*
Amstrad CPC 664: 1970-1971; sistema operativo, *véase Lenguaje máquina*
Amstrad CPC 6128: 2109-2111
Amstrad DDI-1 (unidad de disco): 1689-1691
Amstrad PCW 8256: 2269-2271
Análisis de sistemas: 854-855, 2089
AND: 68-69, 92-93, 488, 546, 625
Ant attack: 486
Apocalypse: 836
Apple IIc: 1048-1051
Apple IIe: 349-351
Apple LaserWriter (impresora): 2009-2011
Apple Macintosh: 1949-1952, 2310-2311
Aprendizaje asistido por ordenador: 1621-1623
Aprendizaje de las máquinas: 1828-1831
Aprendizaje evolutivo: 1829
Apricot: 729-731
Apricot F1e: 1869-1871
Apricot Portable: 1489-1491
Aquarius: 290-291
Árbol binario: 1103
Árbol de búsqueda: 1722-1723
Árbol de conocimiento: 1753
Árbol de números: 1436
Archivo: 204-205, 664, 761, 1056
Archivos aleatorios: 724-725, 752-753
Archivos secuenciales: 684-685
Archon: 1740
Arquitectura Von Neumann: 2330
Artic Computing: 900
ASCII: 556-557
Atari: 519
Atari 400: 6, 109-111
Atari 800: 6, 109-111
Atari 130XE: 1789-1791
Atari 520ST: 2029-2031
Atari 600 XL/800XL: 669-671, 1171
Atari 810 (unidad de disco): 543-545
Aterrizaje (EXL 100): 2300
Atic-Atac: 856
Audiogenic: 940
Audiojuegos: 754
Autopista (Alice): 1788

B

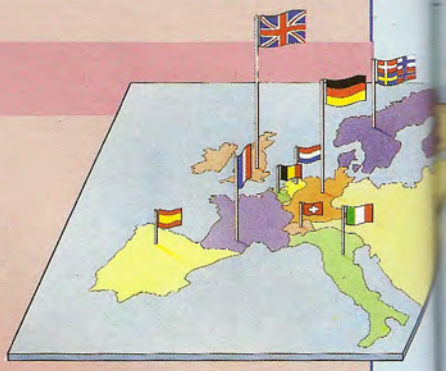
Babbage, Charles: 220
Banana Interface: 1770-1771
Bases de datos: 11, 124-125, 761-763, 806-807, 1577-1578, 1584-1585, 1604-1605
BASIC: 344, 500; *véase Ciencia informática*; principios de programación, *véase Programación*
BASICODE: 721-723
Batalla naval (ZX81): 1668
BBC+: 1929-1931
BBC Micro (modelos A y B): 6, 89-91, 1170, 2251; mapa de memoria, 1359; sistema operativo, *véase Lenguaje máquina*
BCD: *véase* Decimal codificado en binario
BDOS: *véase* Sistema operativo básico de disco
BEAGLE (sistema de aprendizaje): 1801-1803, 2165
Beasty: 1250-1251, 1392
Big Trak: 1563
Bingo (en PASCAL): 1596, 1614, 1616
BIOS: *véase* Sistema básico de entrada y salida
Bit: 28-29, 668
Bombardeo aéreo: Alice, 1468; Dragon, 1608
Bosque encantado, El (juego de aventuras): *véase Programación*
Boulder Dash: 1960
Brainstorm: 1344-1345
Brazo-robot: 314-315, 1321-1323; construcción de un, *véase Bricolaje*
Brother EP-44 (máquina de escribir): 886-887
Bucle: 697, 894
Bucle anidado: 628
Bucle condicionado: 609
Bucle incondicionado: 588
Buffer: 236-237
Bug-Byte: 820
Bugaboo (La pulga): 776
Bush, Vannevar: 400
Bushnell, Nolan: 519
Byte: 28-29

C

C: *véase Ciencia informática*
Cacería de patos (Commodore 64): 2328
CAD: *véase* Diseño asistido por ordenador
Caja buffer: 1279
Caja de salida: 1278
CAL: *véase* Aprendizaje asistido por ordenador
Calculadora de Leibniz: 2173
Cámara Snap: 1229-1231
Cámaras fotográficas: 1389-1391
Camionero del desierto, El: 792
Campo: 706, 761, 1585
Camputers: 740
Cangrejos: Commodore 64, 1296; MSX, 2229, Oric Atmos, 1168
Caracteres de "máscara": 1785

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Aplicaciones



Carreras de informática

Especialidades: 101, 2261
Proceso de datos: 2290
Ventas y marketing: 2301
Ingeniería: 2321
Industria de juegos: 2350

Comunicaciones

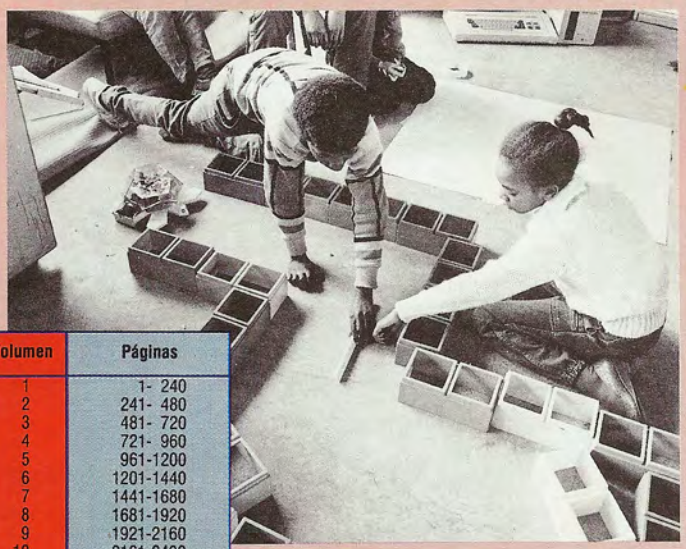
Por medio de redes: 801
Por medio de modems: 1381
Protocolos de terminal: 1401
Elección de terminal: 1421
Sistemas de acceso público: 1443
Tablones de anuncios: 1461
Videotex: 2221

Educación

Perspectiva histórica: 1481
Desarrollo de aptitudes: 1501
Niños minusválidos: 1521
Juegos de aventuras y de simulación: 1541
Robots y LOGO: 1561
Bases de datos: 1581
Lenguajes de programación: 1601
El controvertido CAL: 1621
El ordenador en la escuela: 1641
Contenido de los cursos de informática: 2241
El futuro de la informática educativa: 1661

Industria informática

Fabricación de ordenadores: 901
Marketing: 841
MSX Standard: 621
Ordenadores controlados por el pensamiento: 1681
Ordenadores ópticos: 1981
Producción de software: 861
Traducción de software: 1361
Desarrollos futuros: 2361



Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

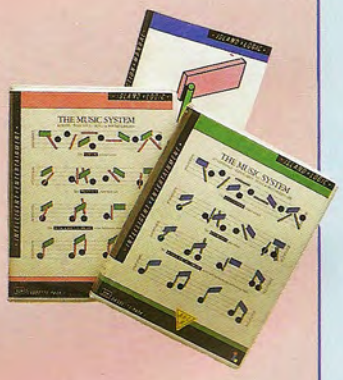
Inteligencia artificial

Perspectiva histórica: 1701
Técnicas de búsqueda: 1721
Estrategias de anticipación: 1748
Técnicas de búsqueda arborescente: 1761
Sistemas expertos: 1781
Sistemas de aprendizaje: 1801
Código genético: 1828
Proceso de lenguaje natural: 1841
Reconocimiento de patrones: 1861
Sistema WISARD: 1881
Sistemas autorreproductivos: 1901
Programación AI: 1924
Representación de conocimiento: 1941
Desarrollos futuros: 1961

Juegos de apuestas

Paquetes de software: 2081
Fórmulas de pago: 2101
Cálculo de probabilidades: 2129
Regla de Bayes: 2152
Simulador de apuestas: 2163

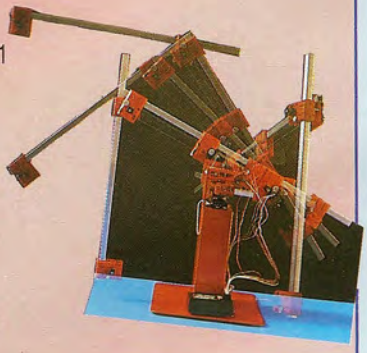
Música

Música electrónica: 961
Sintetizadores: 989
Interface MIDI: 1014
Controlador de MIDI: 1033
Instrumentos basados en MIDI: 1041
Instrumentos de muestreo: 1061
Paquetes de música para micros: 2181

Ordenadores en la industria

Utilización del CAD: 2001
Fabricación de productos acabados: 2021
En la industria editorial: 2041

Robótica

Dispositivos controlados por ordenador: 701
Aspectos históricos y literarios: 1081
Movimiento del robot: 1101
Control del robot: 1121
Actuadores y efectores finales: 1141
Realimentación sensorial: 1161
Brazos-robot: 1181
Movimiento inteligente: 1201
Percepción visual: 1221
Síntesis y reconocimiento de voz: 1241
Percepción del entorno: 1261
Realizar un modelo: 1281
Robots en el mercado: 1301
Brazos-robot en el mercado: 1321
Perspectivas futuras: 1341



Carreras de informática: *véase Aplicaciones*
Cartucho: 484
Casilla de iteración: 894
Casio: 1040
Cassette: 94-95
CCP: *véase* Procesador de instrucciones de consola
CD-ROM: *véase* ROM en disco compacto
Cibernética: 1901-1903
Ciempiés (Vic-20): 2080
Cifras (Alice): 1569
CIM: *véase* Fabricación integrada por ordenador
Cinta de cassette: 484
Circuito (diagrama): 595
Circuito biestable: 708-709
Circuito monoestable: 726-727
Clasificación: 244-245, 286-287
Clasificación Shell: 413
Classic racing: 1080
COBOL: *véase Ciencia informática*
Codificación: 626-627
Codificación digital del sonido: 989-991, 1062
Codificación Hoffmann: 2314-2315
Codificador de eje: 1122
Código de barras: 40
Código de Gray: 348
Código de operación: 578
Código Morse (proyecto): 1128
Código reubicable: 1197
Códigos de enganche: 1878-1879, 1898-1900
Códigos Hamming: 298-299
Coleco Adam: 869-871
Colour Genie: 6, 789-791
COMAL: 344
Comando de misiles: 1140
Commodore 16: 1269-1271, 1171
Commodore 64: 7, 49-51, 490-491, 712-714, 1171, 2250; sistema operativo, *véase Lenguaje máquina*
Commodore Amiga: 2281-2283
Commodore Business Machines: 599-600
Commodore Pet: 430-431, 1788
Commodore Plus/4: 1171, 1189-1191
Commodore SX64: 490
Commodore Vic-20: 7, 230-231
Communicator 104 (modem): 2308
Compaq Plus: 1349-1351
Compatibilidad: 663
Complemento a dos: 758
Complemento a uno: 758
Compresión de textos: 2314-2315, 2335-2337, 2346-2348, 2363-2365
Comprobación de programas: 1086-1087
Comprobación de reacciones: 917
Compunet: 1443-1445
Comunicaciones: *véase Aplicaciones*
Condensador: 595, 619
Conector de páginas: 648
Conector de partes: 648
Conexión en red: 826-827
Conexión mental: 1681-1683
Conjuntos: 1594-1595, 1614-1615
Conquer Chestnut: 2369-2371
Contador de programa: 617
Control con palanca de mando: 1114-1116
Control de dos motores: 1092-1094
Control de realimentación: 1065-1067
Control de relé: 1054-1055
Control del ordenador: 701-703
Convertidor A/D: *véase* Convertidor de analógico a digital
Convertidor D/A: *véase* Convertidor de digital a analógico
Convertidor de analógico a digital: 703
Convertidor de digital a analógico: 703, 1194-1195, 1280
CP/M: 719-720, 1744-1745, 1764-1765, 1784-1785, 1804-1805, 1826-1827
CPU: *véase* Unidad central de proceso
Cray-1 (superordenador): 1884-1885
Criba de Eratóstenes, La (programa en FORTH): 1984-1985
Criptografía: 454-455
Crisis de software: 2333
Cuadrados mágicos: 766-767
Cuatro en raya (Atari): 1648
Curva de Sierpinski: 1105
Curva del copo de nieve: 1104

CH

Chip de video: 2230-2231
Chip Motorola 68000: 523
Chip SID (dispositivo interface para sonido): 1717-1720

D

dBase II: 1632-1633
DBM: *véase* Administrador de bases de datos
DEC VAX 11/780: 2341-2343
Decimal codificado en binario: 675
Decodificación: 626-627
Defensa antiaérea (Commodore 64): 2248
Densidad (en disco): 604-605
Depuración de errores: 432-433, 814, 964-965
Desoldadura: 548-549
Desplazamiento (*shift*): 777-779
Detector de mentiras: 1683
Deus ex machina: 1220
Diagrama de bloques: 508, 529
Diagrama de flujo: 104-105, 508
Diagramas de Karnaugh: 572-573, 586-587
Diagramas de Venn: 128-129, 526
Dibujo de circunferencias: 918-919
Dibujos animados: 181-183
Digital Research: 719-720
Digitalizador: 258-259
Digitaya (juego de aventuras): *véase Programación*
Diodo: 618
Diodo emisor de luz: 595
Direccionamiento: 676-678
Direccionamiento de página cero: 676
Direccionamiento indexado: 676-677
Direccionamiento indirecto: 677-678
Directorio: 665
Disco flexible: 114-115, 484-485, 604-605
Disco láser: 434-435, 681-683
Disco Winchester: 352-353
Discovery 1: 1634-1636
Diseñador de juegos: 521
Diseñador de ordenadores: 2261
Diseño asistido por ordenador: 421-423, 2001- 2003
Diseño de circuitos: 586, 587
Diseño geométrico: 704-705
Diskette: *véase* Disco flexible

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Disposición lógica no comprometida: 388
Documentación: 834-835
Dongle (dispositivo "antipirata"): 193
Dr LOGO: 1690-1691, 1914-1915
Dragon 32: 7, 70-71
Dragon 64: 610-611
Dragon Data: 800
Dragon DOS: 584-585
Dualidad: 526
Dynabook: 2361-2362

E

EasyScript: 2215
Echo Synthesiser: 1821-1822
Editor: 662
Editor de pantalla: 308
Educación: *véase Aplicaciones*
Elección de un lenguaje de programación: 2372-2373
Elite: 1520
En la luna (Atari): 1769
ENIAC (proyecto): 140
"Enigma de un crimen" (LOGO): 1314
Ensambladores: 2354-2356
Enterprise Sixty Four: 1469-1471
Entrada de parámetros: 1137
Entrada/salida: 112-113
Epson HX-20: 7, 169-171
Epson PX-8: 1089-1091
Epson SQ-2000 (impresora): 2090-2091
Equipos de herramientas: 444-445
Ergonómica: 321-323, 1001-1002
Etiqueta: 697
Exocet: EXL 100, 2208; Vic-20, 1274
Expert-Ease: 2201-2203

F

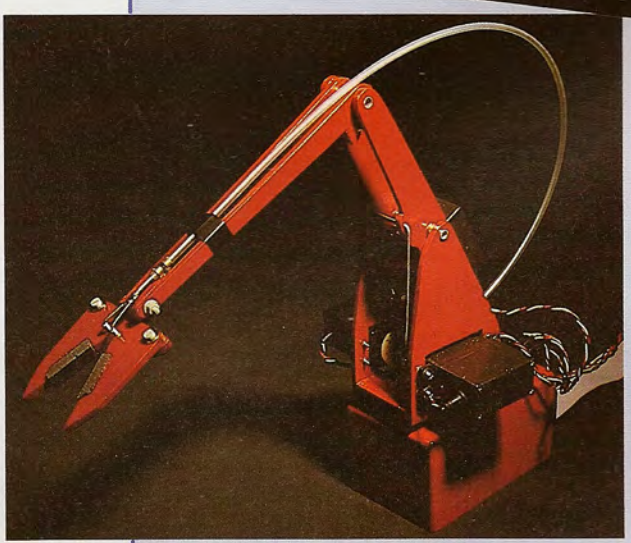
Fabricación de ordenadores: 901-903
Fabricación integrada por ordenador: 1990-1991
Facilidades de ayuda: 1006-1007
Fairlight CMI (Computer Musical Instrument): 1061-1063
Farmfax Suite: 1544-1545
Figuras geométricas: 1452-1454
Figuras *lissajous*: 1215-1217
Fischertechnik Robotics Kit: 1669
Flags de condición: 696
Flip-flop: 708-709, 726-727
Floppy: *véase* Disco flexible
Flyerfox: 1240
Football manager: 1980
FORTH: 345; *véase Ciencia informática*
FORTRAN: *véase Ciencia informática*
Frankie goes to Hollywood: 1995
From chips to systems (libro): 2374

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Bricolaje

Mantenimiento y diseño del micro
Soldaduras: 524
Sustitución de componentes: 548
Uso del multímetro: 566
Componentes electrónicos: 618
Construcción de puertas lógicas: 624
Construcción de un sumador incompleto: 664

Construcción de mecanismos de control
Acceso a la puerta para el usuario: 994
Diseño de la caja del buffer: 1003
Construcción de la caja del buffer: 1026
Control de relé: 1054
Control de realimentación: 1065
Control de dos motores: 1092
Control con palanca de mando: 1114
Construcción de una caja de relés: 1126
Aplicaciones de relés: 1154
Visualización LED: 1165
Convertidor D/A: 1194
Software para el convertidor D/A: 1212, 1226
Trazado del mapa de un laberinto: 1252
Diagramas de circuitos: 1278
Servomotores: 1392, 1403



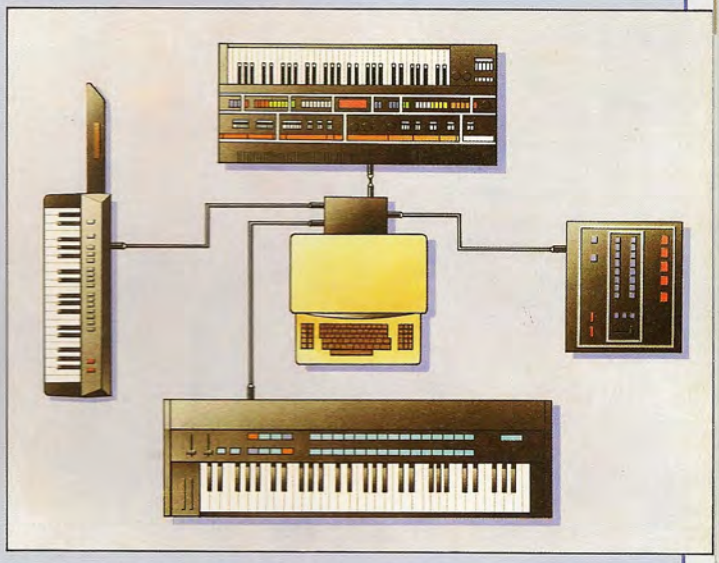
Bricolaje

Construcción de un robot móvil
- Diseño del proyecto: 1290
- Motores paso a paso: 1315
- Control del robot: 1336
- Montaje de los sensores: 1356
- Calibrado del robot: 1374
- Programa de medición: 1423
- Programa de exploración: 1455
- Interface para el Spectrum: 1472, 1492, 1514
- "Vista" para el robot: 1538

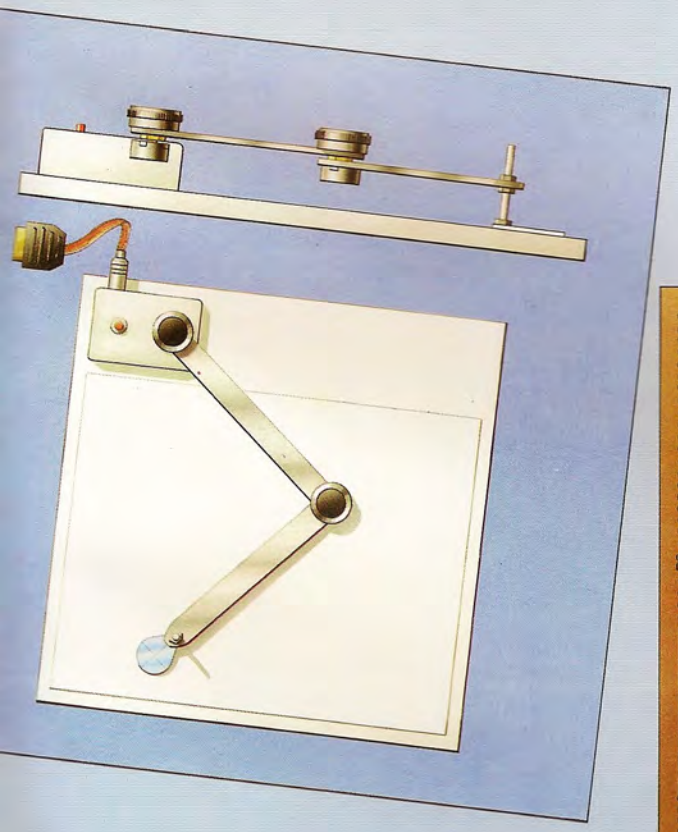
Construcción de un brazo-robot
- Diseño básico: 1552
- Patrones y componentes: 1574
- Ensamblaje del cuerpo principal: 1592
- Mecanismo de prensión: 1606
- Conexiones eléctricas: 1630
- Programador de secuencias: 1655, 1675, 1696
- Curvas cúbicas: 1706

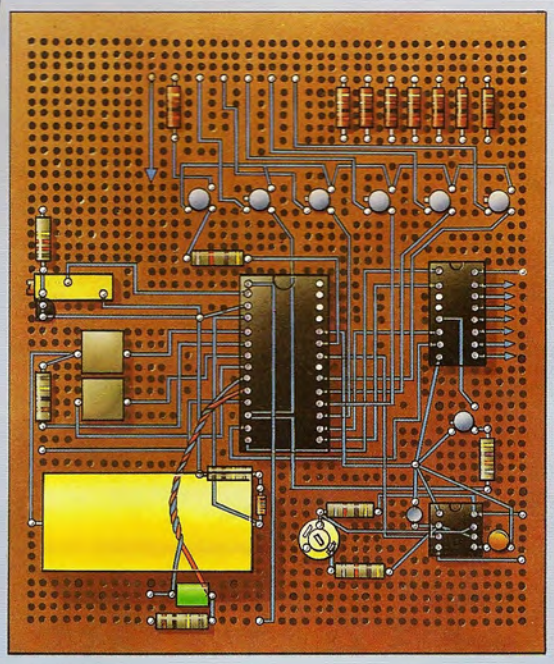
Construcción de un trazador digital
- Líneas generales: 1733
- Etapas del proyecto: 1754
- Calibrado: 1766
- Software de control: 1766, 1786

Construcción de una interface MIDI
- Especificaciones de hardware: 1815
- Diagramas y componentes: 1823
- Conexión del chip ACIA: 1848
- Especificaciones de software: 1866
- Dispositivo digital grabador y reproductor: 1892, 1912
- Adaptación para la gama Amstrad: 2112, 2123



Construcción de un tester digital
- Introducción: 1932
- Circuito del DVM: 1944
- Proceso de conversión: 1964
- El chip 7135: 1996, 2004
- Montaje de los componentes pasivos: 2032
- Comprobación de las conexiones: 2052
- Modificaciones al diseño: 2064
- Ampliaciones opcionales: 2086



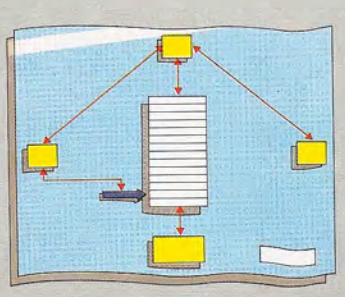


Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Ciencia informática

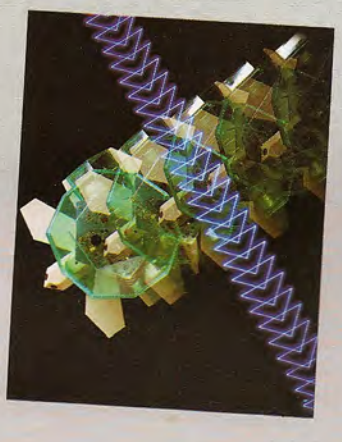
Introducción a la ciencia informática

Álgebra booleana: 488
Circuitos lógicos: 512
Leyes del álgebra booleana: 526
Operadores lógicos: 546
Diagramas de Karnaugh: 572
Diseño de circuitos: 586
Repaso de lógica: 606
Circuitos codificadores y decodificadores: 626
Puertas NAND y NOR: 646
Generadores de bits de paridad: 666
Visualizaciones en siete segmentos: 686
Flip-flops RS: 708
Circuitos temporizadores: 726
Transferencia de datos: 746
Unidad aritmética lógica: 772



BASIC

Primeros pasos: 20
El bucle FOR-NEXT: 37
Sentencia GOSUB: 77
Variables con subíndice: 116
Funciones incorporadas: 146
Función RND: 172
Matrices bidimensionales: 194
Estructuras de control: 212
Programa de base de datos: 232
Búsqueda binaria: 272
Programación *top-down*: 292
Unión de subprogramas: 354
Creación de archivos: 376
Clasificación en serie: 396
Localización de registros: 416



LOGO

Presentación: 986
Gráficos tortuga: 1012
Procedimientos: 1023
Espacio de trabajo: 1044
Parámetros de entrada: 1073
Técnicas recursivas: 1084
Gráficos: 1103
Entrada/salida: 1134
Utilización de sprites: 1146
Juego de estrategia: 1174
Diablos: 1186
Trabajo con números: 1215
Tratamiento de listas: 1234
Juego de aventuras: 1254, 1266, 1288
Investigación de un crimen: 1312

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Capacidades recursivas: 1333
Visualización de gráficos: 1353
Dibujos cicloides: 1366
Transformaciones espaciales: 1395
Patrones en dos dimensiones: 1414
Facilidades matemáticas: 1434
Formas geométricas: 1452

PASCAL

Desarrollo del lenguaje: 1466
Sintaxis y vocabulario: 1484
Sentencias compuestas: 1504
Sentencias IF y CASE: 1526
Funciones aritméticas: 1554
Construcciones iterativas: 1572
Conjuntos: 1594
Registros: 1614
Matrices: 1626
Procedimientos y funciones: 1658
Solidez estructural: 1664
Estructuras de archivo: 1684
Asignación dinámica de memoria: 1714
Recursión: 1726

PROLOG

Variables lógicas: 1752
Sentencias declarativas: 1772
Procedimientos de búsqueda: 1792
Características extralógicas: 1806
Programación de AI: 1832

LISP

Programación de AI: 1856
Listas y funciones: 1864
Problema de ajedrez: 1895

FORTH

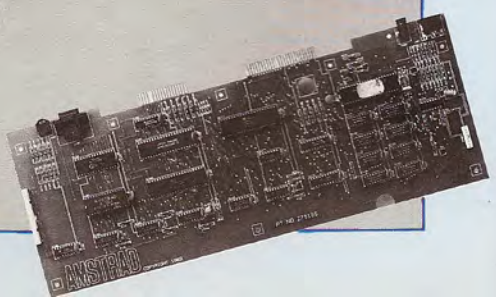
Interactividad y ampliabilidad: 1916
Palabras y diccionarios: 1926
Manipulación de datos: 1955
Estructuras de control condicional: 1966
Facilidades aritméticas: 1984
Programa de compilación: 2015
Estructuras de matriz: 2035
Manipulación de series: 2047
Epílogo: 2092

Lenguajes clásicos

Lenguajes pioneros: 2066
FORTRAN: 2093, 2115, 2134
COBOL: 2143, 2172, 2186

C

Desarrollo del lenguaje: 2204
Estructuras de control: 2235
Funciones c: 2243
Clases de variables: 2274
Punteros y series: 2296
Tipos de datos: 2312
Comandos stándar de E/S: 2324
Interface Unix-C: 2352



Funciones definidas por el usuario: 712-713
Funciones trigonométricas: 634-635, 654-655

G

Gates, Bill: 500
GEM: 2194-2195, 2206-2207
Generador de aplicaciones: 406-407, 553
Generador de bits de paridad: 666-667
Generador de caracteres: 269
Generadores de caracteres definidos por el usuario: 1052-1053, 1068-1069, 1096-1097
Generadores de gráficos: 612-614
Generadores de juegos: 521-522
Generadores de programas: 1854-1855, 1876-1877
Gestión de memoria virtual: 2050
Ghostbusters: 1480
Go (juego de estrategia): *véase Programación*
Grabación digital: 1061-1064
Grabadora de cassettes: 8
Gráficos: 44-45, 152-154; aplicaciones, *véase Lenguaje máquina*; en programación de juegos, *véase Programación*
Gráficos de barras: 1353-1355
Gráficos de tarta: 1353-1355
Gráficos tridimensionales: 187, 812-813, 932-933
Grafpad (digitalizador): 649-651
Gran Premio: Atari, 1408; MSX, 2169
Gran Premio 2 (Atari): 1695
Graph Plan: 1232-1233
Guía del comprador: 1421-1422

H

Hacker's handbook, The: 2374
Hacking (acceso ilícito): 966-967
Hashing: 752-753
Hisoft Devpac (paquete ensamblador): 2354
Hitch-hiker's guide to artificial intelligence, The (libro): 2374
Hitchhiker's guide to the galaxy, The: 1880
Hojas electrónicas: 11, 158-160, 786-787, 1172-1173, 1192-1193, 1204,1205, 1264-1265, 1372-1373; programación de una, *véase Programación*
Hollerith, Herman: 240, 380
HOPE: 2332-2334
Hopper, Grace: 440, 2093
HULK: 561-563, 2083
Hybrid Music 500 (sintetizador): 1570-1571

I

IBM PC: 569-571
IBM PC/AT: 2049-2051
Imagine: 559-560
Impossible mission: 1700
Impresoras: 8, 74-76; *véase Hardware*
Impresoras de chorro de tinta: 372-373, 2090-2091
Impresoras de rueda margarita: 844-845
Impresoras matriciales: 784-785, 804-805, 824-825
Impresoras sin impacto: 550-552
Impresoras térmicas: 1741-1743
Impresoras-plotters: 769-771
Indexación: 752-753
Industria informática: *véase Aplicaciones*
Informática
Cronología, 86-88; la informática en el transporte, 341-343; en la astronomía, 346-347; en la ciencia-ficción, 381-383; en la enseñanza, 81-83; en la industria, 1990-1991, 2021-2023; en la industria editorial, 2041-2043, 2061-2063; en la medicina, 126-127; en la meteorología, 248-249; en los juegos de apuestas, 461-463, 2081-2083, 2101-2103; estudios de, 2241-2242
Ingenieros de hardware: 2321-2323
Ingenieros de mantenimiento: 2263
Ingenieros de software: 2321-2323
Inmos Transputer: 2329-2331
Input/output: *véase* Entrada/salida
Inserción de parámetros: 1137
Integración a gran escala: 468
Integración a muy gran escala: 468
Inteligencia artificial (AI): *véase Aplicaciones* y *Ciencia informática* (LISP y PROLOG)
Interacción hombre-máquina: 1841-1844
Interface: 206-208
Interface Electron Plus 1: 929-931
Interface hombre-máquina: 992-993
Interface MIDI, Construcción de una: *véase Bricolaje*
Interferómetro Mach Zehnder: 1982-1983
Intérprete: 662
Interrupciones: 1858-1859
Interruptor: 595
Introducción a la ciencia informática: *véase Ciencia informática*
Invertir, juego: 879
Iteración: 894

J

Jerga industrial: 1991
Jerga informática: 428-429
Jet Pac: 875
Juegos de apuestas: *véase Aplicaciones*
Juegos de aventuras: 161-163; programación de un, *véase Programación*
Juegos de azar: 1761-1763
Juegos de estrategia: 1748-1751; programación de un, *véase Programación*
Juegos de guerra: 441-443
Juegos de inteligencia: 361-363
Juegos de laberinto: 288-289
Juegos de naipes: programación de un, *véase Programación*
Juegos de personajes interactivos: programación de un, *véase Programación*
Juegos de simulación: programación de un, *véase Programación*
Juegos recreativos: 221-223
Juegos sonoros: 754
Juguetes didácticos: 401-403
Jupiter Ace: 6, 150-151

K

Knight Lore: 1680
Koala-pad: 1109-1111

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Hardware

Impresoras
Apple LaserWriter: 2009
Epson SQ-2000: 2090
Impresoras de chorro de tinta: 372, 2090
Impresoras matriciales: 784, 804 824
Impresoras de rueda margarita: 844
Impresoras térmicas: 1741
Impresoras sin impacto: 650
Impresoras-plotters: 769
Penman Plotter: 1649

Máquinas educativas
AIM 65: 589
Banana Interface: 1770
BBC+: 1929
Link 480Z: 1369
My Talking Computer: 1450

Micros de gestión
ACT Apricot: 729
Advance 86: 829
Amstrad PCW 8256: 2269
Apple IIc: 1048
Apple Macintosh: 1949, 2310
Apricot F1e: 1869
Commodore Amiga: 2281
Compaq Plus: 1349
IBM PC: 569
IBM PC/AT: 2049
RM Nimbus: 1808
Sanyo MBC-550: 1589
Tandy 1000: 1729

Micros portátiles
Apricot Portable: 1489
Epson PX-8: 1089
Osborne Encore: 1329
Sharp PC-5000: 690
Tandy Modelo 100: 1130
Wren Executive: 1709

Modems
Communicator 104: 2308
Modem 1000: 2307
Nightingale: 2306
VTX 5000: 2307
WS 3000: 2308

Ordenadores personales
Acorn Electron Plus 1: 929
Advance 86: 829
Amstrad CPC 464: 909
Amstrad CPC 664: 1970
Amstrad CPC 6128: 2109
Amstrad PCW 8256: 2269
Atari 130XE: 1789
Atari 520ST: 2029
Atari 600XL/800XL: 669, 1171
BBC+: 1929
Coleco Adam: 869
Colour Genie: 789
Commodore Amiga: 2281
Commodore Plus/4: 1189
Commodore 16: 1269
Commodore 64: 49, 490, 712
Dragon 64: 610
Enterprise 64: 1469
Memotech RS128: 1429
Oric Atmos: 749

L

Laboratorios Bell: 420
Lápiz óptico: 156-157
Last One, The: 1876-1877
Latching: 726
LCD: *véase* Visualización en cristal líquido
LED: *véase* Diodo emisor de luz
Leibniz, Gottfried: 260
Lenguaje assembly del 6809: *véase Lenguaje máquina*
Lenguaje assembly del 68000: *véase Lenguaje máquina*
Lenguaje assembly del Z80 y el 6502: *véase Lenguaje máquina*
Lenguaje ensamblador: 464-465, 496, 576-579, 596, 641-643
Lenguajes clásicos: *véase Ciencia informática*
Lenguajes funcionales: 2332-2334
Lenguajes orientados hacia el objeto: 2150-2151
LEO: 320
Ley de Ohm: 594-595, 618
Leyes de Morgan: 526-527
Libros de informática: 1420, 1440, 2374-2375
Líneas de flujo: 553
Link 480Z: 1369-1371
Lisa: 261-263
LISP: 345; *véase Ciencia informática*
Listados de juegos: *véase Software*
Little computer people: 2349
LOGO: 164-165, 345; *véase Ciencia informática*
LOGO (paquetes): 1534-1535
Lords of midnight: 975
Lotus 1-2-3: 1124-1125, 1264-1265, 1361-1362
LSI: *véase* Integración a gran escala
Lunar lander: 832-833
Lynx: 6, 130-131, 740

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Pioneer PX-7: 2069
Sega SC300H: 1009
Sinclair QL: 450, 981
Sinclair Spectrum: 17, 386
Sinclair Spectrum+: 1286
Sony Hit Bit: 1149
Spectravideo SV318: 629
Tatung Einstein: 969
Toshiba HX-10: 1149
Yamaha CX5M: 1509

Otras máquinas

Brother EP-44 (máquina de escribir): 886
Cámaras fotográficas: 1389
DEC VAX 11/780: 2341
Inmos Transputer: 2329
Omni-reader: 1909
Ordenadores de bolsillo: 923, 1021, 1409
Ordenadores ópticos: 1981
Ordenadores portátiles: 821
Psion Organiser: 921
Sintetizadores de voz: 1949
Superordenadores: 1884
Sistemas de video interactivo: 683

Pantallas

Monitor III (Apple): 509
Televisores-pantalla: 809
TM90PSN: 511

Periféricos musicales

Echo Synthesiser: 1821
Music Maker: 1441
Hybrid Music 500: 1570

Periféricos para gráficos

Grafpad: 649
Koala-pad: 1109
Magic Mouse: 1890
Print-Technik: 2121
Ratón AMX: 1530
Touchmaster: 1310
Trazador digital: 889

Otros periféricos

Cámara Snap: 1229
Cassette: 604
Palanca infrarroja Cheetah: 1070
Programador EPROM: 1524
Stack Light Rifle: 710
Unidad de disco: 604

Robots

Beasty: 1250
Brazos-robot: 314, 1321
Fischertechnik Robotics Kit: 1669
Movits: 1564

Sistemas de almacenamiento masivo

Acorn Electron Plus 3: 1549
Amstrad DDI-1: 1689
Atari 810: 543
BBC: 564
Commodore 1540: 532
Dragon: 584
Opus Discovery 1: 1634
Quick Data Drive: 1852
Microdrive Sinclair: 514
Torch Disk Pack: 849
Wafadrive Rotronics: 1609

LL

Llamasoft: 860

M

Macintosh: *véase* Apple Macintosh
MacPaint: 2310-2311
MacProject: 1464-1465
MacWrite: 2232-2234
Maestro (MSX): 1845
Magic Mouse: 1890-1891
Making of the micro, The (libro): 2375
Manchester Mark I: 460
Mando de bola: 8,57
Manejador de archivos: 801
Manic miner: 793
Mano a mano (Oric): 1760
Mapa de memoria: 364-365, 538
Máquina Turing: 424-425
Máquinas educativas: *véase Hardware*
Marketing: 841-843, 2301-2303
Mecanismos de control, Construcción de: *véase Bricolaje*
Mecanografía al tacto: 943-945
Medio sumador: 512-513, 644-645
Melbourne House: 1020
Memoria: 58-59
Memotech: 1060
Memotech MTX 512: 390-391
Memotech RS128: 1429-1431
Método de D'Alembert: 2163, 2165
Micro revolution revisited, The (libro): 1440
Microcosmos: 1543
Microchip (historia): 86-88
Microdrive: 224-225, 514-515, 1898-1900
Micrófono: 595
Micronet 800: 581-583
Microordenador
Historia, 478-480; mantenimiento y diseño, *véase Bricolaje*
MicroPen: 1687-1688
Microprocesador: 2141-2142; historia del, 641-643
Microprocesador 6502: 641-643, 1018
Microprocesador 6809: 1019
Microprocesador Z80: 641-643

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Micros de gestión: *véase Hardware*
Micros portátiles: 166-168; *véase Hardware*
Microsoft: 500
Microsoft Project: 1486-1487
Microwriter: 414-415
MIDI (construcción de una interface); *véase Bricolaje*
Minder: 1780
Mindstorms (libro): 1420
Misión espacial (Atari): 1388
Misterio en el "Dog and Bucket" (juego de personajes interactivos): *véase Programación*
Modem 1000: 2307
Modems: 108, 581-583, 1381-1383, 2306-2308; *véase Hardware*
Módulos: 934-935
Monitor (programa): 662
Motor de inferencias: 1781-1783
Motor paso a paso: 1315-1317
Motorola: 840
Movits (*kits* de robot): 1564-1565
MS-DOS: 2224-2225, 2254-2255, 2272-2273, 2293-2295, 2316-2317
MSX Standard: 621-623, 1171
Muestreo: *véase* Codificación digital del sonido
Mugsy: 1260
Multímetro: 566
Multiplan: 1244-1245
Multiplicación: 119
Multi-User Dungeon: 864-865
Music Maker: 1441-1442
Music system, The: 2100
Música: *véase Aplicaciones*
MUSICOMP: 961-963
My Talking Computer: 1450-1451

N

Namal Type & Talk: 1949-1951
NAND: 646-647
Necromancer: 997
Neumann, John von: 140, 2330
New Brain: 6
Nightingale (modem): 2306
NOR: 646-647, 675
NOT: 68-69, 92-93, 489, 625
Notación polaca inversa: 1955-1957
Nuevo Mundo, El (juego de simulación): *véase Programación*
Numerix (EXL 100): 2189
Números aleatorios: 209, 768
Números binarios: 92-93; 497-498
Números primos: 788

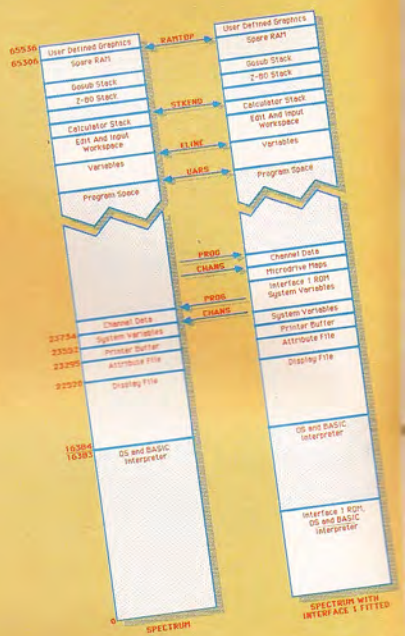
O

Olivetti: 780

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Lenguaje máquina

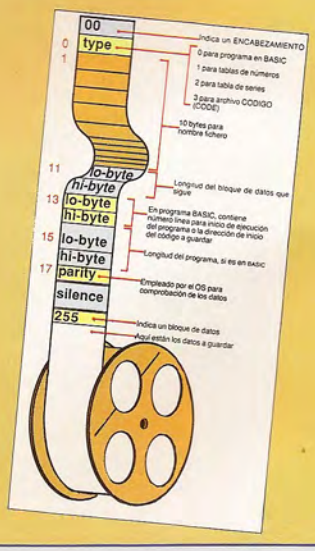
Lenguaje assembly del Z80 y el 6502
Aritmética binaria: 496
Direccionamiento por páginas: 516
Mapas de memoria: 536
Cómo se almacenan los programas en BASIC: 556
Para tratar con la memoria: 576
Desarrollo de un programa: 596
Espacio de memoria: 615
Directrices ensambladoras: 636
Registro indicador de estado: 656
Modalidades de direccionamiento: 676
Etiquetas y bucles: 696
Llamadas a subrutinas: 716
Uso de la pila: 737
Suma y resta: 756
Desplazamiento y rotación: 777
División y visualización: 796
Rutinas de retardo: 958



Aplicaciones para gráficos
Commodore 64: 817, 896, 937, 976
Spectrum: 837, 876
BBC Micro: 857, 918

Lenguaje assembly del 6809
Registros del procesador: 998, 1017
Modalidades de direccionamiento: 1038
Registro de código de condición: 1057
Direccionamiento indexado: 1078
Llamadas a subrutinas: 1098
Direccionamiento indirecto: 1117
Funcionamiento de la pila: 1137
Control de E/S: 1157
Interrupciones: 1177
Codificación reubicable: 1197
Programación estructurada: 1218
Diseño de un programa depurador: 1238, 1257, 1275, 1297, 1318

Sistema operativo del BBC Micro
Empleo de llamadas: 1338
Mapa de memoria: 1359
Utilización de vectores: 1377
Llamadas OSBYTE: 1437
Llamadas OSWORD: 1458



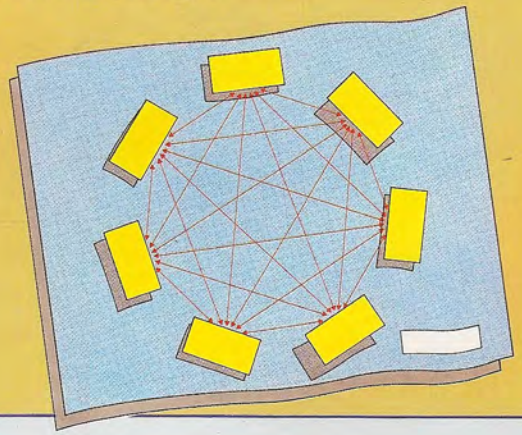
Llamadas OSFILE: 1474
Llamadas OSFIND: 1495
Rutinas BASIC: 1517
Interacción BASIC-OS: 1536
Interrupciones: 1557
Eventos: 1579
Programas de ejemplo: 1597

Sistema operativo del Commodore 64
Punteros, vectores y cuñas: 1617
Entrada/salida: 1637, 1646
Gráficos en color: 1678, 1698
Generación de sonido: 1717
Gráficos tridimensionales: 1736, 1756

Sistema operativo del Sinclair Spectrum
Mapa de memoria: 1776
Canales de E/S: 1797
Flujo de datos: 1812
Sistema de archivo en cinta: 1837
Interrupciones: 1858
Rutinas de la Interface 1: 1878
Códigos de enganche del microdrive: 1898
El enlace RS232: 1918
Códigos de enganche y variables: 1938
Adición de instrucciones BASIC: 1958, 1978
Examen de la pantalla: 1998

Sistema operativo del Amstrad
Mapa de memoria: 2018
Transferencia de archivos: 2038
Temporización: 2058, 2078
Visualización de caracteres: 2096
Generación de sonido: 2118
Creación de nuevas instrucciones: 2137
Gestión del teclado: 2158

Lenguaje assembly del 68000
Generalidades: 2177
Modalidades de direccionamiento: 2195, 2218
Instrucciones: 2238, 2257
Control de programas: 2276
Subrutinas y pilas: 2298
Parámetros: 2318
E/S en serie: 2338
E/S en paralelo: 2357
Operativa del ensamblador: 2376



Omni-reader: 1909-1911
Opcode: *véase* Código de operación
Optoacoplador: 1816
OR: 68-69, 92-93, 489, 546, 625
Ordenador
Arquitectura, 614-617; 2170-2171, 2190-2191, 2212-2214, 2230-2231, 2249-2251; generalidades, 1-4, 8; guía de compra, 226-229; historia, 46-47, 86-88
Ordenador analógico: 238-239
Ordenador del futuro: 466-467; *véase* Conquer Chestnut
Ordenadores de bolsillo: 923
Ordenadores de bolsillo Casio: 1021-1022
Ordenadores de bolsillo Sharp: 1409-1411
Ordenadores en la industria: *véase Aplicaciones*
Ordenadores ópticos: 1981-1983
Ordenadores personales: *véase Hardware*
Ordenadores portátiles: 821-823
Oric-1: 6, 30-31
Oric Atmos: 749-751
Oric Products International: 620
OS: *véase* Sistema operativo
Osborne-1: 410-412
Osborne Encore: 1329-1331
Oscilador: 989

P

Pacman: 1120
Pacman (Oric): 1428
PageMaker: 2062
Página: 516-517
Palanca de mando: 8, 56, 332-333
Palanca infrarroja Cheetah: 1070-1071
Pantallas: 132-133, 509-511, 1600; *véase Hardware*
Paquetes de gestión: 601-603, 632-633
Paquetes educativos: 1386-1387, 1406-1407, 1534-1535
Paracaídas: Commodore 64, 1156; MO5 de Thomson, 2268
Pared, La (Dragon): 1196
Paridad, Control de: 253
PASCAL: 344; *véase Ciencia informática*
Patrones bidimensionales: 1414-1416
Patrones de franjas: 1395-1397
Patrones geométricos: 704-705, 1366-1368
Peddle, Chuck: 180, 599
PEEK: 188
Penman Plotter: 1649-1651
Periféricos: 1209-1211; *véase Hardware*
Persecución: Alice, 1508; Dragon, 1840; MSX, 2140; MO5 de Thomson, 2149
Picturesque Editor/Assembler: 2355
Pila (eléctrica): 595
Pila (zona de la memoria): 617, 737-739, 1137-1139
PIO: 2212-2213
Pioneer PX-7: 2069-2071
Piratería de software: 192-193
Plotter: *véase* Trazador de gráficos
Poesía generada por ordenador: 1235-1236, 1256
POKE: 188
Powers, James: 380
Practicalc II: 1366-1367
Prestel: 612-613, 1443-1445

Print-Technik (digitalizador de video): 2121-2122
Prism: 960
Problemas de programación sencillos: *véase Programación*
Procesador de instrucciones de consola: 1804-1805
Producción de software: 861-863
Programa Alvey: 1773
Programa de terapia: 892-893
Programación de un juego de aventuras: *véase Programación*
Programación de un juego de estrategia: *véase Programación*
Programación de un juego de naipes (veintiuno o *pontoon*): *véase Programación*
Programación de un juego de personajes interactivos: *véase Programación*
Programación de un juego de simulación: *véase Programación*
Programación de una hoja electrónica: *véase Programación*
Programación *top-down:* 956-957
Programador EPROM: 1524-1525
Programadores de juegos: 1030-1032, 2350-2351
Programas de ajedrez: 781-783
Programas de bridge: 924-925
Programas de gestión: 601-603
Programas heurísticos: 732-733
Programas para correspondencia: 806-807
PROLOG: *véase Ciencia informática*
Protocolos: 1401-1402
Prueba de rutinas: 1086-1087
Pruebas de memoria: 852-853
Pseudo-op: 636-637
Psion: 1000
Psion Organiser: 921-923
PSR: *véase* Registro indicador de estado
Psytron: 1200
Puerta: 92-93
Puerta para el usuario: 994-996, 1003-1005

Q

Quick Data Drive: 1852-1853
Quicksilva: 760
Quill, The: 522
Quinta generación: 64, 468-469
Quo vadis: 2260

R

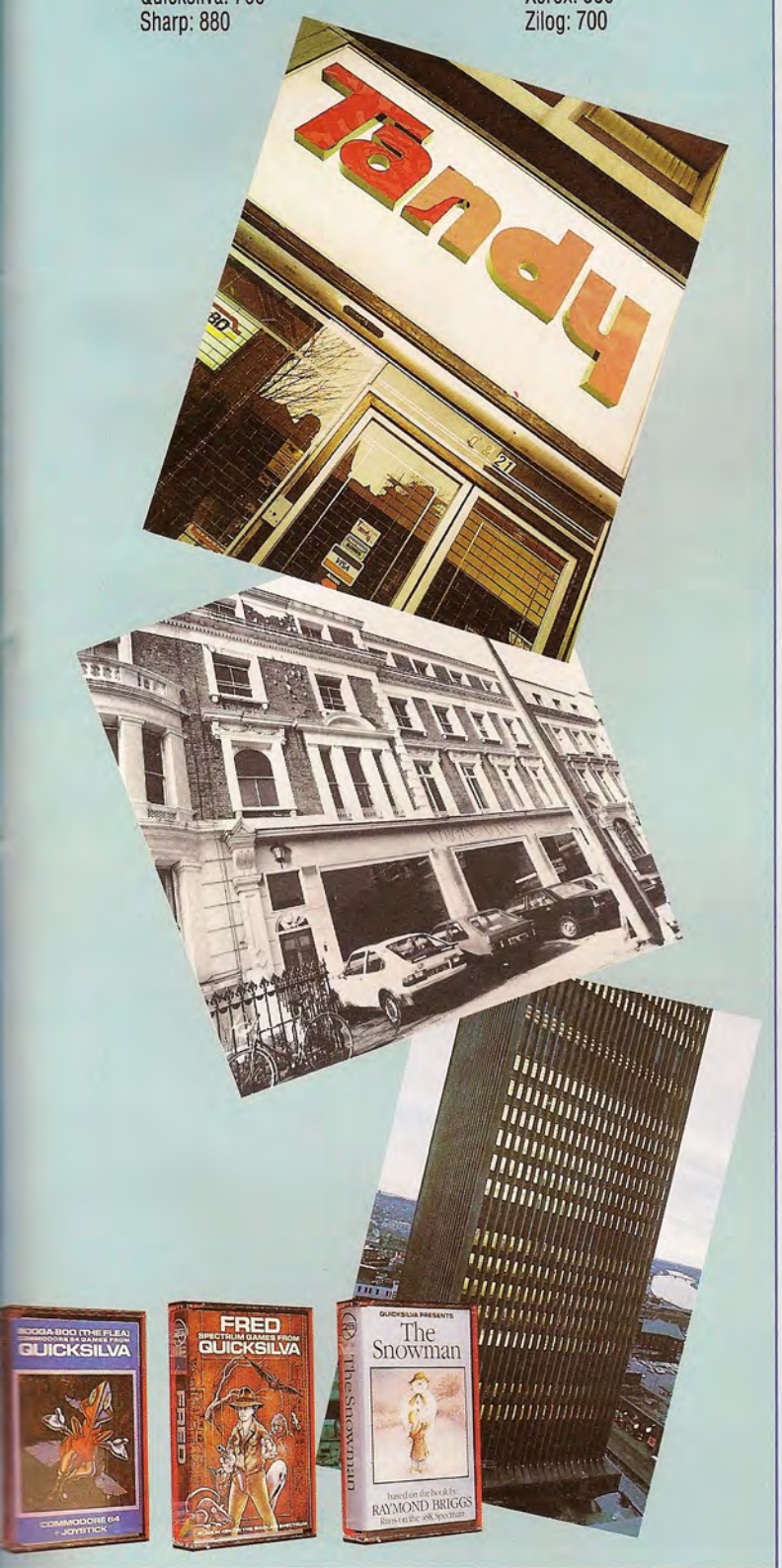
RAM: 96-97, 2170-2171
RAM dinámica: 2190-2191
Ratón: 296-297
Ratón AMX: 1530-1531
Ratón-robot: 1121-1123

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Nombres propios

Acorn: 540
Artic Computing: 900
Atari: 519
Audiogenic: 940
Bug-Byte: 820
Camputers: 740
Casio: 1040
Commodore: 599
Digital Research: 719
Dragon Data: 800
Imagine: 559
Llamasoft: 860
Melbourne House: 1020
Memotech: 1060
Microsoft: 500

Motorola: 840
Olivetti: 780
Oric: 620
Prism: 960
Psion: 1000
Quicksilva: 760
Sharp: 880
Softsel: 980
Sord: 639
Tandy: 679
Texas Instruments: 920
Virgin Games: 660
Xerox: 580
Zilog: 700



Recogedor (MSX): 2108
Reconocimiento de patrones: 1861-1863, 1881-1883
Reconocimiento de voz: 446-447, 1243, 1841-1843
Red de área local: 218-219
Redes: 801-803, 826-827
Redes semánticas: 1941-1943
Registro (*register*): 596, 615-617, 709, 756-759
Registro indicador de estado: 656-657
Registros de datos: 706, 725, 774-775
Registros de datos (en PASCAL): 1614-1616
Regla de Bayes: 2152-2153
Reinventing man (libro): 2375
Relé de red: 1279
Research Machines 380Z: 470-471
Resistencias: 595, 619
RGB (*red, green, blue*: rojo, verde, azul): 809-811
River rescue: 953
RM 480Z: 1369-1371
RM Nimbus: 1808-1811
Robot book, The: 2374
Robot industrial: 281-283
Robot móvil: 176-178; construcción de un, *véase Bricolaje*
Robótica: *véase Aplicaciones* y *Bricolaje* (Construcción de un brazo-robot y Construcción de un robot móvil)
Robots: *véase Hardware*
Robots: Commodore 64, 1629; Thomson MO5, 1732; Thomson TO7, 1332
Rockford's riot: 1960
ROM: 96-97, 2170-2171
ROM en disco compacto: 1662-1663
Rotación: 777-779
RS232C: 801-803
Rutinas de biblioteca: 1046-1047
Rutinas de llamada ROM: 796-799
Rutinas de retardo: 958-959

S

Sabre Wulf: 913
Samna Word III: 2246-2247
Sanyo MBC-550: 1589-1591
Scuba Dive: 755
Sección dorada: 1555
Sector: 604-605
Sectorización hard: 604-605
Sectorización soft: 604-605
Sega SC3000H: 1009-1011
Sensores: 394-395, 1162
Sentencia CASE: 1526-1528
Sentencia IF: 1526-1528
Serpiente: Commodore 64, 1708; Thomson MO5, 1600
Servomotores: 1392-1394, 1403-1405
Shadow of the unicorn: 2287
Shadowfire: 1800
Sharp Corporation: 880
Sharp MZ-711: 309-311
Sharp PC-1251: 1409-1411
Sharp PC-1500A: 1409-1411
Sharp PC-5000: 690-691
Silicio: 121-123
Silicon Valley: 64, 901
Símbolo de decisión: 808
Símbolos auxiliares: 553

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Programación

Principios de programación BASIC

Curso de programación BASIC: 20, 37, 52, 77, 98, 116, 134, 146, 172, 194, 212, 232, 254, 272, 292, 316, 336, 354, 376, 396, 416, 436, 456, 474
BASIC Sinclair: 494, 504
BASIC BBC: 534
BASIC Commodore: 574
Funciones trigonométricas: 634, 654

Problemas de programación sencillos

Patrones geométricos: 704
Programa heurístico: 732
Juegos sonoros: 754
Cuadrados mágicos: 766
El camionero del desierto: 792
Gráficos tridimensionales: 812, 932
Lunar lander: 832
Pruebas de memoria: 852
Juego *Invertir*: 879
Programa de terapia: 892
Comprobación de reacciones: 917
Las torres de Hanoi: 954
Juego del *Ahorcado*: 984

Gráficos en programación de juegos

Subhunter (para Commodore 64): 694, 712, 734, 744, 764, 794
Batalla naval (para una red Spectrum): 826
Campo de minas (para BBC Micro y Acorn Electron): 872, 884, 914, 926, 946, 972
En su moto (para Commodore 64, BBC Micro, Spectrum): 1112, 1133

Utilidades avanzadas de programación

Tratamiento de archivos: 664, 684, 706, 724, 752, 774
Generadores de caracteres definidos por el usuario: 1052, 1068, 1096
Búsqueda y sustitución de variables: 1144, 1180, 1184, 1206, 1224
Técnicas de compresión de textos: 2314, 2335, 2346, 2363

Programación de un juego de aventuras

Para Commodore 64, Spectrum y BBC Micro: 1246, 1272, 1293, 1306, 1323, 1346, 1363, 1384, 1412, 1432, 1446
El bosque encantado (listado completo): 1477
Digitaya (listado completo): 1497

Programación de un juego de estrategia

Para Commodore 64, Spectrum, BBC Micro y Amstrad: 1846, 1873, 1886, 1905, 1934, 1952, 1975, 1992, 2012, 2026
Juego de *go* (listado completo): 1874, 1886, 1906, 1934, 1952, 1976, 1994, 2012

Programación de un juego de naipes (veintiuno o *pontoon*)

Para Commodore 64, Spectrum, BBC Micro y Amstrad: 2192, 2209, 2226, 2252, 2266, 2284

Programación de un juego de personajes interactivos

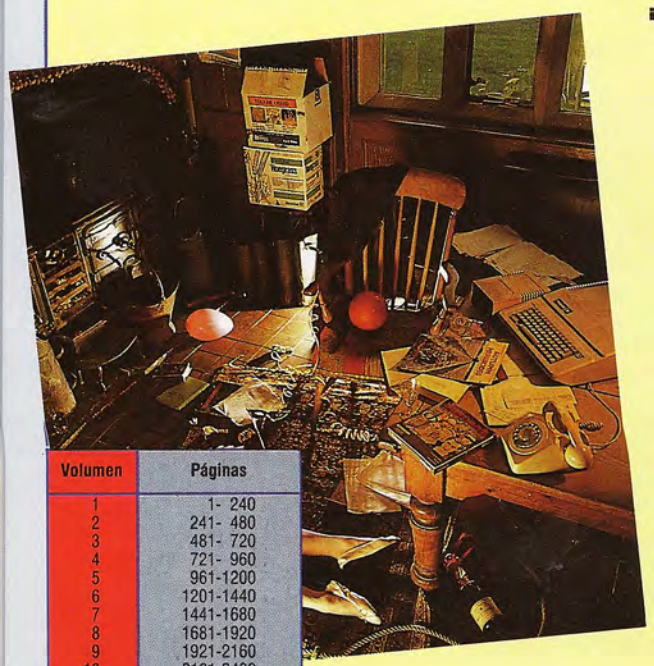
Para Commodore 64, Spectrum, BBC Micro y Amstrad: 1921, 1946, 1972, 1986, 2006, 2024, 2055, 2076, 2084, 2104, 2126
Misterio en el "Dog and Bucket" (listado completo): 2154

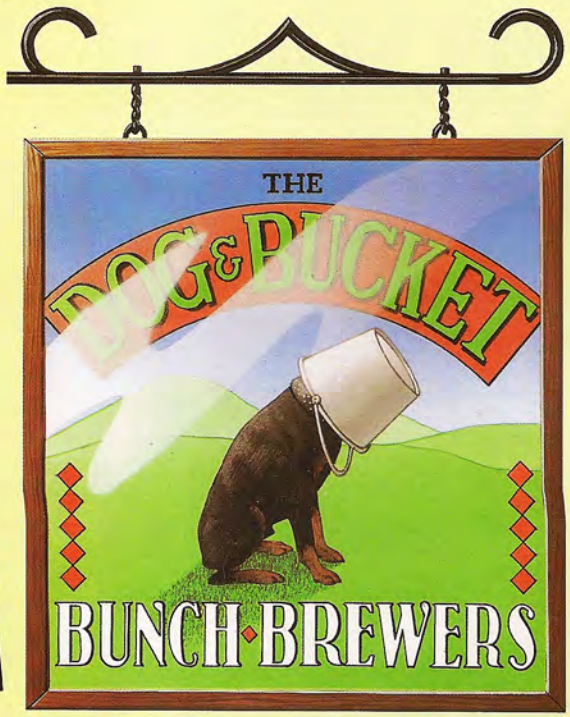
Programación de un juego de simulación

Para Commodore 64, Spectrum y BBC Micro: 1506, 1532, 1546, 1566, 1586, 1612, 1624, 1652, 1672, 1692, 1712, 1724, 1746, 1774
El Nuevo Mundo (listado completo): 1794, 1818, 1834

Programación de una hoja electrónica

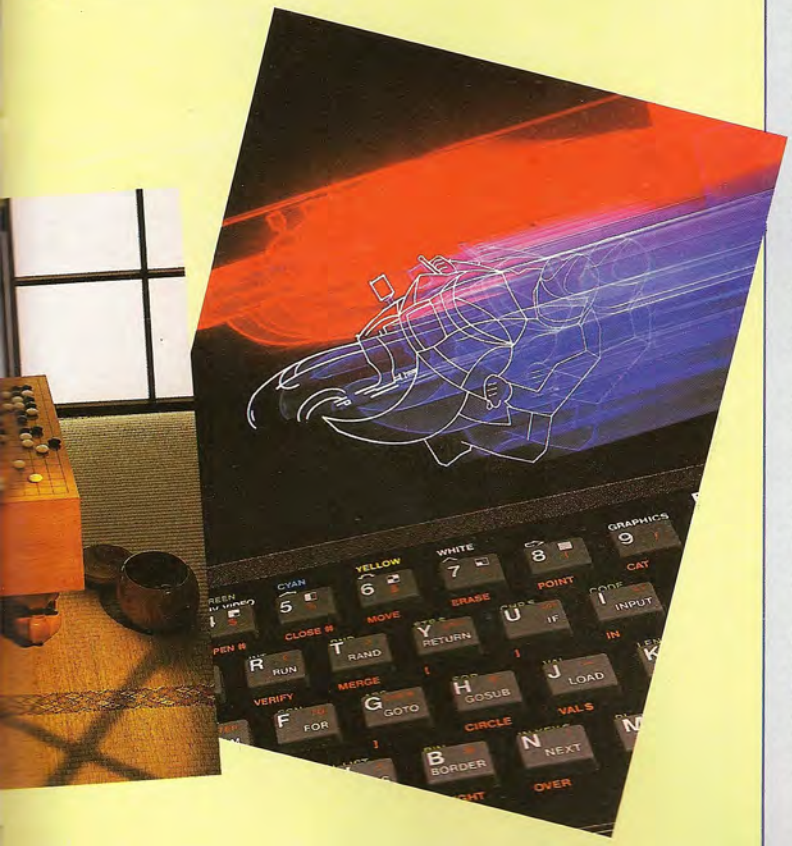
Para Commodore 64, Spectrum, BBC Micro y Amstrad: 2044, 2072, 2088, 2106, 2132, 2146, 2166





Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Programación



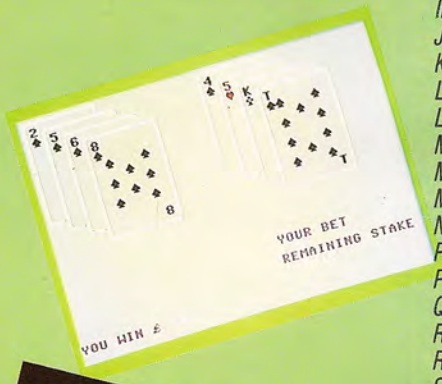
Técnicas de progr.

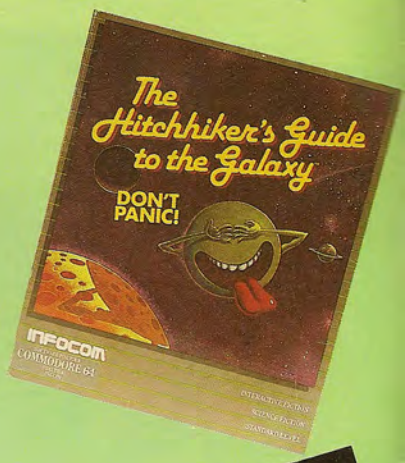
Introducción: 814
Documentación: 834
Análisis de sistemas: 854
Algoritmos: 866
Creación de bucles: 894
Tablas de decisión: 904
Módulos: 934
Programación *top-down:* 956
Tratamiento de errores: 964
Interface hombre-máquina: 992
Facilidades de ayuda: 1006
Sistemas de menús y de comandos: 1036
Rutinas de biblioteca: 1046
Aceleración del BASIC: 1076
Prueba de rutinas: 1086

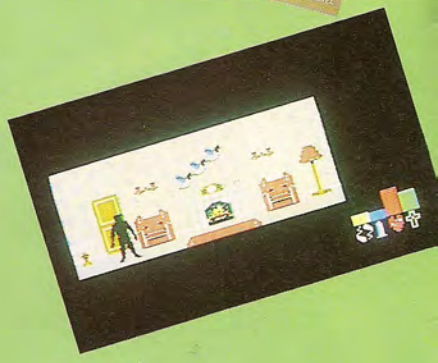
Símbolos de proceso: 553
Símbolos de sistema: 553
Simulación: 267-268, 366-367, 389
Simulador de vuelo: 201-203
Sinclair QL: 450-451, 981-983, 1170
Sinclair Spectrum: 7, 17-19, 386-387, 1170, 2249; sistema operativo, *véase Lenguaje máquina;* teclado, 846-847
Sinclair Spectrum+: 1286-1287
Sinclair ZX81: 7, 210-211, 326-327
Síntesis de voz: 186, 1242-1243
Sintetizadores (música): 141-143
Sintetizadores de voz: 1949-1951
Sistema básico de entrada y salida: 1804-1805
Sistema binario: 54-55, 79-80
Sistema de comandos: 1036-1037
Sistema de menús: 1036-1037
Sistema hexadecimal: 179, 536-537
Sistema operativo: 324-325, 661-663, 1648; *véase Software*
Sistema operativo básico de disco: 1804-1805
Sistemas de acceso público: 1443-1445
Sistemas de almacenamiento masivo: *véase Hardware*
Sistemas expertos: 561-563, 1701-1703, 1781-1783
Sistemas WIMP: 2150-2151, 2161-2162, 2184-2185, 2206-2207
Slalom (Dragon): 1249
Slumbering sentinels, The (libro): 2375
Snooker: 895
Snooker (en PASCAL): 1595, 1615
SoftAid: 2280
Softsel: 980
Software educativo: 541-542
Software integrado: 1106-1107, 1124-1125, 1152-1153
Software vertical: 1324-1325, 1426-1427
Soldaduras: 524-525
Sony Hit Bit: 1149
Sord: 639-640
Sord M5: 7, 250-252
Soul of a new machine, The (libro): 1420
Space invaders: 1095
Spectravideo 318: 629-631

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

Software







Análisis de aplicaciones
- *Abacus:* 1204
- *Acornsoft:* 1534
- *Brainstorm:* 1344
- *dBase II:* 1632
- *EasyScript:* 2215
- *Farmfax:* 1544
- *Graph Plan:* 1232
- *HULK:* 561, 2083
- *Human edge, The:* 1426
- *Logotron:* 1534
- *Lotus 1-2-3:* 1124, 1264
- *Microsoft Project:* 1486
- *Music system, The:* 2100
- *Practicalc:* 1366
- *Samna Word III:* 2246
- *Sycero:* 1876
- *Symphony:* 1124, 1361
- *Tasword II:* 2216
- *TK!Solver:* 1284, 1304
- *View:* 554, 2216
- *Vizastar:* 1366
- *"Word" processor, The:* 1512
- *Wordstar:* 506, 2198, 2216
- *Wordwise:* 554

Análisis de juegos
- *Ant attack:* 486
- *Apocalypse:* 836
- *Archon:* 1740
- *Atic-Atac:* 856
- *Boulder Dash:* 1960
- *Bugaboo (La pulga):* 776
- *Classic racing:* 1080
- *Comando de misiles:* 1140
- *Deus ex machina:* 1220
- *Elite:* 1520
- *Flyerfox:* 1240
- *Football manager:* 1980
- *Frankie goes to Hollywood:* 1995
- *Ghostbusters:* 1480
- *Hitchhiker's guide to the galaxy, The:* 1880
- *Impossible mission:* 1700
- *Jet Pac:* 875
- *Knight Lore:* 1680
- *Little computer people:* 2349
- *Lords of midnight:* 975
- *Manic miner:* 793
- *Minder:* 1780
- *Mugsy:* 1260
- *Necromancer:* 997
- *Pacman:* 1120
- *Psytron:* 1200
- *Quo vadis:* 2260
- *River rescue:* 953
- *Rockford's riot:* 1960
- *Sabre Wulf:* 913
- *Scuba Dive:* 755
- *Shadow of the unicorn:* 2287
- *Shadowfire:* 1800
- *Snooker:* 895
- *Space invaders:* 1095
- *Star raiders:* 1160
- *Starfinder:* 1300
- *Summer games:* 1340
- *Twin kingdom valley:* 816
- *Valhalla:* 936
- *Zaxxon:* 1560

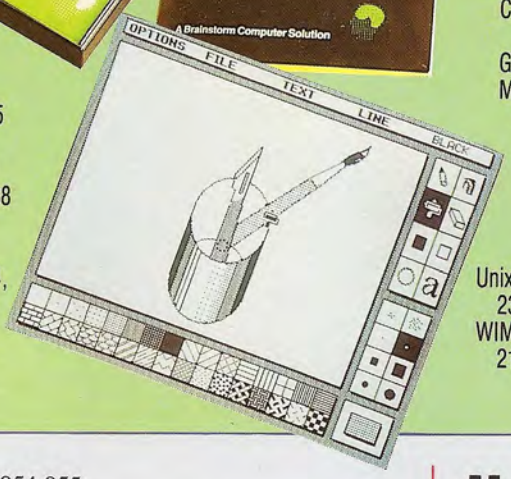
Spectrum: *véase* Sinclair Spectrum
Spooling: 1177
Sprites: 152-154, 837-839, 857-859, 2231
Squash: Commodore 64, 1820; Thomson TO7, 1237
Stack Light Rifle: 710-711
Star raiders: 1160
Starfinder: 1300
Sumador: 92-93
Sumador completo: 527
Summer games: 1340
Superordenadores: 1884-1885
Supersoft Mikro Assembler: 2368
Suspect: 1921-1923
Switch (indicador): 1029
Sycero: 1876-1877
Symphony: 1124-1125, 1361-1362

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400

T

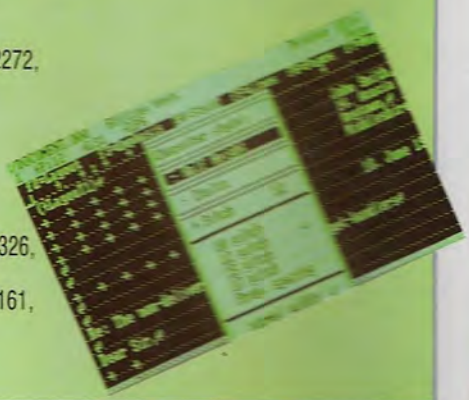
Tablas de decisión: 904-905
Tablas de verdad: 68-69
Tablones de anuncios: 306-307, 1461-1463
Tandy 1000: 1729-1731
Tandy Color: 270-271
Tandy Corporation: 679-680
Tandy MC-10: 330-331
Tandy Modelo 100: 1130-1132
Tasword II: 2216
Tatung Einstein: 969-971
Teach yourself geometry (libro): 2375
Telecom Gold: 1443-1445
Teletexto: 2222
Televisión por cable: 301-303
Televisores-pantalla: 809-811
Teoría de las probabilidades: 2129
Test en cascada: 828, 848, 868
Tester de circuitos: 566
Tester digital, Construcción de un: *véase Bricolaje*
Tests visuales de Ishihara: 1222
Texas Instruments: 920
TI 99/4A: 7, 189-191
TK!Solver: 1284-1285, 1304-1305
Torch Disk Pack: 849, 851

Listados de juegos
Alien (Atari): 2309
Alunizaje (Commodore 64): 1548
Aterrizaje (EXL 100): 2300
Autopista (Alice): 1788
Batalla naval (ZX81): 1668
Bombardeo aéreo: Alice, 1468; Dragon, 1608
Cacería de patos (Commodore 64): 2328
Cangrejos: Commodore 64, 1296; MSX, 2229; Oric Atmos, 1168
Ciempiés (Vic-20): 2080
Cifras (Alice): 1569
Cuatro en raya (Atari): 1648
Defensa antiaérea (C64): 2248
En la luna (Atari): 1769
Exocet: EXL 100, 2208; Vic-20, 1274
Gran Premio: Atari, 1408; MSX, 2169
Gran Premio 2 (Atari): 1695
Maestro (MSX): 1845
Mano a mano (Oric): 1760
Misión espacial (Atari): 1388
Numerix (EXL 100): 2189
Pacman (Oric): 1428
Paracaídas: Commodore 64, 1156; MO5 de Thomson, 2268
Pared, La (Dragon): 1196



Persecución: Alice, 1508; Dragon, 1840; MSX, 2140; MO5 de Thomson, 2149
Recogedor (MSX): 2108
Robots: Commodore 64, 1629; Thomson MO5, 1732; Thomson TO7, 1332
Serpiente: C64, 1708; Thomson MO5, 1600
Slalom (Dragon): 1249
Squash: Commodore 64, 1820; Thomson TO7, 1237
Trampa, La (Vic-20): 1309
Trazos: Commodore 64, 2360; Dragon, 1208; MO5 y TO7 de Thomson, 1352

Principales sistemas operativos
CP/M: 719, 1744, 1764, 1784, 1804, 1826
GEM: 2194, 2206
MS-DOS: 2224, 2254, 2272, 2293, 2316



Unix: 2264, 2288, 2304, 2326, 2344, 2366
WIMP (sistemas): 2150, 2161, 2184, 2206

Torres de Hanoi, Las: 954-955
Torres Quevedo: 360
Tortuga: 1012
Toshiba HX-10: 1149-1151
Touchmaster (dispositivo para gráficos): 1310-1311
Traducción de software: 1361-1362
Tramiel, Jack: 599
Trampa, La (Vic-20): 1309
Transductores: 1163
Transfasor: 1982
Transferencia de datos: 746-747
Transistor: 595, 619, 624
Transputer: 2329-2331
Tratamiento de archivos: 664-665, 706-707
Tratamiento de textos: 11, 61-63, 741-743, 2175-2176
Tratamiento de textos (programas): 2215-2217, 2246-2247
Trazador de gráficos: 198-199
Trazador digital, Construcción de un: *véase Bricolaje*
Trazos: Commodore 64, 2360; Dragon, 1208; MO5 y TO7 de Thomson, 1352
Tres en raya: 1749
Turing, Alan: 200
Twin kingdom valley: 816

U

ULA: *véase* Disposición lógica no comprometida
Unidad aritmética lógica: 616-617, 772-773
Unidad central de proceso: 138-139, 144-145, 496-497, 615-617, 641-643, 746-747
Unidad de disco: 8, 604
Unidad de representación visual: 2309
Unix: 2264-2265, 2288-2289, 2304-2305, 2326-2327, 2344-2345, 2366-2367
Utilidades avanzadas de programación: *véase Programación*

V

Valhalla: 936
VDU: *véase* Unidad de representación visual
Vector: 1377
Veintiuno o *pontoon* (juego de naipes): *véase Programación*
Ventas de ordenadores: 2301-2303
Verificación: 814

Verificadores de ortografía: 404-405
Vic-20: *véase* Commodore Vic-20
Video compuesto: 811
Video interactivo: 683
Videotex: 264-265, 612-614, 2221-2223
View: 554-555, 2216
Vigilancia por ordenador: 941-942
Virgin Games: 660
Visualización en cristal líquido: 278-279
Visualización en siete segmentos: 686-687, 1165-1167, 1280
Vizastar: 1366-1367
VLSI: *véase* Integración a muy gran escala
Voisin, Frédéric (pintor): 2310-2311
VTX 5000 (modem): 906-907, 2307

W

Wafadrive Rotronics: 1609-1611
Watford ROMAS (paquete ensamblador): 2356
Wiener, Norbert: 300
WIMP (sistemas): 2150-2151, 2161-2162, 2184-2185, 2206-2207
Women and computing (libro): 1440
"Word" processor, The: 1512-1513
WordStar: 506-507, 2198-2200, 2216
Wordwise: 554-555
Wozniak, Steve: 155
Wren Executive: 1709-1711
Writing software for profit (libro): 2375
WS3000 (modem): 2308

X

Xerox Parc: 580

Y

Yamaha CX5M: 1509-1511
Yamaha DX7: 1041

Z

Z80: 576-579, 641-643
Zaxxon: 1560
Zilog Incorporated: 700
Zuse, Konrad: 340
ZX81: *véase* Sinclair ZX81

Volumen	Páginas
1	1- 240
2	241- 480
3	481- 720
4	721- 960
5	961-1200
6	1201-1440
7	1441-1680
8	1681-1920
9	1921-2160
10	2161-2400