mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

TOMO 4

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
	Francisco Martín
Asesor técnico:	Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, A. Cuevas

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

ISBN: 84-7598-005-8 (tomo 4) 84-85822-82-X (obra completa)
Depósito legal: B. 52/1984
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona)
Impreso en España - Printed in Spain - 1984

mi COMPUTER

VOLUMEN 4

Editorial Delta, S.A.

Denominador común

El BASICODE proporciona un nuevo procedimiento para solucionar el eterno problema de la portabilidad del software para micros

Ian McKinnell

Estándar común
El BASICODE permite que los micros se comuniquen entre sí a través de un estándar común. Utiliza un juego mínimo de instrucciones de BASIC y su propio formato de cinta para permitir que alrededor de una docena de micros puedan intercambiar sus programas. Los programas en BASICODE se transmiten incluso por emisoras de radio, posibilitando que los oyentes que poseen micros diferentes puedan ejecutar los mismos programas

En la actualidad el BASIC ya está firmemente establecido como el lenguaje estándar para micros personales. Sin embargo, como sabe todo usuario de ordenadores personales, existen enormes diferencias entre las versiones disponibles en el mercado. Aun en el caso de máquinas que compartan una versión común, como el BASIC Microsoft, no existe ninguna garantía de que un programa escrito en un tipo de ordenador funcione necesariamente en un modelo diferente.

Actualmente existen algunos indicios de que esta situación va a cambiar. A principios de 1984, el programa *Chip shop* de la BBC Radio empezó a transmitir programas en una versión única de BASIC denominada BASICODE; tales programas se vienen cargando y ejecutando con éxito en una amplia variedad de ordenadores personales.

El BASICODE es un nuevo enfoque para abordar el programa de la compatibilidad. Se desarrolló por primera vez en los Países Bajos para utilizarlo en *Hobbyscoop*, un programa de ciencia y tecnología producido por Teleac, una especie de Universidad a Distancia que imparte clases por televisión. Cuando *Hobbyscoop* empezó sus emisiones, en 1978, el espacio basaba sus transmisiones en las cuatro máquinas más populares en aquel entonces: el Apple, el Exidy Sorcerer, el Commodore PET y el Tandy TRS-80. Sólo podía haber una transmisión por semana para cada máquina y, como dos de estos ordenadores tenían velocidades de transmisión sumamente lentas, los oyentes tenían que soportar hasta ocho minutos de chirridos. Evidentemente este estado de cosas no era satisfactorio y a medida que iban saliendo al mercado nuevas máquinas, cada una de las cuales exigía su propia emisión, este método de transmisión de programas se volvió prácticamente inviable.

El problema fue abordado por primera vez por un aficionado entusiasta de la radio llamado Klaas Robers, que produjo la primera versión de BASICODE. Éste se basaba en un subconjunto común de instrucciones de BASIC, que resultaba comprensible a todo tipo de ordenadores. No obstante, persistían problemas de interpretación, ya que a pesar de que los distintos tipos de máquinas tenían instrucciones idénticas, cada una de ellas las ejecutaba de forma diferente y por ese motivo no se consiguió la estandarización. En consecuencia, Klaas Robers desarrolló, junto con Jochem Herrman, una versión mejorada del lenguaje, a la que denominaron BASICODE-2.

Las primeras emisiones de BASICODE-2 tuvieron lugar el día de Año Nuevo de 1983 y pronto demostró ser un éxito. Oyentes de lugares tan lejanos como Bélgica, Francia, Gran Bretaña, Alemania y Dinamarca informaron del éxito obtenido al cargar

los programas. Esta atención internacional aumentó cuando el servicio de transmisión holandés empezó a emitir BASICODE-2 por su red externa.

El BASICODE se basa en las 42 palabras clave y 11 símbolos que poseen en común la mayoría de las máquinas que utilizan el lenguaje. Una palabra clave de BASIC no se almacena como los caracteres que la componen sino que se almacena en forma de un código identificativo de un solo byte, cuyo valor representa la palabra clave. Por ejemplo, la palabra clave LEFT$ en el Commodore 64 se almacena en un único byte que contiene el valor 200, en vez de ocupar cinco bytes que contengan los valores ASCII para los caracteres L, E, F, T y $. Esto hace que el trabajo del intérprete de BASIC sea mucho más eficiente y emplee mucha menos memoria RAM. No obstante, a pesar de que todos los ordenadores utilizan este sistema de códigos identificativos para almacenar e interpretar un programa BASIC, cada máquina emplea códigos distintos para sus palabras clave. El problema se resolvió suministrando dos programas de traducción, el BASICODE-Save y el BASICODE-Load. Después de escribir un programa en BASIC, se guarda (SAVE) utilizando el programa BASICODE-Save, que sustituye los códigos estándar de BASICODE por los que utiliza el propio ordenador, almacenando en cinta un programa BASICODE estándar. De esta forma, el programa se puede cargar en otra máquina empleando el programa BASICODE-Load, que sustituye los códigos de las palabras clave de BASIC por la versión BASICODE.

Esto plantea una cuestión trascendental: cómo asegurar que los diversos tipos de ordenador lean y escriban en cinta de la misma forma. Nuevamente, a pesar de que las máquinas utilizan el mismo principio para cargar (LOAD) y guardar (SAVE) programas en cinta, en la práctica una cinta de programas producida por un ordenador puede muy bien ser diferente de la cinta producida por otra máquina. No sólo los datos se pueden escribir y leer en la cinta a distintas velocidades de transmisión, sino que también los bits de comienzo y de final (los indicadores que le dicen al ordenador dónde empiezan y terminan los datos), y los métodos de control de paridad (el sistema por el cual la máquina verifica que los datos se han transmitido correctamente), podrían asimismo diferir de forma radical. La solución adoptada fue suprimir los métodos de manipulación de cinta particulares de cada máquina e imponer un formato de código de audio, o audiocódigo, común para la transmisión.

En este formato los datos se transmiten a 1 200 bits por segundo. Cada byte de datos se precede por un bit de comienzo (valor 0) y se transmite empezando por el bit menos significativo y se termina con dos bits de final (ambos con un valor de 1). Por ejemplo, el valor ASCII de "A" es 65 (01000001 en binario) y en audiocódigo éste se transmitiría como 01000001011. Una marca especial de inicio, compuesta por una secuencia de bits de final transmitida durante cinco segundos, indica el comienzo de un programa BASICODE. A ésta le sigue el código de *principio de texto* (82 en hexadecimal). El programa BASICODE va seguido por un byte de control de paridad, que permite al ordenador verificar la exactitud de los datos transmitidos. Otra secuencia de cinco segundos de bits de final indica el fin de la transmisión de datos.

Aunque casi todas las máquinas se pueden adaptar a BASICODE por software, los modelos I y III del TRS-80 y el Video Genie requieren el uso de una pequeña interface con el fin de que las cintas se puedan leer correctamente. El manual suministrado con la cassette de BASICODE-2 da toda clase de detalles acerca de cómo construir la interface. A quien no desee construírsela él mismo, el grupo de usuarios de TRS-80 de los Países Bajos le puede proporcionar la placa de circuito impreso necesaria.

Para poder escribir un programa en BASICODE, primero se debe cargar el programa BASICODE-Save. Este programa no sólo permite guardar el código recién escrito en cassette de forma estándar, sino que proporciona además una lista de subrutinas exclusivas para esa máquina en particular. Estas rutinas están entre las líneas 0 y 999 y, por tanto, no se hallan disponibles para el programador.

La razón por la que el programa de traducción BASICODE-2 proporciona estas rutinas es porque una instrucción común a varias máquinas (como CLS, la instrucción para limpiar la pantalla) se podría ejecutar de distintas maneras. En lugar de utilizar la instrucción CLS, el programador emplea GOSUB 100, que hace referencia a la subrutina en BASICODE que realiza esta función.

La primera línea escrita por el programador deberá ser de la forma siguiente:

```
1000 A = (valor): GOTO 20: REM nombre del programa
```

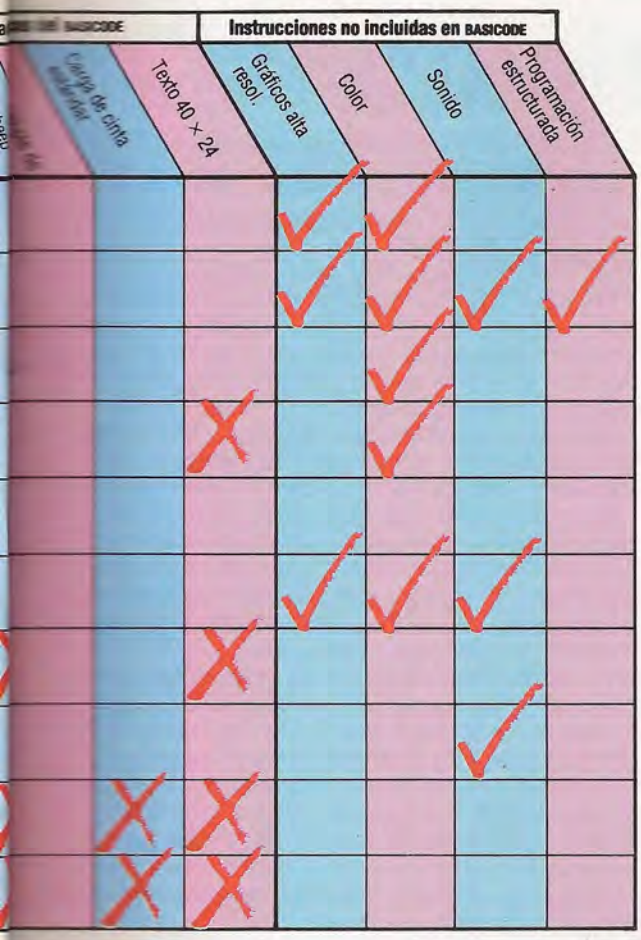
donde (valor) es el número máximo de caracteres que utilizan todas las series juntas. De ahí en adelante, el usuario tiene libertad para programar

Este diagrama muestra cómo se adapta al estándar cada una de las máquinas a las que va dirigido el BASICODE. Aquellos micros cuyas especificaciones no responden al estándar están marcados con una cruz. Las máquinas marcadas con un trazo poseen características que superan lo permitido por el BASICODE

Instrucciones de operación

Ésta es una lista de las instrucciones y operaciones permitidas en BASICODE. Observe que muchas máquinas tendrán gran cantidad de palabras clave que el BASICODE no reconoce

ABS	NEXT
AND	NOT
ASC	ON
ATN	OR
CHR$	PRINT
COS	READ
DATA	REM
DIM	RESTORE
END	RETURN
EXP	RIGHT$
FOR	RUN
GOSUB	SGN
GOTO	SIN
IF	SQR
INPUT	STEP
INT	STOP
LEFT$	TAB
LEN	TAN
LET	THEN
LOG	TO
MID$	VAL
+	<
-	>
*	<>
/	<=
^	>=



como desee. Existen, sin embargo, ciertas restricciones impuestas por el formato del código. Por ejemplo, las variables se deben inicializar antes de realizar alguna operación con ellas, de modo que, por ejemplo, antes de ejecutar la orden LET T = T + 1, se debe poner T a cero.

Existen también limitaciones en cuanto al uso de varias palabras clave de BASIC. Por ejemplo:

```
5000 INPUT "CONTRASEÑA?";A$
```

está prohibido en BASICODE-2. El formato correcto de esta línea es:

```
5000 PRINT "CONTRASEÑA?": INPUT A$
```

Además, la longitud de una línea de programa no puede exceder los 60 caracteres y se asume que el tamaño de la pantalla es de 24 líneas de 40 caracteres.

Llegados a este punto vale la pena considerar el porqué de estas restricciones. A fin de incluir dentro del espectro del BASICODE la mayor cantidad posible de máquinas, el diseño se enfocó desde el punto de vista del "denominador común más bajo". Inevitablemente, hubo una compensación entre la sofisticación del BASICODE y el número de ordenadores capaces de hacer uso del mismo. Esto determinó un cierto "recorte" de sus posibilidades para acercarse a las máquinas menos potentes, lo cual se convirtió en limitaciones a las que hubo que someter a los modelos más avanzados.

Por ejemplo, un cierto número de características que el usuario de un micro personal tendría en cuenta a la hora de comprar un ordenador no están contempladas en el formato del BASICODE. El sistema no tiene incorporada ninguna facilidad para variar el tono y la duración de los sonidos. En este sentido sólo se puede trabajar con la instrucción BEEP, que es más bien primitiva. De modo similar, el BASICODE sólo permite que el programador realice gráficos en la modalidad de baja resolución. Y aun entonces éstos sólo se pueden programar en blanco y negro.

Otro problema es que desde que se inventó el BASICODE se ha producido un enorme adelanto en el desarrollo de las técnicas de programación estructurada del BASIC. El BASICODE no admite sentencias tales como WHILE... WEND, ni siquiera una instrucción DEF FN. La estructuración del programa se deja en manos de la instrucción GOSUB, de la que depende en gran medida el diseño de éste.

Por otra parte, es importante destacar que a pesar de estas restricciones, algunas de las máquinas soportadas por el BASICODE son incapaces de cumplir siquiera los estándares establecidos más modestos. Por ejemplo, el Vic-20, el ZX81, el TRS-80 y el Video Genie tienen visualizaciones inferiores a la estándar de 40×24 caracteres.

No obstante, para el programador aplicado la programación en BASICODE debería ser un desafío. Debido a que las reglas son restrictivas, el programador ha de tener un gran cuidado, asegurándose de que el programa escrito sea portable. El programador debe ceñirse a las aproximadamente 50 palabras clave y operadores que utiliza el lenguaje y usar instrucciones GOSUB para reemplazar las instrucciones no estandarizadas, como CLS. También debe tener siempre presente que algunas de las máquinas a las que se destina el BASICODE tienen una capacidad de memoria muy limitada, y los programas largos, aunque sean totalmente válidos en BASICODE, no cabrían en la RAM disponible de algunas máquinas. Sería muy posible escribir un programa que funcionase en su propio ordenador, pero la prueba de fuego sería guardarlo (SAVE) e intentar ejecutarlo sobre un ordenador distinto.

Dentro del programa principal el programador podría desear añadir ciertas mejoras no permitidas. Lo que se aconseja hacer es usar sentencias REM que expliquen con exactitud qué es lo que el programador se propone realizar. Los autores del BASICODE recomiendan escribir estas sentencias entre las líneas 20000 y 24999, si bien no es obligatorio. Después de cargar el programa, el usuario puede realizar las mejoras descritas adecuadas a su propia máquina.

Al paquete que proporciona la BBC para utilizar con la serie de transmisiones *Chip shop* se adjuntan instrucciones acerca del uso del BASICODE-2. El usuario recibe una cassette con los programas de traducción para las diversas máquinas en la cara uno. Aunque la mayoría de los ordenadores exigen la carga de un único programa de traducción a BASICODE-2, los micros BBC y el Vic-20 poseen programas SAVE y LOAD separados. Todos los programas incluyen instrucciones habladas para ayudar al usuario a encontrar el programa adecuado. En la cara dos hay 18 programas de demostración para dar idea de las posibilidades del BASICODE.

Dadas las inmensas variaciones en cuanto a lo que se supone que es un lenguaje estándar, los autores del BASICODE han conseguido un notable éxito al reunir tantas máquinas bajo el mismo alero.

Ian McKinnell

El manual y la cassette de BASICODE-2 se pueden adquirir enviando un cheque o giro postal por valor de 3,95 libras a nombre de Broadcasting Support Services a:

Broadcasting Support Services
P.O. Box 7
London
W3 6XJ
Gran Bretaña

Selección aleatoria

Los archivos aleatorios ofrecen un acceso mucho más rápido a cambio de consumir más espacio de almacenamiento y requerir un diseño más elaborado que los secuenciales

Las limitaciones de los archivos secuenciales surgen debido a la necesidad de leer la información almacenada en ellos por el orden en que se grabó. Los dispositivos de acceso aleatorio o directo dan solución a estas limitaciones porque los registros que contienen pueden ser accedidos en cualquier orden y muy rápidamente. La palabra «aleatorio» significa que en el archivo se puede escribir o leer cualquier registro de datos sin necesidad de pasar a través de toda la información precedente.

El problema obvio es que todos los archivos contenidos en cinta de cassette deben ser archivos secuenciales. No existe forma alguna de ir directamente a un dato que esté en la mitad de la cinta de cassette, siendo inevitable la lectura de toda la cinta hasta ese punto. La única forma de utilizar archivos de acceso directo en un micro basado en el almacenamiento en cinta consiste en cargar todos los datos en la memoria, aunque esto limite el tamaño del archivo. Para un acceso aleatorio verdaderamente útil son necesarias las unidades de disco, y aun así, unas pocas marcas de unidades de disco no admiten este tipo de tratamiento de archivos.

El usuario notará también que es más sencillo trabajar con archivos de acceso aleatorio que usar las técnicas más bien incómodas que se precisan para los archivos secuenciales. La división del archivo en registros y campos que detallamos en la página 706 es muy importante en el caso de los archivos de acceso aleatorio. Para acceder al archivo es necesario especificar el registro requerido. Este registro será colocado entonces en un buffer de la memoria del ordenador, donde se pueden eliminar, actualizar o imprimir los campos que contiene.

Afortunadamente, el sistema operativo se ocupará de gestionar las estructuras necesariamente más complicadas que requiere este tipo de ficheros. Con objeto de facilitar su rápida localización, todos los registros de un archivo directo tienen la misma longitud. Si cada uno tuviera, por ejemplo, 100 bytes o caracteres de longitud, y el programa solicitara el registro número 83, el sistema operativo posicionaría el cabezal del disco sobre el byte 8300 del archivo. Puesto que el sistema operativo sabe cuántos bytes hay en cada sector del disco, no es difícil calcular en qué sector del disco se halla el registro requerido. Este método de lectura de archivos podría parecer complicado, y por cierto es lento; pero, a pesar de todo, es mucho más rápido que leer a través de un archivo secuencial.

Al normalizar la longitud de los registros, obviamente es necesario elegir un tamaño que sea suficiente para contener el registro más largo almacenado en el archivo. Los registros más cortos se rellenan, por lo general, con espacios (32, en código ASCII). Éste es un problema importante de los archivos aleatorios, ya que el relleno requerido para dejar todos los registros con la longitud elegida implica desperdiciar un precioso espacio de almacenamiento. Esto significa que los archivos aleatorios se utilizan para pequeñas cantidades de información pero cuyo acceso deba ser muy rápido, mientras que los archivos secuenciales se utilizan para el almacenamiento masivo de datos en los que la velocidad de acceso no sea demasiado importante.

Los campos de un registro también se deben definir con una longitud fija. Esto es particularmente relevante para los sistemas que proporcionan facilidad de acceso directo a campos además de registros. Aun para los sistemas que no disponen de esta facilidad, sigue siendo una forma elegante y eficaz de definir sus archivos. El primer paso al diseñar la estructura de los registros de un archivo de acceso aleatorio consiste en estudiar los distintos campos y decidir las longitudes idóneas para los mismos.

La economía es crucial al diseñar un archivo, ya que necesariamente habrá una compensación entre la cantidad de información almacenada y el número de registros distintos. Con bastante frecuencia pueden elaborarse sistemas de codificación que reduzcan la cantidad de espacio ocupado por los datos. Por ejemplo, códigos de color 1, 2 y 3 para representar los colores negro, rojo y verde, o códigos como 841011 para representar la fecha 11 de octubre de 1984. Los sistemas de codificación deben ser

Aleatorio contra secuencial

	ARCHIVOS DE ACCESO DIRECTO	ARCHIVOS SECUENCIALES
PROS	• Rápido acceso a registros determinados	• Conservan espacio • Disponibles para sistemas basados en cintas
CONTRAS	• Desaprovechan espacio • Necesitan discos	• Lentos y engorrosos
APLICACIONES ADECUADAS	• Datos predecibles que estén en un formato definido • Cuando se accede a un pequeño número de registros diferentes; por ejemplo, en una biblioteca donde los clientes solicitan detalles de libros determinados	• Grandes cantidades de datos sin estructurar • Cuando la mayoría de los registros de un archivo se procesan en una única ejecución del programa; por ejemplo, en un sistema de salarios, donde se debe pagar a cada empleado

internos del sistema, y los programas deben convertir los códigos a una forma fácilmente comprensible al visualizarlos por pantalla el operador.

Existen otras dos consideraciones que hay que tener presente al determinar las longitudes de los registros. La mayoría de los sistemas limitan la longitud máxima disponible. Ésta puede variar normalmente desde 128 hasta 2 048 bytes. Además, suele ser más eficaz elegir una longitud que sea un múltiplo del tamaño del sector (cifras como 64, 128, 256 o 512 son las más comunes). Esto evita que los registros individuales queden a caballo de dos o más sectores y, por consiguiente, reduce el número de accesos a disco.

Los archivos directos, por lo general, son más fáciles de tratar que los archivos secuenciales. En ambos sistemas es necesario mantener un contador actualizado del número de registros del archivo, y muy a menudo se usa en archivos directos el registro 0 para guardar esta información y otra información relevante, como la fecha de creación del archivo. Obviamente, este registro tiene una estructura de campos distinta a la del resto.

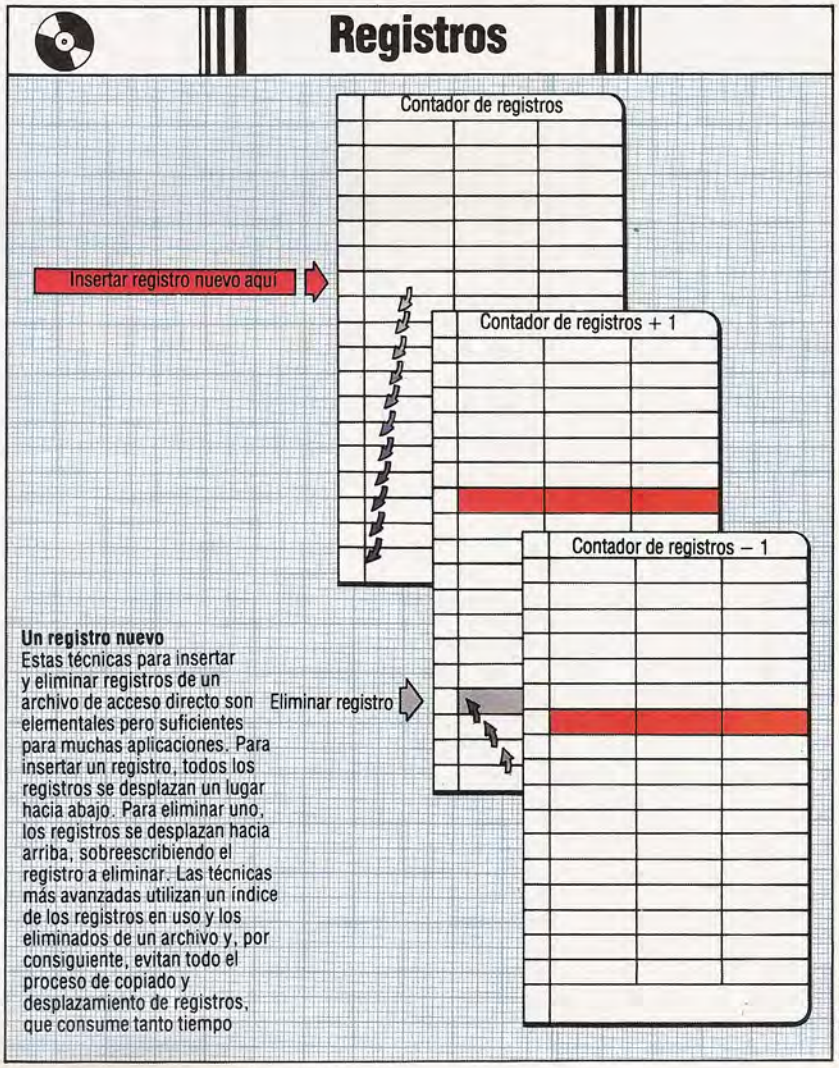
Un registro se puede actualizar leyéndolo, modificándolo y volviéndolo a escribir luego en su sitio. El registro es accedido por su número. Obviamente, no es razonable pedirle al usuario que recuerde el número correspondiente a cada registro. De modo que existe una amplia variedad de técnicas para buscar y localizar registros determinados, similares conceptualmente a las técnicas que se utilizan para acceder a elementos contenidos en matrices de BASIC. Con frecuencia se usa un campo determinado, por ejemplo, un campo de nombre, como clave de acceso del archivo. El ordenador lee el campo de clave y elabora un índice que identifica dónde están almacenados varios nombres.

En los archivos directos sin índice se suele buscar registro a registro, al igual que en los archivos secuenciales. Sin embargo, si los registros están clasificados por el campo clave, se pueden utilizar métodos de búsqueda rápidos. Supongamos, por ejemplo, que deseamos buscar "Jiménez" en un archivo clasificado por apellido. Comenzamos buscando el registro que está en la mitad del fichero y descubrimos que el apellido es "Paredes". Alfabéticamente, "Jiménez" está antes que "Paredes", de modo que podemos descartar todo lo que haya después de este registro. Nuestro siguiente intento es, entonces, un registro que esté a medio camino de la primera mitad del archivo. El siguiente apellido podría ser "Hernández", en cuyo caso deberemos volver a repetir la operación, y así sucesivamente. Esta clase de técnica puede ser muy compleja, y muchos programas mejoran el rendimiento teniendo en RAM un gran número de los registros utilizados más frecuentemente, de modo que estén disponibles con rapidez. Como resultado, en archivos grandes se localizan y almacenan registros a gran velocidad.

Eliminar e insertar registros nuevos puede ser comparativamente lento. El método más burdo para eliminar un registro consiste en copiar, en el espacio que ocupa, el registro que viene inmediatamente tras él, sobreescribiendo por consiguiente la información que contenía. Todos los registros subsiguientes se copian entonces una posición hacia arriba y, por último, el contador de registros se reduce en uno. De forma similar, se puede insertar un registro nuevo en cualquier punto desplazando un lugar todos los registros desde ese punto hasta el final del archivo. Esto crea un "agujero" de un registro en donde se puede colocar el registro nuevo.

Ninguna de estas técnicas es rápida, si bien son más eficaces que las operaciones similares con archivos secuenciales. No obstante, las inserciones y eliminaciones se pueden efectuar de manera mucho más expedita si el archivo posee un índice separado. Cuando se elimina un registro, se lo marca como eliminado en el índice, sin necesidad de modificar sus datos. A medida que se añaden nuevos registros, éstos pueden ser ubicados en los espacios correspondientes a registros eliminados o sin usar y actualizar el índice.

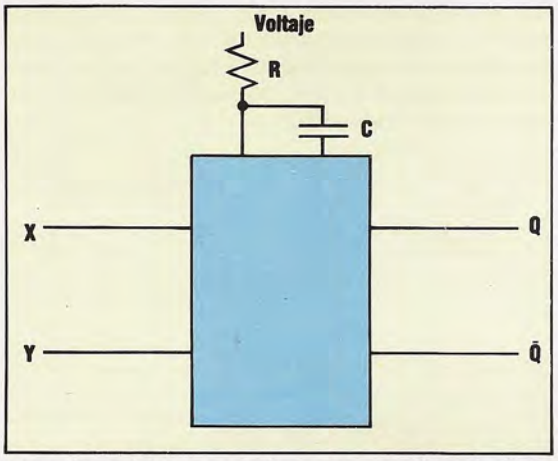
Existen dos ventajas finales a considerar en el sistema de archivos aleatorios o directos. En primer lugar, mientras que ciertamente es más rápido leer y escribir grupos de registros juntos, los archivos pueden quedar desordenados. Por lo tanto, la mayoría de los programas ofrecen una facilidad de reorganización que clasifica los registros por un orden lógico y descarta los registros eliminados. En segundo lugar, el sistema de marcar meramente los registros eliminados como borrados proporciona un útil mecanismo de seguridad, ya que es fácil recuperar estos registros en caso de ser necesario. Este mecanismo de seguridad será válido hasta el momento en que se ejecute un programa de reorganización sobre este fichero.



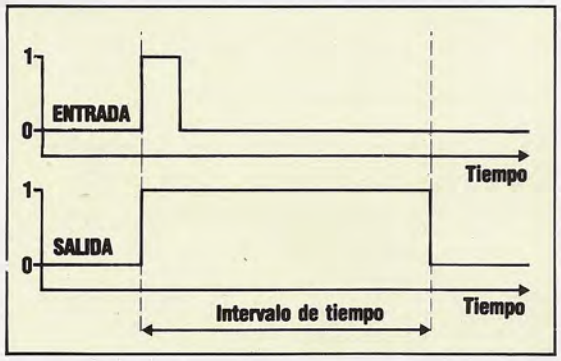
Marcando el compás

Para controlar sus funciones, un ordenador requiere una sincronización exacta. Analicemos tres circuitos que colaboran en esta tarea: el monoestable, el flip-flop tipo D y el flip-flop J-K

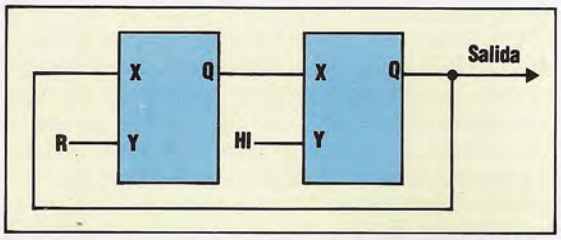
Un *circuito monoestable* proporciona un medio para introducir intervalos fijos de tiempo en las operaciones de los circuitos lógicos. Cuando un circuito monoestable recibe un impulso de entrada, la salida se pone en 1 (HI) durante un intervalo de tiempo fijo antes de volver a su estado normal (LO) de salida cero. El período de tiempo durante el cual la salida es HI está determinado por los valores de ciertos componentes del circuito. Éste es un ejemplo de circuito monoestable:



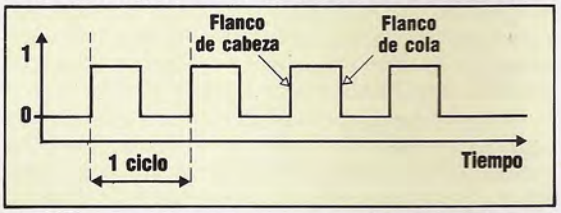
Este dispositivo se puede activar cambiando X de HI a LO o Y de LO a HI. Alterando los valores de la resistencia, R, y del condensador, C, se puede modificar el tiempo de salida. Este gráfico ilustra cómo están relacionadas la entrada y la salida:



La duración de la salida HI se podría utilizar para controlar el motor paso a paso de un lector de cinta o para retardar la transmisión de un bit durante un cierto período de tiempo. Dos circuitos monoestables se pueden unir entre sí para proporcionar un impulso de reloj, que oscila a intervalos fijos entre HI y LO:



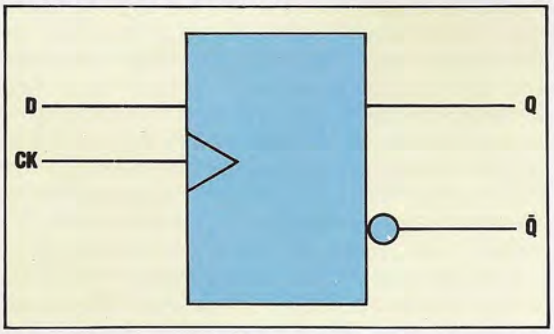
La salida producida tiene el aspecto característico de una "onda cuadrada" (como se ve en nuestros gráficos). Al intervalo de tiempo que media entre los dos inicios de la señal de salida HI se le denomina *ciclo*. Típicamente, éste es una millonésima de segundo. Esta señal de reloj continua es el latido del corazón del ordenador, y temporiza las muchas funciones que se llevan a cabo en la CPU. El siguiente diagrama indica los nombres con que se designan los "flancos" del gráfico de onda cuadrada, donde un impulso cambia de HI a LO, o viceversa:



Analicemos ahora dos nuevos tipos de flip-flop cuyas acciones están gobernadas por los impulsos regulares del reloj.

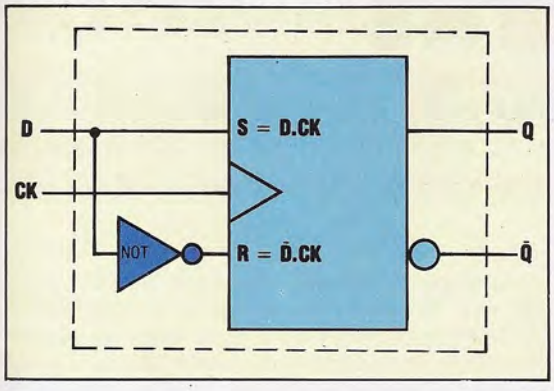
El flip-flop tipo D

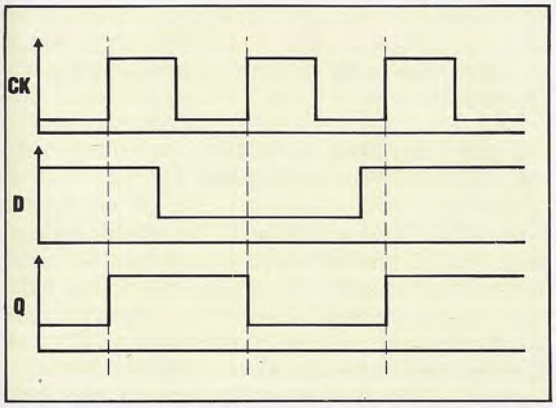
El flip-flop tipo D tiene una entrada lógica (D) y una entrada de reloj (CK):



El diseño del tipo D se basa en el flip-flop R-S, que analizamos en el capítulo anterior. Sin embargo, es la entrada de la señal del reloj lo que produce el método de operación especial conocido como *latching*. La salida del circuito, Q, se determina al comienzo de un ciclo de reloj. Si en ese momento la entrada en D es HI, entonces la salida Q se pone en

HI. Si, por el contrario, la entrada en D es LO, entonces la salida Q se pone en LO.

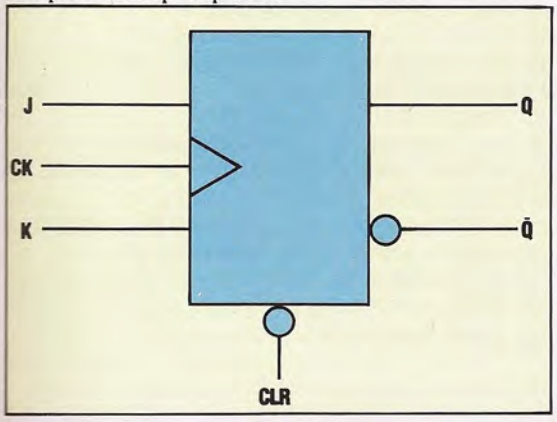




A partir de estos gráficos se puede ver que la salida Q sólo puede cambiar durante una transición del reloj de LO a HI. Por ello, el flip-flop tipo D se dice que es "activado por flanco de cabeza".

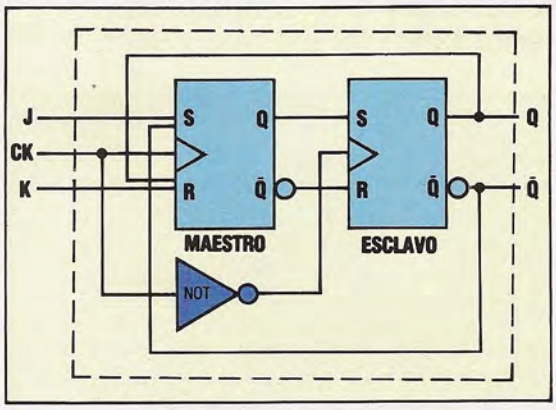
El flip-flop J-K

Se dice que el flip-flop J-K es un dispositivo maestro-esclavo porque se compone de dos flip-flops R-S en relación "dominante-sometido". Un dispositivo maestro-esclavo permite almacenar un impulso de entrada en un flip-flop, dando por la otra unidad una salida dependiente de la entrada previa, dentro de un ciclo de reloj. Un ejemplo es la operación de desplazamiento, común en la mayoría de los procesadores, en la que los bits de un registro se desplazan un lugar a la izquierda o a la derecha. He aquí el diagrama de circuito estándar para un flip-flop J-K:



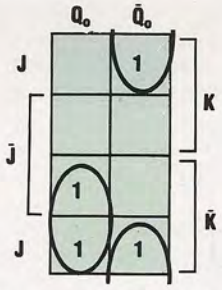
Liz Dixon

El siguiente diagrama muestra cómo están enlazados entre sí los dos tipos R-S. Uno es el "maestro" y el otro el "esclavo". Supongamos que se aplica una entrada en J o K: si la señal de entrada del reloj es HI se activa el maestro; si es LO, alimenta las entradas del esclavo, pues los tipos R-S se activan por flanco de cabeza. Así, sólo un R-S está activado en cualquier momento y queda "bloqueada" la entrada previa en el otro R-S:



En el margen tenemos una *tabla de estados* para el flip-flop J-K. Ésta es similar a una tabla de verdad, pero hace uso de una variable, Q_0, la salida previa. Observe que HI entra simultáneamente por J y K, y esto hace que el flip-flop cambie de estado a cada impulso de reloj. Esto se conoce como temporizado (*toggling*) y se produce por la realimentación de las salidas esclavas a las entradas maestras. Con un flip-flop R-S ésta no es una combinación de entradas permitida, quedando indefinida la salida. Considerando Q_0, J y K como entradas, los resultados de la tabla de estados se pueden colocar en un diagrama K.

Q_0	J	K	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



A partir del diagrama K podemos obtener la expresión lógica:

$$Q = \bar{Q}_0.J + Q_0.\bar{K}$$

Se dice que esta ecuación es la *característica* del flip-flop J-K.

Respuestas al ejercicio 7 de la página 709

1) Al flip-flop también se lo conoce por biestable porque sólo puede estar en uno de dos estados (a saber, cuando Q=1 y Q̄=0 o cuando Q=0 y Q̄=1).

2) a) Éste no es un estado posible.

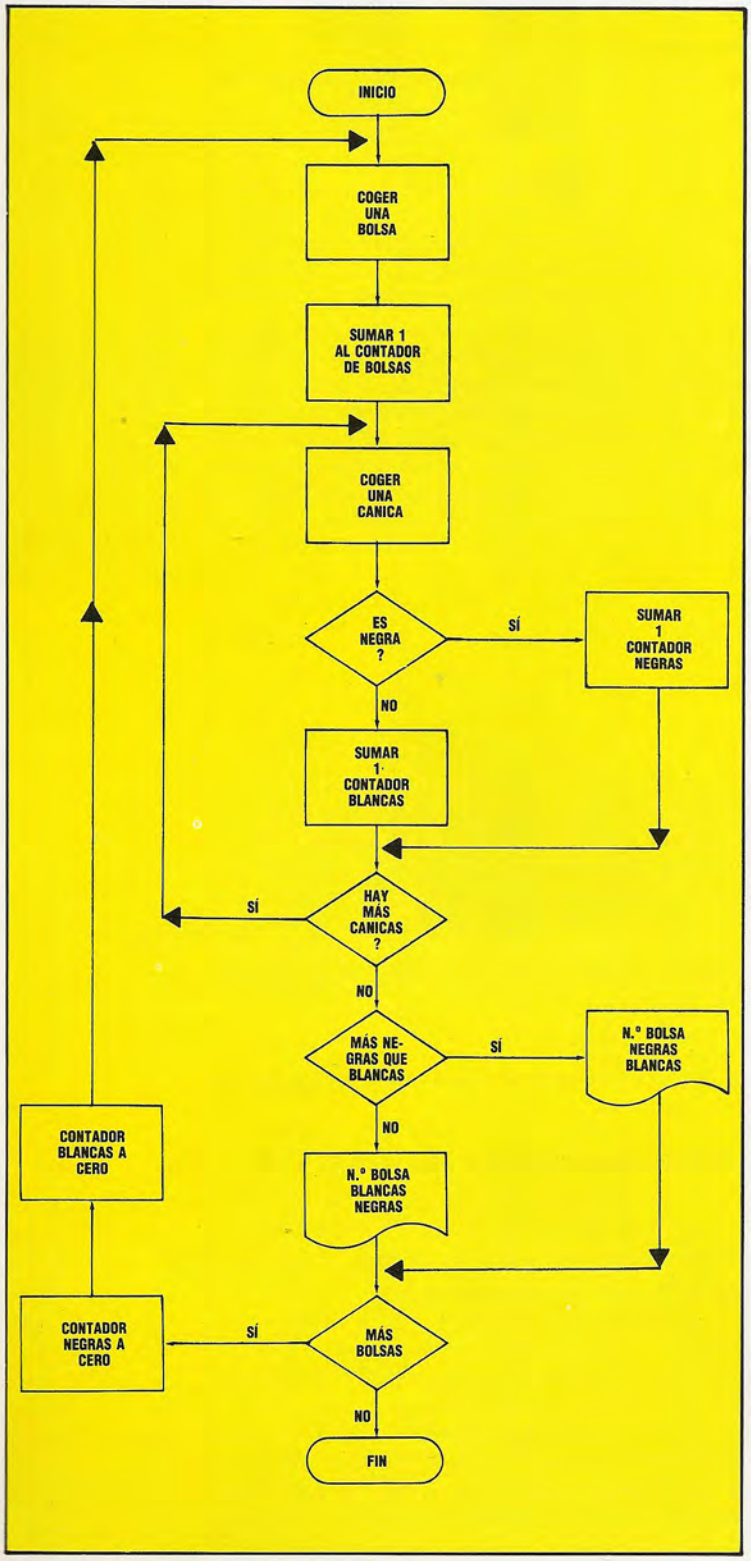
b) El flip-flop cambiará al estado RESET si la entrada superior es considerada primero, o al estado SET si la puerta inferior es considerada primero.

c) Sí (véase respuesta b).

d) Para que todos los registros adquieran un estado predecible al conectar el equipo, se les debe situar a todos en estado SET o RESET en el proceso de inicialización del sistema.

Cuentas claras

Veremos a continuación, mediante un ejemplo práctico, cómo funcionan los contadores



Analicemos el siguiente problema: un niño tiene una serie de bolsas que contienen canicas de dos colores: blancas y negras. Quiere llevar un control de cuántas bolsas posee, así como del número de canicas de cada color que hay en cada bolsa. Todo ello lo escribe en un papel, indicando el número de la bolsa que ha controlado y el número de canicas en orden de color predominante. Veamos un ejemplo:

```
BOLSA 5: BLANCAS 10; NEGRAS 7
BOLSA 9: NEGRAS 20; BLANCAS 12
```

En este caso se observa que el programa va a emplear tres contadores: el de bolsas, que va incrementándose por cada nueva bolsa, y uno para cada color de canica; estos dos irán poniéndose a cero cada vez que se empiece con una nueva bolsa, ya que, de no hacerse así, al cogerse otra bolsa sin borrar el contenido previo del contador, éste indicaría el número de canicas acumulado de todas las bolsas, con lo cual se obtendría un falso resultado.

Al no saberse de antemano el número de bolsas disponibles, ni conocer la cantidad de canicas por bolsa, es preciso obtener esa información a través de consultas al operador.

En caso contrario, es decir, si no se supiera previamente cuál es el número de bolsas con que se cuenta, no sería preciso consultar al operador, por cuanto por programación se controlaría el número final de bolsas.

```
10 REM BOLSAS CON CANICAS
15 PRINT "COGER UNA BOLSA"
20 X = X + 1
25 PRINT "COGER UNA CANICA"
30 INPUT "ES BLANCA" (S/N)";A$
40 IF A$="S" THEN B=B+1 : GOTO 60
50 IF A$ <> "N" THEN GOTO 30
55 N = N + 1
60 INPUT "HAY MAS CANICAS? (S/N)";B$
70 IF B$="S" THEN GOTO 25
80 IF B$<>"N" THEN GOTO 60
90 REM DECISION COLOR PREDOMINANTE
100 IF B>N THEN PRINT "BOLSA"X, "BLANCAS"B,
    "NEGRAS"N: GOTO 120
110 PRINT "BOLSA"X, "NEGRAS"N, "BLANCAS"B
120 INPUT "HAY MAS BOLSAS? (S/N)";C$
130 IF C$="N" THEN END
140 IF C$<>"S" THEN GOTO 120
150 B=0 : N=0 : GOTO 15
```

Atractivo y ligero

El Apricot es un ordenador en tres piezas lo suficientemente ligero como para ser transportado con facilidad

Diseñado en gran parte teniendo en mente al usuario de gestión, el Apricot tiene dos versiones: una con dos unidades de disco flexible de 3 ½" y la otra con una unidad de disco flexible de 3½" y un disco fijo de 10 Mbytes. Aunque dispone de numerosas facilidades diseñadas para atraer al usuario serio, el Apricot hace pocas concesiones al mercado del ordenador personal: no posee gráficos en color, conexión para cassette, accesorios para juegos o salida para televisor. No obstante, se suministran como estándar un monitor monocromático de alta resolución, una puerta en paralelo para impresora, una puerta serial RS232, un conector para un ratón opcional, algo de software y un teclado de calidad.

Lo más llamativo del Apricot es su teclado, versátil e innovador. Una característica novedosa es la Microscreen (micropantalla): una visualización en cristal líquido de dos líneas de 40 caracteres cada una, situadas en la parte superior derecha de las teclas principales. Al conectarse el ordenador, la línea superior de la Microscreen visualiza el día de la semana, el mes, el año y la hora. La fecha y la hora se pueden alterar utilizando uno de los programas de utilidad, y un reloj a pilas mantiene la hora cuando el ordenador no se está utilizando.

Cuando se enciende la máquina comienza automáticamente un programa de autocomprobación. Éste visualiza la cantidad de memoria disponible (lo estándar son 256 Kbytes pero se pueden ampliar a 768 Kbytes) y solicita al usuario que inserte el disco maestro MS-DOS. Para los usuarios que no estén familiarizados con sistemas operativos como el CP/M o el MS-DOS, un "menú" de fácil comprensión para el usuario llamado *Manager* (director) permite la fácil selección del software de aplicaciones (como Supercalc, Multiplan, BASIC Microsoft, etc.) o utilidades (como el configurador del teclado o el editor de pantalla).

El control de la Microscreen es por software, de modo que actúa como algo más que una mera representación visual de la hora y la fecha. Existen seis teclas programables por el usuario y las funciones asignadas a las mismas se pueden mostrar en cualquier momento en la pantalla de cristal líquido. De modo, entonces, que cuando un programa visualiza un menú de opciones en la pantalla principal, el mismo menú se puede duplicar en la Microscreen. Tocar la tecla de función adecuada equivale a seleccionar el ítem del menú en pantalla pulsando las teclas Cursor y Return. La única objeción es que las teclas de membrana al estilo de las que posee el Sinclair ZX81 son menos cómodas y de uso más engorroso que las teclas convencionales tipo máquina de escribir.

Existen también ocho teclas de función ordinarias. Éstas llevan inscritas los nombres de sus funciones normales (HELP, PRINT, MENU, FINISH, etc.). No obstante, como todas las teclas del Apricot, éstas se pueden reconfigurar con el programa Keyedit suministrado. La sensación táctil del teclado del Apricot responde al elevado estándar que cabe esperar de un ordenador de gestión, pero las teclas Control y Escape están en un lugar ligeramente insólito. Para que la máquina resulte fácil de mover, el teclado se conecta a la parte inferior de la unidad principal. La pesada carga que representa el monitor separado es suficiente, no obstante, para desalentar todo intento de considerarla como realmente portátil.

Chris Stevens

Una oferta tentadora
El ACT Apricot es uno de los ordenadores de aspecto más atrayente que existen en el mercado. Al equipo también se le ha fijado un precio modesto tratándose de un micro de oficina, aunque sus especificaciones son elevadas. Utiliza un microprocesador de 16 bits con una memoria estándar de 256 K y viene con un monitor de gran calidad

El software que se proporciona con el Apricot es una amplia gama de utilidades del sistema y Supercalc. El proceso de aplicaciones numéricas tipo "¿Qué sucedería si?" se cubre con Supercalc y Superplanner.

En estos productos se detectan evidencias, al igual que en otros elementos de software adaptados para ser usados en el Apricot, de un intento bastante apresurado de tenerlos a punto justo a tiempo para el lanzamiento del ordenador. Con la máquina vienen dos sistemas operativos, MS-DOS y CP/M-86. Los usuarios del Apricot tienen prometidas copias del Concurrent CP/M-86 cuando éste se encuentre a punto.

Con el ACT Apricot sólo se suministra la versión uno del Supercalc.

Una de las primeras críticas que se le plantearon al Apricot fue que el sistema operativo MS-DOS se había implementado mal y que era lento. Ahora este problema parece haberse superado. El software de aplicaciones que se suministra con la máquina parece funcionar con razonable rapidez y los programas de comprobación (*benchmark*) para probar el MBASIC de Microsoft también son relativamente rápidos.

Microfloppies
Apricot afirma ser el primer micro popular de oficina que utiliza la nueva generación de discos flexibles pequeños. ACT eligió el microflexible de Sony, que utiliza un disco flexible de sólo 3½" de ancho, en el interior de una carcasa rígida. Esto hace que el disco sea mucho más robusto que los discos flexibles tradicionales de 5¼". Una cubierta de carga de resorte protege del polvo al microflexible

Aun así, da la impresión de que el Apricot no opera con la rapidez que uno esperaría de una máquina con procesador 8086.

La documentación del Apricot incluye una introducción para principiantes, una exhaustiva guía del sistema operativo MS-DOS, dos útiles guías para el Supercalc y el Superplanner y extensos manuales para el Wordstar y el Multiplan. ACT proporciona poca información en cuanto a hardware, aunque las utilidades suministradas son suficientes para instalar el ordenador. No hay ningún tipo de detalles acerca de la distribución de memoria ni de las llamadas al sistema, datos muy necesarios para una firma de software que deseara producir software independiente para el Apricot, aunque esta información se puede conseguir fácilmente de manos del fabricante.

El Apricot ha sido diseñado teniendo muy presente su utilización para gestión; no es un sistema para ingenieros de software ni para aficionados a los ordenadores. Si el Apricot alcanzara el éxito del Sirius de ACT, podemos anticipar placas conectables opcionales producidas por fabricantes independientes, así como placas de memoria extra y el modem de la propia ACT. Dejando de lado sus méritos como un ordenador de oficina sumamente versátil y económico, y prescindiendo incluso de su atrayente aspecto, el hecho de que una máquina de este precio disponga de software MS-DOS resulta suficiente para hacer que el ACT Apricot represente una opción muy atractiva.

Apricot XI
La capacidad de los microflexibles es limitada, por lo que ACT ofrece el Apricot XI negro. Éste posee un disco fijo de 10 Mb que reemplaza a uno de los dos microflexibles

Visualización
Un monitor de fósforo [illegible] alta resolución propor[illegible] visualización clara e [illegible]

LCD
Se puede utilizar una visualización en cristal líquido de dos líneas para mensajes o como reloj o calculadora

Sony Microfloppy
Se emplean microflexibles gemelos de 315 K, obteniendo un almacenamiento compacto adecuado

Teclas de función exclusiva
Estas teclas proporcionan funciones estándar como HELP y REPEAT para diversos programas

Teclas de función sensibles al tacto
Estas seis teclas se pueden etiquetar mediante la información visualizada en la pantalla de cristal líquido

RAM de 256 K
Se proporcion[illegible] RAM de 256 K

El teclado del Apricot
Además de las teclas de gran calidad que cabe esperar en todo micro de oficina, el Apricot posee seis teclas sensibles al tacto. Éstas están reservadas para funciones especiales. Debido a que estas funciones cambian de un programa a otro, el Apricot permite visualizar una etiqueta encima de cada tecla, gracias a una pantalla de cristal líquido de dos líneas de 40 caracteres, la cual también se puede utilizar para visualizar la hora

El monitor del Apricot
El monitor tiene una pantalla de sólo 9" de ancho, a pesar de lo cual tiene una visualización de textos de 132 caracteres en 50 líneas, si bien normalmente se utiliza en modalidad de 80×25 caracteres. La calidad es buena y tiene una pantalla antirreflectante incorporada. El peso del monitor reduce las posibilidades de considerar el Apricot como un portátil, aunque algunos usuarios aparentemente han optado por tener un monitor en casa y otro en la oficina, limitándose a transportar de aquí para allá el cuerpo principal del micro

Chris Stevens

ACT APRICOT

DIMENSIONES
488×413×313 mm

CPU
8086 con opción de procesador de cálculo numérico 8087

MEMORIA
256 K de RAM, ampliables a 768 K

PANTALLA
El texto se puede mostrar en 132×50 caracteres o en 80×25 caracteres. La resolución de gráficos es de 800×400 puntos, sólo en monocromático

INTERFACES
Centronics, conector para "ratón", RS232 y ranuras internas para ampliación

UNIDADES DE DISCO
Uno o dos microflexibles Sony de 315 K o 720 K de capacidad cada uno. El Apricot XI tiene un disco rígido de 10 Mb y un microflexible

SISTEMAS OPERATIVOS
MS-DOS, CP/M-86 y Concurrent CP/M-86

TECLADO
90 teclas tipo máquina de escribir más seis teclas de función sensibles al tacto, complementadas con pantalla de cristal líquido para etiquetas

DOCUMENTACION
Amplia y bien presentada

VENTAJAS
Una máquina agradable de utilizar que tiene mejores especificaciones que máquinas de oficina que son éxitos de ventas y cuestan considerablemente más. ¡Y tiene un buen aspecto!

DESVENTAJAS
A pesar de tener una buena relación calidad-precio, el Apricot es más bien caro para el aficionado. Carece, asimismo, de la capacidad para emplear el software estándar para CP/M-80

Juego de animales mágicos

Siguiendo con los programas de entretenimiento, presentamos nuestra propia versión del juego "Animales"

Animales es un juego en el cual el ordenador intenta adivinar el nombre del animal en el que está pensando el jugador. Esto lo hace formulando preguntas tales como "¿Puede volar?", "¿Es peludo?", etc. A cada pregunta, uno sólo puede responder "sí" o "no", y el ordenador gradualmente va abriéndose paso hasta un punto en el que puede aventurar una conjetura "educada". Obviamente, es bastante sorprendente, en especial para aquellas personas que no están familiarizadas con los ordenadores, que el programa sea capaz de hacer esto. Las dos características que contribuyen a que el programa sea particularmente entretenido son la capacidad del ordenador para comunicarse en lenguaje corriente (aun cuando las propias respuestas de uno se limiten a "sí" o "no") y la gran cantidad de conocimientos que necesariamente tiene que desplegar el ordenador para tener la posibilidad de adivinar de qué animal se trata.

En realidad *Animales* es un *programa heurístico* muy simple; es decir, un programa que aprende a mejorar su rendimiento durante su ejecución. Cuando el programa se utiliza por primera vez solamente "sabe" dos nombres de animales y una pregunta. Según su respuesta sea o no un "sí", puede intentar adivinar de qué animal se trata. Si el ordenador se equivoca (lo que ocurrirá casi con toda seguridad la primera vez), el programa le pide a usted que teclee el nombre del animal y una pregunta para distinguirlo de la conjetura del programa sobre cuál era ese animal. Esta información se añade entonces a la base de datos del programa para construir un "árbol del conocimiento" que pueda utilizar en el siguiente juego. Cada vez que usted juegue a *Animales*, el tamaño del árbol aumentará hasta que finalmente el programa llegue a adivinar con una precisión absoluta la mayoría de los animales, descubriendo uno nuevo tan sólo de manera ocasional.

El punto interesante a tener en cuenta es que aun así el programa no "sabe" nada acerca de animales. Sigue a ciegas una guía elaborada a partir del conocimiento combinado de todos los jugadores que han jugado con él. La información muy bien podría referirse a distintos tipos de cerveza, componentes de una motocicleta, enfermedades o a sus amigos y familiares. La versión del programa que le permite al usuario definir la pregunta inicial y dos respuestas se podría utilizar para una gran variedad de tareas diferentes. En otras palabras, no son los datos en sí mismos los que hacen que el programa funcione, sino que éste responde a la forma en que se organizan dichos datos.

Construir el árbol con un sencillo programa BASIC no es especialmente difícil. La mayoría de las estructuras como ésta se mantienen en matrices de BASIC: en este caso utilizando T$() para las preguntas y los nombres de los animales, y Y() y N() para los enlaces entre entradas concretas a T$. Estos enlaces forman el camino a través del árbol. Para cualquier entrada de T$, la correspondiente entrada en Y() le dice al programa dónde mirar si la respuesta a esa pregunta es positiva. De igual modo, la entrada en N() es el enlace para una respuesta negativa. Al final del árbol el texto en T$ no es una pregunta sino el nombre de un animal. Tanto Y() como N() están a cero en este caso y el programa tiene que hacer una conjetura acerca de qué animal en particular está pensando el jugador.

Esta versión del programa se ha hecho pequeña y simple para mostrarle los principios implicados. Si desea mejorarlo, podría perfeccionar la presentación para su máquina agregando gráficos en color, sonido, etc. Una mejora fundamental consistiría en darle al programa alguna forma para almacenar su base de datos en cinta o en disco. Las mejores versiones de *Animales* que se pueden encontrar son aquellas con las cuales unas personas han estado jugando con ellas durante años y que han construido un inmenso árbol de animales, animales míticos, objetos, personas famosas, amigos, etc., todo ello mezclado en una base de datos gigantesca.

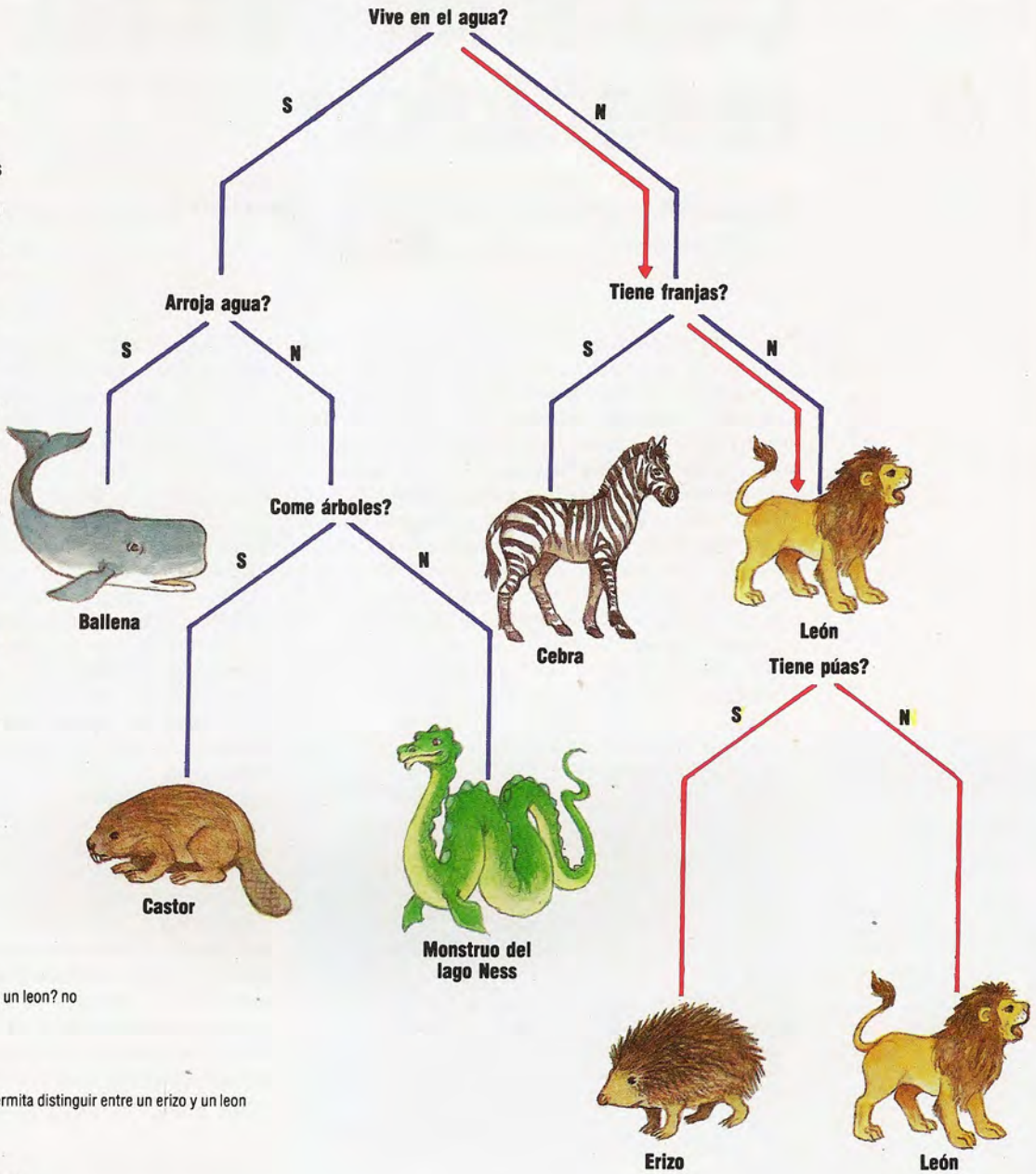
Complementos al BASIC

Este programa está escrito en BASIC Microsoft, de modo que se lo puede ejecutar en la mayoría de las máquinas sin ninguna modificación; el único cambio que quizá se desee efectuar es en relación al formato de las instrucciones PRINT si no le gusta la visualización en pantalla.
En el Spectrum, todas las sentencias de asignación deben comenzar con la palabra clave "LET". Reescriba las siguientes líneas como se indica:

```
45  LET L=40:REM N.° max de cars. de una
    preg.
50  DIM Y(N):DIM N(N):DIM T$(N,L)
150 LET I$=A$(1):LET P$="A "
200 IF A=30 THEN PRINT:PRINT
    "ADIOS":STOP
230 IF Y(P)=0 AND N(P)=0 THEN GOTO 290
```

Árbol de aprendizaje

Este diagrama muestra el árbol de *Animales* después de haber jugado con él varias veces. Se han aprendido cinco animales diferentes, con cuatro preguntas para distinguirlos entre sí. A partir de la ejecución dada como muestra (señalada en rojo), puede verse cómo el ordenador va a utilizar el árbol para contestar a las respuestas del jugador en la próxima ocasión en que se juegue. Esta vez, el jugador está pensando en un erizo y el ordenador tiene que aprender este nuevo animal cuando descubre que el jugador no tiene en mente un león. El ordenador solicita entonces alguna forma de distinguir entre un erizo y un león, de modo que pueda aprender el nuevo animal. Una vez que tenga puesta a punto su propia versión del programa, vea si puede incluir un camello



```
Ejecucion de muestra

Hace una partida? si

Vive en el agua? no

Tiene franjas? no

El animal en el que esta pensando es un leon? no

Me rindo!!!

Cual es su animal? Erizo

Por favor, entre una pregunta que permita distinguir entre un erizo y un leon
Tiene puas?

Para leon, la respuesta seria? no

Ahora ya conozco 6 animales diferentes!

Hace una partida?
```

```
10  REM Juego Animales
20  REM
30  REM ** Preparación **
40  N=100: REM N.º max de animales
50  DIM Y(N), N(N), T$(N)
60  C=3:FOR I=1 TO 3:READ Y(I), N(I), T$(I):NEXT I
70  PRINT:PRINT "A N I M A L E S!":PRINT
80  GOTO 190
90  REM ** Respuesta SI o NO **
100 PRINT:PRINT Q$; "  ";:INPUT A$
110 IF A$="s" OR A$="S" OR A$="SI" OR A$="si" THEN A=1:RETURN
120 IF A$="n" OR A$="N" OR A$="NO" OR A$="no" THEN A=0:RETURN
130 PRINT:PRINT "Por favor responda SI o NO": GOTO 100
140 REM ** Añadir UN o UNA al nombre del animal **
150 I$=RIGHT$(A$,1):P$="un"
160 IF I$="A" OR I$="a" THEN P$="una"
170 A$=P$+A$:RETURN
180 REM ** Empezar un juego nuevo **
190 Q$="Hace una partida?":GOSUB 100
200 IF A=0 THEN PRINT:PRINT "ADIOS!":END
210 P=1
220 REM ** Jugar juego **
230 IF Y(P)=0 AND N(P)=0 THEN 290
240 Q$=T$(P):GOSUB 100
250 IF A=1 THEN P=Y(P)
260 IF A=0 THEN P=N(P)
270 GOTO 230
280 REM ** Hacer conjetura sobre el animal **
290 A$=T$(P):GOSUB 150:T$=A$
300 Q$="Es el animal en el que está pensando"+A$:GOSUB 100
310 IF A=1 THEN PRINT:PRINT "Lo tengo!!!!":GOTO 430
320 REM ** Aprender un animal nuevo **
330 PRINT:PRINT "Me rindo!!!":PRINT "Cual es su animal?";:INPUT N$
340 A$=N$:GOSUB 150
350 PRINT:PRINT "Por favor entre una pregunta que permita distinguir":PRINT "entre"; A$;
    "y";T$:INPUT D$
360 Q$="Para"+T$+"la respuesta seria?":GOSUB 100
370 A$=T$(P):T$(P)=D$:T$(C+1)=A$:T$(C+2)=N$
380 IF A=1 THEN Y(P)=C+1:N(P)=C+2
390 IF A=0 THEN Y(P)=C+2:N(P)=C+1
400 Y(C+1)=0:N(C+1)=0:Y(C+2)=0:N(C+2)=0
410 C=C+2
420 REM ** Final juego & bucle para otra pasada **
430 A=INT(C/2)+1
440 PRINT:PRINT "Ahora ya conozco";A; "animales diferentes!"
450 GOTO 190
460 REM ** Datos iniciales **
470 DATA 2,3, "Vive en el agua?"
480 DATA 0,0, "Ballena"
490 DATA 0,0, "Leon"
```

David Higham

Capacidades de resolución

En nuestro estudio del Commodore 64, esta vez analizaremos sus gráficos de alta y baja resolución

Para baja resolución, la pantalla del Commodore 64 se divide en 25 líneas de 40 posiciones de carácter cada una, lo cual supone 1 000 celdas en total. Como sabemos, cada carácter se construye a partir de una serie de puntos más pequeños, dispuestos en ocho filas; la celda de cada carácter consta, por consiguiente, de 64 pixels. Para conseguir alta resolución, hemos de ser capaces de encender o apagar cada pixel de forma individual utilizando un único bit de la memoria del ordenador para controlarlos uno a uno. Este concepto se conoce como *mapa de bits*. Cada posición de memoria contiene ocho bits y la pantalla 64 000 pixels, luego son necesarias 8 000 posiciones de memoria para almacenar la información de la pantalla en alta resolución.

El Commodore 64 se pasa de la modalidad estándar de baja resolución a la de alta resolución poniendo el bit 5 de la posición 53265 a uno.

Para modificar este bit (que en decimal vale 32) sin perturbar a los otros, es necesario utilizar la siguiente instrucción:

```
POKE53265,PEEK(53265)OR32
```

Una vez establecida la modalidad de alta resolución, la pantalla recibe entonces su información desde un bloque de memoria de 8 000 bytes. El comienzo de este bloque de memoria lo señala la posición 53272. Ésta es la misma posición que se utilizó para la construcción de caracteres definidos por el usuario en el último capítulo de la serie (véase p. 712).

La zona de memoria normalmente asignada a la memoria de pantalla se emplea para contener la información de color para cada celda de ocho por ocho de la pantalla. Los 16 colores disponibles en el Commodore 64 se pueden representar con sólo cuatro bits; de modo que los cuatro bits superiores de cualquier posición de la memoria de pantalla se utilizan para indicar el color de los pixels que están "encendidos" en una celda determinada y los cuatro bits inferiores indican el color de los pixels "apagados". Por lo tanto, es posible tener pares de colores distintos, uno para el carácter y el otro para el fondo, en cada celda de la pantalla. Si deseamos un fondo púrpura, con gráficos de alta resolución dibujados en negro, son necesarios estos códigos:

El código de color para negro es 0 = 0000 en binario

El código de color para púrpura es 4 = 0100 en binario

Juntando ambas partes obtenemos 00000100, o 4 en decimal. Colocando (POKE) 4 en cada posición de la memoria de pantalla (de 1024 a 2023) produciremos los gráficos requeridos en negro sobre un fondo púrpura.

Antes de empezar a dibujar en la pantalla en alta resolución, la zona de 8 000 bytes que controla lo que se va a ver en pantalla se debe limpiar colocando (POKE) un cero en cada posición. En BASIC esta operación llevará varios segundos. De no hacerla, la visualización en pantalla será una mescolanza de puntos, pues esa zona de memoria particular toma valores al azar cuando la máquina se conecta.

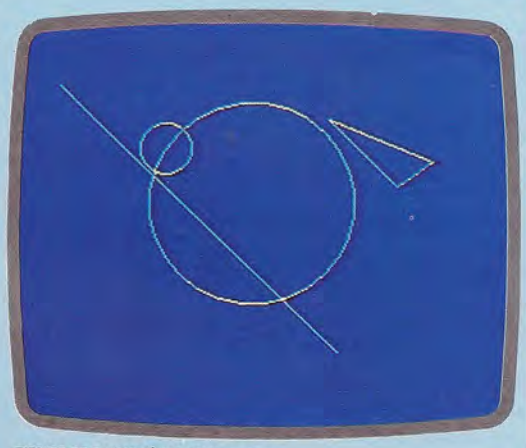
Trazando puntos

Un programa de gráficos de alta resolución ha de ser capaz de encender o apagar pixels individuales de la pantalla. Si a cada punto se le otorga una

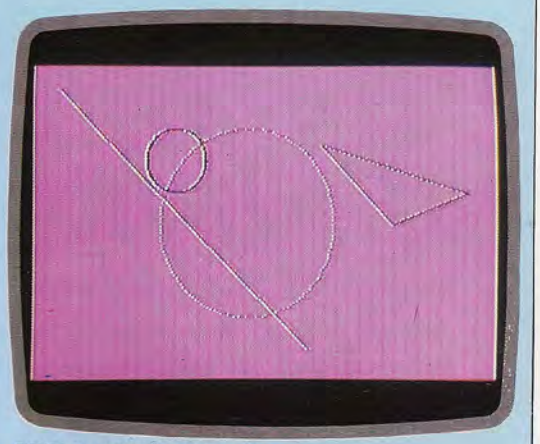
Velocidades relativas

Para producir esta visualización el Commodore 64 tardó cerca de 90 segundos con unas 50 líneas de programa. Producir la misma visualización en el Spectrum costó, sin embargo, alrededor de dos segundos mediante el siguiente programa:

```
4000 REM ******
     DEMOSTRACION ALTA
     RES *******
4050 LET N=18: DIM U(N):
     INK 6: PAPER 1: BORDER
     1: RESTORE 4100
4100 DATA 20, 160, 160, -160
4120 DATA 80, 123, 15
4140 DATA 130, 90,60
4160 DATA 175,140
4180 DATA 40, -40
4200 DATA 20,15
4220 DATA -60,25
4250 FOR K=1 TO N:READ
     U(K):NEXT K
4300 PLOT U(1), U(2):DRAW
     U(3), U(4)
4350 CIRCLE U(5), U(6), U(7)
4400 CIRCLE U(8), U(9), U(10)
4450 PLOT U(11),
     U(12):DRAW U(13),
     U(14)
4500 DRAW U(15),
     U(16):DRAW U(17),
     U(18)
4600 PAUSE 0
```

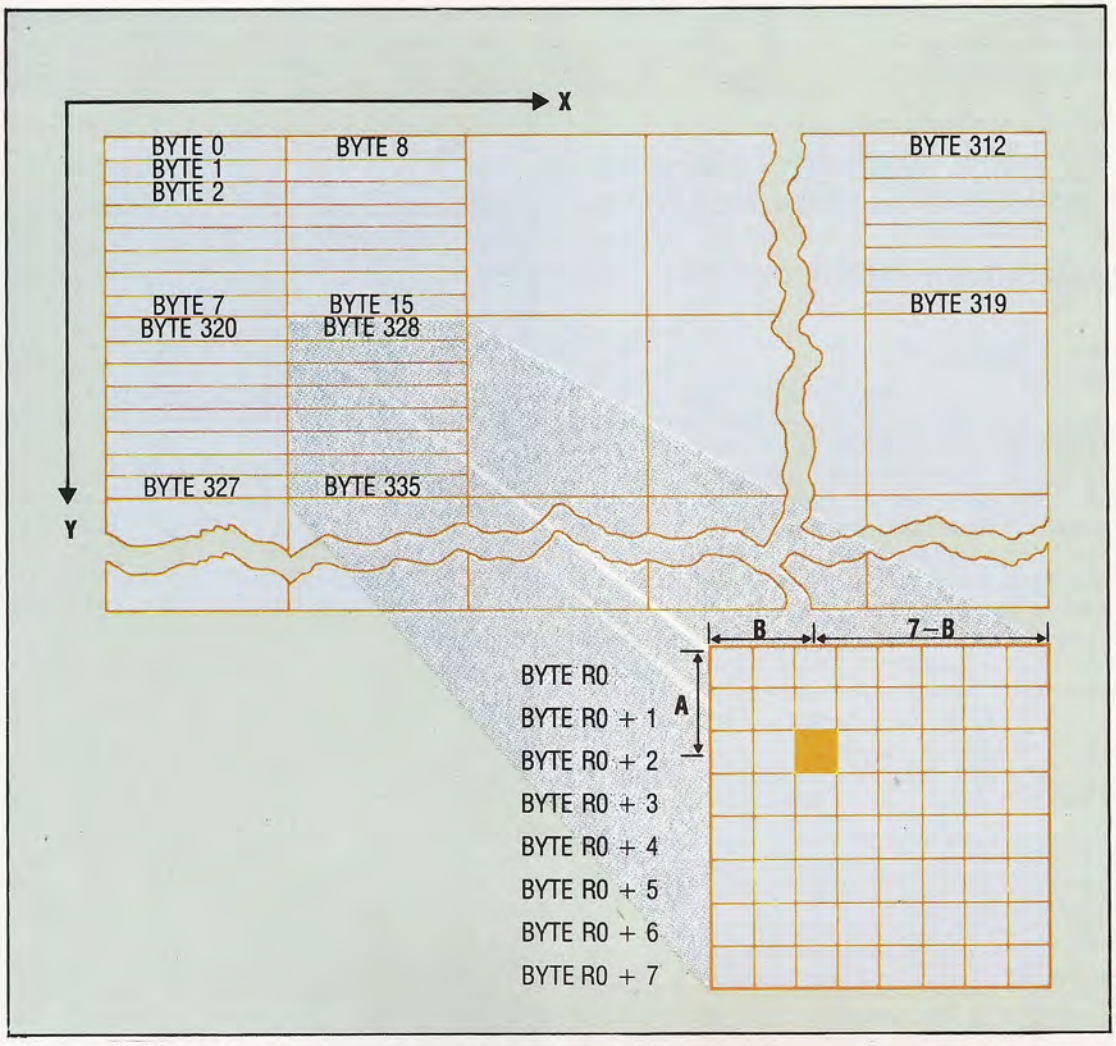


TIEMPO DE EJECUCIÓN: 1,85 segundos



TIEMPO DE EJECUCIÓN: 89,6 segundos

Liz Heaney



Liz Dixon

Punto a punto
A los pixels de puntos que componen la pantalla en alta resolución del Commodore 64 no se puede acceder directamente: la pantalla para texto de 40 × 25 se asocia a un mapa de 8 000 bytes en RAM, siendo descrita cada posición de texto mediante ocho bytes. La posición de un pixel de puntos se describe en alta resolución mediante X, su distancia (en pixels) desde la izquierda de la pantalla, e Y, su distancia desde la parte superior de la pantalla. Estos números se deben convertir en la dirección del byte que contiene el pixel, y el número de bit correspondiente de ese byte

coordenada X y una Y (X e Y están comprendidas entre las escalas de 0 a 319 y de 0 a 199, respectivamente) entonces el programa puede identificar qué bit correspondiente del mapa de memoria de 8 000 bytes se ha de poner a uno o a cero.

La posición horizontal del byte se puede hallar a partir de la coordenada X con la instrucción:

```
HB = INT(X/8)
```

Del mismo modo, el correspondiente byte vertical se puede hallar a partir de la coordenada Y:

```
VB = INT(Y/8)
```

El primer byte de la celda que contiene el bit requerido, R0, se puede calcular a partir de HB y VB:

```
R0 = VB*320 + HB* 8
```

El byte que contiene el bit requerido será R0, más el resto de dividir Y por ocho. Este resto se puede hallar fácilmente a partir de los tres bits situados más a la derecha del valor de Y. Si A = YAND7 y BASE es la dirección del primer byte del bloque de 8 000 bytes, entonces la dirección del byte, BY, que contiene el bit que requerimos se puede hallar de la siguiente forma:

```
BY = BASE + R0 + A
```

El bit dentro del byte BY se puede hallar calculando el resto de dividir la coordenada X por ocho. Si B = XAND7, el siguiente POKE pondrá a uno el bit que corresponde al pixel con las coordenadas X e Y:

```
POKE BY, PEEK (BY)OR(2↑(7–B))
```

Ahora que se ha encendido individualmente cada pixel, se pueden diseñar rutinas que dibujen formas en la pantalla. El siguiente programa muestra cómo se pueden dibujar líneas rectas desde un punto (X1, Y1) a otro (X2, Y2). Un círculo se puede dibujar especificando las coordenadas de su centro (CX, CY) y el radio RA. También hay una subrutina que dibujará un triángulo a partir de las coordenadas de sus tres vértices (XA, YA), (XB, YB) y (XC, YC). Puede que a usted le guste experimentar entrando coordenadas distintas a las dadas en el programa.

Es interesante observar que la estructura de este programa se compone de un conjunto de subrutinas en serie. La rutina de nivel inferior es la que traza un único punto en la pantalla. Esta subrutina es usada por otra rutina de mayor nivel que dibuja una línea recta. A un nivel todavía superior, la rutina TRAZAR TRIANGULO utiliza la rutina TRAZAR LINEA tres veces para trazar sus tres lados. Este enfoque de programación tiene muchas ventajas. Es flexible, porque sería muy fácil diseñar una rutina para dibujar, supongamos, hexágonos regulares. Dicha rutina llamaría a la rutina TRAZAR LINEA, que a su vez llamaría a la rutina TRAZAR PUNTO. Incluso se podría utilizar la rutina DIBUJAR TRIANGULO para construir hexágonos a partir de triángulos equiláteros. En este caso la rutina DIBUJAR HEXAGONO for-

```
65 REM **** DEMOST ALTA RES ****
70 PRINT CHR$(147): REM LIMPIAR PANTALLA
80 POKE 53280,0 : REM COLOR BORDE NEGRO
90 :
100 REM **** ZONA DE MEMORIA COLOR PANTALLA ****
110 FOR I=1024 TO 2023: POKE I,4: NEXT I
120 :
130 REM **** INICIALIZAR APUNTADOR MAPA BITS ****
140 BASE=8192: POKE 53272. PEEK (53272) OR 8
150 :
160 REM **** QUITAR MODALIDAD MAPA BITS ****
170 FOR I=BASE TO BASE + 7999:POKE I,0:NEXT I
180 :
190 REM **** PONER MODALIDAD MAPA BITS ****
200 POKE 53265, PEEK(53265) OR 32
210 :
220 REM **** DIBUJAR LINEA RECTA ****
230 X1=20: X2=190: Y1=15: Y2=180
240 GOSUB 800: REM TRAZAR LINEA
250 :
300 REM **** DIBUJAR CIRCULO ****
310 CX=150: CY=100: RA=60
320 GOSUB 900: REM TRAZAR CIRCULO
330 :
370 **** OTRO CIRCULO ****
380 CX=100: CY=60: RA=20
390 GOSUB 900: TRAZAR CIRCULO
400 :
410 REM **** DIBUJAR TRIANGULO ****
420 XA=200:XB=250:XC=300: YA=50:YB=100:YC=80
430 GOSUB 600: REM TRAZAR TRIANGULO
440 :
450 GOTO 450: REM FIN DEL PROGRAMA PRINCIPAL
460 :
470 :
600 REM **** SUBRUTINA TRAZAR TRIANGULO ****
610 :
620 X1=XA: X2=XB: Y1=YA: Y2=YB
630 GOSUB 800: REM TRAZAR LINEA
640 X1=XB: X2=XC: Y1=YB: Y2=YC
650 GOSUB 800: REM TRAZAR LINEA
660 X1=XC: X2=XA: Y1=YC: Y2=YA
670 GOSUB 800: REM TRAZAR LINEA
680 RETURN
690 :
800 REM ***** SUBRUTINA TRAZAR LINEA *****
810 S=1
820 IF X2 < X1 THEN S=-1
830 FOR X=X1 TO X2 STEP S
840 Y=(Y2-Y1) * (X-X1)/(X2-X1)+Y1
850 GOSUB 1000: REM TRAZAR PUNTO
860 NEXT X
870 RETURN
880 :
900 REM **** SUBRUTINA TRAZAR CIRCULO ****
910 :
920 FOR ANGLE = 0 TO 2 * PI STEP .04
930 X=INT (RA * COS(ANGLE)+CX)
940 Y=INT (CY-RA * SIN(ANGLE))
950 GOSUB 1000: REM TRAZAR PUNTO
960 NEXT ANGLE
970 RETURN
980 :
1000 REM **** SUBRUTINA TRAZAR PUNTO ****
1010 :
1020 IF X > 319 OR X < 0 OR Y > 199 OR Y < 0 THEN GOTO 1070
1030 HB=INT(X/8): VB=INT(Y/8)
1040 RO=VB * 320 + HB*8: A= Y AND 7: B=X AND 7
1050 BY=BASE+RO+A
1060 POKE BY,PEEK (BY)OR(2^(7-B))
1070 RETURN
```

maría un cuarto eslabón en la estructura del programa.

Asegúrese de guardar (SAVE) este programa antes de ejecutarlo, ya que una instrucción POKE incorrecta haría que la máquina se "colgara" o se interrumpiera inesperadamente.

Programa Subhunter

Una parte importante del juego Subhunter que estamos diseñando es la rutina que actualiza el marcador. En un juego así hay muchas formas de asignar la puntuación; nos basaremos en estas reglas:

1) La profundidad y la velocidad del submarino son factores importantes. El que avanza con rapidez y en profundidad es más difícil de acertar que uno lento que se desplace cerca de la superficie. La puntuación asignada tendrá esto en cuenta.

2) Si se hace blanco en el submarino, el valor de su puntuación se suma al marcador del jugador, pero si el sumergible alcanza el borde de la pantalla sin haber sido alcanzado, su valor resta al marcador del jugador. No se permitirán marcadores negativos.

Más avanzado el proyecto nos ocuparemos de la rutina que selecciona al azar la velocidad y profundidad de los submarinos; pero por ahora todo lo que necesitamos saber es que la profundidad del sumergible se almacena en la variable Y3 y su velocidad en DX. El valor del submarino se puede calcular sobre esta base. Para asegurar que sólo se calculen valores de números enteros para el valor del submarino, se utiliza la función INT de la siguiente manera:

```
Valor sub = INT(Y3+DX*30)
```

Almacenaremos el valor en curso del jugador en una variable SC. Todo lo que queda por hacer es sumar o restar el valor del submarino a SC en función de si el submarino ha sido alcanzado o si se ha escapado. La subrutina ACTUALIZAR MARCADOR se utiliza en dos secciones del programa:

1) Donde se comprueba la posición del submarino para ver si ha alcanzado el borde de la pantalla y
2) durante la rutina ACERTAR

En estas dos partes del programa se puede usar el indicador DS para saber cuál de las dos partes está utilizando la subrutina ACTUALIZAR MARCADOR. Poniendo DS=1 en la rutina ACERTAR y DS=-1 en la rutina BORDE DE PANTALLA, se puede incrementar o decrementar el marcador en función del valor del submarino de la siguiente forma:

```
SC = SC+INT(Y3+DX*30)*DS
```

Luego de asegurarse de que el marcador está por debajo de cero, se puede imprimir (PRINT) el nuevo valor de SC en la línea superior de la pantalla. Añada estas líneas a su programa y guárdelo (SAVE).

```
5500 REM **** ACTUALIZAR MARCADOR ****
5510 SC=SC+INT(Y3+DX*30)*DS
5520 IFSC<0THEN SC=0
5530 PRINT<CHR$(19); CHR$(144);
     "MARCADOR";SC;CHR$(157); "  "
5540 RETURN
```

En el próximo capítulo analizaremos la creación de los sprites que se utilizarán para el barco, el submarino, las cargas de profundidad y la explosión.

El método LIFO

La pila es una zona definida de la memoria que juega un papel esencial en las subrutinas. Opera, como veremos, por el método LIFO (último en entrar, primero en salir)

El tratamiento de la memoria es la esencia de la programación en lenguaje assembly y la mayoría de las instrucciones que hemos estudiado hasta ahora en el curso están relacionadas simplemente con cargar datos en posiciones de la memoria o tomarlos de ella. A estas posiciones hemos accedido de diversas maneras (las modalidades de direccionamiento), pero hasta aquí todas las instrucciones que conocemos han empleado siempre una dirección específica de la memoria como parte del operando. Existe una clase de instrucciones, sin embargo, que acceden a un área específica de la memoria pero que no toman como operando dirección alguna. Son instrucciones que operan sobre el área de memoria denominada *pila* (*stack*) y se las conoce como operaciones de pila.

La pila está creada para que tanto la CPU como el programador dispongan de una memoria temporal donde trabajar. Se trata de una especie de "cuaderno de apuntes" cómodo para escribir y también fácil de leer y borrar. Las operaciones de pila copian datos de los registros de la CPU en zonas libres de la pila, o copian datos de la pila en los registros de la CPU. Estas instrucciones no exigen un operando con dirección, ya que un registro específico de la CPU, el llamado *índice de pila* (*stack pointer*), siempre contiene la dirección de la primera posición libre de la pila. Por tanto, todo cuanto se escribe en la pila es depositado automáticamente en el byte señalado por dicho índice, y los bytes que se sacan de la pila se toman de la última posición utilizada. Al ejecutar una operación de pila, su índice se ajusta como parte de la operación.

En los sistemas 6502, la pila ocupa los 256 bytes de RAM que van del $0100 al $01FF; en los sistemas Z80, la situación y el tamaño de la pila son determinados por el sistema operativo, pero pueden ser modificados por el programador. Esta variación refleja las diferencias de los microprocesadores en cuanto a organización interna (véase el diagrama de la página 616): el 6502 posee un índice de pila de un solo byte, mientras que el del Z80 se compone de dos bytes.

La CPU trata el contenido del índice de pila del 6502 como el byte bajo de una dirección de pila, al cual, para completar la dirección, se le suma automáticamente un byte alto con valor $01. Este bit extra siempre está a 1, de modo que en el 6502 todas las direcciones de pila se hallan en la p. 1.

El índice de pila del Z80 es un registro de dos bytes capaz de direccionar cualquier posición comprendida entre $0000 y $FFFF, o sea, todo el espacio direccionable por el propio Z80. Por consiguiente, la pila puede estar situada en cualquier lugar de la RAM, y el programador puede cambiarla de sitio si lo desea. Sin embargo, esto no es aconsejable, dado que el sistema operativo establece inicialmente su situación y va almacenando datos en la misma. Ya que el sistema operativo puede interrumpir la ejecución de cualquier programa en lenguaje máquina en cualquier momento, esperando hallar en la pila los datos adecuados para su funcionamiento, cualquier alteración de la situación de la pila significaría el desconcierto del sistema operativo, que puede quedar colgado a causa de esto.

Un ejemplo de cómo se emplea la pila será la siguiente rutina para intercambiar el contenido de dos posiciones de memoria, LOC1 y LOC2:

6502		Z80	
LDA	LOC1	LD	A,(LOC1)
PHA		PUSH	AF
LDA	LOC2	LD	A,(LOC2)
STA	LOC1	LD	(LOC1),A
PLA		POP	AF
STA	LOC2	LD	(LOC2), A

Primera línea: lo que contiene LOC1 es cargado en el acumulador (LDA). Segunda línea: el acumulador es "empujado" (*push*) dentro de la pila (PHA). Tercera y cuarta líneas: el contenido de LOC2 se carga en el acumulador y seguidamente se almacena (STA) en LOC1. Penúltima línea: el último (*Last*) byte metido en la pila "salta" (*pops*) otra vez dentro del acumulador (PLA). Última línea: el actual contenido del acumulador (que antes estaba en la pila y al principio en LOC1) se almacena en LOC2. Fin del intercambio. Observe cómo las operaciones de pila (PHA, PLA, o bien PUSH y POP AF) no llevan indicación de dirección alguna, a no ser, implícitamente, la primera posición libre en la pila.

Este fragmento de programa nos muestra muchísimas cosas acerca de las operaciones de pila. Ante todo, que son recíprocas y secuenciales. El último ítem empujado dentro de la pila se recupera al primer salto que se ordene desde la pila. Si ha habido varios "empujones" seguidos sin ningún "salto", los datos se escriben en sucesivas posiciones de la pila, cada byte "encima" del anterior, y si se piden varios "saltos" consecutivos, éstos van afectando a las sucesivas posiciones en orden descendente.

Una imagen visual de la pila la tendría si fuera escribiendo durante unos días numerosas tarjetas postales y las fuera apilando en un cajón. La que ocupa en cualquier momento la parte superior del montón sería la más reciente, la del fondo del montón la más antigua, y para sacar ésta debería sacar todas las anteriores. Por esta razón se dice que la pila es una estructura LIFO (*Last In First Out*), o sea, el último que entró es el primero en salir. Su

opuesta, la estructura FIFO (*First In First Out*), o sea, el primero que entra es el primero que sale, es denominada "cola" en lugar de pila. Convencionalmente, al primer byte libre de la pila se le denomina *tope* (*top*, en inglés) de la pila, ya que uno se imagina las pilas creciendo hacia arriba. Aunque, de hecho, tanto en el 6502 como en el Z80, el índice de pila disminuye a cada empujón, por lo que el tope está en realidad en una posición de memoria inferior a la base de la pila.

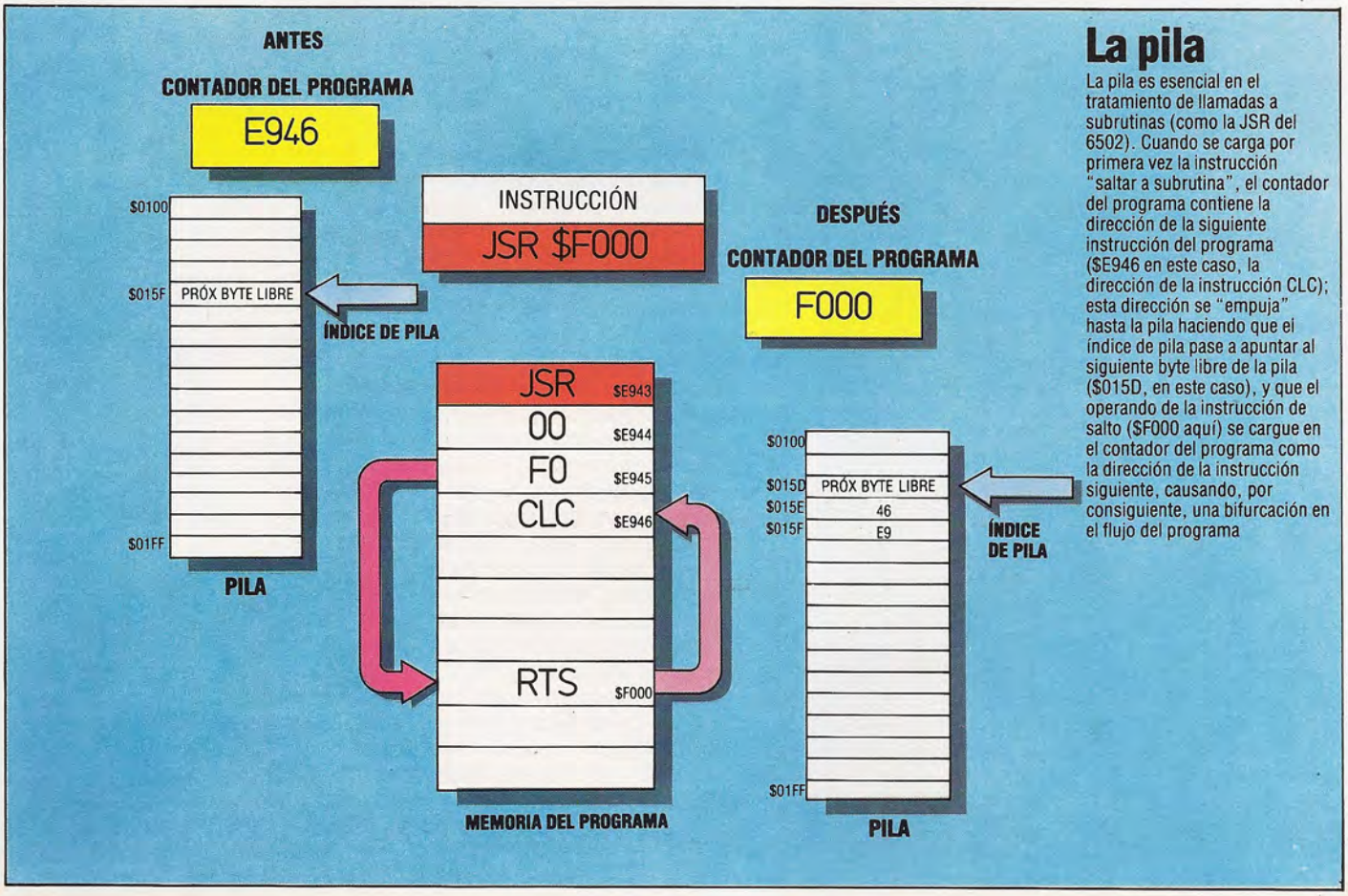
El primer fragmento de programa también constituye un ejemplo típico de los programas que utilizan la pila, en cuanto a que el número de instrucciones de empujón está exactamente equilibrado con el número de saltos. Esto no es esencial, pero el no observar este equilibrio de opuestos al escribir subrutinas podría acabar en un retorno incorrecto desde la subrutina y en el consiguiente fallo del programa. Este error se puede rastrear comparando el número de instrucciones de saltos y empujones del programa.

La versión Z80 de la rutina difiere notablemente de la del 6502 en un aspecto fundamental: el 6502 empuja en la pila un registro de un solo byte, mientras que el Z80 siempre empuja un registro de dos bytes. Cuando usted empuja o hace saltar el acumulador del Z80, también empuja o hace saltar el registro de estado del procesador, porque la CPU trata estos dos registros de byte único como un registro de dos bytes denominado AF (*Accumulator Flag*: acumulador-indicador).

Es una buena costumbre del programador comenzar las rutinas poniendo los contenidos de todos los registros de la CPU en la pila y sacándolos de la pila inmediatamente antes del retorno de la subrutina. Esto asegura que la CPU, después de la llamada a la subrutina, se mantenga exactamente en el mismo estado en que se encontraba antes de la misma, y significa que en la subrutina se puede utilizar cualquiera de los registros sin temor de corromper datos esenciales para el programa principal. Por ejemplo, consideremos esta subrutina:

	6502		Z80	
	LDA	LOC1	LD	A,LOC1
SUM	ADC	#$6C	ADC	A,$6C
GSUB	JSR	SUBR0	CALL	SUBR0
TEST	BNE	SUM	JR	NZ,SUM
EXIT	RTS		RET	
SUBR0	PHP		PUSH	AF
	PHA		PUSH	HL
	TXA		PUSH	DE
	PHA		PUSH	BC
	TYA		PUSH	IX
SUBR1	PHA		PUSH	IY
SUBR2	STA	LOC2	LD	(LOC2),A
	LDA	#$00	LD	A,$00
SUBR3	PLA		POP	IY
	TAY		POP	IX
	PLA		POP	DE
	TAX		POP	BC
	PLA		POP	HL
SUBR4	PLP		POP	AF
	RTS		RET	

Aquí el efecto de las instrucciones entre SUBR0 y SUBR1 es el de empujar hasta la pila los contenidos



La pila

La pila es esencial en el tratamiento de llamadas a subrutinas (como la JSR del 6502). Cuando se carga por primera vez la instrucción "saltar a subrutina", el contador del programa contiene la dirección de la siguiente instrucción del programa ($E946 en este caso, la dirección de la instrucción CLC); esta dirección se "empuja" hasta la pila haciendo que el índice de pila pase a apuntar al siguiente byte libre de la pila ($015D, en este caso), y que el operando de la instrucción de salto ($F000 aquí) se cargue en el contador del programa como la dirección de la instrucción siguiente, causando, por consiguiente, una bifurcación en el flujo del programa

Kevin Jones

actuales del registro, y el efecto de las instrucciones entre SUBR3 y SUBR4 es el de restaurar de nuevo aquellos contenidos en los registros. Las instrucciones substanciales de la subrutina son las dos que empiezan en SUBR2, pero la segunda de éstas es ineficaz, dado que las instrucciones siguientes cambian completamente el estado del acumulador.

Observe que las instrucciones PUSH y POP del Z80 pueden tomar como operando cualquier par de registros, pero el 6502 sólo puede operar sobre el acumulador (PHA y PLA) y sobre el registro indicador de estado (PHP y PLP). De ahí la necesidad de las transferencias registro-acumulador (TXA, TAX, TYA, TAY) de la versión 6502. Observe también que en la versión Z80 hemos cometido un error deliberado al no sacar todos los registros por el orden inverso al que fueron colocados. Ello ilustra el cuidado que es necesario tener en las operaciones de pila, pero demuestra, asimismo, que usted puede empujar a pila un registro y después hacer saltar ese valor hasta un registro diferente: una forma laboriosa pero a veces convenientes de realizar transferencias de datos entre registros.

Las funciones y los usos de los registros de la CPU será tema del próximo capítulo, con el cual terminaremos nuestro análisis general de las instrucciones del lenguaje assembly. Iniciaremos además el estudio de la aritmética en lenguaje máquina.

Ejercicios

1) Reescribir la segunda rutina que empleamos en las soluciones de los ejercicios anteriores, de tal forma que el mensaje de LABL1 se vuelva a almacenar en LABL1, pero en orden inverso, así:

LABL1 EJASNEM NU SE OTSE

Para esta inversión utilice la pila.

2) Desarrollar esta rutina de modo que las palabras del mensaje permanezcan en el orden original, pero que se inviertan los caracteres de cada palabra:

LABL1 OTSE SE NU EJASNEM

Respuestas a los ejercicios de p. 717

1) Esta subrutina almacena los números de $0F a $00 en orden descendente en el bloque de $10 bytes reservado en LABL1 por el pseudo-op DS.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DS $10	LABL1	DS $10
LABL2	DW $7100	LABL2	DW $C100
		OFFST	EQU $0F
BEGIN	LDY #$FF	BEGIN	LD IX,LABL1
	LDX #$10		LD B,OFFST
LOOP0	INY	LOOP0	LD (IX+0),B
	DEX		INC IX
	TXA	ENDLP0	DJNZ LOOP0
	STA LABL1,Y		LD (IX+0),B
ENDLP0	BNE LOOP0		RET
	RTS		

Las diferencias en cuanto a enfoque e instrucciones entre el Z80 y el 6502 son reveladoras. El 6502 utiliza el registro Y como un índice a la dirección LABL1, y el registro X como un contador de bucle y fuente de los datos a almacenar. Observe que el registro X es decrementado dos instrucciones antes de la comparación BNE e ENDLP0, pero como TXA (Transferir el contenido de X al acumulador) y STA no afectan al registro indicador de estado, la comparación actúa sobre los efectos de decrementar X.

La versión Z80 utiliza la modalidad de direccionamiento indirecto IX para retener la dirección de almacenamiento, y utiliza el registro B como contador y fuente de datos. En ENDLP0, en la versión Z80, vemos DJNZ LOOP0, que significa "decrementar el registro B y saltar a LOOP0 si el resultado no es cero". Esta instrucción es casi una estructura FOR...NEXT del lenguaje assembly.

2) Esta rutina copia el mensaje almacenado en LABL1 sobre el bloque que comienza en la dirección almacenada en LABL2. El valor $0D (el código ASCII para Return o Enter) se almacena al final del mensaje como terminador.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DB 'THIS IS A MESSAGE'	LABL1	DB 'THIS IS A MESSAGE'
TERMN8	DB $0D	TERMN8	DB $0D
LABL2	DW $7100	LABL2	DW $C100
CR	EQU $0D	CR	EQU $0D
ZPLO	EQU $FB		
BEGIN	LDA LABL2	BEGIN	LD IX,LABL1
	STA ZPLO		LD IY,(LABL2)
	LDA LABL2+1	LOOP0	LD A,(IX+0)
	STA ZPLO+1		LD (IY+0),A
	LDY $FF		INC IX
LOOP0	INY		INC IY
	LDA LABL1,Y		CP CR
	STA (ZPLO),Y	ENDLP0	JR NZ,LOOP0
	CMP CR		RET
ENDLP0	BNE LOOP0		
	RTS		

La versión 6502 utiliza al registro Y como un índice para la dirección indirecta ZPLO, en modo indirecto postindexado. Esta modalidad sólo es posible con el registro Y, y exige una dirección de operando de página cero; de ahí la inicialización de ZPLO y ZPLO + 1 con la dirección almacenada en LABL2. El sistema operativo de las máquinas 6502 emplea la mayoría de las posiciones de página cero; pero en el Commodore 64 las posiciones desde $FB hasta $FF no se utilizan, así como tampoco se usan las posiciones desde $70 hasta $8F en el BBC Micro, de modo que ZPLO se pone en una de estas posiciones. La versión Z80 emplea IX en modo indexado y IY en modo indirecto indexado.

Ambas rutinas utilizan una instrucción "comparar el acumulador" —CMP CR (6502) y CP CR (Z80)—, en la cual al contenido de éste se le resta el operando, afectando a los flags del indicador de estado (PSR). El contenido del acumulador se restaura luego, mientras el PSR muestra el resultado de la comparación. Cuando el acumulador contenga $0D (fin del mensaje), el resultado será poner a 1 el flag cero. De modo que la comparación ENDLP0 fallará y el control pasará a la instrucción de retorno.

Cambridge Connection

Camputers es otra empresa informática de primera línea, instalada en Cambridge, el "Silicon Valley" británico

John Shirreff

Camputers nació de la inspiración de un solo hombre. En 1976, Dick Greenwood empezó a trabajar como diseñador independiente de electrónica, aceptando contratos de firmas tales como Pye Telecoms. Hacia finales de los años setenta, Greenwood había creado su propia empresa y se había iniciado en un trabajo de desarrollo más especializado, también basado en contratos. La empresa acabó abarcando la creación de software, produciendo un Bar Management System, paquete para control de existencias que permitía a las fábricas de cerveza controlar sus ventas a bares y otros establecimientos.

La empresa pasó luego al diseño de microordenadores, concentrándose en proyectos basados en el chip Z80 de Zilog. En febrero de 1981, Greenwood creó Camtronic Circuits (que luego sería Camputers) y con una subvención estatal empezó a trabajar en el ordenador personal Lynx en el verano de 1981. El deseo explícito de Greenwood era el de "enseñarle al Z80A a bailar en torno a los problemas, no a arrastrarse a través de ellos".

En cuanto a la parte de la creación del hardware del proyecto, quien estaba a su cargo era John Shirreff, graduado por la Universidad de Cambridge, que se incorporó a la empresa después de trabajar en la industria de la música rock. La elaboración del software la llevaba Davis Jansons, quien escribió la versión de BASIC utilizada por la máquina.

Sistema para gestión
En la fotografía vemos el sistema modular Lynx Laureate diseñado para usuarios de gestión. Los 128 K de memoria incorporados le permiten aceptar CP/M

Cortesía de Camputers

Cuando apareció, en 1982, el Lynx se consideró una máquina de apariencia muy profesional. Empaquetado en una atractiva carcasa gris, con un teclado completo tipo máquina de escribir y disposición QWERTY, el ordenador venía equipado con una memoria estándar de 48 Kbytes que se podía ampliar a 192 Kbytes. La máquina tenía capacidad para visualizar hasta ocho colores diferentes, con una modalidad de alta resolución de 248 × 256 pixels. También tenía un altavoz incorporado para aprovechar al máximo sus capacidades de audio.

A decir verdad, el Lynx nunca alcanzó gran popularidad en Gran Bretaña, si bien las ventas en el extranjero alentaron a la empresa a seguir desarrollando el diseño básico. Al cabo de poco tiempo apareció el Lynx 96, que venía equipado con cerca de 37 Kbytes de RAM para el usuario, así como con la capacidad de soportar unidades de disco flexible de 5¼". Además, el Lynx 96 venía con efectos de sonido preelaborados. La máquina disponía, como opciones adicionales, de interfaces para impresora en serie y en paralelo.

Más recientemente, la empresa ha introducido una máquina destinada al sector de pequeña gestión del mercado del microordenador. El ordenador se denomina Lynx Laureate y se basa en el microprocesador Z80, puede operar bajo el sistema operativo CP/M, que proporciona al usuario acceso a la enorme cantidad de software CP/M que se ha escrito en el transcurso de la última década. A pesar de haber sido diseñado como una máquina de oficina pequeña, el Laureate es, sin embargo, compatible con los otros modelos Lynx y está dotado de un bus de ampliación de 40 vías que le permite utilizar toda la gama de paquetes para periféricos Lynx, incluyendo una impresora en paralelo, palanca de mando y software basado en ROM.

Camputers, dirigida por su actual presidente Stanley Charles, continúa desarrollando productos nuevos. En un futuro cercano, la empresa tiene planeado lanzar una versión alternativa de la máquina de oficina Laureate. Este sistema se podrá adquirir como un paquete integrado. También hay planes para relanzar el Lynx 48 en Gran Bretaña, con el nombre de Leisure. Éste estará dirigido específicamente al mercado del ordenador personal y para juegos, sector en el que la empresa considera que sus productos han sido injustamente ignorados. Con los ojos puestos en un futuro algo más lejano, Camputers está trabajando en una máquina que la empresa espera que se convierta en el competidor directo del Sinclair QL.

Devorando caracteres

Un procesador de textos permite que la tarea de producir cartas, informes o ensayos resulte de una gran sencillez para el usuario

En realidad los procesadores de textos no son otra cosa que versiones informatizadas de la máquina de escribir. El texto se digita a través del teclado del ordenador y aparece en la pantalla. Se pueden realizar modificaciones fácilmente sin necesidad de volver a teclear el documento entero y, una vez que la redacción es correcta, el texto se imprime por una impresora de ordenador.

Aparte del temeroso respeto que existe hacia los ordenadores, lo único que probablemente disuade a la gente de hacer uso de un procesador de textos es el problema de no sentirse cómodo al utilizar un teclado. Y, sin embargo, es más fácil iniciarse empleando un teclado con un procesador de textos que con una máquina de escribir normal. Los errores inevitables del mecanógrafo improvisado que al utilizar por primera vez una máquina de escribir usa sólo dos dedos pueden dar lugar a un lío considerable. Con un procesador de textos, estos errores se pueden enmendar en cuestión de segundos.

Para el tratamiento de textos se puede utilizar prácticamente cualquier micro personal, pero algunos no son tan prácticos como otros. En algunos casos ello se debe a que no existe un buen software para tratamiento de textos para esa máquina en particular; en otros, es el propio micro y sus periféricos los que no son adecuados para la tarea. Incluso el precio de un sistema simple para tratamiento de textos puede ser bastante elevado, ya que los accesorios necesarios pueden fácilmente costar el doble de lo que cuesta el propio ordenador. Por lo general el elemento más caro suele ser la impresora. Sin disponer de una buena impresora, de poco vale poseer un paquete para tratamiento de textos. En el futuro inmediato, casi todo el texto procesado ha de terminar imprimiéndose en papel; la edad del correo electrónico, en la que todo el texto se enviará directamente de un micro a otro, aún está muy lejana.

Hasta las impresoras más sencillas son bastante caras y, aun así, la calidad de la impresión que producen es relativamente baja. En muchos casos el tratamiento de textos exige una impresión de gran calidad. Al fin y al cabo, no tiene mucho sentido pasarse el tiempo con un procesador de textos tratando de que el aspecto de una carta sea perfecto, si el resultado se va a imprimir luego en una impresora matricial. Las impresoras margarita ofrecen una superior calidad de impresión, pero son lentas y caras, si bien los precios están descendiendo con bastante rapidez. Algunas máquinas de escribir electrónicas se pueden dotar de interfaces de modo que se las pueda utilizar como impresoras para ordenador.

Una solución al problema de tener acceso a una impresora de calidad es que varios amigos o un club informático compartan el costo de la máquina entre todos. Los usuarios aún tendrían la dificultad de conectar sus micros con interfaces para poder utilizar la impresora. Con algunos ordenadores esto es fácil porque poseen interfaces estándar, de modo que cada usuario sólo ha de adquirir un cable apropiado para conectar su micro a la impresora. Algunos micros, como los Commodore o los Atari, tienen interfaces que los limitan a las impresoras de su propia marca. Unos pocos micros, entre los cuales

Ian McKinnell

BBC Micro
La combinación del BBC Micro y el cartucho de disco Torch permite utilizar software de gestión, incluyendo el excelente Wordstar. El costo del sistema, sin embargo, es más elevado que el de muchos micros de oficina. Usando una impresora más económica y, en lugar del Wordstar, el software Perfect Writer ("el escritor perfecto"), incluido en el precio del cartucho Torch, se reduciría el precio de manera sustancial

está incluido el Sinclair Spectrum, no tienen interface para impresora como tal, por lo que se debe comprar la interface como accesorio adicional.

Disponiendo de una impresora adecuada, su siguiente tarea consiste en elegir el software apropiado para tratamiento de textos. Para las marcas más populares se produce una amplia gama de programas, mientras que las máquinas menos populares disponen sólo de uno o dos. La calidad varía considerablemente entre los distintos programas. Algunos son limitados y sólo permiten una edición sencilla, como insertar y eliminar textos. Otros permiten desplazar párrafos enteros de un lugar a otro del texto, presentarlo en la pantalla tal como aparecerá sobre el papel o justificar sus márgenes (igualar el largo de las líneas, espaciando convenientemente las palabras que las integran, como sucede en el texto que usted está leyendo en estos momentos).

Algunos procesadores de textos pueden buscar una palabra o una frase determinadas, de modo que resulta fácil corregir un error de ortografía que se hubiera deslizado a lo largo de un artículo. Para ciertos procesadores de textos se venden programas que verifican la ortografía de cada una de las palabras empleadas en un texto; y otros programas, como una lista de direcciones o una base de datos, pueden trabajar conjuntamente con el procesador de textos. Los programas para tratamiento de textos más sofisticados están diseñados para aprovechar las posibilidades ofrecidas por ciertas impresoras. Con frecuencia las impresoras matriciales pueden producir varios tipos diferentes de letras (como cursiva, negrita o pequeña) y, por tanto, algunos procesadores de textos permiten mezclar en el mismo artículo distintos tipos. Algunos procesadores de textos, aunque pocos, se pueden ampliar para utilizar la capacidad de producir gráficos de ciertas impresoras matriciales.

Las impresoras margarita pueden utilizar *espaciado proporcional*, dándoles más espacio a las letras anchas como la "w" y menos a las estrechas, como la "i", en vez de concederles a todas el mismo espacio, como lo haría una máquina de escribir convencional. Algunos procesadores de textos pueden utilizar esta facilidad, que hace que el texto resulte mucho más legible, y así y todo arreglárselas para justificar ambos márgenes. Esto es ideal para editar periódicos comunitarios o revistas internas de clubes, porque proporciona un aspecto profesional sin incurrir en el gasto de la correcta composición tipográfica. Las ruedas de impresión intercambiables permiten elegir diversos tipos de letras para elaborar el artículo.

El software para tratamiento de textos se vende en diversos formatos, incluyendo cinta, disco, cartucho y chip de ROM. No obstante, aún más importante es la forma en que se almacena el texto procesado, por lo general en cinta o en disco flexible. Aunque económica, la cinta es engorrosa, lenta y limita la longitud de los artículos a las dimensiones de la memoria disponible. Los discos son mejores porque son rápidos, fiables y permiten escribir artículos extensos. Actualmente están comenzando a aparecer nuevas formas de almacenamiento de datos. El microdrive de Sinclair, por ejemplo, a pesar de ser barato puede almacenar grandes cantidades de datos y hallar en cuestión de segundos una parte determinada del texto. Sin embargo, por ahora, son pocos los procesadores de textos para el Spectrum capaces de trabajar con los microdrives. Otro sistema interesante es la unidad de cinta que utiliza el Coleco Adam, un micro personal evidentemente diseñado teniendo en mente el tratamiento de textos, puesto que incluye una impresora margarita. Utiliza cintas de cassette modificadas para almacenar sus datos y en éstas se puede localizar cualquier palabra en segundos.

Guardar una copia procesada por tratamiento de textos en cinta o en disco permite escribir artículos largos en el transcurso de varios días, utilizar cartas estándar muchas veces, y conservar copias de todo el trabajo. También es una buena idea, al escribir artículos largos, hacer copias en diversas etapas mientras se los va escribiendo. Si esto no se hace existe el peligro de que algún accidente, como un corte de fluido eléctrico, destruya todo el trabajo realizado.

Algunos programas poseen instrucciones extrañas y combinaciones de teclas difíciles de memorizar, mientras que otros son de fácil uso. Algunos

Sinclair Spectrum
Éste es un sistema económico, y a la vez eficaz, para tratamiento de textos, pero aun así es demasiado caro. El Spectrum impone varias limitaciones, incluyendo un teclado pobre, la falta de una interface para pantalla y el uso de microdrives en vez de unidades de disco. No obstante, es posible añadir un teclado de mejor calidad. Tasword Two es uno de los pocos procesadores de textos para el Spectrum que funcionan con los microdrives

Ian McKinnell

micros, como el Sinclair Spectrum, poseen teclados de poca calidad que no contribuyen a facilitar la labor del usuario. Por suerte, para el Spectrum existen teclados accesorios que lo colocan en un estándar aceptable. Los que cuentan con teclas de función extras son útiles porque reducen el número de códigos de funciones que es necesario memorizar. La visualización en pantalla también puede ser una fuente de problemas. Varios micros muestran en la pantalla muy poco texto cada vez, lo que dificulta la escritura de manera notable. El Commodore Vic-20, por ejemplo, sólo visualiza texto en una anchura de 22 caracteres, mientras que las máquinas de oficina suelen tener una anchura de pantalla de 80 caracteres. El micro ideal para una fuerte carga de trabajo de proceso de textos es aquel que proporciona al menos una visualización de 25 por 80 caracteres y que utiliza una pantalla adecuada para visualizar clara y nítidamente la imagen.

El diseño de las letras que se ven en la pantalla del ordenador varía considerablemente. Algunas máquinas construyen cada letra con más puntos que otras máquinas, lo que hace que la visualización resulte más fácil de leer y también que sea más cómodo trabajar con ella. Son pocos los micros personales que utilizan letras de sólo seis puntos de anchura; la mayoría emplea una matriz de ocho puntos de anchura. Unas pocas máquinas de oficina poseen caracteres de 16 por 16 puntos, que son de una calidad superlativa.

Los sistemas personales más modestos son aptos para escribir cartas y otros textos cortos, pero se necesita más equipo para que resulte práctico escribir libros o informes largos. Si ha de procesar gran cantidad de texto, necesita un sistema con una pantalla, dos unidades de disco, una buena impresora, un teclado del tipo máquina de escribir y un buen software para tratamiento de textos. Ampliar un micro personal para que alcance este nivel resulta caro; de hecho, suele resultar más oneroso que adquirir un verdadero micro de oficina.

Los ordenadores de oficina ofrecen otras ventajas. Al estar diseñados para satisfacer necesidades de gestión, poseen buenos teclados, pantallas, unidades de disco o interfaces para impresora. Con todo, su ventaja más importante radica en la gran calidad del software creado para ellos. Tal gama de software de calidad existe porque la mayoría de los ordenadores de oficina utiliza alguno de los escasos sistemas operativos estandarizados, lo que significa que cada programa para tratamiento de textos se vende para muchas máquinas diferentes.

Con mucho, el paquete de gestión para tratamiento de textos más conocido en Gran Bretaña es el Wordstar, disponible para sistemas operativos CP/M, CP/M-86 y MS-DOS. El programa tiene algunas características sofisticadas, pero es caro, ya que cuesta alrededor de 20 veces más que un programa medio para micros personales. El costo de un procesador de textos de gestión es elevado, pero utilizar un micro personal para gran cantidad de proceso de textos por parte de una empresa pequeña o incluso de un eficiente escritor, es una falsa economía. El tiempo que se pierde empleando un sistema limitado superará la cantidad ahorrada.

El costo de los sistemas de gestión serios es elevado, pero está disminuyendo constantemente. Algunos micros personales se pueden ampliar para que utilicen sistemas operativos estándares tales como el CP/M, de modo que las personas que ya han invertido mucho dinero en un sistema de micro personal están empezando a disponer de software de calidad sin pagar demasiados extras. Otra tendencia está contribuyendo a reducir el costo del tratamiento de textos. Varias empresas están incluyendo en sus ordenadores, y sin recargo adicional alguno, programas de tratamiento de textos como el Wordstar.

La novedad más reciente en cuanto a tratamiento de textos es el tratamiento de textos sobre la marcha. Varios ordenadores de pilas se venden con procesadores de textos incorporados para permitir que los ajetreados ejecutivos escriban recordatorios y cartas en cualquier lugar y en cualquier momento. Estas máquinas no están al alcance del presupuesto de la mayoría de los usuarios de micros personales y no poseen muchos otros usos. Pero las máquinas de mano (*hand-held*) parecen sugerir que el tratamiento de textos se está convirtiendo en algo cada vez más común. Tal vez no sea sólo la máquina de escribir la que está condenada a desaparecer, sino también el lápiz y el papel.

Commodore 64
El Commodore 64 ofrece un procesador de textos con una unidad de disco de precio muy razonable. Tener una sola unidad de disco constituye una limitación, y el Commodore 64 sólo produce una visualización de 40 caracteres de ancho

Ian McKinnell

Diseño de sprites

La creación de sprites es una de las características más atractivas de los gráficos del Commodore 64

Un sprite es una gran forma gráfica móvil. Se diseña de modo muy similar a los caracteres definidos por el usuario de ocho por ocho que ya hemos analizado anteriormente en el curso (véase p. 713), pero se construye en un cuadriculado mucho más grande. Una vez definido un sprite, características tales como el color y la posición en pantalla se controlan mediante un juego de registros especiales en el chip de control de video (o VIC) del Commodore 64.

Un sprite se compone de 21 filas de 24 pixels. Cada fila consta de tres segmentos de ocho pixels y se representa mediante tres bytes de memoria, de modo que se requieren 63 bytes en total para almacenar los datos de un sprite. Al igual que sucede con los caracteres definidos por el usuario, cada pixel del cuadriculado del sprite que se iluminará en su forma final se representa mediante un uno binario (y los pixels no iluminados mediante un cero binario). Por consiguiente, para cada fila del sprite podemos calcular los equivalentes decimales de cada grupo de ocho dígitos binarios. Los diagramas que proporcionamos aquí muestran los cuatro sprites que se utilizarán en el juego *Subhunter*. Los números que figuran al lado de cada esquema son los equivalentes decimales que formarán las sentencias de datos para cada sprite (tal como se especifican desde la línea 6000 a la 6370 del programa).

Una vez el sprite ha sido definido y convertido en una serie de sentencias DATA, los datos se deben leer (READ) y colocar (POKE) en la memoria. Los datos de los sprites se pueden situar en varias posiciones de memoria. Por ejemplo, utilizando las posiciones que comienzan en 12288 el sprite se colocará en la zona para programas en BASIC, que va de 2048 a 40960. A medida que se va entrando un programa BASIC en el Commodore 64, éste va ocupando espacio de memoria desde la posición 2048 en adelante. Un programa tendría que tener un tamaño de 10 Kbytes antes de que alcanzara la posición 12288 y, por tanto, machacara los datos del sprite. No obstante, cuando un programa se está ejecutando, todas las variables utilizadas se almacenan en la zona superior a la empleada para almacenar el programa (las variables alfanuméricas o en serie, en particular, se van guardando hacia abajo desde el tope de la zona para programas BASIC). Como el juego *Subhunter* utiliza la variable de temporizado, TI$, la actualización regular de su valor y su subsiguiente almacenamiento pueden machacar la zona en que deseamos almacenar los datos del sprite.

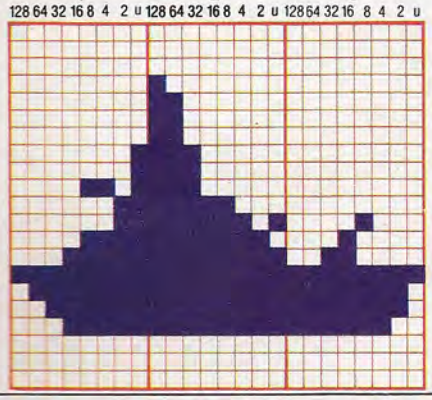
Una solución a este problema consiste en bajar el tope de la zona para programas BASIC colocándolo por debajo de la zona en la que se guardan los datos del sprite. El puntero de la dirección del tope de memoria se guarda en las posiciones 55 (byte bajo) y 56 (byte alto). Normalmente estas dos posiciones contienen los valores 0 y 160 respectivamente, que representan la dirección 40960. En la forma bajo-alto, la posición 12288 se consigna como 0 y 48. Podemos bajar el tope de la memoria a esta posición simplemente colocando (POKE) estos valores en las posiciones 55 y 56 al comienzo del programa (véase línea 90).

Blanco móvil
Un sprite se compone de 21 filas de tres bytes; estos bytes en realidad son patrones de bits, y en las líneas de datos del programa BASIC se almacenan como sus equivalentes en números decimales. Estos valores se ven junto a los diagramas de los sprites y en el listado del programa. El programa coloca (POKE) los valores en una zona exclusiva de RAM, donde son accedidos por el chip controlador de video como datos de sprite, visualizándolos y moviéndolos por la pantalla con un mínimo esfuerzo de programación

Punteros de sprites

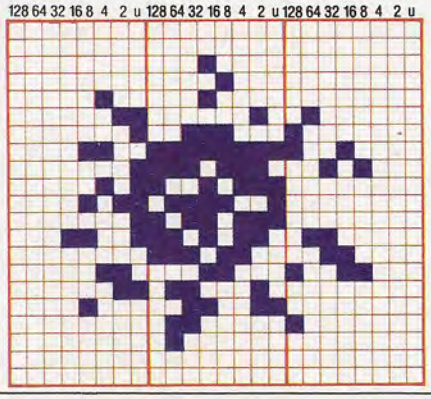
Dado que los datos del sprite se pueden situar en diversas partes de la memoria, se necesita un puntero que indique dónde empiezan los datos. Hay ocho punteros de sprites, guardados entre las posiciones 2040 (para el sprite 0) y 2047 (para el sprite 7). El valor guardado en cada puntero de sprites alude a la zona que contiene los datos del sprite mediante esta fórmula: comienzo de los 63 bytes de datos = (puntero de sprites) x 64. Los datos para el barco de nuestro programa empiezan en 12288 y se designará al barco como el sprite 0, de modo que el puntero de la posición 2040 es 192 (12288/64). El siguiente bloque de datos es para la explosión, el

BARCO-SPRITE 0



1	2	3
0	0	0
0	0	0
0	0	0
0	128	0
0	192	0
0	192	0
0	192	0
1	224	0
1	224	0
13	224	0
3	248	128
3	253	8
15	254	16
31	255	48
255	255	255
127	255	254
63	255	254
31	255	252
0	0	0
0	0	0
0	0	0

EXPLOSIÓN-SPRITE 1



1	2	3
0	0	0
0	0	0
0	16	0
0	8	0
4	16	0
3	2	64
1	56	128
12	255	144
1	238	40
5	151	0
11	121	0
1	183	0
25	214	96
0	236	48
6	24	152
3	98	0
8	51	0
0	96	128
0	64	0
0	0	0
0	0	0

sprite 1. Si ponemos el puntero de la posición 2041 en 193, entonces los datos deben comenzar en 12352. Los valores que utilizamos son éstos:

Número de sprites	0	1	2	3
Puntero de sprites	192	193	194	195
63 bytes de datos del sprite	de 12288 a 12350	de 12352 a 12414	de 12416 a 12478	de 12480 a 12542

Observe que al final de cada bloque de datos de sprite queda un byte sin utilizar. Las partes del programa que leen los datos del sprite de la memoria y especifican los punteros de sprites están contenidos en las líneas 2000 a 2210.

Manipulación de sprites

El chip de control de video (VIC: *VIdeo Control chip*) tiene varios registros especiales que se utilizan para controlar los sprites. La primera posición del chip VIC es la 53248 y para nuestro programa es más fácil describir las posiciones de todos los demás registros en relación a ésta. Si hacemos que V=53248, la siguiente posición del chip VIC, la 53249, se puede denominar V+1, y así sucesivamente. V se debe definir, junto con otras variables, en una etapa anterior (véase línea 100).

El color de cada uno de los sprites se establece colocando (POKE) un número de código de color (comprendido entre 0 y 15) en un registro especial. Cada uno de los ocho sprites posee su propio registro de color; éstos van de V+39 a V+46. Por ejemplo, para colorear de negro el barco simplemente colocamos (POKE) el código de color 0 en la posición V+39. Los otros sprites se pueden colorear de la misma manera (véase desde línea 2220 a 2250).

El posicionamiento de los sprites en la pantalla lo analizaremos con mayor profundidad en el próximo capítulo. Por ahora basta con saber que la coordenada *x* del sprite 0 se guarda en la posición V, la coordenada *y* para el sprite 0 se guarda en la posición V+1; las coordenadas *x* e *y* para el sprite 1 se guardan en V+2 y V+3 respectivamente, y así sucesivamente hasta la posición V+15 (véase desde línea 2260 a 2280).

Los sprites se pueden ampliar horizontalmente, verticalmente o en ambas direcciones en un factor de dos. Los sprites del barco y del submarino podrían parecen más bien aplastados horizontalmente, pero ahora los ampliaremos al doble de su longitud original. De hecho, los cuatro sprites se ampliarán horizontalmente. El registro del chip VIC que controla la expansión horizontal es V+29, que es más fácil de utilizar que los otros registros que hemos visto. En vez de emplear ocho registros diferentes para controlar los atributos de cada uno de los ocho sprites, todo lo que hay que hacer es activar o desactivar la función. Por lo tanto, sólo se requiere un bit del registro para controlar la ampliación horizontal de cada sprite. Para ampliar horizontalmente un sprite, hay que poner a 1 el bit correspondiente del registro V+29. La siguiente tabla muestra el POKE que se requiere para ampliar los cuatro sprites que hemos definido:

Número de sprite	7	6	5	4	3	2	1	0	
Contenido de V+29	0	0	0	0	1	1	1	1	=15 (decimal)

La ampliación en dirección vertical se controla mediante V+23. La explosión, el sprite 1, se amplía vertical y horizontalmente, doblando, por tanto, su tamaño (véase desde línea 2290 a 2310).

Nuestra tarea final consiste en activar los sprites requeridos. Para activar o desactivar cada sprite se utiliza un solo bit del registro, V+21. En el juego *Subhunter* inicialmente sólo se encienden el barco y el submarino (líneas 2310 a 2360).

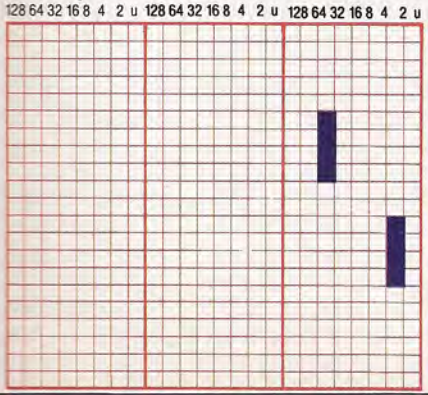
Después de entrar todo el programa, deberá comprobar que los datos de los sprites se hayan leído correctamente. Para hacerlo, ejecute el programa e interrúmpalo mediante RUN o STOP cuando en la pantalla aparezca el reloj. Entrando las siguientes sentencias, sin los números de línea, se posicionarán y visualizarán los cuatro sprites.

```
POKEV,160          (coordenada del barco)
POKEV+2,240:       (coordenadas x e y de
  POKEV+3,100      la explosión)
POKEV+4,160:       (coordenadas x e y de
  POKEV+5,100      la carga de profundidad)
POKEV+6,100:       (coordenadas x e y
  POKEV+7,100      del submarino)
POKEV+21,15        (enciende sprites 0–3)
```

Si el programa se interrumpe con un mensaje "OUT OF DATA ERROR", verifique cuántos números hay en las sentencias DATA. Debería haber 63 para cada sprite. Si el programa se cuelga y el teclado no responde, asegúrese de que en la línea 100 se haya declarado V. Siempre es una buena idea guardar (SAVE) su programa antes de ejecutarlo.

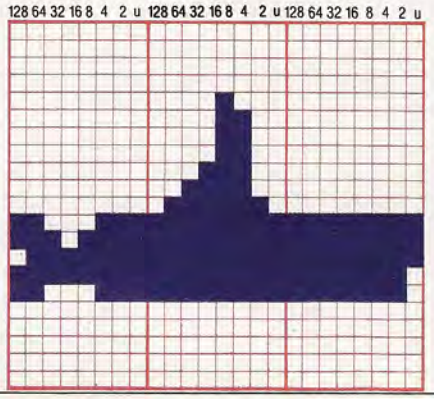
```
1 REM *** GRAFICOS C64 **********
90 POKE 55,0:POKE 56,48:CLR: REM
   BAJAR TOPE MEM
100 V=53248:FL=0:SC=0
110 GOSUB 1000 :REM CREACION
   PANTALLA (véase p. 695)
120 GOSUB 2000 :REM CREACION
   SPRITES
2000 REM *** CREACION SPRITES ***
2020 REM ** LEER DATOS BARCO **
2030 FOR I=12288 TO 12350
2040 READ A:POKE I,A:NEXT I
2060 REM ** LEER DATOS EXPLOSION **
2070 FOR I=12352 TO 12414
2080 READ A:POKE I,A:NEXT I
2100 REM ** LEER DATOS CARGA **
2110 FOR I=12416 TO 12478
2120 READ A:POKE I,A:NEXT I
2140 REM ** LEER DATOS SUBM **
2150 FOR I=12480 TO 12542
2160 READ A:POKE I,A:NEXT I
2180 REM ** ESTABLECER PUNTEROS **
2190 POKE 2040,192:POKE 2041,
   193:POKE 2042,194:POKE 2043,195
2220 REM ** ESTABLECER COLORES **
2230 POKE V+39,0:POKE V+40,1:POKE
   V+41,0:POKE V+42,0
2260 REM ** INIC COORD. BARCO ***
2270 POKE V+1,80:X0=160
2290 REM ** EXPANDIR SPRITES ***
2300 POKE V+29,15:POKE V+23,2
2320 REM ** ENCENDER SPRITES ***
2330 POKE V+21,9
2340 RETURN
2350 :
6000 REM ** DATOS BARCO ***
6010 DATA 0,0,0,0,0,0,0,0,0
6020 DATA 0,128,0,0,192,0,0,192,0
6030 DATA 0,192,0,1,224,0,1,224,0
6040 DATA 13,224,0,3,248,128,3,253,8
6050 DATA 15,254,16,31,255,48,255,
   255,255
6060 DATA 127,225, 254,63,255,254,
   31,255,252
6070 DATA 0,0,0,0,0,0,0,0,0
6100 REM ** DATOS EXPLOSION ***
6110 DATA 0,0,0,0,0,0,0,16,0,0,8,0,4,16
6120 DATA 0,3,2,64,1,56,128,12,255,144
6130 DATA 1,238,40,5,151,0,11,121,0,1
6140 DATA 183,0,25,214,96,0,236,48,
   6,24
6150 DATA 152,3,98,0,8,51,0,0,96,128,0
6160 DATA 64,0,0,0,0,0,0,0
6200 REM ** DATOS CARGA PROF ***
6210 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6220 DATA 0,0,0,32,0,0,32,0,0,32,0,
   0,32,0
6230 DATA 0,0,0,0,0,0,0,0
6240 DATA 2,0,0,2,0,0,2,0,0,2,0,0,2,0
6250 DATA 0,0,0,0,0,0,0,0
6260 DATA 0,0,0,0,0,0,0,0
6300 REM ** DATOS SUBMARINO ***
6310 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6320 DATA 0,8,0,0,12,0,0,12,0
6330 DATA 0,12,0,0,28,0,0,60,0
6340 DATA 0,126,0,199,255,255
6350 DATA 239,255,255,127,255,255
6360 DATA 255, 255, 254, 199,255,254
6370 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

CARGAS DE PROFUNDIDAD-2



1	2	3
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	32
0	0	32
0	0	32
0	0	32
0	0	0
0	0	0
0	0	2
0	0	2
0	0	2
0	0	2
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

SUBMARINO-SPRITE 3



1	2	3
0	0	0
0	0	0
0	0	0
0	0	0
0	8	0
0	12	0
0	12	0
0	12	0
0	28	0
0	60	0
0	126	0
199	255	255
239	255	255
127	255	255
255	255	254
199	255	254
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Liz Dixon

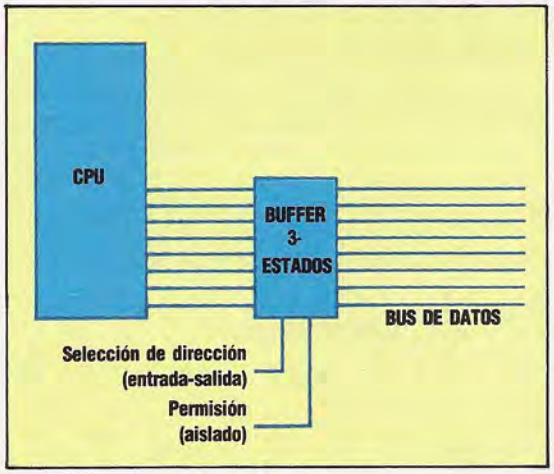
De aquí para allá

En este capítulo veremos cómo se produce el intercambio de datos entre la CPU y la memoria

Cada posición de memoria de un ordenador personal normalmente se compone de ocho bits. Los datos se pueden transferir, de a ocho bit por vez, mediante una serie de ocho líneas paralelas, a la CPU, donde los datos se pueden entonces utilizar de acuerdo a la instrucción de programa que los requiere. Los datos también se pueden enviar en la dirección contraria, para almacenarlos en una posición de memoria. La instrucción en código máquina LDA $1234 hace que el número contenido en la posición $1234 se envíe hasta la CPU por el bus de datos. STA $1234 hace que se envíe un número desde la CPU, nuevamente por el bus de datos, y que se lo almacene en la posición $1234.

Por consiguiente, el bus de datos debe permitir transferencias en ambas direcciones. En ciertas ocasiones también es importante aislar la CPU del bus de datos. Por lo tanto, cada línea del bus de datos puede estar en uno de los tres estados (INPUT —entrada—, OUTPUT —salida— o ISOLATE —aislado—). A fin de conseguir la necesaria conmutación entre estos estados, cada línea del bus de datos posee un pequeño circuito electrónico llamado *dispositivo 3-estados (tri-state)*.

Ocho de esos dispositivos 3-estados se combinan formando un único circuito integrado. El diagrama siguiente muestra cómo este circuito integrado enlaza al bus de datos con la CPU. Asimismo, el diagrama ilustra las líneas "permisión" (*enable*) y "selección de dirección", que ponen los ocho 3-estados en el estado operativo requerido. Los circuitos de este tipo también se pueden utilizar para conectar otros dispositivos, como periféricos de entrada/salida, con el bus de datos.



Siempre que deseamos cargar el contenido de una posición determinada nos referimos a la posición que ocupa mediante su dirección. Cada posición de ROM y RAM posee su propio número exclusivo que hace referencia a ella. Veamos ahora, a nivel de hardware, cómo se puede acceder a cada posición para realizar una transferencia de datos.

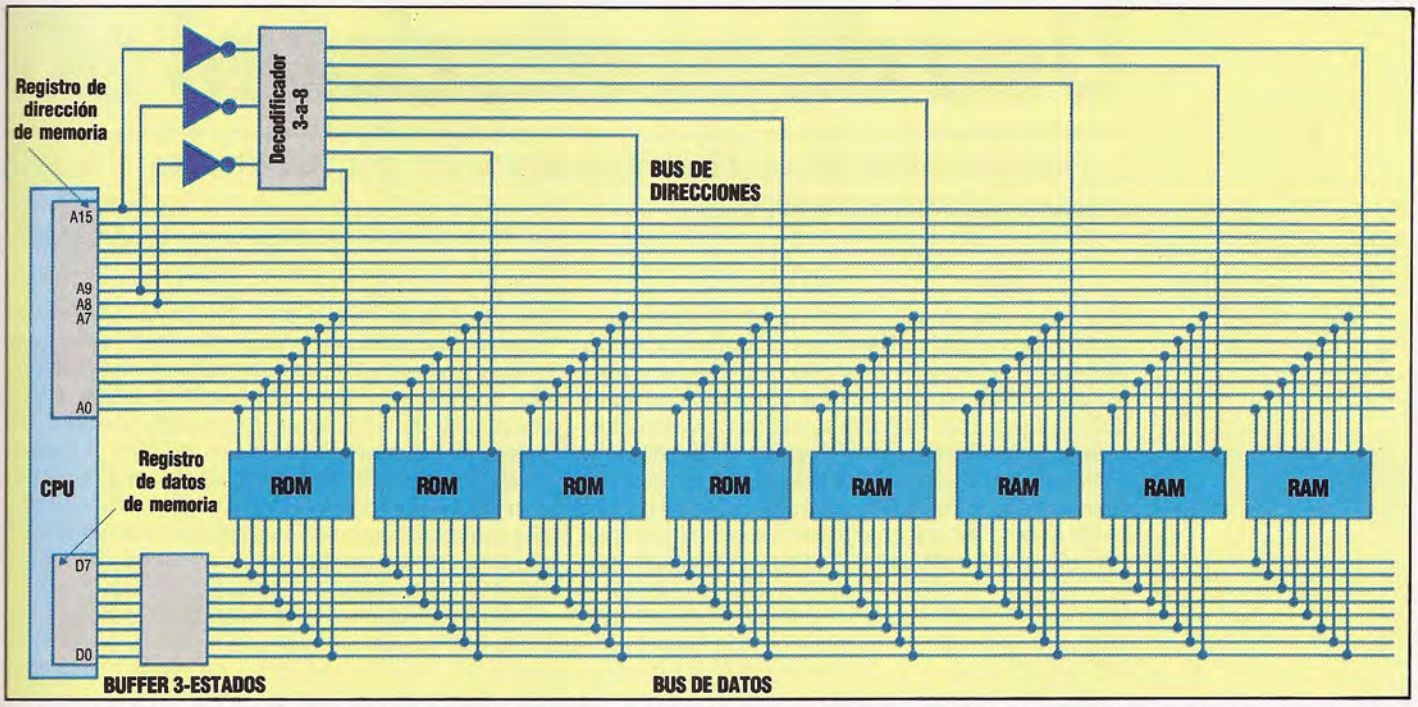
La mayoría de los microordenadores posee una segunda vía entre la CPU y la memoria, que se denomina *bus de direcciones*. Normalmente el bus de direcciones no posee ocho líneas sino 16. Esto significa que se pueden especificar hasta 65 536 direcciones distintas (2^{16}=65 536). Es decir, utilizando un bus de direcciones de 16 bits se puede acceder a 64 Kbytes de memoria. Se puede pensar en el total de memoria como si estuviera dividida en módulos, conteniendo cada uno de ellos 256 posiciones. Los ocho bits inferiores de la dirección se pueden entonces utilizar para hallar la posición concreta dentro de un módulo dado. El módulo propiamente dicho se puede seleccionar utilizando algunos (o la totalidad) de los ocho bits de dirección restantes.

Si consideramos el sencillo ejemplo de un microordenador con un tamaño total de memoria de dos Kbytes, podemos ver cómo tiene lugar la selección de cualquier posición determinada. Dado que cada módulo de memoria contiene 256 posiciones, nuestro ordenador de dos Kbytes requerirá ocho módulos. Para nuestro pequeño ordenador daremos por sentado que la memoria está dividida en ROM y RAM a partes iguales.

La dirección de la posición requerida se guarda en un registro de 16 bits especial de la CPU, llamado registro de dirección de memoria o MAR (*Memory Address Register*). Como los ocho bits inferiores de la dirección seleccionan una posición particular dentro de cualquier módulo, las ocho líneas inferiores del bus de direcciones se pueden conectar a cada uno de los módulos de la memoria. Para seleccionar un módulo determinado, ya sólo necesitamos otros tres bits (2^3=8). Este código de tres bits se debe decodificar en ocho líneas de salida.

El diagrama muestra cómo los módulos de memoria se enlazan con la CPU a través de estos buses de datos y direcciones. Cada módulo de memoria tiene una única línea hacia él desde el decodificador de tres(bits)-a-ocho (líneas). Tres de los bits de dirección superiores se utilizan para determinar qué módulo se ha de seleccionar. Si se hubieran de agregar más módulos de RAM, entonces para seleccionar cualquier módulo individual, se podrían usar algunos de los cinco bits superiores restantes.

Después de haber visto cómo se puede seleccionar una posición determinada de memoria y transferir los datos, analicemos cómo la CPU lleva a cabo una instrucción de lenguaje máquina. Todo programa en lenguaje máquina se suele almacenar en posiciones consecutivas. Una instrucción puede requerir dos o tres bytes para su almacenamiento. Una instrucción como ADD $13FF significa "sumarle al acumulador el contenido de la posición cuya dirección hexadecimal es $13FF". Esta instrucción requeriría tres bytes: uno para guardar el código bi-

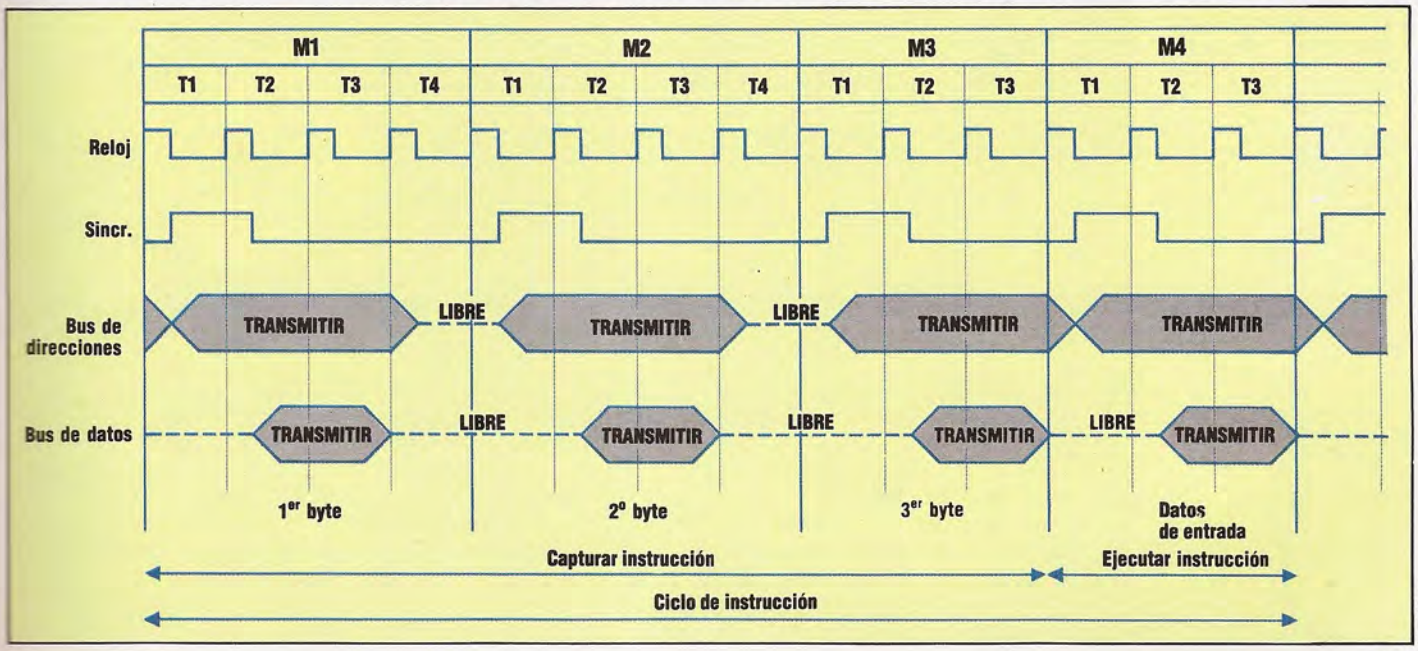


nario de la instrucción ADD y dos para contener la dirección de 16 bits, $13FF. Vamos a suponer que se almacena en las posiciones $1000, $1001 y $1002.

Antes de que la instrucción se pueda procesar, debe ser "capturada" (operación *fetch*) de la memoria. Ello requiere tres accesos separados para llevar los tres bytes a lo largo del bus de datos hasta la CPU. Al completar el ciclo de captura, la instrucción entera está en un registro especial dentro de la CPU. Lo único que hay que hacer entonces es decodificar la instrucción y obedecerla. La instrucción del ejemplo requiere otro acceso a memoria para obtener el contenido de la posición $13FF de modo que pueda ser sumado al acumulador.

Todos los fabricantes de ordenadores publican las características de sus procesadores en forma de diagramas de tiempos. Éstos muestran el orden de los acontecimientos para numerosas operaciones diferentes del ordenador. Podemos trazar un diagrama de tiempos para los ciclos de captura y ejecución de una instrucción de lenguaje máquina. El temporizado de las operaciones se controla mediante los impulsos del reloj (véase p. 726) y nuestro gráfico muestra que en este sistema imaginario el bus de direcciones es activado por el flanco de cabeza del impulso de sincronización, mientras que el bus de datos es activado por el flanco de cola de sincronización. El impulso de sincronización, por su parte, es activado por el flanco de cola del primer impulso de reloj de cualquier fase operativa, o ciclo de máquina. Los ciclos tienen duraciones diferentes porque el procesador necesita más tiempo para decodificar el byte de código de instrucción que para manipular los bytes de los operandos: el código se debe decodificar de inmediato porque especifica el número de bytes de los operandos.

Capturar y ejecutar
Una instrucción en lenguaje máquina que conste de un byte de código de instrucción seguido de dos bytes de operandos se trata en un ciclo de instrucción que consta de fases de captura y de ejecución. Durante la fase de captura, el bus de direcciones accede a las posiciones de memoria que contienen la instrucción, y el bus de datos lleva los bytes de instrucción hasta la CPU. Allí, los buses de datos y de direcciones todavía están atareados durante la fase de ejecución, porque la instrucción que se está ejecutando provoca un acceso a la memoria



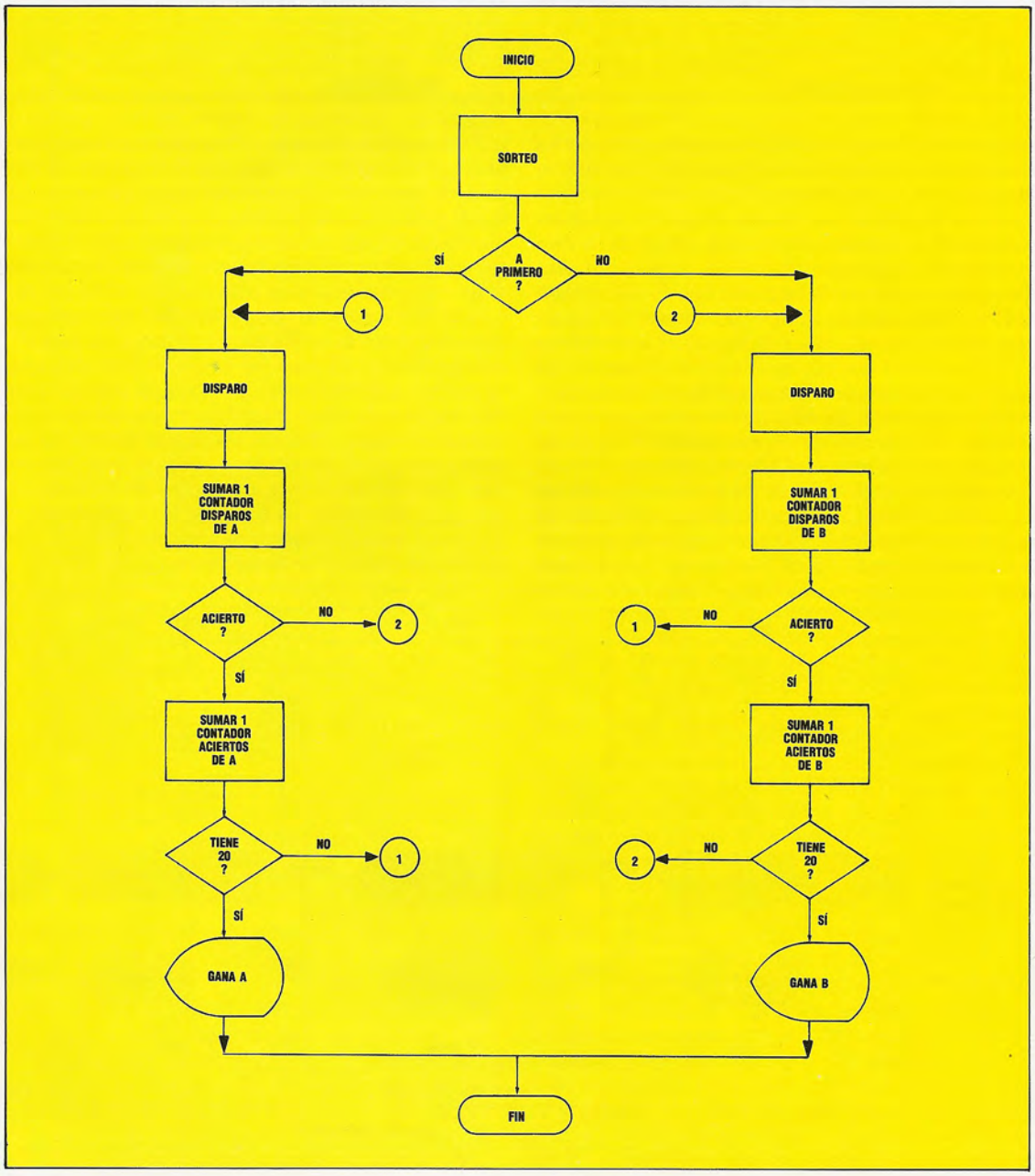
Liz Dixon

Uso de contadores

Un juego de tiro al blanco se puede prestar muy bien a ser controlado por programa

Dos jugadores, A y B, practican el tiro al plato, y deciden que ganará el primero que consiga veinte aciertos. Se sortea quién comienza el juego. Empieza a disparar el jugador seleccionado: A. Éste va tirando hasta que falla un plato; en ese momento cede el turno de disparos al jugador B, que dispara hasta fallar, y así sucesivamente. El juego termina cuando uno de los jugadores llega a contabilizar 20 en su cuenta de aciertos. Entonces se visualizará, por una parte, cuál de los jugadores que participaban en el juego ha ganado y, por otra, cuántos disparos le han hecho falta para conseguir la victoria.

En este caso serán necesarios cuatro contadores, dos por jugador: uno contará el número de disparos efectuados y el otro el número de aciertos. En esta ocasión se prescindirá de cualquier intervención manual, todo será controlado por programa. A este fin, el sorteo del jugador que empieza a disparar, y si el disparo es o no fallido, se determinará según un número aleatorio.



De la misma estirpe

Después del discreto éxito obtenido por el Oric-1 en su lanzamiento en 1983, Oric Products ha creado un nuevo y mejorado modelo

Equipado con un potente BASIC estilo Microsoft, una puerta para impresora Centronics incorporada y un conector para pantalla RGB estándar, el Oric-1 parecía originalmente una buena inversión. Sin embargo, la falta de un buen software, junto con algunos notorios errores en la ROM de BASIC, hicieron que la máquina fuera recibida con indiferencia.

Ahora Oric Products International ha corregido los errores fundamentales y ha relanzado la máquina con el nombre de Oric Atmos. El antiguo teclado tipo calculadora ha sido sustituido por teclas más profesionales, tipo máquina de escribir y de recorrido total, y se ha rediseñado la carcasa con una elegante combinación en rojo y negro. La disposición del teclado es la misma que en el Oric-1, con una tecla Function adicional, que aún está desconectada pero que se suministra en virtud de una "futura ampliación".

El Atmos utiliza un microprocesador 6502 y en operación normal tiene 37 Kbytes de RAM libres para programas BASIC. El Atmos puede visualizar ocho colores y posee una resolución máxima de 240 × 200 pixels. El juego de caracteres se mantiene en RAM, lo que permite al usuario la definición de cualquier carácter. También existe un juego de caracteres alternativo, que ofrece gráficos de bloque de tipo teletexto. A diferencia del Spectrum, que mantiene un archivo de atributos independiente en RAM, el Atmos utiliza "atributos en serie". Éstos emplean menos memoria pero se visualizan en la pantalla como espacios en blanco, por lo que hay que tener mucho cuidado al planificar las visualizaciones en pantalla.

La ROM del Atmos contiene cuatro sonidos preestablecidos (ZAP, PING, SHOOT y EXPLODE) y éstos dan efectos de sonido al estilo de los juegos recreativos. Las órdenes MUSIC, PLAY y SOUND le permiten al usuario sacar el máximo provecho del sofisticado chip de sonido del Oric, disponiendo de una amplia gama de parámetros para hacer variar el sonido. El volumen oscila desde muy débil a muy fuerte, y los tres canales de tonos y un canal de ruido proporcionan una escala de siete octavas.

El BASIC del Oric original incluyó varios molestos *bugs*. La orden TAB no funcionaba correctamente y la visualización a menudo se alteraba por las órdenes de sonido. El Oric también introducía códigos de control erróneos al evaluar la función STR$, y

Ian McKinnell

El sistema Atmos
El Oric-Atmos es un ordenador personal de moderado precio con 48 K de memoria, gráficos en color y efectos de sonido. Oric fabrica dos accesorios para el Atmos, ambos en colores que hacen juego con el conjunto. La unidad de disco proporciona una alternativa rápida en contraposición a la grabadora de cassette, y la impresora-plotter puede trazar líneas o texto en color

El Oric-1
El Atmos es una versión mejorada del Oric-1. Utiliza la misma placa de circuito impreso pero tiene un chip de ROM diferente que contiene una versión perfeccionada de BASIC. Estas modificaciones son suficientes para hacer del Atmos una máquina mucho mejor. Es necesario advertir a los usuarios que gran parte del software de Oric no funcionará en el Atmos, por lo que deberán reconvertir sus programas favoritos

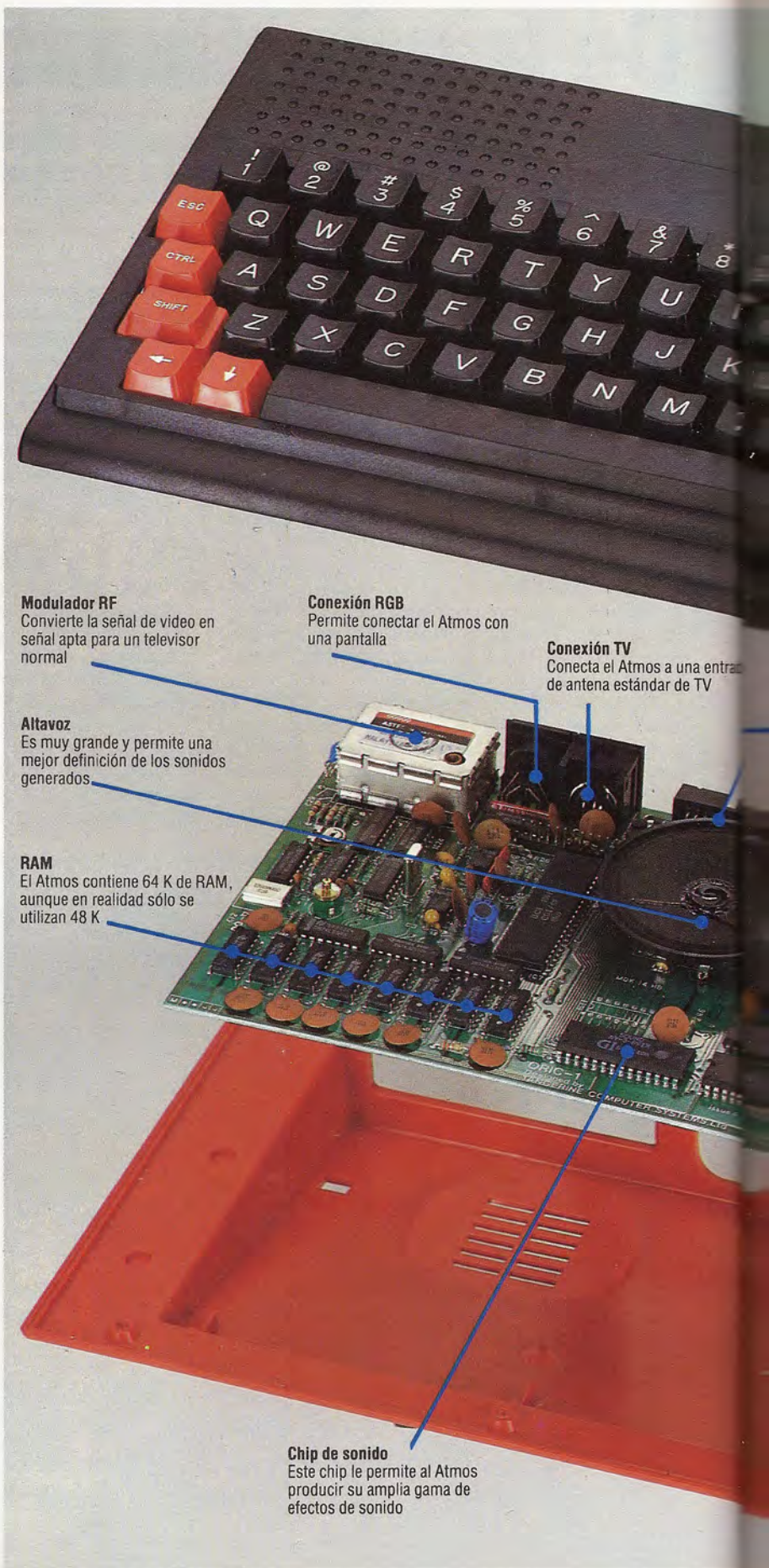
daba resultados incorrectos al usar LEN o VAL. La nueva ROM ha solventado estas dificultades. Un inevitable efecto secundario de estas mejoras es el hecho de que es poco probable que los programas del Oric en lenguaje máquina funcionen con el Atmos, porque varias rutinas de ROM han sido reubicadas en memoria.

El BASIC es una versión ampliada del dialecto Microsoft, desarrollada por Tansoft a partir del BASIC Tangerine original. Admite la estructura IF... THEN...ELSE completa (en esta instrucción el BASIC del Oric-1 tenía un error en el segmento ELSE) y también la instrucción de bucle REPEAT...UNTIL. Una característica inusual es el disponer de las instrucciones POP y PULL, que se utilizan para saltar fuera de rutinas GOSUB y REPEAT...UNTIL sin provocar un mensaje de error. El BASIC del Oric-1 no permitía que el usuario utilizara POKE con un valor hexadecimal; en la nueva ROM esto también se ha corregido.

Las primeras versiones del Atmos tuvieron algunos problemas con la ROM nueva. Al diseñar el nuevo chip, Oric incluyó una rutina para verificación de errores mejorada para cargar las cintas de cassette. Esta rutina era tan eficaz que los usuarios enseguida descubrieron que el software encontraba errores en programas de casi todas las máquinas de cassette. Sin embargo, la ROM diseñada del Oric permite cargar programas de manera satisfactoria.

Coincidiendo con el nuevo Atmos, Oric ha vuelto a diseñar su impresora-plotter, que ahora tiene un acabado en rojo y negro igual que el ordenador. Cuatro pequeños lápices de punta esférica (negro, rojo, verde y azul) están dispuestos en una cabeza giratoria para trazado de gráficos; todos los colores se pueden seleccionar bajo control de software. La impresora-plotter tiene una velocidad lenta para textos, de 12 caracteres por segundo, pero imprime sobre papel normal.

La unidad de microdisco, esperada durante tanto tiempo, también ha sido rediseñada con los colores del Atmos. Oric ha optado por los discos Hitachi de 3"; éstos están metidos dentro de una carcasa rígida de plástico. El Atmos puede utilizar hasta cuatro unidades: una unidad maestra individual con un sistema de interface de disco incorporado y hasta tres unidades esclavas. Por el momento no se



Tecla Function
Esta tecla no está conectada

Interface para impresora
Interface en paralelo tipo Centronics

Puerta de ampliación
Contiene un bus en paralelo de 34 canales para conectar a la unidad de disco

Disipador
Esta placa disipa el calor excesivo que se genera

Versión mejorada de BASIC
El chip individual de ROM contiene la nueva versión del BASIC Tansoft

CPU
La unidad central de proceso es un microprocesador 6502A

Chris Stevens

Unidad de disco
La unidad de disco Oric utiliza discos de 3 pulgadas. Éstos vienen en una carcasa rígida de protección. La unidad posee una capacidad de 160 K en cada cara del disco. Se puede dar la vuelta a los discos y utilizarlos por la otra cara para dar un total de 320 K por disco. Las unidades Oric sólo pueden tratar archivos secuenciales

han producido unidades esclavas, pero se espera que éstas aparezcan pronto. La unidad de disco viene con un transformador de potencia separado lo suficientemente potente para dos unidades de disco y el propio ordenador. En las primeras versiones del sistema operativo en disco surgían problemas cuando se conectaban simultáneamente la impresora y la unidad de disco. Todo intento de editar una línea de programa hacía que ésta se borrara del listado; lo mismo sucedía con todas las líneas de programa numeradas. Oric afirma que las posteriores versiones del sistema operativo han superado esta dificultad.

A pesar de que las primeras versiones han experimentado problemas a la hora de utilizar impresora y cassette, parece ser que Oric Products ha procedido con extremo celo al producir el Atmos y sus periféricos. El equipo de diseño ha tomado nota de las críticas de que fue objeto al anterior Oric-1 y ahora ha enmendado la mayoría de los errores. Los usuarios del Oric-1 estaban muy desatendidos por los productores de software y, para remediar esta situación, Oric Products le ha encargado a Tansoft la producción de un conjunto de programas para utilizar con la unidad de disco. Si la producción de software se incrementa, el Atmos, con toda seguridad, captará un sector más amplio de un mercado que es muy competitivo.

Chris Stevens

Impresora-plotter
Constituye un excelente accesorio para el Atmos. Utiliza cuatro lápices de punta esférica para trazar líneas y texto en color. Puede dibujar textos de tamaño minúsculo hasta varias pulgadas de altura. Entre sus inconvenientes se incluyen la anchura del papel, de sólo 4½ pulgadas, la reducida velocidad y el elevado precio de los lápices de punta esférica. La impresora-plotter no es adecuada para dibujar superficies sólidas de color

ORIC ATMOS

DIMENSIONES
278×178×50 mm

CPU
6502

MEMORIA
48 Kbytes de RAM, 16 Kbytes de ROM

PANTALLA
26 líneas de 40 columnas en modalidad de textos y hasta 200×240 pixels en ocho colores

INTERFACES
Una puerta de ampliación para una interface para impresora Centronics, puerta para cassette y conexión RGB

LENGUAJES DISPONIBLES
BASIC ampliado y FORTH

TECLADO
58 teclas tipo máquina de escribir. La tecla Function no está conectada

DOCUMENTACION
El manual es exhaustivo y está escrito en un agradable estilo coloquial, con la clara intención de que el principiante se inicie fácilmente en la programación en BASIC. Para el usuario más avanzado hay capítulos que cubren programación en lenguaje máquina y técnicas avanzadas de entrada-salida, así como algunos apéndices que proporcionan una completa información técnica

VENTAJAS
El Atmos tiene una amplia gama de facilidades de las que no disponen máquinas de precio más elevado. El BASIC es conciso y contiene numerosas instrucciones que hacen que la programación resulte más sencilla

DESVENTAJAS
El método de visualización en pantalla resulta difícil cuando se trabaja en modalidad de alta resolución. Las unidades de disco son decepcionantes, ya que sólo son aptas para el acceso secuencial

Posiciones clave

En este capítulo estudiaremos dos métodos de acceso a los ficheros aleatorios: el acceso indexado y el acceso mediante hash

Si ha experimentado con los archivos de acceso aleatorio, habrá observado que la programación con archivos es muy sencilla. Usted puede determinar, leer o escribir cualquier registro determinado sin molestarse con los incómodos procedimientos necesarios para recuperar datos almacenados secuencialmente. Sin embargo, los métodos de inserción y eliminación que detallamos en la página 724 no constituyen el procedimiento más eficaz de tratar los archivos aleatorios; acceder a la información utilizando un índice es un método mejor.

Para crear un índice se debe especificar como clave un campo determinado del registro. Así pues, el índice construido se compone de los valores del campo de clave para cada registro, junto con su correspondiente número de registro. Por tanto, si el registro correspondiente a Juan Ruiz es el número 17, y es el octavo de una lista por orden alfabético, la matriz de índice almacenará un 17 en la octava posición. Si el índice está clasificado regularmente, buscar un registro puede ser un proceso muy rápido.

El índice se suele almacenar en RAM con el objeto de que sea accesible de forma inmediata. Se puede generar en el acto, con una rutina que lea todo el archivo, poniendo cada campo de clave en una matriz. Este índice se puede entonces clasificar para su utilización. Un método alternativo consiste en almacenar archivos de índices en disco al igual que archivos de datos. De esta forma, se pueden usar diversos campos como clave, mediante la creación en disco de diversos archivos índices de un mismo archivo. Esto también permitirá indexar el archivo de diferentes maneras: por ejemplo, los registros se podrían dar por orden de nombre (tanto de A a Z como de Z a A), por orden de fecha, etc.

¿Qué estructura debe tener un archivo índice? Cada registro de un archivo índice ha de contener dos campos (el dato de clave y el número de registro) para cada registro. Este registro se lee entero a la memoria, se utiliza y sólo se vuelve a escribir en disco si ha sido actualizado. Ésta es una aplicación ideal para un archivo secuencial en contraposición al archivo aleatorio, porque los datos se requieren en el orden en que están almacenados. Ésta es un área donde los archivos secuenciales y los archivos directos se complementan mutuamente.

Eliminar registros de archivos de acceso directo indexados es sólo cuestión de marcar los registros como eliminados y asegurarse de que ya no estén incluidos en el índice. La forma más económica de hacer esto consiste en grabar un indicador de "eliminado" en el registro (p. ej., un asterisco al principio del primer campo). La clave de ese registro se quita entonces del índice o, como alternativa, al número de registro se le puede poner algún valor especial que indique que el registro está eliminado (podría ser −1).

Cualquiera que sea el método elegido, lo importante es que en el archivo haya constancia de los registros eliminados. Cuando se añaden registros nuevos al archivo, éstos se pueden escribir sobre el espacio que ocupaba un registro eliminado. La entrada original del índice se debe reemplazar por una nueva y, en el momento apropiado, volver a clasificar el índice para que incluya al nuevo registro en la posición correcta dentro del archivo. De esta forma, el programa ofrecerá la posibilidad de recuperar registros eliminados accidentalmente, siempre y cuando en el ínterin no hayan sido sobreescritos.

Es una buena idea dar algún sistema para poner en orden el archivo índice. El sistema de indexación que hemos detallado es propenso a almacenar registros sin orden y con numerosos e innecesarios vacíos entre ellos. A medida que se trabaje con el archivo, la velocidad de acceso irá disminuyendo. La rutina de reorganización tendrá que clasificar los registros por el orden correspondiente y descartar del archivo todos los registros eliminados. La reorganización se puede llevar a cabo como una opción del usuario o de forma automática cada vez que el sistema concluye alguna operación de envergadura.

La indexación no es el único medio de acceder rápidamente a registros de un archivo grande. El *hashing* es un método alternativo muy apropiado para archivos muy grandes y, por consiguiente, se lo ve casi exclusivamente en sistemas de disco rígido o en máquinas con discos flexibles de gran capacidad. No obstante, muchos sistemas operativos y programas utilizan el hashing internamente para acelerar su operación; por tanto, se trata de una técnica que vale la pena conocer.

El hashing reemplaza al índice con una fórmula o algoritmo de hashing. Éste toma el valor del campo de clave y produce a partir de él un número de registro, al que se denomina *hash*. El registro correspondiente a la clave se almacena en esta posición del archivo. La fórmula será elaborada en función del tipo de datos del campo de clave. Si el campo de clave contiene una fecha (para posibilitar el almacenamiento cronológico de los registros), se podría emplear el número de mes, multiplicado por las dos últimas cifras del año, más el número del día. Un campo de nombre se podría "hashear" manipulando los códigos ASCII utilizados para las letras que componen el nombre, y así sucesivamente.

Supongamos que deseamos crear un archivo "hasheado" de registros de empleados utilizando como clave de clasificación los apellidos. El algoritmo de hashing que vamos a emplear es: tomar los códigos ASCII de las primeras cuatro letras y tratarlos como un número de ocho dígitos, elevar ese número al cuadrado, luego tomar los últimos cuatro dígitos del número resultante como el hash.

JONES, por consiguiente, se convierte en el registro 1161, mientras que JONQUIL se convierte en el 0161.

El hashing es muy distinto de un sistema indexado. Con el hashing sólo se puede tener un campo de clave (y un algoritmo de hashing) por archivo y éste se utiliza cuando se colocan por primera vez los registros en el archivo. Un archivo puede tener cualquier número de índices asociados, y éstos se pueden crear en cualquier momento, después o durante la creación del archivo.

El hashing es menos flexible que la indexación pero es mucho más rápido. Para hallar un registro determinado, el programa tan sólo toma la clave, le aplica el algoritmo de hash, y obtiene ese registro en particular. Por consiguiente, se ahorra el tiempo que lleva buscar en el índice, y también se gana el tiempo que cuesta crear el propio índice.

En el hashing se plantea un problema cuando dos registros generan el mismo código de hash y, por tanto, han de ocupar la misma posición en un archivo. Para evitar este problema, los algoritmos del hashing se diseñan cuidadosamente de modo que no haya dos claves (a menos que sean idénticas) que generen el mismo hash. Además, en el archivo los registros están espaciados de modo que dos hashes, que aparentemente están próximos entre sí, en realidad pueden tener entre ellos un espacio vacío de cinco registros.

Ahora podemos describir un sistema de hashing con mayor claridad de la siguiente manera. Cuando se almacena un registro, su clave se "hashea" para producir un número de registro. Si ese registro está ocupado, el sistema mira el siguiente registro en secuencia. Esto lo repite en los próximos cinco (o el número que sea) registros en busca de uno libre. Cuando se ha de recuperar un registro, se "hashea" su clave y entonces se busca secuencialmente en ese grupo de registros para hallar la clave buscada. Podría parecer que esto anula la ventaja de la velocidad, pero lo que el hashing hace efectivamente es reducir, quizá de tres mil a cinco o seis, el número de registros en los cuales buscar.

¿Qué sucede si todos los registros, los cinco o los que sea, para un hash determinado están ocupados? Hay varias formas de enfocar esto, siendo la más obvia de ellas emitir un mensaje de "archivo completo". Con mayor frecuencia, los registros que no se pueden colocar en su posición en el archivo se escriben en un archivo separado de *overflow* (exceso de capacidad) que posee su propio índice y se van incorporando al archivo principal cuando es posible. La mayoría de los sistemas se esfuerzan en evitar el *overflow*, manteniendo normalmente la ocupación de los archivos "hasheados" en un 80 % o menos de su capacidad. Esto evidencia otra limitación del acceso por hash a los archivos directos. Un archivo "hasheado" tiende a consumir más espacio que si se utilizara un índice.

El hashing también acelera la eliminación de registros no deseados. Simplemente se calcula el hash de la clave del registro, se efectúa una búsqueda rápida para localizarlo exactamente y se marca su posición como desocupada. Luego, la próxima vez que se agregue al archivo un registro con un hash idéntico, éste sobreescribirá al anterior.

En el capítulo final de esta serie analizaremos las instrucciones BASIC necesarias para crear archivos en cassette y acceder a ellos.

Unidos por índice

Encontrar "Davids"

Clave	N.º
Andreu	1
Benítez	−1
Bruch	5
Cruz	−1
Davids	7
Dávila	23
Ferrer	15
Gómez	28
Hernández	37
Jofré	25
Klaus	11
Martín	10

Archivo índice

La forma más común de acceder a un archivo directo es mediante un índice. Éste es el esquema de una RAM que refleja los valores para un campo de clave determinado con los registros correspondientes. Cuando se está accediendo a un registro, se lo puede buscar rápidamente en el índice y leerlo luego a la memoria. Los registros eliminados permanecen en el archivo y se los marca como borrados. Luego se sobreescribe en ellos a medida que se van agregando registros nuevos

Archivo principal

	Nombre	Tel. trab.	Tel. part.	Profesión
1	Andreu	242 07 91	727 09 42	Delineante
2	Pérez	636 24 18	221 39 40	Contable
3	Silva	631 08 36	286 81 70	Redactor
4		registro eliminado		
5	Bruch	729 82 13	236 21 90	Dentista
6	Paredes	836 66 22	298 43 10	Fontanero
7	Davids	743 72 16	450 69 26	Jardinero
8		registro eliminado		
9		registro eliminado		
10	Martín	730 63 21	429 75 92	Mecánico
11	Klaus	493 98 99	455 84 31	Abogado
12	Valverde	736 77 00	693 04 52	Peluquero

Haciendo un hash

Encontrar "Davids"

ALGORITMO DE HASHING

El algoritmo de hashing convierte las claves de modo que aludan a un determinado bloque de registros.
Los registros con hash idéntico se agrupan entre sí

Se deja espacio sin utilizar entre los bloques de registros, de modo que se puedan insertar registros nuevos en esas posiciones

Nombre	Tel. trab.	Tel. part.	Profesión
Davidson	629 04 91	430 05 92	Decorador
Daza	436 24 88	362 00 66	Sociólogo
Daroca	730 00 21	626 91 91	Tintorero
Damm	439 99 33	630 49 18	Escritor
Davids	743 72 16	450 69 26	Jardinero
Dávila	830 01 23	340 99 24	Programador
Esteruelas	731 66 66	458 00 21	Delineante
Eduardo	831 82 94	450 62 18	Proveedor

Los archivos "hasheados" ofrecen un acceso a gran velocidad a registros en grandes archivos aleatorios. No obstante, el sistema tiene sus limitaciones y exige una cuidadosa programación.
A la clave de registro se le hace corresponder una posición del archivo mediante la aplicación de un algoritmo de hashing preestablecido. Cada posible hash suele hacer referencia a un bloque de registros en el que se puede buscar secuencialmente para encontrar el registro deseado

Kevin Jones

Sonidos imaginativos

Pocos juegos hacen una utilización creativa del sonido. Veamos algunas ideas para desarrollar "audiojuegos" entretenidos

```
10 REM *** AUDIOJUEGO ***
20 PRINT "UN AUDIOJUEGO SENCILLO": PRINT:PRINT "DIRIJA SU NAVE HASTA EL RADIOFARO ANTES DE QUE SE LE AGOTE EL COMBUSTIBLE"
30 PRINT:PRINT "CUANTO MAS CERCA SE HALLE, MAS AGUDO SERA EL TONO DEL RADIOFARO"
40 PRINT:PRINT "LOS MANDOS SON:":PRINT "I (ARRIBA) M (ABAJO) J (IZQUIERDA) K (DERECHA)"
60 PRINT:PRINT "***** PULSE UNA TECLA PARA EMPEZAR *****"
70 A$=INKEY$(0):IF A$="" "THEN GOTO 70
80 P=INT(RND(1)*500)+1:Q=INT (RND(1)*500+1):F=3*(P+Q)/2:X=0:Y=0
90 XD=0:YD=0
100 PRINT "COMBUSTIBLE=";F, " DISTANCIA=";
110 IF (ABS(P-X)<2 AND ABS (Q-Y)<2) THEN PRINT "LO HAS CONSEGUIDO CUANDO AUN TE QUEDAN";F; "UNIDADES DE COMBUSTIBLE":END
120 A$=INKEY$(0):IF A$="" THEN GOTO 120
130 IF (A$="I" AND YD<3) THEN YD=YD+1
140 IF (A$="M" AND YD>-3) THEN YD=YD-1
150 IF (A$="J" AND XD>-3) THEN XD=XD-1
160 IF (A$="K" AND XD<3) THEN XD=XD+1
170 X=X+XD:Y=Y+YD
180 D=SQR ((P-X)*(P-X)+(Q Y) *(Q-Y))
190 PRINT D
200 SOUND 1, 8,(255-D),2
210 F=F-ABS(XD)-ABS(YD):IF F>0 THEN GOTO 90
230 PRINT TAB(10); "***** CRASH *****":PRINT TAB(11); "**** SE ACABO EL COMBUSTIBLE ****"
240 PRINT:PRINT "LA POSICION ES";D;" UNIDADES ESPACIALES DESDE LA BASE"
260 END
```

Los juegos recreativos se valen del sonido para apoyar sus bien cuidados gráficos. Aunque los efectos sonoros se utilizan para aumentar el realismo y hacer más emocionante la acción, raramente constituyen el centro de atención del juego propiamente dicho. No obstante, si podemos utilizar nuestro sentido de la vista como la base para muchos y muy variados juegos por ordenador, no existe ninguna razón por la cual no podamos generar juegos que se centren en nuestro sentido del oído.

El que veremos aquí es uno de estos "audiojuegos". Aunque no pretendemos que se lo considere como el juego más emocionante jamás desarrollado, ciertamente es un juego muy interesante. Lo que al principio parece ser una tarea bastante sencilla enseguida se convierte en un desafío fascinante, sencillamente porque uno tiene que utilizar una entrada sensorial distinta de la que se suele emplear cuando se controla alguna visualización en pantalla con brillantes colores.

En nuestro juego usted está situado en los controles de una nave espacial que se está quedando sin combustible. Su única esperanza de supervivencia está en acoplarse a una estación de abastecimiento de combustible cercana. Lamentablemente, debido a un defecto de funcionamiento de su ordenador, carece de toda información visual para orientarse en sus esfuerzos para el acoplamiento; el único método de navegación es dirigirse hacia la estación utilizando una señal audible. El tono de la señal del "radiofaro de navegación" se va volviendo más agudo a medida que uno se va acercando; de modo que todo es cuestión de escuchar atentamente e ir respondiendo con precisión mediante los controles.

Al igual que en una nave espacial auténtica, una vez que se dirige la nave en una dirección determinada, seguirá yendo en esa dirección hasta contrarrestar ese movimiento utilizando el mando opuesto. Si utiliza dos U para avanzar rápidamente hacia arriba, entonces necesitará pulsar dos D para detenerse de nuevo. Esto hace que el juego sea mucho más difícil de lo que parece.

Dave Cooper-Smith

Se requerirá bastante práctica para ser capaz de completar el juego utilizando sólo las indicaciones de sonido. Sin embargo, agregando unas pocas líneas se puede dar una indicación en la pantalla acerca de dónde se halla la nave en relación a la estación de abastecimiento. Ésta podría sencillamente indicar la distancia entre la nave y la estación (la variable D) o podría informar al jugador acerca de cuáles son los botones más eficaces (si SGN(p−x)=−1, entonces hay que ir hacia la izquierda, p. ej.) Estas útiles pistas harán que el juego resulte mucho más sencillo.

Después de probar nuestro programa, quizá quiera crear sus propios juegos. Hacer uso del sonido para localizar o evitar objetos es un campo con considerables posibilidades de programación. ¿Qué le parecerían juegos de sonar para submarinos o un campo de minas en el cual fuera necesario navegar utilizando un detector que emite una señal audible?

Los efectos de sonido más avanzados de máquinas como el BBC Micro y el Oric-1 evidentemente serán una ventaja en este caso. Los juegos sonoros pueden utilizar varias voces de forma simultánea, o dotarlas de ruidos ligeramente diferentes, cada uno con su propio significado en el contexto del juego.

Complementos al BASIC

Este programa se escribió en un BBC Micro en Mode 7 y, por lo tanto, es BASIC Microsoft casi estándar. Sus instrucciones PRINT presuponen una visualización en pantalla de 40 columnas y habrá de reformatearse para tamaños distintos. La instrución SOUND de la línea 200 es exclusiva del BASIC BBC. El valor del parámetro (255-D) es (*pitch*) el de la nota a tocar, mientras que los otros parámetros controlan el volumen, la duración y el canal. INKEY$(0) en la línea 120, y el empleo de RND en la línea 80, requerirán algo de atención en otras máquinas:

Spectrum
Insertar LET en todas las sentencias de asignación. Cambiar INKEY$ (0) por INKEY$. Cambiar RND(1) por RND. Cambiar la línea 200 por:

```
200 BEEP 0.4,(255-D)
```

e insertar:

```
15 RANDOMIZE
195 IF D > 254 THEN LET D=D-254
```

Commodore 64 y Vic-20
Cambiar INKEY$(0) por GET A$. Remitirse al manual del usuario para las instrucciones de sonido del Commodore. Insertar:

```
75 X=RND (-TI)
```

Dragon
Cambiar INKEY$(0) por INKEY$. Cambiar RND(1) por RND(0). Cambiar la línea 200 por:

```
200 SOUND (255-D),10
```

e insertar:

```
195 IF D > 254 THEN D=D-254
```

Oric Atmos
Cambiar INKEY$(0) por KEY$. Cambiar la línea 200 por:

```
200 SOUND 1, (255-D), 9:WAIT 40:PLAY0,0,0,0
```

e insertar:

```
195 IF D>254 THEN D=D-254
```

Deporte acuático

"Scuba dive" constituye un refrescante cambio respecto a los juegos de disparos del tipo "invasores" y guerras intergalácticas

Existen a la venta tres versiones del *Scuba dive* (Submarinista): una para el Spectrum, otra para el Oric-1, y una tercera para el Commodore 64. La versión para el Oric-1 está siendo adaptada para que también funcione en el Atmos.

El jugador asume el papel de un submarinista que está recogiendo un tesoro del lecho marino, tarea en la cual arriesga su vida. El principal objetivo de nuestro intrépido héroe consiste en recoger perlas que acumulan puntos, que se extraen de ostras y de almejas gigantes. En una etapa más avanzada del juego se debe recoger un tesoro del interior de unos cofres que se hallan en las profundidades del cavernoso mundo submarino.

Sin embargo, existen muchas y buenas razones para ir avanzando con gran cautela. El agua está densamente poblada de criaturas que constituyen una mortal amenaza para el submarinista: medusas, pulpos, calamares, anguilas eléctricas y, en la versión para el Spectrum, ¡tiburones! Si le toca una de estas criaturas, se pierde una vida, aunque ninguna de ellas ataca expresamente; simplemente deben evitarse sea como sea. Otro peligro se descubre cuando se comienza a extraer perlas de las almejas gigantes: éstas tienen capacidad de cerrarse sobre uno, dejándolo atrapado.

Las estrechas entradas a la caverna principal y a las profundidades de nivel inferior están custodiadas por pulpos gigantes. Superar a los pulpos puede ser complicado, ya que sus tentáculos siempre están revoloteando por ahí. Pero, de vez en cuando, uno se las puede arreglar para deslizarse por entre ellos. En la versión para el Commodore no hay pulpos. En lugar de éstos hay una trampilla que impide el paso. Ésta se abre y cierra continuamente y hay que pasar por ella sin que un golpe fortuito lo deje inconsciente.

El programa concede tres vidas por juego y tiene cuatro niveles de dificultad. Se pierde una vida al tocar a alguno de los poco amistosos seres acuáticos o cuando se acaba el oxígeno.

Cada versión del juego empieza con una panorámica de la superficie del mar y una gran parte de las profundidades. El barco desde el que uno se arroja está meciéndose en la superficie. Uno debe tener cuidado desde el mismo momento de tirarse al agua, ya que es posible quedar atrapado debajo de la embarcación. En la versión para el Spectrum es una ventaja tener un buen sentido de la orientación, porque el barco puede girar cuando uno se encuentra sumergido. En la versión para el Oric esto no representa un gran problema, ya que la embarcación siempre se mueve de izquierda a derecha sin que en ningún momento se salga de la pantalla y, en consecuencia, cuando uno emerge a la superficie siempre está a la vista.

La calidad de los gráficos en la versión para el Spectrum (que es con mucho la mejor de las tres) es soberbia. Se ha hecho buen uso del color y el diseño de las criaturas y de la caverna es de un gran realismo. El control sobre el submarinista se consigue utilizando las teclas X y Z para girar en el sentido de las agujas del reloj y en dirección contraria, respectivamente, y las teclas Space y Shift mueven al submarinista hacia adelante. Los usuarios del Spectrum también pueden emplear palancas de mando, aunque el programa no funciona con la interface para palanca de mando Kempston. Los usuarios del Commodore también disponen de la opción de la palanca de mando, pero quienes juegan con el Oric no disponen de este recurso y deben hacerlo con el teclado.

La versión para este último ordenador es, en varios sentidos, mucho menos emocionante que su equivalente para el Spectrum. El movimiento del submarinista y de los seres marinos es muy torpe, y los gráficos (en especial las paredes de las cavernas y los cofres) están mucho menos detallados.

Por su parte, la versión para el Commodore tampoco alcanza el nivel de calidad de imagen que caracteriza a la del Spectrum, pero, sin lugar a dudas, es mucho más satisfactoria que la del Oric.

Comparando calidad
Estas fotografías ilustran la diferencia de calidad entre las distintas versiones del juego *Scuba dive*

Sinclair Spectrum

Oric-1

Commodore 64

Visto en perspectiva

Esta imagen se compuso a partir de varios volcados de pantalla de la versión del *Scuba dive* para el Spectrum. Las imágenes se han unido para mostrar la visión de las cavernas submarinas que tiene el submarinista

Ian McKinnell

Restar como sumar

Ha llegado el momento de estudiar las operaciones aritméticas de suma y resta en código máquina

Precisamente son las operaciones aritméticas las que nos van a mostrar las filosofías de diseño que diferencian al Z80 del 6502. Los numerosos registros del Z80, con su sofisticado juego de instrucciones, dan idea del procesador mismo: elegante, complejo y potente. En cambio, la arquitectura y el conjunto de instrucciones del 6502, mucho más sencillas, indican un procesador más modesto, robusto y práctico sin duda, pero carente, en apariencia, de la categoría de un Z80. A pesar de todo, la riqueza de tipos de direccionamiento del 6502 y el empleo que hace de la página cero como un registro índice adicional le otorgan su carácter ágil y versátil, y sugieren que su actual hegemonía dentro del mundo de los microordenadores personales y de oficina va a durar todavía tiempo.

La ventaja que tienen los registros del Z80 es que son flexibles. Simultáneamente pueden ser tratados como registros de un solo byte o de dos, permitiendo un gran ámbito de direccionamiento. Por el contrario, los registros de dos bytes están ausentes en el 6502, aunque gracias a sus tipos de direccionamiento puede servirse de la página cero como si fuera una tabla de registros de uno y de dos bytes.

Aritmética básica

Ya hemos visto que los registros de la CPU permiten una gran variedad en los posibles accesos a memoria, pero ahora veremos que el manejo de la memoria comporta algo más que cargas, descargas y comparaciones de su contenido. Para un sistema, la posibilidad de realizar las cuatro operaciones básicas de la aritmética es fundamental, y es curioso que tanto el Z80 como el 6502 tan sólo sepan sumar y restar. La multiplicación y la división, e incluso la suma y resta de números superiores a $FF, deben ser previamente programadas. Esto constituye una limitación evidente de ambas CPU, pero su valor pedagógico es incalculable, ya que el programador ha de inventarse los algoritmos para multiplicar y dividir. Los procesadores de 16 bits que han aparecido después del Z80 y del 6502 pueden realizar también estas dos operaciones, debido a la mayor potencia y rapidez de sus CPU.

Para el tratamiento aritmético de un solo byte a la vez ya hemos tenido ocasión de utilizar la instrucción ADC (*ADd with Carry:* sumar con un bit de arrastre) y varias otras del tipo INC (INCrementar) en ambas CPU. En el cuadro adjunto tenemos un ejemplo de cómo sumábamos los contenidos de dos posiciones de memoria cada una de ellas con dos bytes numéricos. El 6502 recurre al método de byte a byte que el Z80 conoce, pero a éste le resulta más fácil emplear otro método basado en sus pares de registros, de los que no hay equivalentes en el 6502. Nótense las estrategias que emplean ambos procesadores para las diversas posibilidades de arrastre en la suma: se comienza con instrucciones como CLC (6502) o bien AND A (Z80) que borran, antes de hacer la suma, el flag de arrastre poniéndolo a cero, y se acaba modificando el tercer byte de SUM, para el caso de que el resultado desborde los dos bytes, caso que jamás se ha de descuidar.

6502			Z80		
ADDR1	DW	$7E60	ADDR1	DW	$7E60
ADDR2	DW	$4A51	ADDR2	DW	$4A51
SUM	DS	$03	SUM	DS	$03
BEGIN	CLC		BEGIN	LD	A,$00
	LDA	ADDR1		AND	A
	ADC	ADDR2		LD	HL,(ADDR1)
	STA	SUM		LD	DE,(ADDR2)
	LDA	ADDR1+1		ADD	HL,DE
	ADC	ADDR2+1		LD	(SUM),HL
	STA	SUM+1		ADC	A,$00
	LDA	$00		LD	(SUM+2),A
	ADC	$00		RET	
	STA	SUM+2			
	RTS				

De la misma manera que tratamos la suma se puede hacer con la resta, ya que ambos procesadores poseen la instrucción SBC (*SuBtract with Carry*) aunque el Z80 puede hasta restar dos bytes a la vez. Pero como la resta nos introduce en el tema de los números negativos, hemos de hacer una pequeña incursión en la representación binaria del signo algebraico.

Para empezar, de los números negativos está dicho todo con entender bien esta expresión:

Si A + B = 0 entonces A = −B

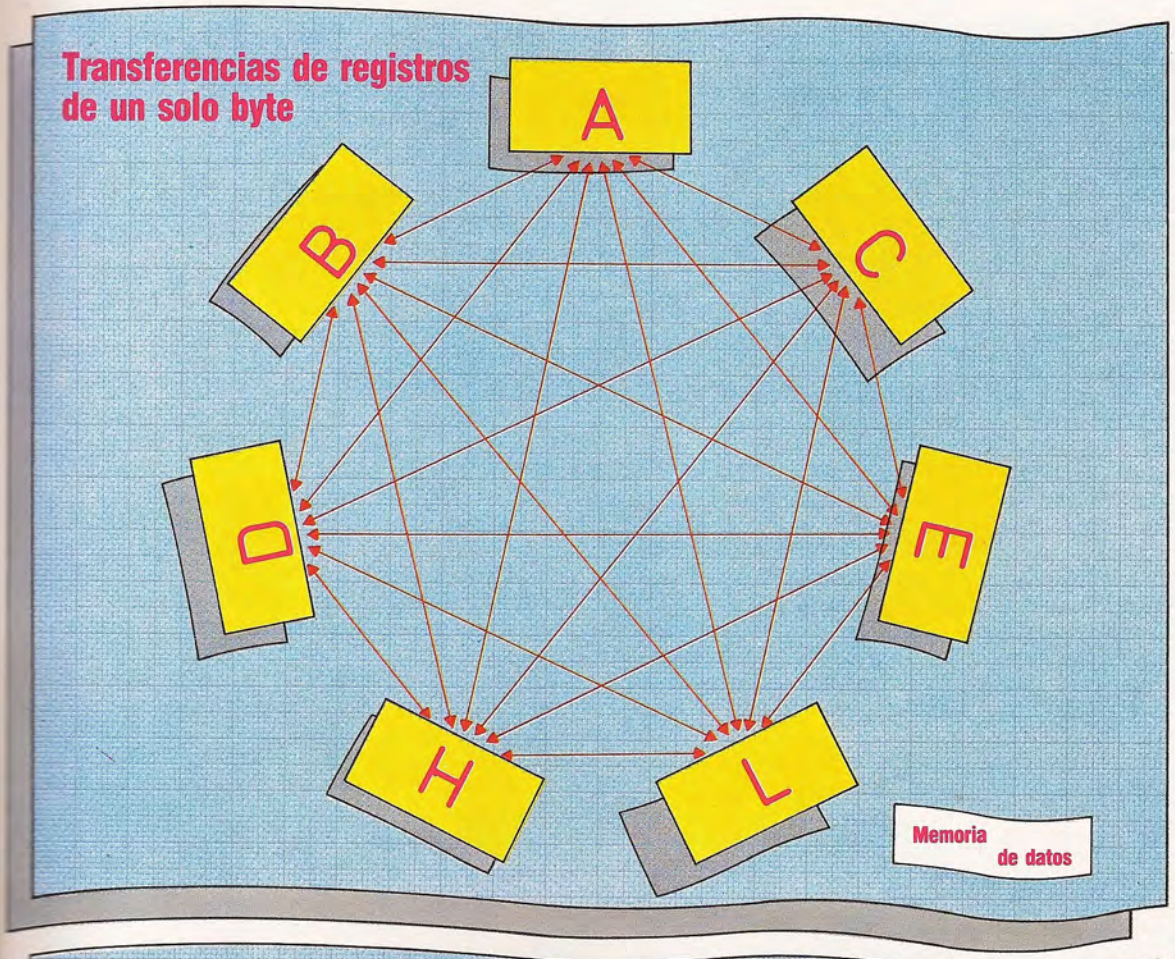
Diremos en este caso que B es el negativo o complemento de A, ya que sumados dan cero. Si tenemos en cuenta los números que caben en un solo byte, no es sorprendente considerar el número $FC como el complemento de $04, ya que

$04 + $FC = $00

Note que el resultado en realidad sería $100 pero ya advertimos que sólo se dispone de un registro con capacidad para un solo byte. Así pues, ambos números son complementarios, o el uno negativo del otro. Este pequeño descubrimiento nos conduce a una conclusión algo chocante pero utilísima: restar es sumar dos números, sustituyendo uno de ellos por su negativo. Es decir:

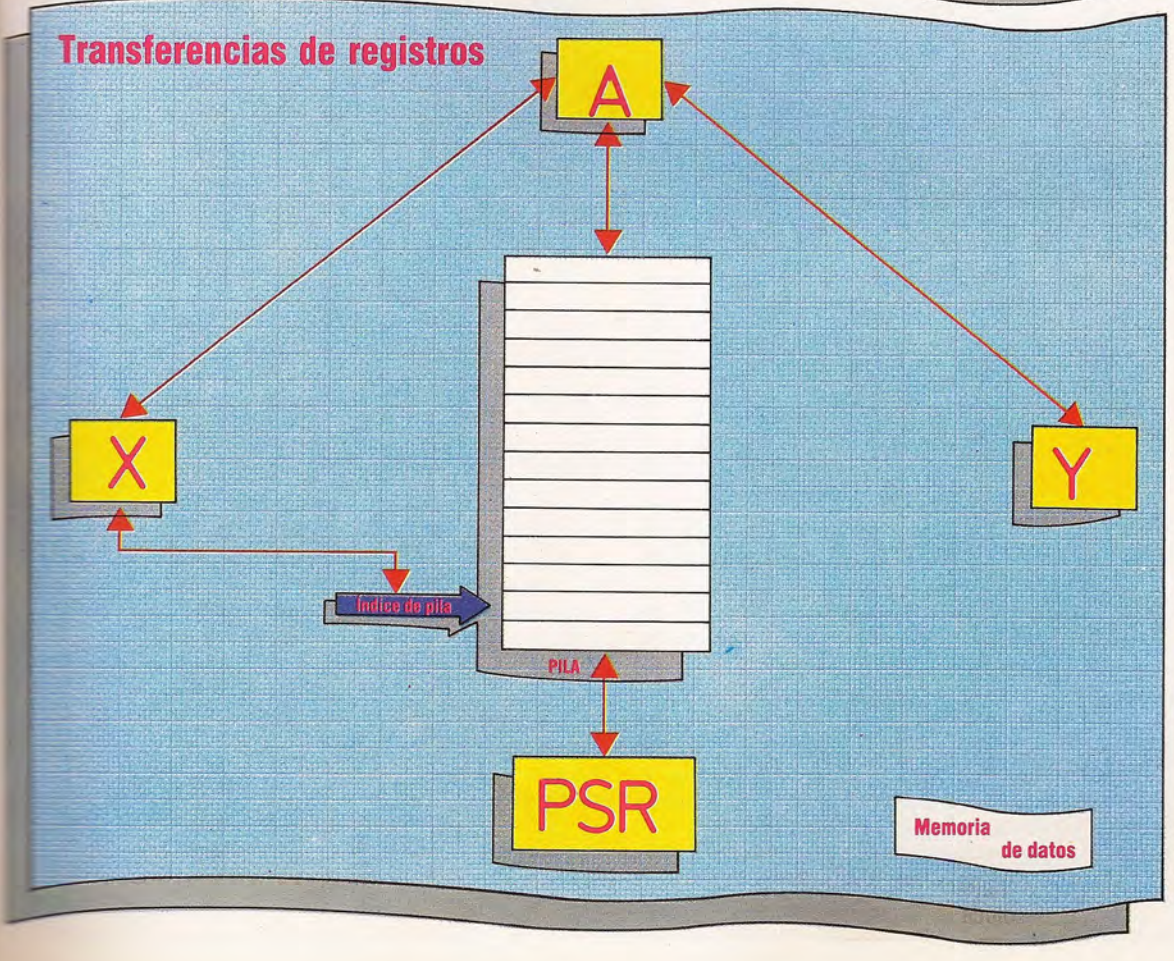
La resta A − B equivale a la suma A + (− B)

Si seguimos teniendo en cuenta el registro de un solo byte, la siguiente resta $09 − $04 se convierte en la suma $09 + $FC, pues ya vimos que el complemento de $04 es $FC. El resultado es $05, que es lo que cabe en un byte, aunque en realidad debería ser, para la suma, $105.



Doble personalidad
Los registros de datos del Z80 pueden comportarse como registros de un byte y comunicar con cualquier otro registro también de un solo byte. Cualquiera de ellos puede comunicar con la memoria en el modo de direccionamiento directo, inmediato, indirecto, absoluto e indexado. Si son tratados en pareja, como BC, DE, HL, pueden transferir datos de 16 bits de la memoria a la pila y viceversa, y funcionar como acumuladores de 16 bits en sumas y restas. Esta mezcla de flexibilidad y riqueza de recursos es lo que explica el inmenso éxito del Z80

Kevin Jones



Claro y sencillo
La comunicación interna del 6502 es estrictamente lineal y limitada a transferencias de datos de 8 bits. Solamente el acumulador puede comunicarse directamente con el X y el Y. Tan sólo X puede comunicarse con el índice de pila; así como los únicos que tienen acceso a la pila son el indicador de estado y el acumulador. Las transferencias de la memoria son posibles en los modos absoluto, directo, indirecto, indexado, inmediato y página cero. El original empleo que hace el 6502 de la página cero compensa el reducido tamaño de sus registros, pues la página cero puede ser tratada como registros de la CPU de dos bytes. ¡Nada menos que 128 registros disponibles!

Kevin Jones

Aquí es esencial la pregunta ¿quién es negativo de quién? Ya que está claro que si $FC es el negativo de $04, entonces $04 es el negativo de $FC. Es norma convenida, por una razón que vamos a explicar, considerar POSITIVOS los números (¡siempre de un solo byte!) que van desde el $00 al $7F, mientras serán NEGATIVOS los comprendidos entre $80 y $FF. En decimal, de los 256 números posibles en un byte, la mitad inferior de ellos son de signo positivo, y la otra mitad, negativo. Y la razón de esto radica en que los números inferiores a 128, representados en binario sobre un byte, tienen el bit superior (el bit 7) siempre a cero (p. ej., en binario 127 es 0111 1111); mientras que el bit 7 de los binarios comprendidos entre 128 y 255, ambos inclusive, está siempre a uno (p. ej., 255 es 1111 1111; 128 es 1000 0000). Era, pues, lógico que se acordara que el bit 7 fuera el *bit del signo*, y que el flag de arrastre se activara o desactivara según indicara el bit 7 del byte de resultado en una operación aritmética o lógica. Con esto se abre la posibilidad de que el resultado sea negativo.

Otro concepto por explicar es el del *complemento a 1*. Hasta aquí hemos utilizado, sin darnos cuenta, la noción de *complemento a 2*: el complemento a 2 de $04 es $FC, y al revés, pues sumados dan $00 en un solo byte. Pues bien, el complemento a 1 de un número expresado en binario como

0000 1101 (hexa 0D)

se obtiene fácilmente poniendo cada uno de sus bits en su estado opuesto, cambiando los ceros por unos, y los unos por ceros:

1111 0010 (hexa F2)

¿Qué utilidad le reporta saber esto? La siguiente: que basta con añadir un 1 al complemento a 1 para convertirlo en complemento a 2. En nuestro ejemplo, el complemento a 2 del número binario de partida será 1111 0011 (hexa F2 + 1). Note finalmente que en hexadecimal un número y su complemento a 1 suman siempre $FF (en el ejemplo, 0D + F2 = FF), y que este resultado se convierte en $100 con sólo sumarle 1. Bien entendida esta lección, usted no tendrá dificultad alguna con las explicaciones del próximo capítulo. Capítulo en el cual anticipamos que trataremos, además, de los algoritmos de multiplicar y dividir.

Respuestas a los ejercicios de p. 739

1) Éste es el programa que permite invertir el orden del string contenido en LABL1:

6502

```
;
ORIGIN   ORG    $7000
LAST1    EQU    $0D
LABL1    DB     'THIS IS A MESSAGE'
TERMN8   DB     LAST1
;
BEGIN    LDX    #$FF
         LDA    #LAST1
         PHA
LOOP0    INX
         LDA    LABL1,X
         PHA
         CMP    #LAST1
ENDLP0   BNE    LOOP0
CLRSTK   PLA
;
BEGIN1   LDX    #$FF
LOOP1    INX
         PLA
         STA    LABL1,X
         CMP    #LAST1
ENDLP1   BNE    LOOP1
         RTS
```

En la versión para el 6502, las instrucciones que van desde LOOP0 hasta ENDLOOP0 emplean el direccionamiento indexado con X en un bucle que sirve para cargar uno a uno los caracteres de LABL1 y "empujarlos" (*push*) en la pila, colocando en la pila primero el valor ASCII del carácter (') para indicar el fondo de la pila. Este mismo valor será también el último que se "empuje" en la pila, indicando ahí el final del string. Con él concluimos el bucle, y una vez conseguido esto lo borramos de la pila en CLRSTK (*clear stack*: limpiar pila).

En el caso del Z80, se usa IX en modo de direccionamiento indirecto para cargar desde LABL en adelante, y coloca en la pila no sólo el acumulador sino hasta el registro flag. Esto quiere decir que en la pila aparecerán intercalados los caracteres del string con los sucesivos valores del indicador de estado.

Z80

```
         ORG    $C000
LAST1    EQU    $0D
LABL1    DB     'THIS IS A MESSAGE'
TERMN8   DB     LAST1
;
BEGIN    LD     IX,LABL1-1
         LD     A,LAST1
         PUSH   AF
LOOP0    INC    IX
         LD     A,(IX+0)
         PUSH   AF
         CP     LAST1
ENDLP0   JR     NZ,LOOP0
CLRSTK   POP    AF
;
BEGIN1   LD     IX,LABL1-1
LOOP1    INC    IX
         POP    AF
         LD     (IX+0) ,A
         CP     LAST1
ENDLP1   JR     NZ, LOOP1
         RET
```

En ambos casos, la codificación entre BEGIN1 y ENDLP1 es similar a la del bucle anterior, pero extrayendo (POP,PLA) de la pila los caracteres en orden inverso, los últimos al principio, y almacenándolos en LABL1. El bucle concluye al encontrar el carácter que dejamos como señal al fondo de la pila.

Hay que remarcar la importancia de administrar bien tanto los llenados de la pila como sus vaciados (PHA o PUSH, PLA o POP), y también es importante saber manejar las condiciones de los extremos (qué se hace para iniciar el bucle, para acabar y para darle continuidad).

La instrucción LD IX,LABL1−1 que encontra-

mos dos veces en la rutina para el Z80 es un ejemplo de la utilidad de un programa ensamblador. Aquí decodifica la expresión LABL1−1 como diciendo "la dirección del byte que antecede inmediatamente al que tiene por dirección LABL1", y ensambla esa dirección en el código generado. La mayoría de los ensambladores admite cierta medida de evaluación de expresiones, permitiendo también normalmente la suma o resta de alguna constante a los operandos.

2) Este programa invierte el orden de los caracteres en cada palabra del string contenido en LABL1, conservando el orden de estas mismas palabras:

6502

```
;
ORIGIN   ORG   $7000
LAST1    EQU   $0D
SPACE    EQU   $20
LABL1    DB    'THIS IS'
TERMNB   DB    LAST1
;
BEGIN    LDX   #$FF
LOOP0    JSR   RVSWRD
         CMP   #LAST1
ENDLP0   BNE   LOOP0
         RTS
;
;****REVERSE A WORD S/R****
LASTCH   DB    $00
LASTX    DB    $00
RVSWRD   TXA
         TAY
         INY
RVSLP0   INX
         LDA   LABL1,X
         PHA
         CMP   #SPACE
         BEQ   CLRSTK
         CMP   #LAST1
ENDRV0   BNE   RVSLP0
CLRSTK   PLA
         STA   LASTCH
         STX   LASTX
RVSLP1   PLA
         STA   LABL1,Y
         INY
         CPY   LASTX
ENDLP1   BNE   RVSLP1
         LDA   LASTCH
         RTS
```

Varios son los detalles que nos interesan aquí (el empleo de JSR y CALL, p. ej.). La subrutina RVSWRD (invertir palabra) se parece a la que encontramos en el ejercicio 1, pero invierte sólo los caracteres dentro de cada palabra. En ambas versiones, la del 6502 y la del Z80, se usa el registro índice (el X y el IX respectivamente) para pasar a la subrutina la dirección de inicio de la palabra, y el acumulador es utilizado para devolver al programa principal el valor del carácter de fin de tarea (que puede ser tanto un espacio como la comilla). Se trata de una técnica de paso de valores muy común en lenguaje assembly, pero se ha de emplear con cuidado, sobre todo si se tiene la costumbre de llevar a la pila todos los registros de la CPU al comienzo de cada subrutina (como se dijo en la p. 738).

Otro punto relevante es el empleo del registro Y en la versión del 6502, utilizado primero para guardar la dirección de inicio de la palabra (mientras el X servía de índice en el bucle que llena la pila), y después sirviendo a su vez de índice en el bucle de vaciado de ésta (mientras el X guarda la dirección del final de la palabra). Decimos "dirección" con imprecisión, pues tanto X como Y son registros de un solo byte, incapaces de contener una dirección completa (que se compone de dos bytes, como sabemos). De todos modos, aquí lo que contienen es el byte de desplazamiento respecto a la dirección de LABL1. El Z80 no tiene problemas: tanto IX como IY pueden contener una dirección completa.

Z80

```
         ORG   $C000
LAST1    EQU   $0D
SPACE    EQU   $20
LABL1    DB    'THIS IS A MESSAGE'
TERMN8   DB    LAST1
;
BEGIN    LD    DE,LABL1−1
LOOP0    CALL  RVSWRD
         CP    LAST1
ENDLP0   JR    NZ,LOOP0
         RET
;
;***REVERSE A WORD S/R***
LASTCH   DB    $00
RVSWRD   PUSH  DE
         POP   HL
         INC   HL
RVSLP0   INC   DE
         LD    A,(DE)
         PUSH  AF
         CP    SPACE
         JR    Z,CLRSTK
         CP    LAST1
ENDRV0   JR    NZ,RVSLP0
CLRSTK   POP   AF
         LD    (LASTCH),A
;
RVSLP1   POP   AF
         LD    (HL),A
         INC   HL
         LD    A,L
         CP    E
         JR    NZ,RVSLP1
         LD    A,H
         CP    D
ENDRV1   JR    NZ,RVSLP1
         LD    A,(LASTCH)
         RET
```

No se utilizan IX ni IY en la versión del Z80, pero sí el par de registros HL y DE. Al igual que X e Y en el 6502, sirven para guardar el comienzo y final de las direcciones de la palabra, pero en lugar de servir de índice a una dirección de base, funcionan como direcciones indirectas: la instrucción LD A,(DE) quiere decir "carga el acumulador con el contenido del byte cuya dirección está indicada en DE". Todos los pares de registros del Z80 se pueden usar así. Una limitación es la falta de una instrucción que compare dos bytes; para comparar DE y HL hay que comparar E con L y después D con H. Igualmente, en la versión 6502, se comparan X e Y de modo indirecto, por medio de una posición de memoria.

El secreto del éxito

Puede considerarse excepcional el hecho de que en la lista de los diez programas más vendidos no aparezca alguno de los creados por la empresa británica Quicksilva

La fabricación del microordenador Sinclair ZX80 fue el impulso inicial para Quicksilva. El éxito de la máquina alentó a un ingeniero de pruebas, Nick Lambert, trabajador por cuenta propia, a diseñar una placa accesoria de tres Kbytes para complementar la escasa memoria de un Kbyte del ZX80. Lambert obtuvo un gran éxito vendiendo esta placa por correo. Cuando al año siguiente Sinclair lanzó el ZX81, Lambert creó Quicksilva, junto con John Hollis y Mark Eyles, para producir una serie de accesorios para la nueva máquina. En 1981 también apareció *Defender*, un juego escrito por Lambert, que fue la primera incursión de Quicksilva en el mercado del software.

El éxito de su juego *Defender* alentó a Quicksilva a producir más software de tipo recreativo, y finalmente se abandonó la vertiente de hardware de la empresa. En abril de 1982 se lanzó Quicksilva como una sociedad anónima.

El segundo gran éxito de software de la empresa fue un juego de aventuras llamado *Timegate*. En la Navidad de 1982 ya había 10 juegos Quicksilva en el mercado, y en enero del año siguiente W.H. Smith encargó 10 000 ejemplares de *Timegate*. Con sus productos (y su nombre) penetrando en el mercado de software comercial, que estaba en rápida expansión, la demanda de productos de Quicksilva subió como la espuma. Habiéndose lanzado con un descubierto de 200 libras, el movimiento total de la empresa durante su primer año de operación fue de 70 000 libras. Los productos Quicksilva ahora se suministran a través de grandes cadenas de tiendas y 150 detallistas independientes. La empresa cree que sus juegos cubren más del 70 % del mercado mundial en la actualidad.

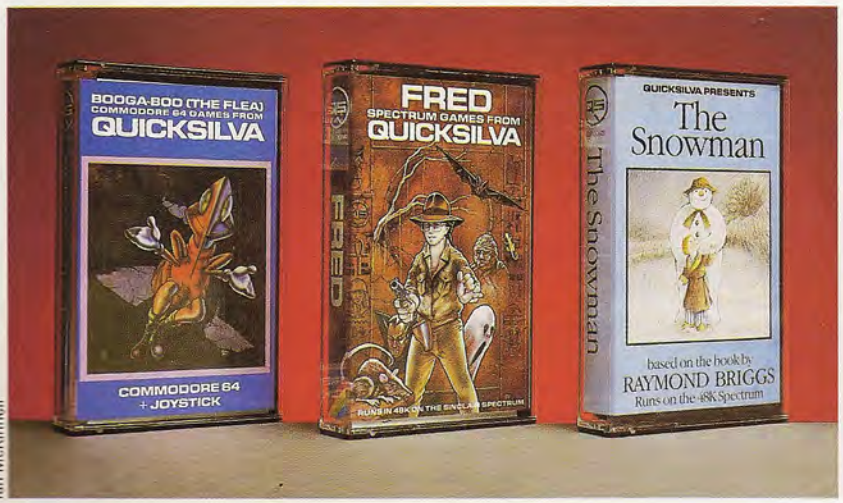
El repentino crecimiento de la demanda significó que la empresa tuviera que ampliarse rápidamente, superando el funcionamiento basado en tres hombres. En la actualidad Quicksilva publica anuncios en la prensa especializada pidiendo autores de software, y un escritor de juegos puede percibir hasta el 15 % por cada una de las cassettes de sus programas que se vendan, en concepto de royalties. Hoy la empresa continúa diversificándose y ya ha dejado de crear exclusivamente para las máquinas Sinclair: el catálogo actual de Quicksilva figura en la actualidad entre los más variados del mercado, por cuanto incluye juegos diseñados para el BBC Micro, el Dragon, el Commodore 64 y el Vic-20.

Quicksilva está considerada en estos momentos como una de las principales empresas editoriales de software de Gran Bretaña. Su prestigio se acrecentó aún más debido al enorme éxito de *Ant attack* (véase p. 486), juego que realizaba gráficos sumamente sofisticados.

Rod Cousens, que asumió el cargo de director gerente de la empresa cuando Lambert decidió concentrarse en proyectos más creativos, fue elegido vicepresidente del Gremio de Casas de Software y "Persona del año" por la Computer Trade Association británica el año 1983.

Quicksilva continúa buscando caminos inesperados hacia los cuales diversificarse. En particular, la empresa ha lanzado recientemente un juego "no violento" denominado *The snowman* (El muñeco de nieve), que está basado en un cuento para niños de Raymond Briggs. Se lo considera como un antídoto, que ha sido bien recibido por el público, contra la enorme plétora de juegos "de disparos" del tipo "invasores del espacio".

En otros sentidos, los planes de la empresa son un poco más predecibles: espera producir enseguida un gran impacto en el mercado de software norteamericano.



A la venta
Éstos son algunos de los últimos juegos que Quicksilva ha sacado a la venta. En la actualidad la empresa produce juegos para una amplia gama de ordenadores

Ian McKinnell

Mapa cortesía de Sinclair

Rod Cousens
Actual director gerente de Quicksilva

Mark Eyles
Director de publicidad y uno de los fundadores de la empresa

Cortesía de Quicksilva

A la orden de sus dedos

Las bases de datos constituyen un sotfware fundamental. Vamos a examinarlas en sus versiones para microordenadores

Una *base de datos* es un conjunto de datos relativos a un tema. Por ejemplo, una base de datos podría mantener uno de los siguientes conjuntos de información: los nombres y las direcciones de los socios de un club; los gastos que representan sus reuniones; las fechas y lugares de tales reuniones, o el pago de las cuotas sociales efectuadas por los socios del club. Sin embargo, un sistema de base de datos podría incluir todos estos datos en un gran archivo.

Un *archivo* es un conjunto de registros, cada uno de los cuales se compone de un cierto número de campos. En algunas aplicaciones, como en los paquetes de contabilidad, la estructura del archivo se determina por el software, pero en un sistema de base de datos es más común que la estructura del archivo se elija en función de la tarea que lo va a explotar. Esto implica determinar el tamaño de cada campo y definir su clase de contenido. En un archivo de nombres y direcciones, por ejemplo, los detalles relativos a cada persona ocupan un registro individual: su nombre se almacena en un *campo de tipo carácter* o *alfanumérico* y cada línea de su dirección en campos separados. Estos campos suelen tener treinta caracteres de longitud.

La alternativa a un campo alfanumérico es un *campo numérico*, que permite al programa efectuar cálculos con los datos almacenados en él. En nuestro archivo de socios del club, el programa podría calcular cuántos miembros han pagado sus cuotas, a cuánto asciende el total de ingresos en lo que va de año, o cuánto queda aún por cobrar. No obstante, no todos los datos numéricos han de ser almacenados en campos numéricos. Los números de teléfono constituyen un buen ejemplo de números con los que no se efectúa ningún cálculo, y éstos se almacenan normalmente en campos alfanuméricos.

Para saber si en su ordenador se puede utilizar cierto archivo, es necesario calcular las dimensiones máximas del mismo (cuántos registros deberá tener), calcular la cantidad de memoria necesaria para cada registro y luego determinar si se dispone de suficiente memoria para almacenar el archivo. Un registro con un nombre y tres campos de dirección de 30 caracteres cada uno y un número de teléfono que ocupe 10 caracteres ocupa un total de 130 bytes. Si se tiene una unidad de disco en el sistema con 200 Kbytes de capacidad, entonces podrá guardar 1 500 registros en cada disco. En un sistema sin unidades de disco con 48 Kbytes de memoria sólo habrá espacio para 300 registros (concediendo alre-

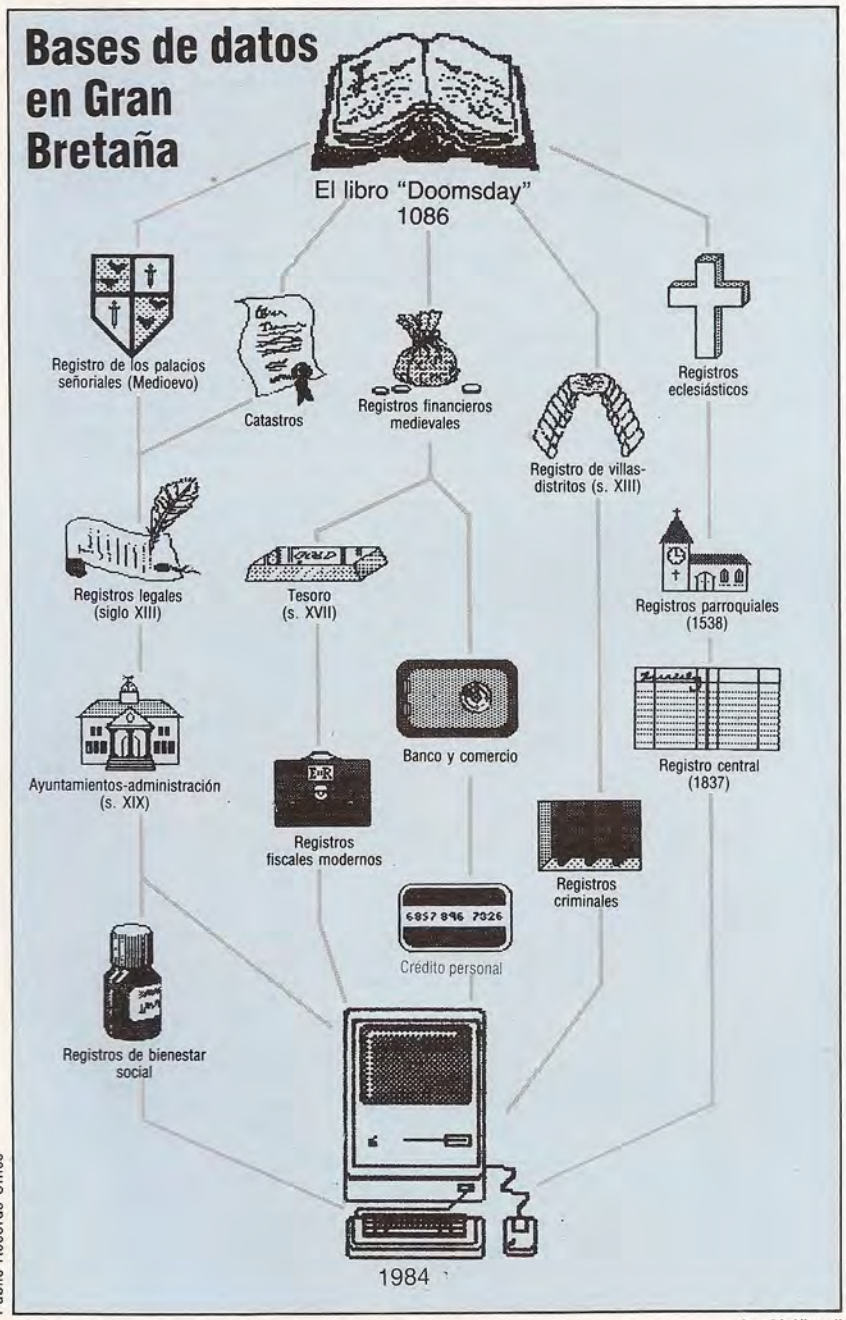
Archivos caseros
La mayoría de las personas conservan información acerca de sí mismas archivadas sin ningún método en trozos desperdigados de papel. Una base de datos por microordenador personal sería útil a modo de almacenamiento central de las pólizas de seguros, detalles sobre las propiedades, números de cuentas bancarias, etc. Además, tendría un valor incalculable para cualquier persona que coleccione sellos, monedas, discos o cualquier otro objeto que se preste a la clasificación sistemática

Mike Brownlow

Sólo basta conectar
Las bases de datos proliferan en las sociedades técnicas que dependen de la acumulación de hechos, y la accesibilidad de la información crece a medida que la tecnología avanza. En el pasado, las bases de datos estaban aisladas unas de otras en virtud de la distancia y el tiempo, pero ahora que tantas bases de datos privadas y de la administración están basadas en ordenador, la posibilidad de que éstas se comuniquen entre sí supone una amenaza potencial para la intimidad y el derecho a la vida privada de las personas

dedor de 10 Kbytes para el programa y el sistema operativo). En la práctica, si se desea clasificar el archivo o hacer otro tipo de tratamiento de esta clase, será necesario algún espacio de trabajo libre, y es conveniente limitar el tamaño del archivo a la mitad de la zona disponible. Los dos sistemas de almacenamiento difieren de forma tan radical debido a que los archivos en cinta se deben leer en la memoria y procesar como un todo, mientras que la velocidad de los accesos a archivos en disco permite que los archivos residan en disco y se procesen en la memoria.

Un sistema de gestión de datos permitiría tomar información de un archivo (el total de gastos del año, p. ej.) y relacionarla o compararla con información de otro archivo, como por ejemplo, el total ingresado en concepto de cuotas sociales. Basándose en esa información, se podría, por ejemplo, tomar una decisión acerca de si iniciar o no una campaña de captación de socios, reducir la cantidad de reuniones o invertir los beneficios en un nuevo ordenador para el club. Quizá podría ser conveniente enviar una carta estándar a los socios del club informándoles de la situación. La base de datos podría proporcionar los nombres y las direcciones, permitiendo su impresión directamente en sobres o etiquetas autoadhesivas. O bien podría enlazarlas con un paquete para tratamiento de textos para escribir cartas e informes. A pesar de que son muchas las cosas que puede hacer la base de datos ideal, no resulta fácil encontrar una que ofrezca todas las facilidades juntas, especialmente si se posee un ordenador personal con un espacio de memoria limitado o el almacenamiento es en cinta y no en disco. Muchos de los paquetes de bases de datos menos sofisticados sólo tratan un archivo cada vez, de modo que aunque se pudieran retener la totalidad de los distintos grupos de información que hemos mencionado antes, no podrían ser relacionados ni comparados en un auténtico estilo de base de datos.



Public Records Office

Ian McKinnell

En todo proceso de datos existe el riesgo de depositar toda una valiosa información en el ordenador y que, a causa de algún accidente, se destruya. Cuando se trata de información importante es esencial contar con copias de seguridad.

Un paquete de base de datos sencillo tratará un único archivo de información. El sistema hará que el trabajo rutinario de introducir los datos en el archivo sea tan fácil como rellenar un formulario, y permitirá el acceso a la información tanto a través de la pantalla como de una impresora. El usuario debería ser capaz de seleccionar registros individuales a partir de criterios de búsqueda y clasificación, por ejemplo "todos los socios que todavía no han pagado este año y que tienen un código postal determinado". Un paquete de esta clase también deberá ser capaz de imprimir campos específicos de un registro, como los campos de nombre y dirección solamente, para hacer listas de correspondencia. Existen paquetes sencillos de este tipo para la mayoría de los ordenadores personales basados en cinta. Sin embargo, si se posee un sistema de disco, la gama del software disponible es mucho mayor.

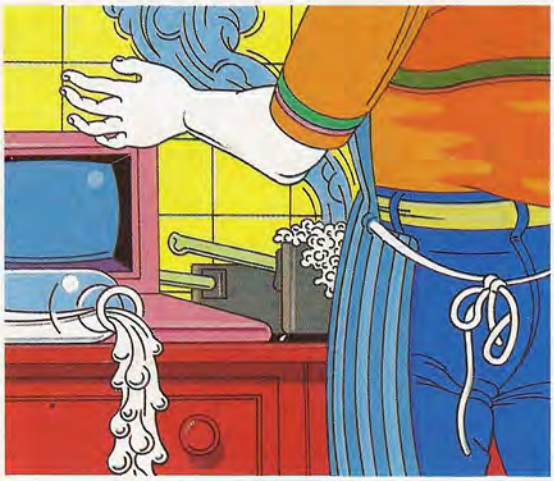
Aparte de los usos que ya hemos sugerido, en los ordenadores personales los sistemas de base de datos se pueden utilizar para catalogar colecciones de discos, libros, referencias bibliográficas, colecciones de sellos, etc.

Los paquetes ligeramente más sofisticados ya permiten diseñar un formulario de entrada de datos por pantalla a tono con un sistema previo basado en papel. Otros paquetes permitirán seleccionar ítems numéricos que sean mayores o menores que una cantidad enunciada (como todos los socios que deben más de 2 000 pesetas, p. ej.). Por consiguiente, es importante rebuscar un poco por las tiendas hasta encontrar el paquete que se adapte mejor a las necesidades específicas.

Un ordenador con un paquete de base de datos es útil para tareas que implican repetición, o cuando hay que clasificar o buscar rápidamente mucha información. Pero hay otras aplicaciones que no son en absoluto adecuadas para los sistemas de base de datos. No parece sensato tener que buscar en un archivo de números de teléfono con un ordenador personal cada vez que uno desea hacer alguna llamada telefónica. En el tiempo que le lleva a usted conectar el ordenador, cargar el programa de

Receta para el desastre

Juan tiene un Vic-20 con el que está muy encariñado. Le ha ayudado a entender los ordenadores y a aprender algo de programación en BASIC, pero lamentablemente Juan no puede distinguir entre lo que es una aplicación seria y lo que es entretenimiento. Insiste en utilizar el ordenador para guardar los números de teléfono de sus (pocos) amigos, y usa una base de datos para almacenar sus recetas de cocina. Cuando uno va a cenar a su casa, invariablemente no puede comer nada hasta pasada la medianoche, porque Juan insiste en verificar los detalles de las instrucciones culinarias en su Vic. Lo enchufa con gesto dramático, carga el programa desde la cinta (para conseguirlo suele realizar varios intentos), espera alegremente durante el interminable período en que el ordenador busca y lee, contempla las instrucciones de unas pocas líneas en la pantalla y, por último, se encamina hacia la cocina. Usted le desenchufa el ordenador y enciende el televisor, sabiendo que la espera será larga. De nada vale llegar tarde a la cena, porque él no empieza hasta que no nos presentamos en su casa, pues para él parte de la fiesta consiste en mostrarte el ordenador en funcionamiento. Es evidente que Juan no usa eficazmente su ordenador.



Roy Ingram

Los discos adecuados

Rosa María siempre había tenido una enorme colección de discos y se las había ingeniado para sacarle el máximo partido. La experiencia le había enseñado que es esencial seleccionar los discos adecuados para cada tipo de reunión. Recientemente Rosa decidió comprarse un ordenador y, por razones de velocidad y de capacidad de almacenamiento, escogió un sistema con dos discos gemelos. Entonces confeccionó sobre el papel los encabezamientos apropiados para las categorías de música que ella deseaba incluir en su programa de base de datos. Había dos grupos principales, cada uno de ellos con subtítulos que a su vez se volvían a subdividir. La división principal era entre música "negra" y música "blanca". La música negra consistía en tres subgrupos principales: reggae, jazz y soul. La música blanca se dividía en rock y nueva ola. El rock blanco se subdividía en *heavy metal* y rock progresivo, y así sucesivamente. Rosa utilizó el gestor de la base de datos para escoger discos según la ocasión (dando, por ejemplo, la orden de listar todos los discos de rock y luego haciendo una selección de nivel inferior de *heavy metal* o bien rock progresivo). Habiendo inspeccionado la lista de los títulos de discos visualizados en la pantalla, hizo un listado por impresora de los títulos elegidos y luego fue a la estantería y cogió los discos que deseaba para esa ocasión.

base de datos, llamar al archivo y encontrar el número de Lupita Pérez, podría realizar seis llamadas telefónicas a números obtenidos de su agenda o de un archivo de fichas de cartera. Se podría pensar en guardar los nombres y números de teléfono en un archivo con alguna otra finalidad, como imprimir cartas y etiquetas estándar, pero utilizar un ordenador cuando los sistemas manuales ordinarios resultan adecuados no tiene el menor sentido.

Un paquete de base de datos más ambicioso ofrecerá todas estas facilidades y proporcionará además la capacidad de utilizar la aritmética para obtener datos tales como la suma total de beneficios obtenidos mediante las cuotas del club, o calcular cuál es el servicio del club que se emplea con mayor frecuencia. Se pueden enlazar varios archivos de modo que los datos de temas relacionados entre sí se puedan utilizar conjuntamente. Por ejemplo, un jugador determinado que pertenezca a nuestro club imaginario tiene sus detalles personales en un archivo y los detalles de su rendimiento en las competiciones del club en otro. Cuando se requiera un historial de su "expediente", los dos archivos se pueden emplear juntos. Para una "instantánea" del rendimiento de todos los socios del club en una fecha determinada, se puede acceder al archivo llamando a cada registro para una cierta fecha, en vez de llamar a cada registro para una cierta persona. Con el fin de ejecutar un programa que posee facilidades tan amplias, casi con seguridad será necesario utilizar un ordenador de sobremesa o de oficina con un mínimo de 64 Kbytes de memoria y unidades de disco.

En los últimos años se ha producido un notable aumento en el empleo de sistemas de información. Los médicos utilizan bases de datos para llevar los detalles del historial clínico de sus pacientes. Quienes trabajan en investigación las emplean para la clasificación especializada y referencias cruzadas de datos. Las empresas llevan listas de correspondencia e informaciones actualizada relativa a los clientes y siempre disponible. Cuanta más información hay disponible, más se la utiliza. En la actualidad las bases de datos pueden tratar cualquier clase de información, desde simples listados hasta complicados sistemas con archivos múltiples, facilidades para crear informes elaborados y la capacidad de procesar información de forma que se pueda transferir a otro software, como los procesadores de textos. La capacidad para comunicarse con bases de datos de acceso público como Prestel y Micronet 800 le proporciona al usuario de un ordenador personal acceso a una enorme cantidad de información. Ello señala el camino hacia un futuro en el que su micro se podría convertir en un terminal enlazado con un gran ordenador central (*mainframe*).

Sprites submarinos

Analizaremos ahora con detalle las rutinas que controlan el barco, el submarino y las cargas de profundidad de nuestro juego "Subhunter"

Para situar un sprite en la pantalla hay que especificar una coordenada X y una Y. Cada sprite se define en un cuadriculado de 21 × 24 pixels y las coordenadas se miden desde la *esquina superior izquierda* del cuadriculado. Las coordenadas se guardan en registros del chip VIC. Éstas se asignan del siguiente modo:

Numero de sprite	0		1		2		3		7	
Coordenada	X	Y	X	Y	X	Y	X	Y	X	Y
Registro VIC	V	V+1	V+2	V+3	V+4	V+5	V+6	V+7	V+14	V+15

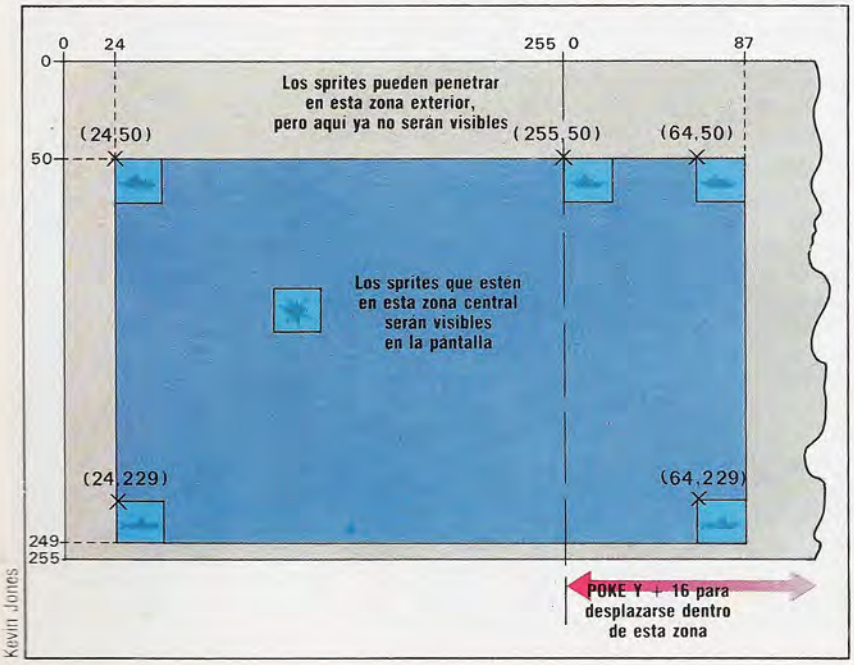
Cada posición puede aceptar números comprendidos en la escala 0 a 255. Esto es más que suficiente para especificar uno de los 200 pixels en la dirección vertical (Y), y la capacidad restante se utiliza para permitir que el sprite aparezca o desaparezca por la parte superior o inferior de la pantalla. Sin embargo, los 320 pixels de la dirección horizontal (X) superan el máximo de 255 que permite un registro de ocho bits, de modo que a cada sprite se le destina otro bit. Estos bits extras se agrupan juntos en un único registro con la dirección V+16. El diagrama inferior muestra los límites de la pantalla para sprites totalmente visibles y sin ampliar.

El movimiento de la embarcación se controla mediante las líneas del programa 230-250 y 270-290. Las coordenadas iniciales para el barco se establecieron en la línea 2270 de la rutina de creación de sprites (véase p. 745). El barco sólo se desplazará en dirección horizontal, de modo que la coordenada Y permanecerá fija. Si ésta se pone a 80, el barco estará correctamente situado en el océano, que se diseñó mediante la rutina de preparación de la pantalla. La coordenada X del barco, X0, se pone inicialmente en 160, que proporciona una posición de comienzo centrada en la pantalla.

El barco se controla mediante el teclado, utilizando las teclas "Z" y "X" para el movimiento a izquierda y derecha. Las líneas 230-250 del bucle principal obtiene (GET) un carácter pulsado en el teclado. Si no ha sido pulsada ninguna tecla, la ejecución del programa continúa (al contrario que la orden INPUT, que interrumpe un programa hasta que se pulsa una tecla). El valor obtenido por GET se almacena luego en A$ y se emplea para modificar la coordenada X del barco: si se pulsa la tecla "Z", el barco se mueve una pequeña distancia hacia la izquierda y es decrementada la coordenada X; si se pulsa la tecla "X" el barco se desplazará la misma distancia hacia la derecha y la coordenada X será incrementada. La segunda parte de las líneas 240 y 250 contiene una condición para ver si el barco ha alcanzado los límites de su viaje. La estructura IF...THEN del BASIC Commodore es tal que si la primera condición de una línea de programa no se cumple, el resto de la línea no se ejecuta. En nuestro programa, la verificación de la condición está dispuesta de modo que el ordenador no pierda tiempo en cálculos innecesarios.

El movimiento del submarino está controlado por las líneas 300-350 y 2500-2570 y opera dentro del bucle principal del programa, tal como refleja el diagrama de flujo contiguo. La subrutina "restaurar coordenadas submarino" que empieza en la línea 2500 utiliza la función RND para seleccionar la profundidad del submarino, Y3, y la velocidad, DX. Y3 siempre será un número entero comprendido entre 110 y 214, lo que asegura que el sumergible no aparezca por encima de la superficie del mar o por debajo del lecho marino. El factor de velocidad, DX, varía entre 1 y 3 y determina la cantidad de pixels con la que se incrementará o decrementará la coordenada X del submarino. La coordenada X del submarino y el bit de registro V+16 que controla las coordenadas X del sprite 3 por encima de 255 están ambas puestas a cero.

En las líneas 300-350 del bucle principal se le suma el valor seleccionado de DX a la coordenada X del submarino y ésta se verifica entonces para ver si éste ha alcanzado el borde de la pantalla. Las líneas 340 y 350 nos permiten tratar el problema de tener valores de X3 que superen 255. Cuando la coordenada X aumenta a, supongamos, 256, deben suceder dos cosas: el bit correspondiente al sprite 3 en el registro V+16 se pone a uno y el registro normal



La posición de los sprites
Este diagrama refleja los parámetros para situar los sprites en la pantalla del Commodore 64. Un sprite situado dentro de la parte central de esta pantalla será visible en la pantalla. Si el sprite se desplaza hasta la parte exterior de la pantalla, entonces no aparecerá a la vista. Los cuatro sprites de la esquina de la zona central muestran los límites hasta dónde se puede mover un sprite y ser todavía completamente visible. Se indican las coordenadas de las esquinas superiores izquierda de los sprites

Kevin Jones

de la coordenada X vuelve a empezar desde cero. La siguiente tabla muestra lo que sucede en los registros a medida que el submarino cruza la frontera de X = 255:

X3	V+16								V+6							
254	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
255	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
256	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
257	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1

La variable H3 se pondrá a uno si el valor de X3 sobrepasa 255. De modo similar, L3 se pondrá de nuevo a cero si H3 se convierte en uno. Los valores de L3 y H3 se pueden colocar entonces (POKE) en los registros V+6 y V+16.

Durante el juego pueden arrojarse cargas de profundidad en cualquier momento. Para hacer un programa directo, obedeceremos la regla de que después de que se haya lanzando una carga no se podrá disparar ninguna otra hasta que:

a) se haya acertado al submarino; o

b) las cargas hayan rozado el submarino; o

c) las cargas no hayan dado en el submarino y hayan llegado al lecho marino.

El bucle principal del programa tiene que realizar dos tareas con respecto a las cargas de profundidad: debe detectar la pulsación de la tecla "M" y, después de disparada la carga de profundidad, debe controlar su movimiento vertical. Asimismo, el programa debe asegurar que no se disparen nuevas cargas de profundidad mientras haya alguna en proceso de caída. Este último problema se puede resolver mediante la utilización de un indicador. Ésta es una técnica que se suele aplicar con frecuencia en el control de programas, señalando que un acontecimiento determinado se ha producido o no. En nuestro programa vamos a emplear la variable FL para indicar el disparo de una carga de profundidad. Su valor será uno si hay una carga cayendo, de lo contrario será cero. En la línea 100 del programa el valor de FL se pone inicialmente a cero. La línea 260 accede a la subrutina "Preparar cargas de profundidad" en la línea 3000 si se pulsa "M" y el indicador está en cero. En la línea 400 se utiliza una segunda subrutina para mover un sprite de carga de profundidad disparada, y a éste se accede por la línea 380.

La subrutina "preparar cargas de profundidad" tiene que realizar tres funciones:

1) Poner el indicador FL a uno como señal de que se ha arrojado una carga de profundidad.

2) Establecer las coordenadas de comienzo: la coordenada X toma su valor a partir de la del barco y la coordenada Y se pone inicialmente en 95, situando la carga bajo la superficie del mar.

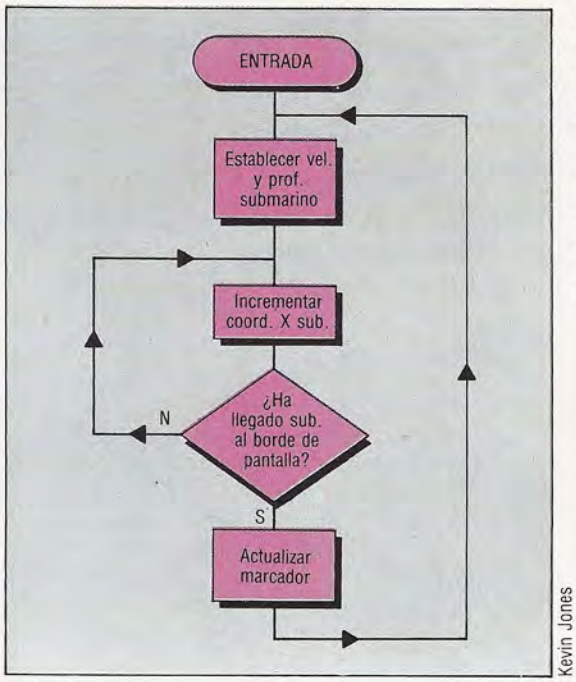
3) Encender el sprite de la carga.

La subrutina "mover carga de profundidad" se utiliza para mover la carga de profundidad hacia abajo por la pantalla. Además, se han de verificar estas condiciones, si:

1) La carga de profundidad ha pasado junto al submarino *o* (OR) alcanzado el fondo.

2) La carga de profundidad ha hecho blanco en el submarino.

Si se ha producido lo primero, se puede apagar el sprite de la carga de profundidad y restablecer el indicador a cero, permitiendo arrojar otra carga de profundidad. El segundo acontecimiento se verifica utilizando otra característica de los sprites del Commodore 64: el registro de colisión de sprites. Al igual que otros registros del chip VIC, este registro, V+30, posee un bit correspondiente a cada sprite. Si un sprite determinado se ve implicado en una colisión con otro sprite, entonces el bit correspondiente de este registro se pone a uno. Por consiguiente, si el submarino (sprite 3) y la carga de profundidad (sprite 2) colisionan, el contenido del registro V+30 será 12 (00001100 = 12). Tomando (PEEK) este registro y verificando su contenido podemos saber si la carga de profundidad ha hecho blanco en el submarino. Si así fuera, en la línea 5000 se accedería a otra subrutina "HIT" (blanco). De esta subrutina nos ocuparemos en el capítulo final del proyecto, así como de las instrucciones para actualizar el MARCADOR MAX y volver a comenzar el juego.



Kevin Jones

Diagrama de flujo subacuático
Este sencillo diagrama de flujo muestra cómo el programa controla el movimiento del sprite del submarino. Selecciona al azar una profundidad y una velocidad, asegurándose de que el sumergible esté por debajo de la superficie y por encima del fondo. El submarino se desplaza entonces de forma uniforme a través de la pantalla hasta llegar al otro lado

Subrutinas de movimiento del Subhunter

```
130  GOSUB 2500:REM ESTABLECER COORDENADAS SUB
230  GET A$
240  IF A$="Z" THEN X0=X0-1.5:IF X0<24 THEN X0=24
250  IF A$="X" THEN X0=X0+1.5:IF X0>245 THEN X0=245
260  IF A$="M" AND FL=0 THEN GOSUB 3000:REM PREPARAR
     CARGAS PROFUNDIDAD
265  :
270  REM ** MOVER BARCO **
290  POKE V,X0
295  :
300  REM ** MOVER SUB **
310  X3=X3+DX
315  :
320  REM ** SUB EN FIN PANTALLA? **
330  IF X3>360 THEN DS=-1:GOSUB 5500:GOSUB 2500
340  H3=INT(X3/256):L3=X3-256 * H3
350  POKE V+6,L3:POKE V+16,PEEK(V+16)OR(8*H3)
360  IF FL=1 THEN GOSUB 4000:REM MOVER CARGA PROFUNDIDAD
370  GOTO 200:REM REITERAR BUCLE PRINCIPAL
380  :
390  :
2500  REM **** RESTAURAR COORDENADAS SUB ****
2510  Y3=110+INT(RND(TI)*105)
2520  X3=0:DX=RND(TI)*3+1
2530  POKE V+7,Y3:POKE V+6,X3
2540  POKE V+16,PEEK(V+16) AND 247
2550  RETURN
2560  :
2570  :
3000  REM *** PREPARAR CARGAS PROFUNDIDAD ***
3010  :
3020  REM ** PONER INDICADOR **
3030  FL=1
3040  :
3050  REM ** ESTABLECER COORDENADAS **
3060  Y2=95:X2=X0
3070  POKE V+4,X2:POKE V+5,Y2
3080  :
3090  REM ** ENCENDER SPRITE 2 **
3100  POKE V+21,PEEK(V+21)OR4
3110  RETURN
3120  :
3130  :
4000  REM ** MOVER CARGA PROFUNDIDAD **
4010  :
4020  REM ** INCREMENTAR COORD Y **
4030  Y2=Y2+2
4040  POKE V+5,Y2
4050  :
4060  REM ** VERIFICAR FONDO Y APAGAR **
4070  IF Y2>Y3+25 OR Y2>216 THEN POKE V+21,PEEK(V+21) AND
     251:FL=0
4080  :
4090  REM ** VERIFICAR SI SUBMARINO ACERTADO **
4100  IF PEEK (V+30)=12 THEN GOSUB 5000:REM RUTINA HIT
4110  RETURN
4120  :
4130  :
```

Sortilegio de la suma

Presentamos un rompecabezas matemático, el "cuadrado mágico", cuya implementación en ordenadores personales resulta muy interesante

Un cuadrado mágico es un cuadriculado de celdas en las que se introducen números enteros positivos (1, 2, 3, 4, 5, etc.), menos el cero, sin que se repita ninguno de ellos. El objetivo de este ejercicio consiste en distribuir tales números de forma que al sumarlos en cada columna y cada fila den el mismo resultado. El cuadrado mágico más simple es una caja de 3x3, y una posible solución es la que ofrecemos a continuación:

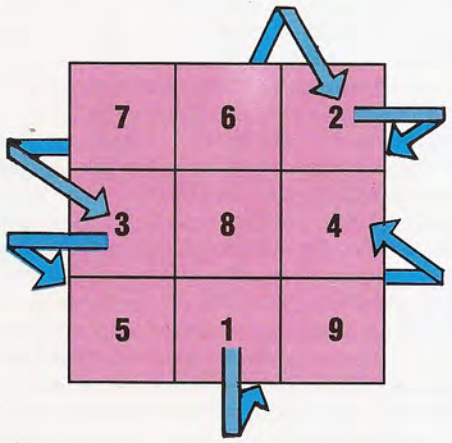
7	6	2	15
3	8	4	15
5	1	9	15
15	15	15	

En este ejemplo, todas las fichas y columnas suman 15. A modo de ejercicio, pruebe a construir usted mismo un cuadrado, pero esta vez utilice uno de 5x5 y use los números del 1 al 25. Descubrirá que no es tán fácil como parece, y de seguro le obligará a un buen número de ensayos erróneos antes de obtener la solución correcta.

Pero su ordenador puede hacer que las cosas resulten más sencillas. Una solución es escribir un programa para rellenar los cuadrados con números y luego verificar la suma de cada fila y cada columna. Sin embargo, éste método podría tardar varias horas en hallar la solución incluso para cuadrados bastante pequeños, de siete o nueve celdas de anchura. Lo que hace falta es un procedimiento para generar los números. Para ahorrarle el esfuerzo de descubrir su propio método, he aquí uno que funciona siempre:

1) Empezar con el número 1 en la celda central de la fila anterior.
2) Después de llenar cada celda, desplazarse una celda hacia abajo y una a la derecha antes de poner el número siguiente. Si al desplazarse hacia la derecha se sale del borde derecho del cuadriculado, vaya a la primera columna. Del mismo modo, si al desplazarse hacia abajo se sale de la parte inferior de la rejilla, pase entonces a la fila superior.
3) Si la celda que toca llenar ya está ocupada, desplazarse una posición hacia la izquierda de la última celda utilizada.
4) Continuar así hasta llenar todas las celdas.

La aplicación de este procedimiento a nuestro ejemplo nos muestra cómo se genera un cuadrado de 3x3.



Es muy fácil escribir un programa que haga esto para un cuadrado de cualquier tamaño. Tan sólo hay que crear una matriz bidimensional vacía de las dimensiones correctas para el cuadrado mágico deseado y luego construir un bucle para rellenarla de acuerdo a las reglas dadas. Observará que el méto-

Cuadrados encantados

Estos cuadrados se generaron con el programa y éste los ha verificado con éxito. En el cuadrado de 15 × 15 se puede observar una configuración en diagonal de números de tres y dos dígitos, que es consecuencia del algoritmo usado para su construcción

52	42	32	22	12	2	73	72	62
63	53	43	33	23	13	3	74	64
65	55	54	44	34	24	14	4	75
76	66	56	46	45	35	25	15	5
6	77	67	57	47	37	36	26	16
17	7	78	68	58	48	38	28	27
19	18	8	79	69	59	49	39	29
30	20	10	9	80	70	60	50	40
41	31	21	11	1	81	71	61	51

Ian McKinnell

do sólo funciona para cuadrados mágicos con una cantidad impar de celdas; su programa deberá rechazar los cuadriculados de número par.

Hay que tener cuidado cuando se visualiza el cuadrado. En aras de la pulcritud, todos los números se deben alinear correctamente en filas y columnas. Esto es bastante fácil de conseguir si su micro posee la orden PRINT USING. De no ser así, es mejor convertir el número a imprimir en una serie. Ésta se puede entonces rellenar con caracteres espacio de modo que el "número" siempre tenga la misma longitud. Una subrutina para conseguirlo es:

```
1000  REM Convertir A en A$ y alinear
1010  A$=STR$(A)
1020  IF LEN(A$)<3 THEN A$=" "+A$:GOTO
1020
1030  RETURN
```

El método exacto dependerá, por supuesto, del ordenador que se esté utilizando.

El siguiente problema es el tamaño de la pantalla; la mayoría de los micros son incapaces de visualizar en pantalla cuadrados mágicos grandes. Una pantalla de 40 columnas tiene espacio para 13 columnas de dos dígitos, pero un cuadrado de 13x13 incluirá números de tres dígitos, de modo que el más grande que se puede visualizar es un cuadrado de 9x9. Una impresora permitirá generar cuadrados mucho más grandes. La mayoría de las impresoras tienen una anchura máxima de 80 o 132 columnas y se pueden imprimir por secciones cuadrados más grandes que pueden ser unidos posteriormente.

El objetivo global de este proyecto es crear el cuadrado mágico más grande que se pueda y presentarlo con la mayor pulcritud posible. Como experimento, intente escribir un programa de ensayo y error y determine cuánto tarda en hallar la respuesta (aunque conseguir un resultado correcto le ocupará bastante tiempo). O intente buscar procedimientos aún más rápidos para generar los cuadrados.

114	98	82	66	50	34	18	2	211	210	194	178	162	146
131	115	99	83	67	51	35	19	3	212	196	195	179	163
148	132	116	100	84	68	52	36	20	4	213	197	181	180
165	149	133	117	101	85	69	53	37	21	5	214	198	182
167	151	150	134	118	102	86	70	54	38	22	6	215	199
184	168	152	136	135	119	103	87	71	55	39	23	7	216
201	185	169	153	137	121	120	104	88	72	56	40	24	8
218	202	186	170	154	138	122	106	105	89	73	57	41	25
10	219	203	187	171	155	139	123	107	91	90	74	58	42
27	11	220	204	188	172	156	140	124	108	92	76	75	59
44	28	12	221	205	189	173	157	141	125	109	93	77	61
46	45	29	13	222	206	190	174	158	142	126	110	94	78
63	47	31	30	14	223	207	191	175	159	143	127	111	95
80	64	48	32	16	15	224	208	192	176	160	144	128	112
97	81	65	49	33	17	1	225	209	193	177	161	145	129

```
10  REM***************
15  REM***CUADRADOS MAGICOS *****
20  REM*******PREPARACION****
30  M=19:DIM A(M,M)
40  PRINT:PRINT"Cuadrados mágicos"
50  PRINT:PRINT"Cuantas filas?(1 a 19)";INPUT S
60  IF S<0 OR S<> INT(S) THEN PRINT"ERROR":GOTO 50
70  IF S>M THEN PRINT"ERROR":GOTO 50
80  IF S/2=INT(S/2) THEN PRINT"ERROR - Solo numeros impares":GOTO 50
90  REM****GENERAR CUADRADO****
100  X=INT(S/2)+1:Y=S:C=1
110  A(X,Y)=C
120  C=C+1:IF C>S*S THEN GOTO 200
130  X=X+1:IF X>S THEN X=1
140  Y=Y+1:IF Y>THEN Y=1
150  IF A(X,Y)<>0 THEN X=X-2:Y=Y-1
160  IF Y=0 THEN Y=S
170  IF X=0 THEN X=S
180  IF X=-1 THEN X=S-1
190  GOTO 110
200  REM*****IMPRIMIR CUADRADO****
210  PRINT:PRINT
220  FOR Y=1 TO S:FOR X=1 TO S
230  A=A(X,Y):GOSUB 380:PRINT"  ";A$;"  ";
240  NEXT X:PRINT:NEXT Y

250  REM****VERIFICAR FILAS Y COLUMNAS*****
260  F=0
270  FOR Y=1 TO S:T=0
280  FOR X=1 TO S:T=T+A(X,Y):NEXT X
290  IF F=0 THEN U=T:F=1
300  IF T<>U THEN PRINT"ERROR - Fila 1 y fila";Y;"no coinciden":STOP
310  U=T:NEXT Y
320  FOR X=1 TO S:T=0
330  FOR Y=1 TO S:T=T+A(X,Y):NEXT Y
340  IF T<>U THEN PRINT"ERROR —Fila 1 y columna";X;"no
     coinciden":STOP
350  U=T:NEXT X
360  PRINT:PRINT"Todas las filas y columnas suman ";T
370  STOP
380  REM***CONV NUM-SERIE***
390  A$=STR$(A)
400  IF LEN(A$)<3 THEN A$=" "+A$:GOTO 400
410  RETURN
```

Complementos al BASIC

Este programa está escrito en BASIC Microsoft, de modo que funcionará sin ninguna modificación en la mayor parte de los mismos. Ya saben los usuarios del Spectrum que deben insertar LET antes de todas las sentencias de asignación. El programa pide que se le suministre el número de filas (y, por lo tanto, de columnas) del cuadrado mágico, y verifica que éste sea un número impar, entero y positivo. Calcula y visualiza el cuadrado mágico y desde la línea 250 verifica su resultado.

Números aleatorios

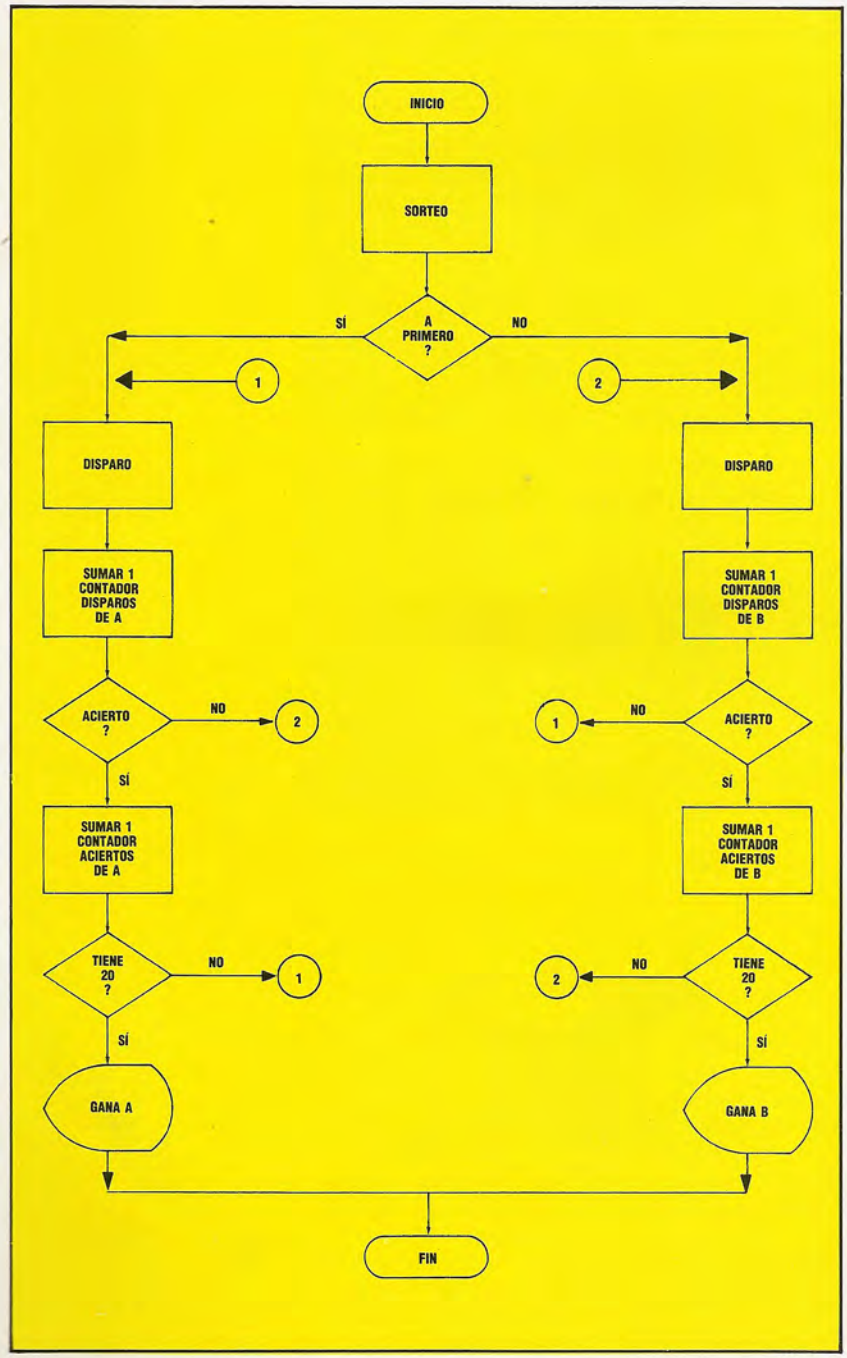
El orden de intervención de los dos jugadores que citamos en nuestro ejemplo lo determina la función RND al generar un número aleatorio

El programa siguiente es la transcripción al BASIC del ordinograma del capítulo anterior, con control del tiro al plato con contadores.

La parte que más difícil puede parecer a la hora de programar es la correspondiente al sorteo inicial. Este punto se ha resuelto con la utilización de la función RND, que permite generar números aleatorios. Éstos se crean mediante algoritmos que recurren a cálculos ya determinados y que dan una serie de números que parecen elegidos al azar.

Así, en el caso presente, se va a generar un número, el 1 o el 2 solamente, de modo que se pueda comparar el número y en función de él dar la señal del inicio al jugador A en caso de aparecer el 1 y al jugador B en caso de aparecer el 2 (sentencia 20). Igual proceso de comparación ocurre en las sentencias 50 y 110 cuando se trata de saber si se ha acertado el disparo (número 1) o se falló (número 2).

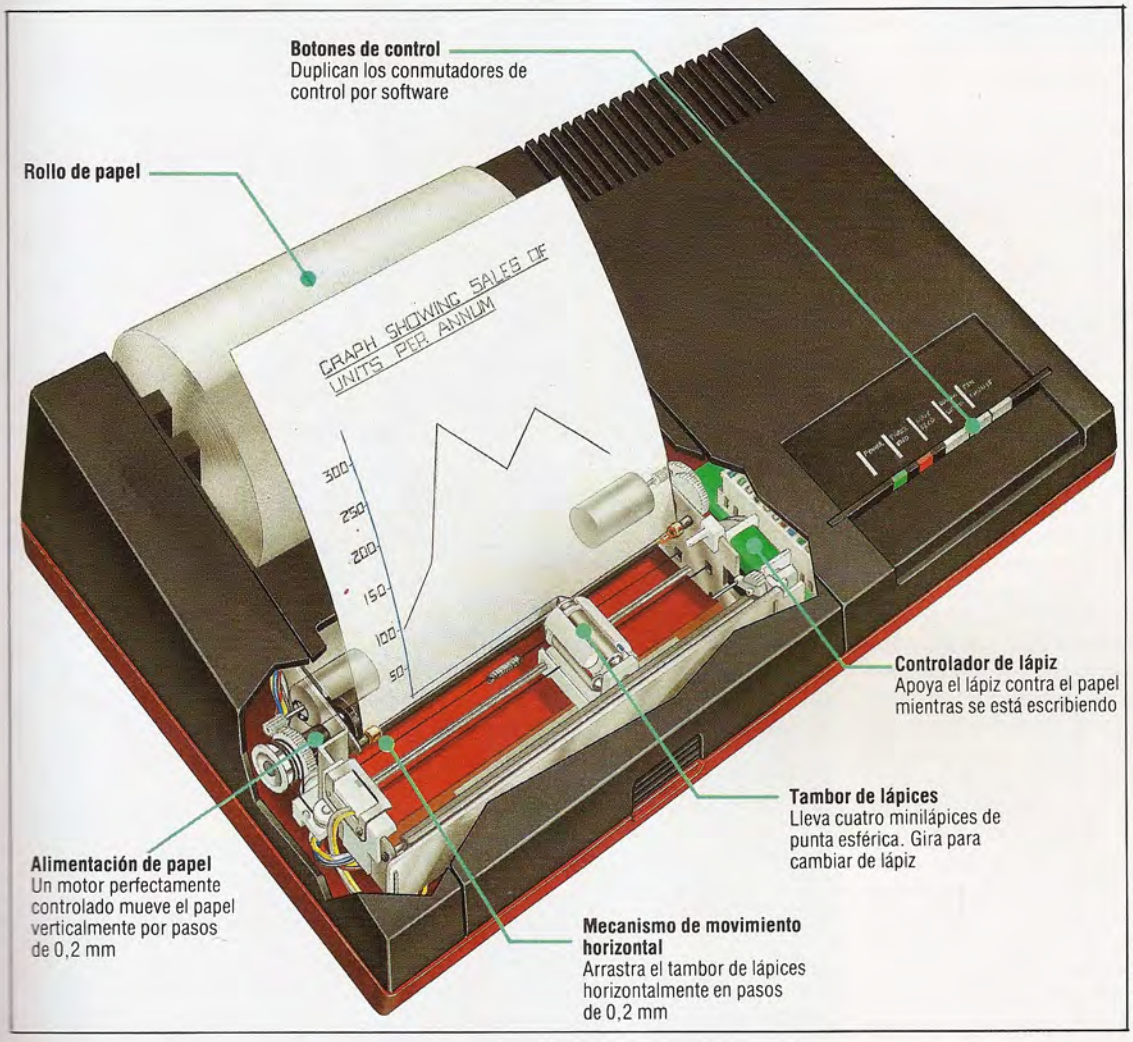
El programa se ha realizado para un Commodore 64; los contadores de disparos y los contadores de aciertos de uno y otro jugador se han denominado CA y CB, y A y B, respectivamente.




```
10  REM****TIRANDO AL PLATO****
15  REM****SORTEO*******
20  S=INT(RND(1)*2)+1
25  IF S=1 THEN PRINT"COMIENZA EL JUGADOR
    A" : GOTO 45
30  PRINT"COMIENZA EL JUGADOR B"
35  GOTO 100
40  REM*****DISPARANDO EL JUGADOR A
45  PRINT "DISPARA A"
50  X=INT(RND(1)*2)+1
55  CA=CA+1
60  REM****DECISION ACIERTO O FALLO
65  IF X=1 THEN GOTO 80
70  PRINT "FALLO"
75  GOTO 100
80  PRINT "ACIERTO"
83  A=A+1
85  IF A<>20 THEN GOTO 45
90  PRINT "GANADOR JUGADOR A EN" CA "
    DISPAROS"
95  END
99  REM*****DISPARANDO EL JUGADOR B
100  PRINT "DISPARA B"
110  R=INT(RND(1)*2)+1
120  CB=CB+1
130  REM****DECISION ACIERTO O FALLO
140  IF R=1 THEN GOTO 170
150  PRINT "FALLO"
160  GOTO 45
170  PRINT "ACIERTO"
175  B=B+1
180  IF B<>20 THEN GOTO 100
190  PRINT "GANADOR JUGADOR B EN" CB "
    DISPAROS"
200  END
```

Plumas afines

Veremos aquí un modelo de impresora-plotter que se comercializa bajo diversos nombres y que es especialmente adecuada para ser usada con micros



Steve Cross

Trazado de calidad
La impresora-plotter tiene una resolución de alrededor de 500 ×2 000 pixels en modalidad plotter y una gama de tamaños. El color del lápiz se selecciona haciendo girar el tambor de lápices, y trazar o imprimir implica desplazar el tambor de los lápices horizontalmente a izquierda y derecha y enrollar y desenrollar el papel. El lápiz se puede apoyar contra el papel o apartarlo del mismo, para permitir el trazado o el movimiento sin huella, respectivamente. El mecanismo es capaz de realizar un trazado de gran precisión, pero el movimiento repetido del papel y el tambor produce una ligera pérdida de precisión

Atari 1020, Commodore 1510 y Oric MCP-40 son algunos de los nombres con los que se comercializa un plotter para microordenador notablemente barato. La parte interna de todas estas máquinas la fabrica una firma japonesa, adaptándola a las exigencias de cada empresa.

Además de su capacidad para gráficos, el plotter también puede imprimir textos, y por esa razón se lo conoce como *impresora-plotter*. Dentro de la unidad hay un cabezal rotatorio que contiene cuatro pequeños lápices de punta esférica. Para trazar una línea, el cabezal gira para seleccionar el color adecuado (los colores rojo, azul, verde y negro vienen como estándar) y luego el lápiz elegido se desplaza hasta hacer contacto con el papel. Una línea horizontal se traza moviendo el cabezal de un lado a otro; una línea vertical se traza mediante el movimiento hacia arriba y hacia abajo del papel. El texto se produce de forma muy similar a los gráficos; la impresora-plotter almacena los patrones para letras y otros caracteres en su propia memoria. Cuando se recibe una señal proveniente del ordenador, la impresora-plotter simplemente busca el carácter adecuado en su memoria interna y luego lo dibuja como si fuera un patrón de gráficos. La calidad de impresión resultante es excelente; por cierto, superior a la de la mayoría de las impresoras matriciales económicas.

Cuando está en modalidad de texto, al impresora-plotter opera exactamente igual que cualquier otra impresora. Aunque el papel que utiliza sólo tiene 115 mm de ancho, la unidad puede producir 40 u 80 caracteres por línea. La escasa anchura del papel hace que el texto de 80 caracteres quede más bien pequeño, pero la definición de los caracteres es muy buena. El texto se puede impri-

mir en cualquiera de los cuatro colores, con una velocidad de impresión de unos 10 caracteres por segundo. Esto es lento en comparación con la mayoría de las impresoras matriciales, pero resulta más o menos igual a la velocidad de las de rueda margarita.

Para producir gráficos, la impresora-plotter se coloca en modalidad de gráficos y entonces produce líneas, curvas y letras grandes. Si a la unidad se le envían caracteres estando en modalidad de gráficos, los interpretará como órdenes y no intentará imprimirlos. Por consiguiente, si necesitamos trazar líneas, emplearemos una orden en BASIC como la que ofrecemos:

```
LPRINT "D 0,100,100,100,100,0,0,0"
```

La D es la instrucción para dibujar líneas y los números que siguen dan las posiciones de los puntos a través de los cuales pasarán las líneas. En este caso (suponiendo que el cabezal de impresión esté posicionado en las coordenadas (0,0) al principio) se dibujará un cuadrado. Otras órdenes siguen un formato similar.

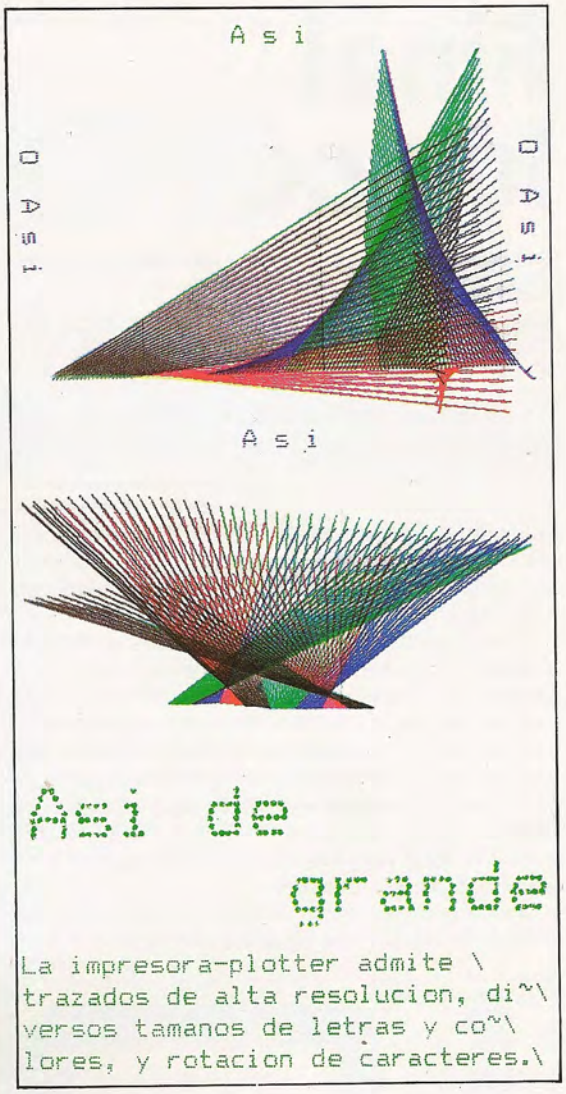
La anchura del papel se considera dividida en 480 pasos horizontales, que se utilizan para posicionar el cabezal de impresión. El papel empleado se suministra en un rollo de varios metros de longitud. Esto podría hacer suponer que no existen límites para la longitud vertical de un dibujo, pero en realidad el ligero resbalamiento del papel al moverlo verticalmente puede ocasionar que las líneas no encajen de la forma requerida, de modo que la impresora-plotter no permitirá que el papel se vuelva a enrollar más de una cierta distancia.

Aunque con la impresora-plotter se pueden realizar algunos gráficos muy complejos, escribir los programas para hacerlos lleva mucho tiempo; no hay ninguna forma sencilla de copiar una imagen directamente a partir de la pantalla del ordenador. El trazado de líneas es simple, pero la ausencia de una orden PAINT o FILL significa que sólo se pueden obtener bloques de color sólido trazando muchas líneas finas. Esto es lento y supone un desperdicio de tinta. El dispositivo tampoco puede dibujar en toda la anchura del papel, sino que hay que dejar un pequeño espacio vacío a cada lado. La unidad es especialmente apta para trazar gráficos y dispone de una orden que dibuja automáticamente los ejes de un gráfico, junto con las unidades correspondientes marcadas a intervalos elegidos.

En la modalidad de texto, la impresora-plotter produce dos tamaños de letra distintos. Sin embargo, utilizando la modalidad de gráficos se dispone de tamaños de texto mucho mayores y se permite imprimir letras de lado, pudiendo así imprimir mensajes largos en toda la longitud del papel. A los lápices de punta esférica se les acaba enseguida la tinta y es probable que se sequen si se los deja en el cabezal de impresión. Cada vez que la unidad se conecta, todos los lápices se prueban de forma automática, dibujando un cuadrado pequeño en cada color.

Los plotters profesionales son sumamente precisos y producen gráficos en color de una gran calidad. No sería razonable esperar que una unidad de este precio igualara estos niveles, pero hay que reconocer que los resultados conseguidos con la impresora-plotter tienen un nivel sorprendentemente bueno.

Chris Stevens



Interfaces para plotter
La tabla muestra las unidades que se pueden utilizar con los micros más usuales. El único problema será obtener el cable adecuado para conectarlos entre sí. Oric, Sharp, Commodore y Atari incluyen los cables necesarios para sus propios micros. Un ordenador popular que no está incluido en la lista es el Sinclair Spectrum, que carece de una interface adecuada

	Atari 1020	Commodore 1520	MCP-40	Oric MCP-40	Sharp MZ-80P5(K)	Tandy CGP-115
Atari 400/600/800	●					
BBC Micro			●	●		●
CGL/Sord M5			●	●		●
Colour Genie						●
Commodore Vic-20		●				
Commodore 64		●				
Dragon 32/64			●	●		●
Memotech MTX			●	●		●
Oric-1/Atmos			●	●		●
Sharp MZ-700					●	
Tandy TRS-80 Colour						●

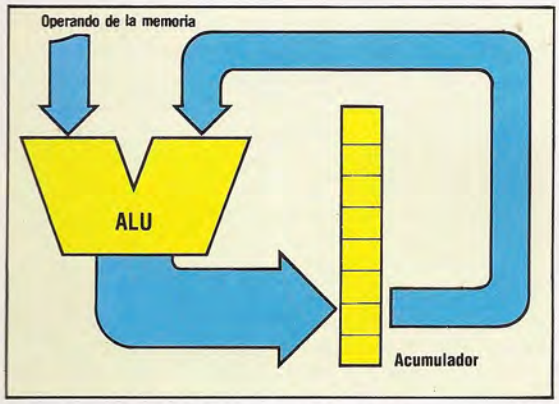
Final lógico

Concluimos esta serie estudiando la lógica de la CPU, y en concreto las funciones de la unidad aritmética lógica (ALU)

Son dos los tipos de funciones desempeñadas por la ALU: las *aritméticas* (suma, resta e incremento en una unidad), y las *lógicas* (que se resumen en tres: OR, AND y XOR).

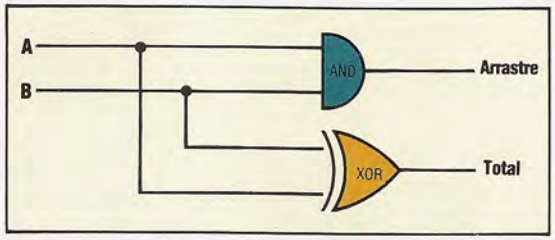
Funciones tales como la suma necesitan dos operandos (o sea, dos números con los que se realiza la operación). Otras, como el incremento, se bastan con un operando que se toma de un registro especial de la CPU llamado acumulador. Cuando son necesarios dos operandos, uno de ellos procede de la memoria principal. Ambos se moverán por los circuitos de la ALU donde tiene lugar la operación escogida. El resultado de la operación queda, finalmente, en el acumulador.

Veamos el recorrido de los datos a través del chip de la ALU en el siguiente dibujo:

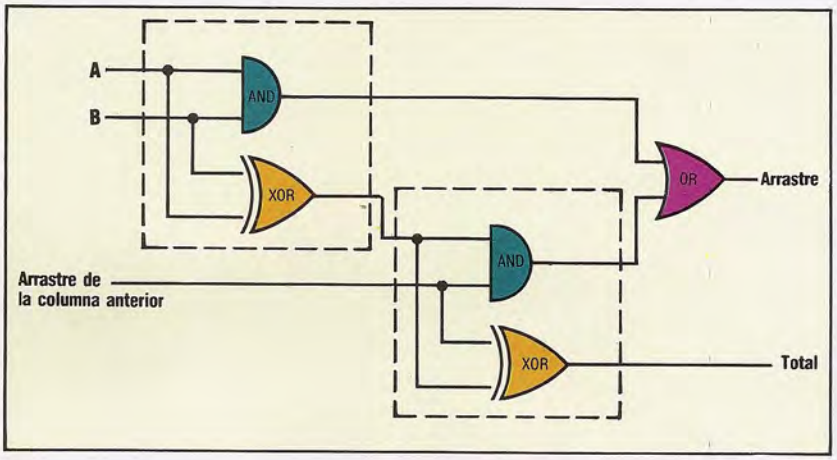


Los números, tanto en el acumulador como en la memoria, tienen una longitud de ocho bits. Estos bits que conforman los números circulan en paralelo por los circuitos de la ALU y la operación se realiza con los ocho bits a la vez. Entenderemos mejor los circuitos de la ALU si nos fijamos en uno solo de estos bits diseñando un circuito que realice con él las seis funciones mencionadas. Al bit del primer operando lo denominaremos A y al del segundo operando, B.

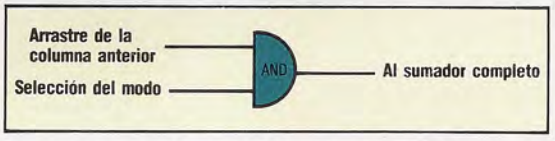
En la base de todo el dispositivo se encuentra el sumador completo. Ya lo dejamos diseñado en un capítulo anterior (véase p. 526), formado por dos circuitos sumadores incompletos que combinan las puertas AND, OR y NOT. Pero estos sumadores incompletos pueden simplificar su circuito usando la puerta XOR (o puerta OR exclusiva), como se ve en el primer diagrama de la columna derecha.



Acoplando dos sumadores incompletos, obtenemos un circuito sumador completo (segundo diagrama de esta columna).



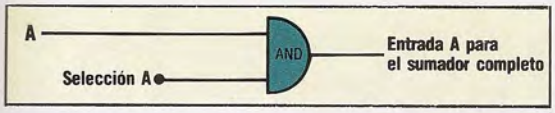
Vamos ahora a ver cómo añadiendo unos pocos circuitos, este sumador completo es capaz de realizar las restantes funciones de la ALU. Para ello estableceremos una serie de señales de control, la más importante de las cuales es la conocida como señal de *selección del modo*. Las operaciones aritméticas necesitan la entrada que llamamos "arrastre de la columna anterior", pero no así las operaciones lógicas. La señal de selección del modo conmutará esa entrada de arrastre y la pondrá en off o en on. Esto se logra con introducir una puerta AND:

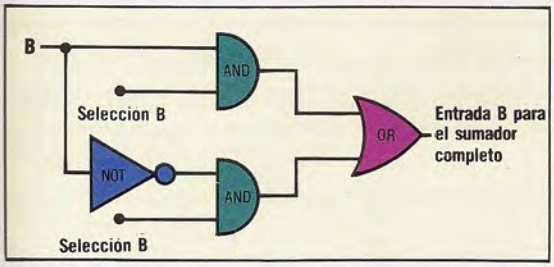


Si ponemos en 1 la señal de selección del modo, la entrada del arrastre, necesaria en las operaciones aritméticas, se respeta gracias a la puerta AND. Para los casos en que el arrastre no es necesario, bastará poner a 0 esta señal. Como podemos añadir estas puertas AND en una u otra entrada, es posible seleccionar el bit A, el bit B o ambos.

En el proceso de resta utilizando la suma en complemento a dos, es necesario calcular el complemento a dos del sustraendo. Se deben cambiar, como dijimos, los unos en ceros, y los ceros en unos. Esto significa que debemos añadir a nuestro circuito sumador, si lo queremos emplear en las restas, algún otro circuito que seleccione la negación del bit B. Se trata de afectar la entrada B con una puerta NOT e incluir la señal de selección del modo empleando de nuevo la puerta AND. Tendríamos, para las entradas A y B, el diseño de circuitos de los dos primeros diagramas de p. 773.

Con estas cuatro señales de control es posible realizar cualquiera de las tres funciones aritméticas



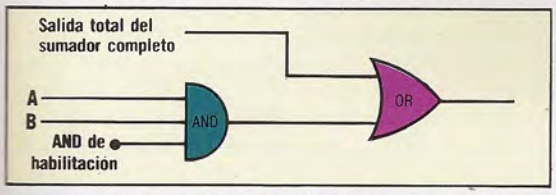


antes mencionadas. La tabla ilustra las combinaciones necesarias:

Función	Selección modo	Selecc. A	Selecc. B	Selecc. B̄
Suma	1	1	1	0
Resta	1	1	0	1
Incremento de A (el 1.er arrastre se pone en 1)	1	1	0	0
Increm. de B	1	0	1	0

Para las operaciones lógicas, ya que prescinden del "arrastre de la columna anterior", la señal de selección del modo puede ponerse en cero. Esto significa que para obtener la función lógica XOR debemos colocar en HI (alto) tanto la selección A como la selección B, y en LO (bajo) la selección del modo.

La función AND no se obtiene tan directamente, pues necesita las entradas A y B por separado, junto con una señal de selección con AND.



Por último la función OR puede crearse combinando las salidas de XOR y AND a través de una puerta OR. Lo demuestra la tabla de verdad que ofrecemos a continuación:

A	B	Salida	Comentario
0	0	0	
0	1	1	Función XOR
1	0	1	
1	1	1	Función AND

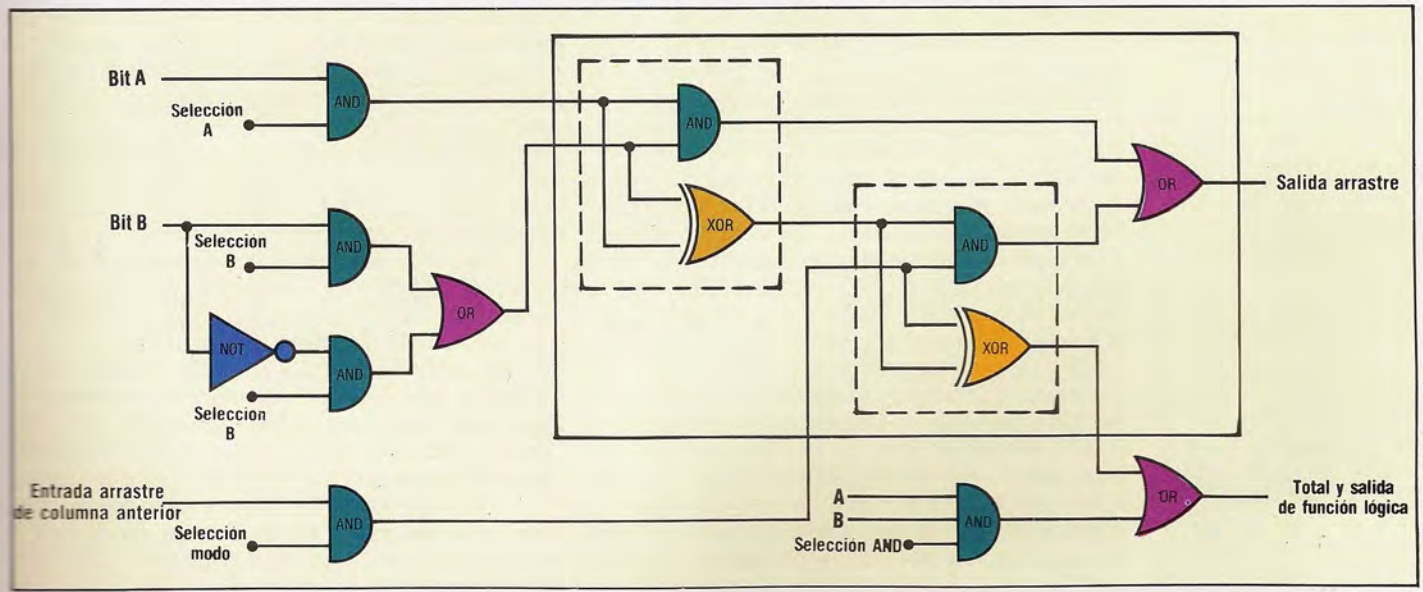
Esta otra tabla muestra la manera de obtener cada una de las funciones lógicas mediante combinaciones de las señales de control:

Funcion	Selección modo	Selección A	Selección B	Selección B̄	Selección AND
XOR	0	1	1	0	0
AND	0	0	0	0	1
OR	0	1	1	0	1

El diseño final que concluye este capítulo nos muestra un circuito de la ALU de un bit, que incluye el circuito sumador completo y demás circuitos adicionales para las señales de control. El circuito completo se compone de ocho circuitos como éste en paralelo. La salida de arrastre procedente de la octava columna sirve de indicador de arrastre al registro de estado del procesador (PSR).

Así concluimos nuestra serie de lecciones sobre lógica. Recuerde que la iniciamos con conceptos tan abstractos como el álgebra de Boole y los diagramas de Venn, para aplicarlos a circuitos lógicos muy sencillos, examinando sus resultados. Pero los últimos capítulos se alejaban de los conceptos abstractos y se referían ya a temas relacionados con circuitos mucho más complicados a un nivel cercano al que realmente existe en un ordenador.

Para el final hemos dejado la explicación de cómo es posible combinar varios circuitos de tal modo que proporcionen al microprocesador los medios para realizar sus típicas operaciones aritméticas y lógicas. Operaciones que sólo exigen, para su perfecta ejecución, los patrones adecuados de unos y ceros en las líneas de instrucciones de la ALU. Estos patrones no son más que las instrucciones del lenguaje máquina en su auténtica forma binaria. Hemos, pues, estudiado la lógica del sistema físico de los ordenadores hasta el mismo borde donde empieza el terreno del software.



Diagramas: Liz Dixon

El método adecuado

En este último capítulo examinaremos la manera de utilizar, en micros basados en cassette, las técnicas descritas anteriormente

Cada micro utiliza su propia técnica de tratamiento de archivos, y, por lo tanto, a menudo es necesario adaptar un programa estándar para ejecutarlo en una máquina determinada. Por este motivo es importante saber qué relación hay entre las facilidades e instrucciones que ofrece su sistema con los métodos generales que hemos analizado. A modo de ejemplo, examinemos el almacenamiento de archivos de datos en un micro estándar basado en cassette. El primer punto a considerar es que los sistemas de cassette no pueden manipular archivos de acceso directo (véase p. 724), y hay que acceder a los datos por el orden en que están almacenados, es decir, secuencialmente.

Dado que el tratamiento de un archivo secuencial implica leer la información del archivo, trabajar sobre esta información y luego escribir los datos modificados en un segundo archivo, es obvio que debe haber dos archivos en uso ("abiertos") al mismo tiempo. Una grabadora de cassette es incapaz de moverse directamente y con precisión en dos porciones de cinta a la vez, de modo que este sistema de "dos archivos" no es apto para la mayoría de los micros basados en cassette. Las excepciones a esta regla son esos pocos micros, como el Newbrain y el Commodore PET, con dos puertas para cassette: una para leer, otra para escribir.

La mayoría de las máquinas personales están, por consiguiente, limitadas a un archivo secuencial por vez. En la práctica ello supone algunas limitaciones importantes. El archivo debe ser leído sobre la memoria antes de utilizarlo y luego, si se producen cambios, debe ser grabado de nuevo en cassette. Esto se podría hacer a intervalos durante la ejecución del programa, o una vez al finalizar su ejecución. Los archivos de datos deben ser suficientemente pequeños para residir en RAM en el espacio libre dejado por el programa que los trata. La mayoría de los micros personales están restringidos, por tanto, a archivos de datos de pequeño tamaño.

Se han desarrollado tres métodos principales para almacenar información en cassette. El sistema más simple no utiliza archivos de datos separados del programa, sino que todas las variables en curso se almacena junto con el programa cada vez que se utiliza la instrucción SAVE. Este procedimiento se emplea en el ZX81 y también está disponible para el Sinclair Spectrum. Cuando se requiere un nuevo archivo de datos se utiliza una copia "fresca" del programa, que después se guarda (SAVE) junto con sus datos particulares. Cuando esta versión se carga (LOAD) posteriormente, las variables quedan automáticamente con los valores que tenían en el momento del SAVE. La virtud de este método es su simplicidad: todo lo que el usuario tiene que hacer es asegurarse de que todo el programa se guarde (SAVE) y se cargue (LOAD) correctamente.

Un sistema ligeramente más sofisticado exige un BASIC que sea capaz de almacenar y volver a leer matrices específicas. En el Oric Atmos, por ejemplo, la orden STORE A$,NOMBRE" grabará la matriz A$ en cinta, y RECALL A$"NOMBRE" la volverá a leer sobre la memoria. Toda la matriz (A$(1), A$(2), etc.) se guarda (SAVE), a pesar de que las instrucciones STORE y RECALL no especifican las dimensiones de la matriz, que se determina cuando la matriz se DIMensiona al comienzo del programa.

Este sistema tiene el problema de que es necesario llevar la cuenta del número de entradas en la matriz que se utiliza en el programa. Una solución consiste en almacenar el contador de elementos usados antes de guardarla (SAVE). La mayoría de las máquinas admiten un subíndice cero, de modo que se puede emplear para el contador de registros el elemento A$(0,0). El contador de registros será una variable numérica (en nuestro programa utilizamos R), pero si se está utilizando una matriz de strings, ésta se debe convertir en un string. Esto es bastante fácil de hacer con una línea como: A$(0,0) = STR$(R). Después de que se haya vuelto a cargar la matriz en el ordenador, R se restablece mediante: R = VAL(A$(0,0).

Aunque muchos micros no admiten procedimientos sofisticados para tratamiento de archivos, éstos se pueden simular, como muestran los listados de la página siguiente. Una vez el archivo instalado en la memoria, es fácil usar matrices de BASIC para tratarlo como a un archivo de acceso directo.

Supongamos que se emplea una matriz de caracteres bidimensionales para almacenar los datos; ésta se podría definir con una instrucción como DIM A$(100,3). El primer subíndice de la matriz se podría utilizar para referenciar un registro determinado, y el segundo subíndice apuntaría al campo determinado dentro de ese registro. Esto permite almacenar los datos en el formato familiar de una tabla y equivale, en cuanto a operación, a un archivo de acceso directo.

Otra facilidad útil de que disponen algunas máquinas es la instrucción APPEND. Ésta permite que uno agregue datos al final el archivo secuencial sin tener primero que leerlo y crear luego una nueva versión. Algunos ordenadores disponen de una instrucción que permite saltarse un cierto número de campos en un archivo secuencial, lo que proporciona una facilidd de acceso directo simple.

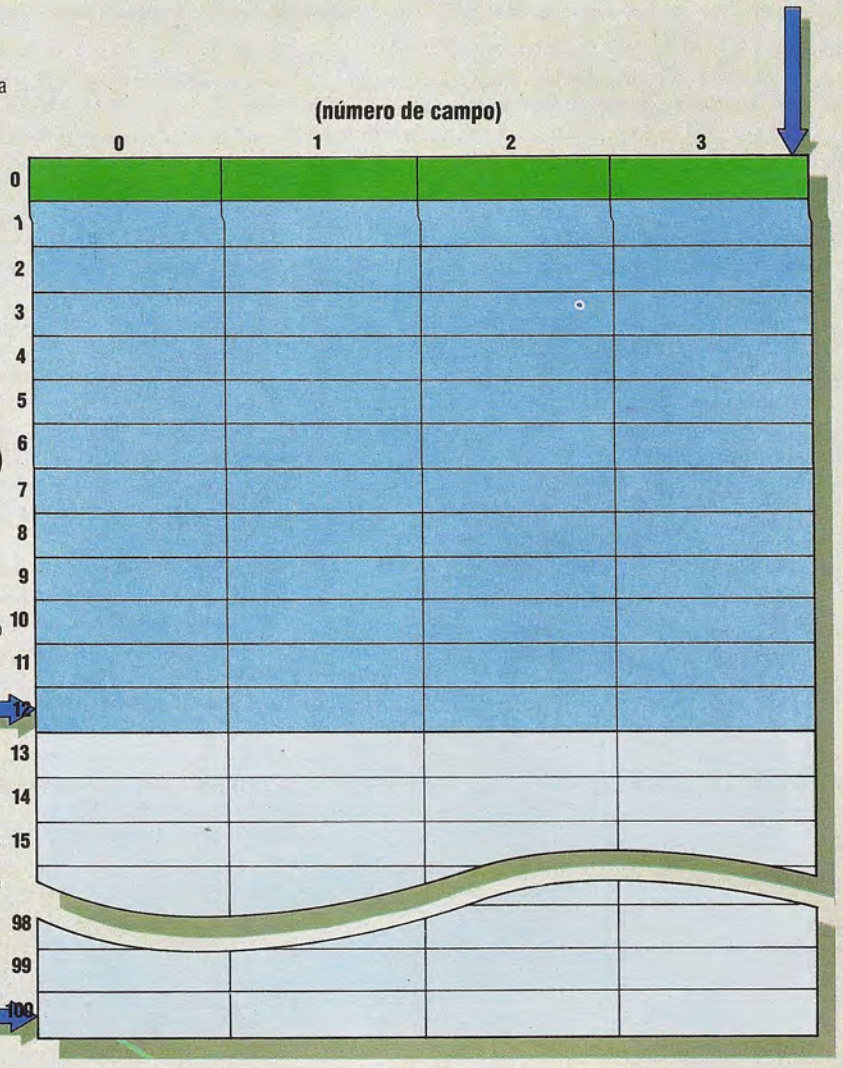
Esta serie de capítulos ha hecho hincapié en los aspectos fundamentales de un tema complejo. El tratamiento de archivos depende mucho del ordenador y será necesario adaptar a su propia máquina todas las ideas que hemos ofrecido en estos capítulos. Pero los principios básicos serán los mismos, independientemente de qué micro se emplee, y cuando escriba sus propios programas gran parte del material le resultará muy útil.

Almacenamiento de registros y campos en una matriz de BASIC

Una matriz de caracteres bidimensional se puede utilizar para emular un archivo de acceso directo. El primer subíndice se emplea para identificar un registro determinado y el segundo especifica campos dentro del registro

Número de campos
Es cómodo tener este número definido en una variable al principio del programa. Esto hace que resulte más fácil utilizar más tarde registros mayores, porque será necesario alterar una sola instrucción

Registro de cabecera
A$(0,0) se utiliza para almacenar un contador del número de registros. Cuando la matriz se guarda (SAVE) en cinta, este valor se almacena con ella



Número de registros
Una variable guarda el número de registros que están siendo usados en cada momento

Número máximo de registros
Se utilizará una variable para guardar el número máximo de registros que se pueden almacenar en la matriz. Éste es el número especificado en la sentencia DIM que crea la matriz

Eliminación de un registro de la matriz
Este fragmento del programa elimina de la matriz el registro número N utilizando el método que ya hemos detallado. Todos los registros por debajo de N se desplazan un lugar hacia arriba, machacando los datos almacenados en la posición N

```
200 FOR I=N TO R-1
210 FOR J=0 TO F
220 LET A$(I,J)=A$(I+1,J)
230 NEXT J:NEXT I
240 LET R=R-1
250 RETURN
```

Inserción de un registro en la matriz
Este otro fragmento inserta un nuevo registro en la matriz en la posición N del archivo. Todos los registros después del punto de inserción se desplazan hacia abajo para crear un vacío para el nuevo registro almacenado en N$, C$, D$ y E$

```
100 LET R=R+1
110 IF R>M THEN PRINT "MATRIZ LLENA":RETURN
120 FOR I=R TO N+1 STEP-1
130 FOR J=0 TO F
140 LET A$(I,J)=A$(I-1,J)
150 NEXT J:NEXT I
170 LET A$(N,0)=N$:LET A$(N,1)=C$
180 LET A$(N,2)=D$:LET A$(N,3)=E$
190 RETURN
```

Cómo cargar y guardar una matriz de registros

Variables guardadas con el programa
El Spectrum guarda todas sus variables junto con los programas (aunque también puede guardar matrices solas). El programa de muestra llena una matriz con datos y la instrucción SAVE almacenará la matriz en cinta junto con el programa. Cuando éste se vuelve a cargar, empieza automáticamente por la línea 700, porque la instrucción SAVE lo remite a aquella línea. La instrucción RUN no se debe utilizar porque borra todas las variables

Spectrum

```
100 REM ***DEMO SPECTRUM***
200 DIM A$(100,20)
300 FOR K=1 TO 100
400 LET A$(K)="Numreg"+STR$(K)
500 NEXT K
600 STOP
700 PRINT"LA MATRIZ CONTIENE..."
800 FOR K=1 TO 300
900 PRINT A$(K).
1000 NEXT K
110 STOP
SAVE "PROGDEMO" LINE 700
LOAD "PROGDEMO"
```

Cómo guardar matrices con nombre
El Oric Atmos dispone de las instrucciones STORE y RECALL para guardar y cargar en cinta matrices concretas. Gracias a estas instrucciones es fácil almacenar un archivo contenido en una matriz

Oric Atmos

```
100 REM Guardar matriz en cinta
110 A$(0,0)=STR$(R)
120 PRINT "Por favor posicione la cinta, pulse
    PLAY y RECORD y después RETURN"
130 A$=KEY$:IF A$="" THEN 130
140 STORE A$,"ARCHIVO",S
150 PRINT "CONCLUIDO"
160 RETURN

200 REM Cargar matriz desde cinta
210 PRINT "Por favor posicione la cinta y pulse
    PLAY, luego RETURN"
220 A$=KEY$:IF A$="" THEN 220
230 RECALL A$,"ARCHIVO",S
240 R=VAL(A$(0,0))
250 PRINT "CONCLUIDO"
260 RETURN
```

Utilización de archivos seriales
El BBC Micro es uno de los diversos ordenadores personales que admiten auténticos archivos secuenciales en cassette. Estas dos subrutinas almacenan y vuelven a cargar la matriz mediante la creación de un archivo secuencial. El primer campo es el contador de registros

BBC Micro

```
100 REM Guardar matriz en cinta
105 PRINT "Por favor posicione la cinta"
110 X=OPENOUT ("ARCHIVO")
120 PRINT £X,R
130 FOR I=1 TO R
140 FOR J=1 TO F
150 PRINT £X,A$(I,J)
160 NEXT J:NEXT I
170 CLOSE £X
180 PRINT "CONCLUIDO"
190 RETURN

200 REM Cargar matriz desde cinta
205 PRINT "Por favor posicione la cinta y pulse
    PLAY después RETURN"
210 IF INKEY(0)=-1 THEN 210
215 X=OPENIN ("ARCHIVO")
220 INPUT £X,R
230 FOR I=1 TO R
240 FOR J=1 TO F
250 INPUT £X,A$(I,J)
260 NEXT J:NEXT I
270 CLOSE £X
280 PRINT "CONCLUIDO"
290 RETURN
```

La pulga, o ¿cómo salir del abismo?

"Bugaboo" (La pulga) es un insólito juego para el Spectrum y el Commodore 64: el jugador es una pulga que queda atrapada en un gran agujero

La impresionante secuencia de títulos del juego *Bugaboo*, de Quicksilva, le informa que se está aproximando al planeta Cebella-7, en el que existen indicios de vida. Después de aterrizar con toda seguridad, usted va brincando por la fantástica flora de la superficie del planeta hasta perder pie y despeñarse en un profundo agujero, que se abre a una enorme caverna. El objetivo del juego es escapar del abismo.

Unos salientes que sobresalen de rocas de llamativos colores permiten apoyar los pies en un paisaje que recuerda las pinturas del Bosco: setas y flores multicolores proliferan por doquier y las arañas trepan por las paredes de la caverna. Usted ha de intentar escapar saltando de saliente en saliente, evitando a la vez a un dragón devorador de pulgas. Sólo en la versión para el Commodore, las plantas atrapamoscas proporcionan un nuevo aliciente: otro obstáculo floral.

A saltar

La característica más sobresaliente del *Bugaboo* son sus detallados gráficos. Estos representan un profundo agujero que tiene muchos salientes entre los cuales el jugador (que personifica a una pulga) tiene que ir saltando para poder ascender hacia la libertad. Los gráficos del Commodore le llevan una ligera ventaja a los de la versión para el Spectrum

Bugaboo (La pulga) en el Spectrum

Bugaboo (La pulga) en el Spectrum

Booga-boo (La pulga) en el Commodore

En ambas versiones, la pantalla actúa como una ventana de la zona de juego, y al jugador se le van presentando nuevas secciones de la caverna a medida que se va desplazando y va trepando hacia arriba. La caverna tiene un ancho aproximado de tres pantallas y una altura de alrededor de cinco pantallas, de modo que descubrir todos los obstáculos lleva su tiempo. Se puede tomar una cualquiera de varias frutas diferentes y los salientes están situados de forma muy astuta para aumentar el grado de dificultad.

La escenografía es original, pero el juego pierde mucho debido a la codificación del programa.

El jugador sólo tiene una vida, y la secuencia de instrucciones de la introducción se repite cada vez que uno pierde la vida, lo que sucede con muchísima frecuencia, ya que evitar al dragón resulta casi imposible.

A medida que avanza el juego, los salientes son cada vez más difíciles de alcanzar. Pero en los niveles superiores no se introducen obstáculos, lo que es lamentable, ya que el programa produciría mucha más adicción si en las sucesivas etapas fueran apareciendo otros depredadores.

El almacenamiento de las múltiples pantallas utiliza muchísima memoria, dejando poca para el uso de reglas complicadas y acción, pero hay otros muchos programas que consiguen más: *Jet Set Willy* es un buen ejemplo. La versión para el Commodore 64 realiza gráficos más complejos que la versión para el Spectrum, pero, por el contrario, no consigue obtener el máximo partido de su memoria adicional.

La versión para el Spectrum se controla mediante el teclado, haciendo que las teclas "1" y "0" controlen los saltos a izquierda y derecha. La fuerza de estos saltos está determianda por el tiempo durante el cual se mantenga pulsada la tecla en cuestión: cuánto más tiempo se mantenga pulsada la tecla, más arriba se saltará.

La versión para el Commodore 64 es para utilizar sólo con una palanca de mando, con el pulsador de disparo controlando la exploración: un arreglo más satisfactorio en caso de que se posea una palanca de mano.

A pesar de las críticas hechas al *Bugaboo* acerca de su calidad de adicción a largo plazo, hay que decir que el juego es muy recomendable. Los controles son sumamente sencillos, permitiendo que el usuario empiece a jugar de inmediato, sin tener que remitirse una y otra vez a la pantalla de instrucciones. Y los imaginativos gráficos son, sencillamente, magníficos.

Artimañas aritméticas

Esta vez estudiaremos cómo resta y multiplica un microprocesador. Para ello debemos presentar dos operaciones lógicas: desplazamiento y rotación

Decíamos en el capítulo anterior que tanto el 6502 como el Z80 disponen de la instrucción SBC (*SuBtract with Carry*: restar con arrastre), pero el modo de ejecutarla es distinto. En el 6502 el flag de arrastre sirve para "quitar uno", de la misma forma que servía para "añadir uno" en la suma. En el assembly del Z80, sin embargo, el funcionamiento de SBC se corresponde perfectamente con el de ADC: el flag se pone a 1 o queda a cero, según sea el resultado de la operación.

Así, por ejemplo, si usamos ADC para obtener la suma $E4+$5F (poniendo antes, como de costumbre, el flag a cero) en el acumulador obtendremos $43 y el flag de arrastre estará a 1, lo cual nos está indicando que el resultado completo es $0143. Ha habido un desbordamiento (*overflow*) que activó al flag ya que el resultado no cabía en el acumulador, capaz tan sólo de un byte.

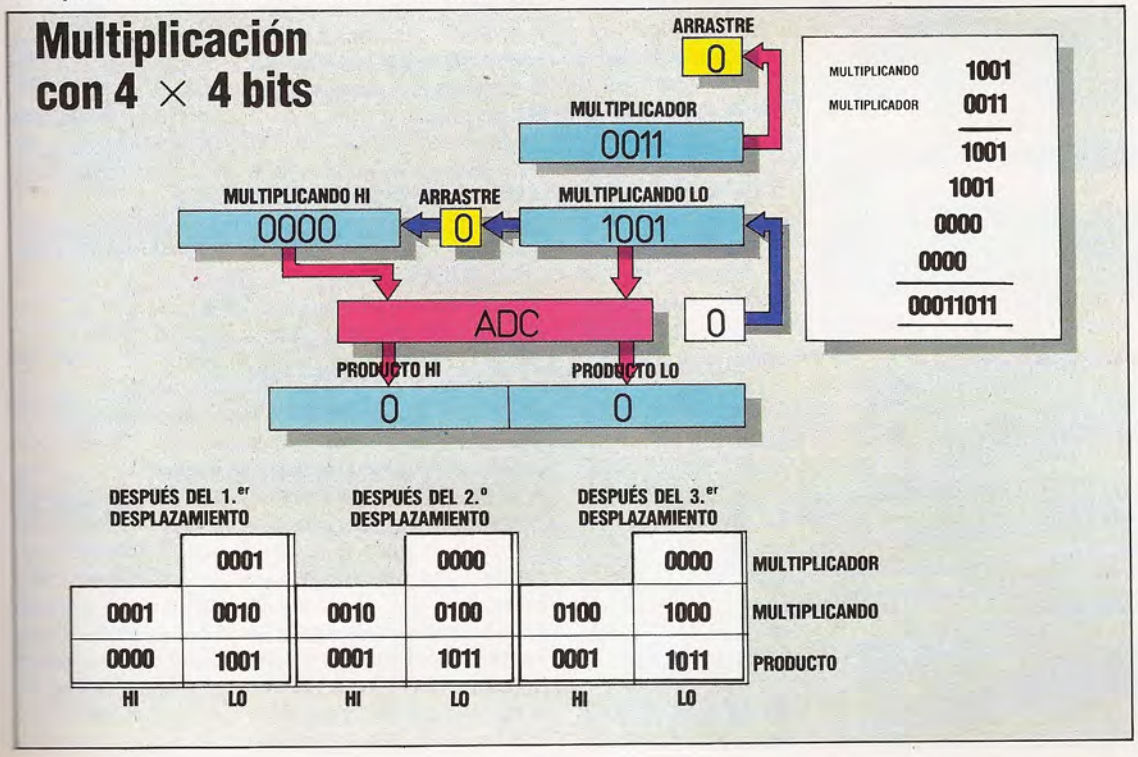
Supongamos que deseamos realizar con el Z80 la resta $5F−$E4. En el acumulador obtendremos $7B y el flag se activará. ¿Qué resultado es éste? En decimal equivale a escribir 95 − 228 + 123 (valor del hexa 7B), pero el significado que para el Z80 tiene el flag activado después de la resta nos permite dar con el resultado verdadero. El flag en 1 significa que debemos tomar el número contenido en el acumulador como *el complemento a 2* del resultado verdadero, ya que nos está diciendo que se ha producido un resultado negativo. Para hallar la solución hemos de "desandar" el camino que en el capítulo anterior describimos hasta dar con el complemento a 2. Veámoslo:

$7B en binario	= 01111011
quitamos 1	− 1
complemento a 1	01111010
cambio de unos y ceros	
y obtenemos el compl. a 2	10000101 = $85

Ya tenemos el resultado auténtico negativo: el número −$85 (decimal, −133).

Naturalmente, las cosas son distintas cuando la resta se realiza a dos bytes. Habremos de distinguir entre los bytes inferiores (los llamados bytes *lo* y los bytes superiores (bytes *hi*). Por ejemplo:

HI	LO		
$37	5F	=	14175 en decimal
− $21	E4	=	−8676 en decimal
$15	7B	=	5499 en decimal



Desplazamientos
Este ejemplo muestra, para mayor claridad, una multiplicación con sólo 4 bits para cada número. El algoritmo no cambia si hay más bits. Observe cómo se forma el producto por medio de la suma de ceros o del multiplicando desplazado, según el bit del multiplicador sea cero o uno. Los bits del multiplicador se desplazan hacia la derecha pasando por el flag de arrastre, mientras que los bits del multiplicando se desplazan hacia la izquierda pasando del byte *lo* al byte *hi* a través del flag de arrastre

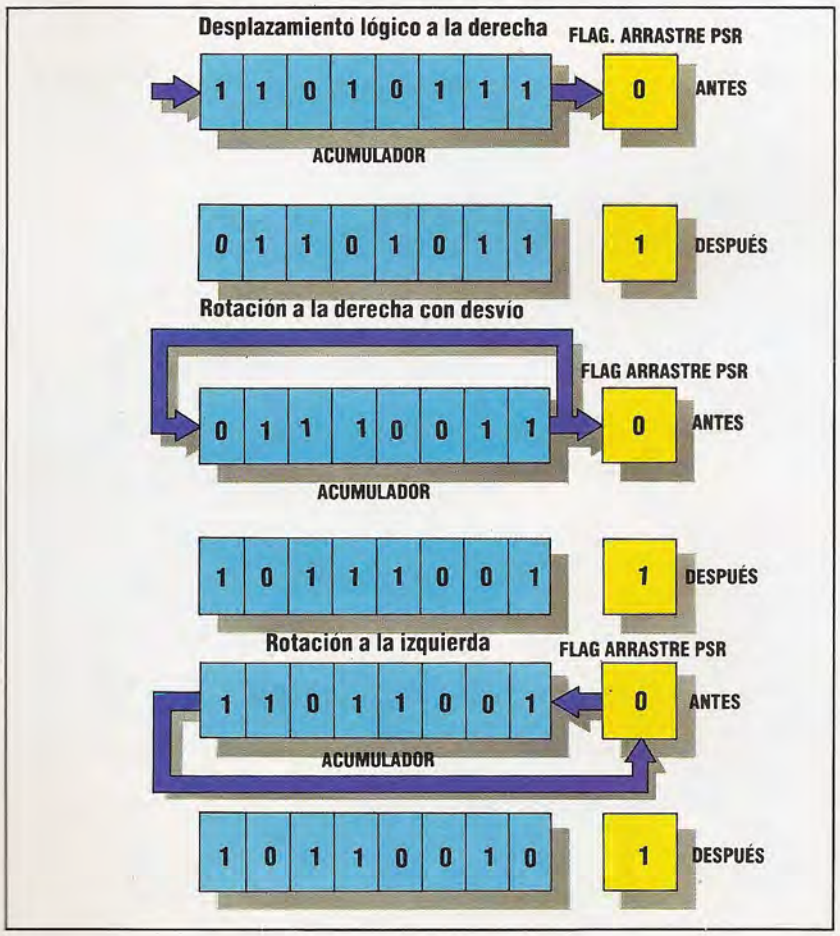
En restas de este tipo, la resta en los bytes *lo* sabemos que da $7B en el acumulador y la activación del flag de arrastre. Esto hace que en los bytes *hi* el sustraendo $21 se convierta en $22, que restado con el minuendo $37 da $15. Vemos que el resultado $375F−$21E4=$157B es correcto, según comprobamos por su versión decimal.

Resumiendo, la operación a dos bytes en un Z80 sigue un procedimiento muy sencillo, que ofrecemos a continuación:

1) Se pone a 0 el flag de arrastre.
2) Se restan los bytes *lo* con arrasstre.
3) Se restan los bytes *hi* con arrastre.

El 6502 se diferencia sobre todo en el paso 1). Lo que se hace previo a la resta es poner el flag en 1, pues en 0 significa que se debe quitar una unidad en los bytes *hi*, según resulta de la resta de los bytes *lo*. Si éstos no necesitan tomar ninguna unidad de aquéllos, entonces la resta transcurre sin más dificultades, y el flag permanece en 1 preparado para la resta de los bytes *hi* que deberá realizarse con idéntica normalidad. Pero si en la resta de los bytes *lo* se tuviera necesidad de una unidad adicional, el flag de arrastre actuaría como un "noveno bit" del acumulador. Lo cual permite que se obtenga el resultado correcto y pone a 0 el flag. En este estado el efecto sobre la resta de los bytes *hi* es idéntico al que realiza el flag activado en el Z80: el número minuendo sufre una disminución de una unidad antes de ser restado. Ambos métodos recuerdan el tradicional "no cabe, me llevo diez, resto y quito una" de nuestras restas infantiles. Veamos la versión para el 6502 más detenidamente.

Instrucciones
Las instrucciones de desplazamiento y rotación sirven ante todo para examinar el contenido de un registro bit a bit. A cada desplazamiento, el bit del extremo (izquierda o derecha) se traslada al flag de arrastre del registro indicador de estado (PSR), pudiéndose emplear su estado para ejecutar una bifurcación del flujo del programa. Las instrucciones de rotación se emplean cuando queremos conservar el contenido del registro, pero las instrucciones de desplazamiento lógico van colocando ceros por un lado a medida que desplazan los bits hacia afuera por el otro. Por tanto, un desplazamiento a la izquierda multiplica por dos el contenido del registro, y un desplazamiento a la derecha lo divide por dos



Si mantuviéramos el flag de arrastre en 0 y realizáramos la resta $5F−$E4, en el acumulador tendríamos, para nuestra sorpresa, $7A y el flag no se inmutaría. Pero ya hemos visto que el "supuesto" resultado verdadero es $7B más una señal del flag de que el número es negativo (o sea, que es el complemento a 2 del resultado definitivo). Pero $7A es el complemento a 1, si bien se ve, de dicho resultado definitivo. De lo que se deduce que el estado del flag indica el complemento a 2 si está activado, y el complemento a 1 si está a cero.

Si, por el contrario, activamos antes de nada el flag y después hacemos la resta, el acumulador contendrá $7B y el flag se pondrá en cero. Para el caso de que la resta sea de dos bytes y no se pare aquí, el flag en cero se encarga de quitar una unidad al byte *hi* minuendo: la unidad que se tomó de más en la resta de los bytes *lo*.

Multiplicación

Examinemos una multiplicación en decimal:

174	multiplicando
× 209	multiplicador
1566	1.er producto parcial
000	2.º producto parcial
+ 348	3.er producto parcial
36366	producto total

No hace falta, para entender bien este método, más que seguirlo en su modo de disponer los productos parciales y saberse la tabla de multiplicar. La multiplicación se reduce a una suma si se ha cumplido su norma más fundamental: la de ir colocando cada producto parcial desplazado un lugar hacia la izquierda respecto de la situación del anterior producto parcial (incluimos, para mayor claridad, el producto con sólo ceros).

Esta alternancia de productos parciales desplazados y tablas de multiplicar es lo que nos hacía difícil la multiplicación en nuestros años de escuela. Pero en binario la tabla de multiplicar es sencillísima: $1 \times 1 = 1$ y $1 \times 0 = 0$. Sólo nos queda respetar la colocación

1101	(decimal, 13)
× 1001	(decimal, 9)
1101	1.er prod. parcial
0000	2.º prod. parcial
0000	3.er prod. parcial
1101	4.º prod. parcial
1110101	(decimal, 117 = 13 ×9)

Se observan claramente en este ejemplo los desplazamientos de los productos parciales, así como la extremada sencillez de la multiplicación en binario. Note que un producto parcial cualquiera o es todo ceros o es idéntico al multiplicando. Lo que inmediatamente nos recuerda el test al que estamos acostumbrados como prueba del lenguaje assembly. Para realizar una multiplicación binaria, debemos examinar cada uno de los bits del multiplicador por orden, e ir sumando ceros (si el bit es cero) o el multiplicando desplazado (si el bit es uno) al total.

Vamos a ver cómo se pueden examinar los bits, uno a uno, del multiplicador y cómo se desplaza el multiplicando.

La comprobación del estado de un bit en particular dentro de un byte se puede realizar, en ambos microprocesadores, el Z80 y el 6502, mediante la instrucción BIT. En el Z80 esta instrucción tiene como operandos una dirección y el número del bit a investigar, poniendo el flag de cero a 0 s el bit es uno, y a 1 si el bit es cero. En el 6502 el operando es sólo una dirección. Con el contenido de esta dirección realiza la operación lógica AND sirviéndose del acumulador, y según el resultado sea verdadero o falso del flag de cero permanece en cero o cambia a uno.

Estas instrucciones permitirían una acertada programación, pero ningún método nos conviene por el momento. Mejor sería si el bit de que se trata nos sirviera como flag de arrastre o flag de cero, de tal modo que el flujo del programa se desviara automáticamente según el estado de cada uno de los bits. Esto naturalmente es posible con las instrucciones de que disponen ambos procesadores, y en particular con las "instrucciones de desplazamiento" (shift instructions). Según ya indica su nombre, permitirán también resolver el problema de cómo desplazar el multiplicando.

Varias son las instrucciones de "desplazamiento" (*shift*) o de "rotación (*rotate*) en ambos procesadores. Tienen por efecto general el de desplazar cada bit contenido en un registro un lugar hacia la izquierda o hacia la derecha. Las diferencias estriban en el modo de tratar los bits que están en los extremos del registro: un bit debe desplazarse fuera del registro por un extremo mientras que por el opuesto hay que emplazar otro bit. En el caso de que se desplace fuera el bit 7 y este mismo aparezca inmediatamente como bit 0, tendríamos una "rotación a la izquierda" (*rotate left*). Lo contrario, el bit 0 va desplazando al bit 7, será una "rotación a la derecha" (*rotate right*). Es claro que tras ocho rotaciones del mismo tipo volvemos a obtener el registro de partida.

Si no se utiliza la rotación debemos encontrar un destino para el bit desplazado fuera del registro, y una fuente de donde sacar el bit de relleno. La mayoría de las veces, ambas cosas las suministran los diversos flags, condición del registro indicador de estado (PSR), y en concreto el flag de arrastre. Para construir una subrutina que multiplique números de dos bytes es necesario desplazar el multiplicando a la izquierda y el multiplicador a la derecha. Los bits del multiplicando irán siendo desplazados hacia el byte *hi* de éste, mientras van apariendo ceros en los bits que se vacían. Por su parte, los bits del multiplicador irán pasando por un flag del PSR para su comprobación, pero su destino final así como el estado de los bits que van rellenando el multiplicador según se va vaciando no tiene mayor interés, a menos que no necesitemos conservar el multiplicador en su forma primitiva. Lo que sí es relevante para lo que nosotros queremos es saber si el bit que acaba de desplazarse fuera del multiplicador es un uno o un cero.

Indicando que el multiplicador se encuentra en la dirección MPR, el multiplicando en MPDLO y el producto en PRODLO y PRODHI, podemos describir estas subrutinas del modo como expresamos a continuación:

MULTIPLICACIÓN CON OCHO BITS

6502			Z80		
	ORG	$C100		ORG	$D000
START	LDA	#$00	START	LD	BC,(MPR)
	STA	PRODLO		LD	B,$08
	STA	PRODHI		LD	DE,(MPDLO)
	STA	MPDHI		LD	D,$00
	LDX	#8		LD	HL,$00
	CLC		LOOP0	SRL	C
LOOP0	ROR	MPR		JR	NC,CONT0
	BCC	CONT0		CALL	ADDIT
	JSR	ADDIT	CONT0	SLA	E
CONT0	ASL	MPDLO	ENDLP0	DJNZ	LOOP0
	ROL	MPDHI		LD	PRODLO
	DEX			RTS	
ENDLP0	BNE	LOOP0	MPR	DB	$E2
	RTS		MPDLO	DB	$7A
MPR	DB	$E2	MPDHI	DB	$00
MPDLO	DB	$7A	PRODLO	DW	$0000
MPDHI	DB	$00	ADDIT	ADD	HL,DE
PRODLO	DB	$00		RET	
PRODHI	DB	$00			
ADDIT	CLC				
	LDA	PRODLO			
	ADC	MPDLO			
	STA	PRODLO			
	LDA	PRODHI			
	ADC	MPDHI			
	STA	PRODHI			
	RTS				

Por el ejemplo podemos ver que la programación del Z80 es mucho más fácil gracias a sus registros de 16 bits y las instrucciones asociadas. Compare la subrutina ADDIT en ambos programas. La versión del 6502 emplea ROR (*rotate right*) para hacer rotar el multiplicador hacia la derecha pasando por el flag de arrastre, y se sirve de ASL (*shift left*) y ROL (*rotate left*) para desplazar el multiplicando hacia la izquierda dejando MPDLO y pasando a MPDHI a través del flag de arrastre. El bucle se controla con el registro X actuando de contador.

La versión del Z80 emplea SRL para desplazar el multiplicador hacia la derecha pasando por el flag de arrastre, y SLA y RL para desplazar el multiplicando contenido en DE hacia la izquierda pasando por el flag de arrastre. El control del bucle está a cargo del registro B que hace de contador. La instrucción ADD suma a dos bytes y no es afectada por el flag de arrastre (al contrario de ADC).

El próximo capítulo tratará de la división y de las diversas maneras de controlar la pantalla en las visualizaciones. Quedará así cubierta la teoría y podremos dedicarnos a realizar ejercicios con el 6502 y el Z80 en las siguientes lecciones.

Ejercicio 15

1) Escriba una subrutina para multiplicar con un multiplicando de 16 bits y un multiplicador de 8 bits elegidos por usted.

2) La multiplicación no es sin la suma reiterada. Escriba una subrutina de multiplicación de 8 bits por 8 bits, sin recurrir a las instrucciones de desplazamiento y rotación.

Elegancia italiana

Olivetti, la prestigiosa multinacional europea, actualmente está considerada como una de las principales empresas especializadas en automatización de oficinas

Ángulo de diseño
El ordenador de oficina Olivetti M20 es una máquina de 16 bits que salió al mercado en 1981. Incorpora un microprocesador Zilog Z8000 y utiliza el sistema operativo Olivetti PCOS. Recientemente Olivetti ha lanzado una versión de esta máquina que es compatible con el IBM-PC

Camillo Olivetti fundó su empresa en 1908. Con una plantilla de 20 empleados, instaló una tienda de máquinas en Ivrea, entonces una pequeña ciudad rural del norte de Italia, y comenzó a producir la primera máquina de escribir de la empresa, la M1. En aquellos tiempos la economía italiana, todavía era básicamente agrícola y carecía de la industria pesada que había favorecido la expansión en Alemania, Gran Bretaña y Estados Unidos. A pesar de esto, la producción de Olivetti fue aumentando de forma constante, pasando de lanzar al mercado cuatro máquinas diarias en 1914 a producir 50 por día en el año 1929.

En los años treinta Adriano Olivetti, hijo de Camillo, comenzó a reorganizar la empresa, introduciendo alumnos de la propia escuela nocturna de Olivetti, que se había fundado en 1924. También se realizaron mejoras sociales y se proporcionaría el alojamiento por parte de la empresa, una política "de la cuna a la tumba" que recuerda a la que luego utilizarían con tanta eficacia los japoneses. Mientras en el resto del mundo la industria luchaba inmersa en la depresión económica de entreguerras, Olivetti seguía creciendo, y hacia 1933 la empresa había vendido 15 millones de productos para oficina. En 1937 se lanzó la primera teleimpresora Olivetti, a la que siguió, en 1940, la primera calculadora de la empresa.

La segunda guerra mundial supuso una interrupción temporal de la expansión de Olivetti, pero en los años de la posguerra la empresa se concentró en el desarrollo de nuevos mercados. El éxito de la empresa se cimentaba en la elegancia de sus diseños y en la calidad de los productos, e incluso los ejecutivos de IBM se vieron obligados a admitir que los productos Olivetti "se acoplaban entre sí como un hermoso puzzle de imágenes".

Exterior elegante
Olivetti siempre se ha destacado por el avanzado diseño de sus productos. Pero su preocupación por el factor elegancia también alcanza a la arquitectura de los edificios de la empresa. Esta oficina se construyó en 1959 y muchas de sus características fueron posteriormente adoptadas por otros arquitectos

En la década de los cincuenta y de los sesenta, Olivetti se concentró en el desarrollo de ordenadores de oficina. Este proceso empezó con la introducción de una máquina de calcular numérica, en 1955, y unos años después, producía el primer ordenador central, el Elea.

La empresa siguió diversificándose, abandonando la maquinaria mecánica de oficina y volcándose hacia los equipos basados en la electrónica, que Olivetti identificó con la principal tendencia de la automatización de oficinas. Se introdujo una nueva gama de microordenadores, a la cual no tardaría en agregarse la de los terminales bancarios y equipos de comunicaciones.

En la actualidad Olivetti fabrica una amplia gama de máquinas de oficina electrónicas, y la empresa invierte grandes sumas de dinero en desarrollar software de apoyo para sus máquinas. En 1982 Olivetti fue el segundo mayor fabricante de ordenadores de Europa (sólo superado por IBM), con el ordenador portátil M10 y la máquina de oficina M20. Tanto una como otra obtuvieron una excelente acogida en el mercado.

El ordenador de mano M10 pesa alrededor de 1,7 kg y viene equipado con una pantalla de 8x40 caracteres. La máquina funciona a pilas y tiene 8 Kbytes de RAM interna que se pueden ampliar a 64 Kbytes. La máquina de oficina M20, de 16 bits, trabaja sobre un microprocesador Z8001, no muy apreciado por otros fabricantes de máquinas de 16 bits. Asimismo, contiene un procesador 8086 que permite alguna compatibilidad con CP/M-86 y MS-DOS.

Olivetti también está planeando una nueva máquina compatible con el IBM-PC que, según la empresa, valdrá menos que su rival de IBM. Llamado M24, posee un procesador 8086-2 y tiene la opción de una tarjeta Z8001 para que sea compatible con el Olivetti M20. Esta compatibilidad ha tenido como consecuencia que Olivetti se viera en la obligación de abandonar su propio sistema operativo PCOS.

Recientemente Olivetti ha firmado un acuerdo con AT & T (la empresa de telecomunicaciones más grande del mundo) para colaborar en un proyecto para desarrolar el sistema operativo Unix. Y no cabe ninguna duda de que en el futuro la red de ventas internacional y el activo departamento de investigación y desarrollo de Olivetti mantendrán el prestigio de que goza la empresa a causa de la óptima factura, de la descollante calidad de todos sus productos para oficina.

Ajedrez por ordenador

Analizamos aquí algunas de las ideas en que se basa el desarrollo de programas de este apasionante y milenario juego

Campeón de ajedrez
David Levy es un maestro internacional de ajedrez que abandonó la competición contra seres humanos en 1978. En 1968 aportó una gran suma de dinero, diciendo que ningún programa de ajedrez por ordenador sería capaz de ganarle a él durante los diez años siguientes. Desde entonces, el período cubierto por la apuesta se ha ampliado, pero él continúa invicto. Levy, una autoridad de primera línea en ajedrez por ordenador, dirige Intelligent Software, una empresa que proporciona las estructuras de programación exclusivas sobre las que se basan muchos ordenadores de ajedrez y paquetes de ajedrez por ordenador. Levy piensa que actualmente los micros están empezando a acercarse al rendimiento de los grandes ordenadores en cuanto a jugar al ajedrez, y calcula que dentro de cinco a ocho años un microordenador podrá derrotar a Belle (una máquina para jugar al ajedrez exclusivamente) y a Cray Blitz (un ordenador central que en 1983 ganó el Campeonato Mundial de Ajedrez por Ordenador), posiblemente mediante la utilización de microprocesadores paralelos

Pocos juegos han captado jamás la imaginación tanto como el ajedrez: este juego lo han practicado millones de personas en todo el mundo durante miles de años, y en la actualidad se juega con reglas que prácticamente no han sufrido ninguna modificación desde el siglo XVII. Hay quienes dedican toda su vida al estudio y al dominio de este juego de estrategia, encontrando satisfacción en su exigencia de rigor y agilidad intelectual. El juego ha generado una gama de variantes que intentan introducir mayores niveles de complejidad: por ejemplo, el ajedrez tridimensional implica varios tableros suspendidos en el espacio y exige muchísima más concentración. Otra variante es el ajedrez para tres personas, que se juega en un tablero en forma de Y. En las diagonales donde se intersectan las tres "alas", al movimiento de las piezas se le aplican reglas especiales. La teoría sobre la que se apoya esta versión es que dos de los jugadores se unan contra el tercero y luego luchen entre sí por la victoria. Pero ninguna de estas variantes ha logrado desplazar a la confrontación esencial entre dos personas, una de ellas con las fichas blancas y la otra con las negras, que se enfrentan sobre el tablero dividido en 64 casillas.

Uno de los motivos que inciden en esto es el número casi infinito de variantes dentro del propio juego. En 1949, el matemático Claude Shannon escribió un ensayo titulado *Programming a computer for playing chess* (Programación de un ordenador para jugar al ajedrez), en el que calculaba que hay 10^{120} partidas posibles de 40 movimientos. Esto significa que una persona que juegue al ajedrez las 24 horas del día, siete días a la semana, a razón de una hora con cada partida (que no es mucho tiempo para 40 movimientos) ¡tardaría un poco más de 10^{17} años en efectuar todas las partidas posibles! Por supuesto, en la actualidad el ajedrez se ha analizado tan profundamente que esta vasta gama de posibilidades en la práctica disminuye, según un factor que depende de la experiencia que tenga el jugador.

Dada esta complejidad, no es sorprendente que la programación de ordenadores para jugar al ajedrez haya consumido tanto tiempo y tanto esfuerzo. Los programas de ajedrez se han ejecutado en grandes ordenadores durante muchos años, y en la actualidad existen numerosas versiones para microordenadores personales. El desarrollo de programas de ajedrez de gran calidad para microordenadores está relacionado con las innovaciones en materia de hardware; los elementos problemáticos siempre han sido la falta de memoria suficiente y la relativamente baja velocidad de proceso, pero los avances que se han ido produciendo durante estos últimos años en la tecnología han significado que la calidad de dichos programas depende ahora del software.

Dado que los ordenadores son, esencialmente, calculadoras de gran velocidad, el ajedrez por orde-

Estrategias de software

En un esfuerzo por evaluar algunos de los programas de ajedrez más populares para micros personales, esta obra organizó un minitorneo entre estos productos: Sargon III, participando en un Apple IIe (disco de Hayden Software, programación de Dan y Kathe Spracklen); Cyrus IS Chess, en un Spectrum de 48 K (cassette de Sinclair Software, programación de Intelligent Software); Colossus 2.0, en un Commodore 64 (disco de CDS MicroSystems, programación de Martin Bryant); y Grand Master 64, también para el Commodore 64 (cassette de Audiogenic, programación de Kingsoft).

Aunque estos programas ya se habían enfrentado entre sí en torneos internacionales de ajedrez por microordenador, nosotros deseábamos una evaluación informal basada en características, manejabilidad y competencia. El minitorneo consistió en un mínimo de dos partidas para cada programa, la primera en el nivel de juego más simple y la segunda en el nivel de competición. No hubo ningún intento por designar un ganador general.

Uno de los problemas que surge cuando juegan un ordenador de ajedrez contra otro es que suele ser difícil identificar el nivel de juego que les proporciona a los dos programas "inteligencia" similar. Los niveles se suelen definir por el tiempo que el ordenador se concede a sí mismo para el mejor movimiento, pero podría no existir una correlación directa entre un lapso de 10 segundos en un programa y el mismo límite de tiempo en otro. Por ejemplo, Sargon III le "roba" tiempo a su oponente y mantiene su generador de movimientos en marcha mientras está moviendo su oponente. Todos los otros programas apagan sus generadores de movimientos en este punto. No obstante, se hizo lo posible para ser imparciales, si no absolutamente precisos, al emparejar los programas.

Calidad de juego

En general, todos los programas jugaban un ajedrez sólido, aunque sin inspiración, en el nivel inferior (tardando aproximadamente 10 segundos por movimiento). Y todos hacían algunos movimientos muy extraños, inútiles en apariencia, hacia el final de la fase intermedia de la partida. Probablemente esto fuera consecuencia de una posición "tranquila" en la que los ordenadores tan sólo consumían su tiempo hasta que sucediera algo interesante. En el nivel superior, de competición (alrededor de 10 minutos por movimiento), los cuatro programas mostraron una jugada táctica inteligente y en ocasiones brillante. El cuadro de la página contigua refleja los resultados del torneo.

nador se diseña basándose en cálculos numéricos, que se utilizan para evaluar los dos elementos esenciales del juego: el material y la movilidad. El "material" de un juego se refiere al número y la fuerza de las piezas del tablero. El programa de ajedrez le asigna a cada pieza un valor numérico. Al rey se le puede dar o bien un valor infinito o uno arbitrariamente alto, como 10 000 (esto se hace porque la pérdida del rey da el juego por terminado); a la reina se le asigna un valor de nueve; la torre vale cinco; los alfiles y los caballos valen tres, y los peones, uno. Al considerar si vale la pena sacrificar una pieza con el objeto de capturar una de las piezas del contrincante, el programa compara sus valores. La mayoría de los programas de ajedrez por ordenador conceden una gran importancia a los valores relativos y raramente cambian piezas si ello produce una desventaja material, a menos que se obtenga una notable mejora en cuanto a la fuerza posicional.

La "movilidad" reviste una gran importancia en el ajedrez, ya que todas las piezas tienen poco valor si su movimiento está restringido. Por el contrario, su valor aumenta si se la puede colocar de modo que ejerza influencia sobre varios cuadros a la vez. El programa de ajedrez necesita, en consecuencia, evaluar la movilidad así como las consideraciones relativas al material. Además, el programa debe ser capaz de planear con anticipación, determinando la mejor secuencia de movimientos a partir de cualquier posición dada. Es aquí donde los programas de ajedrez pueden demostrar su superioridad, utilizando la velocidad del ordenador para examinar un gran número de posibles jugadas en un lapso muy corto.

La mayoría de los programas de ajedrez utilizan una técnica de "fuerza bruta", buscando tantos movimientos como sea posible en el tiempo permitido. El tiempo destinado para cada movimiento se determina seleccionando un "nivel" de juego al principio de cada partida; cada nivel da una medida de tiempo diferente durante la cual el ordenador debe realizar un movimiento. Estos períodos varían desde algunos segundos hasta varias horas, y cuanto más tiempo se le conceda al ordenador para pensar, mayores son las probabilidades de que encuentre la mejor línea de ataque para la posición en la que se encuentra.

En cada movimiento el ordenador determina si el rey está o no en jaque, y luego investiga si hay piezas amenazadas en cualquier bando, si se pueden ocupar posiciones clave y muchas otras consideraciones similares. Cuantos más criterios examine el ordenador, mejor será el resultado. La cuestión final consiste en descubrir si se puede forzar al rey del contrincante a una posición de jaque mate.

En los juegos entre ordenadores y seres humanos, los primeros tienen una clara ventaja en cuanto a velocidad y alcance de búsqueda; aun así, un jugador humano excelente siempre derrotará a un programa para ordenador excelente, debido a la capacidad de los humanos para ver y crear nuevas aperturas y posiciones. Los ordenadores practican un ajedrez táctico soberbio, pero, incluso entre los

Cyrus IS Chess hizo tablas con Colossus y venció a Grand Master en el nivel simple, e hizo tablas con Sargon III en el nivel de competición.

Colossus hizo tablas con Cyrus IS Chess en el nivel simple, venció a Grand Master en el nivel de competición e hizo tablas con Sargon III en el nivel de competición.

Sargon III perdió con Grand Master en el nivel simple e hizo tablas con Cyrus y Colossus en el nivel de competición.

Grand Master venció a Sargon III y perdió con Cyrus IS Chess en el nivel simple, y perdió con Colossus en el nivel de competición.

Características	Grand Master 64	Colossus	Cyrus IS	Sargon III
Jugada invisible	NO	SI	NO	NO
Lista movimientos posibles	NO	SI	NO	NO
Juega "la siguiente mejor"	NO	NO	SI	NO
Vuelve a jugar una partida	NO	SI	SI	SI
Muestra listado	NO	SI	NO	SI
Imprime listado	NO	NO	SI	SI
Retira movimientos	SI	SI	SI	SI
Enseñanza	SI	NO	NO	SI
Promociona a no reina	NO	SI	SI	SI
Partida entre humanos	NO	SI	SI	SI
Cambia de lado	SI	SI	SI	SI
Analiza problemas	NO	SI	SI	SI
Muestra búsqueda	SOLO 1 JUGADA	SI	NO	SI
Invertir tablero	SI	SI	SI	SI
Tablas ofrecidas	NO	NO	NO	SI
Imprime tablero	NO	NO	SI	SI
Guarda partida	NO	SI	SI	SI
Inhabilita librería	NO	NO	NO	SI
Partida automática	NO	SI	SI	SI
Reloj de tiempo real	SI	SI	NO	SI

Características

Todos los programas de ajedrez competentes deben incluir la capacidad de enrocar, promocionar un peón en una reina y capturar al paso, y determinarán situaciones de tablas por ahogado. Algunos de estos programas poseen interesantes características adicionales. Sargon III es el programa que tiene más extras e incluye un segundo disco que contiene 107 partidas de ajedrez clásicas y 45 problemas de ajedrez. La documentación es excelente, con 75 páginas en un cuaderno de hojas sueltas. Por supuesto, Sargon III se ejecutó en un Apple IIe y salió tres veces más caro que los otros programas. Por precio, Cyrus IS Chess y Colossus también ofrecen algunas atractivas características, como se puede apreciar en la tabla.

Conclusión

En términos de manejabilidad, Cyrus IS Chess y Colossus son los más fáciles de utilizar porque los movimientos son entrados empleando el cursor, mientras que Sargon III y Grand Master exigen que se entren los movimientos en notación algebraica, como E2-E4 para peón 4 rey. Colossus y Sargon poseen las mejores visualizaciones en pantalla.

maestros humanos de ajedrez, un buen jugador posicional siempre le ganará a un buen jugador táctico. Los programadores de ajedrez por ordenador se han centrado en la táctica porque, para un ordenador, el juego táctico implica un proceso numérico muy simple. Si un contrincante humano realiza un movimiento poco convencional, a menudo el ordenador no ofrecerá la mejor respuesta. Por este motivo, muchos programas de ajedrez tienen dificultades para tratar con las posiciones "tranquilas", en las que ninguno de los movimientos disponibles ofrece una determinada ventaja táctica. En estas situaciones, que se producen con frecuencia en la fase final de la partida, el programa a menudo revolverá las piezas por ahí en vez de aprovechar la oportunidad que se le brinda para planear sus futuros movimientos.

Un estilo de programación que se ha desarrollado recientemente implica la "búsqueda selectiva". Utilizando esta técnica, el ordenador imita a un jugador humano, analizando con mayor profundidad un menor número de posibles movimientos. Hegener y Glaser, en Alemania, han utilizado la técnica de búsqueda selectiva en su programa *Mephisto III*, que busca todos los movimientos posibles para los dos primeros movimientos de un jugador, luego va estrechando la búsqueda y examina una gama de movimientos más pequeña pero en mayor profundidad. El *Mephisto III* también intenta distinguir entre posiciones tranquilas y posiciones tácticas. Las técnicas de este tipo convertirán a los ordenadores en un auténtico desafío para los jugadores humanos.

Corazón contra cabeza

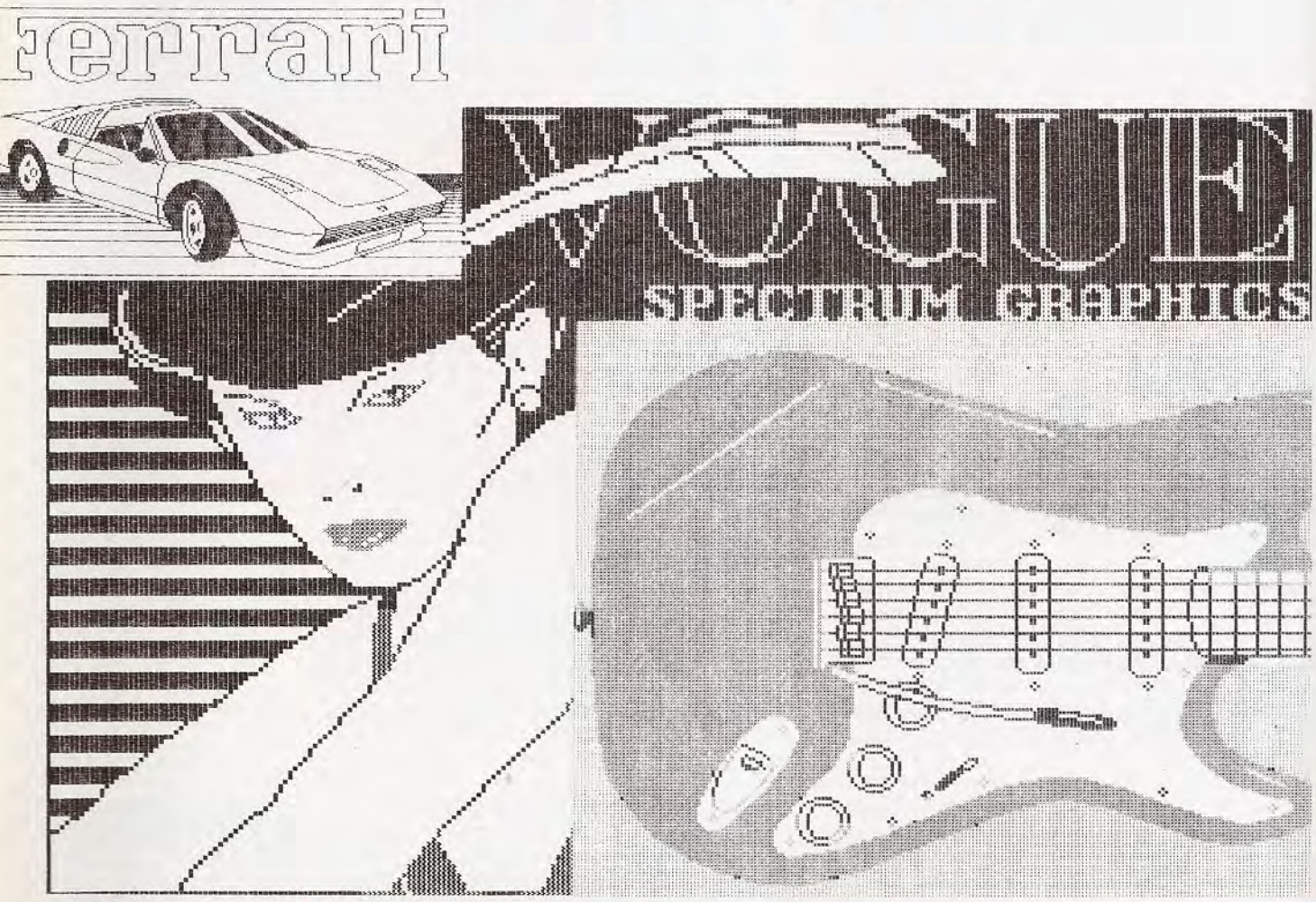
La capacidad para examinar todos los movimientos hasta nueve niveles por adelantado casi garantiza a los programas de ajedrez una superioridad táctica respecto a los seres humanos. La destreza especial de un maestro de ajedrez humano reside en seleccionar unos pocos movimientos cruciales en los que concentrar una enorme destreza en el análisis de hasta los próximos 30 movimientos.
Aquí, Moritz (negras) juega con Emmerich (blancas) en 1922. Las negras pueden dar mate sacrificando su reina y efectuando luego tres movimientos muy elegantes con el caballo; la mayoría de los jugadores de ajedrez preferirían esta secuencia a todas las demás. Moritz no lo comprendió así y lamentó su error. Todos nuestros paquetes descubrieron el mate, pero ninguno sugirió la secuencia de movimientos del caballo, aunque algunos la debieron considerar. Esta incapacidad del ordenador de percibir ese final como el "mejor" parece ofrecer a los humanos la única defensa posible contra el ajedrez de la máquina

Movimientos del caballo:

1		H5-H2
2	G1-H2	E5-G4
3	H2-G1	F4-H3
4	G1-F1	G4-H2 mate

Roy Ingram

Punto por punto



Impresión artística
Estos listados ilustran el tipo de gráficos que se pueden realizar con ciertas impresoras matriciales. Cada aguja del cabezal de impresión se controla de forma individual, y se pueden producir algunos patrones complejos y muy satisfactorios. En futuros capítulos del curso se proporcionarán detalles sobre cómo hacerlo. Estas imágenes se crearon utilizando Paintbox, de Print'n'Plotter Products

Una impresora simplifica la labor del programador y resulta imprescindible para el tratamiento de textos

Es probable que el usuario primerizo de un ordenador se quede asombrado ante la gran cantidad de impresoras disponibles en el mercado, dado que existen casi tantas máquinas diferentes como marcas de ordenadores personales. La primera decisión a tomar es determinar el tipo de impresora requerido; es probable que uno se incline por un modelo matricial o de rueda margarita, si bien existen otras variantes, como las impresoras térmicas o las de chorro de tinta. Un modelo margarita produce los resultados de mejor calidad (por lo general, a un precio proporcionalmente elevado) y, por lo tanto, es mejor para el tratamiento de textos; una impresora matricial suele ser más barata, más rápida e ideal para listados y tareas generales de programación. En este capítulo nos concentraremos en las impresoras matriciales.

La velocidad de impresión y la calidad del texto producido por una impresora matricial son puntos importantes a tener en cuenta; los modelos más caros poseen características adicionales, como espaciado proporcional (es decir, a los caracteres más estrechos, como la "i", se les asigna menos espacio que a los anchos, como la "m") y diferentes juegos de caracteres.

La velocidad de impresión es importante, ya que el uso de la impresora "paraliza" al ordenador, ya que el texto se almacena en la memoria de éste hasta que la impresora está preparada para imprimirlo. Por consiguiente, mientras se está llevando a cabo la impresión, el ordenador no se puede utilizar para otras tareas. Las velocidades de impresión se miden en función de los "caracteres por segundo" (cps), de modo que mientras que un modelo caro de 200 cps puede tardar un minuto en imprimir el listado de un programa largo, un modelo más económico con una velocidad de 30 cps tardará más de seis minutos en realizar el mismo listado (y durante esos seis minutos el ordenador no se puede emplear en ninguna otra tarea). Este problema se puede superar utilizando un *buffer* de impresión. Éste consiste simplemente en una placa de circuito

impreso que contiene chips de RAM, que se conecta entre la impresora y el ordenador y almacena los datos mientras la impresora trabaja, dejando libre al ordenador para otras operaciones.

No obstante, las velocidades de impresión que anuncian los fabricantes se deben aceptar con reservas. Al igual que en el caso de las cifras que se dan en relación al consumo de combustible de los automóviles, éstas siempre se dan suponiendo condiciones ideales y con frecuencia tienen poco que ver con la realidad. Las velocidades de impresión se calculan para la impresión de una única línea de texto compuesta del mismo carácter. El texto normal, con sus diferentes caracteres, espacios, saltos de línea y retornos de carro, retarda el cabezal de impresión. Por consiguiente, una impresora con una velocidad nominal de 160 cps probablemente dará un promedio de unos 100 cps al imprimir el listado de un programa.

La calidad de los caracteres producidos sobre el papel varía de modo considerable de una impresora a otra, y depende básicamente de cuántas agujas se utilicen en el cabezal de impresión. Los modelos más baratos emplean apenas siete agujas en el cabezal de impresión, mientras que las máquinas más caras pueden tener 16 o más. En la impresora Commodore, que sólo posee siete agujas, los caracteres se producen como una matriz de puntos de siete por seis. La Canon PW1080, sin embargo, utiliza una matriz de 16 por 23. En consecuencia, los puntos individuales no se ven y los caracteres tienen un aspecto "sólido", claramente definido.

Una impresora matricial es, en realidad, un microordenador dedicado; utiliza chips de memoria ROM y RAM y posee un microprocesador. Como tal, se la puede programar para que haga otras cosas aparte de imprimir texto. Esto se hace enviando códigos de control especiales desde el microordenador a la impresora, o moviendo pequeños interruptores, conocidos como interruptores DIP *(Dual In-line Package)*, situados dentro de la carcasa de la impresora. Por ejemplo, el juego de caracteres ASCII estándar, que está almacenado en la memoria de la impresora, se puede modificar para adaptarlo a distintos alfabetos. En Gran Bretaña, el signo numérico (#) se suele cambiar para que se imprima como el signo de libra esterlina (£).

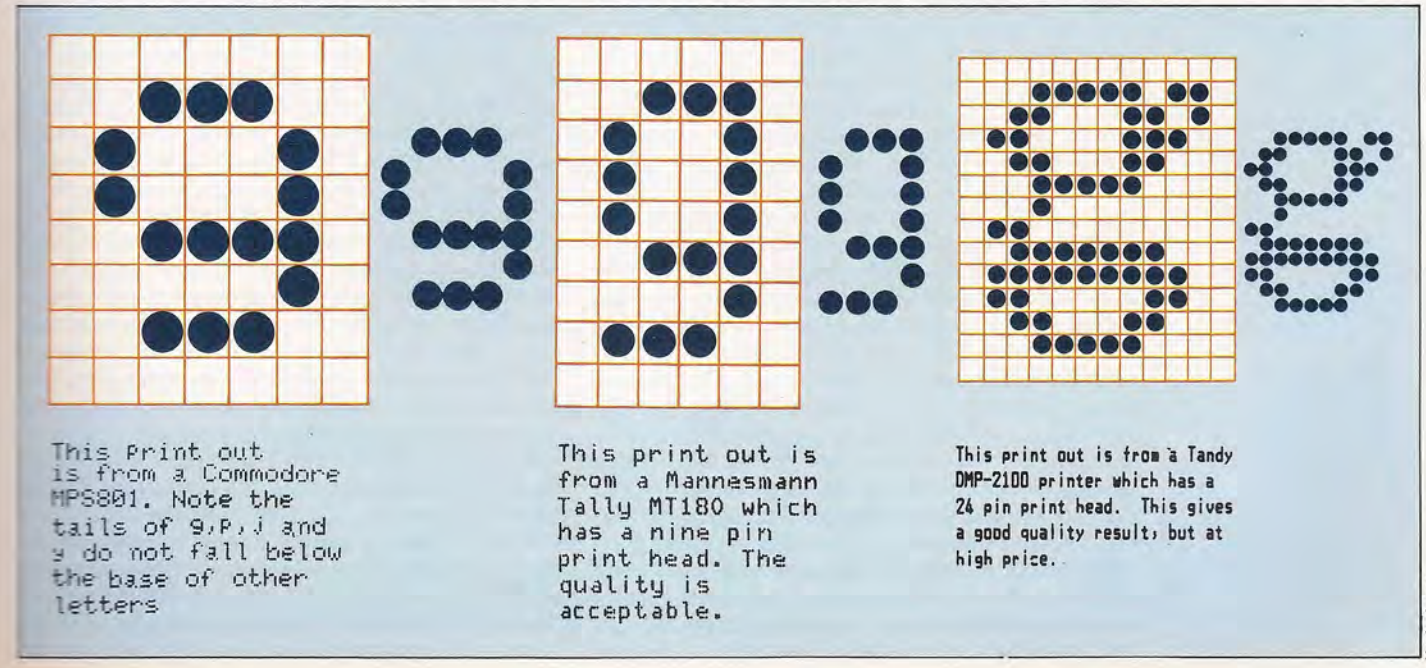
Otros efectos especiales incluyen caracteres de doble ancho, texto enfatizado (más oscuro, más grueso) y distintos espaciados entre líneas. La Epson FX80 es una de las impresoras matriciales más versátiles y posee más de 70 de estas características de impresión.

La gama de impresoras Epson se ha convertido en algo así como un "estándar industrial". Esto significa que gran parte del software que requiere el uso de una impresora (paquetes de tratamiento de textos, programas de facturación, etc.) asumen que se dispone de una Epson. Éste es un punto importante, ya que el software escrito para una marca puede fácilmente no funcionar en otra.

Hay otras consideraciones que bien podrían influir en la elección de una impresora: la fiabilidad es, por cierto, un factor a tomar en cuenta. Una impresora barata podría estar muy bien para realizar el listado de vez en cuando, pero es poco probable que resista el uso continuado que se le dará a diario a una impresora de oficina. Del mismo modo, el ruido es un factor que no se suele considerar; si a usted le gusta trabajar por las noches, algunas impresoras pueden ser realmente ensordecedoras a la una de la madrugada. ¿Posee una tracción por fricción? Todas las impresoras matriciales se suministran con el mecanismo estándar de tracción por arrastre, que trabaja sólo con papel continuo, ese papel que tiene agujeros para rueda dentada a los lados. Sin embargo, si hay que imprimir hojas de papel indivuales, es necesaria la tracción por fricción.

Por último, el factor que quizá sea el más importante: ¿funcionará con su micro? La mayoría de las impresoras matriciales vienen o bien con un conector Centronics en paralelo o con una interface en serie RS232. Si una impresora no posee la interface adecuada para su micro, en ocasiones se puede instalar una interface alternativa. Aun con la interface adecuada es necesario el cable apropiado para conectar la impresora al ordenador.

Tres distintas impresoras
Estas muestras de impresión ilustran la diferencia entre tres impresoras matriciales. La principal razón de la variación radica en el número de agujas del cabezal de impresión: las que poseen más agujas son las que ofrecen los caracteres más detallados
La muestra de la izquierda corresponde a una Commodore MP8801. Nótese que las "colas" de las letras *g, p, q* e *y* no caen por debajo de la base de las otras letras
En el centro vemos una impresión realizada por una Mannesmann Tally, que cuenta con un cabezal de impresión de nueve agujas. La calidad es aceptable
Por último, a la derecha, podemos apreciar la composición de una impresora Tandy DMP-2100, que posee un cabezal de impresión de 24 agujas y que proporciona un resultado de buena calidad. Su precio es superior al de las dos anteriores impresoras



Kevin Jones

Un programa versátil

Una hoja electrónica puede servir como "generador de ideas" y constituirse, además, en valiosa ayuda en el tratamiento de la información

Al igual que un procesador de textos o una base de datos, la hoja electrónica posee muchas facilidades que sus usuarios raramente investigan. La mayoría de las personas que disponen de un sistema para tratamiento de textos utilizan pocas veces sus instrucciones más sofisticadas, al igual que los programas de bases de datos tienden a emplearse como sistemas de índice y administración de archivos, desaprovechando sus capacidades para el proceso de datos. No obstante, la mayor parte de los usuarios de ordenadores personales no poseen un programa de hoja electrónica y no son conscientes de la conveniencia de contar con alguno. Muchos piensan que un programa de este tipo les resultará aburrido y de escasa utilidad práctica, y por lo general se sienten intimidados por la asociación de la hoja electrónica con su uso para fines financieros y de gestión. Considerarla desde este punto de vista es menospreciar la importancia de la administración financiera en el hogar y pasar por alto el hecho de que las hojas electrónicas son sencillamente procesadores de ideas que han sido encasillados en la imagen de su uso contable. De hecho, las hojas electrónicas son a los conceptos lo que los procesadores de textos son al texto.

Una hoja electrónica es en realidad un editor de textos y una calculadora, todo en uno. Se la denomina hoja electrónica porque está dividida en filas y columnas, como una hoja de contabilidad, con los datos dispuestos en celdas o casilleros. Éstos se parecen a los casilleros de una hoja de papel en el sentido de que se las puede utilizar de muchas formas diferentes: se puede depositar texto en una celda en la cual permanecerá para su visualización, se pueden entrar datos numéricos para visualización y cálculo, o se pueden entrar fórmulas matemáticas que operen con el contenido de otros casilleros. Una vez que las fórmulas están en su sitio, la hoja electrónica se convierte en un programa generado por el usuario que está a la espera de una entrada. Cada vez que se da entrada a datos nuevos (en forma de texto, datos numéricos o datos algebraicos), todas las celdas de fórmulas se recalculan incorporando los nuevos datos, manteniendo de esta manera la hoja electrónica constantemente actualizada. Ésta puede, por consiguiente, utilizarse para tareas simples de visualización por pantalla-impresora, haciendo que resulte fácil formatear e imprimir no sólo cálculos que podría hacer usted mismo (si no fueran tan tediosos), sino también otros en los que, de otra forma, jamás se le hubiera ocurrido pensar.

En muchos casos, utilizar una hoja electrónica ayudará a revelar necesidades de las que el usuario no era consciente, como llevar inventarios, analizar resultados deportivos, diseñar formularios, generar devoluciones de impuestos, decidir si alquilar o comprar un televisor, etc. Todas estas tareas las podría programar alguien que tuviera conocimientos prácticos de BASIC, pero desarrollar cada una de ellas ocuparía horas, y la mayor parte de este tiempo se perdería elaborando y depurando las infinitas instrucciones PRINT TAB, PRINT AT e INPUT necesarias para formatear la salida por pantalla. La gran ventaja de la hoja electrónica es que uno va formateando la visualización a medida que elabora las relaciones entre las variables. Esto se efectúa con la misma naturalidad con que uno prepararía una hoja de papel, escribiendo el texto, los datos y los resultados de los cálculos en el lugar en donde desearía que se visualizara.

Las hojas electrónicas disponen de diversas instrucciones para simplificar el trazado: uno puede copiar, desplazar o eliminar bloques de celdas, insertar y eliminar filas y columnas y definir el formato de una celda o un bloque en términos de tamaño, justificación (alineación con otros ítems de la misma columna) y posición de la coma decimal. Éstos son precisamente los detalles que son tan difíciles de controlar en la mayoría de las versiones de BASIC, pero que son vitales para el aspecto y la sencillez de uso de cualquier informe.

Las funciones de cálculo son análogas a estas facilidades de formato. Con una única instrucción se puede calcular el valor medio de una fila o columna de datos, contar las entradas de una tabla distintas a cero, elaborar la suma de una matriz de valores, hallar los valores máximo y mínimo de una lista y utilizar estas facilidades en expresiones matemáticas con operadores y funciones más familiares, como "+" y "/", SQR y ABS. No obstante, no todas las hojas electrónicas disponen de estas facilidades.

Quizá la instrucción más práctica de la hoja electrónica sea REPLICATE (repetir). Su utilización permite duplicar un cálculo o valor entrado en una o más celdas, de modo que en una docena de pulsaciones de teclas se pueden obtener tablas de acumulación de datos (como el interés hipotecario de mes a mes, o los gastos domésticos semana a semana). La programación de hojas electrónicas se convierte muy rápidamente en una extensión natural del BASIC aritmético, permitiendo expresar complicadas expresiones matemáticas en una forma más directa de la que permite este lenguaje.

Las hojas electrónicas ya completadas se pueden guardar (SAVE) en cinta o disco y cargar (LOAD) desde los mismos, y muchas versiones ofrecen la opción de guardar sólo el texto y los datos en un formato de archivo que se puede tratar por software de base de datos y tratamiento de textos. Esto hace posible incorporar en bloque los resultados de cálculos en un archivo de textos o de datos, y cons-

Resultado proporcional

Formato
FORMAT establece la anchura de la columna D, justifica por la izquierda las celdas de texto y visualiza los números con dos posiciones decimales

Copia
Cualquier bloque de celdas se puede copiar en cualquier parte de la hoja mediante la orden COPY

Factor de peso
Se multiplica por una calificación real para producir su correspondiente calificación proporcional

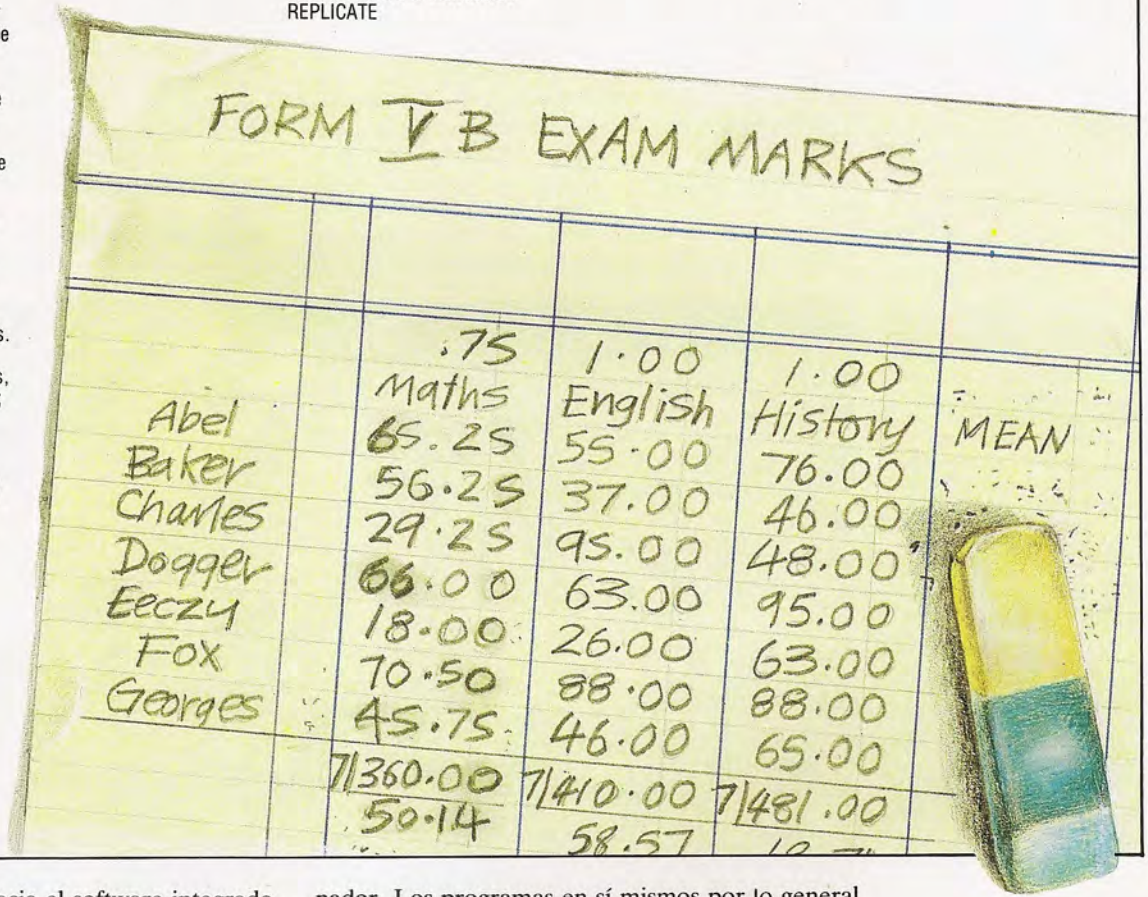
```
    |  C  |  D  |  E  |  F  |  G  |  H  |  I  |  J  |  K  |  L  |  M  |
 1| MARKS ADJUSTMENT EXAMPLE
 2|*************************************************************************
 3|                ACTUAL   MARKS        *        SCALED   MARKS
 4|*************************************************************************
 5|                                      *   .75     .86     .73
 6|          Maths    Eng     Hist   MEAN  *  Maths    Eng     Hist    MEAN
 7|Abel    : 87.00   55.00   76.00  72.67  *  65.25   47.30   55.48   56.01
 8|Baker   : 75.00   37.00   46.00  52.67  *  56.25   31.82   33.58   40.55
 9|Charles: 39.00   95.00   48.00  60.67  *  29.25   81.70   35.04   48.66
10|Dogger  : 88.00   63.00   95.00  82.00  *  66.00   54.18   69.35   63.18
11|Eezy    : 24.00   26.00   63.00  37.67  *  18.00   22.36   45.99   28.78
12|Fox     : 94.00   88.00   88.00  90.00  *  70.50   75.68   64.24   70.14
13|George  : 61.00   46.00   65.00  57.33  *  45.75   39.56   47.45   44.25
14|                                      *
15|  MEAN  : 66.86   58.57   68.71  64.71  *  50.14   50.37   50.16   50.23
16|                                      *
17|*************************************************************************
18|
```

Repetición de texto
Un solo asterisco digitado en esta celda rellena la fila entera gracias a la función REPEAT TEXT

Autocalc
Después de entrar la fórmula para una celda, se la puede copiar automáticamente en otras utilizando la instrucción REPLICATE

Promedio
Se calcula mediante la instrucción AVERAGE (cell#1:cell#2)

Para comparar el rendimiento de sus alumnos en distintas materias, el profesor desea graduar todos los resultados de los exámenes de modo que la calificación media de cada materia sea la misma. Tiene que experimentar con distintos factores de peso para cada materia, calculando una y otra vez las calificaciones, lo que constituye una tarea tediosa y muy susceptible de error que una hoja electrónica podría efectuar en cuestión de minutos. En la hoja electrónica todo, excepto las calificaciones reales, se calcula de forma automática; cambiando el factor de peso, por ejemplo, se produce una nueva columna completa de resultados proporcionales para esa materia en unos segundos



tituye un valioso paso hacia el software integrado. Esto se suele aplicar sólo en los paquetes más caros.

Disponiendo de un razonable conjunto de instrucciones, un programa de hoja electrónica está limitado principalmente por la imaginación del usuario o la cantidad de memoria disponible en el ordenador. Los programas en sí mismos por lo general son extensos, y las aplicaciones con grandes tablas y sofisticadas facilidades para proceso de datos pueden llenar rápidamente el resto de la memoria. Además, los cálculos complicados pueden retardar de forma perceptible la respuesta de cálculo del programa.

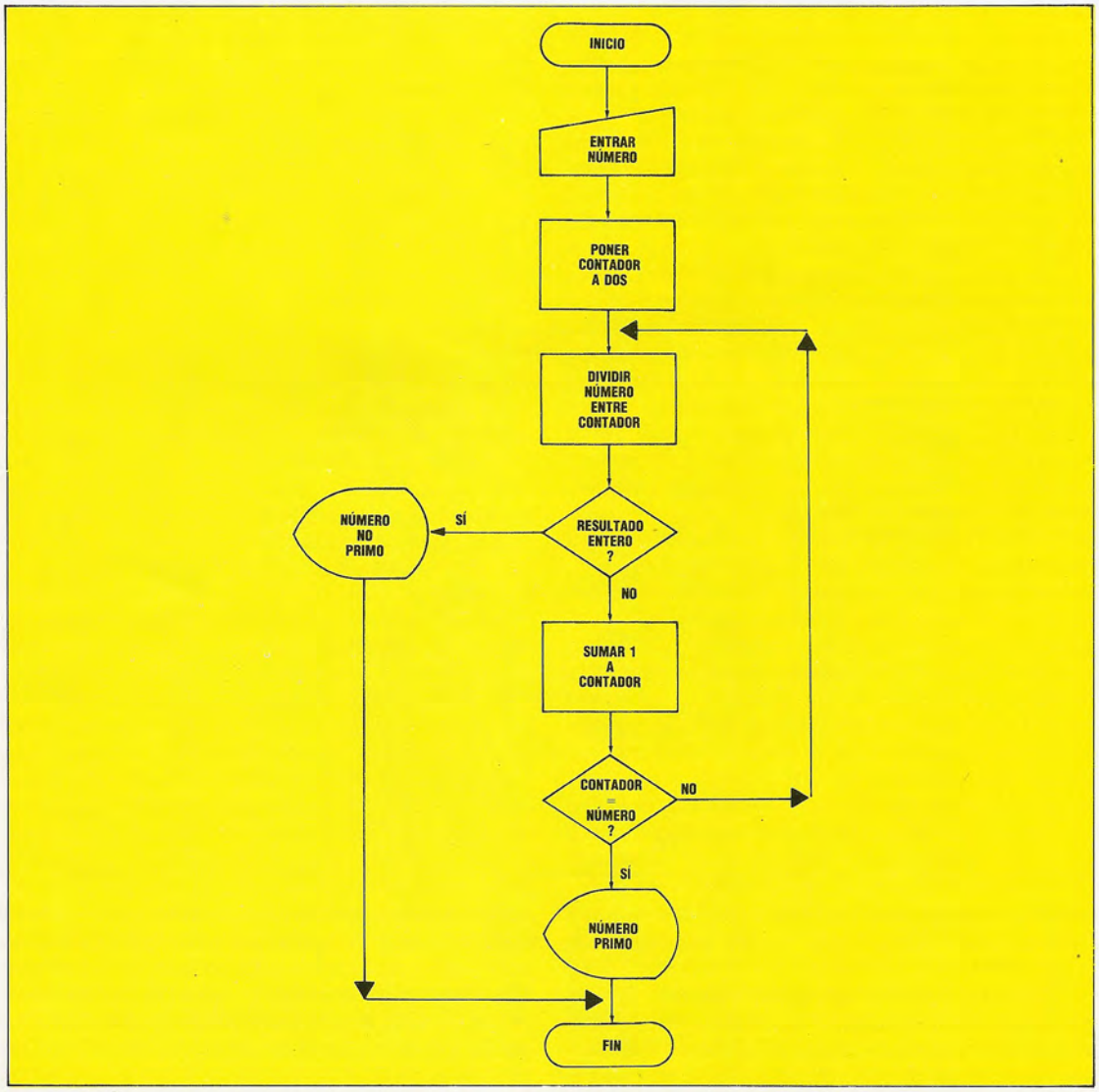
Números primos

Veamos a continuación cómo se puede averiguar si un número determinado es primo

En numerosas ocasiones surge la duda de si un determinado número es o no es primo. Entonces se debe recurrir a una serie de operaciones que pueden prolongarse según sea la complejidad de tal número. El ordinograma que ofrecemos en esta página muestra cómo se puede saber con certeza si un número es primo.

El sencillo programa descrito a continuación determina si un número entrado por teclado (contenido en la variable N) es primo o no. Para ello, el programa va dividiendo dicho número por un contador C que empieza con el valor 2 y se va incrementando de uno en uno. Si el resultado de alguna de las sucesivas divisiones es un número entero, significa que el número N es divisible por ese valor de C y, por tanto, no es primo. Si, en cambio, el contador C alcanza el valor de N sin hallar ninguna división con resultado entero, significa que el número es primo.

```
10 REM ********** = NUMEROS PRIMOS
20 INPUT"ENTRE UN NUMERO";N
30 C = 2
40 X = N/C
50 IF X = INT(X) THEN PRINT "EL NUMERO
   ENTRADO NO ES PRIMO" : END
60 C = C +1
70 IF C <> N THEN GOTO 40
80 PRINT "EL NUMERO ENTRADO ES PRIMO"
90 END
```



Atractiva solidez

Colour Genie, una máquina grande y sólida, diseñada para uso personal, ofrece una interesante relación posibilidades-precio

Basado en el popular microprocesador Z80, el Colour Genie posee un teclado tipo máquina de escribir con 62 teclas. Éstas incluyen cuatro teclas de función, dos Reset (que se deben pulsar simultáneamente) y una Mode Select (selección de modo), que permite obtener desde el teclado caracteres gráficos predefinidos.

La máquina tiene 32 Kbytes de memoria. De éstos, hay dos Kbytes separados para el sistema, y los gráficos de alta resolución utilizan otros cuatro Kbytes. Los 16 Kbytes de ROM contienen una versión ampliada del BASIC Microsoft, que no ofrece ninguna de las características de programación estructurada de que disponen versiones de BASIC más recientes. Sin embargo, admite variables enteras, doble precisión, matrices multidimensionales de cualquier tipo de variables, y amplias facilidades para manipulación de strings. Incluye muchas y muy prácticas instrucciones para realizar efectos de sonido y gráficos de alta resolución.

Las facilidades de sonido son relativamente sofisticadas: ofrecen tres canales (que permiten tocar cuerdas) y posibilitan la salida a través del televisor. Dos instrucciones de BASIC controlan la generación de sonido: PLAY da un sonido predefinido similar a un repique de campanas, mientras que SOUND permite generar otros ruidos.

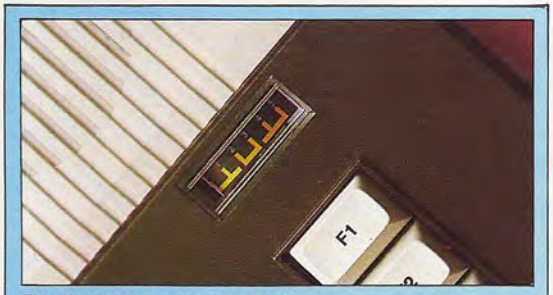
A pesar de ser amplias y potentes, las facilidades para gráficos del Colour Genie están actualmente algo anticuadas. La pantalla se considera como dos "páginas" (de hecho, dos zonas distintas de la memoria de pantalla), una de las cuales almacena y visualiza texto, bloques de caracteres para gráficos y caracteres para gráficos predefinidos, mientras que la otra página se utiliza para la visualización de gráficos de alta resolución. En modalidad de textos, el Color Genie puede visualizar hasta 25 líneas de 40 caracteres. En modalidad de gráficos, las dimensiones de visualización son de 160 ×102 pixels, lo que apenas es "alta resolución" para los estándares actuales.

Tratamiento de gráficos

La tecla Mode Select accede a la página de alta resolución, reservando 4 Kbytes de memoria. El BASIC dispone de numerosas instrucciones para tratamiento de gráficos: se pueden trazar líneas, rellenar superficies con bloques de color sólido y definir, dibujar y borrar formas. Cuando se la incorpora en un programa en BASIC, la instrucción FGR visualiza la página para gráficos, aunque al final del programa el ordenador vuelve automáticamente a la modalidad de textos. El BASIC incluye, asimismo, instrucciones para limpiar la página de gráficos (FCLS) y cambiar los colores de fondo (FILL) y primer plano (FCOLOR). Este sistema es más incómodo que la disposición de página única adoptada por

Chris Stevens

El tímido Genie
Un ordenador que nunca ha alcanzado la fama del Spectrum ni del Commodore 64 es el Colour Genie, a pesar de llevar en el mercado aproximadamente el mismo tiempo. Sea como fuere, tiene un grupo de partidarios pequeño pero exclusivo. La máquina tiene 32 Kbytes de memoria y unas palancas de mando bastante inusuales (vienen dos, con teclados numéricos incorporados y con una base propia)



El indicador de cassette del Genie
Intentar adecuar una grabadora de cassette al nivel de volumen correcto para el ordenador puede ser sumamente difícil, pero el Colour Genie posee su propio indicador de cinta para simplificar de manera considerable la escritura y lectura de una cassette

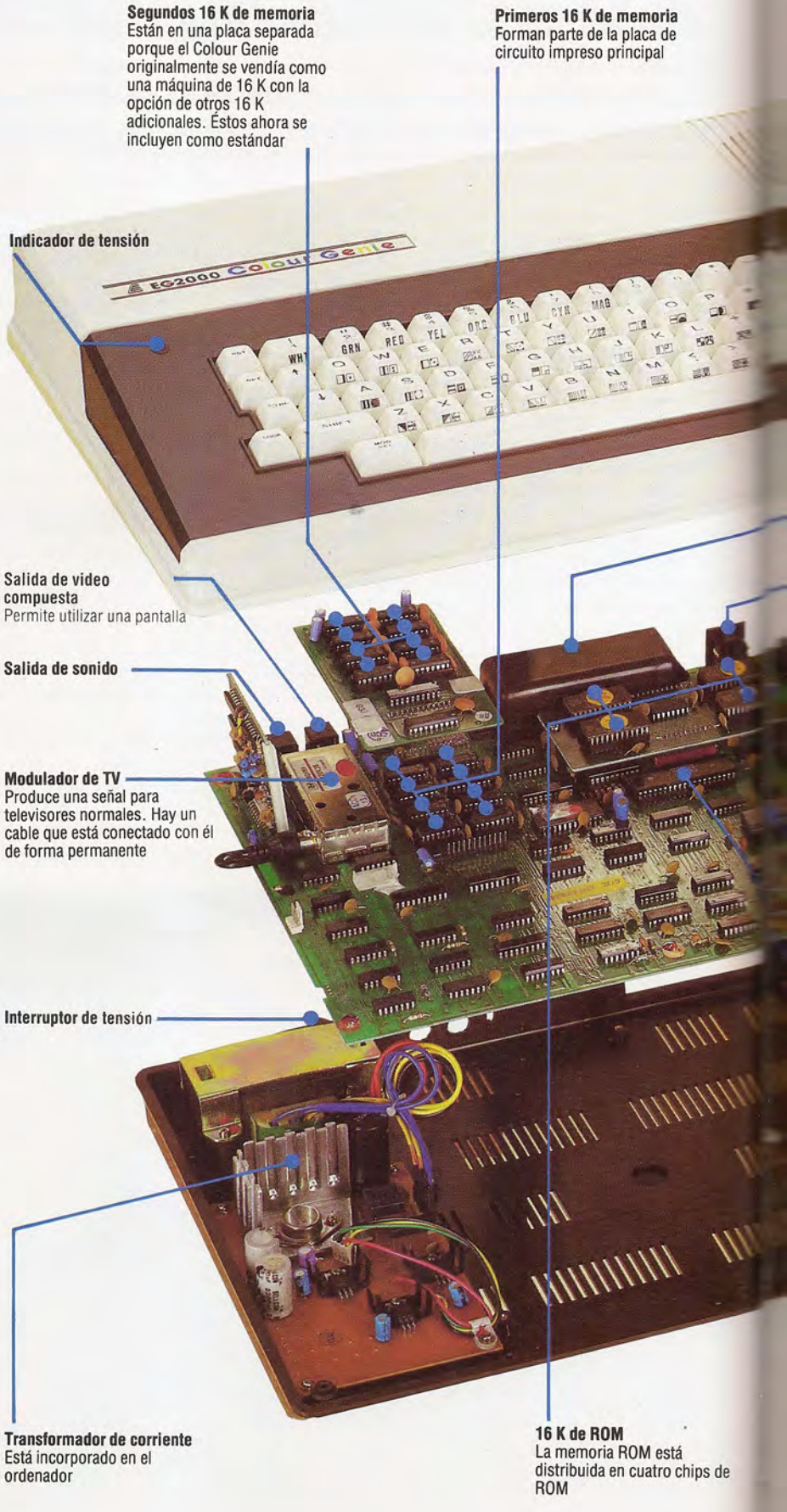
la mayoría de las máquinas nuevas, pero permite colorear cada pixel individualmente (a diferencia del Spectrum, por ejemplo, que posee una resolución más alta pero limita los colores que se pueden visualizar dentro de cada bloque de ocho por ocho pixels). La mayor parte del software de tipo recreativo utiliza la pantalla de textos por la velocidad, con caracteres definidos por el usuario para dar un efecto de alta resolución.

La visualización en pantalla es clara y uniforme, pero el juego de caracteres utilizado hace que resulte algo difícil leer el texto. El Genie dispone de ocho colores (blanco, rojo, verde, amarillo, cyan, magenta, azul y naranja), todos los cuales se pueden visualizar en la pantalla de textos al mismo tiempo. Los gráficos de alta resolución limitan al usuario a cuatro colores (rojo, azul, verde y negro), pero hay una instrucción adicional (BGRD) para poner el fondo de la página de gráficos en rosa.

Se incluyen varias interfaces: una salida RS232 para impresoras y modems; una puerta de ampliación de 50 canales, que se utiliza para conectar unidades de disco; una salida de video compuesta; una salida de audio; un conector para lápiz óptico y una puerta para palanca de mando. Entre los periféricos disponibles se incluyen palancas de mando, una interface para impresora Centronics, un cartucho Prestel (que requiere un modem) y unidades de disco. Las palancas de mando dobles tienen pequeños teclados incorporados pero son difíciles de utilizar (hace falta presionar muy fuerte para que respondan, y las palancas de mando no vuelven a la posición central "neutral" cuando se las suelta). Eaca, la empresa que fabrica el Colour Genie, no ofrece una unidad de disco para la máquina. Hay una disponible gracias a una empresa británica que ofrece su propia unidad de disco utilizando un sistema operativo llamado QDOS, similar al TRS-DOS de Tandy.

La carcasa lleva incorporado un indicador de nivel de grabación para paliar los problemas de carga desde la cassette; el usuario simplemente ajusta el volumen hasta que la aguja se centra, después de lo cual las cintas de cassette se cargan con facilidad. Además, se puede instalar un "estabilizador de datos" entre la grabadora de cassette y el cable para cassette del Genie; éste "limpia" la señal y también ayuda a la operación con las cintas.

El Colour Genie se suministra con dos manuales: una guía para el principiante y un manual básico. Ambos están escritos con claridad pero no son lo suficientemente detallados y ninguno posee índice. En realidad, el manual básico ni siquiera tiene una



página de contenido. Para los usuarios más avanzados existe un manual técnico que se ha de pagar aparte.

A pesar de su aspecto anticuado, el Colour Genie parece ofrecer una buena relación calidad-precio. Cae de lleno en la categoría de "ordenador personal" y tiene poco que ofrecer para el usuario científico o de oficina. La construcción robusta, las buenas capacidades de sonido, la completa gama de periféricos y un BASIC bastante normalizado harán que esta máquina sea especialmente atractiva para el principiante.

Martian raider (Invasor marciano)

Ian McKinnell

Opciones de software
La disponibilidad de software para el Colour Genie es bastante limitada, pero la calidad de lo que se puede obtener por lo general es muy buena. La mayor parte del software son juegos y con frecuencia éstos son versiones traducidas de juegos creados para las máquinas más conocidas

COLOUR GENIE

DIMENSIONES
90 x 280 x 340 mm

CPU
Z80, 2,2 MHz

MEMORIA
32 K de RAM, 16 K de ROM

PANTALLA
Hasta 25 filas de 40 columnas de texto; gráficos hasta 160 x 102 pixels. Ocho colores en modalidad de textos; 4 colores para gráficos

INTERFACES
Palancas de mando (2), puerta RS232, puerta para lápiz óptico, puerta para ampliación

LENGUAJES DISPONIBLES
BASIC incluido

TECLADO
Tipo máquina de escribir, con 62 teclas y cuatro de función

DOCUMENTACION
La máquina se suministra con una guía para el principiante y un manual básico. Ambos son demasiado breves como para ser de mucha utilidad, y ninguno de los dos tiene índice. También hay a la venta un manual técnico

VENTAJAS
El Colour Genie es una buena máquina "familiar". Es de construcción robusta, utiliza BASIC Microsoft, ofrece gráficos razonables y buen sonido, con salida por el televisor

DESVENTAJAS
El diseño del Genie es anticuado; posee un procesador lento y la pantalla se manipula en forma de dos "páginas". Hay poco software para esta máquina

Palancas de mando del Colour Genie
Las palancas de mando del Colour Genie son muy atractivas, pero caras y difíciles de utilizar. Se necesita mucha presión para moverlas y no vuelven a la posición central al soltarlas. Los teclados numéricos incorporados son inusuales y no muy prácticos

Chris Stevens

Arenas peligrosas

Conducir un camión por el desierto no es tarea sencilla cuando no se puede llevar todo el combustible necesario

Nuestro juego se desarrolla en un desierto de 1 000 km de anchura. Aproximadamente cada 100 km hay una posta de aprovisionamiento donde se pueden almacenar tanques de combustible. De vuelta en la base, uno puede hacerse con tantos tanques de gasolina como crea que va a necesitar, cada uno de ellos suficientemente grande como para abastecer al camión de una etapa hasta la siguiente. La travesía a través del desierto sería bastante sencilla si no fuera por un especial detalle: el camión sólo tiene espacio para un máximo de ocho tanques a la vez. Por consiguiente, para conducirlo a través de las arenas es necesario proveerse de combustible en varios puntos de la ruta, yendo y viniendo de un puesto a otro.

Obviamente, el primer objetivo del juego consiste en asegurarse de que no se quedará sin gasolina: el camino hasta la base es muy largo y el desierto no es el lugar más apropiado para realizar una agradable caminata. En segundo lugar, se debe completar la travesía cubriendo la distancia más corta posible y utilizando la menor cantidad posible de tanques. Solucionar esta doble dificultad le resultará bastante sencillo con el programa preparado para ocho tanques.

Sin embargo, podemos modificar el juego para hacer que el problema resulte un poco más acuciante. ¿Qué sucede, por ejemplo, si únicamente se pueden llevar cuatro o seis tanques cada vez? Para investigar estas variantes, sólo hay que alterar el valor de la variable M de la línea 60. Descubrirá que está utilizando la misma técnica pero que se modifican los intervalos entre sus aprovisionamientos de combustible y la cantidad de jornadas. ¿Puede usted elaborar un algoritmo que lo conduzca con seguridad a través del desierto todas las veces? Este algoritmo, una vez encontrado, podría proporcionar la base para establecer un programa que *resuelva* este problema en particular.

Nuestro enigma demuestra una valiosa técnica para resolver problemas en el desarrollo de un programa. Lo primero que debe hacerse es experimentar con la información dada, probar cierta cantidad de ejemplos calculados y luego, si todo va bien, descubrir un patrón común. A partir de éste se puede elaborar un algoritmo, transformándolo luego en un programa. Si se desea desarrollar nuestro juego *Desert trucker* (El camionero del desierto), se pueden agregar gráficos y otros refinamientos, introduciendo dificultades como, por ejemplo, el tener que llevar agua además de gasolina.

Complementos al BASIC

Este programa está escrito en BASIC Microsoft, de modo que se podrá ejecutar en la mayoría de las máquinas que posean una visualización en pantalla de 40 × 25. En el Spectrum se debe insertar LET antes de las sentencias de asignación.

CHR$(26): Sustituir por CLS en el Spectrum, Oric-1, Atmos, Dragon y BBC; y por CHR$(147) en el Commodore 64 y el Vic-20.

MID$(STR$(A(1),2)): Reemplazar por STR$(A(1)) en el Spectrum, y en todas las máquinas en las que PRINT LEN(STR$(2)) dé 1 como resultado.

THEN 1260 & THEN 1300: En el Spectrum, cambiar por THEN GOTO 1260 & THEN GOTO 1300.

```
10 REM ****Camionero del desierto
30 DIM A(10)
40 A(1) = 80: REM Tanques al comienzo
50 S = 1:REM Posición del camion
60 M = 8: REM Max. de tanques en el camion
70 N = 0: REM Total de tanques usados
100 REM Proxima vuelta
110 PRINT CHR$(26): REM Limpiar pantalla
120 PRINT "PUESTO: 1 2 3 4 5 6 7 8 9 10"
130 PRINT "TANQUES:  ";
140 FOR I = 1 TO 10
145 A$ = MID$(STR$(A(I)),2)
147 IF LEN(A$) < 2 THEN LET A$ = "  " + A$:GOTO 147
150 PRINT A$;:  ";:NEXT I:PRINT
160 X = 9 + (S-1)*3:PRINT TAB(X);"*";:PRINT TAB(X-1);"-";:PRINT TAB(X-2);"CAMION";:PRINT
    TAB(X-1);T
175 IF S = 10 THEN PRINT:PRINT"Lo ha conseguido!!!" GOTO 1400
180 IF T = 0 AND A(S) = 0 THEN PRINT: PRINT "Amigo... sera mejor que empiece a caminar" GOTO
    1400
190 PRINT: PRINT"Tus opciones son:": PRINT
200 IF T > 1 THEN PRINT "A abandonar tanques de combustible"
210 IF A(S) > 0 THEN PRINT "P Un puesto hacia adelante"
220 IF T > 0 AND S > 1 THEN PRINT "R Un puesto hacia atras"
230 IF T > 0 AND S > 1 AND (A$ = "P" OR A$ = "p") THEN S = S + 1:T = T-1:N = N + 1:GOTO 110
240 PRINT:PRINT "Cual?";:INPUT A$
250 IF T > 1 AND (A$ = "A" OR A$ = "a") THEN 1260
260 IF A(S) > 0 AND (A$ = "C" OR A$ = "c") THEN 1300
270 IF T > 0 AND (A$ = "P" OR A$ = "p") THEN S = S + 1:T = T-1:N = N + 1:GOTO 110
280 IF T >0 AND S > 1 AND (A$ = "R" OR A$ = "r") THEN S = S-1:T = T-1:N = N + 1:GOTO 110
290 GOTO 110
1250 REM Abandonar tanques
1260  PRINT:INPUT "Cuantos?";A
1270 IF A > T OR A <> INT(A) OR A < 0 THEN PRINT:PRINT"Por favor pruebe otra vez":GOTO 1260
1280 A(S) = A(S) + A:T = T-A:GOTO 110
1290 REM Coger tanques
1300 PRINT:INPUT "Cuantos?";A
1310 IF A > A(S) OR A <> INT(A) OR A < 0 THEN PRINT:PRINT "Por favor pruebe otra vez":GOTO 1300
1320 IF A + T > M THEN PRINT: PRINT"Lo siento, solo hay lugar para llevar";M;"tanques":GOTO 1300
1330 T = T + A:A(S) = A(S)-A
1380 GOTO 110
1390 REM Final del juego
1400 PRINT:PRINT "Ha utilizado";N;"tanques de gasolina ".
1410 PRINT"de los que traia";T
1420 A = 0:FOR I = 2 TO 9:A = A + A(I):NEXT I
1430 PRINT "consigo y ha dejado";A;"en el desierto."
1440 PRINT:PRINT"Entre RUN para volver a jugar"
1450 END
```

Adrian Morgan

Las tres vidas de Willy

"Manic miner" combina hábilmente humor y gráficos originales, lo que lo ha convertido en un programa de entusiasta aceptación

Manic miner, disponible para el ZX Spectrum de 48 Kbytes y el Commodore 64, es fundamentalmente un juego muy sencillo basado en un best-seller anterior llamado *Kong*. El objetivo de este juego era trepar por escaleras y ramas mientras se evitaban obstáculos, en un intento por rescatar a la afligida doncella que el Gran Simio mantenía cautiva. En *Manic miner*, el jugador asume el papel de Miner Willy, prospector del centro minero denominado Surbiton. Willy encuentra el pozo de una mina olvidada del que una civilización perdida extraía oro y otros metales preciosos. Lamentablemente, los antiguos habitantes de la mina se han olvidado de desactivar los Manic Mining Robots (robots mineromaníacos), por lo cual la tarea de recuperar el tesoro, en principio no demasiado ardua, resulta sumamente difícil.

La mina tiene 20 cavernas y cada una de ellas contiene cuatro llaves que hay que coger, para poder abrir la puerta que conduce a la etapa siguiente. Cada caverna tiene diversos salientes sobre los que hay que saltar para poder alcanzar las llaves. Algunos de estos salientes están en malas condiciones (presumiblemente por su antigüedad) y ceden cuando Willy llega a ellos. Las cavernas están rotuladas con frases, como "La guarida de Eugene" (alusión al programador Eugene Evans, joven prodigio de Imagine), "El minero Willy se encuentra con el Rey Bruto", "El ataque de los teléfonos mutantes" (otra "broma para los del ramo", esta vez dirigida al programador Jeff Minter, cuya obsesión son las llamas mutantes) y "Bahía para el aterrizaje del Skylab". Todas las cavernas están habitadas por numerosos seres extraños cuyo solo contacto significa la muerte instantánea. Hasta las plantas son letales.

El jugador dirige a Willy para evitar todos estos problemas con estas tres órdenes simples: "Derecha", "Izquierda" y "Salta". Ésta es parte de la atracción del juego: la simplicidad de los mandos significa que no se requiere un largo período de aprendizaje y uno puede seleccionar las teclas con las que se siente más a gusto.

Willy tiene tres vidas y en cada encarnación dispone de una provisión de aire limitada, indicada mediante un medidor que hay en pantalla. Como la pérdida de la tercera vida lleva a Willy de regreso a la Caverna Uno, el juego puede resultar muy frustrante y no es sorprendente que algunos usuarios se las hayan arreglado para modificar el programa con el fin de empezar por la caverna de su elección.

La traducción para el Commodore es una copia casi exacta de la versión para el Spectrum y no consigue sacar partido de las instrucciones para sonido del 64, más versátiles, y de sus gráficos de mayor resolución. En el 64 la zona de juego se ha hecho considerablemente más pequeña que el tamaño de pantalla disponible, de modo que concuerde exactamente con la versión para el Spectrum.

Pero ambas versiones son, sin duda alguna, divertidas. El ritmo del juego y la dificultad de los problemas planteados se han elaborado con sumo cuidado, haciendo que el juego resulte cautivador para el usuario. Y ahora Matthew Smith ha producido una continuación, *Jet Set Willy*, que rápidamente se está creando su propio culto, tal como ha sucedido antes con *Manic miner*.

Manic Miner: Para el Spectrum de 48 K y el Commodore 64

Editado por: Software Projects, Bear Brand Complex, Allerton Road, Woolton, Liverpool, Merseyside L25 7SF

Autor: Matthew Smith

Palancas de mando: Ambas versiones

Formato: Cassette

"Manic miner" en el Spectrum

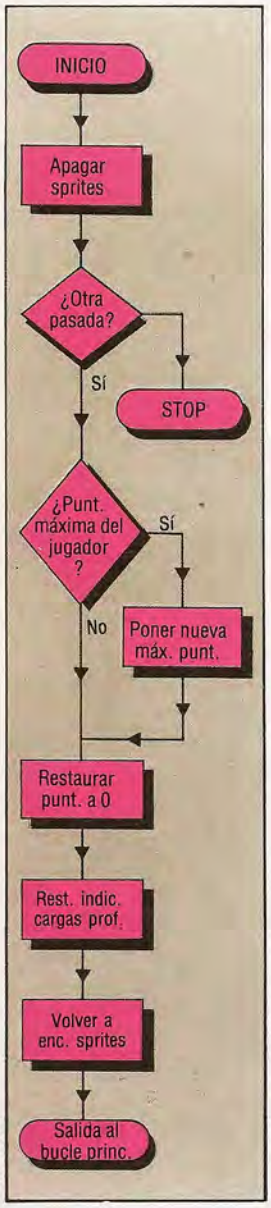
"Manic miner" en el Commodore 64

Bajo tierra
Los misteriosos y maravillosos objetos con que uno se encuentra en el mundo subterráneo de *Manic miner* han contribuido a que el juego sea objeto de multitudinaria aceptación. Los jugadores ya experimentados suelen jactarse de los objetos más insólitos que han llegado a descubrir en las cavernas

Ian McKinnell

Cargas mortíferas

Escribimos hoy las rutinas que crean una explosión cuando el submarino es alcanzado por una carga de profundidad y explicamos el final del juego



Kevin Jones

En el penúltimo capítulo del curso descubrimos lo fácil que es detectar colisiones entre los sprites utilizando un registro de colisión de sprites, V + 30. Cuando esto sucede, la subrutina HIT (que comienza en la línea 5000) tiene que realizar tres tareas. En primer lugar, debe producir una explosión en el punto de la pantalla donde han colisionado los dos sprites, y luego debe incrementar el marcador del jugador según el valor del submarino, que se calcula a partir de su velocidad (DX) y su profundidad (Y3). Por último, debe restaurar las coordenadas para que el próximo submarino comience a desplazarse a través de la pantalla. Analicemos el código de la subrutina HIT (líneas 5000-5250).

La línea 5010 coloca (POKE) un cero en el registro de colisión V + 30 para limpiarlo. Commodore sostiene que el registro de colisión de sprites se limpia él solo después de que dos sprites hayan pasado el uno sobre el otro y ya no estén en colisión. No obstante, la experiencia demuestra que el registro no siempre se limpia él solo con suficiente rapidez, creando efectos inesperados tales como que se produzcan explosiones sin ningún motivo. La solución consiste en limpiar manualmente el registro de colisión después que haya una. Hecho esto, se puede posicionar y encender el sprite de la explosión.

La línea 5030 le da a la explosión una coordenada X de diez pixels a la derecha con respecto a la de la carga de profundidad. Esta ligera desviación sitúa la explosión más centrada respecto a las cargas de profundidad. Como X2 toma su valor a partir de la coordenada X del barco (X0), su valor tiene un límite máximo de 245. Esto significa que el valor máximo de la coordenada X de la explosión es 255. La coordenada Y para la explosión se toma directamente de la del submarino.

Al sprite de la explosión se lo ha designado sprite 1. La línea 5040 pone el bit 1 del registro V + 21 a uno, encendiendo el sprite 1 sin perturbar los valores de otros bits del registro. Llegados a este punto es interesante notar que el sprite de la explosión aparecerá encima de los sprites del submarino y la carga de profundidad, o frente a ellos. Esto se conoce como *prioridad de sprites*, y se rige por la sencilla regla de que los sprites de número inferior aparecen sobre los que llevan un número superior.

El color del sprite de la explosión se controla mediante la posición V + 40 del chip VIC. Se puede obtener un efecto interesante utilizando un bucle FOR...NEXT para colocar (POKE) números de código de color entre 1 y 15. Un bucle FOR...NEXT exterior repite este proceso 20 veces (líneas 5060-5100). Cuando la explosión se ha completado, los tres sprites (explosión, cargas de profundidad y submarino) deben desaparecer de la pantalla. La línea 5130 apaga los sprites 1, 2 y 3.

Como ya hemos mencionado anteriormente, es necesario actualizar la puntuación del jugador utilizando la subrutina que comienza en la línea 5500. Dado que la puntuación se ha de incrementar en función del valor del submarino (en vez de disminuirla, como ocurre cuando un submarino alcanza ileso el borde derecho de la pantalla), el valor de DS se pone a uno para señalarla. Por último, antes de que otro sumergible pueda viajar a través de la pantalla, es necesario restaurar sus coordenadas empleando la subrutina de la línea 2500 y se debe volver a encender el sprite del submarino. Además, el indicador que señala el lanzamiento de una carga de profundidad se debe restaurar a cero de modo que el jugador pueda comenzar a disparar cargas de profundidad nuevamente.

Al cabo de tres minutos el programa abandona el bucle principal y salta a la línea 400. Cuando analizamos por primera vez la utilización del reloj del Commodore 64 (véase p. 714), la línea 400 era una simple sentencia END. La rutina Final del Juego permite volver a jugar el juego y grabar las puntuaciones máximas. El diagrama de flujo muestra las tareas que se han de incorporar en esta rutina. En el listado del programa, estas tareas se llevan a cabo entre las líneas 400 y 660. La mayor parte del código es autoexplicativo, recordando que CHR$(19) lleva el cursor hasta la esquina superior izquierda de la pantalla y CHR$(144) hace que las siguientes letras que se impriman (PRINT) se coloreen en negro.

En este corto proyecto de programación para el Commodore 64 hemos aprendido a crear un sencillo juego animado. Al construir el programa hemos cubierto los aspectos principales que intervienen en la programación de esta clase de juegos en BASIC. Quizá usted desee agregarle al programa refinamientos por su propia cuenta, aplicando los principios que hemos aprendido (p. ej., incorporando los cuatro sprites que han quedado sin utilizar).

Tabla de las variables utilizadas en el Sunhunter	
V	Comienzo de los registros del chip VIC
FL	Indic. cargas prof.; se pone a 1 si se arroja una carga
SC	Puntuación actual del jugador
HS	Máxima puntuación hasta el momento
TI$	Reloj del propio Commodore 64
X0	Coordenada X del barco
X2,Y2	Coordenadas X e Y de la carga de profundidad
X3,Y3	Coordenadas X e Y del submarino
H3,L3	Byte *hi* y byte *lo* de la coordenada X del submarino
DX	Número de pixels; en razón de él se incrementa coord. X
DS	Indicador para ver si se ha de aumentar (DS = 1) o disminuir un marcador

"Subhunter": el listado final

```
10 REM ********************************************************
30 REM **PROYECTO PROGRAMACION 64**
70 REM ********************************************************
90 POKE55,0:POKE56,48:CLR: REM BAJAR TOPE MEM
100 V = 53248:FL = 0:SC = 0
110 GOSUB1000: REM CREACION PANTALLA
120 GOSUB2000: REM CREACION SPRITES
130 GOSUB2500: REM ESTABLECER COORD SUB
140 TI$ = "000000"
200 REM ****BUCLE PRINCIPAL****
210 REM **RELOJ**
220 PRINTCHR$(19);:PRINTTAB(14)CHR$(5);"TIEMPO"MID$(TI$,3,2)"
    RIGHT$(TI$,2)
225 IFVAL(TI$) > 259 THEN 400: REM FINAL JUEGO
230 GET A$
240 IF A$ = "Z" THEN X0 = X0−1.5:IF X0 < 24 THENX0 = 24
250 IF A$ = "X" THEN X0 = X0 + 1.5:IF X0 > 245 THENX0 = 245
260 IF A$ = "M" AND FL = 0 THEN GOSUB3000: REM PREPARAR
    CARGAS PROFUNDIDAD
270 REM **MOVER BARCO**
290 POKE V,X0
300 REM **MOVER SUB**
310 X3 = X3 + DX
320 REM **SI EL SUB LLEGA AL BORDE DE LA PANTALLA**
330 IF X3 > 360 THEN DS = −1:GOSUB5500:GOSUB2500
340 H3 = INT(X3/256):L3 = X3−256*H3
350 POKE V + 6,L3
360 IF H3 = 1 THEN POKE V + 16,PEEK(V + 16)OR8:GOTO380
370 POKE V + 16,PEEK (V + 16)AND247
380 IF FL = 1 THEN GOSUB4000:REM MOVER CARGA PROFUNDIDAD
390 GOTO 200: REM RECOMENZAR BUCLE PRINCIPAL
400 REM ****FIN DE CONDICIONES DEL JUEGO****
410 REM **APAGAR SPRITES**
420 POKE V + 21,0
430 REM **RESTAURAR COORD SUB & BARCO**
440 X0 = 160:GOSUB 2500
450 INPUT"OTRA PARTIDA?(S/N)";AN$
460 IF AN$ <> "S" THENEND
480 REM **BORRAR MENSAJE**
490 PRINT CHR$(19):REM PONER CURSOR EN SU SITIO
500 FOR I = 1 TO 120
510 PRINT"    ";
520 NEXT I
540 REM **ESTABLECER MAX PUNTUACION**
550 IF SC > HS THEN HS = SC
560 PRINT CHR$(19);CHR$(144);"PUNTUACION 000":SC = 0
570 PRINT CHR$(19);
580 PRINT TAB(26);CHR$(144);" MAX PUNTUACION;HS
600 REM **RESTAURAR RELOJ E INDICADOR**
610 TI$ = "000000":FL = 0
630 REM **ENCENDER SUB & BARCO**
640 POKE V + 21,9
660 GOTO200: REM RECOMENZAR BUCLE
1000 REM ****CREACION PANTALLA****
1010 PRINT CHR$(147): REM LIMPIAR PANTALLA
1030 REM **COLOR MAR**
1040 POKE 53281,14:POKE 53280,6
1050 FOR I = 1264 TO 1943
1060 POKE I,160:POKE I + 54272,6
1070 NEXT
1090 REM **FONDO MAR**
1100 FORI = 1944 TO 2023
1110 POKE I,102:POKE I + 54272,9
1120 NEXT
1130 POKE 650,128:REM REPETIR TECLAS
1150 REM **PUNTUACION**
1160 PRINT CHR$(19);CHR$(144);"PUNTUACION 000";SPC(16);"MAX
    PUNTUACION 000"
1170 RETURN
2000 REM ****CREACION SPRITES****
2020 REM **LEER DATOS BARCO**
2030 FOR I = 12288 TO 12350
2040 READ A:POKE I,A:NEXT I
2060 REM **LEER DATOS SUB**
2070 FOR I = 12352 TO 12414
2080 READ A:POKE I,A:NEXT
2100 REM **LEER DATOS CARGAS**
2110 FOR I = 12416 TO 12478
2120 READ A:POKE I,A:NEXT
2140 REM **LEER DATOS EXPLOSION**
2150 FOR I = 12480 TO 12542
2160 READ A:POKE I,A:NEXT
2180 REM **ESTABLECER PUNTEROS**
2190 POKE 2040,192:POKE 2041,193: POKE 2042,194
2200 POKE2043,195
2220 REM **ESTABLECER COLORES**
2230 POKE V + 39,0:POKE V + 40,1:POKE V + 41,0
2240 POKE V + 42,0
2260 REM **ESTABLECER COORD INICIALES**
2270 POKE V + 1,80:X0 = 160: REM COORD BARCO
2280 POKE V + 29,15:POKE V + 23,2
2300 REM **ENCENDER SPRITES 0 & 3**
2310 POKE V + 21,9
2320 RETURN
2500 REM ****RESTAURAR COORD SUB****
2510 Y3 = 110 + INT(RND(TI)*105)
2520 POKE V + 7,Y3:POKE V + 6,0
2530 X3 = 0:DX = RND(TI)*3 + 1
2540 POKE V + 16,0
2550 RETURN
3000 REM ***PREPARAR CARGAS PROFUNDIDAD****
3020 REM **PONER INDICADOR**
3030 FL = 1
3050 REM **ESTABLECER COORD**
3060 Y2 = 95:X2 = X0
3070 POKE V + 4,X2:POKE V + 5,Y2
3090 REM **ENCENDER SPRITE 2**
3100 POKE V + 21,PEEK(V + 21)OR4
3110 RETURN
4000 REM ****MOVER CARGA PROFUNDIDAD****
4020 REM **DECREMENTAR COORD Y**
4030 Y2 = Y2 + 2
4050 REM **VERIFICAR FONDO MAR & APAGAR**
4060 IF Y2 > Y3 + 25 OR Y2 > 216 THEN
    POKEV + 21,PEEK(V + 21)AND251:FL = 0
4070 POKE V + 5,Y2
4090 REM **VERIFICAR SI SUB ACERTADO**
4100 IF PEEK(V + 30) = 12 THEN GOSUB 5000:REM
    RUTINA HIT
4110 RETURN
5000 REM ****RUTINA HIT****
5010 POKE V + 30,0:REM LIMPIAR REG COLISIONES
5020 REM **ENCENDER SPRITE EXPLOSION**
5030 POKE V + 2,X2 + 10:POKE V + 3,Y3
5040 POKE V + 21,PEEK(V + 21)OR2
5060 REM **DESTELLOS COLORES**
5070 FOR I = 1 TO 20
5080 FOR J = 1 TO 15
5090 POKE V + 40,J
5100 NEXT J:NEXT I
5120 REM **APAGAR SPRITES 1, 2 & 3**
5130 POKE V + 21,PEEK(V + 21)AND241
5150 REM **ACTUALIZAR PUNTUACION**
5160 DS = 1:GOSUB 5500
5180 REM **RESTAURAR COORD SUB & INDIC.**
5190 FL = 0:GOSUB 2500
5210 REM **VOLVER A ENCENDER SUB**
5220 POKE V + 21,PEEK(V + 21)OR8
5230 RETURN
5500 REM ****ACTUALIZAR PUNTUACION****
5510 SC = SC + INT(Y3 + DX*30)*DS
5520 IF SC < 0 THEN SC = 0
5530 PRINT CHR$(19);CHR$(144)"PUNTUACION";
    SC;CHR$(157);"    "
5540 RETURN
6000 REM ****DATOS BARCO****
6010 DATA0,0,0,0,0,0,0,0
6020 DATA0,128,0,0,192,0,0,192,0
6030 DATA0,192,0,1,224,0,1,224,0
6040 DATA13,224,0,3,248,128,3,253,8
6050 DATA15,254,16,31,255,48,255,255,255
6060 DATA127,255,254,63,255,254,31,255,252
6070 DATA0,0,0,0,0,0,0,0
6100 REM ****DATOS EXPLOSION****
6110 DATA0,0,0,0,0,0,0,16,0,0,8,0,4,16
6120 DATA0,3,2,64,1,56,128,12,255,144
6130 DATA1,238,40,5,151,0,11,121,0,1
6140 DATA183,0,25,214,96,0,236,48,6,24
6150 DATA152,3,98,0,8,51,0,0,96,128,0
6160 DATA64,0,0,0,0,0,0,0
6170 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6200 REM ****DATOS CARGAS PROFUNDIDAD****
6210 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6220 DATA0,0,0,32,0,0,32,0,0,32,0,0,32,0
6230 DATA0,0,0,0,0,0,0
6240 DATA2,0,0,2,0,0,2,0,0,2,0,0
6250 DATA0,0,0,0,0,0,0,0
6260 DATA0,0,0,0,0,0,0,0
6300 REM ****DATOS SUBMARINO****
6310 DATA0,0,0,0,0,0,0,0,0,0,0,0
6320 DATA0,8,0,0,12,0,0,12,0
6330 DATA0,12,0,0,28,0,0,60,0
6340 DATA0,126,0,199,255,255
6350 DATA239,255,255,127,255,255
6360 DATA255,255,254,199,255,254
6370 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Éste es el listado final para nuestro programa Subhunter junto con una tabla de las variables empleadas en el mismo. El listado contiene muchas sentencias REM para facilitar su comprensión. Éstas se pueden pasar por alto al entrar el programa en el ordenador propio, pero hay que tener cuidado en no eliminar alguna línea REM que se requiera en otra parte del programa. Por ejemplo, se podría optar por eliminar el REM de la línea 400, pero este número de línea se utiliza como parte de una sentencia GOTO en la línea 225. Si elimina la línea 400 por completo aparecería un mensaje "ERROR DE SENTENCIA NO DEFINIDA EN LA LINEA 225" (UNDEF'D STATEMENT ERROR AT LINE 225) y el programa se colgaría. La mejor forma de evitar esto consiste en omitir sólo aquellas REM que aparezcan al final de una línea y aquellas líneas que utilicen dos puntos (:) para separar las instrucciones

Dividir y ver

Con este capítulo damos fin al estudio básico del lenguaje máquina. En él explicaremos cómo se divide y cómo se programa la visualización por pantalla

Lo mismo que nos ocurrió con la multiplicación al estilo tradicional, que nos sirvió de modelo algorítmico para la multiplicación en sistema binario (véase p. 778), nos pasa ahora con la división. Consideremos el siguiente ejemplo:

```
10011010 |1011          dividendo-divisor
-1011↓   |00001110      cociente
010000
 -1011↓                 restas sucesivas
0001011                 del divisor
  -1011↓
00000000                resto
```

El meollo de este algoritmo está en cómo resta una y otra vez el divisor a los bits superiores que van quedando del dividendo. Según sea el resultado, se va colocando en el cociente un 1 o un 0. El resto es el resultado de la última sustracción.

Por desgracia, no son tan fáciles de explicar las diversas maneras de adaptar este método en lenguaje máquina, como lo fueron en la multiplicación. Digamos al menos que el Z80 no deja de aprovechar el potente y flexible recurso que le brindan sus registros de 16 bits, mientras que el 6502 sigue con su tratamiento de 8 bits por vez.

El divisor se sitúa en la dirección etiquetada con DIVSR, el dividendo en DVDND, el cociente (inglés, *quotient*) en QUOT, y el resto *(remainder)* en RMNDR. A continuación pasaremos, sin más, a ofrecerle el programa en assembly, tanto para el Z80 como para el 6502.

Observe cómo en ambos casos el divisor resta al dividendo parcial y, si el resultado es negativo, cómo se recupera este dividendo sumándole de nuevo el divisor. En el 6502 es notable el empleo que se hace del registro de estado del procesador (PSR) tras la resta del divisor: hay un desplazamiento rotatorio del flag de arrastre hacia el cociente, sin llegar a perder su estado, pues se necesita para indicar además el resultado de la resta. Por eso es depositado el PSR en la pila antes de la rotación, para ser recuperado posteriormente, devolviendo al flag el estado que tomó después de la resta.

Concluimos así el análisis de las cuatro reglas aritméticas. Usted habrá comprobado su interés pedagógico como medio de estudio, pues nos revela cómo trabaja la máquina, pero no vale la pena entretenerse mucho más en ellas. Entre otras razones, porque los programadores hace tiempo que han puesto a nuestra disposición, tanto en revistas como en libros de texto las diversas rutinas que se suelen usar. Si, más adelante, necesitáramos alguna de ellas, se la proporcionaremos o la expondremos en algún ejercicio.

DIVISIÓN DE 16 BITS POR 8 BITS

Z80			6502		
START	LD	A,(DIVSR)	START	LDA	#$00
	LD	D,A		STA	QUOT
	LD	E,$00		STA	RMNDR
	LD	HL,(DVDND)		LDX	#$08
	LD	B,$08		LDA	DVDHI
LOOP0	AND	A		SEC	
	SBC	HL,DE		SBC	DIVSR
	INC	HL	LOOP0	PHP	
	JP	P,POSRES		ROL	QUOT
NEGRES	ADD	HL,DE		ASL	DVDLO
	DEC	HL		ROL	A
POSRES	ADD	HL,HL		PLP	
	DJNZ	LOOP0		BCC	CONT1
	LD	(QUOT),HL		SBC	DIVSR
	RET			JMP	CONT2
DIVSR	DB	$F9	CONT1	ADC	DIVSR
DVDND	DW	$FDE8	CONT2	DEX	
QUOT	DB	$00		BNE	LOOP0
RMNDR	DB	$00		BCS	EXIT
				ADC	DIVSR
				CLC	
			EXIT	ROL	QUOT
				STA	RMNDR
				RTS	
			DIVSR	DB	$F9
			DVDLO	DB	$E8
			DVDHI	DB	$FD
			QUOT	DB	$00
			RMNDR	DB	$00

La pantalla

Hasta aquí nos hemos limitado a tratar la memoria RAM y la CPU como un sistema de cálculo y los resultados los íbamos dejando en la misma RAM, en algún lugar, disponibles para ser visualizados por medio de un programa monitor. Esto no basta, desde luego, pero tampoco había necesidad alguna de ocuparnos de la pantalla cuando lo que nos interesaban eran las subrutinas aritméticas.

Muchos micros cuentan con una imagen en memoria de la pantalla. Esto quiere decir que hay reservada un área de la RAM para retener la imagen de la pantalla. Cualquier visualización se realiza a base de puntos o *pixels*, que están o encendidos o apagados. Son, pues, representables por medio de unos (encendido) y ceros (apagado), en binario, y el contenido total de la pantalla puede considerarse como un "mapa" de puntos que se corresponden con los bits que componen los bytes pertenecientes a la RAM de pantalla. Lástima que esta técnica

empleada tanto por el BBC Micro como por el Spectrum y el Commodore 64 no sea utilizada por ninguno de ellos de manera inmediata. Lo más fácil para nosotros sería que cada fila de la pantalla quedara determinada por bytes correspondientes a pixels numerados consecutivamente de izquierda a derecha, siendo el byte más extremo por la izquierda el que sigue al byte extremo derecho de la fila anterior. Pero esto no ocurre así, por varias razones. Lo que significa que debemos examinar cada máquina por separado.

El Spectrum funciona siempre en alta resolución y dispone de un área de la memoria para la imagen de la pantalla. Esta correspondencia memoria-para-pantalla con pantalla es compleja, pues la pantalla se divide en tres bloques horizontales de ocho filas de PRINT cada uno, y a su vez cada una de estas filas se divide en otras ocho filas de pixels. Dentro de ellas el direccionamiento de los bytes que las componen es secuencial, pero no así de una fila a otra. El BBC Micro y el Commodore 64 no siguen este esquema, pero resultan tanto o más intrincados. Pensamos que la mejor manera de entenderlos es estudiar de momento sólo la forma en que hacen aparecer en pantalla los caracteres ASCII. Veámoslo.

Al tratarse de una tarea de uso frecuente, sus correspondientes rutinas en lenguaje máquina se hallan en la ROM. Contando con una descripción detallada de cómo operan, tales rutinas pueden ser llamadas desde nuestros programas en assembly. Necesitamos conocer tan solo la dirección de llamada, los registros de comunicación y cualquier otro requisito previo.

El Spectrum prescinde de todo paso preliminar: el acumulador hace de registro comunicador, y allí debemos colocar el código ASCII del carácter y visualizar. Basta después con usar la instrucción RST $10 para obtener en pantalla en el lugar indicado por el cursor, el carácter correspondiente a dicho código. Algo así es lo que exigen los otros dos sistemas, sólo que el opcode RST (ReSTart: volver al inicio) es una instrucción típica del Z80; se trata de una instrucción bifurcadora relacionada con la página cero y, por ello, implementada con un solo byte: admite uno de los ocho operandos siguientes: $00, $08, $10, $18, $20, $28, $30 y $38. En algún sitio de la página cero ha de hallarse la dirección de inicio de una rutina ROM, y a esta dirección es hacia la que apuntarán los posibles operandos. Tales rutinas toman (o llevan) caracteres o (o de) la pantalla, y no son llamadas directamente por su dirección de comienzo sino a través de la instrucción RST. Esto se debe, en parte, a razones de rapidez (pues, aunque sólo la CPU note la diferencia, es más rápido usar RST que CALL); y en parte, previendo la portabilidad del programa. Si todo programador del Z80 sabe que RST $10, por ejemplo, está llamando a la rutina PRINT, ninguno de ellos se molestará en saber dónde situó el ingeniero de una máquina que incorpore el Z80 esta rutina, por lo que dicho ingeniero podrá colocarla donde más le interese, con tal de que la página cero quede dispuesta de manera que las posiciones de RST conduzcan el programa hasta las direcciones de las rutinas convenidas. El procedimiento es muy semejante para el BBC Micro. El carácter cuyo código se ha colocado en el acumulador aparecerá en pantalla, en la posición que indica el cursor, mediante la instrucción JSR $FFEE. Estamos ante la conocida rutina OSWRCH, perfectamente descrita en la *Guía avanzada del usuario*.

El Commodore 64 también sigue este esquema. Para visualizar el carácter cuyo código ASCII está en el acumulador se emplea JSR $FFD2. Es la rutina CHKOUT, y su descripción se encuentra en la *Guía de referencia del programador*.

Éste es, pues, el esquema general para el empleo de las rutinas de la ROM, que a la vez nos ilustra el funcionamiento de los registros de comunicación. Tiene doble sentido toda comunicación entre el programa que llama a una rutina y la rutina llamada. Por ejemplo, una rutina que sirve para tomar información externa (rutina input) toma dicha información desde el dispositivo externo y lo lleva a la CPU siempre a través del acumulador. Y si falla el proceso, la misma subrutina se sirve de uno de los registros para devolver un código de error. Todos estos protocolos se explican exhaustivamente en muchas obras de referencia hoy en el mercado, dedicadas a exponer las características específicas de cada máquina.

En capítulos venideros trataremos este tema de la recogida de información desde el teclado u otro periférico y estudiaremos igualmente el trazado en alta resolución desde lenguaje máquina. Resumiremos a continuación el camino que ya hemos recorrido hasta este momento en nuestro curso de lenguaje máquina.

En resumen

Comenzamos el curso dando una idea general del lenguaje máquina y tratando de quitarle la mística que habitualmente le envuelve. Por eso insistimos en considerarlo un lenguaje más entre los utilizados por nosotros y por las máquinas. Vimos cómo una misma secuencia de bytes dentro de la RAM podía representar un string de caracteres ASCII en un momento dado y a renglón seguido ser interpretados como una sentencia de BASIC, o bien una serie de direcciones de dos bytes, o bien una secuencia de instrucciones en lenguaje máquina. Por poco que se entretuviera practicando con un programa monitor de lenguaje máquina, éste ya le habrá evidenciado que una secuencia de bytes pueden ser "desensamblados" como tres secuencias de instrucciones tan diferentes como válidas, a poco que a usted le de por comenzar a partir del primero, segundo o tercer byte de la secuencia. No hay nada en este lenguaje que impida tal hecho, y, además, la misma CPU no es capaz de indicarle si lo que está ejecutando es lo que usted escribió o bien una versión embarullada que se metió de rondón en la memoria.

Consideramos después la distribución de la memoria y los modos convencionales de direccionamiento. Para entender todo esto no tuvimos más remedio que estudiar la aritmética en sistema binario, lo que nos permitió conocer más a fondo la CPU: vimos que los procesadores de sólo ocho bits, salvo algunas excepciones, no nos permitían más que el tratamiento de un byte por vez (o sea, de los números decimales comprendidos entre el 0 y el 255). Pronto nos dimos cuenta de las limitaciones del sistema decimal dentro del lenguaje assembly, en comparación con el más apropiado sistema binario. Así, tratando el tema de la memoria distribuida

por páginas (paginación) vimos que la numeración de éstas y su tamaño debían estar en función del número-base, el dos, y sus potencias. Dos elevado a ocho es igual a 256, o, por decirlo en otras palabras, el número "mágico" en el sistema de un procesador de ocho bits.

Otro descubrimiento inmediato fue lo difícil que resultaba operar continuamente en binario, cosa que arreglamos pasándonos a los números hexadecimales (numeración en base 16). Observamos como un byte, que son ocho bits, podía escribirse con sólo dos dígitos "hexas" (como familiarmente les llamamos), desde el $00 al $FF, uno de los cuales se corresponde con los cuatro primeros bits, y el otro, por su parte, lo hace con los cuatro restantes bits de un byte.

Después examinamos con todo detalle cómo son almacenados los programas en BASIC dentro del área para programas de la memoria. Una buena parte del sistema operativo (SO) se pudo entender gracias al artificio de encerrar en un solo byte toda una orden que literalmente requiere varios (en inglés este byte es conocido por *token*). Tratamos incluso las marcas de fin de línea, viendo cómo el intérprete se las ingeniaba para decidir dónde acaba y dónde empieza un fragmento de lenguaje. Por su parte, el direccionamiento de enlace típico del Commodore nos presentó otro convencionalismo más: el direccionamiento byte inferior-byte superior (que llamamos *lo-hi*, por el inglés *low-high*) y el direccionamiento indirecto.

A partir de aquí ya estuvimos a punto para encarar el lenguaje assembly. Comenzamos por las operaciones más primitivas de la CPU, originadas por opcodes de ocho bits que son las instrucciones del programa. De esta primera exposición fue fácil pasar al concepto de la mnemotécnica, base del assembly. Y así nos resultó claro que escribir código máquina, o assembly o BASIC era programar y lo que cuenta es saber resolver con lógica un problema antes de darle forma codificada a la solución. Éste fue el tema central del curso. Aunque la oscuridad de algunos conceptos del assembly nos obligó a prestar más atención en aclarar las numerosas confusiones que en la mayoría de los casos generan los lenguajes de bajo nivel en quienes abordan su estudio por vez primera.

Por eso nos extendimos practicando con programas de carga, traslado y manipulación de información útiles dentro de muchos programas en BASIC. Estudiamos también las variables del sistema y los punteros de que disponen el BBC Micro, el Spectrum y el Commodore 64, aprendiendo a "robar" espacio del mismo BASIC.

Examinamos la arquitectura de pequeños sistemas computerizados, las CPU del Z80 y del 6502, pasando ya a escribir programas en assembly que manejaban la memoria y el acumulador. Aquí es donde hablamos de directrices o pseudo-opcodes del ensamblador, que nos acercaban más a la realidad práctica, aun a costa de alejarnos del lenguaje máquina, el ensamblaje manual y la fatigosa programación a bajo nivel.

Necesitábamos en ese momento comprender la construcción lógica de un lenguaje de programación, por lo que presentamos el registro de estado del procesador (el PSR: inglés *Processor Status Register*). La instrucción ADC (sumar con arrastre) que empleamos nos sirvió para explicar cómo este dispositivo servía de registro de los resultados de las operaciones de la CPU. Vimos que el indicador o flag de arrastre era en las operaciones aritméticas una pieza indispensable. Desde este momento no dejamos de hablar del PSR y de las instrucciones que se referían a él.

Echamos una breve ojeada a los diversos modos de direccionamiento, subrayando el direccionamiento indexado por su importancia a la hora de tratar bucles, listas y tablas. Comenzamos entonces a echar de menos las instrucciones capaces de cambiar el flujo de control del programa y así, mientras seguíamos viendo las posibilidades del direccionamiento indexado y del indirecto, iniciamos el estudio de las instrucciones de bifurcación condicionada. Con éstas y con las cuatro reglas aritméticas junto con las estructuras matriciales, observamos que teníamos ya vertebrado cualquier lenguaje de programación. Sólo nos faltaba ahora darle forma por medio de una investigación y de unas prácticas sistemáticas.

El último aspecto del sistema operativo que debíamos estudiar era la pila, y lo hicimos examinando a la vez la manera cómo se llama en assembly a una rutina y cómo se efectúa el retorno (inglés: *call* y *return*). Vimos para qué servía la pila, cómo funciona y cómo usarla, al tiempo que ampliábamos nuestro acopio de trucos para programar en lenguaje máquina y obteníamos una visión más rica de los registros de la CPU y su capacidad para manejar la memoria y el microprocesador.

Las postreras lecciones, ya provistos de un buen conocimiento de la arquitectura del microprocesador y de un amplio vocabulario de instrucciones opcodes, fueron dedicadas al estudio de las operaciones aritméticas en binario. No nos arredramos ni ante un concepto tan sutil como el complemento a dos, ni tampoco ante operaciones que resultan tan difíciles de explicar como la resta, la multiplicación o la división.

¿Qué nos queda por ver todavía? Ilustrar prácticamente las posibilidades de la programación en lenguaje máquina, a través de tareas específicas que confiaremos tanto a los procesadores que hasta aquí han representado nuestros puntos de apoyo (es decir, el Z80 y el 6502) como a algunos otros, como pueden ser el 6809 CPU, por ejemplo, que utilizan el Dragon 32 y 64.

Soluciones a los ejercicios de p. 779

1) La solución de más rápida ejecución es sin duda una rutina que se escriba específicamente para multiplicandos de 16 bits, siguiendo la pauta de la rutina para 8 bits del capítulo anterior. Por otra parte, si usted separa la multiplicación con 16 bits en dos multiplicaciones de 8 bits cada una (al multiplicador del byte *lo* deberá seguir el del byte *hi*), entonces le bastará con hacer una llamada *(call)* por dos veces a esa rutina para 8 bits de que dispone, previendo la posibilidad de un arrastre procedente del byte *lo* y almacenando después el resultado en los bytes reservados al producto.

2) Una rutina multiplicadora basada en la adición reiterada consta sencillamente de un bucle cuyo contador vale lo que el multiplicador; cada vez que se ejecuta dicho bucle, el multiplicando se va agregando al producto.

ROR — ROTAR A DERECHA — 6502

Registro -6A (1 byte)

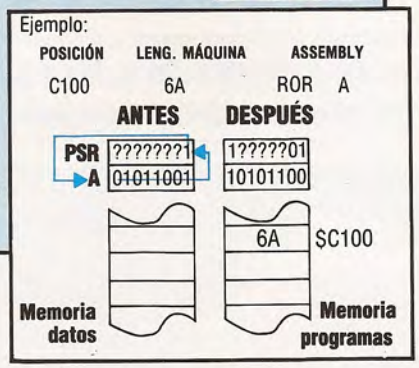
El contenido del byte gira un byte a la derecha por medio del flag de arrastre.

EFECTO EN EL PSR

	S	V		B	D	I	Z	C
MSB	X						X	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	6A	ROR A



RR — ROTAR A DERECHA — Z80

Registro -CB (2 bytes)

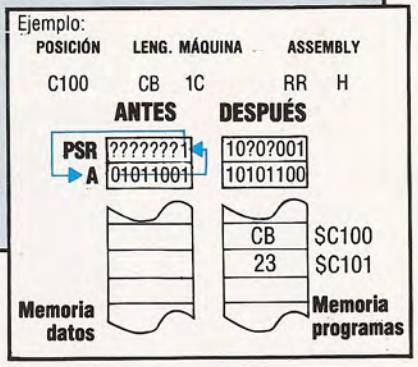
El contenido del byte gira un byte a la derecha por medio del flag de arrastre.

EFECTO EN EL PSR

	S	Z		H		V	N	C
MSB	X	X		0		X	0	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	CB 1C	RR H



SBC — RESTAR CON ARRASTRE — 6502

Absoluto -ED (3 bytes)

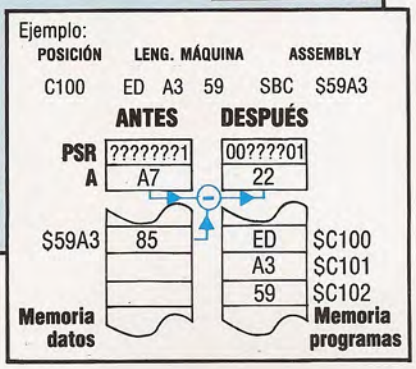
El contenido de la posición de memoria es restado del acumulador, y el flag de arrastre indica si hay que "quitar uno". *(borrow)*.

EFECTO EN EL PSR

	S	V		B	D	I	Z	C
MSB	X	X					X	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	ED A3 59	SBC $59A3



SBC — RESTAR CON ARRASTRE — Z80

Inmediato -DE (2 bytes)

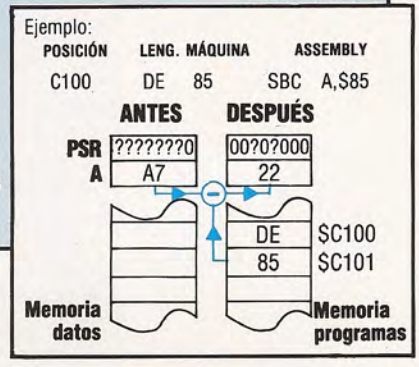
El contenido del byte que sigue al opcode es restado del acumulador.

EFECTO EN EL PSR

	S	Z		H		V	N	C
MSB	X	X		X			X	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	DE 85	SBC A,$85



ASL — DESPLAZ. ARITMÉTICO A IZQ. — 6502

Registro -0A (1 byte)

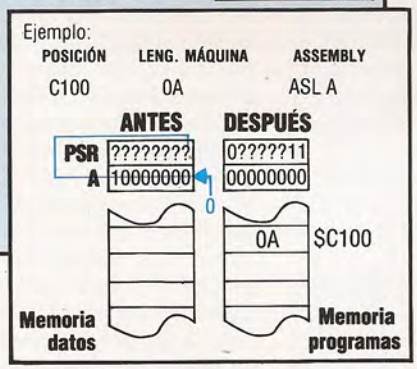
Desplaza el contenido del byte, por medio del flag de arrastre, un bit a la izquierda, poniendo a cero el bit inferior.

EFECTO EN EL PSR

	S	V		B	D	I	Z	C
MSB	X						X	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	0A	ASL A



SLA — DESPLAZ. ARITMÉTICO A IZQ. — Z80

Registro -CB (2 bytes)

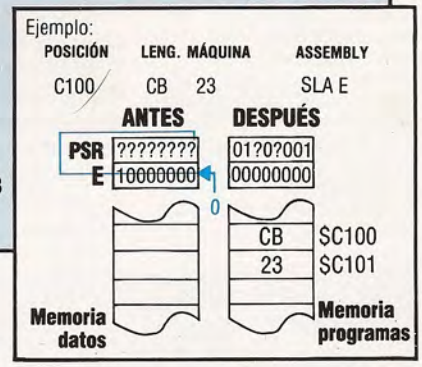
Desplaza el contenido del byte, por medio del flag de arrastre, un bit a la izquierda, poniendo a cero el bit inferior.

EFECTO EN EL PSR

	S	Z		H		V	N	C
MSB	X	X		0		X	0	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	CB 23	SLA E



CMP — COMPARAR ACUMULADOR — 6502

Absoluto -CD (3 bytes)

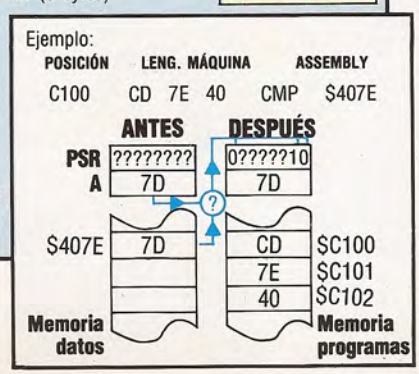
Compara el contenido de la posición de memoria con el del acumulador.

EFECTO EN EL PSR

	S	V		B	D	I	Z	C
MSB	X						X	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	CD 7E 40	CMP $407E



CP — COMPARAR ACUMULADOR — Z80

Inmediato -FE (2 bytes)

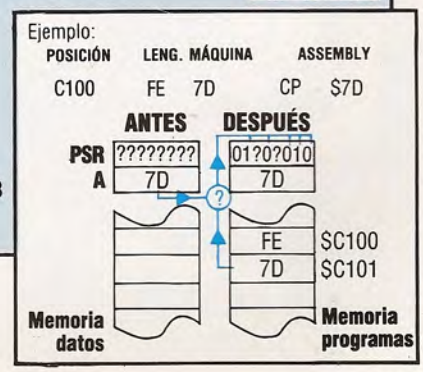
Compara el contenido del byte que sigue al opcode con el del acumulador.

EFECTO EN EL PSR

	S	Z		H		V	N	C
MSB	X	X		X		X	1	X LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
C100	FE 7D	CP $7D



Porvenir incierto

Los problemas financieros de Dragon Data han sembrado dudas acerca del porvenir de la empresa y la comercialización de sus productos

Dragon Data se estableció originalmente como una subsidiaria de Mettoy, fabricantes de juguetes, en 1981. La intención de Mettoy no era otra que aprovecharse del *boom* de los ordenadores personales, que entonces acababa de iniciarse en Gran Bretaña. Con la ayuda financiera de la Welsh Development Agency (Agencia Galesa para el Desarrollo), se montó una fábrica en Swansea, y el Dragon 32 hizo su aparición en agosto de 1982.

La empresa optó por el microprosador 6809 de Motorola, en lugar del Z80 o del 6502, usados por la mayoría de los otros fabricantes de ordenadores personales. El sistema de circuitos del Dragon seguía el trazado recomendado por Motorola, lo que suscitó acusaciones en el sentido de que Dragon Data había basado su diseño en el ordenador Tandy Color, otro modelo que utilizaba el formato Motorola. Un efecto colateral de esto fue que los usuarios descubrieron enseguida que parte del software escrito para el CoCo se podía ejecutar también en la máquina galesa.

Los principales puntos para la venta del Dragon 32 fueron su BASIC Microsoft (la versión de BASIC más ampliamente utilizada) y su teclado tipo máquina de escribir de tamaño natural. En el momento en que se lanzó la máquina, en Gran Bretaña, en el sector del mercado por debajo de las 200 libras, el teclado del Dragon sólo era equiparable al del Vic-20. La estrategia de marketing de Dragon Data también desempeñó un gran papel en el éxito que obtuvo la máquina: en los meses que precedieron a la Navidad de 1982, los suministros tanto del ZX Spectrum como del BBC Micro eran escasos, y el Commodore 64 aún estaba por lanzar. Las existencias del Dragon 32 eran muy grandes y para comienzos de 1983 la empresa había vendido 32 000 máquinas. Ello se debió, en parte, a la conexión con Mettoy; las cadenas de tiendas comerciales más importantes, que siempre habían tenido grandes existencias de los juguetes de la empresa, se sentían felices al poder vender el nuevo ordenador.

Sin embargo, en el verano de 1983 Dragon Data se encontró inmersa en un serio problema financiero. La empresa estaba en plena expansión cuando Mettoy entró en bancarrota, sembrando dudas acerca del futuro de su subsidiaria galesa. Finalmente Dragon pudo salvarse gracias a un consorcio de empresas encabezado por Prutec, la rama de inversiones en alta tecnología de la gigantesca empresa de seguros Prudential. Se reunió una suma de 2,5 millones de libras y la firma contrató los servicios de un nuevo director gerente, Brian Moore, antiguo ejecutivo de GEC. Estos cambios permitieron que Dragon Data superara sus problemas de movimiento de caja, invirtiera en una nueva factoría en Port Talbot y continuara desarrollando el Dragon 64 y la unidad de disco.

El Dragon 64 tiene 64 Kbytes de RAM, un teclado perfeccionado y una interface en serie RS232C. La unidad de disco utiliza discos flexibles estándar de 5 ¼ pulgadas que operan bajo el Dragon DOS, que puede ser utilizado tanto por el modelo 32 como por el 64. Para el Dragon 64 también hay disponible una versión del eficaz sistema OS9.

Pero una sombra se cernió sobre todos los planes de Dragon en junio de 1984, cuando Prutec y la Welsch Develoment Agency se negaron a aportar más capital y la firma se vio enfrentada a profundas dificultades financieras. Había pocas perspectivas de encontrar un comprador para la empresa. En aquellos momentos ya había tres nuevas máquinas dentro de los planes para 1984, además de otros productos relacionados con los ordenadores. Uno de los micros planificados iba a responder al estándar MSX que introducían los japoneses (véase p. 621). Pero el dragón galés se ha convertido en una especie en peligro de extinción.

Nace un dragón
Dragon Data es poco habitual entre las empresas británicas de ordenadores personales por el hecho de que fabrica sus propias máquinas, cuando la mayoría de las empresas, como Sinclair y Acorn, subcontratan la producción de sus ordenadores. Recientemente Dragon ha construido una nueva fábrica en Port Talbot (West Glamorgan, en el País de Gales), que vemos en la fotografía

Expectativas frustradas
Richard Wadman, director de marketing de Dragon Data, tenía planes para vender toda una nueva gama de ordenadores Dragon, cuando la empresa se vio súbitamente en dificultades financieras

Líneas de transmisión

En este capítulo veremos cómo se organiza una red de información utilizando un grupo de ordenadores personales

Interface 1
Las redes no implican necesariamente kilómetros de cables y equipos costosos: los Spectrum equipados con la Interface 1 se pueden comunicar entre sí y compartir microdrives y una impresora, conformando un sistema de precio muy reducido

Paul Chave

Una red es un sistema de ordenadores enlazados entre sí para compartir datos y equipos. Sin embargo, cada ordenador posee su propio sistema operativo y sus propios "protocolos" (procedimientos, reglas de formateado, etc.) para la comunicación con el mundo exterior. Debido a estos problemas de compatibilidad, las estaciones individuales o *nodos* de una red deben estar siempre constituidos por ordenadores similares: todos Spectrum, o todos Apple, por ejemplo.

Para poder realizar el análisis, vamos a suponer que hay un grupo de personas que poseen cinco ordenadores y desean conectarlos a una misma impresora. Nuestro grupo ha de ser capaz de enviarle información a la impresora desde cualquiera de los nodos. ¿Qué sucede si dos o más nodos tienen texto para imprimir al mismo tiempo? Y, aún más importante, ¿qué pasa si el nodo 3 tiene texto para imprimir pero necesita seguir trabajando mientras la impresora está en funcionamiento? Para resolver estos problemas debemos instalar un sexto ordenador, llamado *manejador de impresión*. El papel de esta máquina consiste en controlar el flujo de datos hacia la impresora y, por lo tanto, no se puede destinar a ningún otro fin. El manejador de impresión almacenará los documentos por orden de prioridad. Una vez que un nodo ha enviado su texto al manejador de impresión, el nodo puede realizar otros trabajos.

El empleo de una máquina exclusiva que actúe como manejador es esencial para una red, porque es a través del manejador que se puede compartir la información. Además de un manejador de impresión, algunas aplicaciones de red requieren un *manejador de archivos* para manipular las unidades de disco compartidas y controlar el flujo de información de nodo a nodo.

El siguiente paso consiste en crear un enlace entre los ordenadores-nodos. Esto se hace tendiendo un cable (ya sea un cable de dos hilos trenzados o un cable coaxial) de máquina a máquina. Aunque existen varias disposiciones posibles para los nodos y la estación del manejador —configuraciones en forma de estrella *(star)* o de anillo *(ring)*—, el concepto es esencialmente el mismo, de modo que describiremos el proceso de manera general. Para efectuar la conexión se suele requerir una interface especial para red para cada nodo. Dicha interface puede ser una conexión RS232 simple o una placa de circuito impreso conectable. Además, la estación del manejador requiere una unidad de almacenamiento con capacidad suficiente para manipular toda la carga de trabajo. La estación del manejador requiere, asimismo, suficiente RAM para administrar la red.

Es el software que utiliza un ordenador lo que determina que una máquina realice correctamente su función, y esto es particularmente cierto aplicado a redes. Antes que nada debe haber una "capa" de software sobre el sistema operativo de la máquina que establezca la conexión entre cada nodo y la red. Conocida como *software para red*, esta capa

coloca la estación del manejador al mando de las operaciones especializadas de manejo de archivos y el manejo de impresión, y también le proporciona el control sobre el flujo de datos dentro de la red. Además de establecer esta cadena de mando, el software para red informa al ordenador sobre cada nodo que está en una red, que hay un manejador instalador y cuántos otros nodos hay. Por último, el software para red les proporciona a los ordenadores de los nodos el protocolo que éstos precisan indefectiblemente al objeto de poder comunicarse con el resto del sistema.

Una vez establecida la capa de la red, los nodos individuales deben tener un programa, o un conjunto de programas, para que sus propias aplicaciones reconozcan la red y sepan cómo comunicarse con la misma. Éste es un software escrito especialmente para aplicaciones de red, y se puede ejecutar desde una cassette o una unidad de disco en el nodo, o a través del manejador de archivos. El software sólo es tan complicado como la operación. Si el nodo 1 de nuestra red está ejecutando un programa de tratamiento de textos y está enviando los resultados a la impresora independientemente de los otros nodos, la única modificación requerida para un procesador de textos estándar es la inclusión de protocolos para red. Por otra parte, si los nodos 2 y 4 han de utilizar los mismos datos, y ser capaces de ver los resultados de cada uno, las cosas se vuelven más complicadas. En este caso, el software de aplicaciones (ya sea un procesador de textos, hoja electrónica, base de datos o incluso un juego) y el hardware del sistema deben tener la capacidad de trabajar en multitarea. Dicho en otras palabras, la CPU ha de ser capaz de manejar más de una tarea al mismo tiempo, y debe tener, asimismo, la capacidad de gestionar las comunicaciones que reciba procedentes de al menos otras dos CPU simultáneamente.

Tony Sleep

Experiencia compartida
Esta escuela se equipó con 16 micros BBC Micro con pantallas en color, una impresora, una unidad de disco doble y Econet (la LAN de Acorn para el BBC Micro), invirtiendo menos dinero del que hubiera costado instalar una unidad de disco y una impresora para cada ordenador. (Las siglas LAN corresponden a *Local Area Network*: red de área local.) La velocidad de la red es tal que cada puesto de trabajo parece tener el uso exclusivo de la unidad de disco, incluso con 30 alumnos trabajando; los terminales disponen de una "cola de red" para usar la impresora. El Econet le permite al profesor de informática proporcionar experiencia práctica a todos sus alumnos regularmente; la comunicación interterminal es de gran valor cuando la red se utiliza para la enseñanza de asignaturas

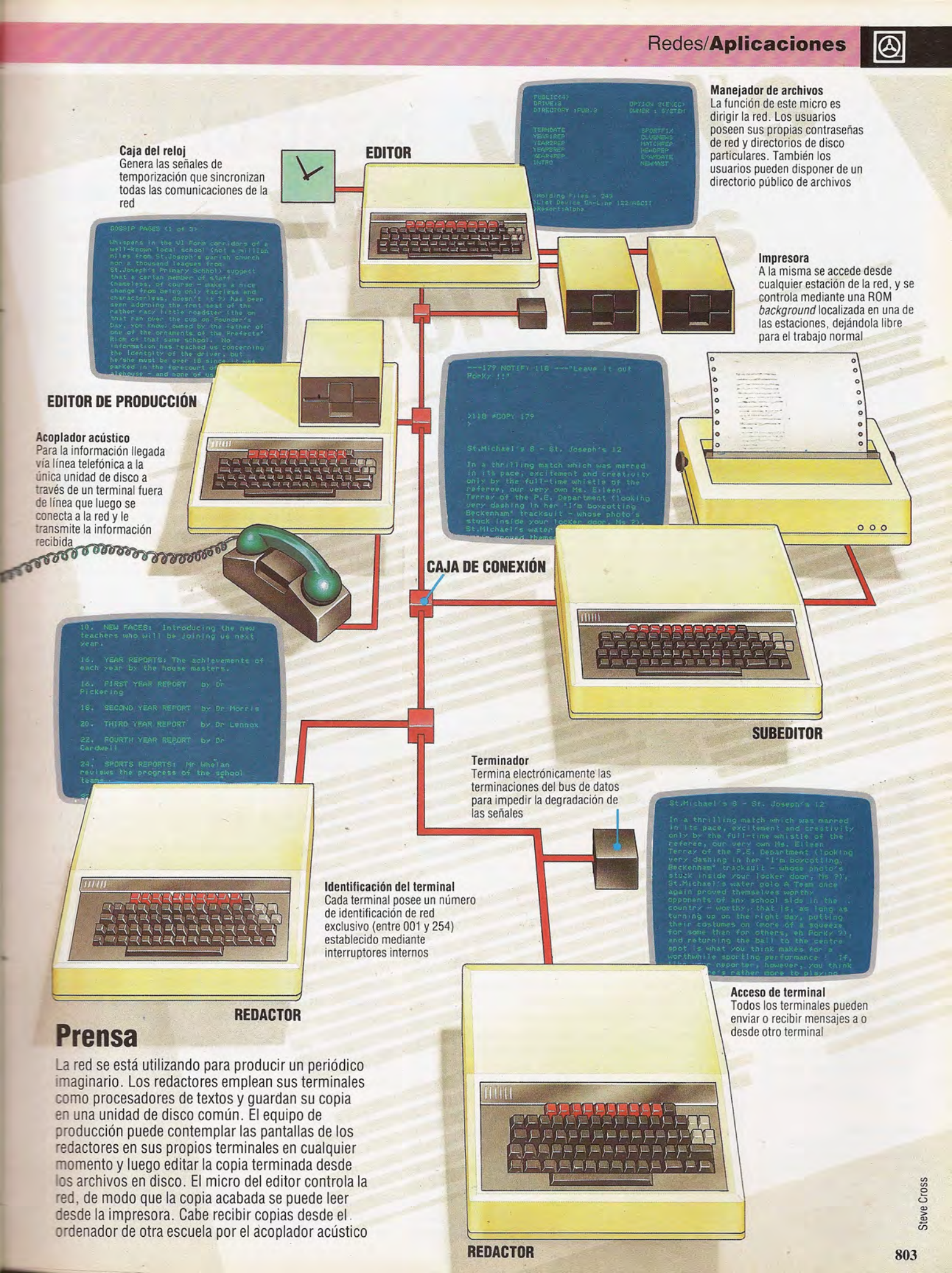
La empresa que se ha mostrado más activa e interesada en poner las redes al alcance de los usuarios de ordenadores personales es, sin duda, Sinclair Research. La unidad para accesorios Interface 1 del Spectrum lleva incorporada una interface para red. Esta unidad se está vendiendo bien porque es necesaria para usar los microdrives con el Spectrum. Es probable que cuando se hayan vendido suficientes unidades de la Interface 1 se ponga en venta software apropiado para hacer uso de la interface para red.

El último micro de Sinclair, el QL, lleva incorporada como estándar una interface para red similar, que debería ser compatible con la versión para el Spectrum. Aunque esta interface es bastante primaria, la popularidad de estas máquinas debería hacer que valiera la pena producir software de redes para ellas. Los programas para juegos son los primeros y obvios candidatos. Más allá de ellos, las posibles aplicaciones para usuarios de ordenadores personales pueden considerarse, lamentablemente, más bien limitadas.

Antes de que la conexión de ordenadores en red sea auténticamente viable hay varios elementos que son necesarios. El más simple de todos, por supuesto, es que debe haber al menos dos micros a conectar entre sí. En segundo lugar, los micros deben estar lo bastante próximos el uno del otro para que se puedan unir entre sí mediante cables. Esto significa que han de estar en el mismo edifico. Por último, debe haber suficiente "tráfico" como para que se justifique el uso de una red. Esto significa que es necesario que existan usuarios que se intercambien datos muchas veces al día, o bien que deseen compartir equipos caros (tales como impresoras o unidades de disco) como fórmula para amortizar el costo de la red.

Si sólo se hubiera de enviar por la red una pequeña cantidad de datos, sería más fácil que un usuario se los pasara al otro en cinta o disco. Del mismo modo, si la red sólo está formada por unos pocos micros, sería más barato suministrarle a cada uno de ellos su propia impresora y unidad de disco, en vez de invertir en el costo adicional de ordenadores para la gestión de la red.

Por consiguiente, aparte de los juegos, los únicos casos prácticos para conectar micros personales en red se dan en pequeñas empresas y en los centros de enseñanza. El Sinclair QL ha hecho que el costo de las redes disminuya hasta un nivel en el que resulta rentable proporcionarle un ordenador al personal que no necesita utilizar los ordenadores de forma intensiva. Son muchas las personas que antes de que haya transcurrido mucho tiempo se encontrarán con ordenadores conectados en red encima del escritorio de su oficina.



Prensa

La red se está utilizando para producir un periódico imaginario. Los redactores emplean sus terminales como procesadores de textos y guardan su copia en una unidad de disco común. El equipo de producción puede contemplar las pantallas de los redactores en sus propios terminales en cualquier momento y luego editar la copia terminada desde los archivos en disco. El micro del editor controla la red, de modo que la copia acabada se puede leer desde la impresora. Cabe recibir copias desde el ordenador de otra escuela por el acoplador acústico

Steve Cross

Todo bajo control

Veamos cómo se obtienen efectos especiales sobre papel y cómo se realiza la conexión entre ordenador e impresora

ÉLITE

```
ABCDEFGHIJKLMNOP
QRSTUVWXYZabcdef
ghijklmnopqrstuv
wxyz0123456789 !
"£$%&'()*+,-./:;
<=>?@[\]^_`{|}~
```

PICA

```
ABCDEFGHIJKLM
NOPQRSTUVWXYZ
abcdefghijklm
nopqrstuvwxyz
0123456789 !"
£$%&'()*+,-./
:;<=>?@[\]^_`
{|}~
```

CURSIVA PICA ENFATIZADA

```
ABCDEFGHIJKLM
NOPQRSTUVWXYZ
abcdefghijklm
nopqrstuvwxyz
0123456789 !"
#$%&'()*+,-./
:;<=>?@[¥]^_`
{/}~
```

ÉLITE ALARGADA

```
A B C D E F G H
I J K L M N O P
Q R S T U V W X
Y Z a b c d e f
g h i j k l m n
o p q r s t u v
w x y z 0 1 2 3
4 5 6 7 8 9  !
" £ $ % & ' ( )
* + , - . / : ;
< = > ? @ [ \ ]
^ _ ` { | } ~
```

PICA CONDENSADA DE PERCUSIÓN DOBLE

```
ABCDEFGHIJKLMNOPQRSTUVW
XYZabcdefghijklmnopqrst
uvwxyz0123456789 !"£$%&
'()*+,-./:;<=>?@[\]^_`{
|}~
```

Sólo con puntos
Las impresoras matriciales ofrecen una gama de tipos de letras como pica, élite y cursiva, y diversos estilos de tipos, como condensados, alargados y enfatizados. Los ejemplos que vemos aquí fueron obtenidos con una Epson FX-80

Una impresora matricial puede hacer mucho más que producir simplemente listados de programas. Una rápida ojeada al manual de usuario de la impresora le enseñará que se pueden producir sobre el papel diversos "efectos especiales". Incluso las impresoras matriciales más baratas le permitirán modificar el tamaño de los caracteres impresos sobre el papel. Normalmente, el texto se imprime en 80 caracteres por línea, pero este número se puede aumentar seleccionando la modalidad "impresión condensada" (que utiliza caracteres más pequeños) o disminuir seleccionando "impresión alargada". De modo similar, se puede alterar el espaciado entre líneas. Un espaciado grande de cuatro líneas por pulgada, por ejemplo, se podría reducir a, digamos, ocho líneas por pulgada, lo que proporciona un listado de mayor densidad.

La impresora que vamos a examinar en profundidad en este capítulo, la Epson FX-80 es un buen ejemplo de máquina con una amplia gama de posibilidades de impresión. La modalidad "enfatizada", que imprime el texto en negrita, y la modalidad alternativa, que cambia de los tipos de letras normales a los caracteres en cursiva, son dos de sus facilidades estándar. Pero quizá su característica más interesante sea su capacidad para cambiar cualquiera de los caracteres almacenados en la memoria de la impresora, una facilidad sumamente útil para los alfabetos extranjeros o para la impresión de símbolos científicos. Antes de pasar a investigar cómo se producen estas configuraciones, consideremos cómo lleva a cabo una impresora la sencilla tarea de imprimir el listado de un programa.

La forma en que un ordenador le "habla" a una impresora varía de una máquina a otra. El Dragon, por ejemplo, utiliza una variante de la instrucción LIST (LLIST) para indicar a la impresora que produzca el listado de un programa. Otras máquinas exigen la apertura de "canales" o "corrientes" *(streams)* para tener acceso a la impresora. Como el procedimiento exacto varía tanto, lo mejor es consultar en el manual de usuario de su ordenador (es poco probable que en este caso el manual de la impresora sea de alguna utilidad).

Habiendo establecido la comunicación entre las dos máquinas, puede que su primer listado sea decepcionante. Los problemas más comunes son que todo el texto se imprima en una línea negra indescifrable, o que haya líneas en blanco entre cada línea del programa. La explicación para estos dos fallos radica en la diferencia entre un carácter de "salto de línea" y un carácter de "retorno de carro". Después de que su ordenador le envía una línea de texto a la impresora, también le hace llegar un carácter de retorno de carro, que mueve el cabezal de impresión otra vez al margen izquierdo, listo para imprimir una nueva línea. Algunos ordenadores también envían un carácter de salto de línea para mover el papel una línea hacia arriba; otros dan por sentado que es la propia impresora la que hace esto de forma automática. Para complicar las cosas todavía más, la mayoría de las impresoras posee un interruptor interno que decide si éstas generan sus propios saltos de línea o no. Si se plantea alguno de estos problemas, busque este interruptor (consultando el manual de la impresora) y colóquelo en la posición alternativa.

Aparte de producir listados de programas, una impresora también se puede utilizar como un dispositivo de salida: en vez de visualizar los caracteres en la pantalla, éstos se pueden imprimir en papel. Nuevamente, el procedimiento exacto para hacer esto varía de un ordenador a otro; la instrucción estándar en BASIC es LPRINT y ésta la emplean el Spectrum y el Oric. En el Commodore 64, OPEN1,4 seguida de PRINT#1,"HOLA" imprimirá la palabra "HOLA". En un micro Dragon, la misma tarea se realiza utilizando PRINT#−2,"HOLA". El BBC Micro emplea VDU2 seguido de PRINT "HOLA" y la instrucción VDU3. Los ejemplos de programación que proporcionamos aquí utilizan LPRINT, de modo que quizá deba ser modificada para su máquina.

Etiquetas de direcciones

```
10 LPRINT "SR. JOSE GOMEZ"
20 LPRINT "C/ DEL PEZ, 9"
30 LPRINT "CIUDAD"
40 LPRINT "ABC 123"
50 FOR I = 1 TO 7
60 LPRINT
70 NEXT I
100 GOTO 10
```

Este listado es un programa sencillo para producir etiquetas de direcciones. Éstas se pueden comprar en un rollo con agujeros para rueda dentada en ambos lados, de modo que se puedan utilizar con el arrastre de tracción de la impresora. Dado que no emplea códigos de control especiales, el programa funcionará en impresoras de cualquier marca. Tal como está, imprime el mismo nombre y la misma dirección una y otra vez. Quizá usted desee modificarlo con el fin de que pueda entrar distintos nombres y direcciones, o incluso leerlos de un archivo de datos. El bucle FOR...NEXT entre las líneas 50 y 70 imprime siete líneas en blanco y sirve para posicionar el cabezal de impresión justo al comienzo de cada etiqueta. Tal vez sea necesario ajustar el número exacto de líneas en blanco para su máquina.

Nuestro programa es bastante adecuado simplemente para imprimir etiquetas, pero para imprimir algo más complicado, como una factura o un membrete, vamos a tener que aplicar algunos de los efectos especiales que hemos mencionado antes. Éstos se producen enviando a la impresora códigos de control además de los caracteres del texto.

Junto con tener un código para cada carácter del teclado, el juego de caracteres ASCII (véase p. 557) posee un grupo de caracteres "invisibles" que no imprimen nada ni en pantalla ni sobre papel. Son estos códigos los que se utilizan para activar los efectos especiales de la impresora: en el juego ASCII estándar hay cuatro códigos (17, 18, 19 y 20) que están reservados como órdenes para control de periféricos. Lamentablemente, el juego de caracteres ASCII no posee suficientes caracteres de control reservados para las setenta y pico posibilidades de una Epson FX-80, y para superar este inconveniente la mayoría de los efectos se producen enviando a la impresora *códigos de escape*. Éstos se componen de dos o más códigos de carácter, empezando con un carácter ESC (código ASCII 27). Por ejemplo, para activar la facilidad de espaciado proporcional en una Epson, se envía ESC-p, o sea, el carácter Escape seguido del carácter "p" minúscula.

En BASIC esto se escribe de la siguiente forma:

```
LPRINT CHR$(27);"p"
```

Normalmente el carácter EC no se puede generar mediante la pulsación de la tecla Escape en el teclado y, por consiguiente, se emplea la función CHR$.

En el BBC se utilizaría:

```
VDU2
VDU1,27,1,112
VDU3
```

La instrucción VDU2 activa la impresora; VDU1 significa "enviar el siguiente carácter sólo a la impresora" (PRINT enviaría el siguiente carácter ESC también a la pantalla, con resultados no deseados). VDU3 desactiva la impresora.

Estas secuencias de instrucciones se aplican sólo a la Epson FX-80. Si se enviara el mismo código a una impresora diferente, o bien no tendría ningún efecto, o realizaría algo inesperado, o haría que la impresora se "colgara" (es decir, se negara a responder al ordenador).

Creación de una factura

Nuestro segundo listado ilustra el uso de algunas de las facilidades de la Epson para crear el membrete de una factura como la que podría emplear un pequeño garaje. Los códigos que hemos aplicado aquí son los que se utilizan para la Epson FX-80. La gama de impresoras Epson es una de las más populares; tanto es así que otros fabricantes hacen modelos "compatibles con Epson". No obstante, si su impresora es incompatible, deberá modificar los códigos de control.

```
999 REM MEMBRETE DE FACTURA
1000 LPRINT CHR$(12)
1010 LPRINT CHR$(14);TAB(11);"MOTORES PLIM, S.A."
1020 LPRINT CHR$(13);CHR$(13)
1030 LPRINT CHR$(27);"E";
1040 LPRINT TAB(36);
1050 LPRINT CHR$(27);"-";CHR$(1);
1060 LPRINT "FACTURA";
1070 LPRINT CHR$(27)"-";CHR$(0);
1080 LPRINT CHR$(27);"F";
1090 LPRINT CHR$(13);CHR$(13);CHR$(13)
1100 REM IMPRESION DETALLES FACTURA
```

Para empezar, la línea 1000 le envía a la impresora el carácter cuyo código es 12. Éste es el carácter de "salto de página", que le indica a la impresora que mueva el papel al comienzo de una nueva hoja. Luego tenemos el código ASCII 14; éste es el carácter denominado *shift out* (SO) y en la Epson hace que todo el texto subsiguiente se imprima en letras alargadas. En nuestro programa se utiliza para el membrete, obteniéndose el nombre del garaje en letras grandes. La función TAB se emplea para centrar el membrete.

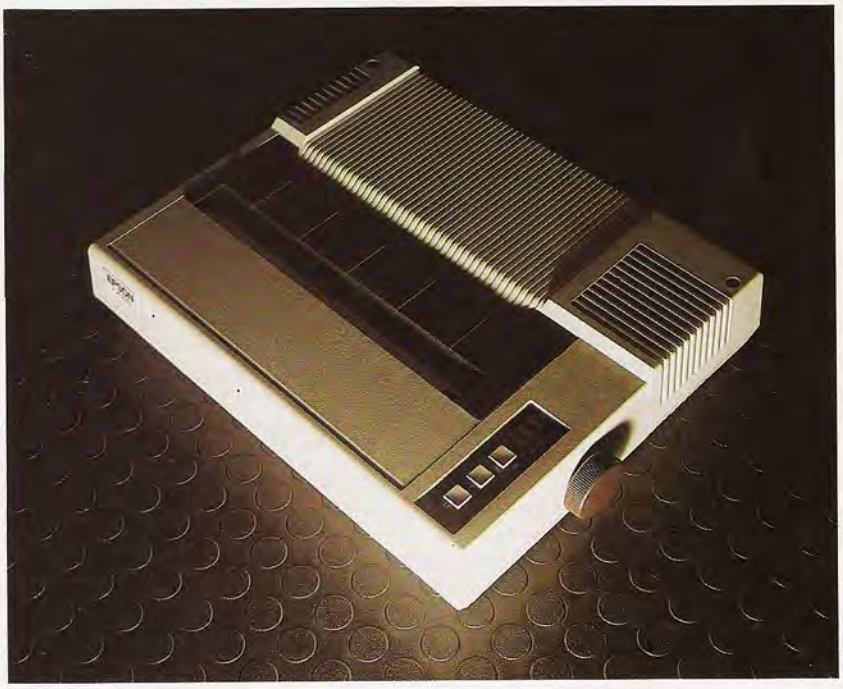
CHR$(13) es el carácter de retorno de carro, que al ser enviado a la impresora produce una línea en blanco. Entre las líneas 1020 y 1090 se emplea varias veces para espaciar la parte superior de la factura ESC-E. En la línea 1030, activa la modalidad enfatizada, y todo el texto a continuación se imprime en negrita (que se obtiene imprimiendo varias veces la misma letra). La línea 1050 activa la característica de "subrayado" y la línea 1070 la desactiva, después de imprimir y subrayar la palabra "factura". ESC-F desactiva la modalidad enfatizada. El listado tendrá el siguiente aspecto:

```
MOTORES PLIM, S.A.

FACTURA
```

Aquí sólo hemos reseñado la parte inicial del programa; un programa de facturación completo incluirá instrucciones para imprimir los detalles del cliente: nombre, marca del coche, cantidad adeudada, etc. Estos detalles se habrán obtenido a partir de una serie de preguntas realizadas al principio del programa y las respuestas se habrán almacenado en variables dentro de éste.

Los dos programas que hemos ofrecido son ejemplos sencillos de los tipos de usos alternativos que se pueden dar a una impresora matricial. En la actualidad son muchas las personas que están explorando la utilización de una impresora para algo más que para hacer listados de programas. De hecho, programando la impresora, el usuario puede disfrutar tanto como cuando programa el propio ordenador.



Ian McKinnell

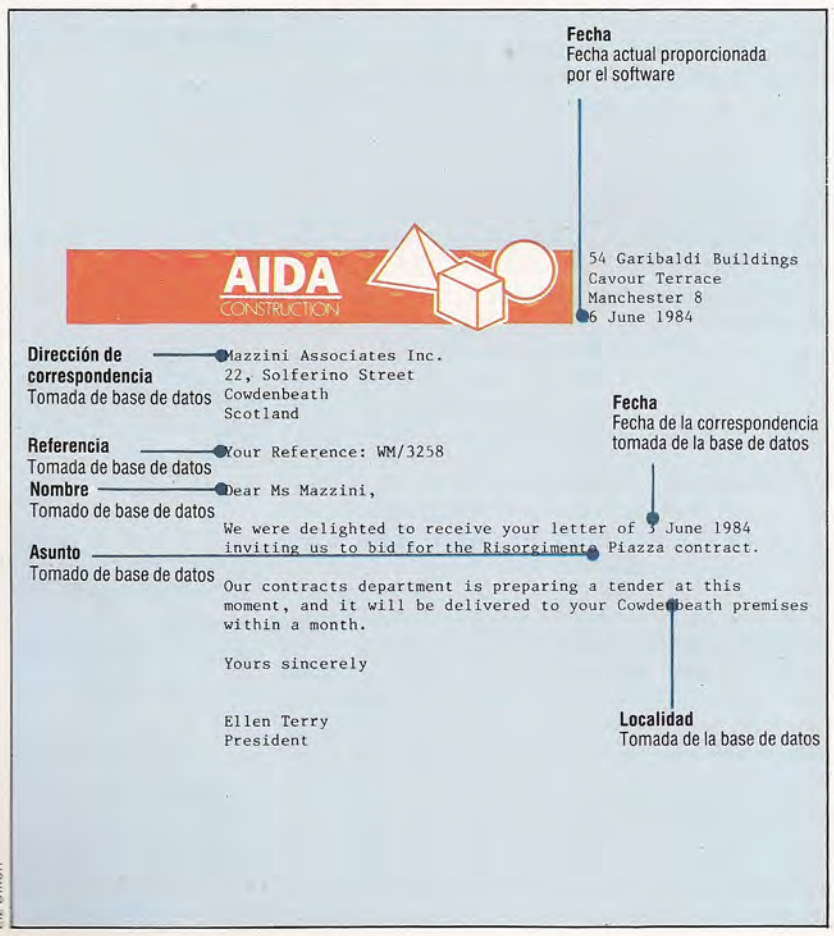
Epson FX-80
Aunque cara, la Epson FX-80 es una impresora popular entre los propietarios de micros personales y de gestión. La FX-80 tiene un cabezal de nueve agujas y una velocidad de impresión máxima de 160 caracteres por segundo. La mayoría de los paquetes de software (tratamiento de textos, p. ej.) admite impresoras Epson o tipo Epson

Entrega inmediata

Un sistema de correspondencia para un micro necesita una impresora, un paquete para tratamiento de textos y un programa de base de datos

El correo informatizado puede llegar muchísimo más allá de la simple impresión de etiquetas de dirección. En realidad, si todo lo que necesitáramos fuera alguna forma de colocar direcciones en sobres, el tratamiento de textos no sería necesario. Un paquete de base de datos sería suficiente, dado que preparar un archivo de nombres y direcciones es muy fácil en una base de datos estándar. A pesar de que inicialmente cada registro de nombre y dirección habría que entrarlo campo por campo, una vez que todo el sistema estuviera preparado podríamos utilizar el paquete de base de datos, una y otra vez, para generar todas las etiquetas precisas.

Dado que los paquetes de base de datos, por poco sofisticados que sean, poseen facilidades para informes, sería posible seleccionar las etiquetas a imprimir de acuerdo a cierto criterio: por ejemplo, todas las direcciones que lleven Barcelona en el campo de la ciudad. Éste se encargaría de seleccionar y crear las etiquetas para los sobres, pero aún dejaría en manos del usuario la realización de la otra mitad de un trabajo tedioso. Si, por ejemplo, se deseara enviar a un conjunto determinado de clientes una carta normalizada, también sería necesario digitar los nombres y las direcciones en el interior de cada una de las cartas.



Liz Dixon

Además, aunque cada cliente tuviera escrito su propio nombre y dirección en la cabecera de la carta, el texto de ésta permanecería anónimo e impersonal. No tiene mucho sentido, por ejemplo, dejar agujeros vacíos en el texto para rellenarlos luego con el nombre de cada cliente, porque los nombres tienen distintas longitudes. Se tendrían que dejar agujeros lo suficientemente largos como para dar cabida al nombre más largo que se deseara incluir, y la carta "personalizada" resultante tendría un aspecto desastroso. De modo que personalizar la carta implicaría volver a digitarla tantas veces como nombres de distinta longitud hubiera en la lista de correspondencia.

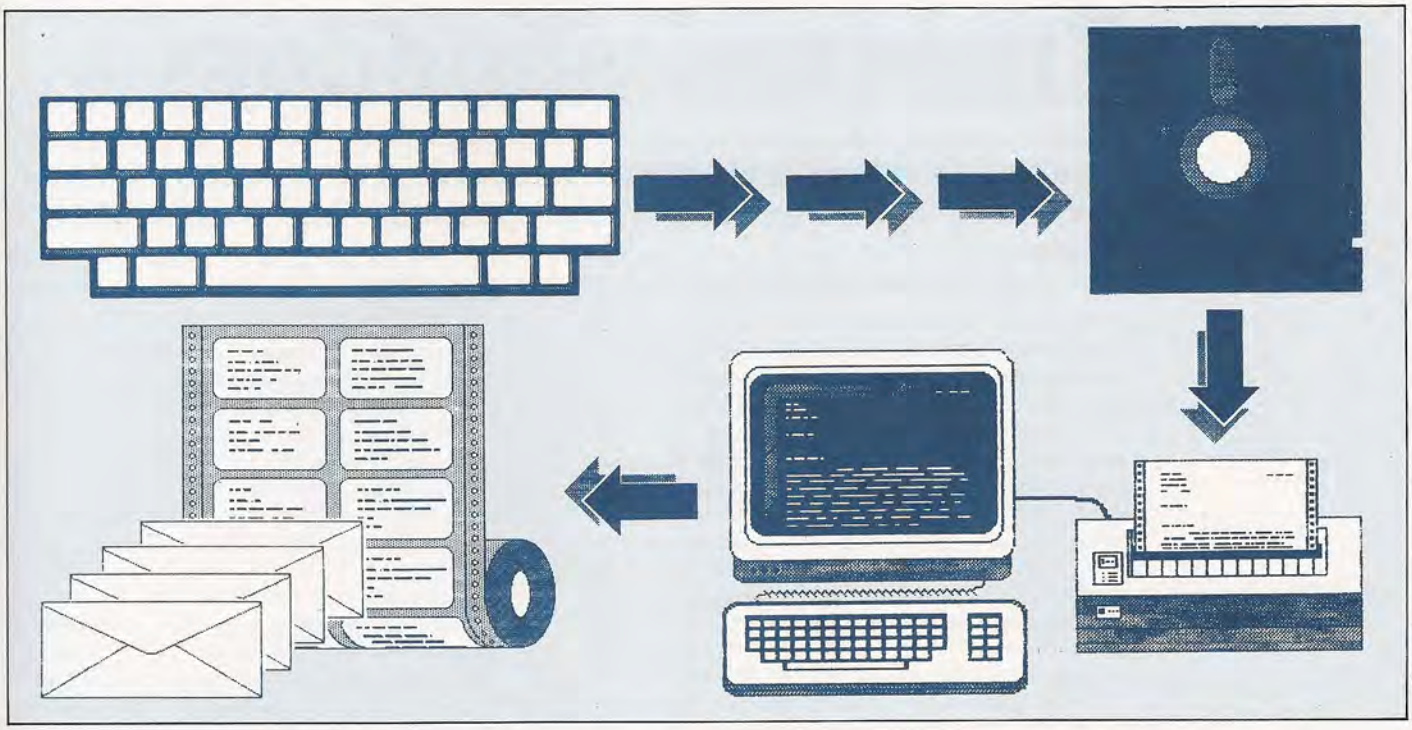
La solución ideal para este problema consiste en acomodar el sistema de modo que el usuario pueda componer una carta estándar empleando todas las facilidades de un procesador de textos, y después dejar que el ordenador extraiga automáticamente los detalles correspondientes a cada cliente de un archivo de base de datos, para agregarle a cada carta los detalles personalizados.

Existen varias maneras de hacer esto. Una de las más sencillas es la utilizada por dos paquetes basados en disco producidos por Acorn, denominados Memoplan y Fileplan, un procesador de textos y una base de datos, respectivamente.

Los paquetes no pueden producir cartas personalizadas independientemente y, por tanto, deben usarse juntos. Los detalles de nombre y dirección se deben preparar con el Fileplan, y la carta estándar (o formulario) se crea mediante el Memoplan. Cada campo del archivo de nombres y direcciones está numerado y en los puntos pertinentes de la carta estándar el usuario simplemente teclea el número del campo cuyo contenido necesita transferir a la carta. El ordenador trabajará entonces secuencialmente con la lista de correspondencia y en la carta estándar aparecerán los contenidos de los números de campo para los registros seleccionados.

El procesador de textos ajusta automáticamente el texto circundante de modo que los destinatarios reciben una carta que parece haber sido escrita personalmente para ellos. El Memoplan también permite que la persona que escribe una carta estándar incluya un recordatorio como ayuda para identificar el contenido de un campo determinado. Por ejemplo, 2(APELLIDO) indica que el segundo campo contiene el apellido de cada destinatario.

Los paquetes de correspondencia especializados destinados a ordenadores de oficina como el IBM PC y el Sirius, tales como el Mailmerge, producido por Micropro (el paquete de correspondencia más



Ian McKinnell

Correo en cadena
El procesador de textos proporciona un esquema de carta con espacios en blanco donde deberán consignarse los datos específicos como fecha, nombre y dirección, y la base de datos suministra los datos correspondientes contenidos en sus registros. Después de las cartas, se imprimen las direcciones en etiquetas autoadhesivas

conocido en Gran Bretaña), o el Mailing List Manager, creado por Peachtree, incluyen numerosas facilidades muy útiles y sofisticadas. No sólo se puede crear un gran número de archivos de direcciones diferentes, sino que también se dispone de amplias facilidades de búsqueda y selección que permiten realizar envíos especiales de correspondencia sólo a partes de una lista. Por ejemplo, la secretaria de un club de golf que utilice este sistema no tendría ningún problema para enviarles correspondencia a todos los nuevos socios del club con un récord inferior a 15 y que hayan pagado sus cuotas.

La búsqueda y la selección se consiguen mediante las técnicas estándar para bases de datos de indexación y claves de registros, llevando a cabo después comparaciones lógicas de los campos seleccionados del registro.

Estos avanzados paquetes también poseen facilidades de formateo muy precisas. Ésta es una característica particularmente útil, porque los nombres y direcciones producidos por el sistema se pueden confeccionar a medida para que se ajusten al tamaño y forma de las etiquetas. Con el sistema de Peachtree, por ejemplo, seleccionar la opción LABEL FORMAT en el menú principal produce en pantalla la imagen de una caja, junto con una lista de todos los campos del registro de la lista de correspondencia. Ello le proporciona al usuario una representación visual de la etiqueta que se está creando. Asimismo, hay facilidades para informar a la impresora de que, por ejemplo, la hoja de las etiquetas es de tres etiquetas de ancho, de modo que a cada pasada del cabezal de impresión se deben imprimir tres etiquetas, y así sucesivamente.

En general, los programas de correspondencia son de mayor utilidad cuanto mayores listas de correspondencia se explotan. Si usted sólo tiene que hacer unas cuantas etiquetas, probablemente lo más práctico sea mecanografiarlas una por una.

Un ejemplo de un paquete diseñado para el BBC Modelo B estándar es el que suministra GCC, una empresa de software con sede en Cambridge. Durante algún tiempo GCC ha comercializado su propia base de datos en ROM, llamada Starbase, que contaba con todas las facilidades necesarias para una base de datos (sólo listas de correspondencia). Pero ahora la empresa ha lanzado una versión de Starbase en 16 K de ROM, que proporciona completas facilidades para correspondencia, conjuntamente con el paquete de tratamiento de textos basado en ROM denominado Wordwise.

Starbase se suministra con un manual y un disco de utilidades. Junto con Wordwise, sus facilidades para correspondencia incluyen personalización de cartas normalizadas y creación de etiquetas. El disco de utilidades hace posible el envío de órdenes a la impresora durante la creación de etiquetas. De modo que en máquinas con impresoras adecuadas se puede utilizar una selección de distintos tipos de letras.

Una característica especialmente útil de Starbase como paquete para correspondencia es su capacidad para llevar a cabo operaciones aritméticas en los campos de un archivo de direcciones. De esta forma se pueden crear informes personalizados y facturas para los socios de clubes, realizando el ordenador todos los cálculos individuales necesarios.

Los usuarios del Commodore 64 pueden disponer de diversos paquetes para correspondencia. Un ejemplo es Visawrite, producido por Visa Software. A diferencia de Starbase, Visawrite funciona tanto con cassette como con disco, y se puede emplear con cualquier paquete de base de datos capaz de crear un archivo secuencial. Por otra parte, aplicado solo, como paquete de proceso, puede retener hasta 500 nombres y direcciones. Éstos se preparan utilizando cada página de documento como una ficha de registro y se pueden mezclar con una carta estándar.

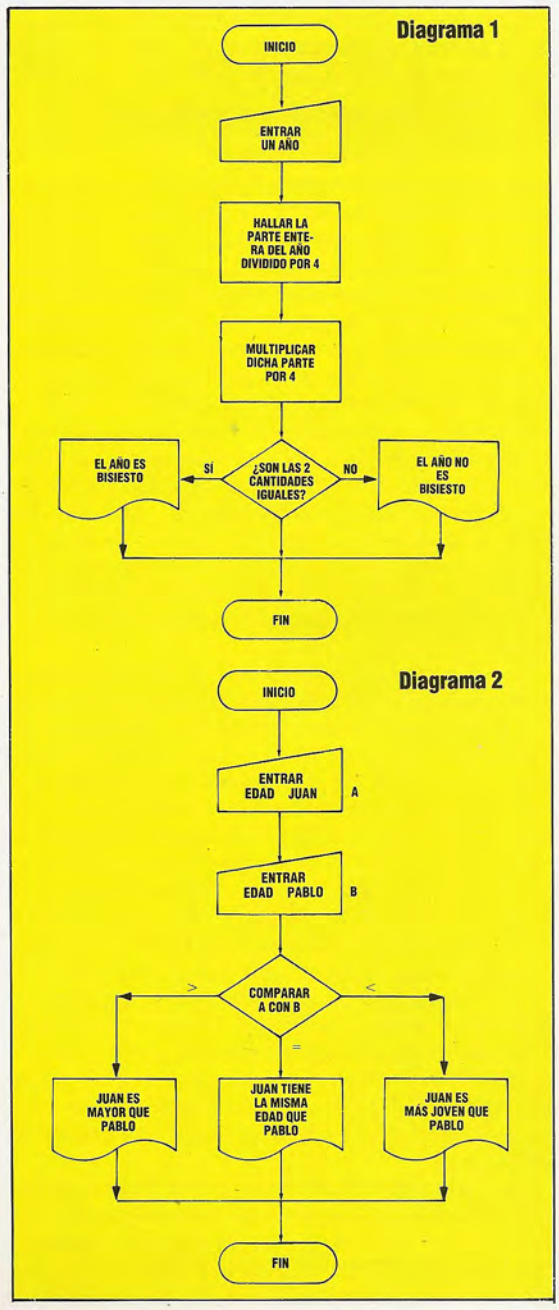
Aplicado solo, Visawrite no posee facilidades de búsqueda selectiva, aunque los usuarios pueden repasar a mano todos los nombres y direcciones y marcarlos individualmente para su inclusión en una partida de correspondencia.

Tres posibilidades

El símbolo de decisión o comparación determina si una afirmación es verdadera o falsa, pero puede proporcionar además una tercera salida

El siguiente programa determina si un año cualquiera entrado por teclado es bisiesto (suponiendo, en una primera fase, que un bisiesto es un múltiplo de 4).

El diagrama 1 muestra paso a paso lo que se ha transcrito en primer lugar (programa 1), mientras que el programa 2 se ha elaborado con la misma lógica, pero tratando de simplificar al máximo las operaciones.



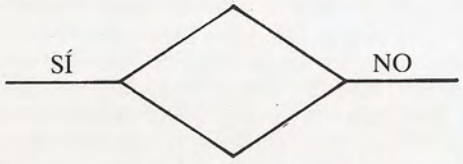
Programa 1

```
10 REM********BISIESTO
15 INPUT "ENTRAR UN AÑO";A
20 E = INT(A/4)
30 M = E*4
40 IF M = A THEN GOTO 60
50 PRINT "EL AÑO NO ES BISIESTO": END
60 PRINT "ES BISIESTO"
70 END
```

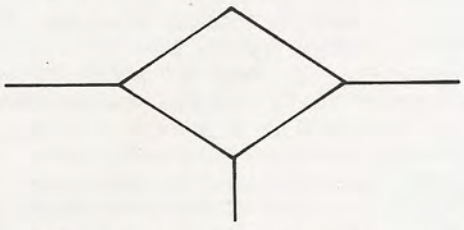
Programa 2

```
10 REM********BISIESTO
15 INPUT "ENTRAR UN AÑO";A
20 IF A/4 = INT(A/4) THEN PRINT
   "ES BISIESTO":END
30 PRINT "NO ES BISIESTO"
40 END
```

Hasta ahora, siempre que se ha precisado recurrir a una decisión, a base de confrontar elementos, esta comparación se ha hecho a nivel de verdadero o falso.



Para ello, el rombo ha proporcionado dos posibles salidas. Ahora bien, este mismo símbolo puede ofrecer una tercera vía de salida para que, en caso de que se necesite, tener la oportunidad de poder discernir las dos posibilidades de desigualdad, es decir, las de salida por mayor o menor además de la salida por igual.



En el diagrama 2 puede observarse cómo se utiliza esta técnica al comparar las edades de dos personas. Como se puede comprobar, la secuencia seguirá una de las tres posibles salidas, adquiriendo el resultado una precisión que no se hubiera obtenido en caso de preguntar solamente si Juan es mayor que Pablo, ya que en caso de que la respuesta hubiera sido negativa, habría quedado la duda de si se ha tomado esta vía por ser menor o por tener ambos la misma edad.

La combinación ideal

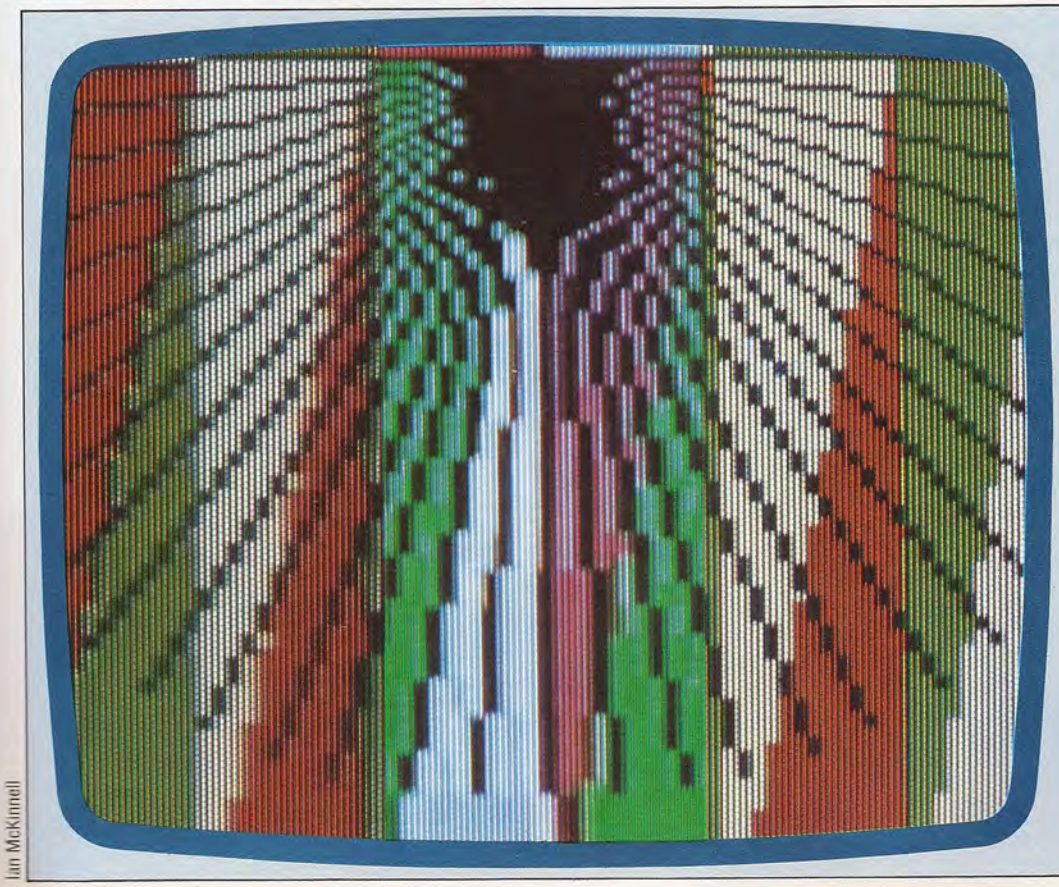
Un televisor combinado con una pantalla proporciona una visualización de gran calidad y permite, además, recibir transmisiones televisivas

La mayoría de los usuarios de ordenadores personales se acostumbra enseguida a las imágenes inestables y los colores confusos que producen sus máquinas cuando están conectadas a un televisor normal. Sin embargo, para aquellos afortunados que tienen acceso a una pantalla de ordenador, la calidad de la imagen se convierte en una revelación: los colores son claros y bien diferenciados, toda la visualización es más estable y hay una notoria ausencia de hormigueo de puntos que deben soportar los usuarios de televisores (se denomina *hormigueo de puntos* al brillo trémulo que se observa especialmente en los márgenes del texto visualizado en pantalla). Pero esta mayor calidad tiene su precio: las pantallas son más caras y no se pueden utilizar para recibir programas de televisión.

Actualmente, sin embargo, existe una tercera opción: el televisor-pantalla combinado proporciona a los usuarios lo mejor de ambas alternativas. Se compone de un aparato de televisión normal con un conector adicional que proporciona calidad de pantalla cuando se lo conecta a un microordenador. Quizá algunos usuarios ya posean uno de estos híbridos sin saberlo, ya que muchos de los televisores modernos vienen equipados con enchufes para la conexión de grabadoras de video, y éstos son igualmente apropiados para utilizarse con un micro.

El problema de emplear televisores convencionales como "visualizadores" de ordenador radica en la forma en que reciben las señales. Los programas de televisión se transmiten en forma de ondas de radio: éstas son recogidas por la antena de televisión y convertidas en imágenes. Un ordenador personal simplemente imita este proceso haciendo pasar su salida por un modulador (la pequeña caja del interior del ordenador en la cual se conecta el cable de la antena). Después de que el modulador ha alterado la señal convirtiéndola en la forma de onda de radio aceptable por el televisor, el receptor vuelve entonces a cambiarla para visualizar la imagen. Esto significa que la señal se puede degradar en dos lugares: en el modulador y dentro del propio receptor. Una pantalla omite la modulación; funciona directamente a partir de la señal primaria, proporcionando una visualización de gran calidad.

Un factor a considerar en el momento de adquirir una pantalla es el formato que utilice su ordena-



Ian McKinnell

Tres grados

Las señales de visualización que producen los ordenadores se clasifican en tres tipos principales. Todos los ordenadores personales emiten señales para televisión, pero esto a menudo significa una imagen pobre. La mayoría de los ordenadores producen, asimismo, señales para pantallas. La visualización que éstas proporcionan es mejor, pero son caras. Los televisores-pantalla combinan la calidad de una pantalla con la capacidad de captar imágenes de televisión. En cuanto a calidad, la principal diferencia entre los televisores y las pantallas es el tipo de señal con la que trabajan. Estas tres imágenes se produjeron en el mismo televisor-pantalla pero utilizando los tres tipos diferentes de señales: televisión (algunas veces llamadas RF), video compuesto y RGB. La señal de televisión (que aparece en el medio) proporciona la calidad más pobre; el video compuesto (a la izquierda) es un poco mejor, y la señal de RGB (a la derecha) es la que da mejor resultado

dor. En la actualidad hay dos tipos de señal para monitor: RGB (*Red, Green, Blue*: rojo, verde, azul) y video compuesto. El RGB da una imagen mejor, pero los dos tipos son considerablemente superiores a la salida de un televisor.

Existen, asimismo, dos clases de televisor-pantalla: los receptores de televisión normales que se han adaptado para tomar una señal para pantalla, y los aparatos construidos especialmente. Estos últimos son más adecuados, porque los aparatos convertidos a menudo se modifican sin el conocimiento del fabricante y, por lo tanto, no están amparados por ningún tipo de garantía. Los televisores-pantalla construidos especialmente están diseñados para ser utilizados de manera primordial, con grabadoras de video. Éstas por lo general disponen de entradas de video compuesto (busque un conector señalado como "video" o "audiovisual"). Los diagramas que acompañan este capítulo le mostrarán cómo conectar su ordenador (en el supuesto de que posea un modelo de video compuesto) a uno de estos conectores. Una vez hecho esto, hay que sintonizar el aparato para la visualización del ordenador de la misma forma en que seleccionaría un canal de televisión.

La principal ventaja del televisor-pantalla combinado respecto a la pantalla estándar es la facilidad de sonido. Muchos ordenadores personales (en especial los modelos Atari, Commodore y Dragon) se basan en el aparato de televisión para producir efectos sonoros. Una pantalla estándar no posee facilidades para sonido, mientras que los televisores-pantalla poseen altavoces y amplificadores incorporados.

Si su ordenador está dotado de una salida de RGB, sus opciones son más limitadas. Básicamente existen tres televisores-pantalla para entrada de RGB: el sistema Sony Profeel, los televisores con conectores Peri-TV (en particular la gama Normende) y el modelo ITT. El Sony Profeel acepta tanto señales de RGB como de video compuesto, pero utiliza un conector no estandarizado. El televisor-pantalla ITT posee un conector RGB que es compatible patilla a patilla con la salida del Oric y el Atmos, pero que también se podría utilizar con otros ordenadores RGB. El Normende es especialmente popular entre los usuarios de ordenadores personales, porque posee un conector Peri-TV. Éste es un conector de expansión para televisión, estandarizado internacionalmente, que acepta tanto señales de RGB como de video compuesto.

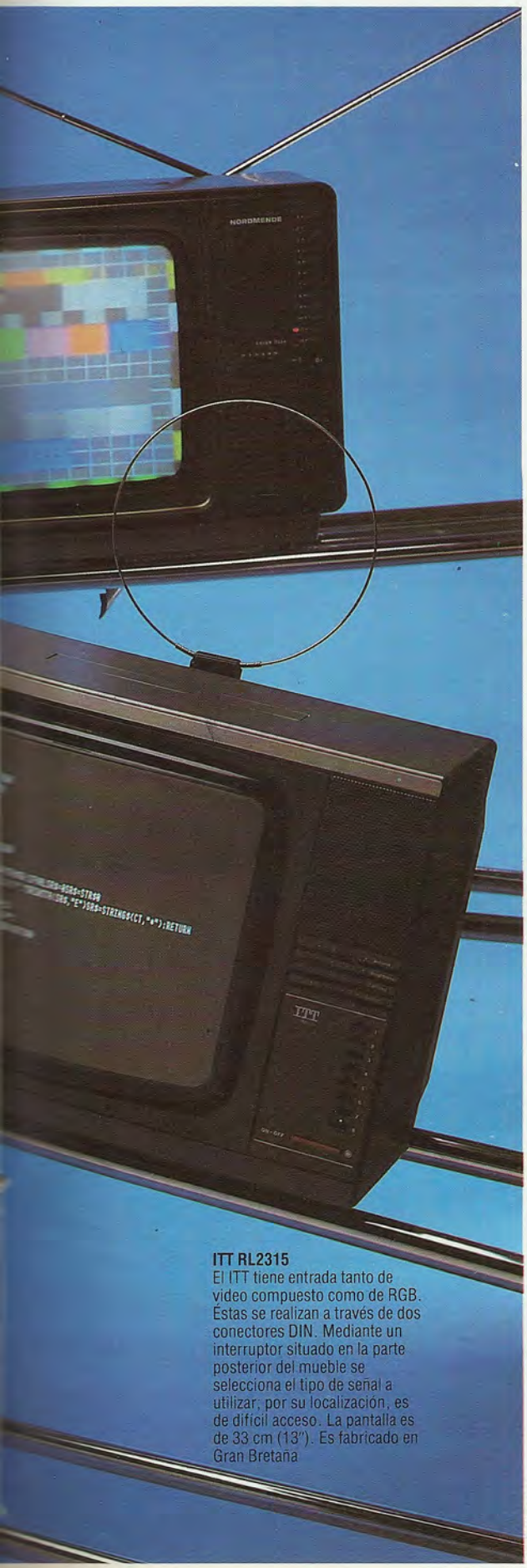
Otros aparatos de televisión se pueden equipar con entradas Peri-TV (también llamada *Scart*); compruebe, si su televisor es uno de ellos. El único problema de este sistema radica en que, en algunos televisores, para pasar de una modalidad a otra, se debe enchufar o desenchufar el conector Peri-TV. Esto es mucho menos conveniente que hacerlo conmutando los canales, tal como se haría en un receptor con conector de video. El hecho de que el sistema Peri-TV sea compatible tanto con las entradas RGB como con las de video compuesto significa que se puede conservar el mismo televisor-pantalla, aun cuando se cambie el ordenador.

Pero, independientemente del sistema escogido, el superior rendimiento de un televisor-pantalla combinado debería hacer que éste fuera el único tipo de aparato de televisión que el usuario de un ordenador adquiriera.

Ferguson TX con RGB
Este televisor-pantalla puede aceptar tanto entradas de RGB como de video compuesto a través de dos conectores DIN. Mediante los interruptores que hay en la parte delantera del TV se seleccionan los tres metodos de visualización. Están situados aquí porque la mayoría de los usuarios pasará a menudo de utilizar la unidad para mirar la televisión a trabajar con el ordenador. La pantalla es de 33 cm (13"). Se fabrica en Gran Bretaña

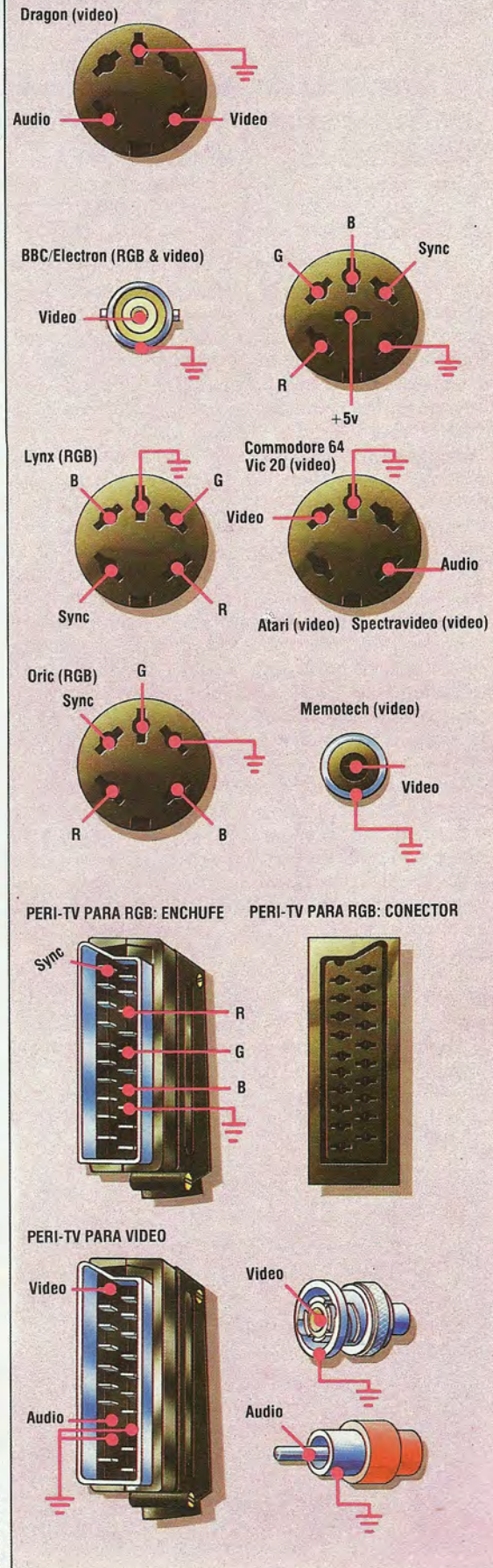
Normende 1534
Éste es uno de los muchos televisores Normende, todos los cuales poseen enchufes Peri y, por consiguiente, pueden aceptar señales de video compuesto o de RGB. Hay siete tamaños de pantalla diferentes disponibles y para la mayoría de los mismos existe la opción de control manual o control a distancia. El aparato que vemos aquí posee una pantalla de 33 cm (13"). Es fabricado en Singapur

Fidelity CM14
Existen dos modelos. Uno es pantalla solamente, con las entradas de RGB y de video compuesto a través de un enchufe Peri. El tamaño de la pantalla es de 33 cm (13"). Es fabricado en Gran Bretaña



ITT RL2315
El ITT tiene entrada tanto de video compuesto como de RGB. Éstas se realizan a través de dos conectores DIN. Mediante un interruptor situado en la parte posterior del mueble se selecciona el tipo de señal a utilizar, por su localización, es de difícil acceso. La pantalla es de 33 cm (13"). Es fabricado en Gran Bretaña

Chris Stevens



Kevin Jones

Conexiones correctas

Existen dos tipos distintos de señales para pantalla: video compuesto y RGB. Las señales de RGB se componen de señales separadas rojas, verdes y azules más una señal de "sincronización". Las entradas y las salidas de un monitor RGB utilizan conectores de patillas múltiples (por lo general DIN). Una señal de video compuesto tiene todas las señales de color y la de sincronización combinadas en una sola señal. Los conectores para entrada y salida de video compuesto por lo general son conectores *phono* o BNC (tipo bayoneta). No obstante, algunos ordenadores incluyen la salida de sonido desde el mismo conector que la señal de video; en estos casos, se deben utilizar patillas múltiples.

Los diagramas ilustran las conexiones que se pueden encontrar en los ordenadores personales. Si su televisor-pantalla posee una de las conexiones ilustradas, todo lo que tiene que hacer es preparar un cable con los dos enchufes, según las conexiones dadas (p. ej., de R a R, de sync a sync, etc.).

Los televisores-pantalla con conectores Peri-TV exigen realizar la conmutación de modalidad televisión a modalidad pantalla mediante la conexión o desconexión del enchufe. Éstos suelen requerir una salida de cinco voltios desde el ordenador (como la del BBC) y algún tipo de circuito de conmutación como el que se ilustra.

Hay dos importantes ordenadores, el Sinclair Spectrum y el ZX81, que no están incluidos en la lista porque carecen de interfaces para pantalla; a pesar de ello se los puede modificar para que proporcionen una señal de video compuesto. Hay al menos una empresa que fabrica un adaptador para que el Spectrum dé una señal RGB de mejor calidad. Este adaptador se enchufa en el micro y, por lo tanto, no anula la garantía.

El Commodore Vic-20, el Atari, el Spectravideo y las primeras versiones del Commodore 64 utilizan la misma entrada para video compuesto. Los modelos más recientes del Commodore, sin embargo, emplean un enchufe DIN de ocho patillas. El cableado de los mismos se debe realizar con la patilla dos como tierra, la patilla tres como sonido y la patilla cuatro como video

Imágenes en relieve

En esta ocasión explicaremos cómo crear gráficos tridimensionales introduciendo en un programa distintas funciones matemáticas

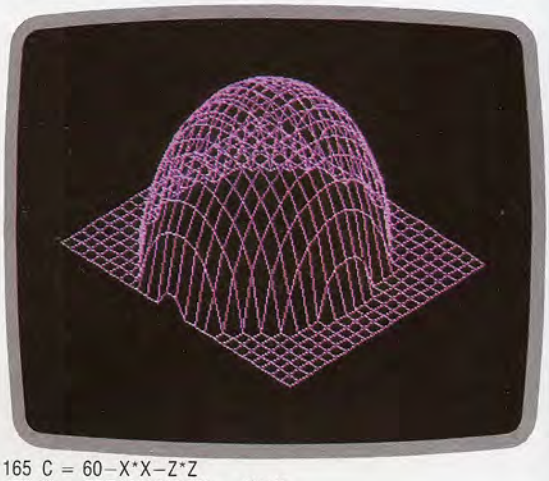
La sola mención de senos y tangentes, por no hablar de gráficos en tres planos, es suficiente para atemorizar a aquellos que se sintieron aliviados al haberse liberado de las matemáticas de la escuela. Pero con la ayuda de un ordenador estos temas se pueden convertir en una fuente de diversión, aun cuando no se comprendan cabalmente los principios sobre los que se sustentan.

La capacidad para gráficos de la mayoría de los ordenadores los hacen ideales para visualizar gráficos de ecuaciones matemáticas. Es posible que para la mayoría de nosotros dichas ecuaciones no tengan ningún significado cuando están escritas sobre el papel, pero al trazarse en forma de gráficos producen figuras muy atractivas. Incluso quienes aborrezcan las matemáticas podrían sentirse inspirados, al verlas, para realizar sus propias ecuaciones.

Todas las figuras que ilustran este capítulo se produjeron con un micro personal utilizando los programas listados. Todos están calculados como gráficos en tres dimensiones. Todo el mundo sabe el aspecto que tiene un gráfico normal en dos dimensiones. Un gráfico tridimensional se compone de varios gráficos bidimensionales visualizados al mismo tiempo, con ligeras diferencias entre cada uno de ellos. Dado que los ordenadores sólo pueden visualizar imágenes en dos dimensiones, el resultado no es auténticamente tridimensional, pero tal como se forman las imágenes se obtiene una ilusión de relieve.

Los programas listados aquí calculan los valores de una ecuación con dos variables, X y Z. El resultado, Y, se calcula para muchos valores de X y Z. Cada valor de Y se utiliza para situar un punto en la pantalla, correspondiendo los valores de Y a puntos sobre el eje vertical; o sea, cuanto más elevado es el valor de Y, más cerca del límite superior de la pantalla se trazará el punto. Los puntos vecinos se unen entre sí mediante líneas rectas, lo que proporciona un efecto de curva. Las curvas en una dirección representan gráficos de X e Y, manteniéndose Z constante, mientras que las curvas que intersectan a éstas son gráficos de Y y Z, con X manteniéndose constante (en este caso se trazan en el plano formado por los ejes Y y Z, que forman un ángulo recto respecto al plano X-Y de los gráficos bidimensionales normales). Dichas visualizaciones son útiles para comprender funciones complicadas.

Estas figuras también pueden ser un estímulo para las personas que normalmente no se sienten demasiado interesadas por las matemáticas. Es di-



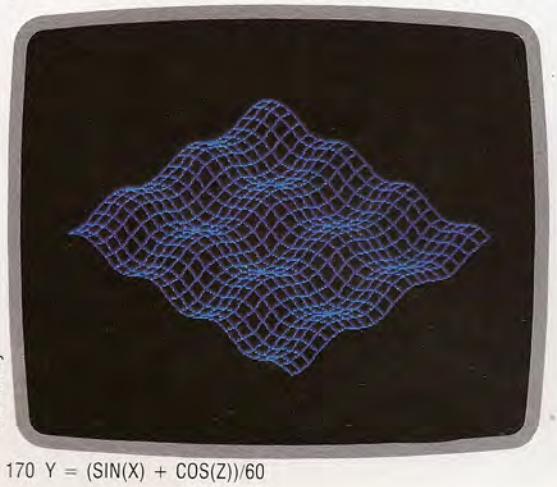
```
165 C = 60-X*X-Z*Z
170 Y = SQR(C*(SGN(C) + 1))/45
```

vertido (y bastante difícil) dar con la ecuación adecuada para producir una forma determinada. Para modificar el gráfico visualizado es necesario cambiar la función de la línea 170 del programa BASIC. Algunas funciones pueden ser relativamente complicadas y, por consiguiente, podrían requerir más de una línea de programa; de ser éste el caso, se podrían utilizar todas las líneas entre 151 y 179.

Además de elegir una función que produzca una forma atractiva, hay que tener cuidado en que los valores generados no sean tan grandes como para que el gráfico salga fuera de los límites de la pantalla. Para mantener la figura dentro de éstos, puede que sea necesario dividir la función por un número grande.

Posiblemente, el programa ofrecido funcionará en varios ordenadores personales. A modo de ayuda para efectuar la conversión, hemos diseñado el programa de una forma estándar. Esto significa que una ecuación que funcione en una máquina determinada también tendrá que funcionar en otras. La segunda parte del programa tiene como finalidad almacenar los valores utilizados para trazar los

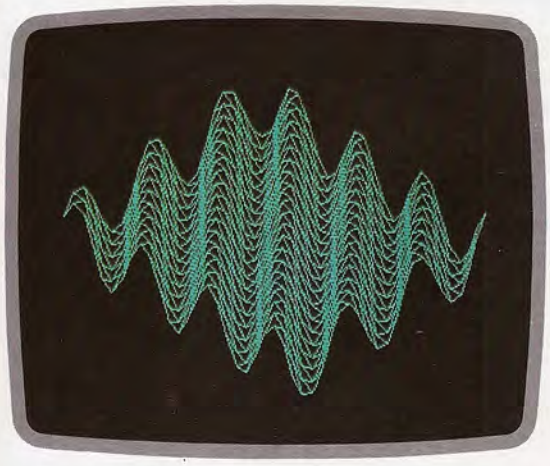
Creación de formas
Modifique el programa tal como se indica debajo de cada fotografía para producir estos gráficos tridimensionales



Liz Heaney

```
170 Y = (SIN(X) + COS(Z))/60
```

puntos en el gráfico. Estos resultados se guardan en una matriz y calcularlos requiere su tiempo. Los cálculos dependen de la función elegida y pueden



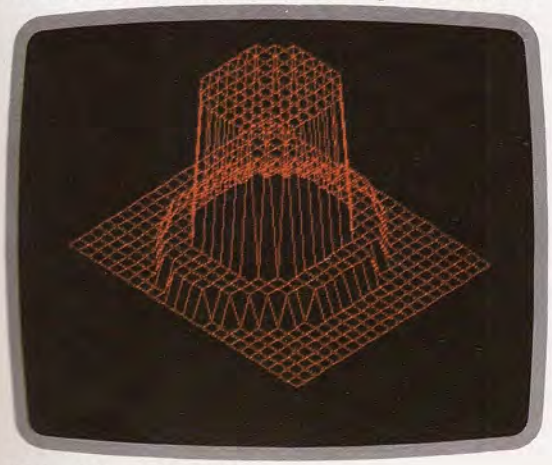
170 Y = SIN(X + Z)/12

consumir varios minutos; en este período parecerá que el ordenador está sin hacer nada. Calcular la función al principio ahorra tiempo a la larga. Si los cálculos se efectuaran mientras se están trazando las líneas, el programa tardaría casi el doble de tiempo en dibujar el gráfico.

Hemos listado varias funciones diferentes para que se experimente con ellas. Las ilustraciones reflejan los resultados que se pueden esperar. Usted también debería intentar desarrollar sus propios gráficos entrando al programa distintas funciones. Tenga cuidado al hacer esto; asegúrese de que el gráfico quepa en la pantalla y no intente ninguna operación matemática ilegal. Los dos errores más comunes son intentar dividir por cero (lo que da infinito) e intentar hallar la raíz cuadrada de un número negativo (lo que no existe).

Para evitar la división por cero, súmele una constante muy pequeña (p. ej., 0,00001) a cualquier variable que pueda convertirse en cero. La única forma de protegerse contra la raíz cuadrada de los números negativos es emplear la función ABS para hacer que todos los números sean positivos antes de hallar sus raíces cuadradas.

Se pueden producir algunas visualizaciones interesantes mediante funciones matemáticas comunes como SIN, COS, LOG, etc. Otras se pueden obtener



165 C = X*X + Z*Z + 0.00001
170 Y = SGN(INT(23/C))/3 + SGN(INT(55/C))/15

utilizando funciones de las que sólo disponen los ordenadores (pruebe con INT, SGN y ABS).

Este programa se podría mejorar de muchas maneras. Trate de adaptarlo de modo que cualquier función gradúe automáticamente sus valores para ajustarse a los límites de la pantalla, o bien intente trazar puntos en una tercera dirección, dando curvas para X y Z mientras se mantiene a Y constante (esto es relativamente complicado). Pero aun si usted simplemente utiliza el programa tal como está escrito aquí, descubrirá lo entretenido que es probar con las ecuaciones más absurdas que se le ocurran. Quizá los resultados le sorprendan.

Este programa está escrito en BASIC BBC.

```
 10 REM *TRAZADO DE GRAFICOS
 20 :
 30 REM *PREPARAR PANTALLA
 40 ACROSS = 1280:TAL = 1024:UP = -1
 50 XGAP = 25:ZGAP = 15
 60 WIDE = INT(ACROSS/XGAP/2)
 70 DEPTH = INT(TALL/ZGAP/3)
 80 MODE4:CLS:PRINTTAB(12)"CALCULANDO"
 90:
100 REM *CALCULAR GRAFICO
110 START = 20
120 DIM G(WIDE,DEPTH)
130 FOR A = -DEPTH/2 TO DEPTH/2
140 FOR B = -WIDE/2 TO WIDE/2
150 X = A*20/WIDE:Z = B*20/DEPTH
160 REM *INSERTAR FUNCION AQUI ABAJO
170 Y = (SIN(X) + COS(Z))/60
180 G(B + WIDE/2,A + DEPTH/2) = Y*UP*TALL
190 NEXT B:NEXT A:CLS
200 :
210 REM *DIBUJAR GRAFICO; PLANO X-Y
220 FOR Z = 1 TO DEPTH
230 XBASE = XGAP*Z
240 ZBASE = TALL/2 + Z*ZGAP + START*UP
250 XOLD = XBASE + XGAP
260 ZOLD = ZBASE-ZGAP-G(1,Z)
270 FOR X = 1 TO WIDE
280 XNEW = XBASE + X*XGAP
290 ZNEW = ZBASE-X *ZGAP-G(X,Z)
300 PLOT 4,XOLD,ZOLD:PLOT 5,XNEW,ZNEW
310 XOLD = XNEW:ZOLD = ZNEW
320 NEXT X:NEXT Z
330 :
340 REM *DIBUJAR GRAFICO; PLANO Z-Y
350 FOR X = 1 TO WIDE
360 XBASE = XGAP*X + DEPTH*XGAP
370 ZBASE = TALL/2-X*ZGAP + DEPTH*ZGAP + START*UP
380 ZOLD = ZBASE-ZGAP-G(X,DEPTH-1)
390 XOLD = XBASE-XGAP
400 FOR Z = 0 TO DEPTH-1
410 XNEW = XBASE-Z*XGAP
420 ZNEW = ZBASE-Z*ZGAP-G(X,DEPTH-Z)
430 PLOT 4,XOLD,ZOLD:PLOT 5,XNEW,ZNEW
440 XOLD = XNEW:ZOLD = ZNEW
450 NEXT Z: NEXT X
460 :
470 REM *MANTENER VISUALIZACION
480 GOTO 470
```

Complementos al BASIC

Spectrum

Insertar LET en todas las sentencias de asignación.
Insertar las siguientes líneas:

```
 40 LET ACROSS = 256:LET TALL = 176:LET UP = -1
 50 LET XGAP = 5:LET ZGAP = 3
 80 CLS
290 PLOT XOLD,ZOLD:DRAW XNEW-XOLD,ZNEW-ZOLD
410 PLOT XOLD, ZOLD:DRAW XNEW-XOLD,ZNEW-ZOLD
```

Oric-1/Atmos

Insertar las siguientes líneas:

```
 40 ACROSS = 239:TALL = 199:UP = 1
 50 XGAP = 5:ZGAP = 3
 80 HIRES
300 CURSET XOLD,ZOLD,1:DRAW XNEW-XOLD,ZNEW-ZOLD,1
430 CURSET XOLD,ZOLD,1:DRAW XNEW-XOLD,ZNEW-ZOLD,1
```

Dar en la diana

Iniciamos una serie de capítulos que pretenden dar a conocer las técnicas que utilizan los buenos programadores

La buena programación se desarrolla mediante la experimentación y la experiencia. El programador novato, que suele resolver los problemas gracias a un enorme entusiasmo y un inmenso esfuerzo, se va transformando gradualmente en un técnico consciente de los métodos expeditos y empíricos que consiguen los resultados deseados. Finalmente, el programador desarrollará la claridad y el enfoque directo del experto. Pero no existe ninguna razón por la que no se pueda acelerar el progreso personal de un programador de ordenadores personales mediante el aprendizaje en función de los errores de otros que han seguido su mismo camino. Las lecciones están allí, para aprenderlas, y la programación de todos se puede beneficiar con ellas. Nuestro curso comienza con un análisis de algunas de las pistas más útiles que pueden serle de ayuda al principiante.

La programación es un proceso para resolver problemas, y gran parte de ella se debe llevar a cabo en la mente y con lápiz y papel mucho antes de que se pueda escribir una línea de código. Las etapas de este proceso son bien conocidas: un claro y amplio enunciado del problema en términos prácticos, repetido reiteradas veces con creciente precisión, hasta formularlo con tanto detalle y exactitud como sea posible. Esta descripción casi siempre contiene o implica la solución esencial, que se debe entonces exponer con mayor detalle y de forma más práctica de modo que se convierta en un método de trabajo. En programación, sólo la última etapa incluye codificación y ésta debe ser una concreción directa de las etapas precedentes. Cuando la etapa de codificación se superpone con la verdadera solución del problema, el resultado son soluciones pobres y código malo.

A las soluciones se las suele denominar *algoritmos*, procesos de cálculo analizados en etapas lógicas. La eficiencia de un programa depende básicamente de la eficiencia de su algoritmo, y éste se juzga en función de lo completo que sea y de su corrección. Estas cualidades de puro sentido común se refieren a la capacidad teórica y práctica del programa para hacer frente a la gama previsible de condiciones de entrada y a la coherencia de su lógica interna. Huelga decir que es mucho más fácil reconocer su ausencia que demostrar su presencia, pero todos los problemas deben someterse a esta valoración y lo mejor es realizarla en la etapa más temprana posible de su desarrollo.

Las soluciones deben ser *fiables*, además de completas y correctas. No sólo deben tratar el ámbito de problemas prescritos, sino que también deben funcionar de forma predecible y segura en condiciones fuera de su ámbito. Esto por lo general significa posser la capacidad de reconocer condiciones de error potenciales y ser capaz de detener la operación con todos los datos intactos, así como visualizar algunos mensajes de estado significativos. Juzgar si un programa es suficientemente fiable es difícil: es más fácil reconocer un programa que no es fiable que reconocer uno que sí lo es. La experiencia es la fuente de una mejor valoración.

Hacer que los programas sean fiables y sólidos es una tendencia meritoria que casi siempre entra en conflicto directo con un objetivo que es igualmente deseable: lograr que los programas sean *económicos*. Todo cuesta dinero, aun cuando sólo se trate del tiempo que se tarda en escribir programas como mera diversión. Siempre llega un momento en el que uno tiene que decidir entre seguir trabajando en un programa que parece "a prueba de todo", o abandonarlo para comenzar un proyecto nuevo. Incluso aunque se disponga de tiempo ilimitado, la memoria y la velocidad de operación del ordenador no lo son. Es bastante posible rodear el algoritmo central con tanto código preventivo y de detección de errores, que protegerlo contra casos extraños pueda ocupar más tiempo que resolver el problema original.

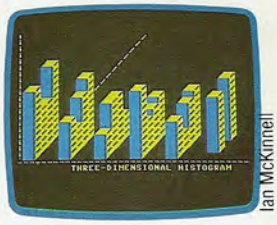
Verificación y depuración

Resolver problemas lógicos y analíticos en teoría es sumamente importante, pero los programas están hechos para realizar una tarea. Después de solucionar los primeros errores lógicos y de sintaxis, es hora de empezar la *verificación*. Ésta es una idea tan familiar que apenas si parecería merecer alguna explicación, por no hablar de hacer hincapié en ella. Pero, en realidad, es un proceso que no se suele comprender bien. En todos los programas, exceptuando los triviales, suelen existir demasiadas posibles combinaciones de las condiciones de entrada como para poder realizar pruebas exhaustivas, de modo que se deben poner a punto juegos de ensayo para comprobar al máximo las partes del programa más vulnerables (y que han de ser las más fuertes). Generar juegos de ensayo exhaustivo no es una labor sencilla y lleva tiempo y dinero. El enfoque profesional a la verificación es que no existen programas perfectos, sino pruebas malas.

Las pruebas adecuadas revelan las insuficiencias de un programa, y deben hacerlo de una forma lógica, de modo que la *depuración (debugging)* consuma el menor tiempo posible. Al igual que la verificación, la depuración, o eliminación de errores, es un proceso esencial que fracasa muy a menudo precisamente porque personifica los fallos humanos que la hacen necesaria. Un error en un programa se debe enfocar como otro problema a resolver, exactamente como hemos descrito antes (enunciación, análisis, algoritmo, verificación), pero lo más frecuente es abordarlo como una plaga a machacar, envenenar o pisotear, con consecuencias predecibles desastrosas para las áreas circundantes.

A medida que se van completando estas etapas de desarrollo, la familiaridad y la satisfacción se van combinando para convencer al programador de que el programa funciona ahora, funcionará siempre y no requerirá jamás ninguna modificación y que, de cualquier forma, el código es un modelo de claridad. Pero los programadores, y no los programas, necesitan documentación. No hay ningún programa que sea autoexplicativo y siempre existen razones para cambiar programas que ya funcionen. Al igual que cualquier otro mecanismo, necesitan *mantenimiento*, es decir, manuales. Los programas han de estar documentados internamente (utilizando líneas REM) para beneficio del programador, y documentados exteriormente con literatura alusiva por el bien del usuario, aun cuando éste sea el propio programador.

Todas estas lecciones las tuvieron que aprender una vez los programadores de ordenadores centrales y han sido ignoradas y dolorosamente redescubiertas por los programadores de microordenadores. Tomadas en conjunto constituyen una "estructura" de programación, un enfoque unificado a la solución de problemas mucho más amplio que un libro de consejos relativos a evitar los GOTO o a adoptar WHILE...WEND. Los programas eficientes los escriben programadores eficientes, por supuesto, pero en todo caso sobre la base de la experiencia estructurada y el pensamiento lógico. Esta serie tiene por objeto fomentar ambos.



Ian McKinnell

Diagramas de barras
Los colores y la profundidad de las barras que conforman el diagrama se ajustan fácilmente por programa cambiando los valores de las variables de control

```
399 REM ************************************
400 REM*GRAFICOS DE BARRAS TRIDIMENSIONALES*
401 REM ************************************
500 GOSUB 1500:          REM INICIALIZAR
720 YY = 22:XX = 2:      REM ORIGEN COORD
760 GOSUB 3200:          REM DIBUJAR EJES
800 FOR E = LT - 1 TO 0 STEP -1
820 OC = XX + E*DB:OL = YY-E*DB
840 GOSUB 2200:          REM INPUT DATOS
900 FOR D = 1 TO NN
920 HT = DT(D)
940 X0 = OC + (D-1)*(BB + LT*DB):Y0 = OL-HT-DB
960 GOSUB 4000:          REM IMPRIMIR BARRA
980 NEXT D
1000 NEXT E
1100 XP = 10:YP = 23:GOSUB 3500
1120 PRINT"HISTOGRAMA TRIDIMENSIONAL"
1200  A$ = INKEY$:IF A$ = "" THEN GOTO 1200
1400 END
1499 REM ************************************
1500 REM*INICIALIZAR S/R                    *
1501 REM ************************************
1520 CLS = CHR$(147):    REM LIMPIAR PANTALLA
1540 PRINT CL$
1560 PO$ = CHR$(19):     REM SITUAR CURSOR
1580 RT$ = CHR$(13):     REM <RETURN>
1600 BB = 2:DB = 1:      REM DIMENSIONES BARRAS
1620 SW = 40:SD = 25:    REM DIMENSIONES PANT.
1640 HB = SD-DB:         REM ALT. MAX. BARRAS
1660 DIM B$(HB + DB)
1680 FOR K = 1 TO SD:PO$ = PO$ + RT$:NEXT K
1800 DIM DT(SW)
1900 GOSUB 2400:         REM CONSTRUIR BARRA
2100 LT = 4:             REM FACTOR PROFUNDIDAD
2190 RETURN
2199 REM ************************************
2200 REM*        ENTRAR DATOS S/R MATRIZ     *
2201 REM ************************************
2220 READ NN
2240 FOR Z = 1 TO NN:READ DT(Z):NEXT Z
2310 DATA 6,12,10,4,7,8,10
2320 DATA 5,7,8,8,6,7
2330 DATA 6,7,4,8,5,3,9
2340 DATA 5,11,6,4,11,6
2390 RETURN
2399 REM ************************************
2400 REM*      S/R CONSTRUIR BARRA ENTERA    *
2401 REM ************************************
2500 TC$ = CHR$(158):    REM LADOS = AMARILLO
2520 FC$ = CHR$(31):     REM FRENTE = AZUL
2540 RV$ = CHR$(18):     REM ACTIVAR REVERSE
2560 NL$ = CHR$(146):    REM DESACTIVAR REVERSE
2580 CR$ = CHR$(29):     REM CURSOR DERECHA
2600 CH$ = CHR$(32):     REM CAR. ESPACIO
2620 C1$ = CHR$(169):    REM CAR. "◤"
2640 C2$ = RV$ + C1$:    REM CAR. "◢"
2680 FOR K = 1 TO SW
2700 SP$ = SP$ + CH$
2720 RC$ = RC$ + CR$
2740 FF$ = FF$ + CH$
2760 NEXT K
2800 TL$ = SP$ + "/"
2820 BL$ = SP$ + NL$ + C1$
2840 FL$ = LEFT$(FF$,BB)
2900 L$ = TC$ + C2$ + RV$ + RIGHT$(TL$,BB)
2920 FOR K = 1 TO DB
2940 B$(K) = LEFT$(RC$,DB-K) + L$ + LEFT$(SP$,K-1)
2960 NEXT K
3000 L$ = FC$ + RV$ + FL$ + TC$ + RIGHT$(TL$,DB)
3020 FOR K = DB + 1 TO HB
3040 B$(K) = L$
3060 NEXT K
3100 L$ = FC$ + RV$ + FL$ + TC$
3120 FOR K = 1 TO DB
3140 B$(HB + K) = L$ + RIGHT$(BL$,DB + 2-K)
3160 NEXT K
3190 RETURN
3199 REM ************************************
3200 REM*          DIBUJAR EJES S/R          *
3201 REM ************************************
3300 PRINT TC$:           REM ESTABLECER COLOR
3320 FOR Y = 2 TO YY-1
3340 XP = XX-1:YP = Y:GOSUB 3500:PRINT"!"
3360 XP = XX + YY-Y:GOSUB 3500:PRINT"/"
3380 NEXT Y
3400 YP = YY
3420 FOR X = XX-1 TO SW-1
3440 XP = X:GOSUB 3500:PRINT"-"
3460 NEXT X
3490 RETURN
3499 REM ************************************
3500 REM*       PONER CURSOR @XP,YP S/R      *
3501 REM ************************************
3600 PRINT LEFT$(PO$,YP)TAB(XP-1);
3620 RETURN
3999 REM ************************************
4000 REM*       IMPRIMIR BARRA PARCIAL S/R    *
4001 REM ************************************
4100 FOR V = 1 TO HT
4120 XP = X0:YP = Y0 + V-1
4140 GOSUB 3500           REM COLOCAR CURSOR
4160 PRINT B$(V)
4180 NEXT V
4200 FOR V = 1 TO DB
4220 XP = X0:YP = Y0 + HT + V-1:GOSUB 3500
4240 PRINT B$(HB + V)
4260 NEXT V
4490 RETURN
READY.
```

Estructura
Modular y razonablemente autoexplicativa

Documentación
Obviamente esta rutina es crucial, pero está completamente sin explicar

Nombres de las variables
Bien comentadas, pero los nombres no son muy significativos

"Completitud"-corrección
¿Funciona esto para todos los valores? ¿Produce la salida correcta en todos los casos, por ejemplo, cuando un valor es menor que cero?

Detección de errores
Ninguna suma de control, ninguna graduación, ningún tratamiento de errores

Documentación
Buena: no hay "números mágicos" ni caracteres de control misteriosos: todo es traducible

Cal y arena

Este programa para visualizar gráficos de barras tridimensionales es una molesta mezcla de buen y mal estilo: la documentación interna es buena donde existe y la estructura es modular, pero el programa no es autoexplicativo, no hay detección de errores y ninguna documentación para el usuario

Complementos al BASIC

Este programa está escrito en BASIC Microsoft y funcionará sin ninguna modificación en micros con una visualización en pantalla de 40×25. Las dimensiones de la pantalla se establecen en la línea 1620. Los valores de los caracteres de control inicializados en las subrutinas 1500 y 2400 son para el Commodore 64; consulte la tabla ASCII en su manual para otras máquinas

Aventura medieval

Analizamos aquí un interesante juego de aventuras: "Twin kingdom valley" (El valle de los dos reyes), creado por Bug-Byte

Los juegos de aventuras para ordenadores derivaron originalmente del juego *Dungeons and dragons* (Calabozos y dragones). En los años sesenta los programadores de ordenadores centrales comenzaron a desarrollar las primeras versiones para ordenador, utilizando las enormes cantidades de memoria disponible para almacenar detalles de un complicado mundo fantástico lleno de hechiceros y monstruos, enanos y gnomos. Todas las aventuras actuales para microordenadores descienden de aquellos primeros ejemplos, pero están ambientadas en una gama de lugares mucho más amplia, desde naves espaciales abandonadas a calles de Chicago en los años treinta, la época de los gángsters.

Pero todas las buenas aventuras tienen una característica común: deben hacer que el jugador sienta que el mundo de fantasía es real. Las mejores aventuras son casi como novelas, implicando por completo al jugador en las situaciones descritas. Originalmente, todas las aventuras eran solamente de textos, pero la nueva generación emplea gráficos de alta resolución para conferir a los juegos una sensación adicional de realismo. La primera aventura con gráficos que obtuvo un gran éxito de ventas fue *The Hobbit*, basado en la novela del mismo nombre de J. R. R. Tolkien. La aventura que vamos a analizar en este capítulo, *Twin kingdom valley* (El valle de los dos reyes), utiliza gráficos en un escenario tradicional de aventuras, con bosques y castillos medievales.

Si bien los gráficos le suelen conferir a la aventura una cierta brillantez, debe decirse que por sí solos no pueden disimular una falta de imaginación por parte del programador, y esto es lo que sucede con *Twin kingdom valley*. La historia en que se basa el juego es muy sencilla. El jugador asume el papel de un vagabundo que se adentra en el valle, que está gobernado por dos reyes en perpetua lucha (el rey del desierto y el rey del bosque). En el valle hay varios ríos (todos los cuales tienen un aspecto parecido) que confluyen en el lago mágico Watersmeet. Al recorrer el reino, el jugador (una especie de héroe mercenario) debe recoger la mayor cantidad posible de tesoros. Cuando se han acumulado los suficientes y el marcador del jugador ha llegado a 1 024, sucede algo sorprendente (no vamos a estropear el interés del juego revelándole de qué se trata).

El movimiento y las acciones se controlan mediante instrucciones. El programa acepta 23 verbos, que se combinan con sustantivos que aluden a objetos del juego. Se aceptará una instrucción como "golpear al guardián con el martillo", siempre que se esté en posesión de un martillo y que haya un guardián cerca. Las direcciones se indican mediante los puntos de la brújula, más las palabras *up* (arriba) y *down* (abajo). En el reino habitan otros personajes y se puede emplear el verbo *ask* (pedir) para adquirir sus posesiones. En la mayoría de los casos, sin embargo, si intenta dirigirles la palabra obtendrá sin ningún motivo una respuesta violenta.

La función de los gráficos es ilustrar los 175 lugares en que se desarrolla el juego. El BBC Micro, en particular, utiliza una gran cantidad de memoria para producir visualizaciones de alta resolución, de modo que la mayor parte de las imágenes se componen de distintas combinaciones de las mismas formas básicas; por ejemplo, un bosque consta de 10 o 12 formas de árboles repetidas formando distintos patrones. En el caso del Commodore, su mayor memoria y sus gráficos tipo sprite permiten la animación en algunas pantallas, con ardillas trepando por los árboles y el agua goteando por las estalactitas. Los gráficos se pueden omitir, pero aun así utilizan un espacio de memoria que se podría haber empleado para hacer que la aventura resultara más emocionante.

Twin kingdom valley sólo es moderadamente difícil de resolver y en cuanto a concepto, apenas si es original. Existen muchas otras aventuras de texto solamente que son mucho más complicadas, si bien, por otra parte, le proporcionan al jugador una sensación mucho más intensa de vivencia en el mundo que describen.

Reinos de experiencia

Las visualizaciones del BBC Micro suelen ser distintas combinaciones de figuras básicas, mientras que al Commodore 64 su mayor capacidad de memoria y sus gráficos tipo sprite le permiten más variedad y algo de movimiento

Commodore 64

BBC Micro

Ian McKinnell

Twin kingdom valley: Para el BBC Micro y el Commodore 64
Editado por: Bug-Byte Software, Mulberry House, Canning Place, Liverpool LI 8JB
Autor: Trevor Hall
Palancas de mando: No se necesitan
Formato: Cassette

Cómo usar el pixel

Comenzamos con una interesante aplicación del lenguaje máquina: la pantalla en alta resolución del Commodore 64

Ya conocemos los diversos procedimientos y pasos para obtener gráficos de alta resolución en un Commodore (véase p. 734). Sabemos que ante todo debemos activar el chip de la interface para video (el VIC) poniéndolo en la modalidad de alta resolución, y cambiar el puntero de la dirección de base del juego de caracteres. Debemos asimismo limpiar el bloque de 8 Kbytes que empieza en la posición 8192 que albergará el mapa de la pantalla, e inicializar después el mapa de la pantalla normal (posiciones 1024 a 2033, o sea, $0400 a $07E7), que va a ser usado para contener la información del color de la pantalla. Tarea esta última algo complicada en visualizaciones polícromas, pero que a nosotros nos resultará trivial pues nos limitaremos a un solo color de fondo en la pantalla.

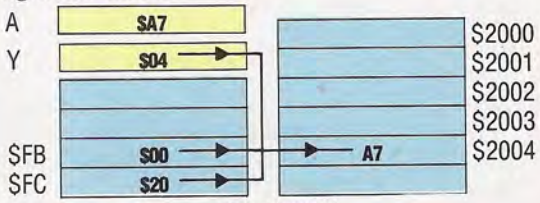
El programa debe permitir la activación y la desactivación de la modalidad de alta resolución, realizando posteriormente los cálculos necesarios para trazar un punto sobre la pantalla en alta resolución. Por tanto, la primera parte de dicho programa se va a ocupar de dos importantes tareas previas al uso de la alta resolución: colocar la información del color en cada una de las posiciones de la pantalla normal, y limpiar el mapa de bits de 8 Kbytes.

Para poder llamar a la misma rutina al activar o desactivar la modalidad de alta resolución se empleará un flag o indicador, que llamaremos HRSFLG (en inglés, **H***igh* **R**e**S***olution* **FL***a***G**). Ahora bien, no siempre desearemos limpiar el mapa de bits al entrar en modalidad de alta resolución, sobre todo si queremos que en la pantalla permanezca una figura anteriormente dibujada. Para esto, un segundo flag, el CLRFLG (en inglés, **CL***ea***R FL***a***G**), servirá de indicador si deseamos limpiar la pantalla o no. A la derecha encontrará usted el diagrama de flujo que emplea estos dos flags dentro de la rutina en lenguaje máquina.

Vayamos ahora a la tarea relativamente fácil de acceder, por medio de un bucle en lenguaje máquina, a bloques de memoria de 256 bytes o menos. El siguiente fragmento de programa coloca el número $03 en cada una de las posiciones que van desde la dirección BASE a la dirección BASE + 255 (en total, 256 posiciones), empleando un direccionamiento indexado absoluto (véase p. 676).

```
      LDY $00
      LDA $03
NEXT  STA BASE,Y
      DEY
      BNE NEXT
```

Observe que con esta técnica se accede primero a BASE, y al bloque restante se accede descendiendo desde BASE + 255 hasta BASE + 1. Pero nuestro programa nos va a pedir el acceso a más de un bloque de 256 bytes, por lo que debemos servirnos también del direccionamiento llamado indirecto postindexado. El sistema operativo del Commodore 64 necesita casi toda la página cero para él, pero deja unos cuantos bytes a disposición del programa de lenguaje máquina, por ejemplo, los bytes 251 y 252 (o sea, $FB y $FC). En este tipo de direccionamiento, el ordenador asume que el byte *lo* de la dirección en cuestión se encuentra en la posición de la página cero especificada, y el byte *hi* en la posición siguiente, siempre en esa misma página cero. Luego una instrucción como STA ($FB),Y, suponiendo que $FB y $FC contienen $00 y $20, e Y contiene por su parte $04, calculará la dirección pedida del siguiente modo:



STA ($FB),Y
$2000 + $04 = $2004

Un método similar es el que puede servirnos para acceder a un bloque entero de 256 bytes de la memoria. La potencia de este procedimiento se basa en la posibilidad de acceder al byte *hi* de la dirección BASE. Un incremento de una unidad en este byte equivale a un incremento de la dirección BASE en 256 (lo que quiere decir el comienzo del bloque siguiente de 256 bytes de la memoria). Esta técnica puede resultarnos útil en las tareas de colocar la información del color en el área de la pantalla normal o de limpiar el área del mapa de bits.

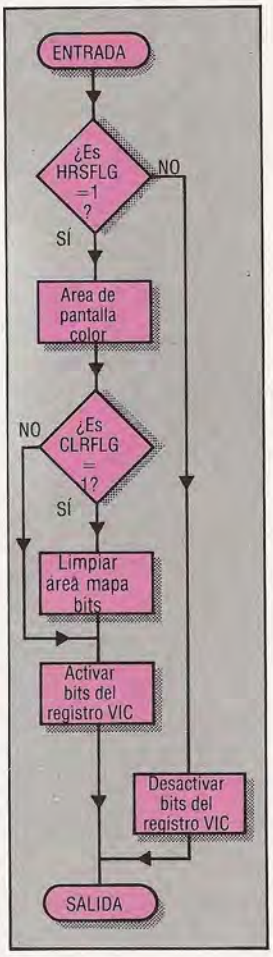
El área de la pantalla se extiende desde $0400 hasta $07E7. Es decir, se compone de tres bloques de 256 bytes más un resto de $E7 bytes. El trozo de assembly que denominamos "Área colores pantalla" en la p. 819 emplea el direccionamiento indirecto postindexado para poner en cada byte una información del color. Se sirve de las variables SCBLO y SCBHI que representan los bytes *lo* y *hi* de la dirección inicial de la memoria correspondiente a la pantalla normal SC, así como de SCBLK y SCBREM para el número de bloques de 256 bytes y el resto (en inglés, **BL***oc***K** y **REM***ainder*). Por último, PTR es la posición en la página cero donde almacenaremos el byte *lo* de la dirección base.

Cálculo de la posición de un pixel

La segunda parte de esta rutina en lenguaje máquina trata del cálculo del bit correspondiente a unas coordenadas (x,y) específicas dentro del área que contiene el mapa de bits. En una pantalla de alta

La rutina A. RES

Realiza tres tareas diferentes. Establece la información del color de la pantalla, limpia (*clear*) el área del mapa de bits y activa y desactiva los bits del registro de A. RES. (en inglés, HIRES), según sea el estado de los flags HRSFLG y CLRFLG



resolución, la abscisa *x* adquiere valores que van del 0 al 319, y la ordenada *y*, del 0 al 199. Estos últimos valores de *y* pueden ser representados por un byte, pero los de *x* necesitan dos, ya que pueden superar el valor 255. En BASIC el bit correspondiente a una *x* e *y* determinadas se calculaba tal como se expone a continuación:

```
1000 HB = XAND248:VBYTE = INT(Y/8)
1010 RMY = YAND7:RMX = XAND7
1020 ROW = VBYTE*320+HB
1030 BYTE = BASE+ROW+RMY
1040 POKEBYTE,PEEK(BYTE)OR(2 ↑ (7−RMX)
```

Los mismos cálculos debe realizar ahora la rutina en lenguaje máquina. El proceso aritmético se dificulta, ya que a veces emplea dos bytes HB, ROW, BASE y BYTE. Hay dos fragmentos de la codificación que deben ser explicados: la división de *y* por ocho y la multipliación de VBYTE por 320. Es fácil dividir por potencias de dos:

```
45      = 00101101
45/2 = 22 = 00010110
22/2 = 11 = 00001011
11/2 =  5 = 00000101
5/2  =  2 = 00000010
2/2  =  1 = 00000001
1/2  =  0 = 00000000
```

Observando los resultados binarios en cada división sucesiva por 2, vemos cómo los bits que forman el resultado binario van desplazándose hacia la derecha hasta desaparecer. De donde se deduce que la división por 8 se puede obtener mediante tres "desplazamientos lógicos a la derecha" (LSR:*Logical Shifts Right*). Y como sólo necesitamos la parte entera del resultado, podemos muy bien despreciar todo bit que "caiga" por el extremo derecho del número. La operación LSR coloca, no obstante, el bit desechado en el flag de arrastre, por lo que podemos utilizarlo si hace falta. Es fácil adivinar que la multiplicación por 2 se realiza desplazando un lugar a la izquierda el número binario (operación ASL). Esto nos permite crear una rutina para multiplicar VBYTE por 320. Debemos tomar nota de que 320 = 5×64 = 5×2×2×2×2×2×2. De aquí deducimos que si sumamos cinco veces VBYTE consigo misma y después realizamos seis operaciones ASL, habremos conseguido el producto deseado: VBYTE por 320. Sólo hay que tener en cuenta que el resultado puede superar el número 255 y, por tanto, necesitamos dos bytes.

Para concluir diremos que las dos rutinas estudiadas pueden ser llamadas por un programa BASIC por medio de la instrucción SYS y que es importante asegurarse de que, una vez concluida la rutina, continuará el programa BASIC. Durante la ejecución de un programa BASIC el intérprete hace uso de los registros X, Y y A, que son también usados por las rutinas en lenguaje máquina; por consiguiente, debemos salvar sus contenidos al iniciar dichas rutinas y restaurarlos al terminarlas. Para esto nada mejor que usar la pila. Los valores de Y, X y A se colocan en la pila al entrar en la rutina en lenguaje máquina y se recuperan de ella poco antes de abandonarla. Los valores de las coordenadas, de los colores y de los flags se pasan al programa en lenguaje máquina mediante POKE, que los coloca en las posiciones especificadas, tal como vemos en el programa BASIC que hemos transcrito.

```
1 GOTO 200
2 POKE 53265,PEEK(53265)AND223
3 POKE 53272,PEEK(53272)AND240OR4
4 STOP
200 REM ****DEMO ALT. RES. C64****
210 :
220 POKE 56,32:CLR: REM BAJAR TOPE MEMORIA
250 GOSUB 3000: REM INICIALIZAR S/R
350 :
360 REM ****ACTIVAR MOD. ALT. RES.****
370 :
380 PRINT CC$:PRINT:PRINT
390 INPUT"COLOR PRIMER PLANO";FG
400 INPUT"COLOR FONDO";BG
410 TT = FG*16+BG: REM CALC. COLOR
420 POKE COLOUR,TT: REM COL. S/R LENG. MAQU.
430 POKE HRSFLG,1: REM ACTIVA ALT. RES.
440 POKE CLMFLG,1: REM ACTIVA PANT. ALT. RES.
450 SYS BEGIN: REM ENTRA S/R DE LENG. MÁQU.
460 :
500 REM ****DIBUJO DEL MODELO****
510 :
515 Z = 0:Y1 = 50:Y2 = 150:X1 = 160:SP = 6
520 FORY = Y1 TO Y2 STEP SP
530 FORX = Y2−Z TO Y2+Z STEP SP
540 GOSUB 1000: REM DIBUJAR PUNTO
550 NEXT X
555 Z = Z+SP
557 NEXT Y
560 :
565 GETJ$:IFJ$<>""THEN 565
570 GETA$:IF A$ = ""THEN 570:REM ESPERA OPRESION TECLA
580 :
600 REM ****LIMPIA PANT. ALT. RES.****
605 POKE HRSFLG,1:POKE CLMFLG,1
610 SYS BEGIN
620 :
630 REM ****DEMO DE LINEA****
640 X1 = 0:X2 = 300:Y1 = 0:Y2 = 190:SP = 1
670 GOSUB1500: REM TRAZADO LINEA
690 :
695 GETJ$:IFJ$<>""THEN695
700 GETA$:IF A$ = ""THEN700:REM ESPERA OPRESION TECLA
710 :
720 REM ****RESTABLECER PANTALLA****
730 :
740 POKE HRSFLG,0: REM DESACTIVAR ALT. RES.
750 SYS BEGIN
760 PRINT CC$:PRINT:PRINT
770 PRINTTAB(9)"****FIN DE PROGRAMA****"
780 END
999 :
1000 REM *** S/R TRAZADO EN A. RES. ***
1010 :
1020 XHI = INT(X/HX):XLO = X−XHI*HX
1030 POKE XBYTE,XLO:POKE XPAGE,XHI:POKE YBYTE,Y
1035 SYS PLOT: REM ENTRADA S/R TRAZADO
1040 RETURN
1500 REM *** S/R TRAZADO LINEA ***
1550 C9 = (Y2−Y1)/(X2−X1):C8 = C9*X1−Y1
1600 FORX = X1 TO X2 STEP SP
1650 Y = X*C9−C8
1700 GOSUB 1000: REM DIBUJAR PUNTO
1750 NEXT X
1800 RETURN
3000 REM *** S/R INIC. ***
3020 CC$ = CHR$(147): REM LIMPIA PANTALLA
3025 HX = 256
3030 HRSFLG = 49408: REM $C100
3040 CLMFLG = 49409: REM $C101
3050 COLOUR = 49410: REM $C102
3060 XBYTE = 49411: REM $C103
3070 XPAGE = 49412: REM $C104
3080 YBYTE = 49413: REM $C105
3085 BEGIN = 49422: REM $C10E
3090 PLOT = 49539: REM $C183
3095 PRINT CC$:PRINT:PRINT
3100 PRINTTAB(9)"****CARGA LENG. MAQ.****"
3150 PRINTTAB(9)"1) LENG. MAQU. EN CINTA"
3200 PRINTTAB(9)"2) LENG. MAQU. EN DATA"
3250 PRINTTAB(9)"3) LENG. MAQU. EN MEM."
3300 PRINT "DIGITE NUMERO ESCOGIDO"
3350 FOR LP = 0 TO 1 STEP 0
3400 GET OP$
3450 IF OP$>"0" AND OP$<"4" THEN LP = 1
3500 NEXT LP
3600 ON VAL(OP$) GOSUB 4000,5000,6000
3900 RETURN
4000 REM** S/R CARGA LENG. MAQU. DESDE CINTA**
4100 PRINT "S/R INSERCION CINTA CON LENG. MAQU."
4200 IF A = 0 THEN A = 1:LOAD "PLOTSUB.HEX",1,1
4900 RETURN
```

```
5000 REM** S/R CARGA LENG. MAQU. DESDE DATA**
5005 PRINTTAB(11)"*****CARGANDOSE*****"
5010 FOR I = HRSFLG TO HRSFLG+312:READ A
5020 POKE I,A:S = S+A:NEXT I
5030 READ CC:IF CC<>S THEN PRINT"CHECKSUM ERROR":END
5040 DATA 2,0,255,255,2,2,255,255,2,18
5050 DATA 255,255,2,2,72,138,72,152,72
5060 DATA 173,0,193,240,83,169,0,133,251
5070 DATA 169,4,133,252,162,3,160,0,173
5080 DATA 2,193,145,251,136,208,251,230
5090 DATA 252,202,48,8,208,244,145,251
5100 DATA 160,231,208,238,173,1,193,240
5110 DATA 24,169,0,133,251,169,32,133
5120 DATA 252,162,32,160,0,169,0,145,251
5130 DATA 136,208,251,230,252,202,208
5140 DATA 246,173,24,208,41,240,9,8,141
5150 DATA 24,208,173,17,208,9,32,141,17
5160 DATA 208,76,125,193,173,24,208,41
5170 DATA 240,9,4,141,24,208,173,17,208
5180 DATA 41,223,141,17,208,104,168,104
5190 DATA 170,104,96,72,138,72,152,72
5200 DATA 173,4,193,141,7,193,173,3,193
5210 DATA 41,248,141,6,193,173,3,193,41
5220 DATA 7,141,8,193,173,5,193,41,7,141
5230 DATA 10,193,162,3,78,5,193,202,208
5240 DATA 250,173,5,193,141,9,193,169,0
5250 DATA 141,11,193,141,12,193,162,5
5260 DATA 173,11,193,24,109,9,193,141,11
5270 DATA 193,202,208,243,162,6,14,12
5280 DATA 193,14,11,193,144,3,238,12,193
5290 DATA 202,208,242,173,11,193,24,109
5300 DATA 6,193,141,11,193,173,12,193
5310 DATA 109,7,193,141,12,193,173,11
5320 DATA 193,24,105,0,141,11,193,173,12
5330 DATA 193,105,32,141,12,193,173,11
5340 DATA 193,24,109,10,193,141,11,193
5350 DATA 173,12,193,105,0,141,12,193
5360 DATA 173,11,193,133,251,173,12,193
5370 DATA 133,252,169,1,141,13,193,56
5380 DATA 169,7,237,8,193,170,14,13,193
5390 DATA 202,208,250,160,0,177,251,13
5400 DATA 13,193,145,251,76,125,193
5410 DATA 38698:REM*CHECKSUM*
5900 RETURN
6000 REM** S/R LENG. MAQU. EN MEM.**
6100 RETURN
```

Uso de PLOTSUB

Este programa BASIC de demostración ilustra las diversas fases necesarias para el uso de las rutinas para alta resolución en lenguaje máquina:

1) Si usted dispone de un ensamblador, puede digitar el programa en assembly, ensamblarlo, guardarlo como archivo fuente y a continuación guardar el código objeto contenido en $C100 hasta $C238 con el nombre de "PLOTSUB.HEX". No intente ejecutar la subrutina en esta fase, pues puede que la pantalla de alta resolución machaque el propio ensamblador, colgando la subrutina.

2) Para usar en un programa estas rutinas para alta resolución se debe bajar el tope de la memoria MEMTOP (véase línea 220 del programa) y cargar el código desde la cinta (subrutina de línea 4000).

3) Se puede guardar la subrutina de línea 5000, si se quiere, como un programa BASIC (denominado, p. ej., "MCODELOAD"). A la hora de usarlo baje el MEMTOP, cargue después y ejecute este MCODELOAD (lo que hará es cargar el lenguaje máquina en la memoria). Digite NEW y cargue a continuación el programa que desea ejecutar: las rutinas para alta resolución están ahora dentro de la memoria y pueden ser accedidas mediante las instrucciones SYS adecuadas.

4) El último grupo de datos de la subrutina 5000 es una suma de control: la suma que totaliza los datos anteriores. Si el programa se para con un "CHECKSUM ERROR", quiere decir que usted introdujo mal los datos y debe corregir el error.

```
;*****************************************
;*****************************************
;**                                     **
;**          RUTINA DE TRAZADO          **
;**                                     **
;**          EN ALTA RESOLUCION         **
;**                                     **
;**        PARA UN COMMODORE 64         **
;**                                     **
;*****************************************
;*****************************************
;
;
PTR     EQU $FB
MPBLO   EQU $00
MPBHI   EQU $20
SCBLO   EQU $00
SCBHI   EQU $04
SCBLK   EQU $03
SCREM   EQU $E7
MPBLK   EQU $20
        ORG $C100
;
HRSFLG  DB  $00
CLRFLG  DB  $00
COLOUR  DB  $00
XLO     DB  $00
XHI     DB  $00
YLO     DB  $00
HBLO    DB  $00
HBHI    DB  $00
REMX    DB  $00
VBYTE   DB  $00
REMY    DB  $00
ROWLO   DB  $00
ROWHI   DB  $00
BPOS    DB  $00
;****PASO A LA MOD. A. RES.*****
;****Y LIMPIEZA AREA MAPA BITS***
        PHA
        TXA
        PHA
        TYA
        PHA
        LDA HRSFLG
        BEQ RESET
;****AREA COLORES PANTALLA*******
        LDA #SCBLO
        STA PTR
        LDA #SCBHI
        STA PTR+1
        LDX #SCBLK
        LDY #$00
        LDA COLOUR
AGAIN STA (PTR),Y
        DEY
        BNE AGAIN
        INC PTR+1
        DEX
        BMI CLTEST
        BNE AGAIN
        STA (PTR),Y
        LDY #SCREM
        BNE AGAIN
CLTEST  LDA CLRFLG
        BEQ HRESON
****LIMPIEZA AREA MAPA BITS*******
        LDA #MPBLO
        STA PTR
        LDA #MPBHI
        STA PTR+1
        LDX #MPBLK
        LDY #$00
        LDA #$00
NEXT    STA (PTR),Y
        DEY
        BNE NEXT
        INC PTR+1
        DEX
        BNE NEXT
;****PASO A LA MOD. MAPA BITS*********
HRESON LDA $D018
        AND #$F0
        ORA #$08
        STA $D018
        LDA $D011
        ORA #$20
        STA $D011
        JMP EXIT
;****VUELTA A PANTALLA NORMAL***
RESET   LDA $D018
        AND #$F0
        ORA #$04
        STA $D018
        LDA $D011
        AND #$DF
        STA $D011
;****SALIDA DE LENG. MAQ. ******************
EXIT    PLA
        TAY
        PLA
        TAX
        PLA
        RTS
;****CALCULO TRAZO EN A. RES***
        PHA
        TXA
        PHA
        TYA
        PHA
;****CALCULO BYTE HORIZ.*****
        LDA XHI
        STA HBHI
        LDA XLO
        AND #$F8
        STA HBLO
        LDA HLO
        AND #$07
        STA REMX
;****CALCULO BYTE VERT.*****
        LDA YLO
        AND #$07
        STA REMY
        LDX #$03
SHIFT   LSR YLO
        DEX
        BNE SHIFT
        LDA YLO
        STA VBYTE
;****CALCULO FILA (ROW)************
        LDA #$00
        STA ROWLO
        STA ROWHI
        LDX #$05
        LDA ROWLO
        CLC
        ADC VBYTE
        STA ROWLO
        DEX
        BNE FIVE
        LDX #$06
        ASL ROWHI
        ASL ROWLO
        BCC NCARRY
        INC ROWHI
        DEX
        BNE MULT
;****SUMA BYTE HORIZ.******
        LDA ROWLO
        CLC
        ADC HBLO
        STA ROWLO
        LDA ROWHI
        ADC HBHI
        STA ROWHI
;****SUMA BASE MAPA A. RES.*******
        LDA ROWLO
        CLC
        ADC #MPBLO
        STA ROWLO
        LDA ROWHI
        ADC #MPBHI
        STA ROWHI
;****SUMA RESTO DE Y LO*****
        LDA ROWLO
        CLC
        ADC REMY
        STA ROWLO
        LDA ROWHI
        ADC #$00
        STA ROWHI
;****COMBINACION DE 2 BYTES*******
;****DE DIRECC. EN PAG. CERO*****
        LDA ROWLO
        STA PTR
        LDA ROWHI
        STA PTR+1
;****CALCULO POSIC. PIXEL****
        LDA #$01
        STA BPOS
        SEC
        LDA #$07
        SBC REMX
        TAX
        ASL BPOS
        DEX
        BNE POWER
;****ENCENDIDO DEL PIXEL************
        LDY #$00
        LDA (PTR),Y
        ORA BPOS
        STA (PTR),Y
        JMP EXIT
```

Para todo el mundo

Bug-Byte es una empresa de software que ha crecido sostenida y paralelamente a la industria británica de ordenadores personales

Bug-Byte se fundó en la primavera de 1980, cuando Tony Baden y Tony Milner, dos estudiantes de química del University College, de Oxford, empezaron a escribir programas para el ZX80 que se acababan de comprar. Comprendiendo que era muy poco el software existente para la máquina, decidieron comercializar sus juegos, compraron 40 cassettes vírgenes e hicieron publicidad en una revista de ordenadores anunciando un paquete de cinco juegos. Los pedidos empezaron a llegar en un promedio de 15 por semana y los socios reinvirtieron sus beneficios en más publicidad y escribieron otros programas para el ZX80. Luego, durante ese mismo año se lanzó al mercado el Acorn Atom, y Bug-Byte amplió su ámbito para satisfacer la demanda de software para la nueva máquina

Tony Milner

Tony Baden

A principios de 1981 apareció el ZX81 y la demanda de juegos para el ZX80 disminuyó enseguida, de modo que Baden y Milner se dedicaron a escribir software para el ZX81. Obtuvieron su licenciatura en Oxford en junio de 1981 y entonces Bug-Byte se trasladó a Liverpool, la ciudad natal de Tony Baden. A partir de entonces la empresa comenzó a trabajar con dedicación exclusiva y al cabo de poco tiempo duplicaron las ventas de programas.

En la Navidad de 1981 la competencia en el mercado de software de juegos se habían hecho más dura. Para mantener las ventas Bug-Byte contrató una agencia de publicidad para que se encargara de la comercialización y empezó a cuidar el anuncio de sus cassettes así como su presentación, hecha a todo color, utilizando al mismo tiempo una empresa de reproducción de cintas profesional para asegurarse cassettes de alta calidad.

Este nuevo enfoque a la presentación de sus productos, junto con la introducción de una red de distribuidores a nivel nacional, tuvo un marcado efecto sobre las ventas y la empresa contrató más personal. A medida que hacían su aparición más ordenadores personales, Bug-Byte contrataba programadores por cuenta propia para satisfacer la creciente demanda. Muchos de estos programadores abandonaron luego la colaboración con Bug-Byte para formar sus propias firmas de software, como Quicksilva y Software Projects. La empresa les paga a sus programadores un porcentaje fijo por cassette y afirma que un autor cuyo juego consiga situarse entre los veinte mejor vendidos puede ganar entre 10 000 y 40 000 libras en el primer año.

A finales de 1982 la red de ventas de Bug-Byte estaba compuesta por más de 200 distribuidores independientes, además de las más importantes cadenas de tiendas, concluyendo la época en que la empresa hacía sus ventas por correspondencia. El aprovisionamiento de cintas estaba resultando un gran dolor de cabeza, ya que las empresas de reproducción eran incapaces de satisfacer la demanda de las firmas de software que estaban preparando sus

Roger Sanders

Oficinas centrales de Bug-Byte, en Liverpool

existencias para el *boom* de ventas de la Navidad, por lo que Bug-Byte creó su propia empresa de reproducción, Spool. En junio de 1983 la empresa se trasladó a unas instalaciones más grandes, en Canning Place (Liverpool); éstas se diseñaron siguiendo las especificaciones de Bug-Byte.

Entre los mayores éxitos de Bug-Byte figuran el juego de aventuras con gráficos *Twin kingdom valley* (para el BBC y el Commodore 64) y *Manic miner*, que funciona con el ZX Spectrum y el Commodore 64. El autor de *Manic miner*, Matthew Smith, recientemente ha dejado la empresa para formar Software Projects, llevándose consigo los derechos de autor *(copyright)* del juego.

La expansión de Bug-Byte ha sido ininterrumpida y el software de la empresa en la actualidad se vende en la mayoría de los países de Europa occidental, así como en Australia, Nueva Zelanda y República de Sudáfrica. Un reciente acuerdo con la CBS británica, representante comercial de Bug-Byte en Europa, podría significar la penetración de la empresa en el lucrativo mercado norteamericano. John Phillips, director de marketing de Bug-Byte, ha manifestado a este respecto que "utilizando el contacto con CBS como modelo, esperamos conseguir que la operación tenga una proyección auténticamente mundial".

Útil compañía

La última generación de ordenadores portátiles convierte a su pionero, el Osborne, en una máquina incómoda y pesada

La definición de "portátil" se ha tenido que revisar desde que apareciera la última generación de ordenadores "de mano". En realidad, a los micros portátiles que se introdujeron hace unos pocos años ahora se los denomina "transportables". Los ordenadores que en la actualidad ofrecen una genuina portabilidad son los que llevan su propia fuente de alimentación eléctrica, visualización y dispositivos de almacenamiento en un paquete no más grande que una guía telefónica.

El Epson HX-20 fue uno de los primeros que ofrecieron este tipo de portabilidad, pero ahora su diminuta visualización en cristal líquido de 20 caracteres por 4 líneas, refleja la edad de la máquina. Los últimos portátiles, como el Tandy 100, el NEC PC-8201A y el Olivetti M10 tienen todos un precio similar pero pueden visualizar en sus pantallas cuatro veces más caracteres.

¿Y qué pueden hacer estos ordenadores? ¿Cuáles son sus ventajas y sus desventajas respecto de los micros convencionales de sobremesa? La razón más obvia para adquirir un portátil es la de disponer de un potencial informático completo en cualquier lugar y en cualquier momento. Muchas personas pasan gran parte de su tiempo alejadas del ordenador de su escritorio, y muchas horas improductivas transcurren en otras oficinas, habitaciones de hotel, aeropuertos y trenes. El ordenador portátil permite aprovechar este tiempo muerto.

La más reciente generación de portátiles proporciona un adecuado potencial informático personal para trabajos científicos y de ingeniería, contabilidad, administración financiera y tratamiento de textos; de hecho, prácticamente para cualquier aplicación para la que se utilicen los ordenadores personales convencionales.

Los ordenadores de mano suelen llevar al menos tres programas incorporados: un intérprete de BASIC, un programa para tratamiento de textos y software para comunicaciones. El Tandy 100 y el Olivetti también están equipados con programas de planificación y agenda de direcciones; así, el usuario puede disponer de señas, números de teléfono y las entrevistas previstas para cada día.

El programa para comunicaciones es sumamente importante porque permite que el portátil se ponga en contacto con otros micros y bases de datos a través de la red de teléfonos. Esta facilidad también puede convertir el micro portátil en un terminal de télex o receptor y transmisor de correo electrónico. Por supuesto, para conseguirlo se ha de emplear un modem o un acoplador acústico. De esta forma, un ejecutivo que no se halle en su despacho se puede mantener en contacto con su oficina central. Un periodista puede escribir su artículo *in situ* y transmitirlo inmediatamente al ordenador del periódico.

Los ordenadores portátiles más caros, como el Sharp PC-5000 y el Epson PX-8, utilizan los sistemas operativos MS-DOS y CP/M, que son comunes a sus equivalentes de sobremesa. Son, por consiguiente, capaces de ejecutar una amplia gama de software de oficina.

Tony Sleep

Sobre la marcha
La informática sobre la marcha se está haciendo cada vez más usual, principalmente entre los hombres de negocios. Algunos están utilizando la nueva generación de ordenadores "de mano" para ganar unos pocos minutos y emplearlos en tratamiento de textos durante las esperas en salas y aeropuertos antes de ir de un taxi a un tren y de éste al avión. Otros, como los vendedores, están abriendo un nuevo camino llevándose el ordenador consigo en sus visitas a clientes, pudiendo generar al instante presupuestos cuya preparación, de otra forma, ocuparía varios días. Los ejecutivos pueden, sobre la marcha, enviar datos a la oficina central utilizando un modem y las líneas telefónicas normales, o, al final del día, regresar a la oficina y enviar los datos directamente a un ordenador más grande

El Epson PX-8 viene con el popular paquete de tratamiento de textos Wordstar ya instalado en sus chips de ROM. El Sharp utiliza cartuchos conectables de memoria de burbujas que proporcionan 128 Kbytes de almacenamiento extra cada uno. Estos cartuchos tratan los datos a una velocidad mucho mayor que las unidades de disco.

Los portátiles menos caros utilizan programas para aplicaciones concretas que suelen cargarse en la RAM del ordenador desde cassette. Éste es un proceso mucho más lento que cargar desde cartucho de memoria de burbujas o desde unidad de disco. El NEC PC-8201A viene con una cassette que contiene varios programas de aplicaciones.

Informática a bordo
Todo hombre de negocios que desee utilizar su ordenador portátil durante un viaje en avión tendrá que poner especial atención al elegir la compañía aérea. Oficialmente, las autoridades de aviación civil afirman que los equipos electrónicos y a pilas pueden producir interferencias en los controles de vuelo de los aviones. Pero cada compañía interpreta estos informes de forma

distinta. La línea alemana Lufthansa y la australiana Qantas prohíben todo equipo eléctrico a bordo. La norteamericana Pan Am no permite radios ni grabadoras, pero sí autoriza juegos electrónicos y ordenadores. Japan Airlines no desautoriza equipos eléctricos de ninguna clase, a diferencia de las líneas británicas, que prohíben los equipos electrónicos

Entre ellos se incluyen una calculadora con memoria, un formateador de textos, un gestor de cartera de inversiones y un evaluador de préstamos. La calculadora con memoria convierte a la máquina en una calculadora que puede recordar hasta 99 entradas. El formateador de textos prepara para la impresión archivos entrados por el programa para tratamiento de textos, pudiendo especificar anchura de márgenes, división del texto en páginas, asignación de números de página, etc. El gestor de cartera de inversiones es para que lo empleen las personas que desean evaluar cómo marchan sus acciones y sus participaciones. Este programa analiza una cartera de hasta 50 inversiones, calculando pérdidas y beneficios.

Al igual que cualquier otro ordenador, el micro portátil se puede conectar a periféricos como impresoras, grabadoras de cassette y modems. Aparte del factor obvio de tamaño y peso, la prueba de fuego de un auténtico portátil es que ha de funcionar a pilas, tener su propia visualización y transportar consigo en ROM sus programas para tratamiento de textos y comunicaciones.

Hay máquinas, como el Apple IIc y el Apricot, que se anuncian como portátiles. Pero éstas no se pueden utilizar en tránsito, ya que se tienen que enchufar a una toma de corriente, conectar a una pantalla y cargar sus programas en RAM desde disco. Prescindiendo de su tamaño más pequeño y de su menor peso, estas máquinas tienen más cosas en común con el micro de sobremesa que con el ordenador de mano a pilas.

Además de sus pilas principales, los ordenadores portátiles están equipados con pequeñas pilas de níquel-cadmio recargable que pueden proporcionar energía en casos de emergencia. Esto es esencial, ya que de no contar con este apoyo se perderían todos los datos si se produjera un fallo de las pilas principales.

La mayoría de los portátiles también poseen una interface para lector de códigos de barras, de modo que se los puede utilizar para el control de existencias. El lector de códigos de barras se pasa por encima del código de barras impreso en los paquetes de productos. Este decodifica la información relativa a precio y fecha, que puede ser procesada por el ordenador para proporcionarles a los comerciantes una lectura exacta del inventario de su stock. De las máquinas que hemos ilustrado en estas páginas, el Tandy Modelo 100, el NEC PC-8201A y el Olivetti M10 vienen equipados con lector de códigos de barras, pero el Casio FP 200 no lo posee. Las tres primeras máquinas en realidad son muy similares en numerosos aspectos, porque todas están hechas en la misma fábrica japonesa. Las diferencias significativas entre ellas son que la Olivetti tiene una pantalla inclinable, la NEC tiene menos software incorporado y las memorias del Tandy y del Olivetti no se pueden ampliar a más de 32 Kbytes, mientras que el NEC se puede ampliar hasta 64 Kbytes. Asimismo, el NEC utiliza cartuchos de memoria de 32 Kbytes intercambiables que retienen sus datos aun cuando se desconecten del ordenador. Todas las mejoras de memoria para ordenadores manua-

Epson HX-20
Aunque posee una pantalla pequeña, el HX-20 tiene la ventaja de llevar incorporadas una grabadora de cassette y una diminuta impresora. También incluye un modesto procesador de textos

Casio FP-200
El Casio es uno de los ordenadores de mano más baratos, pero carece de un procesador de textos incorporado. En cambio, ofrece una versión de hoja electrónica

les son caras debido al tipo de chip que se emplea.

Así como las máquinas de escribir portátiles son complementarias de las máquinas de escribir de oficina y no constituyen un sustituto de ellas, del mismo modo los micros portátiles no están llamados a reemplazar al ordenador personal de sobremesa. Por mencionar alguna razón, sus pequeñas visualizaciones en cristal líquido (LCD) limitan sus idoneidad para prolongadas sesiones frente al teclado. La visualización LCD es más difícil de leer y más lenta que el terminal de rayos catódicos.

A diferencia de los micros más grandes con sus teclados inclinados, los ordenadores portátiles más económicos poseen teclados planos cuya utilización es incómoda para el usuario. Además los portátiles del extremo inferior del mercado no pueden ejecutar los programas de oficina basados en disco.

Pero, con todo, el ordenador portátil de mano está aquí y su permanencia está asegurada. La difusión del empleo de micros les está mostrando a muchas personas cómo el potencial del ordenador las puede ayudar a llevar sus vidas con mayor eficacia. El portátil les permite acceder a este potencial estén donde estén.

Epson PX-8
Puede utilizar software de gestión del tipo CP/M incluyendo el insuperable procesador de textos Wordstar, que viene con la máquina

Tandy TRS-80 Modelo 100/NEC PC-8201A/Olivetti M10
Constituyen distintas versiones reempaquetadas del mismo micro. Comparten un excelente procesador de textos incorporado y una buena gama de interfaces. En la fotografía también vemos un modem a pilas de Olivetti

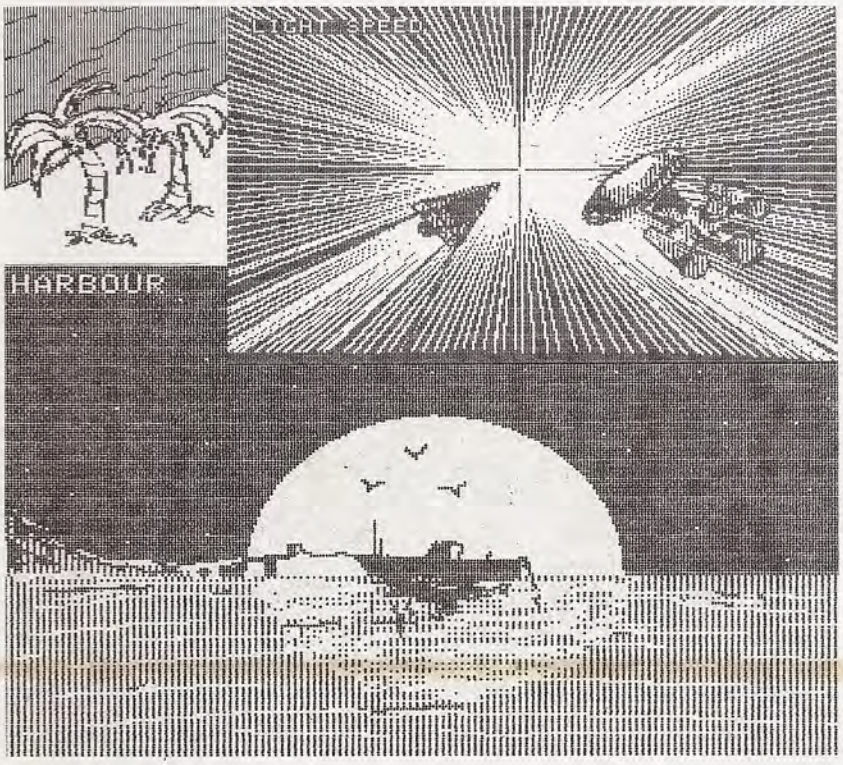
Modelo	Memoria estándar	Memoria máxima	Pantalla	Peso
Casio FP-200	8 K	32 K	8 × 20	1,4 kg
Epson HX-20	16 K	32 K	4 × 20	1,8 kg
Epson PX-8	64 K	64 K + 120 K*	8 × 80	2,3 kg
NEC PC-8201A	16 K	64 K + 32 K*	8 × 40	1,8 kg
Olivetti M10	8 K	32 K	8 × 40	1,8 kg
Tandy TRS-80 Modelo 100	8 K	32 K	8 × 40	1,8 kg

* El NEC puede usar un cartucho de RAM de 32 K y el Epson PX-8 un disco de RAM de 120 K.

Marcus Wilson-Smith

Primeras impresiones

En este capítulo mostramos cómo preparar una impresora para producir atractivos gráficos y cómo crear un programa de "vuelco de pantalla"



Simon Dayton

Impresión por pantalla
Estos diseños se dibujaron en pantalla utilizando una pastilla para gráficos. El contenido de las pantallas se volcó luego a una impresora Epson FX-80, mostrando las posibilidades para gráficos de una impresora matricial

La mayoría de los ordenadores personales poseen una modalidad de gráficos en baja resolución en que las imágenes se construyen en la pantalla a partir de caracteres para gráficos, cada uno de éstos de igual tamaño que un carácter para texto convencional. Estos caracteres de "bloques" poseen códigos de caracteres superiores a 127, ya que los números del 0 al 127 están reservados para el juego de caracteres ASCII. De modo que PRINT CHR$(90) imprimirá en la pantalla un carácter ASCII ("Z", en este caso), mientras que PRINT CHR$(128) visualiza un carácter para gráficos (o un rectángulo negro si se está utilizando un micro Dragon).

Para imprimir la letra "Z" en una impresora digitaríamos LPRINT CHR$(90), por lo que se podría pensar que, del mismo modo, LPRINT CHR$(128) imprimirá en el papel un rectángulo negro. Pero, lamentablemente, no es éste el caso. Ello se debe a que los caracteres por encima del código 127 varían muchísimo de una marca a otra de micro y, obviamente, los fabricantes de impresoras no pueden producir una impresora especial para cada ordenador que haya en el mercado. Lo que los fabricantes tienden a hacer es o bien copiar el juego ASCII estándar en los códigos del 128 al 255 o, alternativamente, programar sus propios caracteres.

La gama de impresoras Epson no viene con ningún carácter para gráficos. En cambio, el usuario puede cambiar cualquiera de los caracteres ASCII estándar para producir sus propios caracteres para gráficos. Esto se consigue enviándole a la impresora "códigos de escape" (véanse pp. 804 y 805).

Los gráficos de ordenador en alta resolución se construyen a partir de pequeños puntos o pixels y no a partir de caracteres enteros. De un modo similar, la impresión en alta resolución utiliza pequeños puntos de tinta. El cabezal de impresión de una impresora matricial posee cierta cantidad de agujas dispuestas en una línea vertical que se desplaza a través del papel a medida que va imprimiendo. Por lo general, los caracteres se componen en una matriz de puntos (de 8 × 8 puntos, p. ej.). No obstante, se pueden producir gráficos controlando las agujas individualmente.

El primer paso consiste en colocar la impresora en modalidad para gráficos. Al igual que si se tratara de cualquier otro ejercicio de impresión, esto se realiza enviándole un código de escape que es específico para el tipo de impresora que se está utilizando. En la Epson FX-80, por ejemplo, las instrucciones necesarias son:

```
LPRINT CHR$(27);"K";CHR$(N1);(N2);
```

La letra "K" indica modalidad de gráficos y los números (N1) y (N2) establecen la anchura de cada línea de gráfico; en otras palabras, el número de puntos que entrarán a lo ancho de la página.

Estando en modalidad de gráficos estándar, la FX-80 puede imprimir un máximo de 480 puntos en una línea. Otras modalidades permiten resoluciones desde 576 a 1 920 puntos por línea. Por lo tanto, si deseamos utilizar la anchura total, la longitud de línea requerida será 480. En nuestro código son necesarios dos números para establecer la anchura porque el tamaño máximo de cada número es 255. El segundo número (N2) se multiplica, por consiguiente, por 256 y se le suma al primero, (N1). De modo que para 480 los números son 1 y 224 (480 = 256 × 1 + 224). En consecuencia, en la Epson FX-80 se requiere la siguiente instrucción:

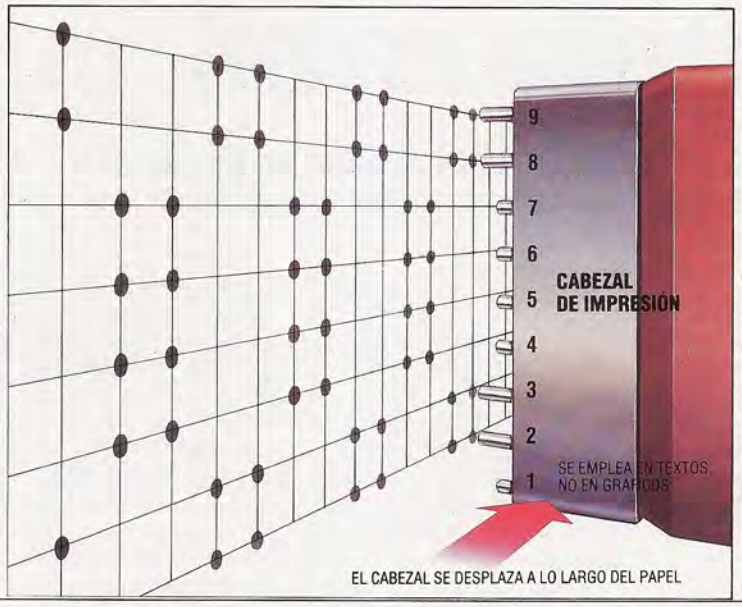
```
LPRINT CHR$(27);"K";CHR$(224);CHR$(1);
```

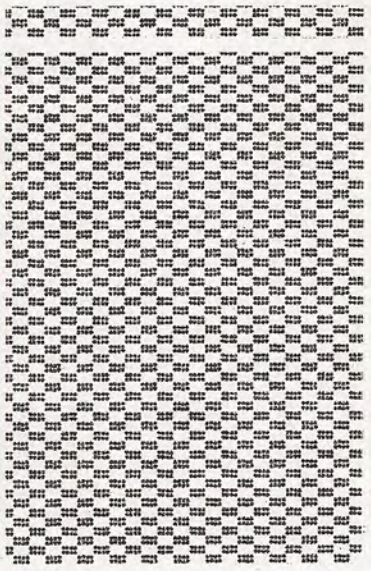
Habiendo programado la impresora con la longitud de línea de gráficos, sólo falta enviar los datos del gráfico. Aun cuando en el cabezal de impresión de una Epson FX-80 hay nueve agujas, en la mayoría de las modalidades para gráficos únicamente se pueden emplear las ocho superiores. Empezando desde la patilla inferior, las numeramos 1, 2, 4, 8, 16, 32, 64 y 128. Los datos para las ocho agujas se pueden representar, entonces, mediante un único número, entre 0 y 255, y éste se le envía a la impresora utilizando LPRINT CHR$(X), donde X es el número. De modo que si sólo deseáramos "disparar" la aguja inferior, enviaríamos CHR$(1) a la impreso-

Puntas de agujas
El patrón se produjo en una impresora matricial enviándole al cabezal de impresión pares alternos de los números decimales 195 y 60. El cuadro (abajo) muestra cómo son interpretados estos números en binario por las agujas del cabezal de impresión (ilustradas a la derecha). El salto de papel controlado hace que la línea siguiente se sobreimprima en el vacío dejado por la aguja 1

NÚMERO AGUJA	VALOR AGUJA		
9	128	•	○
8	64	•	○
7	32	○	•
6	16	○	•
5	8	○	•
4	4	○	•
3	2	•	○
2	1	•	○
		195	60

• LA AGUJA SE DISPARA
○ LA AGUJA NO SE DISPARA





Steve Cross

ra; para activar sólo la aguja superior enviaríamos CHR$(128). Para una combinación de patillas simplemente vamos sumando los números de cada aguja. Este proceso se repite luego para cada una de las 480 posiciones de pixel de la página.

En la ilustración hay dos patrones de agujas: CHR$(195) y CHR$(60). Así, para imprimir las cuatro primeras columnas del patrón de línea se digita:

```
LPRINT CHR$(195);CHR$(195);CHR$(60);CHR$(60);
```

Después de cuatro columnas el patrón se repite, por lo cual un bucle FOR...NEXT se ocupa del resto de la línea.

Es importante comprender que, en el ejemplo, CHR$(60) no indica a la impresora que imprima el carácter ASCII con código 60: es una manera de representar los datos para las agujas del cabezal de impresión. La impresora lo reconoce como tal porque previamente hemos transmitido la secuencia CHR$(27);"K" para activar la modalidad de gráficos.

Este método de impresión, que se conoce como *impresión de imagen de bits*, se describe aquí para una impresora Epson FX-80; otras impresoras utilizan un procedimiento similar, pero los detalles exactos pueden variar. Producir gráficos de esta forma es bastante laborioso y sólo es realmente adecuado para patrones. Una forma mucho mejor de imprimir gráficos es mediante un *vuelco de pantalla*. Éste es un programa que copia en el papel lo que está visualizado en la pantalla.

Explorando a lo ancho y a lo alto de la visualización en pantalla, el programa verifica si el pixel está encendido en cada posición. Si lo está, entonces es necesario que una aguja del cabezal de impresión se dispare en la posición correspondiente del papel. La exploración se realiza utilizando la función POINT(x,y) o instrucciones similares disponibles en la mayoría de micros; si un pixel está iluminado, la función es 1; si no lo está, ésta corresponderá a 0. Las distintas resoluciones de los diferentes micros implican la realización de algunos ajustes.

Una pregunta que quizá se le ocurra es: ¿cómo manipula un programa de vuelco de pantalla las visualizaciones en color? La solución habitual consiste en utilizar distintos patrones de puntos para cada color. Un pixel de pantalla que fuera negro se podría imprimir aplicando cuatro puntos en forma de cuadrado; uno que fuera rojo se podría representar mediante dos puntos, y uno que fuera blanco no utilizaría ningún punto. La función POINT(x,y) produce un número diferente según el color del pixel.

Los programas para vuelco de pantalla se suelen añadir en forma de subrutinas al final del programa que produce la imagen. Para "volcar" la imagen a la impresora se podría pulsar la tecla "P": el programa saltaría, entonces, a la subrutina. Un programa para vuelco de pantalla escrito en BASIC tiende a ser bastante lento, tardando alrededor de cinco minutos en imprimir una pequeña imagen. Las versiones en lenguaje máquina son ligeramente más rápidas.

Como puede verse, las capacidades para gráficos de las impresoras matriciales son avanzadas, aunque un poco incómodas de utilizar. Sin embargo, una vez dominadas, la página impresa puede ser tan atractiva como la visualización en pantalla.

Salpicones de color
Esta imagen del teclado del Spectrum se produjo en una impresora de chorro de tinta en color; en realidad se trata de una impresora matricial en la cual las agujas se han sustituido por chorros de tinta

Ian McKinnell

Batalla naval

He aquí un interesante programa de juegos para la red más asequible para el usuario: Sinclair ZX Net

La batalla naval es un juego clásico que se acostumbra jugar con lápiz y papel. Cada jugador posee dos cuadrículas, que mantiene ocultas a su adversario. En una de ellas tiene señalados sus propios barcos, en la otra va marcando sus resultados a medida que "dispara" contra los barcos de su oponente; es decir, se intenta adivinar la posición de éstos.

El programa que hemos desarrollado funciona con dos Sinclair Spectrum conectados mediante una red. Ambas máquinas han de estar equipadas con la Interface 1. Cada jugador se sienta frente a su propia pantalla, y los dos ordenadores se envían mensajes que informan hacia dónde están disparando los jugadores y qué resultados obtienen.

La primera dificultad que se encuentra es el identificar a los jugadores. Con el fin de comunicarse, cada Spectrum posee un número de estación de red diferente. Los dos ordenadores empiezan con programas idénticos pero de alguna manera deben concluir con números de estación diferentes. Esto se trata automáticamente mediante una rutina en la línea 2000. Cuando el programa se ejecuta (RUN), las dos máquinas intentarán ser la estación 1 en el canal de "transmisión" público.

La máquina que primero empiece a ejecutar el programa se convertirá en la estación 1 y a la otra le corresponderá ser la estación 2. Este sistema funciona bien en batallas navales. El programa entonces da por sentado que quien sea que esté en la estación 1 es el jugador 1 y, por consiguiente, le permite disparar primero. Sin embargo, si los dos programas se inician separados por una fracción de segundo, los dos mensajes "Soy la estación número 1" colisionan y el sistema ZX Net dejará de funcionar hasta que los jugadores pulsen la tecla BREAK.

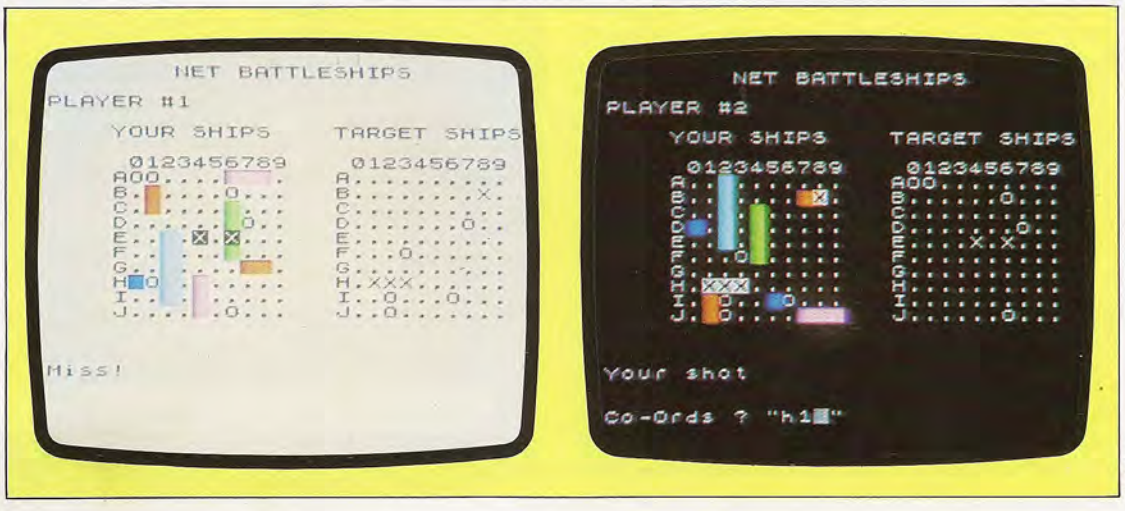
Una vez que se sabe quién es quién, para el programa es fácil comunicarse con su número oponente a través de la red. Mientras un jugador está escogiendo en su máquina un cuadrado al cual disparar, el otro está esperando conocer su elección. Los ordenadores intercambian sus funciones. Una máquina calcula los resultados del disparo y devuelve un mensaje, mientras la otra espera recibir el resultado y actualizar su visualización en pantalla en consecuencia.

Siempre y cuando se logre que los dos programas se complementen (uno enviando, el otro recibiendo), el juego se desarrollará con fluidez y sencillez. No es necesario preocuparse por la sincronización, por enviar mensajes demasiado tarde ni por perderlos después de haberlos enviado, ya que el ZX Net se interrumpe hasta que las dos estaciones están preparadas y luego transmite los datos. De modo que no tiene importancia que un jugador sea demasiado lento al seleccionar un objetivo ni que el programa tarde mucho tiempo en actualizar su pantalla.

Hay otro punto que es interesante destacar: la cantidad de datos que se transmiten se debe mantener en un mínimo. Sin embargo, no hay necesidad de enviar grandes bloques de datos. Siempre y cuando ambos programas sepan lo que significa la información, usted se puede comunicar utilizando códigos cortos. En las batallas navales, el programa devuelve el resultado de un disparo como una serie de dos caracteres. El primer carácter es un código:

1 Agua
2 Barco tocado
3 Barco tocado y hundido
4 Barco tocado y hundido y juego ganado

El segundo carácter es la clase de barco que ha sido tocado (o un cero para agua). El programa al otro lado puede decodificar esta información y visualizar mensajes adecuados. Este método hace que el tiempo que ocupa ejecutar cada jugada sea tan breve, que no parece en absoluto que haya otro ordenador implicado en el asunto.



Liz Heaney

Flotas en acción
La flota de cada jugador está formada por barcos de diferentes tamaños (desde lanchas torpederas hasta portaaviones) situados en cualquier lugar de la cuadrícula. La pantalla visualiza la posición y el estado de los barcos de un jugador y los disparos que el oponente ha efectuado contra ellos, además de la posición y el efecto de sus disparos contra la escuadra del enemigo: "X" indica tocado, "O" indica agua

Desarrollo del juego

Este programa requiere un procedimiento de comienzo ligeramente complicado, porque se ejecuta por separado en dos Spectrum. Se debe cargar una copia del programa en cada ordenador. Las dos se deben cargar desde cassette, pero es mucho más rápido cargar un Spectrum desde cassette (o microdrive) y después transmitirle el programa a la otra máquina a través de la red. Para hacer esto, digite LOAD *"n";0 en el receptor y SAVE* "n";0 en la máquina que tiene el programa en memoria. A continuación, los jugadores han de decidir cuál de ellos va a disparar primero. Este jugador debe lanzar (RUN) el programa un poco antes que el segundo jugador. El programa luego les asigna números de red a ambas máquinas y averigua qué copia del programa está jugando en primer lugar y cuál en segundo.
Cuando empieza el programa, los dos jugadores deben fijar las posiciones de sus barcos. Esto se realiza especificando la situación de un extremo de cada navío en la cuadrícula del juego de 10 × 10 y diciendo si el resto del barco está hacia arriba, hacia abajo, a izquierda o derecha de esa posición. Esto suena complicado pero en la práctica es conveniente. Cada jugador posee dos lanchas torpederas (MTB) (1 cuadrado de longitud), dos cruceros (2 cuadrados de largo), dos acorazados (3), un destructor (4) y un portaaviones (5).
Los jugadores entonces se turnan para disparar contra un cuadrado de la cuadrícula del otro y el programa calcula el resultado de cada disparo. Gana quien antes destruye todos los barcos del oponente. Para volver a jugar, ambos jugadores tienen que digitar RUN, recordando que el que desee empezar primero debe pulsar RUN antes que el otro

```
 10 REM juego de la batalla naval por red
 11 REM
 12 REM 2 Spectrum con Interface 1
 13 REM Junio/84/Version 1.6
 15 REM iniciar todo
 30 GO SUB 2000: REM lucha por la red
 40 LET s$ = "                              ": REM 34 espacios
 50 DIM s(8): REM tipos de barcos
 60 DIM n$(8,12): REM nombres barcos
 70 DIM a(10,10): REM papel cuadriculado!
 80 FOR i = 1 TO 8: READ s(i),n$(i): NEXT i
 90 DATA 1,"MTB",1,"MTB",2,"Crucero",2,"Crucero"
100 DATA 3,"Acorazado",3,"Acorazado",4,"Destructor",
    5,"Portaaviones"
110 LET sc = 8
120 GO SUB 3000: REM inic pantalla
130 GO SUB 4000: REM colocar barcos
140 REM ahora saltar segun que
150 REM jugador seamos. #1 dispara primero
160 IF sta = 2 THEN GO TO 400
200 REM ***Disparar!
210 LET m$ = "Su disparo": GO SUB 6000
220 GO SUB 7000: IF e = 1 THEN GO TO 210
230 OPEN #4;"n";el
240 PRINT #4;a$
250 CLOSE #4
260 REM esperar & obtener resultado
270 OPEN #4;"n";el
280 INPUT #4;a$
290 CLOSE #4
300 LET r = VAL (a$(1 TO 1)): LET x = VAL (a$(2 TO))
310 LET r = 1 THEN LET m$ = "Agua!":PRINT AT
    6 + q,18 + p;"0";:GO SUB 6000
320 IF r > 1 THEN LET m$ = "Tocado!": PRINT AT
    6 + q,18 + p;"X";: GO SUB 6000
330 IF r > 2 THEN LET m$ = "Ha hundido un "+ n$(x) + "enemigo":
    GO SUB 6000
340 IF r = 4 THEN LET m$ = "Felicitaciones ... ha ganado!!!":GO SUB
    6000: STOP
400 REM ***Disparo del enemigo
410 LET m$ = "Enemigo disparando": GO SUB 6000
420 OPEN #4;"n";el
430 INPUT #4;a$
440 CLOSE #4
450 LET p = VAL (a$(2 TO)) + 1: LET q = CODE (a$)−64
460 LET m$ = "El enemigo dispara a "+ a$: GO SUB 6000
470 LET x = a(p,q): LET a(p,q) = 0
480 IF x = 0 THEN LET r = 1: GO TO 530
490 LET r = 2
500 LET s(x) = s(x)−1
510 LET s(x) = 0 THEN LET r = 3:LET sc = sc−1
520 IF sc = 0 THEN LET r = 4
530 LET a$ = STR$ (r) + STR$ (x)
540 OPEN #4;"n";el
550 PRINT #4;a$
560 CLOSE #4
570 IF r = 1 THEN LET m$ = "Es agua": PRINT AT 6 + q,4 + p;"0";
580 IF r > 1 THEN LET m$ = n$(x) +"tocado": PRINT AT
    6 + q,4 + p;"X";
585 IF r > 2 THEN LET m$ = m$ +"y hundido"
587 GO SUB 6000
590 IF r = 4 THEN LET m$ = "Lo siento...ha perdido": GO SUB 6000:
    STOP
600 GO TO 210
2000 REM ***decidir quien es quien
2005 CLOSE #4
2010 OPEN #4;"n";0
2020 PRINT #4;"1"
2030 CLOSE #4
2040 OPEN #4;"n";0
2045 INPUT #4;a$
2050 CLOSE #4
2060 IF a$ = "1" THEN OPEN #4;"n";0: PAUSE 5:PRINT #4;"2": LET
     sta = 1
2070 IF a$ = "2" THEN LET sta = 2
2080 CLOSE #4
2090 FORMAT "n";sta: LET el = 3−sta: RETURN
3000 REM ***preparar pantalla
3010 LET col = 0: IF sta = 2 THEN LET col = 7
3020 PRINT;: BORDER 7−col: PAPER 7−col: INK col: CLS
3030 PRINT TAB4;"BATALLAS NAVALES RED"
3040 PRINT: PRINT "JUGADOR#";sta
3050 PRINT: PRINT "      BARCOS SUYOS      BARCOS ENEMIGOS"
3060 PRINT: PRINT "      0123456789      0123456789"
3070 FOR i = 1 TO 10
3080 PRINT "      ";CHR$ (i + 64;".....      ";CHR$
     (i + 64);"....."
3090 NEXT i
3100 RETURN
4000 REM ***Preparar barcos
4010 LET m$ = "Por favor coloque sus barcòs";GO SUB 6000
4020 FOR s = 1 TO sc
4030 LET m$ = STR$ s + "." + n$(s) + "long." + STR$ (s(s)):GO
     SUB 6000
4050 GO SUB 7000: IF e = 1 THEN GO TO 4030
4055 IF s(s) = 1 THEN LET xd = 0: LET yd = 0: GO TO 4130
4070 INPUT "A, B, I o D?";a$: LET xd = 3: LET yd = 3
4080 IF a$ = "A" OR a$ = "a" THEN LET xd = 0: LET yd = −1
4090 IF a$ = "B" OR a$ = "b" THEN LET xd = 0; LET yd = 1
4100 IF a$ = "I" OR a$ = "i" THEN LET xd = −1: LET yd = 0
4110 IF a$ = "D" OR a$ = "d" THEN LET xd = 1: LET yd = 0
4120 IF xd = 3 AND yd = 3 THEN GO TO 4070
4130 LET l = s(s): LET x = p: LET y = q
4140 IF x < 1 OR x > 10 OR y < 1 OR y > 10 THEN LET m$ = "Alejar
     el barco del borde": GO SUB 6000: GO TO 4030
4150 IF a(x,y)<>0 THEN LET m$ = "Por favor vuelva a colocar el
     barco": GO SUB 6000: GO TO 4030
4160 LET x = x + xd: LET y = y + yd
4170 LET l = l−1: IF l > 0 THEN GO TO 4140
4180 LET l = s(s): LET x = p: LET y = q
4190 LET a(x,y) = s: INK s(s): PRINT AT 6 + y,4 + x;" ";: INK col
4200 LET x = x + xd: LET y = y + yd
4210 LET l = l−1: IF l > 0 THEN GO TO 4190
4220 NEXT s
4230 LET m$ = "Listos para la acción!!!": GO SUB 6000
4240 RETURN
6000 REM ***IMPRIMIR m$
6010 PRINT AT 20,0;s$;AT 20,0;m$: PAUSE 100: RETURN
7000 REM ***Verificar coordenadas
7010 LET e = 0
7015 INPUT "Coord.?";a$
7020 IF LEN a$<>2 THEN LET e = 1: GO TO 7100
7030 FOR i = 1 TO 2
7040 LET c = CODE (a$(i TO i)): IF c > = 97 AND c < = 122 THEN
     LET a$(i TO i) = CHR$ (c−32)
7050 NEXT i
7060 LET q = CODE (a$(1 TO 1)): LET p = CODE (a$(2 TO 2))
7070 IF q < 65 OR q > 74 THEN LET x = q: LET q = p: LET p = x
7080 IF q < 65 OR q > 74 THEN LET e = 1
7090 IF p < 48 OR p > 57 THEN LET e = 1
7100 IF e = 1 THEN LET m$ = "Por favor vuelva a entrar las coord.":
     GO SUB 6000: RETURN
7110 LET q = q−64: LET p = p−47
7120 RETURN
```

Test en cascada (1)

En esta ocasión efectuaremos una comparación entre un valor variable y unos valores constantes

Un test en cascada consiste en una serie de preguntas continuas eliminatorias que tienen como finalidad comparar el contenido de una variable con unos valores constantes, para seguir por una u otra vía de salida de la secuencia en caso de igualdad. Tal vez el ejemplo más sencillo sea el de saber en qué día de la semana nos encontramos a base de preguntarnos el número de orden que ocupa (figura 1). Puede observarse que sólo se han empleado seis preguntas. Ello se debe a que la séptima, en este caso, es innecesaria.

Véase a continuación (figura 2) un supuesto en el que una máquina pesa unos cojinetes que llegan mezclados hasta ella, luego de haber pasado previamente por otras máquinas. Dicha máquina pesa los cojinetes uno por uno. Si su peso es de 10 g, los envía al almacén 1; si pesan 25 g, los remite al almacén 2, y si su peso es de 50 g, los envía al almacén 3. Aquellos cojinetes que no se ajusten a los pesos prefijados, son desechados y desviados a un contenedor especial, destinado a los productos defectuosos.

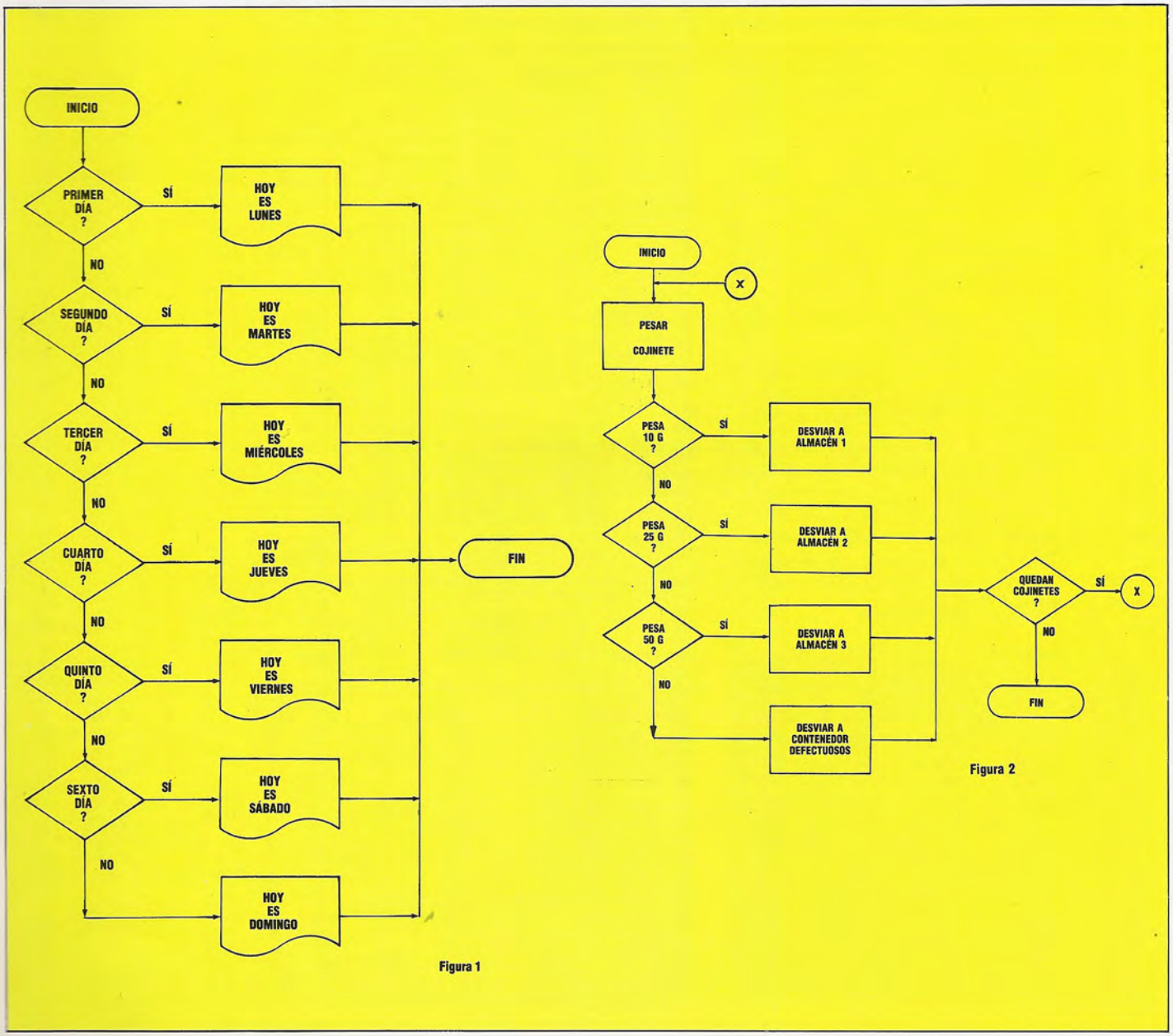
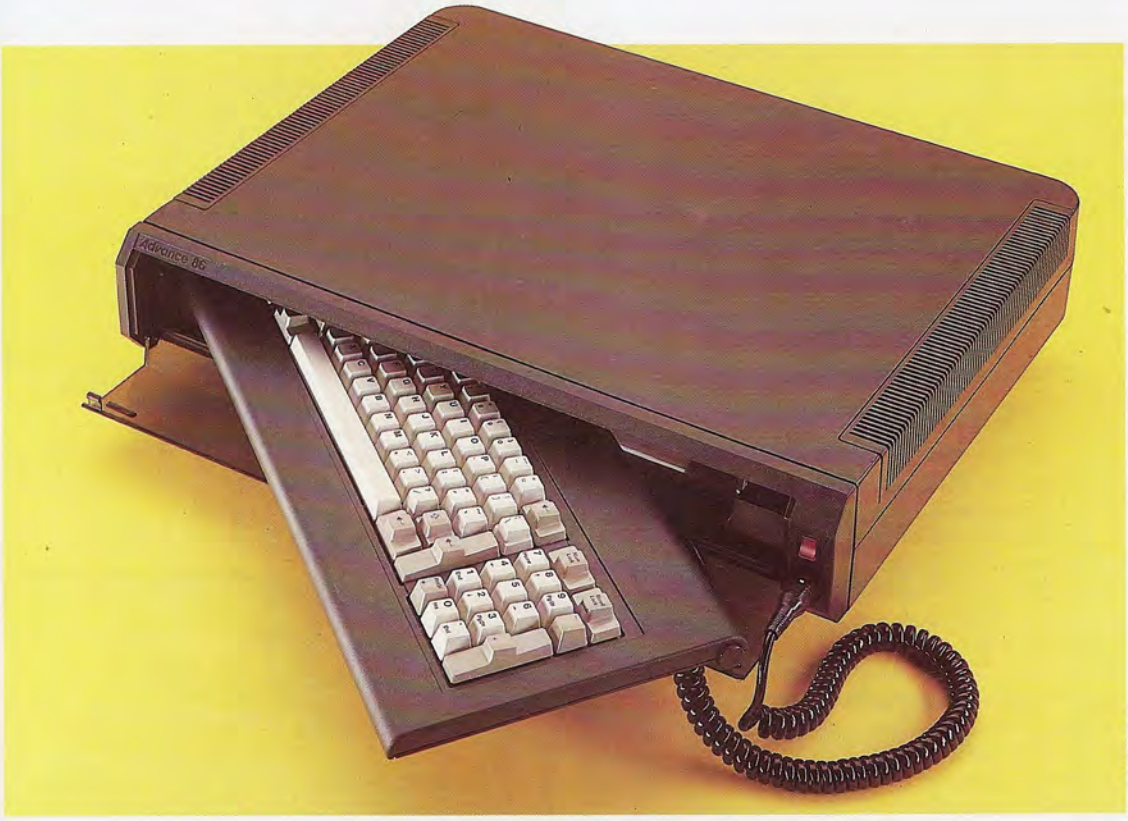


Figura 1

Figura 2

Un micro avanzado

El micro personal Advance 86 permite como pocos la posibilidad de transformarse en una máquina de oficina



Chris Stevens

Advance 86a
El Advance 86 se vende en dos versiones: el Advance 86a, que vemos en la fotografía y que se comercializa como ordenador personal, y el Advance 86b, micro de oficina compatible con el IBM

El Advance 86 se suministra en dos configuraciones: el 86a, microordenador personal basado en cassette cuyo precio está al nivel del BBC Micro pero que puede presumir de un total de 128 Kbytes de RAM, y el 86b, una máquina similar, aumentada por unidades de disco gemelas de 5 ¼" y con un BASIC más potente. Los usuarios del 86a lo pueden ampliar al nivel del 86b con sólo acoplarle un "paquete de ampliación" que contiene las unidades. El resultado es una máquina de oficina compatible con el IBM, por la mitad del precio del IBM PC.

El 86b se fabrica en dos partes: el teclado y la caja del microprocesador. Este último es considerablemente más grande de lo que sería necesario, midiendo 520 × 400 × 95 mm, y viene dentro de una carcasa de plástico negro con una puerta de metacrilato ahumado estilo equipo de alta fidelidad, que permite alojar en su interior el teclado. Todos los conectores se montan en esta unidad y el transformador de potencia está ubicado dentro de ella.

El teclado se conecta a la unidad del procesador mediante un cable coaxial y un conector DIN de cinco patillas, sobre el cual está el interruptor on/off con un indicador LED. El teclado es, por cierto, el mejor de todos los ordenadores personales existentes y es considerablemente mejor que los de la mayoría de las máquinas de oficina. Posee 84 teclas dispuestas en tres grupos: las principales teclas alfanuméricas, un conjunto de 10 teclas de función y un teclado numérico que desempeña también la función de un amplio conjunto de controles del cursor.

Las teclas de función proporcionan algunas de las funciones que se utilizan más comúnmente (RUN, LIST, SAVE, LOAD, etc.). Es muy fácil modificar estas funciones; a cada tecla se le asigna una serie de hasta 15 caracteres para identificar la instrucción, y la línea inferior de la pantalla visualiza una etiqueta de seis letras para cada una. El teclado numérico se controla mediante un interruptor de palanca rotulado como "Num Lock". En uso normal, el teclado actúa como una calculadora, pero la pulsación de Num Lock proporciona un efecto totalmente diferente: las teclas controlan entonces el cursor, permitiendo desplazarlo por la pantalla con precisión y velocidad.

En el interior de la caja del microprocesador hay una placa de circuito impreso relativamente pequeña que contiene el microprocesador de 16 bits Intel 8086 (compatible con el 8088 del IBM, pero más rápido) y 128 Kbytes de RAM. También están instalados los conectores que permiten doblar la RAM, aunque la memoria disponible para BASIC se limita a 62 Kbytes. Esto apenas si es un inconve-

Teclado inteligente
Se dice que el IBM Personal Computer cuenta con el mejor teclado entre los micros existentes. El teclado del Advance imita en gran medida el diseño de éste, aunque, de manera insólita, posee dos teclas Return y algunas teclas están cambiadas de sitio

niente, porque esta cantidad es más que suficiente para la mayoría de las aplicaciones.

El Advance está muy bien provisto de conectores e interfaces. Entre éstos se incluyen un conector especial que hace posible que un televisor o una pantalla funcionen con la fuente de alimentación eléctrica del ordenador; un conector que permite utilizar como visualización un televisor; tomas de video compuesto y RGB para monitores de video compuesto o bien de RGB (*red, green, blue:* rojo, verde, azul); una interface Centronics estándar para conexión a una impresora en paralelo; dos puertas para palanca de mando y un conector DIN de cinco patillas para utilizar con una grabadora de cintas. Cuando se usa la ampliación, se dispone de un conector RS232 para conectar una impresora en serie o un modem. Todas las interfaces aceptan cables de tipo IBM.

En modalidad de textos, el Advance visualiza 25 líneas de 40 caracteres o bien una pantalla del tipo IBM de 80 × 25; esta última apenas si es legible a menos que se emplee un monitor. La línea inferior de la pantalla normalmente visualiza rótulos de teclas de función, pero también se pueden apagar para disponer de la pantalla completa. En esta modalidad se pueden utilizar 16 colores (ya sea fijos o intermitentes). En resolución media la pantalla admite cuatro colores, con una visualización de gráficos de 320 × 200 pixels o texto en formato de 40 × 25. La modalidad en alta resolución ofrece una visualización en blanco y negro de 640 × 200 pixels o texto de 80 × 25. En la RAM se pueden almacenar y recuperar al instante siete pantallas completas de 40 × 25, lo que es muy cómodo para menús, páginas de instrucciones, etc. En la modalidad de 80 columnas se pueden almacenar y recuperar cuatro pantallas. A pesar del gran número de colores disponibles, su empleo está sometido a algunas molestas restricciones. En modalidad de textos el fondo se limita a uno de ocho colores, si bien para el primer plano y el margen se puede utilizar la gama completa. En resolución media se pueden visualizar cuatro colores, pero éstos no se pueden escoger de entre la gama completa sino que se debe seleccionar uno de dos grupos (o "paletas").

Las órdenes para gráficos del 86a se limitan a PSET (que establece el color de un pixel individual de la pantalla) y LINE (una rápida orden para trazado de líneas o recuadros). Existen instrucciones más útiles, como CIRCLE, PAINT (para rellenar con color cualquier forma en la pantalla), DRAW (que permite definir y dibujar cualquier forma) y GET y PUT (que hacen posible copiar zonas de la pantalla para gráficos en matrices, que luego se devuelven a la pantalla con tamaños o colores diferentes), pero éstas sólo se proporcionan en el Disk BASIC del 86b. Es una lástima, porque el deseo lógico de un usuario de ordenador personal sería contar con un juego completo de órdenes para gráficos. El juego de caracteres del Advance incluye la gama ASCII normal, junto con símbolos matemáticos y gráficos de bloques que permiten dibujar palos de la baraja, notas musicales, letras griegas, etc., y crear caracteres definidos por el usuario.

El resto del BASIC Advance casi no merece ningún reparo. Aunque carece de algunas de las facilidades "estructuradas" del BASIC BBC, es rápido y muy fácil de usar. Entre las características útiles se incluyen numeración y renumeración automática de líneas, PRINT USING, que permite formatear fácilmente las visualizaciones en pantalla, y la orden SWAP, que permite intercambiar los valores de dos variables. Las facilidades de sonido son buenas, aunque no asombrosas, pero nuevamente se requiere el Disk BASIC para obtener el juego completo de instrucciones. La operación de la cassette es directa: los programas en BASIC y en lenguaje máquina se cargan con una orden LOAD y los programas se ejecutan automáticamente después de cargarlos si se les agrega a la instrucción la letra "R".

Una de las características más impresionantes del Advance es el editor de pantalla. Utilizando la tecla Num Lock para que el teclado numérico actúe como control del cursor, el usuario puede desplazar éste libremente por la pantalla, haciendo correcciones e inserciones en cualquier punto.

En conjunto, el Advance parece cumplir su promesa de proporcionar un ordenador personal que se pueda mejorar para alcanzar un status completo de gestión. Indudablemente, las facilidades que ofrece el Disk BASIC del 86b, más amplio, son considerablemente superiores a las que suministra la máquina más económica, pero el 86a puede, en efecto, resistir la comparación con cualquier micro. Puede que el BASIC no esté a la altura del estándar del BBC, pero su excelente teclado y su inmensa memoria hacen que el Advance sea una opción más atrayente por el mismo precio.

Interface para impresora Centronics

Salida para RGB
También se puede utilizar una pantalla a color de gran calidad

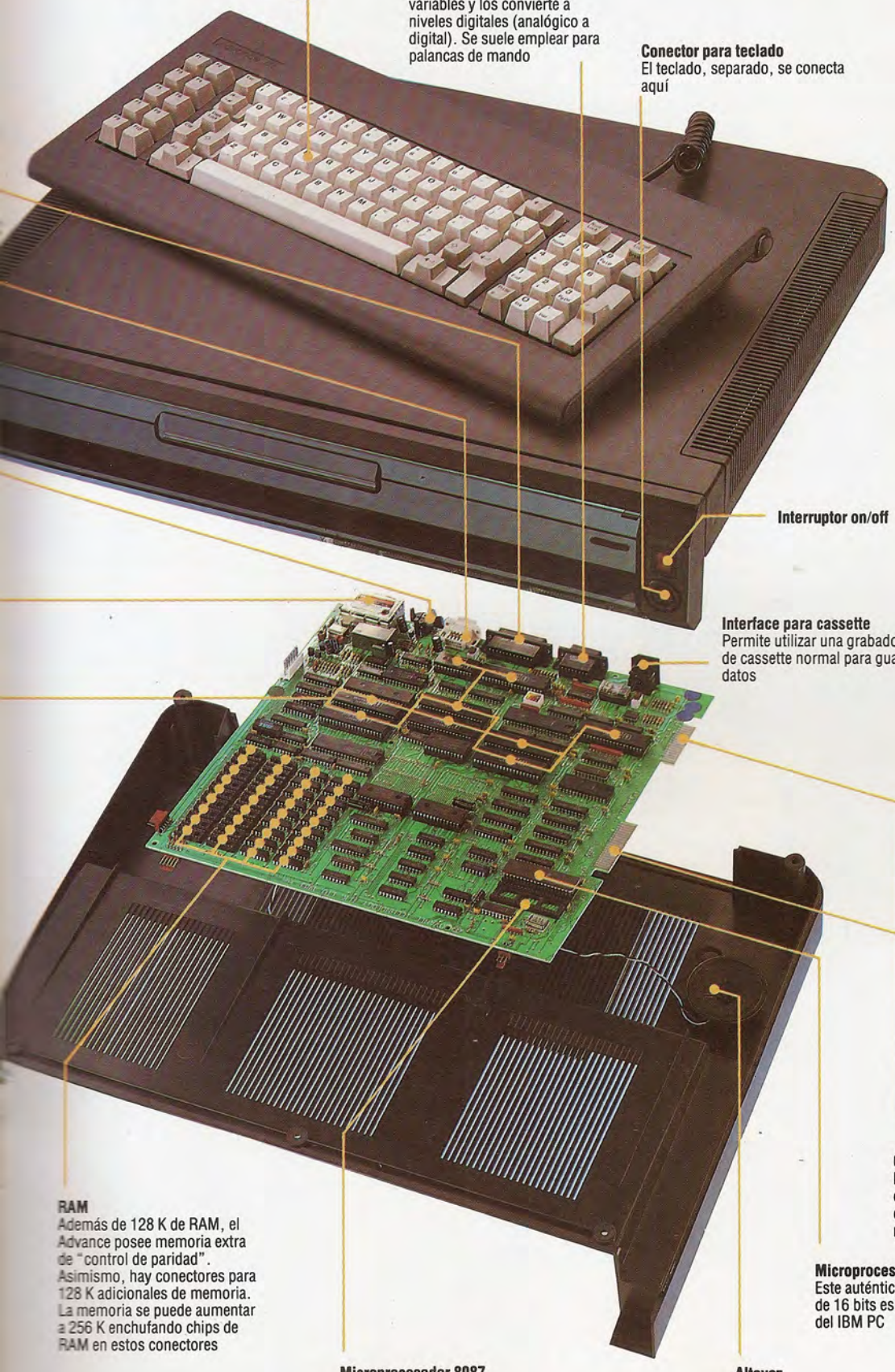
Salida para video compuesto
Permite emplear una pantalla en color o monocromático

Modulador y salida para TV
Para poder utilizar un aparato de televisión normal

Chips ULA Ferranti
El costo del ordenador se mantiene reducido mediante la combinación de muchos circuitos en nueve chips ULA (matriz lógica de uso general) diseñados especialmente

Sistema compatible

El principal atractivo del Advance es que se lo puede adquirir como un ordenador personal y después ampliarlo hasta un nivel similar a las especificaciones de uno de oficina.
El sistema ampliado es casi por completo compatible con el IBM. En la fotografía vemos al Advance ejecutando el Flight Simulator (simulador de vuelo) de Microsoft, que se considera como la prueba definitiva para demostrar la compatibilidad. El kit de ampliación se instala en la parte superior de la unidad principal del 86a. Debe ser instalado por un distribuidor



Teclado tipo IBM

Palanca de mando y convertidor A/D
Este conector lee voltajes variables y los convierte a niveles digitales (analógico a digital). Se suele emplear para palancas de mando

Conector para teclado
El teclado, separado, se conecta aquí

Interruptor on/off

Interface para cassette
Permite utilizar una grabadora de cassette normal para guardar datos

RAM
Además de 128 K de RAM, el Advance posee memoria extra de "control de paridad". Asimismo, hay conectores para 128 K adicionales de memoria. La memoria se puede aumentar a 256 K enchufando chips de RAM en estos conectores

Microprocesador 8087
Aquí se puede agregar el microprocesador de matemáticas de gran velocidad

Altavoz

Conectores para ampliación
El Advance 86a se amplía hasta convertirse en el 86b compatible con IBM acoplando una unidad mediante estos conectores

Microprocesador 8086
Este auténtico microprocesador de 16 bits es más potente que el del IBM PC

ADVANCE 86A

DIMENSIONES
95 x 400 x 520 mm

CPU
Intel 8086, 4,77 MHz

MEMORIA
128 K de RAM (ampliables a 256 K), 64 K de ROM

PANTALLA
25 filas de 40 columnas o 25 filas de 80 columnas de texto, gráficos de 320 x 200 (4 colores) o 640 x 200 (blanco y negro). Dieciséis colores en modalidad de texto

INTERFACES
Monitores de RGB y video compuesto, palancas de mando (2), interface Centronics para impresora en paralelo, puerta para cassette, conector con toma de corriente. Puerta RS232 (sólo en el 86b)

LENGUAJES DISPONIBLES
BASIC en ROM (86a), Disk BASIC (86b)

TECLADO
Tipo máquina de escribir, 84 teclas, incluyendo 10 teclas de función y teclado numérico

DOCUMENTACION
Se suministra con guía para el usuario; también disponible un manual de BASIC. Adecuada pero no abunda en detalles

VENTAJAS
Teclado y editor de pantalla excelentes. El Disk BASIC es muy amplio y dispone de más memoria que ninguna otra máquina personal. La compatibilidad con IBM significa que puede contar con una considerable cantidad de software

DESVENTAJAS
El BASIC en ROM carece de instrucciones para hacer un mejor uso del sonido y los gráficos. Existen limitaciones para los colores en las modalidades para gráficos

Chris Stevens

De la Tierra a la Luna

"Lunar lander" es un juego sutil y refinado que contrasta con los ruidosos y frenéticos juegos actuales

En el juego *Lunar lander* hay que guiar una nave en el aterrizaje sobre la superficie de la Luna (o algún otro planeta). El ordenador de la nave no funciona, de modo que el jugador debe hacer aterrizar la nave con sumo cuidado mediante breves explosiones de su motor cohete. Obviamente, se debe tocar la superficie a una velocidad razonable y el juego implica un delicado equilibrio entre dejarse caer demasiado rápido y gastar la limitada cantidad de combustible evolucionando por la superficie del satélite.

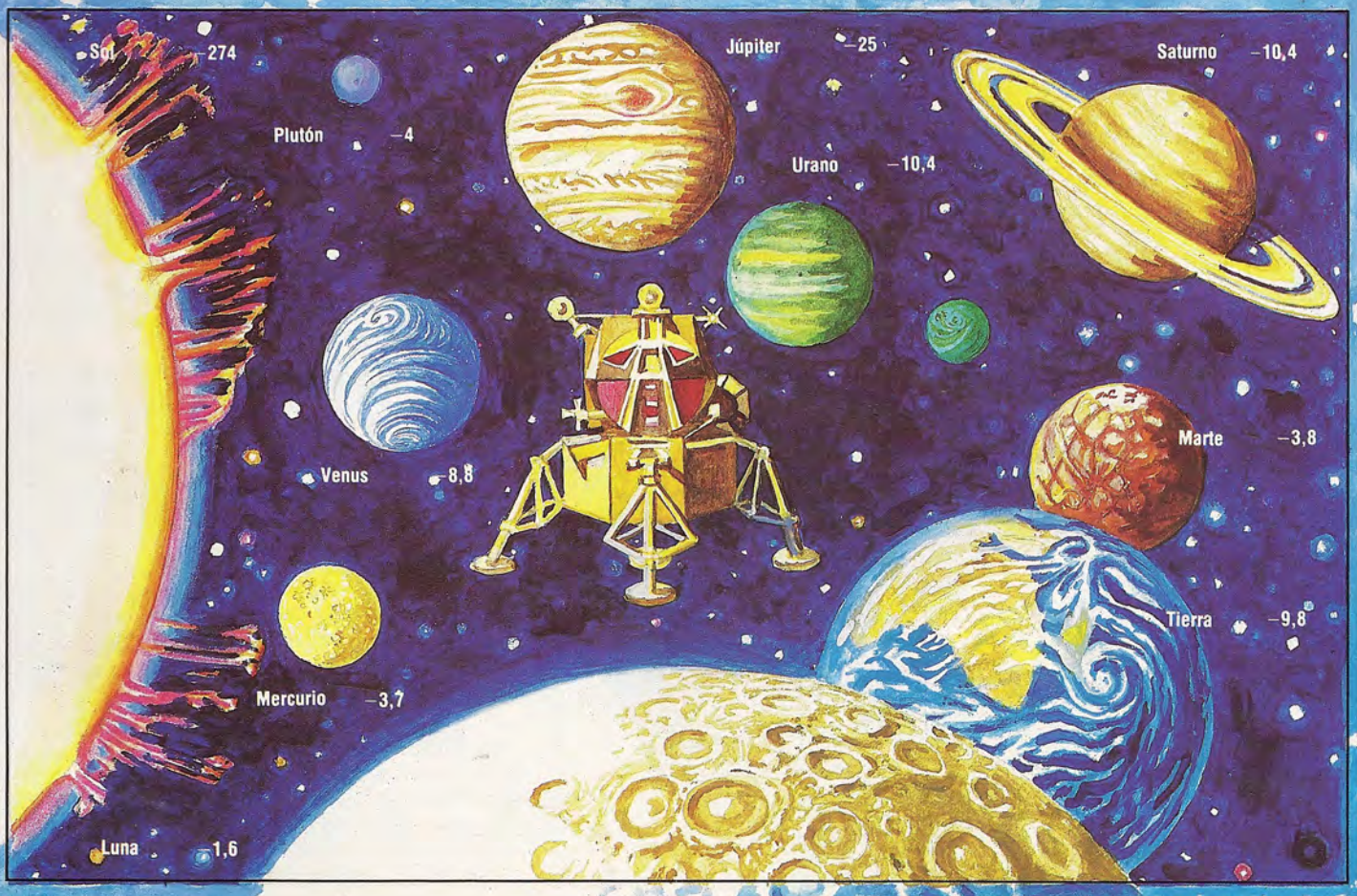
El elemento esencial del juego es que en realidad se trata de una simulación. Si se lo programa de modo que disparar el cohete durante dos segundos siempre reste 10 km/h a su velocidad de descenso, entonces dominar el juego es demasiado sencillo. La idea es que el programa refleje el verdadero comportamiento de una nave espacial en condiciones lo más reales posible. Como resulta evidente, las operaciones matemáticas que hay que efectuar para conseguir esto son muy complejas, de manera que el programa que proporcionamos aquí es una versión simplificada.

Analicemos con mayor profundidad el problema del alunizador:

- El planeta sobre el que uno está descendiendo tiene cierta gravedad. Ésta hará que la nave espacial experimente una aceleración hacia el planeta a medida que vaya descendiendo.
- La nave espacial posee un motor cohete que contrarrestará los efectos de la gravedad empujando la nave hacia arriba.
- La nave espacial posee una masa (o peso). Cuanto mayor sea la masa, menor será el efecto del motor cohete al empujar la nave hacia arriba. La masa de la nave espacial representa su propio peso más el peso del combustible que transporta. A medida que se va consumiendo combustible, la nave espacial se va volviendo más ligera.

Por consiguiente, para reflejar la forma en que se comporta el alunizador necesitamos un conjunto de ecuaciones que implican aceleración, masa, velocidad, etc. Éstas pueden ser muy sencillas o muy

Atracción planetaria
Las cifras reseñadas representan la atracción gravitacional aproximada del Sol, la Luna y los planetas del sistema solar expresados en metros por segundo al cuadrado. Estos valores para la variable *g* del programa deben entrarse en la línea 20. Hay, por supuesto, ciertos cuerpos celestes sobre los que aterrizar sería absurdo o mortal (como el Sol, Júpiter y la Tierra), pero quizá, al efecto del juego, fuera comprensible ignorar estas cuestiones



Adrian Morgan

complicadas, según lo detallistas y exactos que deseemos ser. Hemos procurado mantener estas ecuaciones en un nivel de relativa sencillez.

El principal dato que necesitamos conocer es la altura de la nave espacial. Es evidente que la nave se mueve de manera continua, ya sea cayendo debido a la gravedad o acelerando fuera del planeta debido a un uso excesivo del motor cohete. Para poder calcular dónde se halla la nave en un momento dado, se divide el "tiempo" en una serie de pasos o períodos.

En cada período se puede calcular hasta dónde se ha desplazado la nave, cuál es el cambio en cuanto a velocidad y masa, etc. Estos períodos pueden ser de la duración que se desee: cuanto más cortos sean, más exacta será la simulación. Una vez introducida la idea de los períodos, escribir las ecuaciones es fácil.

La velocidad se mide en base a unidades por hora. En dos horas, un coche que viaja a 10 km/h recorrerá 20 kilómetros. En tres horas, recorrerá 30, y así sucesivamente. Esto nos da la fórmula:

Distancia = Tiempo × Velocidad

De modo que en cada período podemos calcular lo que se ha desplazado el alunizador, hacia arriba o hacia abajo, multiplicando su velocidad por la duración del período (que definimos como unidad). Podemos entonces ajustar la velocidad acelerando la nave en función del empuje gravitacional del planeta y desacelerándola en función del empuje de los motores cohete.

La aceleración debida a la gravedad siempre es constante (la variable g del programa) y dependerá de a qué planeta se esté uno acercando. La ilustración refleja los valores para los planetas de nuestro sistema solar, pero se puede experimentar con otros valores o hacer que el programa genere g al azar para dotar al juego de mayor dificultad.

Simular el motor cohete es algo más complicado. En esta versión, el jugador puede quemar entre una y nueve unidades de combustible en un período dado, y el programa calcula la aceleración resultante, teniendo en cuenta la masa de la nave. La fórmula exacta depende de la potencia de los motores cohete y del tipo de combustible utilizado. En este programa las cifras se han elegido para que ganar el juego sea más laborioso; trate de alterarlas para ver cómo ello incide en el juego.

Un requisito que se le puede agregar al juego es que deba jugarse en *tiempo real*. Ésta es una frase de la que se abusa mucho y, en la actualidad, significa que el programa (juego, simulación, etc.) se ejecuta de forma continua, sin detenerse nunca para esperar la entrada de datos u órdenes. Con frecuencia esto no es más que la diferencia entre emplear una orden INPUT para obtener información y utilizar INKEY$ o GET.

Obviamente, *Lunar lander* es mejor, como juego, si el programa no produce ninguna interrupción para calcular cuánto combustible quemar. Si así lo hiciera, el jugador tendría tiempo para considerar la situación, efectuar algunos cálculos, etc. En la vida real, todo sería cuestión de un pensamiento más rápido.

En nuestro programa del alunizador, tenemos un bucle que se ejecuta una vez para cada intervalo de tiempo del programa. Ajustando el período de modo que sea realmente el tiempo que se tarda en ejecutar el bucle, la simulación opera en tiempo real. En una simulación hacer aterrizar la nave debería tomar tanto tiempo como el que sería necesario en la vida real. Aunque esto es de desear en una simulación, puede ser bastante difícil de conseguir en un juego como éste. La simulación suele ocupar demasiado tiempo para que resulte interesante como juego.

Son muchas las mejoras que se pueden introducir al programa de aterrizaje básico. La más obvia es agregar una visualización gráfica del descenso. En este sentido, las ideas van desde un sencillo dial redondo para el altímetro hasta una vista lateral de una pequeña nave, o incluso una panorámica a escala, hacia abajo, del punto de aterrizaje. También se podría tratar de añadir un movimiento lateral de modo que la nave no sólo tuviera que bajar hacia el punto de aterrizaje sino también situarse sobre él.

En el espacio, la nave normalmente no girará hacia los lados porque no hay nada que la empuje hacia otra dirección que no sea descendente. Pero si se está aterrizando en un planeta con atmósfera, se podría añadir el problema de un viento de superficie, por ejemplo. Algunas refinadas versiones del programa incluyen varias zonas de aterrizaje, localizadas en el fondo de túneles y cráteres, de modo que el descenso exige numerosísimas maniobras.

El *Lunar lander* podría parecer algo muy anticuado comparado con los rápidos y frenéticos juegos recreativos actuales. Pero programarlo y jugar con él es, para muchas personas, el primer encuentro con una simulación por ordenador y con todo el complicado campo de hacer que los programas reflejen situaciones del mundo real. Programar un aterrizaje lunar puede ser el primer paso hacia una nueva gama de proyectos de programación.

Alunizaje

```
10 REM Juego de aterrizaje lunar
20 LET g = −1,6
30 LET t = 1
40 LET f = 1000
50 LET v = 0
60 LET h = 2000
70 LET m = 2000 + f
80 LET g = g*t
100 REM Actualizar pantalla
110 PRINT AT 0,0
120 PRINT"          Aterrizaje lunar"
130 PRINT: PRINT "Altura.....";INT h;"  "
140 PRINT: PRINT "Velocidad....";INT v;"  "
150 PRINT: PRINT "Combustible..";f;"  "
160 PRINT
165 IF h < 0 THEN GO TO 400
170 IF f < = 0 THEN LET f = 0: PRINT "***SIN COMBUSTIBLE
    ": GO TO 190
180 PRINT "Pulsar encendido cohete 0−9"
190 LET b = 0: IF f > 0 THEN LET a$ = INKEY$: IF a$ < >"" THEN
    LET b = VAL a$
200 IF b > f THEN LET b = 0
210 LET h = h + v*t
220 LET v = v + g
230 LET v = v + (b*3000)/m
240 LET f = f−b: LET m = m−b
250 FOR i = 1 TO 50: NEXT i
300 GO TO 110
400 REM Sobre la superficie del planeta
410 IF v > −10 THEN PRINT "***Aterrizaje seguro... Bien hecho": GO
    TO 500
420 IF v > −20 THEN PRINT "***CATACRAC!... Ha destruido la nave
    pero la tripulación ha sobrevivido!": GO TO 500
430 PRINT "***BUUUM!... Nave destruida... No hay supervivientes"
440 PRINT: PRINT "Acaba de crear un nuevo cráter de";INT (−v*2.1);
    "km de ancho"
500 PRINT: PRINT "Vuelve a jugar (S/N)?";
510 LET a$ = INKEY$: IF a$ = "" THEN GO TO 510
520 IF a$ = "s" OR a$ = "S" THEN RUN
530 IF a$ <> "n" AND a$ <> "N" THEN GO TO 510
540 CLS: STOP
```

Complementos al BASIC

Éste es un listado para el Spectrum: en otras máquinas no es necesario utilizar la palabra LET. En el BBC Micro, reemplazar la línea 110 por:

```
110 PRINT TAB(0,0)
```

Reemplazar INKEY$, en las líneas 190 y 510, por INKEY$(0). Reemplazar VAL a$, en la línea 190, por VAL(a$). Reemplazar INT h e INT v, en las líneas 130 y 140, por INT(h) e INT(v).

En el Commodore 64 y en el Vic-20, reemplazar la línea 110 por

```
110 PRINT CHR$(19)
```

Reemplazar LET a$ = INKEY$, en las líneas 190 y 510, por GET a$. Reemplazar VAL a$, de la línea 190, por VAL(a$). Reemplazar INT h e INT v de las líneas 130 y 140 por INT(h) e INT(v).

En el Oric Atmos, reemplazar la línea 110 por:

```
110 PRINT @0,0
```

Reemplazar INKEY$, en la línea 190 y 510, por KEY$. Reemplazar VAL a$, de la línea 190, por VAL(a$). Reemplazar INT h e INT v, de las líneas 130 y 140, por INT(h) e INT(v)

Asesor de estilo

Un programa bien documentado es capaz de indicar lo que está haciendo y de qué manera lo está realizando

Consideremos la primera versión de nuestro programa (listado 1). Es, a todas luces, un gran misterio: es difícil adivinar lo que hace. Aparte de decir que "se entran (input) dos números, son multiplicados por otros dos números, se suman entre sí los dos resultados y se imprime la respuesta", hay muy pocos indicios de la tarea exacta que realiza el código. Ahora observemos la segunda versión del programa (listado 2). Se revela el misterio. Pero no se ha agregado ningún comentario, no hay títulos del programa ni se han insertado líneas de REM ni se ha producido documentación externa.

Vale la pena analizar de forma detallada las diferencias entre estas dos versiones. En primer lugar, los números del primer listado, carentes de todo significado, se han reemplazado por nombres (UNANY y UMES). Los números cuyos valores no cambian en el curso de la ejecución del programa se denominan *constantes*. Algunos lenguajes, como el PASCAL, poseen una notación especial para las constantes (en el listado 2 las dos constantes se definen aparte de las variables), mientras que otros lenguajes, como el BASIC, no. (Las líneas 10 y 20 del programa en BASIC emplean variables para definir las constantes.) Darles nombres a las constantes sólo vale la pena si se han de utilizar con frecuencia, de lo contrario los comentarios incluidos en el programa servirán igualmente para ese cometido.

La segunda diferencia crucial es que a todos los confusos nombres de variables se les han dado nombres más largos y significativos. Los que hemos incluido aquí (NANYS en vez de A, segsedad en lugar de e, etc.) los escogimos porque cada uno de ellos posee menos de 10 caracteres de largo, y los dos primeros caracteres son suficientes para distinguirlos entre sí. La razón para este último requisito la explicaremos enseguida.

En general, es una buena costumbre dar a las variables nombres que guarden alguna relación con el papel que desempeñan en el programa. Por ejemplo, al contador de un bucle se lo podría llamar BUCLE (en vez de los nombres habituales, J o I), y los primeros y los últimos valores del contador se pondrían en constantes o variables con nombres apropiados. Por consiguiente, un bucle como éste:

```
FOR J = 1 TO 10...NEXT J
```

podría escribirse así:

```
FOR BUCLE = PRIMERO A DECIMO...NEXT BUCLE
```

Los nombres de variables largos, por supuesto, llevan más tiempo en digitarlos y ocupan más memoria, pero poseen la ventaja de hacer que los programas resulten más fáciles de comprender y aceleran el proceso de depuración (*debugging*). Si su lenguaje utiliza sólo los dos primeros caracteres de los nombres para distinguirlos entre sí, asegúrese de que los nombres que elija difieran en sus dos primeros caracteres. De no ser así, dos nombres de variables largos (p. ej., CODIGO y COMP) podrían parecerle bien distintos al programador, pero para el ordenador serían indistinguibles.

Otra importante diferencia entre los listados es que el segundo utiliza indicaciones extensas y plenas de significado para sus input y les agrega a sus salidas una explicación razonable (las líneas PRINT en BASIC, las líneas write en PASCAL). Con ello se consiguen dos cosas importantes. La primera es que el programa sea más legible. Aun cuando las variables constaran de una sola letra, el programa seguiría teniendo mucho más sentido que anteriormente. La segunda ventaja, aún más importante, es que hace que el programa sea asequible incluso para quien nunca antes lo hubiera visto.

Trazado de programa

Los usuarios del PASCAL ya serán conscientes de las ventajas de trazar un programa adecuadamente en la pantalla. Cosas muy sencillas (como indentar las líneas, dejar líneas en blanco y combinar mayúsculas y minúsculas) pueden convertir una masa impenetrable de símbolos en un trozo de lógica sensato y legible. Formatear un programa para la pantalla o la impresora realmente adquiere todo su valor cuando sus programas utilizan construcciones de bucle (FOR...NEXT, WHILE...WEND, REPEAT...UNTIL) y, en especial, cuando hay bucles anidados dentro de otros.

Habiendo dicho esto, es lamentable que la mayoría de los BASIC den muy pocas opciones sobre la forma en que uno puede trazar el programa. En este sentido, los lenguajes compilados, como el PASCAL, son mucho más flexibles, puesto que suelen escribirse con un editor de textos (o procesador de textos). Por el contrario, editar un programa en BASIC generalmente es un asunto bastante imperfecto (a menos que, como en el MBASIC de Microsoft, su intérprete tome una versión ASCII del programa y la convierta en un programa ejecutable). Lo que es aún peor, muchos BASIC toman los programas que usted escribe ¡y los vuelven a reformatear para eliminar la indentación! Hay otros que, por el contrario, agregan una indentación que uno no había incluido. El BBC Micro es bastante eficiente en este sentido, pero hay que recordar darle la orden LISTO. La mayoría de los sistemas PASCAL incluyen un formateador y por lo general son muy útiles. No obstante, en aras de su propia claridad de pensamiento, es una buena idea concebir algunas convenciones de formateado, siempre dentro de los límites de su lenguaje.

Los comentarios, por supuesto, son la forma principal de documentar los programas dentro de los propios programas. Nuevamente, las convenciones varían de un lenguaje a otro. El BASIC utiliza

la sentencia REM. La palabra REM debe aparecer al principio de cada comentario y posteriormente el intérprete ignorará todo lo que encuentre hasta el siguiente indicador de final de sentencia (: o (cr)). En otros lenguajes (PASCAL, PL/1, PROLOG, etc.) los comentarios se encierran entre /* y */ (algunas veces { y }), y el compilador hace caso omiso de todo cuanto haya entre estos signos. Una ventaja que ofrece este sistema es que los comentarios pueden ocupar más de una línea. La desventaja es que si uno se olvida del segundo */, ¡el resto de su programa se toma como un comentario y se ignora!

Utilice un comentario cada vez que piense que puede ser necesaria alguna explicación: cuando está definiendo constantes, inicializando variables, empezando un programa o un nuevo procedimiento (subrutina), definiendo una función o escribiendo algún código que, debido a su complejidad, no se entienda a simple vista. Los comentarios no han de ser extensos ni redundantes, y a menudo sólo se requiere un recordatorio. Cuando usted está intentando comprender la lógica de un programa de aventuras del año anterior, los largos bloques de comentarios generales que interrumpen el código y no proporcionan detalles suficientes pueden ser más un obstáculo que una ayuda, de modo que trate que sus comentarios sean breves y concisos. Colóquelos antes de secciones tramposas del código, e inclúyalos en éste sólo cuando no existan posibilidades de que interfieran la lectura de la estructura lógica del programa. Nuestro programa final (listado 3) ofrece algunos ejemplos.

La documentación externa, en forma de guías y especificaciones escritas, es la más dura y tediosa de realizar. Para los programadores, los estudios han demostrado que la documentación escrita se suele consultar sólo como último recurso. Sin embargo, cuando se la utiliza puede significar un considerable ahorro de esfuerzo. Si su programa no es demasiado extenso y está bien documentado internamente, es poco probable que alguna vez se encuentre con la necesidad de una documentación externa del *programa*. La documentación para el *usuario* es otra cuestión y la analizaremos en un capítulo posterior. No obstante, suele ser útil tener a mano alguna documentación escrita cuando se trata de revisar un programa antiguo o de depurar uno nuevo. Una de las formas en que los lenguajes de la llamada "quinta generación" intentan aumentar la productividad del programador es mediante la generación automática de documentación. Esto se conseguirá utilizando información desde la fase de diseño del desarrollo de un programa. No es sorprendente que una de las mejores formas de documentar sus propios programas sea aplicando esta misma técnica.

Vaya formando un archivo para sus programas a medida que los escriba. Coloque en el mismo todas las notas que tome mientras va diseñando el programa, incluyendo borradores de algoritmos y diagramas de flujo. Y, lo que es más importante, conserve la versión final del diagrama que ha utilizado para escribir la versión final. Si posee una impresora, conserve un listado del programa acabado. Observe que en nuestra versión completa del programa, el primer comentario incluye el nombre de éste y una fecha. Cada vez que modifique un programa, cambie también su fecha; así sabrá que se trata de la última versión.

Correctamente documentado

Listado 1

BASIC

(a)
```
10 INPUT A,B
20 C = A*31536000
30 D = B*2592000
40 E = C + D
50 PRINT E
```

PASCAL

(b)
```
program abcde (input,output);
var a,b,c,d,e:integer;
begin
read(a,b);
c: = a*31536000;
d: = b*2592000;
e: = c + d;
writeln(e);
end.
```

Listado 2

BASIC

(a)
```
10 UNANY = 31536000
20 UMES = 2592000
30 PRINT"Entre su edad (en formato AA,MM)";
40 INPUT NANYS,NMESES
50 ASEGS = NANYS*UNANY
60 MSEGS = NMESES*UMES
70 SEGSEDAD = ASEGS + MSEGS
80 PRINT"Su edad en segundos es (aproximadamente)":SEGSEDAD
```

PASCAL

(b)
```
program edadensegs (input,output);
  const
    unany = 31536000;
    umes = 2592000;
  var
    nanys,nmeses,asegs,msegs,segsedad:integer;
  begin
    write('Entre su edad (en formato AA,MM)');
    read(nanys,nmeses);
    asegs: = nanys*unany;
    msegs: = nmeses*umes;
    segsedad: = asegs + msegs;
    writeln('Su edad en segundos es (aproximadamente)',segsedad);
    end.
```

Listado 3

BASIC

(a)
```
10 REM "EDADENSEGUNDOS" Junio 1984
20 REM Entra edad en años y meses (AA,MM) y
30 REM utiliza una conversion aproximada (mes = 30 dias)
40 REM para dar la edad en segundos.
50 REM
60 UNANY = 31536000:REM      segundos en 365 dias
70 UMES = 2592000:REM        segundos en 30 dias
80 PRINT"Entre su edad (en formato AA,MM)";
90 INPUT NANYS,NMESES
100 REM la edad en segundos es (edad en años*segundos en el año) mas (meses desde ultimo
    cumpleaños*segundos en el mes)
110 ASEGS = NANYS*UNANY
120 MSEGS = NMESES*UMES
130 SEGSEDAD = ASEGS + MSEGS
140 PRINT"Su edad en segundos es (aproximadamente)":SEGSEDAD
```

PASCAL

(b)
```
program edadensegundos (input,output);
/* Junio 1984
   lee edad en años y meses (AA,MM) y usa una conversion aproximada (mes = 30 dias) para dar la
   edad en segundos.                                                                          */
const
  unany = 31536000;      /* segundos en 365 dias */
  umes = 2592000;        /* segundos en 30 dias */
var
  nanys,nmeses,asegs,msegs,segsedad:integer;
begin
  write('Entre su edad (en formato AA,MM)');
  read(nanys,nmeses);
  /*la edad en segundos es (edad en años*segundos en el año)
    más (meses desde el ultimo cumpleaños*segundos en el mes)*/
  asegs: = nanys*unany;
  msegs: = nmeses*umes;
  segsedad: = asegs + msegs;
  writeln('Su edad en segundos es (aproximadamente)', segsedad;
end.
```

Guerra en familia

"Apocalypse" (Apocalipsis), el juego de la devastación nuclear, es un paquete tradicional del tipo "Monopoly", que exige aptitudes tácticas y reacciones rápidas por parte del usuario

La mayoría de los juegos por ordenador son diversiones solitarias, antisociales. Los anuncios publicitarios muestran familias enteras reunidas alrededor del ordenador personal, pero los juegos por lo general están diseñados para un único jugador.

Sin embargo, *Apocalypse* recupera la tradicional rivalidad y la intriga propias del *Monopoly* y otros juegos de mesa de este tipo y, por tanto, es especialmente apropiado para reuniones de familia. De hecho, la versión para el BBC permite la participación de hasta 15 jugadores. Se trata de un auténtico juego para varias personas, ya que no se limita simplemente a que éstas se vayan turnando para participar en un juego pensado para una sola persona.

Originalmente *Apocalypse* era un juego de tablero, basado en otro denominado *Diplomacy*, que ahora ha sido ampliado para sacar partido de las capacidades para gráficos y proceso de números del microordenador. El juego principal comprende cuatro "escenarios de guerra" (Europa, el Caribe, Gran Bretaña y Londres) y se pueden adquirir kits de ampliación para incluir acciones en Sudáfrica, Levante, el Ártico, Estados Unidos, el Sudeste asiático, la campaña del Pacífico durante la segunda guerra mundial, el espacio exterior y campañas históricas basadas en la caída del Imperio romano y las batallas napoleónicas. Estas opciones están disponibles en tres cintas que se combinan con el juego principal.

La pantalla está dividida en una serie de cuadrados, visualizados como una matriz de 40 × 20 (BBC) o 20 × 20 (Spectrum). Se utilizan cuadrados azules para representar el mar y mediante otros colores se describen distintas características geográficas: zonas rurales o urbanas, ciudades, montañas o desiertos. También hay símbolos para representar las fuerzas oponentes. Estos gráficos dependen de las posibilidades de la máquina. Los gráficos del Spectrum, más simples, ofrecen la ventaja de la claridad: los cuadrados se rellenan o se dejan vacíos, por lo cual es fácil detectar el territorio que está ocupado por una fuerza atacante. La visualización del BBC, si bien permite apreciar más detalles gráficos, aparece como desordenada y los símbolos de los jugadores tienden a perderse en el fondo.

Los jugadores se turnan para desplegar sus fuerzas de la forma apropiada al escenario elegido. Las fuerzas del ejército y de la marina se desplazan por la zona de juego y lanzan o repelen ataques, mientras el ordenador actúa como árbitro. Los movimientos se realizan seleccionando opciones de diversos menús y, en este sentido, la visualización se podría mejorar para hacer que las cosas resultaran más sencillas. La opción *nuke* (ataque nuclear) se puede seleccionar en el juego principal, con consecuencias previsiblemente devastadoras, pero esta opción sólo se puede utilizar en su correcto contexto histórico: el programa no permitirá que usted lance un ataque nuclear preventivo contra legiones romanas enemigas, por ejemplo.

Éste es un juego largo pero, si se encuentran jugadores bien dispuestos, es muy interesante. Si es o no bueno para jugarlo con un microordenador, ya es otro tema. Tal vez a muchas personas el juego sobre tablero les resulte igualmente atractivo, pero por ordenador es posible que llegue a apasionar.

Apocalypse: Para el BBC Micro (2-15 jugadores) y el ZX Spectrum (2-4 jugadores)
Editado por: Red Shift, 12 Manor Road, London N16
Autores: H. Watson (BBC); R. Tyler (Spectrum)
Diseño original del juego: Mike Hayes
Palancas de mando: No se necesitan
Formato: Cassette

Zonas de guerra

Apocalypse es un juego para toda la familia porque pueden participar hasta 15 personas simultáneamente (en la versión para el BBC), o hasta cuatro, en la que vemos aquí, para el Spectrum. Siempre son necesarios al menos dos jugadores. Éstos se ven envueltos en una confrontación armada que a la larga puede desembocar en un conflicto nuclear. Afortunadamente, la visualización se limita a mapas como éstos y no ofrece imagen alguna de la devastación

Liz Heaney

Paradero conocido

Estudiamos una rutina en lenguaje máquina, casi imprescindible para los creadores de programas de juegos

Un juego recreativo de calidad necesita estar escrito, por lo menos parcialmente, en lenguaje máquina. Esto constituye un auténtico desafío para el principiante; por este motivo hemos decidido presentar una rutina para sprites concebida especialmente para el Spectrum.

El BASIC del Spectrum es muy limitado, y eso se nota especialmente a la hora de dar movimiento a los gráficos. Los juegos animados requieren el uso de sprites de varias formas y tamaños que han de desplazarse suavemente por la pantalla en todas las direcciones.

No es fácil escribir rutinas de gráficos en assembly, pero el programa que aquí presentamos probablemente sugerirá al lector un buen puñado de ideas con las que podrá afrontar la tarea. Este programa imprime un fondo de asteriscos colocados al azar y permite además dar movimiento a cualquier figura escogida por el usuario (nosotros elegimos una cruz) que recorrerá la pantalla con sólo tocar las teclas del cursor. La cruz se mueve paso a paso, un pixel cada vez, arriba, abajo, a derecha e izquierda, sin modificar el fondo. La rutina que mueve el sprite es fácilmente incorporable a cualquiera de sus programas en BASIC.

Para introducirla en un Spectrum, hay que entrar primero el programa en BASIC. Sólo entonces se puede entrar el código máquina ya sea por medio del programa de carga en BASIC, ya sea empleando un ensamblador adecuado. Es posible conservar en cinta de cassette ambos programas, el de BASIC y el de lenguaje máquina, con las líneas 9000 y 9010 del programa BASIC.

Comencemos examinando el fragmento en BASIC, con el fin de comprender plenamente el funcionamiento de todo el programa. La subrutina de la línea 1000 recoge (READ) la forma del sprite definida mediante las sentencias DATA y la coloca (POKE) en la memoria RAM donde le sea posible utilizarla al código máquina. El fondo se imprime con las líneas 90 hasta la 110, y la posición inicial de la cruz se establece en la 120. Con PRINT AT 10,16 el intérprete de BASIC calcula la dirección de la pantalla correspondiente a estas coordenadas de carácter, almacenando dicha dirección en la variable de sistema DFCC (direcciones 23684 y 23685), donde también es accesible por parte del código máquina. La línea 130 llama a la sección de inicialización del programa en código máquina. El bucle de las líneas 140 a 180 espera la pulsación de cualquier tecla, coloca (POKE) el valor numérico de la tecla pulsada en una posición de la memoria que pueda ser leída por el código máquina y posteriormente hace que este código máquina mueva el sprite un pixel en la dirección que determina la misma tecla.

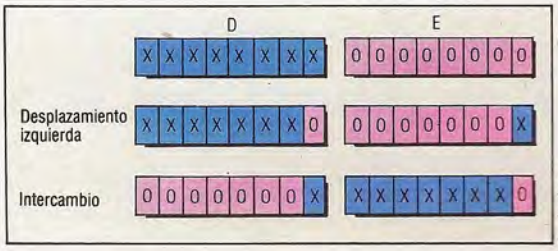
El programa en assembly comienza con la definición de las posiciones de memoria empleadas. Así, KEY (*tecla*, en inglés) indica la posición donde se almacenó el valor de la tecla. SPRPOS significa la dirección de la memoria que contiene la *posición* del *sprite* en la pantalla. SPRTAB es la tabla en la que el programa almacena la definición del sprite y el contenido de las posiciones de la pantalla sobre las que se superpone el sprite. Éste se mueve a cualquier punto de la pantalla; no se limita, pues, a saltar cuadrados de carácter enteros. Y sucede así porque los ocho bits de cada fila del sprite pueden dividirse en dos bytes de memoria de pantalla y la tabla emplea dos bytes para almacenar esos ocho bits, partidos igual que en la pantalla. Usamos BITPOS para almacenar el número de bits que, desde el inicio del byte, se han ido desplazando los datos del sprite.

El trozo del programa que sirve para inicializar lee de DFCC la dirección inicial de la pantalla, salta a la sección etiquetada con SAVSCR (recuerde que en inglés pantalla se dice *screen*), donde guarda la dirección de la pantalla en SPRPOS y carga el valor 1 en el registro D para llamar finalmente la subrutina UNDER. Si D está en 1, la rutina UNDER copia el contenido del área de la pantalla en la cual aparecerá el sprite a fin de restaurarlo, una vez desplazado el sprite. La otra subrutina que llama ahora el programa es la PRSPRT que visualizará (PRint) el sprite en la pantalla.

Liz Heaney

El sprite en movimiento
El programa de demostración en BASIC desplaza el sprite (especificado como una cruz en las sentencias DATA) a través de la pantalla con un fondo de estrellas. Sólo hay que apretar las teclas del cursor

Del movimiento del sprite se encarga la sección de programa que empieza en MOVSPR. Lo primero



Kevin Jones

Desplazamiento e intercambio
El desplazamiento de los bytes del sprite (que representamos con las X), dentro del registro DE, queriendo obtener un efecto de movimiento a la izquierda o a la derecha, quizá cause que algún bit caiga por un extremo. En ese caso D y E se intercambian, volviendo a reunir los bits del sprite

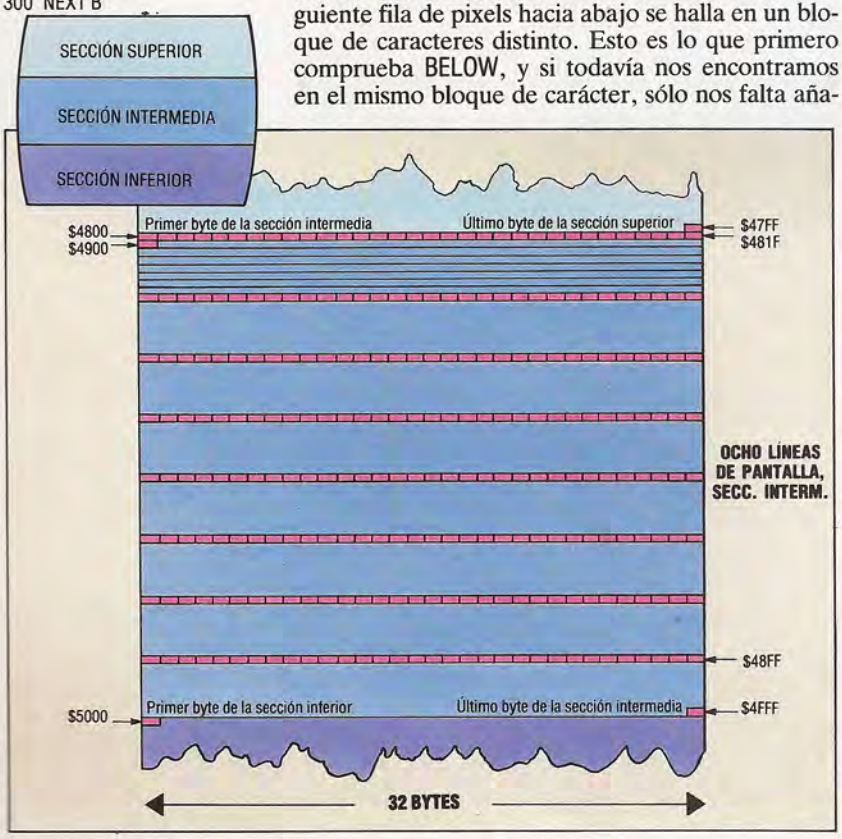
Direccionamiento de pantalla
A efectos de representación en memoria, la pantalla de 24 líneas del Spectrum está dividida en tres secciones de ocho líneas cada una; en el dibujo inferior se muestra el detalle del direccionamiento de la sección central. Cada línea de la pantalla está dividida en ocho líneas de alta resolución de 32 bytes, dentro de los cuales cada bit representa un punto o pixel. Los bytes son numerados consecutivamente a lo largo de la fila de alta resolución; al terminar una de ellas se empieza con la misma fila de alta resolución de la línea inmediatamente inferior: por tanto, los bytes de las ocho filas superiores de las ocho líneas de cada una de las tres secciones de la pantalla tienen direcciones consecutivas. Al terminar las ocho primeras filas, se sigue numerando por el primer byte de la segunda fila de la primera línea; todos los bytes de las ocho segundas filas de alta resolución se numeran correlativamente a partir de ese byte, y así sucesivamente. La dirección del primer byte de la fila superior de la primera línea de una sección se obtiene sumando una unidad a la dirección del último byte de la octava fila de la octava línea de la sección anterior. El siguiente programa coloca un valor $FF en todos los bytes de la pantalla:

```
50 LET INIPANT = 16384
60 LET FINPANT = 22527
100 FOR B = INIPANT TO
    FINPANT
200 POKE B,255
300 NEXT B
```

que hace es poner 0 en el registro D y llamar la subrutina UNDER. Si D está en 0, UNDER devuelve a la pantalla el contenido del fondo que salvó previamente, al tiempo que borra el sprite de la pantalla. Entonces es cuando el programa toma la posición del sprite y el valor de la tecla, lo comprueba y llama a la rutina que preparará el movimiento en la dirección adecuada, saltando finalmente a SAVSCR que, como antes, nos guardará el contenido del fondo de la pantalla y visualizará el sprite.

Dos son las rutinas que se encargan del movimiento vertical del sprite, ABOVE (*arriba*, en inglés) y BELOW (*abajo*). Para entenderlas, es inevitable examinar la extraña manera como el Spectrum relaciona direcciones de memoria con posiciones de pantalla. El capítulo 24 del Manual del Spectrum la explica. Mirando las direcciones en hexadecimal, se observa que cada byte de los ocho que componen un carácter (y son 256 caracteres los que integran una sección de pantalla) tiene una dirección, compuesta a su vez de dos bytes: el byte inferior o byte *lo* coincide con el número del carácter dentro del bloque, mientras que el byte superior o byte *hi* se incrementa en una unidad cuando nos desplazamos una línea de pixels hacia abajo en la pantalla. Ésta es la razón por la que las ocho filas de pixels de un carácter poseen direcciones que van del $4000 al $47FF para el tercio superior de la pantalla, del $4800 al $4FFF para el tercio central, y del $5000 al $57FF para el inferior. (Recuerde que $ significa número en hexadecimal, aunque algunos ensambladores emplean el símbolo #).

La subrutina BELOW espera una dirección de pantalla en el registro doble HL, calcula la dirección del byte inmediatamente inferior a esta posición de la pantalla y deja el nuevo valor en HL. Observadas en binario, si las direcciones de la pantalla tienen los tres bits inferiores de H con el valor 111, la siguiente fila de pixels hacia abajo se halla en un bloque de caracteres distinto. Esto es lo que primero comprueba BELOW, y si todavía nos encontramos en el mismo bloque de carácter, sólo nos falta añadir 1 a H. Si nos hallamos en un bloque de carácter distinto, añadiremos $20 a L (puesto que hay 32 caracteres en una línea). Si el nuevo valor de L está entre 0 y $1F (tres bits altos a 0), esto significa que estamos en un bloque de pantalla distinto. En cuanto al valor de HL, éste corresponde a la dirección de pantalla en curso.



Kevin Jones

Si estamos en el mismo bloque de pantalla, tenemos además que restar 7 a H. Todo esto lo entenderá mejor si examina el código máquina y observa lo que ocurre con las direcciones mostradas en la tabla.

Por lo demás, ABOVE funciona como BELOW, sólo que su misión es calcular la dirección del pixel que está encima de la posición de la pantalla.

Las subrutinas LMOVE y RMOVE desplazan los bits del pixel como un todo a la izquierda y a la derecha. Como son también semejantes, veamos tan sólo cómo funciona LMOVE. El apuntador de la posición de los bits pasa al acumulador, siendo aquél un número de un solo byte entre el 0 y el 7 que numera a cada bit dentro del byte. Para efectuar el movimiento se resta 1 al valor actual del acumulador y el resultado también será un número entre el 0 y el 7 (a menos que el valor original del apuntador fuera 0, en cuyo caso obtendríamos el resultado de 255). La instrucción AND 7 se encarga de que el valor del acumulador quede siempre entre 0 y 7. Seguidamente encontramos un bucle para las ocho filas de pixels de un sprite. La tabla va proporcionando dos bytes cada vez, correspondientes a cada fila, que se cargan en el registro doble DE, para realizar después una rotación de 16 bits a la izquierda en el mismo DE. Si ningún bit queda fuera del extremo superior de D para colocarse en el inferior de E, entonces pondremos de nuevo en la tabla todos los bits del sprite ya desplazados y pasaremos a la siguiente fila de pixels. Pero si se trasladó un bit del sprite desde el extremo superior de D al inferior de E debemos intercambiar D y E antes de almacenarlos de nuevo en la tabla. La rutina, por último, restará 1 a HL para que el sprite aparezca en pantalla una posición a la izquierda.

La última subrutina es PRSPRT, que tiene como misión la de visualizar el sprite sobre la pantalla. La forman dos bucles anidados, uno para las ocho filas de pixels del sprite controlado por el registro C, y otro, controlado por el registro B, para los dos bytes en que se reparte la fila de pixels. El núcleo de la rutina está en su sección central, encargada de almacenar los bits del sprite en pantalla sin afectar los otros bits en pantalla previos que el sprite debe dejar intactos. La dirección de la pantalla se coloca en el registro doble HL y la dirección de la tabla del sprite en el registro IX. Nuestra PRSPRT toma un byte de la trama de pixels del sprite y le aplica la operación OR con el que ya está en pantalla, de modo que nos resulten los puntos apagados del sprite superpuestos al contenido previo de la pantalla.

Este programa no es de uso universal. Por ejemplo, el tamaño máximo de un sprite está limitado a 8 × 8 pixels, además sólo admite un sprite y éste no traslada consigo sus propios colores. Pero una vez entendido, de seguro que usted mismo sabrá añadir al programa algunos detalles. Con o sin modificaciones, se trata de un muy útil aditamento para muchos programas en BASIC y en lenguaje máquina.

Sprites con vida

```
  5 REM *Programa 1*
 10 CLEAR 45055: REM AFFF EN HEXA
 20 LOAD "MOVESPRITE"CODE
 30 LET KEY = 45056
 40 LET BITPOS = 45059
 50 LET SPRTAB = 45060
 60 LET INIT = 45312: REM B100 HEX
 70 LET MOVSPR = 45317: REM B105 HEX
 80 GO SUB 1000: REM SET UP SPRITE
 90 FOR I = 1 TO 20
100 PRINT AT 21*RND,31*RND;"*";
110 NEXT I
120 PRINT AT 10,16;
130 RANDOMIZE USR INIT
140 LET X$ = INKEY$: IF X$ = "" THEN GO TO 140
150 LET X = VAL X$
160 POKE KEY,X
170 RANDOMIZE USR MOVSPR
180 GO TO 140
1000 POKE BITPOS,0
1010 FOR I = 0 TO 7
1020 READ X
1030 POKE SPRTAB+2*I,X
1040 POKE SPRTAB+1+2*I,0
1050 NEXT I
1060 RETURN
2000 DATA BIN 00011000
2010 DATA BIN 00011000
2020 DATA BIN 00011000
2030 DATA BIN 11111111
2040 DATA BIN 11111111
2050 DATA BIN 00011000
2060 DATA BIN 00011000
2070 DATA BIN 00011000
```

```
  5 REM *Programa 2*
 10 LET A = 45312
 20 FOR L = 1000 TO 1420 STEP 10
 30 LET S = 0
 40 FOR A = A TO A+7
 50 READ B
 60 POKE A,B
 70 LET S = S+B
 80 NEXT A
 90 READ C
100 IF C<>S THEN PRINT "ERROR DE LINEA ";L: STOP
110 NEXT L
120 READ B
130 POKE A,B
140 READ B
150 POKE (A+1),B
200 PRINT "INSERTE CINTA PROGRAMA"
250 SAVE "MOVESPRITE"CODE 45312,400
1000 DATA 42,132,92,24,44,22,0,205,561
1010 DATA 61,177,42,1,176,58,0,176,691
1020 DATA 254,5,32,5,205,165,177,24,867
1030 DATA 24,254,6,32,5,205,109,177,812
1040 DATA 24,15,254,7,32,5,205,136,678
1050 DATA 177,24,6,254,8,192,205,228,1094
1060 DATA 177,34,1,176,22,1,205,61,677
1070 DATA 177,205,35,178,201,42,1,176,1015
1080 DATA 221,33,4,176,14,8,229,6,691
1090 DATA 2,203,66,32,6,221,126,16,672
1100 DATA 119,24,4,126,221,119,16,35,664
1110 DATA 205,83,178,221,35,16,234,225,1197
1120 DATA 13,200,221,35,197,213,205,109,1193
1130 DATA 177,209,193,24,217,62,7,164,1053
1140 DATA 254,7,40,2,36,201,17,32,589
1150 DATA 0,25,62,224,165,32,4,205,717
1160 DATA 83,178,201,124,214,7,103,201,1111
1170 DATA 62,7,164,40,2,37,201,17,530
1180 DATA 32,0,167,237,82,62,224,165,969
1190 DATA 254,224,32,4,205,76,178,201,1174
1200 DATA 124,198,7,103,201,221,33,3,890
1210 DATA 176,221,126,0,61,230,7,221,1042
1220 DATA 119,0,79,221,35,6,8,221,689
1230 DATA 94,0,221,86,1,203,3,245,853
1240 DATA 203,11,241,203,18,203,19,62,960
1250 DATA 7,185,32,3,122,83,95,221,748
1260 DATA 115,0,221,114,1,221,35,221,928
1270 DATA 35,16,220,62,7,185,192,43,760
1280 DATA 205,76,178,201,221,33,3,176,1093
1290 DATA 221,126,0,60,230,7,221,119,984
1300 DATA 0,79,221,35,6,8,221,94,664
1310 DATA 0,221,86,1,203,11,245,203,970
1320 DATA 3,241,203,26,203,27,62,0,765
1330 DATA 185,32,3,122,83,95,221,115,856
1340 DATA 0,221,114,1,221,35,221,35,848
1350 DATA 16,220,62,0,185,192,35,205,915
1360 DATA 83,178,201,42,1,176,221,33,935
1370 DATA 4,176,14,8,229,6,2,221,660
1380 DATA 126,0,47,87,126,162,221,182,951
1390 DATA 0,119,35,205,76,178,221,35,869
1400 DATA 16,237,225,13,200,197,205,109,1202
```

```
1410 DATA 177,193,24,224,62,63,188,192,1123
1420 DATA 38,87,201,62,88,188,192,38,894
1430 DATA 64,201,265
```

```
KEY     EQU   £B000
SPRPOS  EQU   £B001
BITPOS  EQU   £B003
SPRTAB  EQU   £B004
DFCC    EQU   £5C84
        ORG   £B100
INIT    LD    HL,(DFCC)
        JR    SAVSCR
MOVSPR  LD    D,0
        CALL  UNDER
        LD    HL,(SPRPOS)
        LD    A,(KEY)
        CP    5
        JR    NZ,L0
        CALL  LMOVE
        JR    SAVSCR
L0      CP    6
        JR    NZ,L1
        CALL  BELOW
        JR    SAVSCR
L1      CP    7
        JR    NZ,L2
        CALL  ABOVE
        JR    SAVSCR
L2      CP    8
        RET   NZ
        CALL  RMOVE
SAVSCR  LD    (SPRPOS),HL
        LD    D,1
        CALL  UNDER
        CALL  PRSPRT
        RET
UNDER   LD    HL,(SPRPOS)
        LD    IX,SPRTAB
        LD    C,8
LINE    LD    B,2
BYTE    BIT   0,D
        JR    NZ,SVESCR
WIPOUT  LD    A,(IX+£10)
        LD    (HL),A
        JR    CONT
SVESCR  LD    A,(HL)
        LD    (IX+£10),A
CONT    INC   HL
        INC   IX
        DJNZ  BYTE
        DEC   C
        RET   Z
        INC   IX
        DEC   HL
        DEC   HL
        PUSH  BC
        PUSH  DE
        CALL  BELOW
        POP   DE
        POP   BC
        JR    LINE
BELOW   LD    A,7
        AND   H
        CP    7
        JR    Z,BDIFCB
BSAMCB  INC   H
        RET
BDIFCB  LD    DE,£20
        ADD   HL,DE
        LD    A,£E0
        AND   L
        RET   Z
BSAMSB  LD    A,H
        SUB   7
        LD    H,A
        RET
ABOVE   LD    A,7
        AND   H
        JR    Z,ADIFCB
ASAMCB  DEC   H
        RET
ADIFCB  LD    DE,£20
        AND   A
        SBC   HL,DE
        LD    A,£E0
        AND   L
        CP    £E0
        RET   Z
ASAMSB  LD    A,H
        ADD   A,7
        LD    H,A
        RET
LMOVE   LD    IX,BITPOS
        LD    A,(IX+0)
        DEC   A
        AND   7
        LD    (IX+0),A
        LD    C,A
        INC   IX
RMOVE   LD    IX,BITPOS
        LD    A,(IX+0)
        INC   A
        AND   7
        LD    (IX+0),A
        LD    C,A
        INC   IX
        LD    B,8
RPAIR   LD    E,(IX+0)
        LD    D,(IX+1)
        RRC   E
        PUSH  AF
        RLC   E
        POP   AF
        RR    D
        RR    E
        LD    A,0
        CP    C
        JR    NZ,RSTORE
        LD    A,D
        LD    D,E
        LD    E,A
RSTORE  LD    (IX+0),E
        LD    (IX+1),D
        INC   IX
        INC   IX
        DJNZ  RPAIR
        LD    A,0
        CP    C
        RET   NZ
        INC   HL
        RET
PRSPRT  LD    HL,(SPRPOS)
        LD    IX,SPRTAB
        LD    C,8
PRLINE  LD    B,2
PRBYTE  LD    A,(IX+0)
        CPL
        LD    D,A
        LD    A,(HL)
        AND   D
        OR    (IX+0)
        LD    (HL),A
        INC   HL
        INC   IX
        DJNZ  PRBYTE
        DEC   C
        RET   Z
        DEC   HL
        DEC   HL
        PUSH  BC
        CALL  BELOW
        POP   BC
        JR    PRLINE
        LD    B,8
LPAIR   LD    E,(IX+0)
        LD    D,(IX+1)
        RLC   E
        PUSH  AF
        RRC   E
        POP   AF
        RL    D
        RL    E
        LD    A,7
        CP    C
        JR    NZ,LSTORE
        LD    A,D
        LD    D,E
        LD    E,A
LSTORE  LD    (IX+0),E
        LD    (IX+1),D
        INC   IX
        INC   IX
        DJNZ  LPAIR
        LD    A,7
        CP    C
        RET   NZ
        DEC   HL
        RET
```

Cómo usar estos programas

1) Digite y guarde (SAVE) "Programa 1"
2) Digite y ejecute "Programa 2", que tomará el código máquina de la memoria y lo guardará (SAVE) en la cinta, a ser posible después de "Programa 1"
3) Cargue (LOAD) el "Programa 1", que se ejecutará automáticamente

En primera línea

Desde sus modestos comienzos, Motorola ha ido creciendo hasta llegar a la posición que ocupa actualmente: uno de los primeros fabricantes de componentes microelectrónicos del mundo

Robert Galvin, presidente de Motorola

Al igual que muchas otras empresas de éxito, Motorola empezó como el negocio de un solo hombre. La fecha de creación de la firma se remonta a 1928, cuando Paul Galvin fundó la Galvin Manufacturing Corporation, en Chicago, que se especializó en la producción de receptores de radio para el hogar. Durante los años treinta la empresa se diversificó, fabricando radios para la policía y para automóviles bajo la marca comercial "Motorola". En los años cuarenta la empresa (cuyo nombre ahora es Motorola Incorporated) fue una de las primeras firmas de electrónica en producir semiconductores.

Paul Galvin falleció en 1959 y lo sucedió como presidente su hijo, Robert. Durante la década siguiente otros fabricantes, en particular japoneses, empezaron a competir con Motorola en los mercados del semiconductor y de la electrónica de consumo. La recesión mundial de mediados de los setenta hizo que la empresa sufriera enormes pérdidas y se viera forzada a replantear su estrategia. Se contrató nuevo personal, gran parte del cual provenía del rival por antonomasia de Motorola, Texas Instruments, y se tomó la decisión de abandonar el campo de la electrónica, en el cual la compañía ya no podía competir, para concentrarse, en cambio, en la microelectrónica de alta tecnología.

Ello implicó la venta de parte del activo de la firma (en particular el negocio de televisores en color), la inversión de fuertes sumas en investigación y desarrollo, y la adquisición de empresas en zonas nuevas en las que Motorola deseaba causar un gran impacto. Ello representaba un riesgo considerable, pero en aquel momento las alternativas que se le presentaban eran escasas.

Las oficinas centrales de Motorola en Illinois (Estados Unidos)

La apuesta parece haberse ganado. Durante la segunda mitad de la década de los setenta Motorola quedó muy a la zaga de las principales firmas fabricantes de semiconductores, pero después de grandes inversiones en nueva tecnología la empresa ahora afirma estar pisándole los talones a Texas Instruments, líder del mercado. Como comenta Robert Galvin: "Las empresas que solían hacerle la competencia a Motorola han quedado en el camino porque no se han adaptado al medio".

Motorola ha seguido teniendo problemas para lograr que sus productos salieran a la venta en el momento adecuado. A mediados de los setenta, cuando la industria del microordenador estaba en pañales, el microprocesador Motorola 6800 fue superado, en cuanto a volumen de ventas, por el Mostek 6502, que fue adoptado por Apple para sus ordenadores personales, de tan fabuloso éxito, y por el Intel 8085 y el Zilog Z80, que utilizan los ordenadores con CP/M. La empresa introdujo el 6809 en 1976; éste fue reconocido a nivel general como el mejor microprocesador de ocho bits de todos los existentes, pero la carrera por el mercado masivo ya se había perdido y el chip sólo apareció en unos pocos micros personales, como el Tandy Color y el Dragon.

No obstante, la empresa ha seguido realizando grandes inversiones en investigación ("para sacar la máxima ventaja lo antes posible", según declara Robert Galvin) y ahora está mucho mejor situada en la carrera por el mercado de 16 bits. El microprocesador 68000 se lanzó en 1979, aunque no estuvo disponible ampliamente hasta 1982. Este procesador es el que ha adoptado Apple para sus microordenadores Lisa y Macintosh, y Sinclair Research para su QL. Se trata de un dispositivo extremadamente potente que contiene 17 registros de 32 bits, un bus de datos de 16 bits y un bus de direcciones de 24 bits.

Motorola continúa desarrollando nuevos productos en sus centros de investigación de Phoenix (Arizona), Ginebra (Suiza) y East Kilbride (Escocia). La fábrica de East Kilbride fabrica chips CMOS (semiconductores de óxido metálico complementario) y MOS (semiconductores de óxido metálico) para una amplia gama de aplicaciones. En la actualidad, la empresa está organizada en cinco grupos, que se ocupan de comunicaciones, semiconductores, sistemas de información, electrónica para la automoción e industrial, y electrónica para la administración. A pesar de la baja rentabilidad de algunos de sus departamentos, Motorola vio ascender sus ventas a 1,26 billones de dólares en el primer trimestre de 1983 y parece dispuesta a mantener la sólida posición que ocupa en estos momentos en el mercado mundial de la microelectrónica.

La imagen ideal

Liz Heaney Software, cortesía de Pilot Software

La importancia de la presentación
Estos dos programas basados en disco cuestan alrededor de 9 000 pesetas cada uno, pero uno de ellos se ha empaquetado en una forma imaginativa, que atrae la atención y sugiere que vale la pena pagar su precio, mientras que el otro, en comparación, parece una caja demasiado convencional cuyo precio resulta muy elevado

Tanto en la comercialización de un nuevo ordenador como en la de un paquete de juegos es básico crear la "imagen del producto"

El *marketing* consiste en construir un puente entre el producto que los vendedores tienen que vender y el dinero del que disponen los potenciales compradores. Al diseñar un puente que estimule y resista la máxima cantidad de tráfico, los agentes de marketing tienen que hacer suposiciones (algunas veces basadas en prospecciones de mercado) sobre las necesidades, deseos y caprichos de su público, y después respaldar esas conclusiones con importantes inversiones.

Las decisiones relativas a la comercialización empiezan a tomar forma cuando la producción de una nueva máquina se halla en fase de planificación. Las funciones que tendrá la máquina, la cantidad de unidades que se fabricarán y cuánto se puede gastar en la producción de cada unidad, todos son factores que incidirán en la comercialización.

A la comercialización del software se le aplican consideraciones similares. No tiene sentido gastar grandes sumas de dinero desarrollando y vendiendo un juego que, para recuperar la inversión, resultará más caro de lo que se pueden permitir sus compradores potenciales. Pero una firma de software que escatime dinero en desarrollo se encontrará vendiendo un producto que se queda corto en prestaciones o que está plagado de errores, o ambas cosas a la vez, corriendo el riesgo, por lo tanto, de que su marca comercial se desprestigie, con un efecto potencialmente pernicioso para sus futuros productos. Por el contrario, si se invierte muchísimo dinero en desarrollo pero nada en comercialización, se encontrará vendiendo un producto espléndido pero que nadie conoce.

También es importante determinar ya en una etapa temprana el lugar que ocupa un producto en relación a artículos similares existentes en el mercado. Es aquí donde la "imagen" del producto adquiere una importancia vital. Los fabricantes de cigarrillos y de jabón en polvo comparten una dificultad: a la hora de la verdad, todos los productos se parecen mucho entre sí. Los fabricantes de hardware para ordenadores no se hallan exactamente en esta misma situación, pero no siempre es fácil explicar con precisión el carácter individual de una máquina a quienes acuden a adquirir una.

Por este motivo los fabricantes de hardware y de software, al igual que los fabricantes de cigarrillos y de jabón en polvo, recurren a la creación de una

"imagen del producto", una especie de proyección psicológica sintetizada del producto. Es más eficaz vender una idea que un mero producto, principio tan bien reflejado en el axioma publicitario que afirma que "vende más el olor que la salchicha".

El "diseño industrial" de la máquina (su aspecto exterior y el trazado exterior de sus interruptores y teclas de función) con frecuencia es un punto de partida útil para desarrollar la imagen de un producto de hardware. El BBC Micro, por ejemplo, es sobrio y utilitario, como corresponde a su publicidad, que subraya siempre su especial valor educativo. Las carcasas de otras máquinas suelen estar decoradas con molduras y logotipos llamativos para seducir al amante de los juegos. La gama Atari XL (cuyo nuevo estilo responde a un esfuerzo de marketing por rescatar al producto de la depresión de mediados de 1983) tiene un estilo austero, de bordes romos y aspecto militar; muy adecuado para jugar a *Tank battle* (Batalla de tanques). Commodore, en su campaña para introducirse en el mercado norteamericano, le encargó a Ferdinand Porsche, el famoso diseñador de automóviles, que mejorara la presentación de su gama con formas elegantemente redondeadas, que le dieran, con sutileza, un toque futurista.

Por el contrario, el aspecto físico del Sinclair Spectrum, con su pequeña carcasa negra y sus teclas multifunción cuidadosamente rotuladas, supone una máquina que ofrece mucho en muy poco espacio, con una clara sugerencia de que lo que proporciona supera con creces lo que vale.

La imagen de un producto va más allá de consideraciones sobre su aspecto físico. La imagen se debe difundir mediante una plataforma publicitaria coherente que refuerce la idea. La supuesta "memoria de elefante" del Commodore 64 se vende muy bien incluyendo un paquidermo en los anuncios en revistas y televisión. Para el BBC, el logotipo del búho evoca en la mente del público la idea de la "sabiduría".

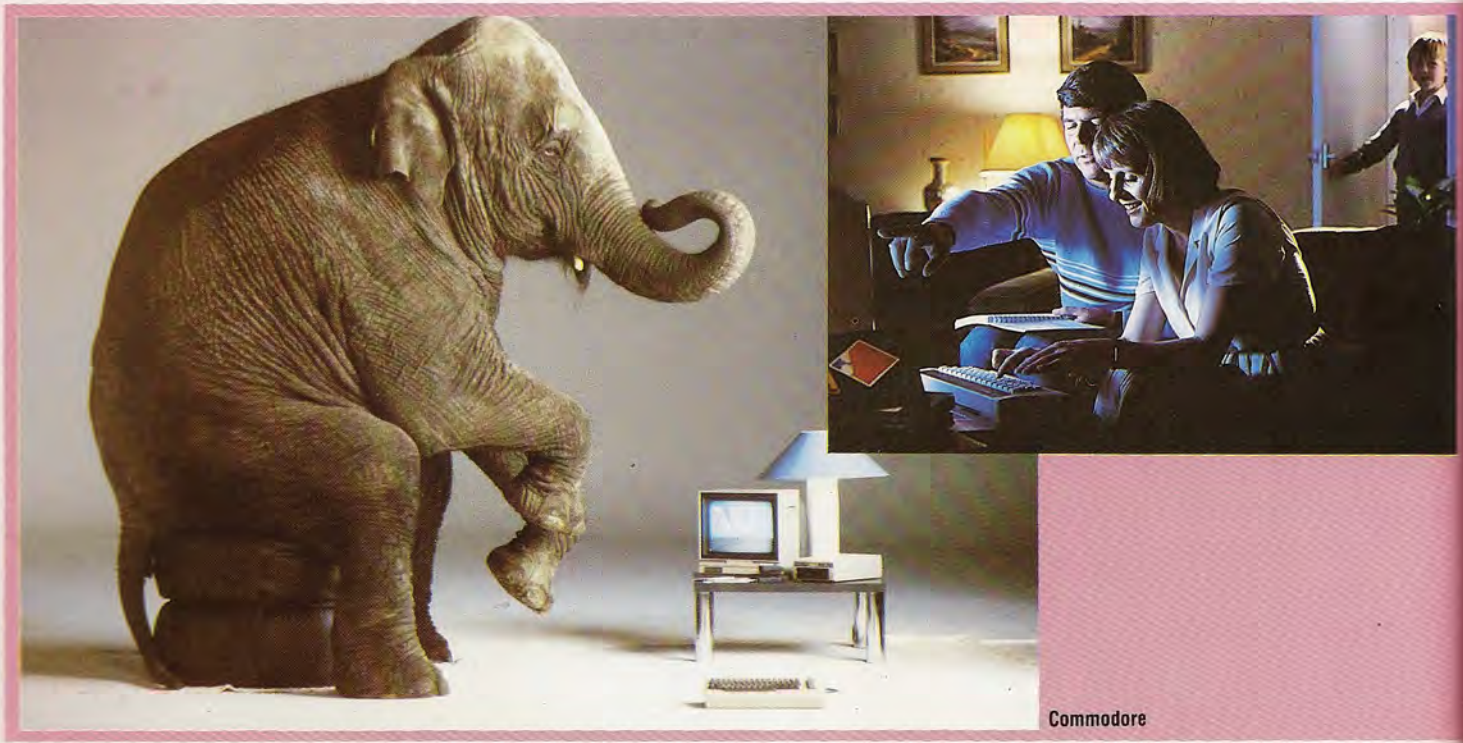
Naturalmente, el nombre es un punto importante. Los primeros ordenadores personales tenían que lidiar arduamente contra una imagen arraigada en la mente del público por las películas de los años sesenta y principios de los setenta en las que el ordenador aparecía invariablemente como un ente inhumano cuyo control escapaba a los simples mortales, a imagen y semejanza del "Hermano Mayor", el omnipresente dictador de *1984*, la novela de anticipación de George Orwell. Debido a ello a los micros se les asignaban nombres domésticos que deliberadamente rechazaban toda posible asociación con una tecnología elevada. De ahí el PET (mascota) y el Apple (manzana).

El Macintosh se anunció en Gran Bretaña y Estados Unidos mediante una insistente campaña de televisión filmada en aquélla por el director cinematográfico Ridley Scott, en la que se veía a una mujer haciendo añicos la pantalla de un "Hermano Mayor", para representar la recién hallada libertad que ofrecía el último producto de Apple. El slogan era "Gracias al Macintosh, 1984 no será como 1984". Para el mercado norteamericano, "Mackintosh" (con "k") es una variedad de manzanas.

Pero la gente quiere estar segura de que no se le está vendiendo un mero juguete. El PET encontró una gran resistencia en sus ventas en este punto cuando Commodore intentó desarrollar el mercado de gestión a finales de 1970. La primera línea de ataque fue sugerir que el nombre era un acrónimo de Personal Electronic Transactor (gestor electrónico personal), que sonaba mucho más científico, pero la táctica no resultó muy convincente y hubieron de recurrir a cambiar el nombre del micro por el de Commodore Business Computer.

La idea de la alta tecnología se puso de moda a medida que la gente se iba sintiendo cada vez más cómoda con la idea de disponer de ordenadores en su entorno. En esta categoría, la efectividad de los nombres de los productos depende de apartarlos lo más posible del lenguaje cotidiano, de modo que se

Imágenes
En marketing, la importancia de crear una imagen de producto poderosa se aprecia claramente en nuestro ejemplo: el paquidermo del Commodore sugiere la "memoria de elefante" del Commodore 64; las escenas hogareñas y escolares del BBC transmiten la flexibilidad, cordialidad e importancia educativa del Electron; mientras que el escenario de Apple, bastante más fantástico con sus alusiones orwellianas, les promete a los usuarios del Apple la liberación de los trabajos fatigosos en un mundo lleno de ordenadores humanizados. Sin embargo, al igual que la mayoría de las técnicas de gran poder, pueden tener efectos imprevistos: proverbialmente, al elefante lo aterrorizan los ratones (a diferencia del Macintosh y el Lisa) y podría sugerir un producto anticuado; a los potenciales compradores del Electron el ambiente familiar les podría resultar aburrido y estereotipado, y la asociación con la escuela, intimidante; los clientes de Apple podrían pensar que se los está retratando como clones sin inteligencia



Commodore

prefieren acrónimos o incluso grupos de letras que no quieren decir nada, antes que palabras que se encuentran en el diccionario. Las letras más raras, las que más puntos valen en el *scrabble* —juego que consiste en crear el mayor número de palabras cruzadas con el menor número de letras—, también son las que más valen aquí. Y de ahí máquinas como el ZX81, el MTX500 y el MZ-700.

El empaquetamiento, que en el caso del hardware es meramente protector, reviste una trascendental importancia en la imagen del software. En líneas generales, los vendedores de software pueden seguir dos estrategias: gastarse sólo un mínimo en empaquetamiento (la caja de la cassette con un folleto en color en su interior) y un correspondiente precio "de ganga", o construir el "valor percibido" del producto poniéndolo en una caja más grande, que a menudo tiene la apariencia de un libro, y añadirle extras.

La imagen se proyecta a través de la publicidad. En términos generales, la relación entre marketing y publicidad es la relación entre estrategia y táctica. Las cuestiones acerca de cuándo anunciar y a cuánto debe ascender el presupuesto de publicidad son decisiones de marketing.

Pero la promoción de la imagen del producto no siempre ayuda a vender cuando la imagen es mala. Hubo una campaña de comercialización para cigarrillos que ya es leyenda en el ambiente publicitario. El nombre comercial «Strand» se promocionó ampliamente en anuncios para cine y televisión, asociado a un hombre solitario enfundado en un impermeable blanco; el slogan rezaba: "Nunca estás solo si tienes un Strand". Pero el mensaje que se deducía era: "Los solitarios fuman Strand", y la marca no se vendió.

La imagen de algunos ordenadores existentes puede ser como un *boomerang* en este mismo sentido. El elefante del Commodore podría servir para recordar que la escritura de programas en el 64 es pesada.

Se podría decir que la creación y la proyección de las imágenes son el lado "creativo" de la comercialización (así lo creen en el ambiente publicitario). Pero la logística del marketing es igual de importante o quizá aún más, y con mucha frecuencia éste es el eslabón débil de la cadena. Una cosa es estimular la imaginación del público sobre un nuevo producto, y otra muy distinta, y más ardua, es introducirlo en sus hogares u oficinas.

¿Ha de realizarse la distribución en forma de pedidos por correspondencia, al estilo Sinclair? Esta es una forma económica de hacerles llegar los productos a los clientes, pero ¿cómo les asegura un buen soporte? Probablemente habrá que vender a precios de ganga para contrarrestar la seguridad que proporciona comprar en una tienda, pero si se reducen demasiado los márgenes de beneficio podría haber dificultades para financiar la producción en masa, con las consiguientes demoras en la entrega.

Vender a través de las cadenas de tiendas ya establecidas les proporcionará a los clientes una fuerte sensación de seguridad en su compra, pero estos establecimientos comerciales exigirán la contrapartida de un mayor porcentaje para la tienda, reduciendo nuevamente los beneficios del fabricante. Un paquete de software cuyo precio de venta al público sea de 1 100 pesetas reportará menos de 500 pesetas después de que la cadena haya deducido su parte. También cabría pensar en instalar una red de distribuidores seleccionados, cada uno de ellos capaz de dar a sus clientes ayuda y consejo especiales sobre sus productos.

En la lucha por la supervivencia en el mundo de los micros, la desaparición de algunos fabricantes de excelente hardware y software ha puesto de manifiesto el hecho de que fabricar productos con frecuencia es menos de la mitad de la batalla. Donde la historia empieza realmente es en crear y mantener mercados. Y ahí es donde todo empieza a resultar difícil.

Una flor de calidad

Las impresoras de rueda margarita realizan una labor eficaz y poseen características tan útiles como el espaciado proporcional

A primera vista, una impresora de rueda margarita podría parecer una compra extraña para el usuario de un ordenador personal. No es realmente adecuada para listar programas, es lenta y cuesta más que una impresora matricial. No obstante, para algunas aplicaciones es una buena elección. El área en la cual la impresora de rueda margarita gana mucho terreno es en la calidad de la impresión. Una impresora matricial construye cada carácter imprimiendo un patrón de puntos: independientemente de cuántas agujas se utilicen en el cabezal de impresión, los puntos individuales siempre se ven en el texto impreso.

Con una impresora de rueda margarita, por el contrario, los caracteres se producen mediante un conjunto de tipos que golpean contra una cinta entintada, al igual que en una máquina de escribir. Estos bloques de tipos, uno para cada carácter, están dispuestos en círculo, como los pétalos de una flor, de ahí su nombre: *rueda margarita*. La impresión es más fácil de leer y parece más profesional.

La contrapartida por esta superior calidad de impresión se encuentra en la velocidad de la máquina: las impresoras de rueda margarita son mucho más lentas que las matriciales de igual precio. El motivo de esta lentitud radica en sus distintos métodos de impresión. Para imprimir un carácter utilizando una impresora margarita, la rueda de impresión ha de girar primero hasta que el "pétalo" requerido quede en la parte superior; luego el martillo de impresión golpea el tipo seleccionado y el carro se desplaza para producir el carácter siguiente.

Compare esto con el funcionamiento de una impresora matricial, donde los puntos se imprimen a medida que el carro se desplaza a través del papel, y podrá comprender la diferencia de velocidad entre los dos tipos de impresora. Una rueda margarita suele imprimir alrededor de 20 caracteres por segundo (cps); una impresora matricial Epson FX-80 vale aproximadamente lo mismo, pero puede imprimir a una velocidad de 160 cps. Algunas impresoras de rueda margarita son más rápidas, pero su precio asciende a más del doble que el de las impresoras margarita normales.

Para aumentar la velocidad de impresión, tanto las impresoras de rueda margarita como las matriciales con frecuencia poseen dos características extras: impresión bidireccional y búsqueda lógica. *Bidireccional* significa simplemente que una línea de texto se imprime de izquierda a derecha y la línea siguiente se imprime de derecha a izquierda. La impresora no ha de esperar que el carro retorne y, por lo tanto, imprime más rápido. *Búsqueda lógica* significa que el carro se salta espacios para llegar a la siguiente palabra del texto; las impresoras menos sofisticadas tardan el mismo tiempo en "imprimir" un espacio que cualquier otro carácter.

Las impresoras matriciales tienen las formas de sus caracteres almacenadas en memoria ROM dentro de la impresora; las impresoras de rueda margarita tienen el carácter almacenado como bloques de tipos en la rueda de impresión. Cada método posee sus ventajas: con una impresora matricial las formas de los caracteres se pueden redefinir enviándole a la impresora, desde el micro, códigos de escape adecuados. Con una impresora de rueda margarita el proceso es mucho más sencillo: simplemente se cambia la rueda corriente por una distinta.

Las ruedas margarita vienen en diversos estilos y espaciados de tipos; el *estilo de tipos* alude al diseño de los caracteres, y el *espaciado* a su anchura. Algunos de los estilos de tipos más comunes son el Courier, Romano, Gótico y cursiva. El espaciado normalmente es de 10 o 12 caracteres por pulgada. Una rueda margarita puede ser de plástico o de metal. Las metálicas poseen la ventaja, respecto a las plásticas, de que tienen una vida útil mucho más larga, pero son mucho más caras que éstas.

Sin embargo, estos diferentes estilos presentan el inconveniente de que pocos de ellos son exactamente iguales que el juego de caracteres que utiliza un micro. Esto significa que algunos de los caracteres del teclado de su micro se imprimirán como algo totalmente diferente. A menudo el carácter "numérico" (#) imprime un signo de libras (£), o un corchete ([) se imprime como una fracción (½). En muchos estilos de tipos el número "0" (cero) no se diferencia de la letra "O" mayúscula y, del mismo modo, la "l" minúscula se puede tomar por un número "1". Mientras que las impresoras de rueda margarita más caras poseen ruedas de impresión de 127 caracteres, la mayoría sólo pueden imprimir 92 o 96 caracteres y son éstas las que sufren más este tipo de problema.

Como usted se puede imaginar, tratar de depurar un programa no resulta nada sencillo si uno no está seguro de qué caracteres son "unos" y cuáles son "eles". Por esta causa, y también debido a su lentitud, una impresora de rueda margarita no es aconsejable si el usuario utiliza su ordenador básicamente para programar.

Efectos especiales

Una impresora de rueda margarita se puede programar para que produzca diversos efectos especiales, al igual que las impresoras matriciales. A pesar de que el número de estos efectos es limitado, el método para programar la impresora es idéntico al que se utiliza para una impresora matricial, es decir, enviando códigos de escape (véase p. 804). Por ejemplo, en una impresora de rueda margarita Diablo, el código ESC-E activa el subrayado automático y ESC-R lo desactiva. Utilizando el BASIC Mi-

crosoft estándar, se entraría LPRINT CHR$(27);"E"; y LPRINT CHR$(27);"R"; para enviar a la impresora los códigos antes mencionados. Otros códigos incluyen ESC-1 para fijar una tabulación; ESC-9 para fijar el margen izquierdo; y ESC-U para hacer saltar el papel media línea hacia arriba (útil para subíndices). En la rueda margarita, el equivalente de "enfatizar" en una impresora matricial se denomina *destacar* (cada carácter se imprime cuatro veces para que resalte).

Algunas impresoras de rueda margarita permiten que se varíe tanto la distancia entre caracteres como el espaciado de líneas. En estos casos se puede utilizar la impresora para crear imágenes gráficas o vuelcos de pantalla, al igual que en una impresora matricial (véase p. 824). Si un pixel de la pantalla está "encendido", la rueda margarita imprime un punto; si está "apagado", entonces se imprime un espacio. Reduciendo la distancia entre caracteres se puede imprimir sobre papel una línea horizontal completa de la pantalla. De forma similar, el espaciado entre líneas se puede reducir hasta no dejar ningún espacio vacío entre una y otra línea. El proceso, sin embargo, es muy lento.

Una característica de la que carecen las impresoras matriciales es la capacidad de centrar títulos automáticamente. Enviándole ESC= a una impresora Diablo, ésta centrará el resto de esa línea entre los márgenes. Otra característica novedosa es el tabulador decimal: ESC–H hará que todos los números se impriman con las comas decimales alineadas, lo cual es muy útil para sumas de dinero.

Las impresoras de rueda margarita más caras por lo general pueden efectuar espaciado proporcional. Con esta característica el espaciado no es el estándar de 10 o 12 caracteres por pulgada, sino que varía a tenor de la anchura del carácter que se esté imprimiendo. Por ejemplo, el carácter "w" es mucho más ancho que el carácter "i", de modo que con el espaciado proporcional el carro no se iría tan lejos para una "i" como lo haría para una "w". Esto significa que los caracteres de las sucesivas líneas de texto no quedan exactamente uno debajo del otro, y el efecto global resulta visualmente más agradable. Las líneas de texto que usted lee en este momento están espaciadas proporcionalmente.

En el caso de una oficina, la necesidad de una impresora de rueda margarita puede estar plenamente justificada; para la mayoría de los usuarios de ordenadores personales, probablemente no. Existe, sin embargo, una alternativa: adaptar una máquina de escribir electrónica. Las impresoras de rueda margarita cuestan más que las máquinas de escribir electrónicas, a pesar de que ambas utilizan el mismo procedimiento de impresión. No obstante, hasta hace muy poco tiempo las máquinas de escribir no se podían conectar fácilmente con un ordenador: carecían de circuitos de interface o de conector RS232. Pero en la actualidad todas las máquinas de escribir electrónicas que existen en el mercado vienen con una interface para ordenador incorporada o bien se pueden equipar con un kit de interface. La gran ventaja de adaptar una máquina de escribir electrónica, aparte del ahorro de dinero que representa respecto a una impresora de rueda margarita comparable, es que se la puede seguir utilizando como una máquina de escribir convencional. Así, usted posee tanto una máquina de escribir como una impresora de rueda margarita.

La calidad tiene su precio

Imprimir muestra con 10 caracteres por pulgada
Imprimir muestra con 12 caracteres por pulgada
Imprimir muestra con 15 caracteres por pulgada

Para subíndices se utiliza el salto de media línea:

H_2O

ESC-E activa el subrayado automático y ESC-R lo desactiva.
La impresión destacada hace que el texto **resalte** del resto del texto.

El código ESC= se utiliza para centrar texto:

Un título centrado

Estas líneas de texto se imprimieron sin la facilidad de espaciado proporcional. Observe especialmente cómo quedan espaciados los números 0 1 2 3 4 5 6 7 8 9. Sin el espaciado proporcional, la anchura de cada carácter es la misma. Por ejemplo, una "w" tiene el mismo ancho que una "i":
WWWWWWWWWWW
iiiiiiiiiii

Estas líneas de texto se imprimieron utilizando la facilidad de espaciado proporcional. Observe especialmente cómo quedan espaciados los números 0123456789. Utilizando el espaciado proporcional, la anchura de cada carácter varía. Por ejemplo, una "W" es mucho más ancha que una "i":
WWWWWWWWWWW
iiiiiiiiiii

El efecto global es que el texto resulta más atractivo.

Más cara y menos flexible que una impresora matricial, la de rueda margarita produce una impresión de calidad similar a una máquina de escribir, con espaciado proporcional. La rueda propiamente dicha (con clara apariencia de flor) se sustituye fácilmente por otra de tipos o estilo diferentes

Leer en el teclado

Aquí le proporcionamos un programa sencillo para mover un "lápiz de gráficos" desde el teclado del Spectrum

El BASIC le permite al usuario entrar la información desde el teclado utilizando INPUT e INKEY$. La sentencia INPUT lee un número o serie de caracteres, terminados por ENTER, sobre una variable numérica o de tipo string (o alfanumérica). La función INKEY$ explora el teclado y devuelve o bien un string conteniendo el carácter de la tecla pulsada o, si no se ha pulsado ninguna, un string vacío. Se trata de dos instrucciones muy útiles, pero para algunas aplicaciones especiales (por ejemplo, cuando se deben leer simultáneamente combinaciones de teclas) la función IN se puede utilizar para leer directamente el teclado.

Las teclas del Spectrum están conectadas al microprocesador Z80 a través de puertas de entrada-salida. Hay 65 536 puertas de entrada-salida y cada una de ellas se puede direccionar individualmente. De la misma manera en que se usan PEEK y POKE para leer o escribir en la memoria, IN y OUT se utilizan para leer y escribir en las puertas de entrada-salida. La función IN(m) devolverá el valor de la puerta m, mientras que la sentencia OUT(m,n) escribirá el valor n en la puerta m.

El teclado se compone de cuatro filas de 10 teclas, estando dividida cada fila en medias filas de cinco teclas. Cada media fila corresponde a una puerta, como refleja la ilustración. Observe que las teclas de las medias filas izquierdas se corresponden con sus puertas de derecha a izquierda, mientras que las medias filas de la derecha se corresponden de izquierda a derecha.

Los cinco bits más hacia la derecha de una puerta se corresponden cada uno de ellos con una tecla, y tomarán el valor 0 si se ha pulsado la tecla que les corresponde, o 1 en caso contrario. Los bits marcados con una X son irrelevantes: pueden contener cualquier valor binario. Esto implicaría que el valor de IN(m) quedaría indeterminado. Este problema se puede superar poniendo por software el valor de los tres bits superiores a cero, mediante la utilización de la siguiente instrucción:

```
DEF FN a(p) = p - INT(p/32)*32
```

donde p es el valor de la puerta (el byte indeterminado devuelto por IN(m). La división p/32 dará un número desde 0 hasta justo por debajo de 8. Tomando INT(p/32) se descartará la fracción decimal de este número, dejando un valor entero entre 0 y 7. Multiplicando éste por 32 se obtiene luego un valor que representa los tres bits superiores (más hacia la izquierda) de p. Restándoselos al propio p

En contacto
El teclado del Spectrum está formado por ocho bloques de cinco teclas, y cada bloque se controla constantemente mediante un byte exclusivo o "puerta". Cada tecla se corresponde con un bit de su puerta; mientras se mantiene pulsada una tecla, su bit "de señal" pasa de uno (su estado normal) a cero. Aquí se están pulsando las teclas Y, I y O, de modo que el valor binario de su puerta, el byte 57342, es XXX01001 (los bits del 5 al 7 son irrelevantes). De forma similar, se están pulsando A y S, de modo que el valor del byte 65022 es XXX11100

Kevin Jones

se eliminan estos tres bits, dejando un valor entero entre 0 y 31, que representa los cinco bits inferiores (más hacia la derecha) de p. El valor de FN a(p), por consiguiente, siempre se definirá con los tres bits superiores a cero, cualquiera que sea el valor de los tres bits superiores de p.

Por ejemplo, si se pulsara la tecla V, la puerta 65278 contendría:

X	X	X	0	1	1	1	1

FNa(IN(65278)), por lo tanto, retornará el valor:

0 0 0 0 1 1 1 1 = 15

Si se pulsara al mismo tiempo Caps Shift y la letra V, la puerta contendría:

X	X	X	0	1	1	1	0

y FN a(IN(65278)), por consiguiente, devolvería el valor 14. Si no se pulsara ninguna tecla, la puerta devolvería el valor 31 (00011111).

Pruebe este programa:

```
10 REM Imprimir el valor enmascarado de las
   puertas
20 REM Definir la función de máscara
30 DEF FN a(p) = p-INT(p/32)*32
40 REM Imprimir las puertas
50 PRINT AT 1,1;"Puerta          valor
   enmascarado"
60 PRINT "32766",FN a(IN(32766))
70 PRINT "49150",FN a(IN(49150))
80 PRINT "57342",FN a(IN(57342))
90 PRINT "61438",FN a(IN(61438))
100 PRINT "63486",FN a(IN(63486))
110 PRINT "64510",FN a(IN(64510))
120 PRINT "65022",FN a(IN(65022))
130 PRINT "65278",FN a(IN(65278))
140 FOR i = 1 TO 250:NEXT i
150 CLS
160 GO TO 50
```

Después de pulsar RUN, intente oprimir algunas teclas y observe cómo cambian los valores enmascarados de las puertas. Intente predecir los valores que se visualizarán para las distintas teclas y combinaciones de teclas. Si pulsara las teclas del cursor (5, 6, 7 y 8) obtendría los siguientes valores enmascarados en las puertas 61438 y 63486 respectivamente (puerta A y puerta B):

	Puerta A (puerta 61438)	Puerta B (puerta 63486)
←	31	15
↓	15	15
↑	23	15
→	27	15

Si pulsara la tecla Caps Shift, obtendría el valor enmascarado 30 para la puerta 65278 (a la que nos referiremos como puerta C).

Consideremos el sencillo programa para gráficos que ofrecemos aquí. Este programa leerá las teclas del cursor para trazar líneas horizontales, verticales y oblicuas, y, en caso de pulsarse la tecla Caps Shift, para desplazar un "lápiz para gráficos" sin dibujar ninguna línea. Las líneas oblicuas se trazan pulsando dos teclas del cursor al mismo tiempo.

En el programa, la línea 30 define una función (FN(a)) para poner a cero los tres bits superiores, como antes. La posición horizontal del lápiz para gráficos se representa mediante x, siendo y su coordenada vertical. La línea 50 establece la posición inicial en la que se encontraba el lápiz en el centro de la pantalla.

Las líneas 70 y 80 leen las puertas que se corresponden con las teclas del cursor. La media fila de 6 a 0 incluye tres de las teclas del cursor y se corresponden con la puerta 61438 (puerta A). La media fila de 5 a 1 contiene la tecla del cursor con la flecha hacia la derecha y se corresponde con la puerta 63486 (puerta B). La línea 90 lee la tecla Caps Shift: la media fila desde V a Caps Shift corresponde a la puerta 65278 (puerta C).

Entre la línea 110 y la 180 se verifican las ocho combinaciones "legales" de las teclas del cursor y se ajusta la posición x,y del lápiz para gráficos:

La línea 110 comprueba si ↑
La línea 120 comprueba si ↑ y →
La línea 130 comprueba si →
La línea 140 comprueba si ↓ y →
La línea 150 comprueba si ↓
La línea 160 comprueba si ↓ y ←
La línea 170 comprueba si ←
La línea 180 comprueba si ↑ y ←

Entre las líneas 200 y 240 se asegura que el lápiz no se desplace fuera de los límites de la pantalla. Por último, de no haberse pulsado Caps Shift, la línea 250 trazará en la pantalla la nueva posición del lápiz para gráficos. La línea 260 cierra el bucle para repetir nuevamente todo el proceso.

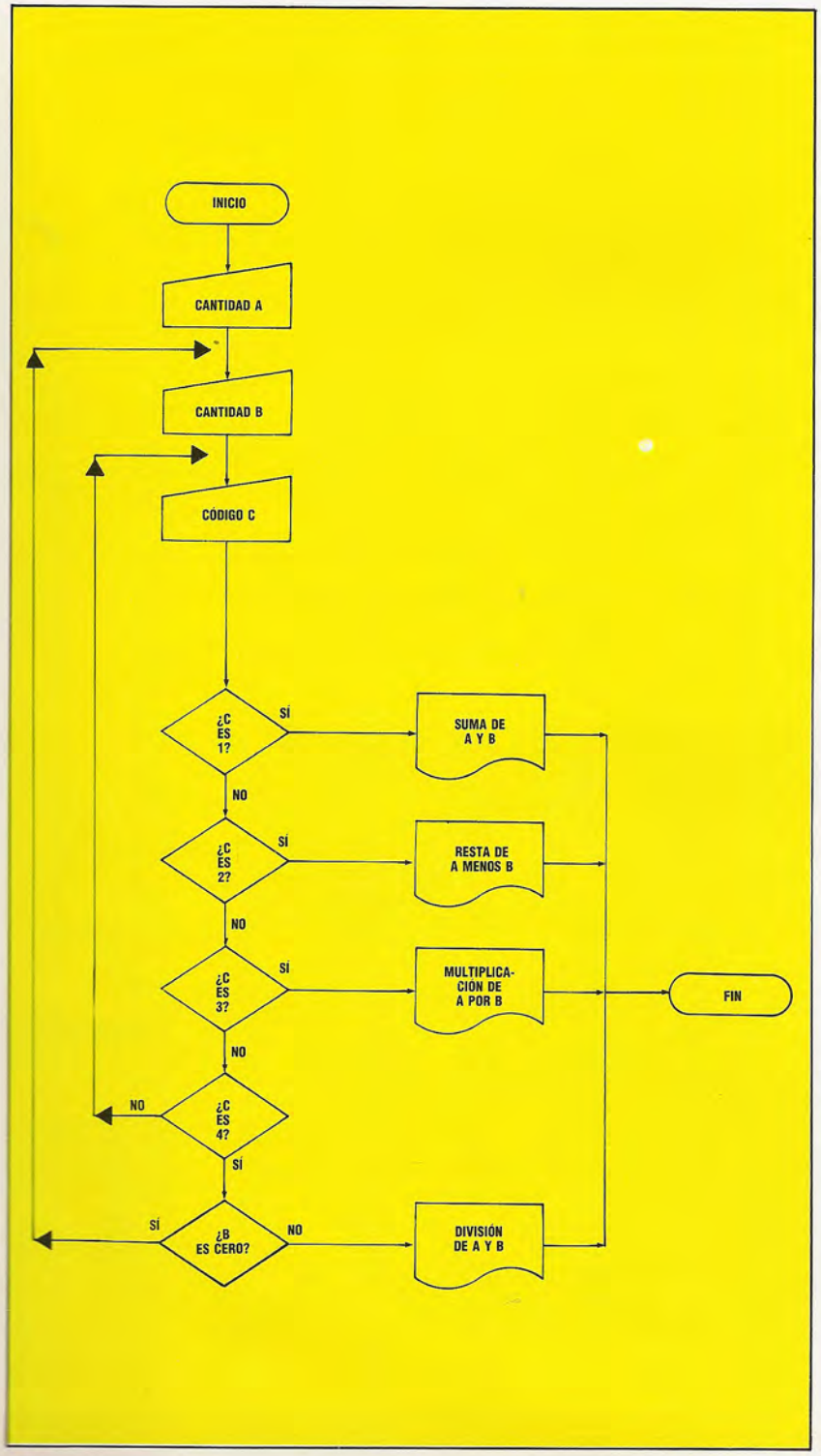
Un asunto cambiante
El programa emplea las técnicas analizadas para detectar múltiples pulsaciones de teclas: las teclas de flecha sin Caps Shift simultáneo permiten trazar líneas horizontales, verticales y oblicuas (oblicua=horizontal+vertical), mientras que las teclas pulsadas junto con Caps Shift producen los mismos movimientos del cursor sin trazado en pantalla. En un próximo capítulo utilizaremos las mismas técnicas para conseguir mejores efectos

Detección de pulsaciones múltiples de teclas

```
20 REM Preparar funcion para enmascarar los tres bits superiores
30 DF FN a(p) = p - INT (p/32)*32
40 REM Inicializar posicion lapiz
50 LET x = 127: LET y = 35
60 REM Leer puertas y eliminar tres bits superiores
70 LET puerta A = FN a(IN 61438)
80 LET puerta B = FN a(IN 63436)
90 LET puerta C = FN a(IN 65278)
100 REM Alterar posicion lapiz segun la tecla pulsada
110 IF puerta A = 23 AND puerta B = 31 THEN LET y = y+1
120 IF puerta A = 19 AND puerta B = 31 THEN LET x = x+1:
    LET y = y+1
130 IF puerta A = 27 AND puerta B = 31 THEN LET x = x+1
140 IF puerta A = 11 AND puerta B = 31 THEN LET x = x+1:
    LET y = y-1
150 IF puerta A = 15 AND puerta B = 31 THEN LET y = y-1
160 IF puerta A = 15 AND puerta B = 15 THEN LET x = x-1:
    LET y = y-1
170 IF puerta A = 31 AND puerta B = 15 THEN LET x = x-1
180 IF puerta A = 23 AND puerta B = 15 THEN LET y = y+1:
    LET x = x-1
190 REM Evitar que el lapiz se salga de la pantalla
200 IF x < 0 THEN LET x = 0
210 IF x > 255 THEN LET x = 255
220 IF y < 0 THEN LET y = 0
230 IF y > 175 THEN LET y = 175
240 REM Trazar punto si no se ha pulsado CAPS SHIFT
250 IF puerta C = 31 THEN PLOT x,y
260 GO TO 70
```

Test en cascada (2)

Continuando con este interesante test que iniciamos en el capítulo anterior, en esta ocasión introduciremos un nuevo elemento



En el capítulo anterior, refiriéndonos al ejemplo de los días de la semana, se hacía alusión a cómo se eliminaba una comparación; en aquel caso, se hacía por lógica, ya que sabemos que los días de la semana son siete. Por deducción, podría realizarse la misma operación, por ejemplo, con los meses del año, o bien con los dedos de la mano. Pero ello ocurre porque pertenecen a unas series limitadas cuyo número de elementos ya está establecido y es fijo; por ello, no podrían considerarse de igual modo otros casos cualesquiera.

Así, en estos casos nos encontramos con que deben controlarse unos valores determinados, pero variables, para poder realizar la eliminación de una pregunta tras la corriente sucesiva de otras. Por este motivo se deberá incluir una pregunta general al principio de la serie para que, de este modo, el dato entrado sea convenientemente filtrado antes de ser comparado.

El siguiente ejemplo lo desarrollaremos de dos formas: una es la que vemos representada a la izquierda; en ella no se ha tenido en cuenta ese filtro inicial, realizándose el mismo por seguimiento de la secuencia. En el próximo capítulo del apartado de Diagramación realizaremos la representación con dicho filtro.

Representación sin filtro inicial

Entradas dos cantidades, debe entrarse una tercera, que hace las veces de código y que, por otra parte, también marca la operación que debe realizarse con las dos primeras:

1: suma
2: resta
3: multiplicación
4: división

Conviene recordar que hay que efectuar un control para que la segunda cantidad, caso de tratarse de un código 4 (división), no sea cero, ya que ello nos daría un error. En este ejemplo, el test de eliminación, el filtro que determina si el número es válido, se lleva a cabo solamente tras haber pasado por todas las preguntas y haberse comprobado que no se puede optar por ninguna salida. Esto se considera válido, aunque no del todo recomendable, ya que supone una pérdida de tiempo y de trabajo, al tenerse que cuestionar cada control. En el caso de que se tratara de una cantidad mayor de decisiones, sería necesario comparar con todas ellas antes de llegar a la conclusión de que el código que se había entrado era erróneo.

Una gran ampliación

El Torch Disk Pack permite ampliar el BBC Micro, convirtiéndolo en un auténtico ordenador de oficina

El BBC Micro se puede ampliar para responder a unas especificaciones de gestión completas. Si bien esta capacidad se incorporó en la máquina desde el principio, la primera empresa que hizo uso de la idea no fue el fabricante, Acorn, sino Torch. El Torch Disk Pack permite que el usuario del BBC utilice la amplia gama de software que emplea el sistema operativo CP/M. Así y todo, el paquete completo vale apenas un poco más que un par de unidades de disco Acorn.

El sistema operativo VP/M requiere un microprocesador Z80 y unidades de disco. El BBC utiliza como microprocesador un 6502, pero el Torch Disk Pack le agrega un Z80, dejando que el 6502 del BBC maneje todas las funciones de entrada y salida. El 6502 controla el teclado, la pantalla y las unidades de disco, mientras el Z80, con sus 64 Kbytes de RAM, ejecuta los programas CP/M y, por regla general, "se hace cargo" del sistema. Los datos se pasan sucesivamente, una y otra vez, de un microprocesador a otro a través de la puerta de ampliación o "tubo" del BBC.

El procesador Z80 y sus 64 Kbytes de RAM se suministran en una pequeña placa de circuito impreso que se instala en el interior del BBC. El sistema combinado no utiliza la fuente de alimentación eléctrica de éste. En cambio, se conecta al BBC una nueva fuente de alimentación, situada en la caja principal del Disk Pack. La caja principal de éste contiene, asimismo, las dos unidades de disco flexible. Éstas son unidades de doble cara de 80 pistas, con unas características muy parecidas a las unidades de 800 Kbytes existentes para el BBC que fabrica Acorn.

Torch también ofrece un sistema alternativo. El ZEP 100 es esencialmente un Torch Disk Pack sin las unidades de disco y está pensado para aquellos usuarios de un BBC que ya posean sus propios discos. Torch también comercializa una gama de ordenadores de oficina que se componen de una placa con el BBC Micro más las unidades de disco, con una pantalla y un modem incorporados.

El Disk Pack está diseñado para instalarse debajo del BBC, conectándose ambos mediante cables cortos. La caja posee exactamente las mismas dimensiones que el ordenador, de modo que el sistema completo tiene un aspecto impecable. Lamentablemente, el Disk Pack hace que el teclado quede levantado unos 7 cm encima del escritorio, lo que puede dificultar un poco la digitación en el sistema completo.

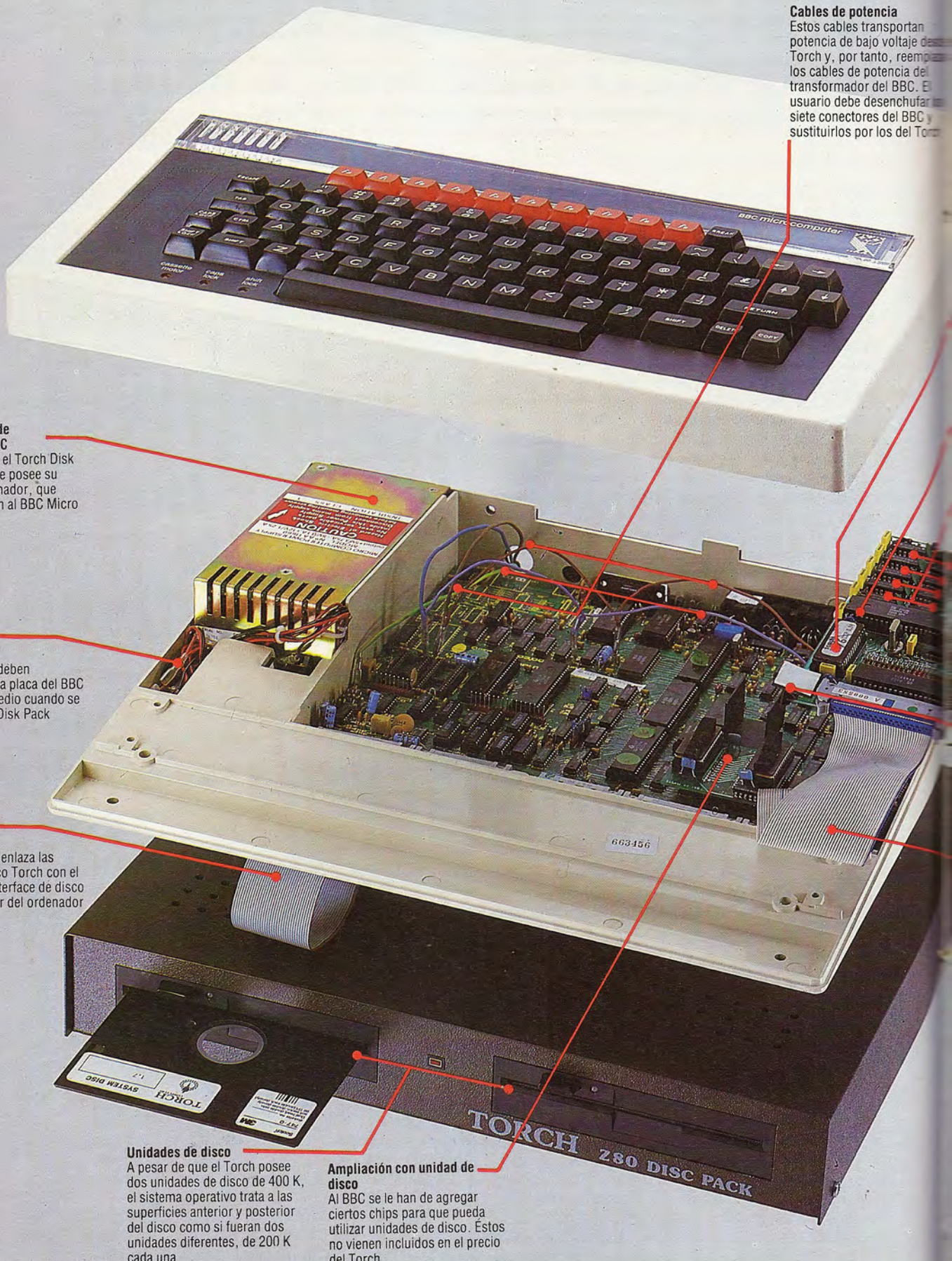
El Torch Disk Pack viene equipado con todos los elementos necesarios para mejorar el BBC, con una importante excepción: no incluye el juego de chips para interface de disco que precisa el BBC para utilizar una unidad de disco. Algunas personas ya las habrán pagado al adquirir un BBC Micro, pero la mayoría se habrán comprado un BBC Micro Modelo B estándar y, por tanto, tendrán que desembolsar una cantidad adicional para añadir los chips de interface. Los BBC Micro modelo A se deben ampliar hasta el modelo B para que puedan emplear el Torch Disk Pack.

Chris Stevens

BBC mejorado
El Torch Disk Pack convierte al BBC Micro en un ordenador de oficina porque incluye un procesador Z80 con 64 K de memoria que permite utilizar un sistema operativo similar al CP/M. Incluye asimismo dos unidades de disco de 400 K y con todo ello el sistema completo vale apenas un poco más que un par de unidades de disco Acorn

Una vez que el Torch Disk Pack está correctamente instalado y el sistema encendido, funciona directamente (en modalidad de visualización 3) como un ordenador CP/M. El sistema operativo en disco utilizado está escrito por Torch y se denomina MCP. Éste es muy parecido al CP/M y ejecutará la mayoría de los centenares de programas CP/M. La diferencia principal entre los sistemas a los que nos referimos estriba en que el MCP está almacenado en una ROM dentro del BBC y no es necesario cargarlo en el micro.

El MCP dispone de muchas instrucciones que a los usuarios del BBC les resultarán familiares. Mediante MODE se pueden seleccionar distintas modalidades de visualización, y también están todas las



Cables de potencia
Estos cables transportan potencia de bajo voltaje desde la Torch y, por tanto, reemplazan los cables de potencia del transformador del BBC. El usuario debe desenchufar los siete conectores del BBC y sustituirlos por los del Torch

Transformador de corriente del BBC
No se utiliza con el Torch Disk Pack, ya que éste posee su propio transformador, que alimenta también al BBC Micro

Salida de potencia
Estos cables se deben desenchufar de la placa del BBC y quitar de en medio cuando se instala el Torch Disk Pack

Interface de disco
Este cable plano enlaza las unidades de disco Torch con el conector para interface de disco de la cara inferior del ordenador

Unidades de disco
A pesar de que el Torch posee dos unidades de disco de 400 K, el sistema operativo trata a las superficies anterior y posterior del disco como si fueran dos unidades diferentes, de 200 K cada una

Ampliación con unidad de disco
Al BBC se le han de agregar ciertos chips para que pueda utilizar unidades de disco. Éstos no vienen incluidos en el precio del Torch

Chris Stevens

ROM/CCCP
El CCCP (Cambridge Console Command Processor) forma parte del MCP, el sistema operativo de Torch. El resto está en un chip en uno de los conectores de ROM del BBC

Microprocesador Z80
Éste asume el puesto del microprocesador 6502 del BBC Micro. Permite utilizar el sistema operativo Torch MCP. Dado que éste es similar al CP/M, se puede emplear una amplia gama de programas de gestión

RAM
Esta memoria es exclusiva para el procesador Z80, dejando los 32 K del BBC para utilizar como memoria de pantalla

Lengüetas de fijación
Estas lengüetas adhesivas pegan la placa del procesador Z80 en el interior de la carcasa del BBC Micro

Cable de conexión del Z80
Este cable plano conecta la placa del procesador Z80 con el BBC Micro a través de la interface "tubo"

instrucciones VDU y *FX. *KEY se utiliza para definir las teclas de función. Cuatro de las teclas de función están ya definidas con una selección de instrucciones de uso habitual.

Hay muchas otras instrucciones asociadas con la carga y tratamiento de archivos en disco. Aparte de ser capaz de cargar programas en código máquina sencillamente digitando sus nombres, el sistema ofrece una instrucción para cargar archivos específicos que construyen imágenes en el disco empleando el sistema operativo del BBC. PRINT saca un archivo de textos por una impresora, mientras que TYPE lo hace por pantalla. COMMAND utiliza un archivo como una serie de órdenes para el ordenador de modo similar a lo que hace EXEC en el BASIC BBC.

Existen otros varios programas de utilidad que residen en un disco de sistema que acompaña al Disk Pack y se cargan en la máquina tecleando el nombre correspondiente. Éstos incluyen una rutina para cambiar el diseño de los caracteres en la pantalla, una utilidad para escribir música, un *debugger* de código máquina, un editor de disco y una utilidad para permitir que el Torch lea discos Acorn y viceversa. Esta última es muy práctica, ya que los formatos que emplean ambos sistemas son diferentes. Esta utilidad permite, por ejemplo, crear un archivo de texto ejecutando un programa en BASIC BBC y luego editarlo con un programa CP/M para tratamiento de textos.

Si bien el formato del disco Torch es diferente del formato del Acorn, emplea la misma curiosa convención de tratar a las dos unidades de disco como si fueran cuatro unidades distintas.

A pesar de todas estas mejoras, el sistema todavía se puede utilizar como un BBC Micro estándar. Tecleando *BASIC el sistema ignora todos los extras añadidos por el Disk Pack. Extrañamente, el BBC Micro que usted debe tener está preparado para emplear cassettes y no discos. A pesar de que ahora está totalmente equipado para utilizar las unidades de disco del Disk Pack, sin necesidad de recurrir al MCP, el usuario debe especificar que se requiere el sistema de archivo en disco. Desde el BASIC BBC, se puede pasar a emplear de nuevo el Z80 en cualquier momento apagando y encendiendo la máquina, o bien tecleando *MCP.

El BBC Micro es un ordenador personal que dispone de muchas facilidades, pero *sólo* es un ordenador personal. Para utilizar un micro para gestión se requiere muchísimo más que lo que puede ofrecer el BBC solo. Cuando se le añade a un BBC Micro un Torch Disk Pack, se sigue teniendo allí el ordenador personal, listo para emplearlo al momento; pero, gracias a haberse producido esta agregación, el ordenador posee entonces todos los extras necesarios para convertirse en una auténtica máquina de oficina.

Instalando el Disk Pack
Instalar un Torch Disk Pack en un BBC es una tarea bastante complicada. Su ordenador debe ser un BBC Micro Modelo B equipado con una interface para unidad de disco. Dado que ya no se necesita la fuente de alimentación eléctrica del BBC, se deben desenchufar los siete conectores que la unen a la placa principal del ordenador. Éstos se sustituyen entonces por un cable que viene de la fuente de alimentación del Disk Pack. Luego se instala un cable plano del Disk Pack en la interface para unidad de disco del BBC. La ROM que contiene el sistema operativo MCP se enchufa entonces en un conector para ROM libre del BBC

Chris Stevens

Software gratis
El Torch Disk Pack viene con varios paquetes de software, que se ofrecen sin costo adicional alguno. Se proporciona un juego de software de oficina: Perfect Writer (un programa para tratamiento de textos), Perfect Speller (un programa verificador de ortografía), Perfect Filer (un programa de base de datos) y Perfect Calc (un programa de hoja electrónica).
Además de estos paquetes, se suministra una versión de BASIC BBC para ejecutar en el segundo procesador Z80. Las instrucciones son las mismas que en la versión en ROM del BBC con la excepción del ensamblador de 6502 incorporado del BBC. Como el segundo procesador es un Z80 y no un 6502, el BASIC BBC del Z80 tiene incluido un ensamblador Z80. La bonificación son 48 K de memoria libre, independientemente de la modalidad de visualización que se esté utilizando. El BBC Micro tiene apenas 9 K libres en algunos modos de visualización y en ningún caso tiene libres más de 28 K

Chip DFS
A la placa de circuito impreso del BBC se le han de añadir unos cuantos chips para que pueda utilizar una unidad de disco. Estos chips no vienen incluidos en el Torch Disk Pack y, por lo tanto, se han de adquirir por separado. Esta ampliación no está pensada específicamente para el Torch y contiene un chip denominado DFS que en realidad el Torch no utiliza. Sin embargo, vale la pena tenerlo, porque el DFS (*disk filing system*: sistema de archivo en disco) es el sistema operativo que utiliza Acorn. Esto significa que el Torch Disk Pack se puede emplear como un par de unidades de disco para ejecutar software en disco para el BBC Micro en el procesador 6502.
Los formatos de disco de Torch y Acorn no son compatibles, de modo que los programas MCP no pueden leer discos Acorn y viceversa. Con el Torch se suministra un programa que convierte de un formato a otro

Opciones para oficina
El primer ordenador que produjo Torch fue una máquina de oficina. Torch produce, asimismo, versiones de la máquina de oficina que utiliza el sistema operativo Unix. Ésta es la serie 700 (que vemos en la fotografía). La firma tiene también en el mercado la serie 300, terminales que enlazan los ordenadores Torch en una red

Color por números

En este capítulo analizaremos un juego por ordenador aparecido hace algún tiempo, que se basa en otro juego muy conocido: el tradicional "Simón dice"

"Simón dice" es uno de los primeros juegos al que muchos de nosotros recordamos haber jugado. El que lleva el juego da instrucciones como "Simón dice colocar las manos sobre la cabeza", o bien "Simón dice pararse", etc. Los jugadores quedan descalificados si responden a una orden no comenzada con "Simón dice...".

Por sorprendente que parezca, este juego típico de fiestas o de clase se convirtió en un juego electrónico cuando se desarrollaron numerosos juguetes electrónicos que utilizaban microprocesadores para controlar una serie de botones, luces y zumbadores. En la versión electrónica, el ordenador es el líder del juego, y es obligatorio seguir todas sus instrucciones.

El juguete realiza una secuencia de tonos y luces, y el jugador, o los jugadores, deben repetirla pulsando las teclas adecuadas. El juguete añade entonces otra nota a la secuencia y vuelve a comenzar.

Por supuesto, si se posee un ordenador personal, la idea de una máquina dedicada exclusivamente a jugar le parecerá más bien extraña. Un juguete puede ser muy portátil, duradero y fácil de usar. Pero hasta los mejores juegos pronto resultan aburridos ¡y la capacidad del ordenador para ejecutar cientos de programas diferentes asegura que éste jamás dejará de resultar interesante!

El programa que aquí listamos se denomina "¡Imítame!" y desarrolla un juego basado en cuatro luces de colores, cada una con su propio sonido y su propia tecla en el teclado, numeradas del 1 al 4. El programa enciende una luz y luego le pide al jugador que la repita. Si lo hace bien, el programa enciende dos luces, y así sucesivamente.

Hay dos formas en las que se puede perder en este juego: tardando demasiado tiempo en responder con el siguiente botón o pulsando un botón equivocado tres veces en una partida. Para hacer que el juego resulte más difícil, esta versión también va más deprisa a medida que se va avanzando, si bien no es preciso reproducir la secuencia a la misma velocidad que el ordenador. En el juego cada secuencia tiene un máximo de 50 luces; es realmente poco probable que usted llegue a este límite; ¡la mayoría de los jugadores consiguen responder hasta una secuencia de 15 luces antes de que el juego resulte demasiado para ellos!

Es interesante observar la forma en que está estructurado el programa. Todas las instrucciones de color y sonido están agrupadas en subrutinas al final del programa del siguiente modo:

```
1000 Visualizar número de luz a
1500 Obtener una pulsación de tecla 1, 2, 3 o 4
2000 Hacer un ruido para el final del juego
2500 Hacer un ruido para avisar que la respuesta
     ha sido errónea
6000 Visualizar un mensaje en la línea 20 de la
     pantalla y, de ser necesario, hacer una
     pausa después
```

En un juguete exclusivo, estas subrutinas podrían corresponder a dispositivos reales de hardware. Extraerlas del cuerpo principal del programa tiene dos efectos positivos. En primer lugar, todas las complejidades de sonido y color específicas de la máquina se mantienen en un lugar, haciendo que resulte más fácil llevar el programa de una máquina a otra. En segundo lugar, sería fácil utilizar las subrutinas para un juego diferente siguiendo aproximadamente las mismas líneas. El mismo programa podría ofrecer variantes del juego, así como sucede con los juguetes exclusivos. Quizá a usted le gustaría inventarse algunas variantes propias y agregarlas al programa. Por ejemplo, éste se podría adaptar para que se visualicen los marcadores individuales para cualquier número de jugadores.

Como usuario de un ordenador, no olvide que cualquier cosa que pueda hacer un juguete exclusivo, su ordenador lo puede hacer mejor.

Kevin Jones

¡Imítame!

```
10 REM Juego Imitame!
30 LET h = 0: LET n = 0: LET w = 3
40 DIM c(4): DIM p(4): DIM a(50)
50 LET p(1) = 5: LET p(2) = 8: LET p(3) = 12: LET p(4) = 15
60 LET c(1) = 1: LET c(2) = 2: LET c(3) = 4: LET c(4) = 6
70 LET s$ = " ": FOR i = 1 TO 5: LET s$ = s$ + s$; NEXT i
80 LET b$ = CHR$ (143): REM un bloque
100 REM *** pagina de instrucciones
110 CLS: PRINT TAB (10):"Imitame!"
120 PRINT: PRINT
130 IF n > 0 THEN PRINT: PRINT "Lo has conseguido ";n;" veces"
140 IF n > h THEN PRINT: PRINT "... Un nuevo record!!!": LET h = n
150 IF h > 0 THEN PRINT: PRINT "Lo mejor hasta ahora son";h;" veces"
155 PRINT: PRINT "Intenta reproducir la secuencia de luces y sonidos del
    ordenador"
160 PRINT "pulsando las teclas del 1 al 4"
170 PRINT: PRINT "Pulsar P para jugar, S para parar"
180 LET a$ = INKEY$: IF a$ = "" THEN GO TO 180
190 IF a$ = "s" OR a$ = "S" THEN CLS: STOP
200 IF a$ <> "p" AND a$ <> "P" THEN GO TO 180
205 REM *** Nuevo juego
210 CLS: PRINT TAB (10); "Imitame!"
220 FOR a = 1 TO 4: GO SUB 1000: NEXT a
230 LET n = 0: LET m = 0: RANDOMIZE
240 REM *** Siguiente turno
250 LET n = n +1
270 LET a(n) = INT (RND*4) + 1
280 IF m = w THEN GO SUB 2000: LET m$ = "*" + STR$(w) + "respuestas
    incorrectas!": GO SUB 6000: GO TO 100
285 LET m$ = "*Ahi va...": GO SUB 6000
290 FOR i = 1 TO n
300 LET a = a(i): GO SUB 1000
310 FOR j = 1 TO 100/n: NEXT j
320 NEXT i
330 LET m$ = "Imitame...": GO SUB 6000
340 LET i = 1
350 GO SUB 1500
360 IF t = 0 THEN GO SUB 2000: LET m$ = "*Demasiado lento!": GO SUB 6000:
    GO TO 100
370 IF a <> a(i) THEN LET m = m + 1: GO SUB 2500: GO TO 280
380 LET i = i + 1: IF i < = n THEN GO TO 350
390 IF n > 50 THEN LET m$ = "*Tu ganas con 50 veces!": GO SUB 6000: GO TO
    100
400 LET m$ = "*Preparate para volver a probar": GO SUB 6000
410 GO TO 250
1000 REM *** Iluminar recuadro a
1010 INK c(a)
1015 LET p = (a-1)*8 + 2
1020 FOR l = 10 TO 14
1030 PRINT ATl,p;
1040 IF l = 12 THEN PRINT b$;b$;a;b$;b$;
1050 IF l <> 12 THEN PRINT b$;b$;b$;b$;b$;
1060 NEXT l
1070 BEEP 2/(n + 1),p(a)
1080 PRINT AT 10,p;"      "
1090 PRINT AT 11,p;"   ";b$;b$;b$;"   ";
1100 PRINT AT 12,p;"   ";b$;a;b$;"   ";
1110 PRINT AT 13,p;"   ";b$;b$;b$;"   ";
1120 PRINT AT 14,p;"      "
1130 INK 0: RETURN
1500 REM *** Leer una tecla
1510 LET t = 250
1520 LET a$ = INKEY$; IF a$ = "" THEN LET t = t-1: IF t > 0 THEN GO TO
    1520
1530 IF t = 0 THEN RETURN
1540 IF a$ <> "1" AND a$ <> "2" AND a$ <>"3" AND a$ <> "4" THEN GO
    TO 1520
1550 LET a = VAL (a$): GO SUB 1000
1560 RETURN
2000 REM *** Despedida
2010 BEEP 3,0: RETURN
2500 REM Advertencia
2510 BEEP 1,0: RETURN
6000 REM *** Imprimir m$
6010 PRINT AT 20,1;s$;AT 20,1;
6030 IF m$(1) = "*" THEN PRINT m$(2 TO): FOR z = 1 TO 200: NEXT z
6040 IF m$(1) <>"*" THEN PRINT m$
6050 RETURN
```

Complementos al BASIC

Commodore 64

Reemplazar CLS por PRINT CHR$(147).
Reemplazar LET A$ = INKEY$ por GET A$
Reemplazar RANDOMIZE por XX = RND(-TI)
Reemplazar (RND*4) por (RND(1)*4).
Reemplazar M$(1) por LEFT$(M$,1).
Reemplazar M$(2 TO) por MID$(M$,2).
Reemplazar CHR$(143) por CHR$(166).
Reemplazar PRINT AT L,C; por PRINT LEFT$(DN$,L + 1)TAB(C); (p. ej., la línea 6010 se convierte en 6010 PRINT LEFT$(DN$,21)TAB(1);S$;LEFT$(DN$,21)TAB(1);
Reemplazar DIM C(4) por DIM C$(4)
Reemplazar b$;b$;a;b$;b$; por BBZBB$; en la 1040.
Reemplazar b$;a;b$; por BZB$ en la 1100
Insertar:

```
20 VL=54296:AD=54277:SR=AD+1:WF=
   AD-1:NO=17:N1=N0:LF=AD-5:HF=LF+1
25 POKE AD,255:POKE SR,48:POKE VL,15
50 C$(1) = CHR$(31):C$(2) = CHR$(28):C$(3)
   = CHR$(30):C$(4) = CHR$(158)
60 P(1) = 51:P(2) = 34:P(3) = 64:P(4) = 38
90 DN$ = CHR$(17):FOR K = 1 TO 5:DN$ = DN$
   + DN$:NEXT K:DN$ = CHR$(19) + DN$
1010 PRINT C$(A);
1015 P = (A-1)*9 + 3:Z$ = RIGHT$(STR$(A),1)
1130 PRINT CHR$(144):RETURN
2010 SD = 15:SP = 4:N1 = 33:GOSUB
     7000:RETURN
2510 SD = 10:SP = 10:N1 = 33:GOSUB
     7000:RETURN
7000 REM *** BEEP SD,SP
7010 POKE VL,15:POKE WF,N1
7020 POKE LF,SP:POKE HF,SP: FOR DD = 1 TO
     SD*50:NEXT DD
7030 POKE HF,0:POKE LF,0:N1 = N0:RETURN
```

BBC Micro

Reemplazar AT Y,X por TAB(X,Y). Por ejemplo, la línea 6010 se transforma en:

```
6010 PRINT TAB(1,20);S$;TAB(1,20)
```

Reemplazar INKEY$ por INKEY$(0). Insertar:

```
20 MODE 2
25 COLOUR 135:CLS
60 C(1) = 1:C(2) = 2:C(3) = 4:C(4) = 5
80 B$ = CHR$(35)
1010 COLOUR C(A)
1015 P = (A-1)*4
1070 SOUND 1,-10,P(A),40/(N + 1):FOR
     DE = 1 TO 2000/N:NEXT
1130 COLOUR 0:RETURN
2010 SOUND 1,-15,2,40
2510 SOUND 1,-15,40,40
```

Plan de acción

En esta ocasión estudiaremos el diseño de un programa y consideraremos las preguntas que se deben formular antes de escribir un código

El diseño de programas está considerado por quienes están implicados en el mismo (el diseñador-programador y el usuario) como un ejercicio formal y excelente de una aplicación de "resolución de problemas". Lamentablemente, los problemas a resolver siempre se asume que son de tipo técnico, cómo formatear la pantalla, cómo lograr que un bucle sea más rápido, dónde colocar todo en la RAM, etc., mientras que los problemas reales están presentes desde el comienzo del proyecto y por lo general se crean en el primer encuentro del usuario y el "experto". Es raro que los usuarios sean muy conscientes de la verdadera naturaleza de sus problemas (esperan que el experto les diga cuál es el problema y cómo resolverlo) y los expertos muy a menudo piensan que conocen el problema y la solución antes de que el usuario empiece siquiera a enunciarlo. El resultado es una mala comunicación inicial, que lleva a una descripción incompleta del problema y las necesidades del usuario. Trabajar partiendo de estos principios da como resultado la producción de un sistema insatisfactorio que el usuario se verá forzado a aceptar.

En los proyectos de informática personal el programador por lo general es también el diseñador (o "analista de sistemas") y el consumidor. Esto debería significar que los problemas de comunicación se redujeran considerablemente. No obstante, como un usuario-diseñador combinado, usted siempre debería hacer el esfuerzo de explicarse a sí mismo los problemas, las soluciones y las necesidades con la misma claridad que si estuviera hablando con otra persona.

Consideremos el caso de un usuario imaginario y su problema: es un modelista de aviones a escala aficionado que también posee un microordenador basado en cassette. Desea almacenar descripciones bastante detalladas de los materiales que utiliza para cada uno de los modelos que construye de modo que cuando trabaje en los últimos modelos pueda buscar en sus registros los empleos anteriores de esta clase de pegamento o de este tipo de juntura. Por consiguiente, lo que el diseñador debe obtener del usuario es una clara enunciación de lo siguiente:

- La función del programa. Esto puede empezar como una vaga enunciación de intenciones, tal como "Deberá almacenar los registros de mis modelos", pero debe irse afinando mediante la persuasión e interrogación del diseñador en algo más parecido a una especificación de requerimientos, como "Debería almacenar mis descripciones del modelo y su construcción y materiales, entrando toda la información por el teclado, y visualizándola cuando yo entre el nombre del modelo o algún aspecto de su construcción". Esto enuncia las necesidades del usuario con bastante más claridad y señala algunas de las tareas específicas de programación implicadas (almacenamiento, búsqueda, indexación, recuperación, etc).
- Cómo se utilizará el programa. Algunos de los detalles físicos de la utilización típica pueden quedar claros a partir de la descripción de la función, pero puede que no estén completos. Por ejemplo, tal vez el usuario no desee que los detalles del modelo se visualicen en la pantalla porque él trabaja en un cobertizo donde no tiene un aparato de televisión. En este caso, tal vez lo más indicado fuera una "copia en papel" de los detalles seleccionados.
- Qué aspecto tendrá: los formatos de entrada y salida. El programador profesional con frecuencia empleará diagramas preimpresos que representen la pantalla para trazar cada una de las visualizaciones que el usuario verá durante las fases de entrada-salida. Estas técnicas no suelen ser necesarias para el uso personal, si bien los gráficos en alta resolución pueden representar la excepción a esta regla. Los formatos de pantalla son un aspecto muy importante de la *interface para el usuario* (lo que el usuario "ve" del programa) y merecen la misma atención y el mismo análisis que a veces se les dedica a aspectos obviamente más ergonómicos de la informática, como la disposición de teclados y pantallas, la altura de la mesa, los niveles de iluminación, etc.
- Cómo se habrá de organizar: formatos de archivo y programa. Tal vez el usuario piense que necesita almacenar al menos 100 descripciones de aviones, y que cualquier cantidad inferior sería insuficiente. Por el contrario, quizá nunca llegue a construir más de media docena de modelos estándar. Las dimensiones del archivo de datos tienen serias implicaciones respecto a su formato y métodos de acceso. Una exploración secuencial a través de seis descripciones de modelos en cassette que tarde, supongamos, cinco minutos, podría ser bastante aceptable para el usuario, mientras que esperar a que se buscaran 100 ya no tendría sentido. Una solución podría ser colocar el programa y el archivo de índice de descripciones en una cinta, y las descripciones propiamente dichas en otras 20 cintas clasificadas por tipo de avión, por ejemplo.

Las dimensiones del propio programa también pueden constituir un problema: si la sección de entrada de texto exige un editor de textos complejo, si el programa está lleno de menús y muchos mensajes significativos, si las secciones de tratamiento de archivos emplean complicadas rutinas de búsqueda e indexación, entonces el programa tal vez se tendrá que dividir en varios programas separados con el fin de que quepa en la RAM disponible.

- Qué deberá hacer: procedimientos y cálculos especiales. En el ejemplo de los aviones a escala, es poco probable que surjan, pero a menudo sí apare-

cen cuando se consideran otros programas. Puede que existan 20 formas equivalentes y perfectamente buenas de pasar por un proceso determinado, pero cabe en lo posible que el usuario insistiera en una y sólo una de ellas.

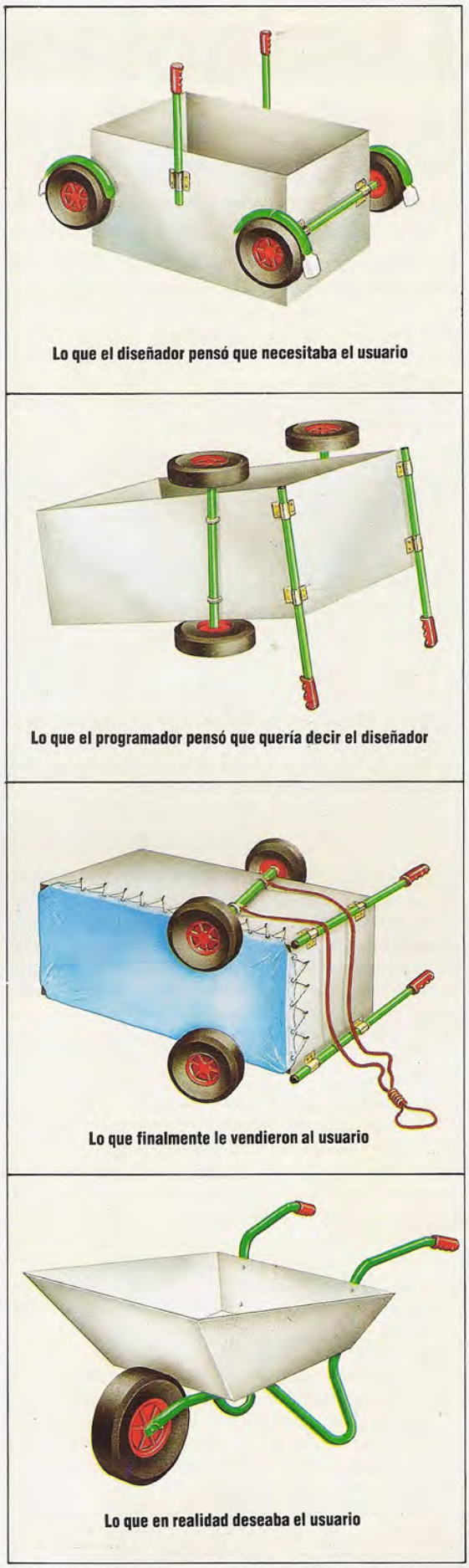
Realizar esto mal hace que el usuario quede de inmediato insatisfecho con el programa. El diseñador se puede sentir tentado a no usar el método preferido por el usuario y sí otros métodos más eficaces, ¡pero cualquier posible ventaja desaparece enseguida cuando el usuario no quiere luego utilizar el programa! Descubrir los procedimientos del usuario puede ser muy útil cuando se trata de diseñar cálculos. ¿Por qué inventar una fórmula para calcular las cargas alares, por ejemplo, si uno simplemente le puede preguntar al que construye el modelo cómo lo hace él?

Teniendo toda esta información apuntada, ya se puede comenzar el trabajo de convertir estas especificaciones en un programa. Un enfoque útil es el de diseñar primero el diálogo usuario-programa, luego los archivos de datos y luego los procesos que controlan todo. La palabra "diálogo" se toma en el sentido de la comunicación bidireccional de información que se produce entre el usuario y el programa. Ésta no consiste sencillamente en la entrada de los detalles de los modelos de avión y su subsiguiente visualización, sino que también incluye todas las preguntas, mensajes y menús que el programa produce y todas las entradas, órdenes o selecciones que entra el usuario. Asimismo, es importante determinar el tipo de diálogo en esta etapa. Para el programa de los aviones podría ser adecuada una elección entre menú y acciones mediante órdenes. Las decisiones tomadas aquí tendrán un considerable efecto en la estructura del programa general. El contenido y el formato del diálogo se debe considerar en detalle, pero la recompensa a este esfuerzo es que todos los datos manipulados por el programa son especificados en esta fase. Ello significa que se puede calcular el tamaño de memoria requerido para mensajes de error y textos de preguntas, y que los archivos se pueden diseñar ahora.

Para el programa de los aviones a escala, donde los archivos contendrán grandes bloques de texto y serán muy extensos, la mejor solución sería la de dividir el archivo en varias cintas de modo que resulte más fácil buscar en cada una de ellas. Si valiera la pena (depende del volumen), los datos se podrían comprimir mediante un algoritmo de codificación antes de escribirlos en cinta, y después decodificarlos durante la lectura.

Ya en esta fase, las funciones necesarias serán evidentes. Habrá rutinas que permitan añadir y editar textos de datos, para archivar los textos recién introducidos (éstas actualizarán todos los índices utilizados por el sistema), para aceptar nombres de componentes, para buscar y visualizar descripciones, etc. Todo ello se le debe presentar al usuario en forma de opciones, y el programa debe ser capaz de detectar y tratar datos que no sean válidos.

En esta etapa es aconsejable que el usuario verifique concienzudamente el diseño para asegurar que hace lo que debería hacer. Si todo está bien, entonces se puede codificar el programa. Por supuesto, es más fácil decirlo que realizarlo, y el acto de convertir el diseño en un programa que funcione eficazmente muy bien podría poner en evidencia otros problemas.



Lo que el diseñador pensó que necesitaba el usuario

Lo que el programador pensó que quería decir el diseñador

Lo que finalmente le vendieron al usuario

Lo que en realidad deseaba el usuario

Steve Cross

Describir, definir, diseñar
Si el equipo de desarrollo de software típico (o sea, usuario, diseñador y programador) se hubiera dedicado a resolver el problema de desplazar cargas pesadas por los jardines, esto es lo que habría hecho. La mala comunicación (entre experto y no experto, y también entre expertos) sigue siendo un importante problema con que se enfrentan todos los equipos de diseño

Los fantasmas del ordenador

"Atic Atac" une a los elementos estratégicos propios de los juegos de aventuras la rápida acción de los juegos recreativos

Atic Atac es uno de los juegos de esa rara casta que consiguen combinar la emoción de la acción de los recreativos con la complejidad de los juegos de aventuras. Respondiendo al estilo clásico de aventuras, el juego se desarrolla en las numerosas dependencias de un castillo habitado por fantasmas. El jugador está atrapado dentro del castillo y sólo puede escapar encontrando la llave de oro que abre las puertas principales. Su vida se ve amenazada por una serie de criaturas perversas: arañas, profanadores de tumbas, brujas, demonios y monstruos hambrientos, todos ellos están allí, por no hablar de Drácula, el monstruo de Frankenstein, la Momia y gran cantidad de murciélagos. Es algo así como hallarse de pronto protagonizando una superpoblada película del más puro horror. Las puertas falsas y los pasajes ocultos abundan y hay objetos útiles o valiosos que el jugador puede recoger.

Hasta ahora todo suena como un típico juego de aventuras, pero está ilustrado con espléndidos gráficos animados; y aquí es donde comienza el aspecto recreativo de *Atic Atac*. Al jugador se le presenta una panorámica en color y tridimensional de cada cuarto o calabozo, donde pueden verse las diversas puertas que dan al norte, al sur, al este o al oeste. También hay adornos que incluyen armaduras, estanterías con libros, relojes del abuelo y cuadros. Vestido con el disfraz elegido de Caballero, Mago o Siervo, el jugador se desplaza por las habitaciones protegido con el arma apropiada para su personaje. Los numerosos monstruos no cesan de aparecer, envueltos en nubes de humo, y su solo contacto hace que el protagonista se debilite muchísimo. Afortunadamente, con el uso de las armas, hacha, espada o conjuro, puede repelerlos.

El movimiento se controla mediante una palanca de mando Kempston o de cursor, o con el teclado. Lamentablemente, al igual que muchos juegos, *Atic Atac* utiliza para el movimiento las teclas Q, W, E y R, y como están dispuestas sobre el teclado en línea recta, el juego resulta difícil de jugar.

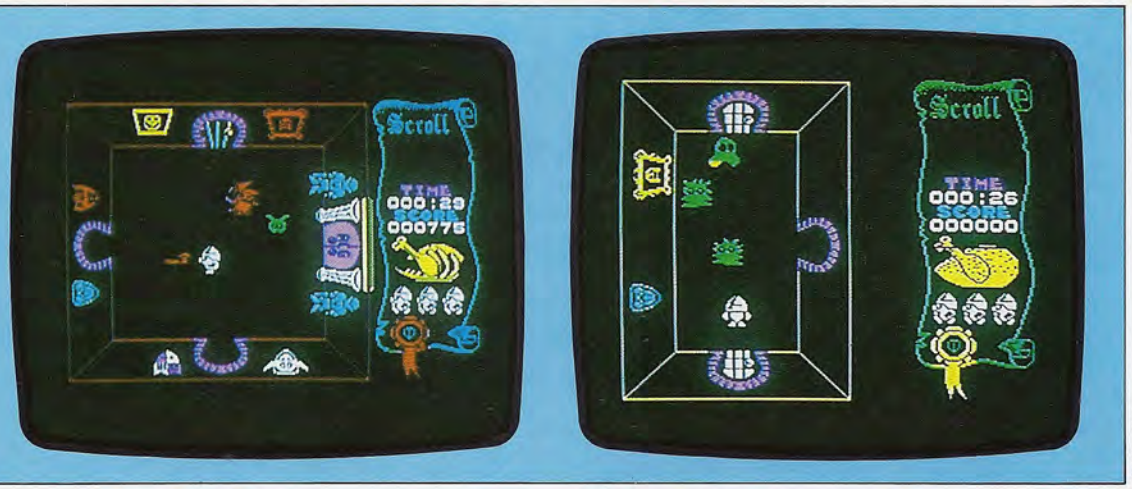
El juego se podría convertir fácilmente en un simple juego vertiginoso si no fuera por el hecho de que algunos de los objetos, por no hablar de la propia llave de oro, son muy útiles. Algunas puertas de colores sólo se abrirán si se posee la llave del mismo color y ciertos objetos protegen contra determinadas criaturas. En un nivel más básico se necesita hallar comida, como indica tan a las claras el pollo que se desintegra rápidamente a la derecha de la pantalla. Si no se lo come, poco a poco la saludable ave se transforma en un montoncito de huesos, lo que significa la muerte del usuario por inanición.

Al principio, *Atic Atac* se tiende a considerar como un simple juego recreativo muy frustrante. Una vez que comience a dominar las técnicas para enfrentarse a los numerosos monstruos, empezará a apreciar el aspecto de aventura del mismo, mientras busca por el castillo los diversos artefactos y tesoros. Pero se lo advertimos: éste es un juego que exige muchas horas de esfuerzo para hallar las llaves con las que poder abrir las puertas.

Atic Atac: Para el Spectrum de 48 K
Editado por: Ashby Computers and Graphics Ltd., Ashby de la Zouch, Leicestershire LE6 5JU
Autores: Ultimate, Play The Game
Palancas de mando: Kempston, y de cursor
Formato: Cassette

Habitantes del castillo

Estas fotografías de pantalla muestran dos etapas del *Atic Atac*, un juego de aventuras que se desarrolla al frenético ritmo de un juego recreativo. Una sucesión de monstruos y otras sorpresas se le aparecen al jugador mientras recorre corredores y pasadizos secretos en busca de una llave de oro



Liz Heaney

Siluetas rutinarias

En este capítulo examinamos una rutina para crear y mover un dibujo empleando el BBC Micro

Muchas son las formas y figuras que pueden adoptar los sprites, pero todas tienen en común el reticulado o cuadrícula. No existe ninguna restricción al elegir el tamaño de la cuadrícula, pero resulta práctico que se componga de un número entero de bytes (ocho, dieciséis, veinticuatro bits, etc.) formando un rectángulo o cuadrado. La rutina que vamos a proporcionarle y comentar se vale de una cuadrícula de 24 pixels de largo por 21 de ancho, que necesita, por tanto, 63 bytes de memoria para representar el sprite. Si trazamos el dibujo que deseamos sobre este rectángulo cuadriculado, sólo queda traducirlo a binario para definir un sprite. Todo pixel "encendido" corresponde a un 1 y los "apagados" equivalen a 0. Una vez obtenidos los valores binarios correspondientes, debemos convertirlos a decimal, o a hexa, y a continuación colocarlos en alguna parte de la memoria. La ilustración que el lector puede ver en la parte inferior de esta página muestra el dibujo del sprite que hemos escogido.

Los 63 números que componen el sprite pueden definirse usando la sentencia DATA y ser leídos (READ) para su tratamiento en BASIC. La instrucción READ se encarga de colocarlos en algún área de la memoria. Esta área puede estar en cualquier zona de la RAM con tal de que no se escriba nada encima durante la ejecución del programa. El sitio más obvio para almacenarlos sería la parte superior de la memoria destinada al programa en BASIC. La dirección superior de esta área está contenida en HIMEM (HI*gh*: alta). Si queremos que ningún otro dato venga a superponerse en ella, bajaremos un poco HIMEM. Esto es lo que hacen las líneas 220 y 230 del programa, al tiempo que fijan la dirección del primero de los bytes de DATA, que llamamos SPRDAT (datos del sprite). Por último, las líneas 1740 a 1770 "leen" (READ) los datos y los colocan en los 63 bytes consecutivos que empiezan en la posición SPRDAT.

La rutina en el código máquina

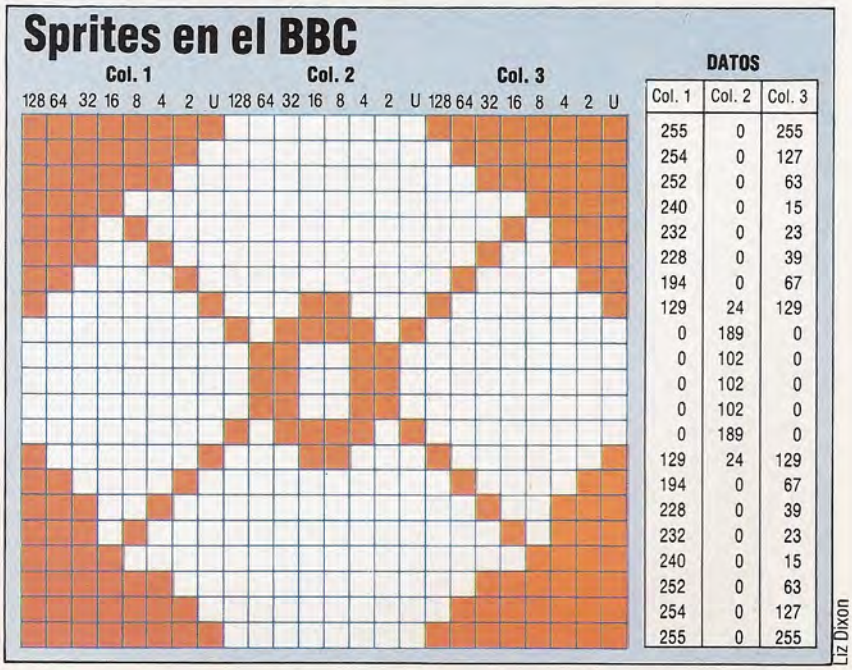
Lo más importante que encomendaremos al código máquina es el análisis del dibujo que define el sprite escogido y la ejecución de las operaciones necesarias para convertir los datos en su correspondiente visualización sobre la pantalla. Esto se consigue examinando cada bit de los referidos 63 bytes, uno a uno, para trazar un punto si el bit está a 1, o bien dejar un espacio si está a 0. La manera más fácil de analizar cada bit de un byte determinado es quizá empleando una de las instrucciones de rotación. La línea 1100 se sirve de ROL (RO*tate Left*: rotar a la izquierda) referida a un byte determinado. Ya sabemos que esta instrucción desplaza cada bit del byte un lugar a la izquierda. Como el valor previo del flag de arrastre se coloca en el extremo derecho del byte, de la misma manera el bit que "sobra" por el otro extremo izquierdo se coloca en el flag. Con lo cual es posible examinar cada bit a su paso por el flag o indicador. Siempre que se tenga la precaución de no alterar el valor del flag durante el proceso, el byte original se recupera intacto después de nueve rotaciones.

Veamos ahora más gráficamente lo que acabamos de exponer:

Byte y flag de partida	C	1 1 0 1 1 1 0 0
Después del 1.er ROL	1	1 0 1 1 1 0 0 C
Después del 2.º ROL	1	0 1 1 1 0 0 C 1
Después del 3.er ROL	0	1 1 1 0 0 C 1 1
Después del 4.º ROL	1	1 1 0 0 C 1 1 0
Después del 5.º ROL	1	1 0 0 C 1 1 0 1
Después del 6.º ROL	1	0 0 C 1 1 0 1 1
Después del 7.º ROL	0	0 C 1 1 0 1 1 1
Después del 8.º ROL	0	C 1 1 0 1 1 1 0
Después del 9.º ROL	C	1 1 0 1 1 1 0 0

Del ejemplo expuesto arriba ya habrá deducido usted que el contenido inicial o final del flag nos trae completamente sin cuidado (por eso lo hemos señalado con una C en ambos casos). En la línea

Dimensionar el sprite
El sprite mide 24 × 21 pixels, por lo que se representa en 63 bytes, ya que a cada pixel corresponde un bit en el método acostumbrado de alta resolución. La variable *logcol* se encarga del color del sprite entero, y el tamaño del sprite viene dado por dos factores, XSCALE e YSCALE. Todos los números deben ser pares



1100 del programa se emplea ROL en el modo de direccionamiento indexado absoluto, lo que permite el acceso a los 63 bytes y su análisis como en el byte del ejemplo.

Ya tenemos cada bit aislado y examinado para saber si corresponde a punto o espacio: ahora falta su trazado sobre la pantalla. El BBC Micro puede realizar esto de dos maneras. La primera consiste en colocar (POKE) los valores directamente en el área para visualización de la memoria. Lo cual no es tan fácil como parece en este ordenador, donde se nos presentan dos problemas principales. Uno de ellos surge de las diferencias que existen entre las áreas de memoria reservadas a la pantalla en el modelo A y en el B y también entre distintos modos. Otro es que la relación matemática entre los pixels y los bits de la memoria de pantalla es de una considerable complejidad. Por ejemplo, en el modo 2 cada byte de la memoria controla tan sólo dos pixels de la pantalla, pues este modo dispone de 16 colores y cada pixel necesita cuatro bits para la definición de su color. Ahora bien, en el modo 0, que es de dos colores, cada pixel no necesita más que un bit para su definición. Desde luego, es factible escribir una rutina en lenguaje máquina para tratar nuestro sprite en el modo 2, pero sólo funcionará en éste y en ningún modo más.

Por suerte, existe una segunda manera de trazar pixels en la pantalla. Cuando nos servimos de los gráficos en BASIC, el sistema operativo del BBC realiza el tipo de operaciones que nosotros estamos buscando. Pues bien, es posible el acceso a esta rutina del sistema operativo. Es más, el fragmento de programa que escribamos para llamar a esta rutina funciona en todos los modos. Es parecido a la instrucción VDU en el BASIC del BBC. Así, en el caso de que nos propongamos trazar un punto en la pantalla, se puede escribir:

```
VDU25,68,300;700;
```

Los últimos números, 300 y 700, son las coordenadas *x* e *y* respectivamente. Una instrucción gemela a ésta en código máquina es OSWRCH, con la que realizamos una llamada al sistema operativo. Tal llamada se repite colocando cada vez un número en el acumulador. Dado que el acumulador tiene cabida para un solo byte, las coordenadas deben ser partidas en un byte *lo* y un byte *hi*, así:

```
VDU25,68,44,1,188,2
```

Para realizar esto en lenguaje máquina, OSWRCH debe ser llamada seis veces. El vector de la dirección inicial de OSWRCH está en la posición &FFEE, y se accede a la rutina por medio de JSR &FFEE. Cualquier orden VDU se puede ejecutar de esta manera, y esta rutina que estudiamos utiliza llamadas a OSWRCH en varias ocasiones. Observe que, si bien OSWRCH no afecta a los valores de los registros X, Y y A, produce una alteración en el contenido del flag de arrastre. Por lo tanto, en el caso de que el valor del flag deba ser conservado, será necesario almacenar éste en algún sitio antes de llamar a OSWRCH. Es lo que sucede con la rutina ROL. La manera más fácil de hacerlo será trasladando el registro indicador de estado a la pila mediante PHP (véase p. 737) antes de la llamada, para recobrarlo, una vez acabada OSWRCH, mediante PLP.

Veamos ahora la estructuración de la rutina en código máquina. El análisis de los datos del sprite y trazado en pantalla se realiza con la subrutina SPRPLT (PL*o*T:trazar) que comienza en la línea 890. Lo primero que hace la rutina es plantearse el método de trazado como una operación OR exclusivo (operación XOR). En BASIC la orden similar es GCOL para el BBC. Para borrar los puntos así trazados basta con volver a escribir sobre ellos los mismos puntos. Cualquier dato bajo el sprite quedará intacto. Al comienzo del fragmento en código máquina se realiza un movimiento absoluto para posicionar la esquina superior izquierda del sprite. Cada fila es analizada ahora tomando tres bytes de los datos del sprite y haciéndolos rotar según explicamos anteriormente.

Un trazado o un movimiento relativo se realiza escogiendo una distancia que proporciona el factor horizontal de escala, XSCALE, dependiendo del valor del bit en curso contenido en el flag de arrastre, perteneciente a uno de los bytes del sprite. Al final de cada fila de tres bytes se vuelve a realizar un movimiento absoluto hacia el mismo punto de la abscisa *x* correspondiente a la esquina extrema izquierda del sprite y hacia otro punto de la ordenada *y* que determina el factor de escala vertical YSCALE. Esto se repite hasta concluir el análisis de los 63 bytes.

En dos ocasiones se usa la subrutina SPRPLT. En la primera el objetivo es borrar el sprite previo mediante un nuevo trazado sobre el lugar que ocupaba. En la segunda, se emplea para trazar el nuevo sprite. Una vez que éste ha sido trazado, sus coordenadas se transfieren a OLDX y OLDY, los cuales quedan de esta forma preparados para un uso ulterior de la rutina.

Empleo de la rutina en código máquina desde el BASIC

Los programadores en BASIC pueden usar fácilmente esta rutina, sin que necesariamente tengan que entenderla. Pasos que deben seguir:

1) Diseñar el sprite y colocar los datos en el área de memoria como se muestra en el programa;
2) Establecer el modo de visualización que se desea usar;
3) Dar los valores de XSCALE e YSCALE conforme se indica en la línea 1870;
4) Establecer el color lógico del sprite según la línea 1890;
5) Establecer las coordenadas de la posición en que se desea que aparezca el sprite y emplear el procedimiento indicado en las líneas 2010 a 2060 para convertir las coordenadas absolutas en la forma "byte *lo*, byte *hi*";
6) CALL SPRITE (llamar al sprite).

El listado en BASIC puede, como aquí, incluir esta rutina en lenguaje máquina. El listado en assembler se puede omitir cambiando la línea 260 así:

```
FOR opt% = 0TO2STEP2
```

O, si prefiere conservar la rutina una vez ensamblada (o sea, tras ser ejecutada), utilizará ■ *SAVE, tomando buena nota de las direcciones de inicio y final del código al visualizar el listado en assembly.

Un sprite BBC

```
10  REM **** SPRITES BBC ****
20  :
30  REM ** PREPARACION VARIABLES P. 0 **
40  TYPE      =&70
50  OLDXLO    =&71:?OLDXLO=0
60  OLDXHI    =&72:?OLDXHI=0
70  OLDYLO    =&73:?OLDYLO=0
80  OLDYHI    =&74:?OLDYHI=0
90  NEWXLO    =&75
100 NEWXHI    =&76
110 NEWYLO    =&77
120 NEWYHI    =&78
130 XSCALE    =&79
140 YSCALE    =&7A
150 logcol    =&7B
160 ROW       =&7C
170 YTMPLO    =&7D
180 YTMPHI    =&7E
190 XTMPLO    =&7F
200 XTMPHI    =&80
210 :
220 HIMEM=HIMEM-150
230 SPRDAT    =HIMEM+
240 OSWRCH    =&FFEE
250 DIM MC% &01FF
260 FOR opt%=0TO 3 STEP 3
270   P%=MC%
280   [
290   OPT opt%
300   \**** MUEVE A LOS ANTIGUOS X,Y ****
310   \
320   .SPRITE     LDA   #25
330               JSR   OSWRCH
340               LDA   #68
350               JSR   OSWRCH
360               LDA   OLDXLO
370               STA   XTMPLO
380               JSR   OSWRCH
390               LDA   OLDXHI
400               STA   XTMPHI
410               JSR   OSWRCH
420               LDA   OLDYLO
430               STA   YTMPLO
440               JSR   OSWRCH
450               LDA   OLDYHI
460               STA   YTMPHI
470               JSR   OSWRCH
480   \
490   \**** BORRADO ANTIGUO SPRITE ****
500   \
510               JSR SPRPLOT
520   \
530   \****MUEVE A LOS NUEVOS X,Y ****
540   .NEWMOV     LDA   #25
550               JSR   OSWRCH
560               LDA   #68
570               JSR   OSWRCH
580               LDA   NEWXLO
590               STA   XTMPLO
600               JSR   OSWRCH
610               LDA   NEWXHI
620               STA   XTMPHI
630               JSR   OSWRCH
640               LDA   NEWYLO
650               STA   YTMPLO
660               JSR   OSWRCH
670               LDA   NEWYHI
680               STA   YTMPHI
690               JSR   OSWRCH
700   \
710   \**** TRAZADO NUEVO SPRITE ****
720   \
730               JSR SPRPLOT
740   \
750   \**** TRASLADO DE LOS NUEVOS X,Y A LOS ANTIGUOS X,Y ****
760               LDA   NEWXLO
770               STA   OLDXLO
780               LDA   NEWXHI
790               STA   OLDXHI
800               LDA   NEWYLO
810               STA   OLDYLO
820               LDA   NEWYHI
830               STA   OLDYHI
840   \
850   \**** VUELTA AL BASIC ****
860   \
870               RTS
880   \
890   \**** SUBRUTINA TRAZADO SPRITE ****
900   \
910   \** ESTABLECER TRAZADO CON OR EXCLUSIVO **
920   .SPRPLOT    LDA   #18
930               JSR   OSWRCH
940               LDA   #3
950               JSR   OSWRCH
960               LDA   logcol
970               JSR   OSWRCH
980   \
990   \
1000  \** INICIALIZACION CONTADORES **
1010  \** X CUENTA BYTES, Y CUENTA BITS **
1020  \** ROW CUENTA FILAS DE 3 BYTES **
1030              LDX   #&00
1040  .NEWROW     LDA   #&00
1050              STA   ROW
1060
1070  .BYTE       LDY   #&09
1080  .BYT        LDA   #65
1090              STA   TYPE
1100              ROL   SPRDAT,X
1110              PHP       \STORE CARRY ON STACK
1120              BCS   DOPLOT
1130              LDA   #64
1140              STA   TYPE
1150  \** INSTRUCCION DE TRAZADO VDU **
1160  .DOPLOT     LDA   #25
1170              JSR   OSWRCH
1180              LDA   TYPE
1190              JSR   OSWRCH
1200              LDA   XSCALE
1210              JSR   OSWRCH
1220              LDA   #&00
1230              JSR   OSWRCH
1240              JSR   OSWRCH
1250              JSR   OSWRCH
1260  \** FIN DE INSTRUCCION TRAZADO VDU **
1270              PLP       \RETRIEVE CARRY
1280              DEY
1290              BNE BIT
1300  \** SI ACABO EL BYTE **
1310              INX
1320              CPX   #63
1330              BEQ   FINIS
1340  \** VER SI FINAL DE FILA **
1350              INC   ROW
1360              LDA   ROW
1370              CMP   #3
1380              BNE   BYTE
1390  \** SI FINAL FILA RESTAR YSCALE DE Y **
1400  \
1410              LDA   YTMPLO
1420              SEC
1430              SBC   YSCALE
1440              STA   YTMPLO
1450              BCS   NOSUB
1460              DEC   YTMPHI
1470  \** MOV. ABSOLUTO PARA COMIENZO FILA SIGU. **
1480  \
1490  .NOSUB      LDA   #25
1500              JSR   OSWRCH
1510              LDA   #68
1520              JSR   OSWRCH
1530              LDA   XTMPLO
1540              JSR   OSWRCH
1550              LDA   XTMPHI
1560              JSR   OSWRCH
1570              LDA   YTMPLO
1580              JSR   OSWRCH
1590              LDA   YTMPHI
1600              JSR   OSWRCH
1610  \
1620  \** FILA SIGUIENTE **
1630              JMP   NEWROW
1640  \
1650  \** FIN DE SUBRUTINA **
1660  .FINIS RTS
1670  \
1680  ]
1690 NEXT
1700 :
1710 REM **** AQUI COMIENZA PROGRAMA EN BASIC ****
1720 :
1730 REM ** LEER DATOS DEL SPRITE **
1740 FOR address = SPRDAT TO SPRDAT+62
1750   READ data:?address = data
1760 NEXT address
1770 :
1780 REM **** ESTABL. PARAMETROS EN LENG. MAQU. ****
1790 :
1800 MODE1
1810 GCOL0,129
1820 CLG
1860 ?XSCALE = 4:?YSCALE = 4
1870 ?logcal = 1
1880 :
1890 X = 700:Y = 800
1900 PROCCOORDS (X,Y)
1910 CALL SPRITE
1920 :
1930 REM **** ESPERA DE TECLAS CURSOR ****
1940 FOR S = 0 TO 1 STEP 0
1950 PROCKEYS
1960 PROCCOORDS (X,Y)
1970 CALL SPRITE
1980 NEXT S
1990 END
2000 :
2010 DEF PROCCOORDS (X,Y)
2020 XH = X DIV 256:XL = X MOD 256
2030 YH = Y DIV 256:YL = Y MOD 256
2040 ?NEWXLO = XL:?NEWXHI = XH
2050 ?NEWYLO = YL:?NEWYHI = YH
2060 ENDPROC
2070 REM **** EXPLORACION DE TECLADO ****
2080 DEF PROCKEYS
2090 LOCAL LT,ZZ:LT = 2
2100 FOR ZZ = 0 TO 1 STEP 0
2110 IF INKEY(-58) THEN Y = Y+50:ZZ = LT
2120 IF INKEY(-42) THEN Y = Y-50:ZZ = LT
2130 IF INKEY(-26) THEN X = X-50:ZZ = LT
2140 IF INKEY (-122) THEN X = X+50:ZZ = LT
2150 NEXT ZZ
2160 ENDPROC
2170 REM **** DATOS DEL SPRITE ****
2180 DATA 255,0,255,254,0,127,252,0,63
2190 DATA 240,0,15,232,0,23,228,0,39
2200 DATA 194,0,67,129,24,129,0,189,0
2210 DATA 0,102,0
2220 DATA 0,102,0
2230 DATA 0,102,0
2240 DATA 0,189,0,129,24,129,194,0,67
2250 DATA 228,0,39,232,0,23,240,0,15
2260 DATA 252,0,63,254,0,127,255,0,255
```

Sprites o burbujas

El sprite que definen los 63 bytes de DATA se mueve por toda la pantalla impulsado por las teclas con flecha. Su relativa lentitud es consecuencia del empleo de la rutina en ROM llamada OSWRCH, pero como contrapartida funciona en todos los modos. Al ejecutar el programa (RUN) el listado en assembly desplaza primero la pantalla y después realiza la visualización del gráfico

LLAMA SOFT

Llamasoft es una empresa basada en el talento de Jeff Minter, considerado una estrella en el mundo del software gracias a sus juegos rebosantes de humor e ironía

La mayoría de las casas de software empiezan como "industrias caseras", con uno o dos programadores trabajando en su casa. A medida que van obteniendo éxito, se contratan otros programadores y los diversos factores que la llevaron por el camino del éxito quedan olvidados en la carrera por producir software nuevo: el "estilo de la firma" desaparece y la producción de una empresa se vuelve muy parecida a la de cualquier otra. Pero Llamasoft es diferente: ninguna otra empresa podría haber producido *Revenge of the mutant camels* (La venganza de los camellos mutantes) y *Sheeps in space* (Ovejas en el espacio).

El estilo distintivo de Llamasoft se debe atribuir a un hombre: Jeff Minter. Sólo tiene 22 años de edad pero, en el transcurso de estos últimos años, se ha convertido en una de las estrellas del mundo del software. Es su obsesión por los camellos, las llamas y otros animales peludos la que ha conformado la imagen de Llamasoft y su adjetivo favorito ("¡imponente!") se utiliza habitualmente en la publicidad de Llamasoft. La fama de Minter (o su notoriedad) es tal que ha sido satirizado en uno de los programas de una empresa rival: uno de los peligros del *Manic miner*, de Software Projects, se describe como "El ataque de los teléfonos mutantes".

Minter empezó a escribir juegos en 1981, después de abandonar la Universidad de East Anglia, donde estudiaba matemáticas y física. A pesar de ser el primero de su clase en la parte de su curso relacionada con la informática, no aprobó los exámenes de matemáticas y tuvo que abandonar la carrera.

Su primera incursión en el mercado del software comercial se produjo cuando escribió una versión del *Defender* para el Vic-20 y fundó Llamasoft, en 1982, para comercializar el producto. Pronto vinieron *Traxx* y *Gridrunner*, que le aseguraron el éxito. *Gridrunner*, en especial, se vendió bien, tanto en Gran Bretaña como en Estados Unidos.

En la actualidad Minter escribe juegos para el Vic-20, el Commodore 64 y la gama Atari, pero no para el Spectrum. En consecuencia, de las adaptaciones de los juegos Llamasoft se han hecho cargo Salamander Software y Quicksilva, que producen bajo licencia las versiones para el Spectrum. Llamasoft está por contratar a un ayudante para adaptar juegos, lo que le permitirá a Minter concentrarse en escribir software nuevo, pero seguirá a cargo de la programación. Según explica, "a mí lo único que me interesa es conseguir un buen juego; todos se basan en ideas mías, así que no hay ningún motivo para tomar a otras personas".

La empresa es, en gran parte, un negocio familiar. Llamasoft se convirtió en una sociedad anónima en abril de 1984 y sus directores son el propio Minter, su padre Patrick y su madre Hazel. La empresa se lleva desde la casa familiar de Tadley (Hampshire). Patrick Minter ayuda a su hijo con la programación, mientras Hazel se ocupa de la administración. Llamasoft también tiene empleados a dos ayudantes, dos contables y un director artístico.

Minter lleva producidos 21 juegos para Llamasoft; muchos de éstos contienen un tema recurrente, por lo general con connotaciones de camellos. Minter afirma que él está "totalmente fascinado por los camellos" y el primer juego que incorporó estos animales lo escribió en 1982 para Atari. Se trataba de *Attack of the mutant camels* (El ataque de los camellos mutantes) y fue seguido posteriormente por *Revenge of the mutant camels*.

Llamasoft acaba de firmar un contrato con la CBS para comercializar los juegos de Minter en Gran Bretaña (con la excepción de las adaptaciones de Salamander y Quicksilva). La distribución en Estados Unidos la realiza Human Engineering Software. La demanda de los juegos de Llamasoft es muy grande: un producto nuevo tiene inicialmente una tirada de 10 000 cassettes, y esta tirada se repite luego tantas veces como haga falta, según las ventas registradas.

Minter no tiene previsto ningún cambio respecto a la forma en que funciona la empresa. Afirma que el aspecto comercial no le interesa y planea seguir desarrollando su obsesión por los animales peludos y escribiendo nuevos juegos "imponentes".

La venganza de los camellos mutantes
El surrealista y obsesivo estilo de Llamasoft queda patente en esta fotografía de *La venganza de los camellos mutantes*, continuación del exitoso *Ataque de los camellos mutantes*

Cortesía de "Your 64 and Vic-20"

Escribir para la pantalla

Las empresas de software profesionales utilizan un hardware muy complejo e invierten enormes recursos en la creación de sus productos

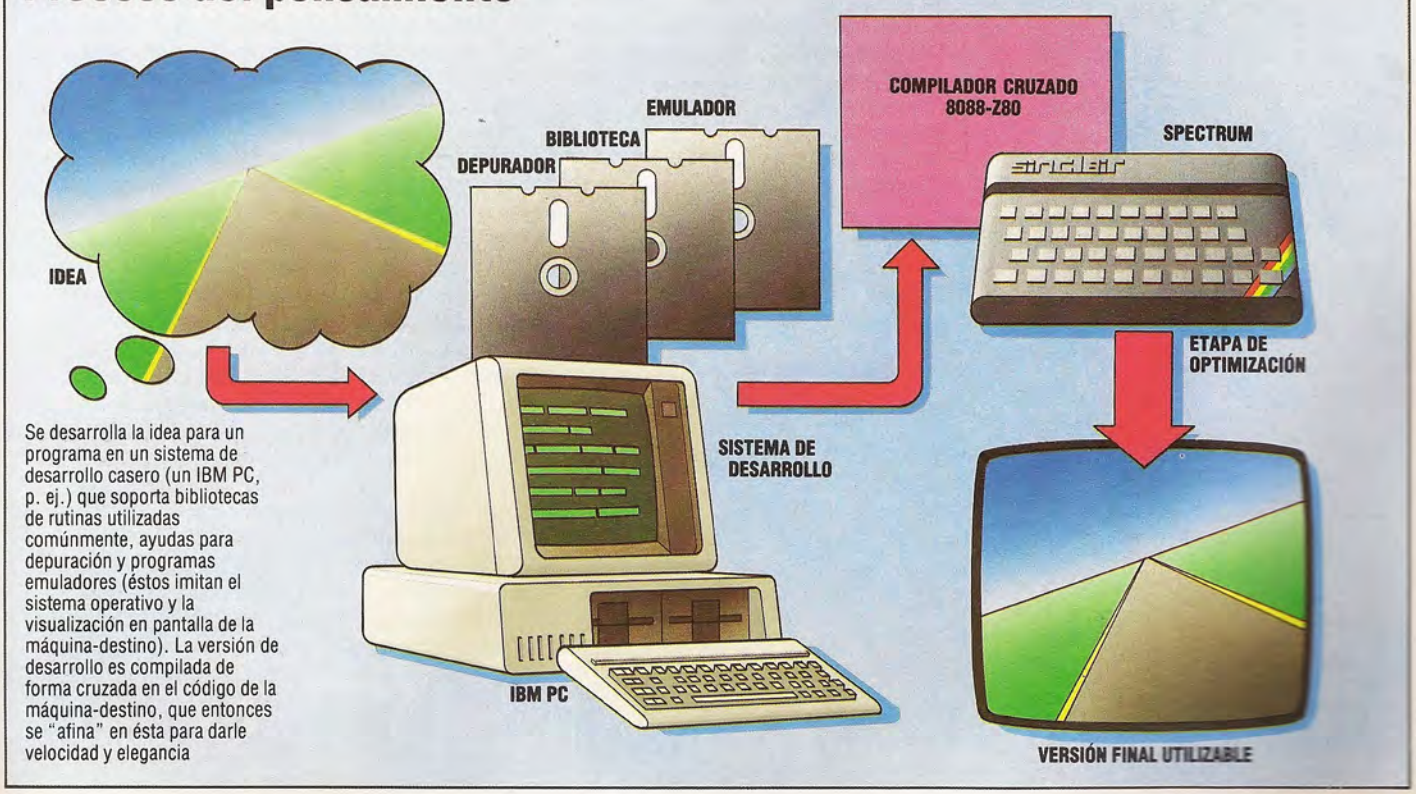
Un equipo caro no significa necesariamente programas de éxito; algunos escritores aficionados han conseguido hacerse una pequeña fortuna con software que han producido en su casa, en un Spectrum. Como quiera que sea, los niños prodigio que programan en sus hogares se están convirtiendo en una especie en vías de extinción, debido en especial al desarrollo de las grandes firmas de software durante estos últimos años.

Uno de los atributos más importantes del software serio para máquinas personales es la velocidad de operación; y ello significa que es necesario escribir los programas (al menos en parte) en código máquina. Pero trabajar con código máquina es sumamente difícil; en particular, los programadores de este lenguaje necesitan piezas de software adicionales que los ayuden a escribir sus programas. Como mínimo se requerirá un programa ensamblador para traducir el código fuente del programador al código objeto que la máquina entiende, y esta parte puede ser bastante desalentadora cuando se trata de un programa largo. Muchos escritores de software trabajan de esta manera.

Básicamente, la calidad de los programas ensambladores existentes para máquinas personales es baja. Hasta el más sencillo de estos paquetes emplea considerables cantidades de memoria y, por consiguiente, limita las dimensiones de los programas que se pueden escribir con el mismo. Asimismo, muchas máquinas personales son muy incómodas de utilizar durante períodos prolongados: los teclados pobres, las visualizaciones imperfectas y, en algunos casos, la falta de unidades de disco, hace que emplear un equipo de estas características sea una ardua tarea.

Por estas razones, la mayoría de las empresas profesionales no utilizan el micro para el cual está destinado el programa (llamado *máquina-destino*), sino que emplean ordenadores de oficina con software especial (denominados *sistemas de desarro-*

Proceso del pensamiento



Se desarrolla la idea para un programa en un sistema de desarrollo casero (un IBM PC, p. ej.) que soporta bibliotecas de rutinas utilizadas comúnmente, ayudas para depuración y programas emuladores (éstos imitan el sistema operativo y la visualización en pantalla de la máquina-destino). La versión de desarrollo es compilada de forma cruzada en el código de la máquina-destino, que entonces se "afina" en ésta para darle velocidad y elegancia

Kevin Jones

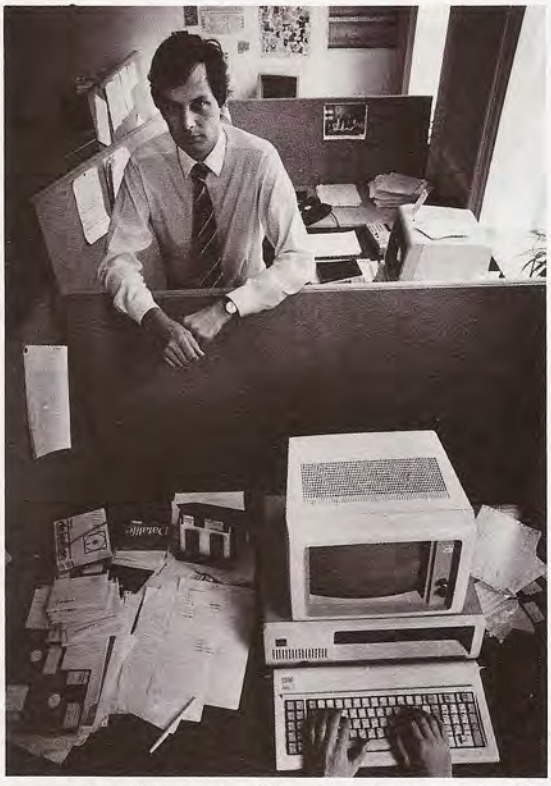
Intelligent Software
Especializada en juegos de estrategia como el ajedrez, IS utiliza máquinas IBM y Apple con sus propias interfaces especialmente creadas para desarrollar el software. A IS, la división de programas en segmentos dependientes de la máquina y universales hace que le resulte más sencillo soportar una gama de ordenadores y máquinas exclusivas para jugar al ajedrez

llo). Los programadores que hacen uso de estas máquinas escriben con frecuencia en lenguajes tales como el PASCAL y el C. Utilizan versiones de estos lenguajes conocidas como compiladores cruzados o ensambladores cruzados, que permiten realizar el trabajo en un micro que cuenta con un procesador 8086, por ejemplo, mientras que los programas así producidos funcionarán en máquinas con procesadores Z80. Estos compiladores cruzados son lenguajes de alto nivel (como el BASIC), lo que los hace sencillos de utilizar por el programador, pero los programas que crean están escritos en código máquina. Los programadores de código máquina experimentados someten a un cuidadoso examen los programas así desarrollados y a menudo logran optimizarlos aún más.

Es evidente que un sistema de desarrollo posee una enorme ventaja respecto al micro personal. Un ensamblador basado en disco, o uno que utilice un espacio de RAM ampliado para almacenar tablas más grandes, trabajará más eficazmente que un ensamblador copiado desde una cinta y que opera en los confines de un micro personal. A la versión de desarrollo del código se le pueden agregar rutinas de depuración, sin necesidad de preocuparse por si el código generado es demasiado extenso para la memoria. Asimismo, es muchísimo mejor trabajar en un ordenador de oficina que posee un buen teclado, una visualización nítida y unidades de disco.

Una de las firmas que se valen de esta técnica de desarrollo de programas es Intelligent Software (IS), fundada en 1981 a raíz de la combinación de la experiencia de David Levy, el especialista en ajedrez, y ANT Microware, de Robert Madge. La empresa se especializa en juegos de estrategia, escritos mayormente por contrato para los micros personales populares. También desarrolla el aspecto software de las máquinas para jugar al ajedrez.

Además de utilizar para el desarrollo las propias máquinas-destino, IS emplea ordenadores IBM PC y Apple con interfaces desarrolladas especialmente para permitir el intercambio de código entre su gama de máquinas. A menudo la empresa se ve comprometida en proyectos de conversión (p. ej., transferir un juego de ajedrez de un ordenador a otro), de modo que sus programadores han aprendido a escribir código en una forma que se puede segmentar fácilmente. Un nivel de segmentación que resulta útil cuando hay que pasar código de un procesador a otro es la división del programa en código de juego y código de entrada-salida. En la nueva máquina, el código de E/S tendrá diferentes direcciones de puertas o de memoria y tal vez también sea distinto desde el punto de vista estratégico (sondeo —*polling*— reemplazado por interrupciones, etc.). Quizá se requiera cierto ingenio para soslayar las limitaciones del hardware, pero normalmente no habrá una cantidad excesiva de entrada-salida. Por el contrario, habrá código de juego en abundancia, pero como éste está aislado del hardware (excepto, por supuesto, del procesador) su conversión será directa.


Tony Sleep

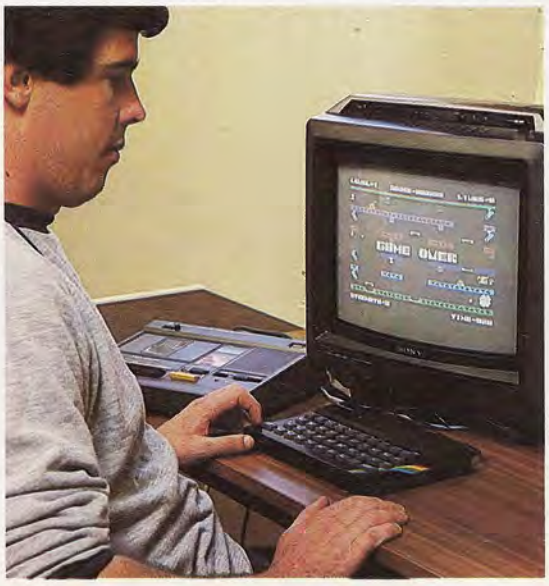
IS desea evitar las restricciones impuestas sobre la capacidad de inventiva de los programadores, de modo que las "reglas de la casa" que regulan la escritura de código son muy pocas. Un punto importante en el que ellos insisten, sin embargo, es en que el código fuente incluya numerosos comentarios, de modo que siempre resulte claro qué es lo que están haciendo las rutinas.

Cuando los programadores trabajan para una firma de software desde su casa, cada uno de ellos desarrollando su propio proyecto, no es posible usar demasiados recursos de la firma en cuestión. En este caso, la individualidad se preserva al precio de una gran cantidad de esfuerzo duplicado, porque cada programador debe reinventar el código para rutinas similares.

Una empresa de software, Psion, está utilizando ordenadores aún más grandes que el IBM PC. De las casas de software que escriben para el mercado de juegos por ordenador personal, Psion es de las pocas que lleva a cabo el grueso de su desarrollo en miniordenadores.

Psion se inició como empresa desarrollando software para el ZX81 (y utilizando equipos ZX81 para hacerlo). Cuando empezó crear la cinta Horizons

Bob Bromide



Visions
Los programadores que trabajan en sus hogares con las máquinas-destino constituyen el grueso de los esfuerzos de programación de Visions. Después de que se hayan decidido la idea del juego y sus escenarios, las rutinas componentes se desarrollan en el lenguaje assembly nativo (Z80 y 6502) utilizando ensambladores como el HiSoft Dev-Pak en el Spectrum

que se edita con cada Spectrum, Psion adquirió un TRS-80 con discos, una máquina que utiliza el mismo procesador Z80, y construyó una interface especial entre las dos máquinas. Pero en agosto de 1982, la empresa decidió que no podía continuar adoptando un sistema de desarrollo completamente diferente cada vez que salía al mercado un nuevo ordenador personal. De manera que reinvirtió los beneficios en comprar hardware pesado de muchísima potencia. En principio, estas máquinas deberían ser suficientemente flexibles para hacer frente a cualquier ordenador que pudiera deparar el futuro. Las máquinas elegidas fueron dos Vax 750, que ejecutan el sistema operativo VMS de DEC.

Las Vax 750 le reportaron a Psion dos ventajas: la calidad del software suministrado por DEC, con la oportunidad que proporciona para crear ayudas de software diseñadas especialmente, y la gran "fuerza bruta" basada en la combinación del sistema operativo y el hardware. Hay muchísimo lugar para un conjunto de ayudas al software, como compiladores, bibliotecas de subrutinas comunes y programas de depuración, todo ello compartido por los programadores, entre 16 y 20, que pueden compartir una sola máquina al mismo tiempo. Las dos máquinas permiten transferir software con toda facilidad de la una a la otra cuando es necesario.

Las bibliotecas de subrutinas comunes ya formaban parte de la filosofía de Psion en los días del TRS-80, pero en un sistema dual de discos flexibles, el intercambio de datos entre discos resultaba tedioso. Las nuevas máquinas Vax permiten que los equipos de programadores trabajen juntos, compartiendo bibliotecas de proyectos comunes de las cuales se pueden llamar módulos casi al instante, y las bibliotecas pueden incluso ser compartidas entre equipos que trabajan en proyectos diferentes. Ésta es la gran ventaja de un sistema de tiempo compartido; y con la ventaja adicional de que también se hará cargo del trabajo administrativo sin tener que interrumpir a los programadores. Psion tiene planeado agregar un tercer Vax para apoyar las tareas administrativas, dejando dos máquinas libres para la producción de software.

Aun cuando se lo pudiera permitir desde el punto de vista económico, estaría equivocado si creyera que sólo con salir y comprarse un Vax usted quedaría instantáneamente al nivel de Psion. De este ambiente de trabajo de Psion, tan bien desarrollado, es muy poco lo que DEC le ha servido en bandeja. Ha llevado muchísimo trabajo, duro y tenaz, hacer que las tareas sencillas se llevaran a cabo con eficacia y fiabilidad, así como lograr las ayudas al software y utilidades forjadas a mano (escritas en c) que Psion ha agregado.

Psion utiliza el c, un lenguaje de "nivel intermedio" que puede producir código objeto razonablemente rápido y compacto para chips de 16 bits como el 8086, que dista mucho del compilador c para ocho bits. De modo que al escribir para máquinas-destino como el Spectrum ha sido necesario que Psion desarrollara sus propias técnicas especiales. Psion es más bien reacia a revelar sus secretos, pero se sabe que utilizó el c para escribir su propio compilador, al cual, a falta de un nombre propio, se lo denomina "nuestro lenguaje de mesa". Éste se parece un poco al c, es portable entre distintos procesadores y crea un código muy eficaz.

Existe una regla universal según la cual el mantenimiento del sistema y la escritura de ayudas para programación caseras, como el lenguaje de mesa, por lo general representan el 30 % de todo el esfuerzo de programación; pero para Psion el tiempo extra bien vale la pena. Desarrollar el código fuente en casa significa la propiedad total: uno puede desmontarlo y mejorarlo o adaptarlo de una forma que sería absolutamente imposible en el caso del software adquirido comercialmente. Si en el software comprado aparece un error, localizarlo es difícil si no imposible y, por lo general, no existe la posibilidad de efectuar cambios internos.

El software especial adquirido por Psion incluye programas que son simulaciones exactas de microprocesadores populares, como el Z80 y el 6502. Por consiguiente, se puede hacer que los gigantescos ordenadores Vax se comporten exactamente como si fueran un Commodore 64 o un Spectrum. A pesar de la potencia de los ordenadores Vax, los simuladores trabajan a una fracción de la velocidad de la máquina destino. La ventaja es que permiten que el programador observe el contenido de todos los registros del interior del microprocesador en cualquier etapa del programa. Esto es especialmente útil para rastrear los errores de los programas. Cuando un programa en código máquina va mal y se cuelga, el programador normalmente no puede decir qué es lo que pasó. Psion puede, por lo tanto, ahorrarse muchas horas en la depuración.

Gran parte de los esfuerzos recientes de Psion en cuanto a desarrollo han estado dirigidos a producir el juego de cuatro programas estándar de gestión que se suministran con el Sinclair QL. La familia de chips Motorola 68000, uno de los cuales es el del QL, se diseñó alrededor de lenguajes de alto nivel, y los programas en c se compilan tan eficientemente en estos chips que hacen innecesaria la escritura en ensamblador. Si todos los ordenadores personales siguieran la ruta señalada por el QL, el c podría reemplazar por completo al ensamblador, y tanto Psion como las casas de software más pequeñas se olvidarían para siempre del trabajo, propio de titanes, que supone la traducción de código a mano.

Tony Sleep

Psion
En 1982 esta empresa adquirió un par de miniordenadores Vax 750 como base de su sistema de desarrollo de software. Cada máquina permite que hasta 20 programadores utilicen simultáneamente la gama de compiladores cruzados, bibliotecas de software y depuradores para crear y traducir programas

Aventura en compañía

Con el MUD muchos jugadores pueden conectar su ordenador a un ordenador central para participar en un juego colectivo

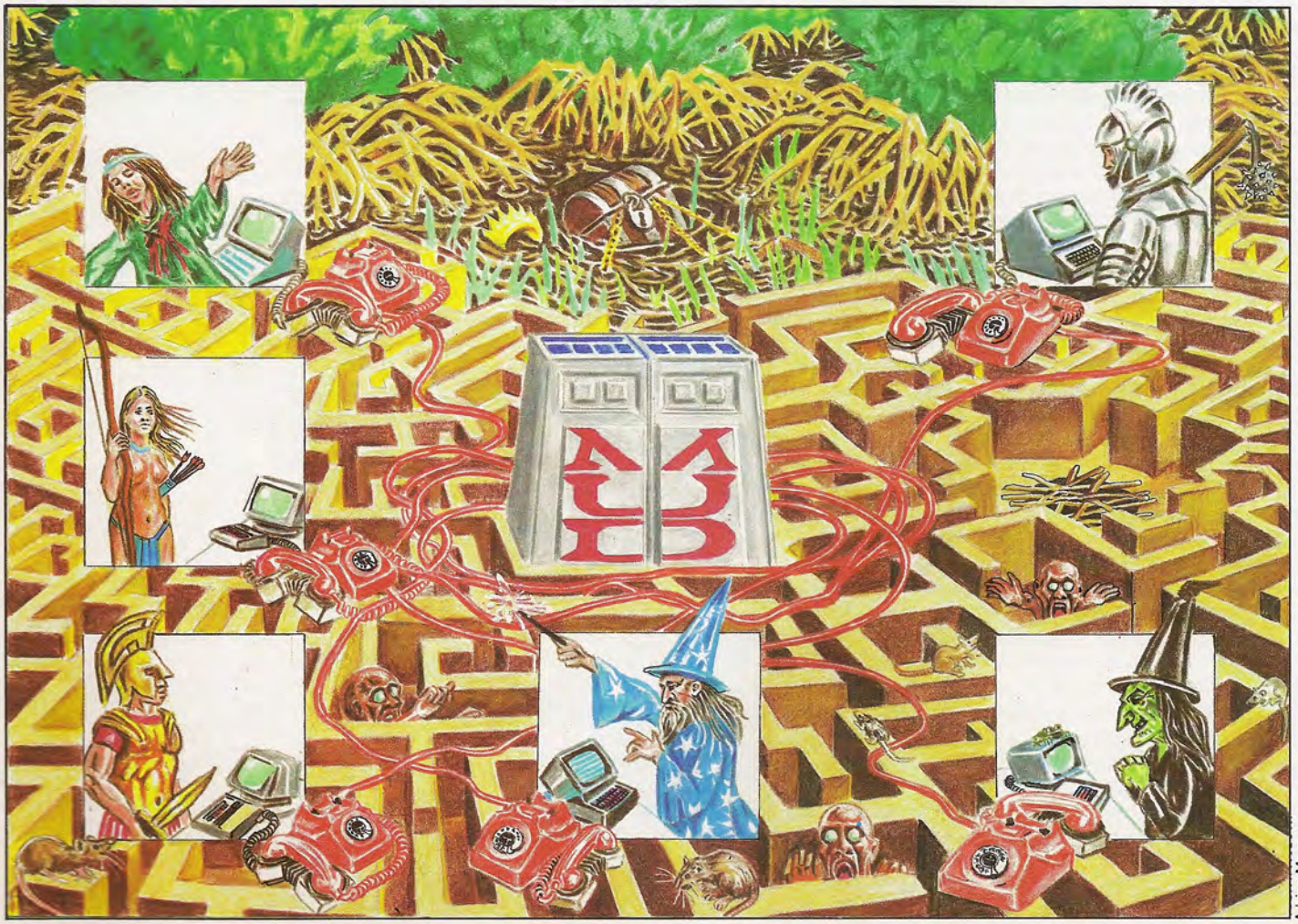
Laberinto de emociones
Las aventuras que utilizan ordenadores centrales permiten la participación de muchos jugadores; en el Multi-User Dungeon, hasta 43 Aprendices, Guerreros, Encantadoras, etc., compiten o colaboran para reunir tesoros y convertirse en Brujos o Brujas omnipotentes. Las aventuras con ordenadores centrales también ofrecen escenarios grandes y detallados: el MUD ofrece El Campo, varias cavernas, un bosque, la isla del dragón, El Mar y El Pantano, todos los cuales pueden contener tesoros y estar habitados por duendes y zombies. La conexión con los jugadores y sus micros personales se realiza mediante seis líneas telefónicas

MUD (*Multi-User Dungeon:* mazmorra multiusuario) es una aventura en tiempo real en la que usted se encuentra con otros jugadores, con los que puede mantener conversaciones, pedirles consejo, unirse contra un enemigo común o luchar contra ellos. Estos jugadores no forman parte del programa: están participando en este juego al mismo tiempo que usted, y, por lo tanto, acciones suyas influyen sobre las de usted.

El MUD se ejecuta en un ordenador DEC10 gigante instalado en la Universidad de Essex (Gran Bretaña) y todo lo que necesita el usuario para jugarlo en su propio microordenador es un programa emulador de terminal, un teléfono, un modem y una cuenta PSS (*Packet Switching System*: sistema de conmutación de paquetes). Un programa emulador de terminal permite que su micro se comunique con el ordenador central a través de un enlace telefónico. El usuario puede escoger entre escribir su propio emulador o comprarse uno. Lo ideal es que permita el desplazamiento de las líneas en la pantalla *(scrolling)* y dé una longitud de línea de 80 caracteres. El software Micronet no es apropiado, ya que no permite esto. Se pueden utilizar micros con visualizaciones de menos de 80 columnas, pero son incómodos. Existen a la venta algunos paquetes excelentes; por ejemplo, Termi y Communicator son chips de ROM para el BBC Micro.

Es necesario que el modem sea capaz de comunicarse con el PSS de la British Telecom (Compañía de Telecomunicaciones Británica) y puede ser de 300/300, 1200/75 o 1200/1200 baudios. Un acoplador acústico Micronet resulta bastante satisfactorio. El PSS es un servicio de conexión en red que permite que el usuario establezca contacto con ordenadores anfitriones distantes, a menudo por el costo de una llamada telefónica local.

El procedimiento para ganar acceso al ordenador central es sencillo y directo. Con el software emulador de terminal ejecutándose, usted llama al núme-



Adrian Morgan

ro de teléfono de la central telefónica del PSS en la que está registrado, conecta su modem y digita su identidad. Luego usted entra la dirección del PSS del DEC10 de la Universidad de Essex. Allí se ha preparado una cuenta especial para permitir el acceso gratuito al MUD desde la medianoche hasta las 7 de la mañana de los días laborables, y desde las 10 de la noche a las 7 de la mañana los fines de semana, cuando la carga del ordenador es menor. Conéctese (*log*) a esta cuenta especial y luego entre RUN MUD.

El programa se actualiza continuamente, de modo que lo primero que el usuario ve es la fecha de la última versión. Se elige un nombre de personaje con el cual se jugará y se le dice al MUD qué es este personaje. Si está jugando al MUD por primera vez, entonces se creará un nuevo personaje para usted, que será clasificado como un Aprendiz. Los niveles de experiencia son:

Nivel	Puntos	Hombre	Mujer
1	0	Aprendiz	Aprendiza
2	400	Guerrero	Guerrera
3	800	Héroe	Heroína
4	1600	Campeón	Campeona
5	3200	Superhéroe	Superheroína
6	6400	Encantador	Encantadora
7	12800	Hechicero	Hechicera
8	25600	Mago	Maga
9	51200	Legendario	Legendaria
10	102400	Brujo	Bruja

Si usted acaba su sesión sin haber sido muerto, el programa anotará los puntos que haya acumulado y la próxima vez que juegue continuará a partir de esa puntuación. El usuario consigue puntos e incrementa su nivel de experiencia arrojando piezas del tesoro en el pantano, superando problemas ocasionales, destruyendo diversos seres repulsivos como ratas y zombies, y ganando una lucha contra otro jugador. Si en el curso de una pelea con otro participante usted muere, su personaje queda eliminado del juego y debe volver a comenzar como Aprendiz, sin ningún punto.

El usuario empieza el juego en un "estrecho sendero a través del campo". Al digitar "WHO" (quién) se le proporcionará una lista en pantalla de todos los que están jugando en ese momento. Usted puede decidir saludar a alguno de ellos. Por ejemplo, al digitar "Jez, hola!, soy nuevo y necesito algún consejo" le transmitiría ese mensaje al terminal del jugador llamado Jez. O bien, les podría hablar a todos los jugadores al mismo tiempo digitando: "SHOUT (gritar), OK, vosotros basuras terminales, cuidaos que aquí vengo yo", pero no es aconsejable que empiece su primer juego de esta manera.

"HELP" (ayuda) le proporcionará alguna información sobre cómo moverse y una breve explicación de muchas de las órdenes. Las opciones de movimiento se explican de la siguiente forma: "Se aceptan la mayoría de las órdenes de movimiento simples, como n, so, o, arriba, saltar, más otras que tú tendrás que descubrir!"

Las instrucciones disponibles se pueden listar digitando "COMMANDS" (instrucciones). Existe mucha información disponible, e irlas transcribiendo en papel mientras las palabras van apareciendo en la pantalla del monitor no es cosa fácil. Algunos programas de emulación terminal le permitirán a usted copiar todo lo que aparece en su pantalla en disco para poder consultarlo después. Esto también le posibilitará diseñar un mapa adecuado del terreno, que se puede actualizar debidamente después de cada juego. Éstas son las instrucciones que estuvieron disponibles un cierto día traducidas al castellano:

INSTRUCCIONES

AutoQuién, ‹segundos›
Breve
Converse
Vaciar ‹bolso›
Seguir ‹nombre›
Dar ‹ítem› a ‹nombre›
Ayuda
Horas
Inventario
Besar ‹nombre›
‹nivel›, ‹mensaje›
Mirar ‹dirección›
NoContraseña
Pronombres
"‹mensaje›"
Vengarse ‹ítem›
Gritar, ‹mensaje›
Robar ‹ítem› a ‹nombre›
decir ‹nombre›, ‹mensaje›
Prolijo
Quién
Detrás
Bicho
Arrojar ‹ítem›
Salidas
Coger ‹ítem›
Ayuda ‹nombre›
Abrazar ‹nombre›
Conservar ‹ítem›
Nivel
Mirar alrededor
Perder ‹nombre›
Contraseña
AnimarQuién
Rehusar ‹nombre›
Salvar
Dormir
Pesar ‹ítem›
Escribir ‹objeto›, ‹mensaje›
Frenético
Adiós
Arrojar todo
Huir ‹dirección›
Coger todo
Ir ‹dirección›
Pistas
Info
Matar ‹nombre›
Conectarse
Mirar ‹bolso›
Meditar
Personaje
Salir
Marcador
Hechizo
Deshacerse
Cuándo

El MUD es una larga aventura basada en textos, con descripciones extensas y detalladas de los lugares. Cuando se está familiarizado con el escenario se puede digitar "BREVE" y no tendrá que leer las descripciones cada vez. Los suscriptores del Prestel ya estarán al corriente de la lentitud y la insuficiencia de los gráficos de teletexto, y si bien las aventuras basadas en gráficos son una novedad interesante, los jugadores de juegos de aventuras exclusivos siempre preferirán un juego basado en textos. Una aventura compuesta sólo de textos permite una implicación en la misma mucho más imaginativa que una basada en gráficos, del mismo modo que una emisión de radio se puede disfrutar más a veces que la televisión. Otra desventaja de las aventuras con gráficos es que cada marca de micro personal requerirá una versión diferente del juego, debido a las diferentes características de gráficos de los ordenadores personales.

Es probable que muchos de los jugadores del MUD utilicen un BBC Micro, un Apple o un Spectrum, pero otros tendrán terminales de segunda mano comprados en tiendas de ocasión y, por lo tanto, la gama de máquinas que utilizan el programa es muy amplia.

Se espera que el MUD sea comercializado pronto. Los autores del programa, Richard Bartle y Roy Trubshaw, están escribiendo una versión para un VAX Computer, que será comercializada por Century Communications.

El juego también se seguirá practicando a través de enlaces telefónicos, como el PSS y el Prestel, si bien de existir suficiente demanda también se podría llegar a ofrecer por cable. Los jugadores del MUD pagarán entonces una tarifa para unirse al juego, más un pequeño recargo por cada hora.

El amo de la mazmorra

Para obtener más detalles sobre el Multi-User Dungeon, contactar con:

Richard Bartle
Department of Computer Science
University of Essex
Colchester
Essex
Gran Bretaña

Componentes primitivos

Veamos ahora cómo la comprensión de los principios de los algoritmos puede ayudarle a mejorar su programación

Un algoritmo es una serie de instrucciones que indican cómo se puede realizar un proceso. Un algoritmo describe éste en función de otros procesos que ya han sido definidos, o bien desde el punto de vista de procesos que son tan básicos que no necesitan ser definidos. De modo que, en una receta de cocina, una instrucción podría ser "preparar una salsa bechamel", habiéndose proporcionado en algún otro lugar del libro de cocina una receta de salsa bechamel (algoritmo). Otra instrucción podría ser "ponga la mezcla al fuego hasta que hierva", donde se supone que el lector puede comprender totalmente la operación de poner algo al fuego hasta que hierva. Desde el punto de vista de la programación, los algoritmos se construyen a partir de instrucciones que utilizan algoritmos (procedimientos, rutinas, funciones) escritas en algún otro lugar del programa, o bien algoritmos ya incorporados en el lenguaje (instrucciones como PRINT y DIM, o funciones matemáticas como LOG y TAN).

En este capítulo analizaremos cómo se construyen algoritmos a partir de otros algoritmos y procesos o funciones *primitivas*. Las primitivas a disposición del programador son las instrucciones y funciones de que dispone el lenguaje. A partir de éstas, se escriben algoritmos que pueden realizar cosas pequeñas (p. ej., mover un sprite o aceptar un número como entrada). Estos algoritmos se utilizan luego para construir algoritmos más generales (actualizar la visualización del juego o controlar un sistema de menú) y estos últimos se emplean a su vez como componentes de otros mayores, hasta que todo el programa, contemplado como un único algoritmo, se escribe en términos de algoritmos de nivel inferior. Este concepto es la base de la llamada programación *estructurada* o *modular*.

Diseño de algoritmos

Un algoritmo posee una entrada y una salida. Esto equivale a decir que, como proceso, el algoritmo trabaja con unos datos iniciales para producir un resultado. Estos datos iniciales se le pasan al algoritmo desde el exterior disfrazados de "parámetros", que permanecen constantes para cualquier empleo particular del algoritmo pero que podrían cambiar para usos diferentes. Pasar parámetros es un hecho familiar hasta para el programador novato, dado que el sencillo programa:

```
procedimiento(parametro1,parametro2,etc.);
```

le pasa el parámetro "Hola mundo!" al algoritmo llamado por la instrucción PRINT. Ejemplos similares son FNA(P), TAN(P), LEFT$(P$,5) y POKE P,5, donde P, P$ y 5 son todos parámetros. Del mismo modo, los resultados de un algoritmo son devueltos como parámetros. Si el lenguaje de programación que se está utilizando posee variables locales (p. ej., el PASCAL y el C), los parámetros se pasarán con una llamada a procedimiento, como en:

```
10 PRINT "Hola mundo!"
```

Al diseñar un algoritmo, es un primer paso esencial considerar el contenido de los parámetros de entrada y salida, sus tipos (entero, de coma flotante, real, serie, etc.) y sus magnitudes y rangos.

Cuando se han definido las entradas y las salidas del algoritmo, el siguiente paso consiste en pensar qué acciones realizar sobre las entradas para obtener las salidas.

Lo más obvio, y lo que más frecuentemente se menosprecia, es tomar prestado el algoritmo de algún otro sitio. Al nivel más simple, las funciones incorporadas de un lenguaje de programación proporcionan muchos algoritmos útiles, como manipulación de series, funciones trigonométricas, entrada-salida y (posiblemente) clasificación y manipulación de matrices. Aparte de esto, puede que el algoritmo necesario ya exista en algún otro de sus programas. El código para el mismo se podría incorporar al programa nuevo (es sumamente útil que cada programador cree su propia biblioteca de algoritmos). Además, existen conjuntos de algoritmos editados que con frecuencia se pueden obtener en bibliotecas públicas.

También es interesante examinar los programas que se publican en las revistas de informática para buscar rutinas que puedan ser de utilidad. Por último, hay algoritmos que se pueden aplicar en otros campos de acción y, a pesar de que jamás estuvieron pensados para la informática, resultan sumamente útiles. La sección de contabilidad de la biblioteca local puede estar llena de libros que contienen fórmulas para calcular balances y depreciaciones. Una pequeña investigación entre estos libros podría simplificar muchísimo la escritura de un programa de cuentas y es probable que el resultado final fuera mucho más fiable. Lo mismo se puede decir de otras disciplinas: ingeniería, electrónica, matemáticas, etc.

Tanto al adaptar un algoritmo ya existente como al crear uno hay ciertos criterios que se le deben aplicar a cada una de las instrucciones que el mismo contiene. Éstos son *precisión* y *eficacia*. Precisión significa que la instrucción no debe ser ambigua en ningún sentido. Es fácil introducir ambigüedad en una primera etapa, cuando el algoritmo se está escribiendo en castellano. Palabras en castellano

como "y" y "o" son muy diferentes del AND (y) y el OR (o) de la lógica booleana. Por ejemplo, si el algoritmo está pensado para seleccionar todos los nombres de una lista que empiecen con "A" *y* todos los que empiecen con "B", se podría fácilmente escribir un código como:

```
IF PRIMERALETRA = "A" AND PRIMERALETRA
="B" THEN
```

lo cual está mal, ¡porque lo que se necesita es un OR (o) lógico! (es evidente que ninguna letra puede ser una "A" y una "B" simultáneamente).

El criterio de la eficacia es la necesidad de que el programa no contenga instrucciones imposibles. Se dice que una instrucción es eficaz cuando se puede llevar a cabo con lápiz y papel en un tiempo finito. Esto significa que instrucciones como "X igual al mayor número primo" no es eficaz (porque no existe un mayor número primo).

Consideraciones generales

Existen, asimismo, criterios para juzgar al algoritmo como un todo. Un algoritmo debe *terminar*. El que ofrecemos no termina (aun cuando sus instrucciones son precisas y eficaces), y si se codificara en un programa sería un bucle sin fin:

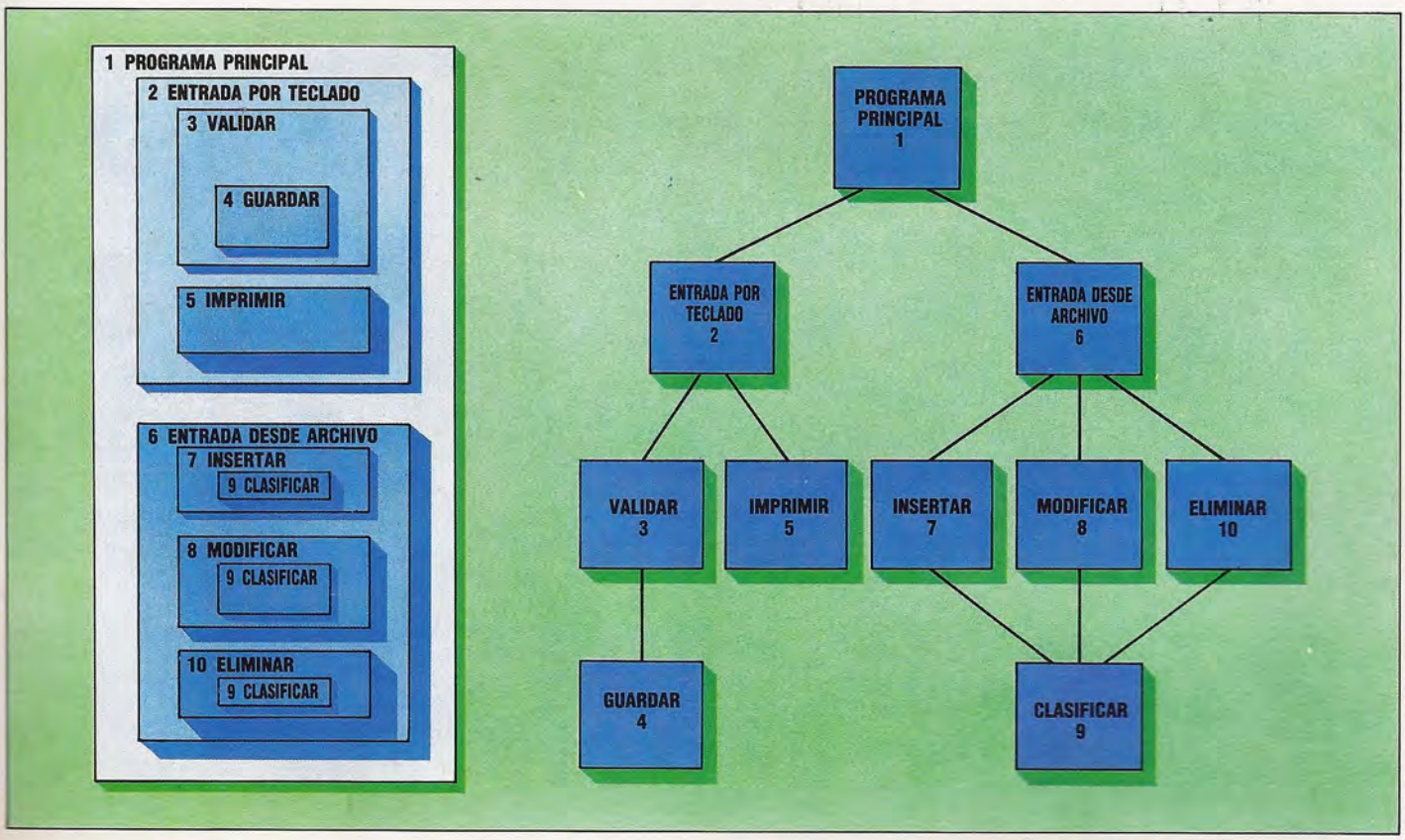
```
paso 1 poner I igual a 1
paso 2 si I > 3 entonces terminar
paso 3 ir al paso 1
```

No siempre es fácil decir si un algoritmo terminará o no pero, en general, los algoritmos que implican bucles verifican una condición determinada antes de terminar (p. ej., si I > 3, en el ejemplo dado) y es necesario verificar que sea posible satisfacer esa condición.

Eficiencia, generalidad y *elegancia* son criterios aplicados a la evaluación de algoritmos. La eficiencia se suele juzgar desde el punto de vista del tiempo de proceso y empleo de memoria. Normalmente los dos son bastante compatibles; un código rápido podría necesitar un espacio relativamente pequeño, aunque esto no es siempre así de manera obligada. Habiendo hallado un algoritmo, se lo puede "afinar" para dotarlo de mayor eficiencia modificando sus detalles. Un cálculo será notablemente más rápido y empleará menos memoria si, por ejemplo, se usa aritmética de enteros y no de coma flotante.

Generalidad es la capacidad de un algoritmo para hacer frente a muchas situaciones diferentes además de aquella para la cual fue diseñado. Vale la pena, a la larga, intentar que todos los algoritmos sean lo más generales posible. Si un programa requiere varias veces una respuesta sí/no, valdría la pena escribir una rutina que le indicara al usuario que "por favor digite S o N", aceptara la entrada, verificara si es "S" o "N", volviera a solicitarla en caso de que no fuera ninguna de ellas y que, en caso de ser correcta, retornara la respuesta entrada. Sin embargo, se podría hacer que la rutina fuera más general escribiéndola para que contemplase distintas preguntas y distintas respuestas potenciales, de modo que se la pudiera utilizar en muchas situaciones diferentes. Elegancia significa hallar algoritmos que sean simples e ingeniosos a la vez. En todos los casos es más sensato hallar algoritmos eficaces y generales que algoritmos elegantes, aunque a menudo los tres conceptos resultan muy compatibles.

Otro aspecto importante de los algoritmos es el flujo de control y de datos dentro de ellos y cómo éste se puede representar mediante diagramas de flujo. Lo analizaremos en el próximo capítulo.



Estructura y niveles de proceso
El diagrama de estructura de bloques de la izquierda ilustra claramente cómo se anidan los algoritmos de un programa, mientras que el diagrama de flujo de procedimiento, a la derecha, hace hincapié en las articulaciones y los niveles del proceso del mismo programa. Los algoritmos más "primitivos" son los que están anidados más profundamente y los que ocupan el lugar inferior en la jerarquía

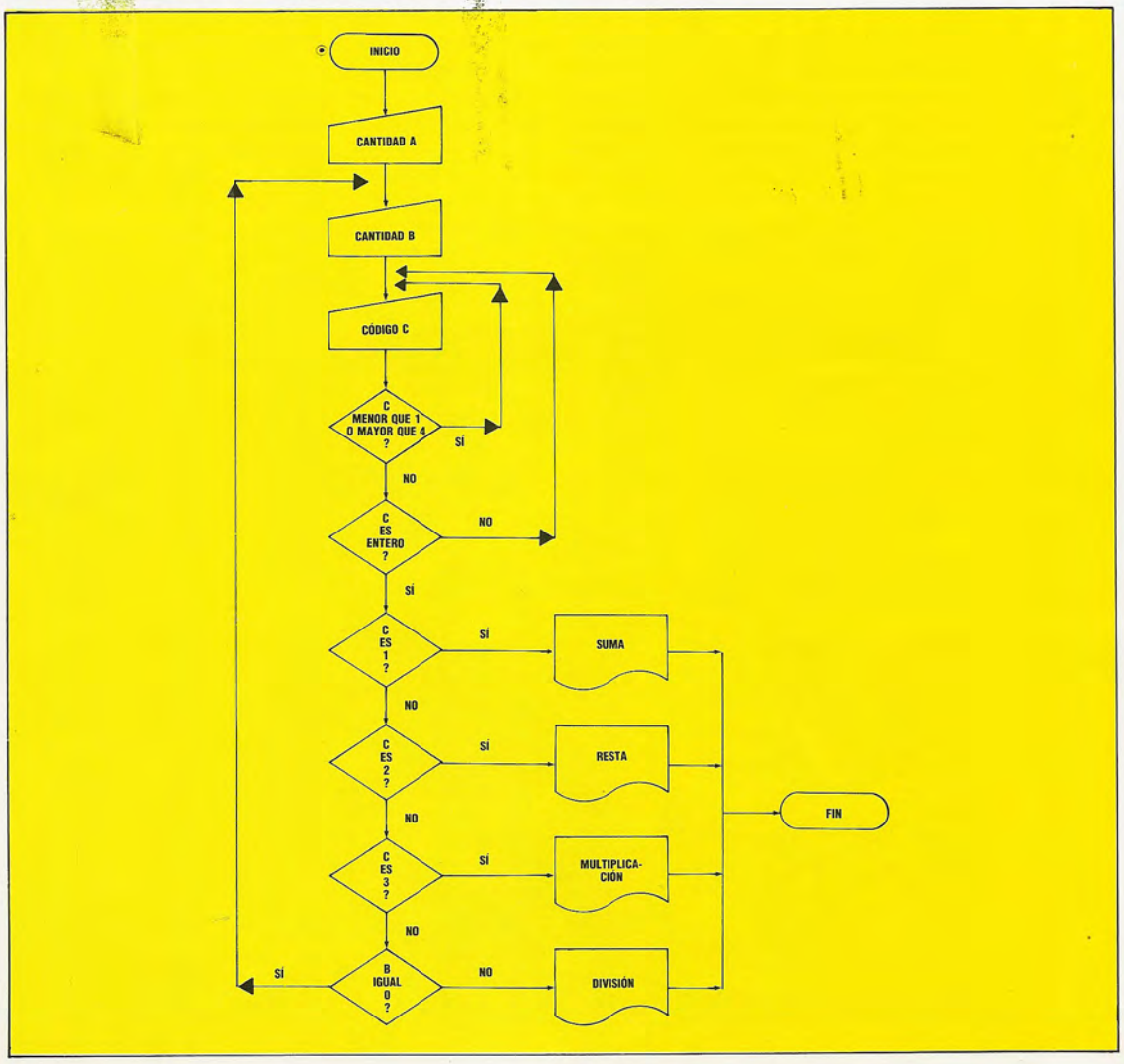
Kevin Jones

Test en cascada (y 3)

Con este nuevo planteamiento, se completa el problema expuesto en el capítulo anterior

En este segundo ordinograma se ha suprimido la última pregunta de la serie eliminatoria de operaciones a realizar. Asimismo, puede comprobarse la inclusión de una comparación, tras la entrada del número, sobre si éste está entre uno y cuatro; pero aun con todo existe un segundo problema: ¿qué pasaría si se introdujera, dentro de ese rango, un número que contuviera decimales? Con la segunda pregunta (que verifica si el número es entero o no) se elimina dicha posibilidad y se tiene la certeza de que no puede entrar ningún elemento extraño. Todo dato introducido se decantará por una vía u otra. Por ejemplo, de no haberse incluido la pregunta inicial, un número erróneo, el 7, habría ido descendiendo y, al no existir control de código 4, tras el pase por el del número 3 con respuesta negativa y la decisión sobre si la segunda cantidad es 0, habría realizado, sin corresponderle, la división.

```
10 REM *********CODIGOS
20 INPUT "PRIMERA CANTIDAD";A
30 INPUT "SEGUNDA CANTIDAD";B
33 REM *********ENTRADA CODIGO
40 INPUT "CODIGO.....";C
45 REM****FILTROS
50 If C < 1 OR C > 4 THEN GOTO 40
60 IF C<> INT(C) THEN GOTO 40
70 REM *********DECISIONES
80 IF C=1 THEN PRINT"LA SUMA DA" A+B :
   END
90 IF C=2 THEN PRINT"LA RESTA DA" A-B :
   END
100 IF C=3 THEN PRINT"LA MULTIPLICACION
   DA" A*B : END
110 IF B=0 THEN GOTO 30
120 PRINT"LA DIVISION DA" A/B
125 END
```



Un sistema eficaz

Examinamos aquí el paquete Adam, que amplía la consola de juegos ColecoVision a un rango de ordenador personal

El sistema Adam está diseñado para apoyarse en la consola para juegos para televisor ColecoVision y los usuarios de esta unidad la puedan ampliar para satisfacer todas las especificaciones del Adam mediante unidades accesorias. De hecho, hasta la fecha todos los Adam que están a la venta se basan en el módulo ColecoVision, aunque hay planificado un Adam individual más compacto.

Cuando se desempaquetan por primera vez las cajas que contienen el sistema Adam, la cantidad de componentes resulta un poco desconcertante. El módulo ColecoVision se engancha en una bandeja plástica grande y el "módulo de memoria" (que contiene asimismo el microprocesador Z80A y la unidad de cinta) se conecta con la unidad de juegos a través de una puerta de ampliación. El módulo se mantiene en su sitio mediante unos clips que lo sujetan a la bandeja. El teclado se conecta mediante un trozo de cable en espiral y uno de los dos controladores de juegos que se proporcionan se puede insertar en una ranura a uno de los lados del teclado para actuar como un teclado numérico. La impresora está conectada al módulo de memoria, y la potencia para todo el sistema se suministra por un cable que va de la red eléctrica a la impresora.

El teclado posee 75 teclas, incluyendo seis teclas de función, un juego de teclas de control para el procesador de textos, un grupo para el cursor y las alfanuméricas habituales. Está bien diseñado y es ligero y fácil de utilizar sobre el regazo.

Cuando se conecta por primera vez el sistema, el Adam está en la modalidad "máquina de escribir electrónica". Pulsando una tecla se visualiza simultáneamente el carácter correspondiente en la pantalla y se imprime en la impresora. La pantalla es una representación de una hoja de papel y del rodillo portapapel de una máquina de escribir. La utilidad del Adam en esta modalidad es limitada, pero la pulsación de una tecla coloca al sistema en modalidad "SmartWriter" (escritor eficaz).

SmartWriter es un procesador de textos simple pero efectivo que obviamente fue diseñado teniendo en cuenta al principiante. Se mantiene la visualización "rodillo de máquina de escribir" y la línea inferior de la pantalla visualiza las instrucciones destinadas a las teclas de función. El uso adecuado de estas teclas, junto con las diversas teclas de control del procesador de textos del teclado, hace que la operatoria del SmartWriter sea sumamente sencilla; de hecho, el manual es en gran parte superfluo, ya que las opciones del menú conducen al usuario a través de todos los pasos necesarios para almacenar, visualizar e imprimir el texto. El formato de la pantalla es de 36 columnas por 20 líneas, pero se pueden manejar líneas más largas utilizando la pantalla como una "ventana" de todo el texto.

El principio de la unidad de cinta es similar al del microdrive de Sinclair, pero posee mayor capacidad (supuestamente 256 Kbytes por cinta). Es mucho más veloz que la cinta de cassette común.

Chris Stevens

La familia Adams
El Adam es una máquina de todo o nada. Se vende como un sistema completo: micro, unidad de cinta de gran velocidad, impresora de rueda margarita, teclado, dos palancas de mando y tres paquetes de software (dos juegos excelentes y un procesador de textos). Todo esto hace que el Adam sea un ordenador personal bastante potente; sin embargo, esencialmente es una forma de mejorar una unidad de juegos para televisión

Jugando con Buck
El juego Buck Rogers viene en cinta para la propia unidad del Adam y la aprovecha al máximo. El juego está dividido en varias fases, que son demasiado largas para caber en la memoria simultáneamente, de modo que el Adam carga una fase desde la cinta y luego, mientras se está jugando, carga la siguiente, y así sucesivamente. Al final de cada juego la cinta se rebobina automáticamente y si se ha conseguido una puntuación elevada, ésta se graba en la cinta. Por consiguiente, la tabla de puntuaciones refleja las mejores marcas que se han conseguido, no sólo la mejor desde que se conecta el ordenador

Asimismo, está totalmente bajo el control del ordenador, por lo que se puede rebobinar la cinta por programa para encontrar un archivo determinado. Las cintas (o los "paquetes de datos", en la jerga de Coleco) ya se suministran formateadas, de modo que no se pueden utilizar cassettes de audio normales. Cuando se enciende el Adam, se carga el programa de la cinta que en ese momento esté en la unidad, que puede ser un paquete de juegos, la cassette de SmartBASIC o un programa del usuario. Si la unidad está vacía, el sistema se coloca automáticamente en "máquina de escribir electrónica".

La impresora de rueda margarita utiliza bobinas de papel o bien papel en hojas sueltas, y su calidad de impresión es alta.

A pesar de los 80 Kbytes de RAM, el SmartBASIC, basado en cassette, sólo deja 28 Kbytes para el usuario. El SmartBASIC es similar a la versión del BASIC de Apple y es igualmente fácil de utilizar, con buenos gráficos y una selección de 16 colores. El manual de BASIC, sin embargo, es verdaderamente delirante, escrito en un tono condescendiente y lleno de expresiones infantiles, tales como *boo-boo* por "error". Pero las similitudes entre el SmartBASIC y el BASIC Apple hacen que los usuarios dispongan de una gran cantidad de libros y otro material al cual remitirse para consulta.

A primera vista el sistema Adam parece ofrecer una excelente relación calidad-precio. El software incorporado es ideal para escribir cartas, informes o artículos cortos y el sistema de cinta es la simplicidad personificada. Una ventaja adicional es la gran cantidad de software para juegos de gran calidad que hay disponible (en el Adam no sólo se pueden utilizar cartuchos ColecoVision, sino que comprando un adaptador también se podrá disfrutar de la inmensa gama de juegos de Activision y Atari). El software más serio se está vendiendo en cinta, si bien el número de paquetes disponibles es más bien limitado. Entre ellos se cuentan una hoja electróni-

Teclado separado
Sólo hay dos micros personales que se venden con teclado separado, aun cuando esto se considera esencial en el caso de los ordenadores de oficina. El teclado del Adam incluye teclas de función y teclas exclusivas para el software de tratamiento de textos que viene con el micro

Mando de la palanca de mando
Aunque el brazo del mando de la palanca es mucho más corto que el de la mayoría de ellas, es suficiente para usarlo con seguridad

Almacenamiento de palancas de mando
Aquí se pueden guardar, cuando no se las utilice, dos palancas de mando. Éstas se enchufan en conectores situados junto a este sector

Sistema de videojuegos ColecoVision
Esta unidad se vende sola como un ordenador exclusivo para juegos, pero se puede ampliar, convirtiéndola en un ordenador, mediante la adición de otras unidades

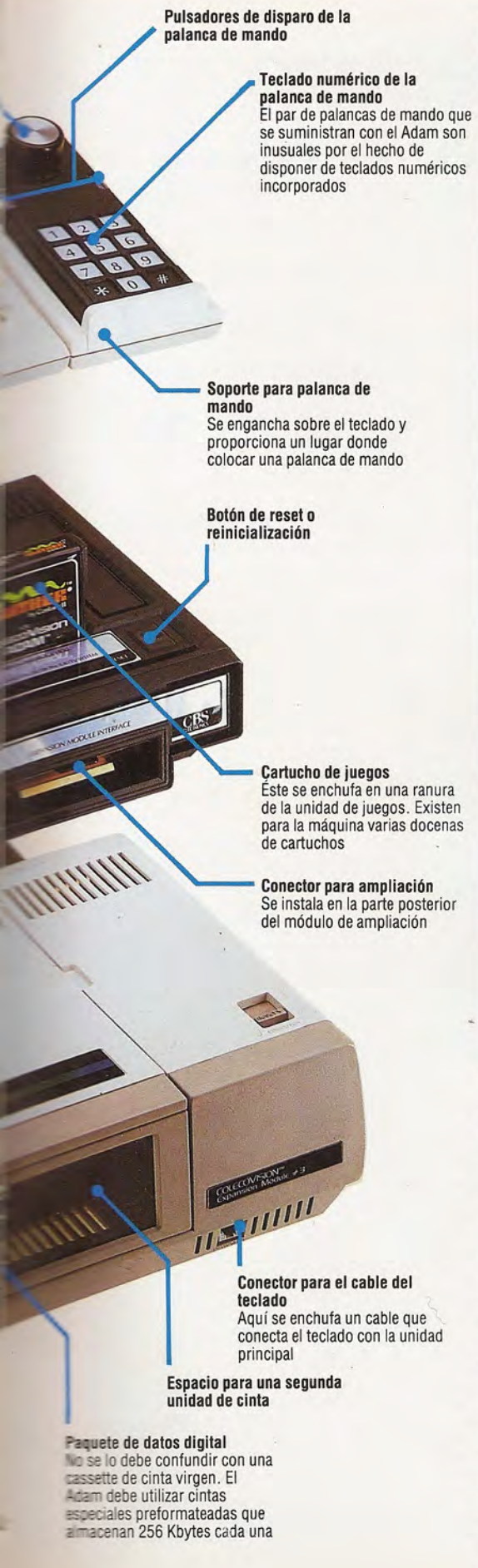
Bandeja del equipo
Se encaja en la parte inferior de las dos unidades para mantenerlas firmemente unidas

Conector para impresora
En este conector se enchufa la impresora que viene incluida con el Adam. A través de ella se le suministra corriente al Adam, de modo que la impresora siempre debe estar conectada, incluso cuando no se la utilice

Unidad de cinta digital
Podría parecer una grabadora de cassette normal, pero se trata de una unidad de cinta de gran velocidad controlada totalmente por el ordenador

Chris Stevens

Chris Stevens



Las palancas de mando del Adam
Con el Adam se suministran como estándar dos palancas de mando. Una de ellas actúa asimismo como teclado numérico y se coloca en un soporte especial a un lado del teclado. Coleco también comercializa dos sistemas *deluxe* para control de juegos. El conjunto Super Action Controller se compone de un par de palancas de mando de acción sencilla con teclados numéricos y gatillos tipo fusil en vez de los pulsadores de disparo normales. Estas se instalan con conectores Atari de tipo D estándares y, por consiguiente, se pueden utilizar con otros sistemas. Se incluye un cartucho de juegos Baseball gratuito. El Roller Controller es un panel de control de juegos de tipo recreativo. Posee dos palancas de mando Coleco estándar y un controlador "de bola" para una respuesta rápida y un control exacto. Dos juegos de pulsadores de disparo dobles permiten la acción de dos jugadores, y un conmutador de modalidades permite seleccionar entre operación por palanca de mando u operación por mando de bola.

ca, una base de datos y una versión del lenguaje LOGO. La adición de la ampliación de memoria prometida, de 144 Kbytes, debería proporcionar algún tipo de capacidad para CP/M.

Pero también hay inconvenientes. La calidad de impresión no es todo lo buena que uno esperaría de una impresora de rueda margarita; la impresora, además, es notablemente lenta y ruidosa, hay una grave carencia de RAM para el usuario y el hecho de que el BASIC se deba cargar desde cinta pronto resulta fastidioso. Con la reciente caída de los precios de las impresoras de rueda margarita, sería posible reunir un sistema que superase al Adam en rendimiento pero costará aproximadamente lo mismo. Dicho esto, parece que el Adam es una adquisición interesante en el caso de que el usuario ya posea la consola de ColecoVision. Para quienes desean adquirir un sistema de ordenador personal con capacidades para tratamiento de textos, el Adam es, por cierto, digno de consideración..., pero bien podría haber otras alternativas mejores.

COLECO ADAM

DIMENSIONES
381 x 279 x 102 mm (unidad de ampliación de memoria)
381 x 355 x 152 mm (impresora)
381 x 152 x 51 mm (teclado)

CPU
Z80A

MEMORIA
80 Kbytes de RAM, de los cuales se utilizan 16 K para la visualización en video. Ampliable a 144 K con el paquete de RAM opcional

PANTALLA
36 columnas x 20 filas (31 columnas x 24 filas en BASIC). 256 x 159 pixels en alta resolución, con 16 colores

INTERFACES
Conector para impresora SmartWriter, enchufe para teléfono modular Adam Net, tres ranuras para ampliación, interface para ampliación ColecoVision

LENGUAJES DISPONIBLES
SmartBASIC, suministrado en cassette; sistema operativo CP/M

TECLADO
75 teclas, trazado QWERTY, teclas esculpidas tipo máquina de escribir. Incluye 6 teclas de función programables

DOCUMENTACION
Con el Adam vienen tres manuales: uno de instalación de 64 páginas, una guía para el programa de tratamiento de textos SmartWriter y un manual de programación en SmartBASIC. Por lo general la documentación es fácil de seguir, pero algo recargada e insustancial. El manual de BASIC es insólito

VENTAJAS
La gran disponibilidad de cartuchos de juegos ColecoVision y la gran calidad de los mismos hace que el Adam resulte ideal para los amantes de los juegos. El software de tratamiento de textos es sencillo de utilizar

DESVENTAJAS
Impresora lenta y ruidosa. Paquetes de datos disponibles sólo de Coleco. Software disponible en cintas limitado

Lento pero seguro

En una serie de lecciones estudiaremos la construcción de un juego utilizando el BASIC BBC. En cada capítulo añadiremos una parte del programa total

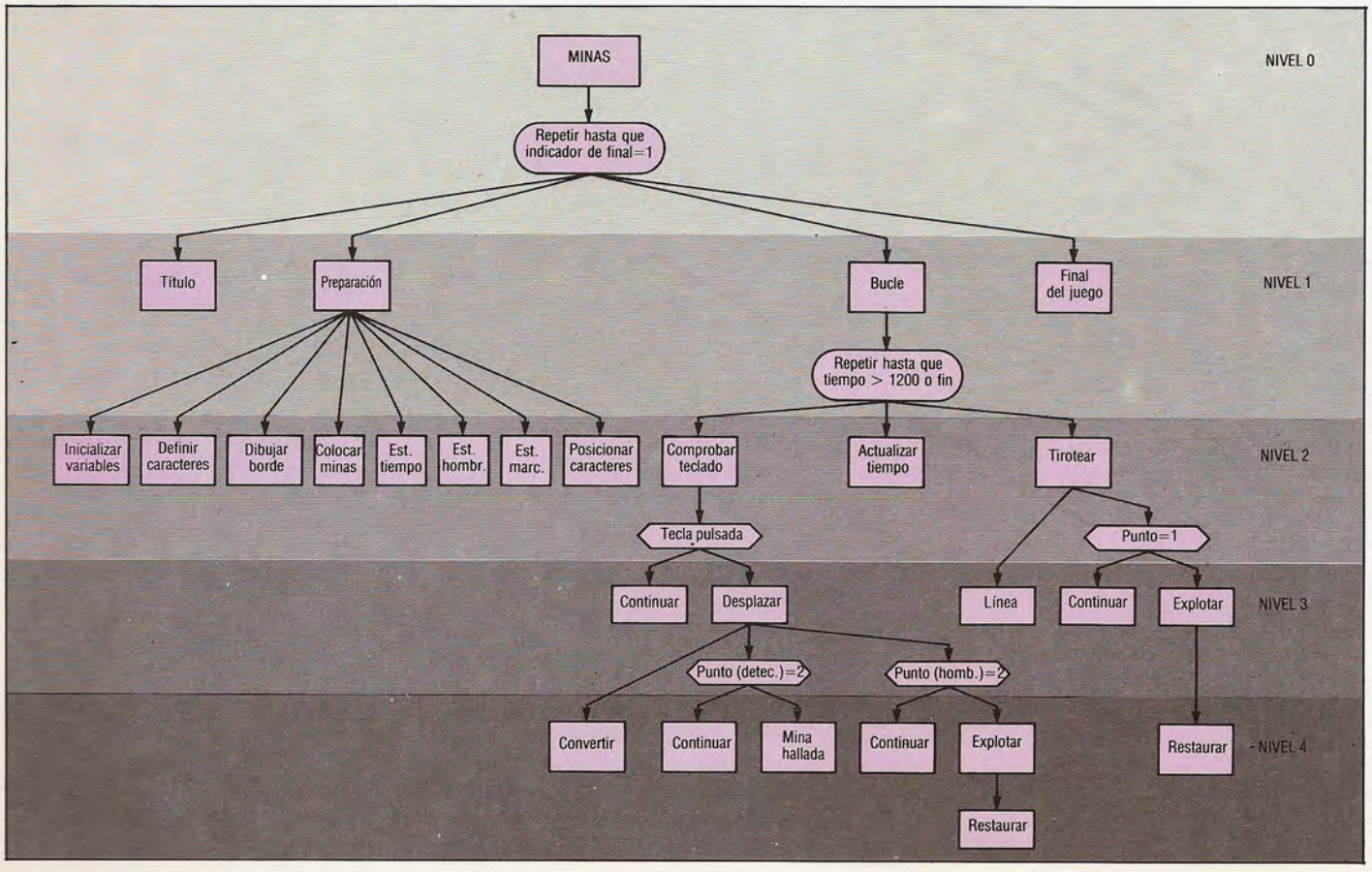
El BASIC BBC le ofrece al programador dos ventajas respecto al BASIC Microsoft estándar: es rápido en ejecución y posee características que permiten la confección de programas estructurados. La esencia de un programa estructurado consiste en desarrollar pequeñas secciones independientes de código que se puedan depurar individualmente antes de ensamblarlas en un programa más grande. Todo programa en BASIC se puede estructurar hasta cierto punto mediante el empleo de subrutinas para codificar cada módulo del programa, pero el BASIC BBC posee tipos especiales de subrutinas, llamadas *procedimientos*. A éstos se los puede considerar como bloques de código diseñados para realizar un trabajo específico dentro del programa. Por ejemplo, imaginemos una porción de programa que ha de hacer una pausa entre cada instrucción durante un tiempo dado. En BASIC estándar, ésta se podría escribir utilizando un *bucle ficticio*, es decir, un bucle que no hace nada más que consumir tiempo en su ejecución y que podríamos transcribir en la siguiente manera:

```
10 PRINT "PRIMERA SECCION"
20 FOR I=1TO100:NEXT I
30 PRINT "SEGUNDA SECCION"
40 FOR I=1TO100:NEXT I
50 PRINT "TERCERA SECCION"
60 FOR I=1TO100:NEXT I
70 PRINT "CUARTA SECCION"
80 END
```

No obstante, sería un enfoque mejor colocar el bucle de retardo dentro de una subrutina:

```
10 PRINT "PRIMERA SECCION"
20 GOSUB 100
30 PRINT "SEGUNDA SECCION"
40 GOSUB 100
50 PRINT "TERCERA SECCION"
60 GOSUB 100
70 PRINT "CUARTA SECCION"
80 END
100 REM**SUBRUTINA**
110 FOR I=1TO100:NEXT I
120 RETURN
```

Procedimiento de rutina
A diferencia de un diagrama de flujo, este diagrama de estructura refleja la estructura de procedimientos del programa en vez de su flujo de control. Una cápsula indica el comienzo de un bucle REPEAT...UNTIL; los rombos son recuadros de decisión: cuando la condición fracasa, el bucle que lo encierra continúa. Los números de Nivel reflejan la estructura de bloques del programa: todos los comienzos de bucles y las llamadas a procedimientos abren un nuevo bloque del programa y un nivel lógico inferior. Compare esto con el diagrama de la página 867



Janet Barrance

Reemplazando la subrutina por un procedimiento en BASIC BBC se obtiene el siguiente código:

```
 10 PRINT "PRIMERA SECCION"
 20 PROCretardo
 30 PRINT "SEGUNDA SECCION"
 40 PROCretardo
 50 PRINT "TERCERA SECCION"
 60 PROCretardo
 70 PRINT "CUARTA SECCION"
 80 END
100 REM**DEFINIR PROCEDIMIENTO**
110 DEF PROCretardo
120 FOR I=1TO100:NEXT I
130 ENDPROC
```

Existen numerosas similitudes entre la construcción de subrutinas y la construcción de procedimientos; por ejemplo, ambos se codifican después de la sentencia END pero se los puede llamar repetidamente desde dentro del programa principal. La ventaja fundamental del procedimiento es que se lo llama por su nombre en vez de por su número de línea. La DEFinición del PROCedimiento puede comenzar en cualquier lugar después de la sentencia END.

Si deseáramos que el programa que hemos dado como ejemplo fuera capaz de esperar durante períodos diferentes antes de cada sección, entonces se podría utilizar una ventaja más importante de los procedimientos: la capacidad de pasar parámetros a una definición de procedimientos. Vamos a suponer que deseamos que la pausa entre la primera y la segunda sección sea de 100 bucles, la pausa entre la segunda y la tercera sección de 200 bucles y la pausa entre la tercera y la cuarta sección de 175 bucles. En BASIC estándar sería necesario asignarle el valor a I cada vez antes de llamar a la subrutina. Utilizando procedimientos, el valor se puede pasar mediante unos paréntesis al final de la sentencia de llamada:

```
 10 PRINT "PRIMERA SECCION"
 20 PROCretardo(100)
 30 PRINT "SEGUNDA SECCION"
 40 PROCretardo(200)
 50 PRINT "TERCERA SECCION"
 60 PROCretardo(175)
 70 PRINT "CUARTA SECCION"
 80 END
100 REM**DEFINIR PROCEDIMIENTO**
110 DEF PROCretardo(N)
120 FOR I=1TON:NEXT I
130 ENDPROC
```

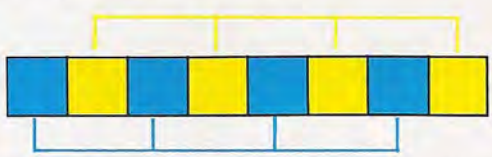
Dentro de un procedimiento se pueden pasar varios parámetros, debiendo quedar separados mediante comas. También se pueden utilizar como parámetros nombres de variables para pasar el valor de la variable en el momento en que se llama al procedimiento.

El juego que construiremos es para un jugador y utiliza las teclas de control del cursor del teclado para desplazar un detector de minas por un campo de minas. Se consiguen puntos por cada mina que se manipula con éxito. Existen, no obstante, varias cosas que dificultan su avance a través del campo de minas. La preocupación fundamental es su ayudante, que representa todos los movimientos suyos. Mientras usted va por ahí ocupándose de las minas, debe asegurarse de que él no pise una. A usted también le está disparando un francotirador y existe un límite de tiempo de dos minutos para el juego. En la versión final, tendrá cuatro ayudantes voluntarios por juego y una opción de factor de dificultad de 0, el más fácil, a 9, el más difícil.

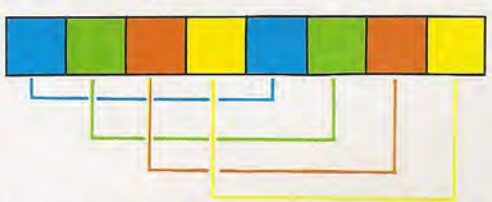
Dado que el BASIC BBC emplea procedimientos, los diagramas de flujo no son de gran utilidad. En cambio, se puede utilizar un *diagrama de estructura* para ilustrar los procedimientos que se requieren y cómo se acoplan entre sí para conformar el programa final. En nuestro diagrama los bucles REPEAT ...UNTIL se indican con forma de "salchicha". Los recuadros de decisión son los recuadros más normales, en forma de rombo, pero con los extremos superior e inferior recortados para ahorrar espacio. Se debe hacer hincapié en el hecho de que el diagrama no se trazó en su totalidad antes de que se comenzara la programación, sino que se desarrolló a partir de una serie de refinamientos sucesivos.

Antes de que podamos comenzar a definir rutinas para colocar objetos en la pantalla, debemos decidir qué modalidad de visualización en pantalla vamos a utilizar. Son tres los factores fundamentales a tener en cuenta: resolución, color y memoria. En términos generales, cuanta más información deba retener la pantalla, más memoria exigirá. De modo que resoluciones más altas y mayor número de colores significan más memoria. Si el programa es corto, esto podría no tener ninguna importancia, pero el programa que estamos diseñando es bastante extenso. También son necesarios algunos colores diferentes para distinguir las minas y el detector-ayudante y para conseguir que el juego sea visualmente atractivo. En el Modelo B se nos ofrecen dos modalidades de resolución media. La modalidad 2 nos proporciona 16 colores con los cuales jugar, mientras que la 5 sólo nos ofrece 4. Analizando cómo el BBC interpreta los patrones de bits en cada modalidad podemos ver por qué la 5 utiliza sustancialmente menos memoria que la 2.

A diferencia de algunos micros, el BBC retiene la información de color y pixel encendido o apagado en un byte para cada pequeña zona de la pantalla. En la modalidad 2 se requieren cuatro bits para representar los 16 colores posibles. Por consiguiente, un byte sólo puede retener información acerca del color de dos pixels. Los bits del byte se disponen de la siguiente manera:

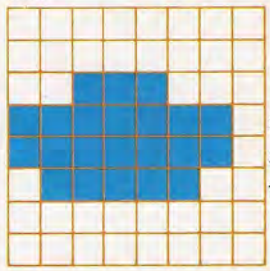


Como la modalidad 5 está limitada a cuatro colores, sólo se necesitan dos bits para retener la información de color. Así, un byte puede representar cuatro pixels, como vemos en la ilustración:

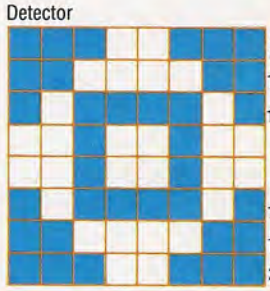


Ambas modalidades poseen una resolución de 160 por 256 pixels, por lo que el número de bytes requeridos para la memoria de pantalla de la modali-

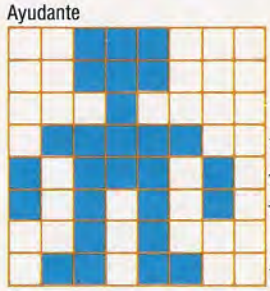
Mina



Detector



Ayudante



Puntos de imágenes
Se describen los caracteres definidos por el usuario que representan la mina, el detector y el ayudante. La información de los pixels está dividida en ocho bytes, siendo cada byte la "imagen" binaria de una fila del dibujo

dad 2 es (160×256)/2=20 Kbytes, mientras que la modalidad 5 necesita (160×256)/4=10 Kbytes de memoria. Eligiendo la modalidad 5 nos concedemos a nosotros mismos 10 Kbytes extras de memoria para nuestro programa. Una ventaja adicional es que las máquinas Modelo A no disponen de la modalidad 2, ¡pero sí de la modalidad 5!

En el BBC la construcción de caracteres definidos por el usuario es muy simple. Cada carácter se compone de ocho números que representan los equivalentes decimales de cada fila. Vemos aquí los diseños de caracteres para las minas, el detector y el ayudante.

La instrucción VDU23 permite que el programador defina caracteres con códigos ASCII del 226 al 255. El segundo número de la instrucción VDU indica el código que se desea asignar a la forma definida por los ocho últimos números. Por ejemplo:

```
VDU23,224,0,0,56,254,254,124,0,0
```

define CHR$(224) como la forma de la mina. Para imprimir este carácter generamos la instrucción PRINT CHR$(224). El siguiente procedimiento define los tres caracteres a utilizar en el juego:

```
2380 DEF PROCdefinir-caracteres
2390 REM**MINAS**
2400 VDU23,224,0,0,56,254,254,124,0,0
2410 REM**DETECTOR DE MINAS**
2420 VDU23,225,231,195,189,36,36,189,195,231
2430 REM**AYUDANTE**
2440 VDU23,226,56,56,16,124,186,170,40,108
2450 ENDPROC
```

Trazado de la pantalla

Después de haber definido las formas, las podemos imprimir en la pantalla. El modo más fácil de hacerlo consiste en utilizar la instrucción PRINT TAB(X,Y). En la modalidad 5 hay 32 filas, cada una de ellas de 20 caracteres. Esto significa que X va de 0 a 19 e Y de 0 a 31. El campo de minas ha de ocupar la superficie que se indica en el diagrama.

Para disponer las minas al azar en la zona dada podemos valernos de la instrucción RND(N). Si N es un número entero, entonces RND(N) devuelve un número entero entre 1 y N. Se debe elegir la coordenada horizontal de cada mina de modo que caiga entre 2 y 17. RND(16) proporciona números del 1 al 16, de modo que RND(16)+1 seleccionará coordenadas dentro de la superficie del campo. Este procedimiento colocará tantas minas como se especifique en el valor que se le pasa como argumento:

```
2560 DEF PROCcolocar-minas(numero-minas)
2570 REM**CAMBIAR COLOR 2 A VERDE**
2580 VDU19,2,2,0,0,0
2590 FOR I=1 TO numero-minas
2600 PRINTTAB(RND(16)+1,RND(25));CHR$(224)
2610 NEXT I
2620 ENDPROC
```

En la modalidad 5 estamos limitados a cuatro colores y normalmente éstos son negro, rojo, amarillo y blanco, que corresponden a los números de color 0, 1, 2 y 3. Sin embargo, no tenemos que utilizar necesariamente estos colores y los podemos cambiar utilizando la instrucción VDU19. Los números del 0 al 3 se conocen como los colores de fondo *lógicos*. Cada uno de los 16 colores del BBC posee un número que no guarda ninguna relación con la modalidad que se esté empleando. Éstos se denominan números de color *reales* y en la guía para el usuario aparece una tabla de los mismos. A cualquiera de los cuatro colores lógicos se le puede asignar uno de los 16 colores reales. Para nuestro juego queremos que las minas sean verdes (número de color real 2). No deseamos el amarillo, que suele ofrecerse como el color lógico 2. La instrucción VDU19 hace esto.

El detector y el ayudante ocupan sus posiciones iniciales en las esquinas inferior izquierda y superior derecha, respectivamente. Como probablemente desearemos volver a posicionar después al detector y al ayudante, el procedimiento para posicionarlos utilizará variables (xdet, ydet) para las coordenadas del detector y para las coordenadas (xhom, yhom) del ayudante.

```
2830 DEF PROCsituar-sujetos
2840 COLOUR 1
2850 PRINTTAB(xdet,ydet);CHR$(225)
2860 PRINTTAB(xhom,yhom);CHR$(226)
2870 COLOUR 2
2880 ENDPROC
```

Las instrucciones COLOUR al principio y al final del procedimiento seleccionan el color lógico que tendrá el futuro texto. COLOUR 1 selecciona el color lógico 1 (rojo) para imprimir el detector y el ayudante, y COLOUR 2 restaura el color verde para la futura impresión. Sin embargo, antes de que se pueda aplicar este procedimiento, se deben asignar valores a xdet, ydet, xhom e yhom. Ello se realiza en otro procedimiento, junto con la inicialización de algunas variables que se usarán en otros sitios.

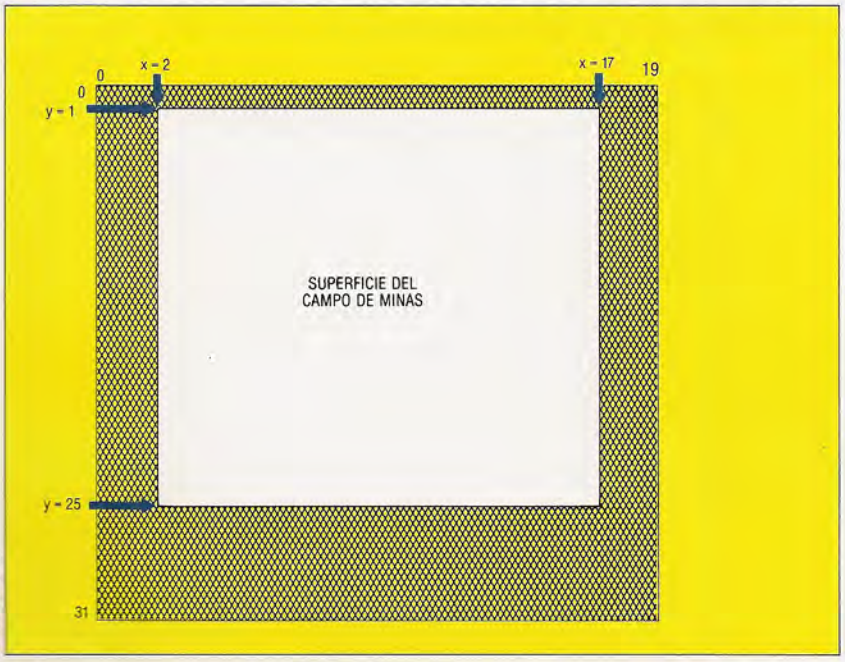
```
2320 DEF PROCinicializar-variables
2330 xdet=2:ydet=25:xhom=17:yhom=1
2340 xcomienzo=120:xfinal=1144
2350 cero$="000000"
2360 ENDPROC
```

Ahora todos estos procedimientos se pueden controlar mediante un breve programa de llamada:

```
 5 MODE 5
10 COLOUR 2
20 PROCinicializar-variables
30 PROCdefinir-caracteres
40 PROCcolocar-minas(40)
50 PROCsituar-sujetos
60 END
```

En el próximo capítulo nos ocuparemos de sincronizar y controlar el movimiento desde el teclado.

Zona de peligro
El campo de minas ocupa 20 filas de 16 caracteres, y las minas se "siembran" al azar durante el procedimiento de preparación



Janet Barranco

Un piloto temerario

"Jet Pac" ha establecido un nuevo estándar de calidad en los gráficos del Spectrum, convirtiéndose en un best-seller

Los juegos recreativos en los que se trata de "aniquilar al extraterrestre" con frecuencia se ven notablemente limitados por su enfoque simplista e infantil. El diseño de un ejemplo de éxito requiere considerable destreza en cuanto a programación. *Jet Pac* es un buen ejemplo de ello.

Como suele suceder con tanta frecuencia, el escenario dibujado en la cartulina de la cassette parece más complejo de lo que es en realidad. Como piloto jefe de pruebas de la Empresa de Transportes Interestelar Acme, su tarea consiste en viajar a través de la galaxia ensamblando naves espaciales en planetas seleccionados, mientras recoge todo el oro y las piedras preciosas que puedan caer en sus manos. El ensamblaje de los cohetes se ve facilitado en gran medida por un Hydrovac Jet Pac, capaz de levantar casi todo, y que le permite manipular componentes con suma facilidad. Usted también está convenientemente armado con *quad photon laser phasers*, que se utilizan para destruir a todo extraterrestre que pueda ser tan obcecado como para quejarse de que se saquee su planeta.

La descripción del juego puede que evoque visiones de un viaje en glorioso Technicolor a través de las variadas ecologías de distintos planetas; pero, como siempre, la verdad es más prosaica. Los planetas que se visitan son virtualmente idénticos, tal es así que el héroe-piloto de pruebas probablemente experimentará un fuerte sentido de *déjà vu* a los pocos aterrizajes. Pero los extraterrestres compensan esto. Cada planeta está habitado sólo por una única especie; de hecho, habría muy poco sitio para cualquier otra especie, ya que da la impresión de que todos los extraterrestres se reproducen como conejos. Estas especies varían en su forma, desde platillos voladores hasta balones que rebotan, pero todas ellas poseen una cosa en común: el contacto físico con cualquiera de ellas significa la muerte.

Desperdigados por la superficie de cada planeta hay tres componentes de una nave espacial; éstos se han de ensamblar para que usted pueda dirigirse hacia el siguiente puerto de llamada. Para esta tarea no se necesitan destornilladores ni llaves inglesas: el usuario simplemente arroja los componentes en la base del cohete y ve cómo ante sus propios ojos la nave espacial se arma sola.

Lo que hace que este juego sea tan divertido son los extraterrestres. Su número es elevadísimo y al principio da la impresión de que avanzan amenazadora e infaliblemente hacia usted; al cabo de unos instantes comprende que, en realidad, están siguiendo caminos preestablecidos que parten desde puntos de partida aleatorios. La primera oleada de atacantes desciende lentamente, dándole tiempo para aniquilarlos con su láser letal. La segunda oleada es de pelotas rebotadoras, que rebotan a través de la pantalla entre los salientes de las rocas y el suelo. Esta mezcla de caminos preestablecidos y movimientos al azar proporciona la combinación precisa, exigiendo destreza y reflejos rápidos.

Esta clase de juego a menudo se suele arruinar a causa de una selección inadecuada de teclas de control. Aquí éstas se han elegido con notable sensatez. Para el movimiento hacia derecha e izquierda, siendo útiles tanto para jugadores diestros como zurdos, se utilizan dos teclas de la fila inferior del teclado. Cualquiera de las teclas de la segunda fila sirve para disparar los láseres, que se pueden dejar disparando de forma permanente. La fila de encima activa los motores de reacción que hacen que se ascienda, y la fila superior permite que usted flote en el aire. Un original detalle es que si no se pulsa ninguna tecla el piloto irá cayendo sobre la superficie del planeta por la fuerza de gravedad.

Los gráficos son excelentes. El dibujo del piloto de pruebas es maravilloso, al igual que el Hydrovac Jet Pac, que despide convincentes volutas de humo cuando se activa "avanzar" o "flotar". El láser atraviesa el cielo con líneas multicolores y los extraterrestres, cuando son alcanzados, explotan convirtiéndose en más nubes de humo.

El juego es bastante similar tanto en el Spectrum como en el Vic-20, si bien el formato de pantalla de este último tiene el efecto de que el piloto de pruebas resulta más bien voluminoso. En ambas máquinas, *Jet Pac* es un juego sumamente atractivo.

Jet Pac: Para el Spectrum de 16 o 48 K y el Vic-20 de 8 K ampliado

Editado por: Ashby Computers and Graphics Ltd.

Autores: Ultimate, Play The Game

Palancas de mando: Kempston Competition-Pro (Spectrum); "la mayoría de las compatibles con Commodore" (Vic-20)

Formato: Cassette

Jet Pac en el Spectrum

Ensamblaje espacial

La tarea del piloto de pruebas de *Jet Pac*, que consiste en ensamblar naves espaciales en diversos planetas, resulta peligrosa debido a la presencia de los decididos extraterrestres, pero el oro y las joyas que llueven del cielo constituyen una buena compensación

Liz Heaney

El cuadro y el marco

Tratamos aquí de un programa en lenguaje máquina para crear "ventanas" o marcos en el Spectrum donde son posibles los desplazamientos visuales

Ventana abierta
La palabra de salutación en inglés "HELLO" es desplazada o bien horizontalmente o bien verticalmente dentro de la ventana de pantalla. Siempre un pixel cada vez

Este programa en lenguaje máquina permite definir "ventanas" rectangulares sobre una pantalla gestionada por el Spectrum, así como desplazarlas (*scrolling*) a derecha e izquierda y arriba o abajo. Tales "ventanas" pueden tener sobre la pantalla toda clase de tamaños y estar situadas en cualquier lugar; no tienen por qué reducirse a cuadrados de caracteres de 8×8 pixels.

Las tablas que emplea nuestro programa en lenguaje máquina comienzan en la dirección $B004 (en decimal,45060) y se destinan al tratamiento de los parámetros de las referidas ventanas y al almacenamiento temporal de datos. Las direcciones $B000 y $B001 (en decimal, 45056 y 45057) contienen la dirección de la tabla de una ventana determinada. Cada tabla de ventana comprende 11 bytes, por lo que si necesitamos más de una ventana, la tabla de la segunda comenzará en $B00F (en decimal, 45071), la de la tercera en la dirección $B01A (en decimal, 45082), etc.

El programa de demostración en BASIC sólo emplea una ventana. Los detalles de ésta se colocan (POKE) dentro de la memoria en las líneas 180 a 230, y la rutina de inicialización de la ventana es llamada en la línea 240. Cualquier otra ventana adicional debe ser definida de este modo antes de poderse emplear. El cambio de ventanas se realiza colocando (POKE) en la memoria la dirección de la tabla de la nueva ventana, concretamente en las posiciones WT y WT+1. Las direcciones de los desplazamientos son dadas mediante la colocación (POKE) de los valores en la posición WNDWTB+DIR (*window table+direction*: tabla de ventana+dirección). Recuerde: para el desplazamiento hacia la izquierda utilice POKE 0; 1 para un desplazamiento hacia la derecha, 2 arriba y 3 abajo.

El programa en assembly comienza con la definición de constantes. PIXADR (*pixel address*: dirección del pixel) es una subrutina de la ROM del Spectrum que calcula la dirección del byte de pantalla, y el número del bit dentro de ese byte correspondiente a un punto de la pantalla definido por sus coordenadas de PLOT. La rutina toma la coordenada *y* que encuentra en el registro B y la abscisa *x* en el registro C para devolver después la dirección de la pantalla que está en el registro doble HL y la posición del bit en el registro A.

La rutina INITW lo primero que hace es comprobar que las coordenadas correspondientes a la esquina inferior derecha de la ventana se hallen por debajo y a la derecha de las coordenadas de la esquina superior izquierda. Igualmente comprueba que los bordes o márgenes izquierdo y derecho no estén en el mismo byte de la memoria de pantalla. Esto asegurará que el ancho de la ventana sea de un cuadrado de carácter, pues se precisaría código adicional para una ventana más estrecha.

Los errores de inicialización de la ventana se visualizan gracias a la rutina de mensajes de error contenida en la ROM. La instrucción RST 8 (línea 2110 del listado en assembly) llama a la rutina de la ROM y vuelve a la modalidad de instrucciones en BASIC, mientras que DEFB 25 de la línea 2120 da el mensaje "Error en parámetro Q".

La última parte de INITW se destina a calcular LFTMSK y RTMASK (*left, right*: izquierda, derecha), empleadas cuando se desplaza el byte de pantalla en los bordes de la ventana, donde parte de ese byte cae dentro de la ventana y parte fuera. Los bits individuales de las máscaras que correspondan a bits de pantalla fuera de los márgenes de la ventana se ponen a 1, y los de dentro a 0.

El programa de desplazamiento propiamente comienza donde está la etiqueta SCROLL. Lo que hace es comprobar la dirección del desplazamiento y llamar o bien a HORIZ para desplazamientos horizontales, o bien a VERT para los verticales.

Los desplazamientos a la izquierda y a la derecha operan de modo similar, por lo que en vez de escribir dos rutinas las hemos mezclado en una sola. El código adecuado a cada dirección del desplazamiento se deduce del bit 0 del byte de dirección. Para darnos cuenta del modo como funciona HORIZ vamos a detenernos en el desplazamiento hacia la izquierda.

Los dos desplazamientos, a izquierda y a derecha, comienzan por la fila superior de pixels de la ventana y de allí van bajando; por tanto HORIZ primero copia la ordenada *y* de la fila superior en una posición de memoria temporal que contiene la fila en curso. Si el desplazamiento es a la izquierda, debemos comenzar por el extremo derecho de cada fila de pixels de la ventana e ir hacia la izquierda. Para preparar esto se copian RMASK y LMASK a MASK1 y MASK2 respectivamente, se calcula y se almacena en el registro doble DE la dirección del byte de pantalla perteneciente al extremo izquierdo de la fila de pixels en curso, y por último se calcula y se almacena en el registro doble HL la dirección del byte de pantalla correspondiente al extremo derecho de la fila de pixels en cuestión. Entonces es cuando se llama a la subrutina HLNSCR para desplazar la fila de pixels. Comprueba si se ha alcanzado la fila inferior y última de la ventana, y en caso contrario toma la fila siguiente de pixels al tiempo que salta a HORIZ3 para seguir desplazando.

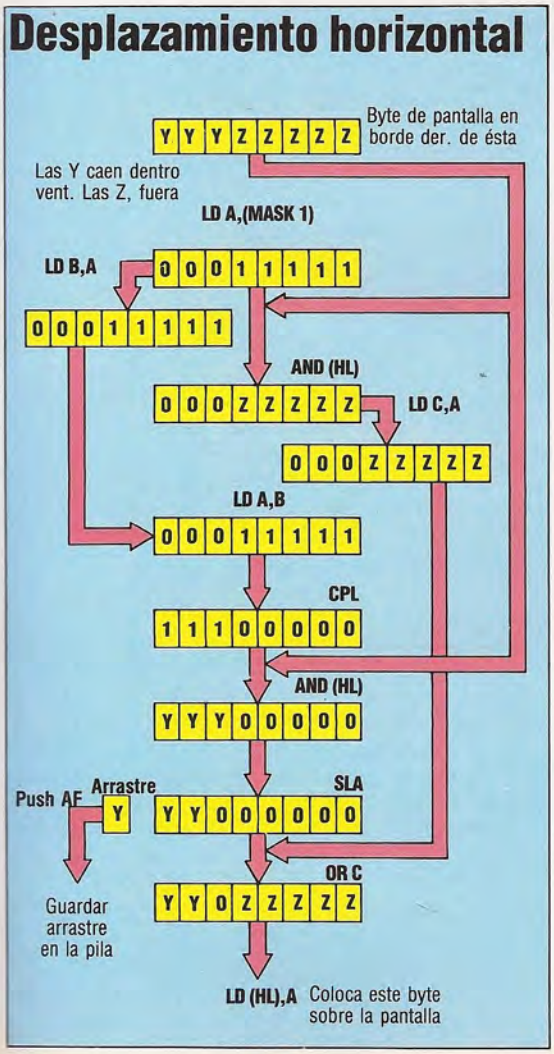
La subrutina HLNSCR comienza por un extremo de la fila de pixels con un byte que puede tener parte de sus bits dentro de la ventana y parte fuera de ella. Sigue con los bytes que caen por completo dentro de la ventana, hasta llegar al otro borde de ésta, donde encontrará de nuevo un byte cuyos bits puede que caigan parte fuera y parte dentro de ella. Creemos que se puede entender mejor todo esto en

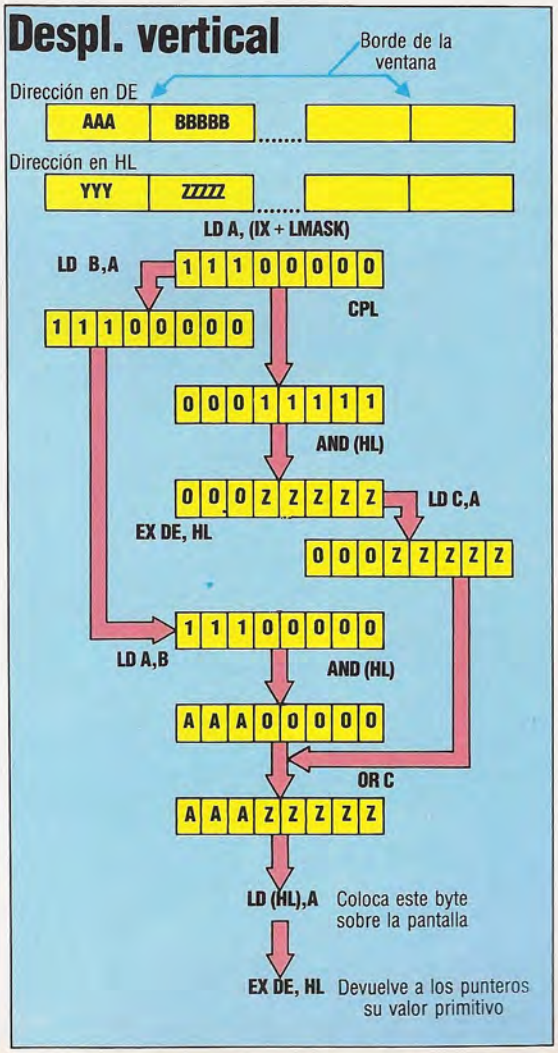
la ilustración adjunta. El fragmento de HLNSCR que comienza en NEXT desplaza los bytes que están dentro del área de la ventana. El bit descartado del byte anterior y colocado en el indicador o flag de arrastre se guardó en la pila mediante PUSH AF para devolverlo a dicho flag con POP AF. Para desplazamientos a la izquierda la rutina escoge la instrucción RL (HL) con la que el byte en pantalla se desplaza a la izquierda, quedando el bit de arrastre ahora en el último lugar de la izquierda del byte y el bit del extremo derecho de éste ocupando el flag de arrastre. Como PUSH AF conserva el flag de arrastre, dicho bit se puede llevar al byte siguiente. La rutina comprueba el final de la fila comparando los registros L y E, pues sabemos que el byte *hi* de una dirección de pantalla es el mismo para todos los bytes de la misma fila.

Las rutinas verticales se combinan de igual manera. Si examinamos lo que sucede cuando VERT está provocando un desplazamiento vertical hacia arriba, veremos que la rutina primero almacena la coordenada *y* de la fila superior en el lugar reservado a la fila en curso. Calcula después las direcciones de pantalla de los que en dicha fila corresponden a los bordes izquierdo y derecho de la ventana y determina la longitud de la fila. La rutina coloca ahora la dirección del byte del borde izquierdo en DE, y la dirección del byte que corresponde a aquél en la fila inmediatamente inferior la pone en HL antes de llamar a la subrutina VLNSCR para que realice el desplazamiento. Después VERT comprueba si ha alcanzado la parte inferior de la ventana. Si no es así, baja una línea antes de volver a VERT5 para desplazar otra línea de pixels. Si la ha alcanzado, el fragmento de la rutina que comienza en CLREDG llena de ceros la fila de pixels inferior para que esa fila no aparezca en la pantalla.

La subrutina VLNSCR trata por separado los bytes de los márgenes, tal como vimos con HLNSCR. (Véase la ilustración de la derecha.) Para mover la parte central de la fila de pixels, la rutina incrementa HL y DE de modo que apunten el primer byte interior de la línea en curso y su correspondiente en la línea superior. Seguidamente calcula la longitud de la parte central (o sea, los bytes que pertenecen por entero al área de la ventana), carga esta información en BC y emplea la instrucción de movimiento en bloque LDIR para que toda esa parte central de la fila se desplace una línea hacia arriba.

Todas estas rutinas de desplazamiento son algo lentas. Esto se debe en parte a que están combinadas las de desplazamiento a izquierda y derecha y las de desplazamiento arriba o abajo, de forma que el programa debe realizar frecuentes comprobaciones con las que decide la parte de código máquina que ha de usar. En parte también se explica porque los bytes que están en los bordes izquierdo y derecho necesitan un tratamiento especial si se encuentran a caballo sobre esos bordes de la ventana.





Línea abajo
El desplazamiento horizontal presenta unos problemas determinados en los bordes de la ventana. Los bytes de estos márgenes deben ser "enmascarados" para separar los bits de pixels que caen dentro y fuera de la ventana, y además estos bytes deben desplazarse. Ambos procesos emplean el AND y el OR lógico y la instrucción SHIFT además de servirse de la pila para guardar el registro indicador de estado (PSR).
El desplazamiento vertical se simplifica mucho con el mapa de memoria de pantalla del Spectrum (véase p. 838). También se superpone una máscara al byte de pantalla para aislar los pixels de dentro y fuera de la ventana, además de la instrucción EX para intercambiar los contenidos de los registros DE y HL, que guardan punteros de direcciones de pantalla

Ventanas en el Spectrum

Instrucciones de operatoria

1) Entre el programa de demostración en BASIC
2) Haga SAVE "SCROLL" LINE 5
3) Digite el programa cargador de código máquina, y ejecútelo (RUN)
4) Haga SAVE "SCROLLMC" CODE 45312,410 directamente en cinta después del programa en BASIC
5) Rebobine la cinta y haga LOAD "SCROLL"

Listado assembly

```
 10 PIXADR EQU  #22AA
 20 WT     EQU  #B000
 30 MASK1  EQU  #B002
 40 MASK2  EQU  #B003
 50 WNDWTB EQU  #B004
 60 LEFTX  EQU  0
 70 TOPY   EQU  1
 80 RIGHTX EQU  2
 90 BOTY   EQU  3
100 LBIT   EQU  4
110 RBIT   EQU  5
120 CURNTY EQU  6
130 DIR    EQU  7
140 LMASK  EQU  8
150 RMASK  EQU  9
160 LENGTH EQU  10
170        ORG  #B100
180 INIT   LD   HL,(WT)
190        PUSH HL
200        POP  IX
210        CALL INITW
220        RET
230 SCROLL LD   HL,(WT)
240        PUSH HL
250        POP  IX
260        BIT  1,(IX+DIR)
270        PUSH AF
280        CALL Z,HORIZ
290        POP  AF
300        CALL NZ,VERT
310        RET
320 HORIZ  LD   A,(IX+TOPY)
330        LD   (IX+CURNTY),A
340        BIT  0,(IX+DIR)
350        JR   Z,HORIZ1
360        LD   B,(IX+LMASK)
370        LD   A,(IX+RMASK)
380        JR   HORIZ2
390 HORIZ1 LD   B,(IX+RMASK)
400        LD   A,(IX+LMASK)
410 HORIZ2 LD   (MASK2),A
420        LD   A,B
430        LD   (MASK1),A
440 HORIZ3 LD   C,(IX+LEFTX)
450        LD   B,(IX+CURNTY)
460        CALL PIXADR
470        EX   DE,HL
480        LD   C,(IX+RIGHTX)
490        LD   B,(IX+CURNTY)
500        CALL PIXADR
510        BIT  0,(IX+DIR)
520        JR   Z,HORIZ4
530        EX   DE,HL
540 HORIZ4 CALL HLNSCR
550        LD   A,(IX+BOTY)
560        CP   (IX+CURNTY)
570        RET  Z
580        DEC  (IX+CURNTY)
590        JR   HORIZ3
600 HLNSCR LD   A,(MASK1)
610        LD   B,A
620        AND  (HL)
630        LD   C,A
640        LD   A,B
650        CPL
660        AND  (HL)
670        BIT  0,(IX+DIR)
680        JR   Z,HLN1
690        SRA  A
700        JR   HLN2
710 HLN1   SLA  A
720 HLN2   PUSH AF
730        OR   C
740        LD   (HL),A
750 NEXT   BIT  0,(IX+DIR)
760        JR   Z,HLN3
770        INC  HL
780        JR   HLN4
790 HLN3   DEC  HL
800 HLN4   LD   A,L
810        CP   E
820        JR   Z,LAST
830        POP  AF
840        BIT  0,(IX+DIR)
850        JR   Z,HLN5
860        RR   (HL)
870        JR   HLN6
880 HLN5   RL   (HL)
890 HLN6   PUSH AF
900        JR   NEXT
910 LAST   LD   C,(HL)
920        LD   A,(MASK2)
930        AND  C
940        LD   B,A
950        POP  AF
960        BIT  0,(IX+DIR)
970        JR   Z,HLN7
980        RR   C
 990        JR   HLN8
1000 HLN7  RL   C
1010 HLN8  LD   A,(MASK2)
1020       CPL
1030       AND  C
1040       OR   B
1050       LD   (HL),A
1060       RET
1070 VERT  LD   C,(IX+LEFTX)
1080       BIT  0,(IX+DIR)
1090       JR   Z,VERT1
1100       LD   B,(IX+BOTY)
1110       JR   VERT2
1120 VERT1 LD   B,(IX+TOPY)
1130 VERT2 LD   (IX+CURNTY),B
1140       CALL PIXADR
1150       PUSH HL
1160       PUSH HL
1170       LD   C,(IX+RIGHTX)
1180       LD   B,(IX+CURNTY)
1190       CALL PIXADR
1200       POP  DE
1210       AND  A
1220       SBC  HL,DE
1230       LD   A,L
1240       DEC  A
1250       LD   (IX+LENGTH),A
1260 VERT3 BIT  0,(IX+DIR)
1270       JR   Z,VERT4
1280       INC  (IX+CURNTY)
1290       JR   VERT5
1300 VERT4 DEC  (IX+CURNTY)
1310 VERT5 LD   C,(IX+LEFTX)
1320       LD   B,(IX+CURNTY)
1330       CALL PIXADR
1340       POP  DE
1350       PUSH HL
1360       CALL VLNSCR
1370       LD   A,(IX+CURNTY)
1380       BIT  0,(IX+DIR)
1390       JR   Z,VERT6
1400       CP   (IX+TOPY)
1410       JR   VERT7
1420 VERT6 CP   (IX+BOTY)
1430 VERT7 JR   NZ,VERT3
1440 CLREDG POP  HL
1450       LD   A,(IX+LMASK)
1460       AND  (HL)
1470       LD   (HL),A
1480       LD   B,(IX+LENGTH)
1490       LD   A,0
1500 CLR1  INC  HL
1510       LD   (HL),A
1520       DJNZ CLR1
1530       INC  HL
1540       LD   A,(IX+RMASK)
1550       AND  (HL)
1560       LD   (HL),A
1570       RET
1580 VLNSCR LD  A,(IX+LMASK)
1590       CALL ENDBYT
1600       INC  HL
1610       INC  DE
1620       LD   B,0
1630       LD   C,(IX+LENGTH)
1640       LDIR
1650       LD   A,(IX+RMASK)
1660 ENDBYT LD  B,A
1670       CPL
1680       AND  (HL)
1690       LD   C,A
1700       EX   DE,HL
1710       LD   A,B
1720       AND  (HL)
1730       OR   C
1740       LD   (HL),A
1750       EX   DE,HL
1760       RET
1770 INITW LD   A,(IX+RIGHTX)
1780       CP   (IX+LEFTX)
1790       JR   Z,ERROR
1800       JR   C,ERROR
1810       LD   A,(IX+TOPY)
1820       CP   (IX+BOTY)
1830       JR   Z,ERROR
1840       JR   C,ERROR
1850       LD   C,(IX+LEFTX)
1860       LD   B,(IX+TOPY)
1870       CALL PIXADR
1880       PUSH HL
1890       LD   (IX+LBIT),A
1900       LD   C,(IX+RIGHTX)
1910       LD   B,(IX+TOPY)
1920       CALL PIXADR
1930       LD   (IX+RBIT),A
1940       POP  BC
1950       LD   A,C
1960       CP   L
1970       JR   Z,ERROR
1980 LFTMSK LD  B,(IX+LBIT)
1990       LD   A,0
2000 L1    SCF
2010       RRA
2020       DJNZ L1
2030       LD   (IX+LMASK),A
2040 RTMASK LD  B,(IX+RBIT)
2050       LD   A,255
2060 L2    AND  A
2070       RRA
2080       DJNZ L2
2090       LD   (IX+RMASK),A
2100       RET
2110 ERROR RST  8
2120       DEFB 25
```

Programa de demostración en BASIC

```
  5 CLEAR 32767
 10 LOAD ""CODE
 20 LET WT=45056
 30 LET WINDOWTABLE=45060: REM B004 HEX
 40 LET LEFTX=0
 50 LET TOPY=1
 60 LET RIGHTX=2
 70 LET BOTY=3
 75 LET DIR=7
 80 LET SCROLL=45322
 90 LET INIT=45312
100 BORDER 6
110 PAPER 4: INK 2
120 CLS
180 POKE WT,4: POKE WT+1,176
190 REM WT & WT+1 NOW CONTAIN ADDRESS 45060
IN LO,HI FORMAT
200 POKE WINDOWTABLE+LEFTX,5
210 POKE WINDOWTABLE+TOPY,80
220 POKE WINDOWTABLE+RIGHTX,250
230 POKE WINDOWTABLE+BOTY,35
240 RANDOMIZE USR INIT
250 FOR Y=0 TO 175
260 PLOT 0,Y
270 DRAW 255,0
280 NEXT Y
290 POKE WINDOWTABLE+DIR,2
300 FOR Y=0 TO 45
310 RANDOMIZE USR SCROLL
320 NEXT Y
400 PRINT AT 12,20;"HELLO";
410 POKE WINDOWTABLE+DIR,0
420 FOR I=1 TO 130
430 RANDOMIZE USR SCROLL
440 NEXT I
470 POKE WINDOWTABLE+DIR,3
480 FOR I=1 TO 35
490 RANDOMIZE USR SCROLL
500 NEXT I
510 POKE WINDOWTABLE+DIR,1
520 FOR I=1 TO 130
530 RANDOMIZE USR SCROLL
540 NEXT I
550 POKE WINDOWTABLE+DIR,2
560 FOR I=1 TO 35
570 RANDOMIZE USR SCROLL
580 NEXT I
590 GO TO 410
999 STOP
```

Cargador de código máquina

```
 100>LET a=45312
 110 FOR l=1000 TO 1500 STEP 10
 120 LET s=0
 130 READ b
 140 POKE a,b
 150 LET s=s+b
 160 NEXT a
 170 READ b
 180 IF s<>b THEN  PRINT "ERROR IN LINE ";L: STOP
 190 NEXT l
1000 DATA 42,0,176,229,221,225,205,69,1167
1010 DATA 178,201,42,0,176,229,221,225,1272
1020 DATA 221,203,7,78,245,204,29,177,1164
1030 DATA 241,196,184,177,201,221,126,1,1347
1040 DATA 221,119,6,221,203,7,70,40,887
1050 DATA 8,221,70,8,221,126,9,24,687
1060 DATA 6,221,70,9,221,126,8,50,711
1070 DATA 3,176,120,50,2,176,221,78,826
1080 DATA 0,221,70,6,205,170,34,235,941
1090 DATA 221,78,2,221,70,6,205,170,973
1100 DATA 34,221,203,7,70,40,1,235,811
1110 DATA 205,103,177,221,126,3,221,190,1246
1120 DATA 6,200,221,53,6,24,215,58,783
1130 DATA 2,176,71,166,79,120,47,166,827
1140 DATA 221,203,7,70,40,4,203,47,795
1150 DATA 24,2,203,39,245,177,119,221,1030
1160 DATA 203,7,70,40,3,35,24,1,383
1170 DATA 43,125,187,40,16,241,221,203,1076
1180 DATA 7,70,40,4,203,30,24,2,380
1190 DATA 203,22,245,24,226,78,58,3,859
1200 DATA 176,161,71,241,221,203,7,70,1150
1210 DATA 40,4,203,25,24,2,203,17,518
1220 DATA 58,3,176,47,161,176,119,201,941
1230 DATA 221,78,0,221,203,7,70,40,840
1240 DATA 5,221,70,3,24,3,221,70,617
1250 DATA 1,221,112,6,205,170,34,229,978
1260 DATA 229,221,78,2,221,70,6,205,1032
1270 DATA 170,34,209,167,237,82,125,61,1085
1280 DATA 221,119,10,221,203,7,70,40,891
1290 DATA 5,221,52,6,24,3,221,53,585
1300 DATA 6,221,78,0,221,70,6,205,807
1310 DATA 170,34,209,229,205,40,178,221,1286
1320 DATA 126,6,221,203,7,70,40,5,678
1330 DATA 221,190,1,24,3,221,190,3,853
1340 DATA 32,209,225,221,126,8,166,119,1106
1350 DATA 221,70,10,62,0,35,119,16,533
1360 DATA 252,35,221,126,9,166,119,201,1129
1370 DATA 221,126,8,205,58,178,35,19,850
1380 DATA 6,0,221,78,10,237,176,221,949
1390 DATA 126,9,71,47,166,79,235,120,853
1400 DATA 166,177,119,235,201,221,126,2,1247
1410 DATA 221,190,0,40,67,56,65,221,860
1420 DATA 126,1,221,190,3,40,57,56,694
1430 DATA 55,221,78,0,221,70,1,205,851
1440 DATA 170,34,229,221,119,4,221,78,1076
1450 DATA 2,221,70,1,205,170,34,221,924
1460 DATA 119,5,193,121,189,40,25,221,913
1470 DATA 70,4,62,0,55,31,16,252,490
1480 DATA 221,119,8,221,70,5,62,255,961
1490 DATA 167,31,16,252,221,119,9,201,1016
1500 DATA 207,25,0,0,0,0,0,0,232
```

Ordenando los naipes

El juego "Invertir" tiene como objetivo acomodar una lista de números por orden ascendente con el menor número de movimientos

El programa genera al azar una lista de números para clasificar. Sólo se puede cambiar el orden de los números invirtiendo grupos específicos dentro de la lista. Por ejemplo, si el ordenador genera la siguiente lista al azar en respuesta a la solicitud de nueve números hecha por el jugador:

2 8 4 7 1 5 6 9 3

y el jugador especifica luego "Invertir? 5", se invertirán los primeros 5 números y la lista quedará en:

1 7 4 8 2 5 6 9 3

Resolver un puzzle como éste no le debería llevar ni mucho tiempo ni mucho esfuerzo, y en principio parece tarea fácil hallar un algoritmo que lo resuelva fácilmente. Sin embargo, en la práctica es difícil definir uno que sea realmente bueno. Supongamos que en la lista hay n números. El algoritmo más obvio es éste:

- Hallar el número mayor de la lista e invertir todos los números hasta su posición. El número mayor está ahora en el extremo izquierdo de la lista.
- Invertir todos los n números, de modo que el número mayor quede en su posición deseada, en el extremo derecho de la lista. Esto sólo ha llevado dos inversiones.
- Hallar el segundo número mayor y volver a repetir todo el proceso. Desplazar este número hasta su posición deseada exige un movimiento "Invertir n-1".
- Repetir el procedimiento hasta conseguir la ordenación completa.

Este algoritmo resuelve *siempre* el puzzle en 2n-3 movimientos. Pero se puede obtener una solución que lleve menos movimientos que ésta. Para demostrar cómo una estrategia directa puede reducir la cantidad de movimientos, consideremos el ejemplo que damos en el recuadro. Nuestro algoritmo llevaría siete (2×5−3) movimientos, pero un jugador hábil lo podría hacer en cuatro.

Este programa es un ejemplo sencillo de toda una serie de juegos de inversión que la gente ha creado e investigado. Quizá a usted le agradará tratar de desarrollar juegos para invertir desde cualquiera de los extremos de la línea, o en los que hubiera de clasificar no sólo una línea de números sino una cuadrícula. Si diseñara su propia versión del juego, tal vez deseara darle más vida utilizando bloques de diferentes colores para reemplazar los números. El objetivo del juego podría ser, entonces, reacomodar una línea de bloques de acuerdo a unos bloques de colores situados en la parte superior de la pantalla. Tal vez también le interesará tratar de incorporarle al programa un algoritmo que les fuera de ayuda a los jugadores que se quedaran encallados.

Invertir

```
20 DIM a(20)
30 CLS : PRINT "Invertir!"
40 INPUT "Cuantos numeros?";n
50 IF n < 0 OR n > 20 OR n <> INT n THEN GO TO 30
60 REM Mezclar la lista
70 FOR i=1 TO n:LET a(i)=i: NEXT i
80 RANDOMIZE
90 FOR i=1 TO n
100 LET r=INT (RND*n+1)
110 LET x=a(r): LET a(r)=a(i): LET a(i)=x
120 NEXT i
130 LET t=1
135 REM Imprimir el tablero
140 CLS: PRINT "Movimiento";t;": La lista es:": PRINT
150 FOR i=1 TO n: PRINT a(i);"  ";: NEXT i
152 REM Verificar si se gana
154 LET i=1
156 IF a(i)=i THEN LET i=i+1: IF i < =n THEN GO
    TO 156
158 IF i > n THEN GO TO 230
159 REM Hacer una pasada
160 PRINT: PRINT: INPUT "Invertir?";r
170 IF r <> INT r OR r < 0 OR r > n THEN GO TO 140
175 REM Invertir r
180 LET t=t+1
190 FOR i=1 TO INT (r/2)
200 LET x=a(i): LET a(i)=a(r−i+1): LET a(r−1+1)=x
210 NEXT i
220 GO TO 140
230 REM Un ganador!
240 PRINT: PRINT: PRINT "Ha terminado en ";t;
    "movimientos"
250 PRINT: INPUT "Juega otra vez (s/n) ?";A$
260 IF a$="S" OR a$="s" THEN RUN
270 CLS: STOP
```

Complementos al BASIC

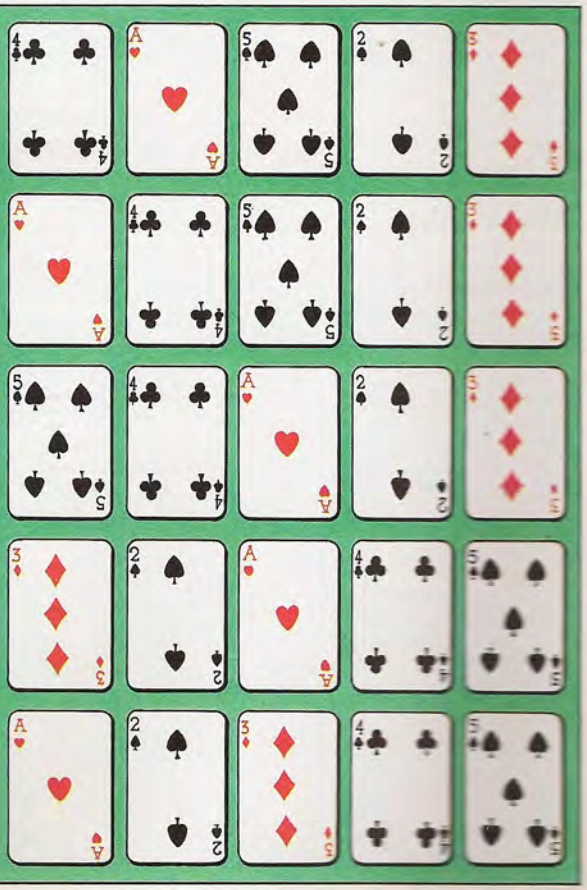
En el Commodore 64 y el Vic-20, reemplazar RANDOMIZE por XX=RND(−TI), reemplazar RND*N por RND(1)*N, y reemplazar CLS por PRINT CHR$(147)

En el BBC Micro eliminar la línea 80, y reemplazar R=INT(RND*N+1) por R=RND(N)

En el Oric-1 y el Oric Atmos eliminar la línea 80 y reemplazar RND*N por RND(1)*N

Cómo proceder

El objetivo de *Invertir* es clasificar una lista invirtiendo reiteradamente subgrupos de la misma. La sección a invertir siempre debe empezar por el elemento situado más a la izquierda, y se describe mediante el número de elementos incluidos. Aquí la secuencia correcta de movimientos es 2-3-5-3, que significa "Invertir los dos naipes más a la izquierda, luego los tres más a la izquierda, etc."



De ámbito mundial

Sharp Corporation fabrica una enorme gama de productos electrónicos, desde transistores a ordenadores y robots industriales

Sharp Corporation siempre ha sido partícipe de las mayores innovaciones tecnológicas. Pero el primer producto suyo que alcanzó el éxito fue un artículo sumamente humilde: el lapicero Ever Sharp ("siempre agudo"). Su creador, Tokuji Hayakawa, creó la Sharp en 1915 para fabricar su invento; y en los años posteriores la empresa fue creciendo de forma continua. En 1925 se inició en la electrónica con un receptor de radio de cristal; y su introducción en los mercados internacionales de la electrónica de consumo se produjo en los años de la posguerra, cuando comenzó a producir aparatos de televisión y otros electrodomésticos. A mediados de los años sesenta la empresa intervino en el mercado de maquinaria de gestión con una serie de calculadoras de sobremesa. En la actualidad es una inmensa multinacional, subdividida en seis grupos de fabricación, con 34 plantas de producción en 30 países, sin contar Japón.

El primer ordenador Sharp que se comercializó en Gran Bretaña fue el MZ80K, que se lanzó en 1981. Al año siguiente, la empresa agregó a su gama el MZ380A y el MZ380B. A pesar de que estos ordenadores se comercializaron como máquinas de oficina, también fueron objeto de una buena acogida por parte de los usuarios de micros personales. Cada modelo viene equipado con una pantalla incorporada y unidades de cassette. Al principio, estos ordenadores se comercializaron como "máquinas limpias", haciendo hincapié en la ausencia de un lenguaje residente en ROM. Este hecho representaba una ventaja: se podían cargar, desde cassette, numerosos lenguajes, incluyendo CP/M.

Grupo de Instrumentos Industriales (IIG)
Los aspectos tecnológicos que son objeto de investigación en el Centro de Ingeniería de Sharp se trasladan a la planta IIG (que vemos en la fotografía) de Yamato-Koriyama-shi, en Nara (Japón), donde se diseñan los productos de la empresa (incluyendo calculadoras y ordenadores personales)

A principios de 1983 Sharp empezó a vender en Gran Bretaña una gama completa de ordenadores personales y de oficina; en esa fecha, además, la lista de productos de la empresa se amplió para incluir la máquina de gestión MZ-3541 y el ordenador de bolsillo PC-1500. El éxito de este último condujo a la comercialización del ordenador de bolsillo PC-1251.

La empresa penetró en el mercado del ordenador personal con el lanzamiento del Sharp MZ-711. Ésta es la versión europea de la serie japonesa MZ-700, y el inmenso juego de caracteres japoneses del original ha permitido un espacio extra para facilidades de gráficos en el modelo europeo. La máquina lleva instalada como estándar una grabadora de datos y se incluye espacio para una impresora-plotter opcional.

En mayo de 1984 la empresa puso a la venta el PC-1500A, una versión mejorada del PC-1500. El nuevo modelo posee 8,9 Kbytes de RAM, que se pueden ampliar a 24 Kbytes. Antes de que finalice 1984 la empresa tiene planeado lanzar la máquina de bolsillo PC-1350, con una visualización de cuatro líneas y capacidades para gráficos. La Sharp Corporation también tiene la intención de introducir el paquete Sharpwriter, que es una fusión de la máquina de escribir eléctrica Sharp ZX-401 y el microordenador MZ-3541. La máquina de escribir se utiliza como un teclado-impresora conectado al ordenador mediante una interface RS232.

Cuando se le pregunta acerca de los futuros desarrollos de Sharp, el director de ventas Rod Goodier afirma que a él "siempre le ha interesado ampliar el mercado del ordenador de bolsillo, que ofrece un potencial tan grande". Ello no significa que se deje de lado el mercado del ordenador personal. "Con la familia MZ-700 realmente no escatimaremos gastos y estamos decididos a mantener una posición sólida en el mercado de ordenadores personales." Peter Fletcher, portavoz de la empresa, explica que "la división de equipos para oficina, que incluye los ordenadores personales, es una innovación relativamente reciente, y representa el 25 % del movimiento total en Gran Bretaña. El plan consiste en aumentar la proporción de movimientos".

En la actualidad Sharp de Gran Bretaña sólo posee un local para almacenaje y comercialización. La empresa está construyendo una fábrica en Wrexham (Gales), donde se ensamblarán grabadoras de video. Con el plan se espera producir 60 000 máquinas en 1985 para distribuirlas a toda Europa.

Al preguntársele acerca de la participación de Sharp con otras empresas en la invasión de MSX que se anunciaba para el otoño de 1984, Rod Goodier replicó que "Sharp ha desarrollado un sistema MSX, pero por el momento no tenemos planeado lanzarlo en Gran Bretaña".

FINANCIAL MICRO TIMES

Poder para el pueblo

A muchas personas les atrae la idea de llevar su propio negocio. Pero, aun cuando se trate de una tienda pequeña, los problemas que entraña una aventura de este tipo pueden ser enormes. Y esto se aplica en toda su dimensión cuando se trata de llevar las finanzas de su empresa. Sin embargo, en este capítulo no vamos a ocuparnos del software de gestión práctico, sino de una selección de paquetes de juegos que simulan algunos de los problemas y los desafíos que plantean los grandes negocios. El más conocido de estos juegos es el *Monopoly*, que es una adaptación del juego de tablero sobre administración y compra-venta de propiedades. Los juegos de gestión son ideales para el ordenador, dado que no hay ni fichas ni papel moneda que se puedan extraviar y que el ordenador efectúa todos los cálculos.

La gama de empresas que se puede llevar es muy diversa: cervecerías, líneas aéreas, fábricas de automóviles y granjas son sólo algunos ejemplos. Obviamente, cada tipo de actividad tiene sus propios problemas.

Independientemente del tipo de empresa que se elija, los programas de simulación de gestión funcionan de forma muy similar. En todos los casos el jugador es el gerente de una empresa determinada y al comienzo del juego dispone de una cantidad inicial de dinero. En el programa *Corn cropper* (Cosechador de trigo), en el que se dirige una granja, esta cantidad es de apenas £ 50 000; pero si se está a cargo de una compañía petrolera, como en el juego *Dallas*, son necesarios cien millones de dólares sólo para arrancar.

Se invierte luego parte de este capital en lo que se denomina "activo fijo" (aviones, fábricas de automóviles, terrenos, etc.). Valiéndose del mismo se puede comenzar entonces a producir sus artículos o servicios (vuelos, coches, trigo) y obtener ingresos.

El grado de realismo varía de un programa a otro. Algunos de ellos, como *Dallas*, están a todas luces pensados como entretenimiento. Otros pueden llegar a ser muy realistas: el programa *Corn cropper* posee datos acerca de las horas de sol y la cantidad de lluvia, las condiciones ideales para cultivar trigo, los salarios de los obreros contratados y los costos de mantenimiento de los tractores, que son sólo algunos de los factores que deben tener en cuenta los granjeros modernos.

Habrá, inevitablemente, algunas personas a las que no les satisfaga la idea de dirigir meramente una empresa, ni siquiera aunque se trate de una corporación petrolífera de $100 millones. Para estas personas la respuesta evidente consiste en ¡intentar gobernar un país! *1984* es un juego que simula la economía de Gran Bretaña, siendo el jugador el primer ministro.

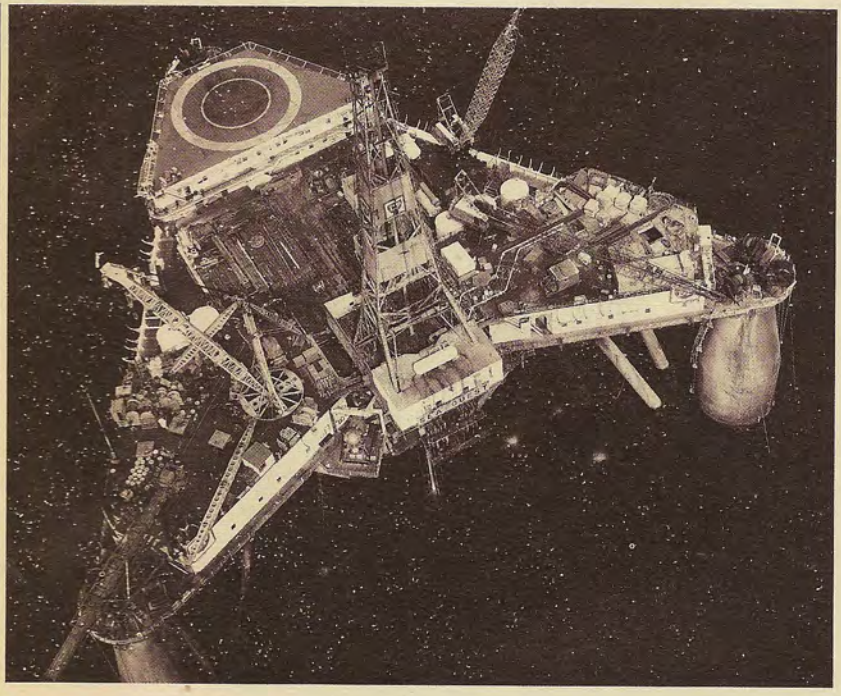
Para simular la economía de este país, los economistas utilizan un "modelo", no un modelo físico, sino más bien un diagrama complejo que ilustra cómo interactúan entre sí todos los componentes de la economía. En los programas *1984* el modelo es muy simple, dado que apenas si contiene cinco secciones: el gobierno, los bancos, la población, la industria y el resto del mundo.

El Ministerio de Hacienda también posee un modelo económico, que el gobierno británico utiliza para hacer diversas predicciones acerca de la economía nacional. Como cabe imaginar, este modelo es algo más complicado. El programa de simulación, que contiene el modelo, se denomina *AMODEL* y se ejecuta en un ordenador central Sperry 1100. Para efectuar sus predicciones emplea 1 100 variables económicas distintas (la tasa de inflación, desempleo, etc.) reunidas durante los últimos 10 años. Sólo estos datos ocupan alrededor de 250 Kbytes de memoria, pero cuando se ejecuta el programa al máximo de su capacidad se utilizan cerca de ocho Mbytes.

Típicamente, el modelo se puede emplear para predecir cifras de desempleo, el nivel de importaciones y exportaciones, el valor al cambio del dólar y muchos otros índices económicos. Para que se haga una idea de la complejidad del modelo, basta con decir que predecir la tasa de inflación para el año que viene llevaría dos o tres minutos de procesador. No parece mucho tiempo, pero hay que tener presente que el procesador ejecuta alrededor de 10 millones de instrucciones en código máquina en apenas un segundo. Imprimir la predicción lleva otros 15 o 20 minutos.

Los juegos de simulación por ordenador le ofrecen la oportunidad de dirigir su propio negocio (o incluso la economía de toda una nación) sin correr el riesgo de perder todos los ahorros de su vida si la empresa se declara en quiebra. Lamentablemente, ¡uno tampoco puede disfrutar de los posibles beneficios!

FINANCIAL MICRO TIMES



Associated Press

ORO NEGRO

"Dallas" es un juego de gestión que coloca al jugador al frente de su propia compañía petrolífera. Comenzando con $100 millones en efectivo, se deben acumular $200 millones con el fin de adquirir a su mayor rival, Euing Associates. (El error ortográfico en el apellido familiar de la popular serie de televisión es intencional.)

En la pantalla hay un mapa del estado de Texas y a medida que avanza el juego se tiene la oportunidad de ir ganando "concesiones" en cada uno de los cuadrados. Para obtener la concesión es necesario hacer una oferta con éxito, pero una vez que se ha efectuado una, hay que perforar un pozo de prueba para buscar el petróleo. Si descubre petróleo en su campo, entonces instala un equipo de perforación y medios de producción, y finalmente un oleoducto. Si se compromete demasiado (comprando, p. ej., muchas concesiones antes de disponer de ingresos por petróleo), entonces tal vez se vea obligado a acudir a un banco para obtener un crédito, con lo cual empieza a pagar intereses. Si el préstamo supera la cantidad de $20 millones, puede ser absorbido por Euing Associates.

Este programa lo vende Cases Computer Simulations para una serie de micros personales, entre ellos el BBC Micro, el Spectrum, el Oric y el Electron.

ALTAS FINANZAS

En "Airline" usted es el presidente de L-AIR, unas líneas aéreas privadas cuyo capital, al comienzo del juego, es de £3 millones. En un período de siete años debe incrementar el activo a 30 millones, punto en el que está en condiciones de adquirir la British Airways y ganar el juego.

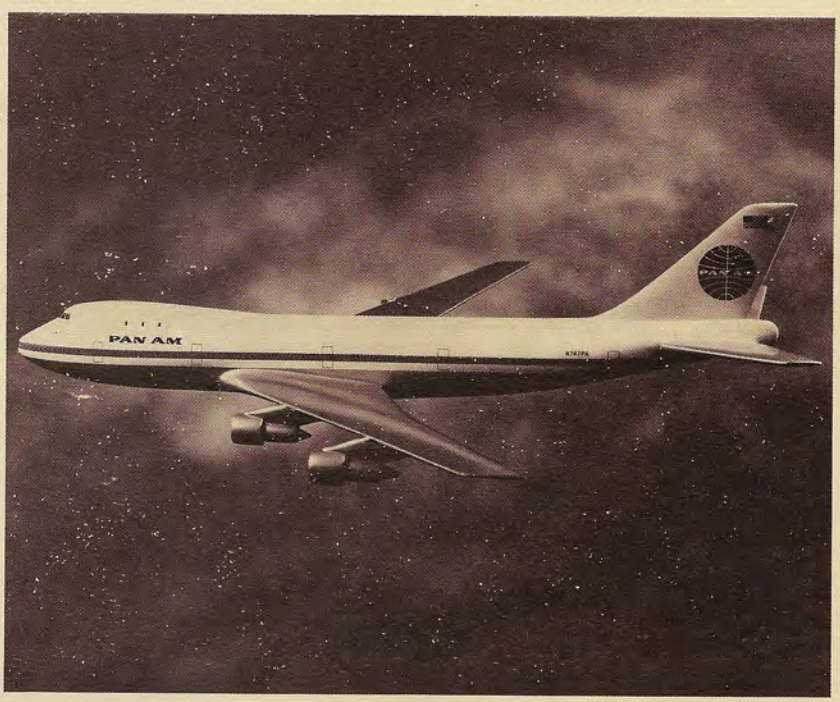
Al principio de cada año financiero hay que decidir con cuántos aviones operar, utilizando estimaciones que predicen el número de pasajeros. Inicialmente no se dispone de suficiente dinero para comprar un avión (valen £10 millones cada uno), de modo que hay que alquilarlos. En años más rentables el jugador habrá de decidir si es mejor alquilar un avión o comprarlo, y esto depende tanto de las tarifas de alquiler como de los tipos de interés.

Se debe decidir, asimismo, el nivel de tripulación y mantenimiento de su avión. Si es demasiado bajo, entonces quizá no se disponga de personal suficiente o de suficientes aviones, y se podría llegar a cancelar algunos vuelos. Si es demasiado alto, se estará derrochando dinero.

En ocasiones en la pantalla aparece una "cinta de teleimpresora" que contiene numerosos mensajes de télex. Por ejemplo, quizá la OPEP haya aumentado el precio de los crudos (y, por consiguiente, el combustible) o al jugador se le ha concedido licencia para una nueva ruta aérea.

Al final del año una hoja de balance proporciona el rendimiento global de las L-AIR y un informe del gerente acerca del éxito obtenido por la empresa. Si en el primer año se pierden £10 millones, la empresa se liquida.

El juego está a la venta para el Spectrum, Oric, Electron y BBC Micro, y lo produce Cases Computer Simulations.



Associated Press

Una oportunidad de primera

Associated Press

En este juego usted es el primer ministro de Gran Bretaña en el año 1984. Su objetivo es el de mantenerse en su cargo el mayor tiempo posible, y su popularidad está determinada por el éxito que consiga al equilibrar las cuentas del país.

Al comienzo de su mandato tiene que tomar diversas decisiones en materia financiera. A modo de ayuda dispone de tres indicadores económicos, incluyendo las tasas de inflación y desempleo, etc. Al inicio de cada año un gráfico indica cómo ha cambiado cada indicador durante sus años de gestión.

Su primera decisión consiste en determinar el tipo mínimo de crédito, que determina el tipo de interés para el año. Usted tiene luego que negociar los incrementos salariales para la Administración Pública, el sector público y el sector privado.

Después de los convenios salariales, debe decidir acerca de los niveles de gastos de los diversos departamentos y ministerios del gobierno. Por último, puede anunciar su presupuesto, que le da la oportunidad de recaudar dinero a base de impuestos. Al final de cada año de su mandato de cinco años, un sondeo de opinión refleja la popularidad de su política y contribuirá al objetivo último que usted persigue, el de ser reelegido al final del juego.

Incentive Software produce el juego *1984* para los modelos BBC Micro y el Spectrum.

COSECHA DORADA

"Corn cropper", de Cases Computer Simulations, disponible para el Spectrum, el Electron o el BBC Micro, simula la administración de una gran granja de trigo. Se empieza a jugar con £50 000 y se tiene como objetivo dirigir la granja de modo que al cabo de cinco años posea un activo total de £250 000.

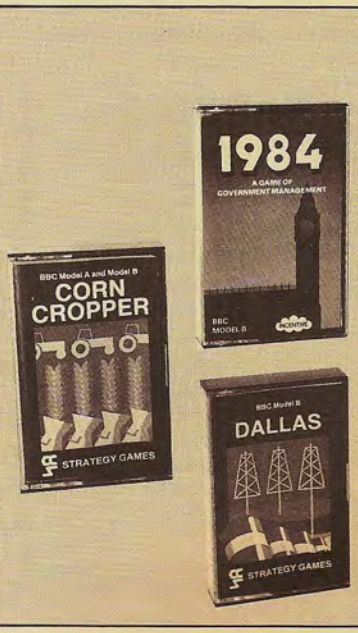
Al principio el participante apenas posee 30 acres de tierra que se han de sembrar. Para hacerlo necesita comprar semilla, alquilar un tractor y pagar el riego en caso de que no llueva lo suficiente. Al principio de cada mes el jugador dispone de las previsiones de precipitaciones y temperatura y decide, en consecuencia, si sembrar o no.

A medida que van transcurriendo los meses tiene que estar atento al "informe del estado de la cosecha" (un calendario que dice si hay trigo listo para su cosecha).

Pueden aparecer otros imponderables: las ratas se podrían comer parte de sus semillas; si no ha pulverizado el grano, los insectos lo atacarán, y la helada puede producir daños considerables.

A medida que avanza el juego, usted intenta obtener unos ingresos constantes producto de la venta del trigo cosechado, de modo tal que pueda reinvertir los ingresos en más tierras y más semillas. Es bastante difícil acrecentar su activo por encima de las £100 000, en especial porque no se le permite girar en descubierto al banco. Como último recurso, siempre puede vender parte de sus tierras para hacerse con algo más de dinero.

Associated Press



Dirección desde la butaca
Estos juegos de gestión por ordenador personal utilizan técnicas de simulación para poner al jugador al mando de unas líneas aéreas, una granja de trigo, una compañía petrolífera y la economía británica. El grado de realismo varía de acuerdo a la cantidad de factores de control que admita el juego y a la complejidad de los modelos de interacción de factores del juego

Ian McKinnell

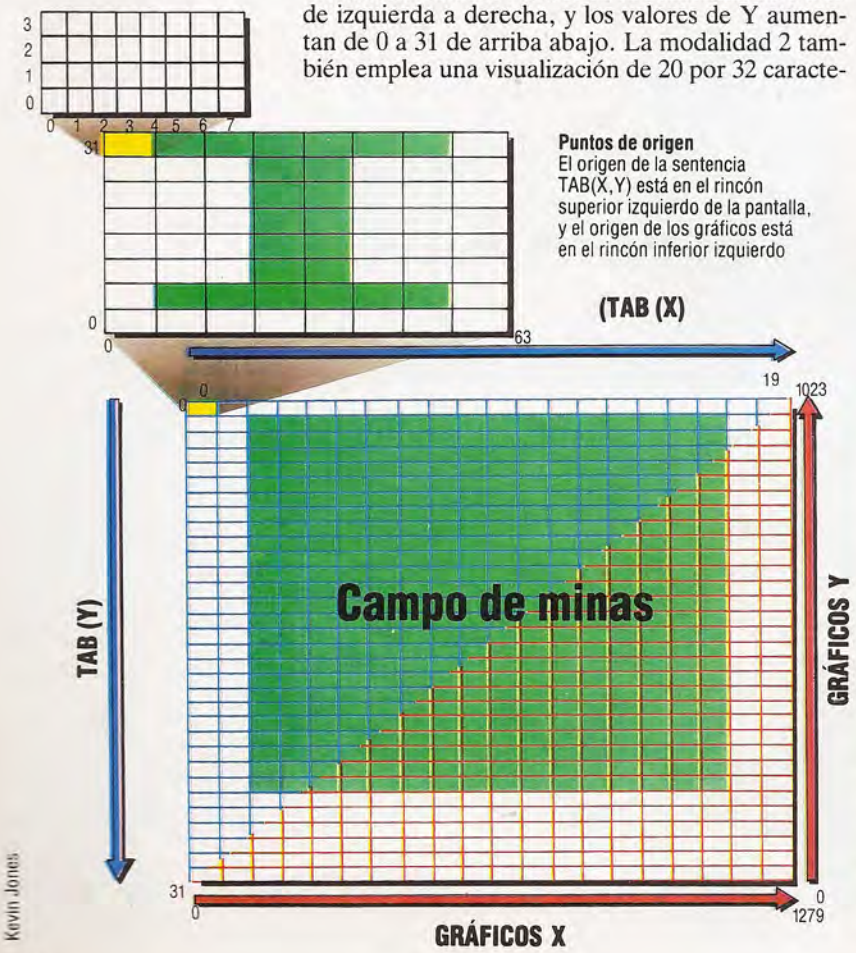
Radio de acción

Vamos a analizar el trazado de líneas, el reloj interno y los demás procedimientos necesarios para el escenario de nuestro juego

En el primer capítulo definimos una superficie de la pantalla del BBC en modalidad 5 como el "campo de minas" en el cual se habrá de desarrollar nuestro juego. Definimos las formas de las minas, del detector y del ayudante, desarrollamos procedimientos para colocar un cierto número de minas al azar en el campo de minas, y situamos el detector y el ayudante en sus posiciones de comienzo. Para hacer que la visualización sea más atractiva podemos dibujar un borde alrededor de la superficie del campo de minas. La forma más sencilla de hacerlo consiste en utilizar las instrucciones en alta resolución MOVE y DRAW.

En el BBC/Electron existe un problema al mezclar gráficos en alta resolución con caracteres en baja resolución: las visualizaciones diferentes utilizan sistemas de coordenadas diferentes. Ya hemos analizado en detalle la visualización en baja resolución cuando la aplicamos para posicionar las minas y los caracteres con la instrucción PRINT TAB(X,Y). En este sistema, el origen (el punto donde tanto X como Y son cero) es el rincón superior izquierdo. En este sistema los valores de X aumentan de 0 a 19 de izquierda a derecha, y los valores de Y aumentan de 0 a 31 de arriba abajo. La modalidad 2 también emplea una visualización de 20 por 32 caracteres, pero el resto de modalidades muestra una cantidad distinta de caracteres y, por lo tanto, utilizan distintas coordenadas para cada carácter.

En un cuadrado
Los caracteres se definen en una matriz de ocho pixels por ocho. En la modalidad 5, en realidad ésta es rectangular, siendo de 64 puntos de alta resolución de ancho por 32 puntos de alto, de modo que cada pixel de caracteres de la modalidad 5 debe medir 8 puntos de alta resolución de ancho por 4 puntos de alto. Trazar (PLOT) o dibujar (DRAW) cualquiera de los puntos de un pixel hará que se ilumine todo el pixel: PLOT 7,3, por consiguiente, equivale a PLOT 5,2



Puntos de origen
El origen de la sentencia TAB(X,Y) está en el rincón superior izquierdo de la pantalla, y el origen de los gráficos está en el rincón inferior izquierdo

Las ocho modalidades de visualización del BBC/Electron proporcionan tres resoluciones distintas (640 por 256, 320 por 256 y 160 por 256) y, aun así, todas ellas emplean el mismo sistema de coordenadas. Aunque al principio este sistema resulta algo confuso, es una verdadera ayuda para la programación, porque los gráficos diseñados para una modalidad se podrán usar para las pantallas de otras modalidades.

El sistema de coordenadas del BBC/Electron trata la pantalla como si tuviera una resolución de 1280 por 1024. Ésta, por supuesto, es una visualización más grande de la que pueden producir ambas máquinas, ¡evidentemente para permitir futuros desarrollos del BBC! Todas las coordenadas se dan como números comprendidos en la escala entre 0 y 1279 a lo ancho de la pantalla y entre 0 y 1023 a lo alto de la misma. El BASIC BBC los convierte automáticamente en los valores apropiados para cualquiera de las tres resoluciones que se esté utilizando.

Mezclar gráficos con caracteres resulta un poco más difícil, porque el origen de coordenadas de los gráficos está situado en el rincón inferior izquierdo de la pantalla, en vez de en el rincón superior izquierdo empleado en el sistema de coordenadas de caracteres.

En la modalidad 5, la resolución de 160 por 256 pixels guarda una relación directa con el número de caracteres que se pueden mostrar. Cada uno de los caracteres se forma a partir de una cuadrícula de ocho por ocho pixels. Dado que a lo ancho de la pantalla hay sitio para 20 caracteres, ello significa que a lo ancho debe haber $8 \times 20 = 160$ pixels. Del mismo modo, a lo alto de la pantalla debe haber $32 \times 8 = 256$ pixels. Para relacionar esto con el sistema de coordenadas en alta resolución debemos recordar que un pixel es la menor superficie de luz de la pantalla que se puede encender o apagar individualmente. En el sistema de alta resolución, en la dirección X hay 1280 coordenadas diferentes. Si dividimos esta cantidad por el número de pixels de la dirección X, obtenemos 1280/160 = 8. Del mismo modo, en la dirección Y, dividiendo las coordenadas de alta resolución por el número de pixels obtenemos 1024/256 = 4. Esto significa que cada pixel se puede encender haciendo referencia a cualquiera de las diversas posiciones del sistema de coordenadas de 1280 por 1024. La ilustración refleja cómo se puede utilizar una gama de coordenadas para encender (o apagar) un pixel. Trazando (7,3) se iluminaría el mismo pixel que trazando (0,0) o (5,2), y así sucesivamente.

Podemos utilizar este hecho para relacionar posiciones de caracteres con coordenadas de alta reso-

Kevin Jones

lución. En el eje horizontal, si un pixel equivale a ocho unidades, la anchura de un carácter equivaldrá a 64 unidades. Cuatro unidades equivalen a un pixel en la dirección vertical, lo que implica que la altura de cada carácter es de 32 unidades. Las fronteras de la pantalla se pueden calcular ahora en términos de coordenadas en alta resolución para las instrucciones MOVE y DRAW. La ilustración del trazado de pantalla muestra cuáles son estas fronteras.

Ahora podemos calcular las coordenadas del rincón inferior izquierdo y del rincón superior derecho del campo de minas (todas las otras coordenadas para el borde se obtienen a partir de estos dos puntos). Como podemos ver a partir del diagrama, las coordenadas del rincón inferior del borde son (120,188). Las coordenadas del rincón superior derecho son (1152,992).

El siguiente procedimiento dibuja un borde alrededor del margen de la superficie definida. GCOL 0,1 establece el color lógico que se utilizará para los gráficos. El primer número define el tipo de trazado, que examinaremos más adelante, y el segundo define el color. Las instrucciones MOVE desplazan el cursor para gráficos (sin dibujar) desde el origen hasta el rincón inferior izquierdo del borde. Las instrucciones DRAW que le siguen dibujan líneas rectas desde la posición en curso de la pantalla hasta el punto especificado.

```
2470 DEF PROCdibujar-borde
2480 GCOL 0,1
2490 MOVE 120,188
2500 DRAW 120,992
2510 DRAW 1152,992
2520 DRAW 1152,188
2530 DRAW 120,188
2540 ENDPROC
```

El reloj interno

El BBC y el Electron poseen un reloj interno al cual se puede acceder fácilmente desde el BASIC utilizando la variable reservada TIME. Cuando se solicita que imprima el valor de TIME, el ordenador devuelve un número que corresponde al tiempo, expresado en centésimas de segundo, dado que a la variable se le dio la última vez el valor de cero. El procedimiento "establecer-tiempo" imprime la palabra "Tiempo", su valor de comienzo y pone el valor cero a la variable TIME. A este procedimiento se lo llama durante la rutina de preparación e inicia el reloj para el juego.

```
2640 DEF PROCestablecer-tiempo
2650 PRINTTAB(2,27)"Tiempo    02:00"
2660 TIME = 0
2670 ENDPROC
```

Dentro del bucle principal del programa, se debe actualizar el tiempo visualizado en la pantalla. Visualizar el tiempo en segundos sería muy directo; simplemente dividiríamos la variable TIME por 100, para convertir su valor a segundos, imprimiríamos este valor en la pantalla, y así sucesivamente. Sin embargo, es posible convertir TIME a minutos y segundos haciendo uso de las instrucciones del BASIC BBC DIV y MOD. TIME DIV 100 devolvería el número de segundos como un número entero; (TIME DIV 100)MOD 60 contaría los segundos de 0 a 59 y luego volvería a comenzar desde cero. Ello se debe a que la instrucción MOD 60 da el valor del resto después de la división por 60. De modo que, por ejemplo, 63/60 es 1, con un resto de 3. (63/60)MOD 60, por consiguiente, sería 3. Los minutos se pueden aislar de la misma manera y visualizar mediante la utilización de (TIME DIV 6000)MOD 60.

Éste es el procedimiento que se aplicará para actualizar el tiempo durante el juego:

```
2900 DEF PROCactualizar-tiempo
2910 seg$ = STR$(((12100-TIME) DIV 100)MOD 60)
2920 min$ = STR$(((12100-TIME) DIV 6000)MOD 60)
2930 REM ** sumar ceros izquierda**
2940 seg$ = LEFT$(cero$,2-LEN(seg$)) + seg$
2950 min$ = LEFT$(cero$,2-LEN(min$)) + min$
2960 time$ = min$ + ":" + seg$
2970 PRINTTAB(11,27);time$
2980 ENDPROC
```

Como puede apreciar a partir de este procedimiento, hemos ido un paso más hacia adelante. Además de estar dividido en minutos y segundos, el tiempo en realidad se contará hacia atrás desde dos minutos a cero. También se incluye una breve rutina de manipulación de series para asegurar que las visualizaciones para los segundos y los minutos siempre tendrán dos dígitos, agregando ceros a la izquierda cuando ello sea necesario.

Para completar la preparación del juego se requieren aún otros dos cortos procedimientos. Durante el juego el participante tiene cuatro vidas; por consiguiente, necesitamos visualizar, en la parte inferior de la pantalla, la cantidad de vidas que le quedan. Al principio éstas serán tres, visualizadas en forma de tres caracteres de "ayudante", tal como lo habíamos definido en el capítulo anterior (véase p. 873). Se empleará un "contador" variable para determinar la cantidad de vidas utilizadas. Inicialmente, éste valdrá uno.

```
2690 DEF PROCestablecer-hombres
2700 hombres$ = CHR$(226) + CHR$(226) + CHR$(226)
2710 contador = 1
2720 COLOUR 1
2730 PRINTTAB(2,30);hombres$
2740 COLOUR 2
2750 ENDPROC
```

La rutina final de preparación inicializa los marcadores y los visualiza en la pantalla. El valor de "maxmarcador$" no se inicializa en este procedimiento. Por tanto, estableceremos su valor inicial sólo al comienzo del programa.

```
2770 DEF PROCestablecer-marcador
2780 marcador = 0:marcador$ = "00000"
2790 PRINTTAB(2,28)"Marcador      00000"
2800 PRINTTAB(2,29)"Max marcador ";max-marcador$
2810 ENDPROC
```

Ahora que ya hemos ensamblado todos los procedimientos de la parte de preparación del programa, podemos construir un procedimiento de mayor nivel para llamarlos a todos ellos. En el último capítulo habíamos llamado a todos los procedimientos que habíamos ensamblado directamente desde un corto programa principal. Ahora usted debe eliminar aquellas líneas y agregar las siguientes:

```
1880 DEF PROCpreparacion
1890 COLOUR 2
1900 bandera-final = 0
1910 PROCinicializar-variables
1920 PROCdefinir-caracteres
1940 PROCcolocar-minas(40)
1950 PROCdibujar-borde
1960 PROCestablecer-tiempo
1970 PROCestablecer-marcador
1980 PROCestablecer-hombres
1990 PROCsituar-sujetos
2000 ENDPROC
```

Estamos ahora en la etapa en la que podemos escribir un corto programa principal para llamar al procedimiento "preparación" y actualizar el tiempo en un bucle REPEAT...UNTIL. Añada a su programa estas líneas:

```
10 REM****PROGRAMA DE LLAMADA****
20 maxmarcador$ = "00000"
30 PROCpreparacion
40 REPEAT
50 PROCactualizar-tiempo
60 UNTIL TIME > 12099
70 END
```

Ejecutante versátil

La Brother EP-44 puede funcionar como impresora y también ser una eficiente calculadora y terminal de comunicaciones

La Brother EP-44 pesa 2,2 k y funciona a pilas, por lo que es una máquina auténticamente portátil; se le puede agregar como accesorio un transformador de corriente. En el teclado, además de las teclas convencionales de máquina de escribir, hay cuatro teclas para tratamiento de textos y siete teclas de calculadora. Pero la característica más notable es la visualización en cristal líquido (LCD) de 15 caracteres. Ésta permite que el usuario visualice y edite el texto antes de imprimirlo.

Para utilizar la EP-44 como máquina de escribir convencional, el interruptor Print Mode Selector (selector de modalidad de impresión) se coloca en "Direct Print" (impresión directa). En esta modalidad, el texto aparece en la "ventana" de LCD y, simultáneamente, se imprime en papel. Las tabulaciones, el espaciado de las líneas y los márgenes se fijan del mismo modo que en una máquina de escribir convencional. Sin embargo, el sistema de alimentación de papel es poco frecuente: en vez de mover manualmente el rodillo portapapel, el usuario debe pulsar uno de dos botones para desplazar el papel hacia arriba o hacia abajo. El teclado no responde del todo al estándar de una máquina de escribir electrónica, pero es perfectamente adecuado.

La EP-44 utiliza un cabezal de impresión térmico para "quemar" una serie de puntos sobre el papel y formar así el carácter deseado. La mayoría de las impresoras térmicas emplean una matriz de puntos de nueve por siete para conformar los caracteres, pero con la matriz de la EP-44, de 24 por 18, se obtiene una mejor calidad de impresión. Otra de las impresoras de la gama Brother, la EP-22, utiliza menos puntos y produce una impresión notablemente más pobre.

Una importante desventaja del sistema de impresión térmico es la baja velocidad de la impresión que, en este caso, está por debajo de los 16 caracteres por segundo. Cuando se emplea la Brother como máquina de escribir esto no es un inconveniente serio, pero cuando se utiliza la máquina como impresora de ordenador la escasa velocidad se hace muy notoria; una buena impresora matricial imprimirá el texto a una velocidad 10 veces mayor que la Brother. El otro problema de este sistema es el papel térmico especial que requiere; éste es caro y su aspecto es muy satinado.

Si se coloca el interruptor Print Mode Selector en "Correction Print" (corregir impresión), la ventana LCD demuestra su verdadera utilidad. En esta modalidad el texto aparece en la ventana pero se produce una demora antes de que sea impreso. De hecho, el texto impreso en papel lleva 15 caracteres

El teclado
Las 51 teclas de impresión están ligeramente esculpidas, son de presión suave y responden a un trazado QWERTY no estándar. Hay dos teclas de cambio de caracter con bloqueo, una de cambio de símbolo y una de cambio de función. Las teclas de caracteres son teclas mulfifunción, que ofrecen una gama de caracteres extranjeros y acentos ortográficos. Hay 21 teclas de función que controlan las modalidades de impresión, de calculadora y de memoria, la colocación de márgenes, tabuladores y el movimiento del papel

de atraso con respecto a la letra que se está digitando en cada momento. Esto produce al principio una sensación extraña, pero permite la alteración de cualquiera de los últimos 15 caracteres antes de que sean impresos, lo que resulta útil si su mecanografía no es muy precisa y, por cierto, representa una gran mejora sobre los líquidos correctores. La edición se realiza en la visualización de LCD mediante la utilización de dos teclas para control del cursor, para seleccionar el carácter a modificar. Hay, asimismo, una modalidad "Line-By-Line" (línea a línea), en la cual la EP-44, antes de escribir una línea completa de texto, espera a que se pulse la tecla Return. Ello posibilita la edición de una línea entera en vez de sólo los 15 últimos caracteres mecanografiados.

Los usuarios de tratamiento de textos se acostumbran enseguida al hecho de que el texto se formatea automáticamente y pierden pronto el hábito de pulsar la tecla Return al final de cada línea de texto. La EP-44 posee una modalidad "Auto Return" (retorno automático), en la cual se inicia automáticamente una nueva línea si se pulsa la barra espaciadora en alguna de las seis últimas posiciones de una línea.

La calculadora incorporada de la EP-44 permite realizar cálculos simples en la visualización LCD sin interrumpir la impresión. Por ejemplo, si se está entrando una factura de venta y se necesita calcular el ITE o los descuentos usuales, puede utilizarse la calculadora para obtener las cifras en cuestión y continuar imprimiendo los detalles de la factura.

Todas las características que hemos mencionado aquí también se pueden encontrar en muchas máquinas de escribir electrónicas modernas. Pero la EP-44 también se puede emplear como un sencillo procesador de textos. Esta característica utiliza los 3,5 Kbytes de memoria interna que emplea la Brother para almacenar texto. Esta memoria puede resultar pequeña si se la compara con aquella de que disponen la mayoría de los ordenadores personales, pero es suficiente para producir una carta de tres páginas. A las instrucciones necesarias para el tratamiento de textos se accede utilizando las teclas normales de máquina de escribir conjuntamente con una tecla "Code" (código) especial de color azul. Por consiguiente, para introducir texto en la memoria lo único que hay que hacer es pulsar la tecla Code y la letra "N" (de *new text*: texto nuevo). Se puede entrar el texto tanto en las modalidades Direct como Correction Print.

Después de haber entrado el texto en la memoria, para editarlo se utilizan la visualización LCD y las teclas del cursor. Las teclas del cursor permiten desplazar la ventana LCD hacia arriba, hacia abajo, hacia la izquierda y la derecha, de modo que se puede contemplar el texto completo e insertar, eliminar o corregir caracteres a voluntad. La única limitación reside en el hecho de que las palabras no se pueden desplazar hasta la línea de abajo si la línea que se está editando es demasiado larga, lo que significa a veces la imposibilidad de reformatear un documento.

La gran ventaja de entrar el texto de esta manera es que éste se puede alterar sin necesidad de volver a mecanografiar todo el documento entero. Por ejemplo, a menudo hay que enviar la misma carta a varias personas diferentes; para hacerlo, simplemente se edita el texto en la memoria para insertar el nombre y la dirección correspondiente, y luego se pulsa la tecla Code junto con la letra "P" (de *print text*: imprimir texto). Entonces se imprimirá su carta modificada.

LCD, ¿OK?

La pantalla LCD de la EP-44 visualiza 15 caracteres, cinco indicadores de modalidad de impresión y una alerta del estado de las pilas. El texto se puede editar en la pantalla, permitiendo, por tanto, un tratamiento de textos sencillo. La intensidad de visualización de los caracteres se puede ajustar de acuerdo a las condiciones de iluminación ambiental

Ian McKinnell

Quitando una pequeña cubierta a uno de los lados de la EP-44 queda al descubierto un conector en serie RS232. Un interruptor señalado como "Terminal" convierte a la Brother en una impresora para ordenador, si bien es necesario un poco de práctica para establecer el valor correcto de la velocidad en baudios, longitud de las palabras, etc., para su máquina en particular. Adaptar la EP-44 es sumamente fácil: la visualización LCD va pasando las diversas velocidades y otros detalles, y usted tan sólo pulsa el interruptor Mode cuando los detalles expuestos son los específicos para su máquina.

Lamentablemente, la Brother no puede manipular el papel de bobina habitual y se la ha de alimentar de forma manual con hojas sueltas. Pero la calidad de impresión es mejor que la de una impresora matricial, por lo que la EP-44 es ideal para la producción de cartas y documentos cortos.

El hecho de que el interruptor se señale como "Terminal" y no como "Printer" (impresora), refleja que la EP-44 puede enviar datos, además de recibirlos. Esto significa que la Brother se puede conectar a un modem, permitiendo que el usuario se comunique, por vía telefónica, con otros usuarios de ordenadores personales o con máquinas más grandes que utilicen los estándares de comunicación apropiados.

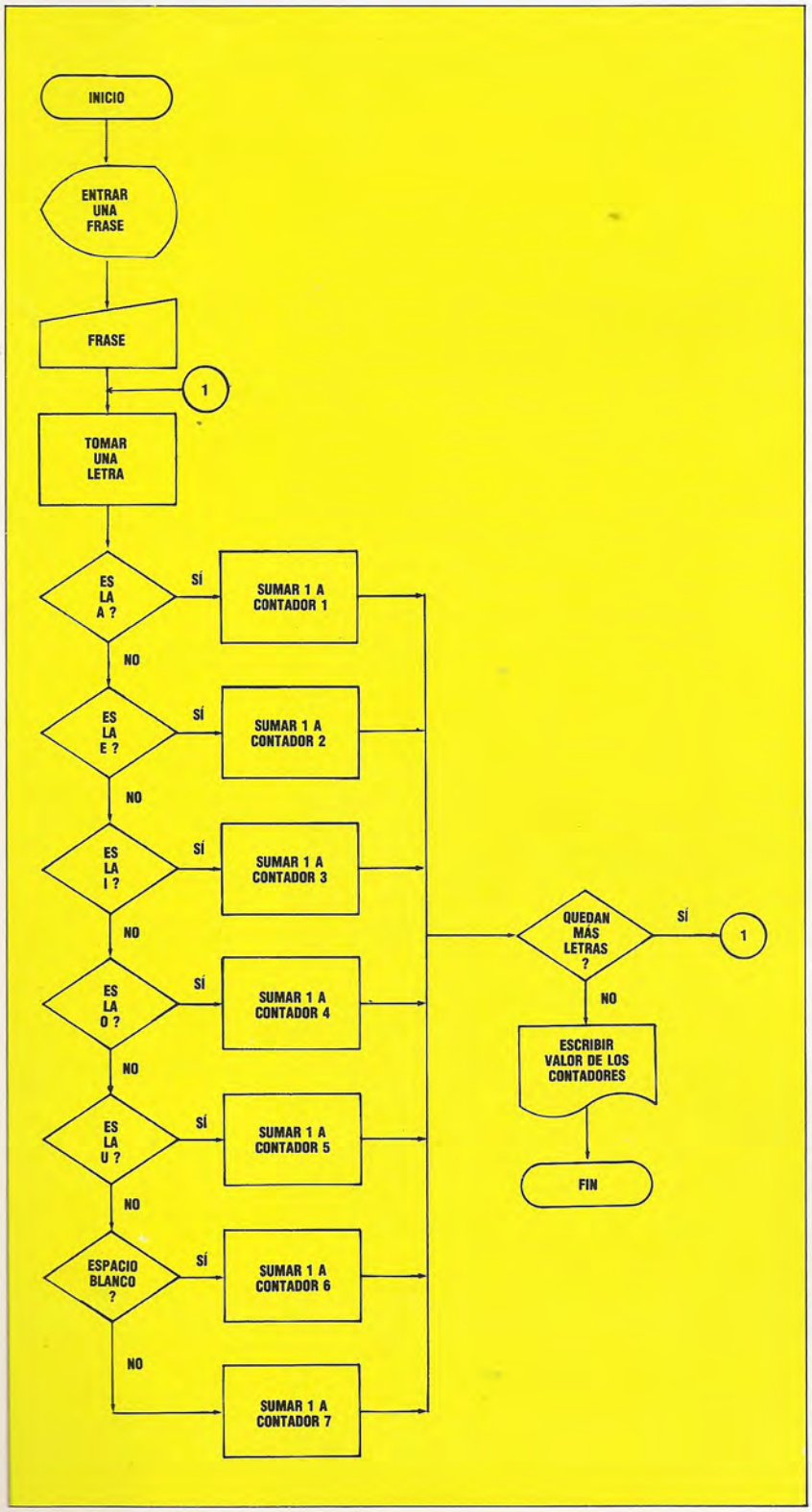
En conjunto, la Brother EP-44 destaca por su versatilidad y por el hecho de que, siendo una máquina pequeña, reúna tantas funciones diferentes.

Salida impresa

```
Con la cinta colocada, se
puede utilizar cualquier
papel de maquina de acabado
satinado, pero la alimentacion
del papel no acepta papel
carbon. La calidad de impresion
es buena con ambas clases
de papel.
El texto se puede:
    Centrar & subrayar
hay caracteres extranjeros
(ÆØ¿¢µπ),
subindices y exponentes
(X² = N₂ + T₀).
```

Contador alfabético

El test en cascada se puede utilizar no sólo en una comparación de datos numéricos, sino también alfanuméricos



Todos los supuestos vistos hasta el momento en los que se ha empleado el test en cascada se han basado en comparaciones con cifras. Ahora bien, esta técnica de diagramación tiene efectividad tanto si se emplea la comparación de datos numéricos como de datos alfanuméricos, tal como se muestra en el siguiente ejemplo. En él, basándose en la introducción de una frase cualquiera entrada por teclado, se pretende conseguir como final lógico una serie de resultados basados en el número de veces que un carácter esté contenido en la frase. Así, se empleará un contador por cada vocal, uno que dé el total de consonantes existentes y otro que lleve el total de espacios en blanco contenidos. Con la visualización de estos siete campos se obtendrá la solución del programa.

Veamos a continuación un ejemplo introduciendo: "LA PELOTA SE ROMPIO".

Nos devolverá el siguiente resultado:

CARÁCTER	TOTAL
A	2
E	2
I	1
O	3
U	0
CONSONANTES	8
ESPACIOS	3

Puede apreciarse que tras las preguntas por cada una de las vocales, se ha inquirido por el espacio en blanco, con lo que, por eliminación, tras el paso por estas seis comparaciones previas, se ha tomado, por defecto, el carácter descendente como una consonante.

Consideraciones generales

El cuerpo del ordinograma podría haberse ampliado sustancialmente en caso de haberse deseado una mayor depuración del resultado, por ejemplo, verificando la inclusión de algún número, o incluso la utilización de algún carácter especial dentro de la frase de nuestro ejemplo.

Con lo que se llega, en un último supuesto, al control total, obtenido tras el empleo de un contador para cada una de las diferentes consonantes de nuestro alfabeto.

Sea cual fuere la opción escogida, obsérvese que su desarrollo no ganaría en complejidad, ya que se trataría de repetir unas mismas fases con la única variación del nombre del carácter y del número del contador correspondiente. Es decir, que la dificultad sería lo laborioso de plasmar el desarrollo del planteamiento, conforme se utilice mayor número de filtros de decisión.

Un brazo eficiente

Los trazadores digitales acaban con la laboriosa tarea de transferir una imagen o un diseño desde el papel al ordenador

Un trazador digital es un instrumento sencillo que permite copiar un dibujo, una fotografía o un diseño a la pantalla del ordenador mediante la técnica de reseguir todos sus trazos con un brazo articulado. La facilidad con que esto se lleva a cabo depende en gran parte del software que acompaña al equipo. Aquí vamos a analizar cuatro trazadores digitales, tres para el BBC Micro y uno para el Sinclair Spectrum.

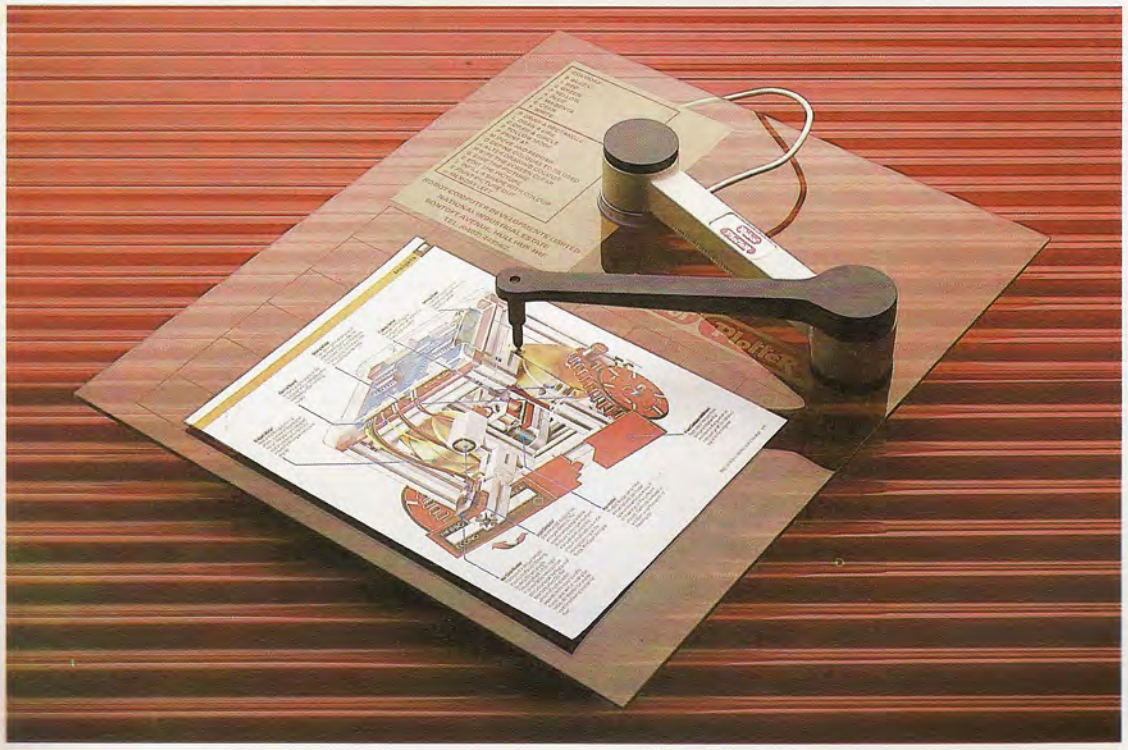
Todos los trazadores funcionan de forma similar. En el extremo de un brazo de doble articulación se fija un puntero que le envía señales eléctricas al ordenador. La intensidad de estas señales varía a tenor de la posición del puntero. El ordenador las convierte en un formato digital y las utiliza para trazar un punto en el lugar correspondiente de la pantalla. Todos los trazadores se suministran con software para llevar a cabo su tarea, ofreciendo diversas opciones, tales como dibujar líneas en distintos colores. El software para los modelos BBC permite seleccionar distintas modalidades de visualización, equilibrando la resolución y el número de colores disponibles dentro de las restricciones impuestas por el limitado espacio de memoria de que se dispone.

El Robot Plotter (trazador robot), de Robot Computer Development, es, de los cuatro que hemos examinado, el de aspecto más impresionante. Este modelo posee una base de perspex que lleva grabado un cuadriculado que muestra las posiciones de los pixels. El brazo trazador está sujeto en uno de los extremos de la base. La imagen a calcar se puede colocar debajo de la cuadrícula y contemplar a través de la base translúcida. El brazo de este modelo es sumamente sólido, construido con metal resistente y plástico. El puntero es una pluma parecida a un lápiz que une el brazo con la placa de la base. Lamentablemente, con este sistema no es fácil ver la imagen a medida que se la va calcando.

El Robot Plotter se vende con una cassette que contiene software para ejecutar en el BBC Micro. Además de las rutinas para calcar, la cassette contiene varias rutinas para dibujar círculos, rectángulos y líneas que se utilizan junto con el brazo.

El programa de trazado almacena todas las imágenes como una serie de líneas; por lo tanto, el trazado de un mapa se almacenará en la memoria como una secuencia de líneas cortas. Esto hace que resulte sumamente fácil eliminar líneas no deseadas sin afectar a las líneas cercanas. Sin embargo, una imagen compleja requiere muchísima memoria y es posible agotar todo el limitado espacio de memoria de la máquina BBC. Debido a que la visualización se almacena como una secuencia de líneas, es relativamente sencillo transferir imágenes creadas con el trazador a otro programa.

El trazador Digigraph también es de construcción sólida. La placa que sirve de base se compone de un gran tablero de madera en cuya superficie hay pintado un cuadriculado rojo. El brazo es de



Robot Plotter
Los originales a calcar se pueden colocar debajo del tablero de trabajo de perspex del Robot Plotter; el software que se suministra incluye rutinas para círculos y rectángulos

Digigraph
Sólidamente construido en metal y plástico, con un tablero para calcar de madera, el paquete Digigraph incluye varias hojas de trabajos para realizar prácticas

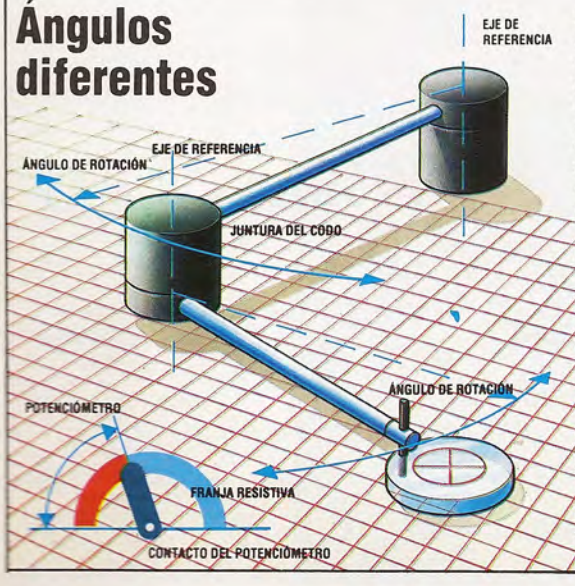
tubos de aluminio, con un disco de perspex en el extremo del puntero. La imagen a calcar se coloca sobre el tablero. Este sistema es el más fácil de utilizar de los cuatro que analizamos en este capítulo; el brazo se mueve suavemente y, a través del disco de perspex, la imagen resulta claramente visible. El software suministrado es menos sofisticado que el que se vende con el Robot Plotter, pero posee facilidades similares. El ordenador no almacena los movimientos del trazador como líneas separadas, sino que las imágenes se dibujan directamente en la pantalla, y el medio para salvar y cargar imágenes no es otro que salvar y guardar la zona de memoria de pantalla. Esto significa que no se puede editar fácilmente una imagen, pero posee la ventaja de que no se utiliza más memoria para dibujar una imagen compleja que para dibujar una imagen sencilla, lo que es muy útil tratándose del BBC Micro, tan escaso de memoria. El sistema Digigraph también incluye varias "hojas de trabajo", que el usuario puede usar para hacer prácticas con el trazador. Junto con los excelentes manuales, estas hojas permiten dominar el sistema dedicando a tal efecto un mínimo esfuerzo.

El trazador RD Labs se comercializa en dos formas: una para el BBC y otra para el Spectrum. Ambas versiones son, virtualmente, kits para ser montados en casa: sólo se suministra el brazo tiralíneas, junto con un material de relleno autoadhesivo que permite pegar el brazo a una base apropiada. El brazo es de material plástico y muy flexible, pero funciona con razonable precisión. El puntero es un dispositivo plástico con dos rayas cruzadas cuya utilización es más bien incómoda.

El precio de la versión para el BBC incluye el brazo propiamente dicho, un manual y una cassette con software. Las imágenes se almacenan de forma similar al Robot Plotter, con el resultado de que se vuelve a plantear el problema de la escasez de memoria. El software es bastante complejo y contiene rutinas para dibujar círculos, rectángulos y líneas así como una facilidad para animación. Ésta utiliza la capacidad del BBC de redefinir la paleta de colores para animar secuencias breves y simples. Una cassette de demostración descubre todas las características de este trazador.

La versión para el Spectrum es la más barata de las cuatro. El paquete incluye un accesorio convertidor de analógico a digital (el BBC Micro posee uno ya incorporado). En la mayoría de los otros sentidos, el modelo para el Spectrum es muy similar a la versión para el BBC. Se puede obtener un dibujo continuo, pero es difícil hacerlo satisfactoriamente debido a la baja respuesta del software, y una curva suave tiende a visualizarse en la pantalla como una serie de líneas rectas. No obstante, están disponibles el resto de facilidades (para dibujar círculos, rectángulos, etc.). El software no produce un aviso de error (a diferencia de los programas nor-

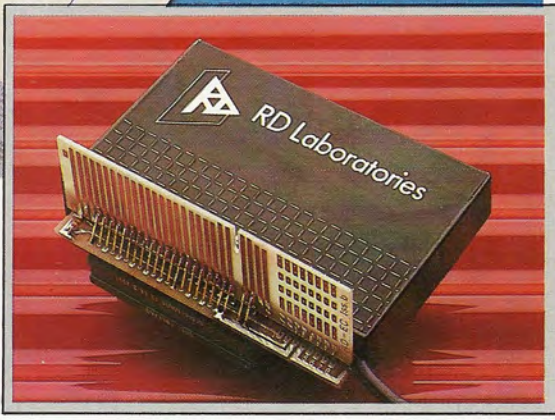
Ángulos diferentes



Ángulo de incidencia
A medida que se va desplazando el cursor de trazado a través del original a calcar, deben cambiar los ángulos entre los brazos del trazador y los ejes principales del tablero de trabajo. Estos ángulos se traducen a las correspondientes corrientes eléctricas mediante los potenciómetros de las junturas de los codos

Ángulo de resistencia
Cada codo del trazador contiene un potenciómetro o resistencia variable. A medida que el brazo del trazador se mueve, también se mueve el contacto del potenciómetro a través de la franja resistiva, haciendo que la corriente que pasa a través de la franja varíe proporcionalmente al ángulo barrido

Trazador digital RD
Esta versión para el BBC Micro está construida con sencillez en plástico ligero; se proporciona una banda de cinta adhesiva, que no se puede quitar una vez usada, para fijarlo a la mesa de trabajo



Interface RD para el Spectrum

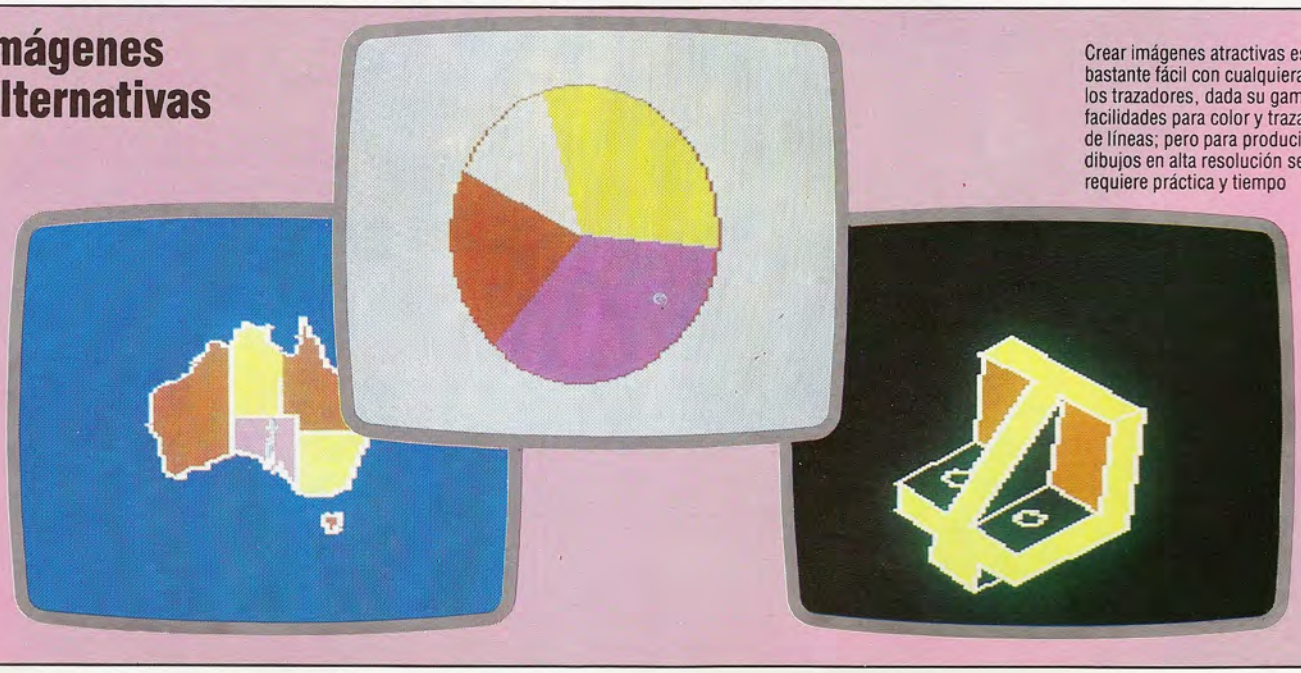
El trazador RD para el Spectrum es exactamente igual a la versión para el BBC cuya fotografía se incluye en este capítulo, pero se conecta mediante su propia interface

males para el Spectrum) si se utilizan estas instrucciones para trazar líneas que salen de los límites de la pantalla.

Es posible escribir programas que lean posiciones directamente de cualquiera de estos trazadores, lo que resulta útil para ciertos fines especiales. No obstante, esto es difícil de conseguir en el Spectrum, porque este ordenador, al contrario del BBC, no posee puerta para palanca de mando y su BASIC carece de las instrucciones pertinentes.

Para los usuarios de un Spectrum, la elección de un trazador se limita al modelo RD Labs, pero éste trabaja suficientemente bien. El usuario del BBC dispone de una gama de opciones más amplia: el tiralíneas RD Labs es el más económico de los tres, pero su construcción es, en cierta manera, endeble; el Robot Plotter está construido sólidamente y tiene el apoyo de un software inteligente, si bien no es particularmente fácil de utilizar; el Digigraph tiene un software menos sofisticado pero es, con mucho, el más fácil de utilizar.

Imágenes alternativas



Crear imágenes atractivas es bastante fácil con cualquiera de los trazadores, dada su gama de facilidades para color y trazado de líneas; pero para producir dibujos en alta resolución se requiere práctica y tiempo

Consulta privada

La comunicación con los ordenadores mediante el lenguaje corriente será pronto una realidad. Veamos un programa que ilustra sus principios

Uno de los pioneros de la informática, Alan Turing, propuso un famoso test de inteligencia artificial: afirmó que si una máquina es capaz de conversar con un ser humano sin que éste pueda darse cuenta de que está hablando con una máquina, entonces se puede afirmar que esa máquina es "inteligente". Hasta ahora, no hay ninguna máquina que haya superado esta prueba con total éxito, pero existen varios programas que se acercan a la misma apartando el interés del hombre por la conversación de las respuestas de la máquina, de manera que sea éste quien mantenga la conversación. Esto puede parecer rebuscado, pero resulta sorprendentemente eficaz. Intente recordar alguna conversación que haya mantenido y le haya producido satisfacción, y probablemente descubrirá que fue usted quien habló la mayor parte del tiempo, mientras

Programa

```
100 GOSUB 2000: REM INIC
200 FOR L = 1 TO 2*LT STEP 2
300 PRINT O$: INPUT I$
400 GOSUB 3000: REM ANALISIS
500 NEXT L
1000 PRINT TAB(5);"-----LA SESION HA TERMINADO-----"
1100 PRINT TAB(3);"-----PULSE CUALQUIER TECLA PARA
     CONTINUAR-----"
1200 A$ = INKEY$: IF A$ = "" THEN GOTO 1200
1300 GOSUB 5000: REM INFORME
1900 END
1999 REM ****************************************
2000 REM **              S/R INICIALIZAR          **
2001 REM ****************************************
2050 LT = 10: AN = 10: T9 = 500: EX = 2*LT
2100 DIM R$(AN): DIM H$(2*LT)
2200 DATA "SI..","DIGAME ALGO
     MAS..","CONTINUE..","Y..","POR TANTO.."
2250 DATA "ES ESO IMPORTANTE..","POR QUE LE
     PREOCUPA ESO.."
2300 DATA "POR FAVOR EXPLIQUEME ESO..","POR QUE LO
     DICE.."
2350 DATA "Y ESO EN QUE LO AFECTA A USTED.."
2500 FOR K = 1 TO AN: READ R$(K): NEXT K
2600 CLS: O$ = "HOLA - CUAL ES SU PROBLEMA.."
2950 RETURN
2999 REM ****************************************
3000 REM **              S/R ANALISIS             **
3001 REM ****************************************
3100 IF I$ = "ADIOS" THEN EX = L-1: L = 2*LT: RETURN
3200 H$(L) = O$: H$(L + 1) = I$
3300 R9 = INT(AN*RND + 1): IF R9 = R0 THEN GOTO 3300
3400 O$ = R$(R9): R0 = R9
3950 RETURN
4999 REM ****************************************
5000 REM **              S/R INFORME              **
5001 REM ****************************************
5050 CLS: PRINT TAB(10);"**INFORME**"
5100 FOR K = 1 TO EX STEP 2
5150 PRINT TAB(5);H$(K)
5200 PRINT H$(K + 1)
5300 FOR D = 1 TO T9: NEXT D
5400 NEXT K
5900 RETURN
```

Michael Brownlow

que la otra persona tan sólo se limitó a proporcionar "ruidos" o señales conversacionales. Éste es el principio sobre el que se basan muchas clases de psicoterapias y asesoramientos.

El programa que listamos aquí simula el comportamiento de un terapeuta completamente no directivo o, dicho de otra manera, un terapeuta que no *dirige* la conversación, sino que mantiene el flujo de la misma mediante respuestas tales como "HÁBLEME MÁS..." y "¿POR QUÉ ESO ES IMPORTANTE?" Conserva una grabación de la charla y la reproduce después de cierto número de intercambios, o bien cuando usted digita "ADIÓS".

El programa se puede desarrollar en la dirección de la pseudointeligencia haciendo que analice las entradas del usuario y elija una respuesta adecuada. Esto ya lo hace en la línea 3100 verificando si la palabra es "ADIÓS". Nosotros podríamos ampliar este análisis verificando las palabras "SÍ" y "NO" y produciendo señales más específicas en las respuestas; algo tan sencillo como "¿POR QUÉ?" o "¿POR QUÉ NO?" funcionaría muy bien. A continuación podríamos comprobar si el usuario está repitiendo una respuesta y, de ser así, replicarle en consecuencia o dar por finalizada la sesión. Podríamos verificar si una respuesta termina en un signo de interrogación o de exclamación, y replicar "¿POR QUÉ ME PREGUNTA ESO?" o "¿POR QUÉ SE EXCITA?"

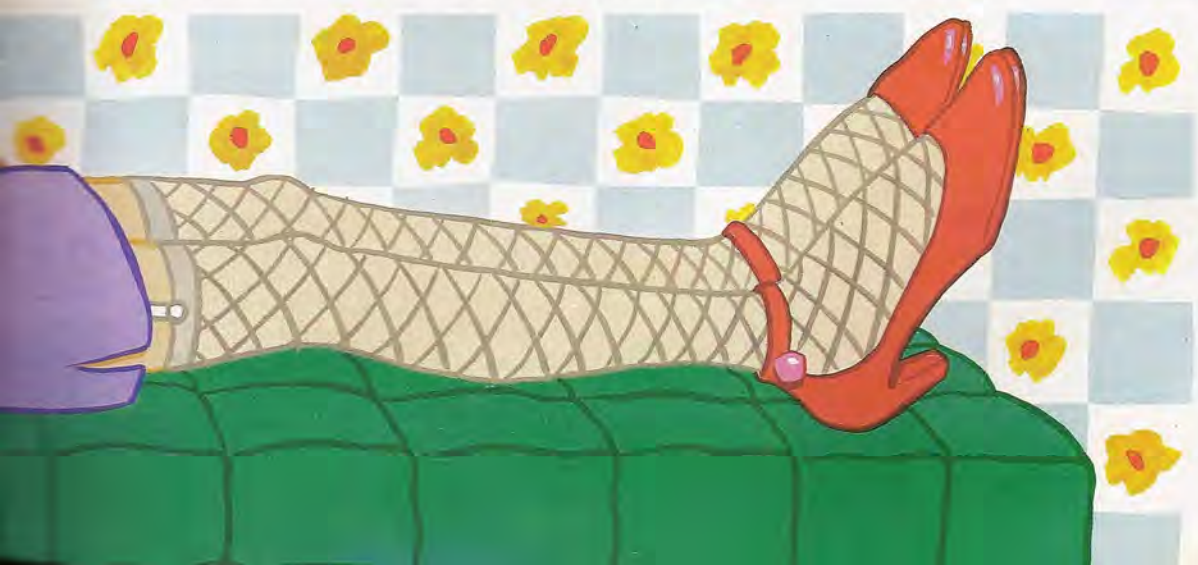
Ahora podríamos preparar una tabla de palabras clave y buscar la respuesta para estas palabras, produciendo una respuesta específica tomada de otra tabla en caso de encontrarla. La selección de palabras clave y de respuestas depende del tipo de conversación que espere mantener; el modo en que su selección afecte a las respuestas del usuario le puede proporcionar algunos indicios esclarecedores acerca de su propio subconsciente o del subconsciente de sus amigos. El inconveniente de este método (y el de todos los métodos que deben buscar el texto) es el tiempo que lleva analizar incluso la oración más corta. En este caso el factor limitante es la velocidad del BASIC, y todo análisis serio exige un programa en lenguaje máquina. Podemos, sin embargo, tolerar demoras en aras de la investigación.

Si pretendemos analizar las palabras del usuario, entonces son muchos los enfoques que podemos adoptar; probablemente el más interesante de éstos sea el *análisis sintáctico*, o sea la reducción de una oración en las partes que la componen, como pronombre, verbo o sustantivo. Ello exige un cuerpo de reglas sintácticas y gramaticales, y tablas de pronombres, preposiciones, conjunciones, transformaciones de palabras, etc., y no es una cuestión sencilla. No obstante, sería fácil elegir la palabra más larga de la respuesta y pedirle al usuario que explique las sensaciones que la misma le produce; es probable que la palabra más larga, en el caso de una oración simple, sea también la más importante. Tal vez pudiéramos introducir una mejora eligiendo al azar una palabra entre todas las palabras que tengan más de, pongamos por caso, cinco letras, o bien utilizar la palabra que aparezca a continuación de "YO", o "MI" o "SU".

Elegir uno de estos métodos y refinarlo es un fascinante ejercicio de programación. Se obtiene un panorama muy claro sobre la enorme complejidad del lenguaje hablado y de su análisis, y puede llegar a sentirse insatisfecho con el BASIC como herramienta analítica. Ello le puede conducir a acercarse a otros lenguajes de programación, tales como el LISP y el PROLOG, que están estructurados de forma muchísimo más sutil, precisamente debido a la necesidad de hacer frente a la complejidad del lenguaje hablado y el pensamiento. Además de esto, por supuesto, también tendrá la oportunidad de hablar todo el tiempo que lo desee con el conversador perfecto: ¡alguien que siempre lo escucha!

```
3420 IF LEFT$(I$,2) = "SI" THEN O$ = "POR QUE ESTA UD
     TAN SEGURO..": RETURN
3450 IF LEFT$(I$,2) = "NO" THEN O$ = "POR QUE NO..":
     RETURN
3500 Z$ = RIGHT$(I$,1)
3520 IF Z$ = "?" THEN O$ = "POR QUE ME LO
     PREGUNTA.."
3550 IF Z$ = "!" THEN O$ = "POR QUE LO PERTURBA
     ESO.."
3600 IF R9 < AN/4 THEN GOSUB 4000
3950 RETURN
3999 REM ****************************************
4000 REM **          S/R PALABRA MAS LARGA        **
4001 REM ****************************************
4050 W = 1: H = 1: WL = 1: S = 1
4100 I$ = I$ + " ": L9 = LEN(I$)
4120 FOR C = 1 TO L9: FOR P = C TO L9
4150 Z$ = MID$(I$,P,1)
4170 IF Z$ = " " THEN W = P-C: H = C: C = P: P = L9
4200 NEXT P
4200 IF W > WL THEN WL = W: S = H
4250 NEXT C
4270 O$ = "QUE SIGNIFICA " + MID$(I$,S,WL) + "PARA
     UD.."
4450 RETURN
```

Agregue estas líneas al primer programa para detección de SI/NO y de la palabra más larga



```
**INFORME**
HOLA—CUAL ES SU PROBLEMA
EL HALCON NO PUEDE OIR AL HALCONERO
    DIGAME ALGO MAS..
LAS COSAS SE SEPARAN— EL CENTRO NO AGUANTA
    CONTINUE..
LA MERA ANARQUIA SE LIBERA POR EL MUNDO
    POR FAVOR EXPLIQUEME ESO..
LOS MEJORES CARECEN DE CONVICCIONES
    ES ESO IMPORTANTE..
LOS PEORES ESTAN LLENOS DE APASIONADA
    INTENSIDAD SI
SEGURAMENTE HAY UNA REVELACION A MANO
    POR FAVOR EXPLIQUEME ESO..
SEGURAMENTE LA SEGUNDA VENIDA ESTA AL
    LLEGAR PORQUE ES ESO IMPORTANTE...
Y QUE TORPE BESTIA— SU HORA LLEGO AL FIN
    CONTINUE..
CAMINA HACIA BELEN PARA NACER
```

Pensamientos estimulantes
Hemos utilizado algunos versos del poema de W. B. Yeats *The second coming* (La segunda venida) para ilustrar la capacidad de un programa para estimular al usuario a pensar

Complementos al BASIC
Este programa está escrito en BASIC Microsoft. Los usuarios del Spectrum deben insertar LET en todas las sentencias de asignación y ajustar los valores de TAB

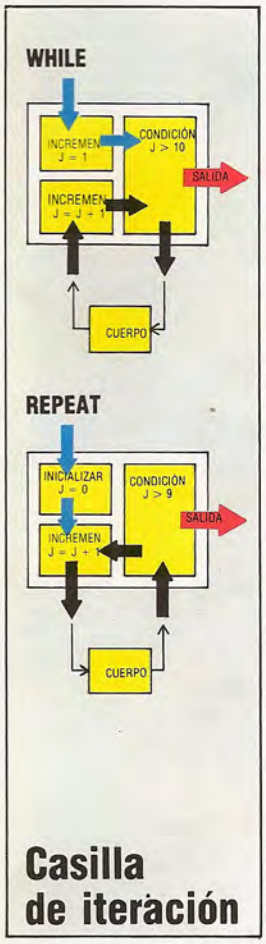
Spectrum
Reemplazar DIM R$(AN):DIM H$(2*LT) por DIM R$(AN,30):DIM H$(2*LT,100)
Reemplazar LEFT$(I$,2) por I$(TO 2)
Reemplazar RIGHT$(I$,1) por I$(LEN(I$))
Reemplazar MID$(I$,P,1) por I$(P)
Reemplazar MID$(I$,S,WL) por I$(S TO S + WL−1)

Commodore Vic-20 y 64
Reemplazar CLS por PRINT CHR$(147)
Reemplazar INT(AN*RND + 1) por INT(RND(1)*AN + 1)
Reemplazar A$ = INKEY$ por GET A$

BBC Micro
Reemplazar A$ = INKEY$ por A$ = INKEY$(0)
Reemplazar INT(AN*RND + 1) por RND (AN)

Líneas de bucle

En esta ocasión analizaremos diversas formas de clasificar los bucles e introduciremos un nuevo símbolo para diagramas de flujo: la casilla de iteración



La iteración, o bucle, es una de las estructuras esenciales de todos los lenguajes de programación. En el curso ya hemos dicho anteriormente que en un algoritmo se utiliza un bucle cada vez que una decisión dirige el flujo de control por un camino que, a la larga, lo lleva de regreso a la misma decisión inicial. Esto describe adecuadamente la estructura: la ejecución repetida de un cuerpo de código. Sin embargo, esta definición no abarca todas sus formas, que son muchas. Dado que los bucles son una estructura primitiva tan esencial que probablemente comprendan el 60 % de toda la actividad del procesador, es sumamente útil analizarlos en mayor detalle. Prestaremos especial atención al efecto que ejercen sobre la estructura del programa-algoritmo general y a las diversas formas en las que se pueden construir y clasificar.

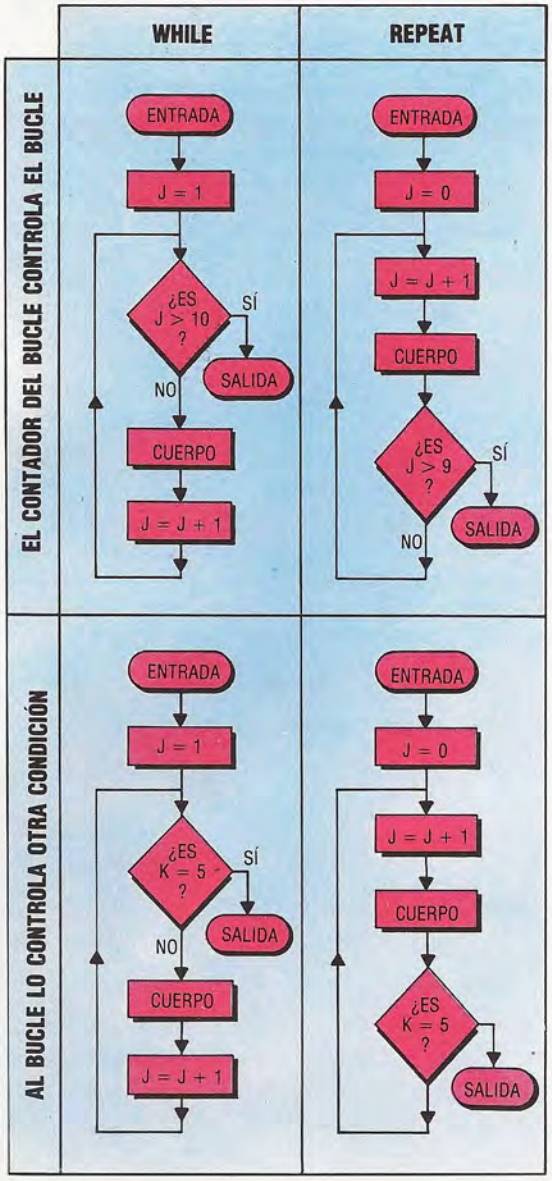
Los bucles se suelen dividir en dos clases, según su similitud con las dos estructuras de bucles de los lenguajes de alto nivel, RETURN...UNTIL y WHILE...ENDWHILE; ambos tipos se utilizan en PASCAL, y el bucle REPEAT está implementado en el BASIC del BBC y del Oric. Los dos tipos difieren en la colocación de la condición de salida del bucle: en un bucle REPEAT la condición va al final del cuerpo del bucle, mientras que en un bucle WHILE va al principio del mismo. Esto significa que el cuerpo de un bucle REPEAT, una vez se ha entrado en él, siempre se ejecutará al menos una vez, mientras que el cuerpo de un bucle WHILE no se ejecutará necesariamente. Esta primera diferencia a la que nos referimos se puede apreciar con bastante claridad en el diagrama de flujo.

Otra forma de clasificar los bucles atiende al hecho de si la variable que actúa como contador del bucle se utiliza en la condición de salida del bucle, o si hay alguna otra condición que controle esta salida. Esto es más difícil de ver en un diagrama de flujo convencional de tipo lineal. En realidad, la clase de diagrama de flujo de "sentido común" con el cual nos hemos familiarizado describe los bucles de una forma que no es nada útil. En estos diagramas de flujo, un bucle tiene exactamente el mismo aspecto que una simple bifurcación, y a menudo es necesario examinar el algoritmo con todo detalle para distinguir a ambos.

Existe una notación más clara llamada "casilla de iteración" que señala con toda claridad el comienzo de un bucle y elimina la confusión entre bucle y bifurcación. Consta de tres casillas unidas: la primera casilla indica la inicialización del contador, la segunda muestra el incremento del contador, y la tercera contiene la condición de salida del bucle. De esta forma se pueden reflejar tanto los bucles REPEAT como los WHILE. Ambos difieren en el flujo de control a través de las casillas, y la casilla de la condición indica si el bucle está controlado mediante el contador o no. Estos puntos se pueden apreciar claramente en el diagrama.

En un bucle REPEAT el flujo de control es "inicializar-cuerpo-condición-cuerpo-condición", mientras que el de un bucle WHILE es "inicializar-condición-cuerpo-condición-cuerpo". Esto se puede observar en la forma en que las líneas de control salen y vuelven a entrar en la casilla de iteración, con el cuerpo del bucle "colgando" de ellas.

Clasificación de los bucles



Billar informatizado

A primera vista el billar puede parecer un juego nada apropiado para la simulación por ordenador, pero la realidad es diferente...

A cualquiera que haya jugado alguna vez al *snooker* (billar) la idea de informatizarlo le parecerá un disparate. Para un jugador profesional constituirá un sacrilegio. ¿Cómo, se preguntarán, se podrían cuantificar y programar todas las sutiles curvas de la trayectoria de una bola o todas las variaciones de las características de la mesa? ¡No, por cierto, dentro de los límites de un Spectrum de 16 Kbytes! Sin embargo, se puede pensar en el juego del *snooker* como una aplicación relativamente sencilla del principio de la conservación del impulso, y un ordenador puede manipular las matemáticas del mismo con toda facilidad.

Cuando se comienza a jugar con el *Snooker* de Visions se visualiza en pantalla un plano de la mesa, con las bolas representadas por círculos de distintos colores. El jugador dirige la bola blanca desplazando una cruz por la pantalla hasta una posición determinada, definiendo de esta manera el camino que seguirá la bola. La bola blanca es disparada entonces hacia su objetivo con una fuerza que está determinada por la duración del período de tiempo durante el cual se mantiene pulsada la tecla.

Después de haber dado unos pocos golpes, un jugador avezado se dará cuenta de dos importantes problemas. El primero de ellos es que el ordenador (ya sea un Spectrum, un BBC o un Commodore) no puede calcular los ángulos de movimiento ni la resistencia por rozamiento suficientemente bien como para dar resultados convincentes en una pantalla de televisión. Ello significa que muchos golpes dan resultados bastante imprevisibles (si bien esto también puede suceder en el juego real). El jugador se dará cuenta de que no puede dar ciertos golpes, que son imposibles en el ordenador. Para compensar esto, las troneras son proporcionalmente más grandes de lo que deberían ser.

El otro problema, que en la práctica es más profundo, es que el jugador carece de perspectiva en su visión de la mesa. Ello significa que cuando le toca dar un golpe se encuentra privado de la ventaja que significa poder mirar a lo largo del taco. Y debido a que una pantalla de televisión es ligeramente curva, juzgar los ángulos resulta el doble de difícil. Es probable que, por todo cuanto acabamos de exponer, los primeros intentos que realice con este juego sean sumamente frustrantes.

No obstante, después de haberse aceptado el hecho de que el juego se parece muy poco al billar auténtico y una vez se han dominado algunas de sus pocas excentricidades, *Snooker* se convierte por derecho propio en un juego de estrategia fascinante. Se comienzan a apreciar algunas sutiles características de la programación, en especial la posibilidad de darle efecto a la bola, aunque los resultados en este caso vuelven a ser impredecibles.

Las tres versiones del juego se diferencian en formas que reflejan los puntos fuertes y los puntos débiles de las máquinas con las que se está jugando. Sorprendentemente, el juego utiliza poca memoria y, por lo tanto, se carga rápidamente en las tres máquinas, incluso en el Commodore, tan conocido por su baja velocidad de carga. Así que la calidad de cada versión refleja las capacidades para gráficos de cada máquina y no su capacidad de memoria.

En este sentido el Spectrum es el más flojo. La mesa es pequeña, y no todas las bolas se visualizan con sus colores auténticos. La versión para el Commodore es mejor, con una mesa más grande, colores realistas y bolas representadas a una escala más exacta. En esta versión el marcador está ilustrado de forma más llamativa y al comienzo del juego se ofrece una opción de demostración (aunque debe completar una partida antes de que se le devuelva el control). La versión para el BBC es la mejor de las tres, con una mesa que cubre virtualmente todo el ancho de la pantalla y fortalecida por un control del taco mucho más preciso.

El juego posee capacidad para uno o dos jugadores, si bien la opción de un único jugador sólo tiene sentido con fines de aprendizaje. Dado que existen tantos juegos por ordenador sumamente antisociales, este juego para dos jugadores es muy valioso en este sentido. Lamentablemente, Visions no ha creído conveniente proporcionar con el software los batines clásicos. El juego da la impresión general de ser digno de consideración, pero cabe la sospecha de que no se ha sabido aprovechar al máximo su potencial, en especial por haber tanta memoria sin utilizar.

Snooker: Para el Commodore 64, el Spectrum de 16 K y el BBC Micro
Editado por: Visions (Software Factory) Ltd, 1 Felgate Mews, Studland Street, London
Autor: Tim Bell (adaptación de Andy Williams para el BBC)
Palancas de mando: Opcionales
Formato: Cassette

La esencia del billar es apuntar con precisión y calibrar con exactitud los tiros. En el *snooker* de Visions se debe situar en la mesa un cursor de puntería mientras se controla la fuerza del golpe a base de pulsar la tecla de disparo más o menos tiempo

De esquina a esquina

BBC Micro

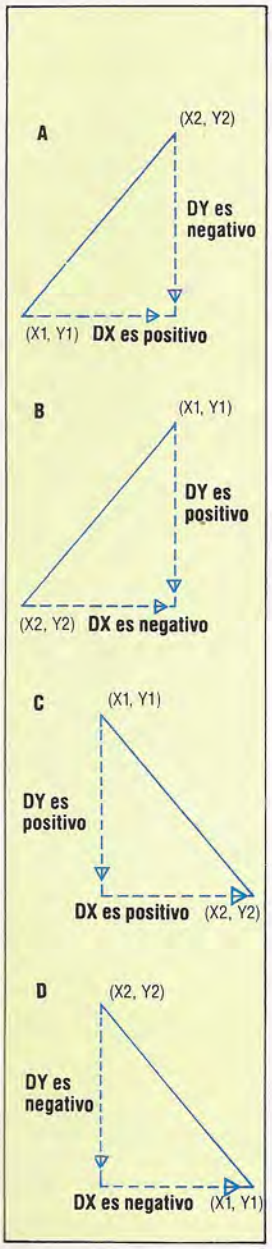
Commodore 64

Spectrum

Ian McKinnell

Rectas y pendientes

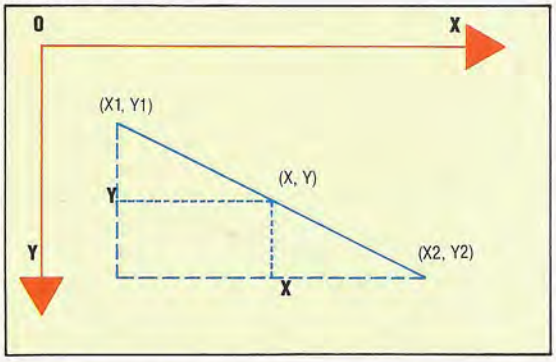
Ya ofrecimos una rutina para el Commodore 64 en lenguaje máquina. La que aquí estudiamos nos servirá para trazar todo tipo de líneas



Cuatro en uno
Cuando el valor absoluto (sin signo) de la pendiente (Y2−Y1)/(X2−X1) es mayor que uno conviene que DX sea positivo, y para ello emplearemos uno de estos algoritmos:
A) Volver a calcular —(DY y decrementar Y a partir de (X1,Y1) hasta alcanzar Y2
B) Intercambiar (X1,Y1) con (X2,Y2) y repetir cálculos
C) Incrementar Y partiendo de (X1,Y1) hasta alcanzar Y2
D) Intercambiar (X1,Y1) con (X2,Y2) y repetir los cálculos

La manera más obvia de trazar una línea de un punto a otro es calculando su pendiente según las coordenadas de los extremos. Así podemos pasar del punto de partida al siguiente mediante un incremento de la abscisa X y otro incremento proporcional de la ordenada Y, que obtendremos de acuerdo a la pendiente. De este punto pasaremos a trazar los sucesivos, hasta llegar al otro extremo de la línea. Esto supone unas sencillas nociones matemáticas que ahora exponemos.

Llamaremos a los puntos extremos (X1,Y1) y (X2,Y2) (no se olvide que un punto está definido por sus *dos* coordenadas), y pendiente P al cociente (Y2-Y1)/(X2-X1). El dibujo muestra una línea con sus puntos extremos y otro punto cualquiera (X,Y):



La pendiente de la semirrecta que va desde el punto inicial (X1,Y1) a ese punto cualquiera es P = (Y−Y1)/(X−X1) que, naturalmente, es idéntica a la calculada para la recta entera. Recordando cómo se despeja en las ecuaciones podemos obtener el valor de la ordenada Y:

$$(Y-Y1)/(X-X1) = P$$
$$(Y-Y1) = (P(X-X1)$$
$$Y = Y1 + P(X-X1)$$

En el caso de que el paso de X a X1 mida la unidad (o sea, los incrementos de la abscisa X valgan uno), entonces

$$Y = Y1 + P$$

En conclusión: si comenzamos por el extremo (X1,Y1) y vamos incrementando la X, los valores sucesivos de la Y se calculan añadiendo repetidamente la pendiente de la línea.

Puede que a usted le tiente probar esta técnica en BASIC mediante el empleo de la rutina Plotsub de la página 817. Pero este método tiene algunas dificultades:

- Si el valor de X2 es menor que el de X1, deberemos decrementar en lugar de incrementar.
- Si la pendiente es mayor que la unidad, la línea no será continua. Esto se debe a que Y quedará incrementado a su vez en más de uno a cada incremento de X.
- Las pendientes negativas son difíciles de implementar en código máquina. Se puede, sin duda, emplear el complemento a dos para representar los cambios negativos en X e Y, pero como la ordenada Y sólo alcanza valores de 0 a 199, y la X de 0 a 319, lo que necesitaremos serán complementos a dos de *dos bytes*.
- Las líneas verticales no pueden trazarse, pues el cálculo de sus pendientes implica una división por cero (X2 = X1 en las líneas verticales).

Algunos retoques

De los problemas enumerados, el primero se evita intercambiando los dos puntos extremos antes de los cálculos (el de partida se convierte en el de llegada, y viceversa).

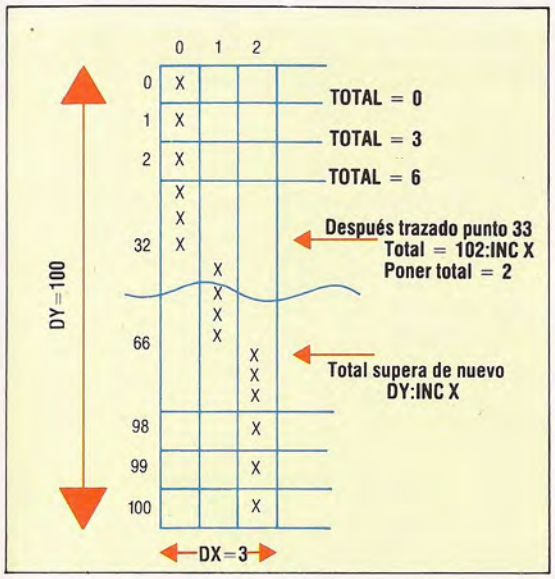
Para trazar una línea continua es necesario incrementar X y calcular Y si la pendiente es menor que uno, o bien incrementar Y y calcular X si la pendiente es mayor que uno. Con una bifurcación que sea consecuencia de tal condición (¿es P > 1?) evitamos usar la división y el complemento a dos de dos bytes.

Ya hemos ingeniado el procedimiento para trazar líneas con pendientes menores que uno. Consideremos ahora el caso de las pendientes que superan la unidad. Sean DX = X2 − X1 y DY = Y2 − Y1. Con sólo comparar DX y DY sabremos si la pendiente supera la unidad. Esto ocurrirá cuando DK < DY.

Para eludir ahora los valores negativos de DX y DY, el procedimiento es algo más complicado. Deberemos tener en cuenta cuatro casos posibles, siempre con la pendiente mayor que la unidad. La ilustración aparte describe estos casos. En nuestro programa debemos determinar con exactitud en qué caso nos encontramos para tomar, después, el camino más adecuado (suponiendo que deseamos mantener DX siempre positiva).

Caso a) Tomar DY como resultado de Y1 − Y2. Comenzando en (X1,Y1) iremos decrementando Y hasta que tome el valor Y2.
Caso b) Intercambiar los puntos y volver a empezar.
Caso c) Comenzaremos en (X1,Y1) incrementando Y hasta llegar a Y2.
Caso d) Intercambiar los puntos y volver a empezar.

Al comienzo de nuestro programa deberemos calcular DX y DY. Podemos colocar los bits en una determinada posición si DX o DY tienen valor ne-



gativo. El bit 0 de este registro (que llamaremos NEGREG) señalará el hecho de ser DY negativo, y el bit 1 indicará que DX es negativo. Seguidamente será consultado NEGREG para determinar en qué caso nos encontramos:

Caso	a	b	c	d
Valor de NEGREG	1	2	0	3
Equivalente binario	01	10	00	11

En cualquier caso debemos trazar la línea mediante el empleo de un bucle, comenzando en (X1,Y1), incrementando o decrementando Y según convenga y calculando el valor correspondiente de X antes de servirnos de la rutina Plotsub para trazar el punto. El bucle ha de terminar cuando Y haya alcanzado Y2.

De Y obtendremos X mediante una expresión reelaborada respecto de la primitiva ya explicada:

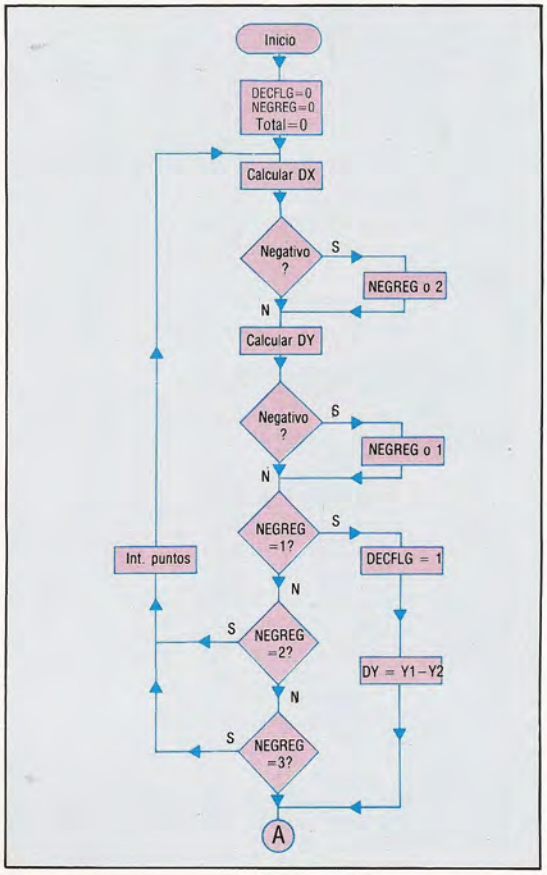
$$P = (Y-Y1)/(X-X1)$$
$$P(X-X1) = (Y-Y1)$$
$$(X-X1) = (Y-Y1)/P$$
$$X = X1 + (Y-Y1)/P$$

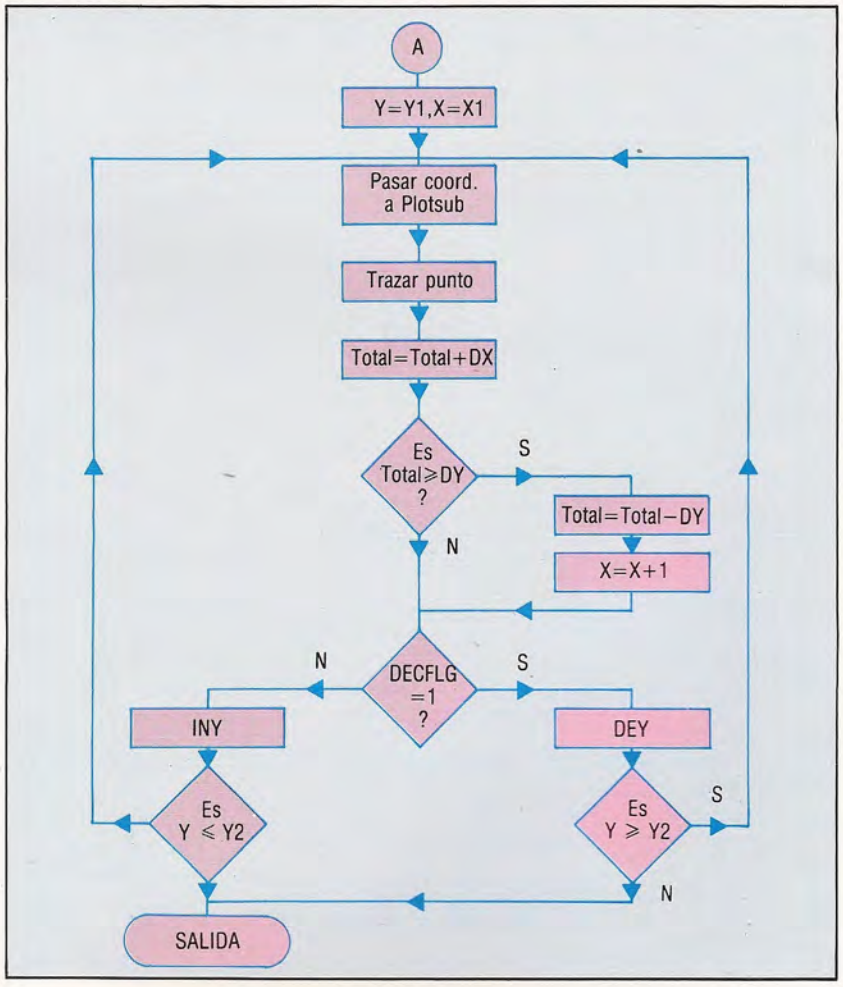
Cuando (Y−Y1) = 1, la expresión se simplifica: X = X1 + 1/P. Y puesto que P = DY/DX tendremos que X = X1 + DX/DY. Pero si P es mayor que uno entonces DY será mayor que DX, de donde se deduce que la división DX/DY da siempre cociente 0 y resto DX. Tomando un total actualizado de los restos a medida que se itera el bucle mencionado, sólo debemos incrementar X cada vez que la suma de los restos sea mayor que DY. El total actualizado se vuelve a reinicializar a continuación.

Esto puede parecer algo complicado, pero veamos el método en acción considerando un ejemplo donde (X1,Y1) = (0,0) y (X2,Y2) = (2,100). La línea se visualizará como compuesta por una serie de tres barras verticales. Una línea bastante escarpada por cierto, pero si disminuimos la longitud de las barras la línea será más compacta y cuando la pendiente sea uno cada barra equivaldrá a un pixel de longitud, dando la apariencia de una perfecta diagonal.

Trama lineal
En el dibujo de la izquierda mostramos cómo puede trazarse una línea vertical u horizontal con exactitud en un campo de puntos discretos. No obstante, una línea en cualquier otro ángulo debe trazarse como una serie de escalones. La línea del dibujo une los puntos (0,0) y (2,100) mediante tres escalones verticales idénticos

Control lineal
El diagrama de flujo de la derecha muestra el método de la subdivisión de la línea previendo los cuatro casos en que la pendiente es mayor que uno. Las variables TOTAL y DECFLG (*flag* de *dec*remento) serán empleados por la rutina en el trazado que contempla el diagrama de flujo inferior. El trazado de puntos comienza en (X1,Y1) y dado que DX se mantiene positivo, la Y se incrementa o decrementa según el estado de DECFLG





La rutina dentro del BASIC

Para utilizar la rutina (que llamaremos Linesub) deberá cargar tanto Linesub como Plotsub tal como se indica en el programa de demostración en BASIC. Denominaremos los programas finales en código objeto PLOTSUB.HEX y LINESUB.HEX. Una vez cargadas las rutinas en código máquina, este breve programa de demostración pedirá al usuario las coordenadas de los puntos, examinará la pendiente y si no es inferior a la unidad empleará Plotsub para limpiar la pantalla en alta resolución y establecer el color. La subrutina en la línea 2000 convierte las X en byte *lo* y byte *hi* antes de colocar (POKE) estos valores en posiciones separadas dentro de Linesub para los extremos de la línea. En la línea 3000 la subrutina imprime los valores de todas las posiciones utilizadas para las variables por Linesub.

Como una opción al mecanografiado del código fuente de Linesub y al ensamblaje de código máquina, el programa cargador de código máquina introduce este mismo programa en la memoria "leyendo" (READ) una serie de valores (DATA) y colocándolos (POKE) en la memoria. Escriba este programa y ejecútelo (RUN) para que Linesub quede cargado en la memoria.

Programa de demostración en BASIC

```
 10 REM ************************************
 12 REM **              LINESUB 64          **
 13 REM ************************************
 14 :
 15 DN = 8:REM PARA CASSETTE HAGA DN = 1
 20 IF A = 0 THENA = 1:LOAD"PLOTSUB.HEX",DN,1
 30 IF A = 1 THENA = 2:LOAD"LINESUB.HEX",DN,1
 50 INPUT"PRIMER PUNTO EXTREMO";X1,Y1
 60 INPUT"SEGUNDO PUNTO EXTREMO";X2,Y2
 70 GOSUB1000: REM MODO ALT RES
 80 GOSUB2000: REM LINESUB
 90 GETA$:IFA$ = ""THEN 90
 95 IF A$ = " "THEN200
100 REM **** PREPARAR PANTALLA ****
110 POKE49408,0:SYS 49422
120 GOSUB3000
125 GETA$:IFA$ = ""THEN125
127 GOTO50
140 :
200 REM **** TRAZAR TRIANGULO ****
205 XA = 30:YA = 10:XB = 310:YB = 98
210 XC = 90:YC = 180
220 GOSUB1000
230 X1 = XA:Y1 = YA:X2 = XB:Y2 = YB:GOSUB2000
240 X1 = XC:Y1 = YC:GOSUB2000
250 X2 = XA:Y2 = YA:GOSUB2000
255 GETA$:IFA$ = "" THEN 255
260 REM **** PREPARAR PANTALLA ****
270 POKE49408,0:SYS 49422
275 PRINTCHR$(147)
280 END
290 :
1000 REM **** USAR ALT RES ****
1010 POKE49408,1:POKE49409,1
1015 POKE49410,1
1020 SYS 49422
1030 RETURN
1040 :
2000 REM **** ENTRADA LINESUB ****
2010 MHI = INT(X1/256):ML0 = X1-256*MHI
2020 NHI = INT(X2/256):NL0 = X2-256*NHI
2030 POKE49920,ML0:POKE49921,MHI
2040 POKE49922,NL0:POKE49923,NHI
2050 POKE49924,Y1:POKE49925,Y2
2060 SYS 49934
2070 RETURN
2080 :
3000 REM **** IMPRIMIR VALORES ****
3001 RESTORE
3002 PRINTCHR$(147):REM LIMPIAR PANTALLA
3005 FORI = 0TO13
3010 READA$
3020 PRINTA$,PEEK(49920+I)
3030 NEXT I
3040 DATA X1LO,XIHI,X2LO,X2HI,Y1,Y2,DXLO
3050 DATA DXHI,DY,TEMP,TOTLO,TOTHI,NEGREG,DECFLG
3060 RETURN
```

Cargador de código máquina

```
 10 FOR I = 49934 TO 50371
 20 READA:CC = CC+A
 30 POKEI,A:NEXT
 50 READA:IFCC<>A THEN PRINT"ERROR SUMA
    COMPROBACION":END
100 DATA72,138,72,152,72,169,0,141,13
110 DATA195,141,12,195,141,10,195,141
120 DATA11,195,173,2,195,56,237,0,195
130 DATA141,6,195,173,3,195,237,1,195
140 DATA141,7,195,16,8,173,12,195,9,2
150 DATA141,12,195,173,5,195,56,237,4
160 DATA195,141,8,195,176,8,173,12,195
170 DATA9,1,141,12,195,173,12,195,201
180 DATA1,240,20,201,2,208,6,32,140
190 DATA196,76,19,195,201,3,208,19,32
200 DATA140,196,76,19,195,173,4,195,56
210 DATA237,5,195,141,8,195,238,13,195
220 DATA173,6,195,24,105,1,141,6,195
230 DATA173,7,195,105,0,141,7,195,238
240 DATA8,195,173,4,195,168,173,7,195
250 DATA201,1,240,115,173,6,195,205,8
260 DATA195,176,107,173,0,195,141,3
270 DATA193,173,1,195,141,4,193,152
280 DATA141,5,193,32,131,193,173,10
290 DATA195,24,109,6,195,176,8,141,10
300 DATA195,205,8,195,144,24,56,237,8
310 DATA195,141,10,195,173,0,195,24
320 DATA105,1,141,0,195,173,1,195,105
330 DATA0,141,1,195,173,13,195,201,1
340 DATA208,31,136,204,5,195,240,3,76
350 DATA161,195,152,141,5,193,173,0
360 DATA195,141,3,193,173,1,195,141,4
370 DATA193,32,131,193,76,134,196,200
380 DATA204,5,195,144,152,76,134,196
390 DATA173,0,195,141,3,193,173,1,195
400 DATA141,4,193,152,141,5,193,32,131
410 DATA193,173,10,195,24,109,8,195
420 DATA141,10,195,173,11,195,105,0
430 DATA141,11,195,173,10,195,56,237,6
440 DATA195,141,10,195,173,11,195,237
450 DATA7,195,141,11,195,48,15,173,13
460 DATA195,201,1,240,4,200,76,104,196
470 DATA136,76,104,196,173,10,195,24
480 DATA109,6,195,141,10,195,173,11
490 DATA195,109,7,195,141,11,195,173,0
500 DATA195,24,105,1,141,0,195,173,1
510 DATA195,105,0,141,1,195,205,3,195
520 DATA208,142,173,0,195,205,2,195
530 DATA208,134,104,168,104,170,104,96
540 DATA173,2,195,141,9,195,173,0,195
550 DATA141,2,195,173,9,195,141,0,195
560 DATA173,3,195,141,9,195,173,1,195
570 DATA141,3,195,173,9,195,141,1,195
580 DATA173,5,195,141,9,195,173,4,195
590 DATA141,5,195,173,9,195,141,4,195
600 DATA96,230
610 DATA50794:REM*SUMA COMPROBACION*
```

Listado assembly

```
                ;*****************************
                ;*****************************
                ;**                         **
                ;**        LINESUB 64       **
                ;**                         **
                ;*****************************
                ;*****************************

                PLTSUB  = $C183
                XLO     = $C103
                XHI     = $C104
                YLO     = $C105
                *       = $C300

                ;**** VARIABLES DE LINESUB ****

                X1LO    * = * + 1
                X1HI    * = * + 1
                X2LO    * = * + 1
                X2HI    * = * + 1
                Y1      * = * + 1
                Y2      * = * + 1
                DXLO    * = * + 1
                DXHI    * = * + 1
                DY      * = * + 1
                TEMP    * = * + 1
                TOTLO   * = * + 1
                TOTHI   * = * + 1
                NEGREG  * = * + 1
                DECFLG  * = * + 1

                ;**** LLEVAR REGISTROS A LA PILA ****

48              PHA
8A              TXA
48              PHA
98              TYA
48              PHA

                ;**** INICIALIZAR VARIABLES ****

                START
A9 00           LDA  #$00
8D 0D C3        STA  DECFLG
8D 0C C3        STA  NEGREG
8D 0A           STA  TOTLO
8D 0B C3        STA  TOTHI

                ;**** CALCULAR DX ****

AD 02 C3        LDA  X2LO
38              SEC
ED 00 C3        SBC  K1LO
8D 06 C3        STA  DXLO
AD 03 C3        LDA  X2HI
4C 86 C4        JMP  FINISH

                NODEY
C8              INP
CC 05 C3        CPY  Y2
90 98           BCC  PLOT

4C 86 C4        JMP  FINISH

                ;**** PENDIENTES MENORES DE 1 ****

                GLESS
AD 00 C3        LDA  X1LO
8D 03 C1        STA  XLO
AD 01 C3        LDA  X1HI
8D 04 C1        STA  XHI
98              TYA
8D 05 C1        STA  YLO
20 83 C1        JSR  PLTSUB

AD 0A C3        LDA  TOTLO
18              CLC
6D 08 C3        ADC  DY          ; TOTAL = TOTAL+DY
8D 0A C3        STA  TOTLO
AD 0B C3        LDA  TOTHI
69 00           ADC  #$00
8D 0B C3        STA  TOTHI       ; COGER ARRASTRE

AD 0A C3        LDA  TOTLO
38              SEC
ED 06 C3        SBC  DXLO
8D 0A C3        STA  TOTLO       ; TOTAL = TOTAL-DX
AD 0B C3        LDA  TOTHI
ED 07 C3        SBC  DXHI
8D 0B C3        STA  TOTHI
30 0F           BMI  ADD         ;RESULTADO NEGATIVO

AD 0D C3        LDA  DECFLG
C9 01           CMP  #$01
F0 04           BEQ  DECY
C8              INY
4C 68 C4        JMP  PAST
                DECY
88              DEY
4C 68 C4        JMP  PAST

                ADD
AD 0A C3        LDA  TOTLO
18              CLC
6D 06 C3        ADC  DXLO        ;SUMAR DX
8D 0A C3        STA  TOTLO
AD 0B C3        LDA  TOTHI
6D 07 C3        RDC  DXHI
8D 0B C3        STA  TOTHI
ED 01 C3        SBC  X1HI
8D 07 C3        STA  DXHI
10 08           BPL  NBRWDX
AD 0C C3        LDA  NEGREG
09 02           ORA  #$02        ;FLAG DX NEGATIVO
8D 0C C3        STA  NEGREG

                ;**** CALCULAR DY ****

                NBRWDX
AD 05 C3        LDA  Y2
38              SEC
ED 04 C3        SBC  Y1
8D 08 C3        STA  DY
B0 08           BCS  NBRWDY

AD 0C C3        LDA  NEGREG
09 01           ORA  #$01        ;FLAG DY NEGATIVO
8D 0C C3        STA  NEGREG
                NBRWDY

                ;**** COMPROBAR ESTADO DE NEGREG ****

AD 0C C3        LDA  NEGREG
C9 01           CMP  #$01
F0 14           BEQ  NEG1

C9 02           CMP  #$02
D0 06           BNE  FALSE
20 8C C4        JSR  SWOP
4C 13 C3        JMP  START

                FALSE
C9 03           CMP  #$03
D0 13           BNE  NEG0
20 8C C4        JSR  SWOP
4C 13 C3        JMP  START

                NEG1
AD 04 C3        LDA  Y1
38              SEC
ED 05 C3        SBC  Y2          :RECALCULAR DY
8D 08 C3        STA  DY                COMO Y1--Y2
EE 0D C3        INC  DECFLG      ;ESTABLECER DECFLG

                NEG0
AD 06 C3        LDA  DXLO
18              CLC
69 01           ADC  #$01        ; SUMAR 1 A DX
8D 06 C3        STA  DXLO
AD 07 C3        LDA  DXHI
69 00           ADC  #$00        ; COGER ARRASTRE
8D 07 C3        STA  DXHI
EE 08 C3        INC  DY          ; SUMAR 1 A DY
AD 04 C3        LDA  Y1
A8              TAY              ; HACER Y = Y1

                ;**** COMPROBAR SI DX >= DY ****

AD 07 C3        LDA  DXHI
C9 01           CMP  #$01
F0 73           BEQ  GLESS
AD 06 C3        LDA  DXLO
CD 08 C3        CMP  DY
B0 6B           BCS  GLESS

                ;**** TRAZAR PUNTO ****

                PLOT
AD 00 C3        LDA  X1LO
8D 03 C1        STA  XLO         ;PASAR BYTE LO DEX
AD 01 C3        LDA  X1HI
8D 04 C1        STA  XHI         ;PASAR BYTE HI DEX
98              TYA
8D 05 C1        STA  YLO         ;PASAR Y
20 83 C1        JSR  PLTSUB
AD 0A C3        LDA  TOTLO
18              CLC
6D 06 C3        ADC  DXLO        ;TOTAL = TOTAL + DX
B0 08           BCS  DOINX       ;EXCEDE LA SUMA 1 BYTE?
8D 0A C3        STA  TOTLO
CD 08 C3        CMP  DY
90 18           BCC  NOINX       ;EL TOTAL ES DY?
                DOINX
38              SEC
ED 08 C3        SBC  DY          ;TOTAL = TOTAL - DY
8D 0A C3        STA  TOTLO
AD 00 C3        LDA  X1LO
18              CLC
69 01           ADC  #$01
8D 00 C3        STA  X1LO        ;INCREMENTAR X1
AD 01 C3        LDA  X1HI
69 00           ADC  #$00
8D 01 C3        STA  X1HI

                NOINX
AD 0D C3        LDA  DECFLG
C9 01           CMP  #$01
D0 1F           BNE  NODEY       ;DECFLG = 1?
88              DEY
CC 05 C3        CPY  Y2
F0 03           BEQ  FINAL       ;SI Y = Y2 TRAZAR ULTIMO
4C A1 C3        JMP  PLOT
                FINAL            ;TRAZAR ULTIMO PUNTO
98              TYA
8D 05 C1        STA  YLO
AD 00 C3        LDA  X1LO
8D 03 C1        STA  XLO
AD 01 C3        LDA  X1HI
8D 04 C1        STA  XHI
20 83 C1        JSR  PLTSUB

                PAST
AD 00 C3        LDA  X1LO
18              CLC
69 01           ADC  #$01
8D 00 C3        STA  X1LO        ; INCREMENTAR X
AD 01 C3        LDA  X1HI
69 00           ADC  #$00
8D 01 C3        STA  X1HI        ; COGER ARRASTRE

CD 03 C3        CMP  X2HI
D0 8E           BNE  GLESS
AD 00 C3        LDA  X1LO        ; HA ALCANZADO X a X2?
CD 02 C3        CMP  X2LO
D0 86           BNE  GLESS

                ;**** SACAR REGISTROS DE LA PILA ****

                FINISH
68              PLA
A8              TAY
68              PLA
AA              TAX
68              PLA

60              RTS

                ;**** FIN DE PROGRAMA ****

                ;**** SUBRUTINA INTERCAMBIO PUNTOS ****

                SWOP
AD 02 C3        LDA  X2LO
8D 09 C3        STA  TEMP
AD 00 C3        LDA  X1LO
8D 02 C3        STA  X2LO
AD 09 C3        LDA  TEMP        ;INTERC. BYTES LO DE X
8D 00 C3        STA  X1LO

AD 03 C3        LDA  X2HI
8D 09 C3        STA  TEMP
AD 01 C3        LDA  X1HI
8D 03 C3        STA  X2HI
AD 09 C3        LDA  TEMP        ;INTERC. BYTES HI DEX
8D 01 C3        STA  X1HI

AD 05 C3        LDA  Y2
8D 09 C3        STA  TEMP
AD 04 C3        LDA  Y1
8D 05 C3        STA  Y2
AD 09 C3        LDA  TEMP        ;INTERCAMBIAR Y
8D 04 C3        STA  Y1

60              RTS

                ****FIN SUBRUTINA ****
```

Rutina Linesub en código máquina

La rutina en código máquina traza líneas con pendientes igual a uno, o mayores-menores que uno. No hay problema alguno con las líneas verticales pues, con el método que hemos descrito, se evita la división por cero. Para líneas con pendiente menor que la unidad son válidos los principios explicados. Lo único que hace falta es incluir una comprobación sobre si DX es mayor que DY (o sea, si la pendiente es menor que uno) con su correspondiente bifurcación.

- La rutina utiliza dos bytes para cada X, pues ésta puede superar el valor 255. Sin embargo, como Y no puede superar 199, cabe en un byte.
- Las posiciones que Plotsub utiliza para guardar las coordenadas de los trazados son asignadas al comienzo del listado en assembly. También se especifica la dirección del inicio de Plotsub.
- A los bits 0 y 1 de NEGREG se les asigna el valor apropiado según sea el resultado de DX y DY. Para acceder a cada bit sin afectar al resto se emplea la instrucción lógica ORA.
- Dado que Y puede contenerse en un solo byte, el registro Y se emplea para retener su valor durante el trazado. Los valores del registro no son alterados por Plotsub.

Nótese bien: para asegurarse de que Plotsub puede utilizarse con Linesub, deberá realizarse el siguiente cambio en el listado que usted conoce del código fuente (véase p. 819). Se debe incluir una instrucción bifurcadora entre SBC REMX y TAX.

Un cambio equivalente se realizará en BASIC, en el programa de demostración en BASIC insertando estas líneas:

```
5010 FOR I = HRSFLG TO
HRSFLG + 314:READ A
5380 DATA 169,7,237,8,
193,240,6,170,14,13,193
5410 DATA 38944:
REM*SUMA COMPROBACION*
```

Una historia clásica

Artic se fundó hace tres años con un capital de 20 libras: hoy es una sólida empresa con planes de ampliación a nivel mundial

En 1980 un estudiante de 17 años llamado Richard Turner empezó a escribir software. Los primeros juegos que escribió fueron *Battleships* (Batallas navales) y *Star trek* (Viaje a las estrellas) para el ordenador ZX80. Optó por escribir juegos de estrategia en vez de juegos más populares de estilo recreativo debido a las limitaciones de la máquina. Como él mismo explica: "El ZX80 limpiaba la pantalla cada vez que algo se movía, de modo que los únicos juegos que uno podía crear eran juegos para pensar y no juegos recreativos, que sólo salieron realmente con el Spectrum".

Su primer gran éxito lo obtuvo con el juego *ZX chess*, que lanzó en el verano de 1981 en la primera "feria del micro ZX". Turner utilizó sus recursos al límite: "La noche anterior todavía estábamos copiando cassettes, usando siete ZX81 y colocándolas en bolsitas de plástico con instrucciones que habíamos reproducido con la fotocopiadora de la escuela". El *ZX chess* fue un gran éxito y, según Turner, ganó 1 500 libras en la feria.

A fines de ese mismo verano Artic Computing se convirtió en una sociedad anónima, pero quedó relegada a un segundo lugar cuando Turner aceptó una beca de la Ford Motor Company para estudiar ingeniería eléctrica en el Imperial College de Londres. Sus estudios duraron sólo un año, al final del cual decidió tomarse otro de descanso para dirigir la empresa. Pero jamás volvió a la Universidad.

En un principio, Artic se dirigía desde el dormitorio de Richard, en la casa de sus padres, en Hull; pero cuando la lista de software de la empresa llegó a incluir 93 títulos, Turner decidió que había llegado el momento de contar con unas oficinas propias. En junio de 1983 la empresa se trasladó a sus oficinas actuales de Brandesburton (Humberside). Se modernizó el catálogo y se contrató más personal. En la actualidad Artic tiene empleadas a 15 personas, incluyendo a tres personas de televentas y cinco programadores de dedicación exclusiva, quienes perciben un sueldo además de los royalties.

Artic tiene planeado abrir su propia cadena de tiendas minoristas distribuidas por toda Gran Bretaña. Las tiendas se llamarán Artic Software Stations y venderán no sólo los juegos de Artic sino también productos de otras empresas. La primera tienda se abrió en julio de 1984 en Acton (West London) y es también la sede social de la empresa en Londres. Hay que destacar que el establecimiento no está situado en un sector de gran movimiento comercial y que está bastante alejado del centro comercial del West End londinense. Al preguntársele el motivo por el cual había elegido ese lugar en particular, Jeff Raggett, director de marketing de Artic para Londres, replicó que "un local céntrico costaría 300 o 400 libras a la semana, y esta tienda nos sale por muchísimo menos, de modo que no nos vemos obligados a vender muchas cassettes para cubrir los gastos. Mucha gente nos ha criticado, nos ha dicho que estábamos locos por el hecho de abrir tiendas, pero al menos así sabemos lo que es vender y podemos hablar con la gente sobre cuáles son los juegos que les gustan".

Otra innovación en la comercialización son los exhibidores de Artic. Estas unidades son cajas exhibidoras con capacidad para 64 cassettes. Actualmente Artic se los está vendiendo a los agentes de prensa, posibilitando que el público compre su software a nivel local en vez de tener que acudir a los grandes detallistas. Jeff Raggett afirma que estos exhibidores están obteniendo un notable éxito.

Artic piensa dirigir la comercialización de sus productos en el exterior ella misma en la medida de lo posible, y en estos momentos la empresa está considerando la posibilidad de una ampliación a nivel europeo. Para el mercado de América del Norte Artic ha firmado un contrato con dos casas de software ya establecidas, Softsync y la International Publishing Corporation.

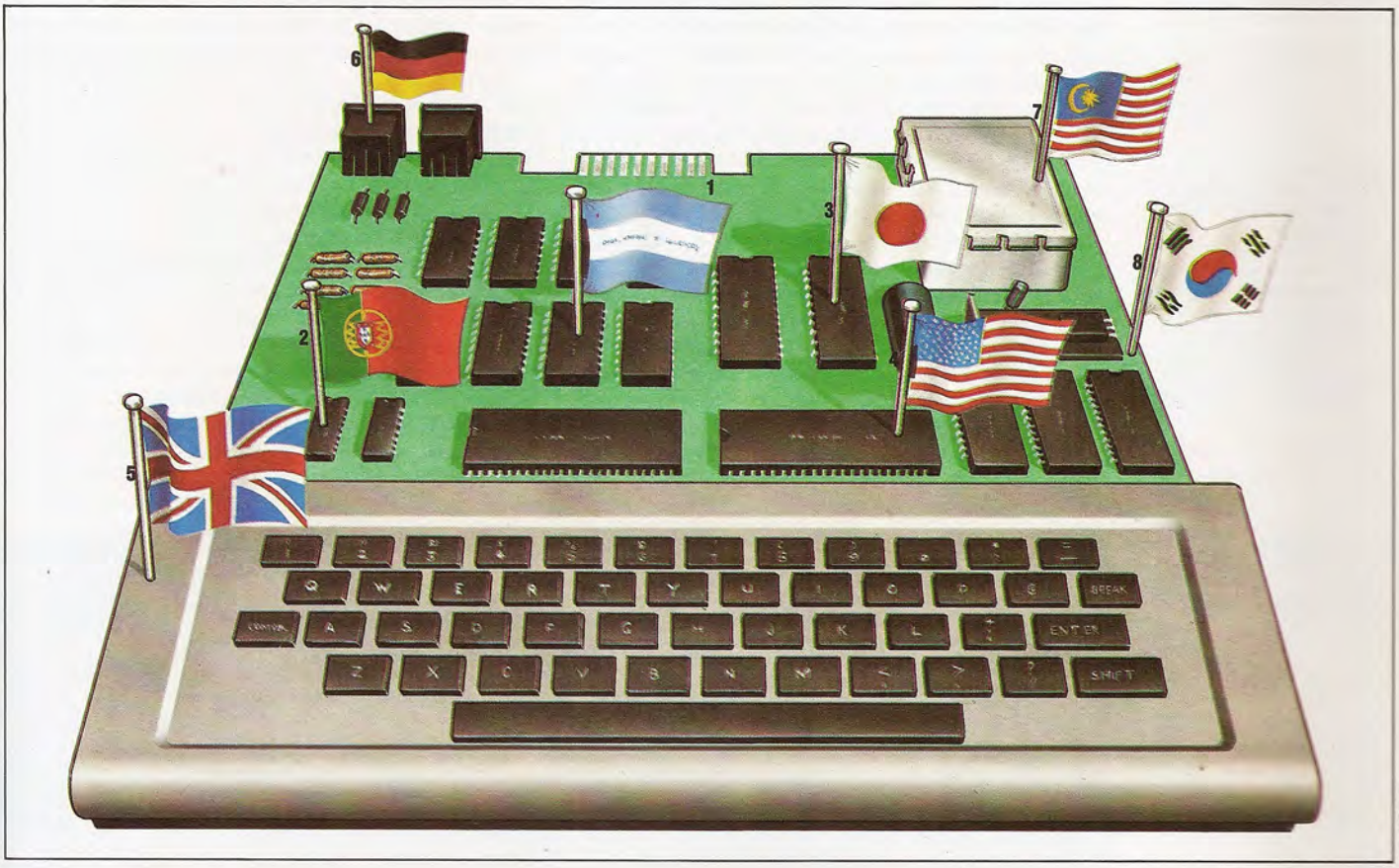
Hasta la fecha los mayores éxitos de ventas de Artic han sido *Bear Bovver* (del cual se han vendido más de 40 000 cassettes), *Galaxians* y *Gobbleman*. Recientemente la empresa ha lanzado un nuevo juego para el Spectrum llamado *World cup* (Copa del mundo), del que en sólo tres semanas se han vendido 5 000 ejemplares.

Los juegos de Artic
Una muestra de los juegos más destacados de Artic, incluyendo los exitosos *Bear Bovver* y *World cup* (Copa del mundo), una versión para el Spectrum del popular deporte del fútbol

Ian McKinnell

Richard Turner

Mercancías internacionales



Steve Cross

Una atenta mirada al origen de los componentes de un ordenador puede revelarnos una sorprendente variedad de países

Fabricar micros es un negocio multinacional. El Amstrad CPC 464, por ejemplo, que describimos en la página 909, se construye enteramente en Corea y una gran proporción de los BBC Micro de Acorn se construyen en Hong Kong. Sinclair siempre ha seguido la política de ser sólo una empresa de investigación y diseño, subcontratando la fabricación de componentes y el ensamblaje final.

La razón de ello es la necesidad de construir las máquinas de la forma más barata posible. Las consideraciones sobre la posterior fabricación son de capital importancia para los diseñadores de ordenadores al principio de todo el proceso. Con el fin de mantener reducidos los costos, la placa de circuito impreso debe ser pequeña y simple. Esto significa que el diseño debe incorporar la menor cantidad posible de chips. La razón no es el costo de los chips en sí mismos, sino porque montar un gran número de éstos en una placa es caro y puede hacer que el producto final sea menos fiable.

Este último punto es el motivo por el cual en los micros más populares se utilizan chips ULA (*Uncommitted Logic Array*: disposición lógica no comprometida). El ULA, aunque es un chip caro de diseñar y construir, sustituye en la placa a docenas de chips más pequeños.

La mayoría de los microchips se fabrican en California, donde se ha acuñado el término Silicon Valley (Valle del Silicio) para referirse a la zona donde tienen sus sedes o sus centros de investigación la mayoría de las empresas norteamericanas dedicadas a la microelectrónica. Una vez fabricados estos chips, es necesario encastarlos dentro de la carcasa de plástico o cerámica. Esta parte del proceso no exige el mismo nivel de pericia técnica y es un trabajo de mano de obra intensiva. Puesto que ésta es más barata fuera de Estados Unidos, los chips se embarcan hacia diversos países.

Finalizado el diseño de la placa, comienza la búsqueda de un subcontratista. La fabricación de pla-

Crisol de razas
Este microordenador imaginario se fabricó con elementos procedentes de muchos países diferentes. Se fabrican chips en: 1) El Salvador; 2) Portugal; 3) Japón; 4) Estados Unidos; la carcasa, el teclado y el montaje final se efectúan en 5) Gran Bretaña; los conectores son de 6) República Federal de Alemania; el modulador de RF se produce en 7) Malasia, y la placa de circuito impreso (PCB) se fabrica en 8) Corea del Sur

cas de circuito impreso, al igual que la fabricación de chips, es una labor compleja que implica grandes inversiones en maquinaria. Existen muchas empresas que se especializan en la fabricación de dichas placas. A partir de copias heliográficas detalladas de la placa, el fabricante produce las familiares placas verdes con franjas metálicas para unir los componentes electrónicos.

Incluso el diseño de la propia placa está supeditado a las limitaciones del costo de fabricación. Es posible hacer construir placas de capas múltiples o multinivel, en las que se superponen varias capas de metal separadas por capas de aislante. Sin embargo, esto es caro y los diseñadores lo evitan. Para las placas de ordenadores siempre se especifica un sistema de patrón de agujeros, en el que los agujeros para todos los cables están metalizados para mejorar el contacto eléctrico, debido a la fiabilidad que proporciona este sistema al producto acabado.

Las fuentes de alimentación eléctrica, los moduladores de televisión, los conectores, teclados y otros componentes, se adquieren en distintos lugares de todas partes del mundo, siendo siempre los costos la principal consideración. Estos elementos se confían luego a otro subcontratista, a menudo extranjero, para el montaje final. Hasta la carcasa de plástico, que se fabrica con costosas maquinarias de moldeado, proviene de otro subcontratista.

El montaje final de un ordenador se puede realizar de dos maneras: ya sea por medios sumamente automatizados, o bien mediante una gran cantidad de mano de obra de reducido coste. En Estados Unidos, Europa y Japón generalmente se opta por la primera posibilidad, mientras que la segunda es común en Hong Kong, Singapur y Corea del Sur.

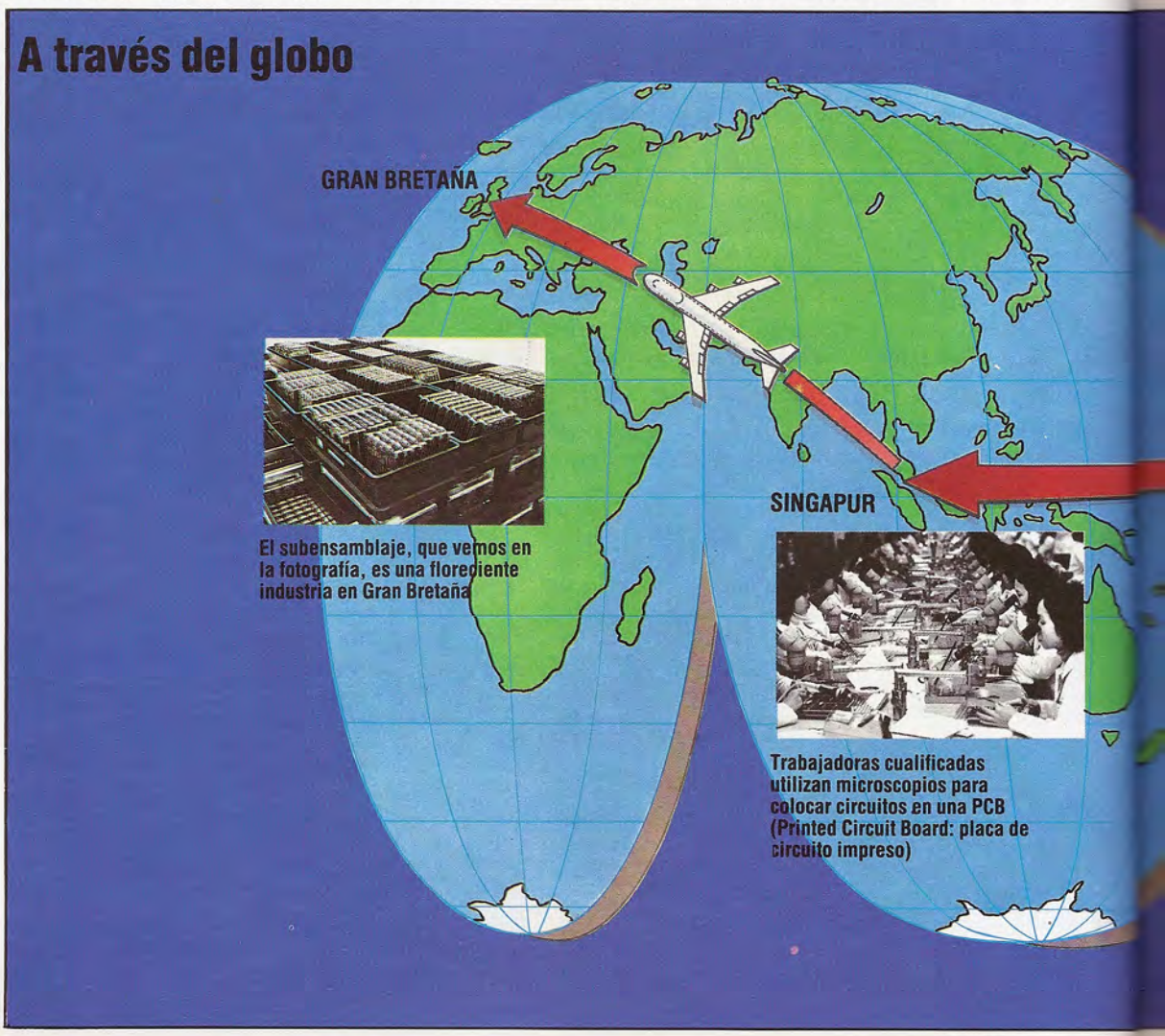
Las líneas de montaje automatizadas utilizan robots para instalar cada componente en las placas de circuito impreso. Los robots se alimentan con largas cintas de componentes, desde condensadores a chips de memoria. Lo que los operarios tienen que hacer es sustituir las cintas cuando éstas se vacían.

Sea cual fuere el sistema que se utilice, las placas son "atiborradas" con los chips adecuados y otros componentes, con sus cables sobresaliendo por debajo de la placa de circuito impreso. Las placas pasan luego a través de una máquina de "soldadura continua" que reviste con soldadura los cables sobresalientes. La soldadura se empuja hacia arriba a través de los agujeros metalizados de la placa y se solidifica para proporcionar una conexión fiable.

Luego las placas terminadas se verifican, se colocan en las carcasas, se empaquetan y se embarcan hacia los almacenes para su distribución, comercialización y venta a los clientes. Esto puede que parezca sencillo, pero cada una de las etapas del proceso posee sus propios problemas específicos.

El primer problema es de sincronización. Todos los componentes, procedentes de sus distintas fuen-

A través del globo



El subensamblaje, que vemos en la fotografía, es una floreciente industria en Gran Bretaña

Trabajadoras cualificadas utilizan microscopios para colocar circuitos en una PCB (Printed Circuit Board: placa de circuito impreso)

tes, deben llegar a un lugar listos para su ensamblaje. La persona responsable de esta operación, el comprador de componentes, es uno de los eslabones vitales de la cadena. Debe tener la habilidad para hacer un astuto trato comercial para adquirir los elementos de forma económica, así como para planificar y sincronizar las fechas de entrega. Todos los fabricantes de ordenadores son conscientes de los desastrosos efectos del retraso en la entrega de un solo componente. Las cadenas de producción sin utilizar cuestan dinero y los atrasos en las entregas significan pérdida de clientes.

El montaje del ordenador, ya sea automático o bien mediante mano de obra, es también un área susceptible de error. Es fácil insertar los componentes con la parte de arriba abajo o la de delante atrás, o, simplemente, omitirlos, arruinando por completo la placa final. La soldadura continua también se puede saltar una patilla de la carcasa de un chip. Del mismo modo, algunos de los componentes de una partida bien podrían no satisfacer sus especificaciones técnicas.

Estos problemas explican la necesidad de los controles, tanto de los componentes como de las placas acabadas. Muchos ensambladores de micros someten a pruebas selectivas los componentes que reciben, y todos ellos efectúan pruebas en las placas con diversos grados de sofisticación. La comprobación de placas es costosa, ya que requiere un poderoso hardware de ordenador. Pero es necesario realizar la inversión: un subcontratista no conseguirá que su contrato dure mucho si cuando entrega las máquinas éstas no funcionan.

El subcontratista que fabrica el Oric ha desarrollado un ingenioso control adicional. Las máquinas terminadas se pesan de forma individual. Si la máquina está por debajo del peso especificado, se deduce que durante el montaje se ha omitido algún componente. Éste es el motivo por el cual todas las cajas Oric poseen una etiqueta azul que indica el peso de la máquina.

La prueba final de un ordenador consiste en enchufar la máquina terminada en una fuente de alimentación eléctrica y un aparato de televisión. Los fabricantes de máquinas de oficina con frecuencia las dejan funcionando durante uno o dos días, "en remojo" o "al fuego". Esta prueba implica simplemente dejar la máquina ejecutando sus rutinas incorporadas o el software que viene con ella para asegurarse de que todo funcione adecuadamente.

Con toda esta cantidad de variables, es fácil apreciar por qué los micros personales pueden aparecer con retraso o no ser fiables. El montaje final depende de que los proveedores de chips y componentes hagan sus entregas a tiempo y según las especificaciones. Las empresas de diseño y marketing dependerán de que el ensamblador final entregue el producto acabado y sin tener los clientes esperando.

Reducción de costos
El objetivo del transporte de componentes para ordenador por todo el mundo es el de ahorrar capital al fabricante y no subir los precios de venta al público. Mediante la utilización de la mano de obra barata de otros países, los fabricantes han sido capaces de reducir significativamente sus costos. Pero recientes avances en la producción automatizada han hecho posible la producción de ordenadores completos en países más avanzados al mismo costo. El Oric Atmos, por ejemplo, se fabrica totalmente en Gran Bretaña, si bien Oric mantiene facilidades de producción en Singapur para sus mercados en el exterior

Componentes viajeros
Este mapamundi ilustra el movimiento de los componentes de microordenador durante el proceso de montaje

Kevin Jones

Movimientos claves

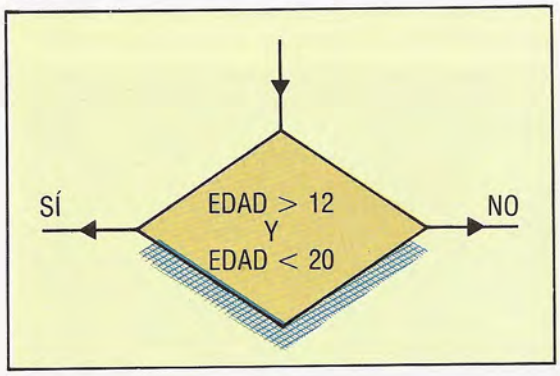
Continuamos con nuestro análisis de cómo utilizar eficazmente los diagramas de flujo en las etapas de planificación de programas

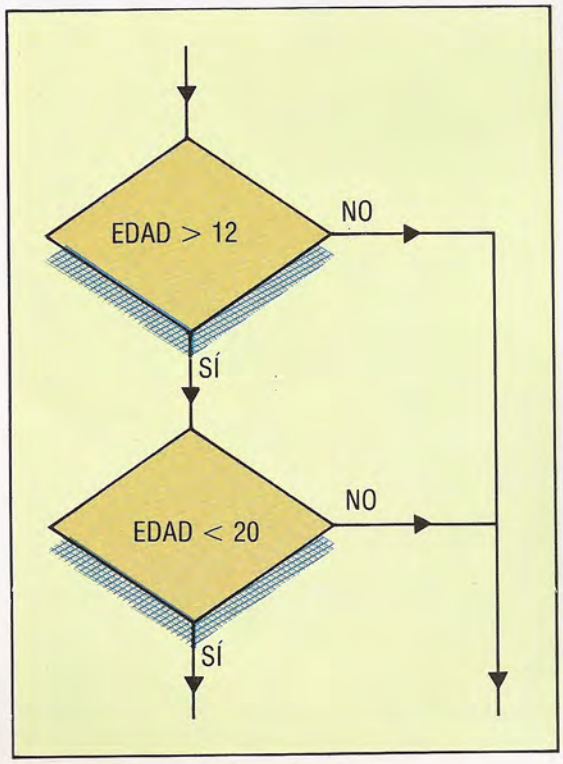
En el presente capítulo de técnicas de programación veremos cómo se pueden dividir las comparaciones compuestas en componentes simples y examinaremos la utilización de *tablas de decisión* en los casos más complejos.

Con frecuencia los programadores desean utilizar en sus programas comparaciones compuestas tales como:

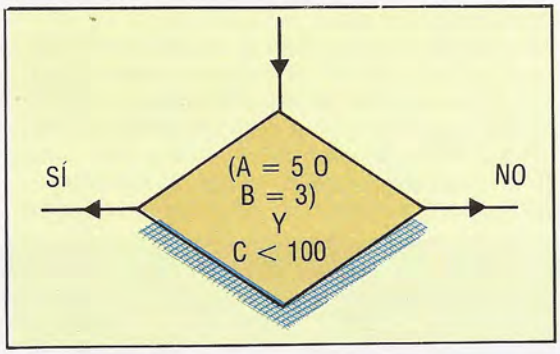
```
IF EDAD > 12 AND EDAD < 20 THEN ESTADO=
"ADOLESCENTE"
```

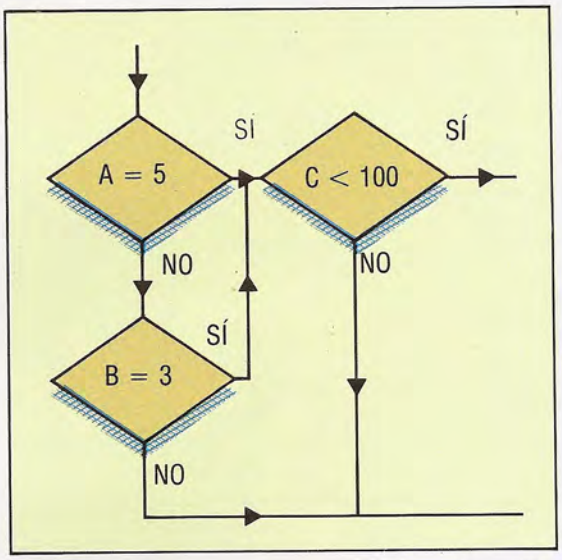
Los algoritmos que contienen instrucciones cómo ésta son más fáciles de entender si se divide la comparación compuesta en las comparaciones que la integran.





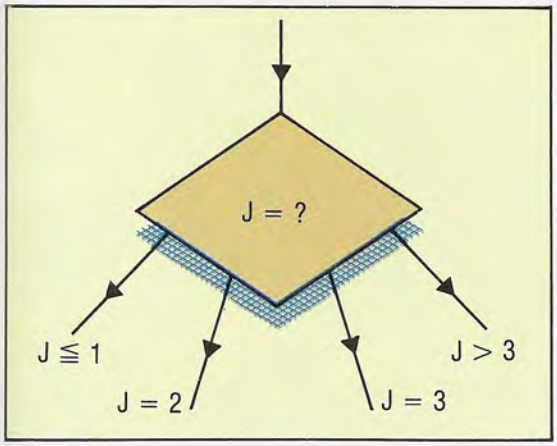
Este ejemplo en BASIC en forma diagramática ilustra cómo la versión compuesta es menos satisfactoria que la simple. El segundo ejemplo (abajo), con tres comparaciones simples, hace que el flujo de lógica a través de las casillas de decisión sea mucho más inteligible que su equivalente compuesto. También muestra una similitud con la lógica booleana, que permite la construcción de circuitos complejos a partir de una combinación de puertas lógicas simples.

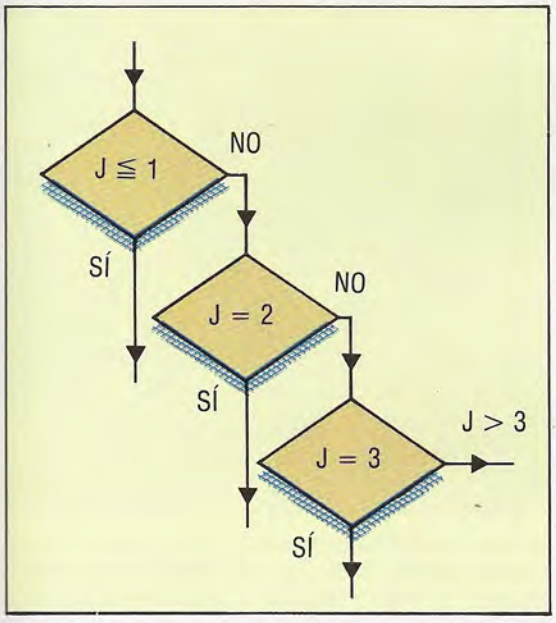




En los ejemplos, todas las comparaciones han sido binarias; no obstante, es bastante común que un algoritmo implique comparaciones con más de dos posibles salidas. Si tratáramos una entrada desde el teclado que representase la selección de una opción de un menú, entonces sería deseable bifurcar a una subrutina, de entre varias diferentes, para emprender la acción requerida. Para hacerlo, la mayoría de los lenguajes disponen de construcciones de bifurcación múltiple tales como CASE...OF en PASCAL y ON...GOTO y ON...GOSUB en BASIC. Las reglas para las comparaciones binarias también rigen para las múltiples: desde una casilla de decisión sólo se puede tomar una ruta y todas las salidas deben

estar bien etiquetadas, siendo todas las posibles rutas mutuamente exclusivas y cubriendo todas las posibilidades. Una comparación múltiple se podría dibujar, como en el ejemplo, con un conjunto de caminos de salida partiendo de la misma casilla de decisión. Sin embargo, es raro, y con mayor frecuencia la comparación será binaria, o sea, con sólo dos posibles salidas.





Todas las comparaciones múltiples se pueden representar como un conjunto de comparaciones binarias.

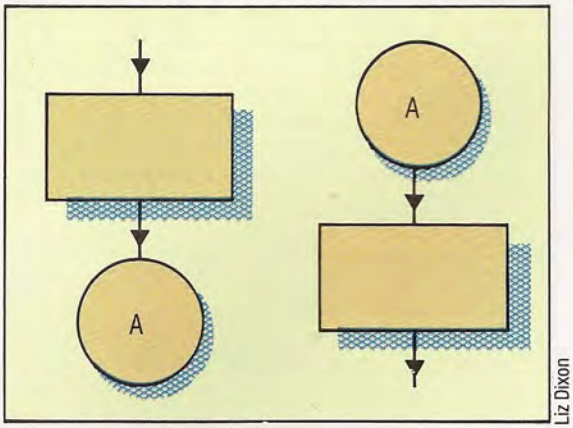
Como una alternativa a los diagramas de flujo, en especial cuando haya muchas comparaciones múltiples, podemos recomendar la utilización de tablas de decisión. Aquí ofrecemos un ejemplo de una tabla de este tipo, que representa un conjunto de reglas para realizar comparaciones. La tabla tiene cuatro secciones principales: texto que describe las condiciones para las reglas, texto que describe las acciones a emprender, una cuadrícula que muestra cómo las condiciones se adaptan a las reglas, y una cuadrícula que indica qué acciones son las apropiadas para cada regla. En la cuadrícula "condiciones/reglas", en los casilleros aparecen valores de variables, mientras que en la cuadrícula de "acciones/reglas" de abajo, un trazo indica qué acción se debe realizar y un valor actúa como un parámetro de entrada para dicha acción. La regla 4, por ejemplo, se lee: "Si se pulsa el botón de disparo y el nivel del juego es 2 y el nivel del jugador es novato, entonces activar escudo de explosiones aleatorias y reducir el nivel de energía en un 2 %".

	REGLAS							
CONDICIONES	1	2	3	4	5	6	7	8
BOTÓN DISPARO PULSADO	✓	✗	✗	✗	✓	✗	✓	✗
NIVEL JUEGO	1	1	2	2	1	1	2	2
NIVEL JUGADOR	NOVATO	NOVATO	NOVATO	NOVATO	EXPERTO	EXPERTO	EXPERTO	EXPERTO
ACCIONES								
ACCIÓN EXTRATERRESTRES							✓	✓
ESCUDO EXPLOSIONES			✓	✓			✓	✓
REDUCIR NIVEL ENERGÍA EN	1 %	1 %	2 %	2 %	2 %	2 %	4 %	4 %

Las tablas de decisión también sirven para combinar comparaciones simples y compuestas y, en formas más sencillas que la que hemos visto aquí, equivalen a las tablas de verdad que se emplean para predecir la salida de puertas lógicas.

Por último, he aquí una importante observación acerca de la utilización de los diagramas de flujo: siempre que sea posible limite la extensión de éstos a una sola página. Puede ser muy incómodo y ocupar mucho tiempo ir pasando una tras otra muchas hojas de papel. Si su algoritmo es demasiado extenso, intente dividirlo en otros más breves. Recuerde que cada algoritmo se puede utilizar como una única instrucción en algún otro algoritmo. De este modo, cada rutina de un programa se podría escribir como una única casilla de proceso en un diagrama de flujo de todo el programa, incluso si la rutina empleara otras rutinas que a su vez utilizaran otras, y así sucesivamente.

Es inevitable que algo vaya mal de vez en cuando y surja la necesidad de que un diagrama de flujo continúe en más de una página. Si esto sucede, corte el diagrama en un lugar apropiado (una comparación, p. ej.) y utilice un círculo con un símbolo identificador dentro para señalar el lugar donde el flujo de control continúa en la página siguiente (representado por otro círculo con el mismo símbolo en su interior, como vemos abajo). Si el control retorna al programa principal, vuelva a emplear círculos para señalar hacia atrás. Otra solución es contemplar la porción que falta como un algoritmo separado, referirse al mismo en una casilla de proceso y representarlo con su propio diagrama de flujo.



Liz Dixon

Haciendo conexiones

El modem Prism VTX5000 es uno de los accesorios de comunicaciones más ingeniosos en el mundo de la informática personal

La base de datos Prestel luchó por encontrar usuarios desde el día en que se puso, por primera vez, "en línea". Los equipos Prestel eran caros y la cantidad de información disponible en la base de datos era muy limitada como para ofrecer grandes ventajas en comparación con la que podía proporcionar un periódico. No obstante, con un modem y software adecuado, la mayoría de los micros personales pueden acceder al Prestel. Y así, con el objetivo de explotar esta posibilidad, Prism creó Micronet, un área separada dentro del Prestel que se dedica a noticias e información variada.

Pronto los modems y el software que posibilitaban el acceso del público al Micronet y al Prestel se pudieron conseguir a través de Prism, y el proyecto alcanzó un enorme éxito. La cantidad de usuarios y la gama de los servicios de información disponibles continúan creciendo.

En la producción de un accesorio para conectar el Sinclair Spectrum al Micronet, Prism se vio frente a un inmenso desafío, dado que la máquina no es adecuada para esta aplicación. No posee ni interface RS232 ni interface en serie, lo que significa que no se pueden conectar los modems normales. Posee una visualización en pantalla de 32 columnas por 24 líneas, y el Prestel exige una visualización de 40 por 24, así como el complicado sistema de gráficos de "teletexto". La empresa produjo una unidad "todo en uno" diseñada específicamente para esta única tarea: el modem Prism VTX5000.

La unidad contiene todas las interfaces necesarias para la conexión con el Spectrum, un modem de 1 200/75 baudios de conexión directa y el software para acceder al Prestel. No sólo proporciona las funciones estándar para conectarse (*log*) en el sistema, sino que también utiliza la pantalla para gráficos del Spectrum para imitar una auténtica visualización de teletexto de 40 por 24.

El VTX5000 se coloca debajo del Spectrum y se conecta a su conector de ampliación. El cable plano entre ambos posee un tercer conector, de modo que se pueden conectar otros periféricos del Spectrum, tales como una impresora o microdrives. Esta unidad se enchufa directamente en la red telefónica en vez de tener que utilizar un acoplador acústico (en el que el tubo del teléfono se encaja en dos tazas plásticas del modem). Esto proporciona una comunicación mucho más fiable.

Para instalar la unidad hay que disponer de unos enchufes telefónicos especiales. En caso de tener

en casa un enchufe normal se debe instalar un conector apropiado. Se desenchufa el teléfono, se enchufa el modem y después se enchufa el teléfono al modem. Este método evita la necesidad de tener en su pared un enchufe telefónico de dos sentidos. Una vez conectado, el teléfono continúa funcionando normalmente.

Cuando se conecta el Spectrum, éste ejecuta automáticamente el software Micronet, que está almacenado en ROM dentro del modem, de manera que no se necesita cargarlo antes de utilizarlo. El paquete Micronet es muy similar al que proporcionan otros micros, de modo que una vez que lo haya utilizado no tendrá ningún problema para emplearlo en una máquina diferente. El software se controla mediante una serie de menús y es fácil de aprender y utilizar.

La primera opción del menú es conectarse (*log-on*), que significa entrar el número de identificación de 10 dígitos que se le otorga a cada usuario del Prestel. El ordenador recordará este número mientras esté encendido, de modo que sólo hay que teclearlo una vez en cada sesión, aun cuando el usuario llame al Prestel en varias ocasiones.

Chris Stevens

El siguiente paso consiste en telefonear a uno de los cuatro ordenadores que mantienen la base de datos Prestel. Cuando éste responde con una nota de tono agudo, se pulsa el interruptor Line (línea) del modem y ya puede colgar el teléfono. Ahora el Spectrum está "hablando" con el Prestel. Lo primero que se debe hacer es entrar una contraseña o palabra clave de cuatro letras. Este sistema impide que cualquier otra persona pueda utilizar la cuenta que usted tiene con el Prestel, y puede cambiarla cuantas veces desee para mantenerla en secreto.

Una vez que se accede a la base de datos, es como utilizar cualquier otro adaptador Prestel. El Prestel requiere teclas de asterisco (*) y numérica (#) para ir pasando página a página. Para las mismas funciones, el Spectrum utiliza Enter y Symbol Shift. Se pueden llamar otras funciones del software Micronet en cualquier momento, siempre que siga conectado al sistema. Éstas permiten copiar páginas de la base de datos y almacenarlas en cinta o imprimirlas por impresora. Por otra parte, se pueden copiar programas completos del sistema en el Spectrum. Las páginas de telesoftware del Micronet incluyen programas gratuitos y programas por los que hay que pagar. Los gratuitos están bien para pasar el rato; pero, como es sabido, no se puede esperar gran cosa de lo que se ofrece gratis.

Tanto el Micronet como el Prestel ofrecen una variada gama de información. Además de noticias y reseñas, hay páginas de consejos técnicos, chistes, juegos, cartas, anuncios de contactos, etc. Incluso se puede enviar correo electrónico a otras personas que utilicen regularmente el servicio. El Micronet también posee un rival: el periódico *Viewfax*, en una parte distinta del Prestel, posee una irreverente sección de micros que es controlada por el misterioso "MicroGnomo".

La "explosión" informativa del Prestel viene muy bien como presagio de lo que será el futuro. Pero algunos de sus problemas originales siguen siendo evidentes, en especial en el Micronet. Muchas páginas de noticias ya son obsoletas cuando se incorporan a la base de datos y a menudo el camino que lleva de una página de información a la siguiente es confuso. Pero si usted se cansa del Prestel, puede tratar de comunicarse directamente con otros usuarios de Spectrum. El VTX5000 también está diseñado para enlazar dos ordenadores Spectrum a través de las líneas telefónicas, de modo que puedan enviarse mensajes y programas directamente el uno al otro. La velocidad de transmisión es de 1 200 baudios. Las primeras versiones del modem se vendían sin el software necesario para hacer esto, pero ahora se incluye con cada unidad una cinta que facilita esta transferencia de datos. Obviamente, este software sólo tiene algún valor para los usuarios de Spectrum que tengan amigos que también posean Spectrum y modems VTX5000.

Existen en uso otros varios estándares de comunicación para los modems, pero el VTX5000 no puede cumplir con todos ellos. La limitación más importante es que el modem carece del estándar necesario para los muchos tablones de comunicaciones y boletines gratuitos que están creando por todo el mundo los entusiastas de la comunicación por ordenador. Éstos operan a una velocidad de datos de 300 baudios, pero el VTX5000 no puede transmitir a esta velocidad tan lenta.

Quien realmente desee explorar toda el área de las comunicaciones por ordenador con su Spectrum quizá encuentre más adecuado comprar una interface RS232 (en especial la ZX Interface 1 de Sinclair) y emplear un modem de propósito general. Sin embargo, esto implicará la escritura de su propio software, conectar cables, etc. El VTX5000 es ideal para quienes sólo deseen utilizar el Prestel.

Conexión directa
El VTX5000 se conecta directamente al Spectrum sin necesidad de ninguna interface adicional. Su teléfono se enchufa en el VTX5000, que se engancha luego directamente en su conector telefónico modular. Observe que este modem sólo funciona en enchufes modulares y no en las cajas para conexión telefónica normales

Enchufes telefónicos modulares BT
El enchufe se introduce en una caja plástica cuadrangular, como se aprecia en la fotografía. Si usted no posee un enchufe modular, es necesario conseguir un enchufe de conversión

Valores variables

Un acumulador, a diferencia de un contador, que trabaja con valores constantes, puede verse aumentado o disminuido en cantidades variables

El diagrama del último programa "contador alfabético" mostrado en esta sección plasmaba una serie de resultados (total de consonantes, vocales, etc.) que habían sido obtenidos tras el incremento de unos contadores con unas cantidades constantes; así, cada vez que aparecía una "u", el contador correspondiente a las "úes" se incrementaba en uno hasta llegar a su valor final.

Pues bien, ésta es la función del contador, la de una variable (una posición de memoria) cuyo valor se incrementa o decrementa en función, siempre, de cantidades fijas: en el citado ejemplo, de 1 en 1, en el caso de los números pares de 2 en 2, o los años bisiestos, que lo hacen con un incremento de 4.

Si este dato fijo no fuera tal, es decir, si se empleasen cantidades diferentes entre sí para formar su valor, ya no se emplearía entonces el término de contador, sino que se daría paso al de *acumulador*.

Con lo cual puede establecerse que la diferencia principal y básica entre uno y otro es la de que mientras que el contador trabaja con valores fijos de incremento o decremento, el acumulador puede verse aumentado o disminuido en cantidades variables sin analogía entre ellas y que, aunque por casualidad, se repitan en alguna ocasión, ello no debe inducir a error, pues puede ser propiciado por pura coincidencia o en orden de alguna regla establecida de antemano que predisponga su esporádica repetición.

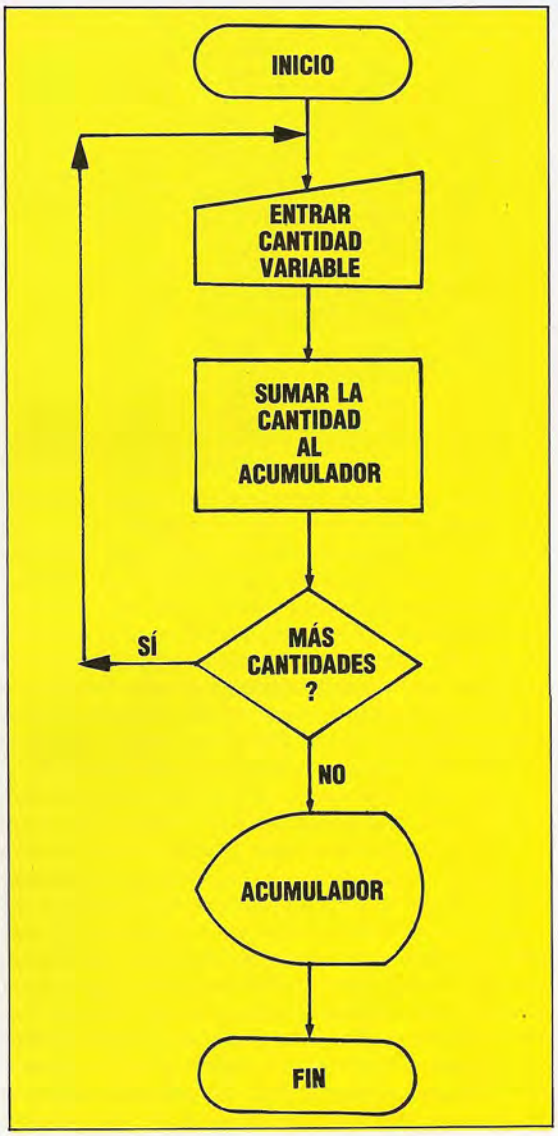
En el programa anteriormente citado, podrían haberse empleado diferentes acumuladores; así, por ejemplo, la función alfanumérica LEN(X$), que determina el número de caracteres de la serie X$, con lo aprendido hasta el momento estamos capacitados para suplirla, mediante el empleo de un contador inicial que irá incrementándose con el paso, uno a uno, de los diferentes caracteres. Pero existe otro procedimiento, propiciado por el uso de un acumulador que obtendría su valor final tras la suma de todos y cada uno de los diferentes contadores empleados.

Lo cual nos daría el total general obtenido con el resultado de sumar todos los totales parciales. En este caso los totales parciales representan datos variables y su contenido cambia según la frecuencia de aparición del carácter que cuenta.

Otro posible acumulador sería el que nos permitiera saber el número total de vocales aparecidas. Para ello, una vez acabado el recuento, se acumularía en una variable la suma conjunta de los cinco contadores.

Pero veamos ahora un pequeño ejemplo en el que se dispone de una serie de cantidades que, entradas por teclado una tras otra, van sumándose en un campo (AC). Al final del programa, es decir, una vez entradas todas las cantidades, se visualizará el contenido de dicho campo acumulador.

```
10 REM ACUMULADOR
20 INPUT"CANTIDAD";K
30 AC = AC + K
40 INPUT"HAY MAS CANTIDADES?(S/N)";F$
50 IF F$ = "S" THEN GOTO 20
60 PRINT "TOTAL ACUMULADOR:";AC
70 END
```



Con pantalla propia

Sus excelentes gráficos y su fiable hardware convierten al Amstrad en uno de los ordenadores más sofisticados del mercado

El ordenador Amstrad se vende en dos versiones. Ambos modelos son el mismo ordenador básico, pero se suministran con pantallas diferentes: una monocromática, la otra en color RGB. La fuente de alimentación eléctrica para el ordenador está alojada dentro de la carcasa de la pantalla y unos cables conectores llevan la energía y las señales de la pantalla hasta el ordenador. Como la grabadora de cassette también está incorporada en la carcasa del monitor, hay un único cable de energía eléctrica para abastecer a todo el sistema, de lo que resulta una mínima cantidad de cables de conexión.

La pantalla monocromática es de color verde y proporciona una visualización muy clara y bien definida, apta para tareas comerciales y otro tipo de tareas con textos. Se observa una ligera ondulación de la imagen, probablemente debida a la proximidad de la fuente de alimentación eléctrica en la carcasa de la pantalla. La pantalla en color sólo es de resolución media. Esto significa que, si bien puede visualizar en toda su plenitud los gráficos multicolores del Amstrad, no es capaz de visualizar texto de 80 columnas de una forma legible.

Para armonizar con la pantalla monocromática de estilo oficina, el Amstrad posee un teclado del tipo máquina de escribir completo, con un teclado numérico separado. Todas las teclas se pueden redefinir para crear caracteres diferentes a los que señalan las teclas. Además, el teclado numérico puede actuar como un conjunto de teclas de función programables. En cada una de éstas se puede programar una serie de hasta 32 caracteres. Por consiguiente se pueden utilizar teclas individuales para cargar o listar un programa o para borrar la pantalla.

Además de las características que el Amstrad tiene incorporadas, también hay prevista una amplia gama de periféricos. Una impresora Centronics se enchufa directamente en el micro a través de un conector marginal bastante tosco. También se pueden añadir unidades de disco, pero éstas todavía no han salido a la venta, si bien está anunciada para dentro de poco tiempo una unidad de disco que le proporcionará al ordenador memoria extra, el lenguaje LOGO y el sistema operativo de gestión CP/M. El Amstrad posee un único conector para palanca de mando que acepta las de tipo Atari. Algunos juegos exigen dos palancas de mando, de modo que Amstrad suministra un par que encajan simultáneamente en el conector.

El Amstrad tiene un altavoz incorporado (completo, con control de volumen), pero la señal de sonido también se puede enviar a un amplificador externo para obtener sonido estéreo a partir de las tres "voces" separadas del ordenador. Una voz se envía al altavoz derecho, una al izquierdo, y la tercera se mezcla a partes iguales entre ambos. Con

Chris Stevens

Opciones de pantalla
La versión más económica del Amstrad viene con una pantalla monocromática, pero una versión con pantalla en color tiene también un precio asequible. Las máquinas no se pueden adquirir sin las pantallas. Los dos programas que se están ejecutando en la fotografía son *Wordhang*, que es una versión de *Hangman* (Verdugo), y *Admiral Graf Spee*, un juego naval ambientado durante la segunda guerra mundial

una programación adecuada, una nave espacial extraterrestre no sólo se puede mover a través de la pantalla sino que da la impresión de que su sonido se desplaza por la habitación.

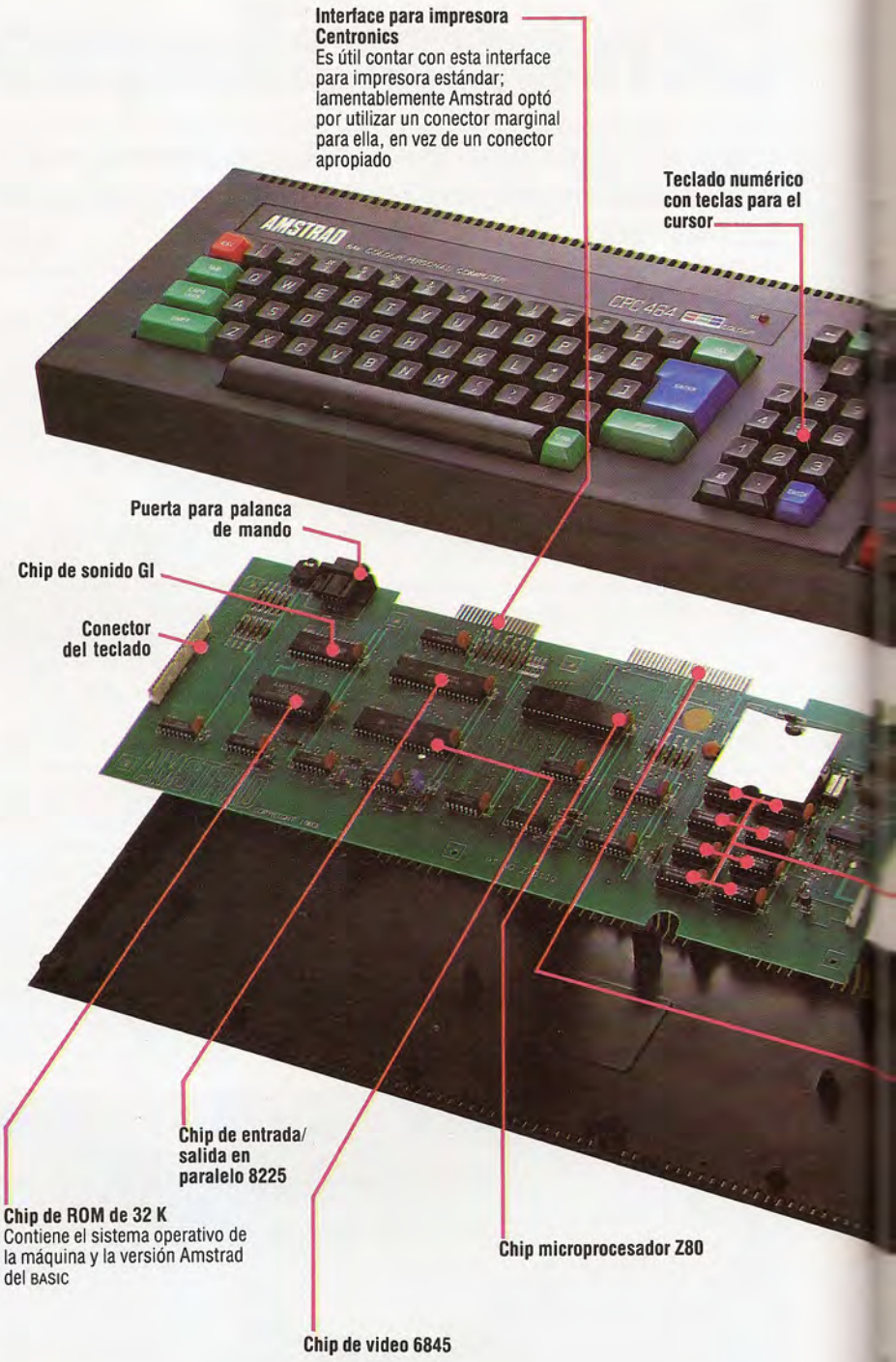
Controlar el sonido desde el BASIC es muy sencillo, a pesar de la sofisticación del sistema. No sólo se puede hacer que las notas empiecen y se interrumpan a una altura y volumen determinados, sino que también se puede controlar su envoltura o "forma". La envoltura de volumen se puede ajustar para hacer que una nota imite el sonido de, por ejemplo, un piano o una campana. La envoltura de tono también se puede controlar independientemente para la creación de efectos sonoros, como sirenas y silbidos, mientras cada voz se puede mezclar con "ruido", haciendo muy sencilla la simulación de explosiones y disparos de armas de fuego.

La mejor característica del Amstrad es la de sus excelentes gráficos. Hay tres modalidades de visualización disponibles, cada una de las cuales suministra una cantidad diferente de caracteres y de colores en pantalla al mismo tiempo. Cada una de estas modalidades utiliza los mismos 16 Kbytes de memoria y hay una compensación entre la cantidad de colores y la resolución y el formato del texto. La modalidad de resolución más alta permite sólo dos colores en la pantalla al mismo tiempo, uno de primer plano y otro de fondo. Hay disponible texto en 80 columnas y la máquina admite una resolución para gráficos realmente impresionante, de 640 × 200. En el otro extremo, la modalidad de 20 columnas admite 16 colores en la pantalla al mismo tiempo. La tercera modalidad admite 4 colores y 40 columnas.

Aunque la cantidad de colores que puede haber en pantalla al mismo tiempo es limitada, éstos se pueden seleccionar entre los 27 colores de la paleta. Se pueden visualizar marrones y tonos pastel además del rojo, azul, etc., habituales. El color del borde alrededor de la superficie de visualización se puede seleccionar de la misma paleta de colores. Se puede hacer que cualquiera de los 27 colores se enciendan de forma intermitente entre dos tonalidades diferentes a velocidades variables. Además de proporcionar un excelente marco de ampliación para imágenes detalladas, los gráficos del Amstrad también proporcionan una buena base para técnicas de animación. La única omisión la constituyen los sprites. No obstante, la manipulación de la pantalla mediante el BASIC es suficientemente rápida como para lograr que esta deficiencia no resulte tan importante.

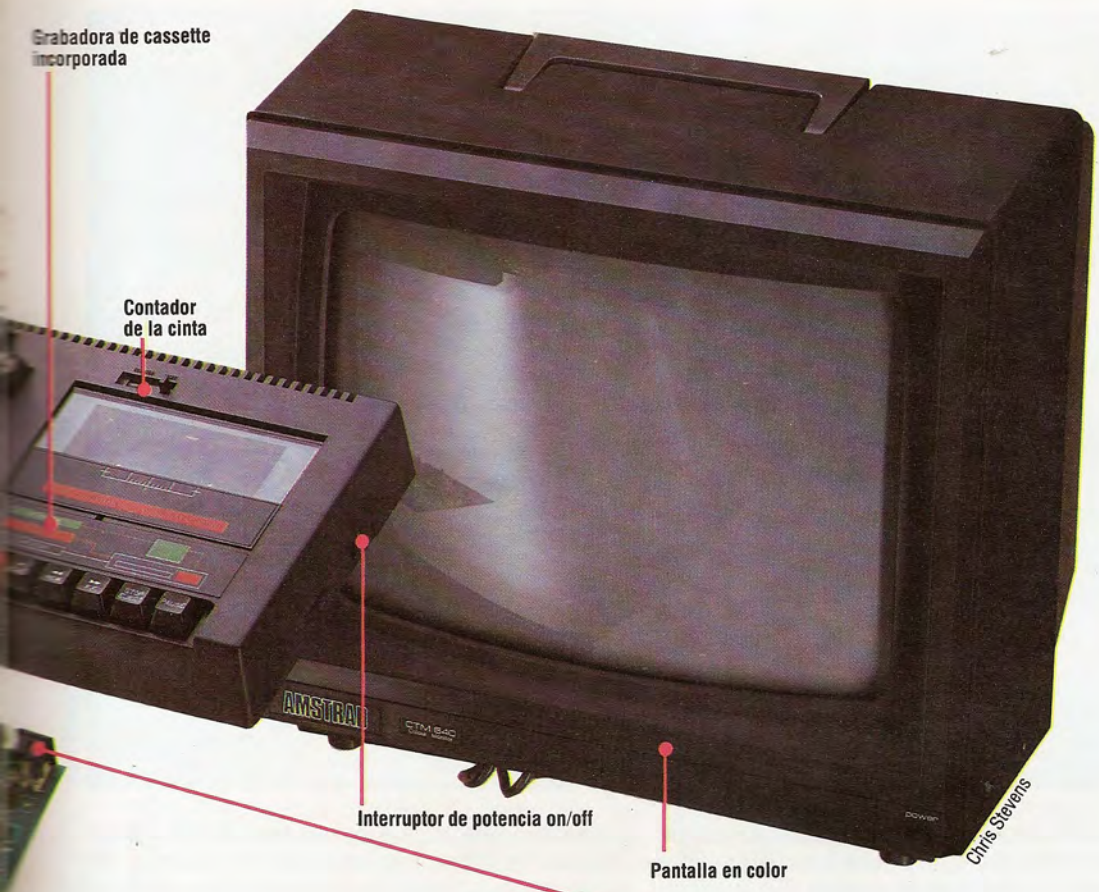
A pesar de que la visualización en pantalla utiliza hasta 16 Kbytes de memoria, no roba espacio de memoria del programa del usuario. La RAM de la pantalla y la ROM de BASIC ocupan la misma área en el mapa de memoria del procesador. Un chip hecho a medida dentro del Amstrad conmuta entre ambas cuando ello es necesario, de modo que para los datos y los programas en BASIC quedan 42 Kbytes completos de memoria.

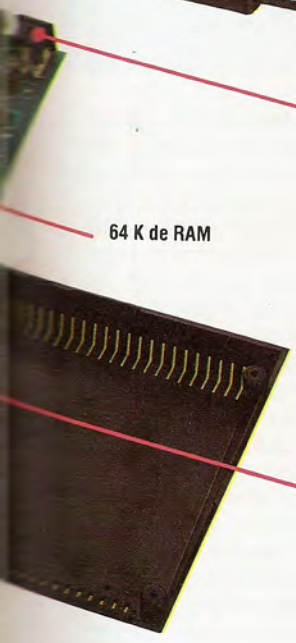
El BASIC Amstrad es una de las versiones del lenguaje más sofisticadas que existen. El excelente hardware para gráficos representa un gran apoyo. Hay varias características útiles para simplificar el trazado de imágenes en la pantalla. El origen de los gráficos se puede redefinir desde el rincón inferior izquierdo de la pantalla a cualquier punto dentro o



Palancas originales

El Amstrad tiene un conector para palanca de mando que permite utilizar una del tipo Atari. No obstante, la firma fabricante también produce su propia palanca, y ésta tiene un conector que permite enchufar una segunda. Las señales provenientes de ésta se transmiten a través de la primera





AMSTRAD CPC 464

DIMENSIONES

Teclado-cassette:
565 x 170 x 70 mm
Pantalla en color:
380 x 350 x 350 mm

CPU

Z80

MEMORIA

64 K de RAM, de los cuales hay 42 K disponibles para programas en BASIC, 32 K de ROM

PANTALLA

Tres modalidades con mezcla total de texto y gráficos:
640 x 200 (2 colores)
320 x 200 (4 colores)
160 x 200 (16 colores)
Una paleta con 27 colores para escoger

INTERFACES

Palanca de mando (2), puerta para impresora Centronics, bus de ampliación (unidades de disco), salida de sonido en estéreo, salida para pantalla

LENGUAJES DISPONIBLES

BASIC (incluido), PASCAL (en cassette)

TECLADO

Tipo máquina de escribir, 74 teclas

DOCUMENTACION

La guía para el principiante que se incluye con la máquina es fácil de entender. También se encuentra a la venta un manual de referencia de BASIC y un manual de referencias técnico

VENTAJAS

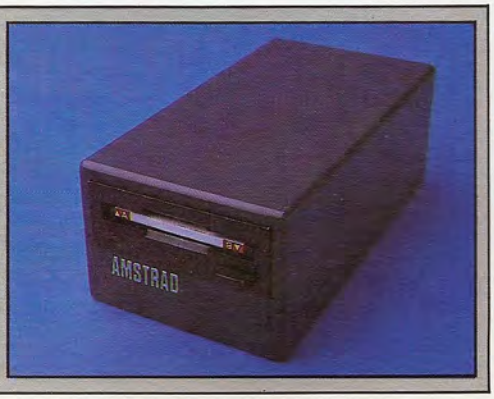
La pantalla y la grabadora de cassette incorporada hacen del Amstrad un sistema completo. Posee una gran memoria, gráficos versátiles y excelentes, y un sofisticado sonido estéreo

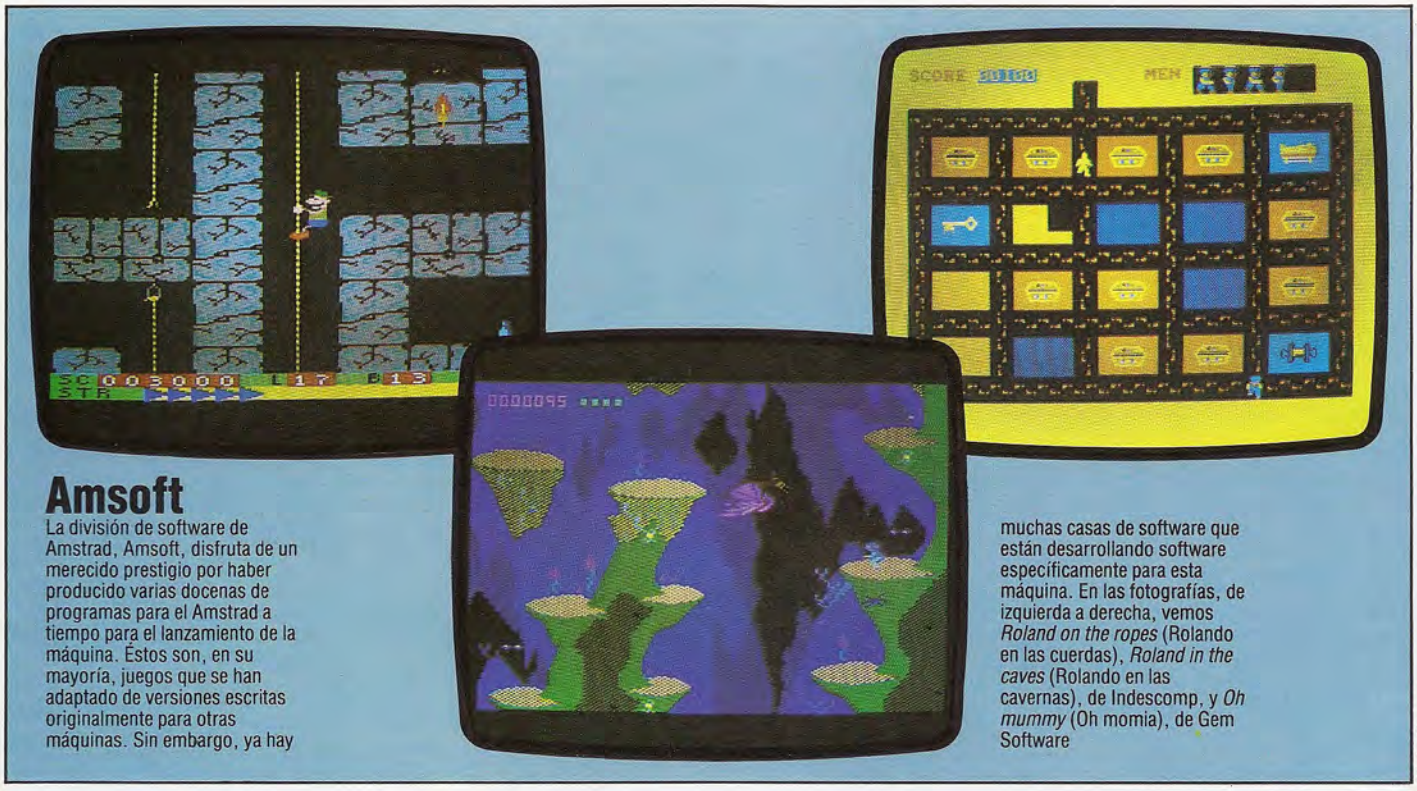
DESVENTAJAS

La grabadora de cassette no es de construcción muy sólida. Las instrucciones para gráficos del BASIC no están a la altura (todavía) del potencial que ofrece el hardware

Futura mejora

Amstrad tiene planeado producir un impacto en el extremo inferior del mercado de ordenadores de oficina lanzando una unidad de disco. Ésta incluirá el sistema operativo CP/M estándar, que necesitan la mayoría de los programas de gestión





Amsoft

La división de software de Amstrad, Amsoft, disfruta de un merecido prestigio por haber producido varias docenas de programas para el Amstrad a tiempo para el lanzamiento de la máquina. Éstos son, en su mayoría, juegos que se han adaptado de versiones escritas originalmente para otras máquinas. Sin embargo, ya hay muchas casas de software que están desarrollando software específicamente para esta máquina. En las fotografías, de izquierda a derecha, vemos *Roland on the ropes* (Rolando en las cuerdas), *Roland in the caves* (Rolando en las cavernas), de Indescomp, y *Oh mummy* (Oh momia), de Gem Software

Chris Stevens

fuera de la misma. Se puede establecer una ventana de gráficos para limitar las operaciones de gráficos a una pequeña porción de la pantalla. En la pantalla se pueden definir hasta ocho ventanas de texto al mismo tiempo, y el texto se puede dirigir a cualquiera de las ocho en cada momento. Se podrían enviar las preguntas a una ventana, por ejemplo, y las respuestas tecleadas a otra.

Los gráficos en BASIC también carecen de un procedimiento para rellenar con color una superficie de la pantalla. No hay instrucciones para dibujar zonas sólidas en la pantalla, ni tampoco para rellenar un esquema ya presente; sólo se pueden dibujar puntos y líneas. La única forma de producir un bloque de color es dibujar muchas líneas las unas muy cerca de las otras, método bastante ineficaz. No obstante, existe la posibilidad de que esta omisión se rectifique en un futuro cercano. Así como la ROM de BASIC se enciende y se apaga en el mapa de memoria para dejarle sitio a la memoria de la pantalla, se podría agregar otra ROM para que ocupara el mismo espacio. Las características de las que carece el BASIC Amstrad bien podrían aparecer en una ROM adicional y entonces las funciones nuevas se fundirían con BASIC antiguo. Lenguajes nuevos completos, como el PASCAL, FORTH y LOGO, se pueden añadir de la misma manera. Estas ROM "laterales" no ocuparían más espacio de memoria que el que ocupa el BASIC actual, de modo que también habría 42 Kbytes de RAM libres para su utilización. De hecho, se puede insertar memoria extra en el mapa de memoria de forma similar, de modo que se ampliarían los 64 Kbytes estándares de RAM.

El aspecto más original del BASIC Amstrad es su tratamiento de las "interrupciones". Muchos micros permiten que los programadores en lenguaje máquina utilicen el sistema de interrupciones incorporado en el sistema operativo de una máquina para activar sus propias rutinas en código máquina. El BASIC Amstrad lleva esta idea un paso más allá permitiendo utilizar las interrupciones desde el BASIC.

La instrucción del BASIC AFTER (después) cederá el control desde un programa a una subrutina especificada después de transcurrido un período de tiempo determinado. EVERY hará lo mismo de forma repetida. Esta avanzada característica hace que el escribir cualquier tipo de programa que dependa del tiempo, desde un programa para recogida de datos de laboratorio hasta un juego recreativo, sea más fácil y más eficaz.

El Amstrad es único entre los ordenadores personales por el hecho de que se suministra con una pantalla en vez de la visualización por el televisor que utilizan sus competidores. Las pantallas ofrecen mejor calidad de imagen, de modo que ésta es una ventaja muy importante. Sin embargo, es probable que los usuarios que hayan adquirido la versión monocromática necesiten contar ocasionalmente con una visualización en color. Tal como están las cosas, no existe ninguna forma de poder utilizar un televisor en color con el Amstrad, si bien hay a la venta, a modo de extra, un adaptador. Con el Amstrad se puede emplear una pantalla en color separada sin la ayuda del adaptador; pero, dado que el micro toma su energía a través de la pantalla monocromática, habría que tener las dos pantallas encendidas simultáneamente.

Con gráficos espléndidos y hardware fiable, un BASIC avanzado y potencial para ampliación, el Amstrad CPC 464 es uno de los ordenadores personales más sofisticados que existen actualmente en el mercado. También ofrece una excelente relación entre calidad y precio. Como inconveniente cabría destacar que la grabadora de cassette no es muy sólida y que las instrucciones para gráficos no están a la altura del hardware.

Selva y laberinto

"Sabre Wulf" tiene como protagonista a un aventurero, en busca de tesoros y en lucha constante contra sus atacantes

A causa de la caída en las ventas de los juegos de acción, las empresas de software han desviado su atención hacia una nueva forma de juego que combina elementos de la estrategia de los juegos recreativos con el sentido de la aventura. *Sabre Wulf*, el nuevo producto de Ultimate, Play The Game, responde a esta fórmula.

Un juego de aventuras, en el que el jugador debe conducir al héroe (a menudo un personaje extraído de la literatura de ciencia-ficción o de fantasía) a través de diversos lugares, resolviendo enigmas por el camino, con frecuencia tiene tramos laboriosos. La acción puede cesar durante prolongados períodos, mientras el jugador intenta hallar la respuesta de un enigma aparentemente insoluble. El juego recreativo, por otra parte, fomenta poco el pensamiento prolongado, exigiendo, en cambio, buenos reflejos y un dedo que dispare con rapidez.

Sabre Wulf intenta combinar lo mejor que poseen estos dos tipos de juegos. Básicamente se trata de un juego de laberinto que se desarrolla en un ambiente selvático, y sus orígenes guardan una estrecha relación con el clásico juego recreativo *Pac-Man*. El héroe, que recuerda a Indiana Jones (el aventurero protagonista de la película *En busca del Arca perdida*), es guiado a través de un laberinto sumamente complicado, evitando atacantes y apoderándose de tesoros para anotar puntos. El objetivo del ejercicio es recuperar los cuatro trozos desperdigados de un amuleto mágico roto.

El escenario de la jungla es magnífico, reproducido vívidamente en algunos de los gráficos más detallados que pueden verse en el Spectrum. Animales, plantas, montañas, cuevas y tesoros están todos ellos ilustrados de forma maravillosa y, en algunas ocasiones, el efecto global recuerda a un cuadro de Rousseau. El laberinto es extraordinariamente complejo y cubrir en una sola partida sólo una quinta parte del mismo constituye una proeza.

La mayoría de los repelentes seres atacantes se despachan simplemente con un rápido golpe de la espada del héroe, si bien él ha de estar justo enfrente de ellos en ese momento. En este sentido, *Sabre Wulf* imita a *Pac-Man*. Pero algunos contrincantes exigen armas especiales y éstas las debe ir descubriendo el jugador a medida que avanza por el laberinto. Otros objetos proporcionarán una vida adicional, lo cual es muy importante en este juego porque sólo los jugadores muy expertos tienen posibilidades de mantenerse ilesos durante mucho tiempo. Algunos objetos, como las orquídeas en flor, pueden ser tanto una ayuda como un obstáculo. Según el color que tengan, pueden hacerlo inmune al peligro, convertirlo temporalmente en un vegetal, doblar la velocidad a la cual se mueve el jugador o (lo que más lo confundirá) invertir el efecto de los mandos. Los efectos de las orquídeas desaparecen pronto.

Sabre Wulf está, en general, muy bien diseñado, aunque adolece de algunos de los defectos comunes que afectan también al software de otros juegos. El sonido, inicialmente muy atractivo y, por cierto, complejo, enseguida se vuelve molesto, y Ultimate no ha incluido la posibilidad de apagarlo. Si bien se supone que es un juego para uno o dos jugadores, en realidad es un juego para un solo jugador que posee dos marcadores. Está el usual Ultimate Hall of Fame (galería de famosos de Ultimate), con espacio para las iniciales de los seis marcadores más altos. Aquí Ultimate ha optado por el estilo recreativo para entrar los nombres: las iniciales se entran utilizando los controles de movimiento, ya sea en la palanca de mando o en el teclado.

Los fabricantes de software están empezando por fin a darse cuenta de que existe una amplia gama de teclados disponibles para el Spectrum y Ultimate ha previsto muchos de ellos. La utilización del teclado es menos satisfactoria porque *Sabre Wulf* emplea las teclas Q, W, E, R y T para movimiento y acción de la espada. Esto es difícil de comprender, puesto que todas estas teclas están en la misma fila y, por lo tanto, son extremadamente difíciles de utilizar.

Pero, dejando de lado errores soslayables acerca de la disposición del teclado, *Sabre Wulf* es un juego excelente que ofrece un notable equilibrio entre la acción de los juegos recreativos y la estrategia de los de aventuras.

Sabre Wulf: Para el Spectrum de 48 K
Editado por: Ashby Computers and Graphics
Autores: Ultimate, Play The Game
Palancas de mando: Kempston, Interface 2 y cursor
Formato: Cassette

La página del título
Esta pantalla muestra los gráficos de la página del título de *Sabre Wulf*. El mismo diseño está reproducido en la cubierta del paquete.
Sabre Wulf es un juego de laberinto con una ingeniosa variedad de personajes detestables. Los márgenes del laberinto están cubiertos por excelentes gráficos que imitan un paisaje selvático

Liz Heaney

Curso de colisión

Ahora examinaremos el control del movimiento y la detección de colisiones entre los caracteres del jugador y de las minas

El BASIC BBC posee no menos de cuatro instrucciones que responden a una sola pulsación de tecla. La elección de la instrucción dependerá, obviamente, del efecto deseado. INKEY$ e INKEY se utilizan normalmente cuando se desea esperar durante un período determinado una posible pulsación de tecla antes de seguir adelante con el resto del programa; GET$ y GET, por otra parte, siempre interrumpen la ejecución del programa hasta que se pulsa una tecla. Estas dos últimas instrucciones tienden a utilizarse cuando se requiere una respuesta a una pregunta, como "Otra partida S/N?". Al utilizar GET$ o GET, el programa aguardará hasta que se produzca una respuesta. En este caso, las únicas respuestas aceptables son "S" y "N". Podemos utilizar un bucle para repetir (REPEAT) la instrucción GET hasta (UNTIL) que la respuesta sea "S" o "N":

```
1000 PRINT "OTRA PARTIDA S/N?
1010 REPEAT
1020    A$ = GET$
1030 UNTIL A$ = "S" OR A$ = "N"
1040 Etc.
```

Si se utilizan GET$ o INKEY$, entonces la tecla pulsada se interpreta como un carácter alfanumérico (tipo string), como en el ejemplo anterior. Si utilizamos GET o INKEY, entonces se devuelve un carácter de tipo numérico; este valor es el código ASCII de la tecla pulsada. Estas opciones permiten que el programador pueda preguntar por la pulsación de aquellas teclas que no tienen un carácter asociado, como las teclas Return o las de movimiento del cursor. La sentencia *FX4,1 se podría usar para que las teclas del cursor devuelvan códigos ASCII. De hacerlo así, las teclas tendrían los valores:

Cursor izquierda	136
Cursor derecha	137
Cursor abajo	138
Cursor arriba	139

Supongamos que deseamos aceptar para nuestro programa pulsaciones de teclas de cursor izquierda y cursor derecha. El siguiente segmento de programa utiliza INKEY para esperar una entrada durante un cuarto de segundo:

```
1000 *FX4,1:REM ENCENDER MODALIDAD ASCII
     DEL CURSOR
1010 REPEAT
1020    A = INKEY(25)
1030 UNTIL A = 136 OR A = 137
1040 *FX4,0:REM RESTAURAR CURSOR A
     MODALIDAD EDICION
```

El parámetro 25 de la línea 1020 le dice al ordenador que espere 25 centésimas de segundo antes de seguir adelante con el programa.

Estas sentencias no leen del teclado mismo sino que lo hacen en un área de memoria dentro del ordenador llamada *buffer del teclado*. Éste es un espacio de almacenamiento temporal para caracteres que se introducen desde el teclado, y es algo así como la cola para entrar en un cine. Los nuevos caracteres tecleados se ponen al final de la cola y el procesador va tomando caracteres del comienzo de la misma. De esta forma, si se teclean caracteres más rápidamente de lo que puede manipularlos el procesador, éstos no se pierden sino que tan sólo aguardan su turno en el buffer del teclado. Puesto que INKEY, GET, INKEY$ y GET$ normalmente leen sobre la parte delantera de la cola del buffer del teclado, no hay forma de saber durante cuánto tiempo ha estado un determinado carácter esperando en el buffer para ser procesado. En los juegos que se controlan desde el teclado, ello puede significar una respuesta lenta, ya que el programa puede estar procesando anteriores pulsaciones de teclas mientras el jugador está realizando otras nuevas. Por ejemplo, si se llena el buffer del teclado con códigos de cursor derecha, será necesario procesarlos todos antes de que el programa pueda responder a una instrucción cursor izquierda. Esto puede dejar al jugador pulsando frenéticamente el botón cursor izquierda ¡y preguntándose por qué razón el objeto que controla se mueve hacia la derecha!

Existen dos soluciones para este problema. La primera consiste en limpiar siempre el buffer del teclado justo antes de investigarlo. Esto se puede efectuar utilizando la sentencia *FX15,1. Alternativamente podemos utilizar otra variación de INKEY. Como hemos descrito más arriba, INKEY() espera durante un período de tiempo, determinado por el número entre paréntesis, a que se pulse una tecla antes de continuar con el programa. Podemos, sin embargo, hacer que INKEY lea directamente del teclado en vez del buffer del teclado especificando entre los paréntesis que siguen a la instrucción un número *negativo*. Con esta finalidad, todas las teclas tienen asignado un número negativo. En nuestro programa emplearemos las teclas del cursor para controlar el movimiento. Los valores de las teclas a utilizar con INKEY son:

Cursor izquierda	−26
Cursor derecha	−122
Cursor abajo	−42
Cursor arriba	−58

El siguiente procedimiento utiliza INKEY para leer del teclado directamente para cada una de las cuatro teclas del cursor sucesivamente. Si se está pulsando una de las teclas, entonces se accede a otro procedimiento ("mover"), pasando dos parámetros. Estos parámetros contienen información relativa a la dirección en la cual se ha de desplazar el carácter que representa el detector de minas.

```
3000 DEF PROCleer—teclado
3010 REM ** ARRIBA? **
3020 IF INKEY(-58) = -1 THEN PROCmover(0,-1)
3030 REM ** ABAJO? **
3040 IF INKEY(-42) = -1 THEN PROCmover(0,1)
3050 REM ** DERECHA? **
3060 IF INKEY(-122) = -1 THEN PROCmover(1,0)
3070 REM ** IZQUIERDA? **
3080 IF INKEY(-26) = -1 THEN PROCmover(-1,0)
3090 ENDPROC
```

El procedimiento "mover"

Este procedimiento es el cuerpo del programa. Con él se desplazan los caracteres del detector y del ayudante y se verifican las colisiones con minas. Vamos a analizar en primer lugar la sección que controla el movimiento de los caracteres.

Desde el procedimiento "leer—teclado" se le pasan a "mover" dos parámetros. Éstos se aceptan en las variables delta-x y delta-y para su utilización en "mover", y corresponden a los cambios a efectuar en las coordenadas x e y del detector de minas. Por ejemplo, si se pulsara la tecla cursor arriba, entonces se le pasarían a "mover" los valores 0 y −1. Las instrucciones xdet = xdet + delta−x e ydet = ydet + delta−y harían que se actualizaran las coordenadas del detector. En el caso de cursor arriba, se le suma 0 a xdet y −1 a ydet, lo que en realidad es restarle uno a su valor. Esto parecería implicar el movimiento de una unidad hacia *abajo* en la pantalla, pero debemos recordar que el origen de las posiciones de los caracteres está en el rincón superior izquierdo y que los valores de y se incrementan hacia abajo en la pantalla. Por consiguiente, decrementar ydet en uno produce el movimiento hacia arriba de una celda de carácter. Tal vez usted quiera verificar que los valores pasados para las otras tres direcciones corresponden, realmente, a las alteraciones correctas de xdet e ydet. Sería factible utilizar este sistema para incluir movimientos en diagonal. Pasándole a "mover" los valores (1,−1) el detector se movería oblicuamente una celda hacia arriba y una hacia la derecha. Sin embargo, se habrían de introducir otras teclas para permitir el control en diagonal desde el teclado. Veamos el listado para el procedimiento "mover":

```
3220 DEF PROCmover(delta−x,delta−y)
3230 REM ** BORRAR POSICIONES VIEJAS **
3240 COLOUR 1
3250 PRINTTAB(xdet,ydet);" "
3260 PRINTTAB(xhom,yhom);" "
3270 REM ** MOVER DETECTOR **
3280 xdet = xdet + delta−x
3290 ydet = ydet + delta−y
3300 REM ** VERIFICAR LIMITES **
3310 IF xdet > 17 THEN xdet = 17
3320 IF ydet > 25 THEN ydet = 25
3330 IF xdet < 2 THEN xdet = 2
3340 IF ydet < 1 THEN ydet = 1
3350 REM ** CALCULAR COORDS. HOMBRE **
3360 xhom = 19−xdet
3370 yhom = 26−ydet
3380 PROCconvertir(xhom,yhom)
3390 IF POINT(xgraf,ygraf) = 2 THEN PROCexplotar(xgraf,ygraf)
3400 PROCconvertir(xdet,ydet)
3410 IF PONT(xgraf,ygraf) = 2 THEN PROCdescubierta−mina
3420 PROCsituar−sujetos
3430 ENDPROC
```

Antes de alterar las coordenadas x e y del detector debemos borrar las posiciones antiguas del detector y el ayudante. Las líneas 3250 y 3260 utilizan los valores antiguos de xdet, ydet, xhom e yhom para imprimir (PRINT) espacios sobre los caracteres viejos. Dado que los nuevos caracteres se imprimirán (PRINT) en rojo (color lógico 1), se utiliza la instrucción de color en la línea 3240 para establecer el color de fondo en curso en 1. Las líneas 3280 y 3290 actualizan las coordenadas del detector tal como hemos descrito anteriormente. Antes de imprimir (PRINT) realmente el detector en su nueva posición, se deben realizar comprobaciones para asegurarnos de que no estemos incrementando o disminuyendo coordenadas por fuera de la zona que hemos definido como nuestro campo de minas. Los límites superior e inferior de xdet e ydet se comprueban entre las líneas 3310 y 3340. Aquí se ha decidido que si el detector llega a un margen, entonces permanecerá allí hasta que se lo mueva en la dirección contraria. Por ejemplo, la línea 3310 verifica si se ha alcanzado el margen derecho del campo de minas, señalado con coordenada x de 17. Si se realizara un intento por incrementar xdet por encima de 17, entonces esta línea simplemente restauraría el valor a 17. Hubiera sido igualmente posible crear un efecto de "rodillo" según el cual el detector, al llegar al margen derecho, apareciese seguidamente por el margen izquierdo de la pantalla. Para producir este efecto en el margen derecho del campo de minas, alteramos la línea 3310:

```
3310 IF xdet > 17 THEN xdet = 2
```

Tal vez se desee alterar esta condición y las otras tres condiciones de márgenes para proporcionar un efecto de rodillo en todos los bordes del campo.

Una de las reglas de nuestro juego es que mientras el jugador desplaza el detector de minas por el campo destruyendo minas, el ayudante del jugador imita todos los movimientos. Para hacer esto, debemos actualizar automáticamente las coordenadas del ayudante, que se relacionan con las coordenadas del detector mediante un sencillo par de fórmulas, que podemos ver en las líneas 3360 y 3370. Para demostrar cómo las mismas producen movimientos de espejo, observemos la relación existente entre las coordenadas x (xhom = 19−xdet).

Inicialmente, xdet es 2 y xhom es 17. Si el detector se mueve un lugar a la derecha, xdet aumenta a 3. Utilizando la fórmula anterior, xhom se calculará como 19−3 = 16. Esto significa que el ayudante se mueve un lugar hacia la izquierda. Si se vuelve a mover xdet hacia la derecha, entonces xdet se convierte en 4 y xhom será 15, y así sucesivamente. Las coordenadas y funcionan de modo similar.

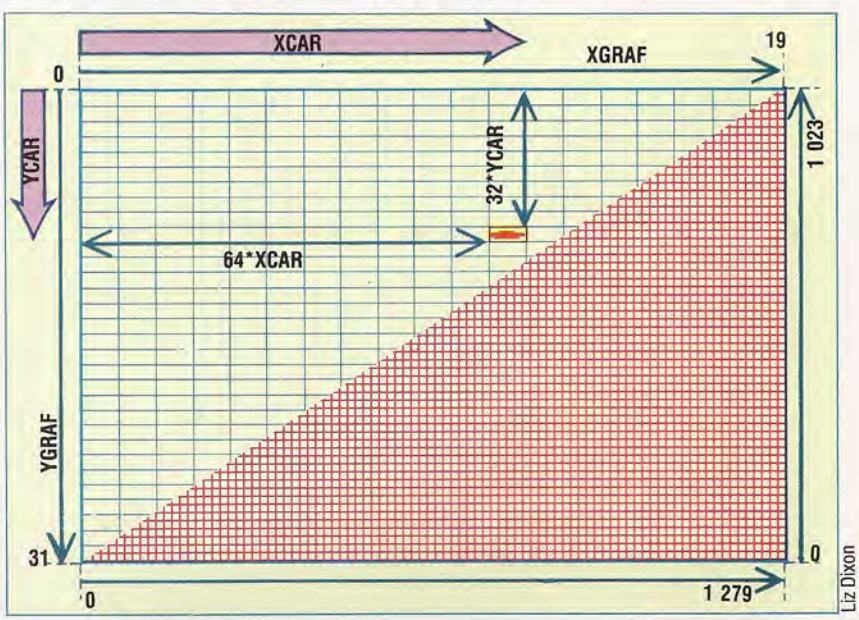
Antes de imprimir (PRINT) el detector y el ayudante nos queda por realizar aún otra tarea. Debemos comprobar si el ayudante o el detector se han movido a una celda de caracteres que ya está ocu-

Mapa

El juego mezcla gráficos en alta resolución con la visualización de textos BBC/Electron. Ello tiene sus ventajas pero significa que se deben mezclar dos sistemas distintos de coordenadas, uno para gráficos y otro para textos. El BBC/Electron posee varios formatos de texto diferentes y, por tanto, cada uno de ellos posee su propio sistema de coordenadas. El juego utiliza la modalidad 5, que proporciona 20 caracteres a lo ancho de la pantalla y 32 a lo alto. Esto se puede apreciar en las partes superior e izquierda del diagrama.

Las máquinas poseen, asimismo, tres resoluciones para gráficos distintas, aunque por suerte todas ellas utilizan el mismo sistema de coordenadas. Éste trata a todas las modalidades como si tuvieran una resolución de 1 280 por 1 024. Esto se puede apreciar en las partes inferior y derecha del diagrama.

El programa utiliza el sistema de coordenadas de alta resolución para leer puntos de la visualización de textos en baja resolución. Ello significa convertir la posición de un carácter a una coordenada de alta resolución. Para hacer esto se debe multiplicar por 64 la coordenada horizontal (XCAR, en el programa) y por 32 la coordenada vertical (YCAR). También se debe superar otro problema. Las coordenadas de la pantalla para texto empiezan desde 0 en la parte superior de la pantalla y se cuentan hacia abajo, mientras que las coordenadas para gráficos empiezan desde 0 en la parte inferior y se cuentan hacia arriba. Esto se resuelve fácilmente restándole 32*YCAR a 1 023



Liz Dixon

pada por una mina. El BASIC BBC nos permite investigar cualquier punto de la pantalla para ver si está encendido con un color determinado. POINT-(X,Y) devolverá el color del pixel de la posición (X,Y). Podemos utilizar esta instrucción para ver si el color de la celda a la que nos estamos moviendo es verde (en cuyo caso contendría una mina). Sólo hay un inconveniente: POINT(X,Y) utiliza el sistema de coordenadas en alta resolución para especificar las coordenadas del punto a examinar. Si deseamos utilizar esta instrucción, debemos antes convertir nuestras coordenadas de celdas de carácter en coordenadas de gráficos.

En el capítulo anterior calculamos que en la modalidad 5 cada celda de caracteres es de 64 unidades de gráficos de anchura por 32 unidades de gráficos de altura (véase p. 884). Multiplicando xcar por 64 obtendremos la coordenada xgraf del margen de la celda en cuestión. Sumándole a xgraf otros 32 obtendremos la coordenada x del centro de la celda. El cálculo de ygraf es algo más complicado porque los dos sistemas trabajan en sentidos opuestos. En la parte superior de la pantalla ygraf es 1023. Yendo hacia abajo, 32*ycar nos llevaría hasta la parte superior de la celda especificada; avanzando otras 16 unidades hacia abajo llegaríamos a la coordenada y del centro de la celda. Así, el siguiente procedimiento se puede diseñar para convertir coordenadas de caracteres en coordenadas de gráficos:

```
3720 DEF PROCconvertir(xcar,ycar)
3730 xgraf = 64*xcar + 32
3740 ygraf = 1023-(32*ycar + 16)
3750 ENDPROC
```

Podremos apreciar el verdadero valor que tiene la posibilidad de pasar parámetros entre procedimientos si volvemos a analizar el procedimiento "mover". El procedimiento "convertir" se utiliza dos veces: en primer lugar, para convertir las coordenadas del ayudante en coordenadas de gráficos, y estos valores se utilizan luego en la línea 3390 para comprobar el color verde. (Recuerde que a pesar de que se han actualizado las coordenadas del ayudante, el carácter todavía no ha sido impreso —PRINT— en su nueva posición.) Si el color es verde, entonces el programa salta a otro procedimiento para visualizar una explosión. El procedimiento "convertir" se utiliza por segunda vez en la línea 3400, pero en esta ocasión se calculan las coordenadas del carácter del detector. La línea 3410 verifica luego si la celda está ocupada por una mina. De ser así, se llama al procedimiento "descubierta-mina". Por último se imprimen (PRINT) el detector y el ayudante en sus nuevas posiciones llamando a "situar-sujetos".

En el próximo capítulo analizaremos el procedimiento "explotar"; por ahora colocaremos un procedimiento ficticio. Entre las siguientes líneas:

```
3550 DEFPROCexplotar(x—explosión,y—explosión)
3560 PRINT "BANG"
3570 END
3580 ENDPROC
```

Por último, examinemos el procedimiento "descubierta-mina", al que se llama cuando el detector se desplaza a una celda que está ocupada por una mina. Sería una buena idea introducir un efecto sonoro que indicara que se ha descubierto una mina. Más adelante en el proyecto analizaremos con más profundidad el sonido, así que de momento todo cuanto necesitamos saber es que la sentencia SOUND de la línea 3790 produce un "ping" de tono agudo. Sin embargo, la principal función de esta rutina es la de incrementar el tanteo del jugador. El programa utiliza dos variables para el tanteo, la primera de las cuales es una variable numérica que se incrementa a razón de 150. Para que el tanteo se imprima siempre como un número de cinco dígitos debemos agregarle ceros por la izquierda. Con el fin de hacer esto, debemos primero convertir el valor numérico del tanteo a una variable string o en serie y utilizar después técnicas de manipulación de series, como las descritas previamente en el proyecto (véase p. 884) para agregar ceros por delante. El procedimiento completo es:

```
3770 DEF PROCdescubierta—mina
3780 REM ** EFECTO SONORO **
3790 SOUND 2,-15,170,3
3800 REM ** INCREMENTAR TANTEO **
3810 COLOUR 2
3820 tanteo = tanteo + 150
3830 tanteo$ = STR$(tanteo)
3840 tanteo$ = LEFT$(cero$,5-LEN(tanteo$)) + tanteo$
3850 PRINTTAB(11,28);tanteo$
3860 ENDPROC
```

En la última parte del proyecto escribimos un corto programa de llamada para los procedimientos que hemos escrito hasta ahora. Los procedimientos reseñados aquí se pueden añadir a su programa con los números de línea consignados. La única modificación necesaria es llamar al procedimiento "leer-teclado" desde dentro del bucle principal del programa de llamada. Por consiguiente, deberá agregar temporalmente a su programa esta línea:

```
55 PROCleer-teclado
```

Ian McKinnell/Liz Heaney

Disparar al objetivo

En este punto del proyecto del Campo de Minas, el programa llenará la pantalla con minas colocadas al azar; creará la imagen suya y una imagen espejo; definirá las tablas del marcador y proporcionará movimiento. Debido a que el programa aún no está completo, si usted cae sobre una mina sólo verá la palabra "BANG" impresa en la pantalla. En el próximo capítulo de este proyecto diseñaremos una rutina que cree una verdadera explosión con los efectos sonoros correspondientes. También es posible que se encuentre con mensajes de error en ciertos puntos de la ejecución del juego. Estos mensajes se producen debido a la estructura incompleta del juego e irán desapareciendo a medida que se vayan añadiendo las subrutinas finales. No obstante, ya hemos llegado a un punto en el cual el programa ha asumido todas las características de un juego de acción rápida

Acción refleja

El programa que presentamos le permite medir sus tiempos de reacción participando en un juego sencillo

A todos nos gusta creer que tenemos muy buenos reflejos; de hecho, tendemos a suponer que nuestras reacciones ante los acontecimientos son casi instantáneas. Sin embargo, es probable que nuestra respuesta más veloz ante un estímulo sea más o menos de un tercio de segundo, que parece suficientemente rápida hasta que uno considera que un coche a gran velocidad habrá recorrido unos 10 metros durante ese lapso. Raramente los tiempos de reacción son constantes; el exceso de alcohol, la fatiga o la enfermedad pueden tener un efecto negativo sobre los reflejos. Nuestro programa está diseñado para probar los reflejos de cualquier persona y, para obtener una mayor exactitud, se calcula un promedio de los tiempos obtenidos en cinco pruebas.

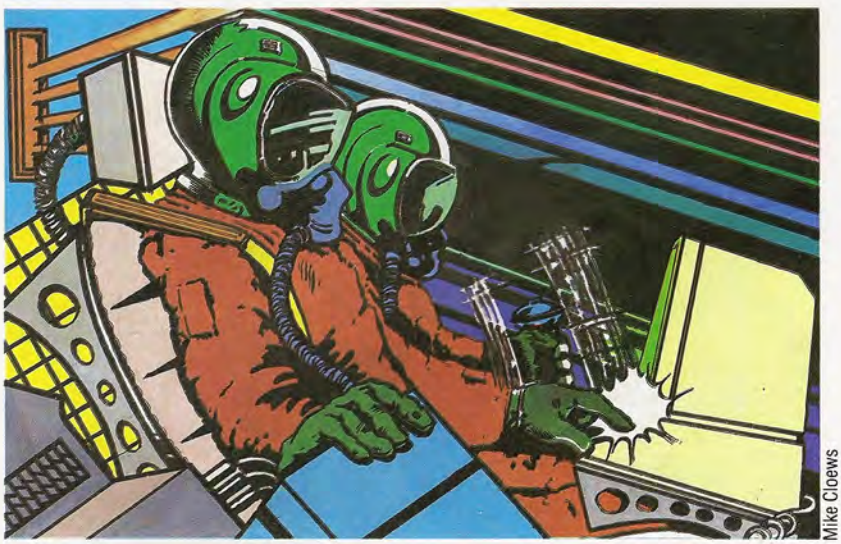
El jugador asume el papel del piloto de una nave espacial que debe entregar con toda urgencia un cargamento de medicinas a una colonia del cinturón de asteroides. La velocidad es vital para que los medicamentos lleguen a tiempo, pero la nave se debe detener tan pronto como exista la inminencia de una colisión. Simplemente pulsando la barra espaciadora la nave se detendrá, y el jugador podrá entonces seguir abriéndose camino a través de los asteroides. El juego visualiza el tiempo que transcurre entre la aparición en pantalla de los asteroides y la pulsación de la barra espaciadora. Después de que se haya detenido la nave en cinco ocasiones, se calcula el tiempo de reacción promedio. Para asegurarse de que el jugador no haga trampas, el programa verifica si la barra espaciadora se está manteniendo pulsada de forma continua; de ser así, los motores se paran y se oye un sonido de advertencia. Al mismo tiempo que aparecen los asteroides en la pantalla, el radar de la nave emite una nota aguda. Como ejercicio, intente cambiar el programa de modo que se reciba un aviso audible o bien visual, pero no ambos a la vez.

Una modificación final podría ser seleccionar entre dos o más opciones presentadas en la pantalla. Ello haría que el programa fuera más una prueba de decisiones y menos una indicación de simples tiempos de reacción.

Tiempo de comprobación

```
 10 REM *PARA EL SPECTRUM
 20 INK 7: PAPER 0: BORDER 0
 30 FOR T = 0 TO 31: READ A
 40 POKE USR "A" + T,A: NEXT T
 60 DATA 123,231,255,255,245,233,123,100
 70 DATA 34,187,208,49,86,96,178,205
 80 DATA 67,234,45,123,198,255,29,245
 90 DATA 78,100,245,60,50,160,245,189
150 LET M = 0: LET A = 0: LET V = 0: DIM R(5): CLS
220 FOR G = 1 TO 5
250 FOR P = 1 TO RND*150 + 50
260 PLOT INT (RND*250),INT (RND*170)
270 IF INKEY$ <> " " THEN GO TO 280
275 BEEP .1,-3: GO TO 265
280 NEXT P
290 FOR A = 144 TO 147
320 PRINT AT RND*20,RND*30;CHR$(A)
330 NEXT A
350 BEEP .05,15: LET C = 0
360 LET C = C + 1: LET A$ = INKEY$
410 IF A$ <> " " THEN GO TO 355
420 LET R(G) = C/66 + .05
430 PRINT AT 19,0;" HA TARDADO";INT (R(G)*100)/100;
440 PRINT " SEGUNDOS EN PARAR"
445 PRINT "LOS MOTORES"
450 LET M = M + R(G)
453 FOR J = 1 TO 300: NEXT J
455 NEXT G
460 LET A = M/5: CLS
480 PRINT "AL CABO DE CINCO PRUEBAS"
490 PRINT "SU TIEMPO DE REACCION MEDIO FUE";INT (A*100)/100;
495 PRINT " SEGUNDOS"
500 FOR G = 1 TO 5: LET V = V + ABS (R(G)-A)
510 NEXT G
520 PRINT: PRINT "SU TIEMPO DE REACCION VARIO EN UN"
530 PRINT INT (V*20/A);" POR CIENTO"
540 PRINT: PRINT "QUIERE OTRA PASADA?(S/N)"
550 LET R$ = INKEY$: IF R$ = "s" THEN GO TO 150
560 IF R$ <> "n" THEN GO TO 550
570 INK 0: PAPER 7: BORDER 7: CLS
```

```
 10 REM *PARA EL BBC MICRO Y ELECTRON
 20 MODE 1: CLS: VDU 23;8202;0;0;0: DIM REAC (5)
 30 REM *PREPARAR CARACTERES (ASTEROIDES)
 40 FOR J = 0 TO 3
 50 VDU 23,237 + J,78,100,245,60,50,160,245,189
 60 VDU 23,223 + J,89,67,34,156,123,200,256,29
 70 NEXT J
 80 SUMA = 0: PROMEDIO = 0: VARIAC = 0: CLS
 90 FOR VEZ = 1 TO 5
100 REM *TRAZAR ESTRELLAS
110 FOR DEMORA = 1 TO 100 + RND(300)
120 PLOT 69,RND(1200),RND(1012)
130 IF INKEY(-99) = 0THEN 150 ELSE
140 SOUND 1,-15,5,1: GOTO 130
150 NEXT DEMORA
160 REM *TRAZAR ASTEROIDES EN PANTALLA
170 FOR ASTEROIDE = 233 TO 240
180 X = RND(28): Y = RND(28)
190 PRINT TAB(X,Y);CHR$(ASTEROIDE)
200 NEXT ASTEROIDE
210 SOUND 2,-15,150,3
220 REM * COMPROBAR BARRA ESPACIADORA
230 TIEMPO = 0
240 IF INKEY(-99) = 0 THEN 240
250 REAC(VEZ) = TIEMPO/100
260 PRINT TAB(3,26);"HA TARDADO ";
270 PRINT;REAC(VEZ);" SEGUNDOS "
280 PRINT TAB(3,28);"EN PARAR LOS MOTORES"
290 FOR I = 1 TO 500: NEXT I
300 SUMA = SUMA + REAC(VEZ)
310 NEXT VEZ
320 FOR I = 1 TO 3000: NEXT I
330 REM *CALCULAR TIEMPO RESPUESTA
340 PROMEDIO = SUMA/5
350 FOR VEZ = 1 TO 5
360 VARIAC = VARIAC + ABS(REAC(VEZ)-PROMEDIO)
370 NEXT VEZ
380 CLS: PRINT: PRINT
390 PRINT "AL CABO DE CINCO PRUEBAS SU TIEMPO"
400 PRINT "DE REACCION MEDIO FUE"; PROMEDIO;" SEGUNDOS"
410 PRINT: PRINT "SU TIEMPO DE REACCION VARIÓ EN UN"
420 PRINT INT(VARIAC*20/PROMEDIO);" POR CIENTO"
430 PRINT: PRINT "QUIERE OTRA PASADA?S/N"
440 RS = INKEY$(0): IF RS = "S" THEN 80
450 IF RS <> "N" THEN 440 ELSE MODE 4
```



Mike Cloews

Círculo luminoso

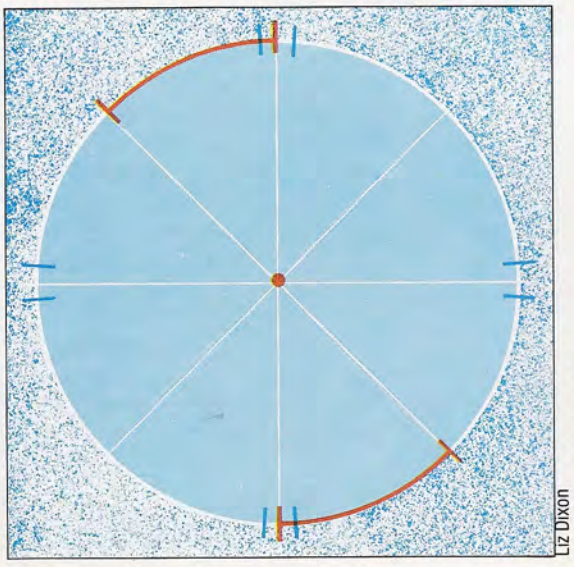
Vamos a comprobar cómo funciona una rutina en lenguaje máquina que sirve para crear círculos y pueda acoplarse con el lenguaje BASIC

Dibujar una circunferencia es una cosa muy sencilla, evidentemente mucho más que dar con la fórmula matemática que permita trazar sus puntos. El modo más sencillo de dibujarla se consigue sirviéndonos de las funciones COS (coseno) y SIN (seno), como en este ejemplo:

```
DEF PROCCIRCLE (XORG,YORG,R)
  MOVER XORG + R,YORG
  FOR ALFA = 0 TO 2*PI STEP PI/32
    X = R*COS(ALFA)
    Y = R*SIN(ALFA)
    TRAZAR X + XORG,Y + YORG
  NEXT ALFA
FIN PROC
```

Pero este PROCedimiento lleva su tiempo de ejecución, porque emplea dos funciones, COS y SIN, cuyos cálculos son lentos. La circunferencia se va dibujando con relativa lentitud. Sin embargo, es posible acelerar su trazado mediante el siguiente algoritmo, derivado de la geometría elemental y del cálculo diferencial:

```
  3 MODO1
  5 PROCCIRCLE (500,600,200)
  7 END
 10 DEF PROCCIRCLE (XORG,YORG,R)
 20 Y = R
 30 FOR X = 1 TO Y*.707
 40 Y = Y - X/Y
 50 PROCPOINTS(X,Y)
 60 NEXT
 70 ENDPROC
 80 DEF PROCPOINTS (X,Y)
 90 PLOT 69,XORG+X,YORG+Y
100 PLOT 69,XORG-X,YORG+Y
110 PLOT 69,XORG-X,YORG-Y
120 PLOT 69,XORG+X,YORG-Y
130 PLOT 69,XORG+Y,YORG+X
140 PLOT 69,XORG-Y,YORG+X
150 PLOT 69,XORG-Y,YORG-X
160 PLOT 69,XORG+Y,YORG-X
170 ENDPROC
```



Liz Dixon

Imágenes especulares
El programa en código máquina utiliza una ecuación seleccionada especialmente por su velocidad para trazar el círculo. No obstante, no hace falta calcular la posición de todos los puntos de éste. En realidad, la mitad superior de la circunferencia es una imagen especular (como reflejada en un espejo) de la mitad inferior, por tanto es suficiente calcular sólo media circunferencia. De igual modo, la mitad derecha es una imagen especular de la mitad izquierda.
En la práctica, el programa únicamente calcula una octava parte del círculo. Cada punto de este octavo es copiado siete veces en otras partes del círculo para llegar a trazarlo en su totalidad

Esta rutina dibuja la circunferencia en ocho zonas a la vez, lo que evidentemente acelera el proceso. Este algoritmo es además mejor que el inicial pues no emplea las funciones de seno y coseno que debían calcularse antes del trazado de cada punto. Sin embargo, todavía resulta algo lenta la ejecución del proceso si se piensa que se debe calcular una división.

Este otro procedimiento alternativo que representamos a continuación no necesita ningún cálculo aritmético complicado:

```
 10 MODO 4
 20 PNUM = 69
 30 PROCCIRCLE(500,600,200)
 40 END
 50 :
 60 DEF PROCCIRCLE(X,Y,R)
 70 VDU29,X;Y;:REM ESTABLECER ORIGEN
    DEL GRAFICO
 80 X = 0:Y = R:D = 3-2*R:REM
    VARIABLES
 90 REPEAT
100 PROCCPLOT
110 IFD < 0:D = D+4*X+6:ELSED =
    D+4*(X-Y)+10:Y = Y-4
120 X = X+4
130 UNTIL X > Y:ENDPROC
140 :
150 DEF PROCCPLOT
160 PLOT PNUM,X,Y
170 PLOT PNUM,Y,X
180 PLOT PNUM,Y,-X
190 PLOT PNUM,-X,Y
200 PLOT PNUM,-X,-Y
210 PLOT PNUM,-Y,-X
220 PLOT PNUM,-Y,X
230 PLOT PNUM,X,-Y
240 ENDPROC
```

El método se conoce como algoritmo de Breshen. Resulta mucho más rápido al no tener que realizar suma, resta o multiplicación alguna por dos o por cuatro (que pueden realizarse mediante desplazamientos de bits). Es éste el algoritmo que usaremos en nuestro programa en lenguaje máquina para dibujar una circunferencia.

Para permitir que el programa funcione en todas

Círculos BBC

```
 30PNUM=69
 40S%=2
 50OSWRCH=&FFEE
 60DIM CODE% 600
 70DIM D% 12
 80X=D%+2
 90Y=D%+4
100D=D%+6
110N1=D%+8
120N2=D%+10
130PROCCOMPILE
140MODE4
150PROC_OLYMPIC
160END
170#
180DEF PROCCOMPILE
190FOR I% =0 TO S% STEP S%
200K%=P%
210P%=CODE%
220[OPT I%
230.CIRCLE
240JSR INIT
250:
260.LOOP
270JSR COMPXY:BMI DOIT:JSR CPLOT:RTS
280.DOIT
290JSR CPLOT
300LDA D+1:BPL D_IS_POS
310:
320.D_IS_NEG
330JSR DNEG
340JSR ADD_4_TO_X
350JMP LOOP
360:
370.D_IS_POS
380JSR DPOS
390JSR ADD_4_TO_X
400JMP LOOP
410:
420.INIT
430LDY #8
440.L7
450LDA &601,Y:STA &80,Y
460DEY:BPL L7
470INY
480LDA (&80),Y:STA X
490LDA (&83),Y:STA Y
500LDA (&86),Y:STA D
510INY
520LDA (&80),Y:STA X+1
530LDA (&83),Y:STA Y+1
540LDA (&86),Y:STA D+1
550LDA #29:STA D%+1
560LDA #0:STA D%
570JSR PSTR
580LDA #25:STA D%
590LDA #PNUM:STA D%+1
600JSR SETD
610RTS
620:
630.COMPXY
640LDA X:STA N1:LDA X+1:STA N1+1
650LDA Y:STA N2:LDA Y+1:STA N2+1
660JSR SUB
670LDA N1+1
680RTS
690:
700.CPLOT
710LDX #4
720.L2
730JSR P2
740DEX:BNE L2
750RTS
760:
770.DNEG
780LDA X:STA N1:LDA X+1:STA N1+1
790JSR TIMES4
800LDA #6:STA N2:LDA #0:STA N2+1
810JSR ADD
820LDA D:STA N2:LDA D+1:STA N2+1
830JSR ADD
840LDA N1:STA D:LDA N1+1:STA D+1
850RTS
860:
870.DPOS
880LDA X:STA N1:LDA X+1:STA N1+1
890LDA Y:STA N2:LDA Y+1:STA N2+1
900JSR SUB
910JSR TIMES4
920LDA #10:STA N2:LDA #0:STA N2+1
930JSR ADD
940LDA D:STA N2:LDA D+1:STA N2+1
950JSR ADD
960LDA N1:STA D:LDA N1+1:STA D+1
970JSR SUB_4_FROM_Y
980RTS
990:
1000.ADD_4_TO_X
1010LDA #4:STA N1:LDA #0:STA N1+1
1020LDA X:STA N2:LDA X+1:STA N2+1
1030JSR ADD
1040LDA N1:STA X:LDA N1+1:STA X+1
1050RTS
1060:
1070.SUB_4_FROM_Y
1080LDA#4:STA N2:LDA #0:STA N2+1
1090LDA Y:STA N1:LDA Y+1:STA N1+1
1100JSR SUB
1110LDA N1:STA Y:LDA N1+1:STA Y+1
1120RTS
1130.SETD
1140LDA #0:STA X:STA X+1
1150LDA D:STA Y:LDA D+1:STA Y+1
1160ASL D:ROL D+1
1170LDA #3:STA N1:LDA #0:STA N1+1
1180LDA D:STA N2:LDA D+1:STA N2+1
1190JSR SUB
1200LDA N1:STA D:LDA N1+1:STA D+1
1210RTS
1220:
1230.P2
1240JSR PSTR
1250JSR SWAPXY
1260JSR PSTR
1270JSR NEGY
1280RTS
1290:
1300.TIMES4
1310ASL N1:ROL N1+1
1320ASL N1:ROL N1+1
1330RTS
1340:
1350.ADD
1360CLC
1370LDA N1:ADC N2:STA N1
1380LDA N1+1:ADC N2+1:STA N1+1
1390RTS
1400:
1410.SUB:\ (N1=N1-N2)
1420SEC
1430LDA N1:SBC N2:STA N1
1440LDA N1+1:SBC N2+1:STA N1+1
1450RTS
1460:
1470.PSTR
1480LDY #250
1490.L1
1500LDA D%-250,Y
1510JSR OSWRCH
1520INY
1530BNE L1
1540RTS
1550:
1560.SWAPXY
1570LDA X:PHA:LDA X+1:PHA
1580LDA Y:STA X:LDA Y+1:STA X+1
1590PLA:STA Y+1:PLA:STA Y
1600RTS
1610:
1620.NEGY
1630LDA #0:STA N1:STA N1+1
1640LDA Y:STA N2
1650LDA Y+1:STA N2+1
1660JSR SUB
1670LDA N1:STA Y
1680LDA N1+1:STA Y+1
1690RTS
1700]
1710NEXT
1720ENDPROC
1730#
1740DEF PROCCIRCLE(P1%,P2%,P3%)
1750CALL CIRCLE,P1%,P2%,P3%
1760ENDPROC
1770#
1780DEF PROC_OLYMPIC
1790PROCCIRCLE(300,600,150)
1800PROCCIRCLE(650,600,150)
1810PROCCIRCLE(1000,600,150)
1820PROCCIRCLE(475,450,150)
1830PROCCIRCLE(825,450,150)
1840VDU29,0;0;
1850MOVE100,250
1860DRAW100,800
1870DRAW1200,800
1880DRAW1200,250
1890DRAW100,250
1900ENDPROC
```

Esta rutina facilita el dibujo de circunferencias en el BBC y en el Electron. Para utilizar la rutina en código máquina basta con colocar los valores adecuados en tres variables INTEGER (p. ej., X%,Y% y R%) y llamarla con la instrucción: CALL CIRCLE,X%,Y%,R%. Dibujará una circunferencia de radio R% y centro en X%,Y%. Obsérvese que el origen del gráfico es desplazado hacia el centro de la circunferencia por la rutina. Pero lo podemos restablecer con VDU29,0;0;. Por desgracia, la instrucción CALL no posibilita utilizar las expresiones como parámetros, por lo que X%,Y% y R% sólo pueden ser variables. Para obviar este inconveniente nos hemos servido del procedimiento PROCCIRCLE (como se ve en el listado)

las modalidades de gráficos, hemos empleado la instrucción VDU 25 para trazar todos los puntos de la circunferencia, a conciencia de que esto retrasa considerablemente la ejecución. El programa está estructurado en subrutinas autosuficientes de modo que ayude a su comprensión. Y para evitar confusiones se han utilizado sólo técnicas directas. El precio en la claridad de comprensión ha sido la lentitud en la ejecución del programa. Así y todo, para el trazado de una circunferencia de 300 unidades de radio se necesitan 0,52 segundos en código máquina frente a los 1,9 segundos en nuestra versión original en BASIC.

Pueden hacerse útiles mejoras a la rutina alterando el valor de PNUM en la línea 30, cambiando del 69 al 5. Esto hará que el programa dibuje líneas en lugar de puntos por lo que creará un disco sólido de color en vez de un círculo. Este cambio provocará el efecto colateral de dibujar una línea no deseada por cada círculo. Para evitarla, hay que añadir esta sentencia:

```
1745 VDU29,0;0;:MOVE P1%,P2%n
```

Lo mismo se puede hacer en assembler con la siguiente línea:

```
575 LDA#25:JSR OSWRCH:LDA#4:JSR
    OSWRCH:LDA#0:JSR OSWRCH:LDA#0:JSR
    OSWRCH:LDA#0:JSR OSWRCH:LDA#0:JSR
    OSWRCH
```

Realizando mejoras más complicadas, el programa llegará a dibujar arcos y elipses.

Estrella solitaria

A la firma norteamericana Texas Instruments se la puede considerar la creadora de la revolución del ordenador personal

El presidente de TI
El presidente y principal ejecutivo de Texas Instruments, J. Fred Bucy, ocupa el cargo de presidente desde abril de 1976

Introducido en 1978, el ordenador personal TI99/4A utilizaba el microprocesador de 16 bits TMS9900, de Texas Instruments, y tenía 16 K de memoria RAM. La máquina disponía de una selección de 16 colores y tres canales de sonido. A pesar de las excelentes ventas, el ordenador, sin embargo, fue víctima de la implacable guerra de precios que tuvo lugar en Estados Unidos en 1982 y 1983. A consecuencia de ella, a principios de este último año TI abandonó por completo la producción de la máquina.

Richard Mann, director de relaciones públicas para Europa de Texas Instruments, afirma que "el TI99/4A se estaba vendiendo muy bien, pero no era rentable y perdimos varios centenares de millones de dólares". A la pregunta de si TI tiene previsto lanzar otra máquina, responde: "Rotundamente, no. Eso lo dejamos bien claro en su momento. A nosotros nos interesa más vender máquinas de cientos de dólares que máquinas de decenas de dólares". No obstante, TI tiene la intención de mantenerse "firmemente en el mercado del ordenador profesional".

Los otros productos de la empresa van desde juguetes electrónicos para niños, como el *Speak and Spell*, a miniordenadores y sofisticados equipos para detección de movimientos sísmicos. Posee 15 plantas de producción distribuidas por todo el mundo y un movimiento anual de cuatro billones de dólares.

Texas Instruments fue fundada en 1930 por dos científicos, John Karchner y Eugen McDermott, y originalmente se denominó Geophysical Service Incorporated. Karchner había estado investigando sobre la idea de hacer rebotar ondas sonoras en los estratos geológicos para calcular su profundidad, y la empresa se estableció en la ciudad de Dallas (Texas), para venderle esta idea a la industria petrolífera.

Geophysical Service Incorporated fue creciendo constantemente durante los años treinta, desarrollando nuevas técnicas y equipos para peritación sismológica, y durante la segunda guerra mundial sus equipos de peritación resultaron de gran utilidad para detectar submarinos, lo que determinó que GSI creara un departamento de laboratorio y fabricación. En 1951 esta rama de la empresa matriz había crecido hasta tal punto que se decidió establecerla por sí sola. La nueva empresa fue bautizada como Texas Instruments.

Al año siguiente, TI obtuvo una licencia de Bell Laboratories para fabricar los transistores, que se habían inventado recientemente, y en 1954 produjo la primera radio a transistores del mundo. En 1958 Jack Kilby, un ingeniero de la empresa, inventó el circuito integrado, y desde entonces TI ha permanecido en la vanguardia de la investigación en este campo. Kilby también colaboró en la invención de la primera calculadora electrónica de mano del mundo, en 1967.

El ordenador TI Professional, que fue lanzado al mercado en enero de 1983, está basado en el procesador 8088 y puede, por consiguiente, ejecutar CPM-86 y MS-DOS. Posteriormente, durante ese mismo año, la empresa introdujo el ordenador de oficina portátil Texas Instruments y el micro de mano CC 40.

Actualmente, la empresa también produce una amplia variedad de componentes electrónicos, incluyendo chips de RAM de 64 Kbytes, de los que declara ser uno de los mayores proveedores de todo el mundo. Los chips de TI aparecen en casi todos los ordenadores personales, incluyendo el Commodore 64 y el Sinclair Spectrum. Al igual que la mayoría de los otros fabricantes de chips, TI ha desarrollado una gama de procesadores de 16 bits, entre la que se incluye el TMS9900, incorporado en el TI99/4A.

Este procesador fue inusual por el hecho de no tener registros internos, que se incluyeron en una "memoria temporal de trabajo" (*sratchpad memory*) especial exterior a la CPU. Richard Mann afirma: "El TMS 9900 estuvo un poco adelantado para su tiempo. Tenía muchas características eficaces, pero no logramos que se aceptara su arquitectura". Ahora Texas Instruments ha comercializado un procesador de 16 bits denominado TMS99000 y la empresa confía en que éste obtendrá un mayor éxito entre los usuarios.

Planta de Texas Instruments
Gran parte de la fabricación de los equipos de Texas Instruments se realiza en estas modernas instalaciones Expressway en Dallas (Texas)

Compresor de datos

"Organiser" (organizador) es un revolucionario y eficaz microordenador que puede caber en el bolsillo del usuario

Ian McKinnell

El Psion Organiser tiene el mismo tamaño y peso que una máquina de afeitar eléctrica y a primera vista tiene precisamente ese aspecto. Una carcasa rígida envuelve el teclado tipo calculadora, que posee 36 pequeños botones. Éstos incluyen las letras del alfabeto (el Organiser sólo utiliza caracteres en mayúsculas) y varias teclas de función. A los dígitos del 0 al 9, a algunas funciones matemáticas y a los signos de puntuación se accede pulsando la tecla Shift (de cambio). Encima del teclado hay una visualización en cristal líquido de 16 caracteres, que dispone de un contraste regulable de modo que se la puede leer desde cualquier ángulo. Cada carácter se compone de una matriz de puntos de cinco por ocho.

En la parte posterior de la máquina hay dos ranuras para insertar los "paquetes de datos" que proporcionan el almacenamiento a largo plazo. El Organiser viene equipado con paquetes de datos de ocho Kbytes, aunque también hay a la venta unos paquetes de 16 Kbytes.

Cuando se pulsa el botón ON de la parte superior izquierda del teclado, la visualización muestra la hora en un reloj de 24 horas y la fecha. Las cuatro funciones principales del Organiser se obtienen mediante la pulsación de un único botón. Como explica claramente el manual, no hay necesidad de teclear instrucciones. Estas funciones principales son:

- SAVE: Desde el teclado se pueden introducir registros de datos de hasta 200 caracteres de longitud, editarlos con los dos controles del cursor (izquierda y derecha) y la tecla Delete (borrar), y después guardarlos (SAVE) en cualquiera de los paquetes de datos.

Archivo rápido

En Psion Organiser, concebido como un ordenador de bolsillo con una poderosa base de datos, proporciona un sistema de archivo rápido e instantáneo. Con funciones de calculadora incorporadas y la capacidad de almacenar y recuperar información vital con suma rapidez, el Organiser puede manipular algunas de las funciones normalmente restringidas a microordenadores de escritorio o manuales más grandes

• FIND: Esta función recupera datos desde cualquiera de los paquetes de datos. Se puede efectuar la búsqueda de una serie de hasta 15 caracteres. La serie requerida se entra sobre la visualización y la máquina la compara con los registros almacenados. La instrucción FIND es muy rápida: el tiempo típico para buscar en una base de datos de 21 Kbytes es de ocho segundos. Los registros que contienen la serie requerida se visualizan por el orden en que están dispuestos. Usando las teclas del cursor se puede hacer rotar un registro en cualquier sentido.

Hemos utilizado el paquete de datos de Psion llamado *Restaurant guide to London* (Guía de restaurantes de Londres), una base de datos de 16 Kbytes que contiene los nombres de 105 restaurantes, cada uno de ellos con dirección, número de teléfono, estación de metro más cercana, tipo de comida, horario de atención al público, orientación sobre precio aproximado del cubierto y un comentario general. Una búsqueda de la serie "ARB" da como resultado un restaurante llamado "BARBARELLAS" y también un restaurante próximo a la estación de metro de "MARBLE ARCH". Esta función se puede comparar favorablemente, en cuanto a velocidad y conveniencia, con muchos programas de base de datos para microordenadores.

• ERASE: Elimina un registro determinado de su paquete de datos. En el teclado del Organiser no hay ninguna tecla de función ERASE (borrar) y a esta instrucción sólo se puede acceder después de haber utilizado la instrucción FIND. Con el registro en la visualización, el usuario pulsa la tecla Mode (modalidad) para obtener la facilidad ERASE. Pulsando entonces el botón Execute (ejecutar), el registro se borrará de la memoria. Sin embargo, esto no deja libre ningún espacio de almacenamiento para su reutilización, sino que tan sólo hace que el archivo quede inaccesible para la CPU. Más adelante en este mismo capítulo analizaremos la naturaleza de la memoria de los paquetes de datos.

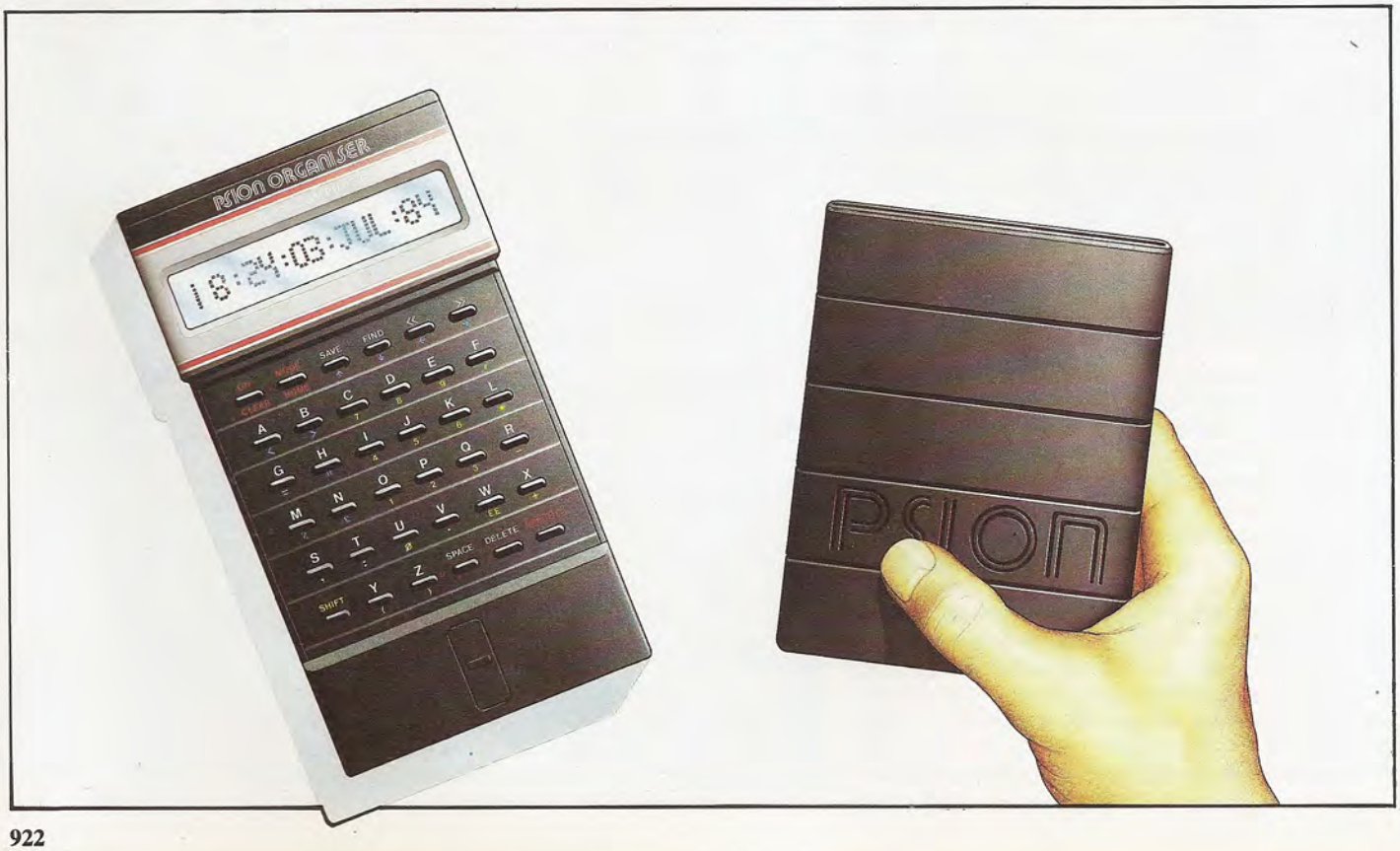
• CALC: Mediante esta instrucción se puede utilizar el Organiser como una calculadora de cuatro funciones (suma, resta, multiplicación, división). El resultado da siete cifras significativas y se utiliza la

Sistemas contrastados

A fines de evaluación, hemos comparado al Psion Organiser con un sofisticado sistema de agenda/diario/notas de hojas sueltas

	PSION	FILOFAX
DIMENSIONES	142×78×29,3 mm	180×124×30 mm
PESO	225 g	200 g
VISUALIZACIÓN	LCD 16 caracteres	Entrada manuscrita-mecanografiada por el usuario
ALMACÉN.	10 900 caracteres por rampak de 8 Kbytes	Hasta 200 páginas
FACILIDADES DE BÚSQUEDA	A partir de cualquier pista de hasta 15 caracteres	Manual, índice alfabético
PAQUETES DISPONIBLES	Rampak de 8 o 16 Kbytes; científicos, ingeniería, matemáticas, contabilidad y financieros, relación de restaurantes	Diario, páginas de teléfonos-direcciones, contabilidad de costos, hoja electrónica, mapas, papel para apuntes y papel pautado para música
OTRAS FACILIDADES	Dice la hora y la fecha. Se puede utilizar como calculadora	Viene en una elegante cartera de piel con lugar para almacenar tarjetas comerciales y de crédito

Potencia a mano
El teclado Psion contiene el abecedario completo, dispuesto por orden alfabético. Este trazado no es intuitivo y puede plantear dificultades a quienes estén habituados a un teclado QWERTY. La tecla de cambio (SHIFT) proporciona acceso a un teclado numérico completo, funciones de calculadora, instrucciones para movimiento del cursor y para la base de datos incorporada



Kevin Jones

notación científica para números muy grandes o muy pequeños. Nuevamente, el teclado del Organiser no dispone de ninguna tecla etiquetada como CALC: a ella se accede pulsando la tecla Mode en cualquier momento, excepto después de haber ejecutado una búsqueda FIND.

La simplicidad y la sencillez de uso de la máquina corren parejas con su reducido precio. El Organiser es viable comercialmente porque la tecnología de baja potencia también es de bajo costo. La máquina puede funcionar mediante una sola pila PP3 de nueve voltios hasta seis meses, porque el sistema de circuitos utiliza chips CMOS (*Complementary Metal Oxide Semiconductor*: semiconductor de óxido metálico complementario) y los paquetes de datos son chips EPROM (*Erasable Programmable Read-Only Memory*: memoria programable de lectura solamente, que puede borrarse), ambos con niveles muy reducidos de consumo de potencia. Hasta hace muy poco, el costo de estos tipos de chip habría significado un precio demasiado alto, que hubiera comprometido su buena comercialización.

En el corazón del Organiser hay un procesador de ocho bits Hitachi 6301X que funciona a 0,93 MHz y está controlado por cuatro Kbytes de ROM. También hay dos Kbytes de RAM para los datos introducidos, información en pantalla y "espacio de trabajo" de la calculadora. Los paquetes de datos EPROM que se introducen en la parte inferior de la carcasa son el equivalente de la memoria RAM de otros microordenadores. Se puede pensar en la EPROM como una RAM "de una sola vez": los datos en ella escritos se almacenan y no se pueden quitar, si bien se puede acceder a ellos como si fuera una ROM normal. Por lo tanto, es una ROM "programable". Utilizando la función ERASE del Organiser en un registro del chip no se libera su espacio de almacenamiento en la EPROM, sino que se marca el registro para indicarle al procesador que está eliminado. Por consiguiente, los paquetes de datos se van llenando hasta que finalmente no se pueden agregar registros nuevos. La única forma de borrar los datos almacenados y restaurar el chip al estado "limpio" original es exponerlo a una luz ultravioleta intensa. Ésta es la función del Psion Formatter. Los chips de paquetes de datos nuevos se pueden formatear hasta 100 veces.

La ventaja que tiene la EPROM sobre la RAM es que, una vez que los datos han sido almacenados, no es necesaria una fuente continua de potencia para mantenerlos, mientras que la RAM es "volátil" y requiere un consumo constante de energía eléctrica. Asimismo, los chips de EPROM son muy fiables, y el costo de cada byte representa una quinta parte del de la RAM. Los utilizan comúnmente los fabricantes de ordenadores en los prototipos de sistemas en vez de ROM, porque en el caso de que se produzcan errores de software, los chips se pueden volver a utilizar y a programar, mientras que una ROM programada erróneamente es irrecuperable. La ROM se puede emplear después de que se hayan descubierto todos los fallos en el software contenido en EPROM.

La comunicación con otros dispositivos informáticos se consigue mediante una interface RS232; ésta se coloca en una de las ranuras para paquetes de datos y permite transmitir o recibir información a velocidades de hasta 9 600 baudios. Los protocolos se pueden programar utilizando el software que se suministra con la interface, de modo que, por ejemplo, se puede formatear la salida a una impresora por longitud de páginas y anchura de línea.

El Organiser se puede emplear en una gran variedad de situaciones para las que se necesita un ordenador potente y portátil. Se podría utilizar para anotar resultados experimentales o para llevar una agenda de citas, aunque el teclado limita seriamente la velocidad de digitación porque está dispuesto de forma alfabética. La mejor forma de utilizar la máquina es la de almacenar información que requiera una gran cantidad de referencias cruzadas (p. ej., listas de precios de vendedores, la relación indexada de una biblioteca, o los nombres, direcciones y números de teléfono de las amistades). También puede ser útil para aplicarle un programa de ordenador a datos experimentales sobre la marcha, o para almacenar información con el fin de procesarla luego en otro lugar.

Ordenadores de bolsillo

Durante estos últimos años se han realizado intentos por comercializar ordenadores de bolsillo de una u otra clase. Éstos se han vendido moderadamente bien, pero nunca han logrado ni siquiera una fracción de las ventas de la calculadora de bolsillo. Algunos ordenadores de bolsillo son poco más que calculadoras con pretensiones. Éstos se pueden programar en sus propios lenguajes, pero suelen atraer sólo a ingenieros y científicos que necesitan un medio de realizar complicados cálculos sobre la marcha. Por encima de ellos, el siguiente peldaño es el de los ordenadores de bolsillo que se pueden programar en BASIC. Este hecho los hace más adecuados para fines generales y, por consiguiente, se han realizado para ellos programas para diversos usos.

Ni siquiera los mejores ordenadores de bolsillo se han hecho auténticamente populares. No se ha escrito software para permitir que la gente lleve con ellos sus diarios de citas ni sus agendas de direcciones. Y aunque así se hubiera hecho, las memorias de los ordenadores suelen ser demasiado pequeñas como para retener una cantidad útil de información. Los teclados, a causa de su reducido tamaño, son difíciles de utilizar, de modo que la mayoría de las personas prefieren usar lápiz y papel para sus diarios y agendas. Las versiones informatizadas de estos dos objetos sólo ofrecen una ventaja cuando son precisas sofisticadas facilidades de búsqueda

Hewlett Packard 41C
Ésta es una calculadora programable muy sofisticada. Está a la venta en tres modelos: HP-41C, HP-41CV y HP-41CX. La 41CX tiene 0,5 K de memoria; los otros dos modelos cuentan con 2 K. A pesar de que la calculadora posee teclado alfabético, éste sólo está destinado a la programación y, por lo tanto, la calculadora no trata textos

Sharp PC 1500

Sharp PC 1251
Sharp fabrica dos ordenadores de bolsillo, el PC 1251 y el PC 1500. El primero es el ordenador de bolsillo más pequeño que existe en el mercado, y el segundo es el más potente. Ambos se programan en BASIC, si bien el PC 1500 posee instrucciones extras. Los dos tienen 4 K de memoria estándar, pero el PC 1500 se puede ampliar a 20 K

Texas Instruments TI-66
Se trata esencialmente de una calculadora programable y, como tal, es atractiva para científicos y matemáticos. No trata textos, de modo que no es adecuada para aplicaciones de carácter más general. Sólo posee 0,5 K de memoria

Casio FX700P
Este es uno de los varios ordenadores de bolsillo fabricados por Casio que se pueden programar en BASIC. Tiene 2 K de memoria y un diminuto teclado alfabético. También se fabrica una versión con una pequeña impresora térmica incorporada

Manos ganadoras

El bridge es un juego de naipes que despierta en los aficionados un apasionado interés y se presta muy bien a ser informatizado

Los programas de bridge disponibles a nivel comercial pertenecen a dos categorías claramente definidas, según estén diseñados como medios de enseñanza o como paquetes para jugar.

Es posible hallar en el mercado algunos buenos paquetes de enseñanza. Uno de los mejores, desde el punto de vista del diseño y la presentación, es la serie Bridgemaster. Existen versiones de este paquete para el Spectrum, ZX81, BBC Modelo B, Electron y Commodore 64. El diseño de esta serie pertenece a Terence Reese, ex campeón del mundo y autor de numerosos libros sobre este juego de naipes.

El principio en el que se basan todos los programas de enseñanza de bridge es que el principiante vaya pasando por una serie de manos preestablecidas escritas en el programa. Ésta es la diferencia esencial entre un paquete de enseñanza y un paquete para jugar, que genera manos al azar. Dado que el tutor "sabe" lo que contiene cada una de las manos, el programa es capaz de guiar al jugador paso a paso a través de las muchas reglas y convenciones que hacen que el bridge sea un juego tan atractivo y exigente desde el punto de vista intelectual.

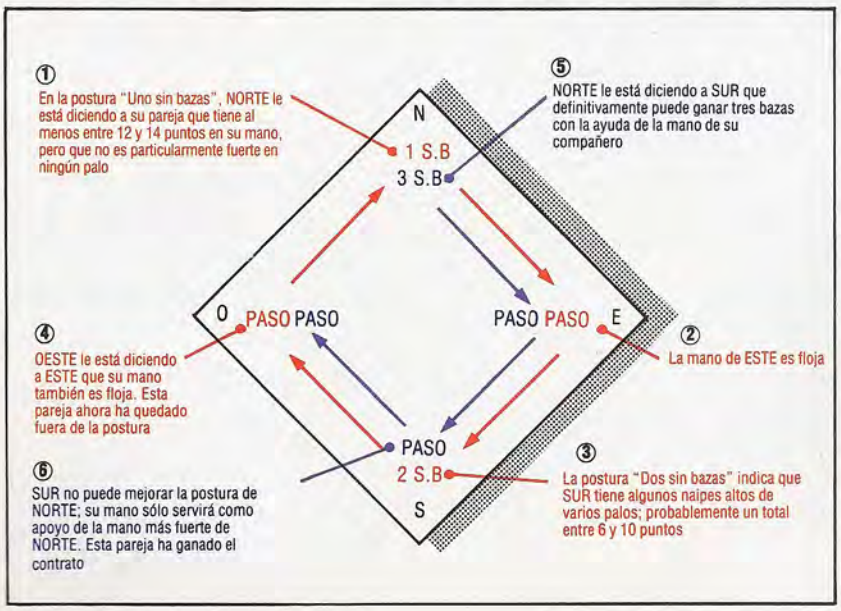
La serie Bridgemaster proporciona un excelente ejemplo de lo que se puede conseguir gracias a un paquete de enseñanza. El cursor consta de dos programas, dirigidos, respectivamente, al principiante absoluto y al jugador aficionado. El primero se denomina *Complete learning package for the beginner at bridge* (Paquete completo de aprendizaje para la iniciación en el bridge); el segundo es el *Expert bridge* (Bridge para expertos). El paquete para principiantes se compone de dos cassettes de programas, dos cintas de comentarios, un escueto folleto de instrucciones y *Begin bridge*, libro de bolsillo escrito por Reese. Este último no se utiliza al mismo tiempo que los programas en la pantalla. Las cintas de comentarios proporcionan todo el apoyo que necesita el principiante.

Los programas de enseñanza de bridge que no poseen comentarios basados en cintas han de proporcionar un manual muy exhaustivo o bien observaciones sumamente explicativas en la pantalla. Por otra parte, también podrían presuponer al menos un conocimiento básico de bridge por parte del usuario y limitarse a mejorar el nivel de juego del mismo.

Para quienes no saben jugar al bridge, la baraja se reparte entre los cuatro jugadores que se agrupan en equipos de dos. En el bridge por ordenador, se adopta generalmente la convención estándar de designar a los cuatro jugadores (Norte, Sur, Este y Oeste). La postura precede a la partida, y en ella los jugadores, sin mostrarse el juego entre sí, deben declarar la fuerza de los naipes que tienen de mano. La fuerza de una mano está determinada por la cantidad de cartas altas o la cantidad de cartas de un palo o ambas. En nuestro diagrama, mostramos un ejemplo de postura. La postura ganadora determinará el contrato. Éste selecciona qué color ha de ser el triunfo así como la cantidad de bazas que el bando ganador debe intentar hacer. Después de la postura, se juega la mano. En la partida, el jugador que ha ganado la postura se convierte en el "declarante" y su pareja en el "muerto". Los naipes del "muerto" se ponen a la vista de todos. El declarante ganará o perderá puntos según se haga o no con la cantidad de bazas especificadas en el contrato.

Todos los programas de enseñanza de bridge representan la postura, tanto si permiten que el jugador la gane como si no. En Bridgemaster, al jugador se le da la oportunidad de hacer su propia postura, habiendo visto visualizados sus naipes en la pantalla. El ordenador declara entonces la postura que se ha de realizar. Ésta podría diferir de la postura escogida por el jugador. El programa de Reese, al igual que muchos programas de enseñanza, está arreglado de modo que el ordenador sólo acepte la postura o la partida que el diseñador del programa desea que uno haga. Reese admite que, en ocasiones, esto lleva a que se rechace una jugada perfectamente sensata del jugador en favor de la ruta predeterminada que se ha trazado de antemano.

Además de visualizar su mano y la postura, el programa debe manejar la jugada de la mano. El método de visualización más común es un cuadrado



Postura ganadora
Los valores de los naipes de la postura son: As-4; Rey-3; Reina-2; Valet-1. Las letras en rojo corresponden a la primera postura de los jugadores y las azules representan la segunda

en el centro de la pantalla con los cuatro lados señalados N, E, S y O. Luego se visualizan al completo las manos del Norte y el Sur. Por lo general se muestran los cuatro palos: piques, corazones, diamantes y tréboles, dispuestos por ese orden, con los valores de los naipes visualizados a los lados.

Se supone que el jugador es Sur y que el Norte o bien el Sur serán el «muerto». En cualquier caso, Sur siempre juega los naipes de ambos. Dicho en otras palabras, Sur juega siempre la mano, con independencia de que la postura la haya ganado Norte o Sur. Esto, por supuesto, constituye un alejamiento del verdadero bridge, en el cual un jugador se puede pasar toda la tarde sin ganar el contrato ni siquiera una sola vez.

En un programa de enseñanza esta diferencia no tiene relevancia, excepto en la importante área del juego defensivo. Se necesitan estas tácticas cuando uno desea impedir que sus oponentes cumplan el contrato después de haber ganado la postura. En un programa para jugar al bridge, las consecuencias son más graves y nosotros no conocemos ningún programa para jugar al bridge que juegue una secuencia en la que Este y Oeste ganen la postura. Si se evita la postura en, por ejemplo, el excelente programa de bridge de CP Software para el Spectrum de 48 Kbytes (es decir, si se permite de forma deliberada que Este u Oeste sean el declarante), el programa imprimirá en la pantalla un mensaje indicándole que no está diseñado para jugar en esas circunstancias e indicándole que pulse la tecla R para generar una mano nueva.

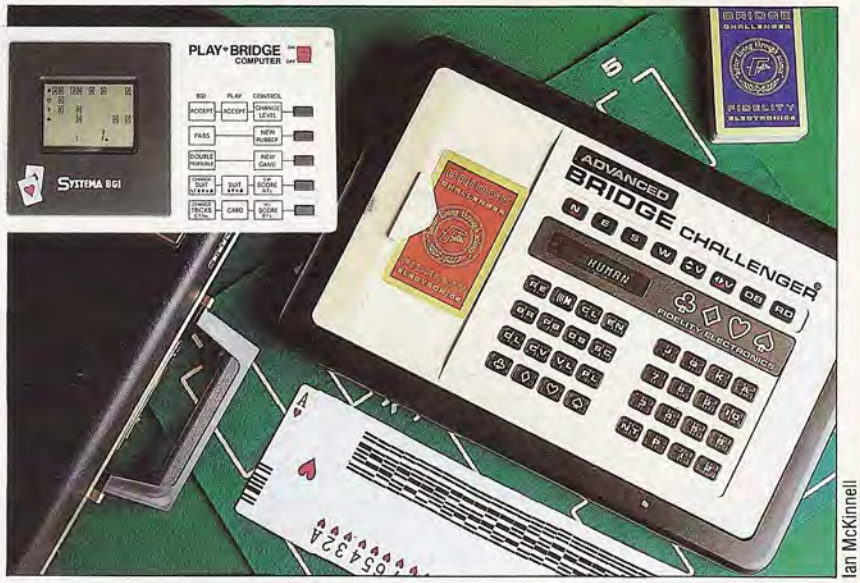
Cuando se ha completado la postura, se visualiza el juego de la mano. De acuerdo a quién gane la baza, el ordenador (que juega por E y O) o bien el jugador (que juega por N o S) selecciona un naipe. Los naipes de E y O aparecen en la pantalla de a uno, tal como sucedería en la mesa en el juego normal.

Los programas basados en ordenador ofrecen una gran cantidad de características útiles. Bridgemaster, por ejemplo, permite que el jugador elija entre cuatro opciones antes de cada turno de dar, según el nivel de competición. P juega las manos con el jugador humano seleccionando juegos para N y S (guiado por Reese); A selecciona *autoplay* (automático), en la que la mano termina sin su colaboración; H visualiza en la pantalla las cuatro manos, mientras que D selecciona un turno diferente. Este juego de opciones es el habitual en la mayoría de los programas de bridge.

La facilidad *autoplay* es especialmente útil en combinación con el comentario de Reese, porque el jugador no se tiene que distraer con la tarea de seleccionar naipes. Todos los turnos se pueden volver a jugar una y otra vez. Los programas para jugar al bridge también disponen de estas características, pero en este caso no actúan como repetición de ejercicio en ciertos puntos del aprendizaje, sino como *post mortems* de la postura y la partida.

Hemos analizado algunos de los principios de los paquetes de enseñanza, pero existen también en el mercado algunos programas para el jugador más avanzado que desea perfeccionar su juego.

El paquete *Expert bridge* de la serie Bridgemaster que hemos comentado previamente es un curso de instrucción avanzado bien presentado que analiza las complejidades de incidencias como manos de alta puntuación, "slam", aprietos y bloqueos.



Ian McKinnell

Desafío exclusivo
Las dos máquinas de la fotografía son ordenadores para jugar al bridge. El primero (derecha) es un juego de mesa llamado Advanced Bridge Challenger, de Fidelity Electronics. Bridge Challenger está recomendado sólo para jugadores de bridge experimentados, porque su dominio exige mucha práctica y concentración. Un gran inconveniente del Bridge Challenger es su diminuta pantalla de ocho caracteres. La segunda máquina (izquierda) es el ordenador Systema Play Bridge. Funciona a pilas. Comprender el Play-Bridge sólo ocupa algunos minutos; no posee ninguna característica avanzada, pero jugar con él puede ser apasionante

Otro paquete que merece la atención del jugador más experimentado es el programa para jugar al bridge de Alligata Software. Existen versiones para el Oric, BBC Modelo B, Electron y Commodore 64. El programa presupone cierto conocimiento de bridge por parte del jugador y con este paquete no se suministra ningún manual, de modo que los jugadores que no tengan buena memoria deberán disponerse a tomar notas.

El programa de CP Software para el Spectrum es muy entretenido para jugar. No se juega con la mano como declarante, de modo que no se puede practicar un juego defensivo. No obstante, como la mayoría de los jugadores aficionados, que constituyen el 98 % de la comunidad de jugadores de bridge de todo el mundo, desean jugar todas las manos de cualquier manera, éste no se considera un inconveniente serio.

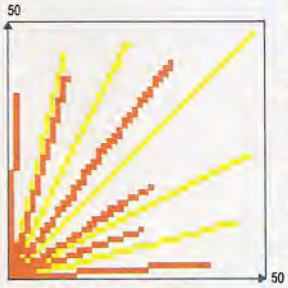
Los programas de bridge, tanto los de aprendizaje como los de juego, constituyen una manera ideal de aprender a jugar y de mejorar los puntos débiles que uno tenga. El bridge es un juego que exige muchas habilidades diferentes: una buena memoria, aptitudes analíticas, capacidad para trabajar en pareja, perspicacia para jugar con cautela y saber cuándo asumir un riesgo, y habilidad para mantenerse inescrutable y conservar la cabeza fría. Los juegos que hemos analizado en este capítulo no simularán a la perfección una partida de bridge entre cuatro jugadores; pero, por cierto, le proporcionarán una práctica valiosa y estimulante.

Ian McKinnell

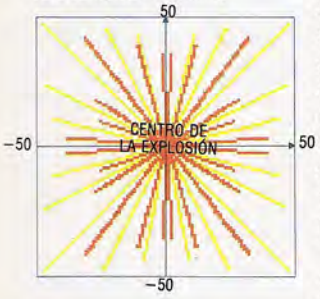
Elija un paquete
Existen varios paquetes de bridge a la venta, que van desde programas pedagógicos a complicados juegos de desafío que ponen a prueba la pericia de los jugadores más experimentados.
En la fotografía vemos (de izquierda a derecha) *Bridge master*, disponible en cassette para el Spectrum, ZX81, BBC Modelo B, Electron y Commodore 64; *Bridge*, de Alligata Software, en cassettes para el Oric, BBC Modelo B, Electron y Commodore 64; y *Bridge player*, de CP Software, que se vende en cassette para el Spectrum de 48 Kbytes

Explosión

He aquí los procedimientos destinados a crear los efectos visuales y sonoros de una explosión



Jugada explosiva
Los gráficos se crean generando líneas de longitud al azar comprendidas entre una escala de −50 a 50 unidades desde el centro. La explosión se construye por cuadrantes y se reproduce a modo de espejo para completar el efecto.



En el último capítulo de nuestro proyecto para crear el juego "Campo de minas", destinado al BBC Micro y al Electron, estudiamos cómo el programa detectaba las colisiones con las minas. En esta ocasión analizaremos los procedimientos que nos permitirán crear los efectos visuales y sonoros de las explosiones.

El BBC Micro y el Electron admiten 16 posibles variaciones de color que podemos utilizar en la creación de un efecto de explosión. En realidad sólo hay ocho colores distintos, siendo las otras ocho variaciones efectos de intermitencia (*flashing*) entre un color y otro. Normalmente, las combinaciones intermitentes de colores se alternan cada medio segundo, pero se puede cambiar esta periodicidad mediante dos instrucciones FX. *FX9 establece la duración del tiempo durante el cual se visualiza el primer color del par intermitente. El tiempo se mide en unidades de cincuentavos de segundo. Por consiguiente, *FX9,20 altera el tiempo de visualización del primer color a 2/5 (dos quintos) de segundo. La otra instrucción FX, *FX10, se puede utilizar para alterar el tiempo de visualización del segundo color de la misma manera. Modificando uno o ambos tiempos de visualización se pueden producir interesantes efectos centelleantes.

El efecto visual de la explosión se puede producir dibujando una serie de líneas cortas en diversos colores que partan desde un punto central. Para mejorar el efecto, también se puede cambiar rápidamente el color de las minas y de las letras del marcador. Como éstos se imprimirán originalmente en el color lógico 2 (que hemos establecido en verde) podemos utilizar la siguiente instrucción para reasignarle al color lógico 2 un color seleccionado al azar entre las 16 posibles combinaciones:

```
VDU19,2,RND(15),0,0,0
```

RND(15) selecciona un número entero entre 1 y 15. Ello significa que el color 0 no se selecciona nunca; pero como el color 0 es negro, el color del fondo, esto no tiene importancia alguna.

El color utilizado para trazar las líneas de la explosión también se puede seleccionar al azar utilizando GCOL 0,RND(3). Luego empleamos un bucle para dibujar una serie de líneas desde el centro de la explosión en una serie de colores seleccionados al azar. Hasta ahora hemos analizado las instrucciones de alta resolución MOVE y DRAW. DRAW(X,Y) siempre traza una línea hasta el punto con coordenadas "absolutas" (X,Y). Sin embargo, existe otra familia de instrucciones para dibujar en alta resolución que nos permiten, entre otras cosas, especificar puntos "relativos" entre sí. Se puede emplear la instrucción PLOTK,X,Y para dibujar líneas entre puntos relativos o absolutos en función del valor de K. La siguiente tabla muestra algunas de las variaciones que son posibles utilizando PLOT:

K=0	Movimiento relativo al último punto
K=1	Trazar línea hasta posición relativa en color de primer plano
K=2	Trazar línea hasta posición relativa en color lógico inverso
K=3	Trazar línea hasta posición relativa en color de fondo
K=4	Movimiento a posición absoluta
K=5	Trazar línea hasta posición absoluta en color de primer plano
K=6	Trazar línea hasta posición absoluta en color lógico inverso
K=7	Trazar línea hasta posición absoluta en color de fondo

PLOT1 y PLOT5 son equivalentes a la utilización de las instrucciones MOVE y DRAW. Para nuestro efecto de explosión usaremos PLOT1 para dibujar líneas relativas al centro de la explosión. Sin embargo, antes de dibujar cualquier línea es necesario saber dónde está el centro de la explosión.

En el último capítulo del curso, cuando llamamos a una versión ficticia de PROCexplosión, les pasamos a x-explosión e y-explosión los valores de xgraf e ygraf. Éstos no jugaban ningún papel en el procedimiento ficticio, pero ahora los utilizaremos para especificar el centro de la explosión. Si se llama a este procedimiento desde la línea 3390 de PROCmover (véase p. 915), entonces xgraf e ygraf serán las coordenadas de gráficos del centro de la celda de carácter que contiene la mina. Por tanto, MOVEx-explosión,y-explosión desplazará el cursor para gráficos hasta el centro del lugar en el que deseamos que se produzca la explosión. El motivo por el cual pasamos las coordenadas a otro par de variables para su empleo en el procedimiento de la explosión, en vez de utilizar simplemente xgraf e ygraf, es que este procedimiento también será llamado desde otra parte del programa en la cual las coordenadas del centro de la explosión se especificarán mediante un par diferente de variables. Pasando parámetros al procedimiento de la explosión lo hacemos más flexible y general.

Después de habernos desplazado (MOVE) hasta el centro de la explosión, trazaremos una línea en una dirección escogida al azar de, por ejemplo, un máximo de 50 unidades:

```
PLOT1,RND(50),RND(50)
```

no realizará del todo esta tarea, porque no especifica coordenadas negativas. (Una coordenada *x* negativa produce una línea trazada hacia la izquierda, mientras que una coordenada *y* negativa da una línea trazada hacia abajo desde el centro de la explosión.) El uso repetido de esta instrucción rellenará sólo la cuarta parte del espacio alrededor del centro (donde ambas coordenadas son positivas).

Para que podamos introducir valores negativos (y dibujar entonces líneas en todas las direcciones alrededor del centro de la explosión) debemos reemplazar RND(50) por RND(100)−50. El valor máximo de RND(100) es, por supuesto, 100, de modo que el valor máximo que puede tener

RND(100)−50 es 100−50=50. Por otra parte, el valor mínimo de RND(100) es uno. Así, RND(100) puede ser tan pequeño como 1−50=−49.

Mediante un movimiento (MOVE) repetido hacia el centro de la explosión y trazando una línea relativa de hasta 50 unidades en cualquier dirección y color podemos producir un efecto visual atractivo.

Consideremos ahora cómo podemos producir efectos de sonido para acompañar los gráficos explosivos de la pantalla. Se pueden crear sonidos bastante sofisticados utilizando las instrucciones SOUND y ENVELOPE del BASIC BBC. SOUND puede generar notas con tonos (de modo que se pueda tocar una melodía) o bien "ruido" para efectos especiales de sonido. La instrucción ENVELOPE se emplea junto con SOUND para "dar forma" al sonido que uno oye y simular, por tanto, instrumentos musicales. Hay cuatro números (o parámetros) relacionados con SOUND que controlan las características del tono producido. Éstos son los siguientes:

```
SOUND C,A,P,D
```

- C es el número de canal, que debe tener un valor entre 0 y 3. Los canales del 1 al 3 producen notas con tono, pero el canal 0 es para efectos especiales. Éste es el canal que vamos a utilizar aquí.
- A es la amplitud o, para decirlo de forma más sencilla, el volumen de sonido producido. El volumen máximo se establece cuando A es −15 y pasa a silencio cuando A es cero. Los valores positivos de A se utilizan para seleccionar la forma del sonido y se emplean junto con la instrucción ENVELOPE.
- P suele ser el tono de la nota; en otras palabras, lo agudo o grave que es el sonido.
- D es la duración o período de tiempo durante el cual suena la nota. Los valores de D de 0 a 254 hacen que la nota se mantenga durante un tiempo medido en unidades de una vigésima de segundo. Por consiguiente, para mantener una nota durante un segundo el valor de D se ha de establecer en veinte. Si el valor de D es −1, entonces el sonido continuará hasta que uno emprenda alguna acción para interrumpirlo.

El efecto sonoro de la explosión necesita tener un volumen máximo, de modo que nuestra instrucción tendrá la amplitud máxima, representada mediante un valor de −15. El valor del tono es algo más complicado. Cuando se utiliza con el canal 0, P toma valores entre 0 y 7 de acuerdo a estas reglas:

P=0	Vibración de alta frecuencia
P=1	Vibración de frecuencia media
P=2	Vibración de baja frecuencia
P=3	Vibración, frecuencia establecida por canal 1
P=4	Ruido de alta frecuencia
P=5	Ruido de frecuencia media
P=6	Ruido de baja frecuencia
P=7	Ruido, frecuencia establecida por canal 1

El grupo de cuatro formas de onda de vibración produce sonidos tipo "zumbador", mientras que las formas de onda de ruido producen algo que suena como un televisor antes de sintonizarlo adecuadamente. La instrucción SOUND que utilizaremos en nuestro programa es:

```
SOUND 0,−15,4,40
```

que produce un silbido agudo y dura dos segundos (40/20) con el volumen establecido en el máximo. He aquí el listado completo para el procedimiento:

```
3550 DEF PROCexplotar(x−explosion,y−explosion)
3560 REM ** EFECTO SONORO **
3570 SOUND 0,−15,6,50
3580 REM ** ESTABLECER VELOCIDAD FLASH **
3590 *FX9,20
3600 *FX10,50
3610 FOR I=1 TO 100
3620 MOVE x−explosion,y−explosion
3630 VDU19,2,RND(15),0,0,0
3640 GCOL 0,RND(3)
3650 PLOT 1,RND(100)−50,RND(100)−50
3660 NEXT I
3670 PROCrestaurar
3680 ENDPROC
```

El procedimiento "restaurar"

Tras una explosión hay varias tareas que realizar:

- Debemos reducir la cantidad de vidas que quedan y verificar si se han utilizado ya todas ellas.
- Debemos poner en orden todo el lío que se ha producido en la pantalla a resultas de la explosión.
- Debemos reposicionar el detector y el ayudante en su posición de comienzo.

La primera tarea se consigue mediante el empleo de un contador que se incrementa cada vez que se llama al procedimiento. Si el contador, después del aumento, es mayor de cuatro, se pone a uno una variable especial, flag final (indicador de final), para señalar que el juego ha concluido.

Habiendo verificado si el juego ha terminado, debemos limpiar la explosión. Para esto se podrían utilizar diversos métodos, como imprimir (PRINT) espacios en blanco encima de la superficie de la explosión. No obstante, el procedimiento que empleamos es limpiar la pantalla, volver a colocar las minas y reimprimir la información relativa a tiempo y marcador. Para mantenernos ecuánimes, deberíamos volver a colocar la misma cantidad de minas que quedaba antes de la explosión. Esto se puede calcular a partir del marcador. Dado que cada mina vale 150 puntos y que originalmente hay 50, podemos calcular la cantidad de minas que quedan y pasarla al procedimiento "colocar minas".

El detector de minas y el ayudante se vuelven a posicionar reinicializando sus coordenadas (llamando al procedimiento inicializar variables) y llamando luego al procedimiento situar sujetos. Abajo se ha listado el procedimiento restaurar completo. Agregue este listado y el procedimiento explosión.

```
3880 DEF PROCrestaurar
3890 contador=contador+1
3900 IF contador > 4 THEN flag−final=1:ENDPROC
3910 CLS
3920 VDU19,2,2,0,0,0
3930 COLOUR 2
3940 PROCinicializar−variables
3950 minas−restantes=50−marcador/150
3960 PROCcolocar−minas(minas−restantes)
3970 PROCtrazar−borde
3980 PRINTTAB(2,27);"Tiempo"
3990 PRINTTAB(2,28)"Marcador"
4000 PRINTTAB(11,28)marcador$
4010 PRINTTAB(2,29)"Max marcador"
4020 PRINTTAB(11,29)max−marcador$
4030 vidas−restantes$=LEFT$(hombres$,4−contador)
4040 COLOUR 1
4050 PRINTTAB(2,30);vidas−restantes$;"  "
4060 COLOUR 2
4070 PROCsituar−sujetos
4080 ENDPROC
```

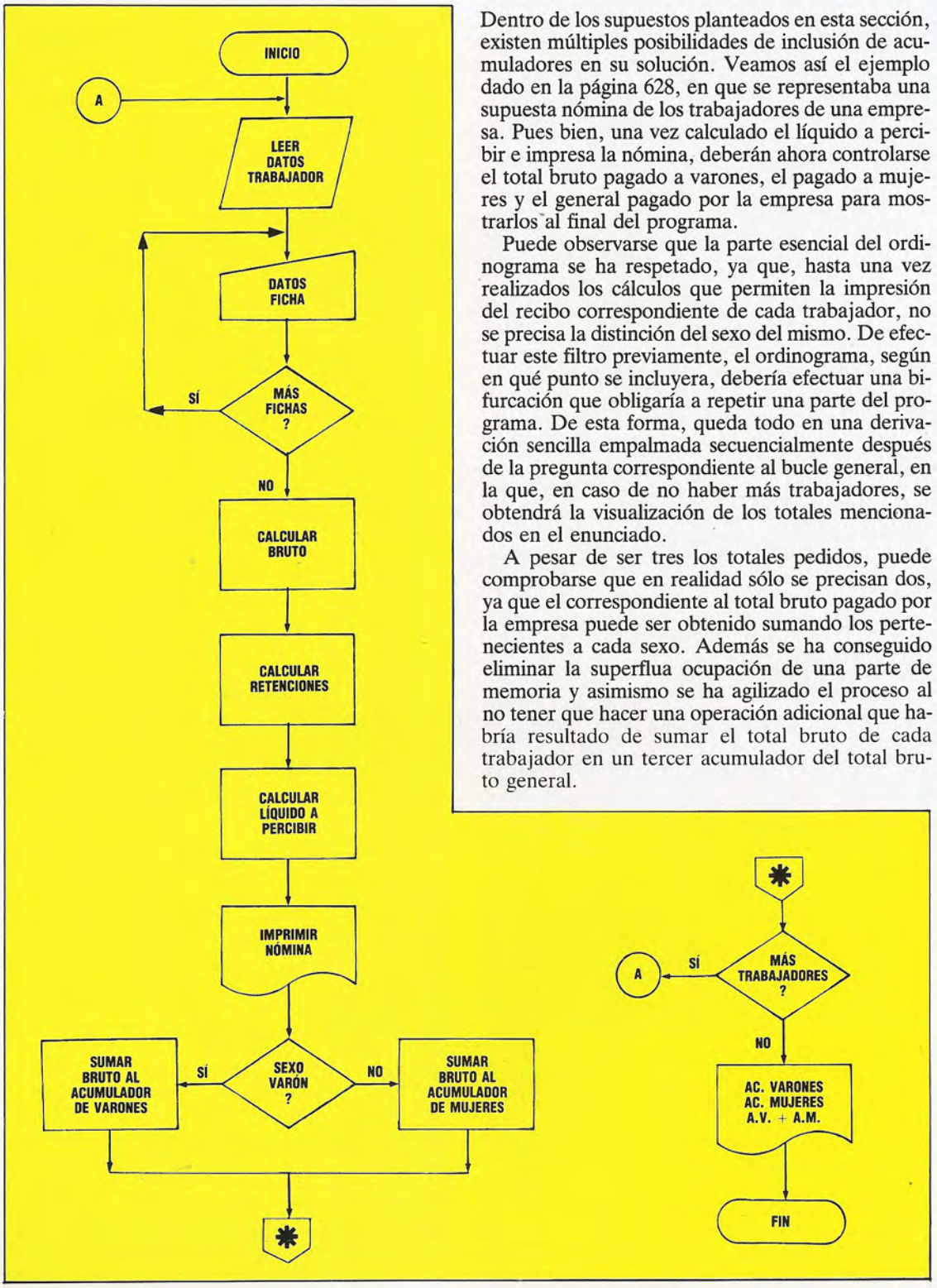
También es necesaria la siguiente modificación en el programa temporal de llamada (véase p. 885):

```
60 UNTIL TIME > 12099 OR flag final=1
```

Kevin Jones

Uso de acumuladores

Puede ser muy práctico incluir acumuladores en la resolución de un problema representado en un diagrama de flujo



Dentro de los supuestos planteados en esta sección, existen múltiples posibilidades de inclusión de acumuladores en su solución. Veamos así el ejemplo dado en la página 628, en que se representaba una supuesta nómina de los trabajadores de una empresa. Pues bien, una vez calculado el líquido a percibir e impresa la nómina, deberán ahora controlarse el total bruto pagado a varones, el pagado a mujeres y el general pagado por la empresa para mostrarlos al final del programa.

Puede observarse que la parte esencial del ordinograma se ha respetado, ya que, hasta una vez realizados los cálculos que permiten la impresión del recibo correspondiente de cada trabajador, no se precisa la distinción del sexo del mismo. De efectuar este filtro previamente, el ordinograma, según en qué punto se incluyera, debería efectuar una bifurcación que obligaría a repetir una parte del programa. De esta forma, queda todo en una derivación sencilla empalmada secuencialmente después de la pregunta correspondiente al bucle general, en la que, en caso de no haber más trabajadores, se obtendrá la visualización de los totales mencionados en el enunciado.

A pesar de ser tres los totales pedidos, puede comprobarse que en realidad sólo se precisan dos, ya que el correspondiente al total bruto pagado por la empresa puede ser obtenido sumando los pertenecientes a cada sexo. Además se ha conseguido eliminar la superflua ocupación de una parte de memoria y asimismo se ha agilizado el proceso al no tener que hacer una operación adicional que habría resultado de sumar el total bruto de cada trabajador en un tercer acumulador del total bruto general.

Unidad transformadora

La nueva interface Electron Plus 1 puede convertir el Acorn Electron estándar en una alternativa económica del BBC Micro

El Acorn Electron se concibió como una versión a escala reducida del BBC Micro. Sus diseñadores conservaron el excelente BASIC BBC y su sistema operativo, pero eliminaron la mayoría de las interfaces que hacen del BBC Micro una máquina tan versátil. De hecho, las únicas conexiones del Electron son una puerta para cassette, dos conectores para pantalla (RGB o video compuesto), una salida de modulador para conectar a un televisor y un conector para la toma de corriente.

La interface Plus 1 le añade a la máquina estándar una puerta para impresora en paralelo, un conector para palanca de mando y dos conectores para cartuchos de ROM enchufables. Acorn espera que las facilidades extras aumenten el potencial de ventas de la máquina. Las ranuras para cartuchos y la puerta para palanca de mando permiten emplear el Electron en juegos de tipo recreativo, mientras que la puerta para impresora es una facilidad útil para aquel usuario que necesite imprimir listados de programas o que desee utilizar software para tratamiento de textos.

La instalación de la unidad de ampliación Plus 1 es muy sencilla: se acopla simplemente en el conector marginal que sobresale de la parte posterior del Electron. Luego se cierran dos pestillos sobre la carcasa del Plus 1 y se fijan a los conectores de rosca de la carcasa del ordenador. Tanto la corriente como los datos se transfieren a través del conector marginal. El software adicional necesario para "activar" la nueva interface está contenido en un chip de ROM dentro de la unidad.

La puerta para impresora es del tipo Centronics en paralelo estándar. Para habilitar la impresora, el usuario ha de entrar la instrucción VDU2, y para inhabilitarla debe entrar VDU3. Estas instrucciones son idénticas a las que se utilizan en el BBC Micro. Igualmente, se emplea VDU1 para enviarle un carácter a la impresora; esto también es estándar en el BASIC BBC.

Las palancas de mando se enchufan en la puerta analógica. Ésta es un conector tipo D de 15 patillas, y las conexiones de las patillas son idénticas a las de la puerta analógica del BBC Micro, lo que permite utilizar cualquier palanca de mando compatible con esta máquina. El Plus 1 puede medir hasta cuatro voltajes analógicos, ya sea provenientes de cuatro paletas para juegos o de dos palancas de mando (cada palanca produce dos voltajes: uno para el movimiento arriba-abajo y otro para izquierda y derecha).

El proceso de conversión de voltajes analógicos a señales digitales que puede tratar el ordenador se lleva a cabo mediante un chip convertidor de analógico a digital; en este caso, un ADC0844. Este chip es menos complejo que el que se utiliza en el BBC Micro, lo que significa que el Electron no sigue el movimiento de una palanca de mando con la misma precisión que lo hace una máquina BBC. En los programas de juegos, en los que la palanca se emplea simplemente como un mando de cuatro direcciones, ello no constituye un problema. Si se emplea la palanca de mando con un programa para gráficos (para dibujar "a pulso" en la pantalla, p. ej.), en el Electron los resultados son bastante inferiores y la imagen resultante menos definida.

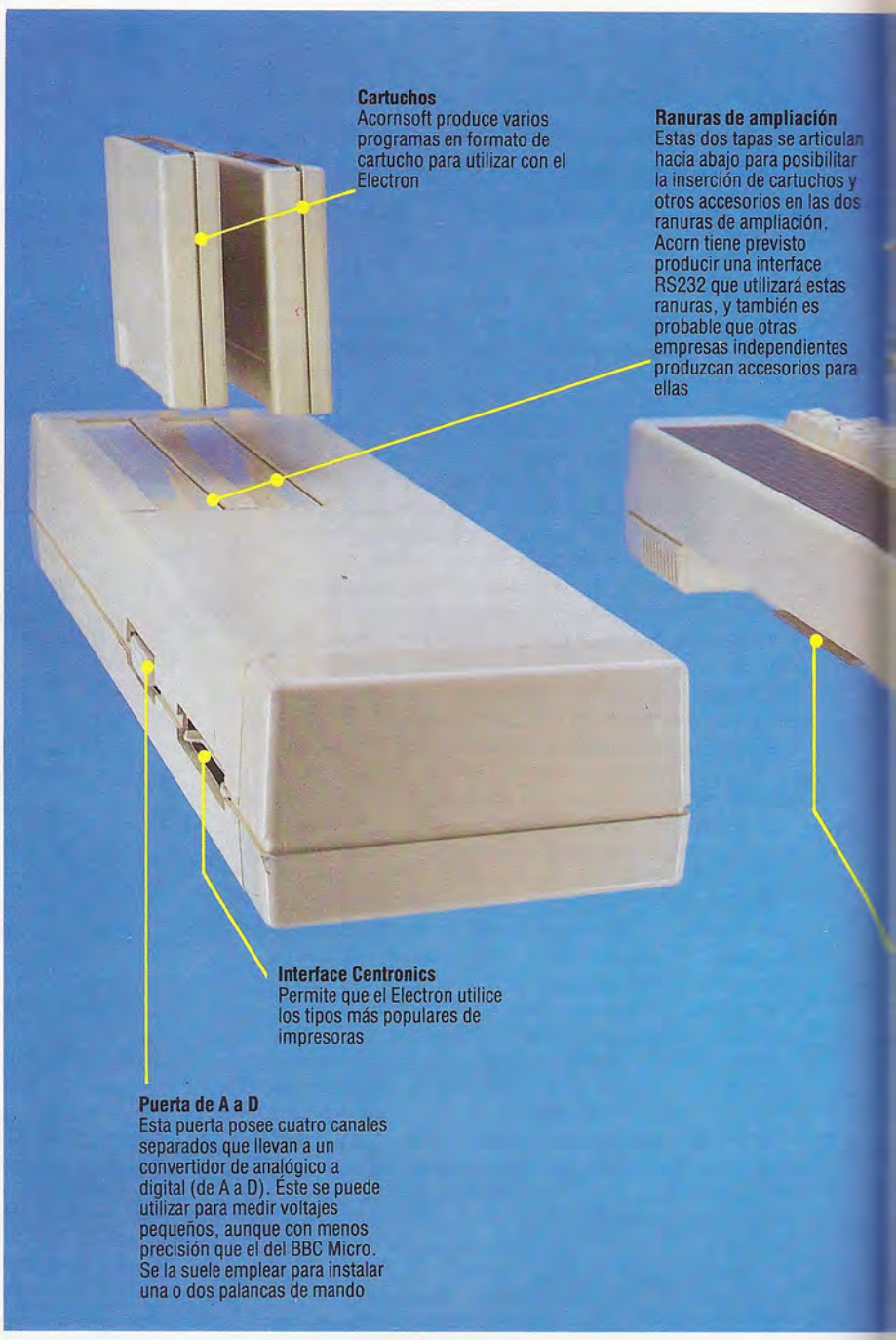
Chris Stevens

Notable ampliación
El Electron se concibió originalmente como una versión de precio reducido del BBC Micro. Carece de casi todas las interfaces del BBC, a pesar de lo cual ofrece los mismos excelentes gráficos y BASIC a la mitad de precio. Ahora Acorn ha producido una unidad denominada Plus 1, que le proporciona al Electron las interfaces más importantes

Cuando se enciende por primera vez la Electron Plus 1, el cartucho del conector de ROM frontal comienza a funcionar automáticamente; para detenerlo se debe pulsar Escape. El sistema de archivo de ROM trabaja de forma muy similar a la versión en cassette (si bien es mucho más rápido), por lo cual interpreta instrucciones como *CAT, LOAD y CHAIN. Para utilizar software en cassette con la nueva unidad, hay que entrar la instrucción *TAPE o quitar el cartucho, recordando desconectarlo de la red previamente. Además de programas de juegos, también se suministran otros lenguajes soportados en cassette. A diferencia de las ROM de juegos, estas ROM de lenguajes se "encienden" para sustituir a la ROM de BASIC usual.

El problema de la mayoría de los accesorios para ordenador es que suelen tener efectos colaterales. Cuando se le conecta al Electron la unidad Plus 1, se pueden producir errores de carga al emplear programas en cassette. Esto afecta especialmente a los programas que contienen archivos de datos; por nuestra parte hemos descubierto que dos de los programas de la cassette de presentación del Electron no se cargan en el Electron ampliado. No obstante, utilizando una serie de instrucciones del sis-

Seis títulos de software
Los cartuchos para el Electron tienen una ventaja sobre la cinta, y es que cargarlos apenas lleva un par de segundos, frente a los varios minutos que lleva la carga de una cinta. En formato de cartucho sólo hay seis títulos a la venta. Éstos son cuatro juegos (entre ellos *Hopper* y *Snapper*), un programa educativo y el lenguaje LISP



tema operativo, uno puede "inducir" al Electron a pensar que no tiene conectada la Plus 1 y, por consiguiente, restaurar su rendimiento normal. En el manual esto casi ni se menciona y puede resultarle confuso al usuario novato.

En cualquier máquina Acorn, teclear *HELP proporciona una lista de las diversas ROM que hay en el ordenador. Tecleándola en el Electron con la unidad Plus 1 instalada, le dice que la principal ROM del sistema operativo es OS 1.00, y también lista Expansion 1.00 ADC/Printer/RS423. Ésta es la ROM del sistema operativo que hay en el interior de la unidad de ampliación Plus 1, pero lo interesante de esto es la última parte de la información: RS423. Ésta es una referencia a una interface en serie estándar, si bien ni el Electron ni la Plus 1 admiten dicha facilidad. De hecho, Acorn tiene en sus planes introducir una interface en serie más adelante.

La principal diferencia entre los dos BASIC es que el Electron carece de la modalidad 7, la modalidad de gráficos compatible con teletexto. Esta modalidad se utiliza para títulos y páginas de instrucciones en los programas para el BBC Micro, porque es económica desde el punto de vista de la memoria. En el Electron, los diversos atributos de videotexto no encienden los colores ni las letras intermitentes de la modalidad 7; en cambio, se visualizan en la pantalla como un galimatías. La mayoría de los programas para juegos utilizan para el juego propiamente dicho una modalidad diferente, de modo que después de las páginas de título e instrucciones ya no existe ningún problema.

La otra diferencia fundamental que existe entre las dos máquinas reside en las instrucciones SOUND y ENVELOPE. El Electron posee un único canal de sonido, frente a los cuatro que ofrece el BBC. Del mismo modo, la instrucción ENVELOPE del Electron sólo afecta al tono y no al volumen. No obstante, las instrucciones son compatibles, de modo que un programa que las utilice funcionará en ambas máquinas, si bien en el Electron el sonido será sensiblemente distinto.

El teclado del Electron tiene mejor "tacto" que el del BBC Micro, pero posee menos teclas. Ello significa que en el Electron, al igual que en el Sinclair Spectrum, la mayoría de las teclas tienen tres o cuatro funciones diferentes. Por ejemplo, la tecla "L" producirá "l" o "L", según se pulse o no la tecla de cambio (Shift); pero si se utiliza junto con la tecla de función (Function), la tecla "L" generará la instrucción LIST del BASIC. Pulsando la tecla de control (Control) al mismo tiempo que "L" se limpiará la pantalla. Quizá hubiese sido preferible que Acorn hubiera conservado las 10 teclas de función rojas que se hallan en el BBC Micro, porque además se pueden programar para realizar diversas tareas útiles. El Electron posee teclas de función, pero éstas son sólo las teclas numéricas que se pulsan conjuntamente con la tecla Function.

El único aspecto en el que las dos máquinas difieren de forma notable es en la gama de interfaces proporcionadas. El BBC Micro tiene más interfaces que ningún otro ordenador personal, lo que significa que se le puede acoplar fácilmente un sinnúmero de periféricos, desde impresoras, modems y unidades de disco hasta segundos procesadores y

Juego rápido
El Snapper, de Acornsoft, es una buena versión del clásico juego Pac-Man. Se vende en formato de cartucho y de cassette y se puede controlar mediante teclas o con una palanca de mando

Conector para ampliación
El Plus 1 se conecta con el Electron a través de este conector marginal. Éste es el único medio de ampliación de que dispone el Electron

brazos-robot. Todos éstos requieren chips y conectores adicionales, lo que inevitablemente aumenta los costos de fabricación. Omitiendo estas interfaces, no sólo ya no son necesarios chips extras, sino que los requerimientos de potencia son inferiores y el ordenador es más pequeño y más económico. Ésa fue precisamente la filosofía que inspiró la creación del Electron original. El hecho de que Acorn haya producido la Plus 1 sugiere que quizás este recorte del costo haya ido demasiado lejos.

El Electron Plus 1 es menos versátil que el BBC Micro y ofrece menos posibilidades de ampliación. Pero muchos usuarios no necesitan tales facilidades y, a un precio considerablemente inferior que la máquina BBC, el Electron Plus 1, con sus excelentes gráficos y facilidades de sonido, su BASIC estructurado y una creciente gama de software, parece ser una inversión rentable.

BBC MICRO MODELO B

DIMENSIONES
75 x 340 x 410 mm

CPU
6502, 1,8 MHz

MEMORIA
32 K de RAM, 32 K de ROM

PANTALLA
Ocho modalidades de visualización. Resolución más alta: texto, 80 x 32 caracteres; gráficos, 640 x 256 pixels. Hasta ocho colores que pueden ser fijos e intermitentes. Modalidad de visualización en teletexto. Caracteres definibles por el usuario

INTERFACES
UHF para televisión; RGB y video compuesto para pantallas; puerta para cassette; interfaces para impresoras RS423 y Centronics; puerta analógica (para palanca de mando, etc.); conectores de ROM (para software); capacidad para segundos procesadores; bus de 1 MHz; interface Disk (opcional); interface Econet para conexión en red (opcional); puerta para el usuario; salidas de corriente auxiliares (para unidades de disco, etc.)

LENGUAJES DISPONIBLES
BASIC BBC (incluido), Assembler 6502 (incluido), LISP, FORTH, BCPL, PASCAL

TECLADO
72 teclas tipo máquina de escribir. Incluye 10 teclas de función programables

DOCUMENTACION
El manual del BBC es una guía excelente para el programador experimentado, pero es de poca ayuda para el usuario principiante

VENTAJAS
Una de las mejores versiones de BASIC existentes, una enorme gama de interfaces, buenos gráficos, sonido y teclado

DESVENTAJAS
Iniciarse con el BBC no es fácil para el principiante

ACORN ELECTRON PLUS 1

DIMENSIONES
65 x 260 x 340 mm (con la Plus 1 instalada)

CPU
6502, 1,8 MHz

MEMORIA
32 K de RAM, 32 K de ROM

PANTALLA
Siete modalidades de visualización. Resolución más alta: texto, 80 x 32 caracteres; gráficos, 640 x 256 pixels. Hasta ocho colores y ocho intermitentes. Caracteres definibles por el usuario

INTERFACES
Salida UHF para televisión; RGB y video compuesto para pantallas; puerta para cassette; puerta para impresora en paralelo; puerta analógica (para palancas de mando, etc.); dos conectores (para software en cartucho de ROM, etc.)

LENGUAJES DISPONIBLES
BASIC, Assembler 6502 (suministrado); FORTH, LISP (también en cartucho de ROM), S-PASCAL

TECLADO
56 teclas tipo máquina de escribir. Tecla de función que permite la entrada de instrucciones de BASIC con una única tecla. Diez teclas programables

DOCUMENTACION
La Guía para el Usuario está bien realizada y es de fácil lectura. Posee una cobertura muy profunda del BASIC Electron y del lenguaje Assembly 6502

VENTAJAS
Los gráficos son buenos; producen una imagen clara. Teclado de gran calidad. Una buena versión de BASIC

DESVENTAJAS
Las teclas multifunción pueden ser fuente de confusión. Sólo dispone de un canal de sonido. Poca memoria disponible. Carece de modalidad de teletexto.

Chris Stevens

Revolución cubista

Analizamos hoy un programa que nos permite dibujar formas tridimensionales y hacerlas rotar

El programa que proporcionamos utiliza algunos principios geométricos básicos para crear proyecciones en perspectiva de objetos que se pueden contemplar desde cualquier ángulo o distancia. Todos los datos para los objetos están almacenados en sentencias DATA dentro del programa. Éstos comprenden coordenadas tridimensionales para cada punto final de una línea del objeto. Para cada punto hay, asimismo, una cifra que indica si está al comienzo de una línea o en su final, que nos informa si la línea se ha de trazar desde o hacia dicho punto.

La conversión de estas coordenadas tridimensionales en valores bidimensionales que representen puntos en la pantalla es una cuestión simple, aunque extensa, y puramente matemática. Las coordenadas *x* e *y* de cada punto (en el plano de la pantalla de televisión) se dividen por un factor que representa la distancia del objeto en relación a nosotros. Además, la coordenada resultante se escala en función de un factor apropiado para el sistema de coordenadas del micro. Cambiando el factor distancia, se puede lograr que el objeto se encoja o agrande a medida que se acerca o se aleja del observador.

Se puede utilizar una tercera constante para cambiar el efecto de la proyección en perspectiva. Incrementando el valor de esta constante, se exagera la vista en perspectiva del objeto, como si se estuviera contemplando con una lente de ojo de pez. La disminución de esta constante da el efecto de aplanar la vista, como si estuviéramos mirando a través de un teleobjetivo.

Además de crear una vista en perspectiva del objeto, el programa también permite hacerlo rotar de modo que se lo pueda contemplar desde cualquier ángulo. Ello se consigue mediante simple trigonometría. Los ejes giran el ángulo deseado de forma que cuando se efectúa la proyección en perspectiva parece que la imagen de la pantalla ha girado. Esto se puede hacer bien como una rotación a través del eje *y* (el punto de vista parece moverse alrededor del objeto) o bien alrededor del eje *x* (el punto de vista se desplaza por arriba o por debajo del objeto).

Aunque nuestro programa puede crear todos estos efectos, su operatoria es notablemente sencilla. El control del punto de vista se ejerce mediante las teclas numéricas del uno al ocho. Éstas proporcionarán respectivamente: un movimiento del punto de vista hacia la izquierda, derecha, arriba, abajo, hacia el objeto o alejándose del mismo; un aumento del efecto de perspectiva (ojo de pez), o una disminución del efecto de perspectiva (teleobjetivo).

Las coordenadas tridimensionales corrientes se almacenan en tres matrices: X, Y y Z. Los cambios en estas matrices y los cambios en las constantes utilizadas para la transformación de la perspectiva se llevan a cabo en una serie de subrutinas. Cada vez que se detecta una pulsación de tecla, se borra la imagen de la pantalla dibujándola en el color de fondo, se realiza el cambio deseado con una llamada a la subrutina apropiada y se vuelve a dibujar la nueva imagen.

La transformación de la perspectiva se realiza en la subrutina que dibuja la imagen. Ésta pasa por cada juego de coordenadas tridimensionales, las traduce a bidimensionales y las traza en la pantalla (desplazándose a los puntos o dibujando líneas de acuerdo al cuarto dato para cada punto).

Crearse sus propios datos de objetos para este programa es una cuestión bastante larga pero relativamente fácil. Los datos se almacenan en sentencias al final del programa, con cuatro valores para cada punto del objeto. El número total de puntos se establece en la primera línea del programa. Ésta habrá de modificarse para adaptarla a los datos propios de cada caso. Los datos que proporcionamos crean un cubo con una línea en diagonal a través de una cara.

Cada juego de cuatro valores está por este orden: valor trazar o dibujar, coordenada *x*, coordenada *y*, coordenada *z*. Los valores se calculan siguiendo mentalmente el boceto del objeto en tres dimensiones. Utilizando un lápiz imaginario, visite cada vértice del esquema del objeto uno a uno. Si su lápiz se desplaza hasta un punto sin dibujar una línea, se utiliza 4 para el primer valor. Un valor de 5 indica que se ha de trazar una línea hasta el punto desde el punto precedente. Se utilizan estos valores en particular porque hacen que el programa para el BBC Micro resulte menos complicado.

El origen de las coordenadas (0,0) está en el centro de la pantalla. Lo mejor es hacer que éste también sea el centro de su objeto. El eje *x* es el eje horizontal, con valores positivos yendo hacia arriba. El eje *z* es el eje hacia dentro y hacia fuera de la pantalla. La dirección positiva de este eje es hacia dentro de la pantalla.

Mantenga los valores de X, Y y Z tan pequeños como sea razonablemente posible. La preparación inicial del efecto de perspectiva y la distancia del punto de vista deben tener en cuenta que la anchura de un objeto de alrededor de 10 ocupará toda la pantalla. Esto se podría modificar alterando el valor de escalado utilizado en la transformación de la perspectiva. Le sugerimos que primero experimente con formas simples. Mantenga reducida la cantidad de puntos. Cuando haya conseguido dominar la digitalización de objetos sólidos simples (una pirámide, un cubo) puede atreverse con otros más complicados.

Nuestro programa se podría ampliar aún más para proporcionar efectos adicionales. Se podría incorporar una facilidad de traslación para desplazar el objeto, sin rotación, en relación al origen de la

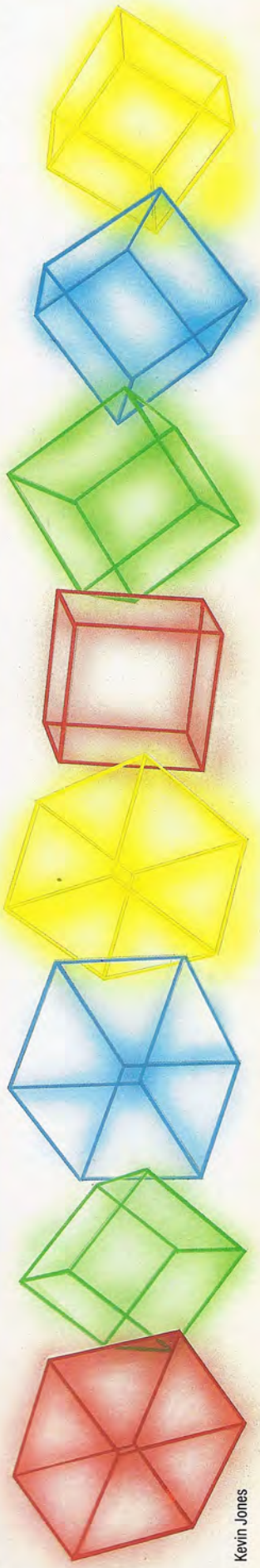
coordenada. Intente incorporar una rutina para eliminar las líneas y las porciones de líneas que quedarían ocultas a la vista. Con ello se lograría hacer que la actual imagen, como construida en alambre, tuviera un aspecto más realista. No obstante, la eliminación de estas líneas ocultas es un asunto enormemente complicado. Exige unas matemáticas muy complejas y haría que el programa resultara considerablemente más lento. Aun si no se le agrega nada al programa y se le deja tal como se ofrece aquí, se pueden conseguir algunos resultados atractivos y espectaculares.

Versión para el Spectrum

```
 10 LET  N=16
 20 DIM  P(50)
 21 DIM  X(50)
 22 DIM  Y(50)
 23 DIM  Z(50)
 24 DIM  A(50)
 25 DIM  B(50)
 40 LET  D=10: LET  P=0.5
 50 LET  SI=SIN  0.09:  LET  CO=COS  0.09
 60 FOR I=1  TO  N
 70 READ  P(I),X(I),Y(I),Z(I)
 80 NEXT  I
 90 :
200 INVERSE  0:  GO  SUB  300
210 IF  INKEY$<>""  THEN  GO  TO  210
211 IF  INKEY$=""  THEN  GO  TO  211
212 LET  I$=INKEY$
230 INVERSE  1:  GO  SUB  300
240 IF  I$="1"  THEN  GO  SUB  1000
241 IF  I$="2"  THEN  GO  SUB  2000
242 IF  I$="3"  THEN  GO  SUB  3000
243 IF  I$="4"  THEN  GO  SUB  4000
244 IF  I$="5"  THEN  GO  SUB  5000
245 IF  I$="6"  THEN  GO  SUB  6000
246 IF  I$="7"  THEN  GO  SUB  7000
247 IF  I$="8"  THEN  GO  SUB  8000
250 GO  TO  200
260 :
300 FOR  I=1  TO  N
310 LET  A(I)=X(I)*300/(P*Z(I)+D):  LET  B(I)=
    Y(I)*300/(P*Z(I)+D)
320 NEXT  I
330 FOR  I=1  TO  N
340 IF  P(I)=4  THEN  PLOT  A(I)+128,B(I)+85
345 IF  P(I)=5  THEN  DRAW  A(I)-A(I-1),B(I)-B(I-1)
350 NEXT  I
360 RETURN
370 :
1000 FOR  I=1  TO  N
1010 LET  X=X(I)*CO-Z(I)*SI
1020 LET  Z=Z(I)*CO+X(I)*SI
1030 LET  X(I)=X:  LET  Z(I)=Z
1040 NEXT  I
1050 RETURN
1060 :
2000 FOR  I=1  TO  N
2010 LET  X=X(I)*CO+Z(I)*SI
2020 LET  Z=Z(I)*CO-X(I)*SI
2030 LET  X(I)=X:  LET  Z(I)=Z
2040 NEXT  I
2050 RETURN
2060 :
3000 FOR  I=1  TO  N
3010 LET  Y=Y(I)*CO+Z(I)*SI
3020 LET  Z=Z(I)*CO-Y(I)*SI
3030 LET  Y(I)=Y:  LET  Z(I)=Z
3040 NEXT  I
3050 RETURN
3060 :
4000 FOR  I=1  TO  N
4010 LET  Y=Y(I)*CO-Z(I)*SI
4020 LET  Z=Z(I)*CO+Y(I)*SI
4030 LET  Y(I)=Y:  LET  Z(I)=Z
4040 NEXT  I
4050 RETURN
4060 :
5000 LET  D=D*0.9
5010 RETURN
5020 :
6000 LET  D=D/0.9
6010 RETURN
6020 :
7000 LET  P=P/0.9
7010 RETURN
7020 :
8000 LET  P=P*0.9
8010 RETURN
8020 :
9000 DATA  4,1,1,1,  5,1,1,-1,  5,-1,1,-1,  5,-1,1,1,
     5,1,1,1
9010 DATA  5,1,-1,1,  5,1,-1,-1,  5,-1,-1,-1,  5,-1,-1,1,
     5,1,-1,1
9020 DATA  4,1,-1,-1,  5,1,1,-1,  5,-1,-1,-1,
     5,-1,1,-1
9030 DATA  4,-1,1,1,5,-1,-1,1
```

Versión para el BBC

```
 10 N=16
 20 DIM  P(50),X(50),Y(50),Z(50),A(50),B(50)
 30 MODE0:VDU29,640;512;5
 40 D=10:P=0.5
 50 SI=SIN(0.09):CO=COS(0.09)
 60 FOR  I=1  TO  N
 70   READ  P(I),X(I),Y(I),Z(I)
 80   NEXT  I
 90 :
200 GCOL0,3:GOSUB  300
210 I$=GET$
220 V=VAL(I$)
230 GCOL0,0:GOSUB  300
240 ON  V  GOSUB  1000,2000,3000,4000,5000,6000,
    7000,8000  ELSE  250
250 GOTO  200
260 :
300 FOR  I=1  TO  N
310   A(I)=X(I)*1000/(P*Z(I)+D):B(I)=Y(I)*1000/(P*Z(I)+D)
320   NEXT  I
330 FOR  I=1  TO  N
340   PLOT  P(I),A(I),B(I)
350   NEXT  I
360 RETURN
370 :
1000 FOR  I=1  TO  N
1010   X=X(I)*CO-Z(I)*SI
1020   Z=Z(I)*CO+X(I)*SI
1030   X(I)=X:Z(I)=Z
1040   NEXT  I
1050 RETURN
1060 :
2000 FOR  I=1  TO  N
2010   X=X(I)*CO+Z(I)*SI
2020   Z=Z(I)*CO-X(I)*SI
2030   X(I)=X:Z(I)=Z
2040   NEXT  I
2050 RETURN
2060 :
3000 FOR  I=1  TO  N
3010   Y=Y(I)*CO+Z(I)*SI
3020   Z=Z(I)*CO-Y(I)*SI
3030   Y(I)=Y:Z(I)=Z
3040   NEXT  I
3050 RETURN
3060 :
4000 FOR  I=1  TO  N
4010   Y=Y(I)*CO-Z(I)*SI
4020   Z=Z(I)*CO+Y(I)*SI
4030   Y(I)=Y:Z(I)=Z
4040   NEXT  I
4050 RETURN
4060 :
5000 D=D*.9
5010 RETURN
5020 :
6000 D=D/.9
6010 RETURN
6020 :
7000 P=P/.9
7010 RETURN
7020 :
8000 P=P*.9
8010 RETURN
8020 :
10000 DATA  4,1,1,1,  5,1,1,-1,  5,-1,1,-1,  5,-1,1,1,
      5,1,1,1
10010 DATA  5,1,-1,1,  5,1,-1,-1,  5,-1,-1,-1,  5,-1,-1,1,
      5,1,-1,1
10020 DATA  4,1,-1,-1,  5,1,1,-1,  5,-1,-1,-1,  5,-1,1,-1
10030 DATA  4,-1,1,1,  5,-1,-1,1
```



Kevin Jones

Estructuras modulares

En este capítulo mostramos cómo mejorar su programación utilizando módulos de código como componentes básicos

Un módulo es un trozo de código que realiza una función determinada. Los puntos de entrada y salida de los módulos, llamados *interfaces*, se deben definir con precisión, y los procesos que se producen entre estas interfaces deben ser absolutamente independientes del resto del programa. Una vez que se ha escrito un módulo, se lo puede tratar como a una "caja negra". Los datos puden entrar y salir del módulo a través de sus interfaces, pero lo que sucede dentro del mismo se puede dejar en sus propias manos.

Los módulos se pueden unir entre sí para construir un programa sin que el programador tenga que preocuparse por la forma en que llevan a cabo sus tareas. Un programador se puede construir un "stock" de módulos para emplearlos cuando los necesite y los programadores pueden intercambiar módulos para usarlos en sus programas. Pero para sacar partido de la programación estructurada modular, al escribir los módulos hay que tomar nota respecto al flujo de control y al flujo de datos.

Para asegurar que todos los módulos se comportan del mismo modo en relación al flujo de control, se debe seguir una regla muy simple: todos los módulos deben tener un único punto de entrada y un único punto de salida. Lo que esto significa en la práctica es que se debe diseñar cuidadosamente el flujo de control dentro del módulo de manera que empiece en un lugar e, independientemente de los bucles y las bifurcaciones en que pueda incurrir, llegue a la misma salida por todas las rutas posibles.

Los módulos corresponden a los algoritmos que hemos estado analizando en capítulos anteriores del curso. Los lenguajes "estructurados", como el PASCAL, permiten que el programador cree rutinas a las que se puede llamar por su nombre y que utilizan sus propias variables. Los lenguajes de esta naturaleza estimulan al programador para que entre y salga de una rutina (denominada *procedimiento*) por puntos únicos de entrada y salida.

En BASIC, utilizando la combinación GOSUB...RETURN se puede llamar a una subrutina desde el programa principal y, después de que la misma se haya ejecutado, el control retornará a la línea que sigue inmediatamente a la de la instrucción GOSUB. No obstante, no existe restricción respecto a la línea de la cual GOSUB le envía el control. Dos instrucciones GOSUB pueden enviar el control a líneas diferentes de una subrutina con un único RETURN, y el resultado puede ser completamente distinto en cada caso. Del mismo modo, no hay ninguna restricción respecto a cuántas sentencias RETURN se pueden emplear en una subrutina.

Todo ello significa que el programador de BASIC debe ser autodisciplinado. Debe empezar por asegurarse de que todas las GOSUB hacia la misma subrutina señalen al mismo número de línea y que todas las subrutinas posean en su interior un solo RETURN. Lo mejor es acostumbrarse a que la primera línea de cada subrutina contenga una sentencia REM que le otorgue un título y utilizar esa línea como punto de entrada. La última línea de la subrutina ha de ser el RETURN. Esto no es esencial, pero ayuda a que las cosas queden más claras.

La regla GOTO

Se debe tener cuidado especial con la instrucción GOTO, que puede hacer estragos en la estructura del programa. En este caso la regla es: utilizar un GOTO sólo para enviar el control a una línea *dentro* de la misma subrutina. Con ello se evita el peligro potencial de saltarse un RETURN o pasarle el control a un RETURN equivocado. En algunas ocasiones es necesario retornar de una subrutina sin ejecutar todas sus líneas. En este caso debe ir hacia (GOTO) la línea que posee el RETURN y de esta forma no se planteará ningún problema.

Utilizar instrucciones GOTO dentro de bucles es aún más peligroso. Si el control salta hacia fuera de un bucle, ¡el BASIC no puede saberlo y supone que el resto del programa es el cuerpo de ese bucle! La regla de seguridad es: estando en el cuerpo de un bucle, no ir nunca hacia (GOTO) una línea que esté fuera del cuerpo de ese bucle. Si necesita salir de un bucle antes de su fin, ponga el contador del bucle o la variable de condición en el valor terminal y vaya hacia (GOTO) la línea de condición (la línea que contiene NEXT o WHILE). Al igual que con la sentencia RETURN, coloque NEXT o WHILE en una línea exclusiva a efectos de claridad. Seguir la pista de la estructura de un programa es muchísimo más fácil si se evitan los GOTO en la medida de lo posible.

Es en las bifurcaciones donde es más probable que el control se extravíe, de modo que trate de que ninguna decisión envíe el control hacia fuera de una subrutina a menos que sea con una llamada correcta a otra subrutina. Recuerde que cada subrutina posee un único punto de salida, de modo que asegúrese de que sea posible seguir el flujo de control a través de todas las bifurcaciones hasta ese punto. Esto es más fácil de controlar si se dibuja un diagrama de flujo para la rutina. Con frecuencia, utilizar un flag o indicador puede reducir la necesidad de instrucciones GOTO en aquellas rutinas que implican bucles y bifurcaciones.

Podemos pensar que los datos entran y salen de los módulos, tal como aplicamos esta idea en el caso de los algoritmos (véase p. 866). Para que los módulos se puedan utilizar de forma independiente los unos de los otros, deben ser diseñados de modo que la única influencia que ejerzan entre sí sea en función de los datos que se pasan entre ellos. El programa principal le pasa datos a un módulo y, después de que éste se ha ejecutado, se retorna el resultado, sea cual sea.

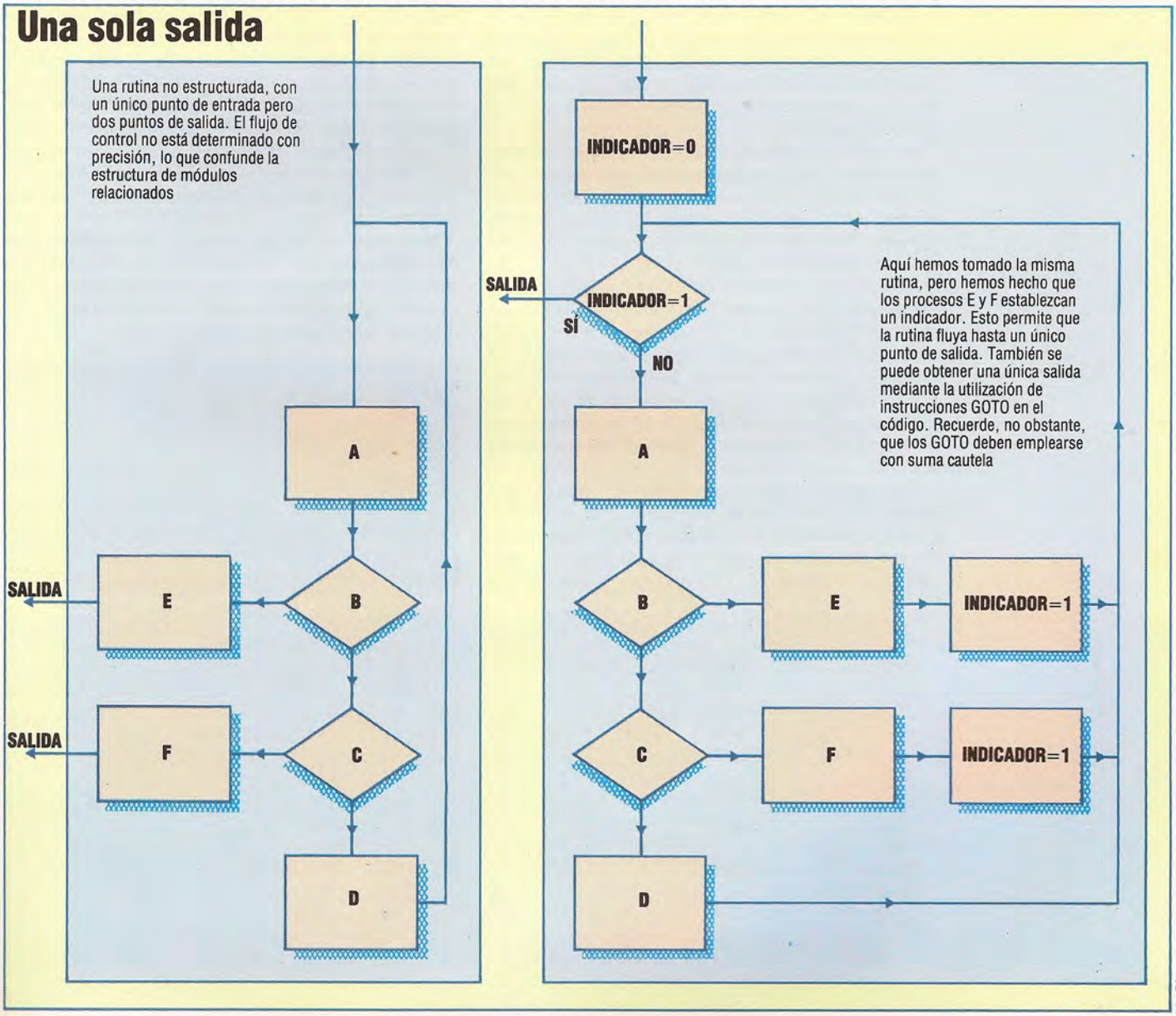
Los datos se mueven por el programa dentro de variables y la libertad de movimiento de una variable se denomina su *ámbito*. Muchos lenguajes de programación pueden limitar el ámbito de una variable a determinadas subrutinas. En PASCAL, las variables que se utilizan en una determinada subrutina (procedimiento) se deben "declarar" para ese procedimiento. Las variables declaradas para el programa principal son *globales* y se podrían emplear en cualquier otro lugar del programa (incluyendo dentro de cualquiera de sus módulos). Sin embargo, las variables declaradas dentro de un determinado procedimiento son *locales* de ese procedimiento y sólo se pueden utilizar allí.

Las variables locales pueden tener el mismo nombre que las globales y la utilización de una no afecta al valor de la otra. El uso de un lenguaje que admita variables locales nos permite escribir subrutinas sin que debamos preocuparnos sobre cómo las variables empleadas en la rutina pueden afectar a las variables de otras subrutinas. Por desgracia, son pocas las versiones de BASIC que admiten variables locales, lo que significa que si deseamos escribir subrutinas independientes debemos simular de alguna forma el efecto de tener variables locales.

La forma más sencilla de simular este efecto consiste en adoptar convenciones de nomenclatura para distinguir a las variables que realizan tareas diferentes. Existen algunas convenciones ya aceptadas y los programadores se sirven de ellas ampliamente. Utilizar I, J y K como contadores de bucles y valores de índice es muy común y es una práctica que se ha adoptado a partir de las matemáticas.

Después de haber descrito un programa mediante un diagrama de flujo, es una cuestión sencilla numerar las subrutinas implicadas o proporcionarles algún otro tipo de código. Las variables globales que es necesario hacer locales para una subrutina determinada se pueden entonces dotar de un subíndice basado en este código para hacerlas exclusivas. Por consiguiente, la rutina número 5 podría utilizar las variables SUMA5 y TOTAL5 para distinguirlas de SUMA12 y TOTAL12 de la rutina número 12. No obstante, ¡tenga cuidado de que el BASIC que esté utilizando no mire sólo los dos primeros caracteres! No es necesario codificar las variables que se emplean para pasar valores entre subrutinas y aquellas que se utilizan sólo en el programa principal.

Una sola salida



Liz Dixon

Jugando con dioses

"Valhalla" es un imaginativo juego de aventuras con constantes referencias a la mitología escandinava y espectaculares gráficos, escrito para el Spectrum y el Commodore 64

En la aventura *Valhalla* existen 81 escenarios, de los cuales 16 están en Asgard, incluyendo la evanescente Valhalla, el lugar soñado y buscado por el usuario. Midgard consta de 20 escenarios, y los 45 restantes componen colectivamente el Infierno. Cada escenario tiene ocho salidas, si bien algunas de ellas pueden estar bloqueadas o exigir la posesión de un objeto mágico para permitirle la entrada. Para complicar más las cosas, hay "caminos de anillos" que conectan escenarios alejados; si lleva un anillo adecuado puede "saltar" hasta el escenario más lejano.

Valhalla tiene un reparto de 36 personajes (en la pantalla habrá al menos tres en cualquier momento dado), que son buenos o malos. Interactúan continuamente, luchan entre ellos, se dan comida o vino los unos a los otros o le arrojan objetos a cualquiera que no les agrade.

El jugador puede interactuar con ellos en cualquier punto, entrando las acciones que desee a través del teclado. Por otra parte, puede también reclinarse en el asiento y contemplar el espectáculo, aunque los observadores silenciosos acabarán muertos invariablemente.

El juego no presenta un único camino hacia el éxito; el jugador puede tomar cualquiera de las muchas rutas posibles. Debe convencer a los personajes buenos de que es digno merecedor de su ayuda. Por lo tanto, si está de un humor pésimo, puede fácilmente optar por gastar todas sus energías comportándose de manera repelente para conquistarse las simpatías y el apoyo de los personajes malos.

Las visualizaciones de gráficos de *Valhalla* son particularmente buenas, en especial en la versión para el Commodore. Primero se dibuja en la pantalla una escenografía para cada escenario, en vivos colores primarios en el Spectrum y con suaves tonos pastel en el Commodore. A continuación aparecen los personajes: en el Spectrum éstos son estilizaciones con palitos y están dibujados en negro, mientras que en el Commodore, su mayor potencial de color y su mayor resolución permiten representar a los personajes con mayor detalle. Por último aparecen los objetos existentes en el escenario: éstos pueden incluir alimentos, vino, joyas, llaves y armas. Nuevamente, las representaciones del Spectrum son muchísimo menos realistas que las de la versión para el Commodore.

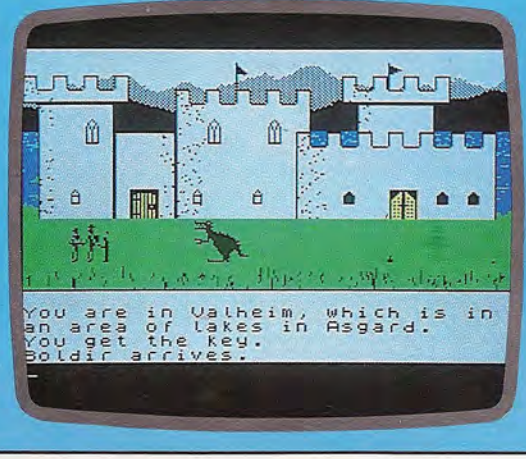
Los personajes interactúan, entonces, entre ellos y sus acciones se describen con palabras en la parte inferior de la pantalla. Y aquí es donde participa el jugador. La gama de acciones que puede emprender éste es amplia: quizá le guste tratar de convencer a uno de los dioses escandinavos de que se desprenda de parte de su tesoro, o puede intentar atacarlo con un arma para poner a prueba su fuerza. Sin embargo, al cabo de un rato deseará marcharse y explorar un nuevo territorio en busca de la mítica Valhalla.

Encontrar los objetos mágicos que finalmente le darán acceso a este Paraíso no es tarea fácil; coger algunos de ellos resulta increíblemente difícil, y la exasperación que esto puede provocar tal vez constituya una limitación del juego.

Valhalla: Para el Spectrum de 48 K y el Commodore 64
Editado por: Legend, Freepost, 1 Milton Road, Cambridge CB4 1UY
Autores: Graham Asher, Richard Edwards, Charles Goodwin, James Learmont, Jan Ostler, Andrew Owen, John Peel
Palancas de mando: No se necesitan
Formato: Cassette

Día y noche

Valhalla es uno de los varios juegos que se han producido recientemente cuyos gráficos indican el paso del tiempo mediante el oscurecimiento de la pantalla. Ello crea el efecto de "día" y "noche"



Liz Heaney

En el redondel

Para el Commodore 64 ya explicamos varias subrutinas que explotaban sus posibilidades en alta resolución. La presente sirve para el trazado de círculos

No es posible dibujar con toda precisión una circunferencia en un ordenador familiar. El grado de precisión depende del método seguido y de la longitud y complejidad de la rutina resultante. El trazado por medio del BASIC por lo general se sirve de las funciones seno y coseno o de raíces cuadradas que proporcionan las coordenadas de los puntos sobre los que pasa la circunferencia. Ambos métodos tienen sus dificultades al tratar de adaptarlos a código máquina. Esto es lo que nos lleva a proponer un método alternativo, especialmente adecuado para su resolución en lenguaje máquina.

El método del que hablamos considera el diámetro de la circunferencia divisible en numerosas partes iguales, cada una de las cuales mide W. Por cada división debemos pensar en una línea que alcanza verticalmente un punto, C, de la circunferencia. La ilustración adjunta muestra una de estas líneas tras N divisiones del extremo izquierdo del diámetro AB. Uniendo A y B con C obtenemos dos triángulos rectángulos ACD y BCD, según puede verse.

Empleando el teorema de Pitágoras, podemos escribir las siguientes igualdades referidas al diagrama:

$$AC^2 = AD^2 + CD^2$$
$$CB^2 = DB^2 + CD^2$$

Sumando ambas igualdades obtenemos esta otra igualdad:

$$AC^2 + CB^2 = AD^2 + DB^2 + 2CD^2$$

Sabemos también por geometría que el triángulo ABC es igualmente rectángulo. Podemos, pues, establecer:

$$AC^2 + CB^2 = AB^2$$

Con lo cual la expresión que escribimos anteriormente queda así:

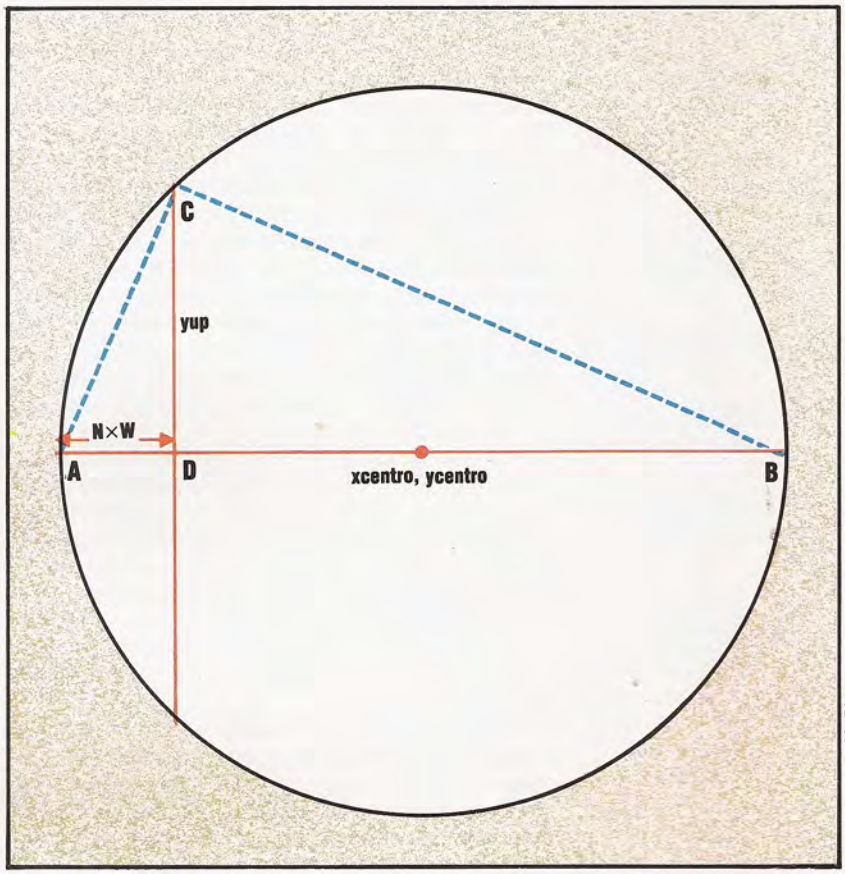
$$AB^2 = AD^2 + DB^2 + 2CD^2$$

A CD llamaremos "yup" (*y* "hacia arriba"), o sea, la distancia del punto del diámetro a la circunferencia. Pero AD vale N×W y AB mide 2×R (R es el radio). Sustituyendo en la ecuación anterior y despejando tendremos:

$$2yup^2 = (2R)^2 - (NW)^2 - (2R - NW)^2$$
$$yup^2 = 2RNW - (NW)^2$$

Si, por ejemplo, decidimos dividir el diámetro en 64 partes iguales, tendremos que W = 2×R/64, o lo que es lo mismo W = R/32. Volviendo a la ecuación:

$$yup^2 = 2RNR/32 - (NR)^2/32^2$$
$$yup^2=(64\ R^2N-R^2N^2)/32^2=R^2(64N-N^2)/32^2$$



Liz Dixon

Tomando raíz cuadrada en ambos miembros:

$$yup=\sqrt{R^2(64N-N^2)/32^2}=R/32\times\sqrt{64N-N^2}$$

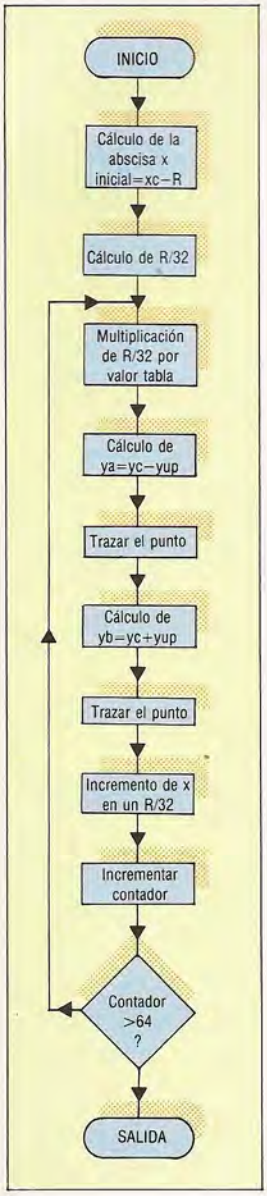
Si comenzamos por x = xcentro − R e incrementamos en 64 pasos iguales, entonces la distancia vertical hasta la circunferencia se obtiene con esa fórmula que acabamos de deducir, siendo N el número del incremento. Y aunque este resultado incluya una raíz cuadrada, la expresión a la que afecta la raíz no depende de las coordenadas del centro ni del radio. Podemos, pues, calcular una tabla de valores para la función raíz que nos proporcione una solución para cada uno de los valores de N, desde el 0 al 64. Esto se hace sólo una vez y se incorpora en el programa como tabla de consulta.

La ordenada absoluta *y* para cada incremento de *x* será:

$$ya = ycentro - yup$$

También podemos servirnos de la simetría de la circunferencia para calcular la ordenada *y* correspondiente a la otra mitad:

$$yb = ycentro + yup$$



Construcción de la rutina

Estamos en disposición de trazar las líneas maestras de nuestra rutina. El diagrama de flujo al margen muestra la manera de calcular cada punto de la circunferencia. Vemos que nuestra rutina, al necesitar tan sólo de una multiplicación, puede trazar cada punto con relativa rapidez. Pero hay dos ligeros inconvenientes. El primero es que no nos dará un círculo continuo sino una serie de puntos que componen una circunferencia. Y el segundo reside en que si bien esta técnica ofrece círculos bien definidos en BASIC, cuando se ejecuta en lenguaje máquina surgen imprecisiones.

El primer problema se resuelve utilizando la Linesub (subrutina de la línea; véase p. 899) para unir los puntos con pequeños segmentos y obtener así un círculo continuo. El segundo problema es debido a las inexactitudes en el cálculo de la raíz cuadrada que se ofrece en forma de tabla de consulta. El hecho de calcular los valores en BASIC y colocarlos (POKE) en una serie de bytes prefijados para la tabla significa que en realidad sólo queda almacenada la parte entera de cada valor. Si deseamos círculos mejor definidos debemos mejorar la precisión de los valores almacenados de la tabla. El valor máximo a almacenar es el 32; podemos, pues, multiplicar cada número por ocho antes de almacenarlo, sin dejar de utilizar un byte por cada entrada de la tabla.

El número 32 es el único que puede, multiplicado por ocho, ser guardado en un único byte. Para simplificar la rutina habremos de aproximar este valor. Así, nuestra rutina en código máquina puede dividir por ocho el valor de la tabla ejecutando tres LSR (*Logical Shifts Right*: desplazamientos lógicos a la derecha). Y lo que es aún más importante, el resto puede quedar guardado después de la división para ser utilizado en posteriores cálculos.

El listado del código fuente reserva 65 bytes al comienzo del programa para almacenar la tabla, etiquetando el primero de ellos. Las siguientes entradas de la tabla son así accesibles por medio del direccionamiento indexado. Este listado puede introducirse y ensamblarse en la memoria de la manera habitual. Aunque antes de guardar (SAVE) el código ensamblado debe introducirse y ejecutarse el siguiente programa en BASIC. Esto no dañará al código objeto, dado que se encuentra ubicado en la parte alta de la memoria. El programa calcula los 65 valores que necesita nuestra rutina Circsub (*sub*rutina *Circ*ular), multiplica además estos valores por ocho y coloca (POKE) el resultado dentro del área de memoria reservada en el programa de código máquina.

Una vez ejecutado el programa de creación de la tabla, el código máquina puede guardarse (SAVE) del modo habitual, pero asegurando que se guarda (SAVE) la tabla junto con el código objeto. La tabla comienza en $C500. Una vez hecho esto, la tabla de consulta se cargará automáticamente siempre que carguemos Circsub.

```
 5 REM *** CREACION TABLA CIRCSUB ****
10 FOR N=0 TO 64
20 X=SQR (64*N-N↑2)*8
25 X %=X: DF=X-X %
27 IFRE >=.5 THEN X %=X %+1
28 IF X %>255 THEN X %=255
29 POKE 50432+N,X %
30 NEXT
```

Este programa que ahora sigue muestra la manera como puede usarse desde un programa BASIC nuestra rutina Circsub. Sólo es necesario especificar las coordenadas del centro y el radio. En la línea 2000 la subrutina parte del valor de *x* en dos, según la forma byte *lo* byte *hi*; después coloca (POKE) en Circsub los valores especificados y hace la oportuna llamada (SYS). Observe que Circsub se vale de Linesub y esta subrutina a su vez de Plotsub (véase p. 817), por lo que las tres subrutinas han de cargarse al inicio del programa. Este programa dibuja círculos por la pantalla con radios cada vez mayores.

```
3 REM****PROGRAMA PRUEBA DE CIRCSUB****
5 DN=8:REM FOR CASSETTE DN=1
10 IFA=0THENA=1:LOAD"PLOTSUB.HEX",DN,1
15 IFA=1THENA=2:LOAD"LINESUB.HEX",DN,1
20 IFA=2THENA=3:LOAD"CIRCSUB.HEX",DN,1
100 GOSUB1000
110 YC=100:R=2
120 FORXC=30TO210STEP7
130 R=R+3
140 GOSUB2000
150 NEXT
160 GETA$:IFA$=""THEN160
170 GOSUB3000
180 END
1000 REM **** ACTIVACION ALT. RES. ****
1010 POKE49408,1:POKE49409,1
1020 POKE49410,3
1030 SYS49422
1040 RETURN
1050 :
2000 REM **** ENTRADA CIRCSUB ****
2010 CHI=INT(XC/256):CLO=XC-256*CHI
2020 POKE50497,CLO:POKE50498,CHI
2030 POKE50499,YC
2040 POKE50500,R
2050 SYS50521
2060 RETURN
2070 :
3000 REM **** BORRADO ALT. RES. ****
3005 RESTORE
3007 PRINTCHR$(147):REM CLEAR SCREEN
3008 PRINTTAB(10)"CIRCSUB VARIABLES"
3009 PRINT
3010 POKE49408,0:SYS49422
3030 FORI=50497TO50497+23STEP2
3035 READA$
3040 PRINTTAB(2);A$,PEEK(I),
3045 READA$
3047 PRINTA$,PEEK(I+1)
3050 NEXT
3060 RETURN
3070 :
5000 DATAXCTRLO,XCTRHI,YCTR,RADIUS,INTR
5010 DATAREMR,TOTX,RESREM,RESLO,RESHI,OLDXLO
5020 DATAOLDXHI,NEWXLO,NEWXHI,OLDYA
5030 DATAOLDYB,NEWYA,NEWYB,XFLAG,YFLAG,OLDY,NEWY
5040 DATAINTTAB,REMTAB
```

Como ocurre con las otras subrutinas en alta resolución para el Commodore 64, el código máquina puede ser introducido también por medio de sentencias DATA si usted carece de un ensamblador. El listado inferior ha de entrarse por teclado y ejecutarse para poder cargar (LOAD) Circsub en memoria. Pero note que los cargadores en BASIC que afectan a Linesub y Circsub deben ser igualmente cargados y ejecutados antes que el programa de muestra. Una vez las tres rutinas en memoria, hay que eliminar las líneas 10, 15 y 20. También verá que el cargador en BASIC de Circsub ya incluye los *data* de la tabla de consulta, por lo cual no es necesario ejecutar en este caso el programa "creación tabla".

Muy pronto

Esta sección de lenguaje máquina ha analizado hasta el momento aquellos ordenadores provistos de un microprocesador Z80 o bien 6502. Nos queda un tercer microprocesador, igualmente popular, el chip 6809, que emplean el Dragon y el Tandy Color. Pronto nuestro curso centrará su atención en este chip para enseñar a programar en lenguaje máquina también a los usuarios del 6809

Programa de carga en BASIC

```
10 REM**** CARGADOR EN BASIC DE CIRCSUB ****
20 FORI=50432TO50432+498
30 READA:POKEI,A
40 CC=CC+A
50 NEXT I
60 READA:IFA<>CCTHENPRINT"ERROR EN SUMA DE CONTROL"
100 DATA0,63,89,108,123,137,149,159
110 DATA169,177,185,193,199,205,211
120 DATA216,221,226,230,233,237,240
130 DATA243,245,247,249,251,252,253
140 DATA254,255,255,255,255,255,254
150 DATA253,252,251,249,247,245,243
160 DATA240,237,233,230,226,221,216
170 DATA211,205,199,193,185,177,169
180 DATA159,149,137,123,108,89,63,0
190 DATA205,0,100,80,2,16,16,0,0,0,29
200 DATA1,31,1,100,100,100,100,0,0,120
210 DATA100,0,0,72,138,72,152,72,173
220 DATA68,197,41,31,141,70,197,173,68
230 DATA197,160,5,74,136,208,252,141
240 DATA69,197,173,65,197,56,237,68
250 DATA197,141,77,197,141,75,197,173
260 DATA66,197,233,0,141,78,197,141,76
270 DATA197,173,67,197,141,79,197,141
280 DATA80,197,169,0,170,141,71,197
290 DATA189,0,197,160,3,74,136,208,252
300 DATA141,87,197,189,0,197,41,7,141
310 DATA88,197,172,68,197,169,0,141,72
320 DATA197,141,73,197,141,74,197,173
330 DATA72,197,24,109,88,197,141,72
340 DATA197,201,8,144,23,56,233,8,141
350 DATA72,197,173,73,197,24,105,1,141
360 DATA73,197,173,74,197,105,0,141,74
370 DATA197,173,73,197,24,109,87,197
380 DATA141,73,197,173,74,197,105,0
390 DATA141,74,197,136,208,198,160,5
400 DATA78,74,197,110,73,197,110,72
410 DATA197,136,208,244,173,72,197,201
420 DATA16,144,9,173,73,197,24,105,1
430 DATA141,73,197,173,67,197,56,237
440 DATA73,197,141,81,197,173,81,197
450 DATA141,86,197,173,79,197,141,85
460 DATA197,32,165,198,173,67,197,24
470 DATA109,73,197,141,82,197,173,82
480 DATA197,141,86,197,173,80,197,141
490 DATA85,197,32,165,198,173,77,197
500 DATA141,75,197,173,78,197,141,76
510 DATA197,173,81,197,141,79,197,173
520 DATA82,197,141,80,197,173,71,197
530 DATA24,109,70,197,141,71,197,201
540 DATA32,144,26,173,71,197,56,233,32
550 DATA141,71,197,173,77,197,24,105,1
560 DATA141,77,197,173,76,197,105,0
570 DATA141,78,197,173,77,197,24,109
580 DATA69,197,141,77,197,173,78,197
590 DATA105,0,141,78,197,232,224,65
600 DATA240,3,76,153,197,104,168,104
610 DATA170,104,96,169,0,141,83,197
620 DATA141,84,197,173,75,197,205,77
630 DATA197,208,3,238,83,197,173,85
640 DATA197,205,86,197,208,3,238,84
650 DATA197,173,83,197,45,84,197,208
660 DATA39,173,75,197,141,0,195,173,76
670 DATA197,141,1,195,173,85,197,141,4
680 DATA195,173,77,197,141,2,195,173
690 DATA78,197,141,3,195,173,86,197
700 DATA141,5,195,32,14,195,96
710 DATA67223:REM*ERROR EN SUMA DE CONTROL*
```

Programa Circsub

```
;  ++++++++++++++
;  ++++++++++++++
;++                ==
;++    CIRCSUB 64   ++
;++                 ++
;  ++++++++++++++
;  ++++++++++++++
;++++ VARIABLES LINESUB ++++
;
LINSUB   = $C30E
X1LO     = $C300
X1HI     = $C301
X2LO     = $C302
X2HI     = $C303
Y1       = $C304
Y2       = $C305
*        = $C500
;
;++++ VARIABLES CIRCSUB ++++
;
TABLE    *=*+65          ;TABLA DE CONSULTA
XCTRLO   *=*+1
XCTRHI   *=*+1
YCTR     *=*+1
RADIUS   *=*+1
INTR     *=*+1
REMR     *=*+1
TOTX     *=*+1
RESREM   *=*+1
RESLO    *=*+1
RESHI    *=*+1
OLDXLO   *=*+1
OLDXHI   *=*+1
NEWXLO   *=*+1
NEWXHI   *=*+1
OLDYA    *=*+1
OLDYB    *=*+1
NEWYA    *=*+1
NEWYB    *=*+1
XFLAG    *=*+1
YFLAG    *=*+1
OLDY     *=*+1
NEWY     *=*+1
INTTAB   *=*+1
REMTAB   *=*+1
;
;++++ LLEVA REGISTROS A LA PILA ++++
;
         PHA
         TXA
         PHA
         TYA
         PHA
;
;++++ CALC INT(R/32) Y RESTO ++++
;
AD 44 C5 LDA  RADIUS
29 1F    AND  #$1F
8D 46 C5 STA  REMR
AD 44 C5 LDA  RADIUS
A0 05    LDY  #$05
4A       LOOP1 LSR  A
88       DEY
D0 FC    BNE  LOOP1
8D 45 C5 STA  INTR
;
;++++ CALC. ABSCISA X INICIAL ++++
;
AD 41 C5 LDA  XCTRLO
38       SEC
ED 44 C5 SBC  RADIUS
8D 4D C5 STA  NEWXLO
8D 4B C5 STA  OLDXLO
AD 42 C5 LDA  XCTRHI
E9 00    SBC  #$00
8D 4E C5 STA  NEWXHI
8D 4C C5 STA  OLDXHI
;
;++++ ESTABLECE VALORES INICIALES OLD Y ++++
;
AD 43 C5 LDA  YCTR
8D 4F C5 STA  OLDYA
8D 50 C5 STA  OLDYB
;
;++++ CONTADOR A ZERO Y TOTX ++++
;
A9 00    LDA  #$00
AA       TAX
8D 47 C5 STA  TOTX
;
;++++ CALC. INTTAB Y REMTAB ++++
;
NEXTPT
BD 00 C5 LDA  TABLE,X
A0 03    LDY  #$03
LOOP2
4A       LSR  A
88       DEY
D0 FC    BNE  LOOP2
8D 57 C5 STA  INTTAB
;
BD 00 C5 LDA  TABLE,X
29 07    AND  #$07
8D 58 C5 STA  REMTAB
;
;++++ MULTIPL. TABLA POR RADIO ++++
;
AC 44 C5 LDY  RADIUS
A9 00    LDA  #$00
8D 48 C5 STA  RESREM
8D 49 C5 STA  RESLO      ;RESPUESTA 0
8D 4A C5 STA  RESHI
;
AD 48 C5 AGAIN LDA  RESREM
18       CLC
6D 58 C5 ADC  REMTAB
8D 48 C5 STA  RESREM
C9 08    CMP  #$08       ;>=87
90 17    BCC  NOCARR
38       SEC
E9 08    SBC  #$08       ;REINIC. REM
8D 48 C5 STA  RESREM
AD 49 C5 LDA  RESLO
18       CLC
69 01    ADC  #$01       ;INC RESULT.
8D 49 C5 STA  RESLO
AD 4A C5 LDA  RESHI
69 00    ADC  #$00
8D 4A C5 STA  RESHI
NOCARR
AD 49 C5 LDA  RESLO
18       CLC
6D 57 C5 ADC  INTTAB
8D 49 C5 STA  RESLO
AD 4A C5 LDA  RESHI
69 00    ADC  #$00
8D 4A C5 STA  RESHI
88       DEY
D0 C6    BNE  AGAIN
;
;++++ DIVIDE RESULTADO POR 32 ++++
;
A0 05    LDY  #$05
LOOP3
4E 4A C5 LSR  RESHI
6E 49 C5 ROR  RESLO
6E 48 C5 ROR  RESREM
88       DEY
D0 F4    BNE  LOOP3
;
AD 48 C5 LDA  RESREM
C9 10    CMP  #$10       ; REM>=16?
90 09    BCC  NOCRRY
AD 49 C5 LDA  RESLO
18       CLC
69 01    ADC  #$01       ; SUMAR 1
8D 49 C5 STA  RESLO
;
;++++ CALC. PRIMERA ORDENADA Y ++++
;
AD 43 C5 LDA  YCTR
38       SEC
ED 49 C5 SBC  RESLO
8D 51 C5 STA  NEWYA
;
;++++ DIBUJA LÍNEA ++++
;
AD 51 C5 LDA  NEWYA
8D 56 C5 STA  NEWY
AD 4F C5 LDA  OLDYA
8D 55 C5 STA  OLDY
20 A5 C6 JSR  DRAW
;
;++++ CALC. 2.ª ORDENADA Y ++++
;
AD 43 C5 LDA  YCTR
18       CLC
6D 49 C5 ADC  RESLO
8D 52 C5 STA  NEWYB
;
;++++ DIBUJA LÍNEA ++++
;
AD 52 C5 LDA  NEWYB
8D 56 C5 STA  NEWY
AD 50 C5 LDA  OLDYB
8D 55 C5 STA  OLDY
20 A5 C6 JSR  DRAW
;
;++++ CAMBIA OLD POR NEW ++++
;
AD 4D C5 LDA  NEWXLO
8D 4B C5 STA  OLDXLO
AD 4E C5 LDA  NEWXHI
8D 4C C5 STA  OLDXHI
AD 51 C5 LDA  NEWYA
8D 4F C5 STA  OLDYA
AD 52 C5 LDA  NEWYB
8D 50 C5 STA  OLDYB
;
;++++ ALTERA ABSCISA X ++++
;
AD 47 C5 LDA  TOTX
18       CLC
6D 46 C5 ADC  REMR
8D 47 C5 STA  TOTX       ;SUM. RESTO A TOTAL
C9 20    CMP  #$20       ;>32?
90 1A    BCC  NOINC
AD 47 C5 LDA  TOTX
38       SEC
E9 20    SBC  #$20       ;REINIC. TOTX
8D 47 C5 STA  TOTX
AD 4D C5 LDA  NEWXLO
18       CLC
69 01    ADC  #$01       ;INC COORD X
8D 4D C5 STA  NEWXLO
AD 40 C5 LDA  OLDXHI
69 00    ADC  #$00
8D 4E C5 STA  NEWXHI
NOINC
AD 4D C5 LDA  NEWXLO
18       CLC
6D 45 C5 ADC  INTR
8D 4D C5 STA  NEWXLO
AD 4E C5 LDA  NEWXHI
69 00    ADC  #$00
8D 4E C5 STA  NEWXHI
;
;++++ INCREMENT COUNTER ++++
;
E8       INX
E0 41    CPX  #$41
F0 03    BEQ  FINISH
4C 99 C5 JMP  NEXTPT
;
;++++ SACA REGISTROS DE LA PILA ++++
;
FINISH
68       PLA
A8       TAY
68       PLA
AA       TAX
68       PLA
60       RTS
;
;++++ BUSCA EL MISMO PUNTO ++++
;
DRAW
A9 00    LDA  #$00
8D 53 C5 STA  XFLAG      ;FLAGS A 0
8D 54 C5 STA  XFLAG
;
AD 4B C5 LDA  OLDXLO
CD 4D C5 CMP  NEWXLO
D0 03    BNE  NOXFLG
NOXFLG
AD 55 C5 LDA  OLDY
CD 56 C5 CMP  NEWY
D0 03    BNE  NOYFLG
EE 54 C5 INC  YFLAG
NOYFLG
AD 53 C5 LDA  XFLAG
2D 54 C5 AND  YFLAG
D0 27    BNE  NODRAW
;
;++++ DIBUJA LÍNEA ++++
;
AD 4B C5 LDA  OLDXLO
8D 00 C3 STA  X1LO
AD 4C C5 LDA  OLDXHI
8D 01 C3 STA  X1HI
AD 55 C5 LDA  OLDY
8D 04 C3 STA  Y1
AD 4D C5 LDA  NEWXLO
8D 02 C3 STA  X2LO
AD 4E C5 LDA  NEWXHI
8D 03 C3 STA  X2HI
AD 56 C5 LDA  NEWY
8D 05 C3 STA  Y2
;
20 0E C3 JSR  LINSUB
;
NODRAW
60       RTS
```

El sonido del éxito

Audiogenic se ha creado un prestigio como una de las principales firmas proveedoras de software de las máquinas Commodore

Nuevas oficinas
Las oficinas centrales de la empresa están situadas en Sutton Park, en las afueras de Reading. Audiogenic se trasladó allí en abril de 1984, abandonando una finca ubicada en el centro de Reading que ya no disponía de suficiente espacio de almacenamiento

El director gerente y fundador de Audiogenic, Martin Maynard, define la empresa a su cargo como un "negocio de comercialización y manufactura". Con anterioridad Maynard trabajaba en la industria de la música, y estableció Audiogenic en Reading (Gran Bretaña) a principios de la década de los setenta como un estudio de grabación y reproducción de cintas de audio. En 1978, Southern Electricity Board encargó a Audiogenic la duplicación de cassettes de datos para ordenador. El equipo con el que contaba la empresa se modificó para hacer frente a las demandas que implica la producción de cintas para ordenador a granel, y Audiogenic firmó un contrato con Commodore para ocuparse de la duplicación del software para el microordenador PET.

La empresa asumió entonces la comercialización y distribución del catálogo de cassettes de Commodore y empezó a vender libros, revistas y otros equipos relacionados con Commodore. Después del lanzamiento del Vic-20, en 1981, Maynard obtuvo licencias para comercializar los productos Vic desarrollados por firmas de software norteamericanas. El software fabricado bajo licencia supone todavía el 80 % del catálogo de Audiogenic y representa el 85-90 % del movimiento total de la empresa.

Este énfasis en la comercialización de software, en vez de la producción casera, parece haber tenido mucho éxito y Maynard calcula que hasta ahora Audiogenic ha producido más de un millón de cassettes. "Nuestro punto más fuerte es que tenemos un catálogo enorme. Esta diversidad significa que podemos comprender lo que va sucediendo en el mercado", explica Maynard. "Habiendo estado en la industria del software durante seis años, ya lo hemos visto todo. De todo el software que se distribuyó el año pasado, entre el 20 y el 30 % todavía está en las estanterías de alguien. Los escritores están perdiendo el contacto con lo que realmente desea la gente."

Este cauteloso enfoque, promocionando software ya probado, contrasta firmemente con las tácticas improvisadas que aplican muchas otras firmas de software.

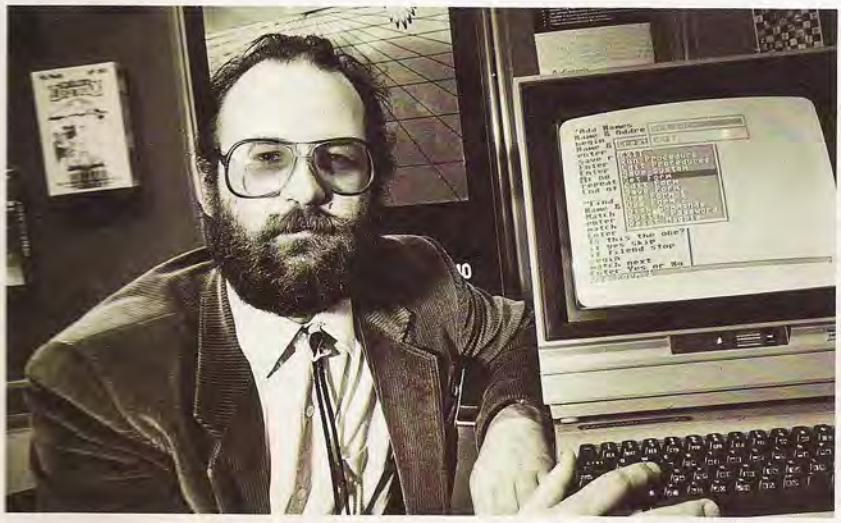
En la actualidad Audiogenic tiene una plantilla de 25 personas. El principal programador de la empresa, Dave Middleton (escritor del paquete de base de datos Magpie, tan bien conceptuado) trabaja en calidad de colaborador independiente. Audiogenic tiende a concentrarse en el software de utilidades para sus propios programas, en vez de perseguir los rápidos beneficios que se suelen obtener en el mercado de juegos. Como explica Maynard, "los ordenadores siempre tendrán un elemento de juegos, pero esa fase ahora está muriendo y el software para ordenador se irá desarrollando hasta convertirse en algo más útil. Cuando estás vendiendo doscientos o trescientos ejemplares de un paquete por mes, piensas que no se está vendiendo bien, pero al cabo de un año todavía se estarán vendiendo en esa misma cantidad".

Esto no significa que la empresa rechace de plano el mercado de juegos. Uno de los mayores éxitos de ventas de Audiogenic fue *Motor mania* y recientemente la empresa ha lanzado *Alice in Videoland* (Alicia en el País del Video) para el Commodore 64. Desarrollado bajo licencia, este juego consta de un impresionante programa de 90 Kbytes distribuido en cinco pantallas que se cargan desde disco a medida que va avanzando el juego.

Al poco tiempo de haberse trasladado a unas instalaciones más grandes, en marzo de 1984, Audiogenic instaló equipos de reproducción de cintas modernizados. La nueva maquinaria reproduce un programa repetidamente en una cinta continua, que se corta y se empaqueta en forma de cassette después de la duplicación. Este sistema es más rápido y también más conveniente que el sistema antiguo, que reproducía los programas en cassettes individuales e implicaba que la empresa necesitaba tener almacenadas grandes existencias de cassettes C10 y C20 vírgenes.

Audiogenic pretende continuar su política de distribuir productos fabricados por otras empresas en Gran Bretaña y Estados Unidos. La diversificación de hardware de periféricos ha llevado a la empresa a comercializar una pastilla para gráficos que desarrolló Koala Technologies. La planificación de proyectos de software incluye una gama de cassettes para máquinas MSX y software para el nuevo ordenador personal Commodore 16.

Brillante gestión
El director gerente de Audiogenic, Martin Maynard, que fundó la empresa a principios de la década de los setenta como un estudio de grabación



¿Está usted siendo espiado?

Steve Cross

Espionaje en el despacho
Existen muchísimos lugares donde se pueden ocultar dispositivos de escucha, desde los más obvios hasta los más insólitos. En este despacho de ejecutivo, los siguientes lugares podrían estar ocultando "micrófonos":
1) El teléfono. El micrófono podría estar en la bocina o el cuerpo del teléfono, o bien en la lámpara, que está muy cerca.
2) Planta en maceta. En la tierra, debajo de la maceta, ¡e incluso simulando un insecto!
3) La pared. Se sabe que se han empotrado micrófonos en la pared a modo de "clavos" o detrás de paneles de corcho.
4) Cuadro colgado. Disimulado como gancho de apoyo u oculto detras.
5) En el escritorio. Debajo de la tabla superior o en cualquiera de los cajones

En este capítulo pasamos revista a los últimos adelantos en técnicas de vigilancia informatizada

El empleo oficial de la vigilancia por ordenador se está ampliando de forma continua, introduciéndose en cada vez más aspectos de la vida cotidiana. Con certeza todos hemos sido incluidos en alguno de estos sistemas en algún momento de nuestra vida.

En Gran Bretaña, por ejemplo, país bastante avanzado en este aspecto, existen vigilancias que parecen inocentes y que revisten vital importancia para las agencias del gobierno, como los registros de automóviles y matrículas de los ordenadores DVLC de Swansea o los registros de los DHSS basados en ordenador que pertenecen a la Seguridad Social. Ahora se ha planteado la posibilidad de enlazar todos estos distintos sistemas entre sí, para correlacionar los archivos de los sistemas DVLC y DHSS con, pongamos por caso, los archivos del ordenador de la Policía Nacional.

La Data Protection Act (Ley para la Protección de Datos) se formuló en un intento por detener los abusos de este tipo de poder. Pero hay quienes creen que la misma está anticuada, a la luz de los rápidos avances que se han producido en la tecnología del ordenador desde que fuera introducida.

Muchas de las técnicas que se utilizan en la vigilancia por ordenador y los sistemas de seguridad implican el reconocimiento de patrones, una técnica en virtud de la cual el ordenador compara lo que "ve" con patrones que ya tiene almacenados en su memoria. El inconveniente del reconocimiento de patrones es que exige grandes cantidades de espacio de memoria y enormes cantidades de capacidad de proceso del ordenador. En la actualidad se dispone de ambas cosas y a un precio reducido, lo que ha facilitado la realización de significativos avances.

Un ejemplo de ello es el nuevo sistema de impresión dactiloscópica que ha instalado Logica para la Policía Metropolitana en la Nueva Scotland Yard de Londres. Han sido necesarios 15 años de desarrollo para crear el sistema, que puede almacenar 650 000 impresiones dactilares y 100 000 "huellas" (las impresiones parciales que se encuentran en el escenario de un delito). El sistema simplemente compara las huellas con todas las impresiones almacenadas, para ver si concuerdan con alguna de las del archivo. Esta aplicación necesita de la potencia de cálculo de miniordenadores Prime, junto con procesadores matriciales sumamente eficientes y

Herramientas de la profesión

Protección informatizada
Muchos ejecutivos y personas que trabajan con información clasificada se protegen a sí mismos contra el espionaje electrónico con artilugios como este mezclador telefónico informatizado. En vez de hablar por el teléfono, el usuario digita sus mensajes en el teclado. El mezclador envía el mensaje en forma "silenciosa" a través de las líneas telefónicas normales. El mensaje se recibe entonces en la pantalla incorporada del sistema. Un sintetizador de voz incorporado también puede convertir el mensaje que se recibe a una forma verbal audible con sólo tocar un botón

Mensaje cifrado
Parecido al mezclador informatizado, este sistema envía un mensaje mezclado a partir de una nota manuscrita. Éste se encuentra protegido contra escuchas furtivos y micrófonos ocultos porque se envía silenciosamente. De esta forma se pueden enviar hasta firmas

Analizador de tensión
Este analizador mide la tensión de la voz y visualiza sus resultados instantáneamente en una sencilla lectura numérica. Se trata en realidad de un sofisticado detector de mentiras y también puede indicar si la persona experimenta ansiedad o se halla en tensión

pantallas y cámaras de televisión de elevado rendimiento. Aun así, sólo puede comparar cada día 200 o 300 huellas con las 650 000 impresiones almacenadas.

Un sistema similar de reconocimiento y comparación de patrones está instalado en un puente que atraviesa la autopista M1, con cámaras que enfocan hacia cada uno de los carriles. En realidad este sistema capta imágenes del número de matrícula de los coches cuando éstos se van aproximando y luego utiliza la potencia de cálculo del ordenador para analizar las imágenes y comparar los números con un archivo de vehículos buscados. La información de que se ha visto uno de los vehículos buscados se puede entonces transmitir por radio directamente a los coches patrulla de la autopista, que lo interceptarán.

Uno de los campos obvios en el que los desarrollos en el hardware para ordenador tuvieron una trascendental influencia fue la producción de dispositivos para vigilancia cada vez más pequeños; es decir, en los micrófonos ocultos. La tecnología del chip ha hecho posible fabricar transmisores de radio del tamaño de un grano de arroz, con una sofisticada electrónica de control incorporada. Un dispositivo típico "chupa" su potencia desde la fuente de alimentación eléctrica de la Central Telefónica hasta el teléfono intervenido, y sólo se conecta a sí mismo cuando efectivamente hay alguien hablando. Luego están los micrófonos de autoalimentación, igualmente diminutos, que se dejan caer en un rincón de una habitación y recogen todas las conversaciones que tienen lugar en ella (al igual que los arriba citados, sólo entran en funcionamiento cuando alguien habla) para transmitir a un receptor remoto.

Más próximo aún al "estilo James Bond", existe un micrófono "a distancia" que dispara un haz de láser hacia una ventana. Las vibraciones que la conversación produce en el cristal se captan como interferencia en el haz de láser reflejado y la información verbal se extrae de esta interferencia (por ordenador, por supuesto).

En las operaciones militares, al contrario que en las tareas de seguridad clandestinas, los operadores de ordenadores se encuentran con el problema opuesto. Intentan evitar ser controlados por otras personas y, ¡faltaría más!, son los chips y los ordenadores los que han venido a auxiliarlos. Los transmisores y receptores de radio actuales para el campo de batalla utilizan el salto de frecuencia (saltar, bajo el control del procesador, de una frecuencia a otra según un código preestablecido) para evitar escuchas secretas y la perturbación con señales de interferencia.

Los ordenadores para vigilancia y seguridad son, en la actualidad, grandes máquinas del tipo y la potencia de las que se emplean para la decodificación de claves complejas en lugares como la CIA y la National Security Agency, en Estados Unidos, por no mencionar el M15 y el M16 de Gran Bretaña. Pero los avances en la tecnología del hardware hacen presumir que pronto el reconocimiento de impresiones dactiloscópicas (y tal vez el de rostros) se automatizará y resultará muy económico.

En el futuro, es probable que los coches de la policía se equipen con ordenadores a bordo que puedan extraer datos instantáneamente de un registro escolar, un registro criminal, un registro médico, de la Seguridad Social o cualquier otro archivo oficial. A todos ellos se accedería introduciendo en la ranura del ordenador una tarjeta plástica de la Seguridad Nacional. La máquina aceptaría huellas dactilares y una fotografía, las compararía con los archivos centrales y confirmaría la identidad del sospechoso.

Un suave toque

Dominar la mecanografía al tacto es esencial para muchos aspectos de la informática personal

La capacidad de teclear palabras enteras sobre el papel o para contemplarlas en una pantalla, en vez de ir pulsando laboriosamente letras individuales en un teclado, elimina un paso que ocupa mucho tiempo en un proceso en el cual la velocidad suele tener importancia, y reduce el margen de error.

Tradicionalmente la mecanografía se ha enseñado mediante manuales de instrucción o en la clase, utilizando diversos métodos. Existe incluso en el mercado una gama de programas de software que conecta las lecciones directamente al ordenador. Independientemente del método de enseñanza, hay cuatro principios que se deben aprender para convertirse en un consumado mecanógrafo al tacto. Éstos son:

- Dominar el teclado
- Acostumbrarse a fijar los ojos en una pantalla (o papel)
- Precisión
- Velocidad

Estas habilidades se desarrollan de modo lento, mediante ejercicios, hasta que el mecanógrafo alcanza un nivel de destreza. La repetición constante es el principal método de aprendizaje: golpear las mismas letras hasta que la secuencia de los dedos se vuelve automática.

La disposición estándar en lengua inglesa para una máquina de escribir se conoce como teclado QWERTY, así llamado porque ésas son las letras situadas al lado izquierdo de la segunda de las cuatro filas de teclas. Los símbolos y las letras del teclado están dispuestos de acuerdo a su frecuencia de uso y cada fila del teclado, al objeto de la enseñanza, está dividida en lado izquierdo y lado derecho.

El aprendizaje de la mecanografía al tacto empieza por lo general por el dominio de las ocho teclas de la fila de letras del medio. Éstas se conocen como *letras base*, porque los dedos retornan a ellas después de haber pulsado cualquier otra tecla del teclado. Las teclas base para el lado izquierdo de la máquina de escribir y, por consiguiente, los cuatro dedos de la mano izquierda, son asdf. Las teclas base para el lado derecho de la máquina de escribir y, por tanto, para la mano derecha, son jkl; (punto y coma). Después de dominar estas teclas, el mecanógrafo aprende la situación de las otras en relación a su posición respecto a las teclas base, palpándolas y desplazándose hacia ellas estando aún en la posición "base". Una sorpresa con la que se encuentra el recién iniciado en la mecanografía es el uso al que se destina el dedo meñique. La mecanografía al tacto exige la utilización de *todos* los dedos y el meñique tiene asignada una cantidad de teclas "a cubrir", al igual que los demás dedos. Después de aprenderse las teclas, el mecanógrafo pasa a la barra espaciadora (que se pulsa con los pulgares) y a la tecla de cambio para las letras mayúsculas y las teclas que tienen asociados dos caracteres.

Lamentablemente para el usuario de un ordena-

Tony Sleep

Vista y oído
Sight and Sound ofrece cursos de mecanografía al tacto para principiantes y ha dado buenas pruebas de éxito. En esta fotografía, se les está enseñando a los estudiantes mediante una combinación de técnicas visuales y auditivas

dor personal, la utilización correcta de las teclas numéricas es un tema que se trata muy de vez en cuando (por no decir nunca) en los cursos o manuales de mecanografía al tacto. Para el programador, en especial, las teclas numéricas constituyen una parte vital e integral del teclado del ordenador. La mecanografía de números al tacto es bastante fácil de aprender una vez dominados el concepto y el funcionamiento de las teclas base. Los dedos de la mano izquierda simplemente se estiran por encima de las teclas alfabéticas para cubrir los números del uno al seis, mientras que los dedos de la mano derecha son responsables desde el siete al cero y de las teclas de signos que haya a continuación.

Se puede efectuar una interesante extensión del enfoque de la "tecla base" cuando se utiliza un ordenador personal con caracteres para gráficos a los que se puede acceder directamente desde el teclado. Aprendiendo la situación de los símbolos de gráficos, sería posible incorporarlos a su programa de mecanografía al tacto.

A fijar los ojos en una pantalla (o en un papel) se aprende al mismo tiempo que se va dominando el teclado. Éste es el aspecto más importante de la mecanografía al tacto, lo que la distingue del "teclado a dos dedos" o de mecanografiar "mirando y buscando". Tapar las teclas con cinta adhesiva o con unas cubiertas diseñadas especialmente es una excelente ayuda para el principiante, ya que impide caer en la tentación de bajar la vista hacia las manos. En este sentido, un teclado y una pantalla son mejores que una máquina de escribir y una hoja de papel. La pantalla está a la altura de los ojos, por lo cual disminuye la tentación de mirar hacia abajo. También se pueden corregir de inmediato los errores, asegurando mayor precisión, que es la siguiente etapa de la mecanografía al tacto.

La precisión es cuestión de práctica y concentración. Debe pulsarse la tecla con un golpecito firme y rápido, directamente en el centro, tocando las teclas con un mismo grado de regularidad, lo que produce un ritmo que, llegado el momento, se vuelve natural. Finalmente, utilizar el dedo equivocado o pulsar la tecla errónea llega a resultar extraño, y este ritmo natural contribuye en gran medida a adquirir velocidad.

La velocidad se mide mediante pruebas de tiempo y ejercicios, y se obtiene sólo después de haber superado las tres etapas que hemos descrito anteriormente. Los mejores mecanógrafos al tacto pueden alcanzar velocidades que superan las 100 palabras por minuto (ppm). Para el principiante, una velocidad razonable para plantearse como objetivo es de alrededor de 30 ppm.

Existen varios manuales de enseñanza diferentes y en el transcurso de los años se han ido desarrollando numerosos métodos. Aun así, todos ellos siguen valiéndose de los principios que hemos explicado. Pitman, la escuela internacional de secretariado, todavía edita un manual que se publicó por primera vez hace 35 años, así como una versión actualizada que enseña el teclado a través del mecanografiado de *palabras* desde el primer ejercicio. Dado que en estos ejercicios iniciales se utilizan palabras cortas y de uso común, la ortografía no exige ningún esfuerzo, lo que deja al principiante libre para concentrarse en adquirir la técnica necesaria.

El Dico Typing Course se precia de haberle ense-

Base de operaciones
Esta ilustración muestra la situación de las teclas base para las manos izquierda y derecha y las teclas que corresponden a cada dedo. Los dedos índice y meñique, por ser los que tienen más libertad de movimiento, son los encargados de pulsar el mayor número de teclas

Kevin Jones

ñado mecanografía al tacto a niños de 11 años de edad en menos de 10 horas. Este enfoque se basa en la obra del psicólogo norteamericano BF Skinner. En este método, el principiante arranca la página del manual y mecanografía la respuesta debajo del ejercicio. Esto responde al tradicional método de enseñanza según el cual el alumno copia exactamente ejercicios preparados. Los ejercicios se acompañan con dibujos de manos colocadas en la posición correcta sobre el teclado.

Una escuela que pone los métodos de mecanografía al tacto directamente en línea con el pensamiento moderno es Sight and Sound. Esta institución posee 11 centros de adiestramiento en Gran Bretaña y otros países. Los métodos de enseñanza de Sight and Sound implican la utilización de luces intermitentes y cassettes grabadas. Este sistema de enseñanza audiovisual simula las respuestas de ver, oír y reaccionar simultáneamente. En un gran tablero elevado una luz intermitente ilumina una letra. La cinta grabada se sincroniza con el tablero y la voz pregrabada del instructor al exclamar "¡Ya!" determina la velocidad a la cual se han de mecanografiar las letras. A medida que uno va adquiriendo más experiencia, se incrementa la velocidad a la cual el instructor exige que se mecanografíen las letras. La técnica de Sight and Sound se ha descrito como "lavado de cerebro", e incluso los instructores admiten que no comprenden del todo por qué este método produce resultados tan eficaces. No obstante, la cantidad de alumnos satisfechos que aprenden a mecanografiar al tacto con eficacia y sin esfuerzo indica que el sistema funciona, sea cual sea la razón.

Sumario del software

Los programas para ordenador que enseñan mecanografía al tacto se basan en juegos o en texto. Los paquetes basados en texto se fundamentan sobre todo en ejercicios y repetición de procedimientos. La mayor parte de las lecciones de los programas de esta clase se basan en texto, por lo general en forma de ejercicios escritos que el usuario debe copiar exactamente. Los paquetes basados en juegos enseñan a través de la utilización de gráficos, acción rápida y sonido

Ian McKinnell

Type Invaders

Editado por: Carswell Computers
Máquina: BBC Modelo B

Éste es un paquete basado en un juego para quienes poseen nociones de mecanografía. Sus sencillos gráficos visualizan letras "atacantes" que se deben destruir mecanografiándolas correctamente. Las palabras y las letras que se han fallado vuelven a atacar y finalmente pueden acabar rompiendo sus líneas de defensa y ocupando su territorio. Hay 10 niveles diferentes de juego, que van desde letras mayúsculas solamente hasta palabras de cinco letras que contienen mayúsculas y minúsculas, mayúsculas y cifras. También hay tres velocidades para escoger: fácil, rápida y veloz. A los mecanógrafos experimentados hasta la opción "veloz" les resultará sencilla. Sin embargo, se trata de un paquete práctico y entretenido para mecanógrafos lentos y normales. Los principiantes deberían practicar primero con el paquete Typeasy, del mismo fabricante

Sprintyper

Editado por: Micro Software International
Máquina: Commodore Vic-20

Paquete basado en texto que afirma mejorar la velocidad y precisión tanto de principiantes como de mecanógrafos avanzados. Posee una biblioteca de 356 635 frases para mejorar la habilidad. Para comenzar, se muestra en la pantalla una frase sencilla que se debe mecanografiar lo más rápidamente posible. Un tono grave señala un error, y sólo cesa al efectuar la corrección. Después de copiar la frase correctamente, aparecen en la pantalla el tiempo de mecanografiado, la cantidad de errores y un tiempo récord. Sprintyper es, esencialmente, una prueba de velocidad, que tiene poco que ofrecer al principiante en cuanto a ejercicios constructivos

Typing Tutor II

Editado por: Microsoft
Máquinas: Apple IIe y Apple IIe+

Para ejecutar este paquete se necesita Applesoft en ROM, 48 K de memoria, una unidad de disco y el sistema DOS 3.3. Éste es un paquete basado en texto conducido por menú, que proporciona una combinación de lecciones, párrafos de práctica y pruebas de velocidad. Su característica más importante es el sistema *Time response monitoring* (Control de tiempo de respuesta), que comprueba el mecanografiado 100 veces por segundo, detectando hasta las pausas más imperceptibles que se producen si los ojos se desvían de la pantalla al teclado. Los principiantes comienzan con cierta cantidad de letras para practicar. Cuando se van familiarizando con estas letras, y cuando la velocidad de la mecanografía equivale a 30 palabras por minuto, esas letras se transfieren a una columna FAST (rápido) y se seleccionan nuevas letras para practicar. Para los mecanógrafos experimentados, el informe de progreso del párrafo de práctica detalla la cantidad de errores cometidos, las teclas con las que se incurrió en esos errores, la velocidad y la precisión. Es un paquete muy aconsejable para los mecanógrafos de cualquier nivel de destreza. Al principio resulta un poco difícil de seguir y es recomendable que los usuarios previamente estudien con sumo cuidado la documentación

Brillo y esplendor

Ahora añadiremos al programa refinamientos que harán el juego más atractivo y emocionante

El primer añadido que efectuaremos es una rutina de "francotiroteo". Ésta simula a un francotirador que dispara hacia el campo de minas tratando de hacer blanco en el detector de minas o bien en el ayudante. Los disparos se visualizarán como una línea en alta resolución que atravesará la pantalla desde el margen izquierdo del campo de minas hacia el derecho. Para introducir en el francotiroteo un elemento de azar, seleccionaremos las coordenadas de los puntos de comienzo y final utilizando la función RND. Los valores de xcomienzo y xfinal se establecen en el procedimiento inicializar-variables. La diferencia entre estos dos valores es de 1 024 unidades de gráficos. Para que la línea de francotiroteo pueda detectar un blanco, ya sea en el detector o en el ayudante, ha de trazar un corto segmento de la línea y verificar la superficie que hay por delante en busca de la presencia del color lógico 1 (utilizando la instrucción POINT) antes de trazar el segmento siguiente. Esta secuencia se debe repetir hasta alcanzar el otro lado de la pantalla o hacer blanco.

Ahora debemos decidir qué longitud deseamos utilizar para los pasos. Si elegimos una longitud de paso muy corta, entonces aumentará el tiempo que se tarda en trazar la línea. Por el contrario, si elegimos una longitud de paso demasiado larga quizá no consigamos detectar los objetivos. Dado que cada celda de caracteres equivale a lo ancho a 64 unidades para gráficos, parece razonable una longitud de paso de media celda de caracteres (es decir, 32 unidades para gráficos). Por consiguiente, si optamos por que nuestra longitud de paso en la dirección *x* (dx) sea de 32 unidades, podemos trazar la línea en un total de 1024/32 = 32 pasos. Si calculamos las coordenadas *y* de los puntos de comienzo y final al azar, entonces la longitud de paso adecuada en la dirección *y* (dx) se puede calcular dividiendo por 32 la diferencia entre los dos valores.

Nuestro último problema es hallar alguna forma de borrar la línea después de haberla trazado. La solución radica en el concepto de colores lógicos del BASIC BBC y su capacidad para realizar operaciones lógicas entre ellos. En la modalidad 5 hay cuatro colores lógicos. A menos que los modifiquemos, éstos son:

Color lógico	0	1	2	3
Equivalente binario	00	01	10	11
Color real normal	negro	rojo	amarillo	blanco

Utilizando GCOL podemos efectuar varias operaciones lógicas entre el color que estamos trazando y el color que ya está allí. La instrucción posee dos parámetros, el segundo de los cuales indica el color lógico a trazar. El primer número establece el método de trazado:

GCOL0	Traza el color especificado
GCOL1	Realiza operación OR
GCOL2	Realiza operación AND
GCOL3	Realiza operación OR Exclusivo
GCOL4	Realiza NOT en el color que ya está allí

Esto puede parecer complicado, pero con unos pocos ejemplos clarificaremos el funcionamiento de la instrucción. Si está presente el blanco (color lógico 3) en la posición que deseamos trazar, y queremos trazar en rojo (color lógico 1), las diversas modalidades de operación de GCOL producirán los siguientes resultados:

GCOL0,1	Borrará el blanco y trazará en rojo		
GCOL1,1	Operará rojo y blanco con OR para producir blanco	rojo blanco OR blanco	01 11 11
GCOL2,1	Operará rojo y blanco con AND para producir rojo	rojo blanco AND rojo	01 11 01
GCOL3,1	Operará rojo y blanco con OR Exclusivo para producir amarillo	rojo blanco amarillo	01 11 10
GCOL4,1	Operará blanco con NOT para producir negro	blanco negro	11 00

¿Y cómo puede esto ayudarnos a solucionar nuestro problema de borrado? Podríamos trazar la línea en blanco y después volver a trazarla en negro para borrarla. Pero si debajo de la línea hubiera algo, como por ejemplo una mina, entonces esto haría que quedara un "agujero". Sin embargo, podemos operar con OR Exclusivo el rojo y el color ya presente en cada punto que atraviesa la línea. Cuando atraviese un área blanca, obtendremos un segmento de línea amarillo. Si operamos después la misma zona con OR Exclusivo en rojo el resultado sería:

rojo	01
amarillo	10
EOR	—
blanco	11

Por consiguiente, se retoma el color original. Quizá desee comprobar que efectuar dos OR Exclusivos siempre restaura el color original. Podemos emplear este factor para borrar nuestra línea. Si trazamos la línea original utilizando una operación EOR y después volvemos a trazar exactamente la misma línea, utilizando otra vez OR Exclusivo, borraremos la línea y restauraremos los colores de fondo a su condición original antes del primer trazado. He aquí el listado completo para el procedimiento francotirador:

```
3110 DEF PROCfrancotirador
3120 ycomienzo = RND(750) + 220
3130 yfinal = RND(750) + 220
3140 dx = 32:dy = (yfinal - ycomienzo)/32
3150 GCOL 3,3
3160 PROClinea
3170 IF POINT(x,y) = 1 THEN PROCexplotar(x,y) ELSE PROClinea
3180 ENPROC
```

Y éste es el listado para el procedimiento linea:

```
3450 DEF PROClinea
3460 SOUND0,-8,4,5
3470 x = xcomienzo:y = ycomienzo
3480 MOVE x,y
3490 REPEAT
3500 DRAW x,y
3510 x = x + dx:y = y + dy
3520 UNTIL x > xfinal OR POINT(x,y) = 1
3530 ENDPROC
```

Como vimos en el último capítulo, el BBC Micro puede generar sonidos bastantes complejos. Para quienes sientan inclinaciones musicales, ahora vamos a añadir al programa una corta melodía. Para hacer las cosas lo más sencillas posible utilizaremos sólo un canal. La melodía se puede ejecutar simplemente especificando la frecuencia y la duración de cada una de sus notas.

```
4090 DEF PROCmusica
4100 REM ** 1A BARRA **
4110 SOUND1,-8,213,5
4120 SOUND1,-8,209,5
4130 SOUND1,-8,213,5
4140 SOUND1,-8,209,5
4150 SOUND1,-8,213,5
4160 SOUND1,-8,193,5
4170 SOUND1,-8,205,5
4180 SOUND1,-8,197,5
4190 REM ** 2A BARRA **
4200 SOUND1,-8,185,20
4210 SOUND1,-8,165,5
4220 SOUND1,-8,185,5
4230 SOUND1,-8,193,20
4240 REM ** 3A BARRA **
4250 SOUND1,-8,165,5
4260 SOUND1,-8,193,5
4270 SOUND1,-8,197,20
4280 ENDPROC
```

Página de títulos: Podemos utilizar las ideas del trazado con OR Exclusivo y el trazado de puntos relativos para crear una interesante secuencia de títulos. Este procedimiento dibuja la palabra MINES (minas) utilizando gráficos en alta resolución. Cada nueva línea dibujada en la palabra se traza en relación a la última, de modo que podemos situar la palabra completa en cualquier lugar de la pantalla simplemente especificando el punto de comienzo. Si trazamos la palabra y luego la volvemos a trazar en OR Exclusivo antes de desplazarla hacia arriba y repitiendo la acción, podemos lograr que la palabra parezca que flote hacia arriba de la pantalla. GCOL0,129 establece el color del fondo en rojo. Efectuando un subsiguiente CLG toda la pantalla se colorea de rojo. Al mismo tiempo, también podemos tocar la melodía definida anteriormente llamando a PROCmusica. La información retenida en PROCmusica se procesa bastante más rápidamente que lo que se toca, de modo que se utiliza un buffer para almacenar la información SOUND hasta que es tocada. Ello significa que el procesador queda libre para realizar otras cosas mientras todavía está sonando la melodía.

Factores de destreza: Para hacer que el juego resulte un poco más emocionante podemos emplear la noción de factores de destreza. Después de haberse visualizado el título, solicitaremos un número entre 0 y 9, que se almacenará en la variable destreza. Éste se puede utilizar entonces para aumentar la cantidad de minas en el campo de minas y la velocidad a la cual el francotirador dispara sobre la zona. Lo primero se puede realizar introduciendo una ligera modificación en el procedimiento preparación que dimos con anterioridad (véase p. 885). Cambie las líneas 1930 y 1940 por:

```
1930 factor = destreza*3 + 30
1940 PROCcolocar-minas(factor)
```

Además, cuando volvemos a colocar las minas durante el procedimiento restaurar, debemos calcular la cantidad de minas que queda modificando de este modo la línea 3950:

```
3950 minas-restantes = factor-marcador/150
```

El listado del procedimiento pagina de titulos es:

```
1300 DEF PROCpagina-titulos
1310 GCOL 0,129
1320 CLG
1330 GCOL 3,3
1340 PROCmusica
1350 Y = 100:X = 0
1360 REPEAT
1370 X = X + 20:Y = Y + 50
1380 FOR I = 1 TO 2
1390 PROCminas
1400 NEXT I
1410 UNTIL Y > 700
1420 :
1430 PROCminas
1440 PRINTTAB(0,20) "Factor destreza (0-9)?"
1450 PROCmusica
1460 REPEAT
1470 destreza = GET-48
1480 UNTIL destreza >-1 AND destreza < 10
1490 ENDPROC
1500 :
1510 DEF PROCminas
1520 PLOT4,X,Y
1530 REM ** LETRA M **
1540 PLOT1,0,200
1550 PLOT1,80,-100
1560 PLOT1,80,100
1570 PLOT1,0,-200
1580 REM ** LETRA I **
1590 PLOT0,40,0
1600 PLOT1,80,0
1610 PLOT0,-40,0
1620 PLOT1,0,200
1630 PLOT0,-40,0
1640 PLOT1,80,0
1650 REM ** LETRA N **
1660 PLOT0,40,-200
1670 PLOT1,0,200
1680 PLOT1,120,-200
1690 PLOT1,0,200
1700 REM ** LETRA E **
1710 PLOT0,160,0
1720 PLOT1,-120,0
1730 PLOT1,0,-200
1740 PLOT1,120,0
1750 PLOT0,-40,100
1760 PLOT1,-80,0
1770 REM ** LETRA S **
1780 PLOT0,280,60
1790 PLOT1,0,40
1800 PLOT1,-120,0
1810 PLOT1,0,-100
1820 PLOT1,120,0
1830 PLOT1,0,-100
1840 PLOT1,-120,0
1850 PLOT1,0,40
1860 ENDPROC
```

Hasta este punto hemos estado utilizando un programa de llamada provisional (véase p. 874) para ensamblar nuestros procedimientos entre sí, pero ahora hemos ensamblado todos los procedimientos que se requieren para el bucle del programa principal del juego. Borre el programa de llamada provisional (líneas 10 a 70) e incorpore lo siguiente:

```
2020 DEF PROCbucle
2030 REPEAT
2040 PROCactualizar-tiempo
2050 PROCleer-teclado
2060 rand = RND(50-destreza)
2070 IF rand = 1 THEN PROCfrancotirador
2080 UNTIL TIME > 12099 OR flag-final = 1
2090 ENDPROC
```

Ahora podemos escribir nuestro programa de llamada. Entre estas líneas:

```
1060 max-marcador$ = "00000"
1110 MODE5
1120 REM ** APAGAR CURSOR **
1130 VDU23;8202;0;0;0;
1140 PROCpagina-titulos
1150 CLS
1160 PROCpreparacion
1170 :
1180 PROCbucle
```

Corte de secuencia

Un proceso general puede representarse dividido en varias partes gracias a la utilización de un símbolo especial

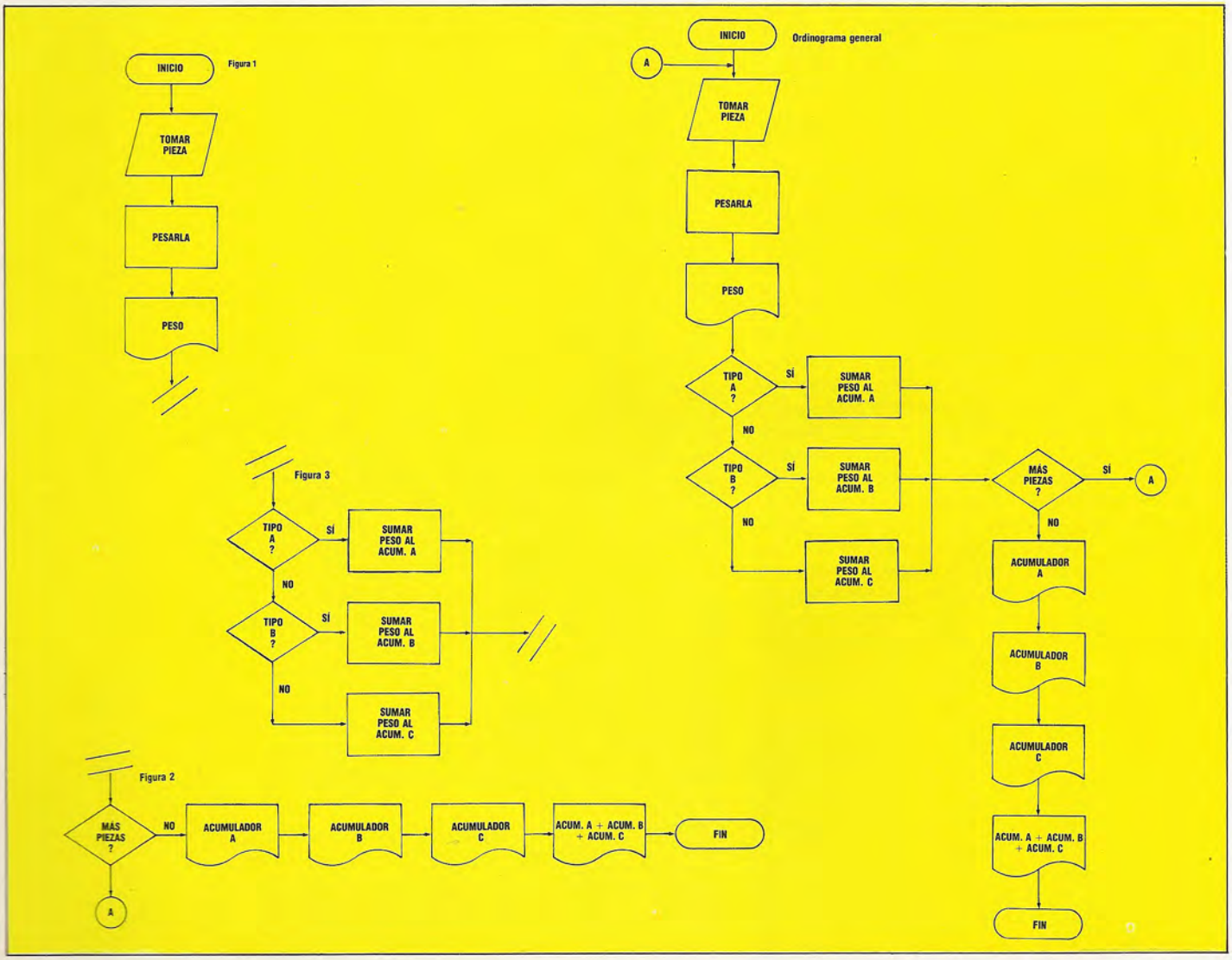
El símbolo // se emplea en diagramación para representar un corte en la secuencia; siempre que aparezca al principio o al final de una representación se entenderá que lo que figura forma parte de un proceso general más amplio. En el ejemplo siguiente se han realizado incisiones en el ordinograma general, diseccionándolo en tres cuerpos.

Una máquina colocada al final de una línea de producción controla los tres tipos de piezas que se fabrican (A, B y C). Su función es pesar cada una de las piezas que, filtradas, llegan a ella e imprimir su peso. Así sucesivamente hasta que finaliza la producción diaria; entonces debe dar los pesos totales de las piezas de cada tipo y el total general.

Haciendo un análisis del problema, llegamos a la conclusión de que hay una parte común para todas las piezas, independientemente del tipo al que pertenezcan, que es la correspondiente a la entrada y pesaje, así como la impresión del peso (fig. 1).

Nuestro problema termina con la pregunta de si quedan más piezas por procesar. En caso afirmativo se retorna al principio del bucle; en caso negativo se llega a la salida de los datos que se precisan recoger como final de jornada, a saber, tres totales parciales y un total general. También en este caso la obtención del peso total general se lleva a cabo mediante la suma de los tres acumuladores parciales, sin tener que recurrir innecesariamente al empleo de un acumulador general (fig. 3).

Pero para obtener estos resultados antes se ha debido llevar a cabo una serie de pasos que vienen marcados por una sucesión de preguntas eliminatorias *(test en cascada)* destinadas a conocer, a través de su peso, el tipo de pieza que se ha controlado, para así sumar su peso al correspondiente acumulador (fig. 2).



Precursor del futuro

El elegante e innovador Macintosh de Apple ha marcado un hito definitivo en el diseño de microordenadores

El Macintosh es distinto de todos los otros ordenadores que hemos analizado hasta ahora. En realidad, es diferente de cualquier otra máquina que existe en el mercado. A pesar de que el Macintosh es principalmente una máquina de oficina, Apple ha optado por crear su propio camino en vez de imitar a otros fabricantes que han adoptado los estándares IBM para los diseños de sus máquinas. Tomando este arriesgado camino, Apple ha mantenido su reputación de innovador en una industria llena de "semejanzas".

La disposición de los elementos del Macintosh es inusual. La elegante y esbelta unidad del sistema es pequeña para una máquina de su capacidad de proceso. La visualización se realiza en una pantalla de nueve pulgadas de alta resolución, y la unidad de disco utiliza discos Sony de 3 ½ pulgadas. En la carcasa se ha incorporado un asa moldeada para transportarlo, de modo que el Macintosh se puede considerar como una auténtica máquina portátil. Junto con el teclado, el "ratón" y un maletín opcional para su transporte, el sistema pesa en total 11,6 k. El maletín tiene compartimientos para todos los componentes del ordenador, al estilo de una cesta de picnic.

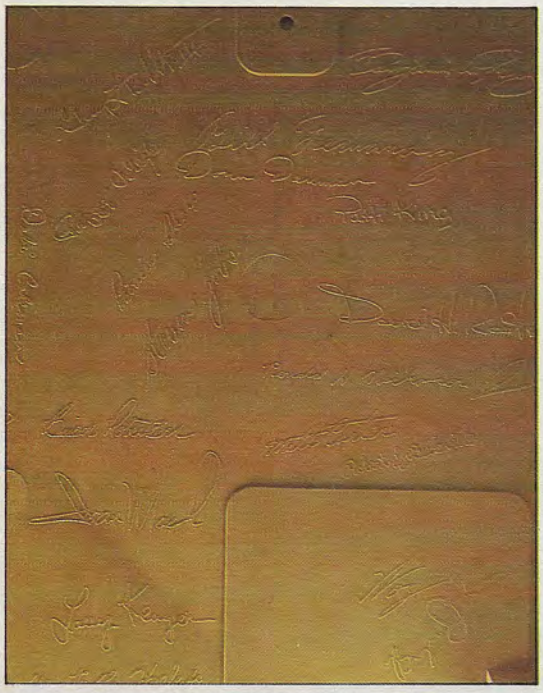
El Macintosh tiene un teclado tipo máquina de escribir, con un "tacto" excelente y adecuado para mecanografía al tacto. El teclado posee su propio procesador para tratamiento de funciones especiales y juegos de caracteres internacionales. El otro componente del "Mac", como se lo llama familiarmente, es el ratón. Llamado así en parte por el "rabo" que lo conecta con la unidad del sistema, este dispositivo manual, del tamaño de un paquete de cigarrillos, se desplaza por una superficie plana y nivelada. El cursor se desplaza en la pantalla de acuerdo al movimiento que el operador imprime al ratón sobre la superficie plana y se lo puede utilizar para seleccionar las actividades que el usuario desee que efectúe la máquina. Este enfoque al diseño de ordenadores se considera mucho más agradable para el usuario que el que emplean la mayoría de las máquinas, que exigen un conocimiento de instrucciones específicas de operatoria. Por ejemplo, si se deseara abrir un archivo de documentos, movería el ratón de modo que el cursor cayera sobre el pequeño símbolo gráfico (icono) que representa una hoja de papel. Pulsando entonces el botón del ratón la pantalla se abriría para esa actividad. Habiendo entrado su archivo desde el teclado, se utilizaría el ratón para retornar al menú principal de instrucciones de iconos y se podría guardar el archivo colocando el cursor sobre el símbolo que representa un disco.

El Macintosh viene con 128 Kbytes de memoria para el usuario, que se pueden aumentar hasta 512 Kbytes mediante la sustitución de la RAM existente por chips de 256 Kbytes. El Mac también posee 64 Kbytes de ROM empaquetados ajustadamente con software operativo, que manipula virtualmente todas las operaciones del sistema, así como algunas características especiales. La unidad de disco Sony utiliza discos de 3 ½ pulgadas, que almacenan hasta 400 Kbytes por una cara y son más fiables que los discos de 5 ¼ pulgadas.

La pantalla del Macintosh es de 512 por 342 pixels y responde a un "mapa de bits", de modo que se puede acceder de forma individual a cada uno de sus más de 175 000 puntos. Ello hace posibles algunas aplicaciones de gráficos verdaderamente asom-

Ian McKinnell

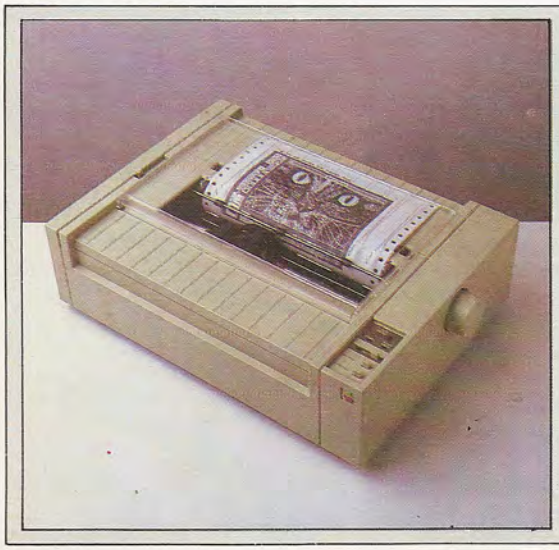
El sistema Macintosh
El Macintosh está diseñado para ocupar el menor espacio posible sobre un escritorio. La altísima resolución de la pantalla permite realizar gráficos que por lo general sólo son posibles en máquinas que cuestan 10 veces más que ésta



Panel de firmas
En el interior de la carcasa del Macintosh, Apple ha grabado las firmas del equipo de diseño. Entre ellas se incluyen las de Steven Jobs y Steve Wozniak, que fueron los creadores de Apple. Fue su genio lo que posibilitó la existencia del Lisa y el Macintosh

brosas. Además de proporcionar una enorme diversión, los trucos de gráficos del Macintosh tienen un gran valor para diseñadores, arquitectos, asesores, relaciones públicas, fotógrafos y muchos otros profesionales. Dado que el Mac está diseñado para funcionar específicamente con la impresora de Apple ImageWriter, de gran velocidad, todos sus espléndidos gráficos se imprimen exactamente igual a como aparecen en la pantalla.

No obstante la gran calidad y fiabilidad del hardware del Macintosh, es la solidez adicional de su software lo que hace que la máquina sea excepcional. Con la integración de hardware y software y las exhaustivas instrucciones operativas basadas en ROM, a las personas que desarrollan software les resulta relativamente sencillo transferir al Macintosh programas escritos para otros ordenadores. El ordenador es tan fácil de usar que literalmente se lo puede enchufar y ponerse en seguida a trabajar con él sin ningún conocimiento previo sobre operatoria de ordenadores.



Mejorando la imagen

ImageWriter es una impresora en serie que genera texto a una velocidad de hasta 120 caracteres por segundo. Su velocidad es aún más evidente en la modalidad de gráficos, porque genera gráficos del tipo mapas de bits, siguiendo el formato de la pantalla

Placa analógica
Esta placa controla la pantalla y la fuente de alimentación eléctrica. El Macintosh no tiene necesidad de ventilador. El exceso de calor se canaliza, a través de placas metálicas, hacia las ranuras de ventilación del mueble

Altavoz incorporado

Cabezal de la unidad de disco

Control de contraste de la pantalla

Unidad de disco Sony de 3 ½ pulgadas
Construida especialmente para Apple; esta unidad contiene 400 K en simple cara. Los discos de doble cara almacenarán, cuando estén disponibles, 800 K cada uno

RAM de video
Parte de los 22 K requeridos por la visualización de pantalla se toman de estos circuitos DMA (*direct memory access*)

Teclado
El teclado separado del Macintosh posee su propio procesador para tratar juegos de caracteres internacionales y funciones especiales. Debido a la existencia del ratón, no se necesitan teclas para el cursor

Conector del teclado

Ratón
Controla el movimiento del cursor y se utiliza para "seleccionar" objetos en la pantalla, actuando luego sobre ellos según instrucciones escogidas de menús

Conexión de analógico a digital
Un cable plano conecta las placas analógica y digital

Salida de audio

Salida para impresora

Bus en serie
El bus en serie, también llamado "ranura virtual", permite añadir diversos dispositivos periféricos

Conector para la unidad de disco externa

Conector del ratón

Adaptador E/S 6522
El 6522, también utilizado para el control de E/S en el Apple IIe, manipula el teclado, el ratón y los circuitos del reloj de "tiempo real"

Controlador de comunicaciones en serie

Controlador del disco
Este chip controla la unidad de disco Sony incorporada, así como la unidad externa, si la hay

CPU
Motorola 68000. Este chip realiza operaciones internas de 32 bits a la vez, pero envía y recibe datos de 16 bits

128 K de RAM para el usuario
Estos 16 chips se pueden reemplazar por chips de RAM de 256 K, que le proporcionarían al Macintosh un total de 512 K de memoria para el usuario. No obstante, 128 K son suficientes para las aplicaciones existentes

APPLE MACINTOSH

DIMENSIONES
343 x 254 x 254 mm (mueble pantalla/unidad disco)

CPU
Motorola 68000, 7,83 MHz

MEMORIA
128 K de RAM, 64 K de ROM

PANTALLA
Monocromática, 512 x 342 pixels, ventanas, menús de selección, "iconos"

INTERFACES
Ratón, impresora, unidad de disco externa, amplificador hi-fi, bus en serie

LENGUAJES DISPONIBLES
BASIC, COBOL, PASCAL

TECLADO
Tipo máquina de escribir, 59 teclas, teclado numérico opcional

DOCUMENTACION
Existe un manual de operatoria con una cassette de audio y un disco de "enseñanza guiada". Se dan manuales para MacPaint y MacWrite, que incluyen enseñanza guiada mediante combinación de disco y cassette

VENTAJAS
El Mac posee un software muy potente y fácil de utilizar. El ratón hace que la operatoria resulte muy sencilla y directa. Los componentes, muy accesibles y estándares, hacen que el mantenimiento y la actualización resulten simples

DESVENTAJAS
El Mac es un ordenador pequeño muy caro y fuera del alcance de muchos usuarios. Con una sola unidad de disco, las operaciones sobre archivos son lentas y engorrosas. Existe carencia de software

Ayuda exterior

La segunda unidad de disco, externa, que Sony ha hecho a la medida para Apple, es un valioso accesorio para el Macintosh. Sin ella, muchas operaciones se retrasan debido a la necesidad de cambiar los discos. Esta unidad externa no requiere interface

Ian McKinnell

Apple realizó fuertes inversiones en la tecnología que permitió crear el Lisa y el Macintosh y está acercando esa tecnología al gran público con agresividad y orgullo. A la vista de su diseño y su construcción, el Macintosh es a todas luces un precursor del futuro.

Teclado
Los primeros envíos de máquinas Macintosh incluían el teclado norteamericano. El teclado de este ordenador posee un juego de caracteres internacionales completo incorporado, de modo que cualquier versión de Macintosh puede generar todos los caracteres especiales que exigen el alemán, francés, castellano, griego y otros idiomas

"Ventana escritorio"
La pantalla del Macintosh continúa la analogía del "escritorio" creada para el ordenador Lisa, y utiliza "ventanas" para mostrar porciones de documentos que son demasiado extensos como para caber en la pantalla. Las ventanas se pueden superponer unas sobre otras, como vemos aquí, de modo que utilizar el Macintosh es bastante parecido a trabajar sobre un escritorio

Ian McKinnell

Aguas turbulentas

Cocodrilos, anacondas, troncos flotantes y helicópteros que arrojan minas son algunos de los peligros que se deben sortear en este juego

River rescue (Rescate en el río) es un juego recreativo del más puro estilo de "disparos" y sin ninguna pretensión de ser nada más que eso. Lo produce Creative Sparks, la división de software de Thorn EMI, y el apoyo de una empresa de tal calibre resulta evidente. Se vende en versiones para cuatro ordenadores personales: Spectrum de 48 K, Commodore 64, Atari y Vic-20 sin ampliar, y se suministra en un paquete burbuja diseñado especialmente en vez de en la caja habitual de cassette de música. Con cada versión se incluye un pequeño pero muy útil folleto de instrucciones.

El juego en sí es básicamente muy simple, pero incorpora acción suficiente como para satisfacer a cualquier adicto a los juegos recreativos. El jugador controla la motora de rescate y tiene la misión de salvar a un grupo de científicos que se han quedado embarrancados en un tramo del río. En primer lugar, no se explican las causas por las cuales es necesario rescatar a los científicos, pero las instrucciones indican que deben ser trasladados a un hospital, de modo que presumiblemente han sido víctimas de algún tipo de accidente.

Mientras intenta recoger a los científicos heridos, el jugador debe conducir su barca, que navega a considerable velocidad, evitando islas y troncos flotantes, apartándose eventualmente de los cocodrilos que puedan aparecer. La versión para el Vic-20 es algo diferente, con peligros adicionales en forma de anacondas y canoas. A intervalos se ven a lo largo de la ribera varios muelles y en ellos se encontrarán los individuos a rescatar. La transferencia con éxito de un científico hasta el otro lado del río incrementa de forma considerable el marcador, pero también se pueden conseguir puntos matando a los cocodrilos que pululan por el río.

Se pueden conseguir puntos extras transfiriendo a los científicos por grupos, si bien la capacidad máxima de la barca es de nueve científicos. Ello hace que las cosas sean un poco más dificultosas, porque toda la tripulación se perderá si el barco choca contra algún obstáculo. Por consiguiente, hay que elegir entre un marcador elevado con el riesgo de perder todo o jugar seguro transfiriendo a los pasajeros de uno en uno. Para empeorar aún más las cosas, es probable que en cualquier momento aparezca un helicóptero (un avión, en la versión para el Spectrum) y arroje minas al agua, que se deben hacer estallar antes de poder seguir adelante.

La versión para el Vic-20 se suministra en formato de cartucho para evitar el tedioso proceso de cargar el cassette. Aquí se ofrece la opción de tres o seis científicos embarrancados y se dispone de seis vidas por juego. Además, los puntos conseguidos en cada "vida" se pasan a las subsiguientes "reencarnaciones", lo que simplifica considerablemente las cosas. En esta versión, no obstante, se tienen tres ríos extras por los cuales se puede navegar.

River rescue es un juego de acción pura y del tipo en el que hay que disparar contra todo lo que se mueva, y se ha diseñado con cuidado para que resulte suficientemente difícil de jugar como para mantenerlo a uno ocupado durante un tiempo, si bien se puede argüir que carece de la imaginación necesaria para hacer del mismo algo realmente especial.

River rescue: Para el Spectrum de 48 K, el Atari, el Commodore 64 y el Vic-20 (sin ampliar)
Editado por: Creative Sparks, 1st Fllor, Thomson House, 296 Farnborough Road, Farnborough, Hants., Gran Bretaña
Autor: Kevin Buckner
Palancas de mando: Kempston/Interface 1 (Spectrum)
Compatibles con Commodore (Vic-20 y Commodore 64)
Compatibles con Atari
Formato: Cassette;
cartucho (sólo el Vic-20)

¡Al rescate!
En la ilustración vemos *River rescue* en un Spectrum. La primera pantalla muestra la página de título, que proporciona buenos indicios sobre los gráficos que aparecerán. En la segunda pantalla vemos el juego ya ejecutándose. El barco transporta a un científico hacia lugar seguro, donde se unirá a los otros que han sido rescatados

Liz Heaney

Salir del círculo

En esta ocasión analizaremos la "recursión", una técnica que se utiliza en programación avanzada

La mejor manera de resumir el tema de esta investigación es mediante una definición que se suele dar comúnmente en broma:

Recursión: *véase* Recursión

Esta definición circular demuestra un aspecto esencial de la recursión: algo que se define en sí mismo. Pero omite otro importante aspecto: para que la recursión sea operativa, debe haber una forma de salir de la circularidad.

El puzzle que hemos utilizado para ilustrar la recursión es *Las torres de Hanoi*. Comienza con un número de discos apilados por orden de tamaño, con el disco más grande en la parte inferior de la pila y el disco más pequeño arriba. Para resolver el puzzle se deben trasladar todos los discos de una pila a otra de acuerdo a las siguientes reglas:

1) Sólo se puede trasladar un disco cada vez;
2) No se puede colocar un disco sobre otro más pequeño;
3) No puede haber nunca más de tres pilas de discos.

El diagrama ilustra cómo utilizamos el concepto de recursión para que el problema sea manejable. Empezamos con una pila de cuatro discos. Asignándole a una variable N el valor de cuatro, indicamos el número total de discos que se deben trasladar. Dado que las reglas prohíben mover más de un disco por vez, utilizamos una fórmula recursiva para reducir el valor de N en uno, continuando luego el cálculo hasta que N sea igual a 1. Cuando N = 1, el programa interrumpe el cálculo y traslada el disco apropiado.

Trabajando con una versión de BASIC que permita la recursión es fácil escribir un programa que siga exactamente el proceso que hemos descrito. En el programa en BASIC BBC, todo el trabajo de calcular los movimientos se realiza entre las líneas 1000 y 1050. ¡El resto del programa sólo produce la visualización gráfica en movimiento!

El poder de la torre

El programa pide al jugador que entre la cantidad de discos a trasladar. La cantidad de discos solicitada se coloca en una pila. El algoritmo recursivo va desplazando los discos de uno en uno a través de las pilas A, B y C, en las primeras tres etapas que vemos abajo. Finalmente, la pila original de N discos se reduce a cero y no queda ningún disco más por trasladar, como se aprecia en el resultado final. Cuando se completen los movimientos se habrán reubicado N discos por orden secuencial en la pila B

Liz Dixon

La versión para el Spectrum

Para convertir el programa *Las torres de Hanoi* al BASIC del Spectrum tenemos que reemplazar un procedimiento recursivo por una subrutina recursiva, que comienza en la línea 1000 de nuestro listado. Cada vez que la subrutina tiene que hacer una llamada recursiva a las matrices M, A, B o C incrementa la variable puntero J y coloca los nuevos valores en M(J), A(J), B(J) y C(J). Posteriormente, estos nuevos valores se pueden utilizar en la siguiente llamada a la subrutina sin alterar los valores antiguos. Al final de la subrutina, se decrementa el valor de J, restaurando, por consiguiente, los valores antiguos. Este método siempre se puede emplear para escribir subrutinas recursivas en BASIC, independientemente de lo complicada que sea la recursión.

La sección del programa para la visualización es directa, imprimiendo un objeto en una nueva posición y borrándolo mediante la impresión de caracteres negros en la posición antigua. El programa muestra la sección transversal de una pila de discos. Para que las pilas parezcan simétricas, hemos terminado cada barra con caracteres de gráficos hechos mitad con un espacio y mitad con un color sólido.

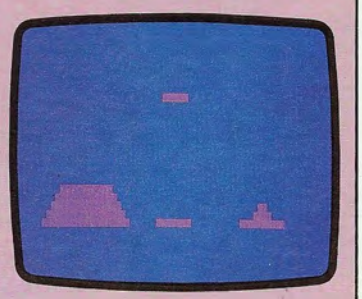
BBC Micro

```
 10 DIM M(10): DIM A(10): DIM B(10): DIM C(10)
 20 DIM D$(10,10): DIM H(3): DIM P(3,10)
 30 GO SUB 3000
 90 DIM M(100): DIM A(100): DIM B(100): DIM C(100)
100 INPUT "CUANTOS DISCOS? ";N
110 IF N < 1 OR N > 10 THEN GO TO 100
120 GO SUB 3100
130 LET J = 1: LET M(J) = N: LET A(J) = 1: LET
    B(J) = 2: LET C(J) = 3
140 GO SUB 1000
200 STOP
1000 IF M(J) = 1 THEN GO SUB 1500: RETURN
1010 LET J = J + 1
1020 LET M(J) = M(J-1)-1
1030 LET A(J) = A(J-1)
1040 LET B(J) = C(J-1)
1050 LET C(J) = B(J-1)
1060 GO SUB 1000
1100 LET M(J) = 1
1110 LET A(J) = A(J-1)
1120 LET B(J) = B(J-1)
1130 LET C(J) = C(J-1)
1140 GO SUB 1000
1200 LET M(J) = M(J-1)-1
1210 LET A(J) = C(J-1)
1220 LET B(J) = B(J-1)
1230 LET C(J) = A(J-1)
1240 GO SUB 1000
1300 LET J = J-1
1310 RETURN
1500 LET PA = A(J): LET PB = B(J)
1510 LET M$ = D$(P(PA,N + 1-H(PA)))
1520 FOR I = 22 - H(PA) TO 7 STEP -1
1530 PRINT AT I -1,10*(PA - 1.;M$;
1540 PRINT AT I,10*(PA-1); B$;
1550 NEXT I
1560 FOR I = 10 * (PA -1) TO 10*(PB - 1) STEP SGN (PB - PA)
1570 PRINT AT 6,I;M$;
1575 PRINT AT 6,I;B$;
1580 NEXT I
1590 FOR I = 6 TO 20 - H(PB)
1600 PRINT AT I,10*(PB - 1);B$;
1610 PRINT AT I + 1,10*(PB - 1);M$;
1620 NEXT I
1640 LET H(PB) = H(PB) + 1: LET P(PB,N +
    1-H(PB)) = P(PA,N + 1 - H(PA))
1650 LET P(PA,N + 1 - H(PA)) = 0: LET H(PA) = H(PA)-1
1660 RETURN
3000 LET B$ = "                ": LET C$ = CHR$
    143 + CHR$ 143 + CHR$ 143 + CHR$143
3010 LET C$ = "": FOR I = 1 TO 10: LET
    C$ = C$ + CHR$ 143: NEXT I
3020 FOR I = 1 TO 9 STEP 2
3030 LET D$(I) = B$( TO 4 - INT (I/2)) +
    CHR$ 133 + C$ ( TO 2 * INT (I/2)) + CHR$
    138 + B$( TO 4-INT (I/2))
3040 LET D$ (I + 1) = B$( TO 4 - INT (I/2)) +
    C$( TO I + 1) + B$( TO 4-INT (I/2))
3050 NEXT I
3060 RETURN
3100 INK 3: PAPER 6: BORDER 6: CLS
3120 FOR I = 1 TO N
3130 PRINT AT 21-N + I,0;D$(I);
3135 LET P(1,I) = I; LET P(2,I) = 0: LET P(3,I) = 0
3140 NEXT I
3150 LET H(1) = N: LET H(2) = 0: LET H(3) = 0
3160 RETURN
```

Spectrum

```
 10 DIM D$(12),H(3),P(3,12)
 20 PROCINIC
100 INPUT "CUANTOS DISCOS?(1-12)";N
110 IF N < 1 OR N > 12 THEN 100
120 PROCVISUALIZ(N)
130 PROCHANOI(N,1,2,3)
200 END
1000 DEFPROCHANOI(M,PA,PB,PC)
1010 IF M = 1 THEN PROCMOVER(PA,PB):
    ENPROC
1020 PROCHANOI(M-1,PA,PC,PB)
1030 PROCHANOI(1,PA,PB,PC)
1040 PROCHANOI(M-1,PC,PB,PA)
1050 ENDPROC
1100 DEFPROCMOVER(PA,PB)
1110 D$ = D$(P(PA,N + 1 - H(PA)))
1120 FOR I = 24-H(PA) TO 10 STEP -1
1130 PRINT TAB(13*(PA-1),I);B$;
1140 PRINT TAB(13*(PA-1),I-1);D$;
1150 NEXT I
1160 FOR I = 13*(PA-1) TO 13*(PB-1) STEP
    SGN(PB-PA)
1170 PRINT TAB(I,9);D$
1180 NEXT I
1190 FOR I = 9 TO 22-H(PB)
1200 PRINT TAB(13*(PB-1),I);B$
1210 PRINT TAB(13*(PB-1),I + 1);D$;
1220 NEXT I
1240 H(PB) = H(PB) + 1:P(PB,N + 1-H(PB)) =
    P(PA,N + 1-H(PA))
1250 P(PA,N + 1-H(PA)) = 0: H(PA) = H(PA)-1
1260 ENDPROC
3000 DEFPROCINIC
3020 FOR I% = 1 TO 11 STEP 2
3030 D$(I%) = CHR$150 + STRING$(5-I%DIV2," ")
    + CHR$234 + STRING$(2*(I%DIV2),CHR$255)
    + CHR$53 + STRING$(5-I%DIV2," ")
3040 D$(I% + 1) = CHR$150 + STRING$(5 -
    I%DIV2," ") + STRING$(I% + 1,CHR$255) +
    STRING$(5-I%DIV2," ")
3050 NEXT I%
3060 B$ = CHR$150 + STRING$(12," ")
3070 VDU 23,1,0;0;0;0;
3080 ENDPROC
3100 DEFPROCVISUALIZ(N)
3110 CLS
3120 FOR I = 1 TO N
3130 PRINT TAB (0,23-N + I);D$(I);
3135 P(1,I) = I:P(2,I) = 0: P(3,I) = 0
3140 NEXT I
3150 H(1) = N:H(2) = 0:H(3) = 0
3160 ENDPROC
```

Problemas recursivos
Ésta es una fotografía del programa *Las torres de Hanoi* ejecutándose en el Spectrum. El color de los bloques se puede cambiar muy fácilmente. Si intenta seguir el modo en que el ordenador resuelve el problema, observe con gran atención, ¡porque las acciones se suceden con bastante rapidez!



Ian McKinnell

El diseño clave

Aquí estudiaremos cómo utilizar los módulos en el desarrollo de un programa completo

Cuando se construye un programa, es una buena idea desarrollar una estructura global, compuesta de un nivel básico de rutinas de uso general que son empleadas por otras rutinas de creciente especialización a niveles superiores, todo bajo la dirección de un único módulo de control ubicado en la parte superior. Esta estructura "piramidal" nos permitirá utilizar un método de diseño denominado *refinamiento de programas* o *diseño top-down* (de arriba abajo).

El diseño top-down, como su nombre indica, implica diseñar primero el programa de control de más alto nivel. Sus funciones se describen en términos de llamadas a rutinas de nivel "inferior" y, por el momento, no es necesario que nos preocupemos demasiado sobre cómo funcionarán estos módulos de nivel inferior. Después de hacer esto, nos desplazamos un nivel hacia abajo y describimos el funcionamiento de cada una de las rutinas a las que llama el módulo de mayor nivel. Cada rutina se describe en términos de las rutinas que debe llamar, y este proceso se repite nivel a nivel hasta que llegamos al nivel inferior.

En esta etapa, las funciones que realiza la rutina que estamos describiendo son tan simples que se pueden definir utilizando el propio lenguaje de programación.

Como ejemplo, analicemos el diseño del juego *El ahorcado*. En vez de que el jugador trate de adivinar una palabra escogida por el programa, como sucede en la mayoría de las versiones para ordenador que existen de este juego, deseamos que el programa adivine una palabra que hemos elegido nosotros. Una forma de conseguirlo, sin proporcionar al programa una extensa lista de palabras en castellano, consiste en entrar los datos relativos a las probabilidades de aparición de secuencias determinadas de letras.

```
100 REM Inicializar variables y matrices

500 REM ***** Rutina de control **********
510 REM
520 GOSUB 1000: REM Pantallas de títulos y ayudas
530 GOSUB 2000: REM Preparar tablero
540 GOSUB 4000: REM Obtener del jugador la longitud de la
    palabra
550 GOSUB 8000: REM Seleccionar conjunto de datos y cargarlo
560 GOSUB 3000: REM Adivinar una letra
570 GOSUB 4500: REM Verificar conjetura con el jugador
580 GOSUB 5000: REM Actualizar el tablero
590 IF JUEGO—NO—TERMINADO THEN 560: REM Hacer
    nuevas conjeturas hasta que termine el juego
600 IF GANADO THEN GOSUB 10000 ELSE GOSUB 11000: REM
    Dar final adecuado para ganado o perdido
610 GOSUB 6000: REM ofrecer al jugador otra partida
620 IF OTRA THEN 530: REM si otra entonces volver a empezar
630 GOSUB 7000: REM decir adios y parar
640 END
```

Antes de empezar sabemos que se deben hacer varias cosas: es necesario inicializar las variables, se han de dimensionar las matrices, se debe preparar la visualización del "tablero" y actualizarla cuando sea necesario, y se han de escribir rutinas que lleven el marcador, hagan conjeturas y que den por terminado el juego.

Nuestro primer intento por diseñar la rutina de control posee una simple sentencia REM para indicar que se deben inicializar variables y matrices; podemos ir colocando todos los detalles en una etapa posterior. La rutina de control en sí misma es sencillamente un par de bucles. El bucle exterior (línea 620) verifica si el usuario está indicando el fin de una sesión, mientras que el bucle interior (línea 590) verifica si se ha acabado el juego.

Si necesitáramos comprobar la rutina de control, deberíamos preparar subrutinas ficticias para las GOSUB. Cada GOSUB de la rutina de control debe tener una sentencia REM que explique su función y ha de comenzar en un número de línea conveniente, preferiblemente uno que sea una cifra redonda, como 1000 o 5000. Es una buena idea asegurar que todas las rutinas de funciones similares tengan números de línea estandarizados; esto hará que las cosas sean más fáciles cuando se trasladen rutinas de un programa a otro. Por ejemplo, las instrucciones del juego podrían incluirse en una subrutina que empiece en la línea 1000, mientras que una línea de programa GOSUB 7000 puede dar siempre un juego por terminado llamando a una rutina estándar.

Nuestra rutina de control inicial es corta y sencilla. Cabrá en la pantalla y, por lo tanto, será más fácil de comprender y depurar que un programa que ocupe varias pantallas. Las tres variables, JUEGO-NO-TERMINADO, GANADO y OTRA, son todas indicadores o flags que se establecen en las diversas rutinas a las que llama la rutina de control y se utilizan aquí para determinar si el programa de control funciona del modo que nosotros pretendemos. En esta simple rutina de control sería bastante fácil detectar cualquier posible error de lógica.

En esta etapa es preciso analizar la estructura del programa con ojo crítico: necesitamos asegurarnos de que éste se comporte como debe en todas las circunstancias. También podemos comenzar a introducir mejoras en el diseño del programa; por ejemplo, tal vez queramos poder disponer de las instrucciones en cualquier etapa del juego, y también sería una buena idea llevar un registro de cuántas partidas han ganado el ordenador o el jugador y una lista de las palabras que vencen al programa. Cualquiera de estas modificaciones, o todas ellas, se pueden efectuar en esta etapa.

El siguiente paso consiste en especificar cada una de las subrutinas. Nuestros listados muestran el as-

pecto que podrían tener dos de estas subrutinas. La primera (que comienza en la línea 4000) simplemente solicita al usuario un número entre 1 y 20 (la longitud de la palabra). Utiliza una subrutina de uso general que da por sentado existe en la línea 51000, que tomará una serie especificada en PROMPT$, la imprimirá y luego aceptará una entrada numérica hecha por el usuario. Si este número no es un entero comprendido entre los límites establecidos por MIN% y MAX%, generará un mensaje de error y pedirá al usuario que entre un nuevo número. Esta subrutina se podría utilizar fácilmente en otros programas y se podría crear una biblioteca de este tipo de módulos de usos generales para emplearlos en proyectos ulteriores.

```
4000 REM Descubrir longitud de la palabra del jugador
4010 REM
4020 PROMPT$ = "Cuantas letras tiene su palabra?"
4030 MIN% = 1
4040 MAX% = 20
4050 GOSUB 51000: REM entrar un numero entero entre MIN%
     y MAX%
4060 LONGPAL% = RESP%: REM RESP% es usada por la
     subrutina de la 51000 para devolver la respuesta
4070 RETURN
```

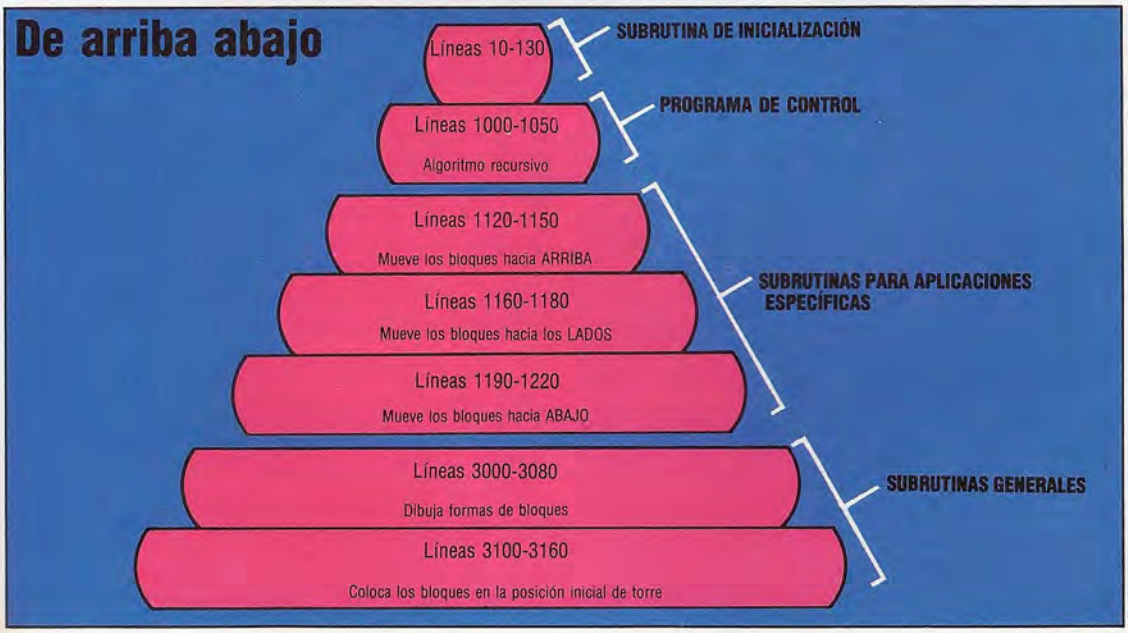
```
8000 REM seleccionar juego datos y cargarlo
8010 REM
8020 IF LONGPAL% > 7 THEN FICH—L% = 8
                     ELSE FICH—L% = LONGPAL%
8030 FICHNO—L$ = STR$(FICH—L%)
8040 NOMFICH$ = "TABLA" + FICHNO—L$
8050 GOSUB 9000:REM OPEN,READ & CLOSE el fichero con
     datos de probabilidad para la longitud de palabra
     adecuada.
8060 RETURN
```

La otra rutina (que empieza en la línea 8000) utiliza variables locales (FICH—L% y FICHNO—L$). Hemos dado por sentado que los datos necesarios para adivinar una letra están en ocho juegos de tablas que dan las probabilidades de hallar una letra determinada junto a cualquier otra. Dado que sólo deseamos tener un juego de datos en RAM en cualquier momento dado, debemos construir una serie en NOMFICH$ para retener el nombre del archivo de datos y después llamar a la subrutina de la línea 9000 para que lea el archivo.

En muchos casos nos encontraremos con que nuestro programa pasa directamente de una rutina a otra. Sin embargo, por lo general será mejor crear una rutina extra que llame a las otras dos sucesivamente. Esto puede parecer una complicación innecesaria, pero nos permite mantener un firme control sobre el "flujo" del programa y ofrece la ventaja adicional de mantener separados los módulos del programa de modo que se los pueda agregar fácilmente a otros programas.

Este uso de las subrutinas que son transportables de un programa a otro implica un trabajo extra y se debe tener cuidado al diseñar las rutinas, de modo que sean aptas para utilizarse en una amplia gama de circunstancias. Con frecuencia esto se puede conseguir simplemente reemplazando constantes por variables. Es importante que todas las subrutinas estén bien documentadas. La documentación ha de especificar el objetivo exacto de la rutina, dando detalles de las variables empleadas, los valores que se esperan como entrada y salida y cualquier posible efecto colateral (desplazar la posición del cursor, cambiar el mapa de memoria, cerrar archivos, etc.).

Un trazado estándar también resulta muy útil; debe asegurarse de que todos los números de línea tengan un intervalo fijado, que los títulos y comentarios se limiten a un grupo de números de línea al comienzo de la rutina y que el RETURN esté siempre en la última línea. Asegúrese de apuntar el primer y último número de línea de cada rutina. Cuando se requiere una rutina de la biblioteca, verifique que el programa tenga un agujero apropiado entre sus números de línea y luego mezcle (MERGE) la subrutina con el programa. Si su micro no posee instrucción MERGE, quizá sea posible utilizar un editor de textos para combinar programas que hayan sido guardados (SAVE) en formato ASCII en vez de en la forma "distintivada" habitual. Si esto no fuera posible, sería necesario digitar las subrutinas de su biblioteca cada vez que las necesite. No obstante, el solo hecho de que no sea necesario volver a diseñarlas bien vale el esfuerzo del trabajo extra.



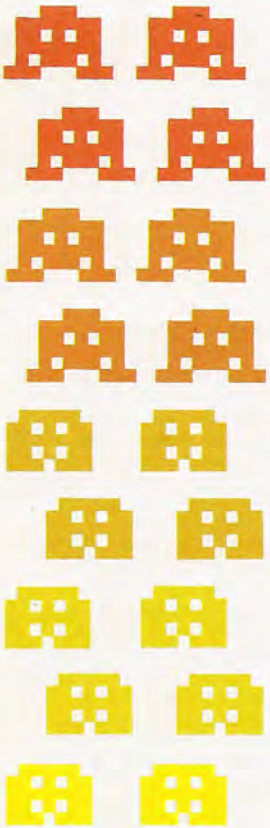
Programación top-down (de arriba abajo)
Este programa ilustra el principio de la programación top-down. Hemos utilizado el programa *Las torres de Hanoi*, que presentamos en la página 955. Los números de línea del diagrama corresponden al listado para el BBC.
El primer estrato de la estructura representa el programa de inicialización, que debe completarse antes de que se pueda ejecutar el resto del programa. El PROGRAMA DE CONTROL de nuestro diagrama representa el algoritmo recursivo, que realiza los cálculos y llama a las otras subrutinas cuando ello es necesario. Las SUBRUTINAS PARA APLICACIONES ESPECIFICAS (de la línea 1120 a la 1220) se utilizan para trasladar las formas de bloques de pila a pila en la visualización. Las dos secciones finales del diagrama, SUBRUTINAS GENERALES, representan las dos últimas secciones del programa que se emplean para formatear la visualización inicial y crear el diseño de los bloques. Compare esta estructura con el listado y verá que el programa está construido exactamente siguiendo esta secuencia

No tan rápido

A veces es necesario que el programa no se ejecute tan aprisa. He aquí uno de los métodos más populares para retardarlo

En el assembly del 6502 se pueden introducir bucles de retardo de diversas formas. El método más simple y natural consiste en cargar uno de los registros índice con un valor y decrementarlo progresivamente hasta que llegue a cero:

BUCLE RETARDO	TIEMPO EMPLEADO EN CADA OPERACIÓN
LDY #$07	Dos ciclos
DEY	Dos ciclos
BNE LOOP	Dos ciclos (3 ciclos si se bifurca a la misma página; 4 ciclos si la página es distinta)



Cada instrucción en lenguaje máquina requiere un determinado número de ciclos de reloj para ser ejecutada. A veces en las descripciones de cómo operan las instrucciones es posible encontrar información sobre tales ciclos. Por ejemplo, la instrucción DEY necesita dos ciclos y LDY también cuando se emplea el modo de direccionamiento inmediato. Dado que cada ciclo tarda un microsegundo (millonésima de segundo), podemos calcular el "tiempo real" empleado en ejecutar el bucle de retardo. El número total de ciclos se calcula así:

1) La instrucción LDY #$07 tarda dos ciclos.
2) El programa da siete vueltas. Por cada vuelta la operación BNE emplea tres ciclos: luego las instrucciones DEY y BNE emplean $(2 + 3) \times 7 = 35$ ciclos.
3) Puesto que el último BNE no provoca la vuelta, sólo emplea dos ciclos.

Número total de ciclos $= 2 + 35 - 1 = 36$. Así pues, el tiempo que consume este retardo es de 36 microsegundos.

Hay varios problemas asociados con el uso de bucles de retardo en código máquina para obtener retrasos "de tiempo real" (o sea, aquellos que pueden ser medidos en segundos o microsegundos con toda precisión). El primero y más importante es que mientras un procesador está ejecutando un programa en código máquina regularmente suspende esta actividad para efectuar servicios requeridos por otras partes del sistema, tales como inspeccionar el teclado, actualizar el reloj interno, etc. Estas discontinuidades en la ejecución del programa son conocidas como "interrupciones". En el chip del 6502 se producen dos tipos de interrupciones: NMI (*Non-Maskable Interrupt*: interrupción no enmascarable) e IRQ (interrupción solicitada). El nombre dado al primer tipo de interrupción implica que nada se puede hacer para evitar (enmascarar) tales interrupciones. Sin embargo, es posible detener las interrupciones IRQ, pues no son esenciales para el funcionamiento del procesador.

Las interrupciones IRQ pueden ser enmascaradas poniendo a 1 un determinado bit del registro indicador de estado. Esto se consigue con la instrucción SEI. Las interrupciones IRQ se vuelven a permitir devolviendo al registro el bit afectado a través de CLI. Enmascarando las interrupciones IRQ antes de entrar en el bucle de retardo, mejoramos su exactitud. Si ocurren interrupciones no enmascarables durante la ejecución del bucle, se producirán errores en el período de retardo. Nuestro bucle inicial presentará este otro listado que evita interrupciones IRQ:

INSTRUCCIÓN	FUNCIÓN	DURACIÓN
SEI	Desactiva IRQ	Dos ciclos
LDY = $07		36 ciclos
DEY		
BNE LOOP		
CLI	Restablece IRQ	Dos ciclos

La tarea de enmascarar las IRQ de esta forma añade cuatro ciclos a la rutina, con lo que su duración será de 40 microsegundos, siempre que no aparezcan interrupciones NMI.

Otro aspecto de los bucles de retardo es el de la "resolución", es decir, cómo varía el tiempo empleado en ejecutar el bucle entre un valor del contador y el siguiente. En nuestra rutina hemos cargado el registro Y con el valor siete, pero si hubiéramos escogido el valor seis, el retardo sería de 35 microsegundos $(2 + 2 + (2 + 3) \times 6 - 1 + 2)$. Si se escogiera el valor cinco duraría 30 microsegundos y así sucesivamente hasta un mínimo de resolución de 5 microsegundos.

Es posible "afinar" nuestro programa (para que el temporizado deje de ser múltiplo de cinco) colocando instrucciones NOP fuera del bucle. La instrucción NOP significa que el procesador no realizará operación alguna (*No OPeration*), lo que hará en dos ciclos. Si deseáramos crear un retardo de 44 microsegundos, por ejemplo, basta con añadir dos instrucciones NOP a nuestro programa antes del bucle (o después):

SEI	Dos ciclos
LDY #$07	Dos ciclos
NOP	Dos ciclos
NOP	Dos ciclos
DEY BNE LOOP	34 ciclos
CLI	Dos ciclos

Este tipo de retardo tiene un tiempo límite determinado por el máximo valor que puede tomar Y. Dado que el registro índice Y es de ocho bits, el valor máximo es 255. Lo que nos da un tiempo máximo de 1 280 microsegundos $(2 + 2 + (2 + 3) \times 255 - 1 + 2)$, aproximadamente una milésima de segundo. A nuestros ojos es un tiempo brevísimo, pero no a los del procesador. A veces son necesarios retardos de mayor duración. Lograremos alargarla agregando instrucciones NOP *dentro* del bucle. Si, por ejemplo, incluimos una instrucción NOP en el bucle el retardo máximo pasa a ser de 1 790 microsegundos $(2 + 2 + (2 + 2 + 3) \times 255 - 1 + 2)$.

Para demoras bastante mayores que éstas deberemos diseñar algún otro método. Dos de los modos más frecuentes de producir largos retardos consisten en utilizar un segundo bucle anidado al primero, o bien producir decrementos en un valor numérico mucho mayor, por ejemplo uno de 16 bits logrado mediante dos bytes de la memoria. Cada uno de estos procedimientos tiene su período de resolución que es posible calcular.

Contador de bucle anidado

```
DELAY SEI
      LDX #$04    ;emplea registro X como contador bucle exterior
LOOP1 LDY #$FF    ;emplea registro Y como contador bucle interior
LOOP2 DEY         ;salida del bucle interior
      BNE LOOP2
      DEX
      BNE LOOP1   ;salida del bucle exterior
      CLI
```

El bucle interior del programa emplea 1 276 microsegundos $(2 + (2 + 3) \times 255 - 1)$ en su ejecución. El bucle exterior controla la ejecución del interior y realiza DEX y BNE cuatro veces. El tiempo total de este retardo es calculable: $2 + 2 + (1\,276 + 2 + 3) \times 4 - 1 + 2 = 5\,129$ microsegundos.

Retardos en el Z80

Cada instrucción en código máquina en el Z80 tarda un tiempo diferente en ser ejecutada (medido en unidades llamadas *estados T*). Además, el Z80 va a diferentes velocidades según las máquinas. Para calcular el tiempo real empleado por cada instrucción se divide el número de estados T de cada instrucción por la frecuencia de reloj del micro. Por ejemplo, una instrucción que emplee cuatro estados T en ser ejecutada por un procesador con 2 MHz de frecuencia de reloj tardará en ejecutarse dos microsegundos.

El libro de Rodnay Zak *Programming the Z80 (La programación del Z80)* contiene las duraciones de todas las instrucciones del Z80. Éstas son las velocidades de reloj de la CPU de las máquinas más populares que incorporan un Z80; el ZX81 (3,25 MHz), el Spectrum (3,5 MHz), el Tandy TRS80/Video Genie (1,7 MHz) y el Amstrad (4 MHz).

Podemos usar la instrucción NOP si deseamos un retardo de muy corta duración. En un micro de 2 MHz esta instrucción producirá un retardo de dos microsegundos. Podemos usarlas sucesivamente hasta un cierto número, pero los retardos de mayor duración se consiguen con rutinas ficticias; por ejemplo, la siguiente rutina produce un retardo de 27 estados T:

```
CALL DELAY
RET
```

En este ejemplo, la instrucción CALL emplea 17 estados T, y la instrucción RET emplea 10. El retardo que se obtendrá en un procesador de 2 MHz será de 13,5 microsegundos. Un retardo algo mayor se obtiene incluyendo instrucciones NOP al comienzo de la rutina.

Para retardos mayores hay que servirse de un bucle. En el siguiente ejemplo se carga un registro con el valor que irá decrementándose dentro del bucle. La rutina proporciona un retardo de 99 estados T (49,5 microsegundos a 2 MHz).

INSTRUCCIÓN	TIEMPO (EN ESTADOS T)
CALL DELAY	17
(BUCLE DE RETARDO)	
LD B,5	7
DEC B	4
JR NZ,LOOP	12 (o bien 7, si verdadero)

Las tres instrucciones que comienzan con LD B,5 son el bucle de retardo en sí. Como ocurre con la rutina en código máquina para un 6502, la duración total de esta rutina varía según el valor que se introduce en el registro. El número total de ciclos de reloj que emplea puede ser expresado así:

$$C = 24 + (N \times 16) - 5$$

siendo N el valor cargado en el registro B.

También se pueden utilizar bucles anidados. Pero debemos tener en cuenta otras consideraciones. Primero, que todo registro empleado durante la rutina debe antes "salvarse" *(pushed)* para no perder su contenido. Segundo, que algunas máquinas tienen interrupciones propias de su hardware que pueden alterar el período de retardo. Las interrupciones enmascarables se pueden desactivar y reactivar mediante las instrucciones DI y EI. La siguiente rutina emplea bucles anidados:

INSTRUCCIÓN	FUNCIÓN	TIEMPO (EST. T)
DI	Desactiva interrupciones	4
PUSH DE	Salva el contenido del reg.	11
LD D,nn	Valor del contador interior	7
LD E,nn	Valor del contador exterior	7
CALL OLOOP	Salto al contador exterior	17
POP DE	Restaura el contenido	10
EI	Reactiva interrupciones	4
:		
(OLOOP)		
DEC E	Decremento del bucle exterior	4
RET Z	Final si es cero	11 o 5
(ILOOP)		
DEC D	Decremento del bucle interior	4
JP Z,OLOOP	Bifurca si D es cero	10
JP ILOOP	Si no, continúa bucle interior	10

En esta rutina el retardo aumenta si aumenta el valor del registro E. La rutina acabará al decrementar el registro E y obtener un valor de cero. Observe que si el bucle interior llega hasta cero y el exterior todavía tiene una valor mayor que uno, el bucle interior será inicializado a 255 e irá contando hasta cero antes de pasar el control al bucle exterior.

Invasión cronometrada
Los retardos en código máquina son necesarios en programas de juegos, y en particular cuando hay un objeto que se mueve en la pantalla y ha de ser manipulado por el jugador. Un ejemplo ya clásico es el juego de los *Invasores del espacio*. Sin retardos cronometrados el movimiento de los extraterrestres invasores sería excesivamente rápido. Mientras que gracias a los retardos cuidadosamente cronometrados el movimiento se controla según sea necesario para que el juego discurra debidamente

Conexiones a todo color

Prism es una empresa en constante expansión desde que se iniciara como distribuidora de los productos Sinclair

El presidente de la empresa
Richard Heath, presidente de Prism, que la fundó como empresa subsidiaria de ECC Publications

Mientras la mayoría de las empresas de la industria del ordenador personal se contentan con una perspectiva a corto plazo del mercado, satisfaciendo la demanda inmediata de hardware y software, Prism piensa en el futuro. Una de las más importantes distribuidoras de hardware, Prism jugó un papel fundamental en el desarrollo del Micronet (la primera base de datos a gran escala que se puso a disposición de los usuarios de máquinas personales) y en la actualidad ha tomado a su cargo la distribución de robots de bajo coste.

La empresa la creó ECC Publications en 1982 para desarrollar el Micronet, bajo la dirección de Richard Heath y Bob Denton. El Micronet utiliza el Prestel, uno de los mayores sistemas públicos de videotexto, para posibilitar a los usuarios de una amplia gama de ordenadores personales cargar software, acceder a información e intercambiar "correo electrónico" (véase p. 581). ECC Publications ya había lanzado la revista *Sinclair User*, aunque en aquel entonces los productos Sinclair sólo se vendían por correspondencia o a través de la cadena minorista W. H. Smith. *Sinclair User* alcanzó un enorme éxito, a pesar del escepticismo inicial de Terry Cartwright, actual director de marketing de Prism. "Yo creía que Sinclair no era más que flor de un día —admite—, pero acudimos a la primera Feria del Micro ZX con 8 000 solicitudes de suscripción y las repartimos todas en unas horas."

Sinclair decidió introducirse en el mercado minorista de las tiendas comerciales céntricas y Prism firmó a tiempo un contrato para distribuir el ZX81 y el Spectrum, recientemente lanzado. De hecho, el nombre de la empresa se eligió de forma deliberada para inducir en la mente del público una asociación con el Spectrum; al fin y al cabo, si uno dirige un haz de luz a través de un prisma, ¡acaba con un espectro de colores! Recientemente Prism ha declarado haber vendido más de 500 000 máquinas Sinclair: alrededor del 25 % de todas las ventas de ordenadores personales realizadas en Gran Bretaña.

En marzo de 1983 Prism lanzó el Micronet, en sociedad con la British Telecom y Telemap. Prism se encargó del hardware, distribuyendo una gama de modems (fabricados por O E Ltd y Thorn EMI) para las máquinas más populares. La contribución más reciente a esta gama fue un modem para utilizar con el Commodore 64.

En la actualidad el Micronet tiene alrededor de 10 000 suscriptores, pero recientemente Prism ha vendido su participación en la red y ahora se está concentrando en comercializar y distribuir hardware para ordenadores. Además del Spectrum, en la actualidad Prism se encarga de comercializar las máquinas Oric y Atmos, y la máquina de oficina portátil Wren ("reyezuelo", en castellano), "marca de la casa".

Prism también se está introduciendo en un área nueva: la distribución de robots personales. Éste es un campo que está despertando un creciente interés y ahora Prism comercializa el "Topo", un robot importado de Estados Unidos, así como diversos kits de robot económicos que se venden bajo el nombre comercial "Movits".

Terry Cartwright considera que los robots son un área de enorme expansión. "Existe un inmenso interés por los robots —afirma—. No sé lo que la gente hará con ellos, pero en 1976 nadie sabía tampoco lo que sucedería con los ordenadores Apple." Asimismo, la empresa pretende empezar la distribución del Sinclair QL a finales de este año. Cartwright espera que continúe la diversificación de Prism en el futuro. "La expansión a nivel exterior será la principal prioridad para los próximos doce meses", afirma.

Bonos de suscripción
La empresa tiene planeado alquilarles estos modems a los clientes, junto con una suscripción de un año al Prestel y al Micronet

Pájaro madrugador
El Wren (reyezuelo), la máquina de oficina portátil de Prism, basada en el Z80, viene equipado con dos unidades de disco y un modem incorporado para conectarlo al Prestel

mi COMPUTER