mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

TOMO 9

Editorial Delta, S.A.


mi COMPUTER

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

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

Realización gráfica: Luis F. Balaguer

ISBN: 84-7598-181-X (tomo 9) 84-85822-82-X (obra completa)
Depósito legal: B. 52/1984
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona)
Impreso en España - Printed in Spain - 1985

mi COMPUTER

VOLUMEN 9

Editorial Delta, S.A.

Bajo sospecha

La programación de juegos de aventuras alcanza cotas de notable refinamiento al incluir personajes interactivos

El software que parece dotar de una «personalidad» al ordenador, mediante el análisis de las entradas del usuario en su lenguaje natural y la producción de respuestas aparentemente llenas de significado, logra siempre generar interés y emoción. Cualesquiera que sean las pautas psicológicas implicadas, programas como ELIZA permiten al usuario un nivel de participación que no guarda relación en absoluto con su relativa simplicidad y, desde el punto de vista del programador, existen pocos desafíos más satisfactorios que el de programar un ordenador para que «escuche» y «responda» como un compañero humano.

Ya hemos desarrollado un programa tipo ELIZA muy sencillo (*Consulta particular*) y, además analizado los principios del proceso del lenguaje natural a lo largo de nuestra serie dedicada a la inteligencia artificial. Ahora examinaremos otra aplicación de estos principios: la programación de «personajes interactivos» en los juegos de aventuras. Analizaremos el papel que pueden jugar tales personajes en estos juegos y luego programaremos nuestro propio manipulador de personajes, el cual se podrá ejecutar como módulo individual, o bien adaptar para ejecutarlo junto con nuestros programas *El bosque encantado* y *Digitaya*.

El nivel de sofisticación del software de aventuras ha ido en aumento desde que Crowther y Woods desarrollaran *Colossal cave* (Caverna colosal) en FORTRAN y en un ordenador de unidad principal, utilizando aproximadamente 170 Kbytes de código. En la actualidad, programas como *The hitchhiker's guide to the galaxy* (Guía de la galaxia para el autostopista) no sólo aceptan entradas complejas sino que también cuentan con personajes que actúan «espontáneamente» y a los que el jugador puede dirigirles la palabra (e incluso, en algunos casos, impartirles instrucciones).

La inclusión de personajes en la *ficción interactiva* (tal como se está empezando a llamar al software de aventuras) les ha permitido a los programadores alejarse del ambiente tradicional de aventuras, que se centraba casi exclusivamente en el proceso de explorar distintos escenarios, recoger objetos y resolver el ocasional enigma. Los personajes interactivos introducen nuevas posibilidades; en primer lugar, la presentación de situaciones que de hecho el jugador no puede resolver sin la ayuda de otro personaje controlado por el ordenador; en segundo lugar, y aun más importante, la presentación de un juego que puede resultar radicalmente diferente cada vez que se juega con él.

El dedo acusador lo señala a usted
Una reina de cuento de hadas, un hombre lobo, un vampiro y muchos otros personajes lo esperan a usted en *Suspect* (Sospechoso), un juego de aventuras de Infocom que incorpora personajes interactivos y vívidas descripciones de escenarios. Usted, invitado a un baile de disfraces de la alta sociedad, tropieza con la escena descrita arriba y descubre, con espanto, que su anfitriona ha sido estrangulada con un lazo que alguien le ha quitado de su disfraz de vaquero. Para salvar su reputación, usted habrá de interactuar de forma a la vez intensa y sutil con el resto de los asistentes al baile

Antes de seguir adelante es necesario que definamos exactamente qué es un personaje de estas características. Un personaje interactivo ha de poseer al menos uno de los siguientes atributos:

- Debe poder desplazarse de un escenario a otro.
- Debe poder recoger y abandonar objetos sin que el jugador tenga que instruirlo en ese sentido.
- Debe poder ser aludido por el jugador y responder de forma significativa (aunque sólo replique «No te comprendo»).
- Debe estar capacitado para dirigirse al jugador sin que éste así se lo indique.
- Debe ser consciente de su entorno y sensible al mismo.

A nivel ideal, el personaje interactivo ha de combinar todos estos atributos. Quizá el mejor ejemplo de los inicios de la programación de personajes interactivos sea *El Hobbit*, una aventura de texto y gráficos de Melbourne House. Este juego incorporaba numerosos personajes, incluyendo, de forma destacable, a Thorin el Enano y Gandalf el Mago. El jugador podía dirigirse a estos personajes mediante la instrucción SAY TO (decir a), como en SAY TO THORIN «GO NORTH» (decir a Thorin «avanzar norte»). El programa empleaba entonces una rutina activada por números aleatorios que decidía si el personaje había de obedecer o no. No obstante, los

personajes de *El Hobbit* eran, para los estándares actuales, comparativamente primitivos: al impartirles una orden con frecuencia el resultado era: THORIN SAYS NO (Thorin dice no) o una respuesta similar. Además, Gandalf y sus compañeros no hablaban particularmente bien, o de hecho no hablaban en absoluto. Thorin, por ejemplo, parece estar limitado a decir HURRY UP (date prisa) si el jugador se demora demasiado tiempo entre las entradas, y tiene la costumbre de «sentarse a pensar en los peces de colores» con monótona frecuencia.

Quizá lo más significativo sea el hecho de que los personajes de *El Hobbit* no poseen ninguna historia personal, es decir que si uno efectúa alguna acción determinada que pueda incidir en su relación con ellos, los personajes posteriormente no la «recordarán». Entre la mayoría de las primeras aventuras es típico que cada personaje esté programado para comportarse de un cierto modo a lo largo del juego y, en gran medida, independientemente de las circunstancias. De modo tal que Thorin, por ejemplo, podría «sentarse a pensar en los peces de colores» en el momento en que se le aproxima un dragón.

En *Valhalla*, programa que obtuvo en Gran Bretaña el premio Microcomputer Game of the Year en su edición de 1984, se efectuó un primer intento por hacer a los personajes dueños de una «historia»; al otorgársele la distinción se premió tanto la incorporación de esta facilidad como su visualización animada. En este juego, cuya acción tiene lugar entre dioses y héroes de la mitología escandinava, uno se encuentra con que los personajes «buenos» siempre se muestran dispuestos a obedecer las instrucciones del jugador (y viceversa) si el objetivo principal de éste es el de acabar con todos los personajes malvados.

La mansión del horror
Suspect se desarrolla en la exclusiva mansión Ashcroft, reflejada en la ilustración. Algunas de las habitaciones más grandes aparecen en el juego como varios escenarios diferentes; por ejemplo, el Salón de Baile junto a la Orquesta, el Salón de Baile junto al Hogar y el Recibidor Grande del Norte. El mapa ilustra la posición de algunos de los protagonistas inmediatamente después del asesinato de Verónica Ashcroft, perpetrado en el despacho

Clave:
1. Arlequín, mirando la televisión
2. Reina de las Hadas (muerta)
3. Gorila (el mayordomo, Smythe)
4. Jeque, esposo de la Reina de las Hadas
5. Muchacha del Harén, bailando
6. Astronauta, dirigiéndose al oeste
7. Barman
8. Vampiro, saliendo del salón de baile
9. Jugador, disfrazado de vaquero

Ello ofreció, entre otras cosas, la interesante posibilidad de personajes susceptibles de someterse a «lavados de cerebro». Por ejemplo, el dios escandinavo Loki inicialmente es un ente maligno, pero uno puede congraciarse con él provocando peleas con, supongamos, Thor (un dios virtuoso) y ordenándole después a Loki que realice buenas acciones, como enzarzarse en peleas con quienes en otro tiempo eran sus siniestros compañeros. Lamentablemente, a pesar de la complejidad del aspecto «histórico» de los personajes de *Valhalla*, los otros atributos de las personalidades participantes se limitan en su mayor parte, a comer, luchar y beber.

Veamos ahora una aventura interactiva muchísimo más avanzada: *Suspect* (Sospechoso), de Infocom (productora asimismo de *The hitchhiker's...*). *Suspect* constituye un ejemplo perfecto del juego en el cual los personajes tienen suma importancia; de hecho, el éxito o el fracaso de éste depende exclusivamente de observar, interrogar, examinar y finalmente acusar a otras «personas» que se hallan bajo el control del ordenador. En consecuencia, y debido a los elementos aleatorios implicados, el juego se puede practicar de numerosas formas.

«Suspect»

La trama de *Suspect* se desarrolla en una mansión campestre (véase el mapa) e incluye al menos 12 protagonistas. La acción transcurre entre las nueve en punto de la noche y las primeras horas del día siguiente, tiempo durante el cual los Ashcrofts (millonarios norteamericanos de la alta sociedad) celebran su baile anual de disfraces. En el transcurso del juego usted descubre (si se molesta en echar una mirada por la casa) que Verónica Ashcroft ha sido estrangulada con el lazo que formaba parte del disfraz de vaquero que usted llevaba. Las sospechas, por consiguiente, recaen ineludiblemente sobre sus hombros y, a menos que descubra al verdadero culpable, pronto se verá entre rejas.

A lo largo del juego, los personajes deambulan por la casa, hablando entre ellos y viviendo su propia existencia. El programa le informa puntualmente a usted acerca de los entornos inmediatos de cada uno. Nuestro mapa muestra las posiciones que ocupan algunos de los personajes en un momento determinado del juego en el cual usted, de pie en el salón de baile junto a la orquesta, recibe la siguiente descripción en pantalla:

> **Salón de baile, junto a la orquesta**
> Éste es el extremo norte del salón de baile. Una superficie elevada constituye una plataforma para la orquesta, y hay una red de circuitos estéreo para usar cuando no hay orquesta. En otras partes de este gran salón de baile hay una multitud de otros invitados a la fiesta vestidos con toda clase de extravagantes indumentarias. En la pista de baile se hallan algunos de los bailarines de mayor edad. La orquesta está tocando el «*Tennessee waltz*». En la periferia de la habitación se observan pequeños grupos que charlan acerca de toda clase de temas, desde política hasta los escándalos locales. Michael se halla junto a la entrada norte. Alicia está en la pista de baile. Ostmann se encuentra junto a la entrada sur. Junto al hogar está el Astronauta. Johnson está en el bar. La orquesta está aquí, haciendo su juego. Ahora el Astronauta se halla junto a la entrada sur.

Observará que en la descripción se menciona dos veces al Astronauta: primero se halla junto al hogar y luego junto a la entrada sur, de modo que usted puede deducir que probablemente en ese momento esté marchándose del salón de baile. Las posiciones de otros personajes, como Smythe, el mayordomo, por supuesto no se mencionan; éstos se hallan en otras habitaciones y usted habrá de buscarlos para ser notificado de su presencia.

Jugar a un juego como *Suspect* constituye una experiencia apasionante, y la presencia de personajes interactivos le añade muchas dimensiones nuevas a un área de la programación de juegos cuya popularidad ya es notable. En el próximo capítulo examinaremos otros ejemplos de la interacción de personajes en el juego y daremos los primeros pasos para escribir nuestras propias rutinas y conseguir efectos similares.

El más adecuado

Veamos cuáles son los lenguajes de programación más apropiados para las aplicaciones de inteligencia artificial (AI)

La mayor parte de los investigadores en el campo de la inteligencia artificial (AI) emplean LISP o PROLOG. Por este motivo, estos dos lenguajes de programación han llegado a conocerse como *lenguajes de AI*. Existen justificadas razones por las cuales quienes trabajan en AI escojan lenguajes que reúnan ciertas características, pero la frase «lenguaje de AI» puede prestarse a confusión, por dos motivos. En primer lugar, sugiere que el LISP y el PROLOG pueden ser inadecuados para el procesamiento convencional de datos. En segundo lugar, y aún más importante, conduce a la suposición de que con el mero hecho de escribir un programa en LISP o en PROLOG uno ya se está introduciendo en el campo de la AI. Sugiere, asimismo, que otros lenguajes para procesamiento de símbolos, como pueden ser el POP-11 y el SNOBOL4, por nombrar sólo dos, son inadecuados para el trabajo serio en AI.

El LISP es un «veterano» entre los lenguajes de programación. Fue desarrollado inicialmente por John McCarthy en el MIT (Massachusetts Institute of Technology) a finales de los cincuenta y, en consecuencia, es tan antiguo como el COBOL. El lenguaje quedó estructurado en 1961, y desde entonces casi no ha experimentado modificaciones. Ya hemos ofrecido una breve serie de introducción al LISP, pero hay un concepto que no ha sido mencionado hasta ahora y que es de gran importancia para las aplicaciones de AI: la *lista de propiedades*.

Cada átomo posee su propia lista de propiedades, que se compone de pares de *valores de atributos*. La lista de propiedades es, en realidad, una descripción del átomo, y proporciona fácil acceso a una clase de estructura de base de datos. Las funciones GET y PUTPROP se utilizan para manipular listas de propiedades. Por ejemplo:

```
(PUTPROP 'AYER 0.1 'PRECIPITACIONES)
(PUTPROP 'MAÑANA 'NUBLADO
'PROBABILIDAD)
```

tiene el efecto de insertar 0.1 como el valor del atributo PRECIPITACIONES del átomo AYER, y de poner NUBLADO como el valor del atributo PROBABILIDAD para el átomo MAÑANA. Los atributos PRECIPITACIONES y PROBABILIDAD son como campos de un registro de base de datos. El valor de cualquier atributo dado se puede recuperar mediante el empleo de GET. Por ejemplo:

```
(SETQ LLUVIA (GET 'AYER 'PRECIPITACIONES)
```

obtendría el valor PRECIPITACIONES de la lista de propiedades de AYER y lo asignaría como nuevo valor para el átomo LLUVIA. (En realidad, el valor de un átomo no es más que un tipo especial de propiedad que está disponible sin necesidad de utilizar GET y PUTPROP.)

Las listas de propiedades se pueden emplear para construir estructuras de datos complejas y flexibles, y se han utilizado para modelar el funcionamiento de la memoria humana.

El PROLOG constituye la principal alternativa al LISP para la programación en AI. En Europa está alcanzando mayor difusión que el LISP, si bien éste aún prevalece en Estados Unidos. Debido a que el PROLOG es un lenguaje declarativo, permite que el programador especifique hechos y reglas acerca de objetos y relaciones. Estos hechos y reglas se pueden entonces utilizar para responder preguntas acerca de los objetos y las relaciones implicados.

En pocas palabras, el PROLOG responde preguntas mediante el uso de un método de provisión de teoremas basado en el *principio de resolución*, que niega la proposición a demostrar e intenta refutarla: una versión sofisticada de la técnica de la *reductio ad absurdum*. Pero en cuanto concierne al usuario, sólo está buscando en una base de datos.

El PROLOG, al igual que el LISP, posee facilidades para construir complejas estructuras de datos (incluyendo listas) y para entrada/salida, cálculos, ma-

Cortesía de Artificial Intelligence Ltd.

Entorno de inteligencia artificial
La red INTERLISP-D de Rank Xerox conforma un entorno de programación completo para investigadores de AI, incluyendo gráficos interactivos, herramientas para depuración y grandes facilidades para almacenamiento en línea en forma de una unidad de disco de 29 megabytes. El sistema de programación del conocimiento LOOPS, utilizado en conjunción con INTERLISP-D, permite al programador de AI seleccionar una combinación de diferentes estilos de programación, tales como una codificación orientada hacia el objeto o una orientada hacia la regla, para construir sistemas de conocimiento de bajo costo

nipulación de archivos, etc. En muchos sentidos no logra responder a los ideales de la programación lógica, pero aun así es potente y muy flexible.

No obstante, el LISP y el PROLOG no son las únicas herramientas de software adecuadas para la programación de AI. El POP-11 (especialmente en el entorno PROPLOG desarrollado en la británica Universidad de Sussex) también es un candidato para los trabajos de AI, y para programar AI es bastante posible utilizar lenguajes tradicionales tales como el C y el PASCAL (o incluso el BASIC). Ello, sin embargo, exige un mayor esfuerzo.

En razón de la complejidad de los problemas de AI, todo aquello que contribuya a facilitar la programación es siempre bien recibido. El tipo correcto de lenguaje es de gran ayuda, por supuesto, pero también cuentan otros muchos factores. Una tendencia moderna que se mantendrá sin ninguna duda es el uso de entornos de AI ejecutados en centros de trabajo de inteligencia artificial.

Ejemplos de tales entornos son el LOOPS y el POPLOG. LOOPS (LISP *Object-Oriented Programming System*: sistema de programación en LISP orientado hacia el objeto) se desarrolló en el Centro de Investigación Xerox de Palo Alto y se basa en un dialecto de este lenguaje denominado INTERLISP-D, pero es mucho más que un sistema de LISP. Proporciona múltiples herramientas de software, algunas convencionales (tales como administración de ventanas e iconos) y otras menos convencionales (como procedimientos de inferencia preempaquetados para uso en sistemas expertos). Todo el sistema opera en un centro de trabajo de AI específico que contiene un procesador optimizado para ejecutar LISP. El POPLOG se ejecuta en la gama de ordenadores VAX, de modo que no requiere hardware especializado; pero desde el punto de vista del software, se trata de una idea similar al LOOPS. Proporciona edición en pantalla y otros útiles paquetes estándares, así como la capacidad de llamar ya sea al LISP o al PROLOG desde POP-11 (su lenguaje principal), ofreciendo de este modo una positiva síntesis de ambos. Además, existe toda una biblioteca de emparejamiento de patrones y otros procedimientos elaborados específicamente para aplicaciones de AI.

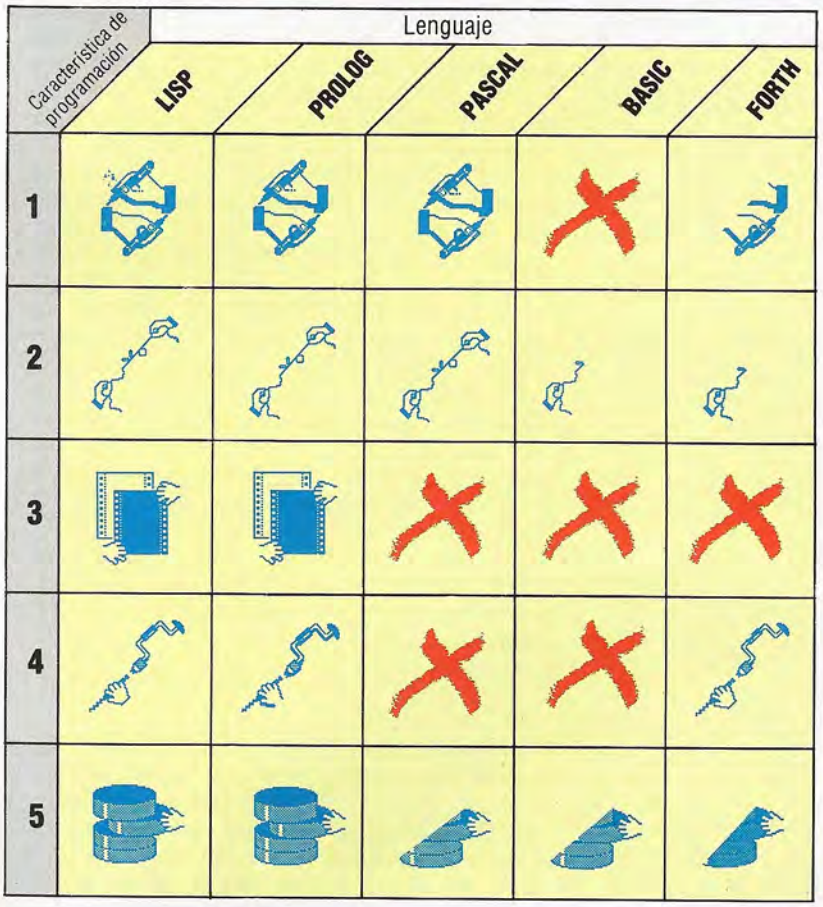
Por encima de todo, lo que ofrecen tales sistemas es un entorno de software muy favorable para la productividad del programador. Su principal desventaja es el aislamiento: se pueden desarrollar aplicaciones inteligentes con mucha rapidez, pero no se pueden ejecutar en el ordenador personal medio.

Al comenzar la programación, es muy raro que el problema de AI se comprenda en su totalidad. En realidad, los investigadores de AI con frecuencia escriben programas como un medio para obtener una visión más profunda de sus problemas. Por consiguiente, en la programación de AI es de capital importancia la *elaboración rápida de prototipos*, estilo de programación en el cual los sistemas evolucionan de forma gradual mediante muchas pequeñas adiciones y alteraciones. Las herramientas de software tradicionales y las metodologías de ingeniería de software no son adecuadas para satisfacer las exigencias de la programación de AI, la cual, como sabemos ahora, es una tarea difícil.

Aptas para el trabajo

Con el correr de los años se han desarrollado numerosas herramientas para facilitar la tarea del programador de AI; éstas poseen algunas importantes características en común.

1. El lenguaje debe soportar repetición.
2. El lenguaje debe disponer de buenas facilidades para manipulación de series y de símbolos, permitiendo la construcción de estructuras de datos flexibles de complejidad arbitraria.
3. Es importante que exista uniformidad de programa y datos: tanto el LISP como el PROLOG representan programas y datos con el mismo formato.
4. La sintaxis ha de ser ampliable: como en el LISP y en el PROLOG, debe ser posible la construcción de un nuevo lenguaje además del original.
5. El acceso a una base de datos incorporada de alguna clase es crucial: el PROLOG posee en su núcleo lo que representa una base de datos relacional, pero el LISP (con listas de propiedades) también ofrece tal facilidad. Además, en los trabajos de AI se necesitan todas las otras herramientas de trabajo que contribuyen a la simplificación de la programación (editores, rutinas para gráficos, verificadores de sintaxis, depuradores, compiladores, productores de documentación, etc.), al menos en la misma medida que en otras áreas de la informática. El gráfico refleja la distribución de estas características entre cuatro diferentes lenguajes de alto nivel



Ian McKinnell

Mundo de palabras

En FORTH, los programas se construyen a partir de palabras que puede definir el mismo programador

Es agradable que todas las instrucciones se utilicen de la misma manera, tanto si están incorporadas en el lenguaje como si se definen luego como parte de un programa. De modo que el FORTH tiene todas sus instrucciones incluidas en un diccionario. Al igual que un diccionario normal, éste consiste en una lista de palabras y sus definiciones, si bien no se almacenan por orden alfabético, sino por el orden según el cual se van definiendo. Cada definición le dice al ordenador exactamente qué hacer cuando se usa esa palabra, ya sea porque se ha digitado directamente (de forma interactiva) o bien porque forma parte de la definición de otro vocablo que se esté utilizando.

En FORTH, todo lo que el sistema reconozca es una palabra, con una definición en el diccionario. Las únicas excepciones a esta norma son los números, que se reconocen como tales de la forma habitual. Algunas palabras se definen como rutinas (con dos puntos o un punto y coma como parámetros), mientras que otras se definen como variables o constantes; pero todas se almacenan en el mismo diccionario. Una palabra puede, en consecuencia, desempeñar varios papeles diferentes, tal como veremos a continuación.

Las instrucciones incorporadas en el sistema son palabras. Por ejemplo, la instrucción ORDS (muchos sistemas utilizan, en cambio, el VLIST, abreviación de *Vocabulary* LIST, lista de vocabulario) visualiza una lista de palabras del diccionario. En algún lugar de la lista puede verse el vocablo WORDS (palabras). Las instrucciones que se le proporcionen en un FORTH ampliado también son palabras, como GRADOS y ELEVACION en el FORTH para telescopios que ya hemos analizado, o como FORWARD y RIGHT en el FORTH para tortugas.

Las instrucciones nuevas que añada el usuario son palabras, como ESTACIONAR y CUADRADO, y se definen entre un signo de dos puntos y un punto y coma, como en:

```
:ESTACIONAR 0 GRADOS ACIMUT 90 GRADOS
ELEVACION;
```

Como puede observar, primero va el signo :, luego la palabra que se esté definiendo, luego la definición y por último el ;. Cuando todo esté incluido en una misma línea, es muy importante asegurarse de que todos los componentes estén separados por espacios. Por otra parte, pueden dividirse en varias líneas, lo que suele resultar más legible:

```
:ESTACIONAR
  0 GRADOS ACIMUT
  90 GRADOS ELEVACION
;
```

Todo el programa es una palabra. De hecho, el FORTH realmente no piensa que nada sea *el* programa. Si las definiciones componen una única instrucción grande (que podría llamarse RUN si así se deseara), entonces podría pensar en ella como si fuera el programa, utilizando las otras palabras como subrutinas; pero no existe razón alguna por la cual no puedan tenerse en el diccionario otras instrucciones/programas independientes al mismo tiempo, con nombres diferentes. Las subrutinas también son palabras. No son otra cosa que instrucciones que se emplean en otra definición.

El FORTH no hace distinción entre instrucciones, programas y subrutinas. Todas ellas se definen utilizando : y ;, y todas se pueden emplear ya sea directamente desde el teclado o bien indirectamente desde otra definición. Debido a que estas definiciones utilizan un signo de dos puntos, se las llama *definiciones de dos puntos*.

Las variables también son palabras. Usted declara una variable (LONGITUD, p. ej.) digitando:

```
VARIABLE LONGITUD
```

La palabra LONGITUD posee, entonces, una definición en el diccionario que incluye algo de espacio de memoria para su valor. Usted debe declararla de este modo antes de poder utilizarla. Si no lo hace, simplemente no habría ninguna definición de ella en el diccionario y no se la reconocería como variable. Puede utilizar variables con los caracteres @ (que significa *traer*) y ! (*almacenar*):

```
LONGITUD @
```

le da el valor de LONGITUD, y:

```
26 LONGITUD !
```

establece su valor en 26 (como en LET LONGITUD = 26 en BASIC).

Es fácil olvidarse del @, pero usted debe utilizarlo para obtener el valor de la variable. Sin él obtendrá la *dirección* de la variable en lugar del valor que podría tener asignado.

Las constantes también pueden ser palabras,

Números como palabras

En la definición:

```
0 CONSTANT 0
```

0 previamente era un número que se podía reconocer en función de la regla 2. A partir de ahora tendrá una definición en el diccionario y, por tanto, será reconocido en función de la regla 1 antes de ser examinado por la regla 2. Al estar definido como el número 0, no hay ninguna diferencia obvia, pero es posible que en el FORTH suyo sea más eficaz de esta forma; de hecho, el figFORTH define al 0, 1 y 2 en el diccionario por la enorme frecuencia con la que se utilizan. Una definición más equívoca sería:

```
12 CONSTANT 13
```

de modo que cuando usted digite 13, en realidad obtendrá 12. (Sin embargo, podría seguir obteniendo el auténtico 13 digitando 013.)

como parece natural. Si emplea repetidamente un número con un cierto significado, quizá prefiera otorgarle un nombre significativo definiendo una palabra como constante. Por ejemplo:

```
66 CONSTANT CLIQUITICLIC
```

A partir de entonces la palabra CLIQUITICLIC significa simplemente 66. Por supuesto, podría haber utilizado en cambio una variable, pero la ventaja de una constante es que no se necesita el @. Evidentemente, con ella usted tampoco puede emplear la instrucción de almacenar (!).

Observe que no puede utilizar declaraciones VARIABLE y CONSTANT dentro de una definición de dos puntos: más adelante verá el motivo cuando analicemos cómo se almacenan las definiciones en el diccionario. Sin embargo, resulta tentador intentar definir palabras como:

```
:INICIALIZAR
  VARIABLE MARCADOR   0 MARCADOR!
  VARIABLE METAS   0 METAS!
;
```

pero no funcionará. Otra consecuencia de esto es que todas las variables son *globales*, como en BASIC. Las variables locales, como las que encontrará en PASCAL, no existen en FORTH.

El FORTH incluye +, –, *, / y muchos otros operadores aritméticos. Como todas las palabras, deben ir separados de otras palabras mediante espacios, y esperan hallar sus argumentos ya allí. De modo que se escribe:

```
2 3 +
```

en lugar de 2 + 3. Este punto lo explicaremos en mayor profundidad en el próximo capítulo, con el funcionamiento de la «pila».

Los dos puntos, VARIABLE y CONSTANT son inusuales porque introducen nuevas definiciones en el diccionario; por este motivo se las denomina *palabras definitorias*. Pero, exceptuando esto, continúan siendo palabras del diccionario, iguales a cualquier otra. En FORTH realmente es posible que uno mismo defina nuevas palabras definitorias.

Los símbolos de estructura del programa incluyen DO y LOOP, que ya hemos visto anteriormente (similar al FOR...NEXT del BASIC), ; (el final de una definición de dos puntos) y otras estructuras de programa como IF...THEN y BEGIN...UNTIL, que se utilizan para bifurcaciones y bucles. Nuevamente tiene usted la posibilidad, aunque no siempre es fácil, de definir sus propias palabras nuevas de estructura de programa.

Dado que ya sabemos lo que ocurre en el diccionario, podemos ver cómo trata el FORTH nuestra entrada. Trata a cada grupo de símbolos sin espacios como una palabra potencial y luego procede de acuerdo con tres reglas principales.

1. Si encuentra la palabra en el diccionario, lleva a cabo lo que indica la definición. Si estuviera definida más de una vez, se utilizaría la definición más nueva.
2. Si la palabra no está en el diccionario, el FORTH comprueba si el símbolo (o grupo de símbolos) es un número y, si así es, lo recuerda temporalmente. (Lo coloca en un trozo de memoria llamado *pila*).
3. Si se encuentra con que el grupo de símbolos no es un número ni está en el diccionario, o sea irreconocible, el FORTH así se lo hace saber.

Palabras y espacios

Una palabra es, simplemente, cualquier secuencia de caracteres. En consecuencia, en FORTH las siguientes son todas palabras (si bien no necesariamente definidas en el diccionario):

```
SALCHICHA
Tubo de gomadesemulsionada
239
pH
25p
+
«merguez»
,!XXX)luego-allíQWERTY
```

El FORTH no realiza el más mínimo intento por otorgarle un significado especial a los caracteres especiales, exceptuando el espacio. Probablemente usted elija palabras como LONGITUD y MARCADOR para sus variables, pero el FORTH no se preocupará en absoluto si usted decide llamarlas 23PI/2 o FORI=1TO10. Quizá se pregunte por qué otros lenguajes son mucho más restrictivos. En muchos, el nombre de una variable, por ejemplo, debe ser una letra seguida de otras letras cualesquiera o dígitos. La respuesta en realidad reside en la actitud de ellos hacia los espacios. La mayoría de los lenguajes tratan los espacios como una especie de elemento decorativo. Pero en esos lenguajes puede omitirlos si así lo desea, y probablemente así lo hará al escribir expresiones: en BASIC normalmente no introducirá espacios en la expresión 2 + 3. Los lenguajes, por tanto, deben ser capaces de distinguir entre los diversos componentes del programa. Si usted pudiera denominar 2 + 3 a una variable en BASIC, el intérprete de BASIC no sabría si eso significaría el nombre de la variable o una expresión; de modo que debe restringir los nombres posibles de las variables para que no se produzcan tales ambigüedades. El FORTH asume un enfoque diferente. Utiliza los espacios de forma inflexible, como separadores entre palabras, y ésa es la única regla que necesita. (Éste es un punto crítico del FORTH y, por consiguiente, al digitar listados se debe tener especial cuidado con los espacios.)

Complementos al FORTH

En figFORTH, VARIABLE debe ir precedida por un número:

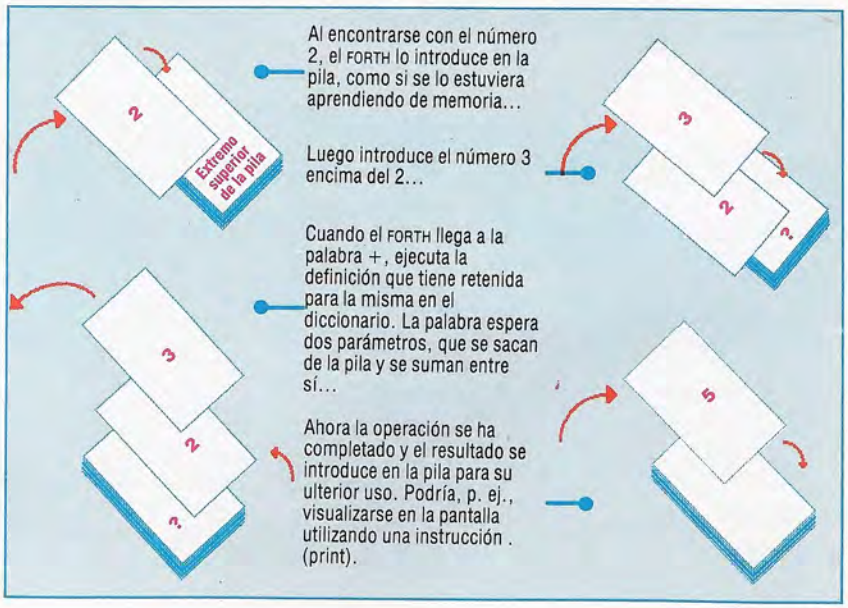
```
10 VARIABLE
   BOTELLASVERDES
```

es el valor con el cual ha de comenzar la variable

Todo tiene sentido

Cuando el FORTH encuentra una palabra, procede de acuerdo con la definición que tiene almacenada en su diccionario para dicha palabra. Sin embargo, cuando halla un número, lo coloca en una zona de la memoria conocida como *pila*: El número permanece allí hasta que posteriormente el FORTH necesite «recordarlo» (en el curso de la ejecución de la definición de una palabra, p. ej.). Aquí vemos lo que sucede cuando el FORTH interpreta la entrada:

```
2 3 +
```



Estas reglas han de modificarse ligeramente en cuanto a las definiciones de dos puntos, porque las acciones apropiadas no se llevan a cabo inmediatamente, sino que se recuerdan en la definición.

Las excepciones obvias las constituyen las palabras (como :, VARIABLE y CONSTANT) y éstas provocan cierta dificultad. Cuando usted escribe:

```
VARIABLE MARCADOR
```

parecería generar un error de la regla 3. Pero en realidad VARIABLE se ocupa de MARCADOR y las tres reglas no la llegan a ver nunca. De lo contrario, nunca podríamos entrar palabras en el diccionario.

Se puede apreciar cómo el diccionario confiere al FORTH su capacidad de interacción y su ampliabilidad. La interactividad está asegurada en función de las tres reglas, y la ampliabilidad se debe al hecho de que todas las definiciones nuevas se incluyen en el diccionario en términos de igualdad respecto a las antiguas.

De vuelta al cuadrado uno

La experiencia en el campo de la programación en código máquina puede facilitar la comprensión del FORTH, pero para quienes carezcan de tal experiencia presentamos el siguiente ejemplo comentado. Para definir una palabra CUADRADO como una subrutina que dibuje un cuadrado en la pantalla con lados de longitud LADO, procederíamos de la siguiente manera. En primer lugar, hemos de declarar la variable LADO:

```
VARIABLE LADO
```

Ésta le indica al FORTH que reserve un espacio en la memoria donde se pueda almacenar un valor numérico. Podemos desplazar valores desde y hacia esta dirección mediante el empleo de @ (traer) y ! (almacenar). De modo que p. ej.:

```
20 LADO !
```

le asignaría a la variable LADO el valor 20.

Ahora podemos definir nuestra nueva palabra CUADRADO utilizando las «palabras»: y; (tal como emplearíamos TO y END en LOGO):

```
:CUADRADO
  LADO !
  4 0 DO
    LADO @ ADELANTE
    90 DERECHA
  LOOP
;
```

Este procedimiento espera dos cosas. Primero, que ya se hayan definido las palabras DERECHA y ADELANTE y que ofrezcan facilidades de tortuga como en LOGO, de modo, que, por ejemplo:

```
50 ADELANTE
```

haría que la tortuga avanzara 50 unidades de pantalla hacia adelante. En segundo lugar, se espera que el usuario entre un valor para LADO, de modo que entrando:

```
50 CUADRADO
```

se dibujaría un cuadrado con lados de 50 unidades.

Trabajemos ahora con la subrutina y veamos exactamente cómo funcionaría si en realidad hubiéramos entrado la instrucción 50 CUADRADO. El FORTH examina su entrada y encuentra que el primer grupo de símbolos está separado por espacios, en este caso 50. El FORTH comprueba entonces si este grupo ya se ha entrado en su diccionario. Suponiendo que usted no haya definido con anterioridad el grupo de símbolos de caracteres 50 como una palabra, el FORTH no logrará hallarlo en el diccionario y comprobará si se trata de un número, como por supuesto sucede en este caso. Por consiguiente, el FORTH toma el número, lo coloca en un almacén de la memoria y comprueba el siguiente grupo de símbolos, que es CUADRADO. Evidentemente, CUADRADO sí está en el diccionario (acabamos de definirlo), de modo que el FORTH recupera su definición y procede a evaluarla. Los primeros términos con los que se tropieza son LADO! En este punto, usted bien podría pensar que algo funciona mal, puesto que LADO! no parecería tener ningún valor que asignarle a LADO; pero el FORTH comprueba su almacén de memoria y encuentra allí el valor que entramos (50) y lo utiliza, asignándole, por tanto, 50 a la variable LADO. Observe que el FORTH, a diferencia del LOGO, no comprueba esto: se limita a dar por sentado que usted ha entrado un número en el lugar apropiado, de modo que si usted simplemente llamara a la subrutina CUADRADO con CUADRADO en lugar de 50 CUADRADO, obtendría algunos resultados impredecibles y no un mensaje de error. 4 0 DO...LOOP se explica bastante por sí solo, pero es interesante observar cómo dentro de ese bucle se pasa el valor de la variable LADO a la subrutina ADELANTE mediante las palabras LADO @ ADELANTE. Éstas primero llaman a la dirección de LADO y luego (utilizando @) traen un valor desde esa dirección, listo para que ADELANTE lo utilice. (Compare esto con el método empleado por la rutina CUADRADO, que utilizaba DUPLICAR para presentar un valor para ADELANTE.) Los términos 90 DERECHA, por supuesto, son necesarios para hacer que la tortuga gire 90 grados tras dibujar cada lado, completando de ese modo el cuadrado. Puede ser que a primera vista el FORTH parezca algo confuso, en especial si usted no es todavía un programador experimentado. No obstante, en su estructura y comportamiento es muchísimo más directo de lo que sugiere en apariencia

El poder de la palabra

Definir una palabra CUADRADO ilustra únicamente un aspecto del potencial de ampliación que posee el FORTH. No sólo es posible construir «diccionarios» de palabras para controlar dispositivos mecánicos, como el telescopio que vemos en la fotografía, sino también idear sistemas de control para aplicaciones más abstractas. Por ejemplo, una casa productora de software británica (Mastertronics) utiliza el FORTH, en la creación de juegos de aventuras, para definir palabras que manipulen personajes, objetos y otras configuraciones propias de los entornos de fantasía

Plus ça change

El BBC+ obvia los principales puntos débiles de su antecesor, pero su precio le impide acceder a un mercado definido

Cuando se le concedió a Acorn el contrato del BBC Microcomputer, en 1981, la máquina era considerada por la mayoría de la industria como un inmenso adelanto en materia de ordenadores personales. Su velocidad y su versatilidad hacían que superara en rendimiento a todos los otros micros personales existentes en el mercado. Sin embargo, desde entonces, el mercado ha cambiado sustancialmente. Aunque la gama de interfaces y periféricos para el BBC Micro sigue siendo incomparable, con el correr del tiempo el ordenador ha dejado traslucir algunas debilidades.

El principal punto débil, que ha marcado al ordenador desde sus comienzos, ha sido la relativa escasez de memoria disponible. Algunas de las modalidades para gráficos de mayor resolución dejan muy poca RAM al usuario para el desarrollo de programas. Ello no se debe a que el BBC Micro carezca de RAM para el usuario (posee los mismos 64 k de otros muchos micros personales), sino a que las avanzadas configuraciones para gráficos del ordenador no caben con comodidad en los confines de las capacidades para direccionamiento de memoria de un procesador de ocho bits.

Hacia 1984 este problema representó una amenaza aún mayor para el futuro a largo plazo de la máquina. Mientras otros fabricantes sacaban partido de la caída del costo de los microprocesadores en el mercado internacional (y en particular de los chips de memoria) para recortar sus precios, Acorn mantuvo inalterado el precio del BBC Modelo B. Acorn pudo hacerlo en parte gracias al valor del prestigio del contrato con BBC, y también en parte al subsidio que concedió el gobierno británico a los establecimientos educativos que adquirieran el ordenador a través del Microcomputer Educational Programme (MEP). Además, mientras los otros fabricantes, como Sinclair, estaban rediseñando sus placas de circuito impreso para sacar ventaja de los chips que estaban apareciendo, de mayor capacidad y menor precio, Acorn no consiguió hacerlo. Ello determinó que el costo de producción del BBC Micro alcanzase un nivel considerablemente superior al de todos sus competidores.

Por último, la tecnología de ocho bits del BBC Micro corría el peligro de quedar anticuada. Sinclair Research, uno de los mayores rivales de Acorn para el contrato con BBC, no tuvo reparos en dar publicidad al hecho de que su ordenador de 16 bits, el QL, estaba diseñado para competir exactamente al mismo precio que el BBC Micro.

Desde entonces, el peligro de que los procesadores de 16 bits conviertan en obsoletos a sus equivalentes de ocho bits ha disminuido, aunque de forma temporal. El repentino descenso en las ventas de ordenadores y la crisis financiera tanto de Acorn como de Sinclair han hecho que la industria informática actúe con mayor cautela. El resultado ha sido que en lugar de introducir máquinas nuevas con todo el margen de riesgo que ello implica, casi todos los principales fabricantes de micros personales hayan optado simplemente por perfeccionar sus productos existentes. Para ser justos, esto responde a los deseos del público. El mercado se está volviendo mucho más sofisticado y el cliente no parece mostrarse interesado por las últimas innovaciones en materia de hardware y sí, en cambio, por la calidad y cantidad de la base de software. Y de allí el desarrollo de ordenadores más potentes compatibles con las bases de software desarrolladas para máquinas anteriores.

La base de software

El BBC + es una respuesta de Acorn a la demanda de los clientes que requieren grandes bases de software ya existentes, pero también mayor memoria para ejecutar programas más largos y sofisticados. El ordenador está equipado con 32 k más de RAM que su predecesor. Al estar basado en un procesador de ocho bits, la cantidad máxima que se puede direccionar directamente se limita, como es obvio, a 64 K. En consecuencia, al igual que Atari y Commodore, Acorn ha utilizado la técnica de *conmutación de bancos*, que permite acomodar las zonas de memoria adicionales.

La técnica de conmutación de bancos permite que el procesador «mire» una de dos zonas de memoria. Se acomodan dos bancos de memoria de modo que ocupen las mismas direcciones de memoria. Por cuanto concierne al procesador, sólo hay una única posición de memoria; pero, según en qué zona de memoria está «depositada», la dirección puede contener varios contenidos diferentes.

Los 32 K de RAM adicionales se hallan en las direcciones 12288 (hexadecimal 3000) y 45055 (hexadecimal AFFF). Ocupa la mayor parte de la zona para programas en BASIC y la zona de memoria ocupada por las ROM paginadas (las ROM paginadas son las ROM de BASIC y aquellas que puedan haber instalado los propios usuarios, como View y LOGO).

La RAM en sombra propiamente dicha está dividida entre estas dos secciones, que vamos a examinar por separado. Como ya hemos visto, el mayor problema del espacio de memoria en el BBC Micro es la gran cantidad de RAM que se requiere para soportar una pantalla de gráficos en alta resolución. Por tanto, la enorme masa de RAM adicional se ha cedido completamente para usarla como RAM de video. De hecho, los 20 K completos que «ensombrecen» la zona para programas en BASIC queda reservada para esta finalidad, lo que representa memoria suficiente para soportar la pantalla más detallada. Al rediseñar la placa de circuito impreso con el objeto de dar cabida a los chips de RAM adicio-

BBC+

MEMORIA

76 Kbytes de RAM, de los cuales hay 64 Kbytes disponibles para programas en BASIC, 32 Kbytes de ROM, ampliables a 192 Kbytes

CPU

Procesador 6512 trabajando a 2 MHz

DISCOS

El BBC+ viene equipado con un sistema DFS Acorn como estándar, si bien las unidades de disco son adicionales

DOCUMENTACION

Se ha actualizado la guía para el usuario del BBC Micro, de por sí ya muy completa, para incluir información adicional sobre la RAM en sombra y otros puntos útiles

VENTAJAS

La provisión de hardware extra del BBC+ es una respuesta a muchas de las críticas que suscribió el BBC Micro original. Ahora se dispone de suficiente memoria como para permitir que los programadores soporten una pantalla de alta resolución a la vez que escriban sofisticados programas en BASIC

DESVENTAJAS

A pesar de ser una máquina mejorada, el BBC+ sigue siendo caro para la mayoría de los usuarios de micros personales. En la actualidad el mercado educativo parece estar optando por máquinas MS-DOS, lo que restringe aún más los posibles mercados para el BBC+

nales, Acorn ha tenido oportunidad de introducir algunos otros cambios. Uno de los más notables es que la nueva máquina opera con un procesador 6512 en vez de con un 6502. Con ello se estandariza parte del intercambio de información entre la CPU y el chip para periféricos 6522 con el que Acorn tuvo algunos problemas en el pasado. La dirección de conmutación de bancos propiamente dicha (hexadecimal FE34) está contenida en una de las ROM (IC36) del sistema operativo que se han vuelto a diseñar. Puesto que el BBC Micro opera en gran medida en base a interrupciones, la adición de esta dirección no ha requerido ningún cambio sustancial en el sistema.

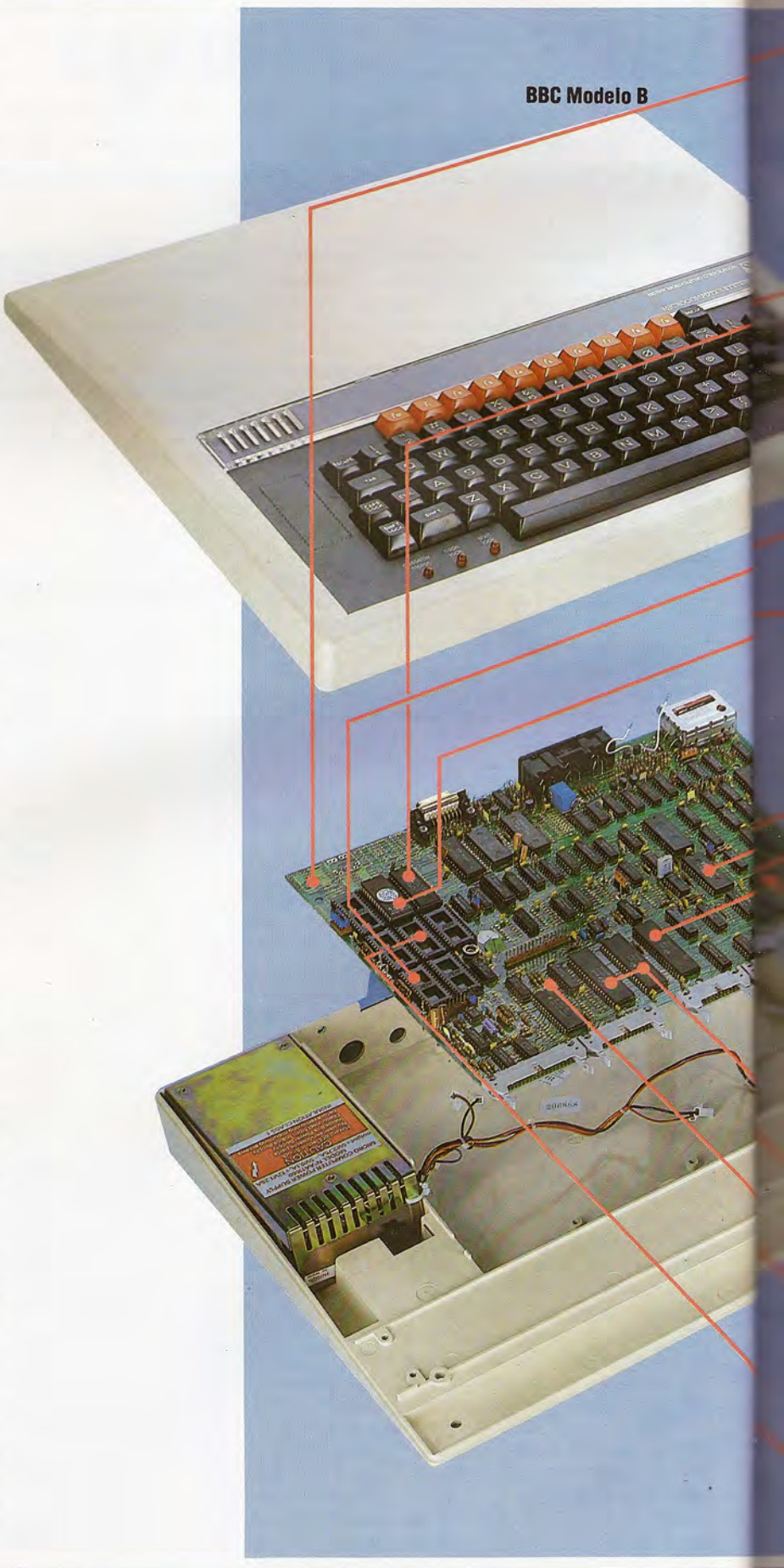
Los 12 K restantes de memoria que se han proporcionado al BBC + se hallan en la zona de direcciones de memoria entre 32768 (hexadecimal 8000) y 45055 (hexadecimal AFFF). Ésta es la zona reservada para uso de las ROM paginadas. Nuevamente la conmutación de bancos permite que las ROM y la RAM en sombra de abajo compartan el mismo tiempo.

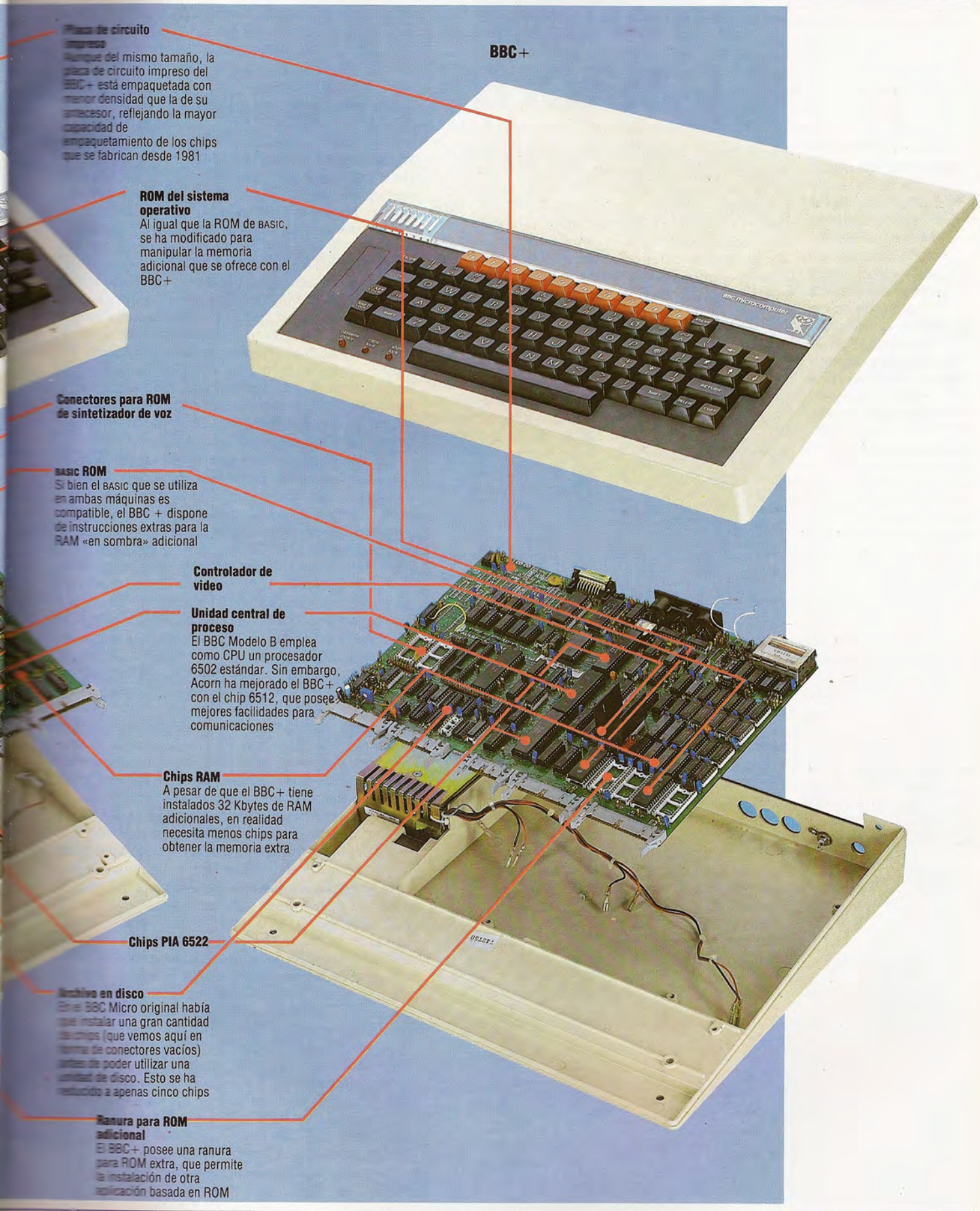
BASIC y OS rediseñados

Los usuarios del BBC + tienen una ventaja sobre sus compañeros de Atari y Commodore, quienes deben utilizar instrucciones POKE para poder acceder a los bancos de memoria adicionales, porque Acorn también ha vuelto a diseñar su BASIC y sus ROM MOS (machine operating system: sistema operativo de la máquina), para proporcionar apoyo de software para las facilidades de conmutación entre bancos.

A la memoria de pantalla en sombra se puede acceder de varias formas. Utilizando la instrucción MODE, seguida de un número entre 128 y 135, se producirá una pantalla en sombra por defecto correspondiente a las ocho modalidades de pantalla entre 0 y 7. Sin embargo, las subsiguientes instrucciones MODE que no estén comprendidas entre 128 y 135, darán por defecto la zona de RAM normal, lo que obviamente constituye un error fatal. La instrucción *SHADOW fijará permanentemente la pantalla en la RAM en sombra, impidiendo que el usuario escriba sobre cualquier programa que hubiera en BASIC. Para salir de esta situación, se ha de ejecutar la instrucción *SHADOW 1. Naturalmente, como todas las instrucciones * BBC, están las correspondientes instrucciones *FX; en este caso, *FX114 y *FX114,1, respectivamente. De modo similar, las instrucciones MODE se pueden suplantar por VDU 22, (128 + n), donde n representa la modalidad requerida.

Al perfeccionar el BBC Micro, Acorn ha optado asimismo por incorporar en el equipo estándar un sistema de archivo en disco (DFS). Éste es útil para quienes deseen adquirir un ordenador y pretendan utilizar desde el principio unidades de disco; la alternativa consiste en instalar un DFS por un recargo adicional. Pero para la mayoría de los usuarios, que sólo desean utilizar con la máquina una unidad de cassette, esta provisión adicional se limita a incrementar el precio de partida del BBC +. Esto parece lamentable, dado que el costo extra tampoco hará nada por silenciar las críticas de quienes sostienen que la gama de micros BBC tiene fijado un precio excesivo.




BBC+
Placa de circuito impreso
Aunque del mismo tamaño, la placa de circuito impreso del BBC+ está empaquetada con menor densidad que la de su antecesor, reflejando la mayor capacidad de empaquetamiento de los chips que se fabrican desde 1981
ROM del sistema operativo
Al igual que la ROM de BASIC, se ha modificado para manipular la memoria adicional que se ofrece con el BBC+
Conectores para ROM de sintetizador de voz
BASIC ROM
Si bien el BASIC que se utiliza en ambas máquinas es compatible, el BBC + dispone de instrucciones extras para la RAM «en sombra» adicional
Controlador de video
Unidad central de proceso
El BBC Modelo B emplea como CPU un procesador 6502 estándar. Sin embargo, Acorn ha mejorado el BBC+ con el chip 6512, que posee mejores facilidades para comunicaciones
Chips RAM
A pesar de que el BBC+ tiene instalados 32 Kbytes de RAM adicionales, en realidad necesita menos chips para obtener la memoria extra
Chips PIA 6522
Archivo en disco
El BBC Micro original había que instalar una gran cantidad de chips (que vemos aquí en forma de conectores vacíos) antes de poder utilizar una unidad de disco. Esto se ha reducido a apenas cinco chips
Ranura para ROM adicional
El BBC+ posee una ranura para ROM extra, que permite la instalación de otra aplicación basada en ROM

Primeras palabras

Iniciamos una nueva serie, en la que esbozaremos los requisitos para construir un tester digital sencillo. En esta introducción consideraremos algunos de los problemas que implica la conversión de analógico a digital y desarrollaremos una estrategia para obtener el diseño adecuado

Por lo general los ordenadores operan de forma totalmente digital. Incluso las interfaces «humanas» de la entrada por teclado y la salidad de video se basan exclusivamente en sistemas de circuitos digitales. La utilidad de los ordenadores se podría ampliar considerablemente si pudieran comunicarse con el «mundo exterior». Pero éste tiende a ser un mundo analógico de voltajes, temperaturas, alturas, pesos, etc., infinitamente variables. En consecuencia, cuando los ordenadores necesitan recoger datos o controlar dispositivos del mundo real, se hace necesaria la conversión entre un entorno analógico y uno digital.

Un ordenador unido a un dispositivo sensible a la temperatura, por ejemplo, habrá de trabajar con números binarios que correspondan a las temperaturas medidas con el fin de decidir lo que debe hacer. Ello implicaría lo que se conoce como un convertidor de analógico a digital, o A/D. Más adelante veremos algunas de las técnicas que se emplean para convertir mediciones analógicas (continuamente variables) en representaciones digitales (binarias).

El contrario de esta situación se produce cuando se utiliza un dispositivo digital, como puede ser un ordenador, para alterar el valor de algo del mundo real. A modo de ejemplo, consideremos un ordenador que se ha programado para producir música. El ordenador crea un valor digital binario, pero éste se ha de convertir en una frecuencia acústica. En este tipo de situaciones se requiere la conversión de señales digitales a señales analógicas.

Medidores digitales

Un multímetro o tester es un instrumento para medir la conductividad (medida en ohmios), la corriente (medida en amperios) y la tensión (medida en voltios) de un sistema eléctrico. En esta serie construiremos un tester digital, que efectuará estas mediciones con un elevado nivel de precisión y visualizará los resultados digitalmente mediante una visualización LCD o LED. El desarrollo de tal dispositivo sería una ardua empresa si no fuera por el advenimiento de los circuitos integrados a gran escala especializados, que combinan muchas etapas analógicas y digitales en un único chip. Al desarrollar nuestro DVM (*digital voltmeter*: voltímetro digital) deseamos diseñar un circuito que pueda trabajar a la vez como tester digital independiente, y también como interface directamente con un ordenador, para que éste pueda leer y procesar la tensión y otras magnitudes.

Un voltímetro digital utiliza una aguja que se desplaza por encima de una escala para proporcionar una lectura directa en voltios, amperios u ohmios. Combinaciones de resistencias en derivación y en serie les permiten a tales testers dar lecturas en términos de tensión, intensidad o resistencia, pero fundamentalmente todos trabajan de la misma manera. En definitiva, una corriente fluye a través de una bobina suspendida en el interior de un campo magnético, y la intensidad de la corriente determina hasta dónde girará la bobina contra la fuerza de un muelle. Una aguja conectada a la bobina muestra hasta dónde ha girado la misma, y la escala se calibra en términos de las unidades que se estén midiendo.

Cuando se ha de medir y visualizar digitalmente una tensión desconocida, los problemas son mucho mayores. Antes de pasar a ver cómo se realiza esto, es interesante considerar el problema inverso, el de convertir valores digitales en valores analógicos.

La conversión de digital a analógico (D/A) es una operación bastante directa. Supongamos que usted desea convertir una palabra de ocho bits (una unidad compuesta por ocho dígitos binarios) en una tensión analógica entre 0 V y 1 V. Una palabra de ocho vits puede representar cualquier valor entre 0 y 255 (binarios 00000000 y 11111111). Esa escala de un voltio se puede resolver, por lo tanto, en pasos de voltaje de 256 puntos, o 0,0039 voltios.

Los valores binarios se pueden convertir en valores analógicos utilizando las señales binarias para establecer corrientes cada vez más intensas. Los diagramas muestran tanto el tipo de convertidor D/A más simple, como el tipo más común, el R2R. En ambos casos, los interruptores mecánicos representan interruptores electrónicos activados mediante señales binarias, significando «cerrado» el 1 binario y «abierto» el 0 binario. Cada dígito binario positivo activa un interruptor electrónico que simplemente permite que fluya más corriente.

La conversión de señales analógicas (p. ej., tensiones) en sus equivalentes digitales es bastante más difícil, y para ello hay muchas técnicas disponi-

El mercado del tester

Comparemos las características que ofrecen tres testers digitales típicos:

Lascar LMM 100 Bench Meter

Visualización:	LCD de 3½ dígitos
Escala:	25 escalas
Precisión:	−0,1% (voltios CC)
Potencia:	Pila
Otras características:	Indicaciones de pausa, polaridad, pila y superación de escala

Soar ME-531 Autoranging Hand-held Meter

Visualización:	LCD de 3½ dígitos
Escala:	Escala automática
Precisión:	0,8% (voltios CC)
Potencia:	Pila
Otras características:	Polaridad, indicación de sobrecarga, comprobación de diodos

Maplin Precision Gold M-5010 Basic Hand-held Meter

Visualización:	LCD de 3½ dígitos
Escala:	29 escalas
Precisión:	0,25% (voltios CC)
Potencia:	Pila
Otras características:	Indicación audible de continuidad, indicación de superación de escala y polaridad, comprobación de diodos

Marcus Wilson-Smith

bles. Al igual que en la conversión D/A, el número de bits utilizado determina la resolución de la medición; no tiene nada que ver con la gama de tensiones que puede medir. Para una mayor resolución serán necesarios más bits, pero para la finalidad de nuestro DVM, ocho bits serán más que adecuados.

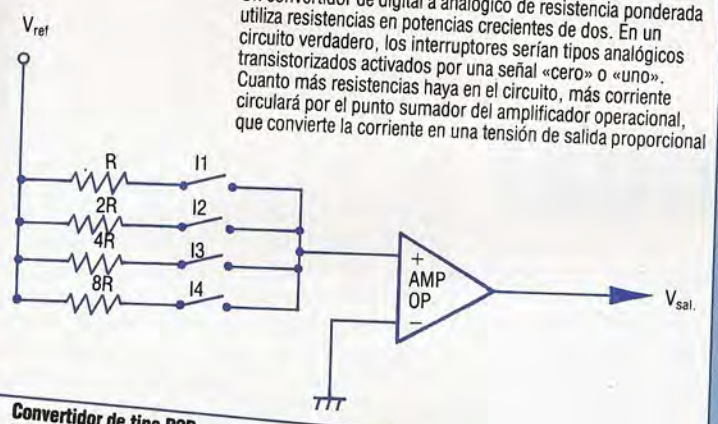
Nuestro diseño tendrá una gama básica de 0 V a 2 V, por lo que con los ocho bits de resolución, los intervalos medibles serán de 0,0078 voltios. Los márgenes de medida de 0 V a 20 V y de 0 V a 200 V se conseguirán utilizando un sencillo circuito divisor de potencial constituido por resistencias.

Circuitos de conversión

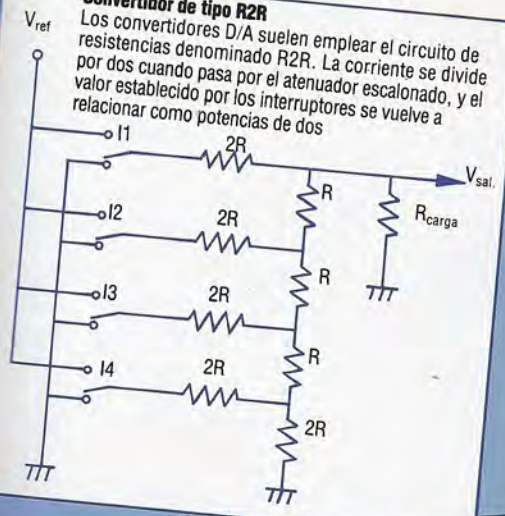
La conversión de analógico a digital es bastante más complicada y cara que la conversión D/A, y en parte depende de cuán rápidamente deba realizarse la conversión y de cuántos bits de resolución se necesiten. Si hay que convertir señales de audio de márgenes de frecuencia de hasta 15 kHz en señales analógicas, por lo menos será necesario convertir 30 000 muestras de la señal cada segundo. Si hay que digitalizar señales de vídeo, la velocidad de muestreo deberá ser mucho más elevada.

Afortunadamente, un DVM sólo tiene que medir tensiones continuas (CC) o alternas (CA) de baja frecuencia, por lo que bastará una baja velocidad de conversión; sin embargo, para una precisión razonable, serán precisos ocho bits de resolución.

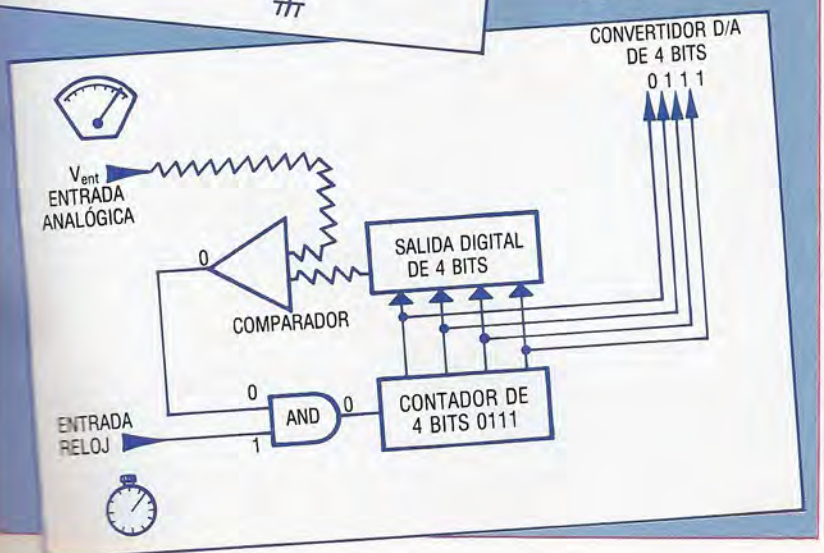
Sencillo convertidor D/A
Un convertidor de digital a analógico de resistencia ponderada utiliza resistencias en potencias crecientes de dos. En un circuito verdadero, los interruptores serían tipos analógicos transistorizados activados por una señal «cero» o «uno». Cuanto más resistencias haya en el circuito, más corriente circulará por el punto sumador del amplificador operacional, que convierte la corriente en una tensión de salida proporcional



Convertidor de tipo R2R
Los convertidores D/A suelen emplear el circuito de resistencias denominado R2R. La corriente se divide por dos cuando pasa por el atenuador escalonado, y el valor establecido por los interruptores se vuelve a relacionar como potencias de dos



Sencillo convertidor A/D
La señal del reloj hace que el contador de cuatro bits se incremente a partir de cero, y el convertidor D/A convierte su salida binaria en una tensión de salida equivalente que se aplica a un comparador. Cuando la tensión de salida se corresponde con el voltaje de entrada que se está midiendo, el comparador produce una salida que detiene el contador. Entonces, el valor de ésta se puede comprobar, calculando el valor de la tensión equivalente



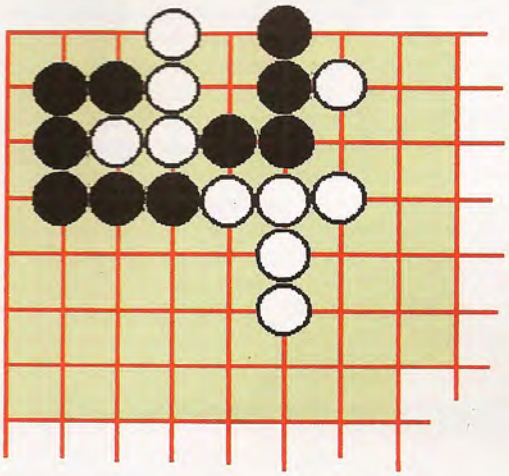
Técnicas de conversión A/D

El método de conversión que se suele emplear consiste en comparar una tensión conocida con otra desconocida (la que se mide). Los chips de comparación integrados (una clase de amplificadores operacionales, descritos anteriormente) resultan muy adecuados para esto, ya que pueden configurarse para dar una salida cuando (y sólo cuando) las dos tensiones de entrada son iguales. Un contactor binario se fija inicialmente a cero y cuenta, dando una tensión de salida (mediante una de las técnicas D/A descritas) que a la larga se hará igual a la tensión de entrada desconocida que se mide. Cuando la tensión desconocida y la tensión determinada por el convertidor D/A son iguales, el comparador produce una salida que detiene el contador binario y permite que se calcule la tensión. A la izquierda puede verse un circuito simplificado para este tipo de convertidor A/D.

Aunque existen muchos IC de un solo chip que pueden activar directamente visualizadores LCD o LED, diseñados específicamente para su aplicación a DVM, normalmente no resulta sencillo acoplarlos a los ordenadores, porque sus salidas sólo son adecuadas para atacar visualizadores digitales. Por tanto, hemos optado por basar nuestro diseño en el chip de medidor de panel digital ICL7135 de Analog Systems. Este chip proporciona salidas decimales codificadas en binario que pueden ser convertidas fácilmente en una forma adecuada para activar visualizadores de siete segmentos o ser acopladas a un ordenador.

El chip presenta una impedancia de entrada muy elevada y una entrada diferencial que permite medir tanto tensiones positivas como negativas. También permite una lectura de cero verdadero con una entrada de 0 V, las indicaciones de fuera margen mínimo y de margen máximo, y de polaridad, E/S lógica para su acoplamiento a un microprocesador y unos requisitos de reloj y de alimentación sencillos. Aunque es un chip caro, es mucho más barato que el circuito equivalente montado con componentes discretos, es más fácil de utilizar y ofrece muchas más posibilidades de funcionar bien a la puesta en marcha.

Para construir un tester de gran precisión sólo se necesitan dos cosas: resistencias de tolerancia restringida para el atenuador de entrada y el empleo de un DVM de precisión de algún amigo para fijar la tensión de referencia. No obstante, aun sin estos elementos, se podrán obtener resultados sumamente satisfactorios.



SEMEAI

Situación comprometida
Un método común para capturar fichas junto al margen del tablero consiste en asfixiarlas, táctica conocida como *semeai*. Es importante la cantidad de licencias que le quedan a cada grupo involucrado en una batalla *semeai*. Aquí, por ejemplo, las cosas parecen habérsele puesto muy difíciles al grupo de blancas en forma de L, dado que sólo tiene tres licencias, mientras que el correspondiente grupo de negras posee cuatro

En su lugar

Continuando con nuestro proyecto de programación de este juego oriental, centraremos nuestra atención en el módulo que le permitirá colocar fichas en el tablero

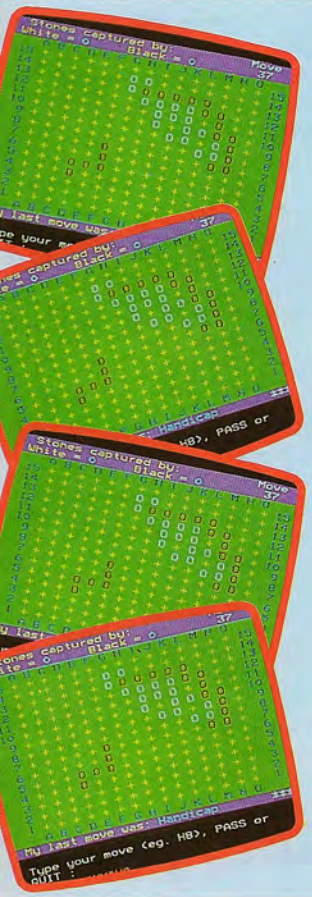
Debido a la forma estructurada y generalizada en la que se ha desarrollado el programa, es sumamente sencillo modificar el programa principal que llama a las rutinas (líneas 10-130) para que usted y algún amigo jueguen entre sí. Más adelante veremos cómo hacerlo, de modo que esté entonces en condiciones de utilizar el programa para partidas entre dos jugadores, con el ordenador actuando a modo de árbitro. La descripción del programa está referida a la versión para el BBC Micro; en las otras tres versiones se mantienen los números de línea y la estructura del programa.

La primera rutina que vamos a considerar es FNlegalidad (de la línea 3890 a la 4000). Esta función aceptará un movimiento, P%, realizado por el color C%, y devolverá un valor que indicará si el movimiento es legal o no. Esta comprobación asegura que:

1. La intersección (P%) esté vacante.
2. El jugador no capture una ficha en Ko.
3. La ficha del jugador no esté cometiendo suicidio.

Si examina la rutina que lee mensajes (de la línea 390 a la 560 en la versión para el BBC Micro), observará que estas tres condiciones coinciden con los mensajes dos, tres y cuatro (tres, cuatro y cinco en la versión para el Spectrum), respectivamente. Por consiguiente, FNlegalidad se ha diseñado para que devuelva alguno de estos números, o bien cero si el movimiento es legal.

El hecho de que la posición esté vacante se maneja mediante la comprobación de un cero en el byte apropiado del tablero, en la línea 3910. Para comprobar las otras dos posibilidades, se coloca



Módulo tres

BBC Micro:

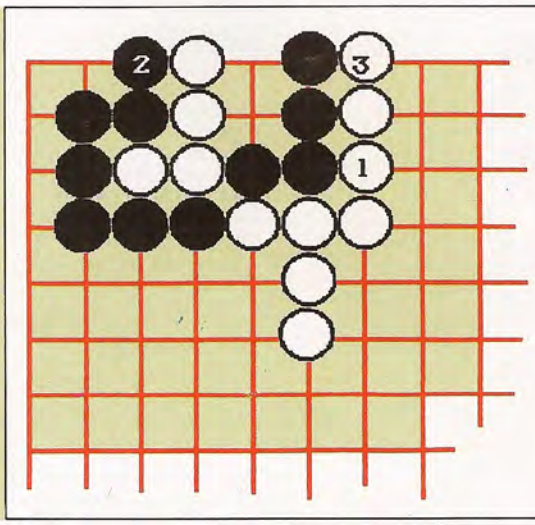
```
1380 ko%=FALSE:fin%=FALSE
2310 :
2320 DEF PROCmovimiento__blancas
2330 LOCAL C%,L%,P%,X%,Y%,A$,C$,Y$
2340 atari2$=""
2350 A$=fnentrada(21,8,4)
2360 IF A$="PASO" THEN ko%=FALSE:PROCimprimir__
     tablero:GOTO 2490
2370 IF A$="ABANDONO" THEN fin%=TRUE:ENDPROC
2380 X%=ASC(LEFT$(A$,1))-64
2390 Y$=MID$(A$,2)
2400 FOR C%=1 TO LEN(Y$)
2410    C$=MID$(Y$,C%,1)
2420    IF C$<"0" OR C$>"9" THEN
     PROCmensaje(23,1,A$):GOTO 2350
2430    NEXT
2440 Y%=VAL(Y$)
2450 IF X%<1 OR X%> 15 OR Y%<1 OR Y%> 15 THEN
     PROCmensaje(23,1,A$):GOTO 2350
2460 P%=16*Y%+X% : L%=FNlegalidad(P%,blancas%)
2470 IF L%>0 THEN PROCmensaje(23,L%,A$):GOTO 2350
2480 PROCefectuar__movimiento(P%,blancas%)
2490 PROCmensaje(23,0,"")
2500 ENDPROC
2510 :
2520 REM ************************
3620 :
3630 DEF PROCefectuar__movimiento(P%,C%)
3640 LOCAL A%,L%,N%
3650 tablero%? P%=C%
3660 FOR L%=1 TO 4 : A%=P%+dir%(L%) : IF
     tablero%?A%<> color%-C% THEN 3700
3670 PROCcontar(A%,color%-C%):IF clib%=0 THEN
     ko%=A%:N%=N%+cstn% :
     PROCsuprimir(A%,color%-C%)
3680    IF clib%=1 AND C%=negras% THEN
     atari$="Atari":atari2$=""
3690 IFclib%= 1 AND C%=blancas% THEN atari2$="Atari"
3700 NEXT
3710 IF N% <> 1 THEN ko%=FALSE
3720 captura%(C%)=captura%(C%)+N%
3730 PROCimprimir__tablero
3740 ENDPROC
3750 :
3760 REM ************************
3770 :
3780 DEF PROCsuprimir(P%,C%)
3790 IF tablero%?P%<>C% THEN ENDPROC
3800 tablero%?P%=0
3810 PROCsuprimir (P%+dir%(1),C%)
3820 PROCsuprimir (P%+dir%(2),C%)
3830 PROCsuprimir (P%+dir%(3),C%)
3840 PROCsuprimir (P%+dir%(4),C%)
3850 NEDPROC
3860 :
3870 REM ************************
3880 :
3890 DEF FNlegalidad(P%,C%)
3900 LOCAL A%,K%,L%,S%
3910 IF tablero%?P%<>0 THEN =2
3920 tablero%?P%=C%
3930 FOR L%=1 TO 4
3940    A%=P%+dir%(L%)
3950    IF tablero%?A%=color%-C% THEN
     PROCcontar(A%,color%-C%):IFclib%=0 THEN
     K%=K%+cstn%
3960    NEXT
3970 IF K%=0 THEN PROCcontar(P%,C%):IF clib%=0 THEN
     S%=4
3980 tablero%?P%=0
3990 IF P%=ko% AND K%=1 THEN  =3
4000 =S%
4010 :
4020 REM ************************
```

temporalmente en el tablero la ficha a jugar. El bucle, L%, cuenta luego las licencias de los grupos enemigos circundantes (utilizando PROCcontar, desarrollado en el último módulo). Dentro de este bucle, K% lleva el registro de la cantidad de fichas que serían muertas mediante el movimiento P%. Ahora se pueden efectuar las comprobaciones finales:

- Si no se mata ninguna ficha (K%=0) y la ficha a colocar no tiene licencias (clib%=0), la ficha debe estar cometiendo suicidio.
- Si se mata una ficha (K%=1) y la ficha a jugar se halla en la posición Ko (P%=ko%), se está violando la regla del Ko.

La comprobación del Ko quizá resulte algo difícil de comprender. Cuando un jugador captura una sola ficha, ko% se establece igual a la posición de la ficha capturada mediante la rutina efectuar_movimiento en las líneas 3670 y 3710. Si la nueva ficha a jugar está posicionada en este punto y está capturando sólo una ficha contraria, debe estar en el punto Ko y estar violando la regla del Ko. La ficha que esté capturando debe ser la última ficha jugada por el oponente.

La siguiente rutina es la que actualizará el tablero tras cada movimiento. PROCefectuar_movimiento comienza por colocar una ficha del color adecuado en el tablero, y un bucle comprueba el estado de cada uno de los grupos que la rodean. Si la nueva ficha llena la última licencia restante de un grupo contrario, ese grupo se suprimirá del tablero mediante una llamada a PROCsuprimir. Por el contrario, si sólo posee una licencia, entonces está en peligro de ser capturada al siguiente movimiento, de modo que se actualiza la serie atari adecuada. Recordará que atari es un aviso habitual, que se consigna cuando usted está a punto de capturar una ficha, o más, de su oponente. Por último, se actualiza la cantidad de fichas capturadas por el jugador

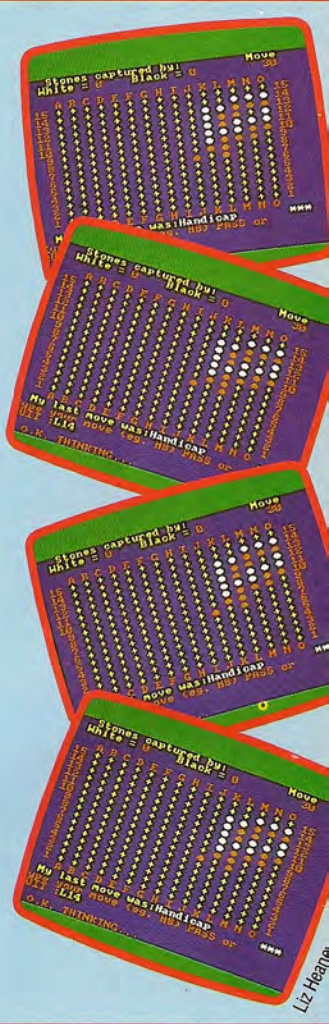


SEKI

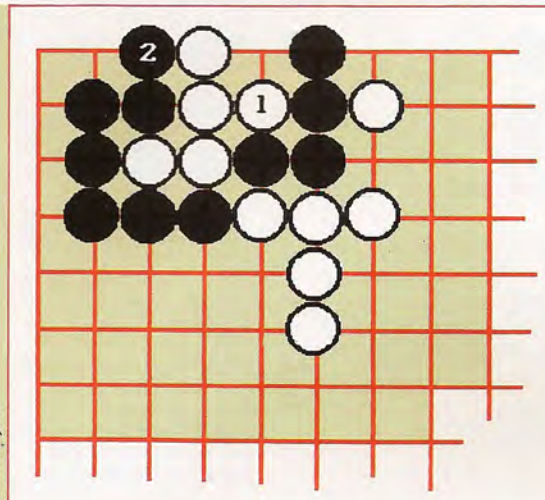
Situación neutra
Las batallas *semeai* casi siempre terminan ya sea con la captura del grupo más débil o bien en la situación de estancamiento denominada *seki*. Si las blancas atacan al grupo de negras desde el exterior, entonces se puede conseguir el *seki*. En la situación que vemos aquí, las blancas tienen su posición asegurada, porque comparten sus dos licencias restantes con el grupo de las negras. Ninguno de los bandos puede jugar en esta zona sin dejar a su grupo a merced de una captura

Amstrad CPC 464/664:

```
  60 MOVIMIENTO%=MOVIMIENTO%+1:GOSUB 2320
1380 KO%=0:FIN%=0
2310 :
2320 REM RUTINA MOVIMIENTO BLANCAS
2340 A2$="                "
2350 IP%=22:IM%=8:IW%=4:GOSUB 1990
2360 IF A$="PASO" THEN KO%=0:GOSUB 1730:GOTO
     2490
2370 IF A$="ABANDONO" THEN FIN%= 1:RETURN
2380 X%=ASC(LEFT$(A$,1))- 64
2390 Y$=MID$(A$,2)
2400 FOR WC=1 TO LEN(Y$)
2410 C$=MID$(Y$,WC,1)
2420 IF C$ < "0" OR C$ > "9" THEN
     MP%=24:MM%=1:O$=A$:GOSUB 2160:GOTO 2350
2430 NEXT
2440 Y%=VAL(Y$)
2450 IF X% > 0 AND X% < 16 AND Y% > 0 AND Y% < 16 GOTO
     2460
2455 MP%=24:MM%=1:O$=A$:GOSUB 2160:GOTO 2350
2460 WP%=16 * Y%+X%:LP%=WP%:LC%=BLANCAS
     %:GOSUB 3890
2470 IF LL% > 0 THEN MP%=24:MM%=LL%:O$=A$:GOSUB
     2160:GOTO 2350
2480 MP%=WP%:MC%=BLANCAS%:GOSUB 3630
2490 MP%=24:MM%=0:O$="'":GOSUB 2160
2500 RETURN
2510 :
2520 REM ************************
3620 :
3630 REM RUTINA EFECTUAR-MOVIMIENTO
3640 N%=0
3650 POKE TABLERO+MP%,MC%
3660 FOR K=1 TO 4:A=MP%+DIR(K):IF
     PEEK(TABLERO+A)<>COLOR%-MC% GOTO 3700
3670 CP%=A:CC%=COLOR%-MC%:GOSUB 4040:IF
     CLIB%<> 0 GOTO 3680
3675 KO%=A:N%=N%+CSTN%:RP%=A:RC%=COLOR%-
     MC%:GOSUB 3780
3680 IF CLIB%=1 AND MC%=NEGRAS% THEN
     A1$="ATARI":A2$="          "
3690 IF CLIB%=1 AND MC%=BLANCAS% THEN A2$=
     "ATARI"
3700 NEXT
3710 IF N% <> 1 THEN KO%=0
3720 CAPTURA%(MC)=CAPTURA%(M)+N%
3730 GOSUB 1730
3740 RETURN
3750 :
3760 REM ************************
3770 :
3780 REM RUTINA SUPRIMIR
3790 IF PEEK(TABLERO+RP%)<>RC% THEN
     RETURN
3800 POKE TABLERO+RP%,0
3805 SK%(PILA%)=RP%:PILA%=PILA%+1
3810 RP%=SK%(PILA%-1)+DIR%(1):GOSUB 3780
3820 RP%=SK%(PILA%-1)+DIR%(2):GOSUB 3780
3830 RP%=SK%(PILA%-1)+DIR%(3):GOSUB 3780
3840 RP%=SK%(PILA%-1)+DIR%(4):GOSUB 3780
3845 PILA%=PILA%-1:RP%=SK%(PILA%)
3850 RETURN
3860 :
3870 REM ************************
3880 :
3890 REM RUTINA LEGALIDAD
3900 LL%=0:LK%=0
3910 IF PEEK(TABLERO+LP%)<>0 THEN LL%=2:
     RETURN
3920 POKE TABLERO+LP%,LC%
3930 FOR K=1 TO 4
3940 LA%=LP%+DIR%(K)
3950 IF PEEL(TABLERO+LA%)<>COLOR%-LC% GOTO
     3960
3955 CP%=LA%:CC%=COLOR%-LC%:GOSUB 4040:IF
     CLIB%=0 THEN LK%=LK%+ CSTN%
3960 NEXT
3970 IF LK%=0 THEN CP%=LP%:CC%=LC%:GOSUB 4040:IF
     CLIB%=0 THEN LL%=4
3980 POKE TABLERO+LP%,0
3990 IF LP%=KO% AND LK%=1 THEN LL%=3
4000 RETURN
4010 :
4020 REM ************************
```



Liz Hearne



Caroline Clayton

Situación límite
Si las blancas cometen el error de jugar en estas licencias internas, como vemos aquí, quedarán en situación de ser capturadas. Después de que las negras juegan la ficha 2, al grupo de las blancas sólo le queda una licencia y la batalla, efectivamente, está perdida

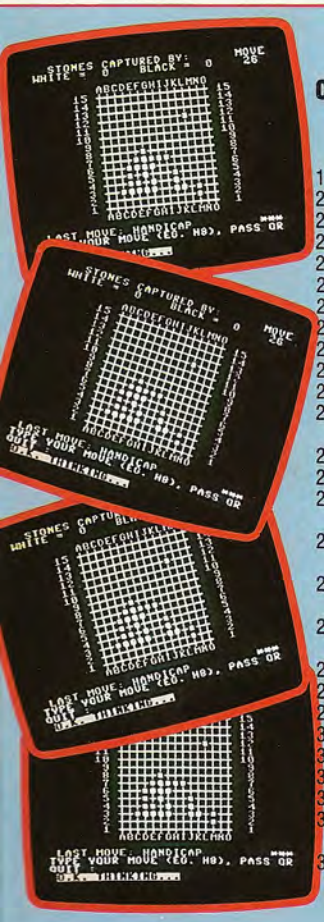
actual y se reimprime la pantalla utilizando la rutina imprimir_tablero.

Cuando se captura una ficha o un grupo de fichas, se llama PROCsuprimir para eliminarlas del tablero. Esta rutina trabaja de una forma repetitiva similar a PROCbuscar. Inicialmente suprime del tablero la ficha de la posición P%. Luego se llama repetidamente a sí misma para eliminar todas las fichas del mismo color que hubiera inmediatamente al norte, este, sur u oeste de la posición actual. Nuevamente, en las versiones para el Commodore 64, el Spectrum y el Amstrad se utiliza la «pila» para simular la repetición.

Concentrémonos ahora en la rutina que le permitirá entrar y realizar movimientos: PROCmovimiento_blancas. Puesto que ya contamos con algunas útiles rutinas a las que podemos llamar, la mayor parte del código tiende a comprobar la validez de la entrada. La rutina FNentrada (línea 2350) le permitirá entrar cuatro caracteres cualesquiera, de modo que tendrá que comprobar si usted ha entrado PASO, ABANDONO o una posición legal del tablero.

PASO sólo requiere restablecer la bandera Ko antes de retornar, mientras que ABANDONO establece en TRUE el valor lógico de fin del juego, de modo que al retornar el programa sale del bucle principal del movimiento, de la línea 60 a la 90. Si la entrada no es PASO ni ABANDONO, el código del programa divide la entrada en coordenadas horizontales (de A a O) y verticales (de 1 a 15), que se combinan en una posición del tablero, P%. Observará que aquí no podemos utilizar con facilidad la función VAL. Ello se debe a que la entrada puede ser de dos o tres caracteres de longitud y VAL(MIDS(«A1?»,2)) nos daría el resultado legal de uno, aunque al final de la serie haya un signo de interrogación adulterado. Si la entrada es válida y FNlegalidad confirma que el movimiento es legal, la rutina llama a PROCefectuar_movimiento, restablece la zona de mensajes llamando a PROCmensaje con el mensaje O.K.ESTOY PENSANDO (listo para que juegue el ordenador) y luego termina.

A lo largo de este programa hemos utilizado las constantes negras% y blancas% en lugar de los verdaderos números uno y dos. La única excepción se produce en PROCimprimir_tablero, donde los elementos de la matriz captura% muestran la cantidad



Commodore 64:

```
  60 mov%=mov%+1:GOSUB 2320
  90 IF NOT terminado% THEN 60
1380 ko%=0:terminado%=0
2310 :
2320 REM **** rutina movimiento blancas ****
2340 atari2$="                "
2350 ip%=21:im%=8:iw%=4:GOSUB 1990:REM entrada
2360 IF a$="PASO" THEN ko%=0:GOSUB 1730:GOTO 2490
2370 IF a$="ABANDONO" THEN terminado%=1:RETURN
2380 x%=ASC(LEFT$(a$,1))- 64
2390 y$=MID$(a$,2)
2400 FOR c%=1 TO LEN(y$)
2410 c$=MID$(y$,c%,1)
2420 IF c$ < "0" OR c$ > "9" THEN
     mp%=25:mm"=1:o$=a$:GOSUB 2160:GOTO 2350
2430 NEXT c%
2440 y%=VAL(y$)
2450 IF x% < 1 OR x% > 15 OR y% < 1 OR y% > 15 THEN
     mp%=25:mm%=1:o$=a$:GOSUB 2160:GOTO 2350
2460 lp%=16 * y%+x%:lc%=blancas%:GOSUB 3890:REM
     comprobar legalidad
2470 IF 11% > 0 THEN mp%=25:mm%=11%:o$=a$:GOSUB
     2160:GOTO 2350
2480 mmp%=lp%:mmc%=blancas%:GOSUB 3630:REM efectuar
     movimiento
2490 mp%=25:mm%=0:o$="":GOSUB 2160
2500 RETURN
2510 :
3620 :
3630 REM **** rutina efectuar movimiento ****
3640 n%=0
3650 POKE (tablero + mmp%),mmc%
3660 FOR k%=1 TO 4:a%=mmp%+dir%(k%):IF PEEK(tablero +
     a%)<>color%- mmc% THEN 3700
3670 cp%=a%:cc%=color%-mmc%:GOSUB 4040:IF clib%=0
     THEN ko%=a%:n%=n%+cstn%:rp%=a%:rc%=color%-
     mmc%:GOSUB 3780
3680 IF clib%=1 AND mmc%=blancas% THEN atari2$="Atari"
3700 NEXT k%
3710 IF n% <> 1 THEN ko%=0
3720 captura%(mmc%)=captura%(mmc%)+n%
3730 GOSUB 1730:REM imprimir tablero
3740 RETURN
3750 :
3760 REM ************************
3770 :
3780 REM rutina suprimir
3790 IF PEEK (tablero+rp%)<>rc% THEN RETURN
3800 POKE(tablero+rp%),0
3805 s(pila%)=rp%:pila%=pila%+1
3810 rp%=s(pila%-1)+dir%(1):GOSUB 3780
3820 rp%=s(pila%-1)+dir%(2):GOSUB 3780
3830 rp%=s(pila%-1)+dir%(3):GOSUB 3780
3840 rp%=s(pila%-1)+dir%(4):GOSUB 3780
3845 pila%=pila%-1:rp%=s(pila%)
3850 RETURN
3860 :
3870 REM ************************
3880 :
3890 REM rutina legalidad
3900 11%=0:lk%=0
3910 IF PEEK(tablero+lp%)<> 0 THEN 11%=2:RETURN
3920 POKE(tablero+lp%),lc%
3930 FOR k%=1 TO 4
3940 la%=lp%+dir%(k%)
3950 IF PEEK(tablero+la%)=color%-lc% THEN
     cp%=la%:cc%=color%-lc%:GOSUB 4040:IF clib%=0
     THEN lk%=lk%+cstn%
3960 NEXT k%
3970 IF lk%=0 THEN cp%=lp%:cc%=lc%:GOSUB 4040:IF
     clib%=0 THEN 11%=4
3980 POKE (tablero+lp%),0
3990 IF lp%=ko% AND lk%=1 THEN 11%=3
4000 RETURN
4010 :
4020 REM ************************
```

de fichas capturadas por cada bando. Una ventaja de este método es que nos permite modificar fácilmente la forma en la que representan los colores los bytes del tablero.

Ahora podemos utilizar esta facilidad para hacer un juego de dos jugadores, cambiando las siguientes líneas (véase *Complementos al* BASIC para las versiones destinadas a otras máquinas):

```
60 movimiento%=movimiento%+1:negras%=
   1:blancas%=2:PROCmovimiento__blancas%
80 movimiento%=movimiento%+1:negras%=
   2:blancas%=1:PROCmovimiento__blancas
```

Con ello tan sólo se cambian los valores reales de negras% y blancas% llamando cada vez a PROCmovimiento__blancas. Puesto que todas las rutinas se han diseñado para permitir cualquiera de los valores, el ordenador permitirá que negras y blancas jueguen alternativamente, utilizando ambas PROCmovimiento__blancas para comprobar la legalidad de cada movimiento y visualizarlos en el tablero. En el próximo módulo tendremos que cambiar estas líneas para que pueda jugar el ordenador. De momento, esta ínfima modificación le permitirá organizar partidas entre dos jugadores.

Complementos al BASIC

Insertando estas líneas en los listados de programa que ofrecemos, podrá disponer de un juego para dos jugadores, con el ordenador actuando como un tablero inteligente, comprobando los movimientos legales y llevando los marcadores.

Commodore 64

```
60 MOVIMIENTO%=MOVIMIENTO%+:NEGRAS%
   = 1:BLANCAS%=2:GOSUB 2320
80 MOVIMIENTO%=MOVIMIENTO%+1:NEGRAS%
   =2:BLANCAS%=1:GOSUB 2320
```

Spectrum

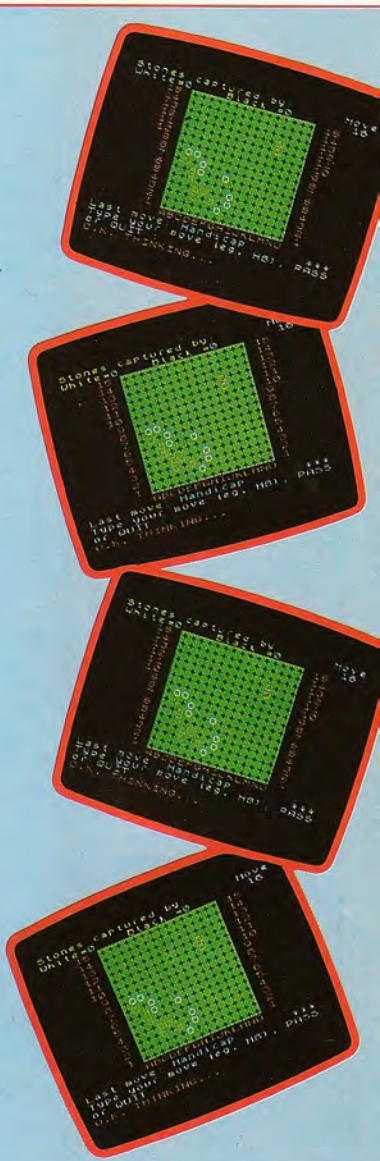
```
60 LET movimiento=movimiento+1:LET negras=
   1:LET blancas=2:GO SUB 2320
80 LET movimiento=movimiento+1:LET negras=
   2:LET blancas=1:GO SUB 2320
```

Amstrad CPC 464/664

```
60 mov%=mov%+1:negras%=
   1:blancas%=2:GOSUB 2320
80 mov%=mov%+1:negras%=
   2:blancas%=1:GOSUB 2320
```

Sinclair Spectrum:

```
  60 LET movimiento=movimiento+1:GO SUB
     2320
1380 LET ko=0: LET fin=0
2310 :
2320 REM rutina movimiento blancas
2340 LET y$="          "
2350 LET ip=19: LET im=9: LET iw=4: GO SUB
     1990
2360 IF a$="PASO" THEN LET ko=0: GO SUB
     1730: GO TO 2490
2370 IF a$="ABANDONO" THEN LET
     fin=1:RETURN
2380 LET x=CODE (a$(1))-64
2390 LET z$=a$(2 TO)
2400 FOR c=1 TO LEN z$
2410 LET c$=z$(c)
2420 IF c$ < "0" OR c$ > "9" THEN LET mp=21:
     LET mm=2: LET o$=a$:GO SUB 2160:GO TO
     2350
2430 NEXT c
2440 LET y=VAL z$
2450 IF x<1 OR x>15 OR y<1 OR y>15 THEN LET
     mp=21: LET mm=2: LET o$=a$: GO SUB
     2160:GO TO 2350
2460 LET lp=16*y+x: LET lc=blancas: GO SUB
     3890
2470 IF 11 > 0 THEN LET mp=21: LET mm=11:
     LET o$=a$: GO SUB 2160: GO TO 2350
2480 LET mmp=lp: LET mmc=blancas: GO SUB
     3630
2490 LET mp=21: LET mm=1: LET o$="": GO
     SUB 2160
2500 RETURN
2510 :
2520 REM ************************
2530 :
3630 REM rutina efectuar movimiento
3640 LET n=0
3650 POKE tablero+mmp,mmc
3660 FOR k=1 TO 4:LET a=mmp+d(k): IF PEEK
     (tablero+a)<>color-mmc THEN GO TO 3700
3670 LET cp=a: LET cc=color-mmc: GO SUB 4040:
     IF clib=0 THEN LET ko=a: LET n=n+cstn:
     LET rp=a: LET rc=color-mmc: GO SUB 3780
3680 IF clib=1 AND mmc=negras THEN LET
     x$="Atari": LET y$="          "
3690 IF clib=1 AND mmc=blancas THEN LET
     y$="Atari"
3700 NEXT k
3710 IF n<>1 THEN LET ko=0
3720 LET c(mmc)=c(mmc)+n
3730 GO SUB 1730
3740 RETURN
3750 :
3760 REM ************************
3770 :
3780 REM rutina suprimir
3790 IF PEEK(tablero+rp)<>rc THEN
     RETURN
3800 POKE tablero+rp,0
3805 LET s(pila)=rp: LET pila=pila+1
3810 LET rp=s(pila-1)+d(1):GO SUB 3780
3820 LET rp=s(pila-1)+d(2):GO SUB 3780
3830 LET rp=s(pila-1)+d(3):GO SUB 3780
3840 LET rp=s(pila-1)+d(4):GO SUB 3780
3845 LET pila=pila-1: LET rp=s(pila)
3850 RETURN
3860 :
3870 REM ************************
3880 :
3890 REM rutina legalidad
3900 LET 11=0: LET lk=0
3910 IF PEEK (tablero+lp)<>0 THEN LET
     ll=3:RETURN
3920 POKE tablero+lp,lc
3930 FOR k=1 TO 4
3940 LET la=lp+d(k)
3950 IF PEEK (tablero+la)=color-lc THEN LET
     cp=la: LET cc=color-lc: GO SUB 4040 : IF
     clib=0 THEN LET lk=lk+cstn
3960 NEXT k
3970 IF lk=0 THEN LET cp=lp: LET cc=lc: GO SUB
     4040: IF clib=0 THEN LET ll=5
3980 POLE tablero+lp,0
3990 IF lp=ko AND lk=1 THEN LET ll=4
4000 RETURN
4010 :
4020 REM ************************
```



Enterados

La Interface 1 del Spectrum también puede servir para acceder a una red de área local. Veamos cómo

Ya hemos analizado el modo cómo un programador en lenguaje máquina puede emplear las facilidades de la Interface 1 para controlar microdrives o acceder a la puerta serial RS232. En lo que llevamos dicho, se han visto ya los principios de trabajo en red así como ofrecimos un sencillo juego de red para el Spectrum. Nos ceñiremos ahora al estudio de las facilidades disponibles para un programador en lenguaje máquina. En nuestro análisis emplearemos el siguiente convenio en la denominación de las máquinas concernientes a una red: SELF representa la máquina de usted, IRIS es otra máquina distinta de la red con la que usted desea comunicar.

Transmisión por red

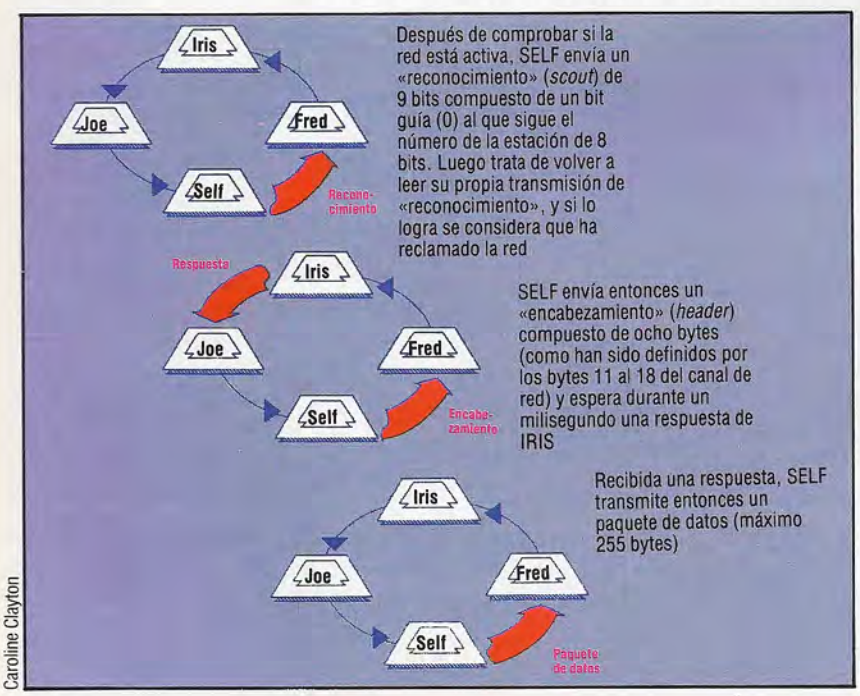
Hay dos maneras de transmitir programas y datos por una red. La primera es la *difusión* (*broadcast*) por las ondas. Toda máquina conectada a la red la puede «oír» si está «sintonizada» cuando se realiza la emisión. Se trata de un modo muy eficaz de distribuir un programa a varias máquinas con gran rapidez. La segunda manera es la *transmisión dirigida* (*directed transmission*), en la que SELF especifica el IRRIS que ha de recibir la información transmitida. Si el IRIS especificado no está a la escucha, la máquina SELF intentará reiteradamente establecer la comunicación.

El método por el cual las máquinas se comunican a través de la red es, sin duda, curioso. Una máquina que desea transmitir datos lo primero que hace es esperar hasta que la red guarde silencio, o «descanse». Para alejar el peligro de competencias en la red, lo que produciría una «confusión» de datos, cada máquina espera durante un intervalo de tiempo aleatorio después de haber comprobado que la red está libre y antes de intentar reclamarla para su uso. Si la red está todavía activa, esperará de nuevo.

Una vez que la red se encuentra libre, la máquina SELF que está transmitiendo los datos envía un *reconocimiento* (*scout*) sobre la red. Se trata del número de la estación de la máquina transmitido de tal modo que puede ser leído en retorno por SELF en la misma red. Cuando esto se cumple, se dice que SELF ha *reclamado* la red.

Ya puede enviarse un *paquete* de datos. Éste se compone de un encabezamiento, que contiene datos sobre el bloque de datos que sigue y un bloque de datos de 255 bytes. Lo que sucede después es función del tipo de transmisión. Si es una transmisión dirigida, SELF espera durante un milisegundo una señal de respuesta desde el IRIS deseado. Si no se recibe tal señal, tras un retardo de ocho milisegundos, SELF envía el reconocimiento y el encabezamiento de nuevo. No envía el bloque de datos hasta recibir la señal de respuesta. Si la transmisión es un difusión, no hay que esperar respuesta y el bloque de datos se envía sin más a la red.

De emisora a emisora
Una operación sobre red ante todo exige «reclamar» la red, a lo que seguirá la transmisión de los datos (iniciada con la necesaria información de encabezamiento). Este esquema muestra el flujo de datos en una red de cuatro estaciones, donde la estación SELF realiza una transmisión «dirigida» a la estación IRIS



Caroline Clayton

Códigos de enganche y variables de sistema

Consideremos ahora que se está utilizando la red desde el lenguaje máquina. Cuatro son los códigos de enganche relativos y dos importantes variables de sistema de la Interface 1:

- NTSTAT: Se encuentra en la dirección 23749 y se establece con el número de estación de la máquina. La instrucción FORMAT, empleada con la red, establece la variable de sistema. El valor por defecto es uno.
- DSTR1: Ya hemos encontrado esta variable al examinar los microdrives. Está en las direcciones 23766 y 23767 y se establece para que contenga el número de estación del IRIS con la que SELF desea comunicar. Poniéndola a cero, dará una transmisión por difusión de los datos.

Respecto al empleo actual de los códigos de enganche, volvemos a un sistema similar al usado con los microdrives, donde hay que abrir un canal de red antes de hacer otra cosa. Un canal de red se establece con el código de enganche 45 (en el dibujo mostramos su estructura). El canal tiene una longitud de 276 bytes, y se inserta en el «área de datos de canal» de la memoria como cualquier otro canal.

El encabezamiento enviado a la red se compone

de todos los bytes que van desde NCIRIS a NCHCS inclusive. El bloque de datos enviados por la red se compone de los bytes 21 al 275 del espacio de canal. Los bytes 19 y 20 de los datos de canal son sólo relevantes cuando el canal está siendo usado para leer datos desde la red. Como para el sistema de canal del microdrive, la escritura y lectura del canal puede hacerse fácilmente dando su dirección a CURCHL y después empleando RST #10 y CALL #15E6 (como con los microdrives). Sin embargo, aquí los datos y el encabezamiento se ponen en la red cuando queremos escribir el byte número 256 en el área de datos de canal. Veamos ahora los códigos de enganche:

- *Código de enganche 45*: Abre un canal de red. Para usarlo, se establece NTSTAT con el número de estación de SELF, y DSTR1 se establece con el número de estación de IRIS. Esto sirve cuando el canal ha sido abierto para lectura o escritura; pero si está leyendo, es evidente que IRIS transmitirá los datos a SELF.

Nuevamente hay que asegurarse de que existen las variables de sistema de la Interface 1. Entonces el código de enganche puede ser usado de la manera habitual, y al retorno el registro IX contiene la dirección de inicio del canal que se ha establecido.

Como ya se ha mencionado, los bytes pueden ser enviados a lo largo de la red por medio de RST =10 y ser recibidos por CALL #15E6. Sin embargo, cuando son empleados con la red, el registro IX puede alterarse. Pero como el contenido de este registro, o al menos la dirección del canal, se necesitará para cerrar el canal una vez hayamos terminado con él, interesa conservar el contenido de IX antes de usar estas rutinas.

- *Código de enganche 46*: Cierra el canal de red, y es llamado con la dirección de inicio del canal tomado del registro IX. Si el canal ha sido usado para escritura, cualquier dato que todavía esté en el área de datos de canal queda escrito en la red. Una vez cerrado un canal, la memoria entre el final del canal y STKEND se desplaza hacia abajo.

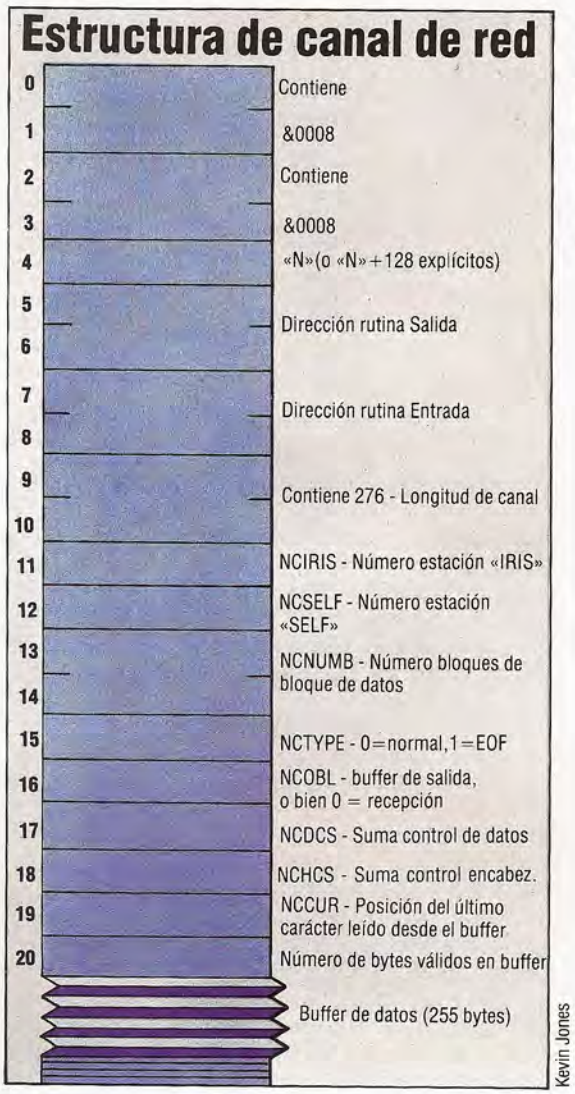
- *Código de enganche 47*: Permite la lectura de un paquete específico desde la red. La variable de canal NCNUMB tendrá el número adecuado de bloque: se supone que NCSELF y NCIRIS contendrán los valores apropiados (generalmente se establecen en el acto de abrir un canal). Tras el empleo del código de enganche, se hará un ensayo de recepción del paquete. Al retorno, el flag C se pondrá a uno si ocurre un final de tiempo, o si los datos estuvieren alterados de alguna manera y no concuerdan las sumas de comprobación. Así, inspeccionando el estado del flag de arrastre (*carry*) puede pedir repetidas veces un mismo paquete de la red. Si todo va bien, y el paquete es recibido, entonces se envía un asentimiento si es necesario y se incrementa NCNUMB.

- *Código de enganche 48*: Este código transmite un paquete de datos a través de la red, y por eso ha de considerarse como la rutina principal de la transmisión de datos. Se emplea con un canal abierto, cuya dirección se pasa a la rutina desde el registro IX. El registro A se activa para contener el tipo de archivo del bloque de datos que está siendo enviado desde el canal. Se pone a uno cuando el bloque que se está enviando es un bloque de datos *end of file* (fin de archivo); de lo contrario, para el bloque de datos normal, estará a cero. Los datos son enviados como difusión o como transmisión dirigida, según el valor contenido en DSTR1 cuando es abierto el canal. En el «extremo receptor», si se aceptan los datos, se leerá el encabezamiento en el área de encabezamiento del canal y quedarán disponibles en NCOBL el número de bytes enviados, indicando NCTYPE si el paquete recibido es o no un paquete *end of file*.

Si se consigue una transmisión acertada, la variable de canal NCNUMB añadirá una unidad a su valor.

Cuando se emplea la red LAN (*local area network*) se puede, pues, escoger entre dos maneras diferentes de enviar y recibir datos según sea la aplicación. Si está en una red pequeña, quizá con una sola máquina distinta de la suya, el método disponible a través de los códigos de enganche 48 y 47 puede quedar algo «saturado», por lo que el sencillo empleo de RST #10 y CALL #15E6 bastará.

Con esto completamos nuestro examen de la Interface 1 en su capacidad para comunicarse con otros ordenadores o periféricos. En un próximo capítulo estudiaremos el empleo de la Interface 1 para aumentar el número de las instrucciones del BASIC del Spectrum.



Estructura de canal de red
El código de enganche 45 establece un canal de red de 276 bytes, de los cuales 20 bytes se emplean como información de encabezamiento y 255 bytes son un buffer de los datos para ser transmitidos. Este esquema muestra la disposición de los bytes en el encabezamiento del canal

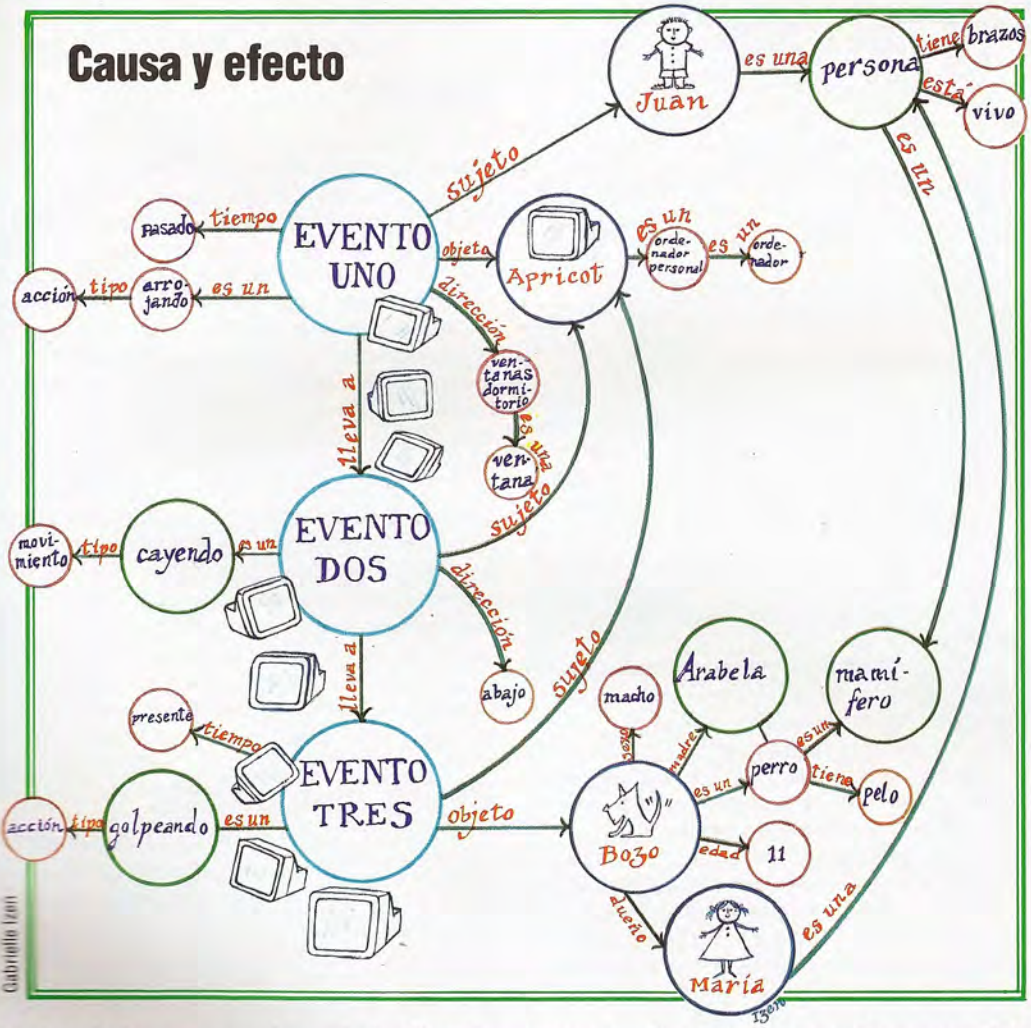
Datos básicos (IV)

Por cortesía de la Commodore Business Machines, reproducimos otro fragmento del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
DFLTN	0099	153	Dispositivo entrada por defecto (0)
DFLTO	009A	154	Dispositivo salida (CMD) por defecto (3)
PRTY	009B	155	Paridad carácter cinta
DPSW	009C	156	Flag: byte de cinta recibido
MSGFLG	009D	157	Flag: $80 = Modo directo, $00 = Programa
PTR1	009E	158	Log error paso 1 de cinta
PTR2	009F	159	Log error paso 2 de cinta
TIME	00A0-00A2	160-162	Reloj instantáneo tiempo real 1/60 seg (aprox.)
	00A3-00A4	163-164	Área datos temporales
CNTDN	00A5	165	Contador sincr. de cassette
BUFPNT	00A6	166	Puntero: buffer E/S cinta
INBIT	00A7	167	Bits entrada RS-232 / Temp. cassette
BITCI	00A8	168	Contador bits entrada RS-232/Temp. cassette
RINONE	00A9	169	Flag RS-232: comprobación bit inicio
RIDATA	00AA	170	Byte entrada RS-232 buffer/temp. cassette
RIPRTY	00AB	171	Paridad entrada RS-232/contador corto cassette
SAL	00AC-00AD	172-173	Puntero: Buffer cinta/ desplazamiento pantalla
EAL	00AE-00AF	174-175	Direcc. fin cinta / fin de programa
CMP0	00B0-00B1	176-177	Constantes temporización cinta
TAPE1	00B2-00B3	178-179	Puntero: inicio de buffer cinta
BITTS	00B4	180	Contador bit fuera RS-232 / temp. cassette
NXTBIT	00B5	181	Siguiente bit a enviar RS-232 / Flag EOT cinta
RODATA	00B6	182	Buffer byte fuera RS-232
FNLEN	00B7	183	Longitud de nombre actual archivo
LA	00B8	184	Número actual archivo lógico

El conocimiento

Examinemos algunas de las maneras en que se representa la información en los sistemas de inteligencia artificial



Secuencia semántica
Las redes semánticas se utilizan para conectar acciones y objetos con el fin de describir sus relaciones. La red que vemos aquí representa: «Juan arrojó el ordenador personal por la ventana. Golpeó al perro de María.» Los círculos son los nudos de objetos y las flechas son los arcos de relaciones. A partir de los enlaces semánticos dados se podría deducir, por ejemplo, que «un ordenador golpeó a un mamífero peludo» o que «algo cayó sobre el hijo de Arabela». En un sistema real, la red sería muchísimo más amplia que esta simplificada que ofrecemos aquí

El objetivo de los programadores de inteligencia artificial (AI) es imitar sistemas complejos, como la memoria humana y el raciocinio, tarea en absoluto sencilla. Éste es uno de los motivos por los que hablan de «representación del conocimiento» en vez de «representación de datos».

La mayoría de los programadores poseen una clara idea acerca de lo que se quiere significar con datos. Pero con frecuencia se muestran escépticos a la hora de utilizar la palabra «conocimiento» para describir la información retenida en un ordenador. A la larga, la diferencia entre datos y conocimiento se reduce a una diferencia de énfasis, proveniente de la naturaleza de los problemas con los que se enfrentan los programadores de AI y las soluciones que idean. Es útil analizar los cuatro componentes principales que distinguen al *conocimiento* de los *datos*.

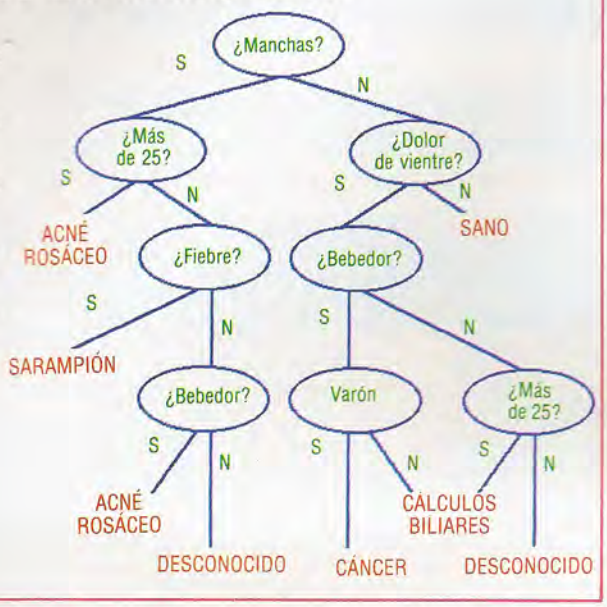
En primer lugar, las representaciones del conocimiento deben ser flexibles en lugar de estáticas. El conocimiento se debe codificar mediante estructuras que pueden reducirse o ampliarse a medida que se ejecuta el programa (utilizando una asignación dinámica de memoria) y no mediante estructuras cuyo tamaño y forma sean fijos para la duración de la ejecución. Lamentablemente, la mayoría de las versiones de BASIC no proporcionan estructuras de datos dinámicas.

En segundo lugar, la representación del conocimiento requiere estructuras de niveles múltiples o estratificadas en lugar de estructuras de un solo nivel. Por ejemplo, las listas pueden contener sublistas; los árboles, contener subárboles; las reglas, aludir a subreglas, y así sucesivamente.

En tercer lugar, las representaciones del conocimiento son típicamente heterogéneas: contienen una mezcla de tipos de datos. Por ejemplo, quizá usted desee establecer un registro para describir a un personaje de un juego de aventuras. El mismo tendría que contener algunas series (un nombre), algunos números (p. ej., edad y altura), señaladores a otros registros (como aquellos de los amigos y

Los caminos hacia la receta

Los árboles de decisión son fáciles de construir y emplear. Sin embargo, se basan en la absoluta precisión de la información utilizada para seguir una cierta ruta a través del árbol. Si los datos son inciertos (como suele ser el caso en la vida real) una representación incorrecta puede conducir a la zona incorrecta del árbol, dando como resultado un diagnóstico erróneo



Caroline Clayton

enemigos de este personaje), reglas que describan el comportamiento de este personaje en situaciones típicas, y muchas otras cosas más. Esto se puede hacer en BASIC, pero no es fácil. Ello se debe a que el principal método para organización de datos (la matriz) sólo puede contener series o números, pero no ambos. Y, ciertamente, no puede contener matrices ni otros objetos de datos exóticos tales como elementos.

Por último, y tal vez lo más importante, las representaciones del conocimiento son activas, mientras que las estructuras de datos son pasivas. Efectivamente, los sistemas basados en el conocimiento, han convertido la separación en dos hileras —programa y datos—, en las tres hileras que son los hechos, las reglas y un motor de inferencias.

Los hechos corresponden claramente a los datos. El motor de inferencias es, obviamente, un programa. Pero las reglas se identifican un poco con ambos. Constituyen *datos activos*. Habiendo esbozado las diferencias clave entre datos y conocimiento, veamos algunas de las formas en que se puede representar y almacenar el conocimiento.

Matrices

La matriz ofrece un método simple para representar el conocimiento. El único problema es que puede resultar demasiado simple para muchas aplicaciones de AI. En el contexto del sistema experto, por ejemplo, imaginemos un paquete simplificado para diagnóstico médico con sólo cuatro diagnósticos (hipótesis) y siete síntomas (evidencias). Sería posible codificar el conocimiento del sistema en forma de una matriz bidimensional (como podemos ver). Este formato de matriz sería adecuado para soportar una estrategia de inferencia sencilla, recogiéndose la evidencia de un elemento por vez en la escala −1 (no), 0 (quizá) y +1 (sí). Una vez producida toda la evidencia, simplemente se la puede multiplicar por los números de las filas correspondientes a cada columna. La columna que registre el total mayor es, entonces, la hipótesis más firmemente sustentada. Surgirían problemas al tratar de ampliar tal representación. Es inflexible: con 200 evidencias y 100 categorías de enfermedades, la matriz habría de contener 20 000 celdas, la mayoría de las cuales estarían vacías. También es demasiado «chata», en cuanto que ignora la estructura jerárquica del conocimiento médico. Las enfermedades se pueden agrupar en tipos. Una vez que un médico sabe que un paciente padece cierta clase de enfermedad, toda una serie de preguntas se vuelven irrelevantes. Una estructura más adecuada sería, en otras palabras, un árbol.

Árboles

Las estructuras arborescentes constituyen una ayuda de incalculable valor para los programadores de AI. Dos aplicaciones importantes para las estructuras arborescentes en el campo de la AI son los *árboles de decisión* y *las jerarquías de herencia*. A modo de ejemplo de la primera estructura arborescente, hemos reorganizado los datos tabulares de nuestro ejemplo anterior en forma de árbol de decisión (véase el diagrama). El problema fundamental de los árboles de decisión en cuanto a la representación del conocimiento es que manipulan muy mal la incertidumbre. Son muy eficientes cuando las pruebas de cada nudo son inequívocas, pero si las respuestas son inciertas, el sistema se puede introducir fácilmente en la parte errónea del árbol. En la vida real muy raramente se puede proporcionar respuestas sí/no definitorias para las preguntas. Los árboles de jerarquía de herencia (véase diagrama de p. contigua) conectan términos y conceptos de modo que la estructura arborescente refleja las relaciones de una forma ordenada. Este tipo de árbol permite realizar inferencias de sentido común cuando se manejan los elementos del árbol.

Redes semánticas

Una red o estructura gráfica es más compleja que un árbol. Un árbol posee un nudo «raíz» especial y todas las ramificaciones se despliegan en una dirección, por lo general hacia abajo. Una red no posee ningún nudo raíz y los enlaces pueden apuntar en cualquier dirección. Las redes semánticas son una clase especial de estructura gráfica que ha sido acogida favorablemente en AI para representar conocimientos acerca del lenguaje y acerca de las acciones y los eventos. En una red semántica, los nudos representan objetos y los arcos o enlaces entre ellos representan relaciones.

La red semántica es una generalización de una construcción del LISP que tuvimos oportunidad de ver en el capítulo anterior: la lista de propiedades. Tanto las listas de propiedades como las redes semánticas codifican la información como pares de valores-atributos unidos a un objeto.

Es básicamente una cuestión de conveniencia; y, de hecho, tanto las listas de propiedades como las redes semánticas se pueden resumir bajo el concepto de *tuples*. Un *tuple-n*, por ejemplo, es simplemente una agrupación de *n* objetos. Las representaciones de objetos-atributos-valores (como redes semánticas y listas de propiedades) emplean ternas, es decir, tuples-3, como en:

Objeto	Atributo	Valor
BOZO	ES-UN	PERRO

Entre la idea de representar el conocimiento como un conjunto de ternas y la noción de «cuadros» hay sólo un corto paso. El cuadro es una estructura muy utilizada en los trabajos de AI, que contiene varias *ranuras*. Cada ranura es como el campo de un registro convencional de un archivo, pero la cantidad de ranuras no es fija.

Por lo tanto, los cuadros son, en esencia, *tuples-n* donde *n* puede variar. La siguiente es una representación tipo cuadro que describe parte de la información de la red semántica que vimos anteriormente, con la adición de algunos hechos más:

```
Nombre:          Evento-3
Tipo:            Golpeando
Tiempo:          Ahora
Sujeto:          PC49
Objeto:          Bozo
Antecedente:     Evento-2

Nombre:          Bozo
Es-un:           —> Perro-prototipo
Dueño:           María
Tipo:            Agente-animado
Sexo:            Macho
Edad:            11
Padres:          (Fido, Arabela)
Implicado-en:    (Evento-3, Evento-7, Evento-20...

Nombre:          Perro-prototipo
Tiene:           Pelo
Sabe:            (Ladrar, Morder, Cazar...)
Patas:           Defecto=4
Ejemplos:        (Bozo, Fido, Arabela, Spot)
Peligroso:       IF pequeño AND dormido OR
                 meneando el rabo
                        THEN No
                        ELSE Sí
Hacer:           IF durmiendo THEN dejarlo
```

Podemos ver aquí que las ranuras se pueden llenar con diversos tipos de datos: Edad posee un número, Padres posee una lista, Peligroso tiene una regla de decisión, Es-un tiene un señalador a otro cuadro, y así sucesivamente. La ranura corresponde a los atributos de una lista de propiedades o a los arcos de una red semántica.

Puesto que cualquier estructura suficientemente flexible puede representar la información de cualquier otra clase de estructura, parecería que la elección entre las representaciones es básicamente cuestión de eficacia. No obstante, a algunos investigadores de AI no les agrada el relativismo que esto implica e intentan definir un formalismo que lo abarque todo y que sea más fundamental que los otros. El formalismo que eligen es la lógica de predicado. La lógica como lenguaje de representación es la base del PROLOG, que utiliza un esquema basado en lo que se denomina *cláusulas cuerno*. Por ejemplo:

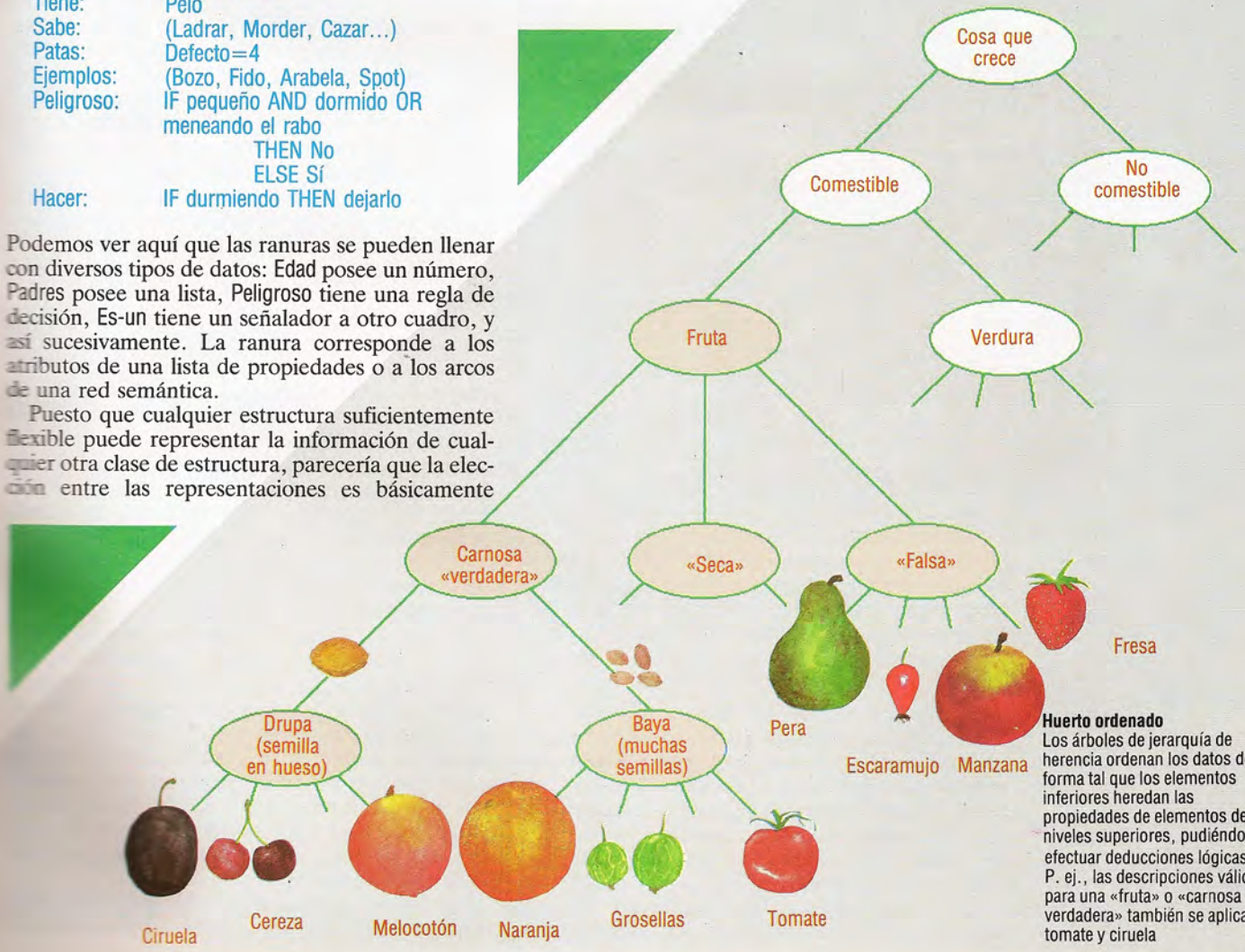
```
peligroso (X):–
        perro (X), no (seguro(X))
seguro (X):–
        pequeño (X), dormido (X)
seguro (X):–
        meneando el rabo (X)
```

es una versión en PROLOG de una de las reglas sobre perros de las estructuras de cuadros anteriores.

En sentido teórico, la lógica es más fundamental que las otras representaciones que hemos analizado. Pero en la práctica, la elección de representaciones viene dictada por consideraciones ajenas a la elegancia teórica.

El ingeniero de conocimiento bien equipado habrá de ser consciente de las variedades de esquemas de representación, que se han utilizado con todo éxito en los proyectos de AI. Adecuar la representación al conocimiento es uno de los artificios de la inteligencia artificial.

Clasificando frutas



Huerto ordenado
Los árboles de jerarquía de herencia ordenan los datos de forma tal que los elementos inferiores heredan las propiedades de elementos de niveles superiores, pudiéndose efectuar deducciones lógicas. P. ej., las descripciones válidas para una «fruta» o «carnosa verdadera» también se aplican a tomate y ciruela

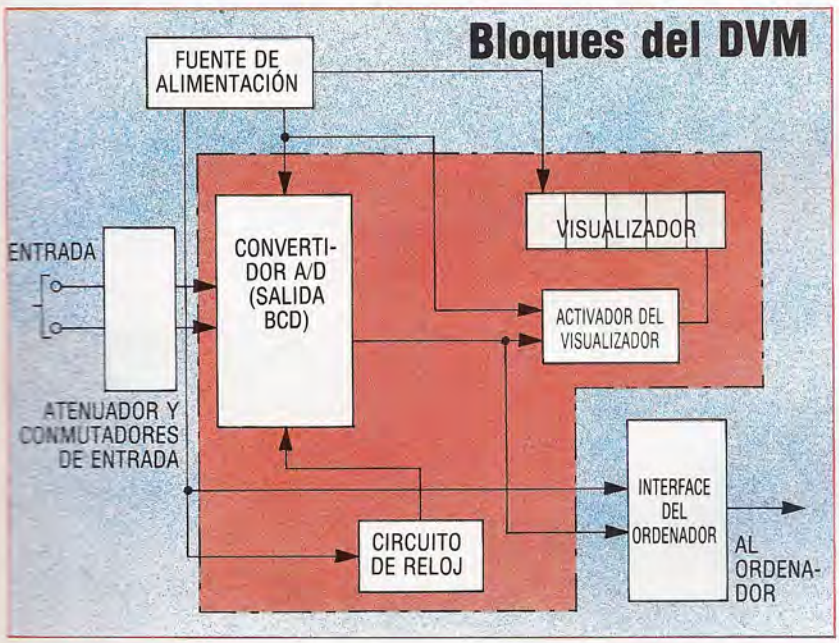
David Devies

De tal palo, tal astilla

En el capítulo anterior estudiamos algunos de los principios de la conversión digital a analógico (D/A) y analógico a digital (A/D). Ahora ofrecemos un detallado análisis del circuito completo del medidor de tensión (DVM) que estamos diseñando en este apartado dedicado al bricolaje

El circuito DVM consta de siete subsecciones o bloques (como se indica en el diagrama). Consideremos cada uno de estos componentes:

- *El atenuador y los interruptores de entrada:* La señal de entrada a medir se alimenta mediante cables con puntas de prueba a los terminales de entrada del DVM, a través del bloque de atenuación y de los conmutadores de entrada. La complejidad del atenuador y de los conmutadores depende de lo amplia que sea la gama de valores a medir (de microvoltios a kilovoltios, p. ej.) y de cuantas unidades eléctricas se midan (voltios CC, voltios CA, corriente CC, ohmios, etc.).
- *La fuente de alimentación:* Este bloque se explica por sí mismo. Todo cuanto se requiere son +5 V y −5 V a una corriente bastante baja, y esto se consigue fácilmente utilizando dos reguladores de tensión de tres terminales, tomando la energía de la red eléctrica o de pilas. Todos los bloques se alimentan, a excepción del bloque atenuador/conmutadores (que sólo consta de componentes pasivos). Es necesario tener cuidado con el cableado de las fuentes de alimentación para evitar que surjan problemas a raíz de bucles a tierra no deseados.



Bloques de construcción
Este esquema ilustra las relaciones existentes entre las diversas secciones del DVM. La zona del esquema de color rojo corresponde al esquema circuital más detallado de la página contigua

- *El circuito del reloj:* El chip convertidor A/D 7135, al igual que casi todos los tipos de convertidores A/D, exige una señal de reloj. La señal de reloj es vital, pero muy simple (a diferencia de las señales de reloj que necesitan algunos microprocesadores) y se puede generar fácilmente mediante un par de puertas TTL (lógica transistor-transistor) con realimentación. Hemos optado por un circuito basado en el versátil chip temporizador 555. Aunque internamente es más sofisticado, los únicos componentes externos necesarios son dos resistencias y un condensador.
- *La interface del ordenador:* Este bloque es una opción extra. El DVM básico se diseñó como unidad independiente, como alternativa de bajo costo a los caros DVM digitales existentes en el mercado. Sin embargo, si usted desea que su ordenador pueda tomar lecturas directas del entorno (en voltios, ohmios, grados de temperatura o lo que sea), entonces se puede añadir este bloque. No afecta a los otros sistemas de circuitos y es opcional.
- *El convertidor A/D:* Éste es realmente el corazón del DVM. Utiliza un único chip configurado para tener una sensibilidad básica de dos voltios (o, para ser precisos, de 1,9999 voltios). Las tensiones medidas más altas se atenúan mediante el atenuador de entrada, de modo que el chip 7135 no recibirá jamás una tensión de entrada superior a dos voltios. Este chip ofrece numerosas ventajas sobre otros convertidores A/D de un solo chip. Convierte la tensión de entrada en una salida BCD (*binary coded decimal:* decimal codificada en binario) que al ordenador le resulta más fácil de leer que las salidas codificadas en siete segmentos de los chips A/D diseñados para activar directamente visualizadores LCD o LED de siete segmentos. Aparte de esto, el chip 7135 posee otras varias configuraciones que lo hacen especialmente atractivo para un proyecto de esta naturaleza. Es sumamente preciso; la precisión básica es de ±1 puntos sobre 20 000 y puede dar lecturas para cuatro lugares decimales en el margen de dos voltios. Posee una lectura de cero garantizada en la visualización para una entrada de 0 V. Tiene *indicación de autopolaridad,* de modo que se pueden leer tensiones tanto positivas como negativas sin tener que invertir las puntas de prueba. Tiene dos formas de indicar una entrada por sobre la escala (unna entrada de más de 1,9999 voltios): ya sea haciendo parpadear la visualización o bien utilizando una línea de señal separada que puede activar un LED para que indique una situación de superación de margen.

El chip posee, asimismo, una impedancia de entrada muy elevada; típicamente la corriente de entrada es de 1 pA (una millonésima, o 1×10^{-12}, de amperio). Una impedancia de entrada tan alta como ésta aplica al circuito que está midiendo una carga muy baja. Por consiguiente, el drenaje de corriente en el instrumento medido no hace caer de forma importante la tensión que se está tratando de medir. No obstante, nuestro atenuador de entrada se ha diseñado por razones de simplicidad y disponibilidad de resistencias de tolerancia restringida, y no obtiene el máximo partido de la altísima impedancia de entrada de este chip.

El chip 7135 también configura una salida por debajo de margen. En nuestro simple diseño, ésta se podría utilizar para activar un LED que mostrara que se habría de seleccionar una posición del conmutador para una escala menor. En un DVM más sofisticado, las salidas por encima y por debajo de margen se podrían utilizar para una *escala automática* (un sistema en el cual la atenuación de entrada y la posición del punto decimal en la visualización se sitúan de forma automática en función del nivel de la señal de entrada).

Los niveles de tensión de salida son compatibles directamente con la TTL, y esto hace que la cone-

sible con interface a microordenadores sea comparativamente simple. Aún más importante es el hecho de que se proporcionan varias señales específicamente para simplificar la conexión con interface al ordenador. Éstas incluyen una línea STROBE para sincronizar la transferencia de datos a los enclavamientos externos (registros que pueden retener datos sin necesidad de suministrar una entrada constante), una línea de entrada RUN/HOLD que le permite al ordenador «congelar» la lectura u operar de forma independiente, y una línea BUSY que le dice a la máquina si las otras salidas son válidas o no. Las cuatro líneas de salida BCD sólo poseen valores significativos en ciertos momentos durante el proceso de conversión A/D, y una línea BUSY activa indica que las lecturas de las líneas de salida no son fiables.

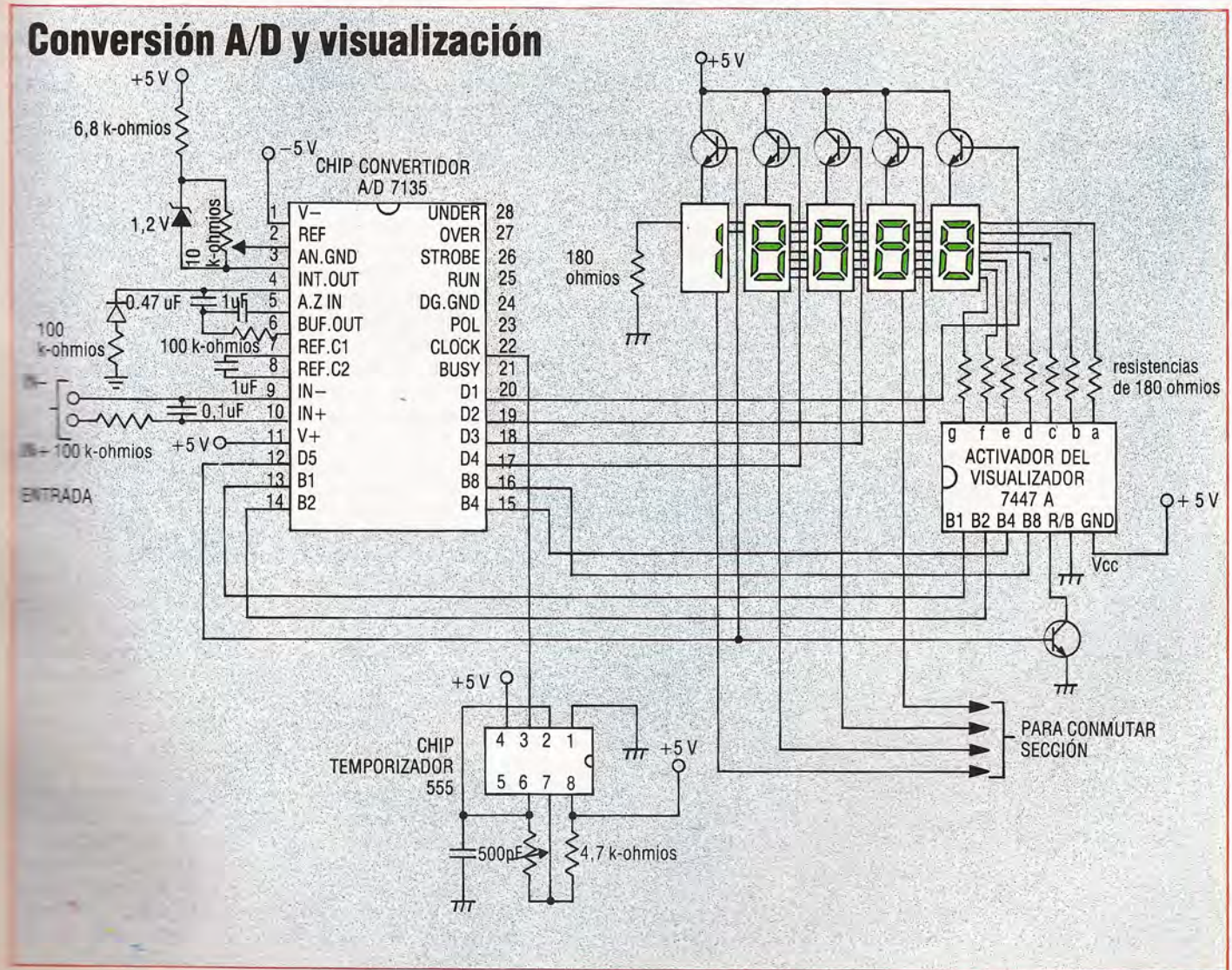
Ante todo, el chip 7135 es sumamente versátil. Muchas de las señales de entrada y de salida disponibles se pueden ignorar, pero existen y están listas para usar si se requiere un sistema más sofisticado.

- *El activador del visualizador:* Dado que el convertidor A/D 7135 proporciona una salida BCD, no se puede utilizar directamente para activar el visualizador. Afortunadamete, un chip individual muy económico es todo cuanto se necesita para conectar con interface el convertidor A/D a un visualizador apropiado. El chip 7447A, por el cual hemos optado, puede convertir una señal BCD en la combinación correcta de siete señales necesaria para iluminar correctamente un LED único de siete segmentos. La salida del 7135 es multiplexada. Es decir, cada dígito de la visualización comparte las cuatro señales de salida BCD, pero éstas sólo son válidas para uno de los cinco dígitos por vez. Al mismo tiempo, las señales de dígitos válidos del chip 7135 indican qué digito se está produciendo en BCD en cada momento. Estas señales de dígitos válidos se utilizan directamente para conmutar transistores, habilitando uno de los LED por vez. Dado que las visualizaciones de dígitos se encienden y se apagan con tanta rapidez, las cinco visualizaciones parecen brillar al mismo tiempo.

- *El visualizador:* En la actualidad las LCD están de moda, en especial para los instrumentos portátiles que funcionan a pila. Sin embargo, los LED todavía poseen algunas ventajas, y hemos optado por una visualización de LED. Los LED son algo más simples que las LCD para las conexiones con interface, particularmente cuando se trata de conmutación de puntos decimales. Los LCD requieren una señal de CA de plano posterior, y en el circuito DP (*decimal point:* punto decimal) se requiere un sistema adicional de circuitos lógicos.

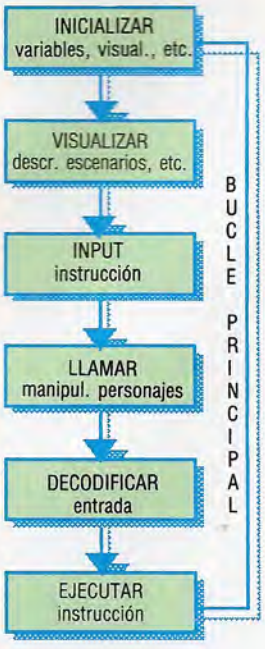
Convertidor kernel
La base de nuestro diseño DVM es el chip convertidor A/D 7135. Este chip lleva a cabo sus operaciones en respuesta a señales de reloj externas que le proporciona el chip 555 de la parte inferior del esquema. El 7135 produce un valor digital en forma BCD que se puede utilizar en conjunción con un chip 7447A, multiplexor activador del visualizador, y cinco transistores de conmutación para iluminar un visualizador LED de 4½ dígitos. En próximos capítulos iremos ofreciendo detalles de construcción completos. Aconsejamos a los lectores que no intenten empezar la construcción en esta etapa, puesto que el diseño no está completo todavía

Conversión A/D y visualización



Kevin Jones

Control del reparto



Caroline Clayton

Posición estratégica
Este diagrama de flujo de un típico juego de aventuras muestra una posible posición para nuestra rutina manipuladora de personajes. Incorporada en el programa en este punto del flujo de control, la rutina se ejecutará cada vez que el jugador pulse ENTER tras haber digitado una instrucción

Para ejercer dominio sobre los personajes podemos recurrir a dos métodos distintos

En nuestro primer capítulo de esta serie vimos cómo los personajes controlados por ordenador del juego *Suspect,* de Infocom, podían desplazarse de un escenario a otro, aparentemente «a voluntad». El movimiento, sin embargo, es una de las características menos importantes de una entidad controlada por ordenador, y el programa proporciona ejemplos excelentes de la interacción jugador/personaje más relevante en su sintaxis de instrucciones. Al jugar a *Suspect,* se pueden utilizar instrucciones que permitan dirigirse a los personajes de numerosas formas diferentes. El programa, por ejemplo, aceptaría todas estas entradas:

Michael, ¿dónde está Veronica?
Acusar al coronel Marston de asesinato
Llamar a mi abogado
Johnson, háblame de ti

Es obvio que existe una esfera de acción para que usted se comunique con los otros invitados en la mansión Ashcroft. Pero lo que hace que el juego resulte particularmente atractivo son los numerosos mensajes que se generan en respuesta a los intentos del jugador por mantener conversaciones amables. Aunque los personajes tienden a ser ligeramente repetitivos, aun así dan prueba de una insólita coherencia y sentido en sus respuestas:

Michael, háblame del caballo
«*Lurking Grue* es el saltador premiado de Veronica. Realmente es un animal muy hermoso.»
Marston, háblame del caballo
«No sé nada acerca de él que pueda interesarle.»

Además, los personajes no sólo se pueden comunicar con el jugador sino que también pueden hablar entre sí. En realidad, no es probable que éste llegue muy lejos a menos que se pasee por el salón de baile e intervenga en conversaciones tales como:

Michael se une a la conversación. «Recuerdo un semental negro que el año pasado se vendió a un precio muy alto. Probablemente haya superado las cien mil.» El coronel Marston lo mira con indignación y dice: «Tengo buena memoria para los números. El precio máximo del año pasado fueron noventa y dos mil.» Cochrane le lanza una mirada a Michael, sintiéndose traicionado. «Tonterías», dice enfadado...

Resumiendo, las «personas» de *Suspect* satisfacen varias de las exigencias que suelen estipularse para unos auténticos personajes interactivos:

1. Pueden desplazarse de un escenario a otro.
2. El jugador puede dirigirse a ellos y obtener una respuesta con sentido.

También pueden recoger y dejar objetos; de hecho, en cierto momento del juego ¡un personaje llega realmente hasta el punto de esconder un objeto dentro de otro objeto! Por último, los personajes pueden dirigirse al jugador sin ser requeridos en ese sentido. Existen muchos ejemplos de ello durante el juego, produciéndose los más obvios cuando usted es arrestado por el detective (si no ha conseguido hallar al culpable).

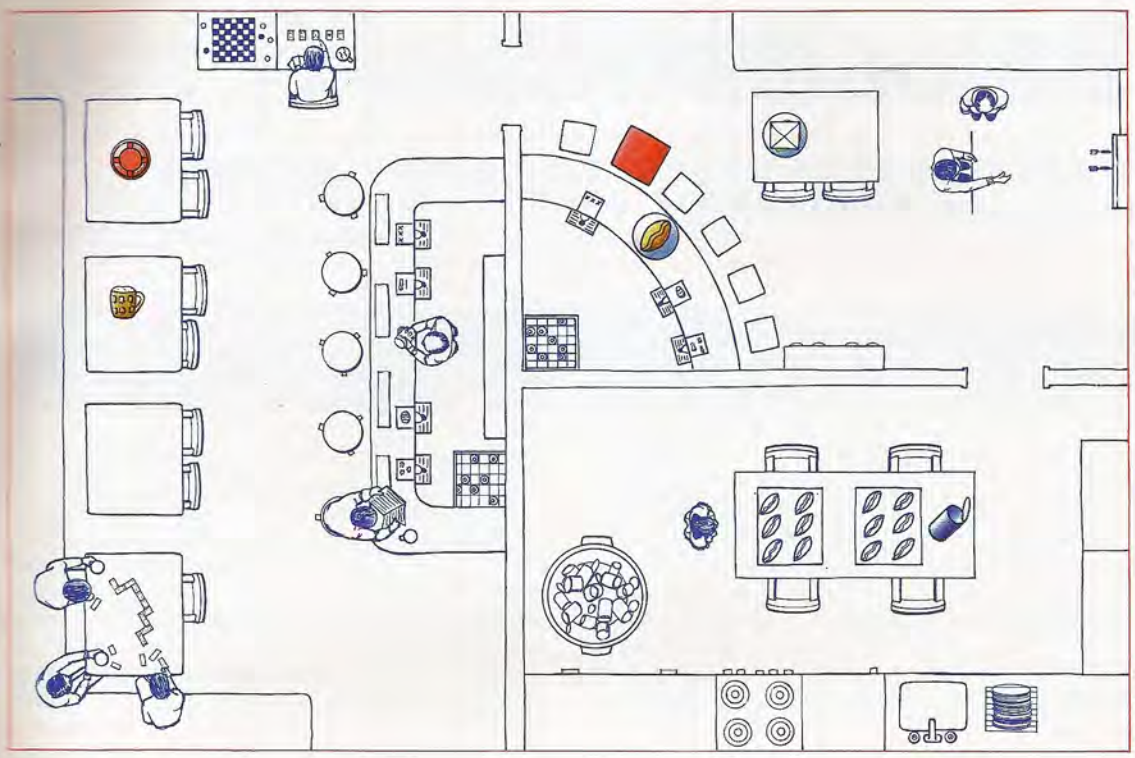
Antes de pasar al siguiente paso (programar nuestra propia rutina de manipulación de personajes) es interesante mencionar brevemente un inconveniente del diseño de un juego que depende mucho del comportamiento y las acciones de los personajes controlados por ordenador. La trama de *Suspect,* por ejemplo, exige que en algún momento «alguien» sea asesinado y que, en otros momentos del juego, los personajes lleven a cabo acciones significativas que, en caso de que el jugador las observe o las deduzca, permitan resolver el misterio.

Por este motivo, los personajes no pueden conducirse de forma totalmente aleatoria. Esto significa que, tras ejecutar varias veces el programa, el jugador enseguida caerá en la cuenta de que, por ejemplo, el personaje A estará en el lugar B en el momento C realizando la acción D. Más adelante veremos que esta necesidad de que las personas realicen acciones específicas para poder cumplimentar la trama, requiere una especial consideración al construir una rutina eficiente para manipulación de caracteres. En *El Hobbit,* por ejemplo, los personajes manifiestan en su conducta un nivel considerable de azar. Esto podría significar que están algo limitados en cuanto a su capacidad para comportarse coherente e inteligentemente, pero por regla general un nivel de azar elevado le confiere a un programa una atmósfera propia de «vida real».

En *Suspect,* por otra parte, existe una sensación (después de haberlo jugado varias veces) de *déjà vu* cuando la Reina de las Hadas sale del salón de baile para limpiarse unas gotas de vino que le han derramado sobre el vestido, sólo para acabar como un cadáver en el despacho. El programador encontrará que equilibrar el grado de azar de la conducta de un personaje con la necesidad de coherencia y pertinencia, constituye uno de los aspectos más difíciles de la programación de una buena aventura.

Dos consideraciones

Habiendo analizado *Suspect* de forma bastante detallada, consideremos nuestras propias rutinas. La primera decisión a tomar es de carácter general: ¿cuán complejo queremos que sea nuestro programa? Al diseñar una aventura, por lo general resulta bastante fácil decidir, por ejemplo, cuántos escenarios queremos que haya; pero, cuando se trata de los personajes, las combinaciones son casi infinitas. Un juego como *Suspect* exige grandes cantidades de almacenamiento de datos sólo para los mensajes que utilizan los personajes, por no hablar de las rutinas para controlarlos. Infocom aborda este problema escribiendo sistemas basados sólo en disco, cargando los datos cuando es necesario. Pero para la mayoría de los usuarios europeos los discos constituyen un lujo y, por tanto, todos los programas deben ejecutarse enteramente en RAM.



Kevin Jones

Operadores locales
Nuestra rutina manipuladora de personajes se desarrollará en Dog and Bucket, un *pub* inglés de dudosa reputación, famoso por su cerveza de tonel y sus excelentes empanadas caseras. Aquí vemos la disposición de los escenarios utilizados en la rutina, junto con las posiciones iniciales de algunos de los objetos más importantes. En el próximo capítulo le mostraremos cómo programarlos en la rutina y le añadiremos personajes a nuestra ilustración a medida que se presenten

Éste no es un factor tan limitador como podría parecer, tal como veremos en el próximo capítulo cuando examinemos *Sherlock,* de Melbourne House, un juego basado en RAM con una manipulación de personajes bastante sofisticada. No obstante, puesto que conviene que las cosas sean moderadamente sencillas, diseñaremos el programa para manipular un máximo de 7 personajes.

La segunda pregunta que es necesario responder no es tan directa; pero, nuevamente, es necesario contestarla antes de iniciar la programación. Existen dos métodos para controlar los personajes de un juego *(síncrono* y *asíncrono)* y hemos de decidir cuál de los dos utilizar. Para explicar cabalmente el significado de estos términos y las implicaciones que tendrá nuestra elección, es preciso refrescar la memoria respecto a qué hace exactamente un juego de aventuras y cómo lo hace.

A partir del diagrama se aprecia que cada vez que el jugador digite una entrada y pulse ENTER, el ordenador la decodificará, llamará a la rutina requerida, actualizará las variables del sistema necesarias, imprimirá mensajes, y luego retornará para aguardar otra entrada. La pregunta a la que hemos de responder es: ¿en qué lugar de esta estructura ubicaremos a nuestro manipulador de personajes?

Volviendo a los términos que mencionábamos anteriormente, un sistema síncrono de personajes ejecutaría la rutina de manipulación de personajes en una etapa fija del programa (como vemos en el diagrama). Aquí, cada vez que el jugador pulse ENTER se llamará a la subrutina manipuladora de personajes. Éste es un sistema síncrono, y *Suspect* representa un buen ejemplo de este tipo de programas. Ofrece ciertas ventajas para el jugador; por ejemplo, si usted se aparta del teclado para ir a servirse una taza de té, tendrá la total seguridad de no ser atacado (tal vez) por otro personaje controlado por el ordenador mientras permanezca en la cocina. Por la misma razón, significa que el ritmo del juego tiende a depender excesivamente de las entradas del jugador.

Por el contrario, un sistema asíncrono por lo general será activado por interrupciones y pasará el control al manipulador de personajes de vez en cuando, independientemente de si usted se halla o no frente al teclado. Éste es el sistema que utilizan *El Hobbit* y *Valhalla,* en los cuales, si el jugador se reclina en la silla y observa la pantalla, verá personajes yendo y viniendo, llevando su propia vida de forma bastante independiente. Este sistema posee la ventaja de conferirle «atmósfera» al juego, pero también es más difícil de programar. En particular, usted ha de tener cuidado en asegurarse de que no exista ningún conflicto variable entre su manipulador de personajes y el programa principal, ya que de lo contrario podría encontrarse con graves problemas.

Utilizar rutinas activadas por interrupciones desde BASIC es muy difícil, a menos que el jugador posea un Amstrad o un micro MSX: por este motivo adoptaremos para nuestra rutina el sistema síncrono. Sin embargo, diseñaremos el programa de modo tal que usted pueda reclinarse en su asiento y mirar cómo se ejecuta sin necesidad de ir introduciendo instrucciones.

Estamos ahora en condiciones de iniciar la programación. Aunque el propósito es construir un módulo transportable que se pueda adaptar para ejecutarlo con juegos de aventuras, como nuestros programas *Digitaya* y *El bosque encantado,* aun así necesitamos proveer un entorno en el cual probar la rutina manipuladora de personajes. Por lo tanto, crearemos una aventura sencilla con tres escenarios en los cuales puedan moverse nuestros personajes, y con numerosos objetos a manipular.

Dado que el programa está diseñado para poner de relieve a nuestros personajes interactivos, desarrollaremos la acción en un jovial lugar de encuentro: el *pub* Dog and Bucket (El perro y el cubo).

Datos básicos (V)

Continuamos con nuestro exhaustivo análisis del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
SA	00B9	185	Dirección actual secundaria
FA	00BA	186	Número dispositivo actual
FNADR	00BB-00BC	187-188	Puntero: nombre actual del archivo
ROPRTY	00BD	189	RS-232 salida paridad/temp. cassette
FSBLK	00BE	190	Cuenta bloques lectura/ escritura cassette
MYCH	00BF	191	Buffer palabras en serie
CAS1	00C0	192	Enlace motor cinta
STAL	00C1-00C2	193-194	Dirección inicio entrada/ salida
MEMUSS	00C3-00C4	195-196	Temps. carga cinta
LSTX	00C5	197	Tecla actual pulsada: CHR$(n)0 = No hay tecla
NDX	00C6	198	Número de caracteres en buffer teclado (cola)
RVS	00C7	199	Flag: imprime caracteres invertidos: 1 = sí; 0 = no empleados
INDX	00C8	200	Puntero: fin de línea lógica en INPUT
LXSP	00C9-00CA	201-202	Posición cursor X, Y inicio INPUT
SFDX	00CB	203	Flag: imprime caracteres con *shift*
BLNSW	00CC	204	Activa parpadeo cursor: 0 = cursor en flash
BLNCT	00CD	205	Temporizador: cuenta el cursor *toggle*
GDBLN	00CE	206	Carácter bajo el cursor
BLNON	00CF	207	Flag: último parpadeo cursor *on/off*
CRSW	00D0	208	Flag: INPUT o GET desde teclado
PNT	00D1-00D2	209-210	Puntero: dirección actual línea pantalla

Dos «conversadores»

Démosle una mirada a dos nuevos sintetizadores de voz y escuchemos qué nos dicen

Los sintetizadores de voz para ordenadores han experimentado un constante progreso durante estos últimos años, y en la actualidad algunos de estos avances están empezando a implementarse en micros personales. El tipo de sintetizador más antiguo tenía en su memoria una lista de «palabras». Cuando recibía algún texto a pronunciar, comparaba la palabra que había recibido con las que tenía en la memoria. Tras hallar una pareja, llamaba a una rutina que producía la serie de sonidos que, unidos entre sí, formaban la palabra requerida. El sintetizador de voz moderno es bastante más sofisticado. En estos sistemas, el usuario puede digitar cualquier frase y el ordenador «pronunciará» las palabras.

Los problemas inherentes a la obtención de una síntesis de voz correcta y amplia utilizando el enfoque texto-habla son enormes. La complejidad de lenguaje es de por sí suficiente para desbordar aun el programa más sofisticado. Tomemos, por ejemplo, en inglés, la palabra *read* (leer). Este vocablo se puede pronunciar tanto «red» (leí, leído), como «rid» (leer, lee), según el contexto. Ejemplos como éste significan que los microordenadores pueden, en el mejor de los casos, tomar una palabra e intentar la pronunciación correcta. Sin embargo, aunque de ningún modo son perfectos, los nuevos sistemas pueden proporcionar resultados notables.

El sistema opera leyendo en un tampón una serie de caracteres ASCII. Luego se toma cada palabra y se coteja con una serie de «reglas» gramaticales que ha establecido el programador. Partes de la palabra corresponderán a ortografías particulares para las cuales se ha establecido una regla. Por ejemplo, en inglés, una de éstas podría ser que si hay una vocal seguida de una consonante seguida de «ie» o «y», luego la primera vocal debe producir un sonido largo. Mediante la implementación de esta regla, la palabra *vary* (variar) se pronunciaría correctamente. No obstante, el problema del idioma, y en particular del inglés, es que su implementación no está normalizada. En consecuencia, al encontrarse con una palabra como *many,* la regla del sintetizador de voz determinará que la pronunciación correcta de la palabra es «meini». Por lo tanto, muchos programadores incluyen asimismo una lista de excepciones comunes a las reglas, aunque, por supuesto, es imposible incluirlas todas en el limitado espacio de que se dispone en un microordenador. Aun cuando el ordenador tuviera un espacio ilimitado de almacenamiento, el tiempo de procesamiento necesario ocasionaría demoras inaceptables para la producción de la voz.

Las dos máquinas descritas aquí son Namal Type & Talk, para el BBC Micro, y el Amsoft Speech Synthesiser, para ordenadores Amstrad: ambas utilizan el sistema de «reglas» para decodificar el texto.

Type & Talk

Type & Talk es el último de una larga línea de periféricos para el BBC Micro que parecen estar dirigidos al sector educativo. El dispositivo viene en una caja metálica color ante, similar al del ordenador. Type & Talk se puede conectar en interface con el ordenador mediante puertas Centronics o RS423, ya que en la parte posterior de la máquina se proporcionan facilidades para los dos formatos, junto con un conmutador para pasar de una a otra. En la parte posterior de la máquina también hay instalado un conector jack miniatura, que permite conectar el sintetizador a un sistema de altavoz externo.

La parte frontal del sistema Type & Talk alberga un interruptor *on/off,* una rejilla para el altavoz interno y el control de volumen. En el interior del dispositivo hay dos placas de circuito impreso; una de ellas contiene el transformador de potencia y la lógica del sintetizador de voz, mientras que la placa hija contiene los circuitos amplificadores de sonido y el chip codificador de voz. Observando primero la placa principal, aparte de chip PIA y la lógica correspondiente, el Type & Talk posee asimismo una EPROM de ocho Kbytes que contiene el firmware del diccionario y las reglas gramaticales. El procesamiento que se requiere para decodificar la información proveniente del ordenador, y antes de pasarla al chip sintetizador de voz, lo lleva a cabo un procesador NEC D780C (que es, esencialmente, un chip Z80). Para almacenar la información de entrada antes de que se la procese, el Type & Talk posee dos Kbytes para RAM en la placa que actúan como un tampón. En la placa

NAMAL TYPE & TALK

CHIP DE VOZ

Votrax SC01A

MODALIDADES

De conversión directa texto-voz y de fonemas

GENERACION DE SONIDO

Amplificador y altavoz en la placa

INTERFACES

Interfaces Centronics y RS423. Dispone de un conector jack miniatura para auriculares o para un sistema de altavoz externo

VENTAJAS

Produce la pronunciación exacta de gran número de palabras. Debido a que el software se basa en ROM, esto deja al ordenador libre para otras aplicaciones

DESVENTAJAS

Muestra tendencia a recalentarse, lo que origina una distorsión de la voz. Su precio no contribuye a que sea un sintetizador de voz demasiado asequible

Crispin Thomas

Con vocación pedagógica
El Namal Type & Talk es un sintetizador de voz diseñado para usar con el BBC Micro. El dispositivo tiene su propio procesador y software almacenados en ROM. Claramente diseñado con vistas al mercado educativo, en el cual el BBC Micro es importante competidor, la máquina es de construcción sólida, su software es excelente, y proporciona una introducción a la síntesis de voz a la vez entretenida e informativa

AMSOFT SPEECH SYNTHESISER

CHIP DE VOZ

SP0256

MODALIDADES

De conversión texto-voz y de alófonos

GENERACION DE SONIDO

Altavoces gemelos que, aunque pueden ser estéreo, reproducen en mono

INTERFACES

Bus de ampliación que se instala en la puerta para disco flexible del ordenador, con un cable flotante a la puerta para *hi-fi*. Jacks miniatura gemelos para los altavoces

VENTAJAS

Una auténtica conversión texto-voz a buen precio

DESVENTAJAS

La lista de reglas gramaticales no es tan amplia como la de Type & Talk. Adolece de restricciones de hardware impuestas por el ordenador

madre también hay incorporados ocho conmutadores DIP que controlan la velocidad en baudios a la cual se manipula la información de entrada, y otros controles para el intercambio de señales de control y selección de paridad. Al menos una tercera parte de la placa principal está ocupada por el transformador de potencia, que los fabricantes han optado por incluir en la placa.

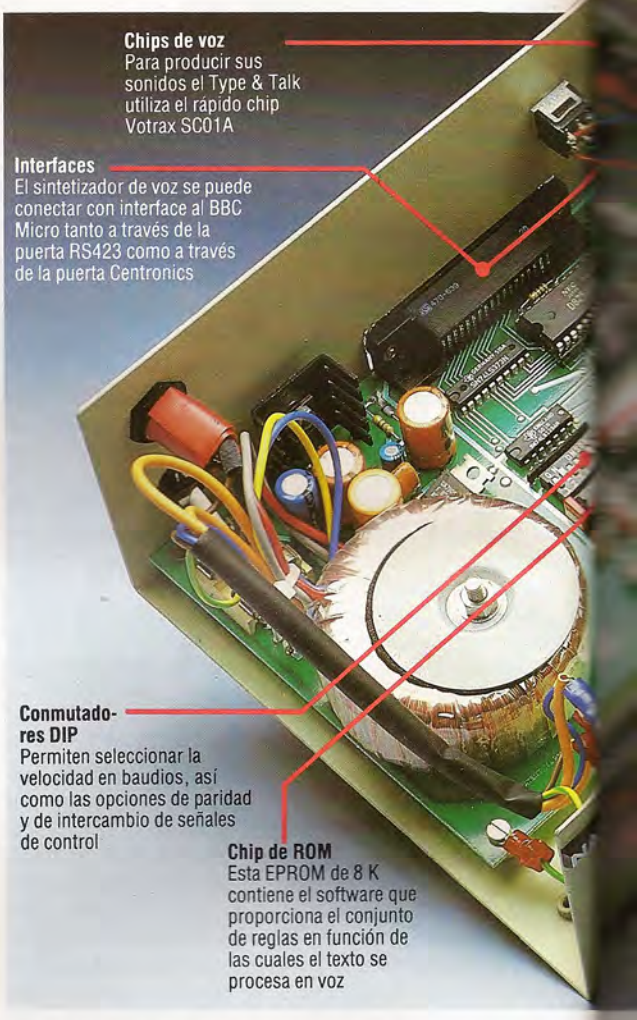
Si bien siempre es preferible tener la fuente de alimentación dentro de una máquina, para reducir la cantidad de cables flotantes, los problemas que entraña tal medida no se han superado por completo en este caso. Después de tener en marcha el Type & Talk durante unas pocas horas, la máquina muestra indicios de sobrecalentamiento, lo que afecta la calidad de la voz producida. La placa hija contiene el amplificador de sonido y un «potenciómetro preajustado» instalado con una tapa atornillada que permite que el usuario regule la velocidad a la cual se produce el habla en el altavoz. La verdadera síntesis de voz la lleva a cabo el chip de voz Votrax SC01A, de gran velocidad, presente en muchos otros sintetizadores de voz para ordenadores.

Una vez conectado el sistema, es sumamente simple producir voz desde el dispositivo. Al encender el Type & Talk, el dispositivo dice Ready Master, indicando que está listo para ser usado. Dado que el ordenador recibe las señales provenientes

Voz amiga
El precio del Amsoft Speech Synthesiser es mucho más asequible que el del Type & Talk; sin embargo, utiliza la misma técnica de «reglas» que éste. El software se carga (LOAD) desde cassette y el texto se convierte en patrones de habla a través del procesador del propio ordenador. A pesar de no ser tan ambicioso como Type & Talk, el dispositivo de Amsoft está pensado como primera máquina para el usuario de un ordenador personal

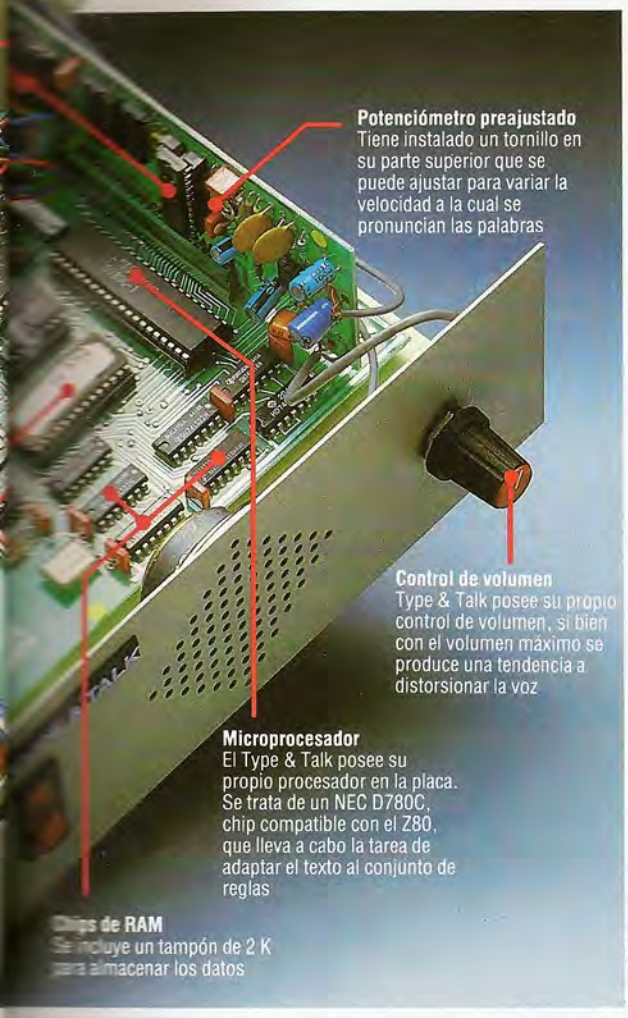
del sintetizador, en forma de códigos ASCII, enviados a través de los activadores de impresora, el ordenador debe estar configurado para producir un «eco» de impresora de lo que aparece en la pantalla. En el BBC Micro esto se hace digitando VDU 2 o, simplemente, enviándole al Type & Talk un carácter CTRL B. A partir de entonces, todas las frases que se digiten en la pantalla serán pronunciadas por el sintetizador de voz.

Ya hemos visto cómo el sistema Type & Talk decodifica las palabras recibidas desde el ordenador, como también las limitaciones impuestas por las restricciones de hardware. Por consiguiente, para obtener la pronunciación correcta de una palabra a menudo es necesario alterar la ortografía, y para conseguir la articulación correcta a menudo se requiere muchísima inventiva. Por ejemplo, el firmware interpreta la palabra *bough* (rama), que se pronuncia «bau», de la misma forma que la pala-



Hablar con facilidad

Al contrario que en la lectura, en la cual podemos reconocer palabras diferentes por la forma en que están escritas, el reconocimiento de las palabras habladas depende de sonidos separados y distintos: los «fonemas». Mientras que el agrupamiento de varios fonemas representa el «grueso» de una palabra, los «alófonos» introducen ligeras diferencias y modificaciones (según el contexto y el lugar en el que se pronuncien las palabras). Por ejemplo, el fonema CH se utiliza tanto en *chip* como en *batch*, pero las pronunciaciones son sutilmente distintas. El alófono, en consecuencia, altera ligeramente el fonema CH para facilitar el reconocimiento de las palabras. Como se puede observar en el siguiente ejemplo, la transcripción fonética a menudo guarda muy poca relación con la forma escrita:



Crispin Thomas

bra *cough* (tos) y, por tanto, la pronuncia «bof». El uso de la ortografía alternativa *bow* (que, según como se pronuncie, puede significar «proa» o «arco») tampoco es de ayuda, puesto que ésta se interpreta para que rime con *low* (bajo). La pronunciación correcta se obtiene escribiendo «bou».

Se puede conseguir un énfasis sobre sílabas determinadas, mediante el envío de caracteres de control entre las palabras a pronunciar. Estos caracteres de control se distinguen de los caracteres ASCII comunes precediéndolos con un !. Un ejemplo de esto lo constituye la palabra HEL!InLO, donde ! significa «entonación» y n corresponde a un número entre 0 y 3.

Otro uso de los caracteres de control que le proporciona al usuario muchísimo más control sobre la pronunciación es !P, que coloca al sintetizador de voz en modalidad de fonemas. Los fonemas son una serie de caracteres que corresponden a todos los sonidos que se efectúan en inglés. (Usted puede observar ortografías fonéticas en los diccionarios, donde por lo general se consignan entre paréntesis tras una entrada en negrita.) Enviando una serie de fonemas, en teoría se puede construir cualquier palabra con cualquier acento.

Speech Synthesiser

El Amsoft Speech Synthesiser es una máquina muy diferente. Se trata de una unidad individual que se enchufa en la puerta para disco flexible de los ordenadores Amstrad. La conversión texto-voz la efectúa el propio procesador del Amstrad, suministrándose el diccionario y las reglas en software basado en cassette. Las instrucciones se envían al sintetizador de voz de una forma algo distinta a la del BBC Micro. Hay nueve instrucciones que permiten operar el sintetizador desde BASIC; éstas, al igual que el sistema operativo de disco Amsoft, están implementadas como ampliaciones del sistema residente (RSX). El problema de este sistema es que para que se les puedan pasar parámetros a las RSX, primero hay que convertirlos en series. Por ejemplo, |SAY (con |ECHO constituye las dos instrucciones para conversión texto-voz directa) exige que antes de poder pasar una oración al sintetizador de voz, primero se la ha de implementar como una serie. Afortunadamente, la instrucción |ECHO, que reproduce mensajes en pantalla a través del sintetizador, posee el traspaso de parámetros incorporado.

Comparación de calidad

Ni el sonido ni la calidad de la conversión texto-voz del Amsoft Speech Synthesiser son tan elevados como los del Type & Talk. Esto tal vez no resulte sorprendente, porque el dispositivo de Amsoft es considerablemente más barato que el del periférico para el BBC Micro. El Amsoft Speech Synthesiser posee muchísimas menos reglas, y están implementadas con menor rigor que en el Type & Talk. El resultado es que muchas palabras que en éste se pronuncian correctamente, en el sintetizador de Amsoft se pronuncian mal. Por ejemplo, *able* («eibl») se pronuncia «able» y *cough* («cof») se pronuncia «couf». No obstante, con una ortografía imaginativa se pueden superar estas dificultades.

Otra instrucción importantísima implementada en el Speech Synthesiser es |APHONE. Esta instrucción, al igual que !P en el sistema Type & Talk, permite construir palabras a partir de sus componentes de sonido básicos. Sin embargo, el sistema Amsoft utiliza alófonos como sus «bloques de construcción», en lugar de fonemas (los alófonos son los sonidos que forman parte de los fonemas, aunque en la práctica hay muy poca diferencia). La instrucción |APHONE va seguida por una serie de números, cada uno de los cuales corresponde a un sonido diferente. Este sistema carece de la prontitud del sistema de fonemas del Type & Talk y significa que el usuario debe ir buscando constantemente el número pertinente del sonido que se requiere, pero resulta bastante fácil acostumbrarse a ello.

En la actualidad, muchos de los problemas que entraña la síntesis de voz ya se han superado en gran medida, a pesar del hecho de que los sintetizadores de voz continúan indefectiblemente sonando como a través de embudos. Pero aún persisten dificultades para idear un programa que tenga en cuenta todas las variaciones que se producen en el habla. Es evidente que la tecnología actual no está a la altura de la tarea, si bien la naciente CD-ROM (*compact disk ROM:* ROM en disco compacto) podría representar un seguro paso hacia la solución de los problemas del almacenamiento de «reglas». Sin embargo, estos dos paquetes son representativos del estado actual de la técnica y constituyen un buen punto de partida para la tecnología en que se basarán los futuros sistemas.

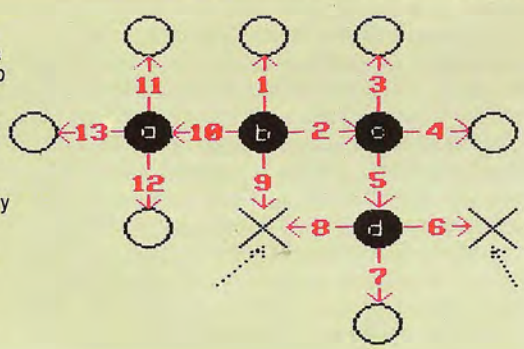
Poder de decisión

Ahora diseñaremos algunos procedimientos que permitan al ordenador «decidir» sus propios movimientos

En este capítulo damos comienzo a la verdadera parte de inteligencia artificial del programa, donde le «enseñamos» al ordenador cómo elegir movimientos razonables. La primera rutina «movimiento del ordenador» le permitirá a éste decidir si algún grupo determinado del tablero se halla o no en problemas. Una vez determinado esto, podemos mostrarle a la máquina cómo seleccionar un movimiento que tenga posibilidades ya sea de poner el grupo a salvo, si fuera uno propio, o bien de atacar y capturar un grupo del adversario.

Antes de comenzar a abordar este problema, veamos en qué forma eligen los movimientos la mayoría de los programas para juegos. En el juego del ajedrez, por ejemplo, se sabe que desde cualquier posición dada del tablero son posibles (como media) aproximadamente 30 movimientos. Tomando esto como base, si un ordenador hubiera de probar 30 posibles movimientos, luego habría de examinar 30 posibles posiciones, decidiendo lo mejor de cada una de ellas. Al intentar decidir la conveniencia de cada movimiento, usted, por supuesto, tendrá que comprobar qué movimientos puede efectuar su oponente en respuesta al suyo: es evidente que un movimiento no es bueno si el oponente puede colocarlo a usted en jaque inmediatamente. Si suponemos que el ordenador va a examinar todos los movimientos que puede efectuar el oponente desde las 30 posiciones que puede alcanzar inicialmente la máquina, luego es necesario examinar algo así como 900 (30 × 30) posibles posiciones. Llevando esto aún otro poco más hacia adelante, podríamos analizar todas las posibles réplicas del ordenador, dando aproximadamente 27 000 posiciones a evaluar, seguidas por 810 000 posiciones al nivel siguiente, y así sucesivamente. Esta evaluación «anticipada» se puede ilustrar gráficamente en forma de *árbol de juego* (similar a un árbol genealó-

Situación crítica
La rutina de evaluación de grupos del programa considera que un grupo se halla en peligro si posee sólo una o dos licencias. El diagrama muestra el orden por el cual se evalúa el grupo de negras, comenzando por la ficha b. Se utilizan dos elementos de matriz, cloc%(1) y cloc%(2), para almacenar las posiciones de las licencias descubiertas, puesto que si el grupo se halla en peligro, el ordenador probablemente querrá jugar en uno de estos puntos



Módulo 4

BBC Micro:

```
  80 movimiento%=movimiento%+1
     :PROCmovimiento_negras
 240 DIM captura%(2),cloc%(2),tloc%(2)
1360 atari1$="          ":atari2$="          ":T$="***"
2530 :
2540 DEF PROCmovimiento_negras
2550 atari1$="          "
2560 posicion%=0
2570 PROCevaluacion_grupos:T$="GP "
2630 IF posicion%=0 THEN ENDPROC
2640 PROCefectuar_movimiento(posicion%,negras%)
2650 PROCmensaje(23,6,"")
2660 ENDPROC
2670 :
2680 REM  ****************************************
2690 :
2700 DEF PROCevaluacion_grupos
2710 LOCAL C%,L%,P%,Q%,S%,hi,marcador
2720 FOR P%=17 TO 255
2730   C%=tablero%?P% AND color%
2740   IF C%=0   THEN 2850
2750   PROCcontar(P%,C%) : IF clib%>2 THEN 2850
2760   tloc%(1)=cloc%(1)
2770   tloc%(2)=cloc%(2)
2780   L%=clib% : S%=cstn%
2790   FOR Q%=1 TO L%
2800     IF FNlegalidad(tloc%(Q%),negras%)<>0 THEN
         2840
2810     IF L%=2 AND clib%<3 THEN 2840
2820     marcador=(8*S%/L%-clib%+2*L%)
2830     IF marcador > hi THEN
         hi=marcador:posicion%=tloc%(Q%)
2840     NEXT
2850   NEXT
2860 ENDPROC
2870 :
2880 REM  ****************************************
4060 cloc%(1)=0 : cloc%(2)=0
4280 IF clib%<3 THEN cloc%(clib%)=P%
```

Amstrad CPC 464/664:

```
  80 mov%=mov%+1:GOSUB 2540
 240 DIM captura%(2),cloc%(2),tloc%(2)
1360 atari1$="          ":atari2$="          ":T$="***"
2530 :
2540 REM **** rutina movimiento negras ****
2550 atari1$="          "
2560 posicion%=0
2570 GOSUB 2700:T$="GR ":REM evaluar grupos
2630 IF posicion%=0 THEN RETURN
2640 mmp%=posicion%:mmc%=negras%:GOSUB 3630:REM
     efectuar movimiento
2650 mp%=25:mm%=6:o$="":GOSUB 2160:REM
     mensaje
2660 RETURN
2670 :
2680 REM  ****************************************
2690 :
2700 REM rutina evaluacion grupos
2710 hi%=-9999
2720 FOR p%=17 TO 255
2730 c%=PEEK(tablero+p%) AND color%
2740 IF c%=0 THEN 2850
2750 cp%=p%:cc%=c%:GOSUB 4040: IF clib%>2 THEN
     2850
2760 tloc%(1)=cloc%(1)
2770 tloc%(2)=cloc%(2)
2780 gl%=clib%:gs%=cstn%
2790 FOR q%=1 TO gl%
2800 lp%=tloc%(q%):lc%=negras%:GOSUB 3890:IF ll%<>0
     THEN 2840
2810 IF gl%=2 AND clib%>3 THEN 2840
2820 marcador=(8*gs%/gl%-clib%+2* gl%)
2830 IF marcador > hi THEN
     hi=marcador:posicion%=tloc%(q%)
2840 next q%
2850 NEXT p%
2860 RETURN
2870 :
2880 REM  ****************************************
4060 cloc%(1)=0:cloc%(2)=0
4280 IF clib%< 3 THEN cloc%(clib)=sp%
```

Commodore 64:

```
  80 MOVIMIENTO%=MOVIMIENTO%+1:GOSUB 2540
 240 DIM CAPTURA%(2),CLOC%(2),TLOC%(2)
1360 :A1$="          ":A2$="          ":T$="***"
2530 :
2540 REM RUTINA MOVIMIENTO NEGRAS
2550 A1$="          "
2560 POSIC%=0
2570 GOSUB 2700:T$="GP "
2630 IF POSIC%=0 THEN RETURN
2640 MP%=POSIC%:MC%=NEGRAS%:GOSUB 3630
2650 RETURN
2670 :
2680 REM ****************************************
2690 :
2700 REM RUTINA EVALUACION GRUPOS
2710 HI=-9999
2720 FOR P=17 TO 255
2730 C%=PEEK(TABLERO+P) AND COLOR%
2740 IF C%=0 GOTO 2850
2750 CP%=P:CC%=C%:GOSUB 4040:IF CLIB%> 2 GOTO
     2850
2760 TLOC%(1)=CLOC%(1)
2770 TLOC%(2)=CLOC%(2)
2780 BL=CLIB%:BS=CSTN%
2790 FOR Q=1 TO BL
2800 LP%=TLOC%(Q):LC%=NEGRAS%:GOSUB 3890: IF
     LL%<> 0 GOTO 2840
2810 IF BL=2 AND CLIB%<3 GOTO 2840
2820 SCR=(8*BS/BL-CLIB%+2*BL)
2830 IF SCR<>HI THEN HI=SCR:POSIC%=
     TLOC%(Q)
2840 NEXT
2850 NEXT
2860 RETURN
2870 :
2880 REM ****************************************
4060 CLOC%(1)=0:CLOC%(2)=0
4280 IF CLIB%<3 THEN CLOC%(CLIB%)=SP%
```

Sinclair Spectrum:

```
  80 LET movimiento=movimiento+1: GO SUB
     2540
 240 DIM c(2): DIM i(2): DIM j(2)
1360 LET x$="          ": LET y$="          ":
     LET t$="***"
2530 :
2540 REM rutina movimiento negras
2550 LET x$="          "
2560 LET posicion=0
2570 GO SUB 2700: LET t$="GP "
2630 IF posicion=0 THEN RETURN
2640 LET mmp=posicion: LET mmc=negras: GO SUB
     3630
2650 LEt mp=21: LET mm=7: LET o$="": GO SUB
     2160
2660 RETURN
2670 :
2680 REM ****************************************
2690 :
2700 REM rutina evaluacion grupos
2710 LET hi=-9999
2720 FOR p=17 TO 255
2730 LET c=PEEK(tablero+p)
2735 IF c>3 THEN LET c=c-4: GO TO 2735
2740 IF c=0 THEN GO TO 2850
2750 LET cp=p: LET cc=c: GO
     SUB 4040: IF clib>2 THEN GO TO 2850
2760 LET j(1)=i(1)
2770 LET j(2)=i(2)
2780 LET gl=clib: LET gs=cstn
2790 FOR q=1 TO gl
2800 LET lp=j(q): LET lc=negras: GO
     SUB 3890: IF ll<>0 THEN GO TO
     2840
2810 IF gl=2 AND clib<3 THEN GO TO 2840
2820 LET marcador=(8*gs/gl-clib+2*gl)
2830 IF marcador>hi THEN LET hi=marcador: LET
     posicion=j(q)
2840 NEXT p
2850 NEXT q
2860 RETURN
2870
2880 REM ----------------------------------------
4060 LET i(1)=0:LET i(2)=0
4280 IF clib<3 THEN LET i(clib)=sp
```

gico), en el cual las bifurcaciones desde cada nudo muestran los movimientos posibles desde esa posición. Este concepto lo hemos analizado profundamente en nuestra serie sobre inteligencia artificial. Estos números, aun pareciendo enormes, se pueden reducir en cierta forma aplicando diversas técnicas (en especial el *algoritmo alfa-beta*) y resultan manejables para un ordenador de gran velocidad.

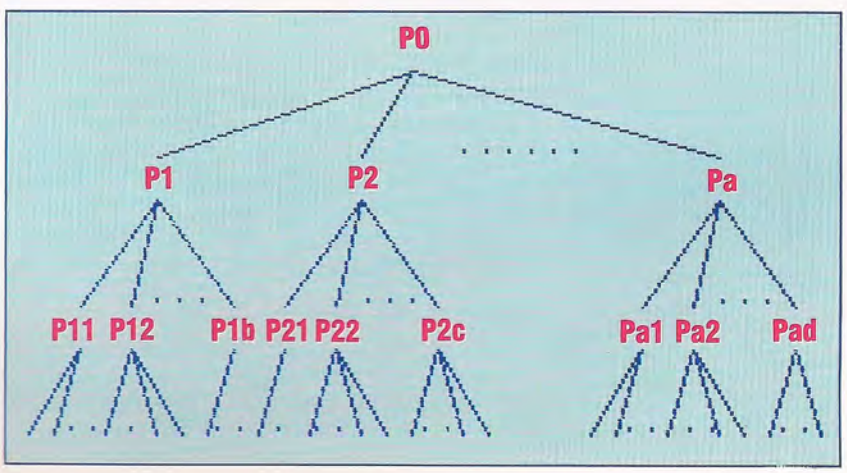
El número promedio de movimientos disponibles desde una posición dada en el juego del *go* (en un tablero de 19 por 19) se calcula en alrededor de 200. En nuestro tablero de 15 por 15, es probable que esta cifra se reduzca a alrededor de 125. Si aplicamos nuestra técnica de anticipación estándar, el crecimiento exponencial da 15 625 posibilidades para las réplicas del oponente, 1 953 125 para el segundo movimiento del ordenador, 244 140 625 posiciones a partir de allí, etc. Con o sin ordenadores veloces, realmente usted no puede esperar que ellos busquen esta cantidad de posibilidades en un tiempo razonable. Además, no es habitual que los maestros del juego examinen posibilidades con hasta 20 o 30 movimientos de anticipación con el fin de decidir el resultado final de una partida determinada.

Una estrategia alternativa

Obviamente, sería absurdo programar nuestro juego del *go* para determinar un movimiento utilizando técnicas de anticipación por «fuerza bruta». Éste es uno de los principales motivos por los que este juego oriental es tan difícil de programar. Una estrategia alternativa consiste en desarrollar una serie de rutinas que examinen la posición actual del tablero, con la esperanza de poder elegir algunos movimientos probables. Finalmente habrá seis rutinas de evaluación, y la evaluación terminará tan pronto cualquiera de las rutinas encuentre un movimiento factible. Por lo tanto, se ha de llamar a las rutinas por el orden correcto. Una variable, posición% (inicialmente cero), se establece en la posición adecuada del tablero si una rutina encuentra un movimiento. Nuestra rutina principal de llamada movimiento_negras será así:

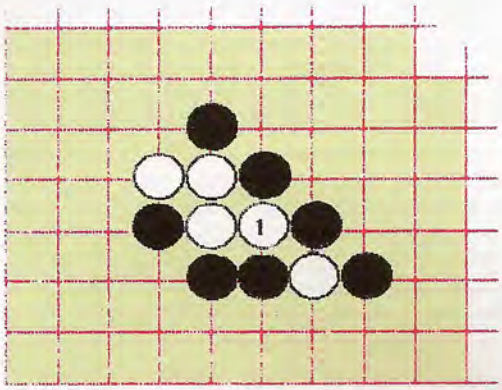
```
LET posicion%=0
CALL                          la primera rutina
                                 de evaluacion
IF posicion%=0 THEN CALL      la segunda rutina
                                 de evaluacion
```

El crecimiento del «go»
Muchos juegos de destreza, como el ajedrez, utilizan árboles de juego de anticipación para decidir el movimiento a realizar. La naturaleza del *go*, sin embargo, significa que se han de considerar alrededor de 200 nuevas ramificaciones tan sólo para analizar un movimiento anticipado; el análisis de dos movimientos implica la comprobación de 40 000 movimientos, y así sucesivamente. Si comparamos esta velocidad de crecimiento con la del ajedrez, podemos ver por qué producir un programa para ordenador para jugar al *go* a un nivel elevado es tan difícil

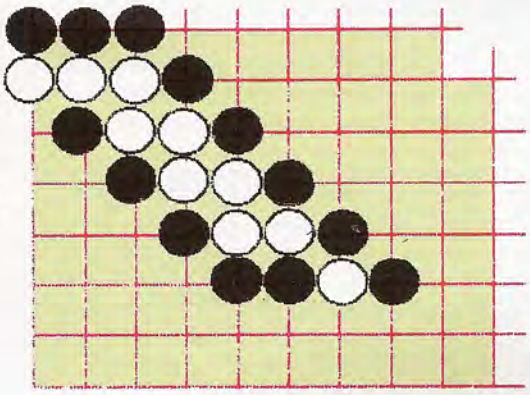


SHICHO

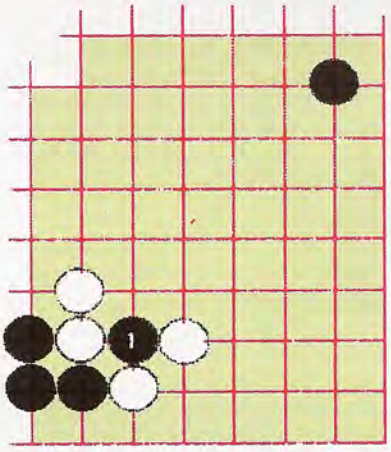
Diagonales peligrosas
La ficha blanca 1 está atacada por las negras y, en vez de sacrificar esta ficha, las blancas intentan extenderse en diagonal hacia arriba y a la izquierda, en un intento por evitar la captura



Fuera de sitio
Las blancas han cometido un grave error al tratar de huir. Finalmente el grupo de blancas en extensión alcanzará el borde del tablero, donde se queda sin sitio y puede ser capturado por las negras. En vez de huir, las blancas deberían haberse preparado para rodear a la ficha original cuando fue atacada por primera vez. Los patrones zigzag en diagonal que se forman durante este tipo de juego se denominan *shicho*



Para romper el «shicho»
En esta situación, las blancas sentirán la tentación de tratar de capturar a la ficha negra marcada como 1, suponiendo que si las negras intentan huir se formará un *shicho* y las negras se quedarán sin sitio en el margen derecho del tablero. Sin embargo, como las negras huyen en diagonal hacia arriba a la derecha, finalmente se unirán a la ficha negra que está sola, rompiendo el *shicho* y permitiendo que las negras escapen a la captura



Caroline Clayton

```
IF posicion%=0 THEN CALL la tercera rutina
                         de evaluación
      .    .    .
      .    .    .
      .    .    .
IF posicion%=0 THEN CALL la n rutina
                         de evaluacion
IF posicion%=0 THEN      no se pueden hallar
                         movs. a realizar
```

Tal como se presenta aquí, PROCmovimiento_negras sólo llama a una rutina de evaluación, a saber, PROCevaluacion_grupos, y si ésta no encuentra un movimiento (si posicion% sigue siendo cero), entonces la rutina simplemente termina. En consecuencia, usted aún no puede esperar que el programa juegue una partida en toda la regla, aunque el ordenador intentará defender un grupo si usted lo ataca.

La rutina «evaluación de grupos» siempre tendrá la máxima prioridad en PROCmovimiento_negras, de modo que siempre se comprobará primero. Tiene por finalidad examinar todos los grupos del tablero y contar sus licencias. Si hay algún grupo que sólo tenga una o dos licencias, se considerará que el mismo está en una situación crítica (es decir, a un solo movimiento de «Atari»), de modo que el ordenador intentará ya sea defender un grupo del ordenador, o bien atacar a un grupo enemigo. La primera acción de la rutina es contar las licencias de cada grupo y guardar las posiciones de éstas si el grupo sólo tuviera una o dos. Nosotros queremos guardar estas posiciones porque nuestro movimiento se producirá en alguna de estas posiciones si deseamos ampliar el grupo para salvarlo, o, atacando, rodear el grupo para capturarlo.

En un capítulo anterior señalamos que la rutina PROCcontar cuenta no sólo las fichas de cualquier grupo, sino también las licencias. Añadiéndole la línea 4060 a PROCcontar y la línea 4280 a PROCbuscar, la cuenta de licencias se almacenará en la matriz cloc%(2).

Evaluación de grupos

Ahora se puede definir PROCevaluacion_grupos. El bucle P% comprueba todas las posiciones del tablero. De hallar un grupo, se cuentan sus licencias y, si posee menos de tres, entonces el grupo es crítico, de modo que se entra en el bucle Q%. Éste comprueba la legalidad de colocar una ficha en la(s) licencia(s) y, de ser ilegal, se le asigna un marcador al movimiento. Esta función del marcador (en la línea 2820) se descubrió básicamente mediante ensayo y error y no es necesariamente la mejor; tal vez a usted le interesa experimentar. Observe que como producto secundario de la llamada a FNlegalidad, ahora clib% y cstn% contienen el número de licencias y fichas del grupo, en el supuesto de que se efectúe el movimiento. Por consiguiente, los valores originales de clib% y cstn% se han guardado en L% y S%.

Por razones de simplicidad, PROCevaluacion_grupos considera todas las fichas del tablero, lo que implica que si un grupo contiene más de una ficha, entonces lo contará más de una vez. Este método, ligeramente ineficiente, se podría mejorar asegurando que los marcadores (establecidos durante PROCbuscar para indicar que se ha contado una ficha) no se borren al final de PROCcontar. Observe que los marcadores de licencias se deben borrar al final de cada búsqueda, puesto que grupos diferentes pueden compartir las mismas licencias. Asimismo, usted debe asegurarse de que sólo se dejen los marcadores cuando se esté llamando a la rutina contar desde PROCevaluacion_grupos. Más adelante, cuando se hayan despejado los marcadores, se habrá de guardar la posición de todas las licencias y cuentas correspondientes, y comprobar la legalidad de cada movimiento. En el próximo capítulo añadiremos una rutina de evaluación de «captura» a fin de proporcionar un movimiento razonable cuando no sea preciso atacar o defenderse.

Pila eficaz

La utilización de la «notación polaca inversa» simplifica las operaciones aritméticas y la manipulación de datos

Es de sentido común que no se puede obtener la suma de dos números hasta saber cuáles son. Esto sugiere que, aunque escribamos 2+3 con el + en el medio, un ordenador prefiriría disponer de los dos números antes de empezar a preocuparse por el +. En realidad piensa en términos de 2 3 + (o sea, tomar los dos números y después sumarlos).

Nosotros estamos habituados a escribir los operadores aritméticos tales como +, −, * y / en el medio («notación de infijos»), como en 2+2. A menudo refleja el lenguaje («dos más dos son cuatro») y separa claramente los operandos entre sí. Éstas son buenas razones para usar la notación de infijos en un lenguaje para ordenador.

Sin embargo, la notación de infijos también posee sus inconvenientes, que aparecen cuando se desea que el lenguaje sea ampliable. En primer lugar, un operador escrito de esta forma debe tener exactamente dos operandos (argumentos o parámetros), uno a cada lado. Para cualquier otra cosa más, se necesita una notación funcional, como FNf(a, b, c, d, e). En segundo lugar, la notación es intrínsecamente ambigua cuando uno escribe algo como 2+3*5. ¿Qué se hace primero, el + o el *? Esto sólo se puede determinar a partir de un conjunto de reglas adicionales, como la que dice que * tiene «mayor prioridad» que +. Si usted quiere ampliar la notación de infijos a nuevos operadores, también ha de poder ampliar las reglas.

El FORTH se inclina por la solución más simple posible: la que, de todos modos, quería el ordenador. Para todos los operadores o funciones (todas las palabras, de hecho), ya sean tradicionales, como +, o nuevas, definidas por el usuario, primero se escriben los operandos y después la palabra: como 2 3 +. Usted puede pensar en esto como si fuera una especie de «notación de libro de recetas»: juntar los ingredientes y luego cocinarlos. Su nombre técnico es *notación polaca inversa*. La notación polaca directa coloca el operador *antes* de los operandos y es la que utiliza el LOGO para las nuevas funciones. Señalemos que todos los números que manipula el FORTH son enteros. Ello se debe a razones de eficiencia, pero significa que la división normalmente no da la respuesta fraccionaria exacta.

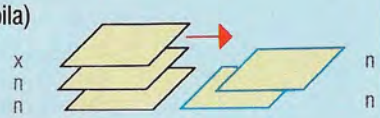
Supongamos ahora que tiene una expresión de infijos más compleja, como 2*3+8/4. De acuerdo a la prioridad usual, el *último* operador a realizar será el +, de modo que el resultado final será la suma de 2*3 y 8/4. Por lo tanto, una primera etapa para escribir esto en notación polaca inversa es:

(2*3)(8/4)+

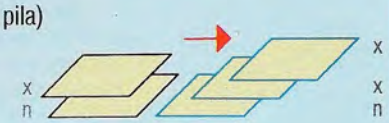
Control de la pila

No siempre es posible acomodar la pila de modo tal que los números queden correctamente ordenados para las operaciones, sin tener primero que manipular sus posiciones. El FORTH ofrece numerosas palabras para la manipulación de la pila:

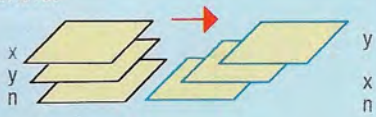
- DROP x−− (Elimina la parte superior de la pila)



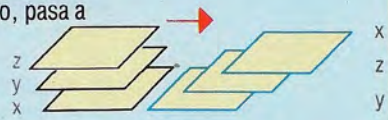
- DUP x−−x,x (Duplica la parte superior de la pila)



- SWAP x,y−−y,x (Invierte los dos elementos superiores de la pila)

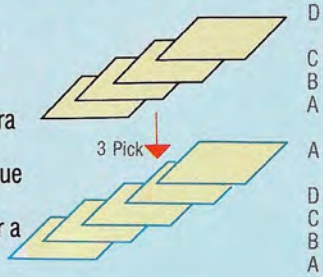


- ROT x,y,z−−y,z,x (Hace rotar los tres elementos superiores de la pila, de modo que el tercer número, que estaba abajo, pasa a quedar arriba)

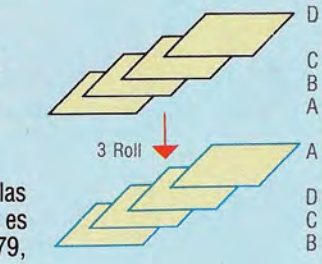


- OVER x,y−−x,y,z (Copia encima el segundo número, que se hallaba debajo)

- PICK xn, . . .,x2, x1, x0, n−−xn, . . ., x2, x1, x0, xn (Igual que OVER, pero se debe proporcionar un número extra (n), para decir qué número es el que se debe copiar arriba. P. ej., 1 PICK es igual que OVER, 2 PICK copia encima el tercer número de debajo, y 0 PICK es similar a DUP)



- ROLL xn, . . ., x2, x1, x0, n−− . . ., x2, x1, x0, xn (Igual que ROT, pero con núm. extra, como en PICK. 2 ROLL es similar a ROT, y 3 ROLL hace rotar 4 núms. 1 ROLL es igual que SWAP)



El figFORTH carece de PICK y ROLL. El FORTH-79 sí las tiene, pero el número extra que usted proporciona es uno más que en FORTH-83. Por lo tanto, en FORTH-79, 3 ROLL es como ROT

Pero esto todavía no es correcto, porque 2*3 y 8/4 aún están en notación de infijos. Reescribámoslos también:

(2 3 *) (8 4 /) +

Quizá piense que necesita los paréntesis para señalar la forma de agrupamiento, pero en realidad la notación polaca inversa *jamás* necesita paréntesis. El agrupamiento siempre es ambiguo, aun sin ellos. De hecho, el FORTH utiliza paréntesis para algo muy distinto (comentarios), de modo que usted está obligado a eliminar los paréntesis. Ello nos deja con la escritura final de 2*3+8/4 en notación polaca inversa:

2 3 * 8 4 / +

Recuerde que en FORTH los espacios son esenciales.

Ahora podemos ver cómo la notación polaca inversa supera los dos problemas de ampliabilidad de la notación de infijos. En primer lugar, no existe ningún motivo por el cual un operador deba abarcar sólo dos operandos. Es tan fácil escribir un operando como tres o cuatro seguidos del operador. El */ del FORTH, por ejemplo, tiene tres operandos: multiplica los dos primeros entre sí, y divide el resultado por el tercero. Esto se ajusta naturalmente al sistema polaco inverso.

El segundo problema que hallamos en la notación de infijos era que, para eliminar la ambigüedad, requería reglas de prioridad y paréntesis. En la notación polaca inversa, este problema no existe. Le dice al ordenador cómo calcular exactamente el resultado de una forma en absoluto ambigua, sin reglas adicionales ni paréntesis.

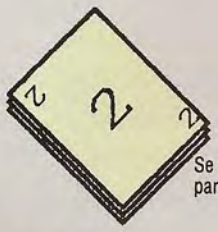
Veamos ahora cómo resuelve el ordenador una expresión en notación polaca inversa. La forma más simple de ilustrarlo es con algo como 2 3 +. El FORTH encuentra el 2, lo memoriza, luego encuentra el 3, y lo «recuerda». Cuando encuentra el + sabe (o, mejor dicho, ha de suponer) que ya ha memorizado dos números. Los busca, los suma entre sí y «recuerda» el resultado en caso de que a continuación venga otro cálculo, o bien si se ha de utilizar el resultado con algún otro fin, como, por ejemplo, imprimirlo en la pantalla.

En el capítulo anterior presentamos una breve explicación acerca de la forma en que el FORTH recuerda los números, «empujándolos» en una pila sobre una base LIFO (último en entrar, primero en salir). A los programadores de lenguaje máquina este concepto les resultará familiar; pero ofrecemos otro ejemplo para quienes encuentren confuso el ejemplo de la pila. Para ver cómo el FORTH «recuerda» un número, piense en cómo lo haría usted a mano utilizando una pila de naipes sobre la mesa. Para recordar un número, lo escribe en un naipe nuevo y lo coloca encima de la pila. Para +, usted debe sacar los dos naipes de encima y usar los números consignados en ellos, pero uno no altera el resto de la pila. Para 2 −3 * 8 4 / + (para (2 * −3) + (8/4)), la pila evoluciona tal como se indica en el diagrama. Observe cómo, en la etapa 6 del procedimiento, el naipe que lleva escrito −6 queda sin modificar mientras se divide 8 por 4.

Llegados a este punto, hemos de destacar que las variables se tratan de forma bastante diferente a los enteros utilizados en este ejemplo. Una variable deja su dirección en la pila. Ésta se puede luego convertir al valor de la variable (usando @, la ins-

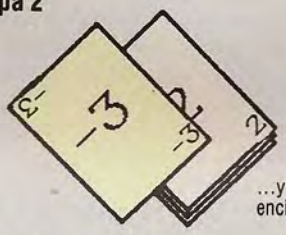
Poniéndolo encima
Este otro ejemplo muestra cómo los números se «empujan» en la pila y se «sacan» de ella, en la ejecución de 2 −3* 8 4 / (2* −3+8/4), en notación de infijos)

Etapa 1



Se coloca el número 2 en la parte superior de la pila...

Etapa 2

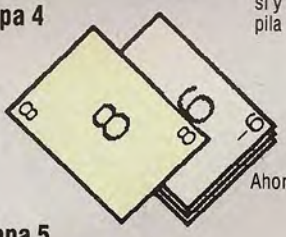


...y después se coloca −3 encima del 2

Etapa 3

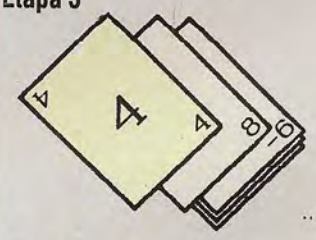
la palabra * toma dos números de la parte superior de la pila, los multiplica entre sí y coloca el resultado en la pila

Etapa 4



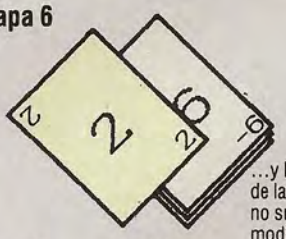
Ahora se pone el 8 en la pila...

Etapa 5



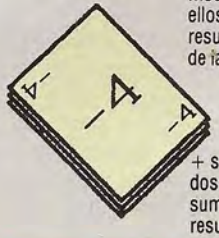
...seguido del 4...

Etapa 6



...y luego / toma dos números de la pila (observe que el −6 no sufre ninguna modificación), se opera sobre ellos y se devuelve el resultado a la parte superior de la pila

Etapa 7



+ saca ahora de la pila los dos números de arriba, los suma y vuelve a colocar el resultado

Caroline Clayton

trucción «traer»), o se puede emplear para actualizar la variable mediante la instrucción ! («almacenar»).

El ejemplo aritmético de la pila demuestra claramente la idoneidad de la notación polaca inversa. El FORTH mantiene su propia imitación interna de la pila de naipes (denominada *pila de datos*) y luego, con cada número o palabra que encuentre, el ordenador puede hacer algo definido sin preocuparse por lo que ocurrió antes o por lo que ocurrirá después.

He aquí otro ejemplo de cómo utilizar la pila. La palabra sustituye a la parte superior de la pila, sea lo que fuere, por sí misma duplicada. Se define:

```
:2* (n -- n*2)
  2 *
;
```

Observe el empleo de los paréntesis para proporcionar comentarios y el símbolo --; ambos se describen en el recuadro «Tomando notas». A modo de ejemplo del empleo de nuestra nueva palabra, 2, la entrada:

```
19 2* .
```

visualiza 38 como resultado. Observe que el símbolo . es la palabra del FORTH para PRINT. Saca la parte superior de la pila y la visualiza en la pantalla. Usted puede ver cómo el * de la definición de 2* espera al menos dos números en la pila. El segundo es el 2 incluido en la definición, pero se espera que el primero (19, en nuestro ejemplo) ya esté en la pila cuando se utilice 2*.

Una de las ventajas de las operaciones basadas en pila es que permiten que una operación produzca más de un resultado. Por ejemplo, la palabra /MOD deja *dos* números en la pila: el «cociente» (respuesta) y el «resto» tras una división. Podríamos expresar esta operación de otra forma, utilizando la notación -- del siguiente modo:

```
m, n -- resto, cociente de m/n
```

Esto sería imposible con la notación de infijos, pero con una pila resulta bastante natural.

La responsabilidad directa de la pila se halla en manos de una potente configuración del FORTH, pero también puede presentar dificultades. Por ejemplo, algunas veces los números tal y como los proporciona la pila no se hallan por el orden correcto y, para reacomodarlos, será preciso efectuar algunas manipulaciones con la pila. En el recuadro «Control de la pila» vemos algunas de las palabras estándares que se utilizan con este fin, y ahora vamos a ver como ejemplo un caso en el cual sería necesario manipular la pila. La palabra */ definida abajo no es tan eficaz como la que se proporciona como estándar (que se cuida de dar la respuesta correcta aun cuando la multiplicación dé un resultado demasiado elevado como para caber en la pila), pero muestra en acción a las palabras manipuladoras ROT y SWAP:

```
:*/ (x,y,z -- x*y/z)
  ROT ROT*
  SWAP /
;
```

He aquí cómo funciona para el ejemplo 4 3 6 */:

Operación de */	Pila (Arriba →)
Al comienzo de */	4, 3, 6
ROT	3, 6, 4
ROT	6, 4, 3
*	6, 12
SWAP	12, 6
/	2
	Vacía (y se visualiza 2)

Afortunadamente, los detalles precisos de la manipulación de la pila por lo general se disimulan debajo de la alfombra ocultándolos dentro de las definiciones de palabras.

Tomando notas

Ha de tener sumo cuidado en colocar en la pila exactamente los números que necesite una palabra. Si coloca pocos, utilizará algunos de más abajo que supuestamente habrían de quedar inalterados, y si coloca muchos, los números extras se situarán en medio. Para decir con claridad lo que una palabra necesita de la pila, y lo deja tras de sí, puede utilizar la notación "--" y después una lista de lo que la palabra deja. P. ej.:

Palabra:	Efecto:
+	m, n -- m+n
.	m --
*/	x,y,z -- x*y/z

Para cada lista, siempre se enuncia en último lugar la parte superior de la pila. La notación "--" no forma parte del lenguaje FORTH, sino que es una configuración añadida que nos permite comprender mejor los listados. Le resultará muy útil incluirla en las definiciones de palabras del FORTH, en los comentarios (encerrados entre paréntesis). Recuerde que es necesario incluir un espacio tras el primer paréntesis, puesto que es una palabra que significa: «ignorar todo cuanto haya hasta el paréntesis de cierre».

Visualizando la pila

Una palabra que le resultará muy útil es .S, que visualiza la pila. No está incluida en el FORTH estándar, pero en FORTH-83 puede definirla así:

```
:  .S (--)
   ( visualiza la pila, de abajo arriba)
   0 DEPTH 1 - DO
      I PICK.
   -1 +LOOP
;
```

En FORTH-79, debido a que PICK trabaja de modo diferente, se debe reemplazar la tercera línea por la siguiente:

```
1 DEPTH DO
```

DEPTH (-- profundidad pila) le dice cuántos números había en la pila antes de que efectuara DEPTH. (El figFORTH no dispone de DEPTH ni de PICK, de modo que en ese dialecto esta definición no funciona)

¿Un poco de ROM?

La Interface 1 puede serle útil para añadir nuevas instrucciones al BASIC del Spectrum. Veamos la «teoría»

Para añadir nuevas instrucciones del BASIC hay que «engañar» a las rutinas de tratamiento de error del Spectrum para que «traguen» algo que normalmente no lo harían. Es, pues, un buen comienzo de nuestro análisis el estudio de cómo se detectan los errores en el Spectrum.

Cuando se digita una línea del BASIC y se pulsa ENTER, la línea es inmediatamente investigada por el intérprete del BASIC por si no fuera sintácticamente correcta. En esta fase se detectan errores tales como la omisión de los nombres de las variables después de instrucciones NEXT, empleo incorrecto de parámetros, etc. Ésta es la causa que impide entrar en este ordenador líneas «torcidas» del BASIC. Si la investigación es positiva se coloca en el lugar adecuado dentro del programa caso de ser una línea de programa, o se ejecuta si es una instrucción en modo indirecto.

Si la línea no es correcta, se ejecutará una instrucción RST#8, seguida de una DEFB, que indica al OS la naturaleza del error cometido. Esto provoca la visualización de un ? parpadeante, del que indudablemente ya habrá tenido usted noticia.

Tras la ejecución de una sentencia, sea en modo directo, sea como línea de programa, se vuelve a comprobar la sintaxis de la línea. Así, en el curso de esta segunda investigación se evalúan las expresiones numéricas contenidas en la línea y se ejecuta mediante llamadas a las correspondientes rutinas de la ROM. En este punto son detectados errores del tipo No Such Variable (No hay tal variable).

El primer chequeo, conocido como *chequeo sintáctico,* no ejecuta la instrucción, sino que sólo asegura su validez sintáctica. El segundo, o *chequeo en fase de ejecución (run-time check),* ejecuta la sentencia. No obstante, cuando se acopla al sistema el hardware de la Interface 1 (IF1), este proceso se altera. Como ya hemos tenido ocasión de ver, el hardware de la Interface 1 se encarga de que siempre que se dé un acceso a la dirección #08 o #1708, la ROM sombra esté paginada y se desconecte la ROM principal. Se dice que ahora el sistema del Spectrum está operando en el «ámbito de la ROM sombra» y la situación se prolonga hasta que hay un acceso a la dirección #700 de la ROM de la IF1 (en este momento se desactiva la ROM sombra y vuelve a reactivarse la ROM principal).

De esta descripción aparece claro que en cuanto se dé un error, la ROM sombra queda paginada y tiene lugar una segunda investigación sobre la condición causante del error. Si el error fue provocado por una instrucción como CAT o FORMAT, que son legales cuando se ha conectado la IF1, éste es tratado por la ROM de la IF1 y se produce un retorno a la ROM principal.

Si la condición del error no fue motivada por ninguna de estas instrucciones, se produce un salto a la dirección contenida en la variable de sistema de la ROM IF1 llamada VECTOR (que se encuentra en las direcciones 23735 y 23736). VECTOR contiene la dirección del tratador de errores de la ROM sombra. Es importante observar que tal dirección puede diferir según las versiones de la ROM IF1. Nosotros hemos empleado la primera versión de la ROM IF1. En el próximo capítulo examinaremos algunas de las principales diferencias entre las distintas versiones de la ROM sombra y le facilitaremos un método para identificar la versión que usted está usando.

Con la primera versión de la ROM IF1, la variable VECTOR contiene el valor #1F0, por lo que una situación de error que reporte una llamada a esta rutina conducirá al editor en tiempo de «chequeo sintáctico» para que sea editado el paso irregular de la línea BASIC que se investiga. En consecuencia, alterando la dirección contenida en VECTOR, desviaremos al tratador de errores de tal modo que ejecute una rutina hecha por nosotros en lugar de la rutina de error habitual. En el próximo capítulo veremos cómo se hace. Pero antes examinemos algunas rutinas de la ROM sombra que nos serán útiles en este proceso.

#0010

Esta rutina permite llamar a la rutina principal cuando se halla en el entorno de la ROM sombra. Es el único medio de llamar a la rutina principal, ya que habremos de poner sumo cuidado en la paginación y despaginación de la ROM principal. Se emplea de la siguiente manera:

```
RST   #0010
DEFW  direccion ;dir. de la rutina por llamar
```

#0020

Esta rutina es la versión para ROM sombra de RST#8 de la ROM principal. Funcionará escribiendo RST #0020, seguido de un solo byte que indique el mensaje de error que ha de ser generado. Los varios mensajes de error se muestran en la siguiente tabla:

#0028

Permite generar un mensaje de error «normal» (es decir, generado en la ROM principal y no en la

De entre las sombras

Además de las facilidades del microdrive, el RS232 y conexión a red que ofrece la Interface 1, la ROM sombra tiene también un buen puñado de útiles rutinas, entre las que se cuenta el convertidor decimal a hexadecimal (que parece no fue asumido por ninguna de las restantes subrutinas del sistema). Para sacar el mayor rendimiento de la interface, recomendamos el libro *Spectrum shadow ROM disassembly*, que proporciona minuciosos detalles de las rutinas de la ROM sombra, así como unos eficaces listados de utilidades

Byte	Mensaje de error
00	Sin sentido en BASIC
01	Núm. no válido de corriente
02	Expresión no válida de dispositivo
03	Nombre no válido
04	Número no válido de drive
05	Número no válido de estación
06	Falta nombre
07	Falta número estación
08	Falta número drive
09	Falta velocidad baudios
10	Error confusión encabezam.
11	Corriente ya abierta
12	Escritura a un archivo lectura
13	Lectura a un archivo escritura
14	Drive protegido contra escritura
15	Microdrive completo
16	No hay microdrive
17	Archivo no hallado
18	Error código enganche
19	Error en CODE
20	Error en MERGE
21	Verificación no lograda
22	Tipo erróneo de archivo
255	Programa concluido

ROM sombra). Antes de llamar a esta rutina, se carga la dirección 23610 con el código de error correspondiente.

#01F0

Genera el cursor parpadeante de *syntax error* mientras se realiza el chequeo sintáctico.

#05B7

Es la rutina a la que se llama para indicar que la sintaxis de la instrucción ha sido comprobada. También comprueba el final de la sentencia BASIC. Más tarde estudiaremos su empleo detalladamente.

#05C1

Empleada para finalizar la ejecución en la ROM sombra, devuelve el control a la ROM principal.

#0700

Provoca un retorno al ámbito de la ROM principal. Puede ser considerada como un medio de despaginar la ROM sombra y de paginar la ROM principal. Examinemos, por último, el código de enganche correspondiente:

• *Código de enganche 50:* Como todos los códigos de enganche *no* ha de usarse en el entorno de la ROM sombra. Le permite a usted llamar a las rutinas de la ROM sombra desde el interior de la ROM principal, y sólo ha de usarse si usted posee un buen conocimiento de las direcciones dentro de la ROM sombra. Se entra con la dirección de la rutina de ROM sombra contenida en la rutina del registro HL, como muestra el siguiente código:

```
LD      HL, direccion;direccion de rutina
LD      (23789), HL
RST     #8
DEFB    50
```

Hay más temas de interés sobre el entorno de la ROM sombra. De lo dicho debe quedar claro que todas las instrucciones *restart* del Z80 son diferentes. El teclado no es explorado cuando nos hallamos en la ROM sombra, ni la variable de sistema FRAMES es incrementada. Consecuentemente, todo espacio de tiempo transcurrido en la ROM sombra tiende a hacer más lenta la variable FRAMES y a hacerla poco fiable en casos de temporización.

A veces un problema que puede presentarse está en el uso de la *calculadora de punto flotante* (*floating point calculator:* FPC) desde la ROM sombra. La rutina que se encuentra en #0010 en la ROM sombra parece incapaz de asumir la llamada a FPC, lo que quiere decir que es mejor salir de la ROM sombra y trabajar en la ROM principal para usarla. Si después usted desea volver a entrar la ROM sombra, lo puede hacer empleando el código de enganche 50.

Empleando un PEEK

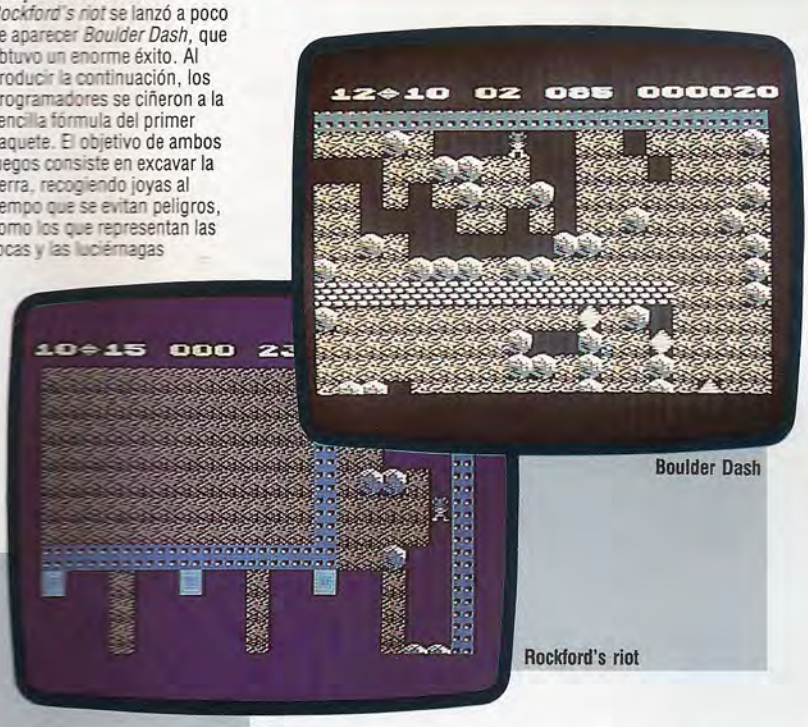
Puede que a usted le agrade examinar algunas de las rutinas aquí mencionadas. El desensamblado de programas ajenos puede ser una experiencia interesante e instructiva y ayudarle a emplear con mucho mayor eficacia las rutinas que se le ofrecen. Por desgracia, directamente desde el BASIC no es posible leer (PEEK) la ROM sombra, ya que sólo está paginada cuando actúa la Interface 1. Este breve programa en BASIC cargará cualquier fragmento de la ROM sombra en la RAM, donde usted sí que podrá examinarlo como guste.

```
10    GO SUB 200
20    INPUT "Cuantos bytes desea ampliar
      (CLEAR)?";b
30    CLEAR r-(b+24)
40    GO SUB 200
50    FOR n=1 TO 23
60    READ d: POKE r+n,d
70    NEXT n
80    INPUT "Direccion inicio en ROM Sombra?";s
90    INPUT "Numero de bytes por copiar?";b
100   LET z=s: GO SUB 300
110   POKE r+12,l:POKE r+13,h
120   LET z=b: GO SUB 300
130   POKE r+18,l:POKE r+19,h
140   LET z=r+9: GO SUB 300
150   POKE r+2,l: POKE r+3,h
160   LET z=r+24: GO SUB 300
170   POKE r+15,l: POKE r+16,h
180   RANDOMIZE USR (r+1)
190   PRINT "Datos almacenados en";r+24
195   STOP
200   LET r=PEEK 23730+256*PEEK 23731:
      RETURN
300   LET h=INT (z/256):LET l=z-256*h:RETURN
400   DATA 33,0,0,34,237,92,207,50,
      225,225,33,0,0,17,0,0,1,0,0,237,176,
      199,201
```

Tesoros ocultos

Comentaremos dos programas de juegos procedentes de Estados Unidos que posiblemente se constituyan en un éxito a nivel mundial

Búsqueda temeraria
Rockford's riot se lanzó a poco de aparecer *Boulder Dash*, que obtuvo un enorme éxito. Al producir la continuación, los programadores se ciñeron a la sencilla fórmula del primer paquete. El objetivo de ambos juegos consiste en excavar la tierra, recogiendo joyas al tiempo que se evitan peligros, como los que representan las rocas y las luciérnagas



Boulder Dash

Rockford's riot

A pesar de que el mercado de software de juegos se ha convertido en una industria multimillonaria, produciéndose literalmente miles de juegos diferentes cada año, son muy pocos los que llegan a provocar impresiones duraderas. Con excesiva frecuencia, aun los juegos de más éxito caen en el olvido a los pocos meses de su lanzamiento. Existen, sin embargo, un puñado de juegos que han tenido una continuación. En Gran Bretaña, por ejemplo, dos de tales juegos son *Manic miner* y su segunda parte, *Jet set Willy*. Este último ha tenido tal éxito que su editora, Software Projects, ha cedido a las presiones del público y ha reeditado el juego con la adición de algunas habitaciones.

En Estados Unidos han surgido ávidos seguidores de Rockford, el héroe de *Boulder Dash* (que también es un juego de «minería») y su éxito ha dado origen a una continuación, denominada *Rockford's riot* (en Gran Bretaña) o *Boulder Dash II* (en Estados Unidos). El nudo de la acción en ambos juegos lo constituye recoger joyas que se hallan enterradas; Rockford debe excavar el terreno para poder alcanzarlas.

Por supuesto, un juego para ordenador estaría incompleto si no hubiera algún obstáculo en el camino. El problema fundamental son las grandes piedras que se hallan diseminadas alrededor de la pantalla y que impiden que Rockford acceda directamente a las joyas. Si cae alguna piedra sobre Rockford, el jugador pierde una vida, aun cuando sea posible que nuestro héroe permanezca de pie con una roca sobre la cabeza. Aunque parecería que las enormes piedras son una molestia, también tienen su utilidad.

Otra de las amenazas son las luciérnagas, criaturas incapaces de horadar la tierra pero que pueden desplazarse por los túneles ya existentes o por los que excava Rockford. Son mortales si se introducen dentro de un cuadrado de Rockford; pero, afortunadamente, tienen su «talón de Aquiles». Primero, cuando disponen de una serie de caminos en los que entrar, siempre optan por la ruta situada más a la izquierda. Segundo, son tan susceptibles de que se les caiga una roca encima como lo es el propio Rockford. Por lo tanto, usted puede predecir hacia dónde se dirigen las luciérnagas, esperar a que lleguen al final de un túnel y, cuando aparezcan, arrojarles una roca encima.

Esta faceta del juego proporciona la solución para otro problema. A menudo las joyas estarán detrás de una pared o de otro obstáculo que le impida el paso a Rockford. Arrojando una roca sobre una luciérnaga que se halle junto a una pared, usted hará explotar tanto a la criatura como a la barrera, donde quedará un agujero.

Para llegar al siguiente nivel se han de recoger una cierta cantidad de joyas, que activarán una puerta situada en algún lugar de la pantalla, a través de la cual pasará Rockford. A los jugadores que no estén familiarizados con el juego esto les resultará desconcertante en un primer momento, porque parece no haber suficientes joyas diseminadas por ahí para que Rockford las recoja. Por ejemplo, ambos juegos contienen una «ameba», una masa verde que se dilata lentamente hasta llenar la pantalla. Sin embargo, si se la rodea con rocas para que no pueda extenderse, la ameba alcanzará una «masa crítica» y se cristalizará en forma de joyas. En otras pantallas, las joyas se crean arrojando piedras sobre objetos determinados.

Gran parte de la acción depende de la rapidez de reflejos, pero la marca de un buen juego es que requiera asimismo movimientos bien pensados.

Los gráficos de *Boulder Dash* y *Rockford's riot*, que son muy similares, son excelentes. Las pantallas se crean en modalidad multicolor y tanto el enrollado de la pantalla como el control del usuario son impecables. Otra característica de estos juegos que hace que jugarlos sea una delicia, es la atención que se ha prestado a los detalles al diseñar las visualizaciones en pantalla.

Boulder Dash y Rockford's riot: Para el Commodore 64 y el Sinclair Spectrum
Editado por: Beyond Software, Competition House, Farndon Road, Market Harborough, LE16 9NR, Gran Bretaña
Autores: Peter Liepa y Chris Gray
Formato: Cassette
Palanca de mando: No se necesita

Mirando hacia el futuro

Fotocomposición y retoque por Helen Zahorodnyj

¿Es el hombre sólo una fase evolutiva en un proceso que conduce a un mundo dominado por máquinas inteligentes? Consideremos los fundamentos de esta inquietante hipótesis

Si consideramos los desarrollos de la informática como si se tratara de la ascensión por una escalera, podemos distinguir, en términos generales, cuatro peldaños de creciente complejidad. En el peldaño inferior tenemos las tareas rutinarias de contabilidad que los ordenadores realizan tan bien. Un tramo más hacia arriba, las cosas se ponen un poco más interesantes. Aquí se hallan los programas que ayudan a las personas a tomar decisiones inteligentes: los sistemas de previsión financiera y la distribución electrónica de información impresa, p. ej. Estas herramientas han de ser flexibles, ya que las demandas del usuario son imprevisibles.

En el tercer nivel encontramos la aplicación de pericia derivada de los seres humanos. Es aquí donde se concentra el grueso de las aplicaciones de la AI, tanto ahora como en un futuro inmediato. Un ejemplo de este tipo de software es el sistema experto PROSPECTOR, que codificó las conclusiones de las investigaciones de varios geólogos. Al aplicar estas reglas en el campo, sus autores se vieron recompensados con el descubrimiento de un enorme depósito desconocido de molibdeno en el estado de Washington, al noroeste de Estados Unidos. Poco después se descubrió otro en Canadá.

Las realizaciones del PROSPECTOR han pasado a formar parte de la mitología de la AI, y, por cierto, son impresionantes. Sin embargo, no nos llevan hasta el cuarto peldaño de la escalera: la producción de máquinas con inteligencia creadora. Para comprender en qué consiste esto, es necesario delimitar claramente productividad y creatividad.

La productividad se basa en seguir las reglas, y los ordenadores son excelentes seguidores de éstas. Pueden ser más productivos que las personas; pero dotarlos de creatividad ha resultado ser increíblemente difícil. Para ser auténticamente creadores, los programas para ordenador habrán de producir sus propias reglas. Ciertos investigadores de las fronteras de la AI están, sin embargo, tratando de hacer que los ordenadores hagan exactamente eso.

Un programa que ha dado un paso adelante hacia el cuarto peldaño de la escalera es EURISKO, desarrollado por Doug Lenat en la Universidad de Stanford. EURISKO es un programa de descubrimiento que ya hemos mencionado con anterioridad en esta serie. Se ha aplicado a varios campos, desde un juego de guerra naval hasta el diseño de circuitos VLSI (integración a muy gran escala). EURISKO parte de un conjunto de reglas y conceptos heurísticos y los aplica a los dominios escogidos. Hasta aquí todo es normal: su novedad radica en el hecho de que puede modificar su propia heurística. Ello le confiere al sistema un enorme poder, dado que puede adaptar sus reglas generales y especializarlas para afrontar situaciones nuevas.

EURISKO ha hecho un descubrimiento por el cual posteriormente se otorgó una patente, de modo que no se lo podría considerar como superficial. Se trata de un nuevo diseño para un circuito lógico 3-D (tridimensional, formado por una puerta NAND/OR) inédito en Silicon Valley —California— (y en todas partes). Y fue generado por la aplicación de una única regla.

Cuando se le aplicó el EURISKO al diseño VLSI, ya contaba con una regla heurística, obtenida a partir de tareas previas, que afirmaba: «si un concepto es interesante, intenta hacerlo más simé-

Realidad más allá de la imaginación
El progreso en el desarrollo de la biotecnología podría conducir a la inquietante perspectiva de los híbridos hombre-máquina, mezclando los procesos intuitivos del cerebro humano con el potencial lógico y matemático de los microprocesadores. En el caso de que esto ocurriera, podríamos ver al proceso de la evolución humana arrancado de las manos de la naturaleza y puesto bajo el control de hombres-máquina capaces de una autorreproducción selectiva

trico». Aplicada a un dispositivo 2-D (bidimensional), condujo a una versión 3-D más simétrica de ese dispositivo, el cual se ha fabricado recientemente con gran éxito. Un dispositivo 3-D, una vez sorteados los problemas de fabricación, se puede empaquetar más densamente, ofreciendo, por lo tanto, una mayor capacidad.

¡Eureka, EURISKO!

Cuando el programa EURISKO, de Doug Lenat, se aplicó al diseño de circuitos lógicos, apareció con un diseño totalmente nuevo para una puerta lógica 3-D, que desde entonces se ha patentado. Aplicándole una regla heurística a un diseño estándar de puerta lógica MOS, que sugería que se podían aumentar los diseños simples para hacerlos más simétricos, el EURISKO desarrolló su nuevo diseño. Además de poder calcular simultáneamente operaciones NAND y OR a partir de cualquier par de entradas, la nueva puerta forma un mosaico en tres dimensiones, posibilitando elevadas densidades de empaquetamiento. Desde entonces, y a partir del diseño del EURISKO, se han construido chips de «gran crecimiento» formados mediante la recristalización de delgadas películas de silicio

Steve Cross

La creatividad se considera la cumbre de la inteligencia humana, envuelta en un velo de misteriosas seudoexplicaciones que entrañan intuición y penetración; pero EURISKO habrá de concedernos una pausa para reflexionar. He aquí un ejemplo de un auténtico descubrimiento obtenido a partir de la aplicación de una sola regla. El ordenador creativo está más cerca de lo que se cree comúnmente.

A lo largo de esta serie, hasta ahora nos hemos venido concentrando en el rostro aceptable de la AI. La hemos visto como la vanguardia de la ciencia informática: una fuente fértil de ideas nuevas y de recursos inteligentes. Cuando estas ideas tienen éxito, se traspasan a otras áreas de la informática. Esto ha sucedido en el pasado con el procesamiento de listas y la informática conversacional, y actualmente está sucediendo con los sistemas basados en el conocimiento. La AI «exporta sus éxitos».

Las perspectivas a medio plazo para este tipo de trabajo son buenas. Podemos esperar progresar en un frente amplio. Puede haber reveses, y puede ser que algunas de las predicciones más atrevidas no se materialicen, pero a finales de este siglo habremos sido testigos de sustanciales avances en los campos de la visión por ordenador, la traducción automática, el aprendizaje de la máquina y los sistemas basados en el conocimiento. Incluso el espinoso problema de la comprensión del habla continua estará próximo a su solución.

La iniciativa japonesa de la quinta generación le ha proporcionado un extraordinario impulso a esta cara de la AI. A raíz de sus ambiciosos planes, los japoneses han redefinido sutilmente las reglas del juego para la industria del ordenador. Para 1990 esperan producir procesadores de información del conocimiento basados en arquitecturas informáticas radicalmente nuevas, y paralélas en gran medida. El software que hará funcionar tales sistemas está firmemente enraizado en la investigación sobre AI. La confianza de los japoneses en las técnicas de AI ha hecho que el resto del mundo se vuelque a ella. Se ha convencido a los gobiernos para que aprueben planes de inversión importantes, a fin de no quedarse rezagados en este terreno.

Existe, sin embargo, lo que para algunas personas es la cara oscura de la AI. Independientemente de lo notables que sean los adelantos en el campo de la inteligencia artificial, quizá no nos agraden algunos de los usos a los que se destinen, en especial si consideramos hasta qué punto depende de los fondos militares la investigación en AI. Más de la mitad de las aplicaciones a corto plazo son de carácter bélico. Éstas incluyen:

- Submarinos inteligentes
- Municiones «sagaces»
- Misiles crucero
- Carros de combate autodirigidos
- Sistemas de sonar basados en el conocimiento
- Torpedos autodireccionales
- Sistemas de imágenes por radar

y muchas aplicaciones más sobre las que la mayoría de nosotros no habrá siquiera oído hablar.

Consideremos por un momento qué es lo que hace que una granada de artillería sea «sagaz». No se limita a caer desde el aire y abrir un agujero en el suelo: selecciona su objetivo. Al acercarse al fin de su vuelo busca objetivos con aspecto de carro de combate y ajusta su trayectoria para asegurarse de que caiga sobre uno de ellos. Dispare en la dirección general adecuada y, con toda seguridad, hará blanco. Así es una de las aplicaciones inmediatas de

la AI: mejores formas de destruir carros de combate.

Aún más alarmantes son las tramas de anticipación que han urdido algunos científicos a partir de éxitos notabilísimos en la búsqueda de *máquinas ultrainteligentes* (UIM: *ultra-intelligent machines*), tal como se ha dado en llamar a los artefactos dotados de inteligencia «suprahumana». Estos hombres de ciencia prevén un futuro dominado por máquinas UIM. Sin «pensar» realmente, estas máquinas serán capaces de hacer casi todo lo que exige un pensamiento razonado y mejor de lo que pueden hacerlo las personas. Nos aventajarán muchísimo en matemáticas y ciencias naturales. En la industria serán unos administradores tan efectivos, que el manejo de economías completas estará bajo su control. En resumen, seremos inferiores intelectualmente. Esta predicción considera una falacia la sugerencia de que siempre podremos «tirar del enchufe» en el caso de que la inteligencia de las máquinas llegara a tal punto que éstas se constituyeran en entes incontrolables.

Durante mucho tiempo, uno de los conceptos recurrentes en las obras de ciencia-ficción ha sido la de los híbridos hombre-máquina; pero sólo recientemente esta idea ha traspasado la frontera de lo fantástico entrando en el área de la conjetura científica a largo plazo. La influencia de dos áreas de la investigación científica aparentemente incompatibles (la ingeniería genética y la ingeniería del conocimiento) es en gran parte responsable de ello.

La ingeniería genética trata de la manipulación del código genético en la materia viva para «programar» mejoras en el campo de la herencia en las generaciones sucesivas. Los ingenieros genéticos ya han perfeccionado técnicas que permiten trasplantar características deseadas en microorganismos ya existentes con el objeto de producir drogas que con anterioridad eran prohibitivas por su precio o bien imposibles de producir a gran escala. Existe la propuesta de que, en un futuro cercano, los métodos de la ingeniería genética serán lo suficientemente sofisticados como para producir los llamados *biochips*. Mediante la programación de los enzimas que dividen y reconstituyen las moléculas se podrá *hacer crecer* circuitos lógicos. La mejora en la densidad de empaquetamiento, obtenida haciendo crecer circuitos a partir de moléculas de proteínas en vez de crearlos utilizando los métodos actuales, tendría como consecuencia una drástica reducción en cuanto a tamaño. La idea de que la materia viva transporte mensajes eléctricos codificados no es nueva: es el mecanismo básico en función del cual trabaja el sistema nervioso humano.

En la actualidad, los ingenieros genéticos alteran la estructura genética del trigo para hacerlo más resistente al ataque de los hongos, pero estas técnicas se podrían utilizar fácilmente para erradicar enfermedades genéticas, como el síndrome de Down, en los seres humanos. Una vez dados los primeros pasos en la manipulación de los genes humanos, a la larga podrían efectuarse otros experimentos; por ejemplo, los científicos podrían tratar de trasplantar en el cerebro humano una bioROM que contenga una base de datos de información. Aunque esto pueda parecer rebuscado, los científicos ya han obtenido cierto éxito implantando aparatos eléctricos en el cerebro. A una estudiante sorda de Oxford se le ha enviado a las áreas auditivas del cerebro la salida eléctrica de un micrófono. Tras un corto período de entrenamiento, pudo descifrar el significado de estas señales y "oír" sonidos.

Algunas personas plantean que en el futuro lejano, nuestros descendientes evolutivos puedan ser híbridos hombre-máquina sumamente avanzados. Puesto que el proceso evolutivo parece hacer uso del material que tenga a mano, sea cual fuere (las aletas se convirtieron en patas; los pulmones que desarrollaron los peces para la flotación se utilizan para respirar aire), parece razonable suponer que estos organismos híbridos pudieran tener en su corazón un cerebro humano rodeado por muchas capas de tecnología avanzada. En realidad, el cerebro humano en su forma actual se compone de varias capas que se desarrollaron durante nuestra evolución.

De máquina a superhombre HAL, el ordenador que coprotagonizó la película *2001: una odisea del espacio*, y que vemos aquí en un fotograma de *2010: el año en que hicimos contacto*, secuela de la anterior, comienza a vivir equipado con unas amplias facilidades para síntesis de voz y un complejo juego de reglas en función de los cuales juzga el éxito de una misión. Tales configuraciones se están implementando cada vez más en el hardware y el software de hoy en día. En la novela de igual título en que se inspiró la segunda de las películas citadas, el autor, Arthur Clarke, llevó el concepto de inteligencia de la máquina varios pasos más hacia adelante, al hacer que HAL trascienda su condición mecánica para convertirse en una figura suprahumana, cuestionando, en consecuencia, el supuesto de que las máquinas no pueden evolucionar independientemente de sus creadores

Cortesía de U.I.P.

Establecimiento de la pendiente para la patilla BUSY
Las fases I y II del proceso de conversión tienen una duración fija, pero el tiempo que lleva efectuar la tercera fase es proporcional a la magnitud de la tensión de entrada original, V_{ent}. Durante esta fase, la tensión de entrada previamente integrada se reduce uniformemente a cero: cuanto mayor sea la tensión inicial, más tiempo se empleará en este proceso. Este hecho nos permite conectar nuestro DVM en interface con un ordenador. La patilla BUSY del chip convertidor 7135 se hace alta (*high*) al comienzo de la fase de integración de señal y permanece así hasta un impulso de reloj después de que la salida del integrador pasa por cero. Si con una AND se relacionan las señales BUSY y CLOCK mediante una puerta lógica, la salida tomará la forma de impulsos de reloj válidos, es decir, impulsos de reloj entre el comienzo de la segunda fase y el final de la tercera. Estos impulsos se pueden transmitir a un ordenador a través de un enlace en serie. Tras restar 10 001 impulsos de reloj para tener en cuenta la fase "integración de la señal" y el impulso nulo al final de la fase "integración de la referencia", la cantidad de impulsos restante será directamente proporcional a la tensión de entrada original. Por lo tanto, tras el calibrado, 20 000 impulsos corresponderán a 2 V, y así sucesivamente

Conversión integradora

Examinemos de cerca el chip integrador 7135, localizado en el centro del tester digital

La mayoría de los convertidores A/D, ya sea los construidos a partir de componentes discretos (un hecho raro en estos días) o bien los integrados en un solo chip, por lo general utilizan uno de dos métodos. Se dice que los dispositivos que incorporan estos métodos son *convertidores de aproximación sucesiva* y *convertidores integradores*. El chip 7135 es un convertidor de tipo integrador, pero antes de examinar su funcionamiento de forma detallada, veamos brevemente cómo trabajan los dos tipos.

Los convertidores A/D de aproximación sucesiva se basan en un convertidor D/A utilizado en un bucle de realimentación junto con un comparador. La salida del convertidor D/A se produce de a un bit por vez, comenzando por el MSB (*most significant bit*: bit más significativo) y avanzando hacia el LSB (*least significant bit*: bit menos significativo). Todos los bits de la palabra del convertidor D/A se fijan inicialmente en uno. A medida que se van efectuando las comparaciones, el bit que se esté considerando se deja en uno si la salida del convertidor D/A es menor que la tensión de entrada, y se establece en cero si la salida D/A es mayor que la entrada. Tras cada comparación, se compara el siguiente bit menos significativo. Tras comprobar todos los bits disponibles (por lo general 8, 12 o 16), los bits que quedaron en estado «uno» permiten que circule una corriente desde el convertidor D/A, que corresponde a la entrada analógica.

Una conversión de ocho bits sólo lleva ocho comparaciones, la conversión de 16 bits sólo lleva 16, y así sucesivamente. Esto hace que la técnica de aproximación sucesiva sea sumamente rápida (100 000 conversiones por segundo o más) y, por lo tanto, ideal para conversiones de audio, video, radar, etc., en las que en un lapso muy breve se han de convertir a forma digital muchísimos datos analógicos. Esta técnica, no obstante, plantea numerosos inconvenientes, dos de los cuales son que los convertidores de aproximación sucesiva son caros y requieren un complicado sistema de circuitos.

Convertidores integradores

Cuando no son esenciales las conversiones a gran velocidad (los voltímetros digitales constituyen un buen ejemplo de ello), el diseño elegido por lo general es el convertidor integrador. Cuando no se pretende digitalizar formas de onda CA de elevada frecuencia, son muy adecuadas velocidades de muestreo de unas pocas lecturas por segundo.

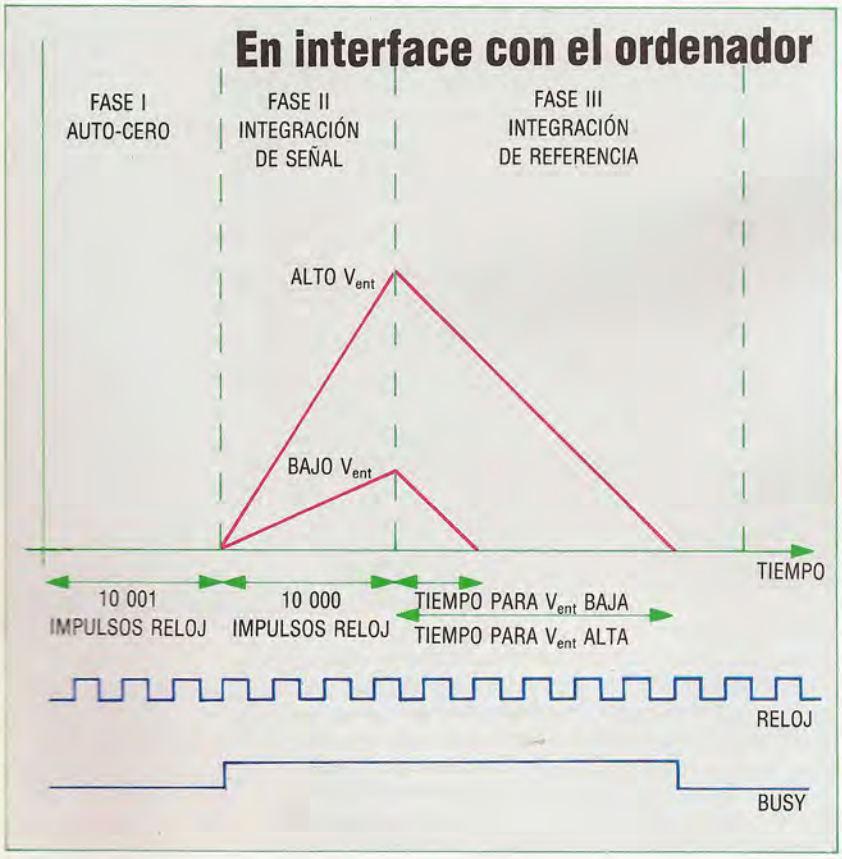
La salida de un convertidor A/D integrador representa el valor promedio de una tensión de entrada analógica a través de un período de tiempo fijo. A diferencia de los convertidores A/D de aproximación sucesiva, que deben «muestrear y retener» la señal de entrada antes de que tenga lugar la comparación y conversión, los de tipo integrador digitalizan la entrada utilizando el tiempo. Para temporizar la conversión se pueden usar señales de reloj.

Entre las ventajas de la técnica del convertidor integrador, que es la que utilizaremos para nuestro diseño, se incluyen una gran precisión, componentes no críticos (aparte de los componentes de la tensión de referencia), excelente rechazo del ruido, ausencia de la necesidad del difícil sistema de circuitos para «muestreo y retención», y componentes de costo comparativamente reducido.

En un circuito convertidor A/D de tipo integrador típico, la conversión se produce en tres fases. Éstas se conocen como la fase de *auto-cero*, la fase de *integración de la señal* y la fase de *integración de la referencia*. Éstas son, por consiguiente, relativamente inmunes a los efectos de fluctuaciones de elevada frecuencia en el "ruido" de entrada (al contrario que los convertidores de aproximación sucesiva para "muestrear y retener").

Todo cuanto se requiere para un convertidor A/D integrador, aparte de una señal de reloj estable, es una tensión de referencia de precisión. Ésta se obtiene, como veremos más adelante, utilizando diodos de referencia de intervalo de banda, que se

En interface con el ordenador



Kevin Jones

consiguen con toda facilidad. Estos diodos limitan la tensión máxima que se pueda aplicar.

La mayoría de los convertidores A/D integradores, incluyendo al 7135, utilizan la técnica denominada de *atenuación doble*. Las tres fases operan así:

Fase 1: auto-cero: Se anulan los errores de los componentes analógicos derivando la entrada a tierra y almacenando la información de error en un condensador de auto-cero.

Fase 2: integración de la señal: Se integra la señal de entrada durante cierta cantidad de impulsos del reloj; para un convertidor de 4,5 dígitos, lo típico son 10 000 impulsos de reloj. Una vez terminado el período de integración, la tensión obtenida es directamente proporcional a la señal de entrada.

Fase 3: integración de la referencia: Al comienzo de la fase, la entrada al integrador se conmuta desde la tensión de entrada a medir, hasta la tensión de referencia. La cantidad de impulsos de reloj contados desde el principio de la fase integración de referencia hasta el momento en el que la salida del integrador pasa por cero, representa la magnitud de la señal de entrada.

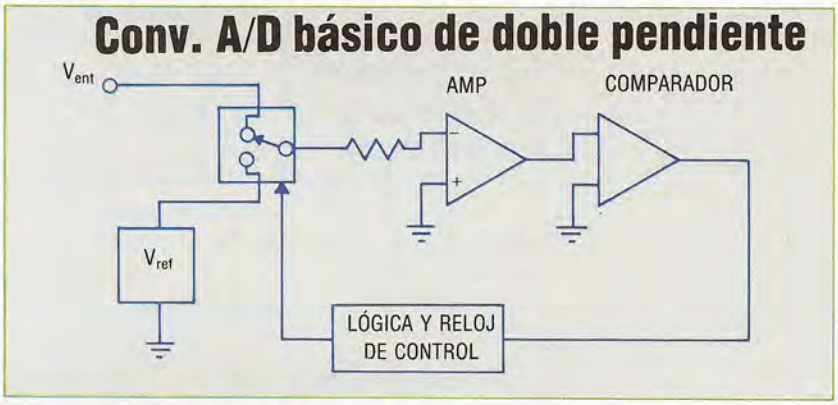
Los convertidores A/D de atenuación doble son intrínsecamente exactos, ya que todo el proceso depende exclusivamente de la absoluta precisión de la tensión de referencia (por ello se debe escoger con cuidado el diodo zener de referencia) y la calidad de los impulsos del reloj. No es necesario que el reloj tenga una frecuencia determinada, ni que cada impulso tenga exactamente la misma duración que los otros. Una de las razones por las que hemos optado por usar el chip temporizador 555 para el reloj es, precisamente, que se trata de un dispositivo básicamente estable y exacto.

La estabilidad de los otros componentes, como el condensador de integración, no tiene especial relevancia a condición de que su valor no cambie durante ninguno de los ciclos de conversión (los convertidores de *aproximación sucesiva*, por el contrario, basan su precisión en la correspondencia exacta de las resistencias de una "escalera" de resistencias). Puesto que es fácil mantener estabilizada la exactitud del reloj en menos de una parte por millón, los convertidores A/D de pendiente doble ofrecen muchísimas ventajas sobre los convertidores de aproximación sucesiva, siempre que no se requiera una gran velocidad de conversión.

El chip 7135

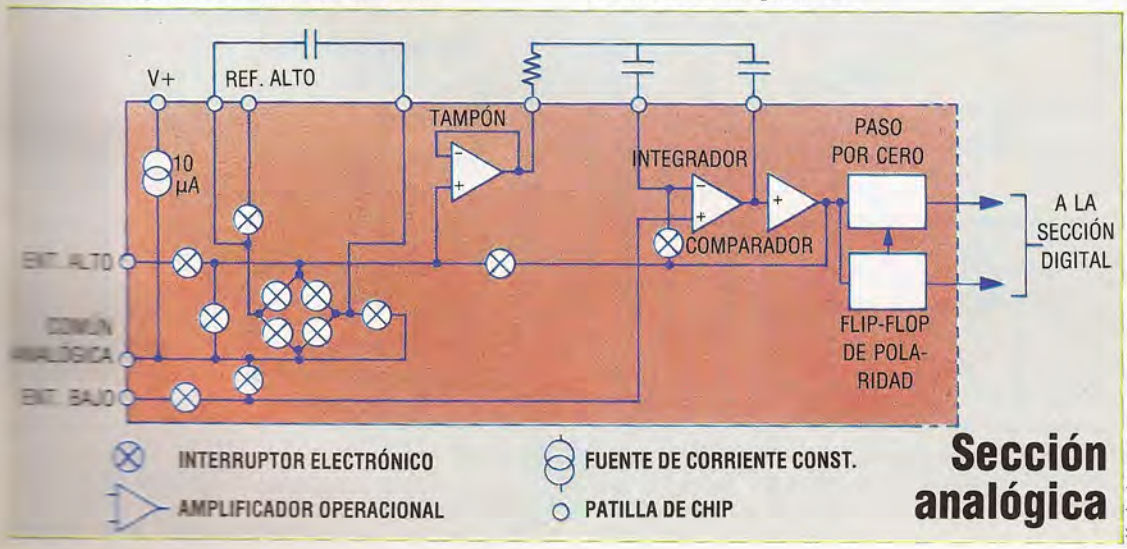
En la ilustración vemos la sección analógica del 7135. La mayoría de los símbolos probablemente le resultarán familiares, exceptuando, quizá, el doble círculo y los círculos con cruces. Éstos representan, respectivamente, fuentes de corriente constante e interruptores analógicos.

Alrededor del circuito
Un convertidor A/D integrador de atenuación doble lleva a cabo su tarea en tres fases diferentes: auto-cero, integración de la señal e integración de la referencia. Durante la fase de integración de la señal, el circuito conmuta la tensión de entrada (V_{ent}) y durante la fase de integración de la referencia se conmuta la tensión de referencia utilizada para calibrar el convertidor (V_{ref}). Una fuente de reloj externa coordina toda la operación



Durante las tres fases de la acción del convertidor (auto-cero, integración de señal e integración de referencia) es importante observar que la cantidad de impulsos de reloj en el tiempo es fija para las dos primeras, y variable para la última. El auto-cero emplea 10 001 impulsos de reloj, la integración de la señal, 10 000 impulsos, y la integración de la referencia, tantos impulsos como sea necesario para que la salida del integrador pase de cero (hasta un máximo de 20 001). Esto permite aplicar una técnica muy sencilla para que el ordenador lea la salida A/D, utilizando una interface en serie.

La frecuencia del reloj, dentro de unos límites amplios, no tiene importancia. Bastarán frecuencias de reloj tan bajas como de 5 KHz (dando un ciclo de medición de alrededor de 10 segundos), o bien algunas tan altas como de 1 MHz. No obstante, las velocidades de reloj muy bajas producen errores debido a las fugas de los condensadores de referencia, y las velocidades de reloj muy altas requieren una ingeniosa compensación del condensador integrador usando resistencias emparejadas cuidadosamente. Basta decir que cualquier frecuencia de reloj de entre 100 KHz y 160 KHz funcionará de forma perfecta.



Kevin Jones

Paso analógico
El diagrama ilustra el trazado de los principales componentes analógicos existentes en el chip convertidor A/D 7135. Los componentes incluidos dentro de la superficie sombreada se hallan realmente "dentro" del chip. Todos los otros componentes son externos. La sección analógica del 7135 es esencialmente un circuito convertidor A/D integrador de doble pendiente, que le proporciona a la sección digital del chip señales de polaridad y paso por cero. En conjunción con las señales externas de reloj, éstas son suficientes para proporcionar a la sección digital la información que necesita para generar la salida BCD digital del chip

Con dos condiciones

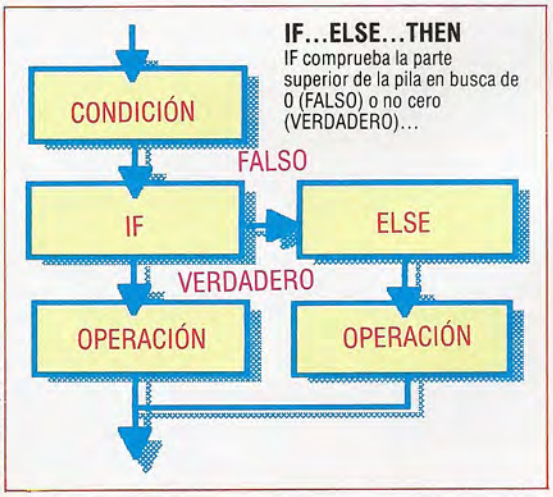
En esta ocasión comentaremos dos estructuras de control condicional del FORTH

Al igual que otros lenguajes, el FORTH posee diversas estructuras para controlar el flujo de un programa. Si usted está acostumbrado a lenguajes que disponen de estructuras de control, como el PASCAL o el C, o los BASIC más avanzados, como los incluidos en el BBC Micro o el Sinclair QL, reconocerá con facilidad lo que las mismas hacen. Sólo recuerde, sin embargo, que debido a la forma en que el FORTH utiliza una pila, a menudo estas estructuras parecen estar escritas al revés.

Otro punto a tener presente es que estas estructuras sólo se pueden emplear dentro de definiciones de dos puntos. Esto se debe a que el FORTH necesita compilar las instrucciones de bifurcación en saltos condicionados y absolutos. Este proceso requiere una cierta cantidad de tiempo, que no estaría disponible si hubiese que compilar las instrucciones durante la ejecución del programa. Insertando las estructuras dentro de definiciones de dos puntos, usted le brinda al FORTH la posibilidad de calcular exactamente lo que está sucediendo mientras la definición está todavía incorporada en el diccionario.

El FORTH estándar dispone de lo siguiente:

- condición IF parte verdadera ELSE parte falsa THEN



Caroline Clayton

Según la condición sea verdadera o falsa, el FORTH ejecuta bien la parte verdadera, bien la parte falsa. También se puede, como en la mayoría de los otros lenguajes, omitir el ELSE y la parte falsa. Si posteriormente la condición es falsa, el FORTH prosigue inmediatamente después del THEN.

La condición y las partes verdadera y falsa pueden ser cualquier secuencia de palabras del FORTH, posiblemente con más IF...THEN...ELSEs. La condición ha de estar en la pila, y luego el IF la sacará. He aquí un ejemplo, que toma un número de la pila e imprime si es impar o par:

```
:PARIDAD      ( n--)
              (imprime n "es par" o "es impar")
  DUP.        (imprime el propio n)
  ."es"
  2 MOD 0 = IF
    ."par"
  ELSE
    ."impar"
  THEN
;
```

MOD da el resto cuando se divide el primer número por el segundo.

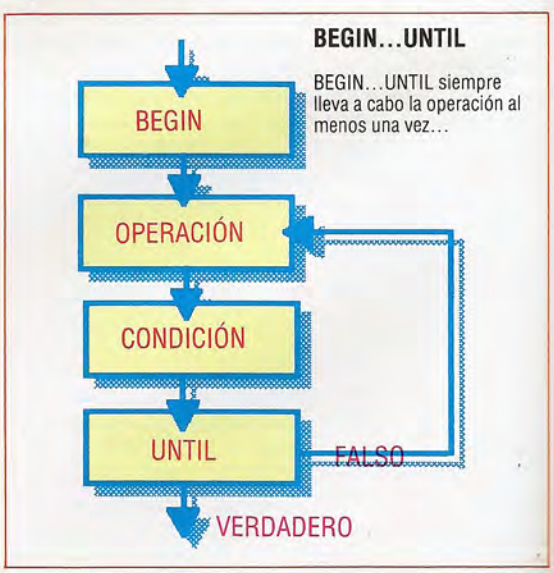
- BEGIN...UNTIL

Es para efectuar un bucle mientras se mantenga una condición. La mayoría de los lenguajes para ordenador modernos, incluyendo los dialectos más nuevos de BASIC, disponen de algo similar. Su formato es:

```
BEGIN parte bucle condición UNTIL
```

y ejecuta la parte bucle y la condición; la condición es simplemente el fin de la parte bucle que deja algo en la pila para UNTIL. Por este motivo, la parte bucle se ha de ejecutar al menos una vez. Por ejemplo:

```
:  2POTENCIAS (--)
              (imprime todas las potencias de 2
              menores que 10 000)
  1           (potencia inicial de 2)
  BEGIN
    CR DUP. (imprimir potencia de 2 en una nueva
            línea)
    2* (tomar siguiente potencia de 2)
  DUP 10 000>UNTIL (comprobar si aún no es
            demasiado grande)
  DROP (dejar última potencia de 2)
;
```



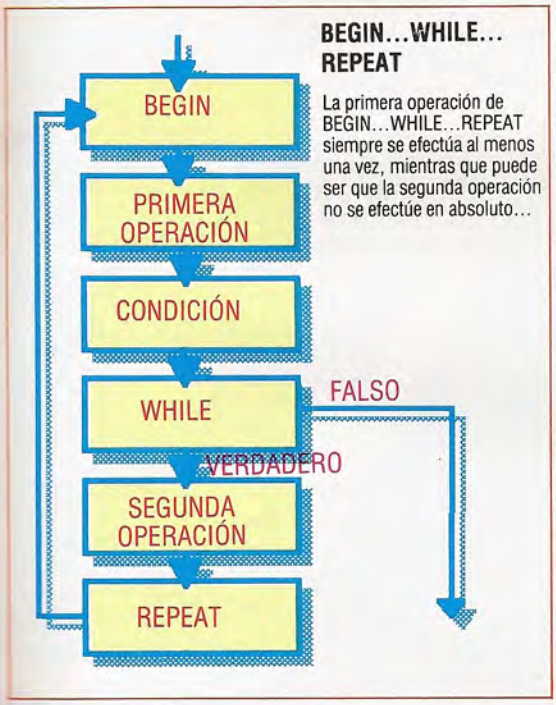
- BEGIN...WHILE...REPEAT

WHILE posee algunas posibilidades más, porque

deja que usted decida, en medio de la parte bucle, si desea salir del bucle sin esperar al fin del mismo. Su formato es:

```
BEGIN 1.ª parte bucle condición WHILE 2.ª parte bucle
  REPEAT
```

Al igual que la parte bucle de UNTIL, aquí la 1.ª parte bucle siempre se ejecuta al menos una vez y termina quedando la condición en la pila. Pero ahora hay dos importantes diferencias respecto a UNTIL. Primero, el bucle ahora se detiene cuando la condición es falsa. Cuando se interrumpe el bucle, el FORTH se salta a la 2.ª parte bucle y continúa a partir del REPEAT. En segundo lugar, si el bucle ha de continuar, el FORTH ejecuta la 2.ª parte bucle antes de volver al bucle desde BEGIN.



• DO...LOOP

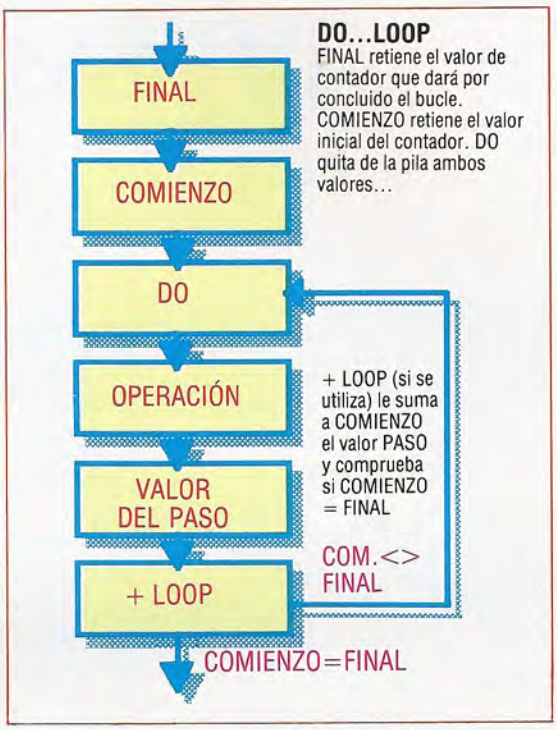
El bucle DO del FORTH hace el mismo trabajo que el bucle FOR del BASIC y muchos otros lenguajes. Comparémoslo con las implementaciones del BASIC:

BASIC	FORTH
FOR X=inicial TO límite	(límite+1) inicial DO
cuerpo del bucle	cuerpo del bucle
NEXT X	LOOP
FOR X=inicial TO límite STEP paso	(límite+1) inicial DO
cuerpo del bucle	cuerpo del bucle
NEXT X	paso+LOOP

Existen algunas diferencias evidentes.

En primer lugar, como cabría esperar, el valor inicial y el límite preceden a DO, de modo que DO pueda tomarlos de la pila. El límite va primero. Menos obvio es el hecho de que el límite es uno más el valor final con el cual se desea efectuar el bucle, de modo que 4 0 DO hace el bucle con los valores 0, 1, 2 y 3 (cuatro veces), pero no con 4.

Como en BASIC, el paso (*step*) es opcional. Si usted lo omite y usa LOOP, el paso se toma como 1.



Si lo usa, entonces necesitará +LOOP en vez de LOOP, indicando el paso justo antes de +LOOP.

En BASIC siempre hay una variable de control (X, en el ejemplo anterior). En FORTH no se utiliza ninguna variable como ésta; en cambio, hay una palabra I que coloca en la pila el valor del bucle. Si tiene un bucle DO dentro de otro, entonces I hace referencia al valor del bucle más interior y J hace referencia al siguiente hacia afuera. De haber más bucles DO, luego sus valores de bucle todavía tienen entidad y se cuentan correctamente.

He aquí un ejemplo que eleva un número a la potencia de otro. A diferencia del BASIC, el FORTH no tiene incorporado ningún operador de este tipo; usted define el suyo propio:

```
: **(m,n--m**n)
  DUP 0 < IF (si n<0 la respuesta se toma como 0)
    DROP DROP 0
  ELSE
```

Condiciones

Las siguientes palabras son útiles para elaborar condiciones para IF, UNTIL y WHILE. Sus resultados, lo que dejan en la pila, son bien −1 para verdadero, bien 0 para falso, pero sólo en FORTH-83. Los FORTH más antiguos dejan 1 para verdadero y 0 para falso. Ni IF, ni UNTIL ni WHILE insisten en que se les dé 0, 1 o −1; 0 es falso y cualquier otro número es verdadero.

=	m,n -- verdadero o falso (verd. si m = n)
<	m,n -- verdadero o falso (verd. si m <n)
>	m,n -- verdadero o falso (verd. si m > n)
0=	m -- verdadero o falso (verd. si m = 0)
0<	m -- verdadero o falso (verd. si m < 0)
0>	m -- verdadero o falso (verd. si m > 0)
NOT	verdadero o falso -- falso o verdadero
AND	m,n -- m AND n
OR	m,n -- m OR n

```
    DUP 0=F (si n=0 la respuesta es 1)
      DROP DROP 1
    ELSE
        1 (multiplicará éste por m n veces)
        SWAP 0 DO (para hacer bucle n veces)
          OVER * (mult. parte sup. de pila por m)
        LOOP
        SWAP DROP (dejar m)
      THEN
    THEN
;
```

Los casos especiales son bastante complicados. Primero, si la potencia es negativa, la respuesta correcta es 1 dividido por la respuesta, con la correspondiente potencia positiva. Dado que el FORTH sólo trabaja con enteros, no se puede efectuar esta división exactamente, de modo que es razonable decir que ** devuelve 0 como resultado. El caso especial en el cual n=0 es más sutil. A la vista del mismo, si multiplica 1 por m cero veces (pasando por el bucle cero veces), entonces obtendrá en cualquier caso el resultado 1. Sin embargo, un bucle DO del FORTH siempre se ejecuta al menos una vez y esto arruina el caso en el cual n=0. De hecho, en muchas ocasiones usted no desea necesariamente que el bucle DO se ejecute en un programa, en cuyo caso ha de tomar precauciones especiales.

He aquí otro ejemplo. Se trata de una versión más amplia de 2POTENCIAS, que ofrecimos anteriormente. Utiliza no sólo I, el valor de bucle, sino también la palabra LEAVE, que interrumpe el bucle inmediatamente y continúa desde LOOP o +LOOP:

```
:2 POTENCIAS    (n--)
                (visualiza índices y potencias de
                2 mientras las potencias sean
                menores que n)
    15 0 DO (2**14 es el mayor que se puede
      imprimir)
      DUP 2 I**< IF (si 2**I>n)
        LEAVE
      ELSE
        CR I. 2 I**.
      THEN
    LOOP
    DROP (deja a n)
;
```

Con estas estructuras de programación, usted puede hacerlo sin el GOTO y sin números de línea; en realidad, el FORTH no los posee. Después de haber utilizado el BASIC, le llevará un poco de tiempo habituarse a esto, pero el hecho es que los números de línea son algo impuesto por el BASIC y no constituyen ninguna parte intrínseca de la forma en que uno concibe un programa. Si usted hubiera escrito 2POTENCIAS, ¿se hubiera acordado del DROP del final? Esto ilustra la importancia de ir observando muy de cerca lo que una palabra hace con la pila. Para facilitar las cosas conserve sus definiciones de palabras lo más cortas posible, descomponiendo el problema, como en la segunda definición de 2POTENCIAS. En ese caso, aunque la palabra hace algo más al dejar que usted imponga su límite sobre la pila en lugar de 1000, no por ello es más complicado. Si hubiera pensado en modificar la primera definición para hacer lo mismo, enseguida habría obtenido toda clase de SWAPs y OVERs. La segunda definición es simple porque usted ha dividido el problema definiendo **, lo que le ofrece dos definiciones más cortas en lugar de una.

Imprimiendo series

Para imprimir una serie desde el interior de una definición de dos puntos, se utiliza ." así:

```
." serie"
```

Se necesita al menos un espacio tras el ." porque se trata de una palabra. Si tuviera más de uno, entonces los espacios que siguieran al primero se considerarían como parte de la serie.

La serie sigue al ." (de modo que no está en notación polaca inversa) porque la pila del FORTH no puede manipular series. Las variables en serie de hecho no se proporcionan como estándar en FORTH, pero existen formas de ampliar el lenguaje para manipularlas.

La palabra CR (retorno de carro) envía a la pantalla una nueva línea

Limitaciones

Si bien las ideas principales tras el DO son bastante directas, también tiene algunas particularidades. En primer lugar, si el paso (antes de +LOOP) es negativo, el valor del bucle realmente puede alcanzar el límite en vez de detenerse justo antes del mismo. De modo que 2 0 DO ... −1+LOOP efectúa el bucle a través de los valores 0, −1 y −2. En segundo lugar, hemos visto que un bucle DO siempre se ejecuta al menos una vez, aun para 0 0 DO...LOOP, que quizá usted quisiera saltarse. El FORTH-83 lo sorprenderá ejecutando el bucle 65 536 veces. −1 0 DO...LOOP lo hará 65 535 veces, y así sucesivamente.

La respuesta más simple no es accidental, sino que tiene sentido si se es consciente de la forma en que se pueden tratar los enteros del ordenador, ya sea con signo o sin signo (complemento de a dos). Si el valor inicial ya ha pasado el límite, entonces subirá hasta 32 768, que es tratado como equivalente de −32 768, y después seguirá hacia arriba a través de los enteros negativos hasta llegar al límite desde abajo. En los FORTH más antiguos, si el valor inicial ya ha superado el límite, el bucle se efectúa sólo una vez. Asimismo, los FORTH más antiguos tratan el LEAVE de forma algo distinta. No saltan directamente fuera del bucle, sino que se aseguran de que se efectuará una salida la próxima vez que se encuentre LOOP o +LOOP

Datos básicos (VI)

Proseguimos el análisis del mapa de memoria del C64, por cortesía de la Commodore Business Machines

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
PNTR	00D3	211	Columna cursor en la línea actual
QTSW	00D4	212	Flag: editor en modo comillas, $00 = NO
LNMX	00D5	213	Longitud de línea física de la pantalla
TBLX	00D6	214	Número línea física cursor actual
	00D7	215	Área temp. de datos
INSRT	00D8	216	Flag: Modo insert, >0 = #INSTs
LDTB1	00D9-00F2	217-242	Tabla enlace línea pantalla/editor temp.
USER	00F3-00F4	243-244	Puntero: pos. RAM color actual pantalla
KEYTAB	00F5-00F6	245-246	Vector: tabla decodificación teclado
RIBUF	00F7-00F8	247-248	Puntero buffer entrada del RS232
ROBUF	00F9-00FA	249-250	Puntero buffer salida del RS232
FREKZP	00FB-00FE	251-254	Espacio página 0 libre para programas usuario
BASZPT	00FF	255	Área datos temp. del BASIC
	0100-01FF	256-511	Área pila del sistema microprocesador
	0100-010A	256-266	Área trabajo flotante para cadenas
BAD	0100-013E	256-318	*Log* para error entrada cinta
BUF	0200-0258	512-600	Buffer de INPUT sistema
LAT	0259-0262	601-610	Tabla núcleo: números archivo lógico activo
FAT	0263-026C	611-620	Tabla núcleo: número dispositivo para cada archivo
SAT	026D-0276	621-630	Tabla núcleo: segunda dirección para cada archivo
KEYD	0277-0280	631-640	Cola buffer teclado (FIFO: *first in, first out:* primero en entrar, primero en salir)

Crispin Thomas

Acertadas mejoras
El Amstrad CPC 664 es una versión mejorada del muy popular micro CPC 464. En lugar de una platina de cassette, esta máquina lleva incorporada una unidad de disco estándar Hitachi de 3 pulgadas. Se han agrandado las teclas del cursor para una mayor facilidad de uso y, además, Amstrad ha mejorado la ROM de BASIC para incluir diez instrucciones adicionales, así como las instrucciones del DOS

Sucesor del CPC 464

El nuevo ordenador presentado por Amstrad, el CPC 664, cuenta con una unidad de disco incorporada y un diseño integral, que se suman a los méritos de su antecesor, el CPC 464

El ordenador personal Amstrad CPC 464 fue muy elogiado cuando se lanzó en 1984. Aunque no representaba ninguna innovación desde el punto de vista técnico, la máquina constituyó un éxito, al que contribuyeron, sin duda alguna, sus excelentes gráficos y su fiable hardware. Un año después, la empresa lanzaba una segunda máquina, el Amstrad CPC 664, esencialmente similar a la anterior pero con una unidad de disco incorporada en lugar de la platina de cassette.

El trazado del CPC 664 es idéntico al del anterior CPC 464, ahora con algunas teclas de color celeste en vez de verdes. Amstrad también ha decidido numerar las teclas del relleno numérico F1, F2, y así sucesivamente, si bien las teclas cumplen exactamente las mismas funciones. La otra diferencia entre los dos teclados reside en el mayor tamaño de las teclas del racimo del cursor, que permiten manipular el cursor con mayor facilidad.

Con una unidad de disco incorporada, obviamente han desaparecido las teclas de la cassette, y en su lugar hay ahora una unidad de Hitachi estándar de 3 pulgadas. La unidad de disco parece mucho más pequeña que la unidad de disco externa, porque la unidad del CPC 664 funciona con una fuente de alimentación de 12 V de la pantalla que viene con el ordenador.

Dado que el CPC 464 tenía su propia platina de cassette incorporada, no había necesidad de incluir una puerta para cassette; pero en el 664 sí se ha añadido esta facilidad, para que los usuarios puedan aprovechar la base de software que hayan ido creando en cinta para la máquina más antigua.

El Amstrad CPC 464 posee un único bus de ampliación destinado no sólo a la adición de una unidad de disco flexible, sino también como interface para periféricos de fines generales. El Amstrad 664 no sólo tiene una interface para periféricos, sino además una puerta adicional para una segunda unidad de disco flexible. Ésta es una facilidad importante si la máquina ha de utilizar CP/M, porque muchos paquetes escritos para trabajar bajo este sistema operativo exigen que el disco de aplicaciones esté en una unidad y el disco de datos en otra. Por supuesto, se puede ejecutar CP/M disponiendo de una sola unidad, pero esto por lo general supone tener que ir cambiando los discos.

Al producir la nueva máquina, Amstrad ha tenido oportunidad de mejorar la ROM de BASIC para que incluya algunas instrucciones nuevas. Están, por supuesto, las instrucciones del sistema operativo de disco, que son idénticas a aquellas incorporadas en la DDI ROM de la interface para disco flexible externo. Muchas de las nuevas instrucciones añaden configuraciones a la ya poderosa lista de instrucciones para gráficos de que dispone el programador de BASIC. Probablemente la más importante de ellas sea la instrucción FILL, una curiosa omisión en el BASIC original. Esta instrucción rellenará una superficie ya sea con el color de primer plano actual o bien con el color que determine el programador.

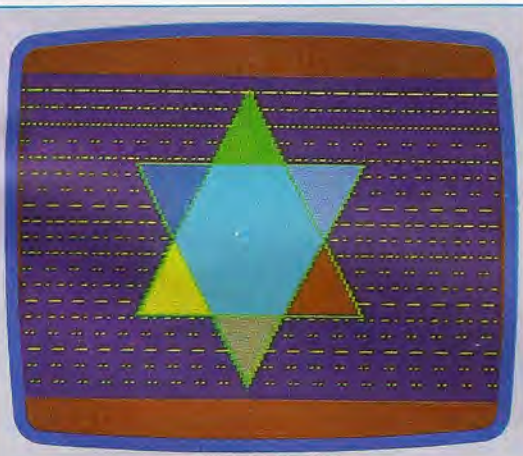
Instrucciones para gráficos

El programador de gráficos dispone asimismo de la instrucción MASK, que puede producir una línea de puntos. La instrucción se escribe con el formato MASK i,p, donde i es un entero entre 1 y 255, y p determina si se debe trazar o no el primer punto. Como su nombre sugiere, la instrucción es una «máscara» que lleva a cabo una operación lógica de gestión sobre la celda de caracteres. De modo que MASK 1,p producirá un solo punto cada ocho pixels, mientras que MASK 17,p producirá dos puntos. Establecida en 255, MASK presentará una línea continua.

GRAPHICS PEN y GRAPHICS PAPER son dos instrucciones relacionadas con MASK. Normalmente, cuando dibujamos una línea no podemos ver el color de fondo. Sin embargo, cuando se traza una línea de puntos con la instrucción MASK, es deseable ver el color del fondo. De modo que GRAPHICS PAPER nos permite establecer los gráficos del fondo ya sea en el color PAPER actual o bien en algún otro color. Del mismo modo, GRAPHICS PEN establece el color de fondo para las líneas o los puntos.

Para refinar aún más el proceso de escribir gráficos en la pantalla, se ha añadido una instrucción adicional, FRAME. Cuando se producen gráficos, se suele observar en la pantalla cierto grado de parpadeo. Ello se debe a que los gráficos están intentando aparecer en la pantalla en medio de una exploración.

El efecto de la instrucción FRAME es interrumpir la ejecución del programa en BASIC hasta que la exploración recomience en la parte superior de la pantalla, de modo que se puedan incluir los gráficos en un solo cuadro. Esta instrucción, por consiguiente, produce una visualización mucho más uniforme.



Control de instrucciones
Esta instantánea de pantalla muestra tres de las nuevas instrucciones de que dispone el Amstrad CPC 664. Las líneas de puntos del fondo se dibujan pulsando MASK, mientras que los puntos de la estrella se trazan mediante DRAW (que dibuja en una posición relativa a la posición actual del cursor). Por último, las superficies del interior de la estrella se colorean mediante FILL

Para simplificar la manipulación de datos en la pantalla se ha añadido COPY CHR$. Esta instrucción significa que los caracteres de una parte de la pantalla se pueden transferir a otra zona. Esto es útil para aplicaciones de gestión, en las que, p. ej., se puede trasladar a cualquier parte una lista de cifras o de direcciones de una ventana con el fin de procesarlas.

Éstos son apenas unos ejemplos de las numerosas instrucciones que se han añadido al BASIC. Aunque se pretende que la ROM sea compatible con la versión de BASIC anterior, ciertos paquetes diseñados para ejecutarse en el CPC 464 no funcionarán en el 664. Ello no se debe a un error de Amstrad. Al igual que otros muchos fabricantes, la empresa se ha reservado el derecho de mejorar el BASIC cuando lo crea necesario. Con el fin de mantener la compatibilidad, la empresa incluyó un bloque de saltos para vectorizar las llamadas a direcciones en ROM.

Lamentablemente, al objeto de proporcionar velocidad adicional, algunos fabricantes de software independientes han pasado por alto el bloque de saltos, llamando directamente a las rutinas. Debido a que en la nueva versión de la ROM muchas de estas rutinas han cambiado de dirección, una gran parte del software ya no servirá.

El Amstrad CPC 664 está destinado a ser un ordenador tanto para pequeña gestión como para aficionados personales. Como tal, parece ofrecer excelentes prestaciones. No obstante, quedan pendientes uno o dos interrogantes acerca de la viabilidad de la nueva máquina como ordenador de gestión. En primer lugar, está el perenne fantasma de la base de software. Aunque Amstrad ha adoptado el CP/M como sistema operativo de disco, hasta el momento los discos que utilizan sus ordenadores no han conseguido hacerse de una gran popularidad. De hecho, esto significa que normalmente los clientes tendrán que esperar a que aparezca el software en el formato correcto.

Sin embargo, gracias a un acuerdo establecido entre Amstrad y la empresa reproductora Timatic, también se pueden copiar discos CP/M de formato de 5 1/4 pulgadas. También existe la posibilidad, si fuera necesario, de adquirir una unidad de disco adecuada de algún proveedor independiente, para usar conjuntamente con la unidad Hitachi.

Algunos de los paquetes CP/M estándares, como el *WordStar* o el *dBase II*, no se ejecutarán en el Amstrad. Ello se debe a que el ordenador sólo tiene 39 Kbytes libres para programas de aplicaciones cuando opera bajo CP/M. No obstante, se ha anunciado la aparición de una placa de ampliación de memoria para el ordenador, que permitirá que los usuarios saquen el máximo partido de la enorme base de datos CP/M.

AMSTRAD CPC 664

INTERFACES

Bus de ampliación, interface para segunda unidad de disco flexible, puerta para cassette, puerta E/S, interface para palanca de mando, interface para impresora, entrada 12 V, entrada 5 V, conector para pantalla

SOFTWARE SUMINISTRADO

El CPC 664 viene con CP/M 2.2 y Dr LOGO

DOCUMENTACION

El manual contiene detalladas explicaciones sobre el BASIC Amsoft, Dr LOGO y el CP/M, y es una recopilación de los manuales del CPC 464 y del disco flexible DDI

PUNTOS FUERTES

Atendiendo a su precio, es difícil encontrarle defectos al CPC 664 como máquina de pequeña gestión. Hace unos pocos años, un sistema similar habría costado varias veces su precio

PUNTOS DEBILES:

Hasta que se rectifique con la introducción de una placa para ampliación de memoria, el ordenador sufrirá de carencia de espacio de memoria disponible para ejecutar los paquetes CP/M utilizados más comúnmente

Parte posterior
Aunque las interfaces de la parte posterior son similares a las del CPC 464, se han efectuado algunas adiciones. A la derecha se ha añadido una puerta para cassette, mientras que en el lado izquierdo se ha incluido una interface para unidad de disco flexible, lo que deja libre el bus de ampliación para otros usos. El cable unido al ordenador se alimenta con la pantalla para operar la unidad de disco incorporada

Preparando el escenario

El diseño de juegos de aventuras exige un enfoque sumamente estructurado

El programa que escribiremos para proporcionarle un entorno a nuestra rutina de personajes interactivos tendrá la estructura que se refleja en el diagrama. Iremos trabajando el programa paso a paso, comenzando por las rutinas de inicialización. Para simplificar, utilizaremos matrices en serie para almacenar todos los datos y leeremos los valores por defecto en las matrices desde sentencias de datos.

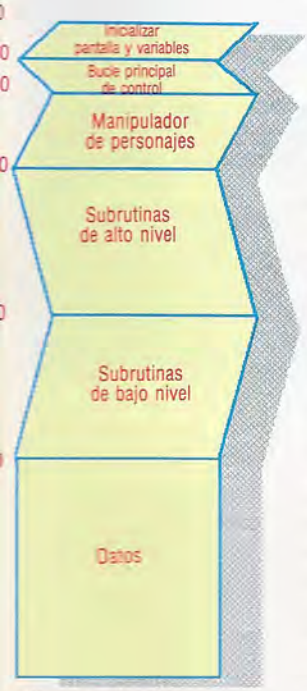
Primero y principal, necesitamos tres descripciones de escenarios: una para cada habitación del concurrido *pub* Dog and Bucket (El perro y el cubo). Asimismo, necesitamos algún medio de saber cómo están comunicadas las habitaciones, de modo que si un personaje avanza hacia el este desde la sala de tertulia, por ejemplo, sepamos que terminará en el salón. La matriz bidimensional I$ retiene toda esta información de la forma indicada en nuestra *Tabla de matrices*.

La decisión de cómo representar los datos para los objetos de nuestro juego dependerá básicamente del papel que se espera desempeñen. En este juego, nos importan los objetos como elementos para que los personajes recojan, dejen o, quizá, se arrojen los unos a los otros. En el caso de las empanadas y el bocadillo de jamón, también queremos considerar la cuestión de si son o no comestibles; y, por supuesto, ¡que haya muchísimo para beber! Por lo tanto, es necesario que las estructuras de datos de objetos sean capaces de abordar las siguientes preguntas de nuestro juego: ¿dónde está, es comestible, es bebestible?

Para hacerlo, utilizamos la matriz bidimensional b$, tal como se indica en la *Tabla de matrices*. Durante el programa podremos comprobar los elementos de esta matriz en cualquier momento, para responder a cualquiera de las tres preguntas que acabamos de enumerar.

Los objetos y los escenarios plantean problemas, de modo que encaremos la tarea, más interesante, de decidir cómo almacenar la información sobre los personajes.

El Dog and Bucket da cobijo a los siguientes alegres parrandistas: Luis Cubas, Lola Fiestas, Pepe Viñas, Mari Tapas, Javi Salado y Gina Fizz. También incluiremos otro personaje: Fred, el barman. Este último personaje no es interactivo; el barman se incluye simplemente en una descripción de escenario y el manipulador de personajes ocasionalmente visualizará mensajes sobre sus «acciones». De esta forma, vemos que un personaje, para ser eficaz, no requiere necesariamente una programación y una manipulación complejas durante el tiempo de ejecución.



Una buena jugada
La adopción de un enfoque modular a la programación nos permitirá modificar nuestro juego en una etapa ulterior si así lo deseáramos, así como adaptar el manipulador de personajes para ejecutarlo con otros programas menos comprometidos. Como en la mayor parte del software de aventuras, el grueso de la memoria lo ocupan los datos, la mayoría de los cuales serán mensajes de texto para imprimir en la pantalla

Tabla de objetos

Número objeto	Descripción	Escenario inicial	¿Comestible?	¿Bebestible?
1	Un vaso de cerveza	2	N	S
2	Una lata vacía de comida para gatos	3	N	N
3	Empanada de la casa	1	S	N
4	Una banqueta de bar	2	N	N
5	Un cenicero	1	N	N
6	Un bocadillo de jamón rancio	2	S	N
7	Una pinta de cerveza amarga	0	N	S
8	Una crema de menta	0	N	S
9	Un whisky con soda	0	N	S
10	Un vodka puro	2	N	S
11	Una pinta de cerveza añeja	0	N	S
12	Una ginebra con ginger ale	0	N	S

Lección objetiva
En el Dog and Bucket se pueden encontrar 12 objetos diferentes. Observe que los elementos del 7 al 12 son «propiedad» de los personajes (ver *Tabla de personajes*) y, por tanto, con la excepción del número 10, parten desde el escenario 0, ya que los llevan consigo sus dueños. Al comienzo, sin embargo, Mari Tapas ha extraviado su trago (número 10), el cual se halla en el salón

Para decidir los atributos que necesitamos almacenar para nuestros personajes, recordemos primero la lista de atributos que considerábamos necesarios para una "persona" controlada por ordenador. Primero, el desplazamiento de un escenario a otro es esencial, por lo que es obvio que necesitamos llevar el registro de la posición de un personaje. Segundo, los personajes han de ser capaces de manipular objetos, por lo que en algún lugar a lo largo de la línea se debe incluir un registro del inventario de cada personaje.

Los otros atributos fundamentales de un personaje están relacionados con la comunicación con el jugador y su conciencia del entorno. En realidad, no es necesario implementar el primer aspecto en la rutina manipuladora de personajes propiamente dicha. Para entender por qué, piense tan sólo en lo que sucede cada vez que usted entra una instrucción en un juego. Si digita, por ejemplo, Coger daga, y el arma está presente, el programa corregirá

Tabla de personajes

	Escenario 2	Inventario 3	Fortaleza 4	Humor 5	Objeto 6	Sexo 7	LCH 8	LCD 9	Frec. manipul. 10	Frec. mov. 11
1 Luis Cubas	2	7	10	10	7	V	0	0	7	4
2 Lola Fiestas	1	8	30	10	8	M	0	0	3	5
3 Pepe Viñas	1	9	8	10	9	V	0	0	4	6
4 Mari Tapas	2	0	20	10	10	M	0	0	5	5
5 Javi Salado	2	11	10	6	11	V	0	0	4	6
6 Gina Fizz	1	12	15	6	12	M	0	0	5	5
7 Ud.	1	0	255	255	–	?	0	0	0	0

David Higham

Dramatis personae
Nuestra tabla refleja los valores iniciales de los diferentes atributos para cada uno de los personajes del Dog and Bucket. Todos los personajes empiezan con sus propios tragos en su inventario, con excepción de Mari Tapas. El personaje número 7 representa al jugador, ¡y se ha incluido para asegurar que los otros personajes le "presten atención" a usted! No obstante, el factor de manipulación del personaje 7 es cero, lo que asegura que sus acciones serán de su entera responsabilidad y no estarán sujetas a la rutina manipuladora de personajes

la variable que retiene la posición de la daga para indicar que ahora se la está transportando, y luego alterará en consecuencia el inventario del jugador. Por el contrario, si usted entra una instrucción para que otro personaje tome un objeto, el programa simplemente lleva a cabo el mismo proceso, pero en esta ocasión altera el inventario del personaje en cuestión.

En este punto no existe necesidad alguna de implicar a la rutina manipuladora de personajes: el manipulador existe sólo para asegurar que éstos puedan llevar su propia vida con total independencia del jugador. Por supuesto, cuando se ejecute el manipulador de personajes, encontrará que ahora la daga está en el inventario del personaje y actuará en consecuencia.

La forma más simple de asegurar que los personajes y el jugador se lleven bien quizá sea también la más obvia: que usted se implemente a sí mismo como una persona más. Todo cuanto debe hacer es definir un personaje llamado Ud., y cada vez que se llame a la rutina manipuladora de personajes, se encontrará a sí mismo profundamente inmerso en las vidas de sus compañeros de ficción.

La comunicación con el jugador no es en principio difícil de comprender ni de implementar eficazmente en el programa (si bien tal comunicación siempre resulta bastante limitada), pero incluir la conciencia del entorno (el último requisito previo para un personaje interactivo) puede resultar muy complicado. En un grado limitado, se puede incluir esta configuración en nuestro manipulador de personajes simplemente sobre la base de los datos que ya hemos listado. La rutina puede comprobar si hay objetos presentes, manipularlos o incluso generar algún comentario inteligente sobre ellos que sea "pronunciado" por un personaje. Asimismo, puede comprobar la posición de un personaje y acaso hacer que éste haga algún comentario sobre la misma. Sin embargo, aunque útiles, las rutinas como éstas son más bien limitadas. Obviamente, lo que se requiere es una conciencia sobre los otros personajes y, en particular, la capacidad de otorgar a una «persona» controlada por ordenador algún tipo de «continuidad», es decir, proporcionar a los personajes un sentido de la historia, del cual ya hemos hablado anteriormente en esta serie.

Para hacer esto introduciremos dos nuevos atributos a almacenar para cada personaje: el código de última instrucción (LCD) y el código de último personaje (LCH). Éstos indican, respectivamente, la última acción efectuada por el personaje (o, en algunos casos, la última infligida a él), y la otra parte (si la hubiera) involucrada en esta acción. Estos atributos se pueden incluir en la matriz de datos de nuestros personajes y, a modo de ejemplo de la forma en que se podrían utilizar, consideremos el caso en el cual Luis Cubas (personaje número 1) le da una empanada a Lola Fiestas (personaje número 2).

El LCD de esta última ahora se establecerá para indicar una acción "recibir", y su LCH retendrá el número 1. La próxima vez que se llame a la rutina manipuladora, ésta comprobará los datos y podrá deducir a partir del objeto retenido en el inventario de Lola (la empanada) qué es lo que acaba de suceder. La rutina puede, entonces, decidir si Lola debe o no decir Gracias; si lo hace, se establecerá su LCD de modo que indique "comunicar" y su LCH seguirá reteniendo 1.

Si, por el contrario, Lola no emprende ninguna acción tras la entrega de Luis, tanto su LCD como su LCH se establecerán en 0, indicando que el pasado, efectivamente, se ha olvidado. El uso del LCD y el LCH puede hacerse muy complejo, como veremos más adelante en la serie cuando lo examinemos con mayor detalle. Mientras tanto, los códigos que utilizaremos para LCD se incluyen en la *Tabla de matrices*.

Seis atributos

Hay otros seis atributos que almacenaremos para nuestros personajes, cada uno de ellos útil y muy directo:

- *Fortaleza.* Este atributo determina la capacidad del personaje para moverse y comunicarse. En este caso en particular, se tratará a un personaje como si estuviera inconsciente cuando su fortaleza sea igual

Tabla de matrices

Matriz	Dimensionada en	Retiene inform. relativa a
I$	3,5 (Spectrum: 3,5,255)	Descr. del escenario (I$(n1)) y los 4 códigos de salida (I$(n,2...5))
b$	12,4 (Spectrum: 12,4,30)	Descr. del objeto (b$(n,1)) y escenario/comestible/bebestible (b$(n,2,...4)
c$	7,11 (Spectrum: 7,11,15)	Nombres de personajes (c$(n,1)) y sus atributos (c$(n,2,...11))

Códigos de última instrucción

LCD Indica

1 La última vez que se llamó al manipulador de personajes, el personaje recibió del personaje que indica el LCH el objeto de su inventario

2 Previamente el personaje dijo algo acerca del personaje que indica el LCH

3 Monólogo. El personaje acaba de hacer un comentario. Este código se utiliza para impedir que los personajes hagan los mismos o similares comentarios

4 Tomar. Durante la última ejecución del manipulador de personajes se recogió el objeto del inventario del personaje

5 Golpear. El personaje acaba de ser golpeado por un objeto que le arrojó otro personaje

6 Comer. El personaje está comiendo el objeto de su inventario

7 Entrar. El personaje acaba de entrar en el escenario actual

Patrón de retención

Las tres matrices principales (I$, b$ y c$) retienen todos los datos necesarios relativos a escenarios, objetos y personajes. Observe que no se utilizan elementos cero (para facilitar la compatibilidad entre los diferentes BASIC, algunos de los cuales no disponen de una facilidad elemento cero). Se emplearán matrices en serie para retener datos tanto alfanuméricos como numéricos (usando STR$ y VAL$). El manipulador de personajes utiliza los códigos de última instrucción (LCD) para conferirles a los personajes algún pequeño grado de continuidad o «historia», y están retenidos en c$(n,9) (ver *Tabla de personajes*).

Tabla de escenarios

Escenario	Descripción	N S E O
1	Usted se halla en la sala de tertulia del Dog and Bucket. En un rincón hay varios personajes dudosos jugando al dominó. Detrás del mostrador está Fred, el barman, con su habitual aspecto festivo. Hay una puerta que da al este.	0 0 2 0
2	He aquí el salón del pub, al cual le vendría muy bien una decoración total. Parece que el suelo ha sido regado regularmente con cerveza derramada. Hay puertas hacia el oeste y hacia el sur.	0 3 0 1
3	¡Uf! Ésta es la cocina, donde se preparan las famosas empanadas de carne de la casa para una clientela que siempre tiene apetito. Hay numerosas latas vacías de comida para gatos, lo que no deja de ser extraño, ya que no hay gato	2 0 0 0

Salida, plataforma oeste

Cada uno de los tres escenarios posee su propia descripción junto con los cuatro códigos de salida, uno para cada punto cardinal. El código de salida retiene el número de destino; de modo que, p. ej., la salida hacia el norte desde la cocina conduce al escenario 2, el salón. El número cero indica que el movimiento en esa dirección es imposible

o menor que cero. Ciertas acciones (como beber demasiado) reducirán la fortaleza de un personaje, mientras otras, por el contrario, la incrementan.

- *Humor.* Determina la forma en que los personajes se tratan entre sí. Por ejemplo, si la rutina manipuladora encuentra que un personaje está transportando un objeto determinado, quizá llegue a un punto en el cual tenga que decidir si es necesario abandonar el objeto o conservarlo. Si el humor del personaje es particularmente escaso (supongamos, de casi cero) el manipulador acaso determine que el objeto deba arrojarse contra alguien. Entre los factores que influyen sobre el estado de ánimo en el Dog and Bucket se incluye el extraviar el trago que se estaba bebiendo.

- *Objeto.* Cada personaje del programa tiene su objeto "propio", registrando este atributo el número de cada uno. En este caso particular, los objetos son todos bebidas de diversas descripciones, y perder o encontrar el trago de uno tiene ciertos efectos sobre la conducta del personaje.

- *Sexo.* Este atributo registra si un personaje es masculino o femenino. Se utiliza, por ejemplo, para determinar los pronombres correctos a emplear en los mensajes.

- *Frecuencia de manipulación.* La frecuencia de manipulación (HF) determina el tiempo que debe transcurrir antes de que la rutina manipuladora de personajes compruebe al personaje. Por ejemplo, sobre un personaje con una HF de 5 se actuará una vez de cada cinco que se llame a la rutina. Un HF de 0, sin embargo, significará que la rutina no pondrá al día a ese personaje en absoluto. Junto con el atributo de frecuencia de movimiento (del que hablamos a continuación), la HF se puede utilizar para «congelar» los personajes (estableciendo HF en 0), demorarlos (aumentando HF) o darles el aspecto de ser más activos (disminuyendo HF).

- *Frecuencia de movimiento.* Quizá deseemos que ciertos personajes sean activos en relación a sus compañeros y su entorno, pero al mismo tiempo necesitemos que se mantengan en un escenario o asegurarnos de que no se vayan desplazando con demasiada rapidez. La frecuencia de movimiento determina la frecuencia con la que la rutina manipuladora intentará desplazar al personaje.

En el diagrama de la página anterior podemos ver los valores iniciales para cada uno de los atributos de los personajes.

Alta estrategia

Diseñaremos una rutina que permita al ordenador realizar movimientos estratégicos

Probablemente el *go* sea uno de los juegos de tablero más difíciles de informatizar. Las dimensiones del tablero y la flexibilidad de los movimientos restringen severamente el uso de árboles de juego anticipados, habituales en la mayoría de los juegos de pericia computerizados, incluyendo el ajedrez, el backgammon y las damas.

En un juego como el ajedrez, donde el número posible de movimientos es relativamente limitado, de una media de alrededor de 30 desde cualquier posición dada, es factible examinarlos a todos de forma detallada, asegurando, de ese modo, que el ordenador encontrará al menos un movimiento para efectuar, ¡aun cuando el movimiento consista en abandonar, devolviendo el rey a su sitio! Lamentablemente, nuestro sistema de evaluación para el *go* no puede trabajar de forma eficiente de esta manera. Nuestra rutina de evaluación de grupos sólo comprobará los movimientos junto a grupos de menos de tres licencias. Si en el tablero no hubiera ningún grupo en esta situación, la rutina no hallaría ningún movimiento. En futuros capítulos crearemos otras rutinas de evaluación basadas en técnicas simples de emparejamiento de patrones, que también hallarán movimientos sólo en ciertas situaciones. Lo que realmente necesitamos es una evaluación «general», que con toda seguridad siempre hallará al menos un movimiento posible, en el supuesto de que haya alguno disponible.

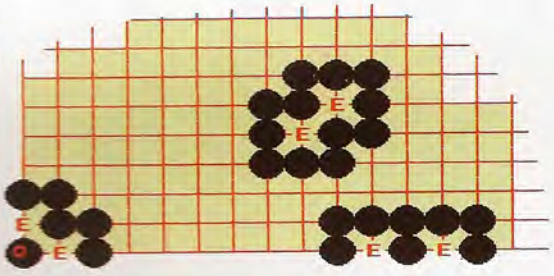
La forma más simple de hacer esto sería limitarse a efectuar cualquier movimiento al azar. Esta rutina sería algo así como:

```
5000 DEF PROCmovimient_azar
5010 LOCAL L%
5020 :
5030 FOR L%=17 TO 255
5040 IF FNlegalidad (L%,negras%)=0 THEN
     posición%=L%
5050 NEXT
5060 ENDPROC
```

Esta rutina devolverá un movimiento legal, en caso de que exista alguno. La rutina se podría mejorar comenzando el bucle en una posición aleatoria del tablero y, posiblemente, asignándole a la ficha colocada (L%) un marcador basado en la cantidad de licencias disponibles; éste se calculará automáticamente y FNlegalidad lo devolverá a la variable clib%. No obstante, aun con estas mejoras, el ordenador no va a efectuar movimientos muy sensatos, ¡cómo no sea por pura casualidad!

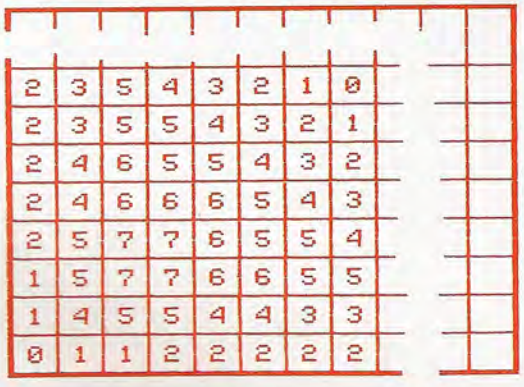
Lo que realmente se necesita es una rutina que efectúe movimientos al azar, pero sólo a posiciones razonablemente sensatas. Una forma de alcanzar este objetivo consiste en "evaluar" el tablero. Ésta es una técnica que se utiliza con frecuencia en juegos como "Otelo", en los que ciertas posiciones del tablero son más ventajosas que otras.

Aunque es difícil decidir las ventajas y desventajas de posiciones específicas, se considera que ciertas zonas del tablero son mejores que otras. Por ejemplo, consideremos la cantidad de piezas necesarias para rodear por completo dos licencias separadas. Recordará que se las denomina *ojos* (E) y que cualquier grupo con dos o más ojos se halla a salvo de captura. El número de fichas que se necesitan para formar un grupo seguro de esta manera es variable, según el grupo se halle en la esquina, a un lado o en el centro del tablero. Para formar un grupo seguro en la esquina sólo se requieren seis fichas y esto, obviamente, es mejor que las doce fichas que se precisan en medio del tablero.



Los primeros movimientos más probables serán en o cerca de las posiciones de inferioridad de la esquina. Desde aquí, en el curso habitual del juego, los jugadores se extienden a lo largo de los lados del tablero (en las líneas tercera o cuarta) en un intento por ganar territorio. Luego, de ser atacado, usted tiene la ventaja de poder utilizar su ficha de esquina de cuarto rango como base desde donde conseguir dos ojos en la esquina.

Habiendo explicado estas tácticas básicas, las estimaciones del tablero que se ofrecen en el diagrama tendrán algún sentido. Éstas son simétricas para las cuatro esquinas del tablero, estableciéndose mediante la rutina entre las líneas 1020-1230. (Observe que en la versión para el C64, la rutina se consigna fuera de secuencia, entre las líneas 363 y 383, debido a la falta de una instrucción RESTORE numero-linea para restaurar el puntero de datos.)



Los números asignados a las diversas posiciones se han escogido sobre la base de las tácticas que acabamos de describir, pero de ningún modo son los mejores. Quizá quiera probar estimaciones diferentes para comprobar su efecto. Una forma de

"regular" estas estimaciones consiste en hacer que el ordenador juegue consigo mismo, con cada bando usando una tabla de estimaciones diferente. Esto se puede hacer de la misma forma en que estableciamos al ordenador como árbitro en un juego para dos jugadores. Simplemente modifique las líneas 60 y 80 para que recen:

```
60 movimiento%=movimiento%+1:negras%=2:
   blancas%=1:PROCmovimiento_negras
80 movimiento%=movimiento%+1:negras%=1:
   blancas%=2:PROCmovimiento_negras
```

(Ver el recuadro *Complementos al* BASIC para los equivalentes para Amstrad, C64 y Spectrum.) Para incluir los cambios, primero modifique la línea 220:

Módulo 5

BBC Micro:

```
220 DIM tablero% 255, estimaciones%255
1010 :
1020 DEF PROCleer_estimaciones
1030 LOCAL A%,B%,C%,D%,X%,Y%,V%
1040 RESTORE 1150
1050 FOR Y%=1 TO 8
1060     FOR X%=Y% TO 8
1070     A%=16*Y%+X% : C%=16*Y% + (16-X%)
1080     B%=16*X% + Y% : D%=16 * X%+(16-Y%)
1090     READ V%
1100     estimaciones%?A%=V%:estimaciones%?(272-
         A%)=V%
1110     estimaciones%?B%=V%:estimaciones%?(272-
         B%)=V%
1120     estimaciones%?C%=V%:estimaciones%?(272-
         C%)=V%
1130     estimaciones%?D%=V%:estimaciones%?(272-
         D%)=V%
1140     NEXT:NEXT
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA   4,5,5,4,4,3,3
1170 DATA     7,7,6,6,5,5
1180 DATA       7,6,5,5,4
1190 DATA         6,5,4,3
1200 DATA           4,3,2
1210 DATA             2,1
1220 DATA               0
1230 ENDPROC
1240 :
1250 REM ***************************************
1350 PROCleer_estimaciones
2620 IF posición%=0 THEN PROChallar_algun_
     movimiento:T$="REND"
2630 IF posición%=0 THEN fin%=TRUE:ENDPROC
2820 marcador=(8*S%/L%-
     clib%+2*L%)*estimaciones%?tloc%(Q%)
3510 :
3520 DEF PROChallar_algun_movimiento
3530 LOCAL L%,hi,marcador
3540 hi=-9999
3550 FOR L%=17 TO 255:
     marcador=RND(1)+estimaciones%?L%
3560     IF (L% AND 240)=0 OR (L% AND 15)=0 OR marcador
         <=hi THEN 3580
3570     IF FNlegalidad (L%,negras%)=0 AND clib%>2 THEN
         hi=marcador: posicion%=L%
3580     NEXT
3590 ENDPROC
3600 :
3610 REM ***************************************
3800 tablero%?P%=0:IF C%=negras% THEN
     estimaciones%?P%=0
```

Commodore 64:

```
220 TABLERO=49152:ESTIMACIONES=TABLERO+256
305 GOSUB 363
362 :
363 REM RUTINA LEER-ESTIMACIONES
364 RESTORE:FOR X=1 TO 4:READ Y:NEXT
365 FOR Y=1 TO 8
366 FOR X=Y TO 8
367 A%=16*Y+X:C%=16*Y+(16-X)
368 B%=16*X+Y:D%=16*X+(16-Y)
369 READ V%
370 POKE ESTIMACIONES+A%,V%:POKE
    ESTIMACIONES+272-A%,V%
371 POKE ESTIMACIONES+B%,V%:POKE
    ESTIMACIONES+272-B%,V%
372 POKE ESTIMACIONES+C%,V%:POKE
    ESTIMACIONES+272-C%,V%
373 POKE ESTIMACIONES+D%,V%:POKE
    ESTIMACIONES+272-D%,V%
374 NEXT:NEXT
375 DATA 0,1,1,2,2,2,2,2
376 DATA   4,5,5,4,4,3,3
377 DATA     7,7,6,6,5,5
378 DATA       7,6,5,5,4
379 DATA         6,5,4,3
380 DATA           4,3,2
381 DATA             2,1
382 DATA               0
383 RETURN
384 :
385 REM ***************************************
1350 GOSUB 363
2620 IF POSIC%=0 THEN GOSUB 3520:T$="RND"
2630 IF POSIC%=0 THEN FIN%=-1:RETURN
2820 SCR=(8*BS/BL-
     CLIB%+2*BL)*PEEK(ESTIMACIONES+TLOC%(Q))
3510 :
3520 REM RUTINA HALLAR-ALGUN-MOVIMIENTO
3540 HI=-9999
3550 FOR I=17 TO 255:SCR=RND(0)+PEEK
     (ESTIMACIONES+I)
3560 IF (I AND 240)=0 OR (I AND 15)=0 OR SCR<=HI
     GOTO 3580
3570 LP%=I:LC%=NEGRAS%:GOSUB 3890:IF LL%=0 AND
     CLIB%>2 THEN HI=SCR:POSIC%=1
3580 NEXT
3590 RETURN
3600 :
3610 REM ***************************************
3800 POKE TABLERO+RP%,0:IF RC%=NEGRAS% THEN POKE
     ESTIMACIONES+RP%,0
3850 SK%(PILA%)=RP%:PILA%=PILA%+1
```

```
220 DIM tablero%255, estimación1%255,
    estimación2%255
```

La rutina PROCleer_estimaciones se puede, entonces, modificar para colocar un conjunto de estimaciones en la matriz de bytes estimaciones1%, y el otro en estimaciones2%. Ahora, añadiendo estimaciones%=estimaciones1% en la línea 60, y estimaciones%=estimaciones2% en la línea 80, el ordenador utilizará diferentes estimaciones del tablero para cada uno de los jugadores.

Habiendo calculado el tablero estimado, podemos añadir la rutina «general» PROC hallar_algun_movimiento. Es similar a la rutina de movimiento al azar, pero usa las estimaciones del tablero para hallar un movimiento «razonable». También se comprueba la variable clib%, para asegurar que el ordenador no juegue a una posición que deje a uno de sus grupos con menos de tres licencias. La línea 2620, con la observación "RND", llama a esta rutina. La indicación alude a la rutina que se utilizó para hallar el último movimiento del ordenador, de la misma forma que el comentario GP de la rutina de evaluación de grupos anterior. Si esta nueva ru-

tina no puede hallar ningún movimiento posible, entonces la línea 2630 establece la variable de fin del juego y sale del programa.

Ya que nos hemos molestado en establecer estimaciones del tablero, también podemos utilizarlas para mejorar otras rutinas de evaluación. En la rutina de evaluación de grupos, esto se puede hacer multiplicando la puntuación por la estimación del tablero (línea 2820). Nuevamente, acaso le interese intentar cambiar esta puntuación, en un intento por conseguir mejorar el rendimiento del ordenador.

Un problema evidente fue la tendencia del ordenador a jugar fichas en zonas de fichas previamente capturadas. Sobre la base de nuestras evaluaciones, obviamente las primeras fichas que se jueguen se colocarán en lo que el programa considera las mejores posiciones del tablero. Si usted captura algunas de estas fichas, entonces estas posiciones previamente "buenas" quedan vacantes. El ordenador, al percatarse de esto, inmediatamente comienza a jugar fichas otra vez en estas zonas, que como es evidente ya están rodeadas por sus fichas, lo cual no es una buena idea. En consecuencia, se ha añadido la línea 3800 para ofrecer una forma sencilla de estimación "dinámica" del tablero. Esto significa tan sólo que las estimaciones del tablero van cambiando a medida que avanza el juego. Por ejemplo, en el ajedrez usted evalúa el tablero para el rey. Durante los primeros movimientos del juego desearía estimaciones altas en la esquina para favorecer que el rey ocupe una posición segura. Sin embargo, al llegar al final de la partida, probablemente querrá cambiar las estimaciones para tentar al rey a una posición central más dominante.

En nuestro programa de *go*, la línea 3800 forma parte de la rutina PROCsuprimir, que suprime las fichas capturadas del tablero. Si estas fichas son negras, esta línea también cambiará la estimación del tablero para esa posición a cero. A partir de entonces, a pesar de que las negras aún pueden jugar en esta posición, esto es muchísimo menos probable.

Amstrad CPC 464/664:

```
220 tablero=&A000:estimaciones=&A100
1010 :
1020 REM rutina leer estimaciones
1040 RESTORE 1150
1050 FOR y%=1 TO 8
1060 FOR x%=y% TO 8
1070 a%=16*x%:c%=16*y%+(16-x%)
1080 b%=16*x%+y%:d%=16*x%+(16-x%)
1090 READ v%
1100 POKE (estimaciones+a%),v%:POKE (estimaciones+272-
     a%),v%
1110 POKE (estimaciones+b%),v%:POKE (estimaciones+272-
     b%),v%
1120 POKE (estimaciones+c%),v%:POKE (estimaciones+272-
     c%),v%
1130 POKE (estimaciones+d%),v%:POKE (estimaciones+272-
     d%),v%
1140 NEXT x%,y%
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA 4,5,5,4,4,3,3
1170 DATA 7,7,6,6,5,5
1180 DATA 7,6,5,5,4
1190 DATA 6,5,4,3
1200 DATA 4,3,2
1210 DATA 2,1
1220 DATA 0
1230 RETURN
1240: :
1250 REM ************************************
1350 GOSUB 1020:REM leer estimaciones
2620 IF posicion%=0 THEN GOSUB 3520:T$=«RND»:REM
     cualquier movimiento
2630 IF posicion%=0 THEN over%=1:RETURN
2820 marcador=(8*gs%/gl%-clib%+2*gl%)*PEEK
     (estimaciones+tloc%(q%))
3510 :
3520 REM rutina hallar algún movimiento
3540 hi=-9999
3550 FOR i%=17 TO 255:marcador=RND(1)+PEEK
     (estimaciones+i%)
3560 IF (i% AND 240)=0 OR (i% AND 15)=0 OR marcador <
     = hi THEN 3580
3570 lp%=i%:lc%=negras%:GOSUB 3890:IF 11%=0 AND
     clib% > 2 THEN hi= marcador:posición%=x i%
3580 NEXT i%
3590 RETURN
3600 :
3610 REM ************************************
3800 POKE (tablero+rp%),0:IF rc%=negras% THEN POKE
     (estimaciones+ rp%),0
```

Sinclair Spectrum:

```
220 LET tablero=64000:LET estimaciones=tablero
    +256
1020 REM rutina leer estimaciones
1040 RESTORE 1150
1050 FOR y=1 TO 8
1060 FOR x=y TO 8
1070 LET a=16*y+x: LET c=16*y+(16-x)
1080 LET b=16*x+y: LET d=16*x+(16-y)
1090 READ v
1100 POKE estimaciones+a,v: POKE
     estimaciones+272-a,v
1110 POKE estimaciones+b,v: POKE
     estimaciones+272-b,v
1120 POKE estimaciones+c,v: POKE
     estimaciones+272-c,v
1130 POKE estimaciones+d,v: POKE
     estimaciones+272-d,v
1140 NEXT x: NEXT y
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA 4,5,5,4,4,3,3
1170 DATA 7,7,6,6,5,5
1180 DATA 7,6,5,5,4
1190 DATA 6,5,4,3
1200 DATA 4,3,2
1210 DATA 2,1
1220 DATA 0
1230 RETURN
1250 REM ************************************
1350 GO SUB 1350: REM leer estimaciones
2620 IF posición=0 THEN GO SUB 3520: LET
     t$=«RND»
2630 IF posición=0 THEN LET fin=1:RETURN
2820 LET marcador=(8*gs/gl-clib+2*gl)*PEEK
     (estimaciones+j(q))
3510 :
3520 REM rutina hallar-algun-movimiento
3540 LET hi=-9999
3550 FOR i=17 TO 255: LET
     marcador=RND+PEEK (estimaciones+i)
3560 IF INT (i/16)=0 OR 1-16*INT(i/16)=0 OR
     marcador < = hi THEN GO TO 3580
3570 LET lp= i: LET lc=negras: GO SUB 3890: IF
     11=0 AND clib > 2 THEN LET hi=marcador:
     LET posicion=i
3580 NEXT i
3590 RETURN
3610 REM ************************************
3800 POKE tablero+rp,0: IF rc=negras THEN POKE
     estimaciones+rp,0
```

Complementos al BASIC

Mediante estas alteraciones se puede conseguir que el ordenador juegue consigo mismo. Estableciendo dos tablas de estimaciones diferentes, el ordenador las seleccionará de forma alterna, lo que permite compararlas

Amstrad CPC 464/664:

```
60  mve%=mov%+1:negras% =
    2:blancas%=1:estimaciones=
    estimaciones1:GOSUB 2540
80  mve%=mov%+1:negras%=
    1:blancas%=2:estimaciones=
    estimaciones2:GO SUB 2540
220 tablero=$A000:estimaciones1=
    &A100:estimaciones2=&A200
```

Spectrum:

```
60  LET movimiento=movimiento+1:
    LET negras=2:LET blancas=1:LET
    estimaciones=estimaciones:GO
    SUB 2540
80  LET movimiento=movimiento+1:
    LET negras=2:LET blancas=1:LET
    estimaciones=estimaciones2:GO
    SUB 2540
220 LET tablero=64000:LET
    estimaciones1=tablero+256:
    LET estimaciones2=tablero+512
```

Commodore 64:

```
60  MOVIMIENTO%=MOVIMIENTO
    +1:NEGRAS%=2:BLANCAS%=
    1:ESTIMACIONES=WL:GOSUB 254
80  MOVIMIENTO%=MOVIMIENTO%
    +1:NEGRAS%=1:BLANCAS%=2
    1:ESTIMACIONES=W2:GOSUB 25
220 TABLERO=49152:WI=TABLERO
    +256:W2=TABLERO+512
```

Por aquí, por favor

Veamos cómo se pueden añadir nuevas instrucciones al BASIC mediante la ROM de la Interface 1

Toda instrucción nueva que usted desee añadir al BASIC del Spectrum tendrá primero que romper los chequeos sintácticos y los chequeos en fase de ejecución que llevan a cabo tanto la ROM principal como la ROM de la Interface. En este caso, el control se pasa a la rutina cuya dirección está contenida en la variable de sistema VECTOR en 23735 y 23736. Hay dos formas de hacer esto.

1. Se modifica una palabra clave para que rompa los chequeos. Esto se logra pasando un número incorrecto de parámetros, sacándolo de su sintaxis normal o añadiendo un carácter (llamado *breaker*) para generar un error. Por ejemplo:

BORDER * (se añade un *breaker* tras la instrucción)
LINE 10,10,1,19 (demasiados parámetros especificados)

2. Se añade una instrucción totalmente nueva precedida de una palabra orden con un *breaker* (como *BEEP o * PRINT).

Estas instrucciones tendrán acceso, claro está, a cualquier variable del BASIC que usted desee. Nosotros nos serviremos del segundo método para añadir nuevas instrucciones.

Supongamos que queremos añadir una nueva instrucción llamada *B, para obtener una nota musical durante un segundo. Lo primero que necesitamos hacer es decidir adónde irá a parar dentro de la memoria el nuevo código. No puede ir dentro de una sentencia REM, dado que la inserción de datos que realizan las variables es imprevisible. En su lugar emplearemos CLEAR nn para ganar algo de espacio.

Para cualquier nueva instrucción que deseemos añadir necesitamos estructurar nuestros programas para incluir los pasos que ilustramos al margen. Cuando se ha entrado la rutina a la que apunta VECTOR después de que el intérprete del BASIC ha encontrado un error, precisamos conservar el carácter en el texto del programa en BASIC que ha causado la anomalía para cerciorarnos si de hecho se trata de una de nuestras instrucciones adicionales. A este carácter apunta la variable de sistema CHADD (situada en 23755 y 23756).

Para recuperar el carácter al que apunta CHADD, necesitamos dos rutinas de la ROM principal: GETCHAR y NEXTCHAR. Las operaciones de estas rutinas vienen descritas con detalle en el recuadro *Direcciones útiles* (página contigua), y mientras se está en el entorno de la ROM sombra pueden ser llamadas, tanto la una como la otra, mediante RST#10 seguido de la dirección (bien #00188, bien #0020) de la rutina requerida.

Al entrar a la rutina apuntada por VECTOR, CHADD contendrá la dirección del carácter que ha generado el error. Por ello, para ejecutar nuestra sencilla instrucción *B necesitamos ensamblar este listado:

```
D7          vector:rst  #10
1800               defw #0018        ; dirección de GETCHAR
FE2A               cp   "*"          ;es un "*"?
C2F001             jp   nz,#01F0     ;si no, error
D7                 rst  #10
2000               defw #0020        ;dirección de NEXTCHAR
FE42               cp   "B"          ;es un "B"?
C2F001             jp   nz,#01F0     ;si no, error
D7                 rst  #10          ;toma NEXTCHAR, sería
2000               defw #0020        ;...fin de sentencia
CDB705             call #05B7        ;comprueba fin de sent.
          runtime:
```

La rutina de ROM sombra que está en #05B7 debe ser entrada mientras el registro A contiene el carácter correspondiente, apuntado por CHADD. Naturalmente, usted puede alcanzar el mismo resultado empleando NEXTCHAR, como antes. En la fase de chequeo sintáctico, la rutina #05B7 provoca un retorno a la ROM principal, pero, en fase de ejecución, el retorno se producirá como si se hiciese desde una subrutina normal y el código situado en la etiqueta RUNTIME será ejecutado.

Es de notar la sencillez con que se comprueba cada carácter del nombre de la instrucción. Usted puede construir fácilmente la rutina de modo que acepte minúsculas y mayúsculas como parte del nombre de la instrucción. En cada comprobación, para el caso de no ser la letra que se espera, hay que salir a través de #01F0 a la ROM sombra (versión 1) para significar un error de sintaxis. También se notará que cuando entramos por primera vez en la rutina apuntada por VECTOR, precisamos tomar el carácter actual apuntado por CHADD, por lo que empleamos GETCHAR en #0018 mejor que NEXTCHAR en #0020.

La codificación en tiempo de ejecución para nuestra sencilla rutina es inmediata y se limita a emitir el sonido musical mediante la rutina *beep* de la ROM principal, como se muestra ahora:

```
110501     runtim:ld   de,#0105      ;duración de 1 segundo
210007            ld   hl,#0700      ;altura
D7                rst  #10
B503              defw #03B5         ;ejecuta la rutina BEEP
C3C105            jp   #05C1         ;salida
```

El salto a #05C1 concluye el asunto limpiamente, devolviendo el control a la ROM principal.

Naturalmente, usted puede añadir una instrucción nueva al BASIC que se sirva de otras rutinas de la ROM principal ya estudiadas a lo largo de esta serie de análisis (además de las rutinas ROM para gráficos que veremos en el próximo y último capítulo), llamándolas por medio de RST#10. Como ejemplo servirá el siguiente listado para la instrucción *B, que se sirve de las rutinas en #1C82 y #1E94 (ambas se detallan en el recuadro *Direcciones útiles*) para evaluar un único parámetro numérico proporcionado por el usuario. El formato de las instrucciones *B n, donde n es cualquier número entre 0 y 255. El parámetro determina la longitud de BEEP en segundos, por lo que *B 255 genera un sonido de 255 segundos de duración.

Un «caza» de la Interface
El Spectrum responde a un error con una RST #08. Tan pronto como el hardware de la Interface 1 detecta que el contador de programas del Z80 contiene esta dirección, pagina la ROM de la Interface 1 y se realiza un salto a la rutina cuya dirección está contenida en VECTOR. Esto nos permite añadir nuevas instrucciones al Spectrum, empleando una estructura de programa como el que aquí se ilustra

```
                org  60000
.     VECTOR:equ  #5CB7          ;dirección de VECTOR
CF       init:rst  #8
31            defb 49             ;establece variables IF1
2169EA        ld   hl,newcom      ;añade nueva instrucción
22B75C        ld   (VECTOR), hl   ;lo toma en VECTOR
C9            ret                 ;fin de inicialización
        ;
D7     newcom:rst  #10            ;VECTOR salta aquí
1800          defw #0018          ;GETCHAR
FE2A          cp   "*"
C2F001        jp   nz,#01F0
D7            rst  #10
2000          defw #0020          ;NEXTCHAR
FE42          cp   "B"
C2F001        jp   nz,#01F0
D7            rst  #10
2000          defw #0020
D7            rst  #10
821C          defw #1C82          ;evalúa parámetro
CDB705        call #05B7          ;comprueba final sent.
D7     runtim:rst  #10
941E          defw #1E94          ;toma parám. en a
47            ld   b,a
C5       loop: push bc            ;guarda contador
110501        ld   de,#0105       ;duración
210007        ld   hl,#0700       ;altura
D7            rst  #10
B503          defw #03B5          ;llama rutina BEEP
C1            pop  bc             ;restaura contador
10F3          djnz loop
C3C105        jp   #05C1          ;salida - final
```

Obsérvese que a causa de que la rutina en #1C82 sólo acepta un valor de ocho bits (ya que este valor es almacenado en el registro A), la entrada de valores fuera del intervalo de 0 a 255 generará error. Otro punto a tener en cuenta es que, durante la generación de notas musicales, el contador FRAMES no se incrementará y no será explorado el teclado, y, por tanto, no hay que entrar el 255 como parámetro a menos que esté dispuesto a esperar un poco...

Parámetros adicionales

Hasta aquí nos hemos ceñido en el método para añadir nuevas instrucciones por medio de entradas no habituales que generen error. Sin duda que es posible también emplear las palabras clave del BASIC Sinclair, aunque habrán de ser modificadas para que generen el error perseguido. La manera más sencilla de conseguirlo consiste generalmente en añadir un parámetro adicional, de aquí que lo que sigue provocará un salto a cualquier rutina apuntada por VECTOR:

```
CIRCLE x,y,z,n
LINE x
BORDER x,y,z
```

Si usted opta por este camino, CHADD apuntará al indicativo (*token*) de la palabra clave irregular en el momento de entrar la rutina apuntada por VECTOR. Esto puede comprobarse con el código del *token* adecuado, que puede encontrarse en el manual del Spectrum.

A pesar de la complejidad aparente del procedimiento, la adición de nuevas palabras clave al BASIC del Spectrum es de hecho bastante fácil. Lo que importa es recordar qué ROM está paginada en cada momento. Si ha de abandonar la ROM sombra, podrá emplear el código de enganche 50, más la dirección adecuada, para llamar la rutina de la ROM sombra.

Identificación

Si trata de acceder directamente a las rutinas de la ROM sombra, le interesa comprobar antes cuál es la versión de la ROM de la Interface 1 que está usando. Por regla general, las IF1 con números de serie superiores a 87315 llevan la versión 2 de la ROM, pero una manera más segura de comprobarlo es entrar esto:

```
CLOSE #0; PRINT PEEK 23729
```

Que dará 0 tratándose de la versión 1 y 80 en la versión 2. Por suerte, las llamadas de la ROM de la IF1 que hemos utilizado para ampliar el BASIC se encuentran en la misma dirección tanto en una como en otra versión. Aun así aparecerán problemas cuando emplee otras rutinas. En caso de duda, y suponiendo que posea un buen desensamblador, puede que prefiera comprobar las rutinas que usted llama empleando el programa *Cargador de la ROM sombra* que ya hemos publicado. Es de notar la adición de dos nuevos códigos de enganche en la ROM de la versión 2. El código de enganche #33 recupera el descriptor de registros de un archivo en el microdrive, y #34 abre un canal «b» RS232. En este último caso, la dirección de base del canal se proporciona en DE

Direcciones útiles

En la ROM sombra:

#01F0	Abandona la ROM sombra a través de una rutina de error de la ROM principal
#05B7	Efectúa un retorno en tiempo de ejecución si el car. contenido en el reg. A es #0D (ENTER) o un punto y coma. De lo contrario genera un error
#05C1	Retorna al intérprete principal cuando ha sido aceptada una nueva instr.

En la ROM principal:

#0018	GETCHAR: Toma CHADD (ver abajo) en HL; toma el car. al que apunta CHADD en A; mira si es un car. imprimible y hace un retorno si lo es. De lo contrario pasa a NEXTCHAR
#0020	NEXTCHAR: Incrementa CHADD y lo coloca en HL. Toma el carácter apuntado por CHADD y comprueba si es imprimible. Si lo es, entonces NEXTCHAR retorna con el carácter an A. Si no lo es, se repite el proceso
#1C82	Evalúa o chequea una expresión numérica, cuyo primer elemento debe ser el apuntado por CHADD. A la salida, CHADD apunta al primer carácter que sigue a la expresión, y el resultado se coloca en la parte superior de la pila calculadora de la ROM principal
#1E94	Toma el valor de la parte superior de la pila de cálculo de la ROM principal y lo almacena en A si está dentro del intervalo de 0 a 255. De lo contrario, genera un error

La variable de sistema CHADD apunta al carácter que se está interpretando, y se sitúa en las direcciones 23755 y 23756

En pos del triunfo

Los juegos de estrategia exigen conjugar numerosos factores y adoptar juiciosas decisiones. «Football manager» constituye un buen ejemplo

Lanzado al mercado a principios de los ochenta, el programa *Football manager* (Entrenador de fútbol) ha generado constantes ventas a largo plazo. En un mercado en el cual la mayoría de los paquetes permanecen en las estanterías de las tiendas sólo durante un período limitado, independientemente del éxito que obtengan al editarse, este juego destaca entre sus competidores. Y, además, no tiene nada particularmente innovador: el concepto del juego es muy simple. Usted dirige un equipo de fútbol y debe conducirlo al éxito a lo largo de la Liga.

El juego comienza, como es natural, al iniciarse una nueva temporada, con el equipo que usted dirige en la Cuarta División. Luego se le presenta un menú de opciones tales como comprobar su posición en la tabla de la Liga, ver detalles de la lista de encuentros y considerar las características de los jugadores de que dispone para integrar el equipo.

Los jugadores se dividen en tres categorías: defensas, centrocampistas y delanteros. Como es obvio, un entrenador sensato formará un equipo equilibrado para cada encuentro. Antes de un partido se le ofrece a usted la opción de introducir modificaciones en la alineación, con el fin de sacar al campo el equipo más adecuado. No obstante, en las primeras etapas del juego sólo dispone de 12 jugadores, por lo cual no habrá mucho donde escoger. Sin embargo, a medida que el juego avanza, aparecen jugadores adicionales provenientes del mercado de traspasos y puede "pujar" por ellos.

Un jugador posee tres características básicas, que se deben tener en cuenta al pujar. En primer lugar, cada uno de ellos posee un "factor de destreza", medido en la escala entre uno y cinco. Segundo, cada uno tiene un "valor" que, aunque relacionado con el factor de destreza (5 000 por cada punto de destreza), puede variar en el mercado de traspasos. Por último, está la "energía" del deportista. Ésta se halla comprendida entre 1 y 20 y disminuye cada vez que un jugador es alineado para un partido. Por lo tanto, no es aconsejable alinear a la totalidad de los mejores jugadores en todos los encuentros, sino que es mejor ir rotándolos de modo que puedan disfrutar de descansos periódicos para poder restablecer sus energías hasta el máximo. Por supuesto, esto es muy difícil con una plantilla de apenas doce jugadores, y, al enfrentarse a un factor de energía en disminución, aun el entrenador más cuidadoso se verá obligado a recurrir tarde o temprano al mercado de traspasos.

Al comienzo de cada "vuelta" se ofrecerá el traspaso de un jugador y se le invitará a pujar por él. Si se acepta su oferta, el deportista pasará a formar parte de la plantilla. Sin embargo, con frecuencia la oferta se rechazará, aun cuando ofrezca más del valor establecido. Entonces se le pedirá que haga otra oferta, sólo que esta vez el valor del traspaso del jugador aumentará debido a que usted se muestra muy interesado por él.

Tras terminar sus incursiones en el mercado de traspasos, acordando préstamos si fuera necesario, y después de examinar su posición actual en la Liga, se le dará la opción de jugar un partido. El ordenador visualiza las fortalezas relativas de su equipo y las del adversario, y usted tiene la opción de cambiar el equipo, sustituyendo los jugadores cansados y lesionados por los de refresco. Tras haber completado esto, el ordenador presentará las "jugadas más interesantes" del encuentro.

Las jugadas más interesantes consisten simplemente en las escaramuzas producidas en la zona de portería de ambos bandos; el tiempo que se dedique a una u otra portería dependerá de las fortalezas relativas de los centrocampistas, mientras que las fortalezas de los delanteros y de los defensores adversarios determinarán la consecución o no de los goles. Aunque los gráficos del partido le proporcionan al juego mucha emoción mientras se contemplan las incidencias del mismo, hay que decir que la visualización de gráficos es muy primitiva; la versión para el BBC, por ejemplo, se compone simplemente de caracteres gráficos en bloque.

Al terminar el encuentro se visualiza el marcador final, junto con los resultados de los otros partidos.

El juego requiere que el entrenador no sólo considere la mejor estrategia para el próximo encuentro, sino que también prevea los partidos venideros, tal como las fechas en las que hay de medirse con equipos que ocupen las primeras posiciones de la tabla y los encuentros coperos, en los que se requiere el mejor equipo posible. Por lo tanto, es necesario tomar decisiones acerca de concederles descanso a los mejores jugadores ahora (y arriesgarse a perder puntos) o hacerlos jugar (corriendo el riesgo de que se lesionen o pierdan energía, lo que incidiría negativamente en su rendimiento en los partidos importantes).

A pesar de que el juego no contiene secuencias "recreativas" propiamente dichas, es fácil comprender las razones por las que ha conservado su popularidad. A pesar de algunas limitaciones, es una simulación suficientemente realista como para lograr que el jugador crea realmente que está dirigiendo un equipo. Y cuando finalmente su equipo gane el campeonato de Primera División, usted se sentirá realmente satisfecho.

Football manager: Para el ZX81, Sinclair Spectrum, Commodore 64, Vic-20, BBC Micro, Electron, Amstrad y Dragon
Editado por: Addictive Games Ltd, 7A Richmond Hill, Bournemouth, Dorset BH2 6HE, Gran Bretaña
Autor: Kevin Toms
Formato: Cassette
Palancas de mando: No se necesitan

Liz Heaney

Impaciente y nerviosa espera Esta pantalla, tomada de la versión de *Football manager* para el Commodore 64, muestra la "secuencia de acciones" del juego. En este punto, el jugador no tiene control alguno sobre los incidentes que se producen en el campo de juego, sino que se limita a contemplar "las jugadas más interesantes" del encuentro. Ésta quizá sea la secuencia del juego más llena de nerviosismo y emoción, puesto que usted permanece a la espera de ver si sus evaluaciones han sido correctas

Futuro luminoso

Acerquémonos a la informática óptica, tecnología que probablemente hará parecer lentas las más veloces máquinas de hoy en día

Fibra añadida
El equivalente óptico del cable eléctrico es el cable de fibra óptica. Este cable permite «doblar» las señales de luz y, en consecuencia, dirigirlas hacia donde deseemos. Las fibras de que está relleno el cable contienen una sustancia transparente que «conduce» la luz en la dirección de la fibra

Science Photo Library Ltd.

Cuando se piensa en un ordenador, casi con seguridad se imagina un dispositivo enchufable, basado en interruptores binarios electrónicos que almacenan y procesan la información. Sin embargo, usted estaría equivocado si pensara que éste es el único tipo viable de ordenador. Se podría concebir un ordenador basado en cualquier método para el almacenamiento de la información de forma física: una carga eléctrica, los dientes y las muescas de las ruedas de engranaje, orificios en tiras de papel, cuentas, piedrecillas, conchas marinas o cualquier otro medio.

El semiconductor electrónico es el mejor de los tipos de almacenamiento de información que se ha inventado hasta ahora, porque su consumo de potencia es ínfimo, su tamaño es microscópico y, lo más importante, es rápido. La velocidad a la que puede operar un ordenador está limitada por el tiempo que requieren sus almacenamientos de información binarios, o *interruptores*, para pasar de una posición a otra, y por la velocidad de la transmisión de información entre interruptores. Mientras que los circuitos de ordenador basados en corrientes de electrones son muy veloces para realizar estas funciones, teórica y técnicamente es posible construir circuitos aún más rápidos, basados en haces de luz.

La investigación en el campo de la informática óptica y disciplinas afines, como la *lógica fotónica*, ya se ha puesto en marcha. Muchos de los componentes que se incorporarían en un ordenador óptico ya están disponibles y en uso. Las fibras ópticas, por ejemplo, se están utilizando para transportar señales telefónicas y de televisión. Se han desarrollado guías de onda ópticas, equivalentes a las pistas metálicas de las placas de circuito impreso. También se han desarrollado discos ópticos. El problema más crucial de todos, la analogía óptica del interruptor binario electrónico, está actualmente en fase de prueba.

Los diseños para un interruptor binario óptico se agrupan en dos tipos principales: los interruptores electroópticos y los interruptores totalmente ópticos. Es interesante analizar cada uno de los dos tipos, empezando por el híbrido electroóptico, puesto que los componentes ópticos probablemente se utilizarán primero en conjunción con componentes electrónicos, en lugar de entrar a competir con ellos. Ambos tipos de interruptores explotan el fenómeno de la interferencia de luz provocada por cambios controlados del índice de refracción de un material óptico no lineal, por lo general un compuesto de galio o litio. El índice de refracción de un material expresa la relación de la velocidad de la luz en el vacío con su velocidad en el material. Por ejemplo, si el índice de refracción de un tipo deter-

El programa de luz
Otro importante componente que será necesario para hacer realidad el ordenador óptico es el dispositivo semiconductor, que se puede regular para que emita una gama de frecuencias de luz pura. Éste será vital a los fines de interferencias en la comunicación, donde se requiere luz a una frecuencia y longitud de onda estándares

Science Photo Library Ltd.

minado de vidrio es 1,5, luego, como la luz viaja a alrededor de 300 000 km/s en el vacío, su velocidad será de alrededor de 200 000 km/s en el vidrio.

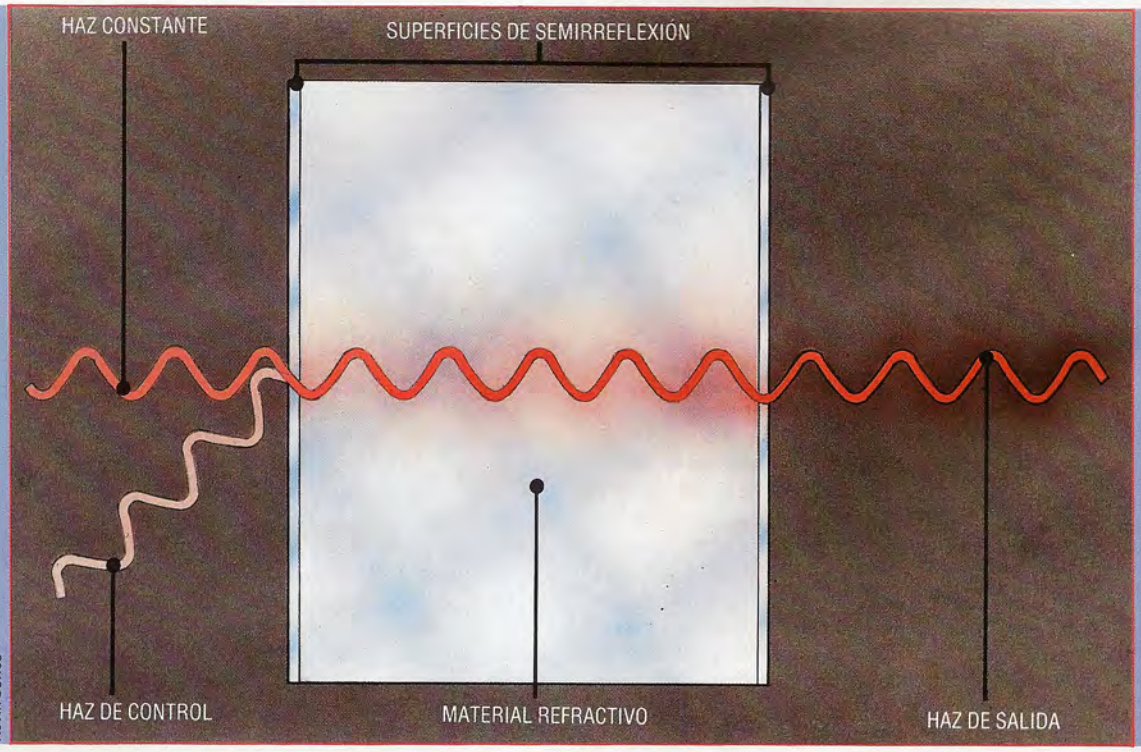
El tipo de interruptor binario híbrido electroóptico se basa en un invento denominado *interferómetro Mach Zehnder*. El principio de este dispositivo es muy simple. Se divide en dos un haz de luz entrante, se le conduce a través de canales separados de material no lineal y después se lo vuelve a recombinar. Si no se aplica ninguna corriente eléctrica a ninguno de los canales del interferómetro, los haces separados llegan a la salida en fase y generan un haz intenso (el estado *on*). Si se altera el índice de refracción de alguno de los canales mediante una señal eléctrica, los dos haces ya no se recombinan en fase, puesto que uno de los dos haces viajará más rápidamente que el otro, y el resultado es un haz de salida mucho más débil (el estado *off*).

Este tipo de interruptor podría conformar la base de un circuito integrado óptico sencillo. El circuito sería más veloz que un equivalente electrónico debido a la capacidad de la luz de transportar la información a mayor velocidad que las señales eléctricas (alrededor de 10 000 veces más rápido, debido a la mayor frecuencia de las ondas de luz). A pesar de este rendimiento superior, la velocidad de este circuito híbrido aún es limitada, ya que para la conmutación se basa en la realimentación eléctrica. Mediante el empleo de luz para operar el interruptor (utilizando un haz más débil para controlar uno más intenso) se pueden obtener velocidades superiores.

Este tipo de diseño es, por supuesto, exactamente análogo al transistor electrónico. Un equipo de la Heriot Watt University, de Edimburgo, ya ha construido una versión prototipo de interruptor totalmente óptico, a la que se ha denominado *transfasor*. El dispositivo realiza la conmutación en alrededor de un picosegundo (una milbillonésima de segundo). El transistor electrónico más veloz, por su parte, posee una velocidad de conmutación de un nanosegundo (una billonésima de segundo). Esto significa que un ordenador totalmente óptico podría operar potencialmente a una velocidad mil veces superior a la de un ordenador electrónico.

El transfasor desarrollado en Heriot Watt se basa en el diseño de un interferómetro inventado por los físicos franceses Charles Fabry y Alfred Perot en 1896. Entre dos superficies reflectantes, se intercala una fina lámina de material no lineal. La luz que entra por una superficie rebota repetidamente dentro de la capa intercalada antes de salir a través de la superficie opuesta. Las ondas de luz atrapada interaccionan de una forma que está de-

En el haz
El bloque de construcción básico del ordenador electrónico es el transistor, cuyo equivalente óptico se denomina *transfasor*. Por uno de los extremos se introduce un haz constante junto con un «haz de control», que se interferirán entre sí en el interior del dispositivo. Una pequeña variación en la intensidad del haz de control producirá un gran cambio en la intensidad de la luz que salga. De este modo el transfasor se puede utilizar como un dispositivo de conmutación lógico



Kevin Jones

terminada por el índice de refracción del material no lineal, que se puede regular utilizando un haz de control. Si el índice de refracción es tal que las ondas de luz están en fase y se refuerzan entre sí, entonces se transmite un haz de luz intenso. Si las ondas están desfasadas, sólo se transmite un haz de luz débil. Los dos estados de transmisión alternativos corresponden a las posiciones *on* (encendido) y *off* (apagado).

Beneficios teóricos

El éxito del ordenador óptico, en definitiva, estará determinado por los límites teóricos y técnicos de las máquinas convencionales. Sobre el papel, los ordenadores ópticos son superiores a sus homónimos electrónicos en numerosos sentidos. El primero de ellos es la velocidad de conmutación. Para que un transistor actúe como interruptor, una corriente de electrones debe atravesar su base. La velocidad a la cual pueden desplazarse los electrones en un semiconductor tiene límites, así como el espesor con que se puede construir un transistor (por supuesto, cuanto más fina sea la base, tanto más corto será el viaje de los electrones). Un ordenador óptico, transmitiendo a la velocidad de la luz, es intrínsecamente mucho más veloz.

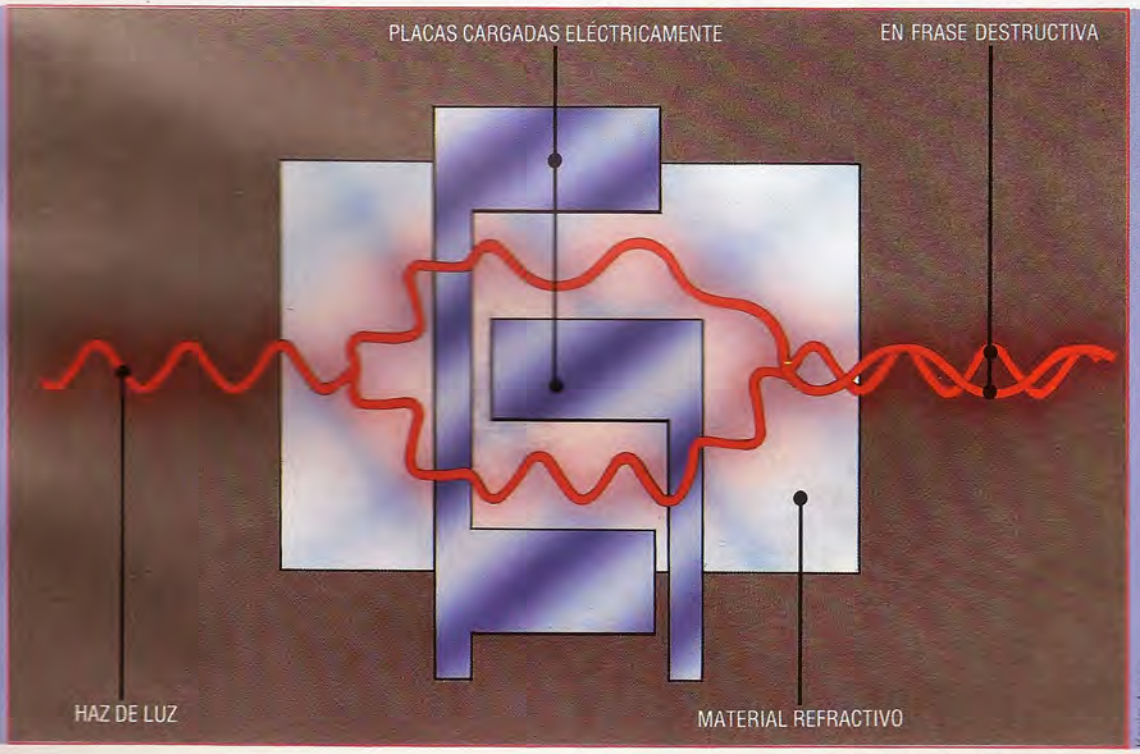
Las máquinas convencionales también están limitadas por lo que se denomina *el cuello de botella de Von Neumann*. Ésta es una característica básica de casi todos los ordenadores y está relacionada con la forma en que se accede a la información desde la memoria a razón de una palabra por vez (estilo «en serie»). En un ordenador óptico, se podría acceder a la memoria en paralelo. Ello se debe a que no es necesario aislar los haces de luz igual que las señales eléctricas. Por lo tanto, se podrían canalizar haces separados a través del mismo transistor óptico y conmutarlos simultáneamente.

La tercera limitación del diseño del ordenador convencional es el limitado espacio disponible para realizar conexiones en un chip plano. Por lo general, la cantidad de conexiones que se pueden efectuar en un solo chip está severamente limitada por el problema de ubicación de patillas, un límite sobre la cantidad de patillas que se pueden montar sobre un trozo plano de silicio. Aplicando técnicas holográficas (ópticas) tridimensionales, en principio sería posible introducir un grado de libertad adicional para la disposición de las conexiones, y cambiar instantáneamente su configuración.

Por último, están los beneficios prácticos de la interconexión entre los ordenadores ópticos y los periféricos. Ya se están utilizando redes de fibras ópticas para enlazar dispositivos electrónicos. Son sumamente rápidas y en la misma fibra óptica se pueden transmitir mensajes de luz sin ninguna interferencia. Los ordenadores ópticos se podrían conectar entre sí mediante estas redes sin necesidad de ninguna interface optoeléctrica lenta.

A la vista de estas ventajas, ¿cuándo podemos esperar que comiencen a aparecer ordenadores ópticos prácticos y asequibles? La respuesta es: no aparecerán por ahora. Los beneficios teóricos de la óptica en comparación con la electrónica no son los mismos en la práctica. Los impresionantes avances en la miniaturización del chip significan que, al menos a corto plazo, los ordenadores ópticos representan un objetivo que retrocede constantemente. La cantidad de dispositivos de transistor que se pueden empaquetar en un chip continúa duplicándose cada dieciocho meses. Para el año 2000, las predicciones hablan de chips con 10 o 100 millones de dispositivos. La integración a gran escala (LSI: *large scale integration*) ha pasado a ser una integración a muy grande (y ultra) escala. El «procesador en paralelo» se ha convertido en el «procesador paralelo en masa».

La última tecnología electrónica de gran velocidad está llamada a seguir perfeccionándose y, con esta clase de competencia, puede ser que el ordenador óptico asequible se halle todavía a años luz.



La fase
El interferómetro Mach Zehnder posibilita el trabajo conjunto de dispositivos electrónicos y ópticos. La luz que entra por un extremo se divide en dos haces. Una vez que los haces han entrado en el dispositivo (regulando el índice de refracción para uno de los haces), la velocidad de la luz se reducirá y quedará «desfasada» con relación al otro haz. Cuando los dos haces converjan en el haz de salida, las fases se interferirán destructivamente y el resultado será off, que se puede considerar como un cero lógico

Kevin Jones

Criba de números

Las habilidades aritméticas del FORTH se ven realzadas en el programa «La criba de Eratóstenes»

Una de las características del FORTH que puede resultar enigmática al principio es su restricción a la aritmética de enteros. Esto es algo que no viene dado por razones de interactividad y ampliabilidad, sino por razones de eficiencia: la aritmética de enteros es mucho más veloz que la de punto flotante.

En FORTH, la forma habitual de manipular fracciones es con aritmética de *punto fijo*, en la que se emplean números que se multiplican o dividen por alguna constante, por lo general una potencia de diez. Esto es como usar milímetros cuando los metros no son suficientemente exactos.

Un número de la pila del FORTH posee 16 bits, y, por lo tanto, puede considerarse como un entero *con signo* (indicándose el signo mediante el bit más significativo) entre −32768 y −32767, o un entero *sin signo* entre 0 y 65535. La forma más rápida de ver esta ambigüedad es digitando:

```
65535.
```

Esto visualiza la respuesta −1, porque el . trata a la parte superior de la pila con un número con signo, y el número sin signo 65535 y el número con signo −1 se representan mediante la misma entrada en la pila. La palabra del FORTH U. es igual que ., con la excepción de que trata a la parte superior de la pila como si no tuviera signo. De modo que:

```
65535 U.
```

visualiza 65535.

Existen algunas otras palabras en diferentes versiones, según esperen que los números de la pila tengan o no signo. La elección reviste importancia si existe la posibilidad de obtener números entre 32768 y 65535. Hay que decidir el empleo de las palabras, ya sea para números con signo, en cuyo caso estos números grandes se cuentan como desbordamiento, o bien para números sin signo, en cuyo caso no se permiten números negativos. Un buen ejemplo es cuando se emplean números como direcciones de memoria (que es como funcionan @ y !). Éstos no tienen signo, de modo que habrá que utilizar palabras sin signo, como U< en lugar de <. Asimismo, pueden utilizarse números de pila como patrones de 16 bits y realizar las combinaciones booleanas entre dos bits habituales.

El ejemplo principal muestra un programa que utiliza la «criba de Eratóstenes» para visualizar todos los primos hasta el 65536. Como es común en los programas en FORTH, resultará más fácil leerlo hacia atrás: empezar por la palabra principal, PRIMOS, e ir hacia atrás para hallar las definiciones de las palabras empleadas. PRIMOS es la palabra a escribir primero, pero deberá digitarse al final, después de sus subpalabras. Para el programa *La criba de Eratóstenes*, se van escribiendo todos los números hasta un límite especificado. Al encontrar un primo, se tachan todos sus múltiplos, que obviamente no pueden ser también primos, y el siguiente primo es el siguiente número que no se ha tachado. Ignorando el 1 (que, por diversas razones, no se considera un auténtico primo), se encuentra el primer primo, 2, y se tachan todos sus múltiplos. El siguiente número que no se ha tachado, el 3, es el siguiente primo, de modo que se tachan todos sus múltiplos y así sucesivamente. Se hace muy fácil con lápiz y papel hasta hallar todos los primos hasta 20.

El programa en FORTH listado aquí almacena ocho Kbytes o 65536 bits, uno para cada número hasta 65536. Al comienzo todos ellos están en 0, pero al tachar uno, su bit se cambia a 1. El espacio de memoria para estos bytes se prepara mediante:

```
CREATE BITS 8192 ALLOT
```

(En realidad hemos reemplazado 8192 por una constante, BYTES.) Más adelante veremos con más precisión cómo trabaja esto, pero por ahora su efecto consiste en asignar un bloque de 8192 bytes a algún lugar de la memoria y crear una nueva palabra, BITS, cuya acción consiste en apilar la dirección del primero de estos bytes. Se puede utilizar, por lo tanto, para que contenga una matriz de bytes.

Para referirse a uno de los bits, se necesitan dos números: uno que muestre qué byte contiene el bit (de modo que para este número pueda utilizarse la dirección de memoria del byte) y otro que muestre qué bit se halla dentro de ese byte, de modo que estará entre 0 y 7. No es demasiado difícil deducir de este par de números el número (entre 1 y 65536) que representa el bit.

Recuérdese que CURBYTES es una variable; su valor, CURBYTES@, es una dirección de un byte de la matriz BITS, y CURBYTES @ C @ es el valor del byte. Si se está familiarizado con el lenguaje c, esta idea no resultará nada extraña. Dado que los primos y las direcciones no tienen signo y posiblemente sean bastante grandes, ocasionalmente se necesitarán palabras sin signo como U. y U<.

He aquí una palabra útil:

```
+! (n, dirección −)
```

Ésta sustituye el número de la dirección dada por el mismo número con n sumado, de modo que típicamente se utiliza para incrementar el valor de una variable con una cantidad dada.

Un punto final acerca de la aritmética es que algunas palabras son para aritmética de *doble longitud*. Esto significa que se utilizan juntos dos números de dos bytes de la pila para formar un único número de cuatro bytes. La mitad más significativa es la más próxima a la parte superior de la pila, y la mitad menos significativa la de abajo.

Bases numéricas

En FORTH es muy fácil cambiar de base numérica, de modo que los números se impriman o se lean desde el teclado como binarios, octales, decimales, hexadecimales o en cualquier otra base numérica que se escoja. Esto se consigue cambiando la variable BASE. P. ej.:

```
54 16 BASE !. DECIMAL
```

Esto visualiza 36 (54 en hexa), vuelve a cambiar la base numérica a diez con la palabra DECIMAL.

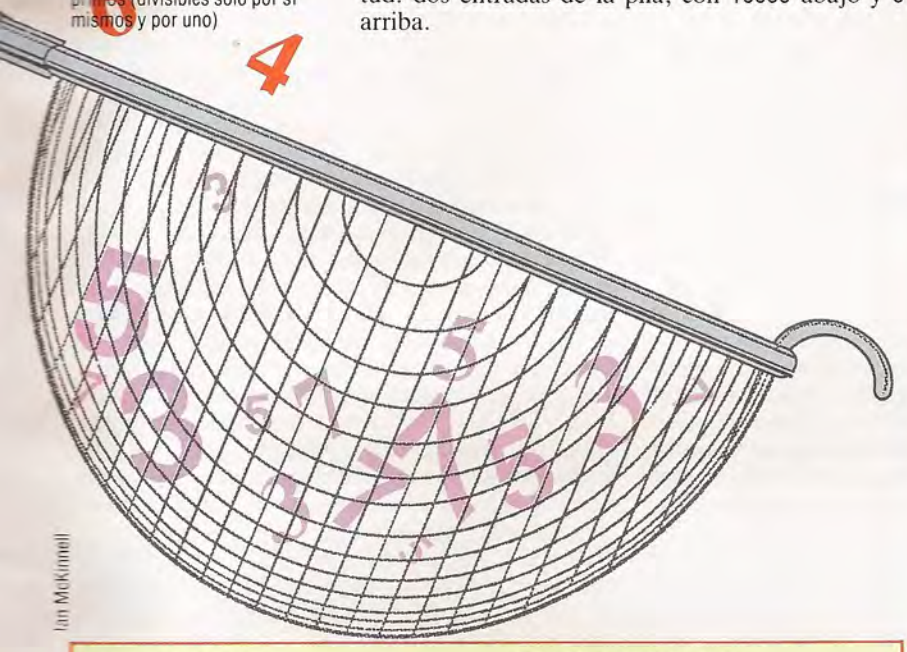
En la práctica resultará más sencillo definir palabras como

```
HEX 16 BASE !
```

Obsérvese que una vez que un número (como 16 aquí) forma parte de la definición de una palabra, no se ve afectado por posteriores cambios de la base numérica

Selección de primos
La criba de Eratóstenes, así llamada en honor del matemático griego que la formuló, es un método simple de hallar números primos. El método trabaja «tamizando» múltiplos de números sucesivos hasta que sólo quedan números primos (divisibles sólo por sí mismos y por uno)

Aunque el estándar FORTH-83 sólo permite digitar números de longitud simple, en la mayoría de las implementaciones, si un número contiene un punto (en cualquier lugar dentro del número), se ignora (no se trata como un punto decimal) y el número se coloca en la pila como uno de doble longitud. Por ejemplo, 10.000 es 10000 de doble longitud: dos entradas de la pila, con 10000 abajo y 0 arriba.



Ian McKimmell

```
8192 CONSTANT BYTES
CREATE BITS BYTES ALLOT BITS BYTES
- CONSTANT BITSEND
( 1 bit para cada número 1 - 65536 )
VARIABLE CURBYTE VARIABLE CURBIT
( mostrar número que se está examinando.
CURBYTE )
( señala byte de BITS, CURBIT a bit )
VARIABLE MASK ( 2**CURBIT )
VARIABLE PRIMO VARIABLE PRIMO/8
VARIABLE PRIMOMOD8
( mostrar primo una vez hallado )
: SETUP ( -- ) ( inicializa variables )
  BYTES 0 DO ( BITS de matriz cero )
    0 BITS I + C!
  LOOP
  BITS CURBYTE ! 0 CURBIT ! 1 MASK! ;

: BITSIG ( -- 0 o -1 - cero )
( regula CURBYTE etc. para hallar siguiente bit )
( y lo coloca en la pila )
  1 CURBIT +!
  MASK @ 2 * MASK !
  CURBIT @ 8 = IF
    0 CURBIT !
    1 MASK !
    1 CURBYTE +!
  THEN
  CURBYTE @ C@ MASK @ AND ;

: PRIMOETC ( -- ) ( regula PRIMO,
  PRIMO/8 )
( y PRIMOMOD8 )
  CURBIT @ 1 +
  DUP 8 = IF
    DROP 0 PRIMOMOD8 ! 1
  ELSE
    PRIMOMOD8 ! 0
  THEN
  CURBYTE @ BITS - + PRIMO/8 !
  PRIMO/8 @ 8 * PRIMOMOD8 @ + PRIMO ! ;

: OTRO-PRIMO ( -- 0 o -1 )
( regula CURBYTE etc. al siguiente primo,
  establece )
( PRIMO etc. deja 0 si se ha llegado al final de
  BITS )
  BEGIN
  BITSIG 0= UNTIL
  CURBYTE @ BITSEND U< IF
    PRIMOETC
    -1
  ELSE
    0
  THEN ;

VARIABLE DELBYTE VARIABLE DELBIT
: DELMASK ( -- 2 ** DELBIT )
  1
  DELBIT @ IF ( si DELBIT no es cero )
    DELBIT @ 0 DO ( multiplicar 1 por 2 )
      2* ( 2 veces DELBIT )
    LOOP
  THEN ;

: OTRO-MULTIPLO ( -- 0 ó -1 )
( regula DELBYTE y DELBIT )
( al siguiente bit si establecido )
( Queda 0 si se ha llegado al fin de la matriz BITS )
  PRIMOMOD8 @ DELBIT +!
  PRIMO/8 @ DELBYTE +!
  DELBIT @ 7 ) IF
    -8 DELBIT +!
    1 DELBYTE +!
  THEN
  DELBYTE @ BITSEND U< BITS
  1- DELBYTE @ U< AND
( asegurarse en caso de que DELBYTE pase de
  65535 ) ;

: BORRAR-MULTIPLOS ( -- )
( prepara bits para ni múltiplos de primo actual )
  CURBYTE @ DELBYTE !
  CURBIT @ DELBIT !
  BEGIN
  OTRO-MULTIPLO WHILE
    DELBYTE @ C@ DELMASK OR DELBYTE @ C!
    REPEAT ;

: PRIMOS ( -- )
( imprime todos los primos entre 2 y 65535 )
  SETUP
  BEGIN
  OTRO-PRIMO WHILE
    PRIMO@ U.
    BORRAR-MULTIPLOS
  REPEAT ;
```

Palabras aritméticas

Suma y resta: no se hace distinción entre números con signo y sin signo. + y − son de longitud simple, D+ y D− de longitud doble.

El menos unario (un solo número negado): NEGATE para longitud simple, DNEGATE para doble longitud. El figFORTH utiliza MINUS y DMINUS.

Multiplicación: * es de longitud simple, con o sin signo. UM * es de longitud «mixta» sin signo: toma de la pila dos números sin signo de longitud simple y deja su producto en doble longitud. En los FORTH más antiguos UM* se denomina U*, y pueden tener un producto M*, de longitud mixta con signo.

División: /, MOD y /MOD son de longitud simple con signo. Dejan el cociente, el resto y ambos (el cociente arriba). UM/MOD es una versión de longitud mixta sin signo de /MOD. Su dividendo (el número que se divide) es de doble longitud. En los FORTH más antiguos se denomina U/ o U/MOD.

Entre el FORTH-83 y los FORTH más antiguos existe una diferencia en la forma de dividir números negativos. Pensando en hacer la división exactamente y después redondear a algún entero, el FORTH-83 redondea en un entero que no sea mayor; por ejemplo, 10/3 se redondea a 3, pero −10/3 se redondea a −4. Los FORTH más antiguos redondean hacia el 0, de modo que −10/3 se redondea a −3. Una vez calculado el cociente en función de estas reglas, el resto se determina mediante la fórmula:

dividendo = divisor*cociente+resto

Multiplicación combinada con división: */ es de longitud simple con signo, teniendo el siguiente efecto sobre la pila:

n1,n2,n3 -- n1* n2/n3

Comparaciones: = <> y U< son de longitud simple. < y > son con signo, U< es sin signo, = cualquiera. D= D< y DU < son de doble longitud, D< es con signo, DU< sin signo y D= cualquiera. MAX y MIN (que dan el mayor o menor de dos números) son de longitud simple con signo. DMAX y DMIN son de doble longitud con signo.

Valor absoluto: ABS y DABS son de longitud simple y doble.

Operaciones booleanas entre dos bits: AND, OR, XOR y NOT son todos de longitud simple. El signo no tiene importancia.

Combinaciones estándares: algunas palabras se utilizan juntas tan habitualmente, que sus combinaciones sin un espacio se definen como una única palabra. Por ejemplo, 1+ tiene exactamente el mismo efecto que 1+ (aunque se podría hacer con mayor eficiencia).

Impresión: . y U. son de longitud simple, con signo y sin signo, respectivamente. D. es de doble longitud con signo.

Direccionamiento de memoria: una dirección, de longitud simple sin signo, puede referirse a los dos bytes que haya en esa dirección (como con @ y !), a un solo byte de éstos (para lo cual se utiliza C@ y C!, donde la C es de «carácter») o al número de longitud doble de cuatro bytes que haya allí (use 2@ y 2!)

Quién, dónde y cómo

David Higham

Comienza el juego
Vemos las posiciones iniciales de los distintos personajes según lo que establecen los valores por defecto en las líneas 6030 y 6040. También se aprecian los objetos, cuyos datos están retenidos en las líneas 6140 y 6150. Comienza el juego en la sala de tertulia

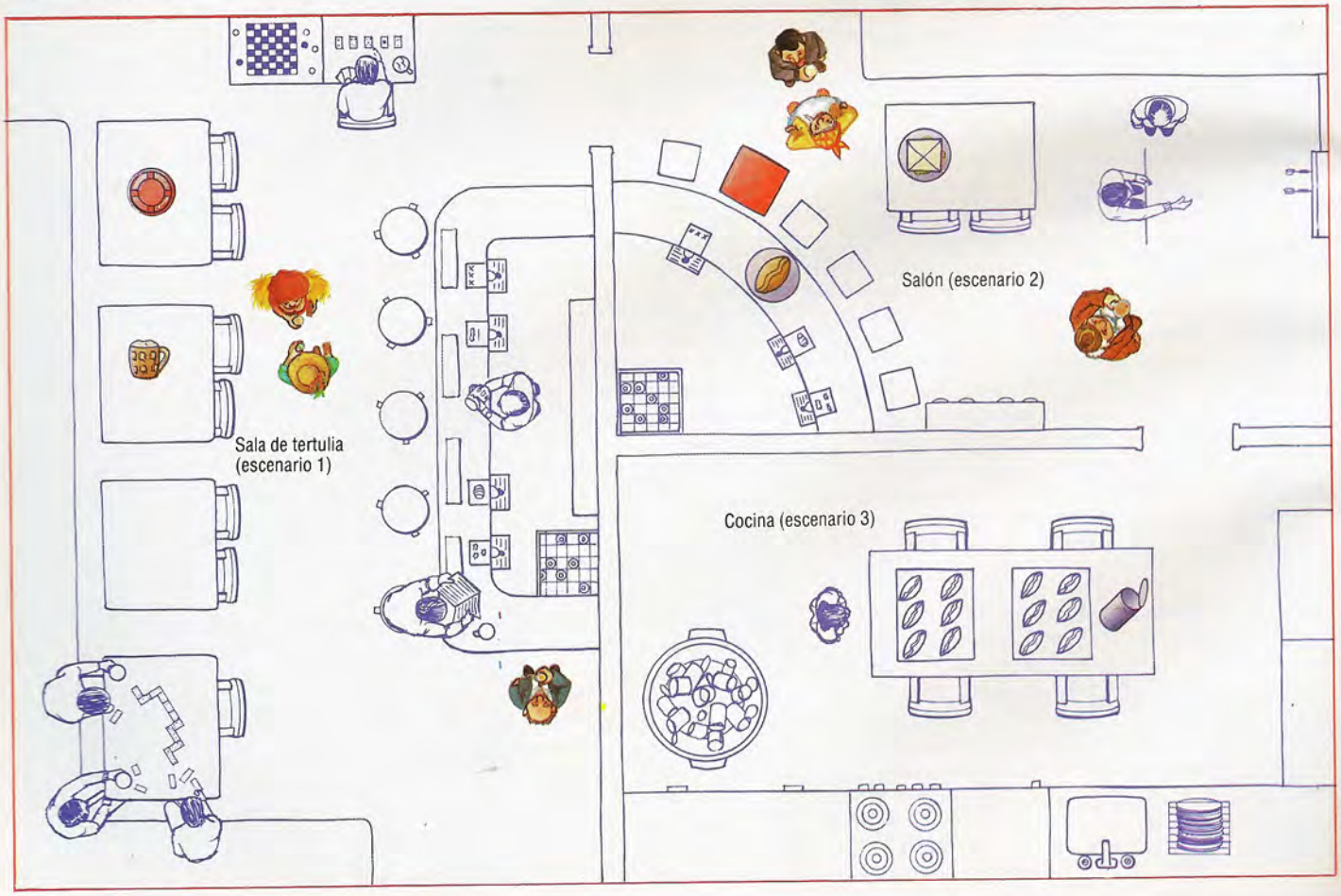
Ya estamos en condiciones de darles algún sentido a las vidas de nuestros personajes

El primer módulo de nuestro programa comienza en la línea 10, inicializando la pantalla y DIMensionando las matrices que analizamos en el capítulo anterior. Luego lee los datos necesarios de las sentencias DATA correspondientes, todos los cuales están almacenados en el módulo DATA que empieza en la línea 6000. A estos dos módulos, por supuesto, se les irán añadiendo más líneas a medida que vayamos agregando otras rutinas e implementando el manipulador de personajes propiamente dicho. Una de las ventajas de adoptar este enfoque modular es que podremos ejecutar (RUN) y probar esta sección del programa apenas la hayamos digitado, y después ir probando cada una de las rutinas individualmente, tal como las vayamos entrando más adelante.

Obsérvese el aviso «Valores por defecto» de la línea 70. Éste, al ejecutar (RUN) el programa, nos permitirá entrar nuestros propios valores para los atributos de los distintos personajes en lugar de los valores por defecto entrados en las sentencias DATA. Esta facilidad de «confección a medida» es muy importante cuando se diseñan personajes interactivos con más de dos o tres atributos, porque permite «ir afinando» los personajes durante el juego. Por ejemplo, quizá encuentre que Mari Tapas es demasiado temperamental, o tal vez que pasa muy rápidamente de un escenario a otro.

La línea 530 es simplemente un bucle de comprobación que llamará, de a una, a cada una de las rutinas principales entradas hasta el momento. La rutina de entrada de la línea 2030 aguarda que se pulse una tecla. Se ignoran todas las teclas que no sean 0, 1, 2 o 3 (valores ASCII del 48 al 51) y se devuelve el control al bucle del programa de la línea 530. Pulsando 1, 2 o 3 el jugador penetra en el escenario especificado, donde podrá ver qué está sucediendo. Usted podría, por supuesto, diseñar una rutina de entrada que le permitiera dirigirse a los personajes directamente, pero en nuestro programa de ejemplo toda la interacción personaje/jugador la llevará a cabo el manipulador de personajes, debido a que hemos incluido al jugador como si se tratara de un personaje extra.

En caso de que usted decidiera adaptar los métodos que estamos analizando aquí para incluirlos en su propio juego de aventuras, recuerde que el manipulador de personajes puede existir independientemente de la rutina de entrada, como veremos más adelante cuando examinemos cómo se podría



Kevin Jones

añadir una rutina manipuladora simple a nuestro programa *El bosque encantado*.

Pulsando la tecla 0 en cualquier momento, se entra en la rutina «Valores por defecto» de la línea 2300, que permite editar los atributos de los personajes como ya hemos mencionado y descrito.

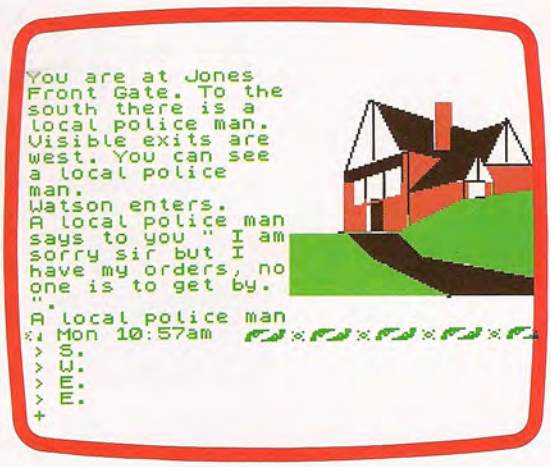
Las líneas 2070-2390 proporcionan las tres subrutinas de visualización esenciales. La primera, de la línea 2100, imprime la descripción del escenario actual. Cuando ejecute el juego por primera vez, ésta será la descripción de la sala de tertulia, escenario número 1, como se establece en la línea 60 del módulo de inicialización.

La rutina de la línea 2150, «Imprimir objetos visibles», realiza un bucle a través de la matriz de objetos b$ y comprueba cada entrada de escenario para ver si corresponde con el escenario actual del jugador (r). Luego se imprimen en la pantalla las descripciones de objetos apropiadas; de lo contrario, se le dice al jugador «Ves: nada». La rutina «Imprimir personajes visibles» opera de forma similar, comprobando el elemento correspondiente de la matriz c$ e imprimiendo el nombre de un personaje si la persona en cuestión está presente.

La subrutina «Inicializar personajes» se llama cada vez que se pulsa 0 durante el tiempo de ejecución, o si durante la secuencia de inicialización se entra algo distinto a s o S en respuesta al aviso «Valores por defecto». Esta rutina utiliza la matriz d$(11) y el nombre de cada personaje para imprimir un aviso y pregunta si usted desea establecer los valores de la persona en cuestión. Si responde en sentido afirmativo, se visualizan los distintos atributos, cuyos títulos están almacenados en la matriz d$, y un valor entrado para cada uno. Al entrar una serie nula para un atributo (pulsando sólo Enter), saltará hasta

Detective con fallos

Sherlock, de Melbourne House, constituye un buen ejemplo de un programa que ofrece avanzadas facilidades interactivas totalmente en RAM, sin tener que cargar datos adicionales desde disco o cassette. El jugador puede charlar con los personajes utilizando «Decirle a...» y «Háblame acerca de...» y debe resolver varios delitos. Las primeras versiones del programa incluían algunos divertidos errores en la manipulación de personajes; en virtud de uno de estos «gazapos», el doctor Watson y Holmes estaban sentados en la misma silla simultáneamente



Módulo de inicialización

Este módulo irá experimentando adiciones a medida que se introduzcan nuevas variables y matrices

```
10 REM *bienvenido al Dog and Bucket*
20 REM
30 REM.....inicializar.....
40 REM
45 GOSUB 4030:REM limpiar la pantalla
50 DIM 1$(3,5),b$(12,4),c$(7,11),d$(11)
60 r=1
70 PRINT «Valores por defecto (s/n)?":GOSUB 4110:IF i$="s"
   OR i$="S" GOTO 90
80 GOSUB 2300:GOTO 100
90 FOR n=1 TO 7:READ c$(n,1):FOR d=2 TO 11:READ
   c$(n,d):NEXT d:NEXT n
100 FOR n=1 TO 3:READ 1$(n,1):FOR e=2 TO 5:READ
   1$(n,e):NEXT e:NEXT n
110 FOR n=1 TO 12:READ b$(n,1):FOR d=2 TO 4:READ
   b$(n,d):NEXT d:NEXT n
120 FOR n=2 TO 11:READ d$(n):NEXT n
```

Bucle del programa

En futuros capítulos iremos alterando esta línea para que llame rutinas nuevas a medida que las vayamos añadiendo

```
500 REM
510 REM bucle del programa de prueba
520 REM
530 GOSUB 2100:GOSUB 2150:GOSUB 2240:GOSUB
   2030:PRINT: PRINT:GOTO 530
```

Módulo subrutinas de bajo nivel

Compruebe el recuadro *Complementos al* BASIC para su micro en particular; la versión impresa es para el Amstrad 464/664 y BASIC compatibles

```
2000 REM
2010 REM rutina de entrada
2020 REM
2030 GOSUB 4110
2040 IF (ASC(i$)<48) OR (ASC(i$)> 51) THEN
   RETURN
2050 IF i$=0 THEN GOSUB 2320:RETURN
2060 r=VAL(i$):c$(7,2)=i$:RETURN
2070 REM
2080 REM impresión escenario
2090 REM
2100 PRINT 1$(r,1)
2110 RETURN
2120 REM
2130 REM imprimir objetos visibles
2140 REM
2150 PRINT «Ves: »;
2160 p=0: FOR b=1 TO 12: IF VAL(b$(b,2))<>r GOTO 2190
2170 p=p+1: IF p> 1 THEN PRINT «, »;
2180 PRINT b$(b,1);
2190 NEXT b
2200 IF p=0 THEN PRINT «nada»;
2210 PRINT: RETURN
2220 REM
2230 REM imprimir personajes visibles
2240 REM
2250 p=0: FOR c=1 TO 6:IF VAL(c$(c,2))<>r GOTO 2290
2260 p=p+: IF p=1 THEN PRINT «Estás en compañía de:
   »;:GOTO 2280
2270 PRINT «,»;2280 PRINT c$(c,1);
2290 NEXT c
2300 IF p=0 THEN PRINT «Aquí no hay nadie.»
2310 RETURN
2320 REM
2330 REM subrutina inicializar personajes
2340 REM
2350 PRINT: RESTORE: FOR c=1 TO 6:READ c$(c,1):GOSUB
   4070:PRINT c$(c,1);« – »;:PRINT «Establecer este
   personaje?»: GOSUB 4110
2360 IF (i$<>«s») AND (i$><«S»)THEN FOR n=2 TO
   11:READ n$:NEXT n: GOTO 2390
2370 FOR d=2 TO 11:READ s$:PRINT d$(d);:INPUT i$:IF i$<>
   «» THEN c$ (c,d)=i$
2380 NEXT d
2390 NEXT c: RETURN
```

el siguiente, dejando sin modificar el valor original. Obsérvese, no obstante, que si se llama esta rutina durante la inicialización, se debe entrar un valor para cada atributo y se deben inicializar todos los personajes, ya que de lo contrario el manipulador de personajes no estará en condiciones de funcionar correctamente.

El módulo de «subrutinas de bajo nivel» que empieza en la línea 4000, se utiliza fundamentalmente para simplificar la conversión a máquinas diferentes.

Los datos se almacenan en último lugar. Naturalmente, a medida que programemos otras rutinas iremos introduciendo considerables cosas en este módulo. En particular, necesitaremos almacenar los distintos mensajes para los personajes.

Una vez que haya entrado el listado, ejecútelo (RUN) y experimente pasando de un escenario a otro pulsando las teclas 1, 2 y 3. También puede comprobar el funcionamiento del editor de personajes pulsando 0 y cambiando las entradas para los escenarios de los personajes, llevándolos de una habitación a otra y verificando que sus nuevas posiciones se visualicen correctamente.

Ahora ya disponemos de un «entorno» en el que dar cabida a nuestros personajes, y dentro del cual podemos verlos en funcionamiento. En el próximo capítulo veremos cómo vamos a «darles vida».

Módulo subrutinas principales

En primer lugar, sólo tenemos cuatro subrutinas principales: entrada, imprimir escenarios, personajes y objetos, y la subrutina «editora de atributos de los personajes»

```
4000 REM
4010 REM subrutinas de bajo nivel del sistema
4020 REM
4030 REM limpiar la pantalla
4040 REM
4050 CLS:RETURN
4060 REM
4070 REM bip
4080 REM
4090 PRINT CHR$(7);:RETURN
4100 REM
4110 REM tomar un carácter del teclado
4120 REM
4130 i$=INKEY$: IF i$="» GOTO 4130
4140 RETURN
```

Módulo de datos

Observe que se han formateado las descripciones de los escenarios para una visualización a 40 columnas. Quizá deba modificarlas para adecuarlas a su micro. Por supuesto, podría alterar cualquiera de los valores por defecto de las líneas 6030-6040, en particular los valores entrados para «Ud.»

```
6000 REM
6010 REM datos de los personajes
6020 REM
6030 DATA "Luis Cubas","2","7","10","10","7",
     "v","0","0","7","4","Lola Fiestas","1",
     "8","30","10","8","m","0","0","3","5",
     "Pepe Viñas","1","9","8","10","9","v",
     "0","0","4","6","Mari Tapas","2","0",
     "20","10","10","m","0","0","5","5"
6040 DATA "Javi Salado","2","11","10","6","11",
     "v","0","0","4","6","Gina Fizz","1","12",
     "15","6","12","m","0","0","5","5",
     "Ud.","1","0","255","255","0","v","0",
     "0","0","0"
6050 REM
6060 REM datos escenarios
6070 REM
6080 DATA "Ud. se halla en la sala de tertulia del Dog
     and Bucket. En un rincón hay varios personajes
     dudosos jugando al dominó. Detrás del mostrador
     está Fred, el barman, con su habitual aspecto
     festivo. Las salidas dan al este.","0","0","2","0"
6090 DATA "He aquí el salón del Dog and Bucket, al cual
     le vendría muy bien una redecoración completa.
     Parece que el suelo ha sido regado regularmente
     con cerveza derramada. Hay puertas hacia el oeste
     y hacia el sur.","0","3","0","1"
6100 DATA "Uf! Ésta es la cocina, donde se preparan las
     famosas empanadas de carne del Dog and Bucket,
     para una clientela que siempre está famélica. Ud.
     observa numerosas latas vacías de alimento para
     gatos, lo cual no deja de ser extraño, ya que no hay
     ningún gato.","2","0","0","0"
6110 REM
6120 REM datos objetos (para b$(12,4))
6130 REM
6140 DATA "un vaso de cerveza","2","n","s"," una
     lata vacía de alimento para gatos","3","n","n",
     una empanada de carne del Dog and Bucket",
     "1","s","n"," una banqueta de bar",
     "2","n","n","un cenicero","1","n","n"
6150 DATA "un bocadillo de jamón rancio",
     "2","s","n","una pinta de cerveza amarga",
     "0","n","s","una crema de menta",
     "0","n","s","un whisky con agua",
     "0","n","s","un vodka puro",
     "2","n","s","una pinta de cerveza añeja",
     "0","n","s","una ginebra con ginger ale",
     "0","n","s"
6160 REM
6170 REM datos atributos personajes (para d$(11))
6180 REM
6190 DATA "Escenario", "Fortaleza", "Inventario",
     "Humor", "Objeto que tiene", "Sexo", "Ultimo
     personaje (Ich)", "Código última instrucción
     (Icd)", "Frecuencia de manipulación", "Frecuencia
     de movimiento"
```

Complementos al BASIC

Spectrum:

```
50   DIM L$(3,5,255), b$(12,3,25),
     c$(6,11,15),
     d$(11,25)
2040 IF(CODE(i$)< 48)OR (CODE(i$)> 51)
     THEN RETURN
4090 BEEP .5,3:RETURN
```

Commodore:

```
4050 PRINT"<shift/CLR> ":RETURN
4090 POKE 54296,15:POKE 54276,17:POKE
     54277,64
4095 FOR A=1TO50:POKE 54273,36:POKE
     54272,85:NEXT:POKE 54273, 0:POKE
     54272,0:POKE 54276,0:RETURN
4130 GET i$:IF i$="" THEN 4130
```

BBC:

```
4130 i$=GET$
```

Datos básicos (VII)

He aquí el análisis del siguiente fragmento del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
MEMSTR	0281–0282	641–642	Puntero: parte inferior de memoria para el OS
MEMSIZ	0283–0284	643–644	Puntero: parte superior de memoria para el OS
TIMOUT	0285	645	Flag: variable núcleo para el Timeout IEEE
COLOR	0286	646	Código color carácter actual
GDCOL	0287	647	Color de fondo bajo cursor
HIBASE	0288	648	Parte superior memoria pantalla (página)
XMAX	0289	649	Tamaño buffer teclado
RPTFLG	028A	650	Flag: tecla REPEAT usada, $80=Repeat
KOUNT	028B	651	Contador velocidad repetición
DELAY	028C	652	Contador demora repetición
SHFLAG	028D	653	Flag: tecla SHIFT/tecla CTRL/C=Tecla
LSTSHF	028E	654	Última configuración de shift
KEYLOG	028F–0290	655–656	Vector: establece tabla teclado
MODE	0291	657	Flag: $00=desactiva teclas SHIFT, $80=activa teclas SHIFT
AUTODN	0292	658	Flag: desplazamiento pantalla hacia abajo, 0=ON
M51CTR	0293	659	RS–232:imagen registro control 6551
M51CDR	0294	660	RS–232:imagen registro instrucciones 6551
M51AJB	0295–0296	661–662	RS-232: BPS no estándar (tiempo/2-100) en USA
RSSTAT	0297	663	RS232: imagen registro estado del 6551
BITNUM	0298	664	RS-232: número de bits que quedan por enviar

Acción industrial

Comenzamos una breve serie en la que esbozaremos el importante papel que desempeñan actualmente los ordenadores en la industria

Para ser eficaz, el proceso de manufacturación (desde el diseño del producto hasta la distribución de los productos acabados) exige conjugar hábilmente numerosos factores: tiempo frente a personas, máquinas frente a material, diseño frente a entrega y una docena de otras consideraciones. A principios de los ochenta, sólo los ordenadores más grandes eran suficientemente potentes como para ser de utilidad en las industrias. Gradualmente, esto ha ido cambiando, no sólo porque los pequeños ordenadores se han hecho más potentes, sino también porque las empresas han comenzado a tomar mayor conciencia de las ventajas que pueden aportar los ordenadores.

Es posible, aunque en muy pocos casos, ver un producto que ha sido realizado con el uso de ordenadores en todas las fases de fabricación, desde la etapa de diseño hasta el ensamblaje por robot. La mayoría de estas plantas avanzadas, como cabría suponer, pertenecen a la propia industria del ordenador. Las empresas de microelectrónica y los fabricantes de ordenadores personales figuran entre quienes marchan a la vanguardia de esta tecnología.

El empleo de ordenadores en todas las etapas del proceso de fabricación se conoce como CIM (*computer integrated manufacturing*: fabricación integrada por ordenador), pero la nomenclatura que se utiliza en esta área de aplicaciones puede ser muy desconcertante, como podemos ver en el recuadro *Jerga industrial*.

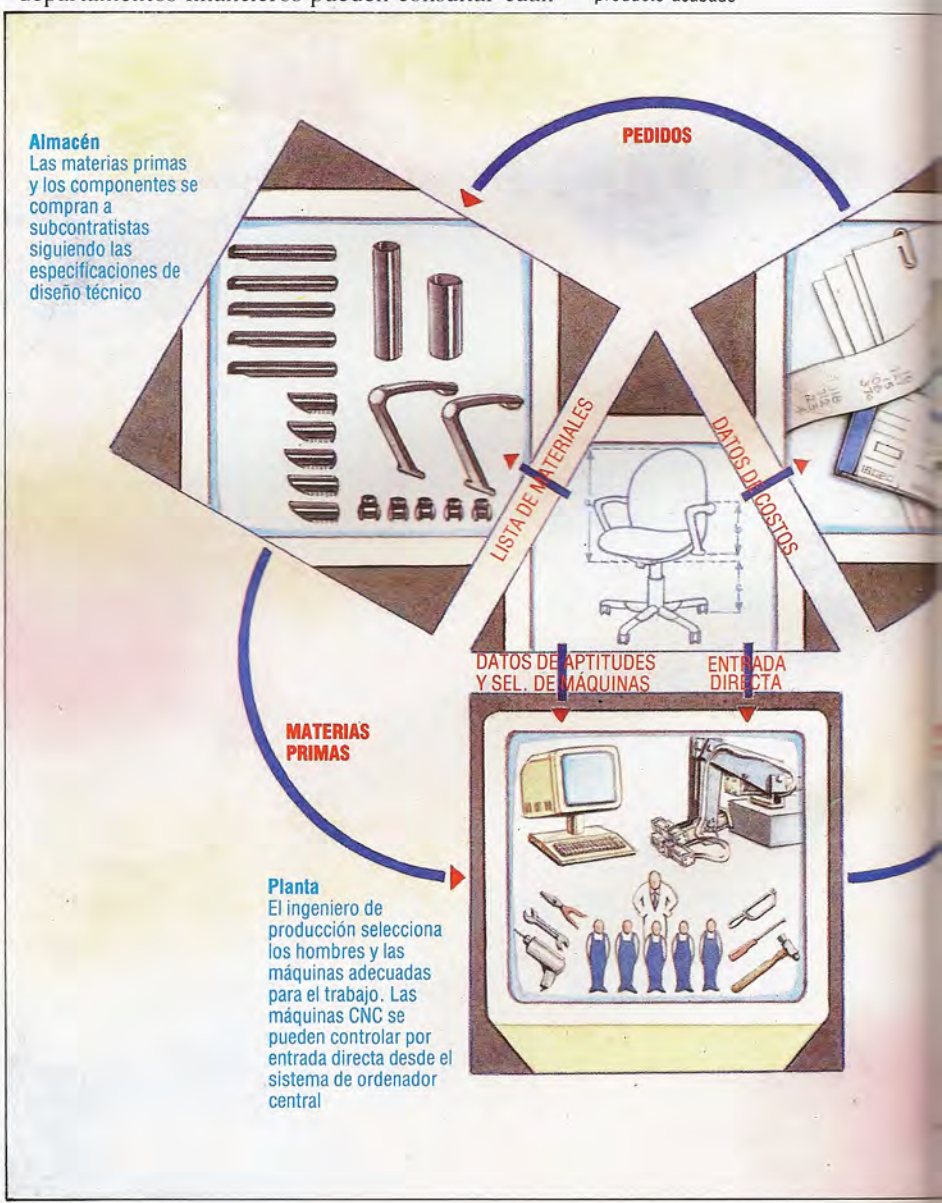
Los fabricantes trabajan sobre una base cooperativa, siendo cada departamento responsable de una parte separada del proceso. El departamento de diseño y dibujo mecánico desarrolla una idea y estipula las especificaciones técnicas que servirán de base a todo el proceso. Se habrán de pedir (y posiblemente fabricar) las herramientas para moldear el producto, y se habrán de programar las máquinas para producirlo. El almacén asegura que haya una constante disponibilidad de stocks de los materiales necesarios.

Luego está el aspecto financiero de la operación. Antes de pedir un producto, el consumidor desea saber el precio que tendrá y el fabricante ha de proporcionar una cotización exacta. Si ésta es demasiado elevada, el cliente irá a buscar a otro sitio, y si es demasiado baja, la empresa perderá dinero. Se debe aceptar el pedido, especificando una fecha de entrega y entregar el producto puntualmente, cobrándolo al final del proceso. El responsable de la planta ha de preocuparse por el despliegue de personal: cuántas personas se necesitan, qué aptitudes deben poseer y cuándo se necesitarán, así como asegurarse de que todos los pedidos aceptados se entreguen en las fechas especificadas.

Compartiendo datos

Por distintas que puedan parecer estas tareas, todas se basan esencialmente en los mismos datos. Ésta es la clave del enfoque CIM, en el que el ordenador retiene la base de datos que proporciona el núcleo de la información, y los distintos departamentos la utilizan en la medida de sus necesidades. El almacén puede consultar la descripción del producto y ver de qué está hecho, el encargado puede consultar qué máquinas se requieren para construirlo y los departamentos financieros pueden consultar cuán-

Valioso aporte
La informática puede aportar al proceso de manufacturación un grado de integración que hace un tiempo era impensable. El concepto CAD constituye la viga maestra de la organización de una fábrica informatizada, desde la etapa de diseño hasta el producto acabado



to cuesta producirlo: toda esta información se incluye en la base de datos.

Echemos ahora una mirada a los datos involucrados y los distintos pasos que supone su procesamiento. El dato crucial es, por supuesto, la *descripción del producto*, producida por el departamento de diseño. Ya hace muchos años que los sistemas CAD, de concepción asistida por ordenador, vienen ayudando en este proceso. Los productos compuestos a partir de un solo elemento son pocos; antes bien se componen de componentes hechos con componentes. Si tomamos un coche, por ejemplo, podemos ver que se trata de un diseño global, pero compuesto por millares de componentes distintos, desde el cableado de las luces traseras hasta el sistema de distribución del motor.

Los mejores sistemas CAD pueden formar una imagen del producto parte por parte, colocándolas en capas superpuestas hasta completar la imagen completa, en una representación tridimensional. A partir de esta imagen, el núcleo central de información, los programas de simulación y de análisis pueden comprobar no sólo el diseño, sino cómo funcionará el producto en la práctica.

La lista de materiales (*bill of materials*: BOM) es esencial para el correcto funcionamiento del almacén, al proporcionar una relación completa de todos los componentes del producto, que se puede descomponer en una lista de las materias primas necesarias. El interés del almacén consiste en cotejar la lista de componentes con la de los que hay en stock. El departamento querrá saber si todos los componentes están allí y si será preciso efectuar un nuevo pedido tras el inicio de la producción. De ser así, tendrán que informar al departamento de compras tanto de los suministros necesarios como de quiénes reciben habitualmente estas mercancías.

Los diseños del producto informan a los ingenieros de producción, responsables de la fabricación propiamente dicha, acerca de las necesidades de máquinas y de personal. A partir de este punto, está el pequeño paso de comparar las exigencias con el equipo y el personal disponibles y trazar un plan de trabajo.

Paul Bryant

La base de datos CIM

La CIM se utiliza no sólo para planificar la fabricación del producto, sino también para la planificación de varios productos diferentes que pasen todos por las diversas etapas de desarrollo de la misma fábrica. El sistema adecuado evitará situaciones en las cuales, por ejemplo, se hayan fabricado tornillos que sean inservibles por el mero hecho de que ninguna de las tuercas esté lista.

En otras palabras, entonces, la base de datos relativa a un producto, compilada en el momento del dibujo, puede detallar todos los procesos que entraña la construcción del mismo. La comparación de esta información con un plan de trabajo produce un programa laboral, y la misma base de datos informará a la oficina de contabilidad sobre los costos de producción. Medirá las cantidades de materiales a utilizar y, al planificar los tiempos-máquina y las aptitudes del personal, dará la información necesaria para proporcionarle al consumidor una cotización realista.

De los datos de la imagen precisa retenida en el dibujo, producido por el departamento de diseño, se puede pasar directamente a las máquinas controladas por ordenador que llevarán a cabo la operación. Ahora hay un tejido de información basado en las ideas originales de la oficina de diseño. El administrador del almacén, el departamento de herramientas, la oficina de ventas, los contables, el encargado e incluso la oficina de expediciones, todos ellos pueden sacar partido del mismo pozo de información relativa al producto.

En algunas de las industrias más nuevas los ordenadores han llegado a representar una parte tan activa del proceso de fabricación, que sin ellos no podrían continuar en funcionamiento. Por ejemplo, el diseño de los microprocesadores de hoy en día ya es demasiado complicado como para llevarlo a cabo sin ordenadores. El proceso llevaría demasiado tiempo si se realizara a mano, y sería imposible probar los diseños dibujados por los delineantes sin construir realmente el chip. Eso en sí mismo sería demasiado oneroso como para ser viable.

La CIM es la administración central de información sobre procesos de fabricación, y para ser eficaz requiere que se la alimente con información fiable. Con este fin se han desarrollado varias técnicas informáticas y en los próximos capítulos analizaremos algunas de ellas, así como las distintas formas en que pueden colaborar de manera insustituible en la industria.

Jerga industrial

Las aplicaciones de la industria basadas en ordenador están envueltas en el misterio de una jerga de siglas. He aquí algunos de los términos más utilizados:

AMT *Advanced Manufacturing Techniques*: técnicas avanzadas de fabricación. Engloba CAD, CAM y CAE

CAD *Computer Aided Design*: concepción asistida por ordenador. A veces se utiliza en alusión al dibujo técnico asistido por ordenador, una aplicación menos sofisticada capaz de dibujar imágenes pero no de analizarlas

CAM *Computer Aided Manufacture*: Fabricación asistida por ordenador. Se refiere a cualquier área en la que se utilicen ordenadores en el proceso de fabricación

CAE *Computer Aided Engineering*: ingeniería asistida por ordenador. El uso de datos (por lo general, la base de datos CAD) para llevar a cabo el análisis de un diseño

Máquina NC Máquina de control numérico. Utilizada para realización automática de piezas en el proceso de fabricación

Máquina CNC Máquina de control numérico por ordenador. Al igual que los ordenadores, las primeras máquinas NC solían ser activadas por instrucciones en cinta de papel. Ahora se pueden utilizar ordenadores para programar directamente las máquinas

CIM *Computer Integrated Management*: proceso de automatización de la administración de una planta industrial

BOM *Bill of Materials*: lista de materiales. Relación de componentes requeridos para la fabricación de un producto

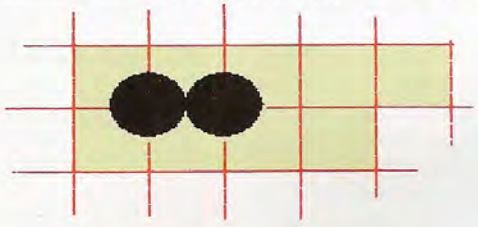
Conciencia de grupo

A partir de ahora nuestro programa tratará todas las posibles agrupaciones de fichas

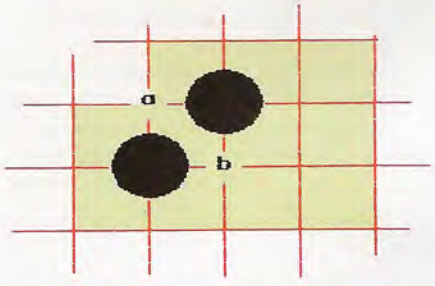
Al terminar el juego, el territorio sólo está rodeado oficialmente si se halla delimitado por una línea continua de fichas. Ésta consistirá en uniones sin interrupción (fig.1) o en uniones en diagonal (fig. 2), donde si las blancas juegan a *a*, las negras pueden formar dos uniones sin interrupción jugando *b*, y viceversa. Sin embargo, por lo general estas uniones sólidas sólo se pueden formar al final de la partida.

Durante la mayor parte del juego, las fichas esta-

1

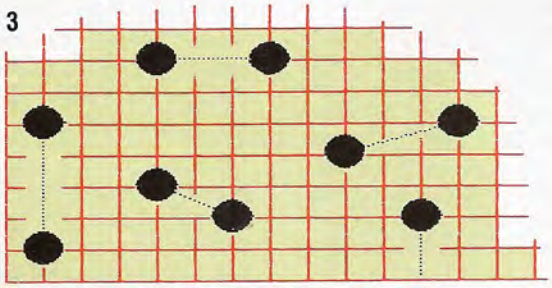


2



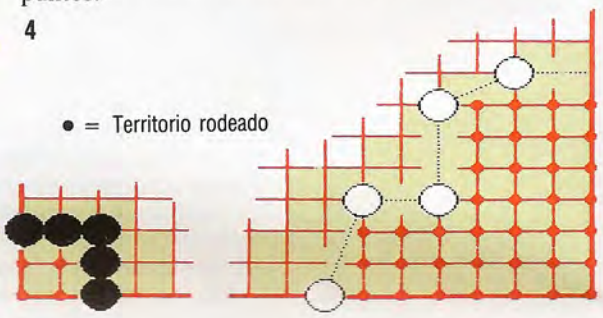
rán separadas por uno o más espacios, formando uniones imaginarias, bien con otras fichas, bien con el borde del tablero (fig. 3). En estos casos, sólo se formarán uniones sólidas cuando no exista ninguna posibilidad de que el oponente abra una brecha en estas «conexiones». No existe una garantía absoluta de que finalmente estas fichas se conecten, pero es bastante probable.

3



Obviamente, si queremos que nuestro programa juegue una partida aceptable, hemos de decirle algo acerca de estas posibles conexiones. Si sólo le enseñamos que para rodear territorio se pueden utilizar conexiones sólidas de fichas adyacentes, el programa se encontrará con problemas. Por ejemplo, en la figura 4, mientras que nuestro programa (negras) está rodeando cuatro puntos de territorio ¡el oponente ha empleado la misma cantidad de fichas para rodear un prometedor territorio de 37 puntos!

4

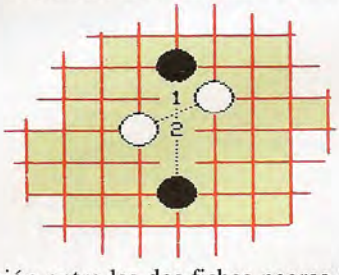


Habiendo decidido que el ordenador comprenda las conexiones, hemos de decidir cómo programarlo. Deseamos que manipule cuatro operaciones principales de conexión. Éstas son:

- Defender una conexión entre dos grupos de fichas cuando el oponente se mueva hacia una posible posición de ataque.
- Atacar una conexión del oponente colocando una ficha entre las fichas conectadas débilmente.
- Jugar una ficha de inicio de ataque. Ésta será una acción previa al ataque, colocando una ficha central, o bien proporcionará un apoyo para una ficha colocada en el centro.
- Iniciar una conexión colocando una ficha a cierta distancia de otra ficha colocada anteriormente. Mediante el empleo del sistema de estimación (desarrollado en el capítulo anterior) se evaluará esta ficha, y automáticamente se hallará una posición atinada para poder rodear territorio.

Estas operaciones se han programado utilizando técnicas sencillas de emparejamiento de formatos. Observe una situación típica como la que muestra la figura 5. Idealmente, las blancas desean romper

5



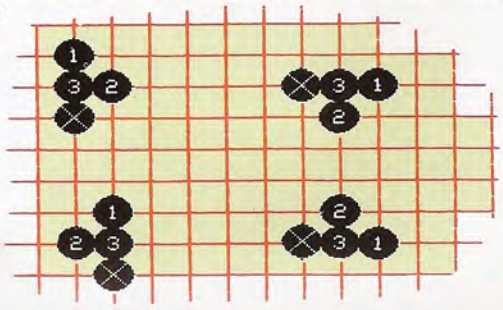
la conexión entre las dos fichas negras jugando en la posición «2», y las negras quieren defenderse jugando en la misma posición. Al mismo tiempo, las negras desean romper la conexión de la ficha blanca bien en «1», bien en «2», mientras que las blancas quieren defenderse jugando en estos puntos. En este caso particular, las blancas tienen ventaja al tener las fichas conectadas más estrechamente.

Es posible implementar los cuatro tipos de operaciones de conexión utilizando un conjunto único de formatos. Estos formatos se establecen mediante la rutina que comienza en la línea 790, que define los formatos ilustrados en las figuras 6, 7, 8 y 9. En todos los casos, la ficha con una cruz es nuestra

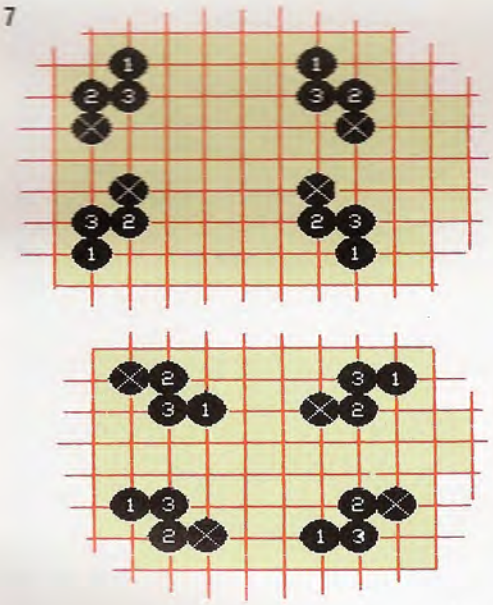
ficha actual (punto cero). Luego las sentencias DATA dan ramales desde este punto en el mapa del tablero%. De este modo, +16, −16, +1 y −1 corresponden respectivamente a norte, sur, este y oeste.

Puesto que a la larga todas las fichas se considerarán como la ficha actual, no es necesario implementar todas las orientaciones de cada patrón. Por ejemplo, en la figura 9 no tenemos un formato con

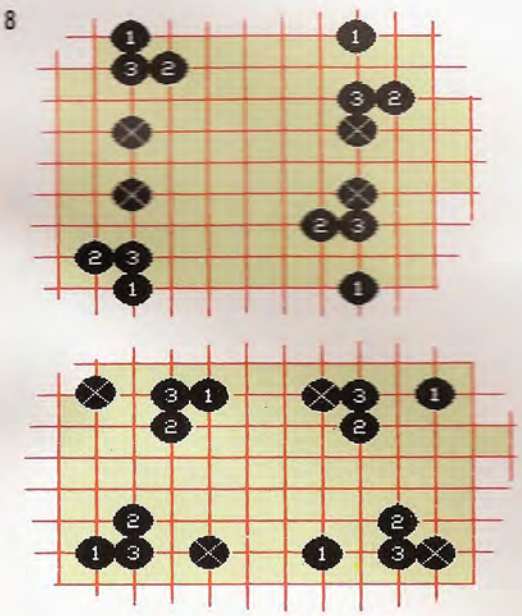
6



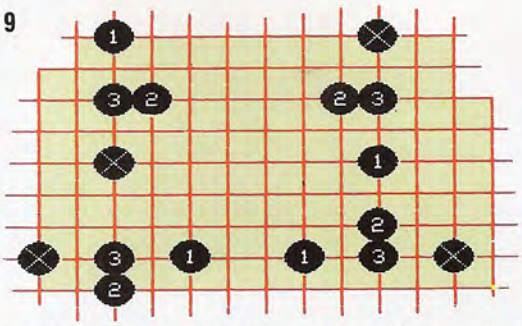
7



8



9



la ficha actual en la parte inferior (0), la primera ficha de la parte superior (+ 64) y la ficha 2 a la izquierda de la ficha 3 (+ 31 y + 32). No obstante, más adelante durante la investigación del tablero, la ficha que ahora está marcada como 1 se considerará como la ficha actual. El formato que vemos arriba y a la derecha de la figura 9 manipulará luego la orientación del formato que acabamos de describir. Por tanto, los formatos simétricos sólo requieren cuatro orientaciones en la memoria, y los formatos no simétricos requieren ocho.

Observará que todas las demás sentencias DATA no son más que la negación de la línea anterior, de modo que podríamos reducir aún más la cantidad de formatos si estuviéramos escasos de memoria, probando específicamente la negación de todos los formatos retenidos.

Las rutinas de emparejamiento de formatos trabajan calculando las posiciones de estos ramales en el tablero y, por tanto, hallando el color de las fichas, si las hubiera, en las correspondientes posiciones. Luego se aplica un conjunto de reglas para decidir si jugar o no una ficha. Si se va a jugar una ficha, a la posición se le asigna un marcador basado en las estimaciones del tablero; al final de la rutina, se utiliza la ficha que tenga el marcador más alto.

Las rutinas se comprueban por orden, de modo que si, por ejemplo, la rutina conexión__de__ataque encuentra un movimiento, entonces no se comprobarán las rutinas iniciar__ataque e iniciar__conexión.

Suponiendo que hubieran de jugar las negras, las reglas aplicadas para decidir si jugar o no una ficha son:

conexión__de__defensa:

IF	ficha con cruz	=NEGRAS
AND	ficha uno	=NEGRAS
AND	ficha dos	=BLANCAS
THEN	intentar movimiento de ficha tres.	

conexión__de__ataque:

IF	ficha con cruz	=BLANCAS
AND	ficha uno	=BLANCAS
AND	ficha dos	=NEGRAS
THEN	intentar movimiento de ficha tres.	

iniciar__ataque:

IF	ficha con cruz	=BLANCAS
AND	ficha uno	=BLANCAS
AND	ficha tres	=LIBRE
THEN	intentar movimiento de ficha dos.	

iniciar__conexión:

IF	ficha con cruz	=NEGRAS
AND	ficha tres	=LIBRE
THEN	intentar movimiento de ficha uno.	

Las cuatro rutinas de conexión que vemos aquí se llaman desde las líneas 2580-2610 de la rutina movimiento_negras. Con ello ahora manipularemos las conexiones entre las fichas del tablero. No obstante, si usted vuelve a observar las figuras 3 y 4, verá que también tenemos conexiones entre fichas y el borde del tablero, que son igualmente importantes. Para tratar con ellas hemos diseñado la rutina PROCfrontera. Recordará que nuestro tablero realmente se define DIM tablero% 256 para un tablero de 15 por 15. Esto da una ocupación de 256 bytes, y también nos deja con un borde de una sola ficha alrededor de todo el tablero. PROCfrontera rellena este borde con el valor pasado como parámetro a V%. Al asegurar que las rutinas de emparejamiento de formatos busquen por todo el tablero, incluyendo el borde (de 1 a 255), también se comprobarán las conexiones con el borde del tablero. Acuérdese de restablecer siempre a cero el borde cuando salga de la rutina, llamando PROCfrontera con un valor de cero.

Debido a la velocidad del programa la cantidad de formatos se ha mantenido en un mínimo. Sin embargo, usted está en libertad de añadir datos o cambiar los que empiezan en la línea 860. Si por algún motivo usted decide hacerlo, no olvide cambiar los parámetros del bucle en las líneas 2940, 3110, 3260 y 3410.

Módulo 5

BBC Micro:

```
 280 PROCleer_formatos
 780 :
 790 DEF PROCleer_formatos
 800 LOCAL L%
 810 DIM pat% 71
 820 RESTORE 860
 830 FOR L%=0 TO 71
 840   READ for%?L%
 850   NEXT
 860 DATA 32,17,16,2,-15,1
 870 DATA -32,-17,-16,-2,15,-1
 880 DATA 33,16,17,31,16,15
 890 DATA -33,-16,-17,-31,-16,-15
 900 DATA -14,1,-15,18,1,17
 910 DATA 14,-1,15,-18,-1,-17
 920 DATA 48,33,32,48,17,16
 930 DATA -48,-33,-32,-48,-17,-16
 940 DATA 3,-14,2,3,-15,1
 950 DATA -3,14,-2,-3,15,-1
 960 DATA 64,33,32,4,-14,2
 970 DATA -64,-33,-32,-4,14,-2
 980 ENDPROC
 990 :
1000 REM ****************************************
2580 IF posicion%=0 THEN PROCconexion_de_
     defensa:T$="DEF"
2590 IF posicion%=0 THEN PROCconexion_de_
     ataque:T$="ATT"
2600 IF posicion%=0 THEN PROCiniciar_ataque:T$="SAT"
2610 IF posicion%=0 THEN PROCiniciar_conexion:T$="SCN"
2890 :
2900 DEF PROCconexion_de_defensa
2910 LOCAL A%,B%,C%,D%,P%,S%,hi,marcador
2920 PROCfrontera(negras%)
2930 FOR A%=1 TO 255: S%=tablero%?A% : IF S% <>
     negras% THEN 3010
2940   FOR P%=for% TO for%+70 STEP 3
2950     B%=tablero%?((A%+?P%) AND
         255):C%=tablero%?((A%+?P%+1)) AND 255)
2960     IF B %<>negras% OR C%<>blancas% THEN
         3000
2970     D%=A%+?(P%+2)) AND
         255:marcador=RND(1)+estimaciones%?D%
2980     IF (D% AND 240)=0 OR (D% AND 15)=0 OR
         marcador<=hi THEN 3000
2990     IF FNlegalidad(D%,negras%)=0 AND clib%>2 THEN
         hi=marcador: posicion%=D%
3000     NEXT
3010   NEXT
3020 PROCfrontera(0)
3030 ENDPROC
3040 :
3050 REM ****************************************
3060 :
3070 DEF PROCconexion_de_ataque
3080 LOCAL A%,B%,C%,D%,P%,S%,hi,marcador
3090 PROCfrontera(blancas%)
3100 FOR A%=1 TO 255 : S%=tablero%?A% : IF S% <>
     blancas% THEN 3180
3110   FOR P%=for% TO par%+70 STEP 3
3120     B%=tablero%?((A%+?P%) AND
         255)):C%=tablero%?((A%+?(P%+1)) AND 255)
3130     IF B%<>blancas% OR C%<>negras% THEN 3170
3140     D%=(A%+?(P%+2)) AND
         255:marcador=RAN(1)+estimaciones%?D%
3150     IF (D% AND 240)=0 OR (D% AND 15)=0 OR marcador
         <=hi THEN 3170
3160     IF FNlegalidad(D%,negras%)=0 AND clib%>2 THEN
         hi=marcador: posicion%=D%
3170     NEXT
3180   NEXT:PROCfrontera(0)
3190 ENDPROC
3200 :
3210 REM ****************************************
3220 :
3230 DEF PROCiniciar_ataque
3240 LOCAL A%,B%,C%,D%,P%,S%,hi,marcador
3250 FOR A%=1 TO 255 : S%=tablero%?A% : IF S% <>
     blancas% THEN 3330
3260   FOR P%=for% TO pat%+70 step 3
3270     B%=tablero%?((A%+?P%) AND
     255):D%=tablero%?((A%+?(P%+2)) AND 255)
3280     IF B%<>blancas% OR D%<>0 THEN 3320
3290     C%=(A%+?(P%+1)) AND
         255:marcador=RND(1)+estimaciones%?C%
3300     IF (C% AND 240)=0 OR (C% AND 15)=0 OR
         marcador<=hi THEN 3320
3310     IF FNlegalidad(C%,negras%)=0 AND clib%>2 THEN
         hi=marcador: posicion%=C%
3320     NEXT
3330   NEXT
3340 ENDPROC
3350 :
3360 REM ****************************************
3370 :
3380 DEF PROCiniciar_conexion
3390 LOCAL A%,B%,C%,P%,S%,hi,marcador
3400 FOR A%=1 TO 255 : S%=tablero%?A% : IF S% <>
     negras% THEN 3470
3410   FOR P%=for% TO pat%+70 STEP 3
3420     C%=tablero%?((A%+?(P%+2)) AND 255):IF C%>0
         THEN 3460
3430     B%=(A%+?P%) AND
         255:marcador=RND(1)+estimaciones%?B%
3440     IF (B% AND 240)=0 OR (B% AND 15)=0 OR marcador
         <=hi THEN 3460
3450     IF FNlegalidad(B%,negras%)=0 AND clib%>2 THEN
         hi=marcador: posicion%=B%
3460     NEXT
3470   NEXT
3480 ENDPROC
3490 :
3500 REM ****************************************
4410 :
4420 DEF PROCfrontera(V%)
4430 LOCAL X%,Y%
4440 FOR X%=0 TO 15
4450   Y%=16*X%
4460   tablero%?X%=V%
4470   tablero%?Y%=V%
4480   NEXT
4490 ENDPROC
4500 :
4510 REM ****************************************
4520 REM ********** FIN DEL PROGRAMA **************
```

Video mágico

«Frankie goes to Hollywood» (Frankie va a Hollywood), de Ocean Software, es un programa creado con notable inteligencia e imaginación

La búsqueda de Frankie
Frankie goes to Hollywood tiene como objetivo buscar objetos en numerosas habitaciones. Cuando se abre un cajón, su contenido se visualiza en una ventana de la pantalla. También se puede «entrar» en ciertos cuadros y jugar a un juego de tipo recreativo

Liz Heaney

Frankie goes to Hollywood no se basa tanto en la música del grupo británico de igual nombre, sino más bien en su filosofía de escapar del mundo cotidiano («Mundanesville») hacia la cumbre del placer («Pleasure Dome»), y, en el camino, convertirse en una «verdadera persona».

El juego es un producto de Denton Designs, un equipo de programación de Liverpool, ciudad de la que también es originario el grupo. Contratado originalmente por Imagine, que cesó sus actividades en 1984, ahora el equipo trabaja en base a colaboraciones independientes para numerosas empresas de software, produciendo, entre otros programas *Shadowfire*, tan bien considerado.

Al tener numerosos puntos de semejanza con su antecesor, *Frankie...* es un juego de aventuras con numerosos lugares para explorar (en este caso una fila de casas con terraza). Mientras usted se va desplazando por las casas, puede ir revisando los muebles en busca de objetos que puedan serle útiles. Estos objetos están escondidos en cajones y otros lugares tan inverosímiles como las lavadoras.

La mayoría de las aventuras exigen que se digiten instrucciones, pero Denton Designs ha adoptado el método de explorar objetos que utilizara por primera vez en *Shadowfire*. Una vez que se ha hallado un objeto, se lo puede recoger o abandonar colocando el cursor sobre el mismo y pulsando el botón de disparo. Entre los objetos más comunes que se descubrirán hay cuatro pequeños iconos que corresponden a los cuatro componentes de su personalidad. Éstos se indican en el lado derecho de la pantalla y van aumentando a medida que el usuario se convierte en una «verdadera persona». Los iconos que corresponden a estos componentes son «píldoras de placer» que, al tomarlas, aumentan su porcentaje de ese aspecto de su personalidad.

El jugador sólo puede llevar consigo ocho objetos al mismo tiempo y, a medida que su inventario va creciendo, tiene la tentación de tomarse las píldoras apenas las encuentra. Sin embargo, éstas son más útiles en una etapa ulterior del juego y, por ello, es interesante conservarlas durante un tiempo.

Otro conjunto de objetos que aparece continuamente son videos que le permiten a usted entrar en una faceta del juego completamente distinta. La mayoría de las casas tienen aparatos de televisión y en ellos se pueden insertar las cassettes de video. Aparte de ganar numerosos puntos para el marcador, el video aparecerá en una ventana de la pantalla. Al desplazar su «persona» dentro de la pantalla de video, el programa cambia a uno de numerosos juegos de tipo recreativo. Es durante estos juegos cuando el comer las píldoras del placer hará que su marcador se dispare al máximo. En algunas de las casas hallará cuadros colgados en las paredes y también podrá entrar en ellos y jugar a un juego recreativo adicional.

En las casas se pueden encontrar, asimismo, varios discos flexibles. En un primer momento parecería que éstos no tuvieran ningún uso práctico, ya que tampoco parece haber ningún ordenador donde colocarlos. Sin embargo, escondida en algún lugar del juego hay una sala de ordenadores y en sus manos está hallarla. Como es natural, cuando finalmente descubre esta habitación, la información retenida en el disco le ayudará en su búsqueda para convertirse en una verdadera persona.

En el curso de sus viajes, usted descubre que se ha cometido un asesinato, y una de sus tareas consiste en identificar al criminal. A intervalos aparecen de forma intermitente en la pantalla pistas para resolver el asesinato. Por ejemplo, una pista podría ser El asesino es socialista, mientras que otra podría ser Miss Mundane siempre vota a los conservadores, con lo que ella quedaría eliminada de la lista de sospechosos.

A pesar de que muchas de las facilidades de *Frankie goes to Hollywood* se han tomado prestadas de otros juegos, es decididamente bueno por derecho propio. La adición en un mismo programa de tantas facetas diferentes es ciertamente insólita, y entretendrá a los amantes de los juegos durante un tiempo considerable.

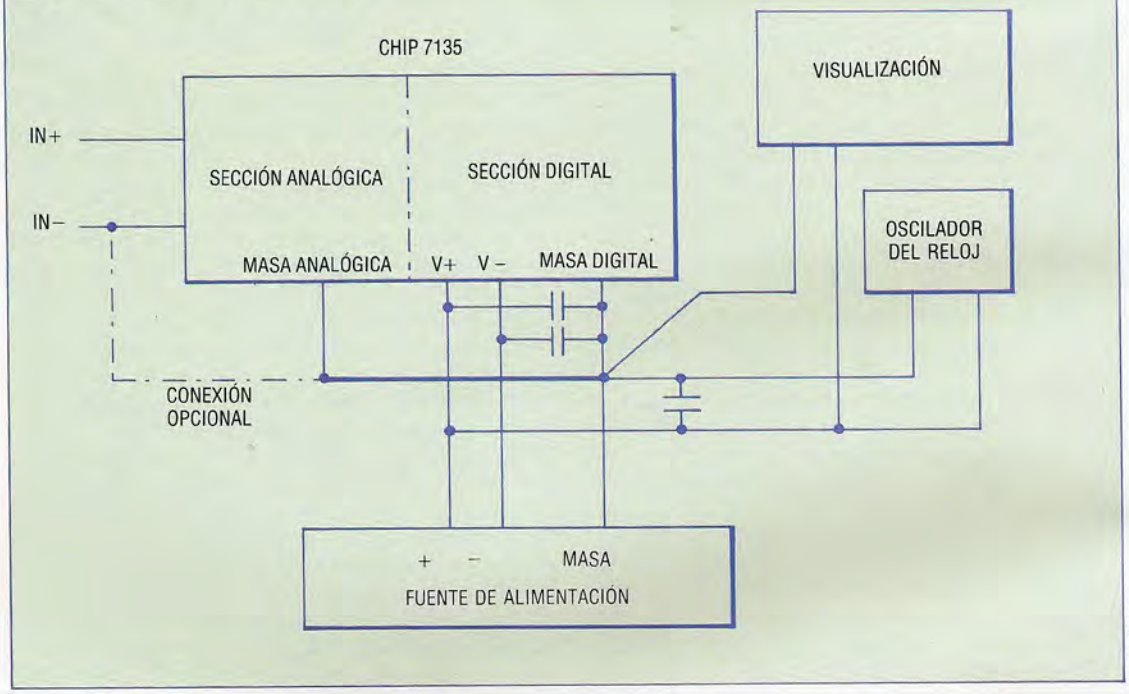
Frankie goes to Hollywood: Para el Sinclair Spectrum y el Commodore 64
Editado por: Ocean Software, Ocean House, 6 Central Street, Manchester, Gran Bretaña
Autores: Denton Designs
Formato: Cassette o disco
Palancas de mando: Necesarias

Centro vital

Examinemos el chip convertidor 7135 y estudiemos algunos puntos importantes referentes al trazado de la placa y la selección de componentes

Principios de la alimentación
Al considerar el trazado del circuito del DVM es muy importante asegurar que las secciones analógica y digital del diseño no entren en contacto entre sí. El esquema general de cableado que vemos aquí minimiza los bucles de masa y el acoplamiento accidental de señales digitales con el circuito analógico

El convertidor A/D



Kevin Jones

Remitiéndonos, en primer lugar, al esquema del circuito realizado anteriormente, que muestra la disposición de las patillas del 7135 y sus respectivos nombres, iremos refiriéndonos a ellas por el orden de los números de patillas. Ulteriormente ofreceremos una lista completa de componentes.

1. *V*–: el 7135 requiere una fuente de alimentación tanto positiva como negativa. Las *especificaciones máximas absolutas* de estas entradas son +6 V para la línea positiva y −9 V para la negativa. Para simplificar las exigencias de la fuente de alimentación, utilizaremos +/− 5V. La corriente de alimentación de −5 V típicamente es de 0,8 mA, de modo que todo cuanto se requiere es una fuente de alimentación muy pequeña y sencilla. En nuestro diseño, la unidad se alimentará con la red eléctrica, pero también se podría utilizar una pila de 6 V (tanto los chips MOS como los CMOS funcionan bien con una fuente de 6 V). En la fase de construcción de esta serie ofreceremos detalles relativos a la fuente de alimentación con la red eléctrica.
2. *Referencia*: La tensión de referencia para el 7135 (o para cualquier convertidor A/D de atenuación doble) es crítica para su precisión general. La desviación de fondo de escala (d.f.e.) —el máximo valor de salida— de un convertidor A/D de atenuación doble es el doble que la tensión de referencia, de modo que para un tester con una d.f.e. de 2 V, el voltaje de referencia ha de ser de 1,000 V.

Existen muchísimas formas de obtener una referencia exacta de 1 V, pero nosotros aconsejamos utilizar un diodo de referencia de intervalo de banda de precisión de 1,22 V, del tipo 9491. La referencia de 1 V requerida se obtiene luego utilizando un potenciómetro para reducir la tensión al nivel requerido. No caiga en la tentación de utilizar un potenciómetro barato del tipo de control de volumen. Utilice un potenciómetro de ajuste multivueltas (de diez o más vueltas) de la mejor calidad. Pida prestado un buen DVM y regule el potenciómetro de ajuste para 1,000 V. Mejor aún, monte un divisor de potencial en la salida de la fuente de alimentación de prueba de baja tensión, y regúlelo para obtener en el DVM prestado una lectura de 1,999 V; alimente éste a la entrada de su DVM y regule el potenciómetro de ajuste de referencia para una lectura de 1,999 V en su visualización.

Si no tiene acceso a un DVM de precisión, lo mejor que puede hacer es crear una fuente de tensión de precisión moderada utilizando dos células de mercurio de 1,35 V en serie y un divisor de potencial construido con una resistencia de 36 K-ohmios y una de 100 K-ohmios. La misma dará aproximadamente 2 V para utilizar como entrada de prueba para el DVM. De nuevo, regule el potenciómetro de ajuste de referencia para obtener en la visualización una lectura de 1,999 V. Emplee resistencias de estrecha tolerancia (1 % o mejores) y células de mercurio nuevas.

3. *Masa analógica*: La patilla del 7135, de apariencia inocua, es la que probablemente cause más

problemas que ninguna otra en su DVM. El problema proviene del hecho de que hay dos señales de masa separadas: masa analógica y masa digital (la masa digital está en la patilla 24). Las señales analógicas, alimentaciones + y −, y circuitos de masa de las secciones analógica y digital, deben mantenerse estrictamente separadas. Se requiere un sistema de cableado que minimice la posibilidad de que las señales digitales se introduzcan en la sección analógica y viceversa, si bien este último caso no reviste tanta importancia.

Los diseñadores de circuitos le dirán que debe adoptar buenos principios de trazado, pero esto no es de gran ayuda si usted no sabe cuáles son los principios de un buen trazado. No profundizaremos demasiado y nos limitaremos simplemente a decirle qué debe hacer y qué no debe hacer.

Todos los puntos de masa analógica (la unión del diodo de referencia y el extremo inferior del potenciómetro de ajuste conectado a la patilla 3) y el extremo a masa de la resistencia conectada al diodo (que está conectado a los condensadores de integración y de autocero en las patillas 4 y 5) se deben soldar en un mismo punto. Suelde el ánodo del diodo de referencia y la resistencia de 100 K-ohmios (conectada a *Int.Out.* a través de un diodo) a un punto único.

Medidas de precaución

Es igualmente importante conectar todos los puntos de masa digital en un mismo punto. Observe que el esquema del circuito utiliza un símbolo diferente para los dos tipos de masa: ⏚ para masas analógicas y ⏚ para masas digitales. Habiendo unido todas las conexiones de masa analógica en un único punto, y todas las digitales en otro punto único y distinto, se deben conectar entre sí las dos tierras. Esto debe hacerse utilizando un solo trozo de conductor de cobre macizo de gran sección.

De no respetarse estas precauciones se produciría una multitud de complicaciones. Las lecturas de la visualización serían inexactas, la visualización podría pasar por varias lecturas (erróneas) alternativas, el DVM podría no captar las lecturas, y todo oscilaría o se clavaría.

La fuente de alimentación para el reloj, el activador de la visualización y la propia visualización deben desacoplarse cuidadosamente empleando tanto condensadores electrolíticos de valor elevado como condensadores de disco cerámicos de bajo valor. Es importante que los componentes analógicos del circuito (como el diodo de la tensión de referencia, las resistencias integradoras, los condensadores, etc.) queden situados en la placa en el lado opuesto de los componentes digitales (tales como la señal CLOCK y las líneas STROBE y BUSY).

De no hacerse así, es probable que señales como BUSY y STROBE, que en el curso de una lectura pasan de un nivel lógico a otro, se acoplen con la entrada y provoquen errores. La regla es: mantener los componentes analógicos y digitales del circuito separados físicamente en la placa, y prestar atención al trazado de las líneas de masa de modo que los bucles no creen problemas adicionales. El diagrama *Principios de la alimentación* ilustra la forma general en que debe plantearse la alimentación.

En tercer lugar, no se debe dejar flotando ninguna señal digital inutilizada (tales como STROBE, UNDER-RANGE, RUN/HOLD, etc.); se las debe dejar altas empleando resistencias de 4,7 K-ohmios o bien dejar bajas mediante resistencias de 440 ohmios.

4. *Int.Out.(Salida integradora)*: Los convertidores A/D de tipo integrador como el 7135 se basan en el supuesto de que el cambio de tensión en el condensador integrador es exactamente proporcional a la corriente que circula en él. El valor absoluto de este condensador no es muy crítico (0,4 o 0,47 µF [microfaradios], p. ej.), pero es vital que tenga unas pérdidas dieléctricas muy bajas. Los condensadores de polipropileno son los mejores, pero los de poliestireno son casi igual de buenos, y también se pueden emplear los de policarbonato. No utilice ninguna clase de condensador electrolítico.

5. *A-Z IN*: El condensador utilizado aquí puede ser de cualquier calidad. El valor no es crítico, si bien cuanto mayor sea, tanto mejor. Emplee condensadores de polipropileno, poliestireno o policarbonato, pero no electrolíticos.

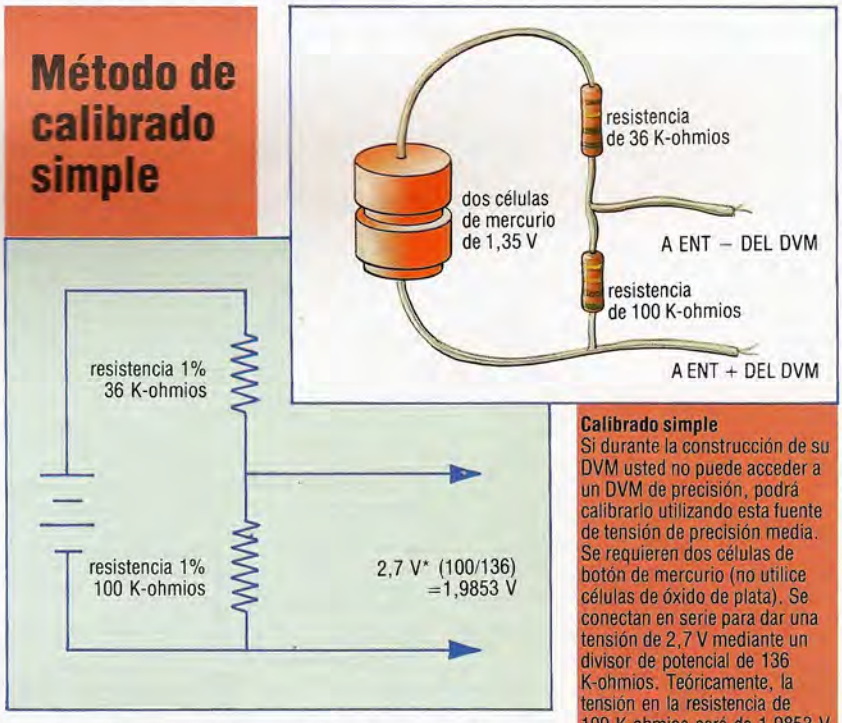
6. *BUF OUT (Buffer Output)*: Éste es el punto donde se conecta la resistencia integradora. El valor de ésta se obtiene con:

$$R = \frac{\text{tensión de fondo de escala}}{20\ \mu A}$$

Servirá cualquier resistencia de 100 K-ohmios, si bien una resistencia de película metálica de estrecha tolerancia (1% o mejor) sería mejor que las resistencias de carbón de baja tolerancia que uno suele tener olvidadas en el fondo del cajón.

El diodo conectado a la salida integradora (patilla 4) es para corregir un pequeño error de deriva. Servirá uno de silicio de pequeña señal de cualquier tipo, de los que suele haber montones en los cajones. Aunque por lo general a los diseñadores de circuitos poco experimentados les agrada que se les recomiende un tipo específico, casi cualquier diodo servirá, aunque ha de ser de silicio y no de germanio, y de pequeña señal en vez de rectificador.

Método de calibrado simple



Calibrado simple
Si durante la construcción de su DVM usted no puede acceder a un DVM de precisión, podrá calibrarlo utilizando esta fuente de tensión de precisión media. Se requieren dos células de botón de mercurio (no utilice células de óxido de plata). Se conectan en serie para dar una tensión de 2,7 V mediante un divisor de potencial de 136 K-ohmios. Teóricamente, la tensión en la resistencia de 100 K-ohmios será de 1,9853 V, pero podría oscilar de forma imprevisible alrededor de +/− un 2% de esta cifra, siempre que se utilicen resistencias del 1% de tolerancia. Existen a la venta resistencias económicas de película metálica del 1% dentro de estos valores. 36 K-ohmios es uno de los valores de resistencias de la «gama ampliada». No intente construirlo con resistencias de la «gama estándar», o correrá el riesgo de perder precisión. El potenciómetro de ajuste deberá regularse hasta que la visualización dé una lectura de 1,9999 V

Nitidez

Nuestro estudio del sistema operativo del Spectrum concluye con un examen de la pantalla

El manejo de los gráficos y de la pantalla no es tan difícil en el Spectrum. En primer lugar examinaremos cómo se usan las rutinas que están en la ROM y que son empleadas por el sistema operativo.

Para direccionar la pantalla primero debe abrirse el canal S seleccionando el flujo 2. Para una operación de impresión (PRINT) normal, el método más fácil es, sin duda, la rutina RST#10. Pero si deseamos imprimir una cadena entera de caracteres, ha de escribirse una pequeña rutina para entrar cada carácter uno a uno, y pasarlos también uno a uno a la rutina RST#10 hasta agotar toda la impresión.

No obstante, en lugar de escribir una rutina propia, es preferible usar la que ya existe en la ROM encargada de hacer esa tarea. Para llamarla, basta con cargar DE con la dirección en la que hay almacenada la cadena, cargar BC con la longitud de la cadena y llamar la rutina Print String que está en #203C. Véase una muestra de lo dicho en el siguiente listado:

```
                ORG  60000
1172EA          LD   DE,MSG
3E02            LD   A,2              ; selecciona flujo 2
CD0116          CALL #1601            ; abre canal S
1172EA          LD   DE,MSG           ; apunta DE a la cadena
012600          LD   BC,38            ; carga BC con la long. cadena
CD3C20          CALL #203C            ; llama PRINT-STRING
C9              RET
160A06    MSG:  DEFB 22,10,6          ; códigos para AT 10,5
54484520        DEFM «VAN TRES»
160C07          DEFB 22,12,7
41445641        DEFM «DEL APOCALIPSIS»
```

Existe otra rutina ROM muy útil: la Print a Floating Point Number (impresión de un número en coma flotante: abreviadamente, PRINT-FP), que está en #2DE3. Ya hemos mostrado cómo se evalúa una expresión numérica y cómo se coloca en la pila de cálculo empleando la rutina que está en #1C82. La rutina PRINT-FP nos puede facilitar las cosas, toda vez que imprimir un número en código máquina es una tarea bastante difícil. Para imprimir un número, primero hay que colocarlo en la pila de cálculo cargándolo, bien en el par de registros BC, bien en el registro A. La rutina STACK-A se llama en #2D28 y la rutina STACK-BC en #2D2B. Una vez en la pila, la impresión se logra con una sencilla llamada a la rutina PRINT-FP.

```
                ORG  60000
013930          LD   BC,12345         ; toma el número en BC
CD2B2D          CALL #2D2B            ; BC empujado a pila cálculo
3E02            LD   A,2
CD0116          CALL #1601            ; abre canal S
CDE32D          CALL #2DE3            ; imprime No. de encima pila
C9              RET
```

La verdadera potencia de los gráficos del Spectrum se hace más patente cuando empleamos las instrucciones de alta resolución PLOT y DRAW. Ya tratamos de la primera. Pues bien, DRAW es tan fácil como PLOT, salvo que son necesarios unos cuantos parámetros más. Ya sabemos que la instrucción tiende a trazar una línea lo más directa posible entre el último punto trazado y un punto definido según el origen. Así, si hubiere que dibujar una línea 20 pixels hacia arriba y 40 pixels a la izquierda, la instrucción sería DRAW −40, 20.

A fin de explicar el empleo de la instrucción DRAW en código máquina, nos serviremos de la expresión sintáctica DRAW x, y. En la ejecución de DRAW en código máquina, los parámetros x e y deben considerarse por separado. Para simplificar el tema, de momento nos despreocuparemos de cualquier factor para curvas. Si ha de hacerse un retorno al BASIC en un determinado punto, habrá que conservar el valor del par alternativo de registros H'L', dado que la rutina DRAW alterará los datos en él contenidos.

El valor absoluto de x (ABS x) se lleva al registro C, mientras que ABS y se pasa al registro B y los valores de signos de y y de x se llevan a la rutina en D y E respectivamente. Pueden emplearse valores superiores a −1, 0 y 1, puesto que la rutina emplea estos valores como incremento efectivo que se añade al último pixel trazado para alcanzar la posición del siguiente pixel de la línea.

Esto significa que si se utilizase un valor de signo 2 o 3, la línea sería dos o tres veces más larga de lo normal, pero se trazaría con un aspecto similar al que se puede obtener en el QL. Obsérvese que los valores de signos que emplea la rutina han de escribirse en complemento a dos.

En el tercer listado pueden verse ejemplos de ambos tipos de dibujo.

```
                ORG  60000
D9              EXX
E5              PUSH HL               ; guarda H'L'
D9              EXX
0658            LD   B,88             ;coordenada Y
0E64            LD   C,100            ;coordenada X
CDE522          CALL #22E5            ;PLOT 100,08
0E28            LD   C,40             ;ABS x=40
0614            LD   B,20             ;ABS y=20
1EFF            LD   E,255            ;SGN y=−1
1601            LD   D,1              ;SGN x=1
CDBA24          CALL #24BA            ;DRAW −40,20
3E58            LD   A,88
327D5C          LD   (23677),A        ;X org=88
3E32            LD   A,50
327E5C          LD   (23678),A        ;Y org=50
0E28            LD   C,40
0614            LD   B,20
1602            LD   D,2              ;SGN x=2 línea de puntos
1E02            LD   E,2              ;SGN y=2 línea de puntos
CDBA24          CALL #24BA
D9              EXX
E1              POP  HL               ;restaura H'L'
D9              EXX
C9              RET
```

La información que ofrecemos en dos cuadros sintetiza todo lo que hemos estudiado en esta parte sobre gráficos y manejo de la pantalla. El primer cuadro contiene una lista de las rutinas más útiles que hemos empleado, y el segundo muestra las variables de sistema más comúnmente relacionadas con esas rutinas.

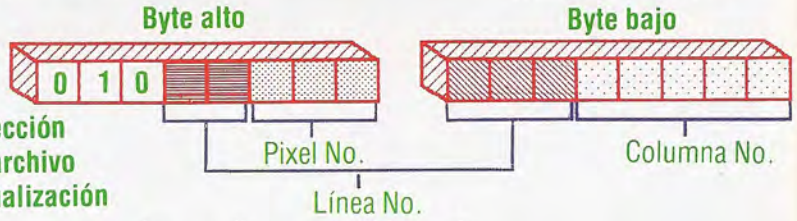
El archivo de visualización

La memoria de pantalla del Spectrum se organiza en dos partes. La primera llamada *archivo de visualización (display file)* se emplea para retener toda la información sobre los pixels actuales. La segunda parte, el *archivo de atributos* (*attributes file*), almacena la información sobre el color en la pantalla. El archivo de visualización va desde 16384 hasta 22527, y comprende un total de 6144 bytes. Escriba este programa y ejecútelo:

```
10 FOR F=16384 TO 22527
20 POKE F,255
30 NEXT F
```

A medida que el programa se va ejecutando, verá cómo la pantalla se llena lentamente. Mire más detenidamente y verá que el relleno se divide en tres zonas. Y apurando la observación de la pantalla podrá observar las líneas que se van dibujando en ese momento. Contrariamente a lo que se podría esperar, estas líneas no se dibujan consecutivamente. La primera línea de pixels aparece en la parte superior de la pantalla, pero la línea siguiente no aparece inmediatamente debajo, sino 8 líneas más abajo, y la tercera aparece 16 líneas más abajo, y así las demás. Cuando se llega a la línea 64, el proceso vuelve a situarse en la parte superior de la pantalla y se dibuja una segunda línea que se traslada a la novena línea. Todo ello se repite en la segunda zona de la pantalla y, por último, en la tercera. La figura muestra en un diagrama todo el proceso.

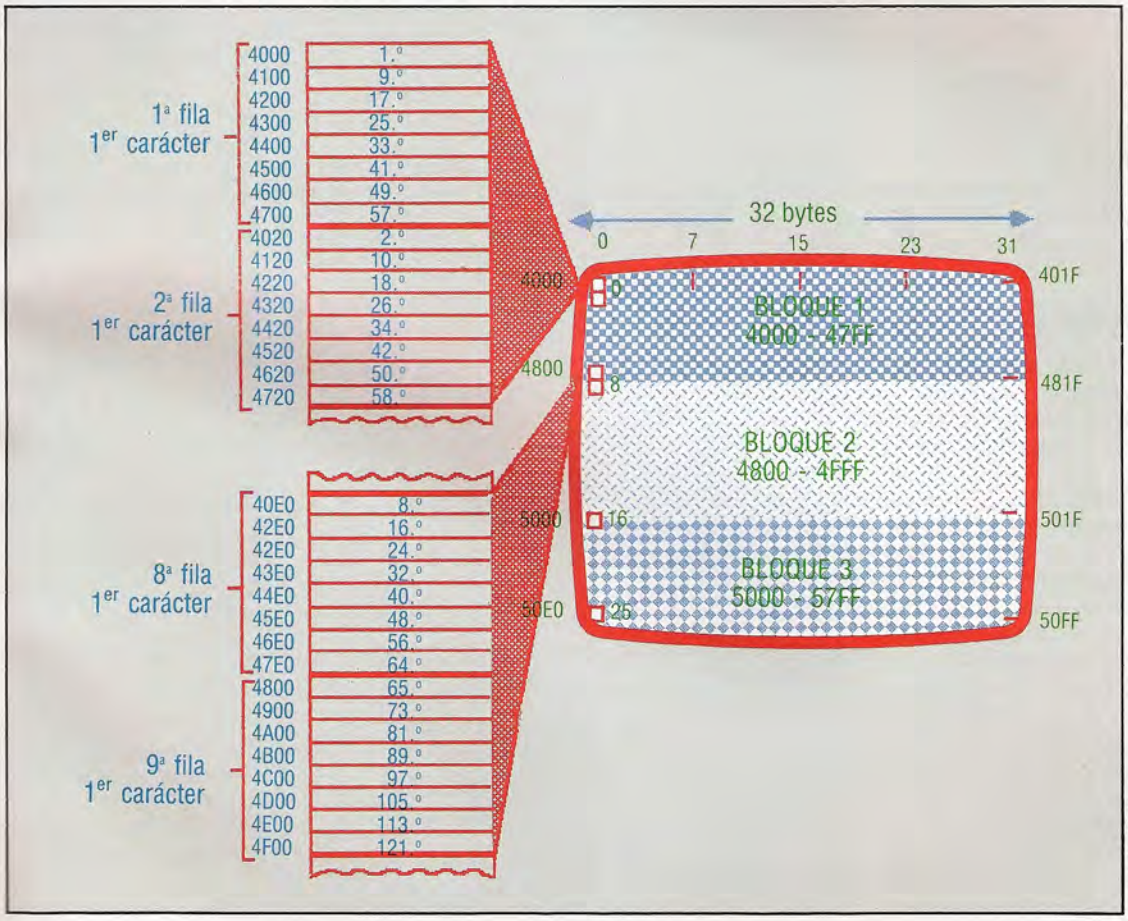
Para utilizar el archivo de visualización (abreviado, *DFile*) y calcular la dirección adonde acudir para imprimir y posicionar los pixels, obsérvese este segundo diagrama.



Direcciones útiles
El dibujo ilustra la importancia de los bits de una dirección dentro del archivo de visualización. Aunque parezca complicada, la estructura del archivo de visualización permite una rápida manipulación de los datos de pantalla empleando instrucciones de registros individuales, en lugar de las más profusas (y lentas) operaciones de empleo de registros pares

Aquí se muestra cómo se disponen los bits en la dirección del *DFile* con el número de pixel referido a la línea de pixel dentro de cada carácter. Así, suponiendo que estamos en el pixel superior de un carácter, el número de pixel sería el 0.

Este esquema muestra también el hecho de que si deseamos hacer bajar un pixel del carácter, todo lo que hay que hacer es incrementar el registro H (alto), o sea, INC H. Si queremos desplazar un carácter a la derecha, emplearemos HL. Para calcular la posición de una coordenada de PRINT AT se empleará la rutina del cuarto listado, la cual exige que el número de columna esté en E y el número de línea en D. Una vez llamada la rutina, la dirección se encontrará en HL.



Memoria de pantalla
Este esquema muestra la distribución del archivo de visualización del Spectrum. La pantalla se divide en tres bloques principales, cada uno de los cuales representa 256 posiciones de caracteres (8 líneas por 32 caracteres). Dentro de cada bloque las líneas de pixels se almacenan según el orden que se indica. Así el primer bloque de 256 bytes de la memoria de pantalla contiene datos para la primera línea de pixels de los primeros 256 caracteres, los siguientes 256 bytes contienen datos para la segunda línea, y así sucesivamente. Esto puede verse en el dibujo detallado de los cuatro caracteres en la izquierda de la pantalla, donde cada carácter tiene sus líneas de pixels numeradas con (1) la dirección correspondiente dentro del archivo de visualización y (2) el orden en que se rellenan las líneas de pixels cuando los datos son POKE consecutivamente dentro del archivo

Caroline Clayton

```
                ORG  60000
3EAF            LD   A,175
92              SUB  D
57              LD   D,A        ;D contiene 175-coord. y
A7              AND  A
IF              RRA
37              SCF
1F              RRA
A7              AND  A
1F              RRA             ;mezcla de bits
AA              XOR  D
E6F8            AND  248
AA              XOR  D
67              LD   H,A        ;H se convierte en...
7B              LD   A,E        ;64+8*INT(B/64)+B(MOD 8)

CB07            RLC  A
CB07            RLC  A
CB07            RLC  A
AA              XOR  D
CB07            RLC  A
CB07            RLC  A;         mezcla de bits
6F              LD   L,A        ;L tiene ahora 32*INT(...
7B              LD   A,E        ;...B(MOD64)/8)+INT(x/8)
E607            AND  7          ;el byte L0 es x(MOD 8)
C9              RET
```

Si deseamos calcular la dirección del pixel, utilizaremos la rutina de este quinto listado. Éste es similar al precedente, ya que también se ha de cargar D con la coordenada y, mientras que E recibirá la coordenada x. Al ser llamada esta rutina, la dirección estará en HL, que retendrá un valor entre 0 y 7, según sea el grupo de bits para el pixel adecuado.

```
                ORG  60000
7A              LD   A,D
E6F8            AND  248        ;desenmascara bits 0-2
CBF7            SET  6,A
67              LD   H,A
7A              LD   A,D
E607            AND  7          ;desenmascara bits 3-7
CB0F            RRC  A
CB0F            RRC  A
CB0F            RRC  A          ;mueve hacia arriba los
                                bits
83              ADD  A,E        ;añade bits columna
6F              LD   L,A
C9              RET
```

Un último detalle que se ha de recordar sobre el D*file* es que cada byte que examinemos (POKE) corresponde a ocho pixels. Estos pixels se almacenan del mismo modo que los almacenados en un byte de gráficos definidos por el usuario.

El archivo de atributos

La segunda parte de la memoria de pantalla (el archivo de atributos) es más inmediata que el archivo de visualización. El primer byte está en 22528 y la longitud de este archivo es de 768 bytes. Ejecute:

```
10 FOR F=22528 TO 22528+767
20 POKE F,0
30 NEXT F
```

Observará que la pantalla se llena como usted esperaba: completada la primera línea pasa al comienzo de la segunda y así hasta llenar toda la pantalla.

Cada byte en el archivo de atributos se corresponde con los valores de INK, PAPER, BRIGHT y FLASH para cada recuadro de carácter. Los tres primeros bits contienen el color de INK, y los bits 3, 4 y 5 el color de PAPER. El valor adecuado de BRIGHT está en el bit 6 y el de FLASH en el bit 7.

Una observación final sobre las alteraciones de color de BORDER. Para cambiar de color el recuadro desde código máquina, A se cargará con el número de color y se llamará #229BH.

Variables del Spectrum

Nombre variable	Posición	Descripción
COORDS	23677	Contiene la coord. x del último punto trazado
COORDS	23678	Contiene la coord. y del último punto trazado
DFCC	23684	Es una variable de dos bytes que contiene la dirección de memoria en el archivo de visualización del primer byte que se empleará para imprimir
S POSN	23688	Contiene la posición actual de impresión para columnas
S POSN	23689	Contiene núm. de línea actual utilizado para impresión
ATTR P	23693	Se usa para almacenar los colores permanentes establecidos por INK, PAPER, FLASH y BRIGHT. Los datos están en el formato ATTR
MASK P	23694	Se usa para desenmascarar ATTR P cuando se emplean colores transparentes (INK9, etc.). Cada bit encendido en el correspondiente bit de ATTR P será ignorado
ATTR T	23695	Como ATTR P, pero sólo se usa para colores temporales; p. ej., para cambios de color en una instrucción PRINT
MASK T	23696	Igual que MASK P, pero para colores temporales
P FLAG	23697	Varios flags empleados para almacenar el estado tanto temporal como permanente. Cada bit muestra si lo que ha de ser transparente es la tinta o el papel

N.B. Ya fueron descritas precedentemente ATTR P, MASK P y P FLAG

Rutinas ROM del Spectrum

Nombre rutina	Posición	Requisitos entrada	Resultado
RST # 10	#0010	A debe contener el código ASCII del carácter a imprimir	Se imprimirá el carácter
PR STRING	#203C	BC debe contener la longitud de la cadena; DE contendrá la posición de la cadena en la memoria	La cadena se imprimirá en la posición sig. disponible en pant.
PRINTED FP	#2DE3	El número a imprimir debe estar encima de la pila de cálculo	Se imprime el núm. sup. en la pila calculadora
STK-A	#2D28	A contendrá el valor a colocar en la pila	El valor en A se sitúa encima de la pila calculadora
STK-BC	#2D2B	BC debe contener el valor a colocar en la pila	El valor en BC se sitúa encima de la pila calculadora
PLOT	#22E5	B y C contendrán respectivamente las coordenadas x e y	Se traza el punto (x, y) y se actualizan las COORD SV
DRAW	#24BA	B contendrá ABS(y) C contendrá ABS(x) D contendrá SGN(y) E contendrá SGN(x)	Si los valores son mayores que 1 la línea será de puntos
CHAN-OPEN	#1601	A debe contener el número del flujo que se va a usar	Abre el canal adecuado

El poder del CAD

El diseño asistido por ordenador (CAD) ya se utiliza ampliamente en la industria. Examinemos esta eficaz técnica

La gran diferencia que distingue a los gráficos por ordenador elementales del auténtico CAD radica en que este último está relacionado con mucho más que el mero dibujo del objeto. Los sistemas CAD más sofisticados poseen un sentido real del objeto alojado en sus memorias y no una simple imagen del mismo. Algunos sistemas son capaces de probar el objeto, montando los componentes, simulando tensiones y fuerzas y asegurando que el producto final no se derrumbe una vez construido.

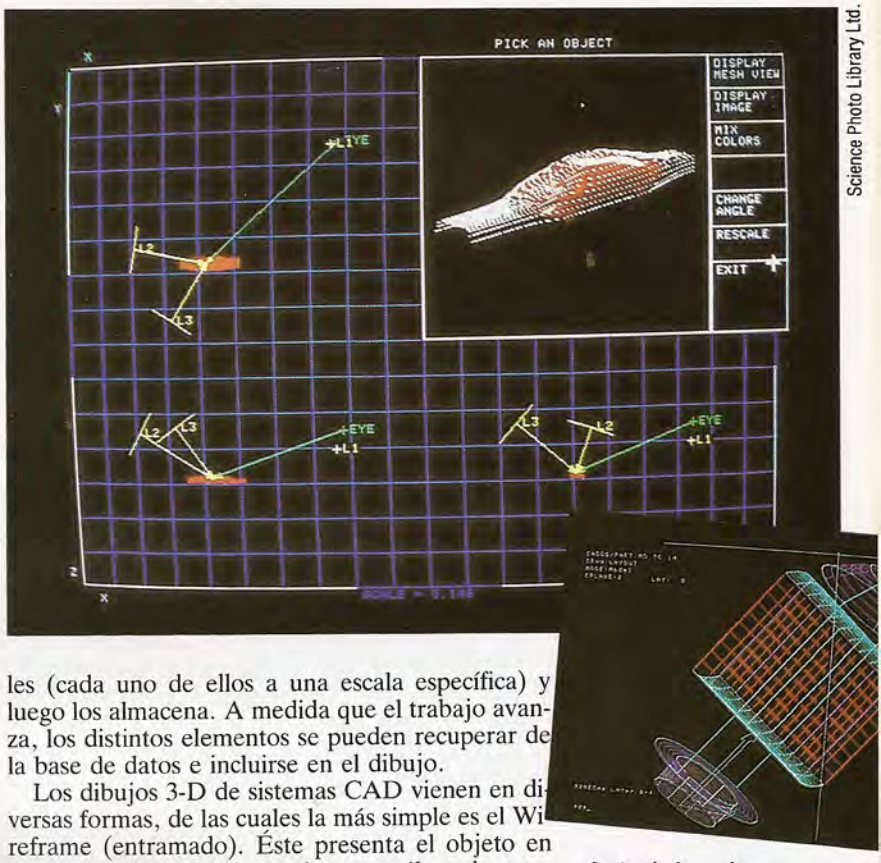
Las imágenes de la mayoría de los sistemas CAD se crean en realidad de forma diferente a las de los sistemas gráficos normales. Los gráficos por ordenador comunes construyen las imágenes con pixels, pero éstos no son lo suficientemente precisos para el CAD, que utiliza, en cambio, el sistema de gráficos de vectores. Éste basa el dibujo en la descripción matemática de las formas. Por lo tanto, en los gráficos por pixels una línea sería una fila de puntos en la pantalla, mientras que en un gráfico por vectores se entendería como las coordenadas de ambos extremos y la distancia más corta entre ambos. Los gráficos por vectores son más exactos que los gráficos por pixels, permitiendo que el procesador los trate con resoluciones mucho mayores de las que puede visualizar el monitor.

Los sistemas CAD más sencillos dibujan en dos dimensiones y en ocasiones pueden añadir la apariencia de una tercera, cosa que generalmente se conoce como dibujo 2 1/2-D (en dos dimensiones y media). Tal sistema puede satisfacer las necesidades de un estudio de arquitectura, donde la «media» dimensión adicional permite que el arquitecto muestre un alzado además de una planta.

Las empresas que aplican el CAD para diseñar sistemas de circuitos electrónicos también utilizan sistemas 2-D, pero de mayor complejidad. Debido a que los sistemas CAD pueden retener información no sólo acerca de un dibujo sino también acerca del producto real, en el campo de la electrónica, por lo general el CAD va acoplado con programas de prueba; una vez que el diseñador de un chip ha elaborado el mapa de los millones de circuitos de un microprocesador, el sistema comprobará que el conjunto de circuitos funcione correctamente.

El uso más complejo del CAD concierne al diseño de objetos para su fabricación. Requiere 3 dimensiones y, por tanto, necesita más memoria y un procesamiento más rápido con el fin de tratar la imagen. Dado que los diseños se han de convertir en productos acabados, ha de ser muy preciso.

El diseñador que utiliza un sistema CAD primero elabora los mapas de los componentes individuales (cada uno de ellos a una escala específica) y luego los almacena. A medida que el trabajo avanza, los distintos elementos se pueden recuperar de la base de datos e incluirse en el dibujo.



Science Photo Library Ltd.

Anatomía de una imagen
El CAD aplica la tecnología de gráficos por vectores para manipular imágenes y visualizarlas en muy alta resolución, como podemos ver en esta instantánea de pantalla de un ensamblaje por soldadura diseñado utilizando técnicas CAD. Los sistemas avanzados, sin embargo, también permiten probar los diseños antes de su fabricación. En la imagen inferior, un superordenador Cray comprueba la aerodinámica de un nuevo vehículo en la General Motors

Los dibujos 3-D de sistemas CAD vienen en diversas formas, de las cuales la más simple es el Wireframe (entramado). Éste presenta el objeto en tres dimensiones en el espacio, pero sólo en boceto. No hay ninguna imagen de las superficies. La 3-D completa y el modelado de sólidos proporcionan las superficies, y es el sistema en sí mismo el que tiene un sentido del objeto tal como es en realidad.

Los sistemas más complejos pueden producir imágenes coloreadas, con hasta 256 colores diferentes en pantalla al mismo tiempo. El sombreado suele añadir profundidad y contribuye a ofrecer una imagen más realista.

Los usuarios de CAD tienen a su disposición una gama de efectos cuando forman sus imágenes, además de la capacidad para manipular formas geométricas. He aquí una relación de los más importantes:

Dimensionado: Una vez terminado el dibujo, el diseñador puede hacer que el sistema añada automáticamente todas las cotas de las medidas y lo deje preparado para dibujar. Dado que el sistema retiene información en forma numérica, ésta es una operación directa.

Plumeado: Los diseñadores necesitan realzar secciones. El plumeado es una técnica de sombreado automático que utilizan los artistas técnicos.

Estratificación: Los diseños se pueden formar por partes. Quizá un arquitecto desee conservar la planta de una casa en una parte y los desagües y tuberías en otra. Ambos podrían aparecer en la misma salida impresa o bien por separado, según cual sea la información que se necesite.

Segmentos: En ocasiones se denominan *entidades* y son los elementos individuales que componen un dibujo. Se retienen como una especie de catálogo de dibujos ya hechos que puede reclamar el diseñador.

Banda elástica: Esta técnica permite que el diseñador estire las líneas o las acorte (como si fueran

elásticas) para colocarlas allí donde las necesita.
Escalado: Permite que el diseñador dibuje en el tamaño que desee y después almacene la información a escala con el resto del dibujo.
"Zoom" y "pan": El primero de estos dos términos cinematográficos define la acción de enfocar una sección del dibujo y ampliarla hasta llenar toda la pantalla. El *pan* permite al diseñador volver hacia atrás, como estaba antes, y contemplar el dibujo en su totalidad.

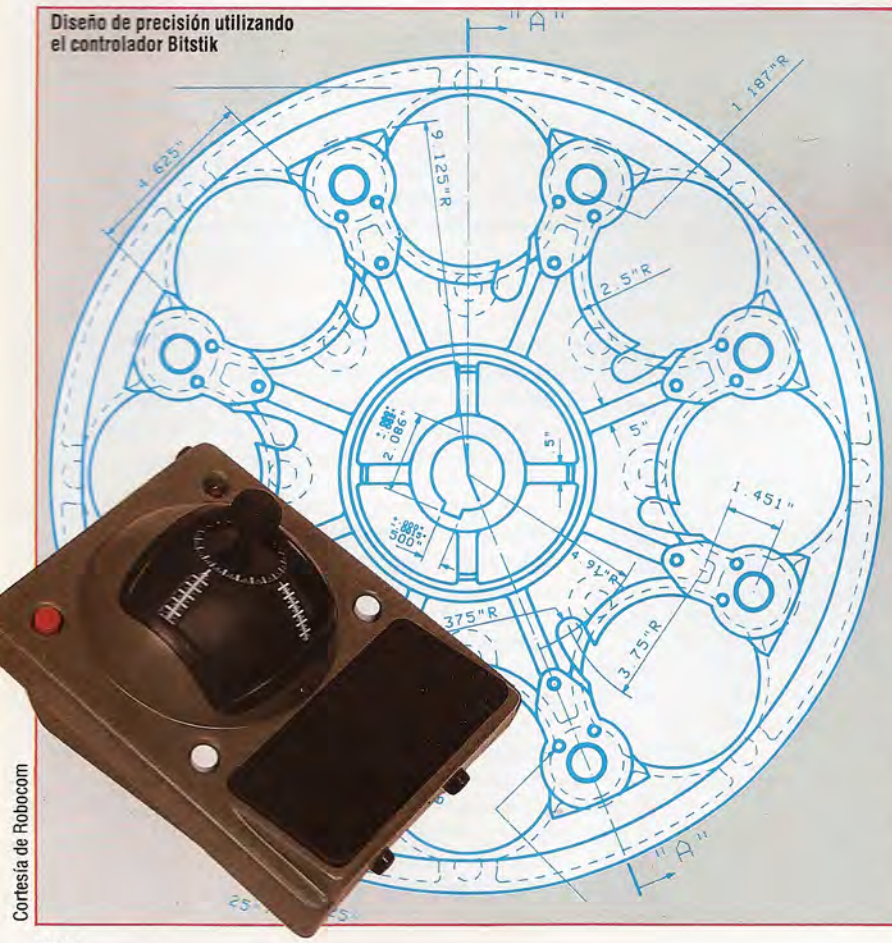
Los usuarios de CAD tienen una gama de dispositivos que pueden emplear según el tipo de trabajo que estén realizando. Entre los más corrientes están la tablilla digitalizadora, el lápiz óptico, la bola seguidora y el ratón.

Los procesadores de imágenes se utilizan para diseños sumamente complejos y fotografías. Trabajan fotografiando la imagen a introducir y descomponiendo la fotografía en los elementos sombreados que componen una imagen de pantalla.

El teclado se emplea frecuentemente para introducir información en sistemas CAD, puesto que es una forma de gran precisión para entrar las coordenadas aplicadas en los gráficos de vectores.

En CAD también se utilizan palancas de mando, pero son mucho más sofisticadas que las que se emplean para los juegos. El Bitstik, de Robocom, posee potenciómetros de precisión que miden el movimiento de la palanca con seis lugares decimales. La palanca controla un cursor en la pantalla y, mediante pulsadores situados en la caja, se puede utilizar para seleccionar funciones del menú. El extremo del brazo del Bitstik se puede regular para efectuar un *zoom* o un *pan*.

Diseño de precisión utilizando el controlador Bitstik



Cortesía de Robocom

CAD en el micro

El diseño asistido por ordenador (CAD) está ganando creciente popularidad entre los diseñadores como herramienta fiable y eficaz desde el punto de vista del costo, para ayudar a producir diseños originales, o bien para probar posibles diseños sin necesidad de construir costosos modelos. Aunque el CAD requiere una gran cantidad de equipos y periféricos para obtener resultados profesionales, muchos de los sistemas que existen actualmente se basan en micros personales bastante modestos. El sistema HRX, para el ordenador Memotech 512, es un paquete que permite mejorar con ordenador las pantallas obtenidas con una cámara de video. Los "fotogramas" de la cámara de video pueden ser leídos por el sistema de ordenador y se pueden obtener numerosos efectos. Entre éstos se incluye la posibilidad de realizar *zooms*, contraer o hacer rotar la imagen. También se pueden cambiar completamente los colores aplicados a la imagen. El sistema de color que utiliza el formato HRX emplea los tres colores básicos de un sistema de video (rojo, verde y azul) y tiene una capacidad de más de 16,5 millones de colores por pixel. Un aparato de televisión normal sólo requiere una capacidad de color de 250 000. Por supuesto, para funcionar eficazmente, un sistema de este tipo requiere muchísimo hardware adicional. Con la necesidad de procesar tanta información, es obvio que se requiere una gran cantidad de memoria. El usuario, en consecuencia, necesita un controlador de doble anchura y tarjeta de memoria, con tarjetas de memoria adicionales si se ha de añadir color. Además, se requiere un convertidor A/D *flash* especial, simple (monocromático) o de tres canales (color) para aceptar información proveniente del ordenador MTX. Un sistema completamente equipado almacenará los datos de una cámara de video a la velocidad de 7,2 Mbytes por segundo. Obviamente, un sistema que ofrezca tal potencia es bastante caro. Mucho más asequible es el sistema Bitstik, para el BBC Micro. Este paquete permite que la máquina produzca dibujos técnicos de calidad profesional. El sistema consta de un controlador Bitstik (una especie de palanca de mando de gran precisión), un disco de sistemas, una ROM de aplicaciones de 8 K y un disco biblioteca que contiene numerosos tipos y símbolos comunes. Sin embargo, para poder operar el sistema, también se ha de dotar al micro de un segundo procesador 6502 y de unidades de disco gemelas de 80 pistas. Para poder sacar el máximo partido del sistema, el usuario también necesita un monitor en color y un plotter en color. De esta manera, el usuario obtiene un sistema que, según afirman los fabricantes, permite que alguien que carezca de formación artística produzca diseños gráficos estándares a nivel profesional. La base del sistema es la biblioteca de formas contenidas en disco y las "primitivas", como líneas y arcos, contenidas en ROM. Los diagramas se forman construyendo diseños más complejos a partir de estas primitivas, que luego se pueden incorporar a diseños mayores. La parte central del sistema es el propio controlador Bitstik, que permite posicionar las primitivas con gran precisión

Haciendo planos en etapas simples

Diseño escenográfico utilizando el Matchbox

Dibujo arquitectónico detallado producido con el Matchbox

Si bien el CAD suele asociarse al desarrollo de productos industriales, como automóviles, las técnicas utilizadas se pueden adaptar a casi cualquier aplicación en la que se requiera "modelado". Uno de los usos que se ha dado al CAD en un entorno no industrial es en el teatro. Una de las principales preocupaciones del escenógrafo es disponer la escenografía para que produzca el efecto visual requerido, sin entorpecer la visión del público. El problema es complicado para las producciones itinerantes que van de teatro en teatro, cada uno de los cuales tiene su propio "punto de vista". Tradicionalmente este problema se ha resuelto mediante la creación de modelos del teatro con reproducciones a escala de la escenografía, que se desplaza a través del escenario hasta hallar la disposición óptima. Sin embargo, un equipo de diseñadores de teatro del Theatre Royal, Stratford, en el East End de Londres, ha colaborado con los asesores de diseño de Robary Ltd. para producir el Matchbox, un paquete que permite crear escenografías en una VDU y contemplarlas desde cualquier ángulo dentro de cualquier teatro determinado. Las ventajas del sistema son que un diseñador puede crear una escenografía y pasarla por todas las vistas y teatros en los que se utilizará. Si la escenografía no se adapta a un teatro dado, se puede modificar, desplazando la utilería por el escenario de forma instantánea, en vez de tener que volver a construir los modelos a partir de cero. Además, el sistema Matchbox proporciona planos y dibujos de precisión, a partir de los cuales pueden trabajar los diseñadores y constructores

Theatre Royal, Stratford East

Hardware CAD

No es sólo el software lo que se elabora a la medida para su empleo en aplicaciones CAD. Un indicio de la importancia que se le otorga al diseño asistido por ordenador es el hecho de que también se están desarrollando algunos ordenadores que le otorgan preeminencia a las necesidades del CAD. El MG-1 Workstation, de Whitechapel Computer Works, es un miniordenador que contiene numerosas configuraciones esenciales para el procesamiento eficaz del CAD. El MG-1 se basa en el procesador de 16 bits National Semiconductor NS32016, que proporciona el tipo de potencia de procesamiento veloz que requiere el CAD. La visualización de gráficos tiene una resolución de 1024 × 800 pixels y, con el fin de producir imágenes rápidas sin parpadeo, incluye numerosas características avanzadas. La máquina está equipada con refresco de memoria directo, con lo que se evita la necesidad de tampones de fotogramas, que reducen la velocidad a la cual se refresca la pantalla VDU. Además, con el objeto de reducir el problema del movimiento desigual del cursor con el ratón, hay un procesador separado para manipular el ratón y la entrada por teclado. El MG-1 opera bajo el GENIX. Ésta es la versión de National Semiconductor del popular OS multitareas Unix. Se han desarrollado numerosos paquetes para correr en el MG-1 bajo este sistema. Por ejemplo, la firma Payfec ha producido un sistema de dibujo profesional, denominado DOGS, mientras que Lattice Logic ha desarrollado un paquete de diseño de procesadores denominado Chipsmith

Cortesía de Whitechapel Computer Centre

Identificar las señales

Proseguimos el detallado examen de las patillas individuales del chip convertidor 7135

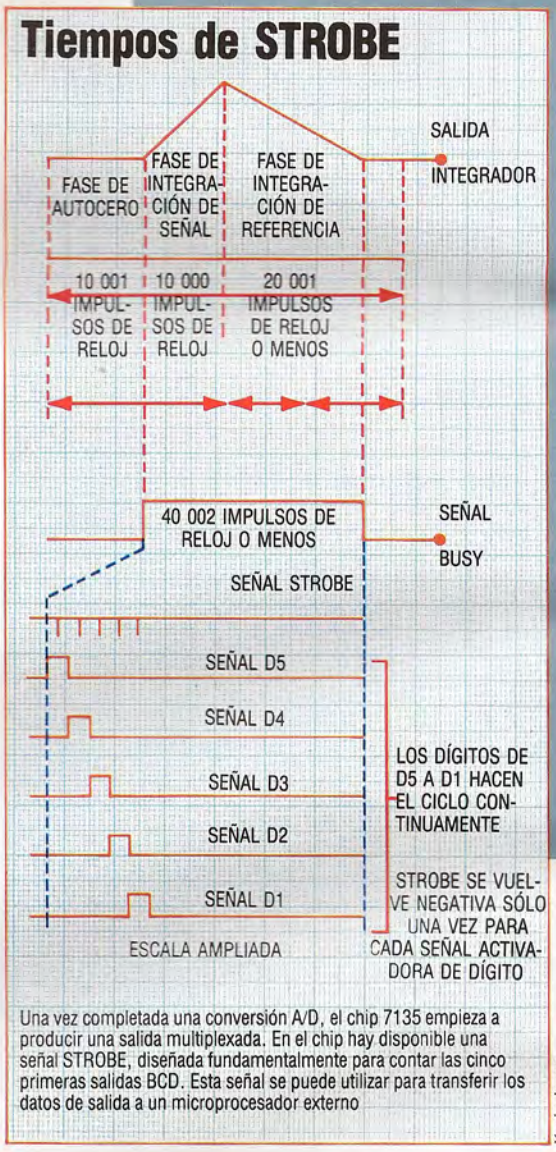
7/8. REF. CAP. (Condensador de referencia): Sorprendentemente, el valor y la calidad de este componente no son muy críticos. Se sugiere un condensador de 1 µF, aunque se podría utilizar uno más pequeño (al precio de una pérdida de eficacia durante los primeros segundos de recuperación tras una sobrecarga). Se han de emplear condensadores de polipropileno, de policarbonato o de poliestireno.

9/10 IN–/IN+: Se dispone de dos entradas así como una masa analógica (patilla 3). Se pueden acomodar entradas diferenciales en cualquier punto entre la escala –4 V y +4,5 V. Estas entradas se pueden dejar flotantes o se puede unir IN– al punto de masa analógica.

11. V+: Para alimentar el chip se requiere una alimentación de +5 V a una corriente máxima de 3 mA.

12. D5: Hay cinco salidas de habilitación de dígitos, de D1 a D5. (Las otras salidas están en las patillas 17, 18, 19 y 20.) Se utilizan para habilitar una por una las visualizaciones LED de siete segmentos, de modo que se puedan multiplexar. Cada salida de habilitación de dígitos es una señal activa positiva que permanece alta durante 200 impulsos de reloj. Los cinco dígitos se exploran continuamente, comenzando por D5, el dígito más significativo (MSD) y terminando en D1, el dígito menos significativo (LSD). A medida que cada salida habilitadora de dígito se hace activa, la tensión se eleva desde el nivel casi de masa (aproximadamente 0,25 V) hasta casi la tensión de la línea positiva (alrededor de 4 V). Cuando sucede esto, el transistor al cual está aplicada se hace conductor y permite que circule corriente por el dígito correspondiente (ver esquema del circuito). Las señales activadoras de dígitos sólo *habilitan* cada una de las visualizaciones LED de siete segmentos, al conectar el ánodo común a la línea de + 5 V. Los segmentos individuales de la visualización se encienden mediante las señales BCD (B1, B2, B4 y B8), decodificadas por el chip activador/decodificador de siete segmentos, el 7447. En nuestra serie sobre lógica hemos visto circuitos lógicos que convierten una entrada BCD de cuatro líneas en una salida de siete segmentos que ilumina las barras adecuadas de una visualización de siete segmentos.

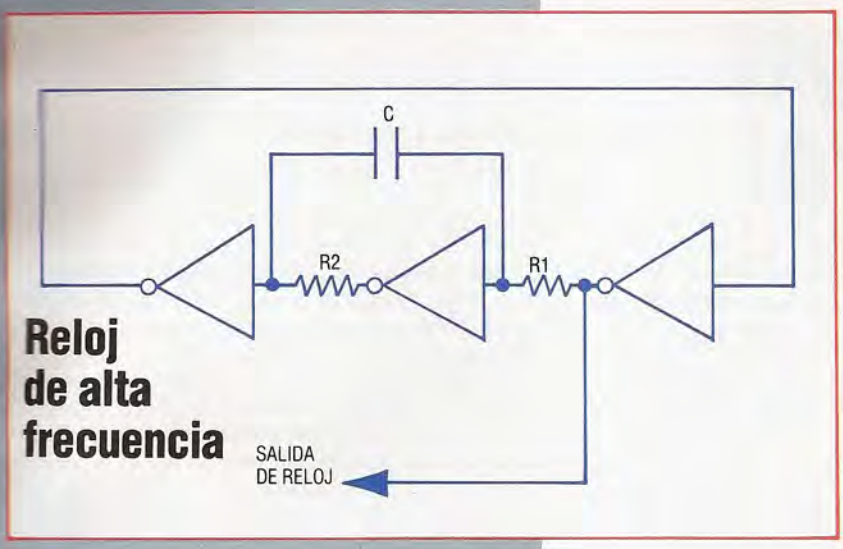
13. B1 (LSB): Hay cuatro líneas de salida BCD (*binary coded decimal:* decimal codificado en binario): B1 en la patilla 13, B2 en la patilla 14, B4 en la patilla 15 y B8 (el MSB) en la patilla 16. Entre ellas



Una vez completada una conversión A/D, el chip 7135 empieza a producir una salida multiplexada. En el chip hay disponible una señal STROBE, diseñada fundamentalmente para contar las cinco primeras salidas BCD. Esta señal se puede utilizar para transferir los datos de salida a un microprocesador externo

Kevin Jones

proporcionan un equivalente codificado en binario del dígito decimal a visualizar en cualquier momento. El código BCD producido será el apropiado para el dígito que esté habilitado entonces. Las señales habilitadoras de dígitos del chip están sincronizadas de manera que el valor BCD para cualquier dígito determinado esté presente en las líneas de salida cuando la correspondiente señal habilitadora sea alta. Los cinco dígitos parecerán estar todos encendidos al mismo tiempo. En realidad están multiplexados, encendiéndose de forma intermitente en secuencia, uno después de otro. Esto da una intensidad de salida luminosa aparentemente mucho menor que si los LED estuvieran iluminados continuamente. Sin embargo, no es práctico encender las cinco visualizaciones al mismo tiempo. Debido a su gran consumo, los LED requerirían casi cinco



Reloj de alta frecuencia

Fijación del tiempo
Esta alternativa al chip de reloj 555 utiliza apenas tres inversores TTL, dos resistencias y un condensador. La fórmula para calcular la frecuencia deseada es:

$F = 2C(R1 + R2)$

Puesto que el chip convertidor 7135 tolera velocidades de reloj relativamente altas, este tipo de circuito de reloj se podría utilizar para sincronizar su acción en caso de que se necesitaran mediciones de alta frecuencia y de poca precisión

veces la energía de un circuito con un solo LED encendido cada vez. Aun más importante es el hecho de que se requerirían salidas BCD separadas para cada dígito (20 patillas en lugar de 9, y cada visualización requeriría su propio chip activador de visualización 7447: cinco chips en lugar de uno). Y la única ventaja sería una visualización más brillante. El multiplexado permite una gran reducción de los costos de cableado y de componentes.

21. BUSY: Ésta es una línea importante si desea conectar el DVM en interface con su micro. Es una señal *alta activa,* lo que significa que, cuando es alta, el convertidor A/D está ocupado (*busy*) haciendo cálculos, y que el dígito y las salidas BCD no tienen ningún significado real y se deben ignorar. Se pone alta al principio de la fase de integración de señal, y permanece alta hasta el primer impulso de reloj después del punto paso por cero. Esto significa que cuando la señal BUSY se hace baja, la salida de señal está fijada en un estado estable y puede ser leída válidamente por, pongamos por caso, un microordenador externo. En anteriores capítulos ya hemos analizado este principio.

22. ENTRADA DE RELOJ: Se utiliza una fuente de reloj externo para disparar las operaciones internas del chip 7135. Esta señal de reloj no debe ser superior a 1,2 MHz. Esta frecuencia máxima proporciona una actualización sumamente rápida de la tensión que se esté midiendo, pero una frecuencia tan elevada puede crear muchos problemas relacionados con la selección de componentes y la precisión. En cualquier caso, el chip 555 que utilizaremos no puede generar tales frecuencias: no subirá por encima de 100 KHz.

Dado que un ciclo de medición completo emplea un máximo de 40 002 impulsos de reloj, la frecuencia del reloj determinará cuántas mediciones se harán por segundo. Un reloj de 100 KHz efectuará aproximadamente 2,5 mediciones por segundo. Si usted necesita mediciones más rápidas, son factibles frecuencias de reloj de hasta 1,2 MHz, pero necesitará un circuito oscilador de reloj alternativo y condensadores de autocero y de referencia de muy alta calidad. Es probable que también la precisión se vea afectada. La flexibilidad del chip 7135 se pone de relieve por el hecho de que puede trabajar en una gran gama de frecuencias, incluso de 1 Hz, si bien, en este caso, ¡tendría que esperar 40 002 segundos para una lectura!

23. POLARIDAD: Esta patilla produce una señal que indica simplemente la polaridad de la tensión que se está midiendo. Es positiva para todas las tensiones de entrada positivas, aun para aquellas tan bajas que la visualización indique +00000. Se utiliza para encender el signo "+" en la visualización LED del ½ dígito situado más a la izquierda.

24. MASA DIGITAL: Aquí es muy poco lo que se puede decir que no se haya dicho ya al hablar de la patilla 3, masa analógica.

25. RUN/HOLD (correr/retener): Cuando esta patilla de entrada es alta, el 7135 está en modalidad "funcionamiento libre", dando lecturas de salida cada 40 002 impulsos de reloj o menos. Si se hace baja (ya sea mediante una señal HOLD desde un ordenador, o bien mediante un interruptor del panel del DVM), la lectura actual se mantendrá con carácter indefinido. Tras esta retención, una señal positiva en esta patilla que dure al menos 300 nanosegundos comenzará un nuevo ciclo de medición. Esta patilla sería útil para una interface de ordenador en paralelo lenta, si no se pudieran procesar con rapidez suficiente las cinco salidas BCD.

26. STROBE (impulsos para sincronismo): Hay cinco impulsos STROBE que se hacen negativos (uno para cada dígito válido) en cada ciclo de medición. Están temporizados para producirse en medio de cada impulso de salida de habilitación de dígito (ver gráfico de tiempos). La señal STROBE se proporciona para simplificar la conexión en interface a microprocesadores y se puede utilizar para transferir los datos BCD a registros de retención externos. La señal STROBE se hace verdadera (negativa) sólo durante medio impulso de reloj, y solamente una vez para cada dígito. Los activadores de dígitos y las salidas BCD continúan efectuando ciclos durante el ciclo de medición actual (para permitir la aparición de una visualización continua), pero una interface de ordenador en paralelo tendría que captar los impulsos STROBE cuando se produjeran, ya que no se producirán más hasta el siguiente ciclo de medición (ver gráfico de tiempos).

27. SUPERACIÓN DE MARGEN: Esta señal indica que la tensión de entrada que se está midiendo es demasiado elevada (más de 20 000 puntos). En un diseño más sofisticado, se podría utilizar como parte de un circuito de escala automática (un circuito que estableciera automáticamente la sensibilidad de la entrada sin el uso de conmutadores mecánicos). En un circuito más simple como el nuestro, se podría utilizar para iluminar un LED que indicara que el atenuador de entrada se habría de situar en una sensibilidad inferior (una escala más alta).

28. MARGEN INSUFICIENTE: Esta señal se hace alta cuando la tensión de entrada que se está midiendo es demasiado baja. Al igual que la señal SUPERACIÓN DE MARGEN, está pensada básicamente para emplearse en un circuito de escala automática. Los DVM de escala automática utilizan estas señales para conmutar el atenuador de entrada a una sensibilidad adecuada para la señal que se esté midiendo, pero la escala automática incrementa la complejidad del diseño y en nuestro circuito no utilizaremos esta señal.

Ensayo general

Una vez situado cada personaje en nuestra representación, vamos a manejar sus atributos

Por lo general se puede reconocer un programa de aventuras escrito en BASIC: un apiñamiento de sentencias IF. En un libro muy conocido sobre programación de aventuras en BASIC se incluye un programa de muestra que incorpora no menos de cuatro páginas de sentencias IF y THEN, lo que podría inducir a la conclusión de que este método para tomar decisiones es inevitable cuando se escribe en BASIC.

En las aventuras el problema surge a raíz de la gran cantidad de variables que necesitamos procesar con el fin de decidir la acción a tomar. Por ejemplo, cada uno de los personajes del Dog and Bucket tiene 10 atributos, y nuestro manipulador de personajes necesita poder juzgar y actualizar cada uno de estos atributos cuando manipula a la "persona" en cuestión. Durante la ejecución del manipulador podríamos, por ejemplo, tener una rutina que opere en líneas generales del siguiente modo: "IF el personaje X está en la sala de tertulia AND el personaje Y está en la sala de tertulia AND IF el personaje Y sostiene la bebida del personaje X AND el personaje X sostiene algo, THEN el personaje X le arroja el objeto al personaje Y".

Si programásemos en nuestro juego esta rutina y otras similares, parecería que fuéramos a tener montones de sentencias IF, tanto si lo quisiéramos como si no. Esto no es de desear, ¡sin olvidar el enorme trabajo que tendríamos para entrar el listado! Lo que se necesita, por lo tanto, es una forma ya sea de reducir drásticamente la cantidad de IFs, o bien (lo que sería mejor) arreglárnosla sin ellas directamente, porque, al contrario de lo que se pueda suponer, *sí* existen alternativas viables a la infinidad de sentencias IF. Estas alternativas no sólo son más compactas, sino que también tienden a ser más rápidas en la ejecución, de modo que los beneficios son incuestionables.

Con el objeto de listar las diversas posibilidades de que disponemos, veamos un programa sencillo que nos permita establecer una serie de condiciones, comprobarlas y actuar en consecuencia. Luego examinaremos distintas maneras de codificarlas y decidiremos cuál es el mejor método a adoptar para nuestro manipulador de personajes. Entre en primer lugar el "módulo subrutinas de bajo nivel" (líneas 4000-4140) que dimos en el capítulo anterior (y con los "complementos" para BBC, Commodore o Spectrum), y después digite esto:

```
10 h$="humano ":a$="animal ":m$ "macho
   ":f$="hembra ":q$="Eres un ": DIM c(4)
20 GOSUB 4050: REM limpiar la pantalla
30 REM establecer las cuatro variables
40 PRINT q$;h$;:INPUT i$:c(1)=ABS(i$="s" OR
   i$="S")
50 PRINT q$;a$;:INPUT i$:c(2)=ABS(i$="s" OR
   i$="S")
60 PRINT q$;f$;:INPUT i$:c(3)=ABS(i$="s" OR
   i$="S")
70 PRINT q$;m$;:INPUT i$:c(4)=ABS(i$="s" OR
   i$="S")
80 PRINT
```

La línea 10 de este módulo primero establece las variables en serie utilizadas en las líneas 40-70, y luego DIMensiona una matriz para cuatro elementos. Esta matriz se emplea luego para almacenar cuatro valores condicionales, según las respuestas dadas a las preguntas formuladas. Para la mayoría de los micros, los valores condicionales son 0 para falso y −1 para verdadero (si bien el Spectrum utiliza 1). La condición negativa explica por qué hemos incluido ABS en cada asignación.

Al ejecutar (RUN) este módulo, se creará la matriz c(4), lista para la comprobación ulterior. Sin embargo, es en este punto donde surge el problema, ya que con cuatro condiciones diferentes hay 16 posibles combinaciones distintas. Si usted sólo quisiera probar un caso (si, p. ej., el usuario hubiera entrado "S" a la primera pregunta y "N" a todas las otras) fácilmente podría utilizar una construcción IF...THEN. No obstante, si quisiera probar, pongamos por caso, 15 de las 16 combinaciones posibles, podría acabar con 15 líneas así:

```
90 REM Imprimir mensajes
100 IF c(1)=1 AND c(2)=1 AND c(3)=1 AND
    c(4)=1 THEN PRINT "Eres un hombre lobo
    mixto!"
110 IF c(1)=1 AND c(2)=1 AND c(3)=1 AND
    c(4)=0 THEN PRINT "Eres un hombre lobo
    hembra!"
```

...y así sucesivamente. Esto, obviamente, resulta más bien largo, y hay otros métodos que podemos considerar.

El primero es usar la construcción ON...GOTO (si su micro la tiene; la mayoría disponen de ella), pero ésta también tiene sus limitaciones, porque requiere una secuencia de valores y tiende a ser bastante inflexible. Sin embargo, por regla general, es una alternativa muy poco utilizada para las sentencias IF y en algunas máquinas (como el Amstrad, p. ej.), también ofrece la ventaja de una ejecución más rápida.

El segundo método a investigar es de hecho el que adoptaremos para nuestro manipulador de personajes. Supone replantearse el problema en cierta medida y nos proporcionará un potente medio para clasificar distintas combinaciones de condiciones a medida que vayan surgiendo en el Dog and Bucket. La respuesta reside en la implementación de "árboles de decisión" para nuestros personajes.

Retrocedamos a las cuatro condiciones que establecimos en nuestro programa y, en vez de seleccionar las condiciones que deseamos responder mediante el empleo de sentencias IF, veamos cómo podríamos representar el proceso de decisión como una estructura arborescente. El diagrama muestra el posible aspecto de este árbol.

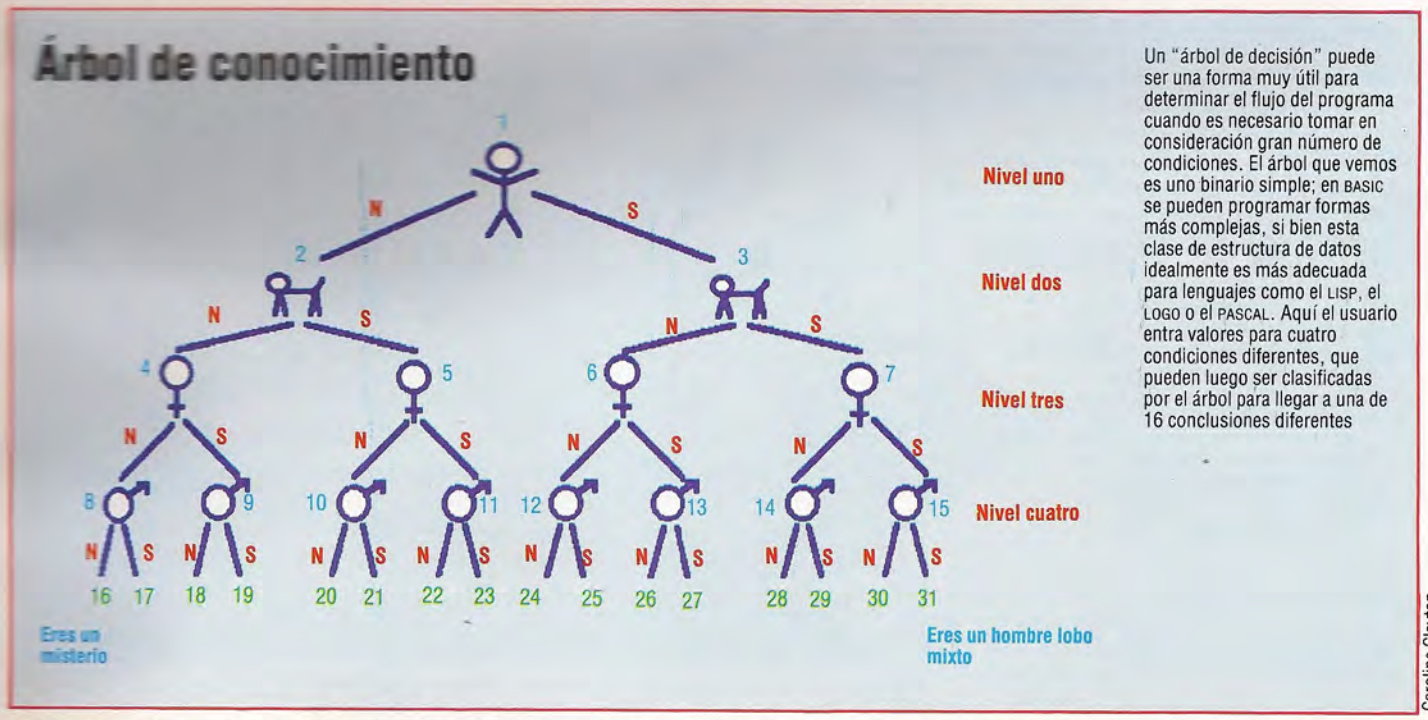
Puede ver que la primera condición a comprobar, en el nudo 1, es "¿humano?" Si la respuesta es sí (en otras palabras, si la variable de matriz

c(1)=1), entonces bifurcamos al nudo 3; de ser falsa, bifurcamos al nudo 2. Este proceso continúa hasta que llegamos a la parte inferior del árbol, uno de los "nudos terminales", numerados del 16 al 31. Cada uno de estos nudos terminales podría tener un mensaje o una rutina. Se han incluido ejemplos para el nudo 16, al que se llega cuando el usuario ha entrado una respuesta negativa a las cuatro preguntas, y para el nudo 31.

Antes de intentar calcular cómo implementar tal estructura dentro de nuestro programa, hay un par de otros puntos que debemos resaltar. En primer lugar, usted puede ver que el árbol está dividido en cuatro "niveles" y que en cada nivel todas las condiciones a probar son las mismas. El nivel 1 comprueba la condición "humano", el nivel 2 la condición "animal", el nivel 3 la condición "hembra" y el nivel 4 la condición "macho". Por el diseño, los números de nivel coinciden con los elementos de la matriz c(4), de modo que c(1) retiene la condición "humano", y así sucesivamente.

```
140 PRINT "Pulsa una tecla para continuar..."
150 GOSUB 4130:REM tomar un personaje
160 GOTO 20
170 REM datos para mensajes
180 DATA "misterio", "mineral macho?!", "mineral hembra?!", "ridículo despilfarrador de tiempo", "animal neutro", "animal macho", "animal hembra", "serpiente? Pueden ser bisexuales, ¿sabes?", "humano, sexo desconocido", "hombre", "mujer", "niño mixto", "hombre lobo neutro", "hombre lobo macho"
190 DATA "hombre lobo hembra", "hombre lobo mixto"
```

Veamos ahora cómo funciona este listado. Deseamos empezar en el nudo 1 y llevar el registro del lugar del árbol donde nos hallamos, de modo que lo primero que hemos de hacer es inicializar una variable n para representar el número de nudo actual. Luego la línea 110 hace todo el trabajo de ir desplazándonos por el árbol, nivel a nivel, acabando en uno de los nudos terminales. Esto lo hace utilizando la variable k para representar el nivel aplicando la fórmula descrita arriba, número de nudo = (2* número de nudo) + c (número de nivel), para decidir hacia qué nudo bifurcarse para ir descendiendo por el árbol.



Un "árbol de decisión" puede ser una forma muy útil para determinar el flujo del programa cuando es necesario tomar en consideración gran número de condiciones. El árbol que vemos es uno binario simple; en BASIC se pueden programar formas más complejas, si bien esta clase de estructura de datos idealmente es más adecuada para lenguajes como el LISP, el LOGO o el PASCAL. Aquí el usuario entra valores para cuatro condiciones diferentes, que pueden luego ser clasificadas por el árbol para llegar a una de 16 conclusiones diferentes

Caroline Clayton

El siguiente punto a observar es la relación entre los distintos números de nudo. Cada nudo elegido se bifurca a dos nudos inferiores, y los números de éstos se podrían determinar mediante:

número del nudo inferior = (número de nudo elegido* 2)+valor de la condición

donde el valor de la condición, retenido en la matriz c, es 0 o bien 1.

Introduzca el tercer listado y ejecute el programa, tras lo cual será saludado con un mensaje adecuado, según cuáles hayan sido sus respuestas a las cuatro preguntas.

```
90 REM ordenado del árbol
100 n=1: REM empezar por el nudo 1
110 FOR k=1 TO 4: n=(2*n)+c(k): NEXT k
120 RESTORE:REM establecer señalador de datos en comienzo mensajes
130 FOR x=1 TO n-15: READ z$:NEXT x: PRINT "Eres un ";z$: PRINT
```

Una vez que hemos recorrido los cuatro niveles y llegado a uno de los nudos del 16 al 31, es muy sencillo restarle 15 al número de nudo terminal para obtener un número entre 1 y 16 y luego, en la línea 130, leer el almacenamiento DATA y seleccionar un mensaje adecuado para imprimir. Las líneas 140-160 simplemente esperan a que usted pulse una tecla antes de repetir el proceso.

Existen otras formas en las que podríamos resolver los problemas de nuestro programa "humano/animal". No obstante, la utilidad de las estructuras arborescentes se hace evidente cuando deseamos aplicarle a nuestro procesamiento de condiciones unas reglas bastante más irregulares.

Datos básicos (VIII)

Proseguimos nuestro detallado análisis del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
BAUDOF	0299–029A	665–666	Velocidad baudios de RS232: tiempo completo del bit (µs)
RIDBE	029B	667	Índice RS232 para final buffer entrada
RIDBS	029C	668	Inicio RS232 de buffer entrada (página)
RODBS	029D	669	Inicio RS232 de buffer salida (página)
RODBE	029E	670	Índice RS232 para final buffer salida
IRQTMP	029F–02A0	671–672	Retiene Vector IRQ mientras las E/S cinta
ENABL	02A1	673	Activa RS232
	02A2	674	Sentido TOD durante entrada/salida cassette
	02A3	675	Almacenamiento temp. para lectura cassette
	02A4	676	Indicador temp. D1IRQ para lectura cassette
	02A5	677	Índice temp. para línea
	02A6	678	Flag PAL/NTSC, 0 = NTSC, 1 = PAL
	02A7–02FF	679–767	No utilizado
IERROR	0300–0301	768–769	Vector: mensaje error impresión en BASIC
IMAIN	0302–0303	770–771	Vector: inicio en caliente BASIC
ICRNCH	0304–0305	772–773	Vector: texto *tokenizado* BASIC
IQPLOP	0306–0307	774–775	Vector: listado texto BASIC
IGONE	0308–0309	776–777	Vector: despacho caracteres BASIC
IEVAL	030A–030B	778–779	Vector: evaluación *token* . BASIC

Calidad suprema

Excelente impresión
Aunque la LaserWriter está pensada para su utilización con micros, es totalmente distinta de las impresoras matriciales y de rueda margarita que se usan con la mayoría de ordenadores. Sería más acertado compararla con las veloces impresoras láser para miniordenadores y superordenadores. Si bien la LaserWriter es más lenta que sus equivalentes para ordenadores centrales, su avanzada tecnología es similar

Chris Stevens

La impresora Apple LaserWriter, gracias a la tecnología óptica y del láser, obtiene una calidad de impresión sin precedentes en ningún micro

La mayoría de las impresoras para ordenadores son de tipo matricial o bien de rueda margarita. Si bien éstas son adecuadas para la mayoría de las aplicaciones, cada una posee sus propios inconvenientes. Las impresoras matriciales o de matriz de puntos, aunque veloces y capaces de producir una enorme gama de caracteres diferentes, producen un tipo de letra que raramente resiste un cuidadoso examen. La impresora matricial promedio, que en su cabeza posee alrededor de 8 × 10 agujas, produce un perfil bastante irregular. Por su parte, las impresoras de rueda margarita pueden proporcionar una impresión de gran calidad, pero carecen de la velocidad y la versatilidad de las de tipo matricial. Por ejemplo, una de rueda margarita sólo posee un tipo (a menos que el usuario desee cambiar la rueda), con una velocidad máxima de alrededor de 80 caracteres por segundo, mientras que una de tipo matricial puede producir textos a velocidades de 2 a 300 caracteres por segundo, aunque a esa última velocidad es probable que la calidad de la letra se vea aún más disminuida. Las impresoras láser son capaces de producir páginas impresas de calidad a velocidades asombrosas. Por ejemplo, las impresoras láser IBM 3800 Modelo 3, para usar con sistemas IBM de unidad principal, puede producir 215 páginas de texto por minuto.

La Apple LaserWriter, que es una versión a escala reducida de la impresora láser estilo IBM, puede producir una salida impresa de gran calidad a 300 caracteres por segundo. Además, al usuario le es dable elegir entre numerosos tipos distintos de un menú que tiene a su disposición. También puede producir diagramas, formas y otros tipos de trabajo artístico con un estándar muy superior al de las impresoras convencionales.

La unidad está diseñada para utilizarse con el Apple Macintosh y se conecta al ordenador a través de la misma conexión en serie de alta velocidad utilizada para otros tipos de impresoras.

La LaserWriter trabaja de una forma radicalmente diferente a la de las unidades convencionales. Mientras que las impresoras matriciales y de rueda margarita son esencialmente desarrollos del sistema de máquina de escribir, con una cabeza móvil que escribe sobre un rollo de papel, la LaserWriter se aproxima más a una fotocopiadora.

La impresora mide 290 por 420 por 475 mm y alberga dos tambores, el primero de los cuales contiene el toner fotocopiador, mientras que el segundo contiene el equipo láser y el cilindro rotativo que se utiliza para imprimir el dibujo o el texto sobre el papel.

El mecanismo de impresión, diseñado por Canon, trabaja tomando la página a imprimir y convirtiéndola en una imagen de bits. La imagen se transfiere luego al cilindro, que está cubierto por millones de puntos eléctricamente sensibles. Antes de que se produzca la transferencia, los puntos se ionizan de modo que retengan una carga eléctrica

positiva. Para transferir la imagen, el láser realiza un barrido sobre el cilindro, "escribiendo" la página en su superficie, neutralizando la carga positiva de los puntos sobre los que incide. El toner, que también está cargado positivamente, se utiliza, entonces, para cubrir la superficie del cilindro. El toner se "pegará" en los puntos neutralizados, pero evitará los demás, que, por tener la misma carga positiva, lo repelerán.

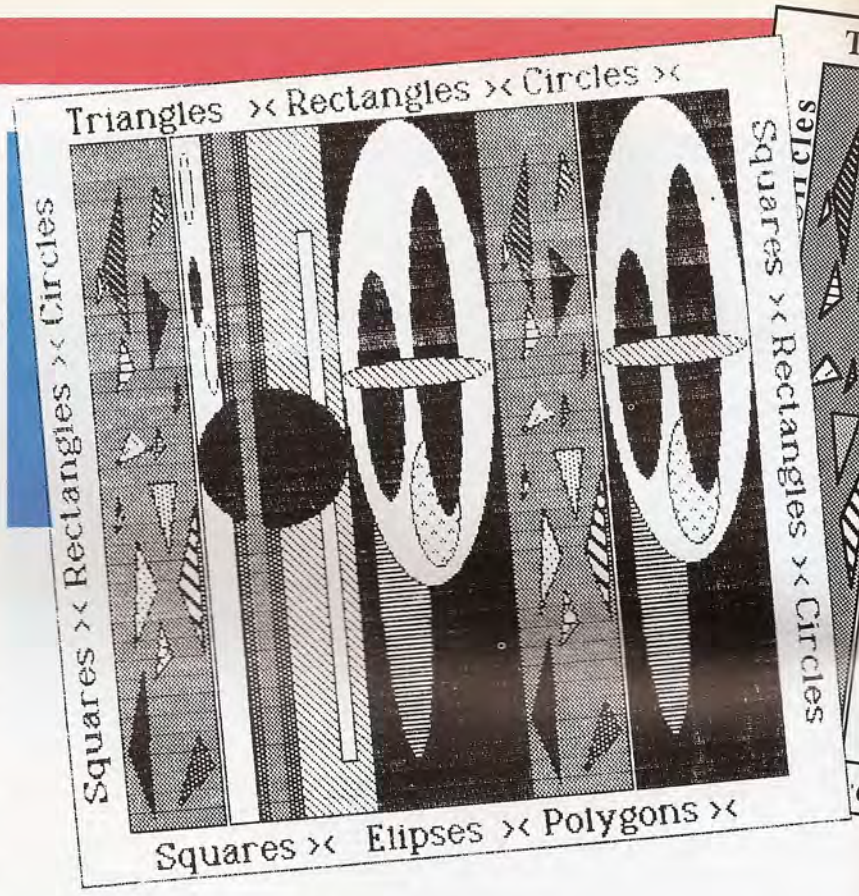
Para transferir el toner al papel, la LaserWriter aplica una carga negativa al papel a medida que éste pasa por la máquina. Cuando el papel pasa por el tambor, el toner cargado positivamente es atraído hacia el papel cargado negativamente y se adhiere a él. Para fijar el toner sobre la página de forma permanente, el papel pasa luego a través de un par de rodillos calentados a una temperatura de 200 °C. El toner se funde sobre el papel, que sale, entonces, de la máquina. Aplicando este sistena la LaserWriter puede producir hasta ocho páginas completas por minuto. Al funcionar, la LaserWriter genera una gran cantidad de calor, si bien éste no lo produce la máquina propiamente dicha, sino los rodillos.

Después de producir aproximadamente 3 000 páginas, algunos de los componentes del sistema láser comienzan a deteriorarse. Estos componentes están fijados al tambor del toner, y cuando el toner se acaba, no existe medio alguno de recargar el tambor, de modo que hay que sustituirlo por completo, junto con los componentes desgastados. Si bien este sistema puede parecer más bien caro, Apple señala que si se tiene en cuenta el precio de cada copia, la LaserWriter está a la altura de la mayoría de las fotocopiadoras corrientes.

La calidad de impresión producida es similar a la de la fotocomposición (como la utilizada para producir esta página). Esto se consigue en virtud de la cantidad de puntos que Canon ha logrado apiñar en la superficie del cilindro: 300 puntos por pulgada. Para una página de tamaño A4, esto significa que para su producción se utilizan casi siete millones de puntos.

Este sistema de impresión láser, aunque rápido y eficaz, no es de ningún modo exclusivo. Lo que diferencia a la LaserWriter de dispositivos similares existentes en el mercado, es la forma en que Apple ha incorporado en la mecánica su propia tecnología, convirtiéndola en la primera impresora "inteligente". Ya hemos visto que la LaserWriter coloca en su memoria una imagen de la página antes de iniciar la impresión. Teniendo en cuenta la altísima resolución de la impresión resultante, es necesario disponer de una considerable cantidad de memoria. La LaserWriter incorpora un ordenador en la placa, destinado exclusivamente a producir la página impresa.

Basada en el procesador Motorola 68000 y trabajando a 12 MHz, el mismo que utiliza el propio Macintosh, la LaserWriter posee 1,5 Mbytes de RAM y 500 Kbytes de ROM. Las ROM albergan juegos de caracteres completos para hasta 13 tipos de letra distintos. No obstante, incluso 2 Mbytes no pueden contener las imágenes completas de mapa de bits para todos ellos y todos los programas de aplicaciones que requiere la LaserWriter. Por lo tanto, con el objeto de ahorrar memoria, la ROM de la impresora se limita a retener la silueta de cada carácter con una rutina especial que "rellena" los caracteres cuando se imprimen. Este sistema no sólo ahorra espacio sino que, al utilizar la misma

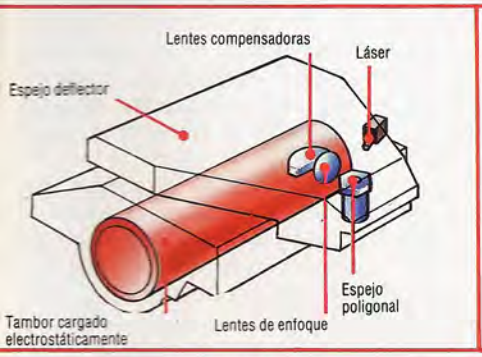


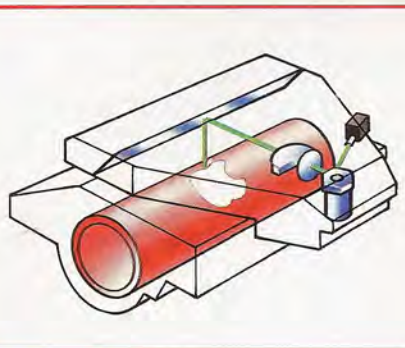
Procedimiento perfeccionado
Esta serie de ilustraciones muestran cómo la LaserWriter fija una imagen en el papel. Si bien la tecnología empleada es sumamente sofisticada, los principios son bastante simples y se basan en las técnicas de xerografía que se han venido utilizando desde hace ya muchos años

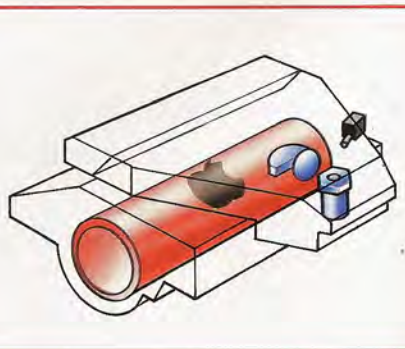
ambor se le aplica una carga electrostática
sitiva

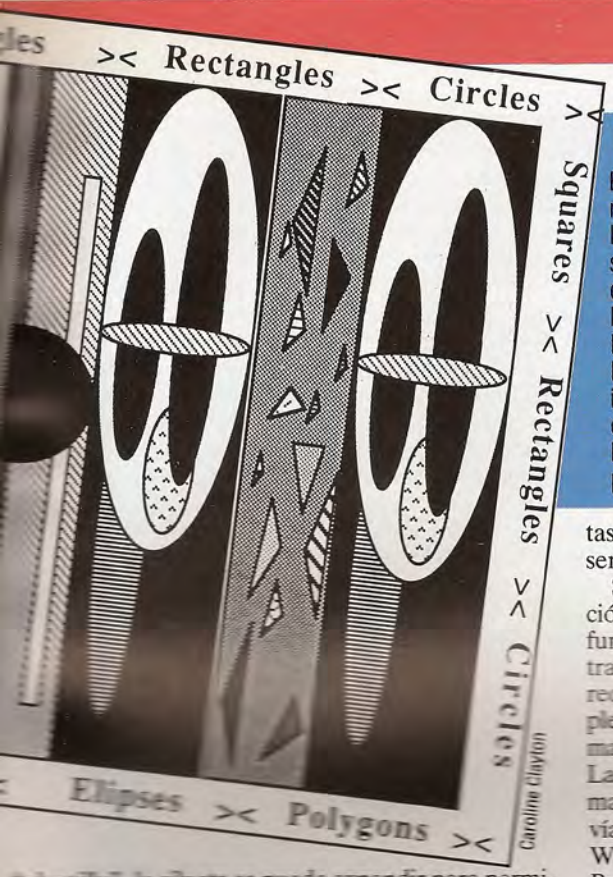
Luego se dispara el rayo láser y se desvía sobre el tambor, donde dibuja la imagen mediante la neutralización de los iones positivos

A medida que la superficie del tambor va pasando por el receptáculo del toner, éste, cargado positivamente, es atraído hacia las zonas "neutralizadas" y adherido a ellas









Comparación ilustrativa

Este dibujo se ha reproducido tanto en la impresora matricial Macintosh Imagewriter como en la LaserWriter. Aparte de los problemas que pueden surgir por la variación de la densidad de tinta en la cinta de la impresora matricial, la resolución de la LaserWriter es muchísimo mayor que la de Imagewriter. Un examen atento del dibujo de la Imagewriter revela la aparición de líneas no intencionales. Es allí donde se superponen los caracteres formados por la cabeza de impresión. Este problema no se presenta en la LaserWriter porque la imagen se fija en un tambor único

"plantilla", la silueta se puede expandir para permitir cualquier tamaño de tipo que se requiera.

También retenido en ROM hay un lenguaje especial desarrollado para usar con impresoras, que se denomina PostScript. Se trata de un lenguaje de programación interactivo que permite que el programador tome las siluetas de los tipos retenidos en ROM (y otras facilidades tales como líneas) y manipularlas para producir muchos estilos y diseños. Las instrucciones de que consta el lenguaje se pueden utilizar para crear rutinas que produzcan círculos, etc., que, a su vez, se pueden enlazar entre sí para producir diagramas complejos. En operación, el lenguaje se asemeja mucho al FORTH en su intenso uso de una pila para pasar datos desde y hacia los procedimientos, y en su capacidad para definir otros procedimientos. PostScript trabaja de una forma muy distinta a la de otros lenguajes de programación de gráficos tales como el LOGO. En vez de decirle a la impresora que dibuje una serie de líneas en términos absolutos, PostScript utiliza una serie de puntos (como los que componen las siluetas de tipos) que se espaciarán de acuerdo a una serie de proporciones.

Sin embargo, el PostScript es más que una opción adicional deseable: es fundamental para el funcionamiento eficaz de la LaserWriter. Para transferir el Mbyte completo de información que se requiere para una página entera, el Macintosh emplearía alrededor de 35 segundos. Pero los programas del disco de sistemas que se suministra con la LaserWriter convierten esta información en el formato Quickdraw del Macintosh. Los datos se envían, entonces, a la línea en serie de la LaserWriter, donde se convierten en instrucciones PostScript. Éstas, a su vez, son interpretadas por el ordenador del LaserWriter para reproducir la página como una imagen por mapa de bits dentro de la memoria del ordenador. El láser usa luego este mapa de bits para "dibujar" la página en el tambor electrosensible. Al reducir así la información, son suficientes 8 K de información, aminorando la velocidad de transmisión y el procesamiento.

La LaserWriter es, sin duda alguna, un paso trascendental en la tecnología de impresoras, no sólo por el avanzado mecanismo láser utilizado para producir una impresión rápida y de gran calidad, sino también por el ordenador proporcionado en la placa y el empleo del lenguaje de programación PostScript. Por supuesto, su precio la coloca muy lejos del alcance de la mayoría de usuarios del Macintosh. No obstante, es tradicional que la tecnología avanzada vaya introduciéndose en las máquinas de precio más económico y, dentro de pocos años, es razonable esperar que gran parte de la tecnología punta utilizada en la LaserWriter esté disponible en equipos más económicos.

APPLE LASERWRITER

DIMENSIONES
290 × 420 × 475 mm

INTERFACES
Interfaces en serie RS422 y RS232C

RESOLUCION DE IMPRESION
300 puntos por pulgada

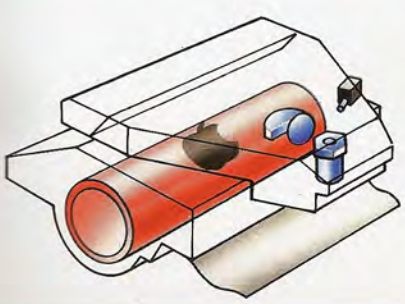
CPU
Procesador Motorola 68000 trabajando a 12 MHz

MEMORIA
1,5 Mbytes de RAM y 500 Kbytes de ROM

VENTAJAS
La LaserWriter produce letras y dibujos de calidad profesional a una velocidad muy superior a la mayoría de las otras impresoras diseñadas para micros

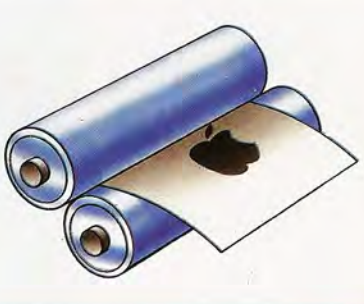
DESVENTAJAS
El tamaño máximo del papel es un poco mayor que el estándar A4, lo que limita severamente la posibilidad de producir trabajos artísticos y diseños. Además, por su precio, la máquina no está al alcance de todos los usuarios de micros

Luego se aplica una carga negativa al tambor del papel...



...que atrae al toner con carga positiva y forma una imagen en la página

Por último, el papel que retiene el toner pasa entre dos rodillos calentados que "fijan" el toner en la página



Estrategias oblicuas

Completamos los programas para los micros Amstrad, Commodore 64 y Spectrum añadiendo las rutinas finales de evaluación de movimientos

Uno de los motivos por los que ha resultado tan difícil programar el juego del *go*, es que es casi imposible simular el funcionamiento de la mente humana utilizando la tecnología actual. Para determinar los movimientos, el jugador humano puede analizar intuitivamente los formatos de grupos y líneas formados por las fichas. Puesto que los micros de hoy en día sólo pueden funcionar secuencialmente (es decir, un paso por vez), el análisis de las formas de los grupos plantea un problema. No obstante, podemos incluir rutinas que analicen las conexiones entre grupos adyacentes, permitiendo que el ordenador conecte y defienda dos de sus grupos, o ataque posibles conexiones entre grupos rivales. En este capítulo ofrecemos rutinas que llevan a cabo estas funciones en los micros Amstrad, Commodore 64 y Spectrum.

Módulo 5

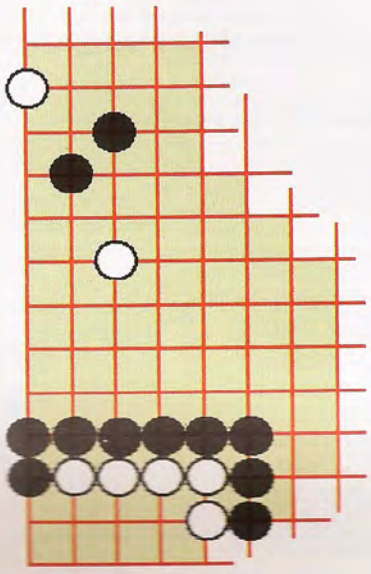
Commodore 64:

```
 330 GOSUB 790
 780 :
 790 REM RUTINA LEER FORMATOS
 810 PAT=TABLERO+556
 830 FOR L=0 TO 71
 840 READ P%:POKE PAT+L,(256+P%) AND 255
 850 NEXT
 860 DATA 32,17,16,2,-15,1
 870 DATA -32,-17,-16,-2,15,-1
 880 DATA 33,16,17,31,16,15
 890 DATA -33,-16,-17,-31,-16,-15
 900 DATA -14,1,-15,18,1,17
 910 DATA 14,-1,15,-18,-1,-17
 920 DATA 48,33,32,48,17,16
 930 DATA -48,-33,-32,-48,-17,-16
 940 DATA 3,-14,2,3,-15,1
 950 DATA -3,14,-2,-3,15,-1
 960 DATA 64,33,32,4,-14,2
 960 DATA -64,-33,-32,-4,14,-2
 980 RETURN
 990 :
1000 REM ************************************
2580 IF POSIT%=0 THEN GOSUB 2900:T$="DEF"
2590 IF POSIT%=0 THEN GOSUB 3070:T$="ATT"
2600 IF POSIT%=0 THEN GOSUB 3230:T$="SAT"
2610 IF POSIT%=0 THEN GOSUB 3380:T$="SCN"
2890 :
2900 REM CONEXION DE DEFENSA
2910 HI=-9999
2920 BV%=NEGRAS%:GOSUB 4420
2930 FOR A=1 TO 255:DCS%=PEEK(TABLERO+A):IF
     DCS%<>NEGRAS% GOTO 3010
2940 FOR P=PAT TO PAT+70 STEP 3
2950 B=PEEK(TABLERO+ ((A+PEEK(P))AND 255)):C
     =PEEK(TABLERO+ ((A+PEEK (P+1))AND 255))
2960 IF B<>NEGRAS% OR C<>BLANCAS% GOTO 3000
2970 D=(A+PEEK(P+2)) AND 255:SCR=RND(0)+
     PEEK(ESTIMACIONES+D)
2980 IF(D AND 240)=0 OR (D AND 15)=0 OR CSR<=HI
     GOTO 3000
2990 LP%=D:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%>2 THEN HI=SCR:POSIT%=D
3000 NEXT
3010 NEXT
3020 BV%=0:GOSUB 4420
3030 RETURN
3040 :
3050 REM ************************************
3060 :
3070 REM RUTINA CONEXION DE ATAQUE
3080 HI=-9999
3090 BV%=BLANCAS%:GOSUB 4420
3100 FOR A=1 TO 255:ACS%=PEEK(TABLERO+A): IF
     ACS%<>BLANCAS% GOTO 3180
3110 FOR P=PAT TO PAT+70 STEP 3
3120 B=PEEK(TABLERO+ ((A+PEEK(P)) AND 255)):C
     =PEEK(TABLERO+ ((A+PEEK (P+1))AND 255))
3130 IF B <>BLANCAS% OR C<>NEGRAS% GOTO 3170
3140 D=(A+PEEK(P+2))AND 255:SCR=RND(0)+
     PEEK(ESTIMACIONES+D)
3150 IF (D AND 240)=0 OR (D AND 15)=0 OR SCR< = HI
     GOTO 3170
3160 LP%=D:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%> 2 THEN HI=SCR: POSIT%=D
3170 NEXT
3180 NEXT:BV%=GOSUB 4420
3190 RETURN
3200 :
3210 REM ************************************
3220 :
3230 REM RUTINA INICIAR ATAQUE
3240 HI=-9999
3250 FOR A=1 TO 255:SAS%=PEEK(TABLERO+A): IF
     SAS%<>BLANCAS% GOTO 3330
3260 FOR P=PAT TO PAT+70 STEP 3
3270 B=PEEK(TABLERO+((A+PEEK(P))AND 255)):D
     =PEEK(TABLERO+((A+PEEK(P+2))AND 255))
3280 IF B<>BLANCAS% OR D<>0 GOTO 3320
3290 C=(A+PEEK(P+1))AND
     255:SCR=RND(0)+PEEK(ESTIMACIONES+C)
3300 IF (C AND 240)= OR (C AND 15)=0 OR SCR < = HI
     GOTO 3320
3310 LP%=C:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%>2 THEN HI=SCR:POSIT%=C
3320 NEXT
3330 NEXT
3340 RETURN
3350 :
3360 REM ************************************
3370 :
3380 REM RUTINA INICIO CONEXION
3390 HI=-9999
3400 FOR A=1 TO 255:SCS%=PEEK(TABLERO+A): IF
     SCS%<>NEGRAS% GOTO 3470
3410 FOR P=PAT TO PAT+70 STEP 3
3420 C=PEEK(TABLERO+((A+PEEK(P+2))AND 255)): IF
     C> 0 GOTO 3460
3430 B=(A+PEEK(P))AND 255:SCR=RND(0)+
     PEEK(ESTIMACIONES+B)
3440 IF (B AND 240)=0 OR (B AND 15)=0 OR SCR< = HI
     GOTO 3460
3450 LP%=B:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%>2 THEN HI=SCR:POSIT%=B
3460 NEXT
3470 NEXT
3480 RETURN
3490 :
3500 REM ************************************
4410 :
4420 REM RUTINA FRONTERAS
4430 FOR BX=0 TO 15
4450 BY=16*BX
4460 POKE TABLERO+BX,BV%
4470 POKE TABLERO+BY,BV%
4480 NEXT
4490 RETURN
4500 :
4510 REM ************************************
4520 REM ************ FIN DEL PROGRAMA *******
```

Amstrad CPC 464/664:

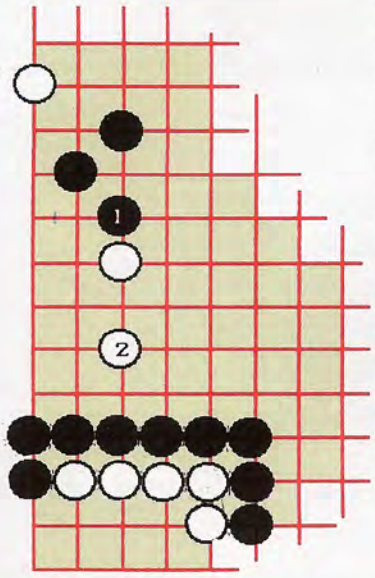
```
280 GOSUB 790: REM leer patrones
780 :
790 REM rutina lectura patrones
810 pat=tablero+&300
820 RESTORE 860
830 FOR I%=0 TO 72
840 READ p%:POKE(pat+I%),(256+p%) AND 255
850 NEXT I%
860 DATA 32,17,16,2,–15,1
870 DATA –32,–17,–16,–2,15,–1
880 DATA 33,16,17,31,16,15
890 DATA –33,–16,–17,–31,–16,–1
900 DATA –14,1,–15,18,1,17
910 DATA 14,–1,15,–18,–1,–17
920 DATA 48,33,32,48,17,16
930 DATA –48,–33,–32,–48,–17,–16
940 DATA 3,–14,2,3,–15,1
950 DATA –3,14,–2,–3,15,–1
960 DATA 64,33,32,4,–14,2
970 DATA –64,–33,–32,–4,14,–2
980 RETURN
990 :
1000 REM ****************************************
2580 IF posicion%=0 THEN GOSUB 2900:T$="DEF"
2590 IF posicion%=0 THEN GOSUB 3070:T$="ATT"
2600 IF posicion%=0 THEN GOSUB 3230:T$="SAT"
2610 IF posicion%=0 THEN GOSUB 3380:T$="SCN"
2900 REM rutina conexion de defensa
2910 hi=–9999
2920 bv%=negras%:GOSUB 4420: REM frontera
2930 FOR a%=1 TO 255:dcs%=PEEK(TABLERO+a%):IF
     dcs%<>negras% THEN 3010
2940 FOR p%=pat TO pat+70 STEP 3
2950 b%=PEEK(tablero+ ((a%+PEEK(p%))AND
     255)):c%=PEEK(tablero+ ((a%+PEEK (p%+1))AND
     255))
2960 IF b%<>negras% OR c%<>blancas% THEN 3000
2970 d%=(a%+PEEK(p%+2)) AND
     255:marcador=RND(1)+ PEEK(estimaciones+d%)
2980 IF(d% AND 240)=0 OR (d% AND 15)=0 OR
     marcador<= hi THEN 3000
2990 lp%=d%:LET lc%=negras%:GOSUB 3890:IF
     ll%=0 AND clib%>2 THEN hi=marcador:
     posición%=d%
3000 NEXT p%
3010 NEXT a%
3020 bv%=0:GOSUB 4420:REM frontera
3030 RETURN
3040 :
3050 REM ****************************************
3060 :
3070 REM rutina conexión de ataque
3080 hi=–9999
3090 bv%=blancas%:GOSUB 4420:REM frontera
3100 FOR a%=1 TO 255:acs%=PEEK (tablero+a%): IF
     cs%<>blancas% THEN 3180
3110 FOR p%=pat TO pat+70 STEP 3
3120 b%=PEEK(tablero+ ((a%+PEEK(p%)) AND
     255)):c%=PEEK(tablero+ ((a%+PEEK (p%+1))AND
     255))
3130 if b% <>blancas% OR c%<>negras% THEN 3170
3140 d%=(a%+PEEK(p%+2))AND 255:csr%=RND(1)+
     PEEK(estimaciones+d%)
3150 IF (d% AND 240)=0 OR (d% AND 15)=0 OR
     marcador< =hi THEN 3170
3160 lp%=d%:lc%=negras%:GOSUB 3890:IF ll%=0 AND
     clib%> 2 THEN hi=marcador:posicion%=d%
3170 NEXT p%
3180 NEXT a%
3190 RETURN
3200 :
3210 REM ****************************************
3220 :
3230 REM rutina iniciar ataque
3240 hi=–9999
3250 FOR a%=1 TO 255:sas%=PEEK(tablero+a%): IF
     sas%<>blancas% THEN 3330
3260 FOR p%=pat TO pat+70 STEP 3
3270 b%=PEEK(tablero+((a%+PEEK(p%))AND
     255)):d%=PEEK(tablero+((a%+PEEK(p%+2))AND
     255))
3280 IF b%<>blancas% OR d%<>0 THEN 3320
3290 c%=(a%+PEEK(p%+1)AND
     255:marcador=RND(1)+PEEK
     (estimaciones+c%)
3300 IF (c% AND 240)= 0 OR (c% AND 15)=0 OR
     marcador < =hi THEN 3320
3310 lp%=c%:lc%=negras%:GOSUB 3890:IF ll%=0 AND
     clib%>2 THEN hi=marcador:posicion%=c%
3320 NEXT p%
3330 NEXT a%
3340 RETURN
3350 :
3360 REM ****************************************
3370 :
3380 REM rutina inicio conexión
3390 hi=–9999
3400 FOR a%=1 TO 255:scs%=PEEK(tablero+a%): IF
     scs%<>negras% THEN 3470
3410 FOR p%=pat TO pat+70 STEP 3
3420 c%=PEEK(tablero+((a%+PEEK(p%+2))AND 255)):
     IF c% > 0 THEN 3460
3430 b%=(a%+PEEK(p%))AND 255:marcador=RND(1)+
     PEEK(estimaciones+b%)
3440 IF (b% AND 240)=0 OR (b% AND 15)=0 OR
     marcador< =hi THEN 3460
3450 lp%=b%:lc%=negras%:GOSUB 3890:IF ll%=0 AND
     clib%>2 THEN hi=marcador:posicion%=b%
3460 NEXT p%
3470 NEXT a%
3480 RETURN
3490 :
3500 REM ****************************************
4410 :
4420 REM rutina frontera
4430 FOR x%=0 TO 15
4450 y%=16* x%
4460 POKE (tablero+x%),bv%
4470 POKE (tablero+y%),bv%
4480 NEXT x%
4490 RETURN
4500 :
4510 REM ****************************************
4520 REM ***************** fin del programa*******
```

Densidad

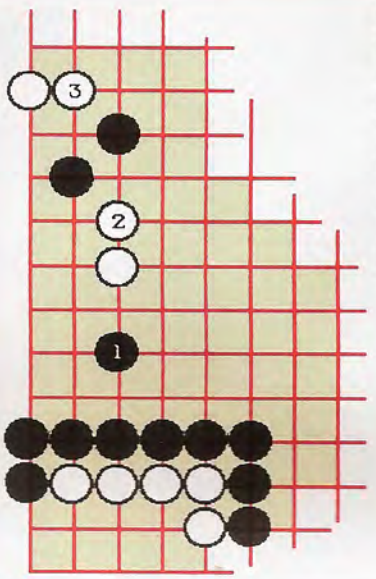


Grupo inexpugnable
Se dice que los grupos formados sólidamente en el exterior, como el de negras que vemos aquí, son "densos". Puesto que en un grupo denso no hay puntos débiles, es probable que los ataques iniciados contra ellos fracasen. Un importante axioma estratégico del *go* es: "manténte alejado de la densidad"

Caroline Clayton



Atacan las negras
Suele ser una buena estrategia utilizar los grupos densos propios como ayuda para un ataque contra fichas enemigas. En la situación que vemos aquí, la negra 1 juega encima de la ficha blanca que está sola, llevando por consiguiente a las blancas hacia el grupo denso de negras que se halla en la parte inferior del tablero. Las blancas tal vez intenten extenderse hacia afuera desde la ficha atacada jugando la ficha 2, pero es probable que a la larga no consigan el éxito, ya que el movimiento se halla más próximo a la zona densa de las negras



Otra frustración
Si las negras juegan en el lado inferior de la ficha blanca, es probable que el ataque fracase, porque las blancas pueden, simultáneamente, defenderse a sí mismas y atacar a las dos fichas negras cercanas jugando la ficha 2. Bajo la amenaza de un movimiento de las blancas en la posición 3, las negras se ven obligadas a actuar a la defensiva y el ataque se desbarata

Sinclair Spectrum:

```
 280 GO SUB 790
 790 REM rutina leer patrones
 810 LET pat=tablero+556
 820 RESTORE 860
 830 FOR I=0 TO 71
 840 READ p:POKE pat+I,p
 850 NEXT I
 860 DATA 32,17,16,2,–15,1
 870 DATA –32,–17,–16,–2,15,–1
 880 DATA 33,16,17,31,16,15
 890 DATA –33,–16,–17,–31,–16,–15
 900 DATA –14,1,–15,18,1,17
 910 DATA 14,–1,15,–18,–1,–17
 920 DATA 48,33,32,48,17,16
 930 DATA –48,–33,–32,–48,–17,–16
 940 DATA 3,–14,2,3,–15,1
 950 DATA –3,14,–2,–3,15,–1
 960 DATA 64,33,32,4,–14,2
 970 DATA –64,–33,–32,–4,14,–2
 980 RETURN
1000 REM ****************************
2580 IF posicion=0 THEN GO SUB 2900: LET
      t$="DEF"
2590 IF posicion=0 THEN GO SUB 3070: LET
      t$="ATT"
2600 IF posicion=0 THEN GO SUB 3230: LET
      t$="SAT"
2610 IF posicion=0 THEN GO SUB 3380: LET
      t$="SCN"
2900 REM rutina conexión de defensa
2910 LET hi=–9999
2920 LET bv=negras: GO SUB 4420
2930 FOR a=1 TO 255: LET
      dcs=PEEK(tablero+a):IF
      dcs%<>negras THEN GOTO 3010
2940 FOR p=pat TO pat+70 STEP 3
2950 LET b=a + PEEK p: LET c=a + PEEK
      (p+1)
2952 IF b> 255 THEN LET b=b–256: GO TO
      2952
2954 IF c> 255 THEN LET c=c–256: GO TO
      2954
2956 LET b=PEEK (tablero+b): LET c=PEEK
      (tablero+c)
2960 IF b<>negras OR c<>blancas THEN GO
      TO 3000
2970 LET d=a PEEK (p+2))
2972 IF d> 255 THEN LET d=d–256: GO TO
      2972
2974 LET marcador=RND+PEEK
      (estimaciones+d)
2980 IF INT (d/16)=0 OR d–16* INT(d/16)=0
      OR marcador<=hi THEN GO TO 3000
2990 LET lp=d: LET lc=negras: GO SUB 3890:
      IF ll=0 AND clib>2 THEN LET
      hi=marcador: LET posicion=d
3000 NEXT p
3010 NEXT a
3020 LET bv=0: GO SUB 4420
3030 RETURN
3070 REM rutina conexión de ataque
3080 LET hi=–9999
3090 LET bv=blancas: GO SUB 4420
3100 FOR a=1 TO 255: LET acs=PEEK
      (tablero+a): IF acs<>blancas THEN GO
      TO 3180
3110 FOR p=pat TO pat+70 STEP 3
3120 LET b=a+PEEK p: LET c=a + PEEK (p+1)
3122 IF b>255 THEN LET b=b–256: GO TO
      3122
3124 IF c>255 THEN LET c=c–256: GO TO
      3124
3126 LET b=PEEK (tablero+b): LET c=PEEK
      (tablero+c)
3130 IF b <>blancas OR c<>negras THEN GO
      TO 3170
3140 LET d=a+PEEK (p+2)
3142 IF d>255 THEN LET d=d–256: GO TO
      3142
3144 LET marcador=RND+PEEK
      (estimaciones+d)
3150 IF INT (d/16)=0 OR d–16* INT (d/16)=0
      OR marcador<=hi THEN GO TO 3170
3160 LET lp=d: LET lc=negras: GO SUB 3890:
      IF ll=0 AND clib> 2 THEN LET
      hi=marcador: LET posicion=d
3170 NEXT p
3180 NEXT a:LET bv=0: GO SUB 4420
3190 RETURN
3230 REM rutina inicio ataque
3240 LET hi=–9999
3250 FOR a=1 TO 255: LET sas=PEEK
      (tablero+a): IF sas<>blancas THEN GO
      TO 3330
3260 FOR p=pat TO pat+70 STEP 3
3270 LET b=a+PEEK p: LET d=a+PEEK (p+2)
3272 IF b>255 THEN LET b=b–256: GO TO
      3272
3274 IF d>255 THEN LET d=d–256: GO TO
      3274
3276 LET b=PEEK (tablero+b): LET d=PEEK
      (tablero+d)
3280 IF b<>blancas OR d<>0 THEN GO TO
      3320
3290 LET c=a+PEEK (p+1)
3292 IF c>255 THEN LET c=c–256: GO TO
      3292
3294 LET marcador=RND+PEEK
      (estimaciones+c)
3300 IF INT (c/16)=0 OR c–16*INT(c/16)=0
      OR marcador<=hi THEN GO TO 3320
3310 LET lp=c: LET lc=negras: GO SUB 3890:
      IF ll=0 AND clib>2 THEN LET
      hi=marcador: LET posicion=c
3320 NEXT p
3330 NEXT a
3340 RETURN
3380 REM rutina inicio conexión
3390 LET hi=–9999
3400 FOR a=1 TO 255: LETscs=PEEK
      (tablero+a): IF scs<>negras THEN GO
      TO 3470
3410 FOR p=pat TO pat+70 STEP 3
3420 LET c=a+PEEK (p+2)
3422 IF c>255 THEN LET c=c–256: GO TO
      3422
3424 LET c=PEEK (tablero+c): IF c> 0 THEN
      GO TO 3460
3430 LET b=a+PEEK p
3432 IF b>255 THEN LET b=b–256: GO TO
      3432
3434 LET marcador=RND+PEEK
      (estimaciones+b)
3440 IF INT (b/16)=0 OR b–16* INT (b/16)=0
      OR marcador<=hi THEN GO TO 3460
3450 LET lp=b: LET lc=negras: GO SUB 3890:
      IF ll=0 AND clib>2 THEN LET
      hi=marcador: LET posicion=b
3460 NEXT p
3470 NEXT a
3480 RETURN
4420 REM rutina fronteras
4430 FOR x=0 TO 15
4450 LET y=16* x
4460 POKE tablero+x,bv
4470 POKE tablero+y,bv
4480 NEXT x
4490 RETURN
4500 :
4510 REM ****************************
4520 REM **** FIN DEL PROGRAMA *******
```

Ahorro razonable

Centramos nuestra atención en las definiciones de dos puntos y los métodos de compilación

Una facilidad importante del FORTH es que sus definiciones de dos puntos son semicompiladas. Si bien no se convierten en código máquina (como en una compilación completa), se procesan para que ocupen menos espacio y se ejecuten más rápidamente. Como resultado de ello, los programas en FORTH son muy eficientes.

Existen varios métodos de semicompilación. El método figFORTH es, con mucho, el más corriente, al punto de que otros métodos se consideran no ortodoxos. Supongamos que usted define una palabra:

```
:CUADRADO (n—n* n) DUP*;
```

En primer lugar, se coloca un *encabezador* en el diccionario. Éste contiene (el verdadero orden de entrada puede variar entre las distintas versiones del FORTH) el nombre CUADRADO y la dirección de memoria del encabezador para la palabra definida previamente. De este modo todos los encabezadores se unen entre sí y, comenzando por el más nuevo, el ordenador puede ir buscando hacia atrás a través de todo el diccionario hasta hallar la palabra que esté buscando. Estas dos partes del encabezador se denominan *campo del nombre* y *campo de unión*.

Tras el encabezador viene la definición. Primero hay un código que muestra de qué tipo de definición se trata; hay códigos diferentes para definiciones de dos puntos (como en este caso), variables, constantes, las estructuras preparadas por CREATE, palabras definidas en código máquina para el sistema y para cualquier otra clase de palabras. Esta parte se denomina *campo del código*.

A continuación viene la parte principal de la definición de dos puntos. Ésta es una lista de las direcciones de las palabras que utiliza (en realidad son las direcciones de los campos del código de esas palabras, también conocidas como sus *direcciones de compilación*). Esta parte principal se denomina *campo de parámetros*.

Usted termina con lo siguiente:

campo del nombre:	"CUADRADO"
campo de unión:	dir. del encabezador de la pal. anterior (2 bytes)
campo del código:	cód. para una definición de dos puntos (2 bytes)
campo de paráms.:	dir. de compilación de DUP (2 bytes)
	dir. de compilación de* (2 bytes)
	dir. de compilación de ; (2 bytes)

Ahora puede empezar a ver cómo se compilan las definiciones de dos puntos. Primero se establecen los campos del nombre y de unión. Esto es igual para cualquier definición de palabra nueva. A continuación, : coloca el código especial para las definiciones de dos puntos, y también coloca al FORTH en un estado de compilación especial. A partir de este momento, cuando entre una palabra, ésta no se ejecuta de la forma normal, sino que se incluye en el diccionario su dirección de compilación. También se suprime el mensaje OK para hacerle saber qué está sucediendo.

Esto continúa hasta ;, de modo que, obviamente, ; ha de ser especial. No sólo se incluye en el diccionario, sino que también tiene el efecto inmediato de decirle al FORTH que vuelva del estado de compilación a su estado normal. Palabras como ésta, que se deben ejecutar directamente como parte del proceso de compilación, se denominan palabras *inmediatas*. No se incluyen en el diccionario a menos que ellas mismas se ocupen de ello. En realidad, lo que ; incluye en el diccionario no es su propia dirección de compilación, sino la de una sombra anónima, la acción en tiempo de ejecución de ;.

Cuando se ejecuta CUADRADO (p. ej., si usted entra 4 CUADRADO .), el FORTH lo busca en el diccionario y encuentra su definición. Ésta le dice entonces, a su vez, que ejecute DUP y * y luego se detenga, pero no necesita buscarlas en el diccionario porque ya posee las direcciones de sus definiciones. En consecuencia, las definiciones de dos puntos se pueden ejecutar bastante velozmente.

Esto describe las definiciones de dos puntos más simples. Tanto los números como las estructuras de programa presentan dificultades.

Está claro que los números no poseen direcciones de compilación, de modo que se compilan de una forma compuesta: primero, la dirección de compilación de un *manipulador literal* (otra sombra anónima) y luego el número propiamente dicho. Cuando se ejecuta desde el interior de una definición de dos puntos, el efecto del manipulador lite-

Campos de código

Los campos de código son bastante inteligentes. El número de código en realidad es la dirección de memoria del código máquina que se ejecuta cada vez que se ejecuta la palabra. Para cualquier definición de dos puntos, el código máquina recuerda a dónde ha de retornar el intérprete de FORTH cuando termine esta definición en particular. Esto se efectúa colocando la dirección de retorno en otra pila, la *pila de retorno*, y volviendo a dirigir el intérprete hasta la definición actual. Hay rutinas diferentes en código máquina para constantes y variables (que en sus campos de parámetros poseen un único número). Las palabras *primitivas*, como +y*, están escritas en código máquina. Éste se almacena en el campo de parámetros y el campo de código contiene su dirección

FORTH Amstrad
El Amstrad CPC 464/664 se une a otros micros personales en la capacidad para ejecutar FORTH en virtud de este paquete de Kuma Software. La versión Kuma del FORTH es una implementación de figFORTH, con facilidades para punto flotante, series y gráficos en color

ral consiste en decirle al sistema que "los dos bytes siguientes que ves no son una dirección de compilación a ejecutar, sino un número a colocar en la pila". ."posee un compuesto similar, formado por un manipulador seguido por la serie propiamente dicha.

La siguiente dificultad se plantea con estructuras de programación tales como IF...ELSE...THEN. Si usted estuviera programando en código máquina (o en los BASIC más primitivos), tendría que hacer esto con saltos, algo así como:

```
1000 Si parte superior pila =0 THEN GOTO 1100
1010 REM hacer esta parte si la condición es falsa
        :
        :
1090 GOTO 1200
1100 REM hacer esta parte si la condición es
     verdadera
        :
        :
1200 REM continuar a partir de aquí en ambos casos
```

El FORTH compilado es bastante parecido a esto. En FORTH hay dos palabras, BRANCH (salto incondicional) y ?BRANCH (quitar la parte superior de la pila y saltar si es cero), pero usted no puede emplearlas en el curso normal de los acontecimientos porque hay que utilizarlas como formas compuestas (como un número compilado): la dirección de compilación seguida por la dirección del lugar hasta donde saltar. Es labor de IF, ELSE, THEN, etc., que son palabras inmediatas, el calcular estas direcciones de salto y compilar las formas compuestas. Por ejemplo, IF incluye la dirección de compilación de ?BRANCH y espacio para la dirección del salto, pero no puede llenar la dirección del salto porque aún no sabe dónde está ELSE. En cambio, deja en la pila la dirección de este espacio para que ELSE pueda llenar la dirección de salto. En realidad, IF también deja en la pila un número de comprobación. Cada clase de estructura posee su propio número de comprobación, de modo que si ELSE no encuentra a IF, entonces sabe que algo anda mal: o que nunca hubo un IF, o bien que algo de en medio se ha mezclado en la pila. Ello impide que se escriban cosas como:

```
IF...BEGIN...ELSE...UNTIL...THEN
```

Veamos algunos ejemplos.

1. Supongamos que su versión de FORTH distingue entre caracteres en mayúscula y en minúscula, y que todas las palabras estándares se entran en el diccionario en mayúsculas. Esto significaría que no se reconocería loop, mientras que sí se reconocería LOOP. Si usted quisiera entrar palabras tanto en minúsculas como en mayúsculas, podría comenzar por:

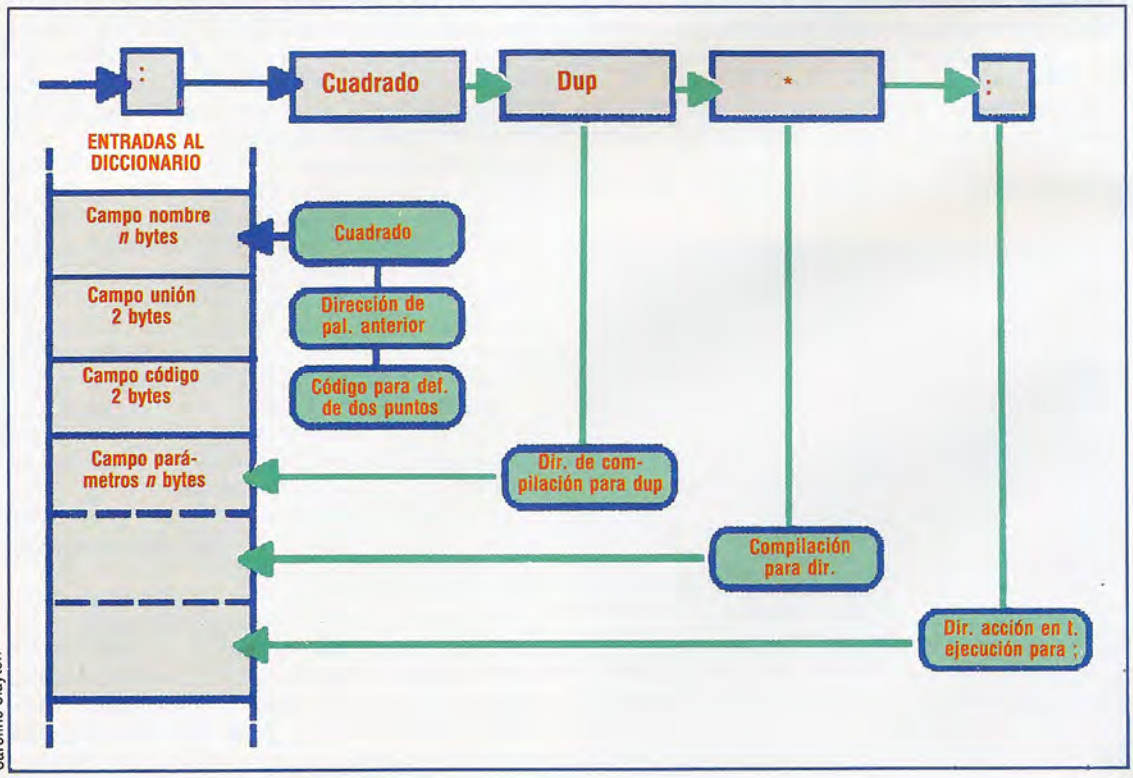
```
:drop DROP;
:roll ROLL;
:variable VARIABLE (sí, esto funcionaría);
```

Tendría problemas con:

```
:loop LOOP (esto no funcionaría);
```

```
:loop [COMPILE] LOOP;IMMEDIATE
```

Usted habrá de tener una mente muy lúcida para ver lo que hace esto, puesto que hay tres circunstancias distintas a tener en cuenta. Primero debe considerar qué sucede cuando se define loop. Aunque LOOP es inmediata, [COMPILE] hace caso omiso de ella, de modo que la dirección de compilación de LOOP pasa a la definición de loop. La segunda circunstancia, tiempo de compilación, es cuando se utiliza loop para redefinir otra palabra. Dado que es inmediata, se ejecuta directamente y tiene por efecto ejecutar LOOP. Éste es el momento que exige mayor concentración, porque es cuando loop cumple su cometido. La tercera circunstancia, tiempo



Almacenamiento de palabras
El FORTH emplea un método de "semicompilación" para facilitar la ejecución de los programas. Una vez entrada una definición de dos puntos, se establecen en el diccionario diferentes campos, comenzando con los campos de nombre, de unión y de código. El FORTH entra, entonces, en una *modalidad de compilación*, en la que la palabra no se entra directamente en el diccionario, sino que es su *dirección de compilación* la que se entra en el campo de parámetros de la definición. Tras encontrar el ; el FORTH vuelve a la *modalidad de ejecución*

Palabras importantes

[y] le permite pasar temporalmente al estado de ejecución estando aún en medio de una definición de dos puntos.

LITERAL, una palabra inmediata, quita la parte superior de la pila y la compila en una definición de dos puntos de la forma normal para un número. Típicamente usted la utilizaría con [y] para evaluar una expresión en tiempo de compilación y compilar el resultado. Por ejemplo:

```
[9 16*] LITERAL
```

en una definición de dos puntos tiene el mismo efecto que 144.

[COMPILE] es una palabra inmediata y dice que la siguiente palabra no se debe ejecutar sino incluir, aun cuando sea inmediata.

COMPILE no es inmediata. Al ejecutarse, incluye en el diccionario la siguiente palabra.

BRANCH y ?BRANCH se utilizan como en el ejemplo cuando usted construye saltos. > MARK y > RESOLVE se utilizan para llenar la dirección de salto para un salto hacia adelante: > MARK se utiliza en el salto y > RESOLVE en su destino. < MARK y < RESOLVE son similares para saltos hacia atrás, con < MARCK en el destino y < RESOLVE en el salto.

IMMEDIATE hace inmediata la palabra anterior.

de ejecución, es cuando se ejecuta la otra palabra, y después entra en vigor lo que fuere que haya compilado LOOP en su definición.

2. He aquí un par de palabras, ?DO y ?LOOP. Son como DO y LOOP, pero si el inicio ya es tan grande como el límite, entonces el bucle no se ejecuta nunca.

```
?DO equivale a      OVER OVER > IF DO
?LOOP equivale a    LOOP ELSE DROP DROP
                    THEN
```

Sin embargo, si usted intenta definir:

```
:?DO (no está bien) OVER OVER > IF DO;
```

tendrá el mismo problema que con loop. La definición correcta es:

```
:?DO
   COMPILE OVER
   COMPILE OVER
   COMPILE >
   [COMPILE] IF
   [COMPILE] DO
;IMMEDIATE

:?LOOP
   [COMPILE] LOOP
   [COMPILE] ELSE
   COMPILE DROP
   COMPILE DROP
   [COMPILE] THEN
;IMMEDIATE
```

3. Las estructuras de caso proporcionan una gama de acciones según cual sea el valor de algo. No se incluyen en el estándar, pero hay una utilizada comúnmente que puede definir por sí mismo. Su forma es:

```
valor CASE
1.ª posibilidad OF...ENDOF
2.ª posibilidad OF...ENDOF
3.ª posibilidad OF...ENDOF
...

Qué hacer si no responde a ninguna posibilidad

ENDCASE
```

Por ejemplo, el valor podría ser el código ASCII de una tecla pulsada y usted puede especificar distintas acciones para los diferentes caracteres que fueran factibles.

Si observa los saltos necesarios para hacer esto, verá que cada OF necesita un salto condicionado a su ENDOF, y que cada ENDOF necesita un salto a ENDCASE. ENDCASE podría necesitar llenar, o *resolver*, varias direcciones de salto. En FORTH no hay ninguna palabra estándar para hacer esto, de modo que está obligado a calcular por sí mismo los saltos para CASE, lo que hará mediante >MARK y >RESOLVE. >MARK apila la dirección del espacio para una dirección de salto, como hace IF, y >RESOLVE lo quita de la pila y coloca el espacio.

```
:CASE  (run: valor —valor
        (compile: —0)
0       (cantidad de OFs—ninguno hasta ahora)
;IMMEDIATE

:OF     (run:valor, posibilidad— [si pareja])
 (      valor, posibilidad—valor [ninguna pareja])
        (compile:—marca para salto a ENDOF)
COMPILE OVER
COMPILE=
COMPILE?BRANCH>MARK
COMPILE DROP (drop valor si pareja)
;IMMEDIATE

:ENDOF (run:—)
   (compile: OF cuenta, marca del OF—
   (              marca para salto a ENDCASE,
                  cuenta + 1 de OF)
COMPILE BRANCH > MARK
SWAP > RESOLVE (resolver salto desde OF)
SWAP 1 +          (incrementar cuenta OF)
;IMMEDIATE
:ENDCASE (run:valor—
           (compile:marcas desde los ENDOF,
           OF
           cuenta—)
   COMPILE DROP
   0?DO
      > RESOLVE
   ?LOOP
;IMMEDIATE
```

Aunque esto es complicado, el mero hecho de que usted pueda hacerlo muestra lo ampliable que es el FORTH.

En la práctica, también debería utilizar números de comprobación para asegurar que una estructura CASE se ensamble de la forma adecuada.

El FORTH-79 carece de BRANCH, >MARK, >RESOLVE, etc., y aunque el figFORTH posee palabras similares, usted tendrá que ahondar mucho más para utilizarlas.

Amstrad para lo que gusten

Al comenzar esta serie sobre el OS del Amstrad CPC 464 y 664 examinaremos los mapas de la ROM y de la RAM

Tanto en el Amstrad CPC 464 como en el 664 queda muy poco de la más reciente tecnología del silicio. Sin embargo, las configuraciones que los ingenieros informáticos suelen implementar en el hardware se compensan en las máquinas Amstrad con un sistema operativo más globalizador. De aquí que se necesitara una cantidad mínima de hardware para proporcionar las facilidades disponibles en las máquinas Amstrad, lo que dio como resultado un micro menos caro.

El CPC 464 emplea una CPU Z80A, acompañada de 64 K de RAM, 32 K de ROM, un PIA 8255 (adaptador de interfaces periféricas), un generador de sonido AY-3-8912, un CRTC 6845 (*cathode-ray tube controller:* controlador de tubo de rayos catódicos) y una matriz de puertas fabricada a medida para el cliente (ULA: *uncomitted logic array:* disposición lógica no comprometida).

El sistema operativo, conocido como *firmware,* ocupa la mitad inferior de la ROM de 32 K. El resto de la ROM contiene la versión del BASIC del Amstrad, denominada Locomotive BASIC. El firmware se divide en subsecciones, cada una de las cuales se encarga de un aspecto particular del sistema. Esto ofrece una cómoda estructuración que sirve de base para nuestro estudio del OS del Amstrad que ahora iniciamos. De momento echemos una mirada general al sistema.

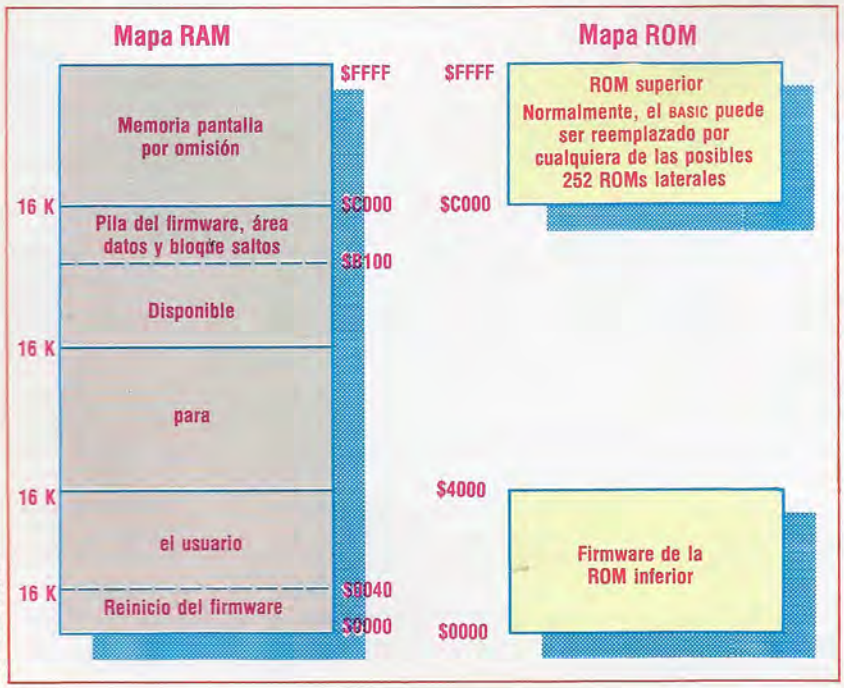
El mapa de memoria del sistema queda reflejado en el esquema (abajo), donde aparecen las dos mitades de la ROM como dos ROM separadas lógicamente y con 16 K cada una, ambas conectables por separado mediante la ULA. Se observará que toda la ROM de 32 K se superpone a los primeros y últimos 16 K de la RAM que quedan siempre libres para lectura y escritura. La ULA ofrece también la facilidad para conectar la ROM superior con cualquiera de las 252 ROM laterales de ampliación. En próximos capítulos abordaremos el empleo de las ROM de ampliación.

La pantalla es controlada por el chip 6845 y requiere 15 K de RAM, que la memoria ofrece por lo general inmediatamente debajo de la ROM superior. No obstante, es posible elegir cualquiera de los cuatro bloques de 16 K para que alberguen los datos de pantalla (lo que permite la conexión de pantallas múltiples a velocidades extraordinarias).

Cuando se ha conectado la ROM superior (p. ej., cuando el BASIC está bajo control) todo lo que se lee de los 16 K superiores de la memoria es organizado por la ULA para que dé posiciones dentro de la ROM. Cualquier cosa escrita en este bloque de 16 K pasa a través de la ROM hasta la RAM subyacente, que habitualmente se encuentra ocupada por la pantalla. La lectura desde la RAM requiere que esté desconectada la ROM superior; el firmware proporciona diversos modos de hacerlo.

Un acceso a la memoria similar se aplica a la ROM inferior que contiene el firmware. Pero ¿qué sucede si la ROM inferior está desconectada? ¿Cómo pueden ejecutarse los programas del sistema operativo si han desaparecido? Este problema se resuelve no accediendo directamente a la ROM inferior a través de programas, haciendo en su lugar todas las llamadas al sistema operativo a través de un *bloque de saltos* (*jumpblock*). Se trata de un área de memoria que tiene una posición fijada y contiene una serie de entradas que sencillamente realizan un salto (o acción equivalente) a distintas rutinas dentro del firmware. El empleo de un bloque de saltos permite el cambio de las direcciones actuales de las rutinas del firmware, sin afectar los programas existentes. Conviene llamar las rutinas del firmware a través del bloque de saltos.

El bloque de saltos reside en la RAM de sistema en el tercero de los bloques de 16 K, por lo que no se puede desconectar igual que la ROM. Se ha previsto la imposibilidad de que el usuario sobreescriba esta área. Aun así, hay situaciones en las que es deseable alterar el bloque de saltos para eliminar o sustituir parte del sistema operativo. Esto se denomina *parcheo del bloque de saltos (jumpblock patching)* que analizaremos con detenimiento en próximos capítulos.



La ROM encima de la RAM
El mapa de memoria del Amstrad muestra claramente cómo el chip de ROM de 32 K "despagina" los 16 K superiores e inferiores de la RAM. El firmware puede emplearse para activar y desactivar ambas páginas de RAM, así como para comprobar la presencia de hasta 252 ROMs laterales. Los datos de pantalla pueden contenerse en cualquiera de los cuatro bloques RAM de 16 K

Caroline Clayton

Cómo trabaja el bloque de saltos

Todos los puntos de entrada al sistema operativo tienen una entrada correspondiente en el bloque de saltos. Cuando el usuario desea acceder al sistema operativo, se hace una llamada a la entrada adecuada del bloque de saltos. Esta entrada emite, entonces, una instrucción RST, la cual, a su vez, llama al firmware. Cada entrada tiene tres bytes de longitud; en la puesta en marcha cada una contiene una instrucción RST seguida de una dirección de entrada del sistema operativo.

Cuando no es necesaria, es posible desconectar la ROM inferior, por cuya razón se mantiene una copia de las rutinas de reiniciación en la RAM contigua a la ROM. Estas rutinas de reiniciación activan la ROM del firmware antes de ser llamada a continuación. Cuando la rutina del sistema operativo retorna, las ROM son restauradas al estado que tenían a la entrada. Esto resuelve el dilema de decidir qué ROM hay que conectar antes de llamar al sistema operativo. La mayoría de las instrucciones de reiniciación se emplean para disponer de un conjunto ampliado de instrucciones; las instrucciones adicionales son principalmente para controlar la activación de la ROM. En la página siguiente se proporciona un cuadro con estas instrucciones de reiniciación y sus empleos.

El sistema de bloque de saltos permite reemplazar las entradas existentes del sistema operativo por rutinas sustitutas. Por ejemplo, si desea escribir sus propias rutinas de manejo de la pantalla, éstas pueden "parchearse" de modo que sustituyan las rutinas existentes alterando la entrada del bloque de saltos para que apunte a una nueva rutina. Damos dos diagramas para mostrar la diferencia entre el flujo de control cuando se pasa una instrucción al sistema operativo de la manera habitual, y la misma cadena cuando ha sido interceptada por el usuario.

Este procedimiento de parchear un bloque de saltos es de hecho el método empleado tanto en el CPC 464 como en el 664 para conectar entre las instrucciones de cassette y de disco.

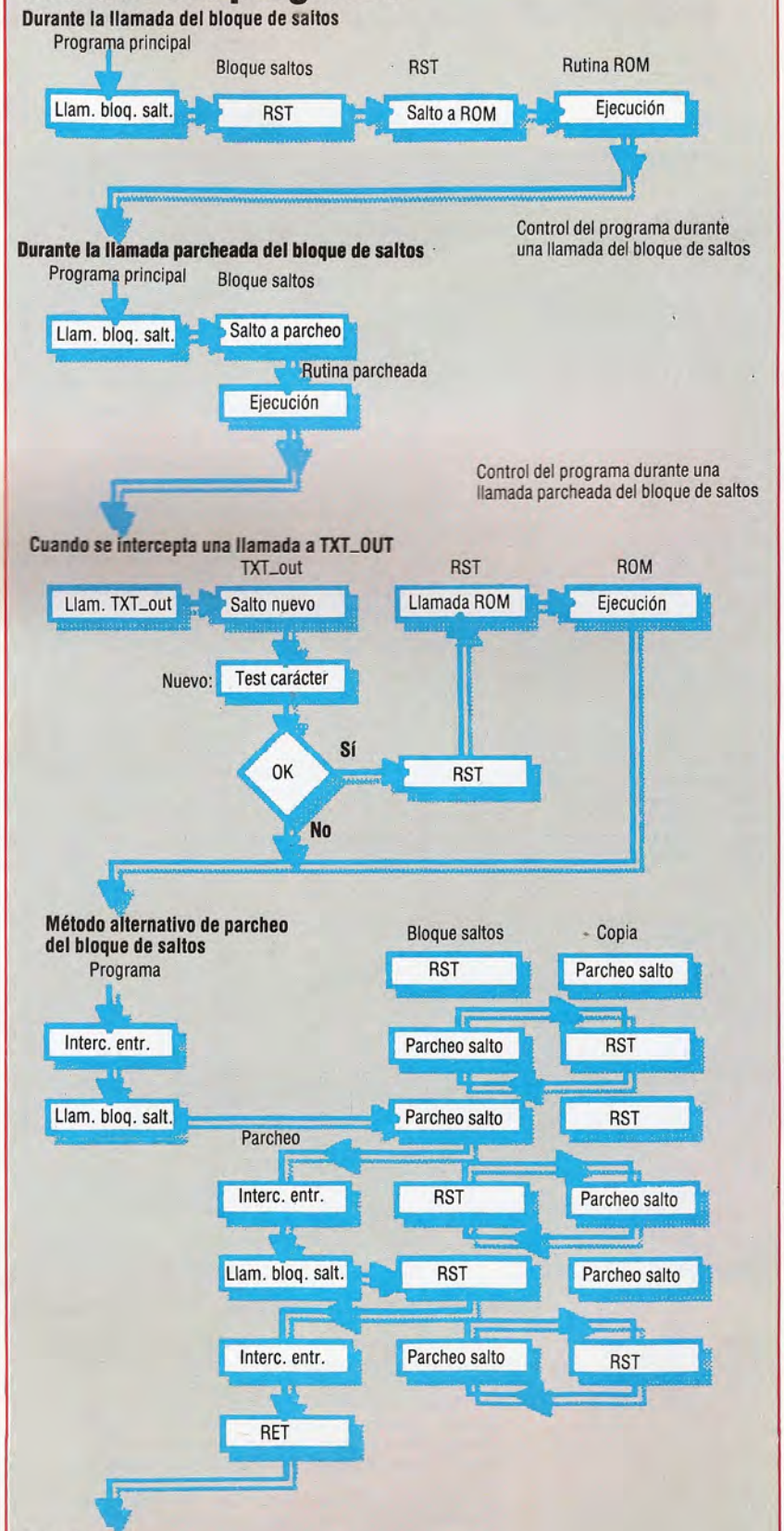
Rutina ilustrativa

Un ejemplo de tal método lo tendría si supone que ha escrito una rutina llamada SEND SCREEN, que envía texto a la pantalla, y desea usar ésta en lugar de la rutina existente del sistema operativo. La entrada en el bloque de saltos para imprimir caracteres en la pantalla se llama TXT WR CHAR y tiene dirección en OBB5DH. El acoplamiento se logra sustituyendo un salto a la nueva rutina en el bloque de saltos.

El procedimiento adoptado antes funciona bien cuando se ha escrito una rutina enteramente nueva y la entrada no se necesita. El problema surge cuando bien se está todavía empleando la entrada original o bien se necesita una llamada desde dentro de la nueva rutina a otra entrada en el bloque de saltos. Pongamos este ejemplo de programa en BASIC que sólo toma códigos de caracteres y los envía a la pantalla:

```
10 MODE 1          limpia pantalla y envía
                   cursor a inicio
```

Control del programa



Salta y brinca

El bloque de saltos permite al usuario emplear el firmware de una manera completa y sin complicaciones, y también redirigir las llamadas del sistema hacia rutinas definidas por el usuario. Este último proceso implica el parcheado del bloque de saltos, y los diagramas que aquí exponemos muestran las distintas fases necesarias, según el método empleado

Caroline Clayton

```
20 WHILE-1                    establece un bucle infinito
30 INPUT "Codigo              toma el codigo de caracter
   de caracter";C;
40 IF C <0 OR C>255           comprueba validez codigo
   then ?:GOTO 30
50 ?CHR$(C);                  lo envía
60 WEND                       itera el bucle
```

El programa permite imprimir todos los caracteres de gráficos y códigos de control. Sin embargo, hay un código que desactiva la pantalla, lo que significa que el programa quedaría "colgado". Por consiguiente, se necesita establecer un test para impedir el envío de este código. Este test podría insertarse fácilmente en la línea 40, pero lo estableceremos en la 50 para favorecer el propósito general de este ejemplo (línea en la que el carácter se imprime efectivamente). Para ello, hay que hacer el test antes de que el carácter sea enviado a la pantalla, empleando la entrada del bloque de saltos TXT OUTPUT en la dirección $BB5A. La entrada original requiere que se pase al registro A el caracter que ha de ser enviado y que se guarden todos los registros a la salida.

El modo más sencillo de lograr el parcheado *(patch)* del bloque de saltos es el que muestra el siguiente listado.

```
        txt_ou:   equ   #BB5A          ;rutina salida bloque
        disabl:   equ   21             ;caracter desactivador pantalla
        ;Sigue nueva rutina - el caracter a visualizar esta en A
FE15    new;      cp    disable        ;es desactivado VDU?
C8                ret   z              ;ignorarlo, si es así
000000  entry:    defb  0,0,0          ;inserta entrada original
        ;La direccion de retorno esta ya en la pila a la entrada
        ;a la rutina, luego es innecesaria una instruccion RET
        ;La entrada del bloque de saltos existente en txt out debe copiarse en ENTRY
        ;La entrada original JMP debe sustituirse por NEW
```

El enfoque adoptado funcionará con cualquiera de las entradas de rutina en el bloque de saltos. Sin embargo, habrá de tener cuidado cuando se está parcheando una rutina y asegurarse de que nadie lo haya hecho antes. El problema está en que algunas entradas del bloque de saltos cuentan con su dirección para funcionar adecuadamente (es decir, si se copia la entrada en otra parte, cuando sea llamada no funcionará bien). Para parchear el bloque con este tipo de entradas se adoptará otro enfoque diferente.

Proporcionamos un diagrama que muestra un método alternativo de parchear el bloque de saltos. Se hace una copia de la entrada actual y se sustituye en su lugar por un salto a la nueva rutina. Ésta copia, entonces, la antigua entrada en su lugar apropiado dentro del bloque de saltos y la llama. Antes del retorno, el salto a la nueva rutina se vuelve a colocar en el bloque de saltos.

Este segundo listado en código máquina es un programa que toma nuestro ejemplo y lo parchea de la manera ya descrita.

```
        txt_ou:   equ   #BB5D          ;rutina salida del bloque saltos
        disabl:   equ   21             ;caracter desact. pantalla
FE15    new:      cp    disable        ;punto entrada rutina
C8                ret   z              ;ignora desactiv.
;ahora conecta entrada antigua al bloque saltos y la llama
CD2529            call  copy           ;conecta entrada
CD5DBB            call  txt_out        ;envia caracter
CD2529            call  copy           ;recopia la entrada
C9                ret

F5      copy:     push  af             ;debe guardar todos los regs
C5                push  bc
D5                push  de
E5                push  hl
0603              ld    b,3            ;ningun byte en entrada
215DBB            ld    hl,tx_out      ;entrada antigua
113F29  cloop:    ld    de,txt_cpy     ;copia area
4E                ld    c,(hl)         ;lee entrada actual
1A                ld    a,(de)         ;y la reemplaza
77                ld    (hl),a         ;por su sustituta
79                ld    a,c
12                ld    (de),a         ;y guarda la antigua
23                inc   hl             ;apunta al byte siguiente
13                inc   de
10F7              djnz  cloop          ;ejecuta los tres
E1                pop   hl
D1                pop   de
C1                pop   bc
F1                pop   af
C9                ret
        ;almacenamiento de nueva entrada bloque saltos hasta que se necesite
C3      txt_cp:   defb  #c3            ;codigo de bloque saltos
1829              defw  new            ;para rutina sustituta
```

Este enfoque a primera vista puede parecer complicado, pero su perfecto funcionamiento está garantizado sea cual fuere el contenido original del bloque de saltos.

Es importante darse cuenta de que el firmware debe ver toda rutina sustituta como si fuera una rutina por omisión (esto es, todos los parámetros pasados y devueltos deben tener la misma definición que los de la rutina sustituida). En el ejemplo que hemos proporcionado, esto se consigue sencillamente guardando los registros devueltos mientras se efectúa la copia.

Merece la pena mencionar aquí la razón para colocar las nuevas rutinas por debajo del HIMEM. La disposición de la RAM de sistema ya quedó descrita, y se mostró que la gran área de RAM libre que está en los tres bloques de 16 K de la parte inferior está a disposición del BASIC. De hecho, el límite inferior está fijado (aunque más adelante mostraremos cómo se puede reservar memoria debajo del área operativa del BASIC), y el límite superior se establece con el valor de HIMEM. En realidad, un programa BASIC "crece" a partir del límite inferior hacia arriba hasta HIMEM.

Para reservar espacio para una rutina en código máquina, debemos asegurarnos de que el BASIC no está dispuesto para acceder a la misma área, y por lo dicho anteriormente está claro que esto se consigue bajando el HIMEM. Para que este proceso sea lo más económico posible en espacio de memoria que queda para el BASIC, el HIMEM bajará sólo la longitud de la rutina que deseamos emplear (más su área de datos) y la rutina se cargará en la RAM que ha quedado libre.

Instrucciones de reinicio
El área de memoria entre #0000 y #003F está duplicada en la RAM y en la ROM, de modo que las instrucciones RST pueden ejecutarse con independencia de que la ROM inferior esté activada o desactivada. El Z80 proporciona ocho instrucciones de reinicio, muchas de las cuales son de uso exclusivo del Amstrad para la administración de la memoria. He aquí la lista de sus funciones

Dirección en hexa		Función
0000	RST 0	Restaura (como en la puesta en marcha)
0008	RST 1	Salta a la rutina de los 16 K inferiores. Los dos bytes que siguen la RST se consideran como una dirección de 14 bits (0000 a 3FFF), con los bits 15 y 14 que señalan el estado de activación de la ROM –bit 15 activado = ROM superior desactivada; bit 14 activado = ROM inferior desactivada
0010	RST 2	Llamada lateral - RST seguida de dos bytes; los bits 15 y 14 indican una de las cuatro ROM laterales; los bits del 0 al 13 dan el desplazamiento a añadir a #C000 para obtener la dirección de la rutina
0018	RST 3	Llamada de lejos - llama cualquier rutina en la ROM o RAM. RST + dos bytes que apuntan a la "dirección lejana" de 3 bytes. Bytes dirección lejana 0/1= dirección; byte 2 = byte de selección de ROM así : #00–#FB = número de ROM lateral, activa la superior, desactiva la inferior; #FC = activación superior e inferior; #FD = activa superior, desactiva inferior ; #FE = desactiva superior, activa inferior ; #FF desactiva superior e inferior
0020	RST 4	Lee byte de RAM apuntado por HL - desactiva ROM superior, activa inferior. Al retorno A retiene el byte leído
0028	RST 5	Llamada del firmware - activa ROM inferior y salta a rutina apuntada por los dos bytes que siguen a RST
0030	RST 6	Reinicio del usuario. Se dejan sin usar los bytes #30 al #37
0038	RST 7	Entrada de interrupción. Se deja solo lo mejor. Si usted desea emplear las interrupciones del modo 1 habrá de considerar el empleo de las facilidades de "eventos" del firmware que analizaremos más adelante en este curso

Los medios de producción

Cortesía del doctor Johannes Heidenhain, RFA

Veamos cómo las máquinas controladas por ordenador fabrican productos acabados

En el pasado, los ingenieros leían los dibujos y preparaban a mano sus máquinas-herramienta. Las máquinas de control numérico (CN) introdujeron un proceso más rápido y de mayor precisión: la máquina-herramienta se prepararía a sí misma de acuerdo a instrucciones recibidas en virtud de un lenguaje de instrucciones. Los diseñadores producirían dibujos técnicos incorporando todas las mediciones adecuadas. Los ingenieros de producción traducirían luego los dibujos a programas de CN. La máquina-herramienta tendría, entonces, toda la información necesaria para mecanizar la pieza.

Los programadores de CN disponen de un conjunto de instrucciones estandarizado en la industria para definir los tipos de movimientos que puede realizar una máquina. El programador define un movimiento y luego añade las coordenadas precisas. Asimismo, necesita definir la velocidad a la cual la máquina está girando o rotando. Las tareas complejas a menudo se efectúan por etapas, avanzando a través de una forma estilizada hasta la pieza ya acabada. Ahora este sistema está siendo superado gradualmente por las máquinas controladas numéricamente por ordenador (CNC: *computer numerically controlled machines*), en las que la cinta de papel que siempre ha alimentado a las máquinas de CN se sustituye por una cinta magnética o por discos flexibles. En los más elaborados de estos sistemas, las empresas están considerando la posibilidad de pasar directamente del CAD a programas de CN generados automáticamente.

No todos los sistemas de CAD son suficientemente precisos para hacer esto, pero, en caso de que lo sean, se puede utilizar de forma cabal la base de datos CAD central en todas las etapas de diseño y producción. El ingeniero de producción, quien tiene una biblioteca de máquinas-herramienta a su disposición, puede acceder a la base de datos CAD central, que posee una descripción geométrica completa de la pieza. Elige la máquina-herramienta que necesita para mecanizar la pieza y, a medida que escribe el programa, puede ver cómo la herramienta se va moviendo a lo largo del contorno del dibujo. De esta forma se pueden evitar errores simples pero costosos, como dirigir la herramienta hacia adelante en lugar de hacia atrás.

El uso de la misma base de datos también acelera el proceso, porque cualquier cambio que introduzca el diseñador en el curso del desarrollo se transmite de forma instantánea al ingeniero. Al diseñador se le pueden indicar directamente los fallos de diseño que podrían hacer imposible la fabricación de una pieza.

El control de procesos modernos es esencial para el funcionamiento continuo de casi todas las industrias, y, por lo general, se lleva a cabo mediante conjuntos de microprocesadores. Un sistema de control de procesos consta de cinco componentes cruciales:

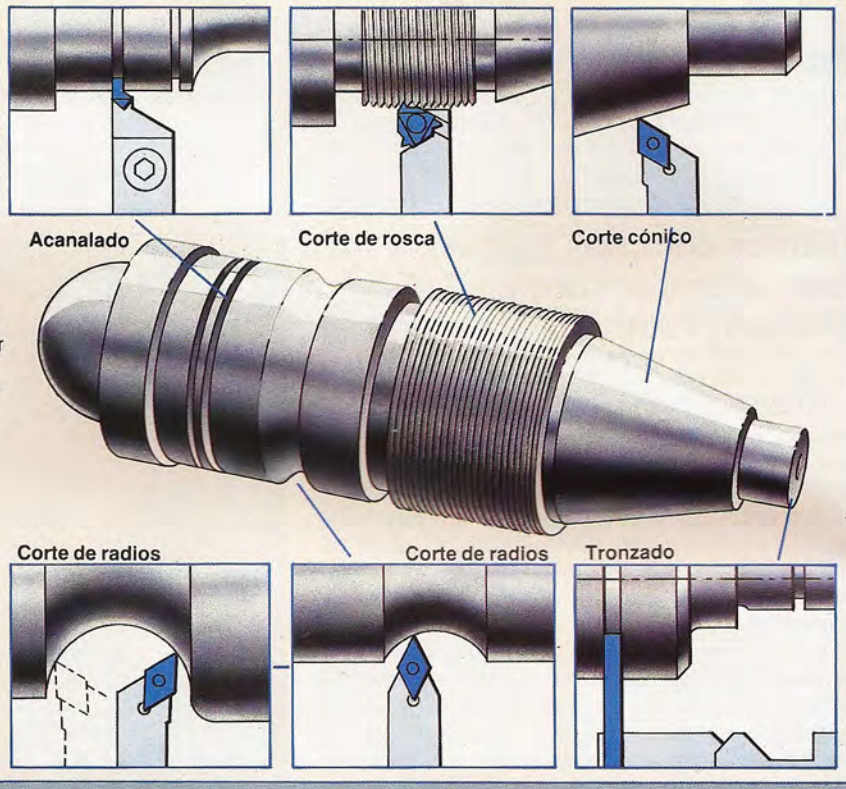
- Un *sensor* para medir la característica determinada, como temperatura, caudal o presión.
- Un *actuador* capaz de iniciar una acción en respuesta a una señal.
- También ha de haber *una válvula de control*, o equivalente. Ésta será activada por el actuador para efectuar el proceso.
- Un *controlador* determina la acción a tomar, basado en la información recibida desde el sensor.

Trabajando

Los progresos en el procesamiento del lenguaje han permitido que se construyan interfaces para máquinas de CNC capaces de aceptar especificaciones en lenguaje corriente. Las primeras máquinas de CN no eran tan eficientes, puesto que necesitaban que un personal altamente entrenado les introdujese una complicada serie de códigos. En el futuro se podrían producir máquinas "inteligentes" que realizaran tareas diferentes en distintos lugares de la fábrica

Herramientas de precisión

Las máquinas de CNC pueden llevar a cabo varias operaciones diferentes sobre el mismo componente mediante la selección de distintas herramientas de corte, tal como le indica el programa maestro, introducido ya sea por el ingeniero o bien directamente desde el sistema de ordenador central. El control por microprocesador permite que una misma máquina domine todas las principales técnicas de mecanizado de un torno convencional con una supervisión mínima



Kevin Jones

Control de procesos

El verdadero trabajo de un sistema de control de procesos lo efectúan los sensores de proceso, que leen, miden e interpretan los datos. La mayoría de ellos contienen microprocesadores.

Medición de temperatura

- *Pila termoeléctrica:* Es la forma más económica y corriente de medir la temperatura. Una pila térmica usa dos metales con distintos coeficientes de dilatación por el calor. Entre ellos se genera una tensión, que es la base de una señal analógica.
- *Termómetro de resistencia:* Se basa en el principio de que el calor aumenta la resistencia eléctrica de los cables conductores. La corriente pasa por un hilo calentado y uno sin calentar. Se mide la diferencia de tensión y se calcula el cambio en la resistencia.
- *Pirómetro de radiación:* Mide el calor radiado y se utiliza para temperaturas más elevadas que los dos dispositivos anteriores. El pirómetro concentra el calor radiado en dispositivos que esencialmente son pilas termoeléctricas y mide la temperatura desde dicho punto de concentración.

Medición de flujo

- *Medidores de presión diferencial:* Calculan la velocidad de flujo midiendo la presión en diferentes puntos a ambos lados de un estrechamiento.

Medición de presión

- *Tubo Bourdon:* Utiliza un tubo de metal en forma de C, en espiral o en hélice. Cuando se aplica presión en un extremo, el tubo intenta enderezarse. La presión se mide por el desplazamiento del extremo libre.
- *Manómetro:* Tubo en forma de U con un brazo más ancho. En éste se monta un flotador. La presión en el brazo más largo y más estrecho varía el nivel de la superficie del líquido en el otro.
- *Diafragma:* Un acoplamiento mecánico mide el movimiento ejercido en un delgado diafragma de goma. En el fuelle hay un sensor más complejo, que se transfiere a la parte inferior y se mide.

Medición de nivel

- *Dispositivos operados con flotador:* Se miden los

- Para hacer esto, el *transmisor* debe tomar la señal del sensor y pasársela al controlador de una forma "comprensible".

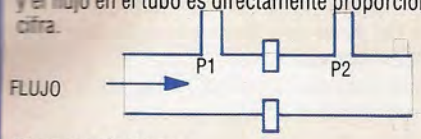
La mayoría de estos dispositivos son analógicos, enviando su información en forma de tensiones. Por este motivo, el sistema contiene un convertidor de analógico a digital que cambia la señal a una forma que le resulte aceptable al ordenador. Los sistemas de control de procesos dependen de tales dispositivos situados en puntos clave de la planta de proceso y casi todos ellos ejecutan el mismo programa una y otra vez. Podrían, por ejemplo, tener que ejecutar un programa una vez por segundo para comprobar el caudal de un flujo en una tubería. Si su velocidad superara los límites establecidos, entonces se habrían de activar las válvulas de control.

Si bien los sistemas de control de procesos son tan diversos como los procesos que controlan, se pueden dividir en dos tipos principales: de control continuo y de control secuencial. Ambos no son mutuamente excluyentes y, de hecho, se pueden incluir en el mismo sistema.

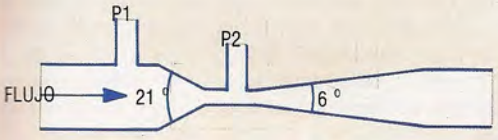
El control continuo ve cómo la materia prima se suministra a un proceso y atraviesa el mismo sin

Flujo rápido

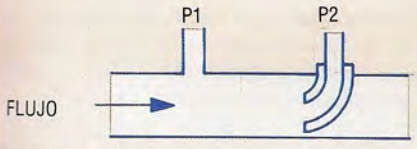
El flujo de líquidos y gases se puede medir y transmitir a un ordenador anfitrión monitorizando la presión diferencial. En cada caso la presión se mide en dos puntos, P1 y P2. La presión diferencial es igual a P1-P2, y el flujo en el tubo es directamente proporcional a esta cifra.



Placa con aberturas



Tubo Venturi

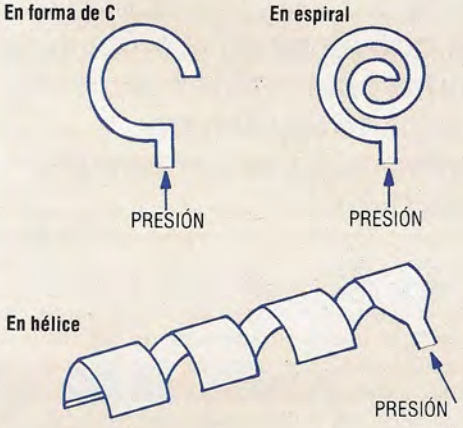


Tubo Pitot

Kevin Jones

Bajo presión

El tubo Bourdon consiste en un tubo metálico que, cuando se le aplica presión al dispositivo, tiende a enderezarse. Entonces se mide el movimiento del tubo para obtener una indicación de presión y el resultado se le transmite al ordenador para su análisis. Los tubos Bourdon se presentan en tres formas: en C, en espiral y en hélice



Kevin Jones

cambios de nivel a medida que el flotador sube o baja.

- *Dispositivos de presión:* La presión en la parte inferior de un depósito mide la cantidad y nivel del líquido contenido.
- *Radiación:* Una célula a uno de los lados del recipiente calcula cuánta radiación gamma está pasando al otro lado desde una fuente. La radiación sólo pasará allí donde el líquido no obstruya su camino y, de este modo, se puede medir el nivel.
- *Sensor:* Calcula la profundidad midiendo el tiempo de respuesta ante una señal ultrasónica.

Medición de peso

- *Células de carga de resistencia eléctrica:* Se basan en que los conductores cambian su resistencia al someterse a tensión mecánica.
- *Células semiconductoras:* Materiales semiconductores, como el silicio, producen tensión bajo presión y pueden usarse como medida.

Análisis químicos

- *Infrarrojos:* Esta forma de análisis se utiliza para medir gases. Se pasan muestras del gas a través de una fuente de infrarrojos hasta un detector de infrarrojos. La cantidad de infrarrojo bloqueado en el camino es una medida del gas presente.
- *Analizadores de conductibilidad térmica:* Los cambios en un gas alteran su capacidad para conducir calor. Se pasa el gas a través de un elemento calefactor a una velocidad constante y se observa si cambia la conducción de calor.

Velocidad

- *Generador taquimétrico:* Mide una velocidad de rotación. El generador se fija a un eje rotativo y, a medida que éste gira, genera corriente. Puesto que la corriente es proporcional a la velocidad, se puede medir esta última.

En los sistemas de control de procesos, la salida de estos sensores, así como la de otros muchos, se aplica a un controlador de tres condiciones. Éste toma el valor medido por los sensores y lo compara con un valor establecido por el operador, que es el nivel al cual se supone debe iniciarse la acción. Si el valor se halla por encima o por debajo del punto establecido, puede activar la válvula y cambiar el proceso en la forma requerida

detenerse. Es necesario el control de la velocidad de flujo a lo largo de todo el camino. El sistema ha de ser capaz de responder a pequeños cambios con el objeto de mantener las condiciones óptimas. En el control secuencial, el proceso se controla por tandas. La materia prima ha de someterse a varios cambios físicos o químicos, de modo que el control se limita a iniciar una serie de acciones en el momento correcto y en las condiciones correctas. Asimismo, debe controlar todas las diferentes etapas para asegurarse de que cada una obtenga el mismo tratamiento.

La gran ventaja del uso de ordenadores en el control de procesos es que permiten hacer muchas cosas al mismo tiempo. Los procesos particulares se pueden controlar individualmente mediante componentes estándares que se pueden intercambiar en caso de una avería. Por sobre todo, el control de procesos informatizado permite que un solo supervisor de planta controle muchas operaciones diferentes desde un único terminal. En vez de tener que leer centenares de diales y medidores, puede seguir la pista de lo que está sucediendo observando los "diagramas simulados" (representaciones estilizadas de la planta y sus procesos) que se visualizan en su VDU.

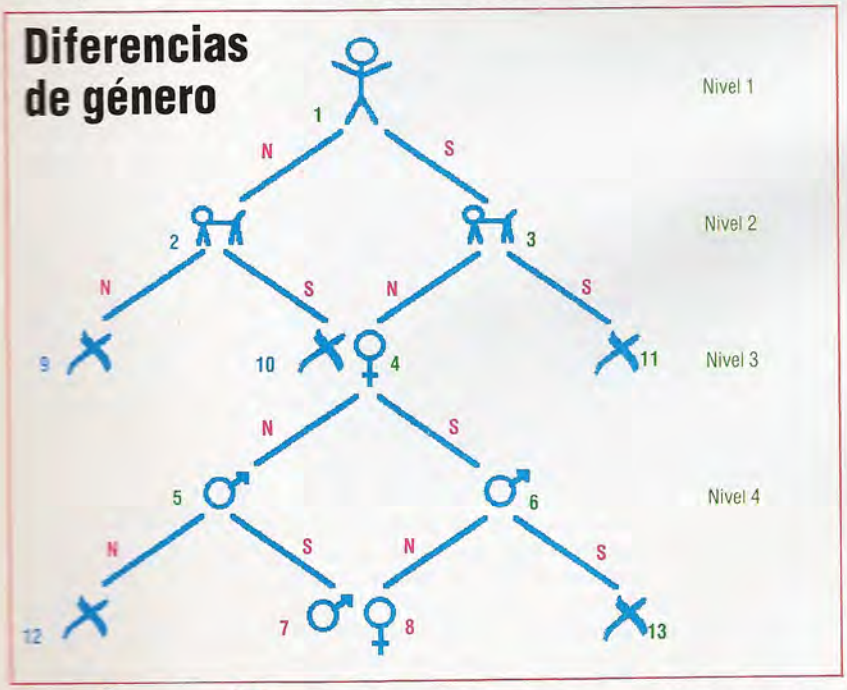
Árbol de decisión

Profundizamos en el empleo de la comprobación de condición como parte de nuestra programación de personajes interactivos

El problema que estudiamos en el capítulo anterior suponía la comprobación de una secuencia de condiciones y la ulterior impresión de un mensaje. Pero supongamos, por ejemplo, que no queremos responder a todas las condiciones, sino sólo a algunas. Esto sería difícil utilizando solamente ON...GOTO, puesto que habríamos de hacer juegos malabares con las distintas condiciones con el fin de terminar con una secuencia de valores para que la sentencia funcionara del modo adecuado. Nuevamente, los árboles de decisión son la respuesta.

Volviendo a nuestro ejemplo anterior de las cuatro condiciones (indicando humano, animal, macho o hembra), el siguiente diagrama muestra un árbol diseñado para comprobar las distintas condiciones y rechazar todas las entradas del usuario, a excepción de aquellas que confirmen que el usuario es un ser humano macho o hembra.

Hombres y mujeres solamente
El descenso a través del árbol que vemos aquí llevará al usuario a un nudo terminal "invalidado", a menos que las preguntas hayan sido respondidas por un humano macho o hembra. Si bien el diseño es semejante al del ejemplo de nuestro anterior capítulo, este árbol posee nudos terminales en niveles distintos, no sólo en el nivel 5. Los nudos de elección se han numerado en color azul, los nudos terminales en rojo y negro. Los números rojos indican resultados válidos (es decir, humanos machos o hembras)



Puede observarse que entre este árbol y el anterior hay ciertas semejanzas. Sigue habiendo cuatro niveles y (remitiéndonos al listado del capítulo anterior) cada nivel comprueba una condición retenida en el elemento de la matriz c correspondiente a ese nivel. Sin embargo, los números de los nudos no guardan la misma relación que antes. Ello se debe a que un nudo terminal no tiene que hallarse necesariamente en la parte inferior del árbol, sino que puede estar en uno de los niveles anteriores, como es el caso del nudo 9, por ejemplo.

Para integrar este nuevo árbol en nuestro programa, primero hemos de numerar los nudos; en este caso, del 1 al 6. Esto nos proporcionará una forma de comprobar, cuando recorramos el árbol, si hemos llegado a un nudo terminal, puesto que cada uno tendrá un número mayor que seis.

En esta etapa podemos simplificar aún más las cosas asegurándonos de que el siguiente conjunto de números que numeremos sean aquellos nudos terminales que requieran que tomemos alguna acción específica: en este caso, los números siete y ocho, cada uno de los cuales representa una de las condiciones que estamos buscando. Por último, numeramos los nudos terminales que no nos interesan.

Las razones que nos mueven a seguir este orden de numeración se harán evidentes más adelante; pero mientras tanto necesitamos representar este árbol dentro de nuestro programa. Para hacerlo, utilizaremos una nueva matriz t(6,1) (t(6,2) en el Spectrum) que retendrá la información de los distintos nudos de elección y de los nudos a los que saltan según de qué condición se trate. La tabla muestra los distintos valores para la matriz t.

Entre el siguiente listado, construido entre las líneas 10 y 80 del listado que ofrecimos en el último capítulo, y las rutinas de bajo nivel de nuestro programa *Dog and Bucket*. Antes ya imprimimos los complementos para las rutinas de bajo nivel en el Spectrum, el Commodore 64 y el BBC Micro. Observe que hemos incluido algunas líneas nuevas entre las líneas 20 y 30, y añadido un número de línea 500 adicional. Estas nuevas líneas leen en la matriz t los datos correspondientes. Asimismo, hemos añadido una sentencia DIM al final de la línea 10, DIM t(6,1).

Tabla del árbol

La matriz t(6,1) —t(6,2) en el Spectrum, que no siempre permite elementos cero en las matrices— retiene los datos necesarios para nuestra estructura arborescente. Si al llegar al nudo 4, p. ej., encontramos que la condición retenida en c(4) es falsa, bifurcamos al número de nudo retenido en el miembro t(4,0) de la matriz. Si es verdadera, la bifurcación se efectúa hasta el número retenido en t(4,1). Los usuarios del Spectrum bifurcarán utilizando t(4,1) y t(4,2), respectivamente

Número de nudo	Falso	Verdadero
1	2	3
2	9	10
3	4	11
4	5	6
5	12	7
6	8	13

```
10 h$="humano ":a$="animal ":m$="macho ":f$=hembra
   ":q$="Eres un ":DIM c(4),t(6,1)
20 GOSUB 4050:REM limpiar la pantalla
22 REM preparar matriz árbol
24 c=6:REM esta es la cantidad de nudos de elección
26 FOR x=1 TO c: READ t(x,1),t(x,0):NEXT x
30 REM preparar las cuatro variables
```

```
  40 PRINT q$;h$;:INPUT i$:c(1)=ABS(i$="s" OR i$="S")
  50 PRINT q$;a$;:INPUT i$:c(2)=ABS(i$="s" OR i$="S")
  60 PRINT q$;f$;:INPUT i$:c(3)=ABS(i$="s" OR i$="S")
  70 PRINT q$;m$;:INPUT i$:c(4)=ABS(i$="s" OR i$="S")
  80 PRINT
 500 DATA 3,2,10,9,11,4,6,5,7,12,13,8
4000 REM
4010 REM subrutinas del sistema de bajo nivel
4020 REM
4030 REM limpiar la pantalla
4040 REM
4050 CLS:RETURN
4060 REM
4070 REM beep
4080 REM
4090 PRINT CHR$(7);:RETURN
4100 REM
4110 REM tomar un caracter del teclado
4120 REM
4130 i$=INKEY$:IF i$="" GOTO 4130
4140 RETURN
```

Ahora añada las siguientes líneas, que recorren el árbol e imprimen un mensaje adecuado:

```
 90 GOSUB 210:REM llamar rutina clasificacion arbol
100 PRINT "Pulsa una tecla para continuar..."
110 GOSUB 4130:REM toma un caracter
120 GOTO 40
200 REM clasifica arbol
210 n=1: k=1:REM empezar en nudo 1,nivel 1
220 n=t(n,c(k)):k=k+1:IF n<=c THEN 220
230 ON n-c GOTO 250,260
240 PRINT "Entrada invalidada...Por favor vuelve a probar":RETURN
250 PRINT "Eres un hombre...":RETURN
260 PRINT "Eres una mujer...":RETURN
```

Si ejecuta este programa, verá que rechaza todas las entradas excepto aquellas efectuadas por "auténticos" hombres o mujeres. El árbol se recorre de forma similar a nuestro primer ejemplo, pero con las siguientes modificaciones. Primero, no sabemos necesariamente cuántos niveles descenderemos antes de encontrar un nudo terminal, de modo que no podemos utilizar un bucle simple FOR n TO numero de niveles. En cambio, inicializamos una nueva variable, k, en la línea 140, para llevar el registro del nivel en donde nos hallamos.

Habiendo establecido n para retener el número del nudo inicial (1), descendemos entonces por el árbol en la línea 150, utilizando la fórmula n=t(n,c(k)) para tomar el siguiente número de nudo de la matriz t. Luego descendemos un nivel (k=k+1) y, si el número de nudo actual es menor o igual que el número de nudos de elección (n=c), sabemos que aún no hemos llegado a un nudo terminal, de modo que saltamos hacia atrás hasta el comienzo de la línea y repetimos el proceso.

Ahora puede ver por qué los números están numerados como describíamos más arriba. La división de los nudos terminales en dos categorías (aquellas que representan las dos condiciones finales [macho y hembra humanos] que queríamos captar seguidas por aquellas que queríamos rechazar) nos permite utilizar una sentencia ON...GOTO para imprimir los diferentes mensajes. Como ya hemos señalado, el principal inconveniente del empleo de ON...GOTO es que requiere una secuencia de valores que a menudo resulta difícil acomodar, pero el árbol nos permite hacerlo sin ninguna dificultad.

Pluralidad de condiciones

Hasta ahora nos hemos concentrado en estructuras que sólo tienen una cosa en común: todas las condiciones comprobadas en cada nivel de un árbol han sido las mismas. Esto ha simplificado las cosas, porque no hemos tenido que averiguar qué condición comprobar en cada nudo. Si, por ejemplo, nos encontráramos en un nudo en el nivel 3, sabríamos que la condición a comprobar está retenida en c(3). Lamentablemente, las cosas no siempre son tan sencillas. A medida que aumenta la cantidad de condiciones a comprobar, la estructura de árbol necesaria para tratar con ellas se vuelve inevitablemente más compleja. No obstante, y ésta es una de las grandes ventajas de utilizar este método para comprobar condiciones, los árboles más complejos no ocupan necesariamente más espacio en su programa.

Los manipuladores de personajes pueden ser tan complejos como pueda hacerlos, y el mejor enfoque consiste en desmenuzar el problema en módulos diferentes, tal como hemos hecho con el cuerpo principal del programa. La primera zona del comportamiento de los personajes que veremos es la forma en que se pueden manipular los objetos en el Dog and Bucket.

Rutinas necesarias y adecuadas

Precisamos, cuanto menos, rutinas para coger, soltar, comer, beber, entregar, recibir y arrojar objetos, y, por supuesto, necesitamos decirle al jugador (si estuviera presente) lo que está sucediendo.

Las rutinas "soltar, arrojar, entregar", por ejemplo, se podrían considerar como un grupo, porque todas ellas dependen de que el personaje comience con un objeto y termine sin él. Para determinar qué rutina es la más adecuada, obviamente necesitamos comprobar condiciones tales como "¿Hay alguien en la habitación?" y, si así fuera, "Ese alguien puede/quiere recibir el objeto?". Asimismo, necesitaríamos comprobar el humor del personaje, para ver si es probable o no que se arrojen objetos, etc. Otras complicaciones incluyen la necesidad de introducir en el proceso un elemento aleatorio, y la necesidad de otras rutinas relacionadas con el mismo (rebajar la energía de un personaje si, p. ej., uno de ellos ha sido golpeado con un vaso de cerveza). En el próximo capítulo dedicado a la programación de personajes interactivos diseñaremos un árbol que tendrá en cuenta todas estas posibilidades.

Complementos al BASIC

Spectrum:

Los complementos para las líneas 4000-4140 ya se han ofrecido en la página 1508. Los usuarios del Spectrum habrán de introducir los siguientes cambios adicionales:

```
10 h$="humano ":a$="animal ":m$="macho
   ":f$="hembra ":q$="Eres un ":DIM
   c(4),t(6,2)
40 PRINT q$;h$;:INPUT i$:c(1)=ABS(i$="s" OR
   i$="S")+1
50 PRINT q$;a$;:INPUT i$:c(2)=ABS(i$="s" OR
   i$="S")+1
60 PRINT q$;f$;:INPUT i$:c(3)=ABS(i$="s" OR
   i$="S")+1
70 PRINT q$;m$;:INPUT i$:c(4)=ABS(i$="s" OR
   i$="S")+1
```

Últimos retoques

Finalizamos nuestro proyecto con algunas sugerencias para perfeccionarlo

Los programas que presentamos se podrían mejorar de muchas formas. Consideraremos algunas de ellas antes de analizar los métodos que utilizan los programas de *go* en los grandes sistemas, algunos de los cuales probablemente se podrían implementar en microordenadores.

Los usuarios del Spectrum, Commodore 64 y Amstrad probablemente habrán notado que estos programas son traducciones de la versión original para el BBC Micro. Aunque en muchos casos se han modificado de forma significativa en razón de las diferencias en cuanto a instrucciones y estructura del programa, se han mantenido intencionadamente lo más similares posible. Su reescritura, utilizando nombres de variables más cortos y sacando mejor provecho de las facilidades de que disponen las distintas máquinas, podría mejorar su rendimiento de modo significativo.

Es poco probable que las versiones para el Spectrum y el Commodore 64 lleguen nunca a funcionar tan rápidamente como la versión para el BBC Micro, debido a que sus BASIC son más lentos. No obstante, quizá desee tratar de convertir algunas rutinas a lenguaje máquina, para acelerar la ejecución. La frecuencia con la que se ejecutan las rutinas se puede comprobar fácilmente colocando una variable de contador al comienzo de cada una, imprimiéndola luego al final del juego. Si hace esto, encontrará que una de las rutinas más críticas es PROCbuscar, que no sólo se utiliza con frecuencia durante el PROCevaluacion de grupos inicial, sino que también se llama al menos una vez para cada llamada a FNlegalidad. Teniendo el tablero (tablero%) establecido como una secuencia de bytes, la conversión a lenguaje máquina resultará bastante sencilla.

En el juego del *go* existen muchas otras tácticas básicas cuya inclusión ciertamente mejoraría el juego del programa. Entre ellas se incluyen configuraciones tales como escaleras, *joseki* (juego de apertura), etc. Una facilidad muy importante que se podría implementar es la de "vida y muerte". El programa ya incorpora una evaluación de "muerte incondicional". Ésta comprueba simplemente si a algún grupo determinado de fichas ya no le quedan licencias, suprimiéndolo en consecuencia del tablero. Dando un paso más hacia adelante, podría fácilmente juzgar si un grupo posee "vida incondicional". Para implementar esto deberíamos:

1. Llenar temporalmente tantas licencias del grupo como fuera legalmente posible con fichas del color contrario.

2. Utilizar luego PROCbuscar para contar las restantes licencias del grupo. Si este valor (clib%) es dos o más, el grupo no se puede capturar; a menos, por supuesto, que el defensor sea suficientemente torpe como para llenar sus propias licencias.

Hay que notar que cuando se llenan licencias deben probarse todas hasta el agotamiento. Si se prueban por orden, puede que no se llenen ojos falsos. Por ejemplo, el grupo quizá tenga una licencia interna que se pueda llenar sólo cuando se hayan llenado todas las licencias exteriores. Habiendo llenado una licencia interna, quizá sea posible llenar otros ojos falsos y verdaderos.

Habiendo implementado la vida incondicional en un grupo, puede pasar a la vida de un grupo en todo el tablero. Esto permite la posibilidad de grupos que compartan ojos. Encontrará un ejemplo de esto si retrocede hasta el primer capítulo de la serie. Para implementar esto, primero debe seguir el procedimiento anterior. De hallar una licencia interna que esté rodeada al menos por un lado por un grupo amistoso diferente, luego la rutina se debe llamar a sí misma recursivamente para comprobar la seguridad de este grupo adyacente, y así sucesivamente. El ojo interno del primer grupo sólo estará a salvo si los grupos adyacentes están a salvo de captura, lo que incide en la seguridad del grupo original.

Si bien el programa se podría mejorar significativamente de muchas maneras, es improbable que llegue a convertirse en un jugador de categoría internacional. Los programas que se ejecutan en grandes sistemas juegan apenas un poco mejor que un recién iniciado, aun después de más de veinticinco años de investigación. Ello se debe a numerosas razones.

Bien guardado

Una útil característica adicional sería la de poder guardar la partida en cinta o disco y retomarla tiempo después. Esto implicaría escribir una nueva rutina PROCguardar_partida, a llamar después de que un jugador digite DEJAR. El procedimiento para guardar tendría que guardar todas las variables de estado del juego y los valores de la tabla de bytes. Ésta se compone de:

tablero% to tablero% + 255	Bytes principales del tablero
estimaciones% to estimaciones% + 255	PROCsuprimir puede haber cambiado los valores de la tabla de estimaciones
movimiento%	No sólo da el número de movimiento, sino que indica el siguiente jugador: impar o par
captura%(2)	Cantidad de fichas capturadas por negras y blancas
ko%	Indica la posición de un punto *ko*, o cero

Asimismo, y de modo opcional, podría guardar:

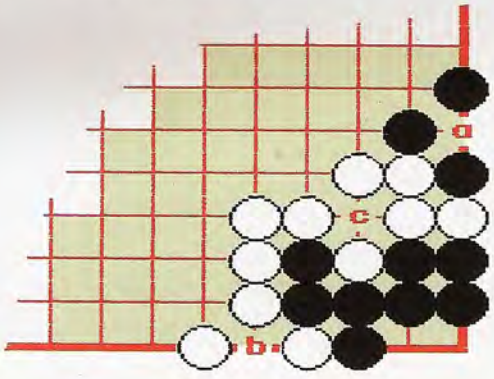
atari1$ y atari2$	Indican una advertencia habitual de "atari" para las negras o las blancas
T$	Método de la última elección de movimiento utilizado por las negras

El propio tablero es fuente de numerosos problemas en virtud de su tamaño. Por lo general se dice que un tablero de 19 por 19 tiene un orden de magnitud de 3^{361}, que es aproximadamente 10^{172}. Esta cifra se calcula como el límite superior de la cantidad de configuraciones de puntos negros o blancos vacantes del tablero. Una evaluación aproximativa de la cantidad media de movimientos potenciales por parte del jugador es 250. Si se aplicara al *go* la técnica tradicional para juegos de la búsqueda arborescente, una búsqueda exhaustiva con sólo tres movimientos de anticipación supondría la generación y evaluación de alrededor de 8 000 000 de posiciones del tablero. Pero informes sobre el juego grabados en cinta de video han revelado que los jugadores aficionados avanzados pueden anticipar la sorprendente cantidad de 30 movimientos, y la literatura sobre *go* con frecuencia contiene secuencias anticipadas aún más profundas.

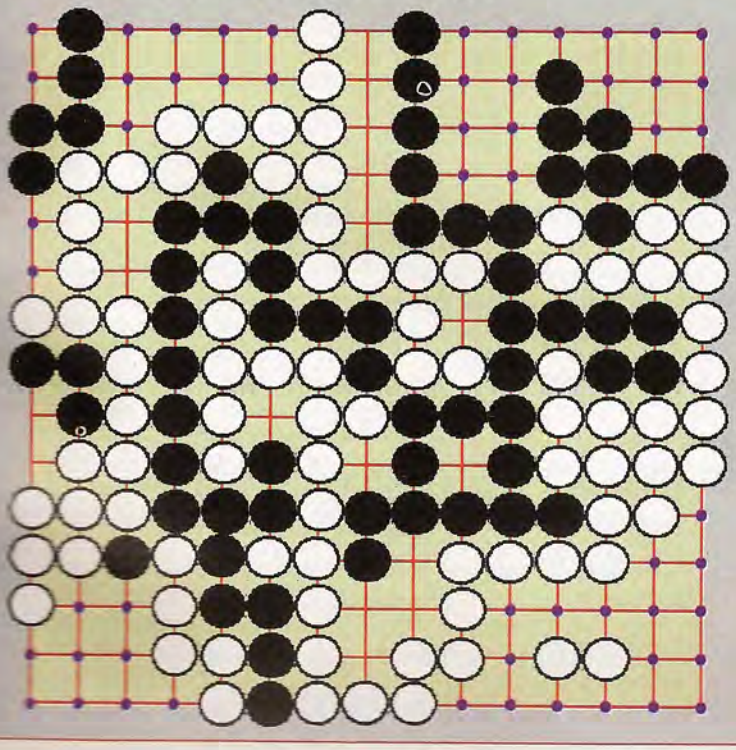
La solución al problema de los árboles grandes consiste en utilizar algún tipo de función de "enfoque". Ésta efectuaría inicialmente una evaluación aproximativa de todo el tablero, al objeto de decidir qué zonas del tablero se hallan en situación crítica. Habiendo elegido una o dos zonas pequeñas, se pueden generar árboles de profundidad total sólo para estos patrones de fichas. Asimismo, los árboles se pueden *podar* utilizando el algoritmo alfabeta y proporcionando rutinas que puedan captar los puntos críticos, las posiciones de los ojos, etc.

Otro de los problemas del *go* es la imprecisión de sus reglas. Por ejemplo, el final del juego se determina cuando ninguno de los jugadores puede obtener ninguna otra ventaja, pero los programas de *go* tienen dificultades para determinar esta condición más bien vaga. Otro problema está relacionado con las condiciones *ko* especiales. Por ejemplo, el diagrama muestra una posición de *triple ko*. Las negras juegan en "b", capturando la ficha blanca de la derecha. Entonces, las blancas capturan la ficha negra situada justo abajo de "a", tras lo cual las negras hacen una captura en "c". Ahora las blancas pueden volver a jugar en la posición de la primera captura, eliminando la ficha negra situada en "a". Las negras recapturan la ficha blanca de "a", y así sucesivamente. En el transcurso de una partida normal, si ninguno de los jugadores está deseoso de abandonar la lucha local jugando en algún otro sitio, entonces el juego se cancela. Aunque no es habitual, se pueden producir situaciones aún más complejas. Para reconocerlas, un programa de *go* habrá de almacenar todas las posiciones del juego anteriores de modo que puedan cotejarse con la posición actual.

En la actualidad los programas de *go* no juegan bien. No obstante, hace muy pocos años eran pocas las personas que reconocían que los ordenadores llegarían algún día a jugar una buena partida de ajedrez. Ahora la cuestión es si un programa de ajedrez por ordenador llegará alguna vez a alcanzar el estatus de campeón del mundo.



Marcador al final del juego



Al final del juego se debe evaluar el tablero y otorgar puntos a cada parte según la cantidad de territorio capturado.

El grupo blanco de la esquina inferior derecha del tablero ha rodeado 17 puntos de territorio. No se incluyen las dos fichas blancas dentro de esta zona: sólo se cuentan los puntos vacantes.

El grupo negro de la esquina superior derecha en realidad son dos grupos, ya que entre las dos mitades sólo hay una «conexión» en diagonal. Sin embargo, todas las salidas están cortadas por fichas negras y, por lo tanto, las negras pueden sumar 17 a su marcador.

Las blancas rodean la esquina inferior izquierda, pero aquí hay una ficha negra suelta. Las reglas del *go* afirman que en realidad no es necesario capturar esta ficha durante el juego. Es en este punto cuando se elimina la ficha, dejando 10 puntos de territorio para las blancas. Al marcador de éstas se le suma un punto para dar cuenta de la ficha negra eliminada.

El bando de las negras parece haber capturado la esquina superior izquierda, pero él mismo se halla rodeado y cortado por un grupo de blancas más numeroso. En consecuencia, se puede eliminar del tablero al grupo de negras y las blancas consiguen 18 puntos territoriales y otros 5 más por las fichas negras capturadas. Todos los otros puntos vacantes son neutrales, porque no forman zonas de territorio rodeadas exclusivamente por ninguno de los colores

Caroline Clayton

Datos básicos (IX)

Por cortesía de la Commodore Business Machines, reproducimos otro fragmento del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
SAREG	030C	780	Almacenamiento para 6502. Registro A
SXREG	030D	781	Almacenamiento para 6502. Registro X
SYREG	030E	782	Almacenamiento para 6502. Registro Y
SPREG	030F	783	Almacenamiento para 6502. Registro SP
USRPOK	0310	784	Instrucción salto función USR (4C)
USRADD	0311-0312	785–786	Byte *lo*/byte *hi* de la dirección USR
	0313	787	No utilizado
CINV	0314-0315	788-789	Vector: Interrupción hardware IRQ
CBINV	0316-0317	790-791	Vector: Interrupción instrucción BRK
NMINV	0318-0319	792-793	Vector: Interrupción no enmascarable
IOPEN	031A-031B	794-795	Vector rutina núcleo OPEN
ICLOSE	031C-031D	796-797	Vector rutina núcleo CLOSE
ICHKIN	031E-031F	798-799	Vector rutina núcleo CHKIN
ICKOUT	0320-0321	800-801	Vector rutina núcleo CHKOUT
ICLRCH	0322-0323	802-803	Vector rutina núcleo CLRCHN
IBASIN	0324-0325	804-805	Vector rutina núcleo CHRIN
IBSOUT	0326-0327	806-807	Vector rutina núcleo CHROUT

El gran competidor

Inmerso en el entorno GEM y basado en el procesador Motorola 68000, el flamante Atari 520ST parece anunciar una nueva era para los microordenadores

Chris Stevens

Es probable que 1985 se constituya en una especie de línea divisoria para la industria del microordenador personal, el año en el que el crecimiento casi exponencial de años anteriores comenzó a disminuir y la industria empezó a madurar. Esto se puede deducir a partir de los problemas dados a conocer públicamente por varios fabricantes de microordenadores, que, a su vez, han fijado un definido margen de precaución en el área de marketing: la mayoría de los productos nuevos han sido versiones remozadas de máquinas de éxito, como el BBC B+, el Commodore 128 y el Atari 130XE.

Sin embargo, 1985 también podrá recordarse como el año en que Atari resurgió de las cenizas para volver a establecerse como uno de los principales fabricantes de micros personales. Esto se debe, en parte, a la influencia del nuevo propietario de la empresa, Jack Tramiel, quien no sólo aportó una aguda perspicacia comercial, sino que también generó, como uno de los "personajes" de la industria, un gran interés en los medios de comunicación.

Por sí solo esto no habría sido suficiente para salvar a Atari, dado que su principal problema era su línea de productos, que, esencialmente, se consideraba pasada de moda. El lanzamiento del Atari 520ST cambia radicalmente esta situación. El ordenador está diseñado alrededor del procesador de 16 bits Motorola 68000, el mismo utilizado en el Apple Macintosh. Este chip por lo general se considera el más avanzado de los disponibles actualmente en grandes cantidades, con arquitectura interna a gran escala de 32 bits, 17 registros de 32 bits, un bus de datos de 16 bits y un bus de direcciones de 24 bits. Obviamente, en base a estas especificaciones se puede diseñar un ordenador muy potente, que es precisamente lo que ha hecho Atari.

Producido en el mismo estilo y aspecto que el 130XE, el 520ST tiene un acabado de resistente plástico gris. El teclado está dividido en cuatro secciones, encima de las cuales hay 10 teclas de función programable, empotradas en la carcasa como las del 130XE. Debajo de éstas hay un teclado estándar, con la adición de una tecla Alternate. Cuando no está conectado el ratón, ésta se utiliza para controlar el cursor de pantalla.

A la derecha hay un racimo de cursor, junto con otras teclas para el editor de pantalla, como Clear e Insert. Justo arriba del racimo del cursor están las teclas Help y Undo y, a la derecha del teclado, hay un relleno numérico con teclas para las funciones aritméticas.

Alejándose de su práctica anterior, Atari ha adoptado para el ordenador varias interfaces "estándares"; las máquinas anteriores de Atari tenían sus propias puertas en serie para todos los periféricos. El Atari 520ST tiene una interface en paralelo Centronics para conexión a impresoras y una puerta en serie RS232 tipo D de 25 vías para la instalación de un modem u otro dispositivo en serie.

Lo más interesante es que el ordenador se ha equipado con una interface MIDI. Ésta está insta-

A la reconquista del mercado
El 520ST representa el intento de Atari por recuperar el lugar que ocupó en el mercado de microordenadores a finales de los años setenta. Basado en el procesador Motorola 68000, el ordenador posee numerosas configuraciones avanzadas, como 512 Kbytes de RAM, puertas MIDI y el sistema operativo GEM. Hasta ahora, el GEM sólo había estado disponible en máquinas mucho más caras

lada como un par de conectores DIN (para MIDI IN y MIDI OUT), lo que significa que el ordenador puede controlar directamente la operación de sintetizadores y otros instrumentos musicales. No obstante, las puertas MIDI tienen otros usos. Puesto que la MIDI es un dispositivo en serie rápido (31,25 Kbaudios), también se puede emplear como método alternativo para la transmisión de datos entre ordenadores. Atari pretende sacar partido de esta facilidad para producir un sistema en red de área local (LAN) utilizando las puertas MIDI.

Otra facilidad interesante que se le ha añadido al nuevo ordenador es una interface para "disco rígido". En la actualidad Atari desea estar entre los primeros en lanzar un reproductor "CD-ROM". Éste es un desarrollo del disco compacto que permitirá almacenar hasta 800 Mbytes de información en un único disco. Por supuesto, ahora mismo los discos compactos son dispositivos de lectura solamente, pero su potencial es extraordinario.

En el precio de la nueva máquina Atari se incluye una unidad de disco, que se enchufa en la única interface para disco flexible. Si se requiere otro disco, se puede enchufar en la primera unidad de disco. No se proporciona puerta para cassette. Atari ha optado por los discos Sony de 3 ½ pulgadas, que se están haciendo sumamente populares para una amplia gama de micros. Las unidades de disco se están produciendo en versiones de una sola cara o de doble cara que, al formatearlas, proporcionan 320 K por cada cara.

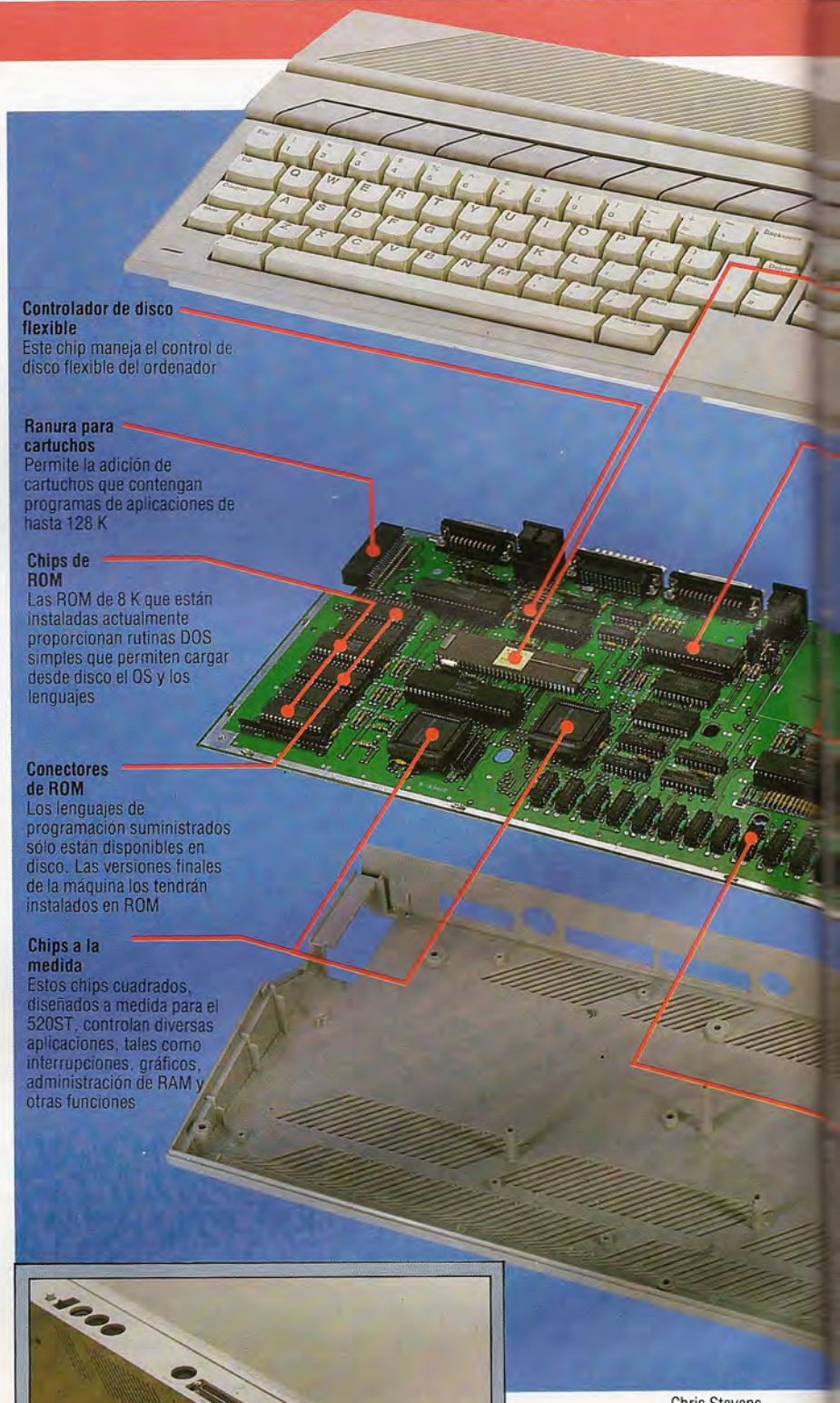
Completando la lista de interfaces que vienen en el ordenador hay un conector de potencia, un monitor RGB y puertas RF (en futuras versiones de la máquina), una puerta para cartuchos capaz de utilizar ROMs de 128 K, y un par de puertas para palanca de mando sobre el lado derecho, aunque éstas no están pensadas específicamente para tal fin, sino más bien para controladores de ratón.

Entorno GEM

Ésta es la última característica que explica el entusiasmo que ha despertado este ordenador. El 520ST es el primer micro de precio reducido que se introduce con el entorno GEM como extremo frontal estándar para el sistema operativo. Desarrollado por Digital Research, el GEM (siglas de *Graphics Environment Manager*: administrador del entorno gráfico) proporciona un sistema WIMP (*Windows, Icons, Mouse Program*), como el que ha obtenido tanto éxito en el Apple Macintosh. De hecho, el sistema GEM, tal como está implementado en el nuevo micro Atari, guarda una sorprendente semejanza con el del Macintosh.

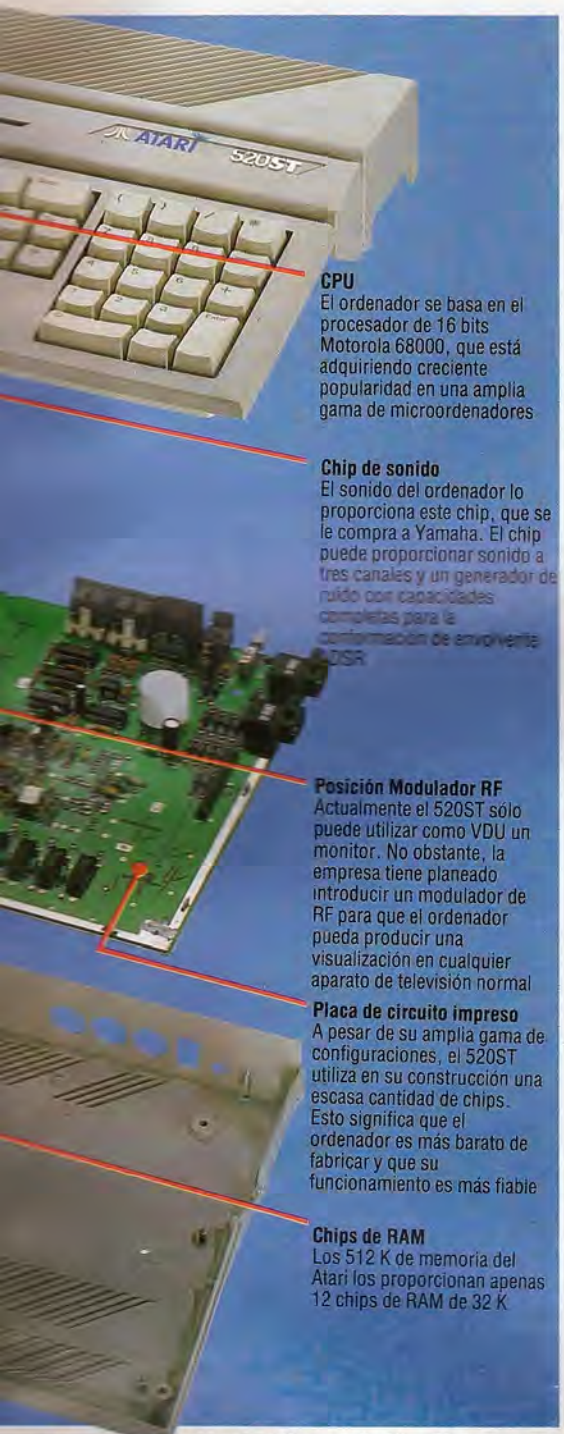
El control del sistema se proporciona desplazando un cursor de flecha (que cuando la máquina está "atareada" se convierte en el icono de una "abeja") por la pantalla a varios iconos. Pulsando el botón del ratón se selecciona el icono determinado. En la parte superior de la pantalla también hay disponibles unos menús que se pueden "bajar" como ventanas. Por ejemplo, una unidad de disco se selecciona eligiendo el icono del "mueble archivador" situado en la esquina superior izquierda de la pantalla inicial. Luego se pueden realizar diversas manipulaciones de archivo tirando del menú "archivos"; "opciones" visualiza el directorio.

El GEM visualiza dos tipos de archivo. Los ico-



Chris Stevens

Algo para todos
Atari ha abandonado el sistema por el cual sus ordenadores sólo se podían dotar de los propios periféricos de la compañía, y ha instalado en el 520ST varias interfaces "estándares". Por su gama de puertas, la máquina es una de las mejor equipadas que existen ahora en el mercado. Entre estas puertas se incluyen una RS232, puertas Centronics y MIDI



nos "carpeta" denotan "directorios", mientras que los iconos cuadrados representan archivos de programas. El programa se carga (LOAD) y ejecuta (RUN) con sólo mover el cursor hasta el archivo. Si quiere borrar un archivo, sólo tiene que desplazar el icono en cuestión hasta el icono de la "papelera". La facilidad con que hasta un novato puede aprender a emplear el GEM es particularmente evidente para cualquiera que haya utilizado alguno de los sistemas operativos más convencionales, como el CP/M y el MS-DOS.

El 520ST dispone de tres modalidades para gráficos: alta, media y baja resolución. La modalidad más alta posee una resolución de 640 por 400 pixels, pero sólo aparece en monocromático, lo que le confiere a la máquina un aspecto aún más similar al del Mac. Las modalidades de menor resolución demuestran las capacidades de color de la máquina, pero muchos usuarios preferirán la modalidad monocromática de alta resolución. Atari está ofreciendo para el 520ST la opción de una pantalla monocromática o bien un modelo en color.

El entorno de "escritorio" del GEM (así llamado porque uno desplaza los iconos de forma muy similar a como movería los papeles por la superficie de un escritorio), opera bajo TOS (*Tramiel operating systems*: sistema operativo Tramiel). Ésta es una versión construida a la medida del sistema operativo CP/M 68 de Digital Research, en sí mismo un desarrollo del popular OS de ocho bits de Digital Research, que se ha vuelto a diseñar para máquinas basadas en el 68000. Sin embargo, a diferencia del CP/M 68, el sistema operativo permite organizar archivos en "directorios" como el MS-DOS, el sistema operativo de 16 bits que es el estándar *de facto*.

El 520ST tiene empaquetados algunos programas de aplicaciones. Primero está *GEM Write*, un paquete de tratamiento de textos, y *GEM Paint*, que, como su nombre sugiere, es un paquete para diseñar gráficos. Similar a los equivalentes *MacPaint* y *MacDraw* del Macintosh, el paquete ofrece varios tamaños de "pincel" y diseños de "relleno", así como una facilidad de *zoom* que permite que el usuario enfoque una pequeña zona de la pantalla con el fin de producir un trabajo especialmente minucioso. Si bien el *GEM Paint* carece de algunas de las facilidades del Macintosh, es, así y todo, un programa muy sofisticado.

Lenguajes basados en ROM

En el ordenador se incluyen también los lenguajes de programación ST-BASIC y ST-LOGO, que en las primeras máquinas se proporcionaban en disco, pero en ulteriores versiones aparecerían en ROM en la tarjeta. Como el TOS, los lenguajes son desarrollos de productos anteriores de Digital Research; el BASIC es la versión propia de DR del BASIC Microsoft, conocido como Personal BASIC, mientras que el LOGO es una versión del popular Dr LOGO, ahora incluido en algunas máquinas. Los lenguajes han sido adaptados para sacar partido de las capacidades del entorno GEM para ventanas.

El Atari 520ST es indudablemente una máquina importante, no sólo para la propia Atari Corporation, sino también para el mercado de ordenadores "económicos". El sistema completo incluye una unidad de disco, una pantalla monocromática y un ratón, así como el software empaquetado. Por su precio está al alcance del usuario personal "serio" y el de pequeña gestión, aunque tal vez sea demasiado caro para generar ventas muy grandes, de las que depende el esencial apoyo de software.

Junto con los paquetes GEM y los 512 K de memoria, el usuario obtiene una configuración similar al *Fat Mac* de Apple por la mitad de precio. En el caso de que saliera una versión más pequeña del ordenador, con, por ejemplo, 256 K de memoria, las ventas resultantes serían muchísimo mayores. Sin embargo, una cosa es cierta: cuando el 520ST esté disponible en cantidades suficientes como para hacer un impacto en las zonas comerciales, el mercado de ordenadores económicos habrá entrado en una nueva era.

ATARI 520ST

DIMENSIONES
470 × 240 × 60 mm

CPU
Motorola 68000 trabajando a 8 MHz

MEMORIA
512 Kbytes de RAM

PANTALLA
Tres modalidades de resolución: la más alta es la modalidad monocromática, con 640 × 400 pixels

INTERFACES
MIDI (dos conectores DIN), interface en paralelo Centronics, puerta en serie RS232, interface para disco flexible, conector de potencia, monitor RGB, RF, puertas para cartuchos y palanca de mando. También hay una interface para "disco rígido" para futuras reproductoras de CD-ROM

UNIDAD DE DISCO
Una unidad de disco Sony de 3 ½ pulgadas y doble cara

SISTEMA OPERATIVO
TOS y GEM como estándar

LENGUAJES DISPONIBLES
ST-BASIC y ST-LOGO

TECLADO
84 teclas más 10 teclas de función

DOCUMENTACIÓN
El manual del usuario está muy bien elaborado, con un gran número de diagramas e instantáneas de pantallas acompañando un texto conciso

VENTAJAS
El ordenador ofrece tecnología de 16 bits, mucha memoria para el usuario y un sistema operativo amable con el usuario que previamente sólo había estado disponible en máquinas mucho más caras

DESVENTAJAS
A pesar de sus notables especificaciones, el sistema aún ha de probarse a sí mismo, y quizá el ordenador sea aún demasiado caro como para generar grandes ventas, de las que depende el vital soporte de software

Punto a punto

Comenzamos la construcción del tester ocupándonos del montaje de los componentes pasivos

Lista de componentes

Cant.	Ref.	Artículo
1	C1	condensador de polipropileno de 0,47 µF (1)
2	C2-3	condensador de policarbonato de 1,0 µF
1	C4	condensador de policarbonato de 0,1 µF
1	C5	condensador de poliestireno de 470 pF
5	C6	condensador de disco cerámico de 22000 pF
1	D1	diodo de ref. de 1,22 V de intervalo de banda 9491 (2)
1	D2	diodo 1N4148
6	TR1-5,TR7	transistor NPN 2N2219
1	TR6	transistor PNP 2N2905
1	VR1	potenciómetro cermet multivueltas de 10 k-ohmios
1	VR2	potenciómetro preajustado cermet de 10 k-ohmios
1	IC1	chip convertidor A/D 7135
1	IC2	temporizador de baja potencia ICM7555
1	IC3	activador de 7-segmentos 7447A
1	—	zócalo DIL de 28 patillas
2	—	zócalo DIL de 16 patillas
1	R1	resistencia de 8,2 k-ohmios (3)
7	R2-8,R14,R17	resistencia de 180 ohmios
1	R9	resistencia de 47 k-ohmios
3	R10-12	resistencia de 100 k-ohmios
1	R13,R15,R16	resistencia de 4,7 k-ohmios

Notas:
1) El condensador C1 puede sustituirse por uno de policarbonato (con alguna pérdida de prestaciones)
2) Este diodo puede sustituirse por un diodo zener de 2,7 V (con alguna pérdida de prestaciones)
3) Se deben utilizar resistencias de 5 % de tolerancia, de carbón o de película metálica

Para la construcción del módulo básico recomendamos utilizar una placa matriz de topos de 0,1 pulgadas en lugar de las placas de circuito impreso de tiras. Esto se debe a que éstas comprometen el trazado del cableado una vez y para siempre, mientras que el cableado punto por punto de una placa de topos permite la introducción de ligeras modificaciones en el posicionamiento de las conexiones para optimizar el rendimiento.

A diferencia de los circuitos digitales comunes, un convertidor A/D tiene entradas analógicas muy sensibles (alta impedancia de entrada). Esto significa que las señales de interferencia y no deseadas pueden introducirse en las entradas analógicas a menos que se tomen medidas especiales. En nuestro diseño todo el sistema de circuitos analógico se mantiene en uno de los lados del chip convertidor A/D (patillas de 1 al 10), y todo el sistema de circuitos digitales en el otro. Aun así, durante la construcción del prototipo encontramos que unas ligeras modificaciones en la posición del cableado podrían mejorar de modo significativo el rendimiento del circuito. Idealmente, tal circuito utilizaría una placa de circuito impreso comprobada, pero su construcción está fuera del alcance del montador medio, de modo que hemos optado por un cableado punto a punto.

El esquema del cableado muestra una vista desde arriba de la placa del circuito; los conductores de interconexión estarán en la cara inferior de la placa. El trazado del cableado está muy estilizado para que resulte más claro, mostrando sólo algunos conductores conectados directamente de un punto a otro.

Este trazado angular responde sólo a razones de claridad y no es necesario respetarlo exactamente. A pesar de que ofrecería un mejor aspecto, es mejor no montar los conductores muy tirantes entre dos puntos, para poder desplazarlos más adelante para optimizar el rendimiento global.

La placa prototipo se cableó empleando terminales para hilo arrollado (*wire-wrapping*) y una herramienta de arrollado especial para conectarlos. Esta técnica es muy conveniente, porque si usted conecta un cable de modo erróneo, puede desarrollarlo y volverlo a conectar con toda facilidad. La alternativa a este método es utilizar hilo de conexión delgado corriente y un soldador.

La mayoría de los chips semiconductores tienen patillas con 0,1 pulgadas (2,45 mm) entre centros, de modo que la placa de circuito más adecuada es una placa matriz de 0,1 pulgadas con cada orificio rodeado por un topo de cobre para realizar la soldadura. Existen versiones sin cobre de este tipo de placa, pero indudablemente es más fácil trabajar con el tipo que tiene topos de cobre.

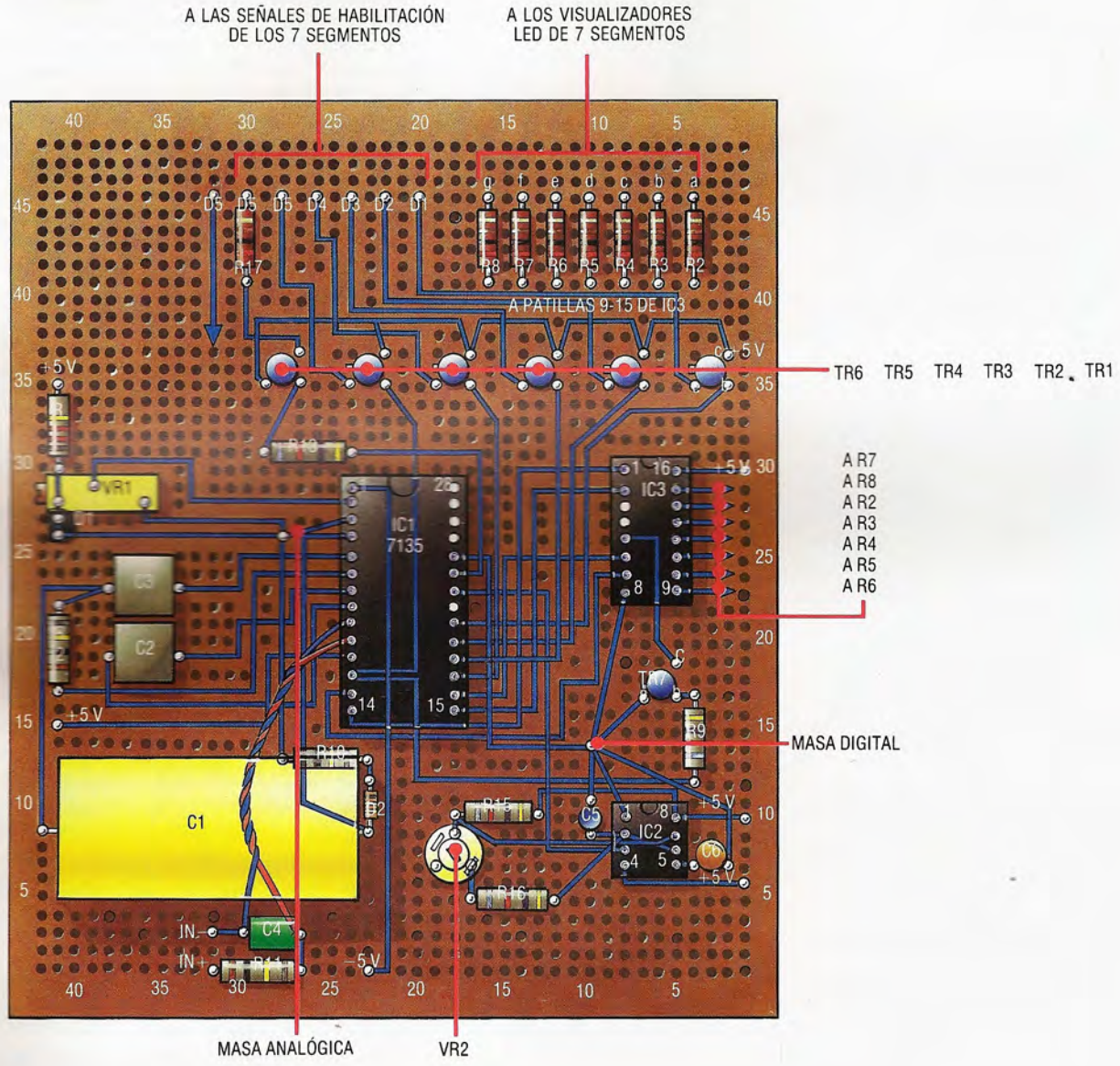
Para que haya amplio espacio para los componentes, la placa matriz ha de tener alrededor de 50 filas por 45 columnas de agujeros. Comience por insertar los tres zócalos de IC: el zócalo DIL de 28 patillas para el convertidor A/D 7135 (ICI), un zócalo de ocho patillas para el chip temporizador 7555 (IC2) y un zócalo de 16 patillas para el activador de siete segmentos 7447A (IC3). Si se utiliza una placa matriz con cobre, suelde dos patillas en esquinas opuestas de cada zócalo DIP por la parte posterior de la placa. Si está utilizando la placa matriz de tipo sin cobre, una pequeña gota de soldadura en las patillas de las esquinas mantendrá a los zócalos en su sitio.

Antes de montar ningún otro componente, suelde o arrolle las líneas de +5 V (Vcc) a los tres IC. Por razones de claridad, no se muestra la línea Vcc completa, pero finalmente todos los puntos de +5 V (como la patilla 16 de IC3) se conectarán juntas y serán alimentadas por la fuente de alimentación. Sólo un chip, el 7135, necesita una fuente de −5 V (Vee). Ésta va desde la patilla 1 de IC1 hasta un punto adecuado en el borde de la placa principal, como se indica.

Puntos de masa

Se necesitan dos puntos de masa separados: uno para la masa analógica (situado junto a la patilla 4 de IC1) y uno para la masa digital. Como ya hemos dicho anteriormente, es importante conectar todas las masas analógicas a un punto único de masa analógica, y todas las masas digitales a otro punto único de masa digital. Conecte la masa analógica de IC1 (patilla 3) a la masa analógica y las masas de IC2 e IC3 (patillas 1 y 8, respectivamente) a la masa digital. La masa digital de IC1 (patilla 24) también se conecta al punto de masa digital.

Disposición de los componentes



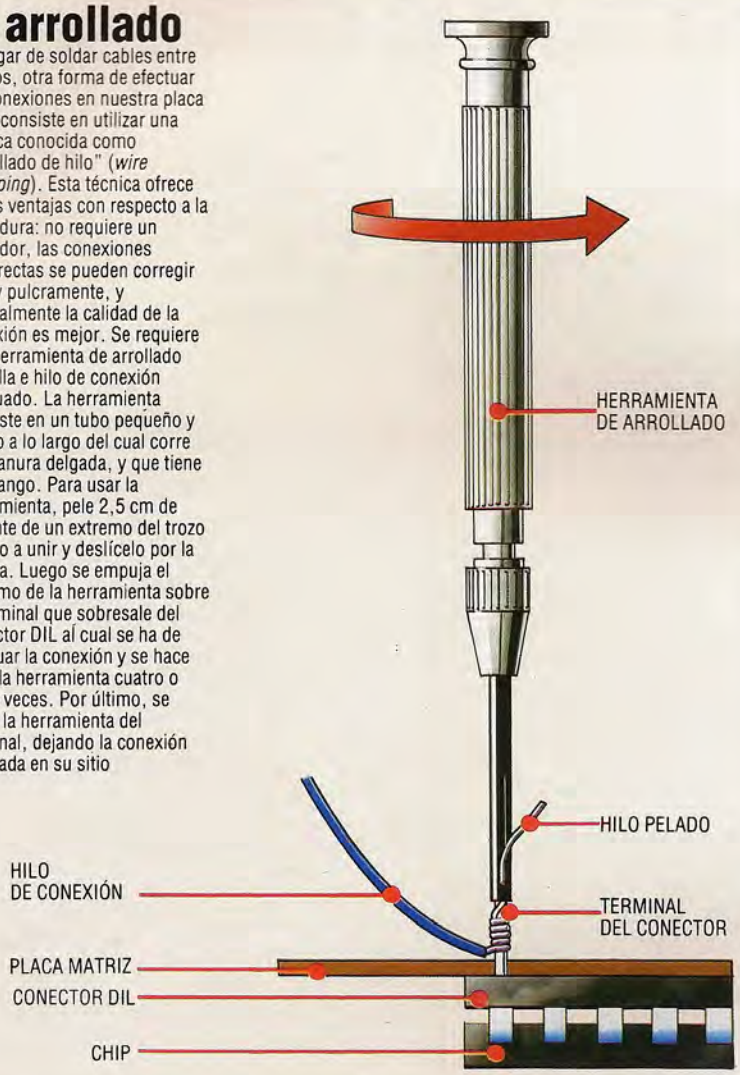
Kevin Jones

Montaje de componentes

En el esquema del montaje vemos los componentes montados en una placa matriz de 0,1 pulgadas. El cableado punto a punto se muestra en estilo angular para facilitar la comprensión. En la práctica, el cableado entre puntos de la placa debe quedar suficientemente suelto como para permitir el movimiento de los cables, ya que la posición del cableado puede incidir en el rendimiento del tester. Observe que no se indican explícitamente las conexiones entre puntos analógicos y digitales a masa, de D5 a la masa digital y todas las líneas de alimentación de +5 V. La conexión entre las masas analógica y digital se debe efectuar con hilo de cobre bastante grueso. El método más simple de proporcionar las alimentaciones de +5 V consiste en conectar todas estas conexiones a un único trozo de cable que corra alrededor del borde de la placa. Como se menciona en el texto, si las tensiones a medir son con respecto a masa, se debe conectar la patilla 9 del chip 7135 al punto de masa analógica. Suelde todos los componentes pasivos en su sitio, tal como se indica, pero todavía no monte los chips en sus conectores, porque éstos pueden dañarse fácilmente. En el próximo capítulo nos ocuparemos de esta parte de la construcción. Asimismo, se deben conectar condensadores de disco de 22 000 pF entre las fuentes de alimentación del chip y masa. En el IC1 conecte un condensador entre la patilla 1 y el punto de masa analógica y otro entre la patilla 11 y la masa analógica. En el IC2 conecte un condensador entre la patilla 8 y la masa digital, y en el IC3 entre la patilla 16 y la masa digital. También se debe conectar un condensador electrolítico de 47 µF entre la patilla 1 de IC1 y la masa analógica. Observe que en este tipo de condensador la orientación tiene importancia. Asegúrese de conectar el terminal negativo del condensador a la patilla 1 del chip y el lado positivo a masa

El arrollado

En lugar de soldar cables entre puntos, otra forma de efectuar las conexiones en nuestra placa DVM consiste en utilizar una técnica conocida como "arrollado de hilo" (*wire wrapping*). Esta técnica ofrece varias ventajas con respecto a la soldadura: no requiere un soldador, las conexiones incorrectas se pueden corregir fácil y pulcramente, y normalmente la calidad de la conexión es mejor. Se requiere una herramienta de arrollado sencilla e hilo de conexión adecuado. La herramienta consiste en un tubo pequeño y hueco a lo largo del cual corre una ranura delgada, y que tiene un mango. Para usar la herramienta, pele 2,5 cm de aislante de un extremo del trozo de hilo a unir y deslícelo por la ranura. Luego se empuja el extremo de la herramienta sobre el terminal que sobresale del conector DIL al cual se ha de efectuar la conexión y se hace girar la herramienta cuatro o cinco veces. Por último, se retira la herramienta del terminal, dejando la conexión arrollada en su sitio



Por último, conecte entre sí los dos puntos de masa utilizando hilo de conexión bastante grueso. (Por razones de claridad, esta conexión no se ha ilustrado, ¡pero no olvide realizarla!)

Antes de instalar ningún otro componente, emplee un óhmetro o algún otro comprobador de continuidad sencillo para verificar la continuidad entre el punto de entrada de la fuente de alimentación y las patillas Vcc de los tres IC. De modo similar, compruebe la continuidad entre todos los puntos de masa.

Una vez montados los zócalos de los IC, se pueden insertar los otros componentes en las posiciones que se indican. Los tamaños exactos que se muestran son para unos determinados modelos de componentes especificados en la *Lista de componentes*. La mayor parte de ellos se pueden reemplazar por otros equivalentes, pero las dimensiones pueden ser diferentes y entonces habrá que modificar ligeramente la disposición. El componente más especial de todos es C1: el condensador integrador de 0,47 µF.

Para un mejor rendimiento, éste ha de ser de polipropileno. En el caso de que tenga dificultades para encontrarlo en el mercado, sustitúyalo por un condensador de policarbonato (que funcionará, aunque no tan bien), o bien encuentre un distribuidor de componentes electrónicos que lo pida a fábrica.

Otro componente que puede resultar problemático son los LED de siete segmentos. Los más corrientes son los dobles (ya sea de dos "8" o bien de un "+/−1" y un "8"), u "8" individuales, pero no "+/−1" individuales. Esto significa que, utilizando los componentes más corrientes, es imposible construir un visualizador de 4 ½ dígitos capaz de mostrar un auténtico signo más o menos sin tener un dígito sobrante.

En el mercado existen LED adecuados (empleados en el prototipo), pero por si hubiese dificultades para obtener estos componentes (y por su elevado precio), hemos modificado ligeramente el circuito original para hacerlo funcionar con LED de 0,5 pulgadas y dígito simple. Por tanto, se utiliza un LED de siete segmentos corriente para visualizar un vacío o un 1, y también para visualizar el signo menos utilizando el segmento horizontal central (segmento C) cuando la señal que se está midiendo es negativa. Las señales positivas se visualizarán sin ningún signo delante de ellas.

VR1 debe ser un potenciómetro de ajuste multivueltas de buena calidad; VR2 puede ser cualquier potenciómetro de corrección preajustado. Lo ideal es que D1 sea un diodo de referencia de intervalo de banda como uno del tipo 9491. Para evitar la incomodidad que entraña conseguir este componente, se puede sustituir por un diodo zener de 2,7 V. Nuevamente, este componente funcionará, pero con un menor rendimiento global.

Tipos de transistores

Los transistores (con la excepción del TR7) se utilizan para activar los visualizadores LED. Observe que los transistores del 1 al 5 son tipos NPN y que TR6 es de tipo PNP. Esto se debe a que TR6 ha de encender en respuesta a una señal LO lógica, mientras que los otros han de activar en respuesta a una señal HI lógica. TR7 se emplea en el circuito de borrado a cero. D2 puede ser un pequeño diodo de silicio cualquiera, si bien el componente especificado se puede obtener fácilmente.

Conecte los puntos de entrada a las patillas IC1 9 y 10, utilizando un trozo de cable trenzado o algún cable coaxial delgado (malla a la patilla 9, y central a la 10). En nuestro prototipo, se consiguieron resultados mucho mejores conectando la patilla 9 de IC1 a la masa analógica (que no se indica en el diagrama). Se aconseja realizar esta conexión si no se requiere una entrada de punto flotante, dado que las tensiones se miden con relación a masa y no como la diferencia entre tensiones positivas y negativas.

Antes de insertar los IC, compruebe concienzudamente su cableado comparándolo con el esquema presentado. Observe, por ejemplo, que TR6 no tiene aleta y que su colector y su emisor están conectados en sentido contrario a los de TR1 a TR5.

Próximamente nos ocuparemos del cableado del panel visualizador y la manipulación de los IC.

Una manipulación torpe destruirá rápidamente los IC, de modo que le aconsejamos que siga atentamente las instrucciones, a menos que esté usted totalmente familiarizado con el manejo de componentes CMOS.

Característica exclusiva

Llegados a este punto en nuestro curso, centraremos nuestra atención en la manera de ampliar el FORTH para tratar estructuras de matriz

Ya hemos visto algunas de las ideas que es necesario combinar para que uno defina sus propias matrices cuando utiliza CREATE en el programa *La criba de Eratóstenes*; equivalía a:

```
CREATE BITS 8192 ALLOT
```

En ese programa, CREATE incluye un encabezador en el diccionario empleando el nombre BITS desde el teclado, y también incluye un campo de código que hace que BITS, cuando usted la utiliza, deje en la pila la dirección de su campo de parámetros, pfa. No obstante, CREATE en sí misma no hace nada para establecer el campo de parámetros, y es aquí donde se introduce ALLOT. ALLOT toma un número de la pila e incluye en el diccionario espacio para el número de bytes correspondiente. Dado que en este caso se incluyen inmediatamente después del campo de código incluido por CREATE, constituyen el campo de parámetros para BITS.

Potencias de tres

He aquí un sencillo ejemplo que define una palabra, 3**, para calcular potencias de tres. Para que sea rápida, emplea una matriz que contiene todas las respuestas posibles, y se prepara utilizando CREATE con la palabra del FORTH, (una coma):

```
VARIABLE INDICEMAX
:POTENCIAS,(--)
     (incluye todas las potencias de tres que pueden
     caber en dos bytes, y coloca en INDICEMAX el
     índice de la potencia mayor.)
   -1 INDICEMAX 9
   1          (potencia más pequeña)
   BEGIN
        DUP,(incluir potencia)
        1 INDICEMAX+!
        3 UM* (siguiente potencia, doble longitud)
              (reemplazar UM* por U* en FORTH-79 y
              figFORTH)
   UNTIL      (hasta que la siguiente potencia lleve
              más de 2 bytes)
   DROP       (bytes menos significativos de potencia
              de doble longitud)
;
CREATE 3POTENCIAS POTENCIAS,
                (no es necesaria ALLOT., realiza la
                inclusión)
   :3**         (n--3**n)
   INDICEMAX @ OVER U < IF
```

En estrecha formación

Definir una estructura de semimatriz no es difícil gracias a CREATE y DOES>. El diagrama refleja la respuesta de FORTH a la entrada 11 1MATRIZ EJECUTA, donde 1MATRIZ se ha incluido previamente en el diccionario como una def. de dos puntos. También refleja el efecto sobre la pila

Def. de dos puntos	11 1MATRIZ EJECUTA	Pila
:1MATRIZ		
CREATE	CREATE toma EJECUTA, le asigna un c. de nombre y un c. de código. Cuando se utilice EJECUTA, dejará su pfa en la pila	11
DUP +	DUP + multiplica 11 por dos y pone el resultado en la pila	22
HERE	HERE coloca en la pila la siguiente dirección disponible en el diccionario. Puesto que acabamos de incluir EJECUTA, ésta es la pfa de EJECUTA	PFA 22
OVER	OVER copia 22 en la parte superior de la pila, encima de la pfa de EJECUTA	22 PFA 22
ALLOT	ALLOT quita 22 de la pila e incluye 22 bytes en el diccionario, que (dado que acabamos de incluir EJECUTA) constituyen el campo de parámetros para EJECUTA	PFA 22
SWAP 0 FILL	SWAP pone 22 encima de la pfa de EJECUTA; coloca 0 en la pila; luego FILL toma los tres parámetros y pone a 0 los 22 bytes del c. de parámetros	0 22 PFA → VACÍA
DOES>	RUN-TIME (después de DOES>)	
	99 2 EJECUTA! (p. ej.) prepara la pila como vemos y luego almacena el valor 99 en el elemento 2 de la matriz. Se EJECUTA del siguiente modo:	PFA 2 99
SWAP	SWAP coloca en la parte superior de la pila el "subíndice de matriz" (2)...	2 PFA 99
DUP +	DUP + lo duplica...	4 PFA 99
+ ;	+ le suma el subíndice (que, efectivamente, está actuando como un valor decalado) a la pfa de EJECUTA...	PFA +OFF-SET
!	! toma 99 y lo almacena en la dirección correcta	VACÍA

```
        DROP  0 (n no entre 0 e INDICEMAX.
              Dejar resultado 0)
    ELSE        (querer contenido de memoria
                en 3POTENCIAS+2*n)
        2*3POTENCIAS + @
    THEN
;
```

3POTENCIAS era una matriz numérica unidimensional. Usted podría igualmente haberla preparado utilizando ALLOT (recuerde que tiene que asignar dos bytes para cada elemento de la matriz) y !. Las matrices de caracteres son similares, pero puesto que cada carácter representa sólo un byte, usted debe utilizar C,, C! y C@ en lugar de ,, ! y @.

La estructura CREATE

Siempre que emplee una matriz como 3POTENCIAS, necesita hacer algo como:

```
2*3POTENCIAS+
```

Ello se debe a que 3POTENCIAS no sabe que es una matriz. Simplemente deja la dirección de su campo de parámetros y deja que usted escoja qué hacer con ella. Para hacer palabras más "inteligentes", hay una construcción muy interesante que emplea CREATE (o<BUILDS en figFORTH) y DOES>. Éstas permiten definir palabras nuevas que sean versiones mejoradas de CREATE, de modo que en realidad son nuevas palabras definitorias. Ellas sí crean (CREATE), pero usted también puede decirles que hagan otras cosas al mismo tiempo que el CREATE. Asimismo, pueden decirle a usted qué ha de hacer la palabra recién creada cuando la utilice (en vez de limitarse a apilar la dirección de su campo de parámetros).

He aquí un ejemplo que crea matrices inteligentes:

```
:1MATRIZ  (n--)
    (crea una matriz numérica unidimensional
    con dimensión n)
  CREATE          (el nombre de copia de 1MATRIZ
                  cuando usted la usa)
  DUP+            (una forma rápida de hacer 2*)
  HERE            (recuerde la dirección del campo de
                  parámetros para FILL)

  OVER ALLOT      (ahora tiene 2*,pfa)

  SWAP 0 FILL     (poner a cero todos los bytes del
                  campo de parámetros)

DOES>             (subíndice, pfa -- dirección de
                  elemento)
  SWAP DUP + (pfa,2*subíndice)
```

Se divide en dos partes: la parte de tiempo de definición, antes de DOES>, se lleva a cabo cuando usted utiliza 1MATRIZ y prepara el campo de parámetros de la palabra siguiente.

Supongamos que dice:

```
11 1MATRIZ EJECUTA
```

Ésta crea una nueva palabra, EJECUTA, y le destina (ALLOT) un campo de parámetros de 22 bytes inicializados en cero. La segunda parte de 1MATRIZ, después del DOES>, es la parte de tiempo de ejecución. Ésta se realiza cuando se utiliza EJECUTA. EJECUTA coloca la dirección de su campo de parámetros en la pila, pero después de que la parte de tiempo de ejecución de 1MATRIZ aplique su inteligencia.

En efecto, ahora usted tiene 11 variables, denominadas:

```
0 EJECUTA
1 EJECUTA
:
:
10 EJECUTA
```

Las emplea tal como si fueran variables comunes, p. ej.:

```
99 2 EJECUTA !
4 EJECUTA @.
```

Otras aplicaciones

Ésta era una aplicación bastante directa de CREATE...DOES>. Pero una vez que aprenda a usarla, descubrirá que puede hacer cosas mucho más inteligentes. Por ejemplo, si ha comenzado a escribir sus propios programas en FORTH, sin duda en muchas ocasiones se habrá olvidado de incluir el @ al definir variables. Con CREATE y DOES> usted puede definir una clase de variable que no emplea el signo @.

Normalmente, sólo deja en la pila su valor, como una constante, pero surge un problema cuando usted desea cambiar su valor. Es mejor, entonces, el uso de una palabra, ->, como en:

```
nuevo valor -> nombre del nuevo estilo de
  variable
```

- > no hace nada, exceptuando el hecho de que deja una "nota" (estableciendo una variable) diciendo que se la ha utilizado, y ésta le dice a la variable de "nuevo estilo" que, en vez de apilar su valor actual, debe tomar un nuevo valor de la pila.

```
VARIABLE -> USADA
0 -> USADA !
: -> -1-> USADA!;
```

Ahora deseamos un recambio para la palabra definitoria VARIABLE.

Esta palabra definitoria (llamémosla VAR) constituye una nueva palabra de este tipo, utilizando CREATE...DOES>:

```
:VAR(--)
  CREATE
  0,(destinar 2 bytes, inicializados en 0)
DOES>
  -->USADA @ IF (nuevo valor, pfa--)
      !          (establecer nuevo valor)
    0 --> USADA !
  ELSE       (pfa--valor actual)
    @
  THEN
;
```

Veamos cómo definir una variable VAR x, establecerla en 99 y luego imprimir su valor:

```
VARx
99--> x
x.
```

La razón por la cual CREATE y DOES> son tan útiles es que permiten que usted combine dos propieda-

des que suelen ir separadas. La mayoría de los lenguajes de programación poseen datos, que son "pasivos", y requieren que se actúe sobre ellos, y rutinas, que son "activas" pero necesitan recibir algunos datos.

Con CREATE y DOES> usted puede establecer datos inteligentes que actuarán sobre sí mismos utilizando una parte DOES> de tiempo de ejecución. Esta idea es muy fructífera; pero para poder valerse de ella debe olvidarse de la distinción entre datos y rutinas en la que hacen tanto hincapié casi todos los otros lenguajes.

Palabras útiles

,(n– –) Incluye n como dos bytes en el diccionario. Hay una versión C, de un byte.

HERE(– – dirección) Apila la dirección del lugar del diccionario donde se llevará a efecto la siguiente inclusión.

FULL(dirección,n,relleno– –) Rellena n bytes, comenzando en la dirección dada, con el byte de relleno dado.

Matriz de matrices

Las matrices definidas por 1MATRIZ sólo son "levemente" inteligentes. Trabajan con bastante rapidez, pero no comprueban que el subíndice que se les da sea razonable, y existe el riesgo de que accidentalmente usted escriba sobre algo fuera de la propia matriz. Si 1MATRIZ almacenara la dimensión al comienzo del campo de parámetros, la parte en tiempo de ejecución podría comprobar que el subíndice proporcionado sea menor que la dimensión, y denunciarlo si no lo fuera. Algunos lenguajes, como el PASCAL y el BASIC, comprueban los subíndices de matrices como ésta, mientras que otros, como el C, dan prioridad a la velocidad y no efectúan comprobaciones. El FORTH obliga al usuario a calcular lo que realmente desea y también a hacerlo en la forma debida. Los lenguajes también difieren en el uso o no del subíndice 0 para el primer elemento o del subíndice 1, o en realidad de cualquier otro subíndice. Nuevamente, en FORTH usted puede elaborar el contenido de las matrices a la medida de su propio gusto. Las mâtrices multidimensionales son similares en principio, pero más complicadas. Normalmente uno se imagina a la matriz multidimensional como un bloque rectangular o tridimensional (o peor), pero el ordenador la almacena fila por fila en una larga lista. Usted convierte sus diversos subíndices en un decalaje en esta lista mediante un proceso consistente en multiplicar por las dimensiones.

Por ejemplo, para una matriz tridimensional de dimensiones 2, 3 y 4 (24 elementos en total), usted convierte los subíndices i, j y k en un:

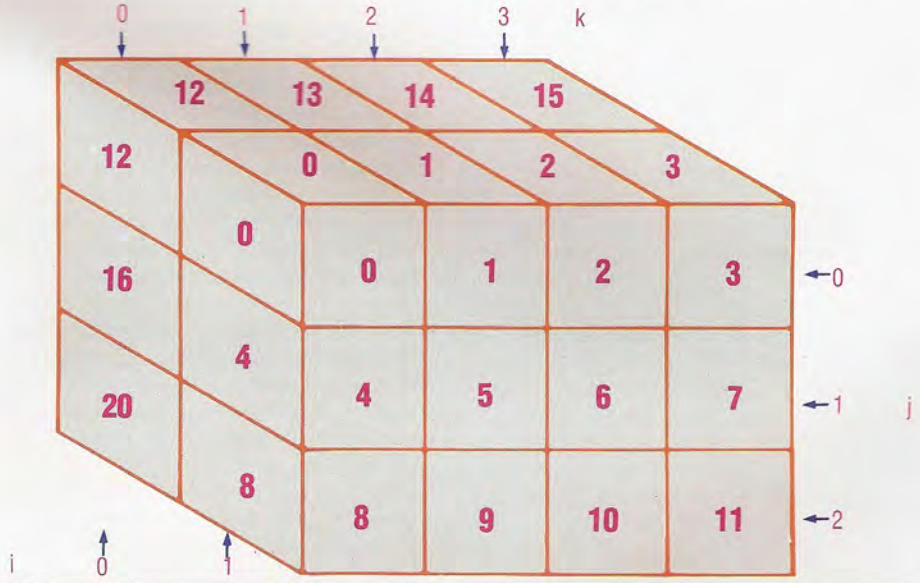
```
(i*3 + j)*4 + k
```

He aquí cómo definiría 3MATRIZ:

```
:3MATRIZ (d,e,f– –)
   (crea una matriz tridimensional con
   dimensiones d, e y f)
CREATE
   OVER,DUP,          (incluir e y f)
   HERE               (d,e,f,pfa + 4)
   2ROT*ROT*ROT*      (pfa + 4,2* d* e* f)
   DUP ALLOT          (incluir 2* d* e* f bytes)
   0 FILL             (inicializarlos en 0)
DOES>                 (i,j,k,pfa– –dirección)
   DUP@               (i,j,k,pfa,e)
   4ROLL*             (j,k,pfa,i*e)
   3ROLL+             (k,pfa,i*e + j)
   OVER2+@            (k,pfa,i*e + j,f)
   *                  (k,pfa,[i* e + j]* f)
   ROT + DUP +        (pfa,2*[[i* e + j]* f + k])
   SWAP 4 ++
;
```

(Observe que en el FORTH-79 el número de antes de ROLL debe ser uno mayor.)

Se pueden escribir palabras análogas (4MATRIZ, 5MATRIZ, etc.) a lo largo de líneas similares. Si esto le parece complicado, la mejor forma de comprenderlo es ver cómo lo hace el C. Incluso se puede escribir una palabra definitoria, NMATRIZ, que tome un número de la pila diciéndole cuántas dimensiones hay debajo



Escritura de un bloque de memoria

El siguiente procedimiento es el que se seguiría al escribir memoria en cinta o en disco:

1. Abrir archivo para salida

B=Longitud nombre archivo
HL=Dirección nombre archivo
DE=Dirección buffer de 2 K

CALL CAS OUT OPEN

2. Escribir los datos

HL=Dirección datos para salida
DE=Longitud de datos
BC=Dirección entrada para encabez.
A=Tipo archivo para encabezamiento

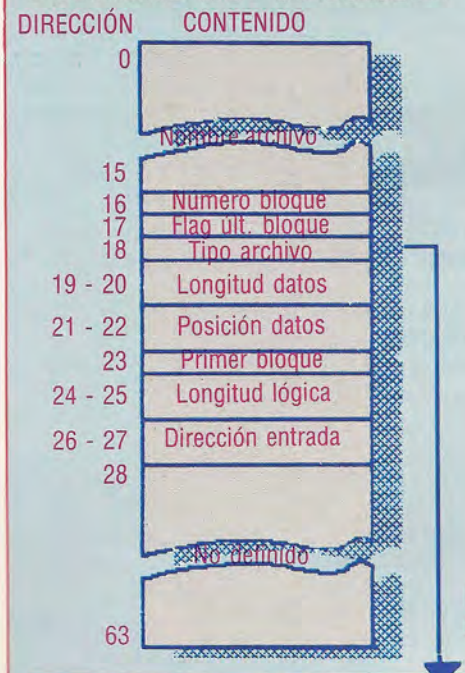
CALL CAS OUT DIRECT

3. Cerrar archivo salida
CALL CAS OUT CLOSE

Caroline Clayton

Formato del encabez.

La apertura de un archivo para su carga mediante CAS IN OPEN hace que el encabezamiento del archivo se escriba en la RAM y su dirección quede en HL. Además, DE contendrá la posición original de los datos (como indica el encabezamiento), BC contendrá la longitud lógica del archivo, y A el tipo de archivo. La información retenida en el byte del tipo de archivo es de bits significativos, como se muestra aquí



El tipo de archivo se define como varios campos diferentes:

Bit 0	Bits 1-3	Bits 4-7
Protección	Contenido archivo	Versión
0=Protegido 1=No protegido	0=BASIC 2=Binario 4=Pantalla 6=ASCII 8=Sin colocar	16=ASCII 32=Sin colocar

Caroline Clayton

A buen recaudo

Tanto la platina de cassette del Amstrad CPC 464 como la unidad de disco del CPC 664 aseguran un almacenamiento eficaz y fiable

El Amstrad CPC 664 ha dado un paso adelante al sustituir la platina de cassette del CPC 464 por una unidad de disco, con lo que ha incrementado aún más la rapidez de acceso. Naturalmente, el equipo físico no serviría para mucho si detrás no estuviera el adecuado software que lo soportara. La pregunta es: ¿qué se ha previsto en el firmware para que el programador en código máquina emplee estas facilidades de almacenamiento?

Un análisis de las características del firmware podría dividirse en tres áreas: las que son generales a la transferencia de archivo, las que son exclusivas del sistema de cinta y las que son exclusivas de la versión de disco.

Por lo general, cada archivo contiene dos secciones independientes de información: la primera es la sección de encabezamiento y la segunda es la de los datos. El encabezamiento contiene todas las informaciones que necesita el firmware para determinar cómo debe tratarse el archivo (por ejemplo, el tipo de archivo, dónde debe cargarse y su longitud). Los datos son simplemente el contenido del archivo. El firmware proporciona al usuario toda la información acerca del archivo y la facilidad encargada de manejarlo en memoria.

La sección del firmware que nos interesa es el gestor de cassette, aunque este término es algo impreciso, dado que también se emplean los mismos elementos para manipular los archivos de disco.

El gestor de cassette emplea dos "corrientes", una para entrada y otra para salida, que pueden estar activas a la vez. Esta configuración no resulta de gran utilidad en un grabador de cinta, pero es una facilidad muy potente cuando se emplea correctamente con una unidad de disco. Las corrientes se han de inicializar antes de poder accederse a ellas. La inicialización consiste en asignar un nombre y un área de trabajo (*buffer*) a la corriente que interesa. A un mismo tiempo sólo puede estar activo un archivo de entrada y uno de salida, de modo que si la corriente de salida no se emplea, puede no ser definida como una segunda corriente de entrada. Las entradas interesadas en la transferencia de archivos se encuentran listadas en la tabla de rutinas de almacenamiento de datos.

Los datos pueden ser transferidos a o desde el archivo en uno de estos dos modos: *transferencia de caracteres* o *transferencia de bloques*. En el modo transferencia de caracteres, el firmware utiliza el *buffer* asignado a la corriente para almacenar los

datos temporalmente. Esto significa que se puede efectuar el acceso tanto hacia como desde una cinta (o un disco) en grandes bloques mientras el programa trata los datos de byte en byte. Naturalmente, un programa que tenga que acceder al disco cada vez que se necesita un carácter sería muy lento. Los dos modos de transferencia se excluyen: una vez efectuado el acceso a un archivo en un modo es imposible cambiar y acceder al mismo en el otro.

En el modo de transferencia por bloques (CAS IN DIRECT y CAS OUT DIRECT) los *buffers* asignados a las corrientes no se emplean: en su lugar, los datos se leen directamente desde o hacia la memoria. La memoria puede estar en la RAM o en la ROM.

Las rutinas CAS IN CLOSE y CAS OUT CLOSE son especialmente importantes porque informan al firmware que la corriente está cerrada y, por ende, disponible para un nuevo archivo. Si CAS OUT CLOSE no se efectúa después de una salida de carácter, el *buffer* de salida puede quedar incompleto y, por consiguiente, no escrito del todo. CAS OUT CLOSE evita este problema obligando al firmware a enviar cualquier bloque incompleto de datos.

Nuestro esquema muestra el proceso por el cual un bloque de memoria se escribe en un archivo. Ante todo, el archivo es abierto por medio de CAS OUT OPEN, que se limita a asignar un nombre al fichero y reserva la corriente de salida para el empleo del archivo. Los datos son escritos desde la memoria empleando CAS OUT DIRECT, y finalmente el archivo se completa llamando a CAS OUT CLOSE que, en este caso, sólo indica que la corriente de salida está disponible para su empleo.

El gestor de cassette emplea un área de datos, conocida como el *encabezamiento*, para almacenar información correspondiente a cada archivo individual. En el caso de una operación de lectura, esta información es tomada desde el comienzo del archivo y, en el caso de una escritura, el encabezamiento se establece en memoria y después se escribe al comienzo del archivo. El planteamiento general del encabezamiento se muestra en el diagrama *Formato del encabezamiento*.

A nivel de firmware, el tipo de archivo no tiene efecto alguno: el archivo puede ser leído en ambos sentidos independientemente del tipo. El tipo resulta importante cuando el firmware es empleado por un programa de alto nivel, p. ej., en BASIC.

La *posición de datos* se define como la posición desde la cual se escribió originalmente el archivo. Se observará que esto sólo es importante cuando el archivo es escrito directamente desde la memoria. Cuando se emplea una salida de carácter, la posición representa la dirección del *buffer* de salida. La

Rutinas de almac. de datos

Sistema cassette:

$BC65 CAS INITIALISE
Esta rutina asegura el establecimiento de las condiciones por omisión: es decir, las corrientes de entrada/salida están cerradas y los mensajes habilitados

$BC6B CAS NOISY
Habilita y deshabilita los mensajes proporcionados desde el gestor de cassette

$BC77 CAS IN OPEN
Inicializa un archivo para leer desde él

$BC80 CAS IN CHAR
Lee un único carácter del archivo entrada

$BC83 CAS IN DIRECT
Carga en memoria un archivo entero

$BC89 CAS TEST EOF
Comprueba el final de archivo entrada

$BC7A CAS IN CLOSE
Cierra el archivo entrada actual

$BC7D CAS IN ABANDON
Cierra el archivo entrada, aún sin llenar del todo

$BC8C CAS OUT OPEN
Inicializa un archivo para escribir en él

$BC95 CAS OUT CHAR
Envía un solo carácter al archivo salida

$BC98 CAS OUT DIRECT
Escribe un bloque de memoria en el archivo salida

$BC8F CAS OUT CLOSE
Cierra un archivo ya completo

$BC92 CAS OUT ABANDON
Cierra el archivo descartando la información no escrita

$BC9B CAS CATALOG
Genera un catálogo de archivos

Sistema de disco:

$BE80 SET MESSAGE
Habilita y deshabilita mensajes dados por el disco

$BE83 SETUP DISK
Inicializa los tiempos de la unidad (velocidad de pasos, retardo carga/descarga y tiempo agotado *on/off*)

$BE86 SELECT FORMAT
Establece el formato del disco

$BE89 READ SECTOR
Lee un sector del disco

$BE8C WRITE SECTOR
Escribe un sector del disco

$BE8F FORMAT TRACK
Formatea una pista

$BE92 MOVE TRACK
Mueve el cabezal de la unidad hasta una pista determinada

$BE95 GET DR STATUS
Proporciona un byte que indica el estado de la unidad actual

$BE98 SET RETRY COUNT
Establece el número de retiradas en caso de error

Las operaciones de cassette o disco se efectúan por medio de llamadas del firmware. Algunos puntos que hay que anotar sobre el sistema de archivado del Amstrad son la facilidad para mantener abiertos tanto los archivos de entrada como los de salida simultáneamente y la casi completa compatibilidad de la síntaxis de manejo de los archivos en cassette y en disco. Los archivos pueden ser cargados o grabados (*read out, read in*) directamente de la RAM (empleando CAS IN/OUT DIRECT) o carácter a carácter a través de un *buffer* de 2 Kbytes (CAS IN/OUT CHAR). No obstante, una vez determinado, el modo de entrada/salida no puede cambiarse

longitud lógica es el número total de bytes de datos en el archivo. Obsérvese que ésta no incluye encabezamiento de información, o cualquier "almohadilla" que se haya requerido para llenar el *buffer*.

La *dirección de entrada* es la dirección en la que se inician los programas en código máquina cuando ocurre un *auto-run* desde el BASIC. Los otros campos son específicos sólo para tratamiento de archivos en cassette y, por ende, su uso es muy limitado.

La detección de errores es efectuada automáticamente por el firmware por medio de un código CRC, aunque son raros los errores que se encuentran merced a la meditada programación del sistema. Los errores que aparezcan son señalados por el firmware para que el usuario decida qué hacer.

A fin de facilitar el tratamiento de errores de lectura, los archivos en cassette no se escriben en su integridad, sino divididos en bloques de dos Kbytes. Cada bloque es tratado como una unidad distinta, y por ello se puede volver a cargar un bloque "malo" sin tener que recargar todos los bloques restantes que le precedieron. Este planteamiento por bloques retarda necesariamente la velocidad de transferencia del archivo, pero la eficacia que se consigue compensa la pérdida de tiempo. La velocidad de escritura a la cassette puede variar bajo control del software por medio de la entrada CAS SET SPEED en $BC68, mientras que la velocidad de lectura es seleccionada automáticamente por el firmware.

Una de las configuraciones más útiles del sistema operativo del Amstrad, en lo que respecta al programador, es la instalación transparente de instrucciones de disco. Es decir, todas las rutinas generales de archivo tienen exactamente los mismos parámetros y, por tanto, un programa escrito para operaciones en cassette puede trabajar automáticamente con la unidad de disco sin modificación alguna. Esto se ha logrado gracias a la técnica del *parcheo*, que ya describimos anteriormente. Sin embargo, esta técnica no es inmediata, por lo que unas cuantas sugerencias harán que su empleo no resulte tan problemático.

Las áreas particulares que hay que vigilar son las que se refieren al empleo de memoria. El sistema de disco es controlado desde una ROM de ampliación, aunque también necesita una cierta cantidad de RAM de espacio de trabajo. Ésta se toma generalmente de la parte superior de la memoria (pero puede reasignarse a otras áreas de la RAM). Por tanto, cualquier programa que quiera usar el disco debe tener en cuenta que esta área de RAM ($504H bytes por debajo de HIMEM) no está disponible.

Otro aspecto, derivado de la implementación de la compatibilidad del nombre de archivo de CP/M, es que los nombres de archivo en disco no pueden contener más de 12 caracteres. Por esto es mejor suponer una longitud de ocho caracteres para evitar así problemas de traslación de cinta a disco.

Los encabezamientos de disco se comportan de una manera algo diferente de los encabezamientos de cinta; las referencias a los bloques aquí ya no tienen importancia y son ignoradas. El encabezamiento es también sometido a una suma de chequeo, para que el firmware pueda distinguir entre archivos CP/M y AMSDOS.

La adición del sistema de disco conlleva entradas adicionales del bloque de saltos, que están a disposición del usuario para el manejo de discos a nivel de sistema operativo. Esto es lo que se detalla en la segunda parte de nuestra tabla.

Por último, damos aquí un programa de ejemplo que ilustra cómo puede extraerse la información de un encabezamiento de archivo. El archivo primero se abre por medio de una llamada a CAS IN OPEN, que genera el encabezamiento en la RAM. Una vez extraída la información pertinente, el archivo se descarta mediante CAS IN ABANDON antes de que tenga lugar cualquier lectura. El programa en BASIC asociado sirve para proporcionar una interface de usuario a la rutina.

```
 10 'Lector encabezamiento archivo
 20 'Emplea CAS IN OPEN en BC77H
 30 routine=&8800: address=&8800: ON ERROR GOTO
    530
 40 MODE 1: MEMORY &5FFF
 50 WHILE -1
 60 CLS
 70 LOCATE 10,10: PRINT "¿Archivo cassette o disco?"
    ;CHR$(18)
 80 a$=""
 90 WHILE a$<>"T" AND a$<>"D"
100 a$=UPPER$(INKEY$)
110 WEND
120 IF a$="T" THEN ITAPE ELSE IF a$="D" THEN IDISC
130 LOCATE 10,12: INPUT "Nombre del archivo"
140 IF LEN(file$)=0 AND a$="D" THEN 130
150 GOSUB 420
160 CALL routine
170 buffer=PEEK(address)+256*PEEK(address+1)
180 LOCATE 20,12: ff=0
190 FOR name=0 TO 7-8*(t$="T")
200 char=PEEK(buffer+name)
210 IF (char<32 OR char>126) AND char>0 THEN ff=1:
    GO TO 230
220 PRINT CHR$(char);
230 NEXT
240 IF ff=1 THEN PRINT CHR$(8); "No hay nombre
    archivo";
250 PRINT
260 'test filetype
270 LOCATE 5,16: PRINT "Tipo archivo: ": LOCATE 20,16
280 type=PEEK(buffer+18)
290 IF (type AND 1)=1 THEN PRINT "Protegido";
300 file = type AND &FE
310 IF file=0 THEN PRINT "Basic"
320 IF file=2 THEN PRINT "Binario"
330 IF file=6 THEN PRINT "ASCII"
340 IF file=6 THEN PRINT "Desconocido"
350 length=PEEK(buffer+24)+256*PEEK(buffer+25)
360 LOCATE 5,18: PRINT "Longitud datos:" :LOCATE
    20,18: PRINT HEX$(length,4)
370 entry=PEEK(buffer+26)+256*PEEK(buffer+27)
380 LOCATE 5,20: PRINT "Direccion entrada:" :LOCATE
    20,20: PRINT HEX$(entry,4)
390 LOCATE 9,25:PRINT "Pulsar una tecla para continuar"
400 WHILE INKEY$=" ":WEND
410 WEND
420 RESTORE 520: READ byte: off=0
430 WHILE byte<>999
440 POKE routine+off, byte
450 off=off+1: READ byte
460 WEND
470 POKE routine+1,LEN(file$)
480 FOR char=1 TO LEN(file$)
490 POKE routine+&100+char-1,ASC(MID$(file$,char,1)).
500 NEXT
510 RETURN
520 DATA &06,&00,&21,&00,&89,&11,&00,&60,
    &cd,&77,&bc,&7d,&32,&0,&80,&7c,&32,&01,&80,
    &cd,&7d,&bc,&c9,999
530 'trampa de errores
540 IF ERL=80 THEN a$="":RESTORE
550 STOP
```

IMPACTO DE UNA NUEVA TECNOLOGÍA

Hoy, muchos periódicos continúan componiendo tipográficamente sus páginas mediante una máquina denominada linotipia. Inventada en 1886, permite obtener líneas completas de matrices en metal fundido a partir de un texto entrado por teclado. La prensa de provincias que se encuentra en dificultades financieras está implantando la composición tipográfica informatizada a fin de reducir los costos. Gracias a los nuevos dispositivos es posible cargar el texto electrónicamente, obteniéndose placas de impresión de plástico llamadas «estereotipos»

Iniciamos una breve serie dedicada a las aplicaciones de la tecnología electrónica en la industria editorial. Analizando los métodos actuales de producción y comprendiendo los aspectos implicados, podremos hacernos una idea más cabal del revolucionario e irreversible cambio que está ejerciendo la informática en la edición de publicaciones periódicas.

Ya casi constituye un lugar común afirmar que la tecnología informática está modificando de manera radical la forma de vivir y de trabajar de la sociedad contemporánea. En efecto, la introducción de nuevos procedimientos basados en ordenador está ocasionando la desaparición de muchas prácticas y técnicas establecidas. La industria editorial ilustra claramente este cambio en las pautas de trabajo, dado que en este caso la tecnología electrónica se está introduciendo en una actividad que, en muchos casos, no ha actualizado sus métodos y maquinaria desde los años treinta. Además, la mano de obra de la industria gráfica está organizada en gremios tradicionalmente muy poderosos. Por tanto, es probable que la introducción de los ordenadores sea una experiencia traumática para quienes trabajan en esta actividad. En esta breve serie nos referiremos a los cambios que se están produciendo en la industria editorial y hacia dónde conducirán probablemente las actuales tendencias. En primer lugar, analicemos la nueva tecnología y veamos cómo se aplica en cada etapa del proceso editorial.

Comencemos por el ámbito más clásico, el de la palabra escrita. En la actualidad la mayoría de los escritores han reemplazado las máquinas de escribir mecánicas (incluso eléctricas) por microordenadores, que les han permitido escribir y corregir sus propios textos, incrementar su velocidad de trabajo y producir copias cuidadas y económicas para las editoriales. Es posible que esté familiarizado con programas de tratamiento de textos en disco y cinta o en ROMs especiales que se pueden instalar en su micro, y que le proporcionan capacidades para tratamiento de textos. De hecho, la aplicación más utilizada en micros personales, después de los juegos, es el tratamiento de textos.

Tanto como si desea enviarles un boletín interno a los socios de un club local como si quiere escribir su primera novela, es probable que el modesto micro personal sea una poderosa herramienta para su objetivo. Las copias se pue-

den comprobar y corregir en pantalla, estableciendo la anchura de los márgenes, numeración automática de páginas, títulos de página y pie antes de la impresión, para producir un único borrador ya acabado del texto. Para facilitar aún más las cosas, muchos paquetes de tratamiento de textos poseen verificadores de ortografía que van explorando el texto y señalando las palabras no reconocidas con el fin de que se las corrija. Si el texto ha de ser el cuerpo de un artículo en una revista o diario, la producción de un original pulcramente mecanografiado es sólo la primera etapa del largo proceso de producir una versión final impresa.

En la mayoría de los actuales métodos de producción de revistas aún prevalecen los sistemas tradicionales. Por lo general, la publicación espera que el autor del trabajo entregue el artículo mecanografiado (normalmente con máquina de escribir corriente o con un sistema de tratamiento de textos/impresora). El personal de la redacción de la publicación se encarga luego de revisar el artículo. El redactor lee y corrige el *contenido* del trabajo, normalmente escribiendo en él a mano o, cuando se requieren modificaciones sustanciales, produciendo hojas adicionales mecanografiadas. El texto pasa luego a un corrector, quien enmienda los errores ortográficos y gramaticales que se hubieran deslizado (nuevamente, por lo general, escribiendo a mano), reescribe, si fuera preciso, algunos párrafos para adecuar el texto al estilo de la publicación y prepara tipográficamente el original. El proceso de *preparación tipográfica* tiene como finalidad esencial indicar al tipógrafo el tipo de letra, el ancho de los textos y la justificación a utilizar y, como antes, se escribe a mano sobre el original. En las publicaciones pequeñas una misma persona suele desempeñar las funciones de redactor y corrector.

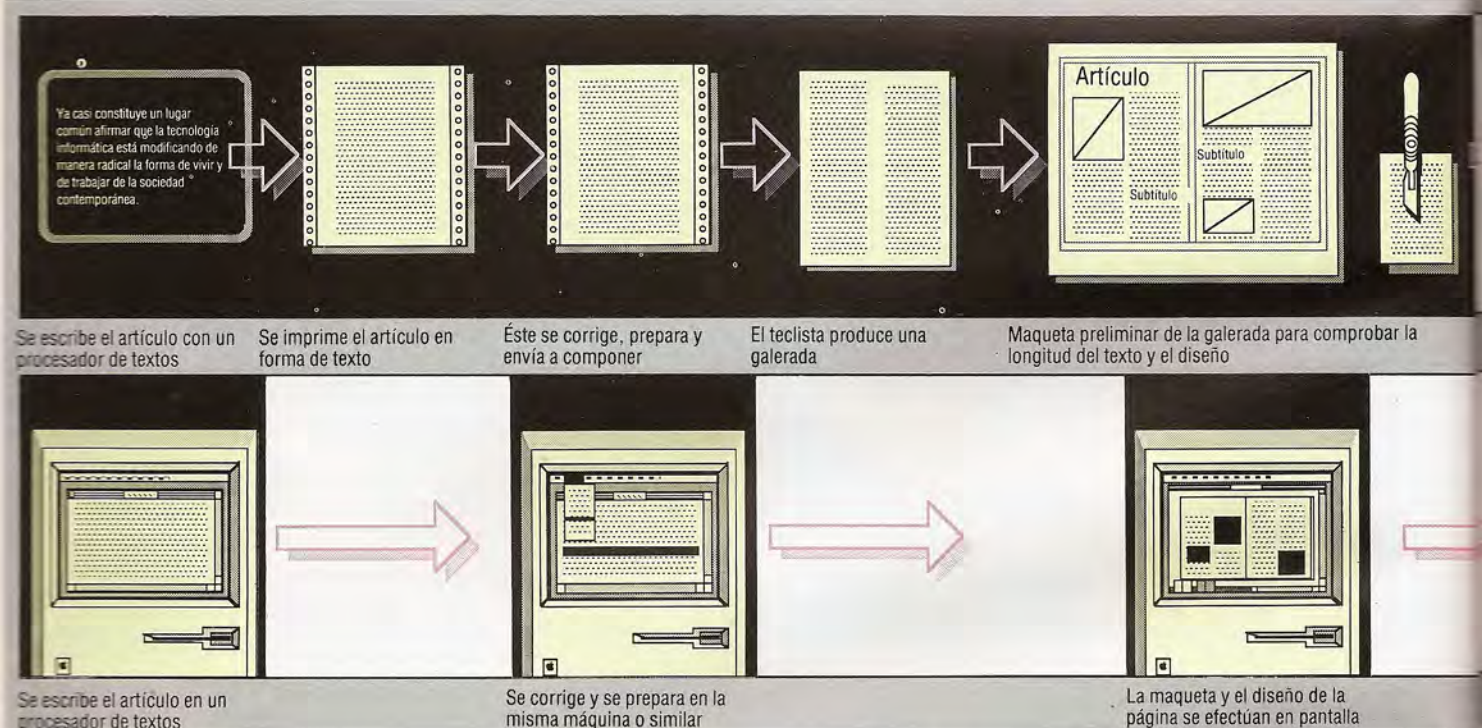
Una vez compuesto el artículo, se espera que ya casi haya adquirido su forma impresa final, aunque es probable que los teclistas o linotipistas hayan incurrido en errores al transcribir el texto original. En esta etapa, el artículo compuesto se denomina *galerada* o *prueba,* y el corrector la debe releer atentamente, marcar las erratas y devolverla al teclista, quien realizará las correcciones señaladas.

En la etapa de la primera prueba, otra persona se une al proceso de producción. El trabajo del maquetista consiste en tomar la prueba impresa del texto y diseñar el trazado de una página: el aspecto que tendrá la forma impresa final. Por supuesto, la mayoría de las publicaciones no contienen sólo texto, sino también fotografías y diagramas, junto con las correspondientes notas al pie. El maquetista ha de combinar equilibradamente las partes que constituirán el contenido final de la página a fin de dar a ésta una forma racional y atractiva. Con este fin se suele confeccionar una *maqueta* de la página y disponer en ella los diversos componentes como si se tratara de un rompecabezas de piezas irregulares. En esta etapa también se puede distribuir el texto de la prueba en la maqueta de la página, aunque algunos maquetistas se limitan a medir la longitud del texto con el fin de determinar el espacio que ocupará en la página terminada.

Durante este proceso sucede con frecuencia que el texto es demasiado largo o demasiado corto como para caber en el espacio que se le ha asignado. En este caso el redactor o el corrector tendrá que suprimir algunos párrafos o bien, si el texto fuera muy breve, añadir líneas para que alcance la longitud necesaria. Las supresiones o añadidos se incluyen en la prueba, de modo que, cuando el teclista la devuelva corregida, el artículo tenga la extensión requerida. Por último, se insertan en la página los diagramas, las fotografías y el texto de acuerdo a la maqueta confeccionada previamente, y entonces ya se puede mandar a imprimir la página.

A la luz de esta breve descripción del proceso editorial podemos apreciar que supone muchas etapas que significan una potencial pérdida de tiempo. En realidad el método que acabamos de reseñar constituye un híbrido en el cual, en ciertas fases, se emplean de forma aislada ordenadores independientes. Es probable que el artículo se escriba en un procesador de textos y que el mismo proceso tipográfico esté también informatizado gracias a una máquina no muy distinta a un procesador de textos normal, pero que incluye algunas instrucciones que posibilitan la selección de dife-

Un cambio revolucionario



rentes tipos, etc. A la vista de un sistema de esta clase, no haría falta un analista de sistemas profesional para sugerir posibles mejoras.

En primer lugar, imaginemos que en la oficina de la editora de la publicación hubiera un sistema de tratamiento de textos compatible con el que utiliza el autor. No sería necesario que éste produjera páginas impresas, sino tan sólo que entregara un disco o cinta con el artículo grabado magnéticamente, o incluso que transmitiera el artículo electrónicamente utilizando la red telefónica y un par de modems. La labor que cumplen el redactor y el corrector se podría realizar en pantalla. Aun cuando el proceso revirtiera, entonces, en esta etapa al método más tradicional, al imprimir la versión revisada del artículo en papel para enviársela a los teclistas, el sistema ya se habría modernizado un poco.

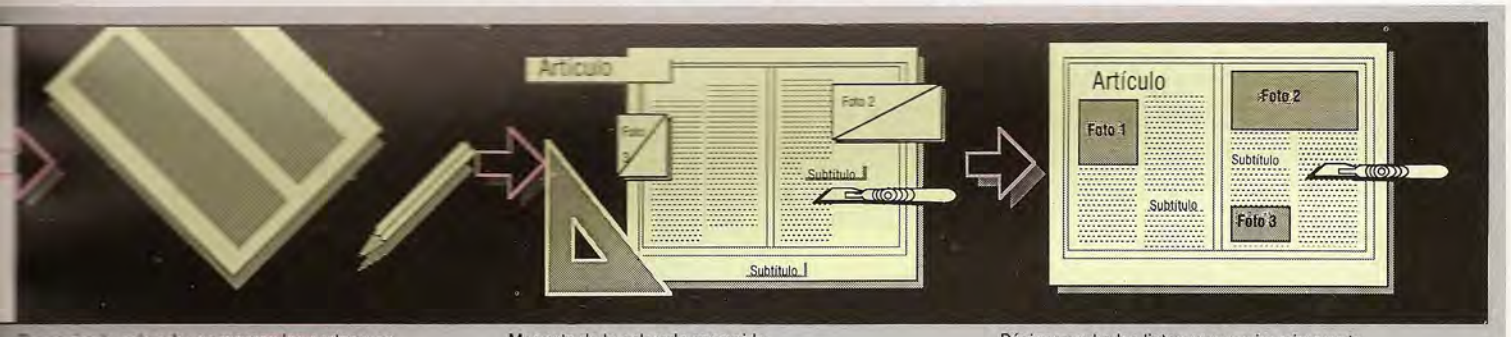
Avanzando otro paso, un sistema racionalizado aseguraría que la máquina tipográfica informatizada fuera capaz de leer el disco en el cual se almacenó la versión original revisada. De ser éste el caso, el artículo pasaría por la traducción del borrador original del autor a su forma tipográfica existiendo sólo como una serie de impresiones magnéticas en un disco hasta que se compusiera tipográficamente.

Se dice que este sistema es de «entrada directa»: el texto del artículo es digitado una sola vez, por su autor, y ulteriormente introducido en otras máquinas mediante carga desde disco o bien por transmisión electrónica entre secciones de un sistema informatizado homogéneo y de mayores dimensiones.

Sin embargo, las posibilidades de la nueva tecnología van mucho más allá de la manipulación de textos. Recientes desarrollos en entornos operativos de ordenador, en particular los sistemas WIMP (Windows, Icons, Mouse Program) han despejado el camino para que la informática intervenga en el trabajo del maquetista. Estos nuevos sistemas permiten que éste monte una página en pantalla, con frecuencia incluyendo fotografías digitalizadas y diagramas. Los diferentes tipos de letra y la disposición del texto se pueden manipular utilizando simplemente un ratón controlador. Luego la versión final de la página se puede imprimir directamente mediante una impresora de alta calidad.

A pesar de que en muchos sentidos la producción de periódicos es diferente de la producción de revistas, para ella rigen muchos de los mismos principios. De hecho, las necesidades de procesamiento rápido de la prensa diaria, sumado a su enorme volumen de ventas, significa que es probable que la nueva tecnología cause un mayor impacto en este campo de la industria editorial. La tecnología que se aplica actualmente en la prensa data, aproximadamente, de los años treinta y emplea procedimientos desarrollados a fines del s. XIX. No obstante, en la industria de la prensa se detectan indicios de cambios inminentes.

Muchos periódicos de provincias, en respuesta al desafío de los folletos publicitarios gratuitos de producción económica, ya han aplicado con todo éxito métodos de producción informatizada. Existen planes para que las empresas pequeñas y más flexibles desafíen al monopolio de los gigantes de la prensa produciendo un diario con alrededor de la quinta parte de personal que se requeriría normalmente en un periódico nacional, usando métodos de entrada directa, montaje en pantalla e impresión informatizada.



Se corrige la galerada, se marcan los cortes y se devuelve al teclista

Maqueta de la galerada corregida

Páginas acabadas listas para enviar a imprenta

Una impresora de alta calidad produce las maquetas de las páginas

Páginas completas ya listas para enviar a imprenta

Un interesante efecto de la producción electrónica de un diario se puede observar en su distribución a través del país. Tradicionalmente, los periódicos nacionales se imprimen en un centro (o posiblemente dos) y luego los ejemplares impresos se distribuyen a través del país por tren y carretera. En el caso de un periódico producido por medios electrónicos, el método de distribución puede cambiar radicalmente. Aunque se puede producir el diario en un sitio, se puede imprimir en cinco o seis prensas de provincias transmitiendo el diario electrónicamente a través de enlaces telefónicos normales, reduciendo de modo significativo los costos de distribución. Además, resulta muy sencillo producir ediciones locales del diario nacional, añadiéndole páginas al contenido principal del periódico en los centros locales.

En la segunda parte de esta serie acerca de la informática en la industria editorial veremos cómo se están abriendo paso las nuevas técnicas de edición electrónica en el mercado de microordenadores, y mostraremos cómo se puede producir una revista, desde la creación del texto hasta el producto final, en un solo micro.

Página en blanco

Comenzamos un proyecto para diseñar un programa de hoja electrónica en BASIC

La mejor forma de pensar en un programa de una hoja de datos es como si se tratara de una hoja de papel electrónica, con una calculadora programable incorporada. El programa permite entrar números y otros datos en esta hoja en blanco de papel, y luego llevar a cabo cálculos específicos a partir de esta información. Para ello usted debe preparar una serie de fórmulas aritméticas o matemáticas en la hoja electrónica. En base a sus instrucciones, el programa de hoja electrónica utiliza estas fórmulas para efectuar cálculos con los datos presentes en la hoja. Luego coloca los datos nuevos (es decir, los resultados de los cálculos) en la hoja, en las posiciones que se especifican en las fórmulas.

De esto modo, si usted tiene un problema o una aplicación que le exijan trabajar con grandes cantidades de datos, la hoja electrónica es una herramienta ideal. Con ella puede añadir filas y columnas con sólo pulsar un botón, una configuración útil para totales de ventas, aplicación de fórmulas matemáticas a una serie de datos, cálculo de porcentajes para el IVA e impuesto sobre la renta, etc. En resumidas cuentas, con una hoja electrónica es posible realizar cualquier cosa que pueda hacer con lápiz, papel y una calculadora, sólo que lo puede llevar a cabo mucho más rápidamente y con mayor exactitud. Otra configuración útil de las buenas hojas electrónicas es que una vez que ha preparado una para que realice un conjunto dado de funciones, usted puede volver a utilizar esa hoja una y otra vez con distintos datos cada vez. Ésta es una facilidad sumamente útil si ha de efectuar con regularidad un conjunto de cálculos, como gastos de viaje semanales o comisiones sobre las ventas.

Por ejemplo, si usted lleva una pequeña empresa, puede reclamar el IVA que pagó sobre las compras en relación al IVA que tenga que indicar en su declaración de impuestos. Obviamente, antes de poder hacer esto, tendrá que calcular el IVA total que ha cargado en los artículos que ha vendido y restar el IVA total que ya ha pagado sobre los artículos que ha adquirido. Utilizando el método normal de lápiz y papel, tendría que calcular el IVA sobre cada artículo individual de forma manual y luego sumar todo el IVA al final, procedimiento que es tan tedioso como susceptible de error. Sin embargo, con una hoja electrónica, todo cuanto ha de hacer es prepararla para que calcule el IVA entrando la fórmula; supongamos, para multiplicar todos los artículos de una columna por 15 y dividir por 100. Luego se puede entrar el costo de las compras, y la hoja electrónica se encargará del resto.

Las hojas electrónicas se pueden utilizar para numerosos tipos de aplicaciones. Entre las domésticas se incluyen el presupuesto hogareño, los gastos del coche, análisis de préstamos y gastos generales. Entre los usos empresariales de las hojas electrónicas se incluyen facturación, libro mayor de ventas y compras, presupuestos, previsión y planificación,

Problema de impuestos

		A2*15/115			A5*15/100
		B2*15/115			B5*15/100
		C2*15/115			C5*15/100
		D2*15/115			D5*15/100

	1.COMPRADO	2.C. BRUTO	3.IVA	4.VENDIDO	5.PRECIO	6.IVA
A	MICRO	399.99	52.17	HOJA/ART. 1	120.00	18.00
B	IMPRESORA	249.99	32.60	PROG. CONV.	2000.00	300.00
C	PAPEL	20.00	2.60	HOJA/ART. 2	120.00	18.00
D	CINTA	7.24	0.94	DATOS CHISME	50.00	7.50
E	LÁPICES...	2.42	0.31			
F	ESCRITORIO	92.47	12.06			
G	UNID. DISCO	199.95	26.08			
H	LÁMPARA	24.87	3.24			
I	ELECTR.	57.32	7.47			
J	SANGRE	10.57	1.37			
K	SUDOR	1.00	0.13			
L	LÁGRIMAS	0.50	0.06			
M		IVA PAGADO	139.03		IVA CARGADO	343.50
N					IVA NETO	204.47

Hoja de balance
El trazado de hoja electrónica que vemos aquí a modo de ejemplo muestra un típico problema de contabilidad: las cuentas de balance del IVA. El usuario ha de preparar las columnas 1, 2, 4 y 5 de la hoja electrónica, y ésta se debe programar para calcular las cifras de las columnas 3 y 6 y los totales del IVA. Por ejemplo, las cifras de la columna 3 se pueden hallar multiplicando la cifra correspondiente de la columna 2 por 15/115, y programarlas preparando las fórmulas en la celda de la columna 3 como se indica. Suele darse el caso, como aquí, de que se requieran fórmulas similares para una fila o columna completas de la hoja electrónica. Para que el usuario no tenga que prepararlas todas de forma manual, la mayoría de las hojas electrónicas incluyen una función para duplicar que permite repetir la fórmula de una celda a lo largo de una fila o una columna

Caroline Clayton

análisis de costos, pérdidas y beneficios, costo y estimación de empleo, cálculos de comisiones sobre las ventas y modelado de proyectos, por nombrar sólo algunos. Los beneficios para el usuario son muchos e incluyen aumento de la productividad, exactitud y mejora de la presentación.

El programa de hoja electrónica que diseñaremos en esta serie le ofrecerá al programador de BASIC una esclarecedora visión de la forma como se escriben los programas comerciales de hoja electrónica y cómo funcionan. A medida que se vayan imprimiendo cada una de las partes del programa, se ofrecerá una explicación detallada acerca de cómo trabaja cada sección y lo que hace. El programa se ha escrito para Spectrum, Commodore 64, BBC Micro y Amstrad CPC 464/664; para cada sección del programa se imprimirá un listado en BASIC Spectrum y tipo Microsoft, junto con detalles para implementar el programa en otras máquinas. Se pretende que el programa sea ilustrativo; se lo podría utilizar como base para un programa mucho más sofisticado, a tenor del nivel de destreza que posea usted como programador y la cantidad de tiempo que desee dedicarle a la tarea.

Debido a la naturaleza compleja de las hojas electrónicas, nos hemos concentrado en reproducir las principales características de un programa de hoja electrónica, con el resultado de que en el programa la comprobación de errores existente es ínfima. Por consiguiente, habrá de tener gran cuidado al entrar sus datos y usar el programa, puesto que cualquier error hará que éste quede colgado. La ausencia de comprobación de errores obedece a dos motivos: el primero es la limitación de espacio y, el segundo, la velocidad de operación.

Puesto que el programa está escrito en BASIC, el tamaño de la hoja queda limitado a 15 columnas por 15 filas. Ello es para asegurar una velocidad de operación razonable y reducir la complejidad del programa. Observe que en todo momento sólo se puede visualizar una pequeña porción, o «ventana» de la hoja, debido a las limitaciones de las visualizaciones en pantalla. En las Commodore 64, BBC Micro y Amstrad, esta ventana es de cinco columnas por siete filas; en el Spectrum es de cuatro por seis.

Estructura celular

Cada uno de los 225 cuadraditos de nuestra hoja electrónica se denomina *celda*. Estas celdas están etiquetadas (en la primera fila) A1, A2, etc., hasta A15; B1, B2, etc. (en la segunda fila), hasta la decimoquinta fila (O1, O2, etc. hasta O15). Para desplazar el cursor por la hoja, usted simplemente usa las teclas del cursor de su máquina. La celda actual que esté señalando el cursor se visualiza en la esquina superior izquierda de la pantalla y se ilumina la propia celda en cuestión.

En cada celda usted puede bien entrar datos numéricos o bien establecer una nueva fórmula para manejar los datos. Éstos se visualizan en la pantalla en la celda adecuada y usted sólo cuenta con cinco dígitos por celda. Si en la celda actual hay una fórmula, la misma se visualiza en la línea de entrada de la parte inferior de la pantalla. Para entrar una fórmula nueva, simplemente seleccione la opción entrar fórmula y digite la nueva fórmula. El evaluador de fórmulas de este programa es bastante sofisticado, evaluando la fórmula de izquierda a derecha y con precedencia aritmética normal. Los operadores que puede manipular son +, −, *, /, ^ y paréntesis. Los operandos permitidos son nombres de celdas o números reales o enteros. Por lo tanto, si usted quisiera sumar los datos de las celdas A1, A2 y A3, y colocar luego la respuesta en la celda A4, entraría la fórmula A1 + A2 + A3 en la celda A4. Nuevamente, si quisiera calcular el IVA sobre el valor, supongamos, de B1, entonces colocaría en la celda B2 la fórmula B1* 0,15 y en la celda B3 la fórmula B1 + B2.

Funciones

Nuestro programa de hoja electrónica en BASIC incluye muchas características que la convierten en una poderosa herramienta para utilizar en el hogar. Éstas incluyen:

- Una pantalla de AYUDA, que imprime una lista de las opciones de que dispone el usuario.
- Una función ALMACENAMIENTO TEMPORAL que le permite guardar la hoja visualizada actualmente. Ésta es especialmente útil cuando quiere utilizar la hoja electrónica para probar valores diferentes.
- También puede RESTITUIR una hoja guardada temporalmente a la pantalla (es decir, después de haber probado otros valores, puede retomar otra vez sus datos originales).
- La función CALCULAR trabaja a través de la hoja de izquierda a derecha y de arriba abajo, calculando todas las fórmulas y alterando en consecuencia los contenidos de las celdas a medida que va avanzando. Por lo tanto, usted debe asegurarse de haber dispuesto los datos por un orden que no afecte a estos cálculos.
- LIMPIAR borra todos los datos de la hoja, de modo que ha de asegurarse, antes de utilizarla, de haber almacenado sus datos o bien haberlos guardado en cinta o disco.
- La función DUPLICAR le permite reproducir una fórmula en una gama de celdas, ahorrando en consecuencia mucho tiempo y mucho trabajo de digitación en caso de que usted tenga muchas fórmulas similares (como cálculos del IVA).
- Puede CARGAR y GUARDAR los datos en su hoja desde o en cinta.
- Asimismo, puede CARGAR y GUARDAR las fórmulas independientemente de los datos de la hoja. Esto es útil, porque le permite utilizar una hoja que se ha preparado para uso frecuente (como para calcular los gastos de viaje semanales) con datos nuevos y sin necesidad de preparar manualmente la hoja cada vez.
- La función TAB le permite llevar el cursor a cualquier posición de la hoja, sin tener que recorrer con él toda la pantalla.

Ofrecemos aquí los listados para preparar la pantalla y manipular los gráficos en el Commodore 64. En el próximo capítulo ofreceremos los listados correspondientes para BBC Micro, Spectrum y Amstrad. Ésta es la única sección del programa de hoja electrónica para la que ofreceremos listado separados. Esto se debe a que la manipulación de pantalla para cada máquina es totalmente diferente, mientras que el resto del programa es similar para cada ordenador (con la excepción de la versión para el Spectrum, que emplea distintos métodos de manipulación de series).

Rutinas gráficas

Aunque una buena parte del programa de hoja electrónica se puede ofrecer en forma de listado común, los métodos en que los cuatro micros manipulan la visualización de información en la pantalla son distintos. Para producir visualizaciones atractivas y manipulación en pantalla en cada máquina, proporcionamos listados separados de las rutinas para gráficos para cada ordenador. Comenzamos ofreciendo los listados para el C64. Estos listados manipulan la impresión de la cuadrícula de la hoja electrónica, la impresión de datos en las celdas de la hoja electrónica y el movimiento del cursor en la hoja. Tras digitar la sección de programa adecuada y ejecutarla, verá en la pantalla la cuadrícula (junto con un conjunto inicial de datos) y podrá desplazar el cursor por la hoja. No obstante, la mayoría de las funciones de la hoja electrónica que mencionamos no estarían disponibles todavía, porque serán el tema que desarrollaremos en futuros capítulos

Commodore 64

Liz Heaney

```
10 REM **** HOJA ELECTRONICA CBM 64 ****
15 REM ** PREPARAR SERIES CARACTERES CBM **
20 FOR I=1 TO 7:L7$=L7$+CHR$(195):NEXT
30 L5$=LEFT$(L7$,5):T$=CHR$(178):B$=CHR$(194)
40 S5$="  ":X$=CHR$(219):I$=CHR$(177)

100 GOSUB 3000:REM PREPARAR MATRICES Y VARIABLES
110 GOSUB 1000:REM IMPRIMIR PANTALLA
120 GOSUB 1700:REM IMPRIMIR DATOS EN PANTALLA
130 GOSUB 1100:REM RUTINA PRINCIPAL EXPLORACION DEL
    TECLADO
999 STOP
1000 PRINT CHR$(147);CHR$(145);CHR$(5):POKE
    53280,6:POKE 53281,6
1005 PRINT"          C O L U M N A S"

1006 PRINT
1007 PRINT "FILA     1.    2.    3.    4.    5."
1010 PRINT "       "CHR$(176);L5$;T$;
    L5$;T$;L5$;T$;L5$;T$;L5$;CHR$(174)
1020 FOR C=1 TO 7
1030 PRINT "   "CHR$(C+64);".      "B$;S5$;B$;S5$;
    B$;S5$;B$;S5$;B$;S5$;B$
1040 PRINT "   ";L7$;X$;L5$;X$;L5$;X$;L5$;X$;
    L5$;X$;L5$;CHR$(179)
1050 NEXT C
1060 PRINT "   "CHR$(C+64);".      "B$;S5$;
    B$;S5$;B$;S5$;B$;S5$;B$;S5$;B$
1070 PRINT "   ";L7$;I$;L5$;I$;L5$;I$;L5$;
    I$;L5$;I$;L5$;CHR$(189)
1080 RETURN
1100 P$=CHR$(Y+64)+MID$(STR$(X),2,2):PRINT
    CHR$(19);"CELDA:";P$;"  "
1110 GET A$:IF A$="" THEN 1110
1120 IF A$=CHR$(29) THEN 1200:REM MOVER DERECHA
1130 IF A$=CHR$(157) THEN 1300:REM MOVER IZQUIERZA
1140 IF A$=CHR$(17) THEN 1400:REM MOVER ABAJO
1150 IF A$=CHR$(145) THEN 1500:REM MOVER ARRIBA
1152 IF A$=CHR$(133) THEN GOSUB 6000:REM F1 IMPRIMIR
    PANTALLA AYUDA
1155 IF A$=CHR$(137) THEN GOSUB 2000:REM F2 ENTRAR
    FORMULA
1158 IF A$=CHR$(134) THEN GOSUB 5150:REM F3 ALMACENAR
    HOJA ACTUAL
1160 IF A$=CHR$(135) THEN GOSUB 2300:REM F5 CALCULAR
    HOJA
1165 IF A$=CHR$(13) THEN RETURN
1170 IF A$>"0" AND A$<"9" THEN GOSUB 2100:REM ENTRAR
    DATOS NUMERICOS
1180 IF A$=CHR$(139) THEN GOSUB 5000:REM F6 BORRAR
    HOJA
1185 IF A$=CHR$(138) THEN GOSUB 5100:REM F4 RETOMAR
    HOJA ANTERIOR
1187 IF A$="G" THEN GOSUB 5200
1188 IF A$=CHR$(136) THEN GOSUB 5700:REM F7 DECIDIR COL
    O FILA
1189 IF A$=CHR$(140) THEN GOSUB 7000:REM F8 RUTINAS
    CARGAR/GUARDAR
1190 GOTO 1110:REM VUELTA A EMPEZAR
1200 REM **** MOVER DERECHA ****
1210 IF X=15 THEN 1100
1220 IF X=H2 THEN GOSUB 1600:X=X+1:H1=H1
    +1:H2=H2+1:GOTO 1270
1230 GOSUB 1660:LET X=X+1:GOSUB 1650:GOTO 1100
1270 GOSUB 1800:GOSUB 1700:GOTO 1100
1300 REM **** MOVER IZQUIERDA ****
1310 IF X=1 THEN 1100
1320 IF X=H1 THEN GOSUB 1600:X=X-1:H1=H1-1:
    H2=H2-1:GOTO 1370
1330 GOSUB 1660:LET X=X-1:GOSUB 1650:GOTO 1100
1370 GOSUB 1800:GOSUB 1700:GOTO 1100
1400 REM **** MOVER ABAJO ****
1410 IF Y=15 THEN 1100
1420 IF Y=V2 THEN GOSUB 1600:Y=Y+1:V1=V1
    +1:V2=V2+1:GOTO 1470
1430 GOSUB 1600:LET Y=Y+1:GOSUB 1650:GOTO 1100
1470 GOSUB 1850:GOSUB 1700:GOTO 1100
1500 REM **** MOVER ARRIBA ****
1510 IF Y=1 THEN 1100
1520 IF Y=V1 THEN GOSUB 1600:Y=Y-1:V1=V1
    -1:V2=V2-1:GOTO 1570
1530 GOSUB 1660:LET Y=Y-1:GOSUB 1650:GOTO 1100
1570 GOSUB 1850:GOSUB 1700:GOTO 1100
1600 REM **** APAGAR CURSOR ****
1610 CU=1023+40*(V(Y+1-V1)+H(X+1-H1):
    POKE CU,PEEK(CU)-128
1620 POKE CU+1,PEEK(CU+1)-128:POKE CU+2,
    PEEK(CU+2)-128
1630 POKE CU+3,PEEK(CU+3)-128:POKE CU+4,
    PEEK(CU+4)-128:RETURN
1650 REM **** ENCENDER CURSOR ****
1660 CU=1023+40*(V(Y+1-V1)-1)+H(X+1-H1):
    POKE CU,PEEK(CU)+128
1670 POKE CU+1,PEEK(CU+1)+128:POKE CU+2,
    PEEK(CU+2)+128
1680 POKE CU+3,PEEK(CU+3)+128:POKE CU+4,
    PEEK(CU+4)+128
1690 GOSUB 1900:RETURN
1700 REM **** IMPRIMIR DATOS EN HOJA ****
1710 FOR I=0 TO 7
1720 PRINT CHR$(19);:FOR C=1 TO V(I+1)-1:
    PRINT CHR$(17);:NEXT C
1730 FOR J=0 TO 4
1735 P$=MID$(STR$(MAT(I+V1,J+H1)),2)
1740 PRINT TAB(H(J+1)-1);"      ";CHR$(145)
1745 PRINT TAB(H(J+1)+4-LEN(P$));P$;
1750 NEXT J,I
1760 GOSUB 1650:RETURN
1800 REM **** IMPRIMIR NUMERO COLUMNA ****
1810 PRINT CHR$(19);CHR$(17);CHR$(17);CHR$(17);
1820 FOR I=H1 TO H2:PRINT TAB(7+6*(I-H1))
    I;CHR$(157);".  ";
1830 NEXT I:RETURN
1850 REM **** IMPRIMIR NUMEROS FILA ****
1860 PRINT CHR$(19);:FOR I=1 TO 5:PRINT CHR$(17);: NEXT
1870 FOR C=V1 TO V2
1880 PRINT TAB(1)CHR$(C+64);CHR$(17)
1890 NEXT C:RETURN
1900 REM *** FORMULA DE CELDA ACTUAL ***
1920 LET D$=F$((Y-1)*15+X)
1930 GOSUB 1950:REM MOVER CURSOR HASTA LINEA
    ENTRADA
1935 PRINT CHR$(18);"FORMULA:
1940 PRINT CHR$(145);CHR$(18)"FORMULA:";D$
1945 PRINT CHR$(19):RETURN
1950 REM **** CURSOR A LINEA ENTRADA ****
1960 PRINT CHR$(19);:FOR K=1 TO 22:PRINT
    CHR$(17);:NEXT K
1970 RETURN

3000 REM **** PREPARAR MATRICES ****
3010 DIM H(5),V(8),ST(20),ST$(20),E$(20),G$(20)
3020 FOR C=0 TO 4
3030 H(C+1)=6*C+10:REM CALC POS X
3040 NEXT C
3050 FOR C=1 TO 8
3060 V(C)=2*C+4:REM CALC POS Y
3070 NEXT C
3075 X=1:Y=1
3080 H1=X:H2=X+4:V1=Y:V2=Y+7
3090 REM ***** MATRICES HOJA *****
3100 DIM MAT(15,15):DIM MC(15,15)
3110 FOR I=1 TO 15:FOR J=1 TO 15
3120 MAT(I,J)=I*J
3130 NEXT J,I
3140 DIM F$(225)
3150 RETURN
```

Jugar con palabras

Finalmente veremos cómo el FORTH manipula las series

Ya hemos visto la palabra ." , que es bastante fácil de usar, pero hay otras manipulaciones de series que son más complicadas. Existen dos problemas principales: en qué zona de la memoria almacenar la serie y cómo hacerlo. Tales consideraciones son engañosas, por lo que, como es natural, querrá definir palabras potentes que en el momento de utilizarlas le permitan olvidar todos los detalles.

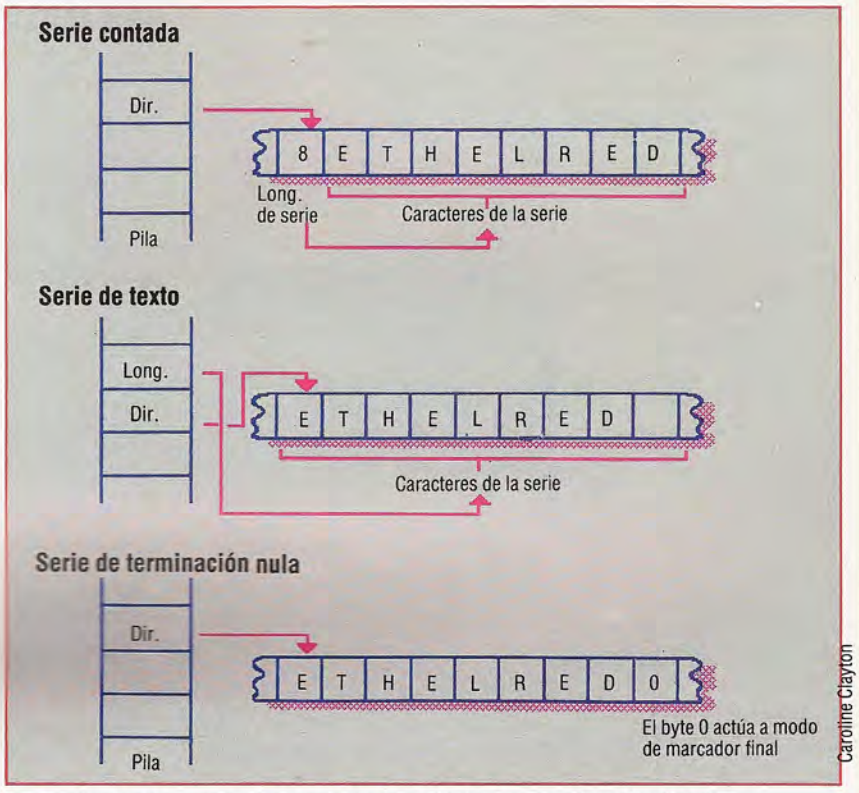
En primer lugar, hay dos lugares principales donde almacenar una serie: en los campos de parámetros asignados (ALLOT) de palabras del diccionario, y en una zona llamada *relleno*. El relleno es para el almacenamiento de series muy temporal, porque el sistema FORTH lo emplea con mucha frecuencia. Por ejemplo, si se incluye algo en el diccionario, o si se interpreta una palabra FORTH desde el teclado, el relleno puede quedar desplazado o bien sobreescrito. En segundo lugar, en FORTH hay dos maneras de almacenar una serie. En ambos casos los caracteres ASCII de la serie se almacenan por orden en algún lugar de la memoria; las diferencias radican en cómo especifique la longitud de la serie.

En una *serie contada,* la longitud (menos de 225 bytes) se almacena como un único byte inmediatamente antes de los caracteres. Para aludir a una serie de este tipo, puede utilizar un número, que es la dirección de este byte de cuenta. Por lo tanto, aunque no puede colocar la serie propiamente dicha en la pila, tras haber establecido la serie puede colocar su dirección en la pila. En el caso de una *serie de texto,* la longitud no se almacena para nada como parte de la serie. Esto significa que para aludir a la serie necesita dos números, la dirección del primer carácter, así como la longitud.

Las series contadas son más fáciles de manipular en la pila (el número adicional necesario para las series de texto hace que las manipulaciones de pila se vuelvan desproporcionadamente complicadas). Sin embargo, tienen un límite de 255 bytes y no se pueden utilizar para subseries, series que constituyen partes de series más grandes, porque en medio de la serie completa, tendría que insertar un byte de cuenta para la subserie.

Veamos algunas palabras que definen variables de serie. Una variable de serie mantiene su texto en su campo de parámetros, de modo que una vez que adjudica (ALLOT) una cierta cantidad de espacio para una serie, el mismo permanece fijo, por lo que nunca podrá destinarle series más largas. $VARIABLE, por consiguiente, requiere que especifique no sólo un valor inicial, sino también una longitud máxima para ulteriores asignaciones. La longitud máxima habrá de ser mayor que el valor inicial.

Una palabra útil es ASCII. La mayoría de los



Caroline Clayton

FORTH tienen incorporada esta palabra, si bien esto no es preceptivo en el estándar. Apila el código ASCII del carácter que le sigue y, si se utiliza en una definición de dos puntos, compila un literal numérico para este número de modo que se lo apila en el tiempo de ejecución.

```
32 CONSTANT BL (código ASCII para un espacio)
:ASCII(– –código ASCII) (utilizado para formar
        carácter ASCII)
   BL WORD (tomar siguiente palabra)
   1+C@ (y tomar el código ASCII para
        su primer carácter)
   STATE @ IF (si compilando definición dos puntos)
      [COMPILE] LITERAL(LITERAL es inmediata)
   THEN
;IMMEDIATE(ASCII tiene acciones de compilaciones
especiales)
;$ VARIABLE(tamaño– –)(utilizado en la forma —)
      (tamaño $VARIABLE nombre inicializador")
      (se prepara el campo de parámetros para conte-
      ner 1 byte para la)
      (longitud máxima y luego una serie contada
      para su valor)
   CREATE
   255 MIN (asegura que el tamaño no sea
   demasiado grande)
   1 ALLOT (para longitud máx.)
   ASCII "WORD(tamaño, inicializador como serie
   contada)
   DUP C@ ROT MAX(inicializador, longitud máx.)
   DUP HERE 1— C!(rellenar longitud máx.)
   HERE SWAP 1+ALLOT (inicializador, dirección
   para colocarlo)
   OVER C@1 + CMOVE (copiar inicializador)
DOES>(pfa– –pfa+1)(apila el valor como
        serie contada)
   1+
;
```

Formatos de series

El FORTH permite retener series en la memoria de tres formas diferentes. Cada método emplea un número de dirección (que se puede manipular fácilmente mediante la pila del FORTH) que señala la zona de memoria que contiene a la serie. Sin embargo, las tres técnicas utilizan distintos métodos para determinar la longitud de la serie. El formato «serie contada» almacena esta información en el primer byte del área de memoria que contiene la serie, mientras que el formato «serie de texto» retiene este número por separado en la pila, permitiendo la manipulación de pila del mismo. Algunas versiones de FORTH soportan el formato «terminación nula», en el cual el fin de la serie se indica simplemente mediante un byte cero

Palabras útiles

STATE (—dirección) Es una variable que lee las palabras inmediatas. Si su valor es verdadero (no 0), se las utiliza para compilar una definición de dos puntos.

WORD (delimitador—serie contada) Se utiliza para leerle la siguiente palabra a lo que ya se ha digitado en el teclado. P. ej., es lo que utiliza CREATE para tomar el nombre de sus nuevas palabras. Suele ser 32 (para un espacio), pero se pueden utilizar otros delimitadores, como " en nuestra definición de $VARIABLE. Algunas palabras del FORTH estándar hacen este truco, como ." and (usando "and) como delimitadores. Tenga cuidado al emplear la palabra con rapidez. Es probable que se almacene en el relleno, donde no permanecerá mucho tiempo.

EXPECT (dirección, número máximo de caracteres—) Se utiliza para tomar nuevas entradas del teclado. Los caracteres, hasta el número máximo especificado, se leen en la memoria comenzando por la dirección dada. Interrumpe la lectura cuando se pulsa ENTER o al digitar la cantidad máxima.

SPAN (–dirección) Esta variable la establece EXPECT para mostrar cuántos caracteres se han leído. Sólo la posee el FORTH-83; los más antiguos ponen caracteres NUL tras la serie para señalar dónde termina.

PAD (—dir. del relleno)

CMOVE (desde dirección, hasta dirección, número de caracteres– –) Copia el número de bytes especificado, comenzando desde la dirección «desde», en la memoria que empieza en la dirección «hasta», de modo que se utiliza para copiar series. Atención: si la zona de memoria «hasta» está después de la zona «desde», y las dos zonas se superponen, entonces algunos de los bytes de la zona «desde» se sobreescribirán antes de que se los lea, por lo que no será una copia fidedigna.

CMOVE> (desde dirección, hasta dirección, número de caracteres– –) Esta palabra, existente sólo en FORTH-83, evita la trampa de CMOVE al copiar los bytes por un orden diferente. No obstante, tiene una trampa propia cuando la zona «hasta» está antes de la zona «desde».

COUNT (serie contada – – dirección, longitud de serie texto) La serie queda igual, pero COUNT realiza una conversión entre las dos formas distintas de referirse a ella en la pila

Ésta es una definición bastante complicada, a lo que en cierto modo contribuye la falta, por parte del FORTH, de variables locales y también sus manipulaciones de pila. No obstante, su complejidad es sólo a escala muy pequeña y una vez que la tenga en funcionamiento podrá olvidarse de su funcionamiento interno, permitiéndole pensar más en términos de series que de secuencias de caracteres en la memoria.

Ahora vamos a definir $@ y$! para estas $VARIABLES, utilizando el formato de series de texto (dirección y longitud) en la pila para hacer referencia a estos valores.

```
:S@ (pfa+ 1 para variable de serie– –serie de texto)
    COUNT
;
:$! (serie de textos, pfa+1 para variable de serie– –)
    (trunca el valor si es demasiado grande para la variable)
DUP 1–C@ROT       (dir,pfa, long. máx., long. real)
MIN               (dir,pfa+1, long. a usar)
OVER OVER SWAP    (dir,pfa+1,long., long.,pfa+1)
C!                (almacenar nueva long.)
SWAP 1+SWAP       (dir,pfa+2,long.)
CMOVE
;
```

Ya puede establecer variables en serie, como en:

```
0 $VARIABLE DIGITOS 0123456789"
```

(El 0 se incrementa automáticamente a 10 para hacer frente a los 10 caracteres presentes en realidad. Pero después de eso, la longitud máxima de la variable en serie queda fija en 10.)

```
DIGITOS $@TYPE
```

(visualiza 0123456789)

```
255 $VARIABLEXPAD XXX"
```

(un relleno alternativo, más estable, con espacio para 255 caracteres.)

Sería agradable poder cambiar los valores de las series desde el teclado. He aquí dos palabras, " y $INPUT, que le permiten realizar el equivalente de las sentencias de asignación del BASIC LET A$="..."

```
:"      (–,–dirección, longitud de serie texto)
        (utilizada fuera de definiciones de dos puntos en forma — "texto")
ASCII"WORD COUNT      (tomar serie de texto)
XPAD $!               (colocarla en XPAD)
XPAD $@               (dejar dir. y long.)
;
```

La palabra " toma su serie de la misma manera en que palabras como VARIABLE toman el nuevo nombre, motivo por el cual no funcionarán adecuadamente en definiciones de dos puntos. Sin embargo, puede utilizarla como en:

```
20 $VARIABLE MASCOTA gato"
"cotorra"MASCOTA $!MASCOTA $@TYPE
"pangolín"MASCOTA $! MASCOTA $@TYPE
```

He aquí una palabra para imitar sentencias de entrada:

```
:$INPUT (pfa+1 de variable en serie– –)
        (espera que digite una serie y la copia en la variable.)
  DUP DUP 1+        (pfa+1,pfa+1,pfa+2)
  SWAP 1–C@2–       (pfa+1,pfa+2,long. máx.–2)
  EXPECT
  SPAN@ SWAP C!
;
```

El programa informa a EXPECT que no tome más caracteres para los que haya sitio en la variable. No obstante, una versión de FORTH permite escribir después de la serie un carácter NUL (nulo) adicional. De modo que, por razones de seguridad, se le ha restado 2 a la longitud máxima. De lo contrario, el NUL adicional podría alterar el encabezador de la siguiente palabra del diccionario.

En FORTH-79 o figFORTH, EXPECT escribirá un NUL o dos en todo caso, que en realidad son vitales. Los FORTH más antiguos no poseen SPAN y, en consecuencia, tendrá que escribir su propia palabra para hallar la longitud de la serie, contando los caracteres hasta el NUL.

Ahora, por fin, podemos definir una versión en FORTH del programa en BASIC más satisfactorio:

```
10 PRINT "Cómo te llamas?"
20 INPUT A$
30 PRINT "Encantado de conocerte, ";A$;"."
40 PRINT "Cuídate! Adiós."
```

Un programa comparable en FORTH, que emplea instrucciones definidas por nosotros, es:

```
30 $VARIABLE NOMBRE

:HOLA!
  ."Hola! Soy el ordenador. Quién eres tú?"CR
  NOMBRE $INPUT
  ."Vaya! Es un nombre increíble, chico,"NOMBRE $@TYPE
  ."."(digitar punto aparte)CR
  ."Cuídate, oyes? Que lo pases bien."
  CR
;
```

El gran mérito del BASIC es que es muy fácil escribir programas cortos, a causa de sus útiles facilidades para cosas como aritmética de punto flotante, matrices y series. Donde el BASIC pierde soltura es cuando se trata de escribir programas más largos. Usted nunca puede abstraerse de la estructura de bajo nivel del programa: los números de línea, por ejemplo. Por el contrario, observe nuestra palabra HOLA! en FORTH. Comprenderla no le supondrá mucho esfuerzo, sabiendo aproximadamente lo que hacen $INPUT, $@ y otras palabras. Si quiere estudiarla con mayor profundidad, vuelva a las definiciones de estas palabras, pero así y todo sólo necesitará saber *qué* hacen (lo que esperan que haya en la pila y lo que dejan en ella) e ignorar todo acerca de *cómo* lo hacen. Por consiguiente, en FORTH no es necesario tener presente la estructura de bajo nivel.

La conclusión de todo esto es: aunque el FORTH parte como un lenguaje menos poderoso que otros, usted puede valerse de su ampliabilidad para restituir el equilibrio. Y, lo que es más, puede continuar ampliándolo hasta que llegue a superar con mucho ese otro lenguaje.

Gran Hermano

Chris S

El recién aparecido PC/AT, de IBM, que se aproxima a un miniordenador en cuanto a potencia y capacidad, estabiece un nuevo estándar para micros

A causa del prestigio alcanzado por IBM, en 1982 los microordenadores se pusieron repentinamente de moda en el ámbito empresarial, y el PC acaparó un considerable sector del mercado. Esto tuvo consecuencias cuyo impacto cabal ni la propia IBM habría podido prever. Gran parte de la tecnología utilizada en el IBM PC original se compró a otros proveedores, como el procesador Intel 8088, las unidades de disco de 5 ¼ pulgadas y, lo más importante, el sistema operativo PC-DOS. En su momento, ninguno de estos componentes representaba una tecnología avanzada, lo que suscitó muchas críticas, no obstante lo cual se vendió un gran número de máquinas, lo que, como es natural, generó una enorme base de software. Otros fabricantes, sintiéndose desplazados del mercado, siguieron el ejemplo de IBM y compraron una tecnología idéntica para producir «clonos IBM»: ordenadores que ejecutaban software IBM. Esto tuvo como consecuencia que el IBM PC se constituyera en el estándar industrial *de facto*.

Desde entonces, la reducción del precio de los chips de memoria, junto con unos procesadores cada vez más potentes, ha dado como resultado que las máquinas de gestión se equipen con 256 K como estándar (el PC original tenía sólo 64 K de RAM), lo que a su vez ha llevado a las firmas de software a producir programas más complejos y más ávidos de memoria.

La cima actual de esta tendencia la representa el PC/AT (*Personal Computer/Advanced Technology*: ordenador personal de tecnología avanzada). Este impresionante ordenador, que comienza a aproximarse a la potencia y capacidad de un miniordenador, está equipado con uno de los últimos procesadores: el Intel 80286. Este chip está equipado con un bus de datos de 16 bits y un bus de direcciones de 24 bits similar al del chip 8086 utilizado tanto en el Olivetti M-24 como en la gama Apricot, permitiendo que la máquina direccione más de 16 Mbytes de memoria. No obstante, el chip posee asimismo la capacidad para *gestión de memoria virtual,* lo que significa que la capacidad de memoria real que puede utilizar es de más de un gigabyte (1 000 Mbytes).

La gestión de memoria virtual es un proceso en virtud del cual el procesador puede tratar a su RAM disponible y el almacenamiento de apoyo como «memoria principal». Al ejecutar un programa, puede ser que en la RAM disponible no quepan todos los datos y el programa, de modo que el resto se retendrá en discos de alta velocidad. Cuando el ordenador requiere una información determinada y ve que la misma no está en RAM, los datos correspondientes se leen del disco en la zona de memoria sin utilizar, donde se actuará sobre ellos de la forma pertinente. De este modo, el ordenador *parece* tener muchísima más memoria de la que en realidad posee.

Esta técnica se ha adaptado para aplicarla en el PC/AT. El objetivo original de la gestión de memoria virtual era el de usar la RAM en conjunción con métodos de almacenamiento más económicos, como discos. Sin embargo, con la caída de precios de los chips de RAM, esto no es fuente de preocupación en el sistema PC/AT. El problema está en el sistema operativo.

El PC-DOS, y los MS-DOS estrechamente relacionados con el mismo, son capaces de direccionar 640 K de memoria. Cuando se desarrolló el PC-DOS, esta memoria se consideraba más que suficiente para cualquier aplicación previsible. No obstante, muchos modernos paquetes de software integrado se están aproximando rápidamente al límite de las capacidades del OS. Por consiguiente, se había de encontrar algún método para añadir memoria dentro de las limitaciones impuestas por el PC-DOS.

La respuesta ha sido tratar la memoria restante como *discos de silicio*. Éstos son bancos de memoria que pueden ser de cualquier tamaño (aunque en el PC/AT el tamaño por defecto es de 64 K) hasta igualar el de la memoria direccionable, y que el procesador trata como si fuera información retenida en una unidad de disco. Esto significa que el procesador no puede procesar la memoria directamente, sino que debe ir a buscar la información en bloques, que se puedan trasladar a la memoria direccionable y manipular allí. Cuando ya no se nece-

Tecnología avanzada
La entrada de IBM en el mercado del ordenador personal determinó un cambio en la actitud comercial hacia los microordenadores. Actualmente, la nueva máquina de la empresa, el IBM PC/AT, es uno de los micros más potentes que existen para uso de gestión. Operando hasta tres veces más rápido que el IBM PC, el AT también es capaz de acceder a hasta tres megabytes de memoria

IBM PC/AT

DIMENSIONES

540 × 420 × 160 mm

CPU

Intel 80286 trabajando a 6 MHz

MEMORIA

Base Model, 256 Kbytes; Enhanced Model, 512 Kbytes ampliables a 3 Mbytes

PANTALLA

Visualización de textos de 80 × 25 caracteres, con una resolución máxima de 640 × 200 pixels

INTERFACES

Ocho ranuras de ampliación para una amplia gama de interfaces

LENGUAJES DISPONIBLES

Todos los lenguajes principales utilizados comúnmente

DOCUMENTACION

Como siempre, tratándose de IBM, la documentación es exhaustiva, si bien algo formal

VENTAJAS

El PC/AT es uno de los microordenadores disponibles a nivel comercial más potentes del mercado, construido por el mayor fabricante de ordenadores del mundo. Es difícil ver en qué podría fallar

DESVENTAJAS

Incompatibilidad con parte del software PC existente

sitan más los bloques, la información actualizada o el programa se pueden volver a «escribir» en el disco de silicio. Por supuesto, aunque se traten como discos, el hecho de que los discos de silicio en realidad sean chips de RAM significa que el acceso es mucho más rápido que el de una unidad de disco convencional.

El PC/AT viene en dos configuraciones básicas. El sistema de precio mínimo, llamado Base Model 1 (modelo básico 1), contiene 512 K de RAM y una sola unidad de disco, mientras que el Enhanced Model (modelo mejorado) posee una unidad de disco rígido adicional de 20 Mbytes y un adaptador serie/paralelo.

Unidades de disco

Las unidades de disco incorporadas en el PC/AT, aun siendo discos flexibles de 5 ¼ pulgadas, representan una sustancial mejora respecto a las incorporadas en las versiones originales del IBM PC, que eran algo lentas y ruidosas, con una capacidad de apenas 160 K; las instaladas en el PC/AT tienen una capacidad de 1,2 Mbytes. Esto se debe en parte a la tecnología mejorada del diseño de las unidades y también a las mejoras introducidas en el DOS. El PC-DOS empaquetado en la máquina es la versión 3, que lee y escribe 15 sectores por pista en lugar de los ocho del PC. No obstante, para conservar la compatibilidad con versiones anteriores del IBM PC, la nueva máquina puede detectar discos de formato PC y adaptarse por sí misma a ellos; los PC, sin embargo, no pueden leer discos del PC/AT.

En el PC-DOS 3, además de las mejoras introducidas para sacar partido de la mayor capacidad de las unidades de disco, se han incorporado algunas instrucciones adicionales para mejorar su propio rendimiento. ATTRIB designa un archivo como de lectura solamente y LABEL permite asignar a un disco una «etiqueta de volumen». SELECT y COUNTRY son instrucciones similares, permitiendo la primera de ellas elegir el trazado del teclado y el formato hora/fecha requerido, y la segunda pasando automáticamente la hora/fecha y trazado del teclado a los de un determinado país. SHARE permite compartir archivos, FCBS especifica el número de bloques de control de archivos que se pueden abrir en un momento dado, y DEVICE permite instalar en el ordenador el tamaño y número de «discos virtuales». Por último, LASTDRIVE permite establecer la cantidad de unidades que puede utilizar el sistema.

En relación al espinoso problema de la compatibilidad, quizá sea sorprendente descubrir que el PC/AT no es totalmente compatible con el PC original. Esto se debe en parte a que las unidades de disco utilizadas en la nueva máquina, aunque mejoradas, son incapaces de hacer frente a algunas de las sofisticadas técnicas que se han desarrollado para impedir copiar el software PC. Otra dificultad reside en que algunos de los chips del PC/AT se han modificado respecto al original y, aunque se supone que son compatibles, el software que interroga directamente a estos chips podría encontrarse con que algunas de las direcciones han cambiado. La más notoria de ellas es la BIOS ROM, núcleo de la compatibilidad con el IBM PC, en la que se han introducido algunas mejoras. Lamentablemente, esto significa que también se han efectuado modificaciones en las direcciones de entrada de ciertas rutinas.

Este problema de incompatibilidad alcanza también al procesador. Aunque al Intel 80286 lo fabrica la misma empresa que produjo el procesador 8088 del PC, algunas de las instrucciones son distintas: el software para una máquina que use las instrucciones incompatibles no se ejecutará en la otra. Por último, el IBM PC original contenía interruptores DIP que en el PC/AT se han sustituido por CMOS RAM.

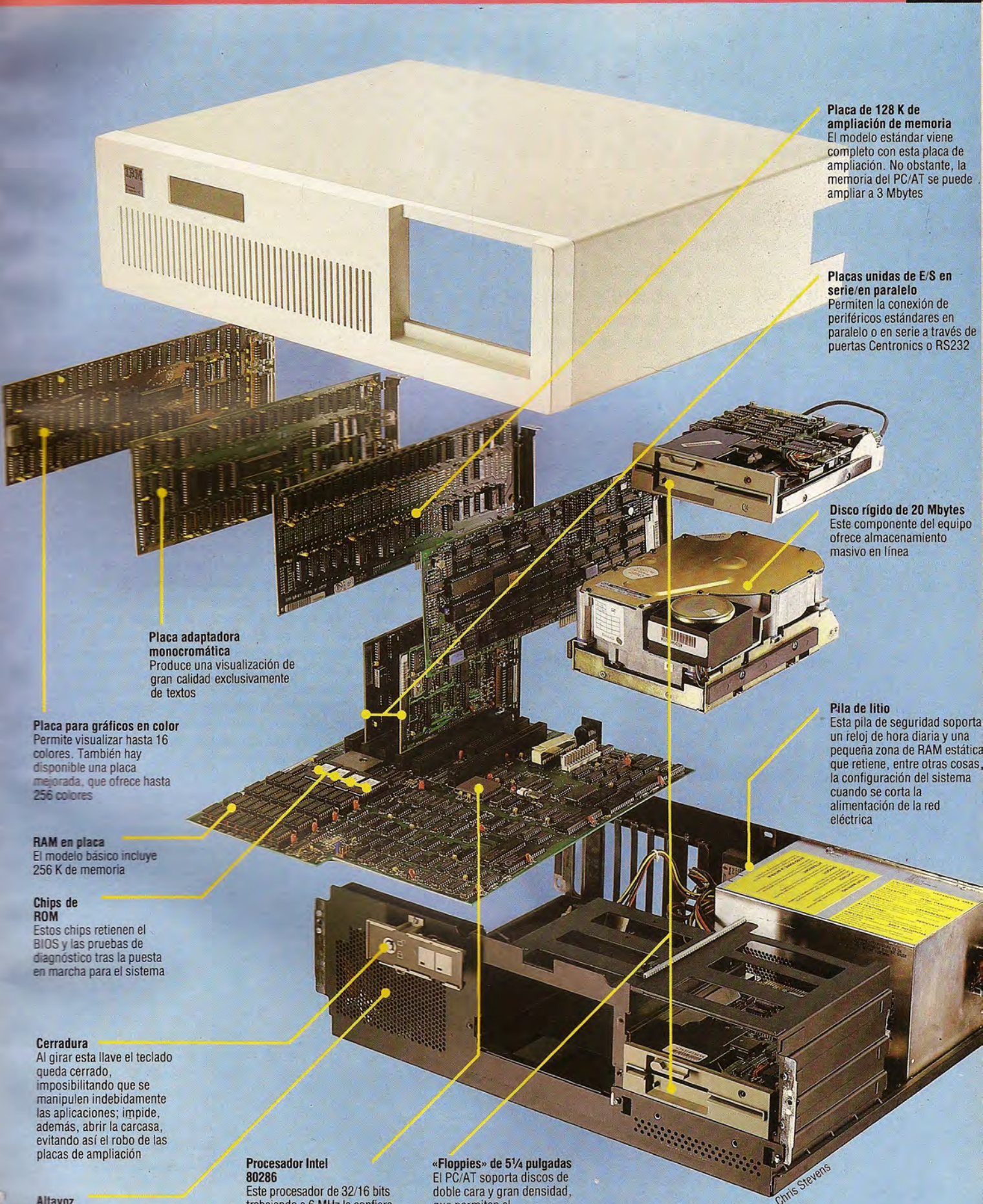
Debido a esto, se han tenido que ajustar muchos productos de software para encajar en el nuevo formato, principalmente con respecto a sus códigos de protección. Quizá el caso más notable sea el del simulador Microsoft Flight (FS1), que se considera como la prueba de compatibilidad IBM por antonomasia. Por lo tanto, Microsoft ha introducido el FS2, un paquete mejorado.

A pesar de estas dificultades, el IBM PC/AT es una máquina excelente, y casi todas las críticas que suscitara el PC original han tenido una respuesta positiva. Entre las mejoras destaca el teclado que, desde el punto de vista del tacto, es sin lugar a dudas el mejor que se haya producido jamás para un micro.

Como cabría esperar, la nueva máquina es considerablemente más rápida que el IBM PC original. La velocidad que en algunas aplicaciones llega a ser tres veces superior. Esto, unido a la vasta memoria a la que puede acceder esta máquina, arroja muy pocas dudas acerca del futuro del PC/AT.

Apariencia engañosa

La técnica de *gestión de memoria virtual* se desarrolló originalmente para usar en ordenadores centrales, y se utiliza para darle a un ordenador la apariencia de tener mucha más memoria de la que posee en realidad. El principio de la gestión de memoria virtual es bastante simple. A menudo, al ejecutar un programa o un conjunto de programas juntos, la memoria disponible resulta insuficiente, y por ello los programas se conservan en un almacenamiento de apoyo de acceso rápido, como puede ser un disco rígido. Cuando se ejecutan los programas, el ordenador sólo carga las partes de un programa que se requieren para llevar a cabo la tarea de que se trate. Una vez acabada ésta, el programa se borra y se carga la siguiente parte del programa. Siempre y cuando el proceso sea suficientemente rápido, «parecerá» que todas las partes estén en la memoria y ejecutándose de forma simultánea. La aplicación práctica de la gestión de memoria virtual que se puede observar más fácilmente es en la multitarea, donde un ordenador ejecutará a la vez varios programas separados. Por ejemplo, podría ser necesario que un ordenador accediera a una carga de datos, los formateara y los imprimiera, manteniendo al mismo tiempo una red y facilidades para tratamiento de textos. Aquí, aunque parecerá que se están ejecutando los tres programas al mismo tiempo, habrá períodos en los que los programas estén «holgazaneando», a la espera de recibir, entrar o enviar mensajes desde un periférico. De este modo, los programas se pueden conservar en disco y cargar y ejecutar sólo cuando se los requiere. Mientras tanto, el ordenador puede utilizar la totalidad de su memoria para llevar a cabo otras tareas



Placa de 128 K de ampliación de memoria
El modelo estándar viene completo con esta placa de ampliación. No obstante, la memoria del PC/AT se puede ampliar a 3 Mbytes

Placas unidas de E/S en serie/en paralelo
Permiten la conexión de periféricos estándares en paralelo o en serie a través de puertas Centronics o RS232

Disco rígido de 20 Mbytes
Este componente del equipo ofrece almacenamiento masivo en línea

Placa adaptadora monocromática
Produce una visualización de gran calidad exclusivamente de textos

Placa para gráficos en color
Permite visualizar hasta 16 colores. También hay disponible una placa mejorada, que ofrece hasta 256 colores

Pila de litio
Esta pila de seguridad soporta un reloj de hora diaria y una pequeña zona de RAM estática que retiene, entre otras cosas, la configuración del sistema cuando se corta la alimentación de la red eléctrica

RAM en placa
El modelo básico incluye 256 K de memoria

Chips de ROM
Estos chips retienen el BIOS y las pruebas de diagnóstico tras la puesta en marcha para el sistema

Cerradura
Al girar esta llave el teclado queda cerrado, imposibilitando que se manipulen indebidamente las aplicaciones; impide, además, abrir la carcasa, evitando así el robo de las placas de ampliación

Altavoz
El PC/AT tiene incorporado un altavoz de 2 pulgadas

Procesador Intel 80286
Este procesador de 32/16 bits trabajando a 6 MHz le confiere al PC/AT su capacidad para «gestión de memoria virtual»

«Floppies» de 5¼ pulgadas
El PC/AT soporta discos de doble cara y gran densidad, que permiten el almacenamiento de hasta 1,2 Mbytes en un único disco

Chris Stevens

La suerte está echada

Con este capítulo finalizamos la construcción de nuestro tester digital y al mismo tiempo indicamos cómo comprobar las conexiones

Por las razones que explicamos ya en el capítulo anterior, se utilizan cinco LEDs idénticos. No es necesario utilizar zócalos para los LEDs, pero soldarlos directamente en las pequeñas patillas es bastante delicado. Si opta por emplear zócalos (que no están incluidos en la lista de componentes), recuerde que los zócalos DIL de 24, 28 y 40 patillas tienen la separación de patillas correcta (siete orificios en una placa matriz de 0,1 pulgadas).

Los LEDs se pueden montar en un trocito separado de placa matriz y conectarla a la placa principal mediante una corta manguera o un cable plano. Ello permitirá montar el visualizador en la ventana de una caja de instrumento pequeño. Las patillas 5 de los LED 3, 4 y 5 tienen conectados conductores que sirven para conmutar el punto decimal.

Se utiliza una fuente de alimentación simple, basada en dos reguladores de *tres terminales*. Esta

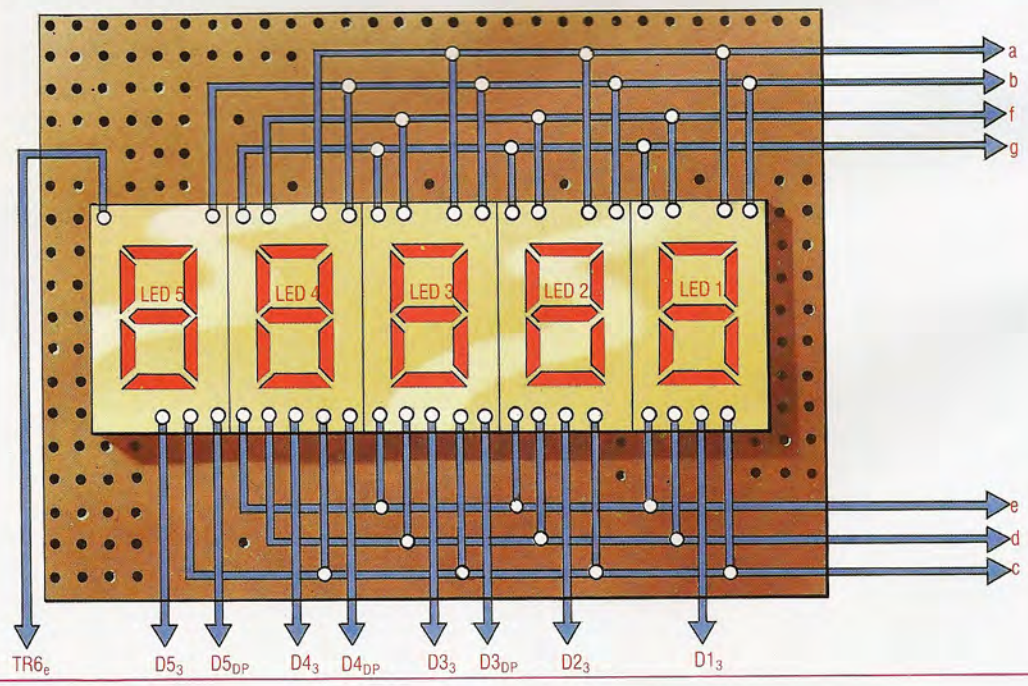
Lista de componentes

Cantidad	Ref.	Artículo
5	LED1-5	LED de siete segmentos de ánodo común
1	T1	transformador de 12 V
1	BR1	puente rectificador
1	C1	condensador electrolítico de 40 V 680 µF
1	C2	condensador de tantalio de 0,33 µF de perla
2	C3,C4	condensador de tantalio de 0,1 µF de perla
2	RG1,RG2	regulador de tres terminales de 5 V
1	—	disipador
2	D1,D2	diodo rectificador 1N4001
Varios		
1,5 m		hilo para arrollar
0,5 m		cable plano de 12 vías

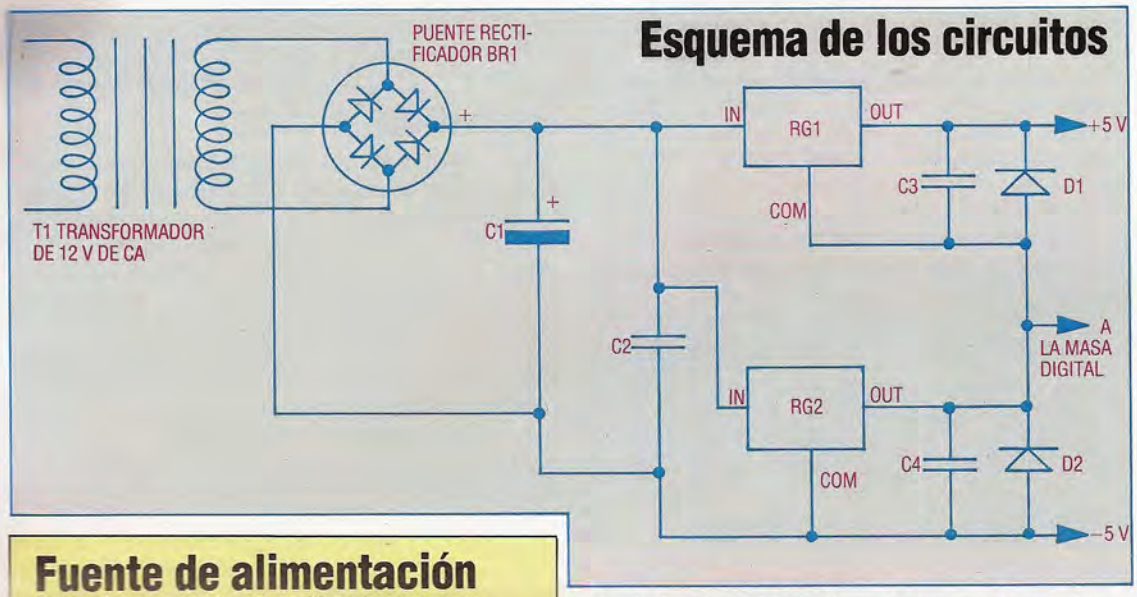
El visualizador LED

El de cinco dígitos para nuestro tester se monta en un trozo separado de placa matriz de 0,1 pulgadas (ver ilustración). Las siete señales que iluminan las barras LED (de a a f) son comunes para los cuatro visualizadores de dígitos situados más a la derecha, dado que estas señales se multiplexan entre los cuatro dígitos. Utilice el sistema de arrollamiento para realizar las conexiones de punto a punto necesarias. Luego se puede utilizar un trozo de cable plano para conectar estas siete líneas a la placa principal. Consultar la disposición de la placa principal para ver cómo se conectan estas líneas. Las señales de habilitación de dígitos desde la placa principal se conectan a la patilla 3 de cada visualizador LED. Nuevamente, remítase al diagrama de la placa principal. Leyendo este esquema desde la derecha, conecte $D1_3$ a D1, $D2_3$ a D2, $D3_3$ a D3, $D4_3$ a D4 y $D5_3$ a la conexión más sobre la derecha de los tres puntos marcados como D5. Vea que las conexiones del punto decimal, señaladas como Dn_{DP} y $TR6_e$ no se deben conectar en esta etapa, ya que serán el tema del último capítulo

Visualizador LED



Kevin Jones



Fuente de alimentación

El circuito es muy directo, utilizando un puente rectificador y dos reguladores de tensión de tres terminales para convertir una corriente de entrada de 12 V de corriente alterna en alimentaciones de + 5 V y − 5 V para la placa principal. Los condensadores del circuito están para reducir «picos» (fluctuaciones súbitas) de la potencia dada por el transformador

configuración permite emplear una fuente única de 12 V de CA y un rectificador de un solo puente. No es necesario que los dos diodos sean del tipo especificado; servirá cualquier diodo rectificador pequeño. Éstos impiden una condición que se conoce como *enganche,* que podría producirse cuando se utilizan dos reguladores en una fuente de alimentación bipolar.

C1 puede ser un condensador electrolítico moderadamente grande, pero asegúrese de que posea una tensión de trabajo suficientemente alta (al menos 35 V). C2, C3 y C4 deben ser condensadores de tantalio sólido de gota, que toleran muy mal las tensiones inversas. Asegúrese de conectarlos con la orientación correcta: el «+» del cuerpo indica el lado positivo.

Los componentes de la fuente de alimentación (a excepción del transformador de red) se pueden montar convenientemente en una placa de tiras. RG2 sólo tiene que suministrar unos pocos miliamperios y no necesitará disipador, pero RG1 estará trabajando mucho más cerca de sus límites y, en consecuencia, habrá de atornillarse a un disipador pequeño. Tenga en cuenta que la cápsula de los reguladores de tres terminales está conectada al terminal central común. Si bien la mayoría de los disipadores son de aluminio anodizado y no son muy conductores (eléctricamente), es muy conveniente asegurarse de que el disipador no toque el metal del chasis ni ningún cable desnudo.

Una vez montada la fuente de alimentación, conecte el transformador de red a la red (tome las consabidas medidas de seguridad al hacerlo) y conecte una resistencia de 100 ohmios entre la salida de + 5 V y masa. Si tiene algún tipo de voltímetro (aunque sea uno barato de tipo analógico), mida la tensión en la resistencia, que deberá ser muy próxima a 5 V. Si dispone de un osciloscopio, ajuste la entrada de corriente continua a una sensibilidad de alrededor de 10 V. Aplique las puntas de prueba en los extremos de la carga ficticia de 100 ohmios y compruebe en la pantalla que no haya un rizado apreciable. Si lo hubiera, podría anular las prestaciones del DVM.

Los reguladores de tres terminales son dispositivos con los que es muy sencillo trabajar, pero pueden generar ruido de alta frecuencia. La disposición que vemos en la ilustración (al contrario de las conexiones indicadas en el esquema del circuito) ayudarán a minimizar este ruido.

Una vez probada la fuente de alimentación con resultados correctos, conéctela a la placa DVM principal, pero *no* inserte los IC en esta etapa. Utilice su tester para comprobar los + 5 V en la patilla 11 de IC1, en las patillas 4 y 8 de IC2 y en la 16 de IC3. Asimismo, compruebe los + 5 V en los colectores de TR1 a TR5, y los − 5 V en la patilla 1 de IC1. Ahora conmute a la escala de ohmios y conecte una punta de prueba del medidor al punto de masa analógica o digital. Ponga en contacto la otra punta de prueba con la patilla 3 de IC1, con la patilla 1 de IC2 y con la patilla 8 de IC3. Las resistencias que indique el medidor deberán ser muy bajas (de menos de 1 ohmio).

IC3 es un chip TTL corriente y se puede manipular sin ninguna precaución especial: tan sólo enchúfelo en el zócalo. Mientras tanto, asegúrese, sin embargo, de que la muesca del chip esté arriba, como se indicó en el último capítulo. IC2 e IC1 no se pueden tratar del mismo modo. IC2 es la versión CMOS del chip temporizador NE555, y se dice que está totalmente protegido contra descargas electrostáticas, pero es mejor no correr ningún riesgo. IC1 se destruirá sin duda alguna a causa de una manipulación incorrecta, y si con anterioridad usted no ha trabajado nunca con dispositivos sensibles electrostáticamente, a continuación le explicamos qué debe hacer.

En primer lugar, corte longitud adecuada de papel de aluminio de cocina y colóquelo sobre el banco. Luego desconecte la fuente de alimentación y coloque la placa del circuito en el centro de la hoja. Introduciendo la mano por debajo del papel, presiónelo firmemente contra el lado de las patillas del circuito. IC1 e IC2 se suministrarán con algún embalaje conductor. Éste podrá tener forma de es-

puma plástica negra, soporte de «goma» negra o soporte de plástico con tratamiento especial. Sea como fuera, independientemente de cómo vengan embalados estos dos chips, colóquelos sobre el papel de aluminio dentro de su embalaje.

Apoye su codo libre sobre el papel y manténgalo allí mientras libera al chip de su embalaje. Manteniendo el codo en la misma posición, inserte el chip en su zócalo. Cuando los chips estén bien insertados, ya puede retirar el codo y tirar el papel de aluminio. (Es muy fácil olvidarse de quitar el papel cuando conecte la fuente de alimentación, de modo que esté atento.)

¡Atención!

Todo proyecto que implique potencia eléctrica es peligroso. Desconecte la alimentación de la red antes de trabajar con la placa

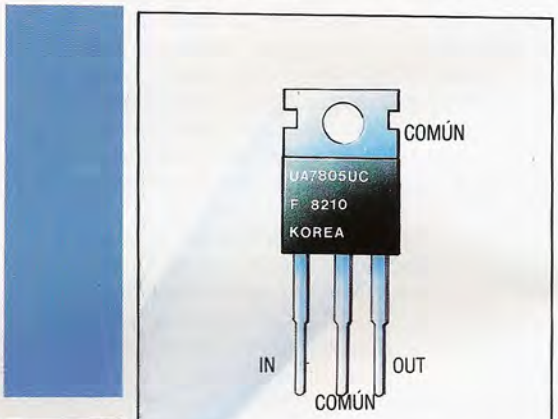
Montaje del regulador

Los componentes para el circuito de rectificación y regulación de la fuente de alimentación se deben montar sobre un trozo de placa de 36 tiras. Este tipo de placa es ideal para circuitos de esta clase. Suelde los terminales de los diversos componentes a las tiras, como se indica, y emplee cables aislados para unir las tiras entre sí. La entrada de corriente alterna del transformador se debe conectar a la placa principal del tester. La alimentación de −5 V se conecta a un punto próximo a la parte inferior de esta placa, y la línea de masa se conecta al punto principal de la masa analógica de la placa. Dado que la alimentación de +5 V ha de alimentar varios puntos de la placa principal, le aconsejamos que monte este cable alrededor del borde de la placa principal y lo derive donde resulte conveniente. Es importante observar la orientación de los condensadores y diodos, el puente rectificador y los reguladores. En especial, se deben insertar correctamente los reguladores de tensión de tres terminales. Para determinar la orientación, ponga el regulador sobre su cara posterior, de modo que los caracteres impresos del cuerpo queden arriba. De izq. a der., las patas son IN (entrada), COMÚN y OUT (salida)

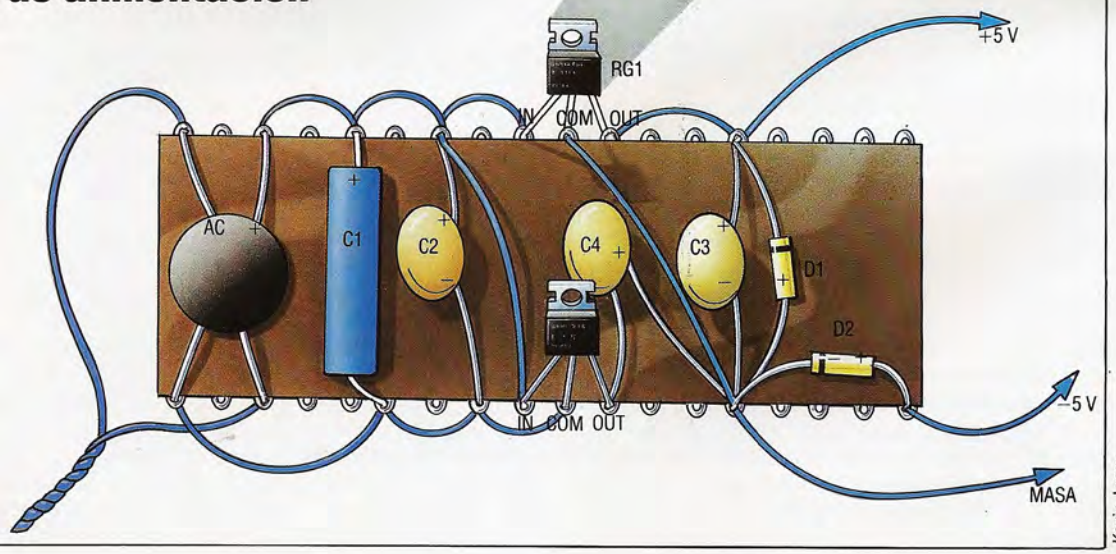
Probando la placa

Si la fuente de alimentación y los puntos de la fuente de alimentación a los IC están bien y los chips se han insertado de la forma descrita anteriormente, ahora puede intentar aplicar la alimentación a la placa. Si no ha conectado la patilla 9 de IC1 a la masa analógica, con un circuito abierto en los puntos IN− e IN+, puede esperar ver cómo en el LED1 y el LED2 se visualizan unos dígitos espurios. Si estos dígitos efectúan un ciclo a través de un patrón repetido, trate de mover la posición del cableado de interconexión de la placa.

Con la patilla 9 de IC1 conectada a la masa analógica y un circuito abierto en la entrada, verá una visualización de 00000 o un 1 o 2 espurios en el LED1. Con la patilla 9 de IC1 a masa e IN− e IN+ unidas, obtendrá una visualización estable (sin intermitencias) de 00000 o 00001. Si usted ha tenido que transigir con D1 (el diodo de referencia de tensión) o C1 (el condensador integrador) quizá no obtenga un rendimiento inicial tan satisfactorio. En cualquier caso, una visualización que efectúe ciclos indica que la disposición del cableado es deficiente, de modo que deberá modificar la disposición de los conductores hasta que la visualización sea estable.

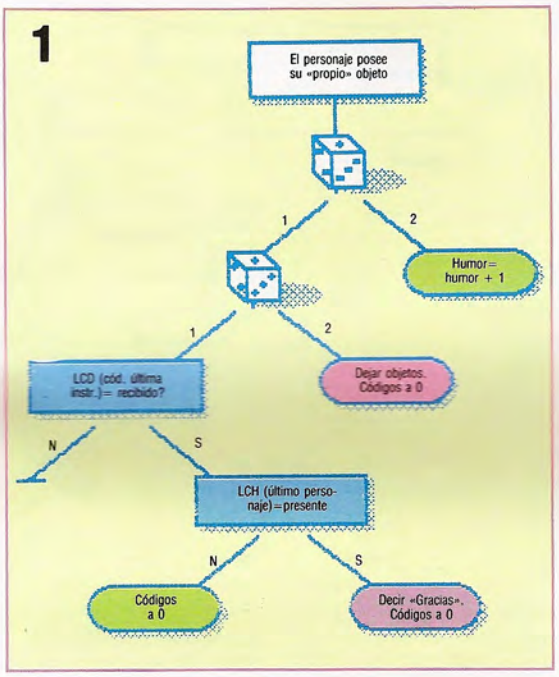


Placa de la fuente de alimentación



Kevin Jones

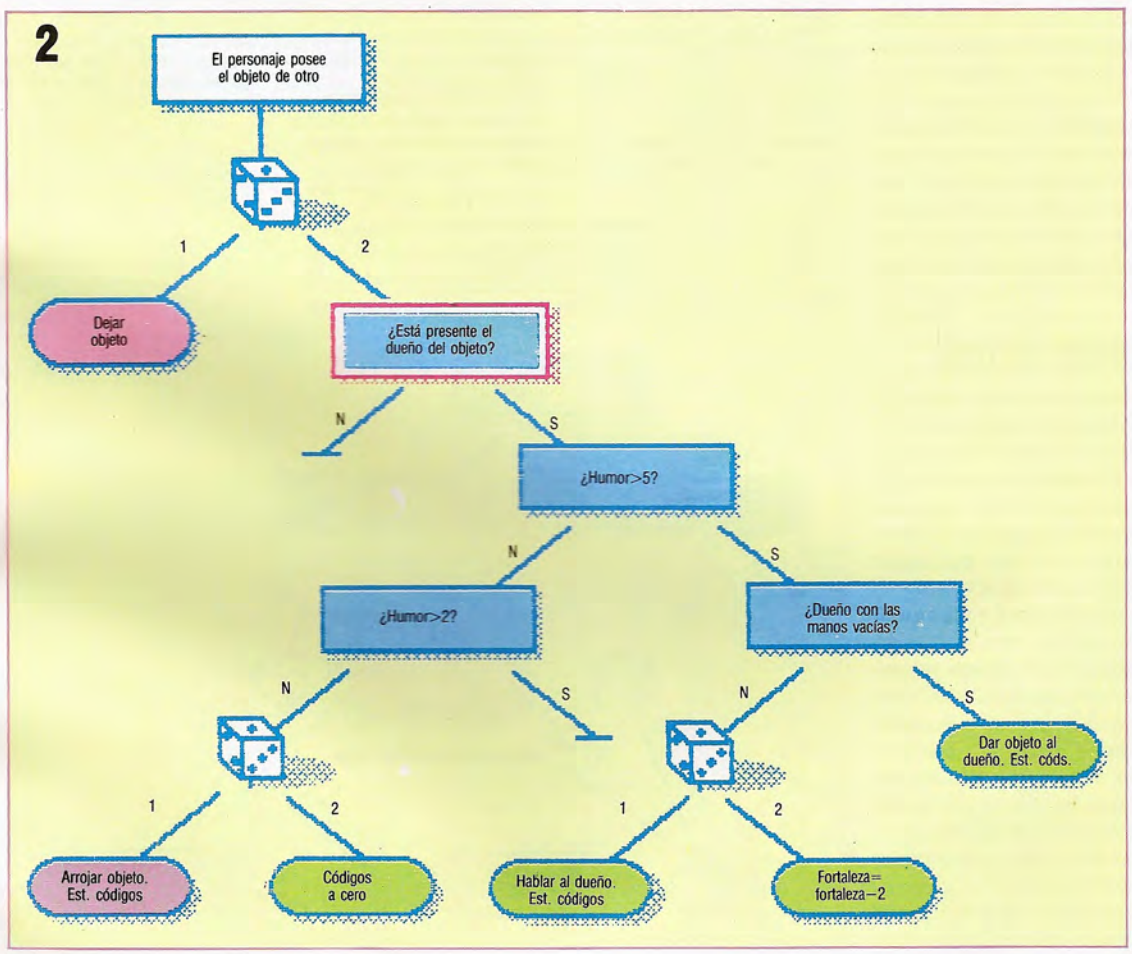
Lección «objetiva»



La manipulación de objetos es esencial en los juegos con personajes interactivos

La forma en que nuestros personajes manipulan los objetos es muy importante en el Dog and Bucket, y se deben tener en cuenta una gran cantidad de factores. En primer lugar, los objetos se integran en dos categorías principales: comestibles (el bocadillo y la empanada) y bebestibles, pero también se incluyen tres objetos (el cenicero, la banqueta y la lata) cuya finalidad es muy limitada, aparte de contribuir a «crear atmósfera». Sin embargo, como veremos, tiene implicaciones en la trama.

El otro atributo fundamental de los objetos es su pertenencia. Ciertas bebidas pertenecen, por ejemplo, a ciertos personajes, y es obvio que necesitaremos llevar el registro de las posesiones de cada uno. Igualmente importante es el hecho de que tendremos que tener en cuenta lo que desea cada personaje. Todos estos factores se pueden comprobar mediante estructuras arborescentes similares a al-



gunas que ya hemos visto con anterioridad, aunque más complejas. Para simplificar las cosas, abordaremos la cuestión de los objetos en cuatro etapas.

En primer lugar, consideremos los posibles cursos de acción que puede seguir un personaje que se halla en posesión de un objeto.

El primer diagrama muestra una estructura arborescente simple para tratar esta eventualidad. Entre los puntos a destacar se incluyen los símbolos de los dados, que representan *nudos aleatorios* cuyo valor (que determinará el nudo hacia el cual se dirigirá la siguiente bifurcación) se fija al azar cuando se encuentren en el recorrido del árbol.

Además de nudos aleatorios, en el diagrama hay otros tres tipos de nudos. Los rojos son nudos terminales, en donde se imprimirá un mensaje y se actualizarán las variables. Los nudos verdes también son nudos terminales, pero éstos sólo actualizarán variables sin alterar la visualización, de modo que cuando entremos el árbol en nuestro programa podremos ocuparnos de ellos por separado. Por último, los nudos azules son simples nudos de elección, cada uno de los cuales retendrá un valor condicionado y se bifurcarán en consecuencia.

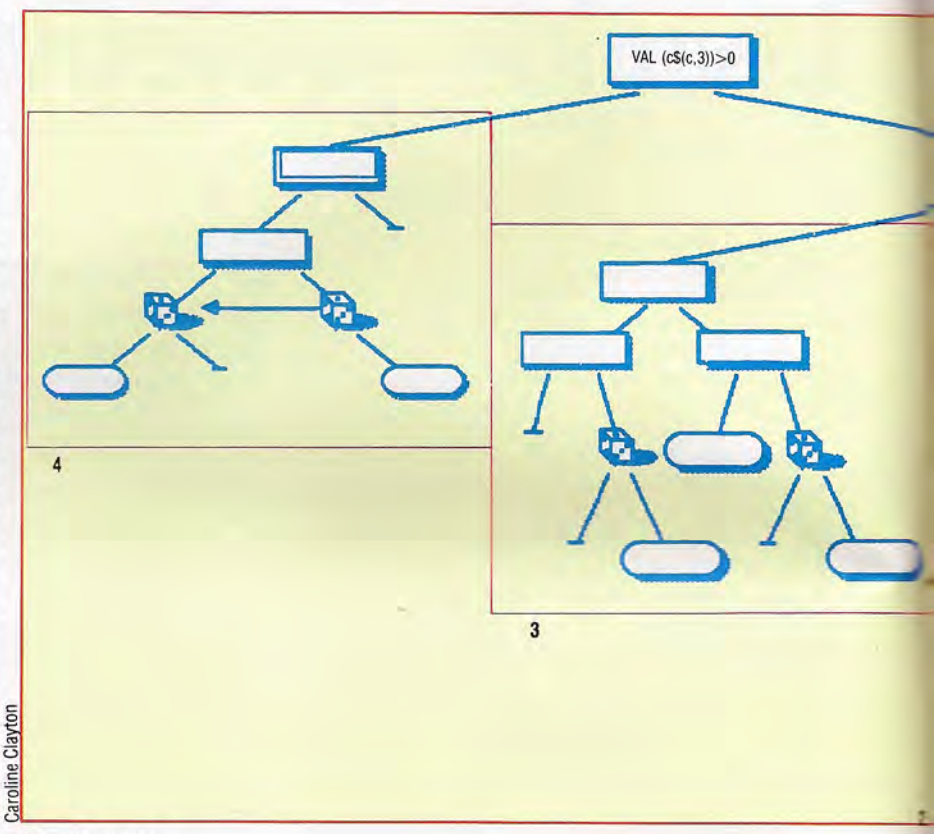
A partir de este diagrama usted puede ver que cuando un personaje posee un objeto «propio», es posible uno de cuatro resultados, representados por los cuatro nudos terminales. Existe la posibilidad aleatoria de que mejore el humor del personaje, en virtud de poder abandonarse a su bebida favorita. Por el contrario, aunque las probabilidades son menores, el personaje podría dejar el objeto o bien, si acaba de recibirlo y quien se lo entrega está todavía presente, decir Gracias. Observe que tres de los nudos terminales resultan en el establecimiento en cero de los códigos LCH y LCD, con lo cual se da por terminada cualquier interacción previa con otros personajes.

Las estructuras como éstas son puramente arbitrarias, lo que confiere la libertad de diseñar árboles bastante diferentes para los mismos fines. Teniendo esto presente, pasemos al segundo diagrama, que muestra qué podría suceder cuando el personaje en cuestión ha entrado en posesión del objeto de algún otro personaje.

Nudos del segundo árbol

Los nudos de este árbol poseen un código de color de estilo similar a los del árbol anterior, con la excepción de un nudo enmarcado en rojo. Éste actúa de modo similar a un nudo terminal, de modo que, al llegar al mismo, el programa saltará fuera del árbol hasta una rutina situada en algún otro lugar del programa. No obstante, una vez ejecutada dicha rutina, y en función del resultado obtenido, el programa podría volver a saltar al árbol y continuar su descenso a través del mismo.

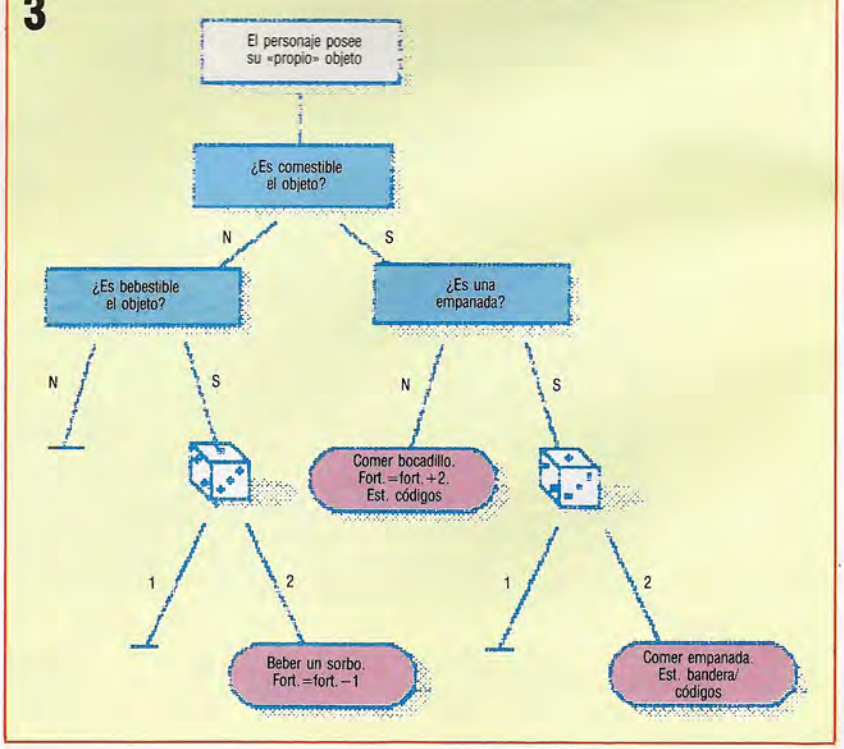
Para averiguar si el propietario del objeto retenido por nuestro personaje se halla en la misma habitación, necesitamos pasar por los atributos de escenario (c$(personaje,2)) de cada persona. En este punto, el nudo en cuestión nos empuja fuera del árbol, a una rutina sencilla que hace esto por nosotros. Si el personaje en cuestión está presente, dará un salto hacia atrás en el árbol comenzando a partir del nudo siguiente (probando humor>5); pero si está ausente, se abandona el descenso y volvemos hasta donde habíamos comenzado.

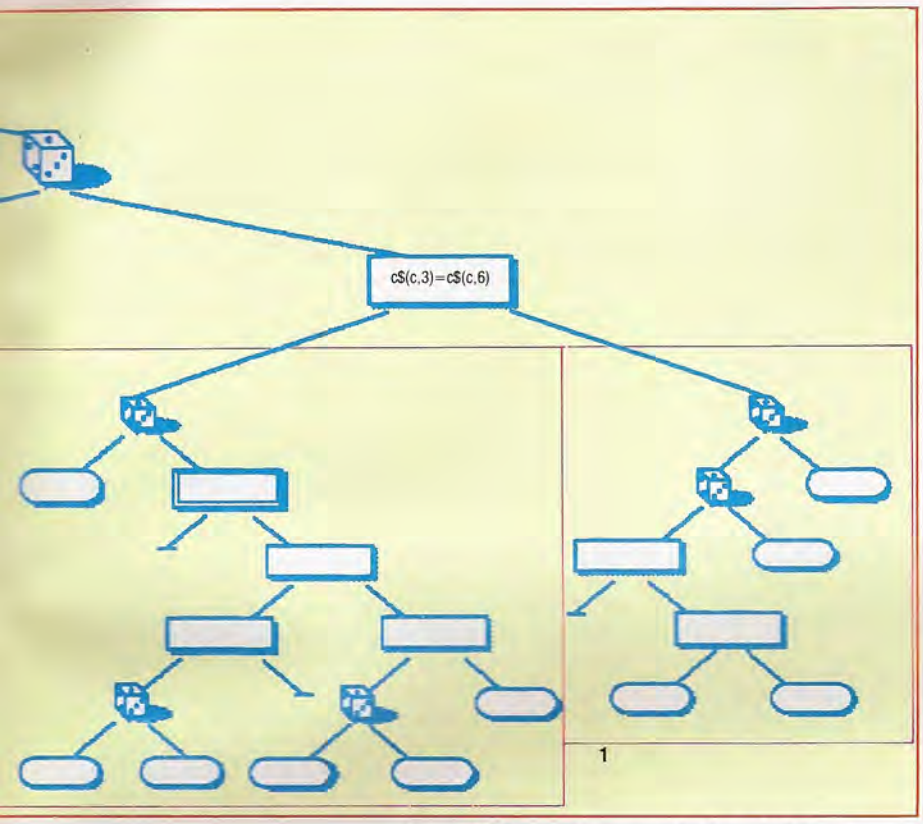


Caroline Clayton

Árbol de árboles
Las cuatro estructuras arborescentes que hemos explicado en este capítulo se pueden combinar entre sí para formar un árbol grande, como podemos ver arriba. El programa comprueba primero el valor de la expresión (VAL(c$(c,3))>0), cuya evaluación dará VERDADERO si el personaje en cuestión (c) tiene consigo un objeto, y FALSO en caso contrario. De dar FALSO, el flujo del control cae hacia abajo del árbol principal hasta la sección que se puede apreciar (con mayor detalle) en el diagrama **4**. De dar VERDADERO, el programa toma una decisión aleatoria respecto a si recorrer la estructura que se indica en el diagrama **3** (comer/beber objetos) o bien seguir adelante y comprobar los objetos «propios» mediante la evaluación de la expresión (c$(c,3)=c$(c,6)). El control pasa luego a una de las dos estructuras que se reflejan en los diagramas **1** y **2**

3





1

Es probable que el personaje deje el objeto; pero si no sucede esto, el programa comprueba la presencia del dueño. En este punto, si éste está cerca, el programa decide que el objeto se le debe entregar (siempre que el receptor esté con las manos vacías) o bien arrojar al propietario, si el personaje se halla de un humor particularmente malo. Si el dueño tiene las manos ocupadas y, en consecuencia, es incapaz de recibir el objeto, existe la posibilidad aleatoria de que el personaje diga: Lo siento, creo que he tomado tu bebida..., o algo por el estilo.

La única otra cosa que debemos observar es que uno de los nudos terminales implica la reducción de la fortaleza del personaje en dos puntos, dado que existen todas las posibilidades de que el personaje beba unos sorbos del vaso si es incapaz de devolvérselo a su propietario original; y, como todos sabemos, ¡cuando se mezclan bebidas los efectos son fatales! Sin embargo, aparte de esto, ninguno de los árboles que acabamos de describir hace ninguna provisión para comer o beber. En consecuencia, necesitamos una estructura que se ocupe de esto, que es la que se muestra en el tercer diagrama.

Este árbol tiene un código de color similar a los dos anteriores y, nuevamente, la fortaleza de un personaje se puede reducir en función de la bebida, si bien no en tanta cantidad, dado que en este punto no hay ninguna prueba para ver si se está muestreando la bebida correcta. Comer el bocadillo aumenta en dos la fortaleza del personaje, y establece el LCD de la forma correspondiente. Sin embargo, comer la empanada establece una bandera, reservada a los fines del argumento.

Y, ya que ha surgido el tema, bien podríamos hacerle conocer algunos secretos y revelarle la trama del argumento de Dog and Bucket. Sin lugar a dudas, se habrá preguntado el significado de las latas de alimento para gatos de la cocina, y habrá adivinado que las famosas empanadas de carne del Dog and Bucket en realidad están elaboradas con un alimento para animales de conocida (pero barata) marca.

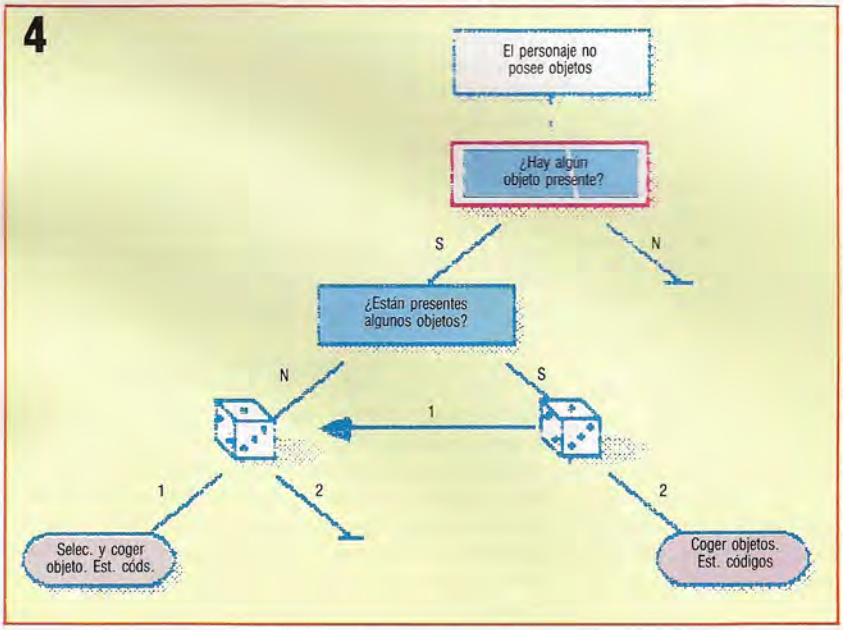
Veneno en la comida

Comer una empanada tendrá como resultado el fallecimiento intempestivo de uno o más de nuestros personajes, cuya identidad exacta diferirá cada vez que se ejecute el juego. En consecuencia, necesitamos establecer una bandera que indique quién, si hay alguno, sufrirá probablemente un envenenamiento alimentario. El hecho de que la muerte de un personaje sea considerada por los otros clientes del *pub* como un misterio o un asesinato, dependerá de otros factores, de los que nos ocuparemos en el próximo capítulo. Mientras tanto, sigamos examinando las estructuras de control de los objetos.

En caso de que un personaje tenga las manos vacías, utilizaremos el árbol esbozado en el cuarto diagrama, en el cual hay otro nudo que nos saca del árbol y nos lleva a una subrutina. Esta rutina comprueba si en la habitación hay otros objetos presentes, retornando si los hubiera y comprobando luego si el «propio» objeto del personaje está o no presente. Si lo está, hay una opción aleatoria de recogerlo; pero si no está, existe la posibilidad de recoger otro objeto. Tenga en cuenta que si los personajes no recogen sus «propios» objetos, existe otra posibilidad de que recojan otros, porque el nudo en cuestión salta a través del árbol, dando lugar, por tanto, a esta posibilidad. Esto significa que no es probable que los objetos se dejen en un sitio durante mucho tiempo, pero que los personajes *tenderán* a recoger sus propias bebidas.

Por último, se pueden unir estos cuatro árboles para conformar una estructura grande, y cada una de las condiciones que es necesario comprobar se puede expresar como una única expresión lógica. Puesto que algunos nudos querrán probar las mismas condiciones en distintos momentos, les asignaremos los valores de las sentencias condicionadas a una nueva matriz, utilizando elementos de ella (cuando fuera apropiado) para construir estructuras arborescentes.

4



Contra reloj

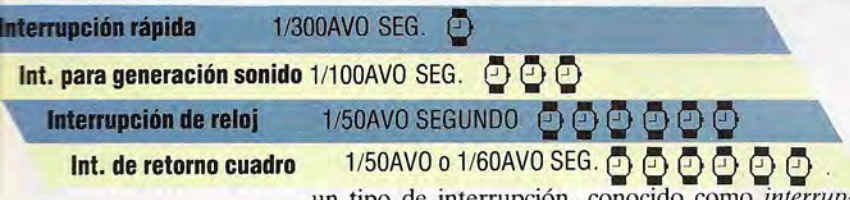
La temporización es una de las características más importantes de un OS. Veamos cómo se pueden tratar las interrupciones y los eventos, los dos métodos de temporización del Amstrad

La temporización del sistema gobierna cualquier operación que haya de efectuarse en un instante dado: por ejemplo, los colores parpadeantes de las letras en la pantalla deben ejecutarse durante un período de *retorno de cuadro* (el tiempo necesario para actualizar la visualización en video). Hay dos métodos distintos para implementar esta temporización en los ordenadores Amstrad; el primer método es controlado por el hardware y se denomina *interrupción,* y el segundo es controlado por el software y se denomina *evento*. Trataremos el empleo que hace el sistema de las interrupciones, como inicio de nuestro análisis.

Una interrupción es una señal generada fuera del microprocesador Z80, el cual suspende la ejecución del programa actual durante el tiempo en que se trata la señal.

Los ordenadores Amstrad operan con el Z80 programado para interrupciones del modo 1, es decir, se pasa el control a la rutina encargada de tratar las interrupciones, residente en la memoria en $0038. En teoría, una interrupción puede ser generada en cualquier momento y por cualquier dispositivo que reclame la atención del Z80, aunque en un Amstrad no ampliado sólo puede producirse un tipo de interrupción, conocido como *interrupción de temporizador* (*timer interrupt*).

Interrupciones del sistema
El procesador del Amstrad puede ser interrumpido por el sistema operativo de cuatro maneras. Estas interrupciones de temporizador se producen a intervalos regulares, para sincronizar las funciones regulares del sistema operativo, tales como el rastreo del teclado, la actualización del visualizador del video y el control de la transferencia de datos al chip de sonidos del Amstrad



La interrupción de temporizador se genera directamente desde el reloj del sistema (que funciona a 4 MHz) por la ULA, y sucede cada 300-ésima de segundo.

Toda temporización interna (tal como la que necesita el BASIC para sus instrucciones AFTER y EVERY) depende de esta interrupción.

La interrupción de temporizador se trata directamente por una parte del firmware denominado *kernel* (núcleo) que dialoga con el resto del firmware señalando diferentes niveles de interrupción, conforme se muestra en el esquema «Interrupciones del sistema». Veamos cada una de estas cuatro señales por separado:

- La *interrupción rápida* se produce cuando el núcleo recibe una interrupción de temporizador.
- La *interrupción para generación de sonido* se produce cada tres interrupciones de temporizador, y es empleada por el firmware para sincronizar la transferencia de los datos al chip de sonido. El usuario no puede detectar cuándo ocurre esta interrupción, salvo si se mantiene un contador de las interrupciones rápidas.
- La *interrupción de reloj* se produce después de seis interrupciones de temporizador y es empleada por el firmware para iniciar un rastreo de pantalla.
- La *interrupción de retorno de cuadro* se produce después de cada cinco o seis interrupciones de temporizador (según el tipo de visualización video) y se emplea para toda temporización dependiente de la pantalla.

El firmware mantiene un contador de interrupciones rápidas que el usuario puede obtener, y permite el establecimiento de bucles de temporización sin tener que suspender las actividades de un programa. El contador puede establecerse llamando KL__TIME__SET, que necesita que el valor para el temporizador sea pasado al par de registros HL. El valor del contador actual puede ser determinado en cualquier momento llamando KL__TIME__PLEASE, la cual da el valor contenido en HL. Un programa que sirva de ejemplo sobre el uso de estas dos rutinas es el que ofrecemos en estas páginas; el programa sondea el teclado durante un intervalo de tiempo especificado antes del retorno. Puede incorporarse a un programa que permita al usuario un determinado tiempo de respuesta antes de ejecutar una operación (como visualizar un mensaje de ayuda).

Se pueden emplear directamente las interrupciones de temporizador parcheando la entrada RST 7 en $0038. El código parcheado primero debe desalojar y copiar en otro sitio la entrada existente de tres bytes, y después cargar allí la nueva entrada (que normalmente será un salto a una dirección en la memoria). Esta nueva entrada deberá ser reubicable, de modo que si es desalojada a su vez por una segunda rutina, pueda funcionar correctamente aun en este caso.

La nueva rutina puede realizar cualquier tarea que se le pida (p. ej., actualizar el reloj) y deberá ejecutar entonces la entrada original saltando a la posición en que fue guardada. Este procedimiento asegura que todavía sean ejecutadas cualesquiera funciones de interrupción de temporizador, tales como sondear el teclado. La idea de insertar una *cuña* de código en una rutina de interrupción del sistema operativo ha sido ya analizada en capítulos anteriores de este curso de lenguaje máquina.

Interrupciones externas

Hasta ahora nuestro estudio ha supuesto que toda interrupción recibida por el núcleo era generada por la ULA, pero es posible que un periférico externo genere una interrupción. Por ejemplo, una interface serial puede generar una interrupción siempre que se haga obtenible un carácter; en este caso una rutina habrá de ser proporcionada para que lea el carácter desde la interface.

Para que esto sea posible, el núcleo debe ser capaz de distinguir entre una interrupción de temporizador y una interrupción externa. Normalmente, cuando se detecta una interrupción, el Z80 se deshabilita para recibir cualquier otra interrupción (y esto es para hacer que no sean violadas otras interrupciones de temporizador). El núcleo activa interrupciones desde dentro de la rutina que trata las interrupciones, y si la señal de interrupción al Z80 es todavía baja (indicando una petición de interrupción), entonces se genera una segunda interrupción. Dado que no se duplica una interrupción de temporizador a este punto, esto puede servir para detectar una interrupción externa siempre y cuando la señal de la interrupción permanezca baja hasta ser atendida. Cuando se detecta una interrupción externa, el control pasa a $003B, por lo que esta posición debe ser acoplada con un salto a la rutina de servicio de interrupciones, la cual atiende al dispositivo que la solicita.

El hardware de sistema acepta tanto *interrupciones no enmascarables* (NMI: *non-maskable interrupts*) como interrupciones estándar. Las NMI sólo difieren en que no pueden ser deshabilitadas en cualquier momento, y su empleo no es recomendable porque puede afectar a actividades de sistema con peticiones limitadoras de temporización. En particular, las secciones firmware de tratamiento de disco y cassette (y partes de las secciones de tratamiento de sonido) continúan operando con interrupciones deshabilitadas, pero quedarían seriamente afectadas si se usaran NMI.

Eventos software

Al nivel más sencillo, los *eventos* pueden ser descritos como el equivalente en software de las interrupciones: las tareas particulares son realizadas por separado respecto de cualquier programa principal por medio de un contador de eventos a ejecutar (cuáles y cuándo) que guarda el firmware.

Aunque los eventos pueden ser empleados para tratar las interrupciones externas, son de gran utilidad para programas complicados que incluyan una simulación, o que requieren una operación independiente del programa principal que aparece en primer plano. Por ejemplo, el sintetizador de voz SSA-1 de Amstrad utiliza un evento para permitir que se genere la voz de manera transparente desde dentro de un programa en BASIC.

Los eventos se presentan al sistema operativo por medio de un *bloque de eventos*. Éste consiste simplemente en un bloque de siete bytes contiguos situados en algún lugar dentro de los 32 K centrales de la memoria. Un bloque de eventos tiene la estructuración que muestra nuestro diagrama; dentro del bloque existen tres campos, esenciales a todo evento, que caracterizan el tipo de evento y dónde ha de ser procesado.

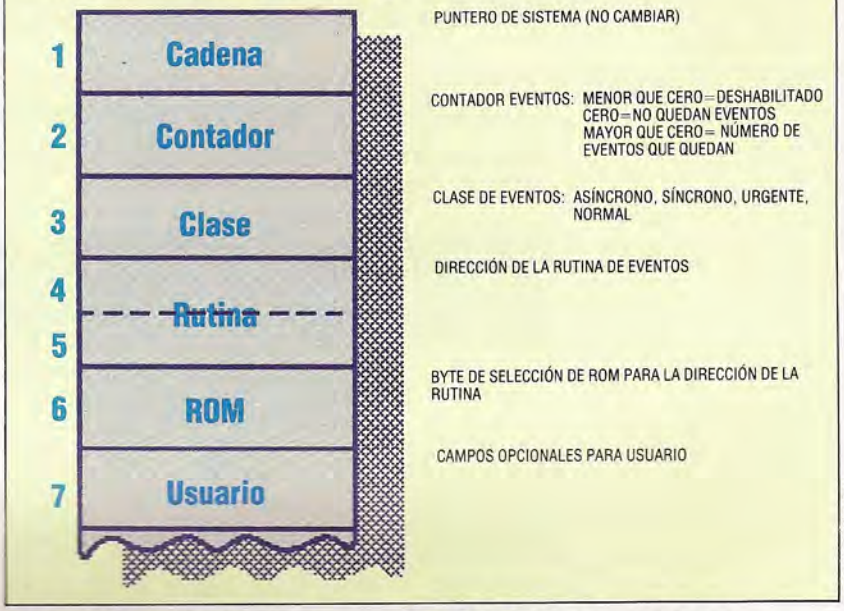
Dentro del bloque de eventos está el *contador de eventos*, que cuenta las veces que ha ocurrido un evento. El proceso de incrementación de este contador es conocido como *puntapié* (*kicking*). Cada vez que se lanza un evento el contador es incrementado en una unidad; una vez procesado el evento con la llamada a la rutina de eventos, el contador es disminuido en una unidad. Un modo de neutralizar un evento es darle al contador un valor negativo.

La *clase de evento* indica si éste es *síncrono* o *asíncrono* con relación al programa principal. Los eventos asíncronos son procesados independientemente del programa principal y se destinan a aplicaciones que requieren casi una acción inmediata.

Los eventos síncronos están situados en una cola cuando se producen, de tal modo que puedan ser procesados por el programa principal cuando sea necesario. Los eventos síncronos son procesados conforme al orden en que ocurren. Además, el evento puede ser también *urgente* o *normal*.

En el capítulo próximo consideraremos las diferencias entre las varias clases de eventos y el modo que tiene el sistema operativo de tratarlos. Puede que a usted le parezca que el empleo de eventos presenta un buen número de complicaciones innecesarias, pero el hecho es que la estructura de eventos en el Amstrad es tan poderosa como flexible, y tiene la ventaja adicional para un programador en código máquina de que es enteramente compatible.

Estructura del bloque de eventos



Caroline Clayton

El evento principal
Los eventos son el equivalente en software de las interrupciones y el bloque de eventos es empleado por el sistema operativo para llevar cuenta de todos los eventos establecidos. Siete bytes consecutivos sirven para retener la dirección de la rutina de eventos a la que se refiere el bloque, el número de eventos (si los hay) que quedan por procesar y la clase de evento

```
; Ejemplo de empleo de rutinas de temporizador del núcleo
;
;
SET.TIME:       EQU     £BD10       ;lee el contador del tiempo del reloj rapido
GET.TIME:       EQU     £BD0D       ;establece el contador del reloj rapido
;
READ.CHAR:      EQU     £BB09       ;busca un caracter del teclado
PAUSE:          EQU     300         ;pausa durante 300 impulsos rapidos
;
                LD      HL,0        ;pone a cero el tiempo
                LD      DE,0
                CALL    SET.TIME
                LD      BC, PAUSE   ;establece periodo de espera
;
;espera que un caracter se haga obtenible
;del teclado
;
WAIT:           CALL    READ.CHAR   ;esta listo el caracter?
                RET     C           ;si es asi, retrocede con acarreo
;
                CALL    GET.TIME    ;halla el tiempo
                SBC     HL,BC       ;un segundo?
                                    (o más)
;
                JR      C,WAIT      ;si no es asi, retirada
;
;si llegamos hasta aqui, la rutina ha agotado el tiempo
;vuelta con arrastre falso para señalar tiempo agotado
;
                RET
;
```

Datos básicos (X)

Con este fragmento finalizamos el detallado análisis del mapa de memoria del Commodore 64 que hemos venido realizando

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
ISTOP	0328-0329	808-809	Vector rutina núcleo STOP
IGETIN	032A-032B	810-811	Vector rutina núcleo GETIN
ICLALL	032C-032D	812-813	Vector rutina núcleo CLALL
USRCMD	032E-032F	814-815	Vector definido por el usuario
ILOAD	0330-0331	816-817	Vector rutina núcleo LOAD
ISAVE	0332-0333	818-819	Vector rutina núcleo SAVE
	0334-033B	820-827	No se usa
TBUFFR	033C-03FB	828-1019	Buffer Entrada/Salida cinta
	03FC-03FF	1020-1023	No se usa
VICSCN	0400-07FF	1024-2047	Área memoria pantalla de 1024 bytes
	0400-07E7	1024-2023	Matriz video: 25 líneas × 40 columnas
	07F8-07FF	2040-2047	Punteros datos sprite
	0800-9FFF	2048-40959	Espacio normal para programas BASIC
	8000-9FFF	32768-40959	ROM cartucho VSP: 8192 bytes
	A000-BFFF	40960-49151	ROM BASIC: 8192 bytes (u 8 K RAM)
	C000-CFFF	49152-53247	RAM: 4096 bytes
	D000-DFFF	53248-57343	RAM color y dispositivos Entrada/Salida o ROM generador carácter o RAM: 4096 bytes
	E000-FFFF	57344-65535	ROM núcleo: 8192 bytes (u 8 K RAM)

Página impresa

Ya es posible diseñar una página como ésta en la pantalla de un micro. Estudiemos el proceso

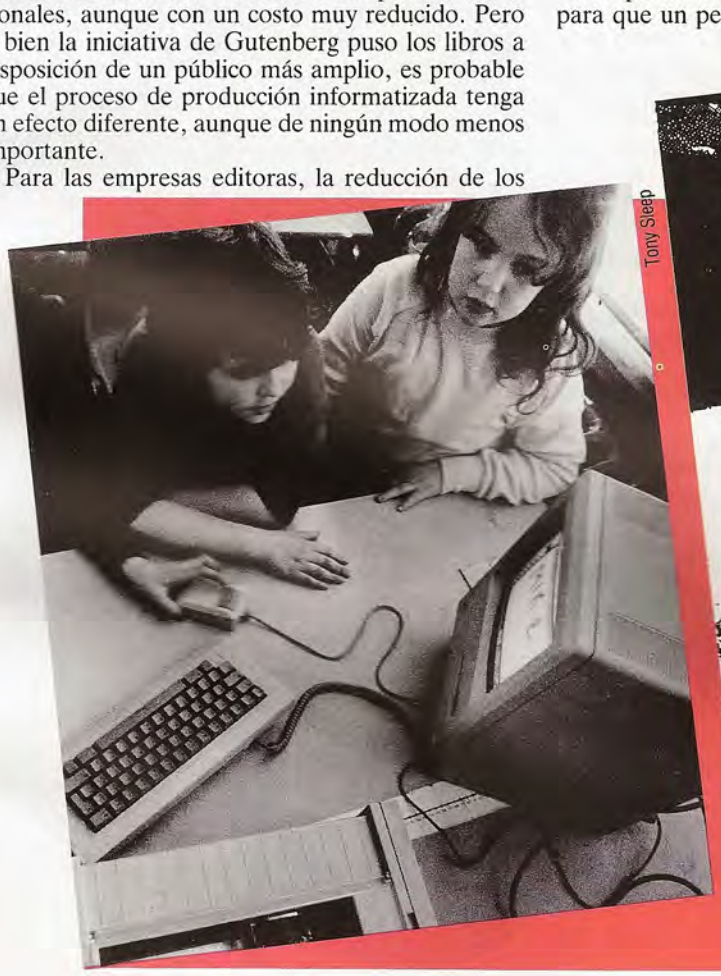
La invención de la imprenta fue un acontecimiento de suma importancia, porque gracias a ella la palabra escrita se puso al alcance de una cantidad mucho mayor de personas. Antes de ella, por supuesto, los únicos libros disponibles se producían artesanalmente, ya sea escritos a mano, o bien compuestos con bloques de madera en los que se había grabado cada una de las letras. Esto significaba que existían muy pocas copias de cualquier volumen y que el costo de su producción era muy elevado. El procedimiento de impresión con caracteres móviles inventado hacia 1440 por el impresor alemán Johannes Gutenberg redujo drásticamente los costos de producción de cada unidad, de modo que los ejemplares de cada obra fueron a partir de entonces mucho más asequibles a la población.

No es razonable equiparar el significado de los últimos avances en el campo editorial con la invención de la imprenta, pero no obstante existen algunas analogías. Básicamente, los nuevos métodos de producción basados en la tecnología del ordenador pueden producir material de calidad similar al producido mediante los métodos de impresión tradicionales, aunque con un costo muy reducido. Pero si bien la iniciativa de Gutenberg puso los libros a disposición de un público más amplio, es probable que el proceso de producción informatizada tenga un efecto diferente, aunque de ningún modo menos importante.

Para las empresas editoras, la reducción de los costos de producción significaba tener que vender menos ejemplares para recuperar los gastos de su producción. Esto podría llevar a un aumento de la cantidad de publicaciones especializadas dirigidas a un público pequeño, antes inviables desde el punto de vista económico.

En el capítulo anterior nos referimos al proceso general que supone la producción de una revista y vimos algunos de los cambios que están teniendo lugar actualmente en ese campo. Llevando la idea de un sistema de producción modernizado un paso más hacia adelante, desviaremos nuestra atención ahora a un único microordenador independiente que se puede utilizar como procesador de textos, diseñador de maquetas y controlador de una impresora de gran calidad.

El Apple Macintosh, equipado con 512 Kbytes de memoria, ha dado lugar al desarrollo de varios paquetes sofisticados de software para usar con él, teniendo en mente precisamente esta finalidad. Entre éstos se incluye un paquete denominado *PageMaker*; si se le añade a este paquete una impresora de primera categoría, se crean las condiciones para que un pequeño grupo de personas produzca



Tony Sleep

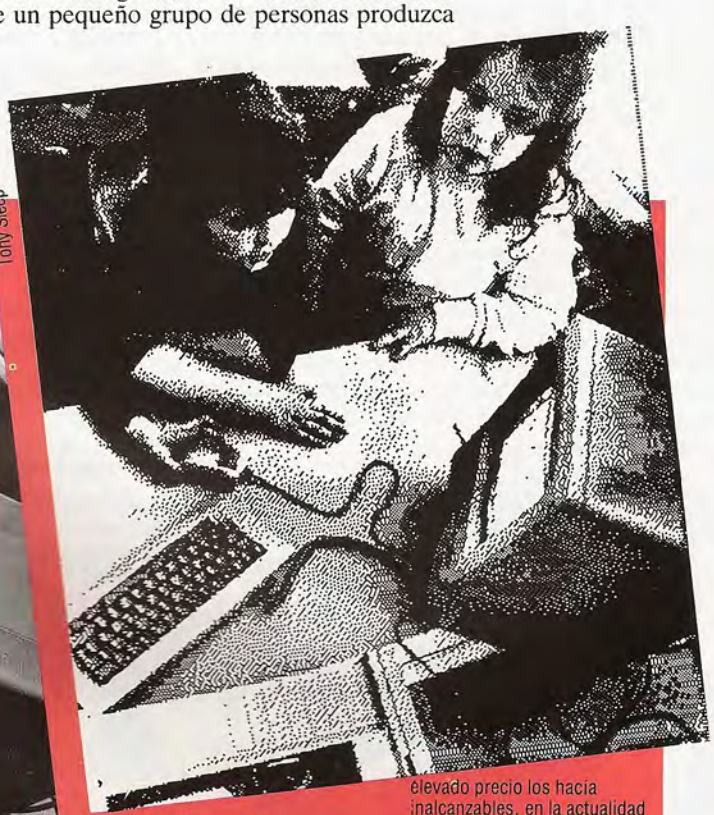


Imagen digitalizada
Los digitalizadores permiten que el usuario almacene y luego reproduzca imágenes a utilizar en el diseño de páginas. Aunque antes su elevado precio los hacía inalcanzables, en la actualidad ya se venden a un precio razonable. La imagen digitalizada de la fotografía que vemos aquí se produjo aplicando el sistema Thunderscan

publicaciones de una calidad casi tipográfica y significativamente más rápida y económicamente que por los métodos tradicionales.

Cómo trabaja el «PageMaker»

El Macintosh, con su enfoque de ventanas, iconos y ratón, constituye un entorno ideal para aplicaciones como *PageMaker*. Cuando se carga el *PageMaker* se presentan los familiares menús y ventanas de pantalla, y la pantalla del Mac queda preparada como un escritorio completo con su caja de herramientas para que trabaje el maquetista.

Normalmente el maquetista comienza a trabajar con una hoja pautada que es la página estándar de una publicación, e incluye líneas de posición que sirven de ayuda para colocar las columnas de texto, los titulares, la numeración de las páginas, etc. Luego se emplean copias del papel pautado para componer cada página de la revista o periódico, cuidando que la misma tenga una apariencia uniforme.

Tradicionalmente, cuando se lanza una publicación, el maquetista traza a mano la hoja pautada, que después se adopta como estándar hasta que se decide un nuevo estilo.

Este tradicional método de diseño se reproduce en el *PageMaker*, que permite definir *páginas maestras*. Para éstas se selecciona el tamaño de la página (p. ej., A4) y la orientación (vertical o apaisada), se posicionan las guías de columnas, se define la anchura de los márgenes y se marcan las posiciones para los números de página, logotipos y titulares. Una vez preparada la página maestra, en el futuro podrá recuperarse en cualquier momento para que se constituya en la hoja pautada del maquetista.

En este punto se pueden maquetar las páginas individuales de la publicación. Un maquetista utilizará un escalpelo para cortar y posicionar texto e imágenes en la hoja pautada, mientras que el *PageMaker* permite cargar texto y gráficos desde disco. Y, en virtud de su compatibilidad con otros paquetes para el Mac, se podría escribir un artículo con el procesador de textos *MacWrite*, cargarlo directamente en *PageMaker* y posicionarlo en el tablero. Los gráficos diseñados con el *MacPaint* y el *MacDraw*, así como numerosas ilustraciones de esta publicación, también se pueden cargar mediante el *PageMaker* y posicionar en la página.

Tras haber cargado los diversos componentes que compondrán la página, se pueden utilizar los diversos dispositivos que contiene la caja de herramientas de la esquina de la pantalla.

Estas herramientas, por ejemplo, permiten añadir o corregir textos en pantalla seleccionando el icono A de la caja de herramientas o, seleccionando el icono «herramienta de siega», reducir, ampliar o recortar los gráficos para que quepan en el espacio disponible.

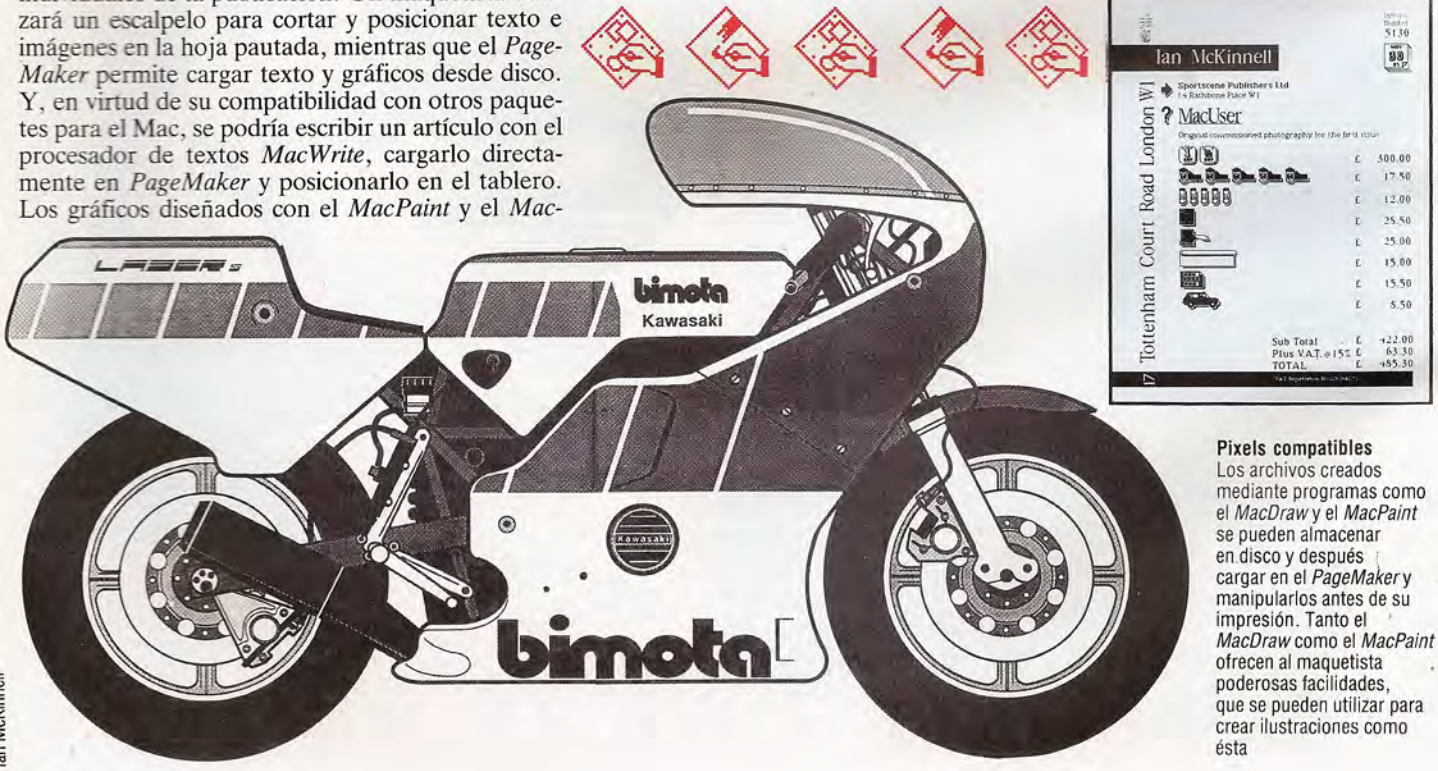
Utilizando el ratón se pueden recoger texto y gráficos y desplazarlos a través de la hoja pautada a voluntad, e incluso «dejar» temporalmente sobre la mesa escritorio mientras se lleva a cabo cualquier otra tarea.

La caja de herramientas del *PageMaker* también incluye facilidades que permiten añadir toques especiales a la página, como dibujar ribetes alrededor de secciones específicas.

Incluye iconos que controlan el trazado de líneas, círculos, elipses y recuadros con o sin esquinas redondeadas.

También se puede seleccionar el espesor de las líneas extrayéndolo del menú adecuado. Estas formas, una vez dibujadas, se pueden rellenar en blanco o negro, varias tonalidades de gris o con formatos. A pesar de ser una herramienta increíblemente útil para los maquetistas profesionales, el *PageMaker* es suficientemente simple como para que el maquetista que no tiene experiencia lo utilice con todo éxito.

El costo de un Macintosh, una LaserWriter y el software para procesar la página necesarios para crear un sistema de producción con un micro es relativamente elevado, pero la introducción de máquinas más económicas basadas en el WIMP, como el Atari 520ST (véase p. 2029), que incorpora el



Iconos de facturas
Esta factura se diseñó y se imprimió usando una copia generada directamente por un Apple Macintosh. El diseño de iconos suele ser una configuración importante de publicaciones tales como *Mi Computer*, ya que facilitan la identificación de las diferentes secciones. De modo similar, los logos y símbolos de productos de las empresas con frecuencia se utilizan en facturas y membretes

Pixels compatibles
Los archivos creados mediante programas como el *MacDraw* y el *MacPaint* se pueden almacenar en disco y después cargar en el *PageMaker* y manipularlos antes de su impresión. Tanto el *MacDraw* como el *MacPaint* ofrecen al maquetista poderosas facilidades, que se pueden utilizar para crear ilustraciones como ésta

entorno GEM (administrador del entorno gráfico), probablemente dará como resultado importantes reducciones del precio de tales sistemas. Aun con los precios actuales, y teniendo en cuenta los costos de trabajo que se ahorran al producir una publicación utilizando sistemas informatizados de diseño de páginas, supone un significativo recorte de los costos de producción convencionales.

Rebajando los costos

¿Cuánto tiempo y dinero se podría ahorrar utilizando un sistema de procesamiento electrónico de páginas como el del Macintosh con la *LaserWriter* y el *PageMaker*? Imaginemos que se está aplicando este sistema para producir el boletín informativo de una empresa, que consta de 16 páginas de calidad profesional. Suponiendo que la publicación la produzca un pequeño equipo de maquetistas y redactores contratado por la empresa, y que todo el trabajo, exceptuando el tipográfico, se realice «en casa», el siguiente detalle representa los costos típicos de producción:

Tarea	Precio	Costo ptas
Preparación tipográfica	5 horas a 800 ptas/h	4 000
Composición y correcciones		20 000
Coordinación	5 horas a 800 ptas/h	4 000
Corrección de galeradas	8 horas a 400 ptas/h	3 200
Maqueta y diseño	6 horas a 800 ptas/h	4 800
Maqueta final	16 horas a 800 ptas/h	12 800
Costos totales		**48 800**

La publicación equivalente podría diseñarla, con el *PageMaker*, una sola persona en unas cinco horas a unas 2 500 ptas. la hora. Por consiguiente, las 16 páginas se podrían producir por unas 12 500 ptas. Pero también hay otras ventajas. El método convencional es, por necesidad, secuencial, y las diversas tareas, que habrían de planificarse cuidadosamente, llevarían alrededor de dos semanas. Utilizando el *PageMaker*, desaparecerían gran parte de los problemas organizativos de la producción, porque las correcciones, supresiones, añadidos y pies de ilustraciones se podrían introducir en pantalla durante el proceso de maquetación.

Fijando el paso

Las modernas técnicas de impresión utilizan placas fotográficas para transferir una imagen al papel. Por lo tanto, las páginas preparadas para prensa han de estar «listas para la cámara» y, por cuanto concierne al texto, ello supone la producción de una imagen perfecta del texto a imprimir, al contrario que en el tipo en relieve que puede producir una linotipia de caracteres fundidos.

Las máquinas de fotocomposición hacen exactamente esto, aceptando texto como entrada desde un teclado o un disco y produciéndolo en forma de «galeradas», largas tiras de papel fotográfico en el cual se ha impreso el texto perfectamente (o así cabría esperar), listo para confeccionar la maqueta. Las primeras máquinas de fotocomposición utilizaban una plantilla giratoria en forma de disco que retenía una copia de todos los caracteres de un determinado tipo. El papel fotosensible pasaba sobre una cara del disco y, cuando la letra deseada giraba colocándose en posición, se encendía una luz, transfiriendo el carácter al papel. Aunque primitivos, estos sistemas alcanzaron considerables niveles de sofisticación y podían generar texto con moderada rapidez.

En la fotografía vemos un moderno sistema de fotocomposición basado en láser y producido por una Linotype-Paul, ejemplo típico de los sistemas controlados por ordenador, capaces de generar textos a 368 000 caracteres por hora.

El sistema ilustrado posee tres componentes fundamentales. A la izquierda se halla la Linotronic 300, una máquina tipográfica láser de formato ancho y gran calidad, con una resolución de hasta un millón de pixels por cm^2. Los tipos se pueden ampliar o condensar, girar e invertir como se desee, y puede haber hasta 32 tipos de letra por línea. El terminal del centro permite entrar texto de forma manual, desde disco flexible o incluso a través de un modem. Esto último es útil para periodistas, quienes con frecuencia necesitan enviar texto al sistema mientras cubren una noticia en el mismo lugar en que se produce.

La unidad de la derecha, una Linotype-Paul Typeview 300, visualiza el texto en el estilo, tamaño y posición que ocupará en el montaje final. Estos sistemas no sólo permiten producir copias listas para la cámara, sino que también se pueden enlazar con sistemas en red mayores, a los que pueden acceder todo tipo de editoriales. Linotype ofrece un sistema capaz de manipular hasta 40 terminales interactivos a la vez, junto con un inmenso almacenamiento en línea y acceso a periféricos compartido. Por tanto, un sistema único, utilizado por un periódico, podría aceptar entradas de los reporteros desde su puesto de trabajo en el periódico o desde fuera, recuperar material gráfico almacenado, pasarlo a los redactores, combinarlo con trabajos artísticos y luego presentar el material al departamento de maquetación antes de producir una placa para su procesamiento e impresión

Máquina versátil

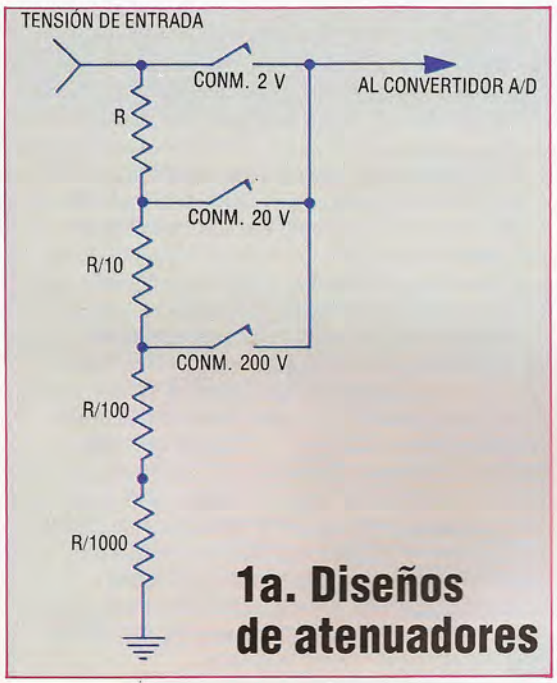
En este penúltimo capítulo analizamos los accesorios que podrían aumentar la versatilidad del tester

Tal como está, el tester digital puede medir tensiones de entre 0,0000 V y 1,9999 V, motivo por el cual decimos que posee una sensibilidad básica de 2 V. Atenuando la señal a medir, teóricamente sería posible medir tensiones de hasta 19 999 V (20 kV). Estas tensiones tan altas pueden ser sumamente peligrosas, de modo que hemos limitado la escala máxima a 199,99 V.

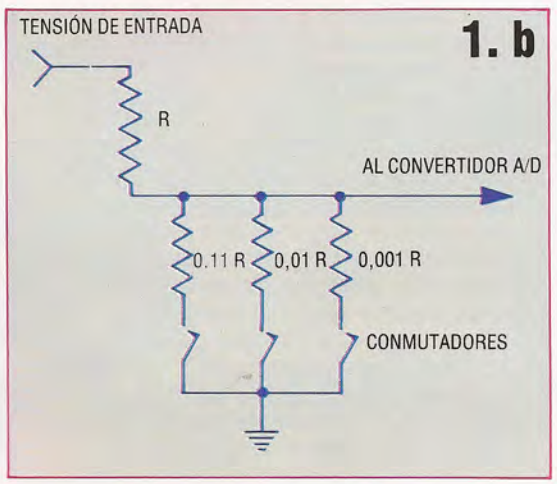
Quizá haya notado, a partir del diagrama del capítulo anterior, que había tres cables sin conectar, procedentes de tres de los LED. Éstos se conectan a los puntos decimales que aparecen a la derecha del dígito de los LED y se utilizan para encender el punto decimal cuando así se requiere. Un conmutador de tres posiciones y de un polo, «acoplado» al conmutador de atenuador de entrada, asegura que esté encendido el punto decimal adecuado para la sensibilidad seleccionada. Consideremos primero el circuito del atenuador de entrada.

Básicamente hay dos formas de hacer un atenuador de entrada. Ambas implican el empleo de un circuito divisor de potencial que distribuye la tensión de entrada en varias resistencias, derivando una fracción de la tensión de la entrada para la medición. En el diagrama 1 se muestran estos dos tipos de atenuadores. El primer tipo ofrece la ventaja de la simplicidad, además de ser fácilmente adaptable a la medición de corriente y resistencia. Esto no es así en el caso del segundo atenuador. En un circuito de escala automática (en el cual la atenuación de entrada se conmuta automáticamente), el segundo tipo ofrece la ventaja de poder utilizar conmutadores analógicos de estado, mientras que el primero sólo puede utilizar conmutadores mecánicos. En nuestro caso utilizaremos el segundo

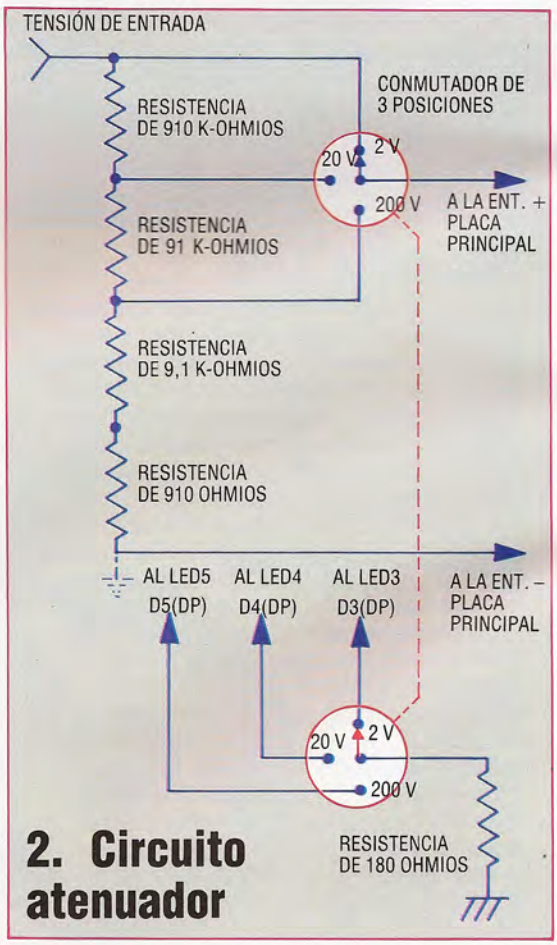
Diseños de atenuadores
Para aumentar la escala de nuestro voltímetro básico necesitamos diseñar un atenuador de entrada que reduzca la escala de tensión a un nivel aceptable para el circuito de conversión A/D principal. Ambos utilizan varias resistencias que se pueden conmutar en el circuito para derivar una proporción del potencial de entrada. Nosotros utilizaremos el primero de los dos diseños que vemos aquí, porque el segundo diseño tiene el inconveniente de requerir un circuito de protección de entrada especial (si bien puede incorporar conmutadores analógicos de estado sólido)



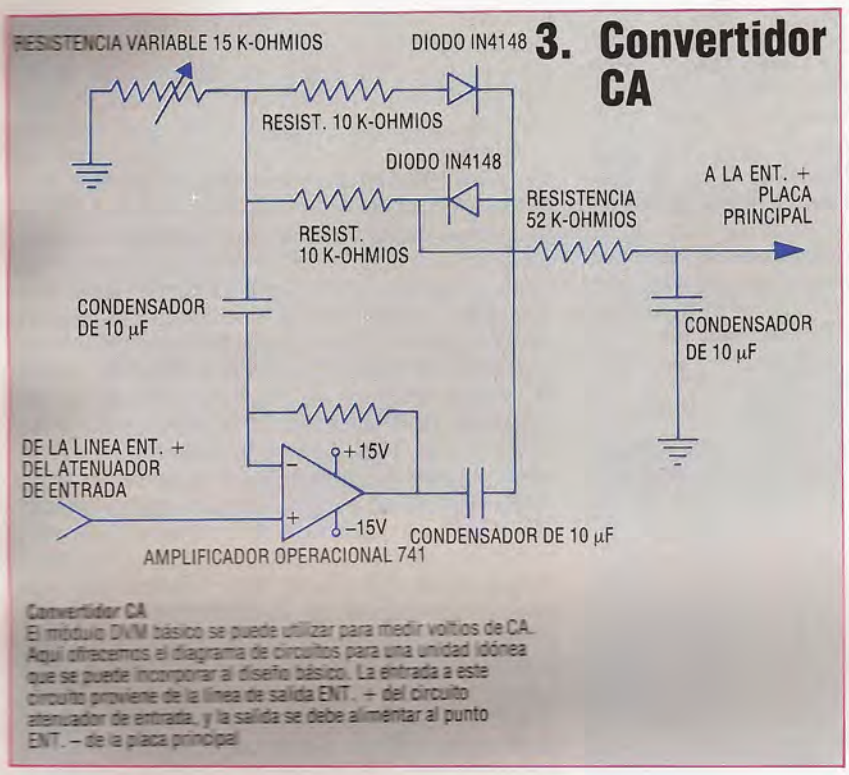
1a. Diseños de atenuadores



Circuito atenuador
Para construir el atenuador de entrada emplee un pequeño trozo de placa matriz de 0,1 pulgadas. Las resistencias han de ser del 1 % de tolerancia sobre los valores nominales. Monte las resistencias a lo largo de una línea de la placa y derive todos los extremos de las resistencias de 910 K-ohmios y 91 K-ohmios a un conmutador de tres circuitos y tres polos. Las salidas negativa y positiva se deben conectar a la placa principal mediante unos trozos cortos de hilo aislado. Los tres terminales restantes del conmutador de dos circuitos se deben conectar a las patillas del punto decimal de los LED 3, 4 y 5 del visualizador. Estas conexiones permiten reposicionar automáticamente el punto decimal de la visualización cuando se selecciona una nueva escala



2. Circuito atenuador



3. Convertidor CA

Convertidor CA
El módulo DVM básico se puede utilizar para medir voltios de CA. Aquí ofrecemos el diagrama de circuitos para una unidad idónea que se puede incorporar al diseño básico. La entrada a este circuito proviene de la línea de salida ENT. + del circuito atenuador de entrada, y la salida se debe alimentar al punto ENT. – de la placa principal

tipo, dada la sencillez con que se puede adaptar para realizar mediciones.

El verdadero circuito (diagrama 2), junto con la conmutación del punto decimal, se puede construir fácilmente en un pequeño trozo de placa matriz, sustituyendo y conectando los componentes especificados en la lista de componentes por los del diagrama de circuitos. Las resistencias especificadas son de tipo de película metálica, de un 1 % de tolerancia.

El trazado no es crítico, pero es aconsejable que todos los trozos de hilo sean lo más cortos posible, puesto que la impedancia de entrada del chip A/D es muy alta y los hilos largos actúan como una antena para señales espurias.

Este circuito atenuador es de diseño simple y utiliza valores de resistencias fáciles de conseguir. Sin embargo, posee la desventaja de presentar una carga relativamente baja para el circuito que se esté probando (poco más de 1 M-ohmio); pero ésta no es una limitación grave: si se está midiendo una señal de 5 V, representa una corriente de carga de alrededor de 5 µA.

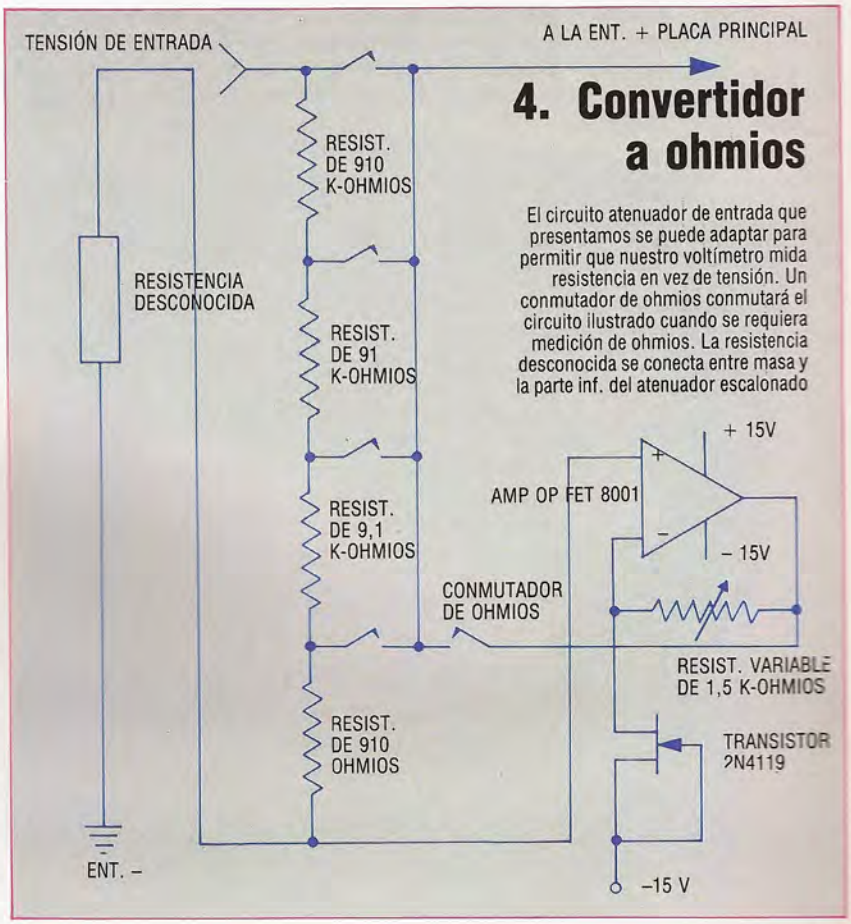
El circuito que hemos presentado puede medir voltios de CC en tres escalas: 2 V, 20 V y 200 V. Modificaciones o adiciones relativamente simples al circuito básico permitirán medir muchas otras unidades, incluyendo ohmios, tensiones CC y temperatura.

Ofrecemos aquí algunas posibles adiciones al circuito básico, pero solamente en forma de diagrama de circuitos.

Para medir voltios CA usando un voltímetro CC (como nuestro DVM), usted sólo necesita rectificar la señal CA a CC y medirla. Lamentablemente, no servirá un rectificador de diodo simple; lo que se requiere es uno de los llamados rectificadores de precisión. El diagrama 3 muestra un posible circuito. Se basa en el amplificador operativo integrado 741, muy conocido y de precio muy bajo. En este caso, la única desventaja es la necesidad de una fuente de alimentación de ±15 V, pero la misma se podría derivar de la salida CC de 12 V del transformador de la red; las exigencias de corriente del 741 son muy bajas.

Medir ohmios en un voltímetro analógico convencional es fácil, dado que la caída de tensión en una resistencia que se está midiendo es proporcional al valor de esta última. Sin embargo, eso no es tan fácil con un voltímetro digital y, en consecuencia, se requiere un circuito convertidor a ohmios. El convertidor de ohmios (ver diagrama 4) funciona aplicándole una tensión constante al atenuador de entrada escalonado, que genera una corriente constante en la resistencia que se está midiendo. Puesto que una corriente constante a través de una resistencia desconocida genera en ella una tensión proporcional a su valor, el DVM podrá leer directamente esta caída de tensión. Nuevamente se requerirá un amplificador operacional, y habrá de ser uno que tenga un tipo de impedancia de entrada muy elevada.

Mediante la aplicación de la salida del convertidor de ohmios al atenuador escalonado se pueden medir cuatro escalas de ohmios: 2 K-ohmios, 20 K-ohmios, 200 K-ohmios y 2 M-ohmios. (Por esta razón se utilizaron dos resistencias separadas en la parte inferior del divisor escalonado, en lugar de las resistencias individuales de 10 K-ohmios que hubieran sido adecuadas si sólo se hubiesen requerido tres escalas de tensión.) Como es natural, se requerirá un conmutador de atenuación de cuatro circuitos en lugar del interruptor de tres circuitos especificado.



4. Convertidor a ohmios

El circuito atenuador de entrada que presentamos se puede adaptar para permitir que nuestro voltímetro mida resistencia en vez de tensión. Un conmutador de ohmios conmutará el circuito ilustrado cuando se requiera medición de ohmios. La resistencia desconocida se conecta entre masa y la parte inf. del atenuador escalonado

Lenguajes clásicos

Iniciamos una serie dedicada a los lenguajes pioneros de la informática. En primer lugar, revisaremos someramente su historia y desarrollo

Para la mayoría de nosotros, la idea de programar un ordenador constituye un hecho corriente. Aunque todavía no podamos hacerlo muy bien, al menos sabemos de qué se trata. Las cosas eran muy diferentes, sin embargo, a finales de los años cuarenta y principios de los cincuenta, cuando se desarrolló por primera vez el concepto de un lenguaje de programación. Este primer período produjo algunos lenguajes que aún hoy se utilizan.

Esta serie de artículos tiene por objeto considerar la evolución de estos lenguajes hasta alcanzar su forma actual, así como rastrear algunas de las influencias que tuvieron en lenguajes aparecidos posteriormente y descubrir su aportación a los microordenadores modernos. Hablaremos con mayor profundidad de dos de estos lenguajes, el COBOL y el FORTRAN, que no sólo se siguen utilizando, sino que la suma de las líneas de código que se están escribiendo supera la de la totalidad de todos los otros lenguajes juntos. Asimismo, nos detendremos en el ALGOL, que si bien continúa siendo un lenguaje importante, ha sido superado por el PASCAL. En primer lugar, consideremos los primeros días de la informática y veamos cómo se desarrolló la idea de un «lenguaje» para ordenadores.

Si pasamos por alto las etapas más tempranas, en las cuales «programar» significaba recablear el hardware para cada tarea diferente, los primeros programas auténticos eran completamente numéricos y por lo general estaban escritos en octal (base ocho), como forma taquigráfica conveniente para el binario. Estos programas se entraban laboriosamente mediante interruptores situados en un panel frontal o, más tarde, empleando cinta de papel o fichas perforadas. Cada programa nuevo se escribía completamente de la nada en el nivel más inferior posible y a menudo los programadores se encontraban a sí mismos escribiendo las mismas rutinas una y otra vez. De allí surgió el concepto de una biblioteca de subrutinas, si bien el vocablo *subrutina* aún no se empleaba. Estas «bibliotecas» se conservaban en cuadernos de apuntes y se copiaban en cada nuevo programa cuando así se requería, y cada programador contaba con sus propias bibliotecas, compartiéndolas sólo ocasionalmente.

La construcción del sistema EDSAC en Manchester (Gran Bretaña), en 1951, supuso un gran paso adelante, y el trabajo de Wheeler, Wilkes y Gill se plasmó en un conjunto coherente de subrutinas generales para la máquina. Ahora que todos utilizaban las mismas subrutinas, los programas fueron más fáciles de escribir y comenzaron a traslucir semejanzas estructurales con sus equivalentes modernos. Unos bloques de código realizaban la tarea determinada con llamadas a las subrutinas para llevar a cabo las funciones estándares tales como entrada, salida y cálculos numéricos, que eran comunes a la mayoría de los programas.

Cuando aumentó la capacidad de memoria de las máquinas, se hizo posible almacenar las bibliotecas de subrutinas internamente, o al menos en línea, y a partir de allí sólo hubo un corto paso hacia el desarrollo de un código que permitiera al programador especificar las subrutinas requeridas utilizando caracteres alfabéticos y una notación matemática. Un programa observaría cada línea de este código, determinaría las subrutinas requeridas, las llamaría y luego regresaría a la siguiente línea. Ésta es la forma en que trabaja un intérprete de BASIC. Un ejemplo de esto fue el *Short code* (Código corto) que produjo John Maunchly en 1952 en el ordenador BINAC y luego en el UNIVAC.

Otro refinamiento de esta primera técnica tomaba cada subrutina cuando así se requería, pero en vez de ejecutarla directamente, primero la copiaba en un dispositivo de salida, acabando, por tanto, con un programa completo ejecutable. Fue durante esta etapa cuando se desarrollaron los primeros compiladores, con el término *compilar* aludiendo a la forma en que se conformaba el programa a partir de una cantidad de componentes dispuestos por un orden lógico, de la misma forma en que se compilaría un conjunto de ensayos.

Fechas clave

Cronología simplificada del desarrollo de los primeros lenguajes de programación:

1951	Se completa en Manchester el EDSAC, primer ordenador digital con programas almacenados
1952	*Short code* de John Maunchly
	Speedcoding de Laning y Zierly
1953	Compilador A2. Primeras especificaciones para el FORTRAN
1954	
1955	Primeros compiladores auténticos para FORTRAN y FLOW-MATIC
1956	Lanzamiento generalizado de compiladores FORTRAN y FLOW-MATIC. Especificaciones para el FORTRAN II
1957	Especificaciones y lanzamiento del primer compilador de ALGOL
1958	Lanzamiento del FORTRAN II
1959	Primeras especificaciones para el COBOL. Primer compilador de LISP
1960	Lanzamiento del primer compilador de COBOL. Lanzamiento de la versión revisada ALGOL 60

Los primeros compiladores

Uno de los primeros compiladores que tuvieron éxito fue el A-2, desarrollado en Remington Rand en 1955 por un equipo dirigido por Grace Hopper.

El mismo utilizaba el concepto de *tres direcciones*, en el que cada operación tenía un nombre mnemotécnico seguido por tres direcciones: dos para los datos fuente y una para el destino. En algunos sentidos, era muy similar a un ensamblador, con la excepción de que las instrucciones no cabían en la arquitectura de ninguna máquina específica. Posteriormente este lenguaje se convirtió en ARITHMATIC, al cual siguió un lenguaje similar de Remington Rand: el AT3, o MATH-MATIC.

A fines de 1951, numerosas personas habían comprendido que, desde el punto de vista del usuario externo, de hecho los ordenadores estaban ejecutando programas que se habían escrito utilizando un código distinto al código máquina nativo. El hecho de que el ordenador realizara primero una tarea de traducción era irrelevante.

En este caso, uno bien podría diseñar su propio lenguaje nuevo con el objeto de hacer más sencilla la tarea de escribir al programador, en lugar de facilitar la tarea de traducción, especialmente porque el hardware se estaba haciendo más rápido y grande y los programadores no podían seguirle el paso.

El siguiente problema que hubo que abordar luego fue que no había forma de estandarización, porque cada máquina utilizaba su propio lenguaje relacionado con un pequeño conjunto particular de problemas. En consecuencia, el paso siguiente fue el desarrollo de un lenguaje independiente de todas las especificaciones de cualquier hardware dado, para abordar una clase más amplia de problemas. Este enfoque se inició en 1954 y condujo al FORTRAN (sistema de IBM de traducción de fórmulas [FOR*mula* TRANS*lation*] matemáticas), el primer auténtico lenguaje de programación.

El FORTRAN es un lenguaje de base matemática, muy adecuado para el trabajo que se realizaba entonces, mayormente numérico, y similar a muchos de los autocódigos de la época. Sin embargo, aún se basaba en gran medida en las arquitecturas de las máquinas disponibles. La comunidad empresarial se mostraba cada vez más interesada en utilizar los ordenadores para el procesamiento de datos a gran escala, pero necesitaban un lenguaje que se pareciera más al inglés comercial común.

No deja de ser divertido que en aquel entonces hubiera muchos programadores que pensaran que esto era imposible, ya que no existía forma imaginable de que el ordenador «comprendiera» palabras en lugar de números. Pronto esta falacia se hizo evidente, gracias (entre otros) a Grace Hopper, quien desarrolló un lenguaje denominado FLOW-MATIC, que podía ser leído y comprendido tanto por la dirección como por los programadores. Hacia 1956, este lenguaje se había convertido en el COBOL (*Common Business Oriented Language*). Estos lenguajes tenían especificaciones rígidas.

La teoría del lenguaje

Para entonces el número de ordenadores y de programadores había aumentado espectacularmente y la gente comenzaba a pensar más en el aspecto teórico de los lenguajes de programación, intentando hallar la forma más eficaz y elegante de expresar los algoritmos. Esto llevó, en 1958, al desarrollo del ALGOL (ALGO*rithmic Language*: lenguaje algorítmico). El ALGOL nunca alcanzó la amplia popularidad del FORTRAN o el COBOL; pero ocupa un importante lugar en el desarrollo de lenguajes. Fue el ALGOL el primero que incorporó el principio de diseño sólido de programa, que ha sido una consideración fundamental en todos los lenguajes posteriores.

Durante aquellos primeros días se desarrollaron otros lenguajes, muchos de los cuales todavía están en uso. Un de los mejores ejemplos es el LISP (LIS*t processing language*: lenguaje de procesamiento de listas), que se desarrolló entre 1956 y 1958. No sólo se sigue utilizando en la actualidad, sino que tiene una importancia creciente en el campo de la inteligencia artificial. Los tres lenguajes, FORTRAN, COBOL y ALGOL, han representado, sin embargo, la principal tendencia de la programación durante las dos últimas décadas. Con el correr de los años han sido objeto de algunas revisiones para incorporar nuevas facilidades y reflejar las modernas tendencias del diseño de lenguajes. Las revisiones actuales son el FORTRAN 77 (especificación que entró en vigor en 1977), el COBOL 74 y el ALGOL 68.

Código corto

En el *Short code* (Código corto) de John Maunchly la conocida sentencia de asignación de BASIC:

```
10 A = B + C
```

se escribiría

```
10 S0 03 S1 07 S2
```

donde 10 es el número de línea y S0, S1 y S2 son símbolos que representan las «variables» A, B y C como palabras individuales en la memoria. 03 y 07 son los códigos de operación para asignación y suma, respectivamente. En el lenguaje A2. la misma sentencia aparecería como:

```
ADD B C A
```

En 1964, en el Dartmouth College (Estados Unidos) se introdujo una versión muy simplificada del FORTRAN (con algunas influencias del ALGOL), para simplificar mucho más la tarea de aprender a programar y para utilizar los nuevos sistemas multiusuario de tiempo compartido que estaban apareciendo. Esta versión se dio a conocer como BASIC (*Beginners All-purpose Symbolic Instruction Code*: código de instrucciones simbólicas con fines generales destinado al principiante).

Cuando se estaba preparando el nuevo estándar 1968 para el ALGOL, Niklaus Wirth se mostró en desacuerdo con la forma en que el lenguaje se estaba volviendo más complejo, y se mostró favorable a un enfoque más simple y elegante. Por lo general se piensa que el ALGOL 68 es demasiado complejo para uso normal y su presencia es rara fuera de las más altas esferas académicas. Sin embargo, la versión de Wirth, muy simplificada y denominada PASCAL, ha obtenido una gran difusión.

El COBOL no ha dado lugar a derivados de la misma forma, dado que se ha utilizado casi exclusivamente en su propio campo. El enorme volumen de programas que hay escritos en COBOL ha significado que se haya utilizado como vehículo para numerosos desarrollos en el área de diseño de programas y sistemas; por ejemplo, generación de programas, diseño de programas estructurados y empleo de bases de datos.

Taquilla

Sistemas simples de reserva de localidades:

FORTRAN IV:

```
C       RESERVA DE LOCALIDADES PARA EL
        TEATRO.
C       PROGRAMA PARA ACEPTAR UN
        NUMERO DE ASIENTO, COMPROBAR SI
        YA ESTA OCUPADO.
C       RESERVADO SI ESTA LIBRE O DAR
        MENSAJE SI YA ESTA RESERVADO.
C
C       DECLARAR VARIABLES.
C
        INTEGER NUMASIENTO, ASIENTO (500)
C       SEÑALAR TODOS LOS ASIENTOS
C       DISPONIBLES.
C
        DO 100 I = 1,500
100     ASIENTO (I) = 0
C
C       LEER NUMERO DE ASIENTO Y
        COMPROBAR DISPONIBILIDAD
C
101     READ (1,10) NUMASIENTO
        IF (ASIENTO(NUMASIENTO).NE.0)GOTO
        102
C
C       EL ASIENTO ESTA DISPONIBLE.
C
        ASIENTO(NUMASIENTO) = 1
        WRITE(1,20)
        GOTO 103
C
C       EL ASIENTO ESTA RESERVADO
C
102     WRITE(1,30)
C
C       SIGUIENTE NUMERO DE ASIENTO
C
103     GOTO 100
C
C       SENTENCIAS FORMAT
C
10      FORMAT(I4)
20      FORMAT(1H,17HASIENTO ESTA LIBRE)
30      FORMAT(1H,19HASIENTO YA ESTA
        RESERVADO)
        END
```

ALGOL 60:

```
comment RESERVA DE LOCAL. PARA EL TEATRO.
PROGRAMA PARA ACEPTAR UN NUMERO DE
ASIENTO, COMPROBAR SI YA ESTA RESERVADO,
RESERVARLO SI ESTA LIBRE O DAR MENSAJE SI
YA ESTA RESERVADO
OBSERVE QUE EL ALGOL NO INCLUYE
SENTENCIAS DE ENTRADA/SALIDA:
  begin
    integer NUMASIENTO, I;
    integer array ASIENTO[1:500];
    comment MARCAR LOS ASIENTOS LIBRES
    for I:=1 step 1 until 500 do
      ASIENTO[I]:=0
    comment LEER NUMERO DE ASIENTO Y
    COMPROBAR DISPONIBILIDAD;
  NUEVO ASIENTO:((read NUMASIENTO));
    if ASIENTO[NUMASIENTO]=0 then
    begin
        ASIENTO[NUMASIENTO]:=1;
        ((print 'EL ASIENTO ESTA LIBRE'));
      end
      else
        ((print 'EL ASIENTO YA ESTA
        RESERVADO'));
      goto NUEVOASIENTO;
  end
```

COBOL 74 [sólo división DATOS y PROCEDIMIENTO]:

```
*RESERVA DE LOCALIDADES PARA EL TEATRO.
*PROGRAMA PARA ACEPTAR UN NUMERO DE
ASIENTO, COMPROBAR SI YA ESTA RESERVADO,
*RESERVARLO SI ESTA LIBRE O DAR MENSAJE SI
YA ESTA RESERVADO.
DIVISION DE DATOS.
SECCION DE ALMACENAMIENTO OPERATIVO.
01  DISPONIBILIDAD-ASIENTO.
    02 ASIENTO PIC 9 OCCURS 500 TIMES.
77  NUM-ASIENTO PIC 999.
77  CONTADOR-BUCLE PIC 999.
77  ASIENTO-DISPONIBLE PIC X(17)VALUE
    'EL ASIENTO ESTA LIBRE'.
77  ASIENTO-RESERVADO PIC X(19)VALUE
    'EL ASIENTO YA ESTA RESERVADO'.
DIVISION PROCEDIMIENTO.
PARRAFO PRINCIPAL.
*MARCAR TODOS LOS ASIENTOS DISPONIBLES
      REALIZAR PARRAFO-MARCAR-ASIENTO-
        DISPONIBLE VARIANDO EL CONTADOR DEL
        BUCLE DE 1 EN 1 UNTIL I>500.
*LEER UN NUMERO DE ASIENTO Y COMPROBAR
      DISPONIBILIDAD.
      REALIZAR PARRAFO-TOMAR-NUMERO-
      ASIENTO.
      STOP RUN.
PARRAFO-MARCAR-ASIENTO-DISPONIBLE.
  MOVE ZERO TO ASIENTO (CONTADOR-BUCLE).
PARRAFO-TOMAR-NUMERO-ASIENTO.
  IF ASIENTO (NUMERO-ASIENTO) IS EQUAL TO 0
     MOVE 1 TO ASIENTO(NUMERO-ASIENTO)
     DISPLAY ASIENTO LIBRE
  ELSE
     DISPLAY ASIENTO RESERVADO
  GOTO PARRAFO-TOMAR-NUMERO-ASIENTO.
```

Reserva de localidades
Ofrecemos aquí listados en tres lenguajes de alto nivel distintos (FORTRAN, ALGOL y COBOL) demostrando la estructura del programa y algunas configuraciones comparativas. Cada programa acepta como entrada un número de asiento, comprueba si el número no se ha entrado ya (en cuyo caso estaría «ocupado») y, si no es así, señala que ese asiento ya tiene dueño

BBC Hulton Picture Library

Función casera

El Pioneer PX-7 puede combinarse con un aparato reproductor de discos láser para crear un sistema propio de video interactivo

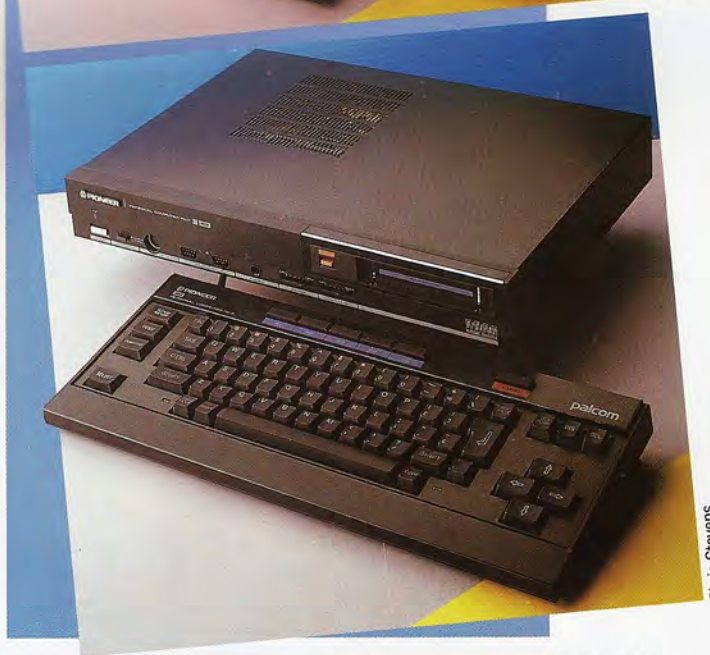
Ya vimos las nuevas disponibilidades que ofrecen los discos láser controlados por ordenador en los campos de la educación y del ocio. El ordenador personal PX-7 de Pioneer, en unión con el reproductor de discos láser LD-700 (que vemos aquí) o el LD-1100, más sofisticado, ofrece la realización de esas posibilidades a un precio al alcance del usuario personal. El PX-7 es un ordenador MSX sumamente desarrollado, mientras que el LD-700 es un reproductor de discos láser de precio asequible que utiliza discos láser de 12 pulgadas estilo Philips.

El PX-7 es muy poco corriente tratándose de un ordenador MSX, siendo el primero de ellos que posee un teclado separado. Esto permite ubicar la unidad CPU junto a un grabador de video, un reproductor de discos láser, un televisor o un sistema de alta fidelidad. En consecuencia, el diseño de la unidad CPU responde más al de un componente de *hi-fi* que a un ordenador personal, y el teclado se conecta mediante un generoso cable de metro y medio. En la parte delantera de la máquina hay un control de volumen, un conector para auriculares y un control de mezcla, que regula el equilibrio entre el sonido generado por el ordenador y el proveniente de una fuente externa como el disco láser. Un conmutador de audio-video aísla efectivamente al ordenador, permitiendo que las señales de video y audio externas pasen directamente a un *hi-fi* o a un televisor conectados a la unidad.

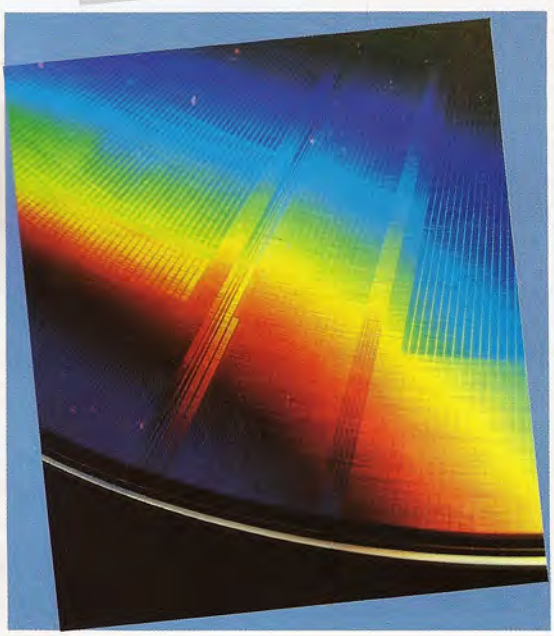
En el interior hay un ordenador MSX estándar de 32 K, que se puede utilizar con la gama de software MSX y los periféricos existentes. Sorprendentemente, a pesar de la sofisticada tecnología del disco láser, el PX-7 utiliza cassettes para almacenar sus programas e información. Para quienes deseen ahondar más en el potencial del micro, un segundo conector para cartuchos en la parte posterior hará lugar para unidades de disco y otros accesorios.

Hay una gama completa de interfaces de ordenador para televisión, pantalla compuesta o en color, cassette, palancas de mando *(joysticks)* gemelas, dos cartuchos y una impresora Centronics. Además, la unidad posee conexiones estéreo para un *hi-fi* y una interface para «control del sistema». Esta última es una interface para fines generales y se puede usar para conectar reproductores de discos láser, grabadores de videocassettes, etc. A medida que vayan apareciendo nuevos equipos, la interface y su software controlador permitirán su adición al sistema. Se pueden conectar varios dispositivos a la vez, teniendo cada uno de ellos un

Bien avenidos
El ordenador personal MSX PX-7 y el reproductor de discos láser LD-700, de Pioneer, están diseñados para trabajar juntos. El PX-7 se puede programar utilizando ampliaciones al BASIC MSX estándar para controlar el reproductor de discos láser de modo de poder seleccionar y visualizar secuencias o fotogramas individuales. También se pueden mezclar gráficos y sonido del ordenador con la salida de video del reproductor de discos, para producir video interactivo bajo el control del ordenador



Chris Stevens



Pistas láser
Los discos láser CAV *(constant angular velocity*: velocidad angular constante), al igual que los discos magnéticos, ofrecen acceso directo a cualquier parte del disco. El mecanismo controlador puede localizar secciones individuales en la superficie del disco en relación a las pistas radiales que vemos aquí. Cada anillo del disco corresponde a un fotograma del programa almacenado, reteniendo el número de fotograma en la parte correspondiente de la pista radial

Conectores a granel
Una de las características sobresalientes del PX-7 es su amplia gama de conectores, que permiten la conexión, mediante cables estándares, a una completa gama de equipos de audio y video

código de dispositivo para permitir que un programa especifique el dispositivo al cual está dirigida una instrucción.

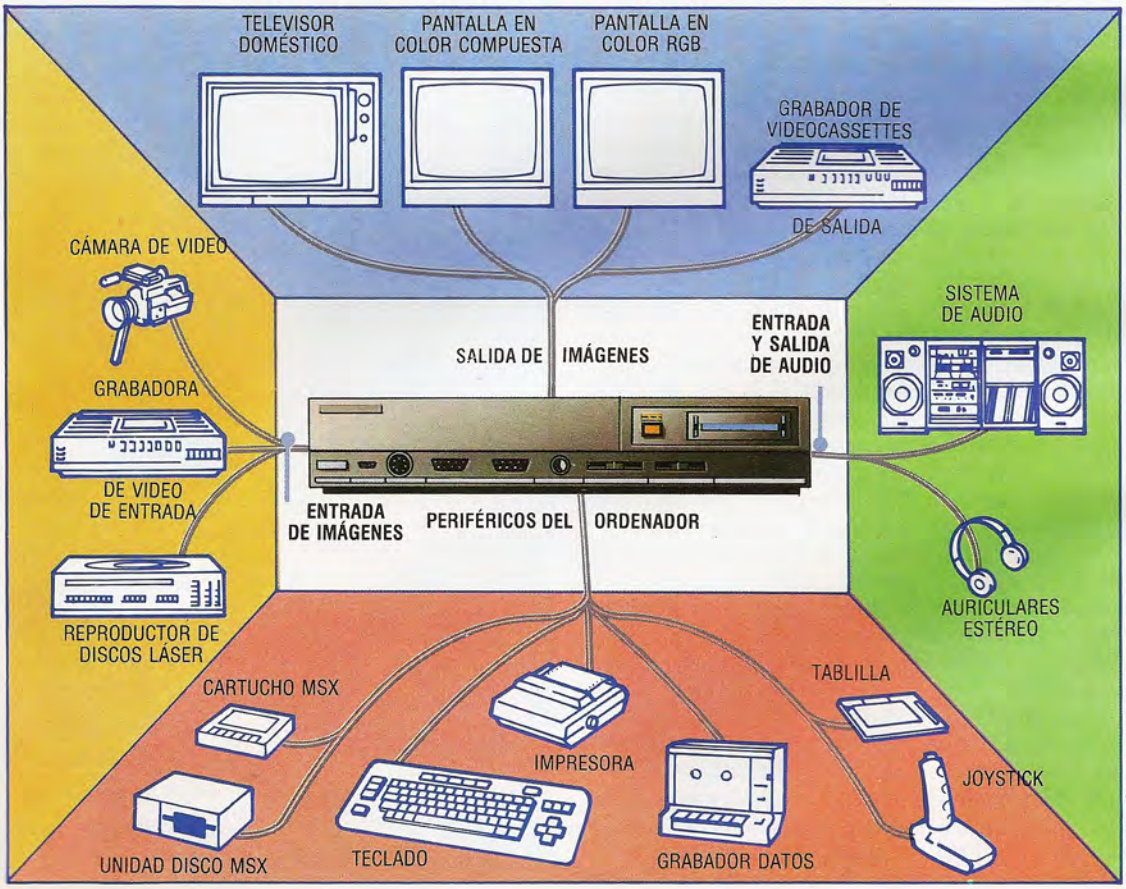
El control de la interface es simple en virtud de un conjunto de ampliaciones del BASIC MSX retenidas en ROM. Cuando se pone por primera vez en marcha el ordenador, usted puede elegir ejecutar el BASIC MSX común o el P-BASIC (BASIC MSX con instrucciones para el control del sistema). Las nuevas instrucciones tienen la forma de sentencias CALL, de modo que el BASIC MSX propiamente dicho es el estándar. La mayor parte de las 16 nuevas instrucciones están relacionadas con el control de las conexiones de video y audio del ordenador. Se puede visualizar la visualización del ordenador, la visualización de video entrante (como un disco láser) o superponer ambas. Esto permite que los gráficos y el texto del ordenador aparezcan encima de las imágenes procedentes del disco láser o de la videocinta. Asimismo, puede alternar entre las modalidades ordenador, video y superposición, utilizando cuatro teclas adicionales del teclado que son las únicas adiciones hechas al MSX estándar.

Las instrucciones de sonido permiten poner sordina a uno de los canales estéreo o a ambos y regular el equilibrio entre ambos. Hay un conjunto de extras para mejorar el BASIC MSX: instrucciones que borran la pantalla de diversas maneras, guardan y cargan la visualización en una unidad de casette, etc. Sin embargo, la instrucción más importante es CALL REMOTE, que envía una instrucción a la interface de control del sistema.

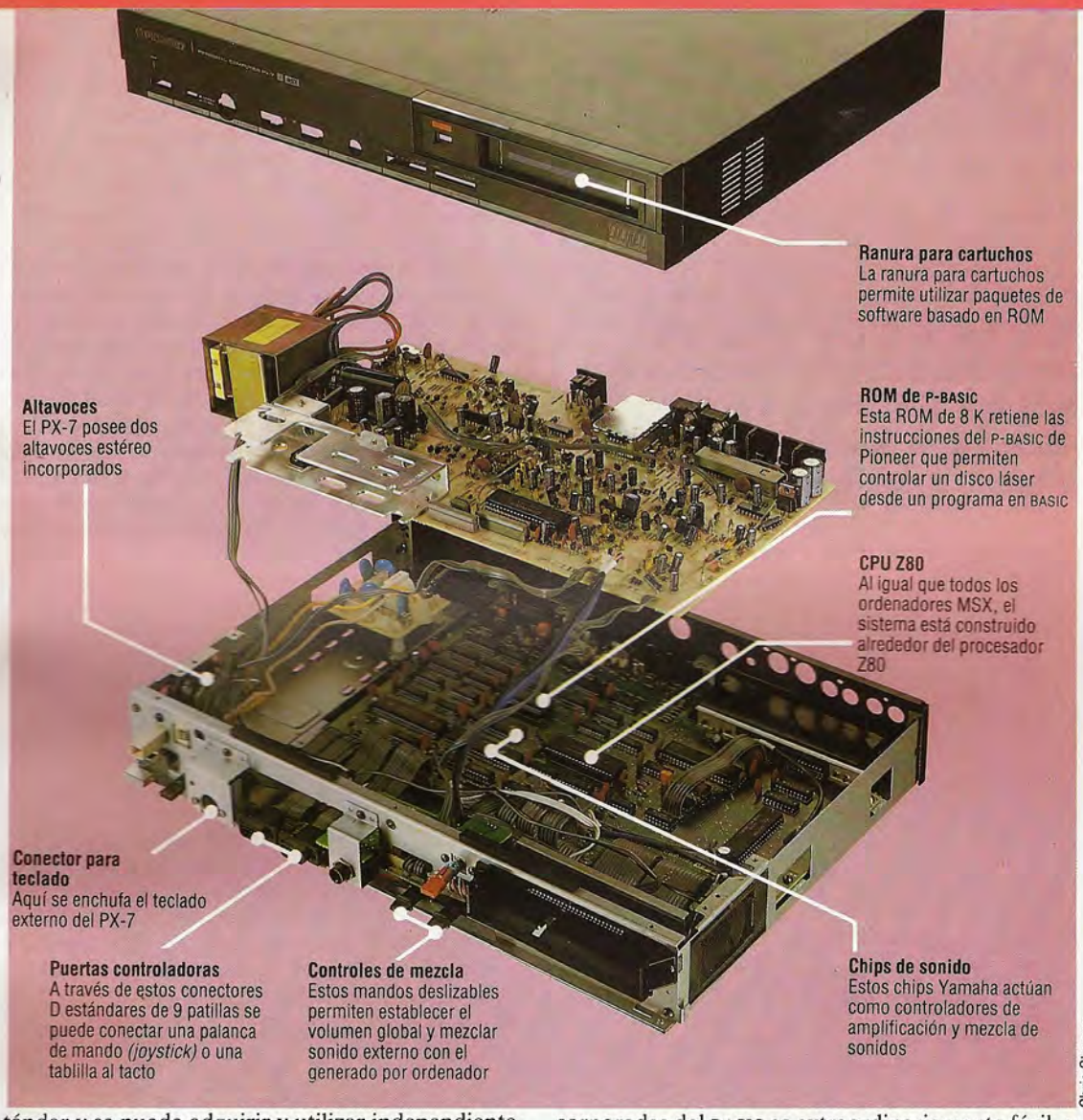
Existen numerosas instrucciones específicas para el control del disco láser. Por ejemplo, CALL SEARCH (0,F,2000) pedirá al reproductor que busque en el disco el fotograma 2 000. CALL FRAME es la instrucción más sofisticada. Una vez ejecutada, la subrutina especificada se ejecutará automáticamente cuando el reproductor muestre un fotograma o capítulo determinado del disco. Esto permite vincular estrechamente un programa al sistema de disco láser. Por ejemplo, en un programa educativo, el ordenador podría preguntar ¿Quieres saber algo más acerca de esto? cada vez que un estudiante reprodujera una parte determinada del disco, pudiendo pasar a mostrar otra sección de la película.

El LD-700 es un reproductor de disco láser es-

Centro de control
Una de las intenciones originales del estándar MSX era la de permitir que los ordenadores conformaran el centro de control de un sistema de entretenimiento doméstico completo, integrado por componentes de audio y video. El Pioneer PX-7 es la primera máquina MSX que puede utilizarse de este modo. La adición del P-BASIC, una ampliación del BASIC MSX estándar, permite que el software controle verdaderas imágenes de video y sonido. Éstos se pueden mezclar con gráficos generados por ordenador, abriendo las puertas de este modo a los juegos de aventuras de video, programas de enseñanza por video interactivo y simulaciones más realistas



Kevin Jones



Chris Stevens

tándar y se puede adquirir y utilizar independientemente del ordenador PX-7. Cada disco de 12 pulgadas puede almacenar hasta 54 000 fotogramas, y esto significa que puede mostrar películas enteras, además de ofrecer fotogramas individuales, cámara lenta y rápida de una calidad muy superior a la que permite el video. El controlador a distancia que se suministra con la unidad permite la búsqueda de fotograma y capítulo (siendo el capítulo una división del disco). Cada disco lleva dos pistas de audio, permitiendo el almacenamiento de la banda sonora en dos idiomas, si bien algunos discos utilizan la segunda pista para almacenar un programa de ordenador ya hecho.

El hardware del Pioneer está bien construido y acabado, aunque el ordenador puede calentarse mucho al trabajar en unión con el reproductor de discos láser. Pioneer ofrece la tablilla para gráficos PXTB-7 y un paquete para gráficos basado en cartuchos denominado *Video Art*, que permite diseñar programas que utilicen gráficos de ordenador superpuestos sobre imágenes de disco láser. No obstante, el software resulta decepcionante: es lento, difícil de emplear y de capacidades limitadas.

El PX-7 hace que la creación y ejecución de programas de video interactivos para el hogar, el despacho o la clase resulten muy sencillas. Escribir software adecuado utilizando las ampliaciones incorporadas del BASIC es extraordinariamente fácil, y el propio BASIC MSX es sofisticado y moderadamente veloz. La velocidad no constituye mayor problema, puesto que el reproductor tarda alrededor de dos segundos en buscar un fotograma determinado.

En vez de quedar limitados a un sistema económico, el PX-7 y el LD-700 proporcionan el potencial para programas de video interactivos completos. Usted podría producir juegos de disco láser, como los que han obtenido tanto éxito en las salas recreativas, crear una gran base de datos de imágenes, etc. El sistema también acepta discos CPE *(computer program encoded*: con programa para ordenador codificado), es decir, aquellos que en una de las dos pistas de audio tienen almacenado un programa para ordenador ya listo.

Sin embargo, es probable que el costo de los discos matriz impida que muchas personas diseñen sus propios discos, y los usuarios habrán de confiar en productos comerciales de empresas cinematográficas o de software. Como es habitual con la nueva tecnología, la aparición del software que explote cabalmente el nuevo hardware llevará su tiempo. Mientras tanto, el Pioneer PX-7 es uno de los más interesantes ordenadores MSX y, en unión de un disco láser, representa cabalmente el futuro de la informática del ocio doméstico y de la educación.

PIONEER PX-7

DIMENSIONES

420 × 323 × 70 mm

MEMORIA

32 K de RAM para el usuario, ampliables a 64 K; 16 K de RAM de video, 40 K de ROM

CPU

Procesador Z80 a 4 MHz

PANTALLA

40 columnas × 24 líneas, 16 colores, gráficos en alta resolución de 256 × 192, con hasta 32 sprites utilizando el chip de visualización 9929

SONIDO

Salidas estéreo y sonido a tres voces utilizando el chip de sonido 8910

INTERFACES

Video compuesto, TV, RGB, impresora Centronics, 2 puertas para palanca de mando *joystick* y 2 puertas para cartuchos, cassette, entrada audio, salida audio, auriculares, control del sistema

LENGUAJES DISPONIBLES

BASIC MSX 32 K y P-BASIC 8 K

DOCUMENTACION

El PX-7 utiliza pequeños folletos, como los que cabría esperar en un sistema *hi-fi*. Algunos están muy pobremente traducidos del japonés y toda la información se presenta en un anodino estilo de informe. Los manuales, sin embargo, son completos e incluyen gran cantidad de información técnica

VENTAJAS

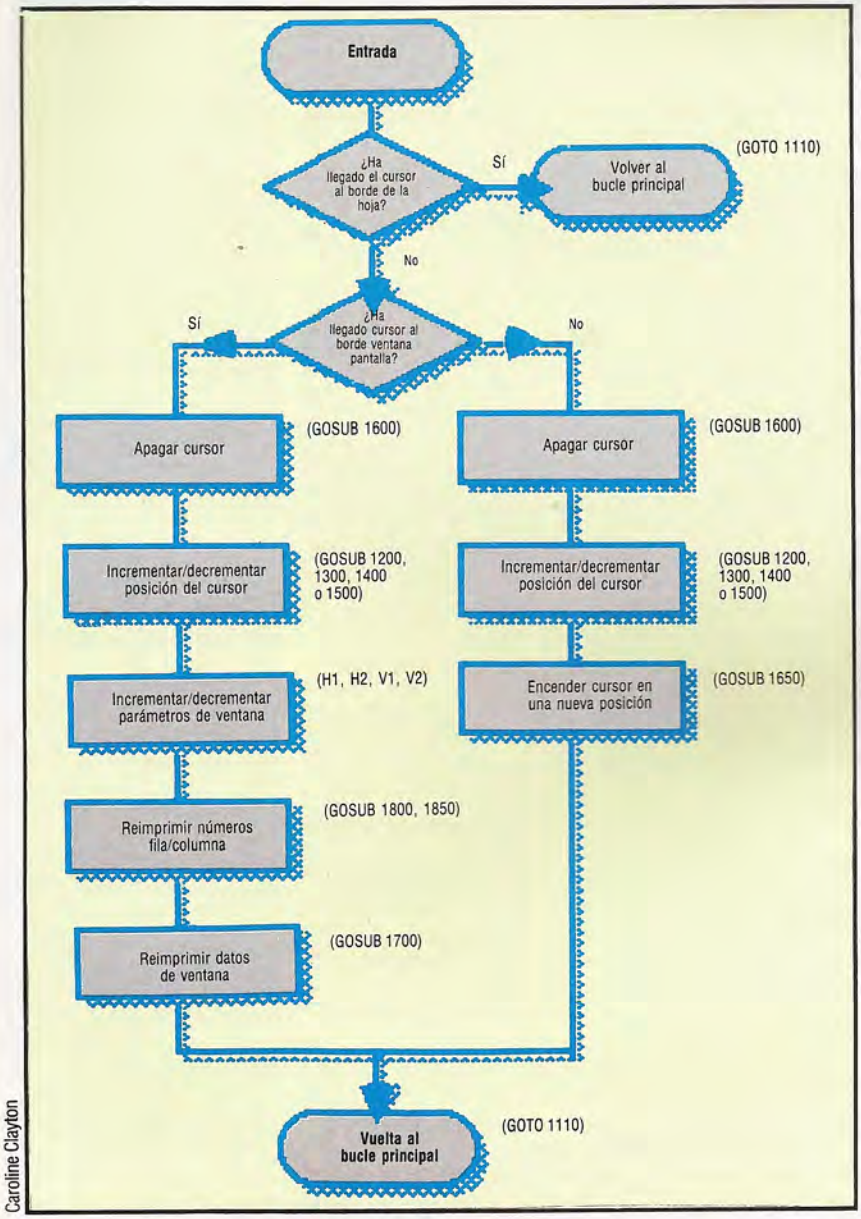
El PX-7 es el más desarrollado de todos los sistemas MSX disponibles. En unión del disco láser, representa un auténtico avance en la tecnología del entretenimiento doméstico. Las unidades están bien construidas, su diseño es agradable y su precio, moderado

DESVENTAJAS

MSX ha recibido un mínimo apoyo por parte de empresas independientes. Asimismo, el PX-7 sufre las consecuencias de la escasez de software y discos láser para sacar partido de sus facilidades

Cuadrícula inicial

Esta vez nos corresponde programar las visualizaciones en pantalla para el Amstrad CPC 464/664, el BBC Micro y el Sinclair Spectrum



Caroline Clayton

Cerca del borde
El diagrama de flujo muestra la manera en que nuestro programa controla al cursor sobre la «hoja». Dado que la pantalla sólo es una ventana de la hoja electrónica, cuando el cursor llega a los bordes superior, inferior, izquierdo o derecho de la zona de pantalla visible, se emprenderá una acción para pasar a la siguiente porción de la hoja

Debido a que los micros para los que se ha diseñado la hoja electrónica poseen métodos diferentes para producir visualizaciones en pantalla, ofrecemos las rutinas gráficas para cada uno de forma independiente. (Anteriormente se ha ilustrado la versión para el Commodore 64.) Las principales funciones de esta parte del programa son imprimir en la pantalla la cuadrícula de la hoja electrónica, junto con los números de filas y columnas, y controlar el movimiento del cursor de la hoja electrónica a través de la cuadrícula.

En las rutinas de manipulación del cursor se incluyen secciones que desplazan el cursor a izquierda, derecha, arriba y abajo. La pantalla sólo puede visualizar en cada momento una porción de la hoja electrónica (le será útil pensar en la pantalla como una ventana a través de la cual sólo se ve una parte de la hoja) y, por tanto, las rutinas del cursor también manipularán el movimiento de la ventana a través de la hoja electrónica.

Las líneas 1000-1080 conforman una subrutina que imprime la cuadrícula de la hoja electrónica. La siguiente sección, que empieza en la línea 1100, es el bucle de control principal del programa, que esencialmente explora el teclado en busca de una pulsación de tecla. Se pueden utilizar teclas para trasladar el cursor por la hoja o para seleccionar una función, como entrar una fórmula en una celda determinada. Desde esta sección se llama la subrutina adecuada.

Movimiento del cursor

La mayoría de las funciones de la hoja electrónica serán tema de futuros capítulos y, por lo tanto, intentar seleccionarlas en esta etapa sólo hará que el programa quede colgado. No obstante, las subrutinas incluidas en este capítulo (en las líneas 1200, 1300, 1400 y 1500) le permitirán desplazar el cursor por la pantalla, puesto que son las rutinas de movimiento del cursor para DERECHA, IZQUIERDA, ABAJO y ARRIBA, respectivamente. Estas rutinas son similares, de modo que sólo veremos una de ellas para hacernos una idea del modo en que trabajan las cuatro.

Al pulsar la tecla para mover el cursor hacia la derecha, el programa salta a la subrutina de la línea 1200, y la línea 1210 comprueba si el cursor ha llegado al borde de la hoja. Esto lo realiza comprobando la coordenada X, para determinar si ha alcanzado su valor máximo, 15. De ser así, se devuelve el control al bucle principal del programa; de lo contrario, la rutina sigue adelante. El siguiente paso consiste en ver si el cursor se halla en el extremo derecho de la ventana de pantalla actual. Se utilizan las variables H1 y H2 para los límites horizontales inferior y superior de la ventana; por ejemplo, si H1=2 y H2=6, en la pantalla están visibles las columnas 2 a 6 de la hoja. Si X ha alcanzado el valor retenido en H2, se pone en movimiento la

siguiente cadena de acontecimientos: se apaga el cursor, se incrementa el valor de X, se incrementan los valores de H1 y H2, la subrutina de la línea 1800 imprime nuevos números de fila y columna, y se imprimen en la hoja los datos de la nueva celda. Por último, se vuelve a encender el cursor.

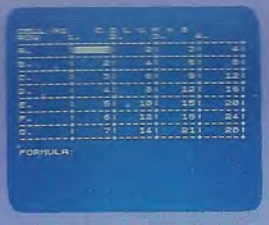
Las subrutinas de las líneas 1600 y 1650 son necesariamente diferentes para cada uno de los cuatro ordenadores, y controlan el encendido y el apagado del cursor.

En todas las versiones, éste se muestra invirtiendo los colores del primer plano y del fondo de la celda adecuada, pero la forma en que se consigue este efecto es peculiar del hardware y el firmware de visualización de cada máquina.

En el Spectrum, los colores del primer plano y del fondo de cada celda de caracteres son controlados por un byte de la memoria llamado *mapa de atributos*. Los tres bits inferiores del byte de atributos controlan el color INK y los bits tres a cinco controlan el color PAPER. En consecuencia, para invertir los colores sólo es necesario localizar el grupo de bytes de atributos que corresponde a la celda actual de la hoja electrónica y colocar (POKE) los valores apropiados.

En las versiones para el Amstrad y el BBC Micro, el proceso es ligeramente más complicado, porque el valor de la celda se debe reimprimir en la celda después de encender o apagar el cursor. Los valores de la celda están retenidos en la matriz M(,); se debe hallar el valor correcto para la celda actual y convertirlo en una serie lista para impresión. En el BBC Micro, los colores se pueden intercambiar simultáneamente utilizando la instrucción COLOUR. En el Amstrad hay disponible un carácter de control, CHR$(24), que se puede incorporar a una sentencia PRINT para intercambiar los colores PEN y PAPER actuales.

Sinclair Spectrum:



```
100 GO SUB 3000
110 GO SUB 1000
120 GO SUB 1700
130 GO SUB 1100
999 STOP
1000 BORDER 1: PAPER 1: CLS : INK 7
1005 PRINT "        C O L U M N A S"
1007 PRINT "FILA     1.    2.    3.    4."
1010 PRINT "-------+-----+-----+-----+-----+"
1020 FOR C=1 TO 6
1030 PRINT CHR$ (C+64);".      |     |
     |     |     |"
1040 PRINT "-------+-----+-----+-----+-----+"
1050 NEXT C
1060 PRINT CHR$ (C+64);".      |     |
     |     |     |"
1070 PRINT "-------+-----+-----+-----+-----+"
1080 RETURN
1100 PRINT AT 0,0;"CELDA:";CHR$(Y+64);STR$
     (X)
1110 LET A$=INKEY$: IF A$="" THEN GO TO
     1110
1120 IF A$="8" THEN GO TO 1200: REM MUEVE
     DERECHA
1130 IF A$="5" THEN GO TO 1300: REM MUEVE
     IZQUIERDA
1140 IF A$="6" THEN GO TO 1400: REM MUEVE
     ABAJO
1150 IF A$="7" THEN GO TO 1500: REM MUEVE
     ARRIBA
1155 IF A$="C" THEN GO SUB 2300: REM
     CALCULA HOJA
1160 IF A$="F" THEN GO SUB 2000: REM ENTRA
     FORMULA
1165 IF A$="E" THEN GO SUB 2100: REM ENTRA
     DATOS NUMERICOS EN CELDA
1168 IF A$="H" THEN GO TO 6000: REM
     IMPRIME PANTALLA AYUDA
1170 IF A$="S" THEN GO SUB 5150: REM
     RECUPERA HOJA ACTUAL EN MEMORIA
1172 IF A$="G" THEN GO SUB 5100: REM TOMA
     HOJA ANTERIOR
1174 IF A$ "Z" THEN GO SUB 5000: REM BORRA
     HOJA ACTUAL
1176 IF A$="R" THEN GO SUB 5700: REM
     REPRODUCE FORMULA
1178 IF A$="T" THEN GO SUB 5200: REM TAB A
     NUEVA CELDA
1180 IF A$="D" THEN GO SUB 7000: REM
     CARGA/GUARDA DATOS/FORMULAS
1190 GO TO 1110
1200 REM ***** MUEVE DERECHA *****
1210 IF X=15 THEN GO TO 1100
1220 IF X=H2 THEN GO SUB 1600: LET X=X+1:
     LET H1=H1 + 1: LET H2=H2 + 1: GO TO
     1270
1230 GO SUB 1600: LET X=X+1: GO SUB 1650:
     GO TO 1100
1270 GO SUB 1800: GO SUB 1700: GO TO 1100
1300 REM ***** MUEVE IZQUIERDA *****
1310 IF X=1 THEN GO TO 1100
1320 IF X=H1 THEN GO SUB 1600: LET X=X-1:
     LET H1=H1-1: LET H2=H2-1: GO TO 1370
1330 GO SUB 1600: LET X=X-1: GO SUB 1650:
     GO TO 1100
1370 GO SUB 1800: GO SUB 1700: GO TO 1100
1400 REM ***** MUEVE ABAJO *****
1410 IF Y=15 THEN GO TO 1100
1420 IF Y=V2 THEN GO SUB 1600: LET Y=Y+1:
     LET V1=V1+1: LET V2=V2+1: GO TO 1470
1430 GO SUB 1600: LET Y=Y+1: GO SUB 1650:
     GO TO 1100
1470 GO SUB 1850: GO SUB 1700: GO TO 1100
1500 REM ***** MUEVE ARRIBA *****
1510 IF Y=1 THEN GO TO 1100
1520 IF Y=V1 THEN GO SUB 1600: LET Y=Y1-1:
     LET V1=V1-1:LET V2=V2-1: GO TO 1570
1530 GO SUB 1600: LET Y=Y-1: GO SUB 1650:
     GO TO 1100
1570 GO SUB 1850: GO SUB 1700: GO TO 1100
1600 REM ***** APAGAR CURSOR *****
1610 LET CU=22528+32*(V(Y+1-V1))
     +H(X+1-H1)
1615 POKE CU,15: POKE CU+1,15: POKE
     CU+2,15
1620 POKE CU+3,15: POKE CU+4,15: RETURN
1650 REM ***** ENCIENDE CURSOR *****
1660 LET CU=22528+32*(V(Y+1-V1))
     +H(X+1-H1)
1665 POKE CU,56: POKE CU+1,56: POKE
     CU+2,56
1670 POKE CU+3,56: POKE CU+4,56
1690 GO SUB 1900: RETURN
1700 REM ***** IMPRIME DATOS EN HOJA
     *****
1710 FOR I=0 TO 6
1720 FOR J=0 TO 3
1730 LET P$=STR$ (M(I+V1,J+H1))
1740 PRINT AT V(I+1),H(J+1);"   "
1745 PRINT AT V(I+1),(H(J+1)+5-LEN (P$));P$
1750 NEXT J: NEXT I
1760 GO SUB 1650: RETURN
1800 REM ***** IMPRIME NUMS
     COLUMNAS*****
1810 FOR I=H1 TO H2
1820 PRINT AT 1,7+6*(I-H1);I;"."
1830 NEXT I
1840 RETURN
1850 REM *** IMPRIME NUMS FILAS ***
1870 FOR C=V1 TO V2
1880 PRINT AT 2*(C-V1)+3,0;
     CHR$(C+4);"."
1890 NEXT C
1895 RETURN
1900 REM * FORMULA CELDA ACTUAL *
1920 LET D$=F$((Y-1)*15+X,1 TO )
1930 PRINT AT 18,0;"FORMULA: "
1940 PRINT AT 18,0;"FORMULA: ";D$
1945 RETURN
```

Rutinas preparación matrices:

```
3000 REM ********************************
3001 REM *        PREPARA MATRICES      *
3002 REM ********************************
3010 DIM H(4): DIM V(7): DIM S(20): DIM S$(20):
     DIM G$ (20): DIM C(20)
3020 FOR C=0 TO 3
3030 LET H(C+1)=6*C+8: REM CALC POS-X
3040 NEXT C
3050 FOR C=1 TO 7
3060 LET V(C)=2*C+1: REM CALC POS-Y
3070 NEXT C
3075 LET X=1: LET Y=1
3080 LET H1=X: LET H2=X+3: LET V1=1: LET
     V2=V1+6
3090 REM ********************************
3091 REM *        MATRIZ VALORES        *
3092 REM ********************************
3100 DIM M(15,15): DIM N(15,15)
3110 FOR I=1 TO 15
3120 FOR J=1 TO 15
3130 LET M(I,J)=I*J
3140 NEXT J: NEXT I
3150 DIM F$(255,20): DIM G$(20): RETURN
```

BBC Micro:


```
 10 REM **** HOJA ELECTRONICA BBC ****
 40 MODE 4
 50 *FX4,1
 60 LET CO$=CHR$(30):CL$=CHR$(8):CR$=
    CHR$(9):CU$=CHR$(11): CD$=CHR$(10)
 70 REM VDU 23,1,0;0;0;0;
100 GOSUB 3000:REM PREPARA MATRICES Y VARIABLES
    PANTALLA
110 GOSUB 1000:REM IMPRIME PANTALLA
120 GOSUB 1700:REM IMPRIME VENTANA DATOS EN
    PANTALLA
130 GOTO 1100:REM RUTINA PRINCIPAL TECLADO
999 STOP
1000 PRINT CHR$(12)
1005 PRINT "                                C O L U M N A S"
1006 PRINT
1007 PRINT "FILA          1.    2.    3.    4.    5."
1010 PRINT "          +-----+-----+-----+-----+-----+"
1020 FOR C=1 TO 7
1030 PRINT " "CHR$(C+64);".  |     |     |     |     |     |"
1040 PRINT "--------+-----+-----+-----+-----+-----+"
1050 NEXT G
1060 PRINT " "CHR$(C+64);".  |     |     |     |     |     |"
1070 PRINT "--------+-----+-----+-----+-----+-----+"
1080 RETURN
1100 P$=CHR$(Y+64)+STR$(X):PRINT
     CO$;CD$"CELDA:";P$;" "
1110 LET A$=GET$
1120 LET A$=CHR$(137) THEN 1200:REM MUEVE CURSOR
     DERECHA
1130 IF A$=CHR$(136) THEN 1300:REM MUEVE CURSOR
     IZQUIERDA
1140 IF A$=CHR$(138) THEN 1400:REM MUEVE CURSOR ABAJO
1150 IF A$=CHR$(139) THEN 1500:REM MUEVE CURSOR
     ARRIBA
1152 IF A$="H" THEN GOSUB 6000:REM IMPRIME PANTALLA
     AYUDA
1154 IF A$="F" THEN GOSUB 2000:REM ENTRA FORMULA
1156 IF A$="S" THEN GOSUB 5150:REM ALMACENA HOJA
     ACTUAL
1158 IF A$="G" THEN GOSUB 5100:REM TOMA HOJA
     ANTERIOR
1160 IF A$="C" THEN GOSUB 2300:REM CALCULA HOJA
1165 IF A$=CHR$(13) THEN RETURN
1170 IF A$>="0" AND A$<="9" THEN GOSUB 2100:REM
     RUTINA ENTRADA DATOS
1180 IF A$="Z" THEN GOSUB 5000:REM BORRAR HOJA
1185 IF A$="R" THEN GOSUB 5700:REM DUPLICA HOJA
1187 IF A$="T" THEN GOSUB 5200: REM TAB A NUEVA CELDA
1189 IF (INKEY(-119)) THEN GOSUB 7000:REM RUTINAS
     CARGAR/GUARDAR
1190 GOTO 1100:REM VUELVE AL COMIENZO
1200 REM ***** MUEVE DERECHA *****
1210 IF X=15 THEN 1100
1220 IF X=H2 THEN GOSUB 1600:X=X+1:H1=
     H1+1:H2=H2+1:GOTO 1270
1230 GOSUB 1600:LET X=X+1:GOSUB 1650:GO TO 1100
1270 GOSUB 1800:GOSUB 1700:GOTO 1100
1300 REM ****** MUEVE IZQUIERDA ******
1310 IF X=1 THEN 1100
1320 IF X=H1 THEN GOSUB 1600:X=X-1:H1=
     H1-1:H2=H2-1:GOTO 1370
1330 GOSUB 1600:LET X=X-1:GOSUB 1650:GOTO 1100
1370 GOSUB 1800:GOSUB 1700:GOTO 1100
1400 REM ***** MOVER ABAJO *****
1410 IF Y=15 THEN 1100
1420 IF Y=V2 THEN GOSUB 1600:LET Y=Y+1:V1=
     V1+1:V2=V2+1:GOTO 1470
1430 GOSUB 1600:LET Y=Y+1:GOSUB 1650:GOTO 1100
1470 GOSUB 1850:GOSUB 1700:GOTO 1100
1500 REM ***** MUEVE ARRIBA *****
1510 IF Y=1 THEN 1100
1520 IF Y=V1 THEN GOSUB 1600:LET Y=Y-1:V1=
     V1-1:V2=V2-1:GOTO 1570
1530 GOSUB 1600:LET Y=Y-1:GOSUB 1650:GOTO 1100
1570 GOSUB 1850:GOSUB 1700:GOTO 1100
1600 REM ***** APAGA CURSOR *****
1610 P$=STR$(M(Y-V1+1,X-H1+1))
1615 IF LEN(P$)< 5 THEN P$=" "+P$:GOTO 1615
1620 COLOUR 1:COLOUR 128
1625 PRINT TAB(H(X-H1+1)-1,V(Y-V1+1)-1);P$
1630 COLOUR 1: COLOUR 128
1640 RETURN
1650 REM ***** ENCIENDE CURSOR *****
1660 P$=STR$(M(Y-V1+1,X-H1+1))
1665 IF LEN(P$)< 5 THEN P$=" "+P$:GOTO 1665
1670 COLOUR 2:COLOUR 129
1675 PRINT TAB(H(X-H1+1)-1,V(Y-V1+1)-1;P$
1680 COLOUR 1: COLOUR 128
1690 GOSUB 1900:RETURN
1700 REM ***** IMPRIME VENTANA DATOS DESDE HOJA EN
     PANTALLA *****
1710 FOR I=0 TO 7
1720 FOR J=0 TO 4
1730 P$=STR$(M(I+V1,J+H1))
1735 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 1735
1740 PRINT TAB(H(J+1)-1,V(I+1)-1);"     "
1745 PRINT TAB(H(J+1)-1,V(I+1)-1);P$
1750 NEXT J, I
1760 GOSUB 1650:RETURN
1800 REM ***** IMPRIME NUM COLUMNA *****
1810 FOR I=H1 TO H2:PRINT TAB(8+6*(I-H1),3);I;"."
1820 NEXT I:RETURN
1850 REM ***** IMPRIME ETIQUETAS FILAS *****
1860 FOR C=V1 TO V2
1870 PRINT TAB(1,V(C-V1+1)-1);CHR$(C+64);"."
1880 PRINT: NEXT C: RETURN
1900 REM ***** FORMULA DE CELDA ACTUAL *****
1910 LET D$=F$((Y-1)*15+X)
1920 PRINT TAB(0,22);"    "
1930 PRINT TAB(0,22);"FORMULA: ";D$
1940 RETURN
```

Rutinas preparación matrices:

```
3000 REM ***** PREPARAR MATRICES *****
3010 DIM H(5),V(8),ST(20),ST$(20),E$(20),G$(20),C(20)
3020 FOR C=0 TO 4
3030 LET H(C+1)=6*C+10
3040 NEXT C
3050 FOR C=1 TO 8
3060 LET V(C)=2*C+4:REM CALC POS Y
3070 NEXT C
3075 X=1:Y=1:REM POSICION INICIAL CURSOR
3080 H1=X:H2=X+4:V1=Y:V2=Y+7
3090 REM ***** DIM MATRICES HOJA *****
3100 DIM M(15,15):DIM N(15,15)
3110 FOR I=1 TO 15:FOR J=1 TO 15
3120 LET M(I,J)=I*J+I
3130 NEXT J,I
3140 DIM F$(225)
3150 RETURN
```

Amstrad CPC 464/664:

```
100 GOSUB 3000:REM PREPARA MATRICES Y VARIABLES
110 GOSUB 1000:REM IMPRIME PANTALLA
120 GOSUB 1700:REM IMPRIME DATOS EN PANTALLA
130 GOTO 1100
999 STOP
1000 REM IMPRIME VISUALIZACION PANTALLA
1005 CLS:PRINT "                                C O L U M N A S"
1006 PRINT
1007 PRINT "FILA              1.    2.    3.    4.    5."
1010 PRINT "             +-----+-----+-----+-----+-----+"
1020 FOR C=1 TO 7
1030 PRINT " "CHR$(C+64);".  |     |     |     |     |     |"
1040 PRINT " -------+-----+-----+-----+-----+-----+"
1050 NEXT C
1060 PRINT " "CHR$(C+64);".  |     |     |     |     |     |"
1070 PRINT " -------+-----+-----+-----+-----+-----+"
1080 RETURN
1100 LOCATE 1,1:P$=CHR$(Y+64)+MID$(STR$(X),2,2):PRINT
     "CELDA "; P$;" "
1110 A$=INKEY$:IF A$="" THEN 1110
1120 IF A$=CHR$(243) THEN 1200:REM MUEVE DERECHA
1130 IF A$=CHR$(242) THEN 1300:REM MUEVE IZQUIERDA
1140 IF A$=CHR$(241) THEN 1400:REM MUEVE ABAJO
1150 IF A$=CHR$(240) THEN 1500:REM MUEVE ARRIBA
1155 IF A$="F" THEN GOSUB 2000:REM ENTRA FORMULA
1160 IF A$="C" THEN GOSUB 2300:REM CALCULA HOJA
1165 IF A$=CHR$(13) THEN RETURN
1170 IF A$>="0" AND A$<"9" THEN GOSUB 2100:REM ENTRA
     DATOS NUMERICOS
1180 IF A$="B" THEN GOSUB 5000:REM BORRA HOJA
1185 IF A$="V" THEN GOSUB 5100:REM TOMA HOJA ANTERIOR
1187 IF A$="G" THEN 5200:REM MUEVE CURSOR A NUEVA
     CELDA
1188 IF A$="R" THEN GOSUB 5700
1190 GOTO 1100
1200 REM ***** MUEVE DERECHA *****
1210 IF X=15 THEN 1100
1220 IF X=H2 THEN GOSUB 1600:X=X+1:H1=
     H1+1:H2=H2+1:GOTO 1270
1230 GOSUB 1600:LET X=X+1:GOSUB 1650:GOTO 1100
1270 GOSUB 1800:GOSUB 1700:GOTO 1100
1300 REM ***** MUEVE IZQUIERDA *****
1310 IF X=1 THEN 1100
1320 IF X=H1 THEN GOSUB 1600:X=X-1:H1=
     H1-1:H2=H2-1:GOTO 1370
1330 GOSUB 1600:LET X=X-1:GOSUB 1650:GOTO 1100
1370 GOSUB 1800:GOSUB 1700:GOTO 1100
1400 REM ***** MUEVE ABAJO *****
1410 IF Y=15 THEN 1100
1420 IF Y=V2 THEN GOSUB 1600:Y=Y+1:V1=
     V1+1:V2=V2+1:GOTO 1470
1430 GOSUB 1600:LET Y=Y+1:GOSUB 1650:GOTO 1100
1470 GOSUB 1850:GOSUB 1700:GOTO 1100
1500 REM ***** MUEVE ABAJO *****
1510 IF Y=1 THEN 1100
1520 IF Y=V1 THEN GOSUB 1600:Y=Y-1:V1=
     V1-1:V2=V2-1:GOTO 1570
1530 GOSUB 1600:LET Y=Y-1:GOSUB 1650:GOTO 1100
1570 GOSUB 1850:GOSUB 1700:GOTO 1100
1600 REM **** APAGA CURSOR ****
1610 LET P$=MID$(STR$(M(Y,X)),2)
1615 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 1615
1620 LOCATE H(X-H1+1)-1,V(Y-V1+1):PRINT P$
1630 RETURN
1650 REM *** ENCIENDE CURSOR ***
1660 LET P$=MID$(STR$(M(Y,X)),2)
1665 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 1665
1670 LOCATE H(X-H1+1)-1,V(Y-V1+1):PRINT
     CHR$(24);P$;CHR$(24);
1680 GOSUB 1900:RETURN
1700 REM **** IMPRIME DATOS EN HOJA ****
1710 FOR I=0 TO 7
1720 LOCATE 10,V(I+1)
1730 FOR J=0 TO 4
1735 LET P$=MID$(STR$(M(I+V1,J+H1)),2)
1740 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 1740
1745 LOCATE H(J+1)-1,V(I+1):PRINT P$;
1750 NEXT J,I
1760 GOSUB 1650:REM ENCIENDE CURSOR
1770 RETURN
1800 REM ***** IMPRIME NUMS COLUMNAS *****
1810 LOCATE 1,3:PRINT "FILA    ";;
1820 LOCATE 7,3:FOR I=H1 TO H2:PRINT TAB(H(I-H1+1)
     -3)I;CHR$(8);".";
1830 NEXT I
1840 RETURN
1850 REM ***** IMPRIME LETRAS FILAS *****
1860 LOCATE 1,4
1870 FOR I=V1 TO V2
1875 PRINT
1880 PRINT " ";CHR$(I+64);"."
1890 NEXT I
1895 RETURN
1900 REM ***** FORMULA DE CELDA ACTUAL *****
1920 LET D$=F$((Y-1)*15+X)
1930 LOCATE 1,22
1940 PRINT "FORMULA:
                "
1950 PRINT '

FORMULA: ";D$
1960 LOCATE 1,1
1970 RETURN
```

Rutinas preparación matrices:

```
3000 REM ******************************************
3001 REM *          PREPARA MATRICES Y VARIABLES          *
3002 REM ******************************************
3010 DIM H(5),V(8),ST(20),ST$(20),E$(20),G$(20),C(20)
3020 FOR C=0 TO 4
3030 H(C+1)=6*C+11:REM CALC POS X
3040 NEXT C
3050 FOR C=1 TO 8
3060 LET V(C)=2*C+3:REM CALC POS Y
3070 NEXT C
3075 LET X=1:Y=1
3080 LET H1=X:H2=X+4:V1=Y:V2=Y+7
3090 REM ******************************************
3091 REM *                MATRIZ VALOR                *
3092 REM ******************************************
3100 DIM M(15,15):DIM N(15,15)
3110 FOR I=1 TO 15
3120 FOR J=1 TO 15
3130 LET M(I,J)=I*J
3140 NEXT J,I
3150 REM *               MATRIZ FORMULA               *
3152 REM ******************************************
3160 DIM F$(255)
3170 LET F$(1)="A1+B1+C1"
3180 LET F$(31)="C1+C2+C3"
3190 LET F$(16)="B1+B2+B3"
3200 RETURN
```

Programa objeto

Proporcionamos las rutinas para manipular los objetos en nuestro juego «Dog and Bucket»

Nuestro árbol de manipulación de objetos se puede programar como se indica, añadiendo las correspondientes líneas a los listados centrales ya proporcionados. Aquí las líneas clave son las 210 y 220, 2430 y 2440, y 5030 a 5090. Examinemos cada uno de estos trozos clave de código de uno en uno.

Primero, las líneas 210 y 220 preparan las matrices necesarias para almacenar los datos del árbol. Recuerde que, a diferencia de algunos de los árboles anteriores que examinamos en esta serie, éste comprueba muchas condiciones diferentes, con independencia del nivel de descenso o del número de nudo. En consecuencia, necesitamos almacenar para cada nudo un registro del valor condicional que está probando y los nudos a los que conducirá, según que las condiciones resulten verdaderas o falsas. La matriz c retiene los distintos valores condicionales, y cada nudo comprueba un elemento de esa matriz. El número del elemento a comprobar se lee en la matriz k(número de árboles, número máx. de nudos de elección).

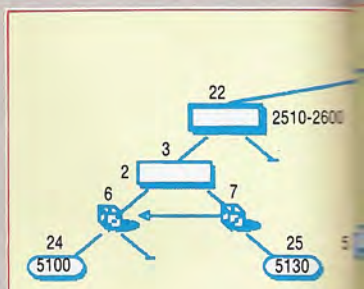
Las líneas 2430 y 2440 inicializan la matriz c. Estas líneas constituyen una subrutina que se debe llamar para cada personaje, dado que obviamente el valor de las condiciones variará para cada caso.

Entre las líneas 5030 y 5060 recorremos el árbol. La línea 5040 comprueba el número de nudo actual, y si se trata de un nudo terminal (es decir, si su numeración es mayor que 21) entonces salta fuera del árbol para seleccionar la rutina de las líneas 5070 a 5090. La línea 5050 comprueba si el nudo está probando la condición 12, que indica un nudo aleatorio, y si es así, llama la rutina de número aleatorio para asignarle un valor (uno o dos) a la condición. Luego la línea 5060 lleva a cabo la parte más importante de la operación, seleccionando el nuevo número de nudo de la matriz t y volviendo a saltar luego a la línea 5040.

La ejecución del programa completo le mostrará al manipulador de personajes en plena acción. Entre S en respuesta a la indicación ¿Valores por defecto? y vea qué sucede. El editor de personajes de la línea 2350 no es totalmente compatible con el manipulador de personajes tal como está. Esta aplicación la analizaremos con mayor detalle en el próximo capítulo. Usted puede cambiar de escenario pulsado 1, 2 o 3.

En esta etapa encontrará que la acción es más bien repetitiva, pero esta situación cambiará en seguida cuando agreguemos las dos rutinas finales (las rutinas de *interacción* y de *trazado*) en los capítulos venideros.

Al módulo de inicialización se le deben añadir las siguientes líneas. Las funciones son útiles para manipular los datos retenidos en las dos matrices de serie principales. La línea 190 prepara una matriz, *t*, que se utilizará para almacenar los datos de nuestras tres principales estructuras arborescentes: el árbol de «objetos» (en este capítulo), el árbol de *interacción* y el árbol de *trazado* (de los que hablaremos en el próximo capítulo). La matriz *k* retiene las distintas condiciones que se deben comprobar en los diversos nudos, y la matriz *c* retiene los valores condicionales (que se inicializan en las líneas 2430 y 2440)




```
130 DEF FNb(y,z)=VAL(b$(y,z))
140 DEF FNc(y,z)=VAL(c$(y,z))
150 DEF FNm$(c$,d)=STR$((VAL(c$))-d)
160 DEF FNi$=b$(VAL(c$(c,3)),1)
180 REM prepara árboles
190 DIM t(3,25,2),k(3,30),c(25)
200 REM árbol objeto
210 FOR n=1 TO 21:REM 21 nudos de elección
220 READ k(1,n),t(1,n,2),t(1,n,1):NEXT n
```

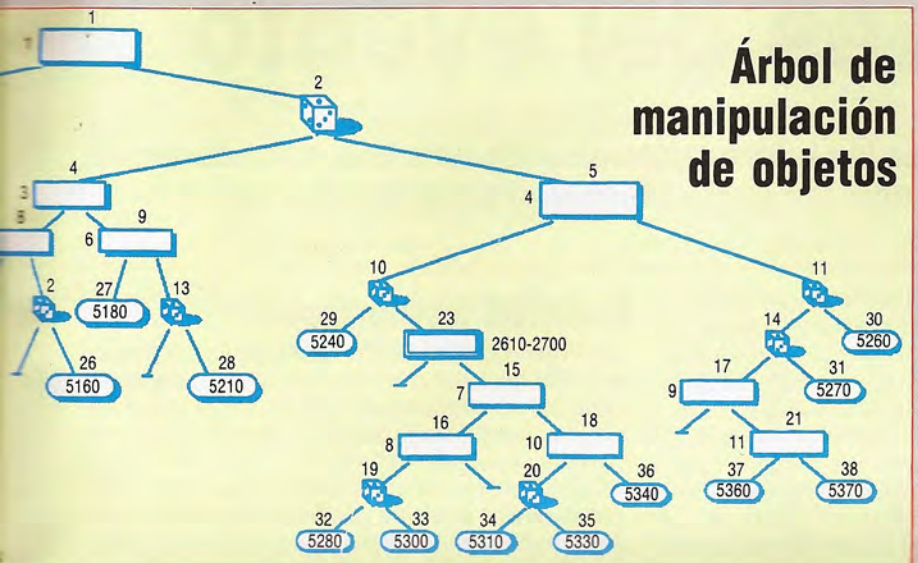
Necesitamos ajustar nuestro bucle principal del programa para tener en cuenta la primera parte de nuestro manipulador de personajes. Las líneas 550 a 800 toman cada personaje de a uno, comprueban si la bandera *manipular* es mayor que cero (línea 560) y, si lo es, la reducen en uno y prosiguen al siguiente personaje. Si el valor es cero, entonces se restablece el factor de manipulación de personaje (c$(n,10)) mediante la lectura de los datos en las líneas 6030 y 6040. La línea 580 comprueba si el valor de manipulación por defecto es cero, en cuyo caso el manipulador no procesará en absoluto al personaje. La línea 590 inicializa las condiciones para el árbol llamando la subrutina de 2430, y después llama el manipulador de la línea 5000

```
500 REM
510 REM prueba bucle programa
520 REM
530 GOSUB 2100: GOSUB 2150: GOSUB 2240:
    PRINT:PRINT
540 REM manipulador de personajes
550 FOR c=1 TO 6
560 IF FNc(c,10)>0 THEN c$(c,10)=FNm$(c$(c,10),1):
    GOTO 800
570 RESTORE:FOR n=1 TO c×10+c-1:READ
    c$(c,10):NEXT n: REM restablece valor manipulación
    por defecto
580 IF FNc(c,10)=0 THEN GOTO 800
590 GOSUB 2430: GOSUB 5000: REM llama arbol objetos
800 NEXT c
810 GOSUB 4260: IF i$="" GOTO 550
820 GOSUB 2040: GOTO 530
```

Las líneas 2430 y 2440 le asignan a la matriz *c* los diversos valores condicionales que se comprobarán durante el recorrido del árbol. En diversos puntos el manipulador de personajes llama las subrutinas de las líneas 2520, 2620 y 2720

```
2400 REM
2410 REM condiciones
2420 REM
2430 h=FNc(c,8): i=FNc(c,3): j=FNc(c,6): c(1)=ABS(i>
     0): c(2)=ABS((FNb(j,2)=FNc(c,2)) AND (q=1)):
     c(3)=ABS(b$(i,3)="y"): c(4)=ABS
     (FNc(c,3)=FNc(c,6)): c(5)=ABS(b$(i,4)="y")
2440 c(6)=ABS(i=3): c(7)=ABS(FNc(c,5)> 5):
     c(8)=ABS(FNc(c,5)> 2):
     c(9)=ABS(VAL(c$(c,9))=1):
     c(10)=ABS(FNc(x,3)=0):
     c(11)=ABS(FNc(h,2)=FNc(c,2)): c(12)=255
2500 RETURN
2510 REM
2520 REM comprueba escenarios para objetos
2530 REM
2540 f=0:REM establece "bandera hallado" en cero
2550 FOR b=1 TO 12
2560 IF FNb(b,2)=FNc(c,2) THEN f=1:b=12
2570 NEXT b
2580 IF f=1 THEN n=3: GOTO 2600
2590 n=39
2600 RETURN
2610 REM
```

Árbol de manipulación de objetos



Caroline Clayton

Líneas de bifurcación
Vemos aquí el árbol de manipulación de objetos, con los nudos de elección numerados en azul y los nudos terminales en rojo. Los nudos 22 y 23 saltan a subrutinas que determinan el nuevo número de nudo. Cada nudo de elección también muestra (en verde) el número del elemento de la matriz *c* que retiene el valor de la condición a comprobar (ver líneas 2430 y 2440); a todos los nudos aleatorios se les ha asignado el elemento 12. Los nudos terminales están etiquetados con el número de línea de la subrutina a la cual el programa principal le pasa el control después de recorrer el árbol. Próximamente ofreceremos complementos de estos listados para el Spectrum

```
2620 REM comprueba presencia del propietario del objeto:
     si está presente, establece x en el número del
     personaje: vuelve a saltar al árbol
2630 REM
2640 f=0:x=0
2650 FOR m=1 TO 6
2660 IF (FNc(m,2)=FNc(c,2)) AND (FNc(m,6)=FNc(c,3))
     THEN f=1: x=m: GOSUB 2430: m=6
2670 NEXT m
2680 IF f=1 THEN n=15:GOTO 2700
2690 n=39
2700 RETURN
2710 REM
2720 REM selecciona objeto al azar del escenario del
     personaje
2730 REM
2740 b=0
2750 FOR s=1 TO 12
2760 IF FNb(s,2)<>FNc(c,2) THEN GOTO 2780
2770 GOSUB 4180: IF q=1 THEN b=s: s=12
2780 NEXT s
2790 IF b=0THEN GOTO 2750
2800 RETURN
```

Aquí añadimos una rutina de numeros aleatorios para los diversos nudos de elección aleatoria, así como una corta rutina para poner a cero los códigos de c$(n,8) y c$(n,9)

```
4150 REM
4160 REM rutina número aleatorio
4170 REM
4180 q=INT(RND(1)*2)+1: RETURN
4190 REM
4200 REM pone a cero códigos personajes
4210 REM
4220 c$(c,8)="0": c$(c,9)="0": RETURN
4230 REM
4240 REM comprueba si se ha pulsado alguna tecla
4250 REM
4260 i$=INKEY$: RETURN
```

Estas líneas recorren el árbol (5030 a 5060) y seleccionan las diversas rutinas tal como lo determinan los nudos terminales (línea 5090). Observe la línea 5080, que selecciona dos subrutinas que devuelven nuevos valores de nudos y vuelve a saltar al proceso de recorrido del árbol

```
5000 REM rutinas árbol objeto
5010 p=0: REM poner a cero bandera imprimir
5020 IF FNc(c,2)=r THEN p=1
5030 n=1: REM empezar en el nudo 1
5040 IF n>21 GOTO 5070
5050 k=c(k(1,n))+: IF k(1,n)=12 THEN GO SUB 4180:
     k=q
5060 n=t(1,n,k):GOTO 5040
5070 IF n>=24 GOTO 5090
5080 ON (n-21) GOSUB 2540, 2640: GOTO 5040
5090 ON (n-23) GOTO 5100, 5130, 5160, 5180, 5210,
     5240, 5260, 5270, 5280, 5300, 5310, 5330, 5340,
     5360, 5370, 5430
5100 GOSUB 2740: c$(c,3)=STR$(b)
5110 IF p=1 THEN PRINT c$(c,1);" recoge";b$(b,1):
     PRINT
5120 b$(b,2)="0": c$(c,9)="4": RETURN
5130 c$(c,3)=c$(c,6)
5140 IF p=1 THEN PRINT c$(c,1);" recoge";FNi$: PRINT
5150 b$(VAL(c$(c,3)),2)="0"; c$(c,9)="4": RETURN
5160 IF p=1 THEN PRINT c$(c,1);" bebe un sorbo
     de";FNi$: PRINT
5170 c$(c,4)=FNm$(c$(c,4),-1): RETURN
5180 GOSUB 4180:IF (p=1) AND (q=1) THEN PRINT
     c$(c,1);" se está comiendo el bocadillo.":PRINT
5190 c$(c,4)=FNm$(c$(c,4),-2): c$(c,9)="6": GOSUB
     4180:IF q=1 THEN GOSUB 4220
5200 RETURN
5210 IF p=1 THEN PRINT c$(c,1);" prueba vacilando un
     bocado de la empanada, gruñe y la tira al suelo.":
     PRINT
5220 g=c: REM establece bandera comida empanada
5230 c$(c,3)="0": c$(c,4)=FNm$(c$(c,4),10):
     b$(3,2)=c$(c,2): RETURN
5240 IF p=1 THEN PRINT c$(c,1);" suelta";FNi$: PRINT
5250 b$(VAL(c$(c,3)),2)=c$(c,2): c$(c,3)="0":
     RETURN
5260 c$(c,5)=FNm$(c$(c,5),-1): RETURN
5270 GOSUB 5240: RETURN
5280 IF p=1 THEN PRINT c$(c,1);"
     arroja";b$(VAL(c$(c,3)),1);" a "; c$(x,1): PRINT
5290 c$(x,4)=FNm$(c$(x,4),1):b$(VAL(c$(c,3)),2)
     =c$(c,2):c$(x,8)=STR$(c): c$(x,9)="5":
     c$(c,3)="0":RETURN
5300 GOSUB 4220: RETURN
5310 IF p=1 THEN PRINT "Creo que he tomado tu bebida,
     le dice ";c$(c,1);" a ";c$(x,1): PRINT
5320 c$(c,8)=STR$(x): c$(c,9)="2": RETURN
5330 c$(c,4)=FNm$(c$(c,4),2): RETURN
5340 IF p=1 THEN PRINT c$(c,1);" le da"; FNi$;" a
     ";c$(x,1): PRINT
5350 c$(x,3)=c$(c,3): c$(c,3)="0": c$(x,8)=STR$(c):
     c$(x,9)="1": RETURN
5360 GOSUB 4220: RETURN
5370 IF p=0 GOTO 5420
5380 IF p=1 THEN PRINT c$(c,1);" le agradece
     ebriamente a ";c$(VAL (c$(c,8)),1);" que le haya
     devuelto su bebida"
5390 GOSUB 5220: RETURN
5400 RETURN
```

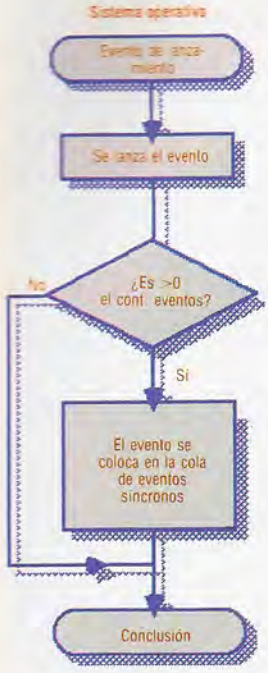
La línea 6230 retiene los datos para el árbol de manipulación de objetos. La línea 220 busca los valores en grupos de tres, asignándolos a cada nudo por turno: el subíndice del elemento de la matriz *c* que retiene la condición a comprobar, el número del nudo al cual debe bifurcar si la condición es verdadera, y el nudo al cual debe saltar si la condición es falsa

```
6200 REM
6210 REM datos árbol objeto
6220 REM
6230 DATA 1, 2, 22, 12, 5, 4, 2, 7, 6, 3, 9, 8, 4, 11, 10,
     12, 39, 24, 12, 6, 25, 5, 12, 39, 6, 13, 27, 12, 23,
     29, 12, 30, 14, 12, 26, 39, 12, 28, 39, 12, 31, 17,
     7, 18, 16, 8, 39, 19, 9, 21, 39, 10, 36, 20, 12, 33,
     32, 12, 35, 34, 11, 38, 37
```

Después del evento

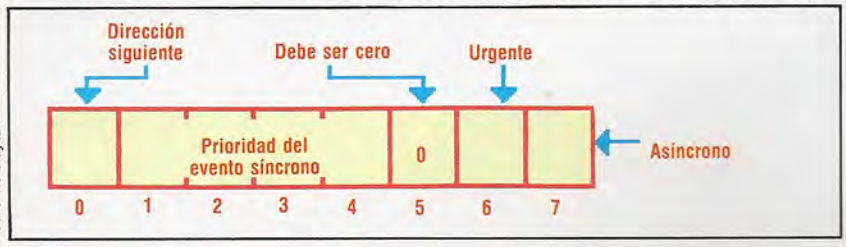
Una vez analizadas las interrupciones y los eventos, ahondaremos en la utilidad que aportan estos últimos al programador en lenguaje máquina

Sincronicidad
Estos dos diagramas de flujo muestran la sucesión de operaciones que realizan el sistema operativo (diagrama superior) y el programa principal (inferior) para procesar los eventos síncronos



Ya hemos visto cómo el OS del Amstrad accede a los eventos de software a través de un *bloque de eventos*, que consiste en siete bytes contiguos situados en cualquier sitio dentro del bloque central de 32 K de la RAM. Los bloques de eventos son establecidos por el usuario llamando a la rutina KL__INIT__EVENT en $BCEF, siempre que se hayan reservado previamente los siete bytes necesarios. La rutina es llamada contando con que el registro pareja HL contiene la dirección del bloque, B contiene el tipo de evento (en forma de bits significativos, como se muestra más abajo), C contiene la ROM seleccionada (0 si es en RAM) y DE contiene la dirección de la rutina que ha de ser llamada. El listado que proporcionamos es un ejemplo de esta operación.

En el diagrama puede verse la clase de evento tal y como se pasa al registro B. Una vez inicializado un evento, el momento exacto en que es llamada la rutina por el sistema operativo dependerá del tipo (síncrono, asíncrono) y de la prioridad (normal, urgente) del evento. Generalmente la rutina que ha de llamarse puede encontrarse en cualquier lugar



Caroline Clayton

de la RAM o en cualquier ROM y la dirección del campo del usuario en el bloque de eventos se pasa a la rutina del evento para su propio uso.

La manera como maneja el sistema operativo los eventos síncronos y asíncronos es muy distinta, por lo que será mejor que las describamos por separado (aunque ambos tipos, como es obvio, pueden emplearse para cualquier aplicación dada).

Los *puntapiés* o incrementos del contador de eventos pueden venir de una de estas cuatro fuentes distintas: la interrupción rápida de reloj, la interrupción de reloj, la interrupción de retorno de cuadro o la entrada de bloque de saltos KL__EVENT. Las tres interrupciones de temporizador tienen cada una una lista asociada de eventos que necesitan ser lanzados cuando ocurre la interrupción. Las rutinas para establecer las listas son llamadas a través de las entradas del bloque de saltos. Se puede inicializar un nuevo bloque de eventos y añadirlo a una lista, o añadir un bloque preexistente a cualquiera de las listas. Los bloques de eventos pueden ser establecidos e inicializados por separado mediante KL__INIT__EVENT. La rutina KL__EVENT es de uso general y también puede servir para lanzar un evento.

Eventos asíncronos

Asociada con los eventos *normales* encontramos una cola pendiente de eventos de interrupción. Esta cola se emplea para que contenga todos los eventos que han sido lanzados durante una interrupción externa.

Consideremos como ejemplo un bloque de eventos asíncronos normales que ha sido dispuesto para ser lanzado por la interrupción rápida de reloj. Cuando ocurre la interrupción rápida de reloj, el sistema operativo busca los eventos que han de lanzarse. En este caso, el evento será lanzado. Si, tras el lanzamiento, el contador de eventos es mayor que cero, el evento no será tratado inmediatamente, sino que será colocado en la cola de espera de eventos de interrupción. Una vez que el sistema operativo ha realizado todas las tareas relacionadas con la interrupción rápida de reloj, se irán llamando una a una las rutinas que están en la cola de espera de eventos de interrupción. Dado que la interrupción rápida de reloj es reactivada antes de llamar las rutinas de eventos, se anotará todo lanzamiento ulterior. Por ello, la rutina puede emplear el tiempo que sea sin que pierda ninguno de los lanzamientos posteriores. El contador se decrementa una vez llamada la rutina del evento.

Si se ha lanzado un evento asíncrono *urgente* y el contador es mayor que cero, éste no se coloca entonces en la cola pendiente de eventos sino que su rutina de eventos es llamada inmediatamente mientras son desactivadas las interrupciones. Sin embargo, si la rutina de eventos es demasiado larga, entonces se perderá cualquier interrupción externa posterior (por lo tanto, la rutina ha de ser tan corta como se pueda). Por lo general, este tipo de evento no se utiliza.

Eventos síncronos

El programa principal decide cuándo han de procesarse los eventos síncronos. Por esto, el sistema operativo lanza sencillamente el evento y, si es mayor que cero, lo pone en la cola pendiente de eventos síncronos según la prioridad que se le asigne en el bloque de eventos. Los eventos síncronos urgentes se sitúan antes que los de tipo normal.

El sistema operativo proporciona varias entradas del bloque de saltos al programa principal para procesar los eventos síncronos, para limpiar la cola si es el caso y para impedir que ocurran eventos particulares. Además, si es necesario, se pueden desactivar todos los eventos síncronos normales.

Cuando el programa principal decide procesar los eventos síncronos, éste efectúa las tareas representadas en los diagramas de flujo. Primeramente se llama la entrada del bloque de saltos KL__NEXT__SYNC para obtener el siguiente evento de turno en

la cola. El evento es entonces procesado mediante la llamada a KL__DO__SYNC; esta rutina busca la dirección de la rutina de eventos en el bloque de eventos y la llama. Entonces se llama la entrada KL__DONE__SYNC para indicar al sistema operativo que ha concluido el procesamiento correspondiente a ese evento. En este punto es decrementado el contador de eventos, y si éste sigue siendo mayor que cero, el bloque de eventos vuelve a ser colocado en la cola de eventos síncronos.

Desactivación de eventos

En un determinado momento puede que sea necesario desactivar o evitar que ocurran determinados eventos particulares. El sistema operativo permite esto con numerosas entradas del bloque de saltos. Para eventos asíncronos, la entrada KL__DISARM__EVENT pone a un valor negativo el contador de eventos relativo a un bloque de eventos dado, y por lo tanto impide que cualquier lanzamiento ulterior llame a la rutina de eventos.

Tres son las entradas que permiten desactivar los eventos asíncronos.

La primera entrada, KL__SYNC__RESET, limpia todos los eventos pendientes de la cola de espera de eventos síncronos pero no altera el contador de eventos. Esto desactiva efectivamente el evento, dado que el contador de eventos no puede decrementarse a menos que el evento esté en la cola. Otra entrada, KL__DEL__SYNCHRONOUS, desactiva y elimina todo evento que pueda encontrarse en la cola.

La última entrada, KL__EVENT__DISABLE, detiene toda formulación de los eventos síncronos normales que se haya colocado en la cola, pero todavía permite que se lancen los eventos.

Los eventos pueden parecer complicados al principio, pero protegen al programador de las complicaciones que normalmente acompañan las interrupciones, aunque se ha de tener mucho cuidado cuando se emplean los eventos asíncronos urgentes para que no interfieran los datos con el programa principal.

Zumbador de fondo

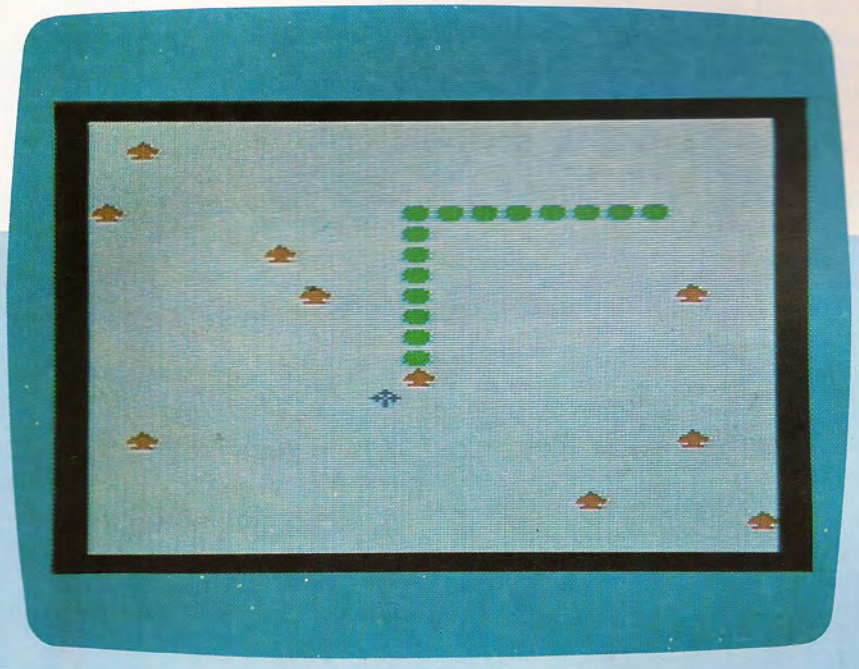
Este listado es un ejemplo de cómo se emplean los eventos, utilizando las técnicas explicadas en el texto. Se establece un evento de reloj y se utiliza como temporizador, lo que hace que un evento cuente los segundos. Éste a su vez establece un evento que obliga al ordenador a emitir un zumbido una vez por segundo. El programa ha de llamarse a su dirección assembly para inicializar los eventos.

Obsérvese que el zumbido continúa de fondo cuando se está ejecutando un programa en BASIC (aunque puede que sucedan extraños efectos si el mismo programa en BASIC emplea a su vez el generador de sonido)

```
;Este programa demuestra el empleo de varios
;tipos de eventos para generar un reloj sencillo que
;emite un sonido cada segundo
;
INIT.EVENT:     EQU  £BCEF ; KL__INIT__EVENT
ADD.TICK:       EQU  £BCE9 ; KL__ADD__TICKER
KICK:           EQU  £BCF2 ; KL__EVENT
TXT.OUT         EQU  £BB5A ; TXT__OUTPUT
;
BLEEP:          EQU  £07
;
;Primero se inicializan los bloques de eventos
;
                ORG  £8000
;
                LD   HL, TICK      ;inicio con reloj
                LD   B, £81        ;asincr, urgente
                LD   C, 0          ;sin selecc. rom
                LD   DE, FRAC      ;direccion de la rutina
;
                CALL INIT.EVENT
;
;Se conserva BC
;
                LD   HL, SEC.      ;contador segundos
                LD   DE, SECND     ;direccion rutina
                CALL INIT.EVENT
;
                LD   HL, BP        ;rutina sonido
                LD   DE, BEEP      ;direccion rutina
                CALL INIT.EVENT
;
;Se cubre ahora la Rutina Reloj
;
                LD   HL, FRBLK     ;direccion bloque reloj
                LD   DE, 1         ;contador
                LD   BC, 1         ;recarga
                CALL ADD.TICK      ;lo registra
;
                RET
;
;Rutinas procesamiento eventos
;
FRAC:
;mantiene un contador de 1/50 –avo de segundo
;
                LD   HL, SEC50     ;apunta al contador
                LD   B, 50         ;un segundo?
                CALL TEST
                RET  NZ            ;si no es asi, concluir
                LD   HL, SEC       ;lanza siguiente segundo
                CALL KICK
                RET
;
SECND:
;Es llamada una vez por segundo – lanza un zumbido
                LD   HL, BP        ;lanza evento zumbido
                CALL KICK
                RET
;
BEEP:
;hace zumbido
                LD   A, BLEEP      ;envia un caracter sonido
                CALL TXT.OUT
                RET
;
TEST:
;comprueba contador
;B contiene valor necesario en entrada
;HL apunta a posicion almacenamiento contador
;
                INC  (HL)          ;lo actualiza
                LP   A, (HL)       ;lee contador
                CP   B             ;conclusion
;
                RET  NZ            ;no, retorno
                XOR  A             ;si verdadero pone cero
                LD   (HL), A       ;restablece contador
                RET                ;y retrocede
;
;ahora los bloques de eventos y reloj
;
FRBLK:          DEFS 6             ;espacio bloque reloj
;
TICK:           DEFS 7             ;bloque evento reloj
;
SEC:            DEFS 7             ;bloque evento 1 seg
;
BP:             DEFS 7             ;bloque evento zumbido
;
SEC50:          DEFB 0             ;contador 1/50 seg
```

Ciempiés

Esta versión de este conocido juego ha sido escrita en BASIC para el microordenador Commodore Vic 20



Trate de dirigir su ciempiés robot durante el mayor tiempo posible. Él ha de alimentarse con las flores azules (las flores rojas están envenenadas) sin salirse nunca del cuadro ni recortar su propio cuerpo. La dificultad reside en que su longitud aumenta una unidad después de cada comida, lo cual hace que los desplazamientos sean cada vez más comprometidos.

```
 10 REM *************
 20 REM * CIEMPIES *
 30 REM *************
 50 GOSUB 1000
100 GET X$
110 D1=(X$="A")-(X$="S")+22*((X$="W")-(X$="Z"))
120 IF D1<>0 THEN D0=D1
125 IF FL=1 THEN FL=0:GOTO 170
130 POKE A(0),CN
140 FOR I=0 TO L
150 A(I)=A(I+1)
160 NEXT I
170 A(L)=A(L-1)+D0
180 C=PEEK(A(L))
190 IF C=CB THEN 300
200 IF C<>32 AND C<>42THEN 500
210 POKE A(L),MP
220 POKE A(L)+M,MC
230 GOTO 100
300 GOSUB 2000
320 POKE A(L),MP
330 POKE A(L)+M,MC
340 L=L+1
350 FL=1
360 GOTO 100
500 PRINT:PRINT:PRINT
505 GOSUB 700
510 PRINT TAB(5)"PUNTUACION:"L*10-70
520 PRINT:PRINT:PRINT
530 PRINT TAB(5)" OTRA ?"
540 GET X$
550 IF X$<>"" THEN 540
560 GET X$
570 IF X$="" THEN 560
580 IF X$<>"N" THEN RUN
590 PRINT CHR$(147);
600 END
700 FOR I=1 TO 6
710 POKE A(L),CN
730 FOR J=1 TO 200
740 NEXT J
750 POKE A(L),42
760 POKE A(L)+M,4
770 FOR J=1 TO 200
780 NEXT J
790 NEXT I
800 RETURN
1000 PRINT CHR$(147);
1010 GOSUB 1200
1020 GOSUB 1400
1030 GOSUB 1600
1040 GOSUB 1800
1050 GOSUB 2000
1190 RETURN
1200 CN=32
1210 CH=65
1220 HC=2
1230 BC=6
1240 DIM A(70)
1250 M=30720
1260 MP=81
1270 MC=5
1280 L=7
1290 D0=1
1300 CB=88
1390 RETURN
1400 FOR I=0 TO L
1410 A(I)=7923+I
1420 POKE A(I),MP
1430 POKE A(I)+M,MC
1440 NEXT
1450 RETURN
1600 FOR I=0 TO 21
1610 POKE 7680+I,160
1620 POKE 7680+I+M,0
1630 POKE 8164+I,160
1640 POKE 8164+I+M,0
1650 NEXT
1660 FOR I=1 TO 22
1670 POKE 7680=I*22,160
1680 POKE 7680+I*22+M,0
1690 POKE 7701+I*22,160
1700 POKE 7701+I*22+M,0
1705 NEXT
1710 FOR I=1 TO 10
1720 GOSUB 1800
1730 POKE P,CH
1740 POKE P+M,HC
1750 NEXT
1760 RETURN
1800 P=INT(RND(TI)*440)+7702
1810 IF PEEK(P)<>32 THEN 1800
1820 RETURN
2000 GOSUB 1800
2010 POKE P,CB
2020 POKE P+M,BC
2030 RETURN
```

En la pista

En esta nueva serie dedicada a los micros y los juegos de apuestas analizaremos, en primer lugar, cuatro paquetes de carácter hípico muy populares en Gran Bretaña

Si pensamos en la inmensa popularidad de los juegos de apuestas, casi no sorprende saber que la introducción de los ordenadores haya llevado a muchos programadores británicos a idear algoritmos destinados a maximizar las ganancias, minimizar las pérdidas y predecir los resultados de las distintas carreras de caballos que se realizan en Gran Bretaña. En esta serie de cuatro capítulos examinaremos algunas de las técnicas y la teoría en las que se basan las aplicaciones de apuestas por ordenador, y adelantaremos nuestra opinión acerca de las posibilidades de que un micro pueda convertir a su usuario en millonario.

Sporting Picture

Ayudar a la suerte Distintos factores inciden en el resultado de una carrera favoreciendo un enfoque anárquico y no rentable a las apuestas. El ordenador actúa como una útil herramienta para el apostador serio, reduce la inversión y maximiza la ganancia potencial al detectar combinaciones acertadas y posibilidades atrayentes

En este capítulo examinamos cuatro paquetes distintos, de gran aceptación en Gran Bretaña, cada uno de los cuales afirma ser de utilidad para el apostador. Asimismo damos una mirada a una de las principales fuentes de información para el apostador interesado en utilizar tales programas, la publicación *The Sporting Life* (que se podría traducir como «Vida de apuestas») y evaluamos los márgenes probables de beneficios (¡o pérdidas!) para cada título.

«Coursewinner»

Uno de los programas más interesantes que tienen a su disposición actualmente los apostadores británicos es el *Coursewinner* (Ganador de carreras), de Selec Software. En él se da por sentado que se dispone del *The Sporting Life* o de un periódico similar especializado en carreras. Los factores escogidos para el análisis son razonables y es probable que den una evaluación exacta.

Una característica muy grata es la capacidad de entrar hasta tres clasificaciones de velocidad tal como se publican en la prensa popular o deportiva o se obtienen de especialistas en clasificaciones como el «Timeform».

El programa, activado por menú, es fácil de usar aunque posee una detección de errores bastante limitada. Un detalle bien pensado es la provisión de un arranque en caliente sin pérdida de datos actuales, de modo que pulsar ESCAPE (en la versión BBC) de forma accidental no nos supone ningún desastre.

Una vez entrados los factores de forma correspondientes, se espera que el usuario proporcione las cotizaciones reales en oferta, porque el programa, además de aventurar las verdaderas posibilidades del caballo, intenta aconsejar si las de los corredores de apuestas representan un buen valor, en cuyo caso recomienda una apuesta «tres estrellas». Sin embargo, dado que el mercado de apuestas sólo se forma unos diez minutos antes del inicio de cada carrera, esto no siempre es posible. En cambio, puede entrarse el pronóstico de apuestas de los diarios matutinos, invalidando en consecuencia las apuestas «tres estrellas» recomendadas, pero no necesariamente las clasificaciones.

Otra opción implica el ajuste de la estimación dada para los diversos factores de forma al calcular las posibilidades de un caballo. Ésta también es una configuración útil, en particular para el apostador experimentado.

El paquete se probó con el caballo *Glorious Goodwood* en 18 carreras de diez corredores o menos. Las apuestas de valores «tres estrellas» no fueron espectaculares, con tres ganadores en 12 selecciones con una ganancia de premios pareja, descontados los impuestos, de un premio de entre £1,60 y £1.

El *Coursewinner* es un programa interesante y bien diseñado, basado en principios sólidos, que sobre la base de una muestra estadística de reconocida solvencia, parece ofrecer al apostador cauteloso la posibilidad de apostar de forma selectiva y rentable.

Obviamente, el mejor consejo para quien intente utilizar seriamente este sistema, u otro, es que lo pruebe concienzudamente antes de invertir dinero auténtico.

Coursewinner: Para el Spectrum, Commodore 64, gama Amstrad CPC, Atari, Apple II y BBC Modelo B Micro
Distribuido por: Selec Software, 37 Councillor Lane, Cheadle, Cheshire, Gran Bretaña

Usando el "Coursewinner"

Al cargar el *Coursewinner*, se presenta un menú; digitando C aparece una lista de pistas. Se digita el número de la pista en la cual se correrá la carrera a analizar y después se entra la distancia para el hipódromo más cercano y el *going* oficial (que está en la parte superior de la Racecard). El programa retorna al menú principal. Se digita I para entrar los datos de la carrera. El programa se bifurca a un submenú con cuatro rutinas para entrar los diferentes tipos de datos requeridos. Hay que pasar por cada rutina entrando los datos en la forma en que lo solicita el programa. En la sección titulada "factores de velocidad" se deben utilizar tres de estos sistemas de clasificación publicados: Stopwatch (*Sporting Life*), Form Ratings (*Sporting Life*), Spotform (*Daily Mirror*), Formcast (*Daily Mail*) y Timeform (por suscripción). De ellos, se recomiendan las clasificaciones del "Timeform". Al retornar al menú principal, hay que digitar P. Aparecerá una solicitud de cotizaciones de partida. Si no dispone de un informe de las agencias de apuestas, use el pronóstico de apuestas; pero recuerde que devaluará las apuestas "tres estrellas" recomendadas. No invalidará el cálculo hecho por el programa de las "auténticas posibilidades" de los corredores.

Cuando reaparezca el menú principal hay que digitar un análisis de la carrera. Cuanto más escasas sean las posibilidades calculadas, tanto mejores serán las probabilidades del caballo en cuestión. Es probable que valga la pena restringir las apuestas a caballos con posibilidades calculadas de 1-1 (dinero saldado) o mejores, dado que éstos ganarán el 50 % de sus carreras con algunas carreras largas como perdedor. Si los corredores de apuestas ofrecen por ellos posibilidades de 6-4 o más, tanto mejor

«Sprint formula»

Brimardon Computer Racing Service ofrece numerosos servicios al apostador británico cuidadoso que desee aprovechar mejor su micro. Éstos incluyen guías para que el usuario escriba sus propios programas, programas ya escritos que se pueden adaptar a las necesidades individuales, programas diseñados a la medida de las especificaciones del cliente y el *Sprint formula* (Fórmula de los *sprints*). Este último es un paquete dirigido al apostador experimentado que conoce el valor en *sprints* de las cifras de velocidad basadas en tiempos de carrera. Se centra en los handicaps de todas las edades hasta siete estadios (un estadio=201) porque estas carreras de notoria dificultad ofrecen apuestas abiertas. Los autores arguyen, con cierta justificación, que esto proporciona una oportunidad de oro para el apostador que posee un medio fiable de evaluar los méritos.

La cifra de velocidad del caballo, que es vital para la fórmula, se puede mejorar o devaluar según el *going* (estado de la pista), la distancia y la clase de carrera en la que se obtuvo la cifra. Además, se juzgan las cifras de forma reciente y se da una estimación en relación a indicadores tan obvios como un segundo lugar obtenido la última vez. Los autores consideran que estos corredores a menudo parten con posibilidades restringidas que no ofrecen ninguna expectativa. El objeto es tomar ganadores con mayores probabilidades.

El sistema, por cierto, hace esto para quienes estén preparados para apoyar a los dos caballos mejor clasificados. Brimardon ofrece detalles de los resultados de la temporada, que se pueden comprobar. Hasta ahora, los dos mejores han producido 28 ganadores en 123 apuestas para una ganancia de premios pareja, descontados impuestos, de 79 puntos, lo que representa un 62 % sobre el movimiento total.

Los autores admiten que, debido a limitaciones de memoria, el resultado del *Sprint formula* no es muy bueno, mientras que las rutinas de entrada son un poco chapuceras. La detección de errores es inexistente y queda en las manos del usuario evitar la entrada de datos erróneos. En compensación, el programa viene con una documentación extensa y eficaz. En conjunto, es un paquete útil para el apostador selectivo, que valorará su inteligencia y su, al parecer, feliz aproximación al problema de la verdadera rentabilidad de las apuestas.

Sprint formula: Para el BBC Modelo B, Acorn Electron, Spectrum, Commodore 64, gama Amstrad CPC, Tandy, Dragon, Oric y Atari
Distribuido por: Brimardon Computer Racing Service, 48 Pierremont Road, Darlington, Gran Bretaña

"Z5 horse race forecast"

El *Z5 horse race forecast* (Pronóstico de carreras de caballos Z5), del profesor Frank George, toma sus datos de *The Sporting Life* y utiliza cifras de velocidad del *Raceform Handicap Book*. Emplea menos datos que el *Coursewinner* 3 y acepta dos clasificaciones: la cifra de velocidad adecuada más el servicio de clasificaciones de forma de *The Sporting Life*.

La información se entra en respuesta a una serie de avisos. Se detectan los errores más obvios y, antes de entrar cada dato, se interroga acerca de su veracidad. El programa clasifica entonces a cada caballo de acuerdo a la siguiente escala: apuesta excelente, muy buena, posible, pobre, de caballo no favorito y de eliminado. No se realiza ningún intento para cuantificar estas clasificaciones, de modo que es obligación del usuario probar metódicamente el programa con el fin de establecer su validez estadística.

La desconcertante característica de este programa es la frecuencia con que se da, hasta cinco corredores de la misma carrera, las clasificaciones máximas en conjuntos de diez corredores o menos. El autor hace dos recomendaciones: o no apostar, o apoyar a todos los caballos de mejor clasificación (si es posible hacerlo conservando, así y todo, alguna ganancia). Sin embargo, el número de verdaderas posibilidades de ganar sobre esta base es limitado.

Z5 horse race forecast: Para el Spectrum, BBC Micro Modelo B y Commodore 64
Distribuido por: Bureau of Information Science, Chalfont, St. Giles, Gran Bretaña

"Hulk" 1 y 2

El *Hulk* (de Warm Boot Ltd) no está diseñado específicamente como ayuda para el apostador por ordenador. Su creador lo describe como un "ingeniero con el conocimiento de un ignorante", en el sentido de que ayuda al usuario a construirse su propio sistema experto. Si se acepta que el apostador con éxito es un experto en la materia, ¿por qué no tratar de imbuirle esa experiencia al ordenador? En consecuencia, *Hulk* es útil para el apostador experimentado a quien le guste trabajar con "sistema": unas reglas basadas en el análisis de datos pasados que, aplicados con rigor, crea ganadores en número suficiente para obtener beneficios.

La primera tarea, y la que más tiempo lleva, es la de preparar una base de datos. En la versión para el BBC Micro, las limitaciones de memoria requieren ingenio por parte del usuario, quien debe tener una idea anticipada y clara de los factores que con toda probabilidad serán relevantes, y seleccionar luego un tipo de carrera específica para el análisis. Una estrategia típica podría suponer el estudio de handicaps para una distancia dada en carreras de diez corredores o menos, concentrándose en los primeros tres o cuatro del pronóstico de apuestas. Es esencial asegurarse escoger una muestra representativa desde el punto de vista estadístico.

Los datos son analizados por un programa llamado *Look*, que permite desarrollar un conjunto de reglas (típicamente siete u ocho) que predicen una hipótesis dada (en este caso, si un caballo es o no ganador). Tras haber obtenido un conjunto de reglas satisfactorio, las mismas se pueden aplicar a acontecimientos futuros mediante otro programa, denominado *Leap*, que realiza sus predicciones desde el punto de vista del porcentaje de probabilidades.

El *Hulk* es un programa fascinante para el apostador "sistemático" dedicado al análisis estadístico como medio para hallar ganadores. Utilizándolo, es posible idear conjuntos de reglas que pronostiquen ganadores sobre una base muy selectiva, pero con un grado de exactitud que supera el 60 %.

Hulk I y II: Hulk I paara el BBC Modelo B Micro; Hulk II para máquinas dBase II, MS-DOS y PC-DOS
Distribuido por: Warm Boot Ltd, Finsbury Business Centre, 40 Bowling Green Lane, London EC1R ONE, y por Brainstorm Computer Solutions, 103a Seven Sisters Road, London, N7 7QN, Gran Bretaña

Para informarse

La Racecard da la hora, la distancia, las condiciones y el valor de la carrera seguidos por una lista de corredores. Leyendo de izquierda a derecha, los detalles para cada corredor son los siguientes: número, posición en las seis últimas carreras, nombre, propietario, entrenador, edad, peso con carga, jockey y *draw*. De vez en cuando también se encontrará información complementaria sobre penalizaciones, ganadores de pistas y distancia, y ventajas para aprendices. A la lista de corredores le siguen las "clasificaciones de cronómetro". El mejor tiempo registrado por un caballo se expresa en forma numérica simple tras haber efectuado varios cálculos teniendo en cuenta *going*, peso, etc. Es importante una buena cifra de velocidad, ya que no tiene sentido apoyar caballos lentos. El pronóstico de apuestas son las ideas del experto acerca de cómo es probable que apuesten los apostadores. Es una guía orientativa y no representa lo que ocurra en realidad cuando se forme un mercado.

El Sporting Life Form de la derecha da el perfil general del caballo, incluyendo sus posiciones durante la temporada, nombre, edad, peso con carga, color, sexo, crianza, lista de carreras ganadas (incluyendo distancia, *going* y pista), triunfos totales y triunfos de la temporada. A continuación siguen unas breves notas que describen las tres últimas carreras

The Racecard

Fourth Race

1¼m

One mile and a half, for three yrs old and upwards

4.10 THE ALYCIDON STAKES (Listed Race)

£12000 added to stakes
Distributed in accordance with Rule 194 (iii)(a)
(Includes a fourth prize)
for three yrs old and upwards which, at starting, have not won a Pattern race since two yrs old
ONE MILE AND A HALF
£24 to enter, £96 extra if declared to run

Weights: 3-y-o colts and geldings....8st 2lb; fillies....7st 13lb
4-y-o and up colts and geldings....9st 1lb; fillies....8st 12lb
Penalties, since 2-y-o, a winner of a race value £4000....3lb
Of a race value £8000....6lb
Allowances: 4-y-o and up which, at starting, have not won a race since 1983 and 3-y-o maidens at starting....6lb
' A trophy value £500, at the option of the winner, is included in the value of this race

A SS

48 entries, 31 at £24 and 17 at £120.-Closed July 17th, 1985

Owners Prize Money. Winner £7297; Second £2393; Third £1136 ; Fourth £508 (Penalty Value £9489.60)

Form		Trainer	Age	st	lb	Draw
2	**PARLIAMENT**		5	9	7	(4)
	Ch h Lord Gayle (USA) - Harbrook					
11—2231 D	**Mrs Pamela Stokes** ROYAL BLUE and WHITE stripes, RED sleeves.	(C. R. Nelson, Upper Lambourn)				P. Cook

(Breeder - Shanbally House Stud.)

Very consistent sort. Victory in listed race last time (Ayr 10f) and close up 2nd in Group 3 Westbury Stakes, (Sandown 10f) behind Elegant Air illustrates that he is not without claims.

4	**SHERNAZAR**		4	9	1	(2)
211245—	B c Busted - Sharmeen (FR) **H.H. Aga Khan** GREEN, RED epaulets.	(M. R. Stoute, Newmarket)				

(Breeder - H.H. Aga Khan.)

Had some very smart form last year, chasing home Commanche Run in the Gordon Stakes here 12f and

SPORTING LIFE FORM

KEY: a—slower than average; b—faster than average; eq—equals average; bl—blinkers; d—disqualified; ow—overweight; £—value to the winner; asterisk denotes apprentice claim, figures appearing after the jockey's name before bracket denote draw position.

2.30—NEWHOLME STAKES (2-y-o). 6f. (£1,244).

AVENTINO (8-11) ch c Cure The Blues — Sovereign Dona by Sovereign Path.

00 BROKERS DREAM (April 7, 5,400gns) (8-11) ch c Free State — Pamora by Blast.

April 27, Sandown, 5f (2-y-o) mdn, good, £2,691: 1 Teetoy(USA) (9-0, 8); 2 Cliveden(USA) (9-0, 11); 3 Sit This One Out (9-0, 10); 10 BROKERS DREAM (9-0, B Thomson, 7), in touch to half-way (33 to 1). 12 Ran. 3l, 1½l, hd, ¾l, 1l. 1m 3.19s (a 2.69s).

April 16, Newmarket, 5f (2-y-o), good, £2,532: 1 Andartis (9-0 , 11); 2 Meadow Moor (9-0 , 8); 3 For Dear Life (9-0 , 5); 9 BROKERS DREAM (9-0 , B Thomson , 7), never beyond mid-division (14 to 1 tchd 20 to 1). 11 Ran. 1½l, nk, ¾l, nk, 2½l. 1m 2.23s (a 2.43s).

3 CONQUERING HERO(USA) (May 2) (8-11) b c Storm Bird — Wave In Glory by Hoist The Flag.

Aug 3, Windsor. See ELNAWAAGI(USA).

311 ELNAWAAGI(USA) (Feb 4, $ 4,000,000) (9-6) b c Roberto(USA) — Gurkhas Band (USA) by Lurullah. **1985, 6f good (Windsor), 6f good to firm (Thirsk). £2,612.**

Aug 3, Windsor, 6f (2-y-o), good, £941: 1 ELNAWAAGI(USA) (9-4 , A Murray , 12), made virtually all, stayed on well when shaken up final furlong (5 to 4 fav op 5 to

41 FEISTY (March 13, 44,000gns) (9-3) b c Pitskelly — Golden Empress by Cavo Doro. **1985, 6f good (Yarmouth). £1,066.**

Aug 8, Yarmouth, 6f (2-y-o) mdn, good, £1,066: 1 FEISTY (9-0 , T Ives , 1), made virtually all, drew clear well over one out, comfortably (100 to 30 op 5 to 2 tchd 7 to 2); 2 Below Zero (9-0 , 5); 3 Pulham Mills (9-0 , 6). 7 Ran. 6l, ½l, ½l, 7l, ½l, 1½l. 1m 16.8s (a 4.2s).

June 17, Windsor, 5f (2-y-o), good to firm, £965: 1 Roaring Riva (8-11 , 14); 2 Dee-Jay-Ess (9-3 , 1); 3 Sitzcarraldo (8-4 , 7*, bl, 8); 4 FEISTY (8-11 , T Ives , 9), always chasing leaders, stayed on inside last (11 to 1 op 5 to 1 tchd 12 to 1); 0 PORTHMINSTER (8-11 , J Reid , bl, 11), (50 to 1). 17 Ran. Sht hd, 2l, 2½l, 2½l, ¾l. 59.9s (a 0.6s).

00 FIRST BILL (Feb 28) (8-11) ch c Nicholas Bill — Angelica by Hornbeam.

Aug 2, Goodwood, 7f (2-y-o) mdn, good, £4,675: 1 New Trojan (9-0 , 11); 2 Mashkour(USA) (9-0 , 8); 3 Bronze Opal(USA) (9-0 , 13); 8 FIRST BILL (9-0 , J Matthias , 9), well placed for over 4f, weakened (50 to 1). 15 Ran. ¾l, 3l, 3l, 1½l, ½l. 1m 31.59s (a 4.99s).

July 13, Salisbury, 7f (2-y-o) mdn, good to firm, £1,697: 1 Atig(FR) (9-0, 17); 2 Ininsky (9-0, 1); 3 Sohail(USA) (9-0, 3); 0 FIRST BILL (8-7 , G Landau , 7*, 12), speed for 4f (33 to 1 op 10 to 1). 19 Ran.

Potenciar los personajes

Centremos nuestra atención en las rutinas de manipulación de objetos que ya hemos analizado

Las líneas que añadiremos a nuestro módulo básico en el último capítulo permiten que los personajes recojan objetos y, en grado limitado, los utilicen. Antes de seguir adelante diseñando los restantes árboles de decisión para controlar la interacción de los personajes y de la trama, es interesante examinar con mayor detalle el proceso de manipulación de objetos.

Es importante comprender que si usted tiene intención de aplicar estas ideas en un juego de aventuras propio debe planificar su árbol de modo que no se impriman mensajes en la pantalla con demasiada frecuencia. Aunque considerara acertada la idea de hacer aparecer gran cantidad de mensajes, éstos se imprimen con excesiva frecuencia pueden dificultar su progreso a través del juego.

Como regla general, puede dividir las acciones de los personajes en tres niveles. El nivel superior implica acciones tales como dar o entregar un objeto y exige que se imprima un mensaje, si es que usted está presente, informándole sobre lo sucedido. El segundo nivel implica aquellas acciones que no inciden necesariamente en la trama del juego pero que, no obstante, podrían visualizar mensajes para contribuir a la creación de una cierta atmósfera. También se podría visualizar un mensaje si usted entrara Examine a un personaje o palabras con esa intención. El tercer nivel (alterar el valor de la bandera de "humor" en nuestro propio programa, p. ej.) jamás tiene como consecuencia la impresión de un mensaje.

Vale la pena tener presente esta idea de "niveles" cuando usted programa su propio juego; puede asignarle un valor a cada acción que determine su nivel y luego tener una sola subrutina para ocuparse de la impresión de mensajes. Tal rutina comprobaría primero si usted está presente, comprobaría el nivel de la acción si lo estuviera, y decidiría si imprimir o no un mensaje.

Volviendo a nuestra rutina de manipulación de objetos, intente primero suprimir la línea 5010. Ello tendrá el efecto de asegurar que se impriman mensajes para todos los personajes, independientemente si se hallan en la misma habitación que el jugador o no. Usted verá que, durante la mayor parte del tiempo, los personajes se concentran en hacerse con sus "propios" objetos y bebérselos, que es lo que deseamos si hemos de mantener un grado de realismo. No obstante, puede experimentar con esto alterando los valores por defecto de las sentencias de datos en las líneas 6030 y 6040. Por ejemplo, si altera los inventarios de Lola Fiestas, Pepe Viñas y Gina Fizz a 9, 12 y 8 respectivamente, y luego ejecuta (RUN) el programa, verá que las acciones de la sala de tertulia siguen un curso bastante diferente. Pruebe de "regular" a los otros personajes de este modo y vea lo que sucede.

Además de cambiar la línea 5010 como indicamos arriba, usted también hallará que "regular" a sus personajes le resulta más fácil saltándose las líneas 560 a 580 de modo que cada personaje se procese mediante cada llamada a la rutina manipuladora, en vez de dejarlos esperando hasta que sus banderas se reduzcan a cero.

Para hacer esto, simplemente edite la línea 550 de modo que rece:

```
550 FOR c=1 TO 6:GOTO 590
```

Ahora verá que las acciones de cada personaje se visualizan una después de otra. Recuerde, si decide que el programa principal no está actualizando a los personajes con suficiente rapidez, que siempre puede alterar los valores para las banderas "manipular" en las líneas 6030 y 6040. (Si ha olvidado exactamente a qué factor alude cada elemento de la matriz c, remítase a p. 1972.)

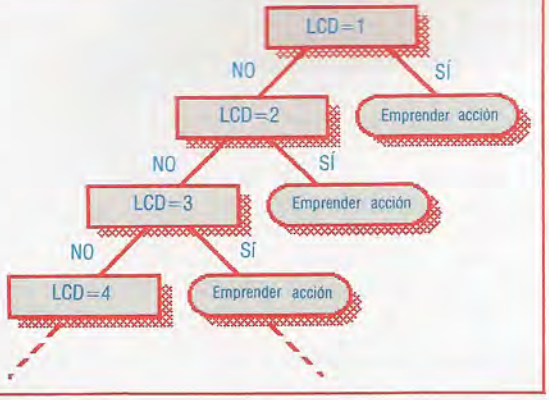
Pasemos ahora a la interacción. En la línea 190 ya hemos DIMensionado la matriz t para retener los datos de tres árboles, cada uno con hasta 25 nudos de elección. Tal vez encontremos que hay que alterar este valor, pero, aún más importante, tal vez encontremos también que la estructura de árbol que hemos utilizado para tratar con los objetos no es lo suficientemente poderosa para la interacción. Para ver por qué esto es así, observemos el árbol de objetos de 2055. Verá que una de las características de diseño fundamentales de este árbol es que cada nudo se bifurque solamente en dos únicas direcciones.

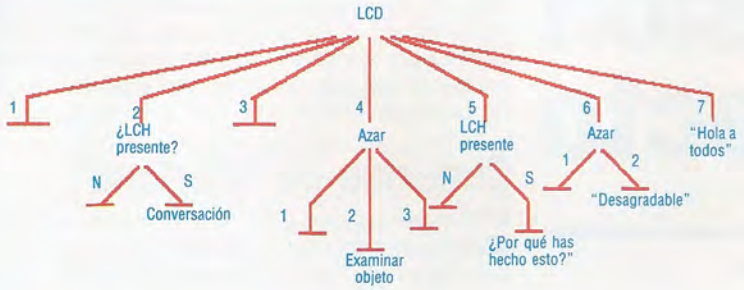
Esta configuración, obviamente, es muy conveniente, puesto que cada una de las condiciones que estamos comprobando en los nudos de elección sólo tiene dos posibles valores: falso o verdadero. Supongamos, no obstante, que las condiciones hubieran de tener más de dos posibles valores. Si quisiéramos comprobar el valor de la bandera LCD (c$(n,9)), por ejemplo, que puede tener siete posibles valores (ver p. 1974) utilizando un árbol como el de p. 2076, necesitaríamos algo como la estructura que se mostraba en el primer diagrama. Es obvio que con esto es bastante incómodo, porque sería mucho más fácil si pudiéramos recurrir a una estructura como la que anteriormente mostraba el diagrama 2.

En realidad, podemos hacerlo con bastante facilidad, empleando aún la matriz t para retener datos

Limitaciones de la bifurcación

La utilización de una estructura arborescente en la que cada nudo sólo pueda bifurcarse en dos direcciones puede imponernos serias limitaciones al comprobar las condiciones para una cantidad de valores diferentes





Opción múltiple

Este árbol muestra cómo el programa podría procesar la bandera de Código de Última Instrucción, LCC (c$(c,9)) utilizando una estructura en la cual cada nudo pueda bifurcarse en más de dos direcciones. Este método permite implementar en nuestro programa diseños arborescentes más compactos

para nuestro nuevo árbol. Nuestro primer árbol utilizaba t(n,n,1) y t(n,n,2), que retenían los números de nudo nuevos a los que habría de bifurcarse según el valor retenido en la matriz c fuera uno o dos. No obstante, podríamos utilizar t(n,n,1) para retener un número de nudo *base* y a t(n,n,2) para retener un *offset*. De modo que, por ejemplo, en el segundo diagrama el número de nudo uno tendría el valor dos leído en el elemento de la matriz t(n,1,1) y el valor de LCD menos uno leído en t(n,1,2). Aplicando este procedimiento podríamos recorrer el árbol mediante la fórmula:

Nuevo codigo = t(numero de arbol, numero de nudo actual, 2) + t(numero de arbol, numero de nudo actual,1)

Éste es el método que emplearemos para nuestro árbol de interacción. Tiene una o dos limitaciones, pero permite diseñar una estructura muchísimo más compacta para el manipulador de personajes. En el tercer diagrama se ilustra una estructura arborescente provisional empleando este procedimiento. Primero, comprueba la fortaleza de los personajes, porque si un personaje está muerto o inconsciente, es obvio que no va a hacer nada como no sea actuar como centro de atención de los demás. El programa decide luego si mover o no un personaje, comprobando (y actualizando si fuera necesario) la bandera "mover" retenida en c$(n,11). Por último, se elige uno de tres subárboles: interacción con personajes, conciencia de objeto e informes de actividad.

Complementos al BASIC

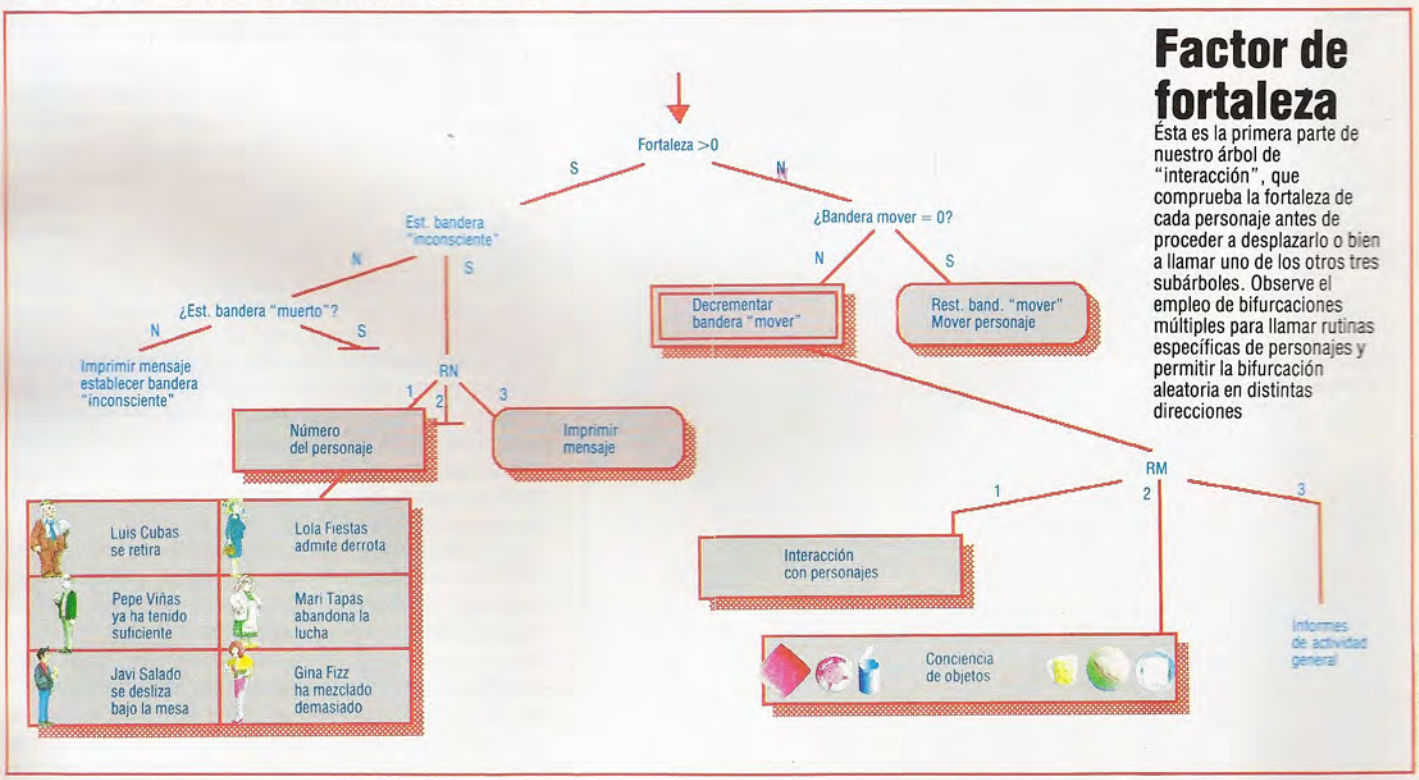
En el listado que ofrecimos en el capítulo anterior se deben introducir las siguientes modificaciones y adiciones:

Spectrum:

```
4180 q=INT(RND*2)+1:RETURN
5080 IF n=23 THEN GOSUB 2540:GOTO 5040
5085 GOSUB 2640:GOTO 5040
5090 RESTORE 9900:FOR e=1 TO (n-23):READ h:
     NEXT e:GOTO h
9900 DATA 5100,5130,5160,5180,5210,5240,
     5260,5270,5280,5300,5310,5330,5340,
     5360,5370,5430
```

BBC Micro:

```
4180 q=RND(2):RETURN
```



Factor de fortaleza

Ésta es la primera parte de nuestro árbol de "interacción", que comprueba la fortaleza de cada personaje antes de proceder a desplazarlo o bien a llamar uno de los otros tres subárboles. Observe el empleo de bifurcaciones múltiples para llamar rutinas específicas de personajes y permitir la bifurcación aleatoria en distintas direcciones

Caroline Clayton

Conclusión del proyecto

En este capítulo final realizaremos dos ampliaciones opcionales al diseño original antes de dar por terminado nuestro tester

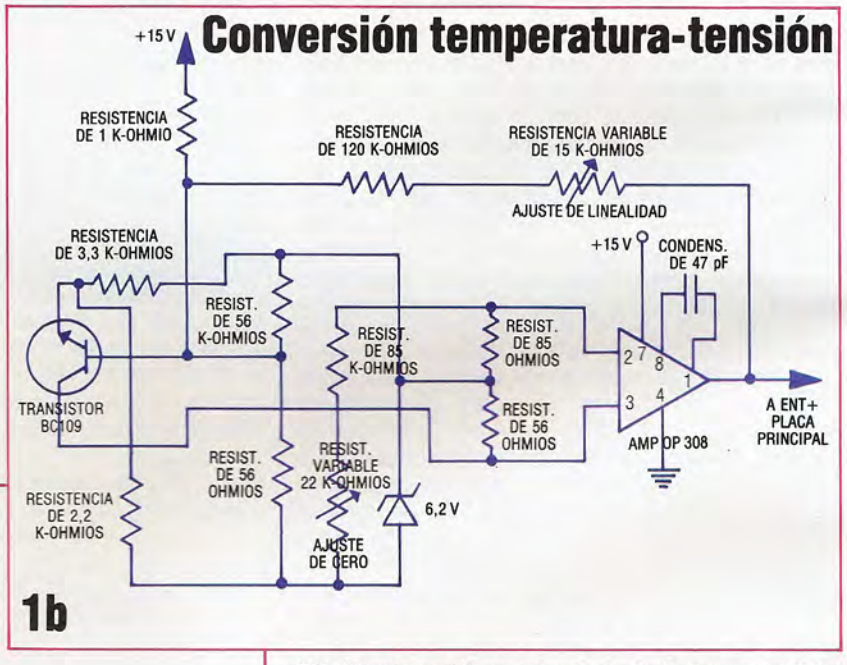
La forma más fácil de medir temperatura con un DVM es comprar una punta de prueba de temperatura ya hecha que produce una salida de tensión proporcional a la temperatura medida. Sin embargo, por lo general no son baratas, y es fácil construir dispositivos similares con un coste bastante menor. En el esquema 1 ofrecemos dos circuitos posibles. El primero, el más sencillo, produce una salida de tensión que aumenta en 10 mv por °C, pero a la salida habrá que restarle tensión constante para obtener la lectura necesaria. Si se conecta el DVM a un ordenador, no habrá ningún problema, porque la resta se puede realizar mediante software. La escala normal es de −10° a +100 °C.

Conexión en interface

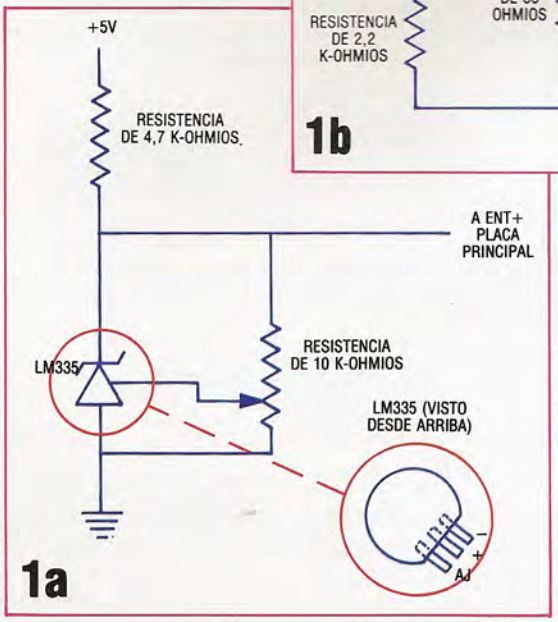
El chip convertidor A/D 7135 se ha diseñado específicamente para que la conexión en interface a un microprocesador sea lo más sencilla posible, y esto se puede hacer de numerosas maneras. En primer lugar, las salidas digitales del chip están en BCD (decimal codificado en binario). Las salidas de "dígito" separadas se vuelven verdaderas para indicar para qué dígito es válido el dato BCD. Hay una salida STROBE (de sentido negativo) que se puede utilizar para enclavar los datos BCD en un UART. El impulso STROBE se vuelve momentáneamente negativo una vez para cada dígito (cinco veces durante cada ciclo de medición). El impulso se produce en el centro del impulso TRUE de activación de cada dígito.

Si el software para lectura de datos por ordenador está escrito en BASIC u otro lenguaje de alto nivel, sería ventajoso enviar una salida desde el ordenador a la patilla RUN/HOLD del 7135 (patilla 25). Si esta patilla se toma baja, la lectura actual del 7135 se mantendrá por tiempo indefinido. Al detectar un LOW en STROBE (patilla 26), el software enviará un LOW a RUN/HOLD y puede tardar cuanto quiera en leer los datos BCD, utilizando como impulsos de reloj los impulsos de activación de dígitos (patillas 20, 19, 18, 17 y 12).

Sin embargo, la forma más simple de conectar el DVM en interface con un ordenador utiliza la salida BUSY (patilla 21). Esta señal se hace alta al principio de la fase de integración de señal y se hace baja un impulso de reloj después del fin de la integración de referencia (cuando se enclavan las salidas de datos digitales). La fase de integración de señal emplea 10 000 impulsos de reloj (y BUSY se pone bajo al final del primer impulso después de



Conversión de temperatura a tensión
Estos esquemas muestran dos circuitos alternativos para la conversión de temperatura a tensión. El circuito *b* es más sofisticado que el circuito *a*, produciendo un cambio de tensión proporcional al cambio de temperatura. Dos resistencias variables permiten establecer con precisión la linealidad y los puntos de cero



Kevin Jones

pasar por cero). Relacionando con ADN a CLOCK y BUSY y contando la cantidad de impulsos transmitidos al ordenador mediante la puerta AND, se podrá, por tanto, obtener la tensión medida simplemente restando la cantidad de 10 001 por software.

La rutina para contar los impulsos habrá de escribirse en lenguaje máquina, porque el BASIC sería demasiado lento. A una frecuencia de reloj de 125 KHz, el impulso será positivo para 4 µS.

Una alternativa aún más simple para contar los impulsos sería una rutina sencilla que esperara a que BUSY se pusidera alto y luego efectuara un bucle hasta que BUSY se pusiera bajo, sumándole uno a COUNT cada vez que se realizara el bucle. Para que esta técnica funcione, habrá de conocerse con exactitud la velocidad de la CPU del ordenador y habrá de calcularse la cantidad exacta de ciclos de máquina requeridos para cada instrucción en código máquina. Asimismo, también será necesario haber medido con exactitud la frecuencia de reloj del DVM. Esto permitirá medir la duración de BUSY con bastante precisión, restando 10 001 veces el período de CLOCK para obtener la duración de la fase de integración de referencia, y dividir este resultado por el período de CLOCK para obtener la cantidad de impulsos CLOCK en la integración de referencia. Cada impulso contado corresponde entonces a 0,0001 V.

Las señales BUSY y CLOCK del chip 7135 se pueden relacionar fácilmente con una AND utilizando un chip 74LS00, que posee cuatro puertas NAND. La figura 2 muestra las conexiones de patillas necesarias para producir la señal BUSY-.CLOCK. Este chip también se puede utilizar para proporcionar únicamente una señal de BUSY a través de un tampón si se desea adoptar el enfoque simple para medir la duración de la fase de integración de referencia que acabamos de describir.

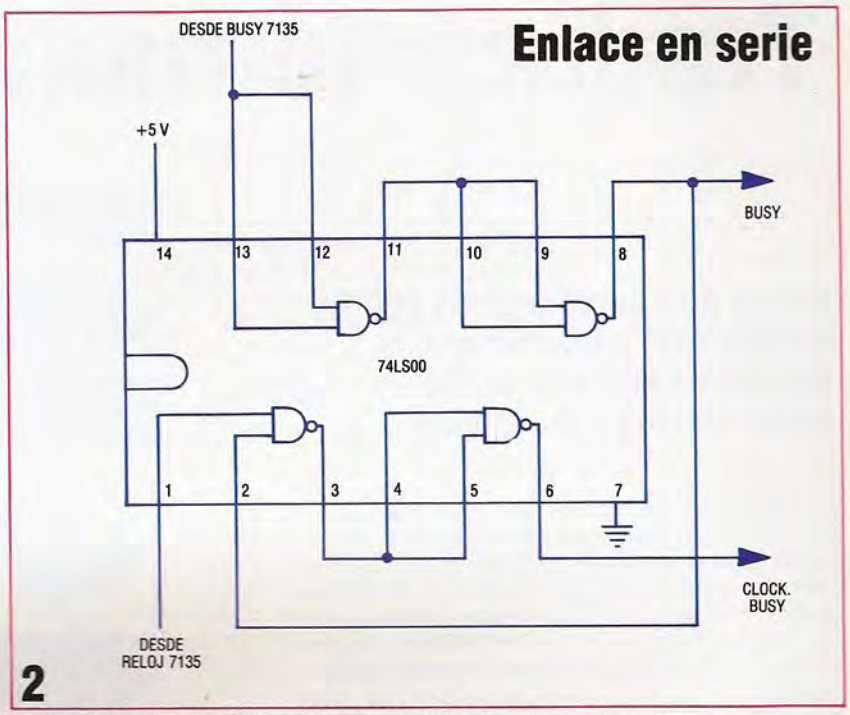
Modificaciones de diseño

Nuestro prototipo para el DVM utilizó un tipo de LED distinto para el dígito 5, con un signo "más" y "menos" separado. Debido a la dificultad para obtener un LED adecuado, modificamos el diseño para permitir el empleo de un LED de siete segmentos común para el dígito 5 (utilizando el segmento *g*, la barra cruzada, como signo menos).

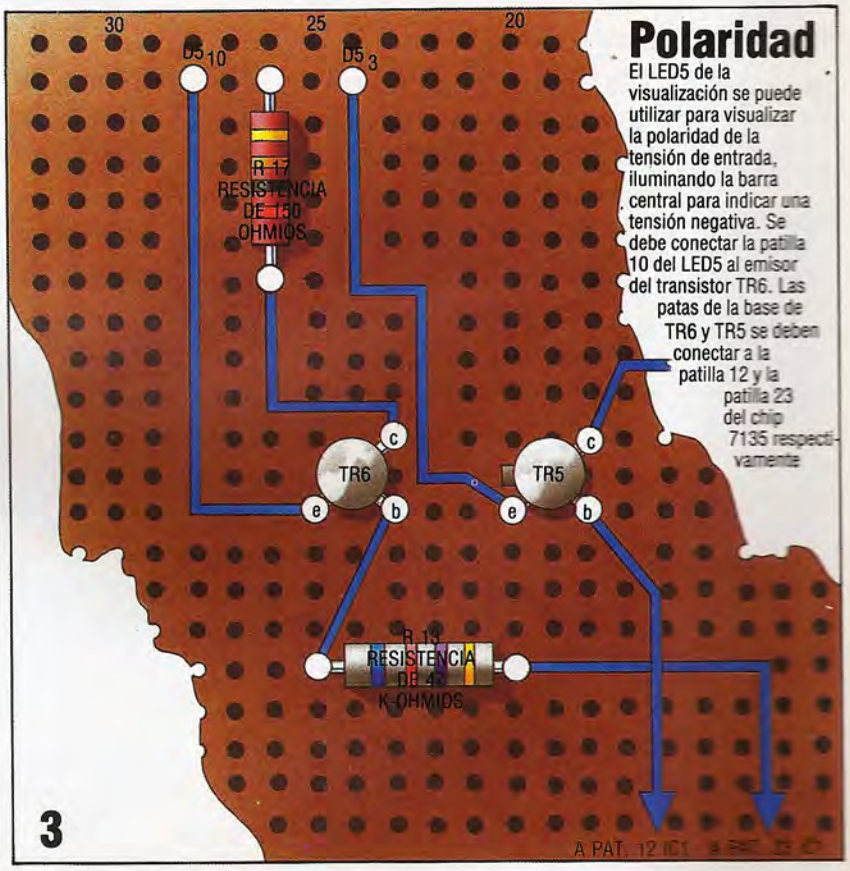
Ello supone la introducción de algunas ligeras modificaciones en el circuito publicado anteriormente. Puesto que el LED de siete segmentos tiene sólo un ánodo común para los siete segmentos, el signo menos sólo puede estar encendido cuando el dígito 5 está totalmente iluminado (en la disposición anterior había un ánodo separado para el signo). Para utilizar el LED 5 de modo que muestre un signo menos, en el diagrama de trazado correspondiente se deben efectuar las siguientes modificaciones. TR5 queda como estaba; el colector va a la fuente de +5 V y el emisor va al ánodo común del LED5 (patilla 3).

TR6 se conecta del siguiente modo: su base va a la patilla 23 (polaridad) de IC1, su emisor se conecta al cátodo del segmento *g* del LED5 (patilla 10) y su colector se conecta a la masa digital a través de una resistencia de 150 ohmios. En el diagrama de construcción, la resistencia limitadora de corriente desde el colector de TR6 a masa se lleva hasta el borde de la placa (columna 30). No olvide conectar este punto a la masa digital.

Y así termina nuestro proyecto del tester. Le hemos ofrecido los detalles completos para construir un medidor básico de 2 V de sensibilidad, junto con detalles bastante completos para numerosos accesorios y una forma sencilla de conectar el medidor en interface a cualquier ordenador. Queda en sus manos decidir el estilo de la caja para alojar la unidad.



Enlace en serie
Las señales provenientes del chip convertidor A/D 7135 se pueden conectar en interface a través de un chip 74LS00 para producir una salida CLOCK. BUSY. Conectando esta salida a un ordenador y contando por software la cantidad de impulsos recibidos, se puede calcular la entrada de tensión en el voltímetro



Traducir fórmulas

Ahora nos dedicaremos a las rutinas que permiten que el programa entre fórmulas matemáticas y las evalúe

Cuando se trabaja con una hoja electrónica, una de las principales consideraciones es que, tal como ocurre con tantas otras aplicaciones, usted y el ordenador poseen diferentes métodos de efectuar cálculos. Supongamos, por ejemplo, que entrara en la celda A5 la fórmula (A1+A2)* (A3−A4). Según el conjunto de reglas que nos han enseñado, primero debemos evaluar el contenido de cada par de paréntesis por separado y después multiplicar los resultados entre sí. Si quisiéramos que el ordenador efectuara este cálculo, habríamos de indicarle que hiciera lo siguiente:

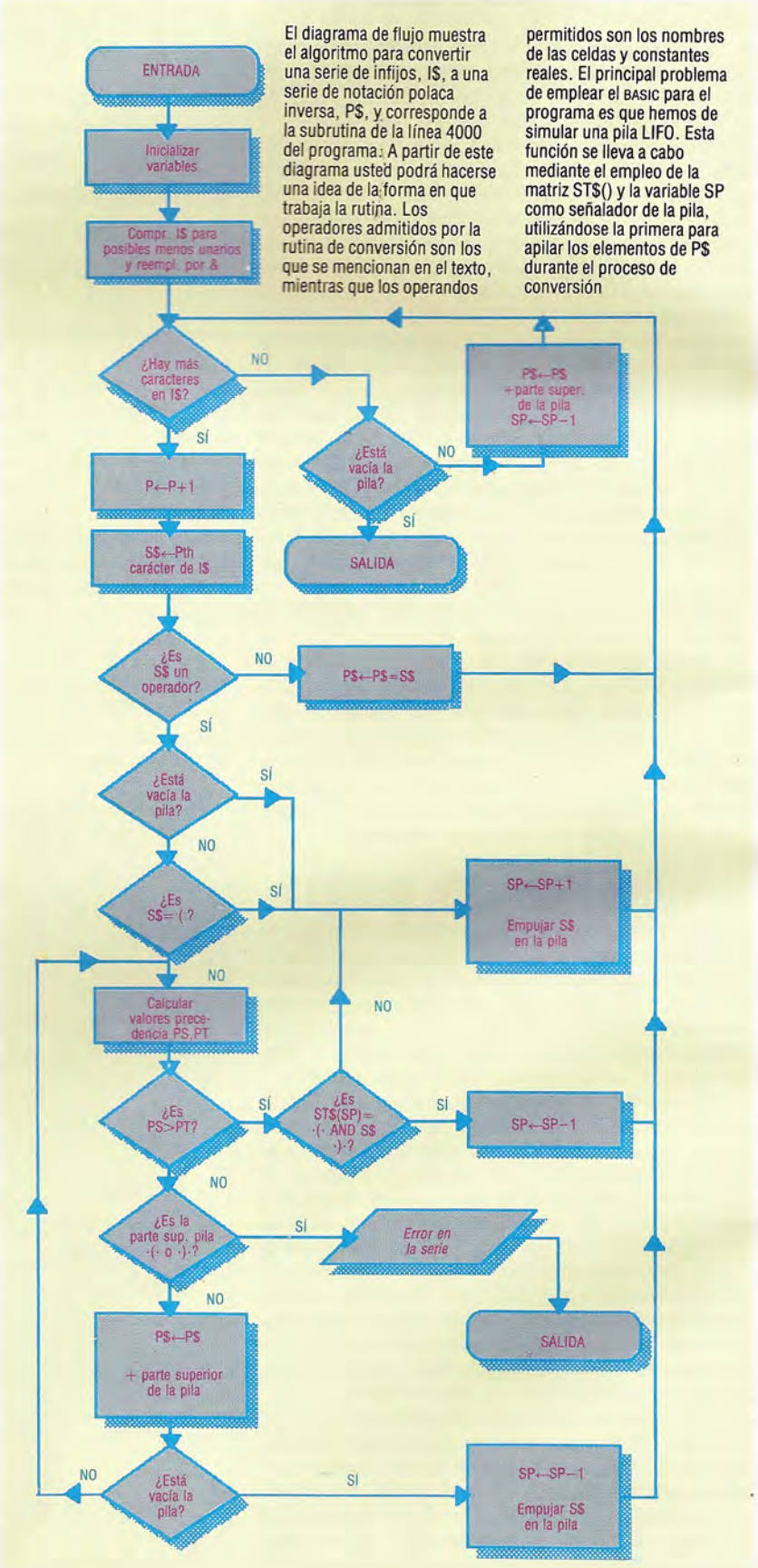
1. Tomar los valores de las celdas A1 y A2.
2. Sumarlos entre sí y almacenar el resultado temporalmente.
3. Tomar los valores de las celdas A3 y A4.
4. Restarle a A4 el valor de A3.
5. Multiplicar el resultado por el resultado de A1 más A2.
6. Almacenar el resultado en la celda A5.

Usted, probablemente, podría escribir un programa para llevar a cabo esta función, pero no sería de mucha utilidad si tuviera que evaluar muchas fórmulas difefrentes. En consecuencia, lo que necesita es una rutina que tome cualquier fórmula y la evalúe por el orden correcto.

El problema de escribir expresiones en la forma habitual (que se conoce como notación de *infijos*) es que un ordenador no las evaluará por el mismo orden. En cambio, para escribir expresiones utilizamos el sistema de notación polaca inversa (*reverse Polish notation*: RPN).

La conversión y evaluación de fórmulas en nuestra hoja electrónica se puede dividir en tres pasos distintos: el paso 1 es el auténtico proceso de convertir la serie de infijos a RPN; el paso 2 consiste en comprobar que la serie RPN producida esté bien constituida (sea sintácticamente correcta); el paso 3 es la evaluación propiamente dicha de la fórmula.

Los operadores que permite este programa son +, −, *, / y ↑. Si usted utiliza en una fórmula el signo − para indicar un valor negativo, el programa exige el carácter & para diferenciar éste del signo menos normal. Para llevar a cabo la conversión de infijos a RPN, necesitamos asignarle a cada uno de los operadores valores de precedencia. Esto determina el orden por el cual se trabaja con los operadores. La división y la multiplicación, por ejemplo, se efectúan antes de la suma y la resta:



El diagrama de flujo muestra el algoritmo para convertir una serie de infijos, I$, a una serie de notación polaca inversa, P$, y corresponde a la subrutina de la línea 4000 del programa. A partir de este diagrama usted podrá hacerse una idea de la forma en que trabaja la rutina. Los operadores admitidos por la rutina de conversión son los que se mencionan en el texto, mientras que los operandos permitidos son los nombres de las celdas y constantes reales. El principal problema de emplear el BASIC para el programa es que hemos de simular una pila LIFO. Esta función se lleva a cabo mediante el empleo de la matriz ST$() y la variable SP como señalador de la pila, utilizándose la primera para apilar los elementos de P$ durante el proceso de conversión

Operador	Valor de precedencia
=	0
(	1
+−	2
*/&	3
↑	4

En la versión final del programa de hoja electrónica, la rutina para convertir fórmulas de infijos a notación polaca inversa se acomodará así. Después de que entre una fórmula que relacione matemáticamente varias celdas, la serie de infijos resultante se coloca en una matriz, F$(), que almacena las fórmulas para cada celda. Luego se borra el elemento correspondiente de una segunda matriz, PS$(), la cual se utiliza para retener la correspondiente serie RPN.

Cuando seleccione la función calcular, el programa recorrerá la matriz PS$(), tomando cada serie polaca inversa de una en una y preparándola para la evaluación. Si desde la última función calcular se hubiera entrado en la hoja electrónica una nueva fórmula para una celda determinada, la entrada RPN para esa celda estaría vacía. Por tanto, la rutina de conversión que presentamos aquí es llamada antes de preparar la fórmula de la celda para su evaluación. Utilizando esta disposición, podemos retener versiones tanto de infijos como el RPN para cada fórmula de la hoja, usando la primera para visualizar en la pantalla y la segunda para la evaluación por ordenador.

Conversión de series

Complementos al BASIC

BBC Micro:

Introducir estas modificaciones en la versión para el C64:

Omitir la línea 50

```
4240 IF ST$(SP)="(" OR ST$(SP)=")" THEN PRINT TAB(0,22);" ERROR EN LA FORMULA ":RETURN
```

Amstrad CPC 464/664:

Introducir estas modificaciones en la versión para el C64:

Omitir la línea 50

```
4240 IF ST$(SP)="(" OR ST$(SP)=")" THEN LOCATE 1,22:PRINT " ERROR EN LA FORMULA ":RETURN
```

La línea 3140 se debe renumerar 3160

Commodore 64:

```
50 CD$=CHR$(17):CU$=CHR$(145):CR$=CHR$(29):CL$
   =CHR$(157):CO$=CHR$(19)
```

Adición a la subrutina de matrices

```
3140 DIM F$(225):DIM P$$(255)
```

Rutina de conversión RPN

```
4000 REM ********************
4001 REM * CONVERSION DE SERIE NORMAL *
4002 REM * A POLACA INVERSA
4003 REM ********************
4005 FOR K=1 TO 20:ST$(K)=" ":NEXT K
4006 LET P=0:LET SP=0:P$=""
4010 LET I$=C$
4015 FOR K=1 TO LEN(I$)-1
4017 J$=MID$(I$,K,1):K$=MID$(I$,K+1,1)
4020 IF J$="(" AND K$="-" THEN
     I$=MID$(I$,1,K)+"&"+MID$(I$,K+2)
4025 NEXT K
4030 IF P<=LEN(I$) THEN 4100
4040 REM *** PILA VACIA *****
4050 IF SP=0 THEN PS$((J-1)*15+I)=P$:GOSUB
     4700:RETURN
4060 IF ST$(SP)="(" OR ST$(SP)=")" THEN 4080
4070 LET P$=P$+ST$(SP)
4080 LET SP=SP-1
4090 GOTO 4050
4095 STOP
4100 LET P=P+1
4110 LET S$=MID$(I$,P,1)
4120 IF S$="+" OR S$="-" OR S$="*" OR S$="/" THEN
     4200
4130 IF S$="^" OR S$="(" OR S$=")" OR S$="&" THEN
     4200
4140 LET P$=P$+S$
4150 GOTO 4030: REM CONTINUAR HASTA EL FINAL DE LA
     SERIE
4200 IF SP=0 THEN 4400
4210 IF S$="(" THEN 4400
4220 GOSUB 4500:REM ** PRECEDENCIA **
4230 IF PS> PT THEN 4300:REM PRECEDENCIA
4240 IF ST$(SP)="("OR ST$(SP)=")" THEN GOSUB
     1950:PRINT" ERROR EN LA FORMULA ":
     RETURN
4250 LET P$=P$+ ST$(SP):ST$(SP)="":LET SP=
     SP-1
4260 IF SP> 0 THEN 4220
4270 SP=SP+1
4280 LET ST$(SP)=S$
4290 GOTO 4030: REM CONTINUAR HASTA FINAL DE LA
     SERIE
4300 IF ST$(SP)="(" AND S$=")" THEN LET ST$(SP)=""
     :SP=SP-1 :GOTO 4030
4400 LET SP=SP+1
4410 LET ST$(SP)=S$
4420 GOTO 4030
4500 REM *** EVALUACION DE PRECEDENCIA ***
4510 LET T$=S$:GOSUB 4600
4520 LET PS=T
4530 LET T$=ST$(SP):GOSUB 4600
4540 LET PT=T
4550 RETURN
4600 IF T$="=" THEN T=0:RETURN
4605 IF T$="(" THEN T=1:RETURN
4610 IF T$="+" OR T$="-" OR T$=")" THEN T=2:
     RETURN
4620 IF T$="*" OR T$="/" OR T$="&" THEN T=3:
     RETURN
4630 IF T$="^" THEN T=4:RETURN
4650 T=0:RETURN
```

Sinclair Spectrum:

Adición a la subrutina de matrices

```
3150 DIM F$(255,20):DIM H$(255,20):RETURN
```

Rutina de conversión RPN

```
4000 > REM ********************
4001 REM * CONV. DE SERIE NORMAL *
4002 REM * A POLACA INVERSA *
4003 REM ********************
4005 DIM S$(20)
4006 LET P=0:LET SP=0: LET P$=""
4010 LET I$=C$
4011 FOR Z=1 TO LEN (I$)
4012 IF I$(Z)=" " THEN GO TO 4014
4013 NEXT Z
4014 LET I$=I$( TO Z-1)
4015 FOR K=1 TO LEN (I$)-1
4017 LET J$=I$(K): LET K$=I$(K+1)
4020 IF J$="(" AND K$="-" THEN LET I$=I$
     (1 TO K)+"&"+I$(K+2 TO)
4025 NEXT K
4030 IF P< LEN(I$) THEN GO TO 4100
4040 REM **** PILA VACIA ****
4050 IF SP=0 THEN LET H$((J-1)*15+I,
     1 TO)=P$: GO SUB 4700:RETURN
4060 IF S$(SP)="(" OR S$=")" THEN GO TO
     4080
4070 LET P$=P$+S$(SP)
4080 LET SP=SP-1
4090 GO TO 4050
4095 STOP
4100 LET P=P+1
4110 LET O$=I$(P)
4120 IF O$="+" OR S$="-" OR O$="*" OR O$
     ="/" THEN GO TO 4200
4125 IF O$="^" OR O$="(" OR O$=")" OR O$
     ="&" THEN GO TO 4200
4140 LET P$=P$+O$
4150 GOTO 4030
4200 IF SP=0 THEN GO TO 4400
4210 IF O$="(" THEN GO TO 4400
4220 GO SUB 4500: REM PRECEDENCIA
4230 IF PS> PT THEN GO TO 4300
4240 IF S$(SP)="("OR S$(SP)=")" THEN PRINT
     AT 20,0;"ERROR EN LA FORMULA ":
     RETURN
4250 LET P$=P$+ S$(SP): LET SP=SP-1
4260 IF SP> 0 THEN GO TO 4220
4270 SP=SP+1
4280 LET S$(SP)=O$
4290 FO TO 4030: REM CONTINUAR HASTA FIN
     DE LA SERIE
4300 IF S$(SP)="(" AND O$=")" THEN LET
     S$(SP)="" : LET SP=SP-1: GO TO 4030
4400 LET SP=SP+1
4410 LET S$(SP)=O$
4420 GO TO 4030
4500 REM * EVALUACION DE PRECEDENCIA *
4510 LET T$=S$:GO SUB 4600
4520 LET PS=T
4530 LET T$=S$(SP):GO$SUB 4600
4540 LET PT=T
4550 RETURN
4600 IF T$="=" THEN LET T=0: RETURN
4610 IF T$="(" THEN LET T=1: RETURN
4620 IF T$="+" OR T$="-" OR T$=")" THEN
     T=2: RETURN
4630 IF T$="*" OR T$="/" OR T$="&" THEN
     LET T=3: RETURN
4640 IF T$="^" THEN LET T=4: RETURN
4650 T=0:RETURN
```

LA JET SET

LA JET SET

En la era del jet
La Epson SQ-2000 es una impresora de chorro de tinta de alta calidad que incorpora una amplia gama de estilos de impresión e impresión de gráficos en alta resolución. Configura el nuevo sistema de Epson de cartucho de tinta y limpieza automática

Espacio para maniobrar
La Epson SQ-2000 no incluye interface para ordenador. En cambio, en la parte posterior de la máquina se puede instalar una de tres interfaces estándares: Centronics, RS232C e IEEE. Aquí también se puede añadir un tampón opcional de 32 K y tipos adicionales en ROM

LA JET SET

LA JET SET

Crispin Thomas

La Epson SQ-2000 es una excelente representante de las impresoras de chorro de tinta ("ink jet printer")

En los últimos años, el precio medio de las impresoras matriciales de gran calidad se ha reducido rápidamente hasta ponerse al alcance de muchos usuarios de ordenadores personales. La nueva gama de impresoras Epson parece estar pronta a hacer lo mismo con la tecnología de chorro de tinta. La SQ-2000, que describimos aquí, es la que actualmente está en el mercado, pero es probable que Epson introduzca otra impresora de chorro de tinta A4 unidireccional (que se llamará HS-80). Esto pondrá a las impresoras de chorro de tinta al alcance de los usuarios de ordenadores personales.

Al igual que sus parientes de la familia (la Epson FX y la gama de impresoras matriciales MX), la SQ-2000 viene alojada en una atractiva carcasa de plástico color crema. Cuando la máquina descansa sobre un escritorio, los pulsadores de control, el interruptor de red y la cámara de cartuchos de tinta quedan fácilmente accesibles. Dado que está diseñada básicamente para pequeñas empresas, la SQ-2000 posee un carro ancho para dar cabida a hojas especiales de contabilidad y a hojas electrónicas. Deliberadamente, Epson ha hecho que su nueva impresora sea lo más flexible posible. El papel se puede alimentar mediante el rodillo portapapel normal y el método de guías de hoja, pero también se pueden adquirir como extras alimentadores de papel de hojas cortadas o de tractor continuo. La unidad básica no tiene incorporada interface para ordenador, pero en la parte posterior de la máquina hay una ranura para enchufar un cartucho de interface. Es posible escoger entre tres tipos de interface estándares distintos: en paralelo Centronics, en serie RS232C o IEEE 488, según el tiempo de ordenador con el que se usará la máquina. El cartucho de interface se instala fácilmente empujándolo en su sitio y asegurándolo con un par de tornillos. Las interfaces estándares tienen un tampón de entrada de 2 K en donde almacenar temporalmente los datos que entran procedentes del ordenador. Un buffer interno de doble impresión permite la decodificación e impresión simultánea de los datos tomados del tampón de entrada, y hay disponible un tampón de caracteres de carga inferior de 5 K para almacenar caracteres definidos por el usuario.

Las impresoras de chorro de tinta presentan muchas ventajas sobre las matriciales y las margarita. Combinan la calidad de impresión de las impresoras margarita con la velocidad y la flexibilidad de las impresoras matriciales, y sin el nivel de ruido. De hecho, la SQ-2000 es considerablemente más silenciosa que ninguna otra impresora matricial o margarita, lo que representa una importante ventaja. Sin embargo, las impresoras de chorro de tinta han experimentado problemas que sólo recientemente han podido solucionarse: las cámaras de tinta tendían a quedarse bloqueadas y las máquinas solían ser difíciles de manejar. Epson afirma que su nuevo sistema de entrega de tinta evita estos

problemas. En este sistema, la tinta se suministra desde un cartucho de tinta sellado que consta de tres cámaras, que contienen una tinta de fórmula especial, un líquido de limpieza y un espacio de almacenamiento para los productos de desecho. El cartucho está diseñado para imprimir más de tres millones de caracteres antes de su sustitución.

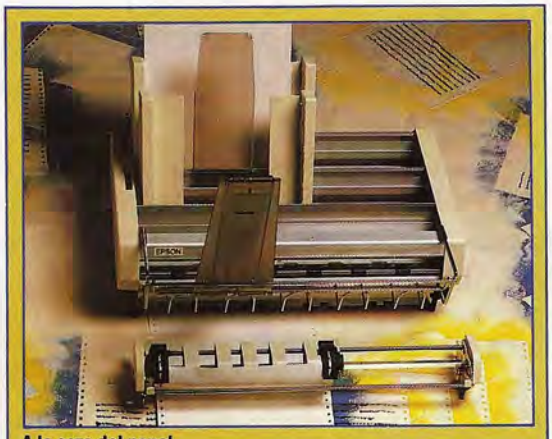
La razón por la cual Epson alega haber mejorado la fiabilidad es que el sistema de suministro de tinta se limpia de forma automática (operación que lleva apenas unos pocos segundos) cada vez que se enciende o se apaga la máquina, y periódicamente mientras la máquina se halla en funcionamiento. Esta operación de limpieza también se puede desencadenar manualmente manteniendo apretado un pulsador situado en el panel de control delantero. Los otros controles de este panel incluyen los conocidos pulsadores *on-line, line* y *form feed,* así como un botón de alimentación de hojas que permite cargar un hoja única desde un sujetapapel. Tres luces indicadoras de color verde señalan que la máquina está en marcha, si la máquina está o no en línea y si está lista para recibir datos desde el ordenador anfitrión. Dos luces rojas indican que se está acabando la tinta o que se ha agotado el papel.

Una de las características más impresionantes de la SQ-2000 es su gama de estilos de impresión. Hay dos modalidades básicas disponibles: borrador y calidad casi de impresión (*near letter quality*: NLQ). La modalidad de borrador es más rápida que la NLQ, pero posee una pérdida proporcional de calidad de impresión. En cada modalidad se pueden seleccionar tipos Pica, Elite y Roman en formas ampliada, condensada, cursiva, subrayada, realzada, proporcional e índice/subíndice.

Al igual que los producidos mediante una impresora matricial, los caracteres de una impresora de chorro de tinta se forman a partir de puntos individuales, permitiendo la flexibilidad de formas de caracteres programables. No obstante, debido a que los puntos de tinta individuales se mezclan entre sí, la forma del carácter resultante es de superior calidad que su equivalente matricial. La SQ-2000 produce sus caracteres a partir de una cuadrícula rectangular. En la modalidad borrador, los caracteres se componen en una cuadrícula de 15 por 24 puntos; en modalidad NLQ, la cuadrícula mantiene su tamaño original pero incluye 29 por 24 puntos para mejorar la calidad de la imagen impresa.

La cabeza de impresión está compuesta por 24 chorros dispuestos en dos filas de 12. Las filas están ligeramente desplazadas de modo que cada chorro se superpone ligeramente con su vecino de la otra fila, y es esta superposición lo que proporciona una imagen de impresión más fuerte. Al igual que sus parientes matriciales, la SQ-2000 también se puede utilizar para producir imágenes por mapa de bits, permitiendo volcar directamente en ella visualizaciones de pantalla en alta resolución. La velocidad de impresión es de 176 caracteres por segundo (cps) en modalidad borrador y 105 cps en modalidad NLQ.

Además de los accesorios de hoja cortada y alimentación por tracción, hay otras numerosas opciones disponibles. Éstas incluyen un tampón de entrada de 32 K que permite que el ordenador anfitrión vuelque rápidamente los datos a imprimir y luego prosiga con otras tareas mientras los datos se sacan del buffer y se imprimen automáticamente.



A la caza del papel
La SQ-2000 viene con un alimentador de hojas individuales, pero Epson también ofrece como opciones una unidad de tracción y una bandeja alimentadora. La unidad tractora posee los conocidos dientes de engranaje que empujan el papel a través de la cabeza de impresión. La unidad de la bandeja alimentadora será de gran utilidad a las pequeñas empresas, ya que permite alimentar con hojas de papel individuales la máquina desde dos bandejas que se pueden seleccionar independientemente, lo que resulta ideal para cartas personalizadas o informes

Además de las fuentes disponibles en la máquina estándar, se pueden añadir otras ROM.

Esta impresora es una compra ideal para el usuario serio de micros, porque es capaz de llevar a cabo las principales tareas de impresión que se requieren en el entorno de una pequeña empresa. Es de agradecer, especialmente, la reducción del nivel de ruido. Dado que su precio la coloca un poco fuera del alcance del bolsillo de la mayoría de los usuarios de ordenadores personales, la esperada introducción de una versión de precio reducido se hace sumamente atractiva.

EPSON SQ-2000

DIMENSIONES
620 × 297 × 188 mm

INTERFACES
En paralelo Centronics, RS232C, IEEE 488

VELOCIDAD
176 cps en modalidad borrador, 105 cps en modalidad NLQ *(near letter quality*: calidad casi de impresión)

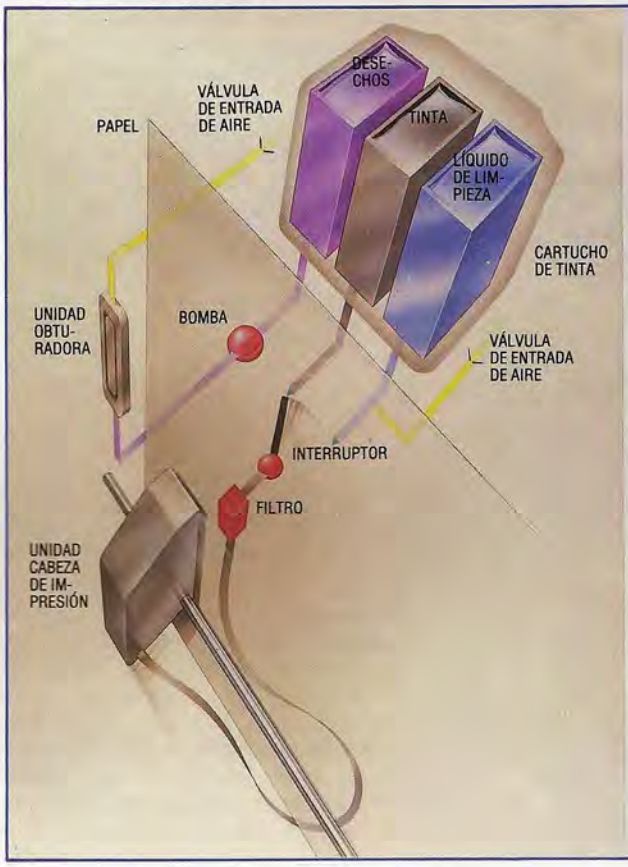
ALIMENTACION DE PAPEL
Por fricción o por tracción

ANCHURA DEL CARRO
140 mm mínimo, 406 mm máximo

DOCUMENTACION
Manual conciso y bien escrito, con elaborados ejemplos de generación de caracteres definidos por el usuario y gráficos por mapa de bits, apéndices completos de códigos de control y juego de caracteres

VENTAJAS
Combina la flexibilidad y la velocidad de las impresoras matriciales con la calidad de impresión de las impresoras margarita, y es mucho más silenciosa que ambas. Puede realizar las principales tareas de impresión que se requieran en una pequeña empresa

DESVENTAJAS
Para evitar que la impresión salga emborronada hay que escoger cuidadosamente el tipo de papel. Su precio la hace poco accesible a la mayor parte de usuarios de ordenadores personales



Haciendo la limpieza

El nuevo sistema de suministro de tinta desarrollado por Epson posee tres depósitos para tinta, líquido de limpieza y productos de desecho generados por el proceso de autolimpieza. El bombeo del líquido de limpieza a través del sistema mantiene a los tubos libres de posibles bloqueos, y una pequeña banda de goma situada a la izquierda del rodillo portapapel va frotando las boquillas del chorro de tinta por delante de la cabeza de impresión

Cortesía de Epson Ltd.

Epílogo

Finalmente demostraremos cómo la ampliabilidad de este lenguaje puede ser de gran utilidad en el desarrollo de un programa para controlar un sistema de semáforos

Como última palabra a nuestra serie dedicada al FORTH presentamos un programa que ilustra cabalmente la facilidad más útil de este lenguaje informático: su ampliabilidad. Como ya hemos visto a lo largo de la serie, este aspecto del FORTH, que da origen a estructuras fáciles de manipular y definir, permite su utilización para aplicaciones que otros lenguajes en realidad no pueden gestionar. Una de tales aplicaciones es el control de maquinarias, con el cual está relacionado el programa que proporcionamos a continuación.

Rojo, ámbar, verde

La máquina que estamos controlando en esta ocasión corresponde a un sistema de semáforos. Si estuviéramos haciéndolo en un plano real, lo haríamos a través de una puerta de salida en paralelo desde el ordenador, controlada mediante alguna palabra dependiente del sistema, como OUT (byte de salida --). Supongamos que las tres luces se controlan mediante tres bits de este byte de salida: el bit 0 para la roja, el bit 1 para la ámbar y el bit 2 para la verde. Si un bit es 1, entonces su luz se enciende, y si es 0, su luz se apaga. En el ejemplo, reemplazamos el OUT correcto por uno que imprime en la pantalla los nombres de las luces que están encendidas.

También necesitamos una palabra que espere durante un período dado. No hay ningún estándar que pueda cumplir este cometido, pero experimentando con su ordenador usted puede hacerlo utilizando bucles DO vacíos.

```
1 CONSTANT ROJO
2 CONSTANT AMBAR
4 CONSTANT VERDE

:OUT(codigo luces --)
DUP ROJO AND IF
   ."rojo"
ELSE
   "  "
   .
THEN
DUP AMBAR AND IF
   ."ambar"
ELSE
   "  "
   .
THEN
VERDE AND IF
   ."verde"
THEN
CR
;
VARIABLE LUCES (codigo ultimas luces
   escrito OUT)
   0 LUCES!
   0 OUT

:ENCENDIDA (codigo color --)
   LUCES @ OR
   DUP OUT
   LUCES!

;
APAGADA (codigo color --)
   NOT(usar-1 XOR en FORTH-79 o
      en ordenador ONE FORTH)
   LUCES @ AND
   DUP OUT
   LUCES!

;
VARIABLE BUCLESSEG
30000 BUCLESSEGS!(pruebe
   alterandolo)
:1SEG (espera un segundo)
   BUCLESSEGS @ 0 DO
   LOOP
;

:SEGUNDOS(segundos a esperar --)
   1 MAX 30 MIN (limite entre 1
      y 30 segundos)
   0 DO
      1SEG
   LOOP
;
;RUN
   ROJO APAGADO AMBAR APAGADO
   VERDE APAGADO
   5 0 DO
      ROJO ENCENDIDO 10 SEGUNDOS
      AMBAR ENCENDIDO 2 SEGUNDOS
      ROJO APAGADO AMBAR APAGADO
         VERDE ENCENDIDO
            10 SEGUNDOS
      VERDE APAGADO AMBAR ENCENDIDO
            2 SEGUNDOS
         AMBAR APAGADO
      LOOP
;
```

Está claro que ahora usted cuenta con un versátil lenguaje para controlar un sistema de semáforos, que podría ampliar fácilmente para cubrir un cruce completo. Es la ampliabilidad del FORTH lo que permite mezclar de una forma tan legible las partes de control de semáforos (como ROJO ENCENDIDO 10 SEGUNDOS) con las estructuras del programa.

Procesador de números

En el primero de estos tres artículos que dedicaremos al FORTRAN comprobaremos su gran capacidad aritmética

El FORTRAN se suele considerar como el primero de los lenguajes de alto nivel del tipo con el que estamos familiarizados en la actualidad. Se diseñó en un momento en que casi todos los ordenadores se utilizaban para tareas puramente numéricas, de modo que sus facilidades para manipular, por ejemplo, series de caracteres, son muy limitadas; pero así y todo sigue siendo uno de los pocos lenguajes capaces de manipular directamente números complejos. Su naturaleza matemática significa que se pueden escribir rutinas numéricas rápidas y eficaces, lo que lo convierte en un buen lenguaje para gráficos; asimismo, existen numerosas máquinas cuyos sistemas operativos se han escrito, al menos parcialmente, en FORTRAN.

Con el correr de los años se han ido construyendo muchas bibliotecas de subrutinas para una variedad de aplicaciones numéricas y éste es uno de los motivos de la continuada popularidad de lo que, al fin y al cabo, es un lenguaje muy anticuado, puesto que la forma más simple de utilizar este gran volumen de software es seguir escribiendo en FORTRAN. Parecería, entonces, que este lenguaje continuará siendo la principal herramienta de programación de la ingeniería y los laboratorios científicos.

Existen algunas versiones disponibles de FORTRAN para microordenadores, si bien pocas de ellas son implementaciones del estándar actual, el FORTRAN 77. La mayoría son versiones del estándar anterior, el FORTRAN IV, con ampliaciones moderadamente coherentes. Nos centraremos primero en el FORTRAN IV, porque ofrece la mejor idea de lo que es el lenguaje, y luego indicaremos las extensiones y matizaciones comunes al estándar, permitidas por la mayoría de las implementaciones para micros. El trazado de un programa en FORTRAN está determinado por el hecho de que la mayor parte de las entradas/salidas solían ser tarjetas perforadas de 80 columnas. Las ocho últimas columnas de tales tarjetas por lo general estaban reservadas para un número de identificación, lo que dejaba 72 columnas disponibles para cada sentencia de programa. En consecuencia, el FORTRAN sólo acepta una sentencia por línea y una línea por sentencia, con una longitud máxima de 72 caracteres. Si es necesario escribir una sentencia en más de una línea, entonces las líneas siguientes han de señalarse especialmente como líneas de continuación. Las 72 columnas restantes quedan aún más limitadas por el hecho de que las cinco primeras están reservadas para números de sentencia, la sexta columna se utiliza para indicar si la línea es o no una continuación, y la verdadera sentencia comienza en la columna siete.

Las líneas no se han de numerar como en un programa en BASIC, sino que a cualquier línea a la que se haga referencia en otra sentencia se le debe otorgar un número a modo de etiqueta. Estos números no deben seguir ninguna secuencia dada, sino que deben ser exclusivos. Una línea se puede utilizar como comentario colocándole una C en la columna uno. Los programas en FORTRAN, al igual que la mayoría de los programas en BASIC, son difíciles de estructurar correctamente, y carecen de tipos de datos y estructuras de control, lo que los hace difíci-

Evolución del lenguaje

Cronología de las versiones FORTRAN:

1954 Se formulan las primeras especificaciones del FORTRAN

1957 Primera publicación general de compiladores para FORTRAN (I)
Formulación de especificaciones para el FORTRAN II

1958 Aparecen los primeros compiladores para FORTRAN II

1962 Se formulan las especificaciones para el FORTRAN IV

1963 Aparecen los compiladores para el FORTRAN IV

1977 Especificaciones para el nuevo FORTRAN 77

1978 Aparecen los primeros compiladores para el FORTRAN 77

Sperry Ltd.

Grace Hopper
Grace Hopper fue una de las fuerzas impulsoras del desarrollo de los primeros lenguajes de programación. A mediados de los años cincuenta, cuando trabajaba en Remington Rand, ayudó a producir uno de los primeros compiladores (A-2) y los subsiguientes lenguajes ARITHMATIC y FLOW-MATIC. Luego IBM refinó el concepto de lenguajes compilados en su versión final de FORTRAN, mientras que el FLOW-MATIC se convertiría posteriormente en el COBOL

les de leer y, por tanto, requieren la inclusión de comentarios como una importante característica.

Como en BASIC, pero no como en PASCAL, las variables no han de declararse; se las puede introducir en cualquier punto de un programa tan sólo con seleccionar un nuevo nombre. Sin embargo, si usted hace esto, entonces acepta la tipología de datos por defecto, que es que todos los nombres de variable empiecen con I, J, K, L, M o N si son de enteros, tomándose como reales todos los demás.

Normalmente, una de las principales dificultades para los programadores de FORTRAN recién iniciados son las complejidades de la manipulación de entrada/salida. El problema deriva de la dependencia básica de las tarjetas perforadas, con su formato estricto. Cada operación de E/S requiere dos sentencias: un READ o WRITE, que especifica tanto el dispositivo de E/S como las variables que se estén utilizando, y una sentencia FORMAT, que especifica los tipos y trazado exactos de los datos de la tarjeta, registro o línea de entrada de terminal. Por ej.:

```
READ(1,100)X,I,ICHAR
100 FORMAT(F7.2,I4,A1)
```

El READ es la sentencia ejecutable que hará que se lleve a cabo una operación de entrada. El primer número dentro del paréntesis especifica el dispositivo de entrada que se está utilizando. La asignación de números a dispositivos depende de la implementación exacta y del hardware, pero un número habrá de referirse al terminal, otros a la impresora (sólo para salida), lector o perforador de tarjetas, o se le podría asignar a archivos en disco o cinta.

El segundo número dentro del paréntesis es un número de sentencia que identifica la sentencia FORMAT correspondiente. (Observe que, en el ejemplo, a la sentencia FORMAT se le ha dado el mismo número.) la sentencia FORMAT no es ejecutable; simplemente da información, de modo que se puede posicionar en cualquier lugar del programa. El número es la única forma de mostrar la relación, y es bastante permisible que más de un READ o WRITE se refieran al mismo FORMAT. Tras el paréntesis que sigue al READ se halla la lista de variables a las cuales se asignan los valores de entrada.

El FORMAT incluye una especificación para cada variable. Las principales especificaciones son:

- F: F7.2, por ejemplo, indica que el valor a entrar en X es real, ocupando las siete primeras columnas de la línea, registro o tarjeta con dos dígitos después del punto.
- I: I4 indica que el valor a entrar en I es un entero, ocupando las cuatro columnas siguientes de la línea.
- A: A1 indica que el valor de la siguiente columna es un carácter.

La única forma de que el FORTRAN pueda manipular caracteres es almacenándolos como variables de enteros. La cantidad máxima por variable depende del tamaño que tenga asignado el sistema para una variable de enteros, de modo que la utilización de enteros de 16 bits significa que sólo se permiten A1 y A2. No obstante, algunas versiones permiten almacenar caracteres también en variables de reales. Dado que las variables se pueden tratar tanto como numéricas como alfabéticas, y dado que la forma en que se almacenan los caracteres no siempre corresponde al código ASCII estricto, esto puede crear muchísima confusión. Las series de caracteres más largas, por ejemplo, se han de manipular utilizando matrices de enteros. En consecuencia, ¡el FORTRAN no es un lenguaje recomendable para aplicaciones que supongan tratamiento de textos!

Los verdaderos valores que se entran deben acomodarse a estas especificaciones, porque de lo contrario se generará un error en tiempo de ejecución. La mayoría de los sistemas modernos, sin embargo, permiten una entrada de la denominada *forma libre*, como READ (1,*)A,B,C o simplemente READ A,B,C, que está pensada específicamente para entradas por teclado, donde es difícil conseguir el espaciado exacto entre valores. Estas entradas de forma libre funcionan, entonces, de forma similar a la sentencia INPUT del BASIC. La salida se manipula de modo similar utilizando una sentencia WRITE:

```
WRITE(1,200)A,B,C
200 FORMAT(3F7.2)
```

Observe cómo se pueden manipular las especificaciones repetidas sin tener que escribir una para cada variable.

Las series de texto se pueden hacer salir usando

Lógica comparativa

Operadores lógicos y conectores:

FORTRAN	BASIC
.EQ.	=
.NE.	<>
.GT.	>
.GE.	>=
.LT.	<
.LE.	<=
.NOT.	NOT
.AND.	AND
.OR.	OR

Eficiente con los números

FORTRAN IV admite varios tipos de datos numéricos:

- INTEGER (enteros) y REAL (reales), que poseen sus significados habituales
- DOUBLE PRECISION (doble precisión), que son números de punto flotante que utilizan el doble de la cantidad de palabras de memoria que los reales comunes
- COMPLEX (complejos) para números complejos con partes real e imaginaria
- LOGICAL (lógicos) equivalentes a un booleano del PASCAL, que puede tomar los valores .TRUE. (verdadero) o .FALSE. (falso) (hay que encerrarlos entre puntos)
- Los nombres de variables pueden tener hasta 6 caracteres de longitud, utilizando sólo caracteres alfabéticos o numéricos, debiendo ser alfabético el primero de ellos
- Las variables sólo se pueden declarar al principio del programa mediante una sentencia de declaración, como:

```
INTEGER NUM1, NUM2
LOGICAL FNSH
```

una especificación H especial, pero ésta es difícil de usar porque se debe enunciar la cantidad exacta de caracteres:

```
      WRITE(1,300)RESP
  300 FORMAT(13HLA RESPUESTA ES,F7.2)
```

Nuevamente, muchos sistemas modernos admiten series entre comillas dentro del FORMAT:

```
      WRITE(1,300)RESP
  300 FORMAT('LA RESPUESTA ES',F7.2)
```

pero esto no es FORTRAN estándar.

Cuando la salida se dirige a una impresora de líneas, el FORTRAN utiliza el primer carácter impreso para el control de papel, si bien los detalles exactos pueden variar de una instalación a otra. Un sistema típico, por ejemplo, podría utilizar un carácter de espacio para la espaciación simple normal, y los caracteres 1, 2, 3, etc., para dejar el adecuado número de líneas en blanco. Es un error común olvidar esto y perder el primer carácter de su salida mientras la impresora hace algo impredecible con el papel. Una salida hacia la impresora podría ser:

```
      WRITE(2,400)I,J,K,L,M,N
  400 FORMAT(1H,2(I6,5X))
```

Observe el 1H extra, que controla el papel. La especificación X simplemente deja la cantidad de espacios en blanco especificada. Se puede encerrar entre paréntesis una secuencia de especificaciones con un multiplicador, como en 2(I6,5X).

En los ejemplos utilizaremos siempre un número 1 de dispositivo para indicar entrada/salida a través de un terminal, y el verdadero valor a utilizar aquí dependerá de la implementación empleada.

El FORTRAN es muy similar al BASIC en la forma en que establece la aritmética y la asignación, y la única diferencia importante es que el FORTRAN utiliza un asterisco doble (**) para la elevación a potencia en lugar de la lfecha hacia arriba (↑). En una expresión se pueden mezclar valores reales, enteros e incluso de doble posición, y asignarle el resultado a una variable de cualquier tipo. No obstante, se debe tener cuidado, ya que de lo contrario se podría encontrar con que todo el cálculo, o parte del mismo, se lleve a cabo en aritmética de enteros cuando se requiere un resultado de reales:

```
I=2
J=3
A=I/J
```

A tendrá el valor 0, porque la división se ha realizado en aritmética de enteros. Se proporciona una función FLOAT para convertir un valor entero en real, de modo que:

```
A=FLOAT(I)/FLOAT(J)
```

funcionará correctamente.

Habiendo sido diseñado como lenguaje aritmético, uno esperaría encontrar en el FORTRAN una gran variedad de funciones estándares. Hay demasiadas como para listarlas aquí, incluyendo todas las funciones trigonométricas y logarítmicas usuales, así como muchas otras que sólo tendrán algún significado para los matemáticos e ingenieros. Pero no sería arriesgado afirmar que el FORTRAN posee como función todas las operaciones numéricas estándares. Por último, la mayoría de las funciones se proporcionan en versiones tanto de reales como de doble precisión y, también, cuando es adecuado, en una versión de enteros.

Una de las principales críticas que se hacen al FORTRAN es su falta de estructuras de control, que en el FORTRAN 77 se ha paliado hasta cierto punto. La familiar sentencia GOTO se utiliza ampliamente y su destino puede ser cualquier sentencia numerada ejecutable (no una sentencia FORMAT).

Existen dos variables de sentencia IF: el IF lógico es similar al del BASIC, tomando la forma:

```
IF(expresion logica) sentencia ejecutable
```

(Los operadores lógicos y los conectores son una cuestión diferente.) La otra forma del IF es el de tipo aritmético, que suele considerarse el peor ejemplo de estructura de control. Éste asume la forma:

```
IF(expresion aritmetica)S1,S2,S3
```

donde S1, S2 y S3 son números de sentencia. El control se transfiere a la sentencia número S1 si el valor de la expresión aritmética es menor que cero, a S2 si es igual a cero y a S3 si es mayor que cero. Esto puede ser muy conveniente en tanto en cuanto permite una bifurcación en tres direcciones, pero puede dar lugar a un *código espagueti* aún peor que un GOTO normal.

Estudiante medio

```
C     PROGRAMA PARA LEER UN CONJUNTO
C     DE NUMEROS E IMPRIMIR SU
      MEDIA.
C     LA ENTRADA TERMINA
C     ENTRANDO UN NUMERO NEGATIVO
C
C     INICIALIZAR CUENTA Y SUMAR
C
      SUM=0
      ICOUNT=0
C
C     LEER SIGUIENTE NUMERO
C
100   READ(1,10)X
C
C     COMPROBAR SI ES NEGATIVO
C
      IF(X)300,200,200
C
C     NUMERO POSITIVO
C
200   ICOUNT=ICOUNT+1
      SUM=SUM+X
      GOTO 100
C
C       FIN DE LA ENTRADA
C
300   AVGE=SUM/FLOAT(ICOUNT)
      WRITE(1,20)AVGE
      STOP
C
C     FORMATS
C
10    FORMAT(F9.2)
20    FORMAT(10HEL PROMEDIO ES,F9.2)
      END.
```

En pantalla

La visualización de caracteres es realizada por el OS del Amstrad de forma diferente de como visualiza los gráficos

La VDU para textos se encarga, entre otras cosas, de la lectura y la escritura de caracteres en pantalla, del cursor, de las ventanas y de los símbolos definibles por el usuario. El firmware del Amstrad permite el empleo en cualquier parte de la pantalla de hasta ocho corrientes con sus respectivas ventanas, cada una de las cuales con sus posibles tintas de fondo y letra *(paper/pen)*, su cursor, modo de fondo opaco o transparente, y la opción de escritura por caracteres de los gráficos. El tamaño de las ventanas de una corriente puede definirse con TXT_WIN_ENABLE, consultarse por medio de TXT_GET_WINDOW y borrarse con TXT_CLEAR_WINDOW.

Dos corrientes cualesquiera pueden tener, además, todos sus parámetros intercambiados gracias a TXT_SWAP_STREAMS. La mayoría de las rutinas del firmware actúan sobre la corriente actual; la entrada TXT_STR_SELECT está pensada para seleccionar la corriente actual, que quedará activa hasta que sea seleccionada otra corriente.

Los caracteres son visualizados en pantalla dentro de un cuadrado de 8×8 pixels. Existen varias entradas del firmware empleadas para imprimir un carácter, y una de las más importantes es la que mostramos en la tabla *Direcciones útiles*.

Por lo general, los caracteres ocupan cuadrados adyacentes y se imprimen en la posición actual del cursor para texto. Pero existe un modo especial, que se selecciona con TXT_SET_GRAPHIC, cuya utilidad consiste en trazar los caracteres en la posición del cursor de gráficos, con lo que es posible superponer texto y hacerlo aparecer en cualquier sitio de la pantalla.

La rutina principal, TXT_OUTPUT, se limita a guardar todos los registros antes de llamar la *indirección* TXT_OUT_ACTION. La rutina que queda apuntada por la indirección se encarga de tratar los códigos de control antes de pasar los caracteres visualizables a TXT_WR_CHAR, o a GRA_WR_CHAR si se activa la opción de escritura caracteres-gráficos. Se puede parchear la indirección para proporcionar un medio alternativo a la interpretación de códigos de control o a la escritura de caracteres hacia la pantalla.

La VDU para texto tiene un cursor correspondiente que indica la posición donde será visualizado el carácter siguiente. Se dispone de rutinas para activar y desactivar el cursor de texto y colocar o eliminar otro cursor en cualquier posición para carácter. Hemos detallado en la tabla citada las rutinas más útiles que guardan relación con el posicionamiento del cursor.

El programa *tabulador* permite el uso de paradas de tabulación a intervalos iguales según la variable TAB. El tamaño de la ventana actual se obtiene primero con una llamada (CALL) a TXT_GET_WINDOW, la cual proporciona en el registro D la columna extrema derecha, pero se hace que apunte a una posición posterior al margen derecho de columnas. Seguidamente, mediante TXT_GET_CURSOR se obtiene la posición efectiva del cursor; el registro H proporciona la columna del cursor, que se compara sucesivamente con una posible parada de tabulación hasta que encuentra la siguien-

Direcciones útiles

Rutina	Dirección	Observaciones
TXT_WIN_ENABLE	BB66	Entrar con H izq.; D derecha; L sup.; E inferior
TXT_GET_WINDOW	BB69	Salida con registros como antes
TXT_CLEAR_WINDOW	BB6C	Limpia ventana texto de la corr. actual
TXT_SWAP_STREAMS	BBB7	B y C retienen los números de las dos corrientes
TXT_STR_SELECT	BBB4	La corriente a seleccionar está en A
TXT_OUTPUT	BB5A	A retiene el carácter
TXT_WR_CHAR	BB5D	A retiene el carácter
TXT_RD_CHAR	BB60	Lee el carácter en el cursor y lo lleva a A
TXT_GET_CURSOR	BB78	H da la columna, L la fila
GRA_SET_ORIGIN	BBC9	Entrar con las coord. DE-X y HL-Y
GRA_SET_WIDTH	BBCF	Entrar con las coord. DE izq. y HL der.
GRA_SET_HEIGHT	BBD2	Entrar con las coord. DE sup. y HL inf.
SCR_SET_BASE	BC08	A retiene byte *hi* de la dir. base
SCR_SET_MODE	BC0E	A retiene núm. de modo (0,1 o 2)

Nota: Para usar adecuadamente estas rutinas, necesitará consultar la guía *Amstrad firmware specification*, editada por Amsoft (SOFT 158), que contiene las condiciones completas de salida y entrada de éstas y las restantes rutinas de firmware

te. Una vez conocida esa posible parada siguiente de tabulación, se efectúa una comprobación del espacio disponible para el campo de caracteres que sigue. Si hay espacio, entonces la columna del cursor se pone de modo que corresponda a la siguiente parada de tabulación, y en caso contrario se obliga al cursor a desplazarse a la línea siguiente mediante la especificación de una columna exterior a la ventana actual. Si fuera necesario, se puede parchear la rutina dentro de TXT_OUTPUT para que puedan admitirse otras paradas de tabulación distintas de las ocho columnas habituales.

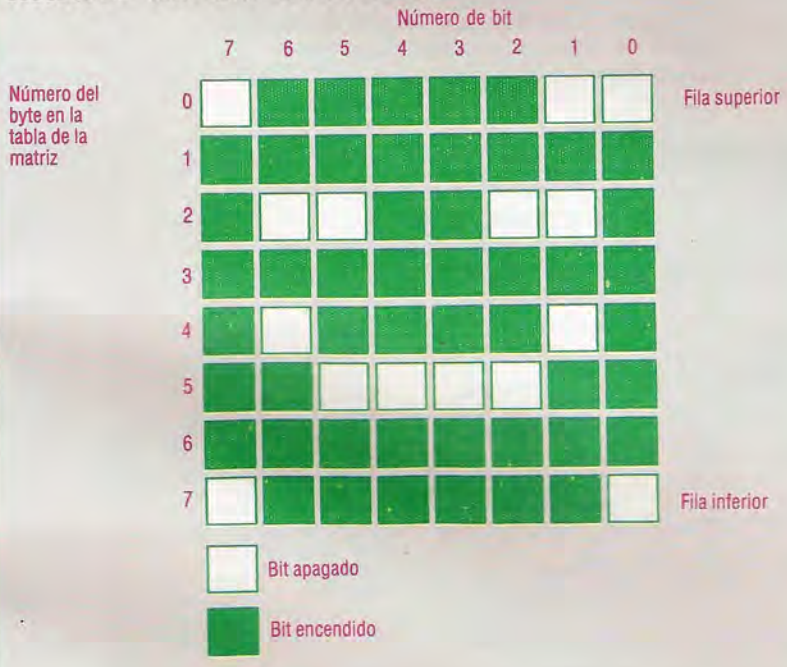
Cada uno de los códigos ASCII comprendidos entre 0 y 255 tienen asociada una matriz que define los bits que conforman el carácter visualizado correspondiente. La matriz se compone de ocho bytes, y cada byte forma una línea del carácter. El dibujo muestra cómo se almacena en memoria la matriz del carácter correspondiente a CHR$224.

Diferencias de modo
Cada byte en la memoria de pantalla retiene datos para un número de pixels entre dos y ocho, según el modo de visualización. El dibujo muestra a qué pixel corresponden los bits de cada byte en los distintos modos

Asignación de pixels

	Número de bit 7	6	5	4	3	2	1	0
Modo 2	1	2	3	4	5	6	7	8
Modo 1	1	2	3	4	1	2	3	4
Modo 0	1	2	1	2	1	2	1	2

Matriz de un carácter



Todo un carácter
Los caracteres son almacenados en matrices de ocho bytes, donde cada byte representa una fila de pixels del carácter. Los bits encendidos indican que un pixel ha de ser impreso en el color actual de la "pluma", y los apagados el color del "papel" (el fondo). El gráfico muestra la matriz del carácter 224 ASCII

El firmware presenta cuatro rutinas para determinar y redefinir estas matrices. Todas las matrices de los caracteres ASCII entre el 0 y el 247 se sitúan, por omisión, en la ROM inferior y por ello no son inicialmente redefinibles; los caracteres que van del 248 al 255 se copian en la RAM, donde puedan ser redefinidos. La rutina TXT_SET_M_TABLE puede emplearse para redefinir cualquier número de carácter comprendido entre el especificado y el 255, permitiendo así un carácter enteramente nuevo listo para su uso cuando se desee.

Los caracteres que han sido ubicados como redefinibles pueden aceptar la manipulación directa de sus matrices por medio de un programa del usuario, o pueden establecerse con la rutina TXT_SET_MATRIX que necesita ocho bytes apuntados por el registro HL y los copia en la matriz correspondiente a un carácter dado.

Se dispone también de la rutina TXT_GET_MATRIX para obtener la dirección de una matriz de carácter, así como de la rutina TXT_GET_M_TABLE para obtener tanto el primer carácter de una tabla definible por el usuario como su dirección.

El programa que presentamos, *Rotación de caracteres*, se sirve de TXT_GET_MATRIX y de TXT_SET_MATRIX para hacer rotar los caracteres en una de las cuatro direcciones. Esto se logra redefiniendo el carácter 248 para que tenga la misma matriz que el carácter requerido y manipulando después la matriz en el carácter 248 para la rotación deseada, antes de imprimirlo con TXT_OUTPUT. Se pueden conseguir efectos textuales muy provechosos enlazando este programa al BASIC.

La VDU de gráficos

La VDU de gráficos tiene por misión dibujar trazos en pantalla a una resolución de pixel. Se emplea una sola ventana de gráficos en la que las tintas de fondo y de primer plano pueden ser definidas independientemente de las usadas para la VDU de texto.

Existen tres sistemas de coordenadas que pueden servir para describir una posición de pixel en la pantalla. La resolución de pantalla efectiva es de 640×200 puntos en el modo 2, de 320×200 puntos en el modo 1 y de 160×200 puntos en el modo 0; éstas son las denominadas *coordenadas de base*. Sin embargo, cada punto en la pantalla tiene una dirección única que se emplea como referencia independientemente del modo actual. Los diferentes modos simplemente determinan cuánta área es asignada alrededor de un pixel al establecer ese punto. Así, en el modo 2 se establece un pixel, en el modo 1 se establecen dos y en el modo 0 se establecen cuatro.

Aunque la pantalla física tiene sólo una resolución de 200 puntos en vertical, se trata como si fuese de 400 puntos por el firmware, por lo que la proporción de 2:1 es aproximadamente aceptable. Por ello cada punto vertical corresponde a medio pixel y siempre se establecen al mismo tiempo las dos mitades de un pixel. Las filas de pixel 10 y 11, por ejemplo, corresponden ambas a la misma línea de pantalla.

Por lo tanto, la pantalla siempre es tratada como si tuviera una resolución de 640×400 en cualquier modo.

El sistema de coordenadas descrito sirve para las posiciones absolutas en toda la pantalla. Sin embargo, es posible definir una ventana de gráficos que emplee un tercio del total de las coordenadas para referenciar un punto.

El tamaño de la ventana se establece mediante GRA_SET_ORIGIN, GRA_SET_WIDTH y GRA_SET_HEIGHT; y puede determinarse empleando las

correspondientes GRA_GET_ORIGIN, GRA_GET_WIDTH y GRA_GET_W_HEIGHT. Decidido el uso de una ventana, todas las coordenadas se denomina coordenadas del usuario. El cursor de gráficos es controlado de dos maneras: el posicionamiento *absoluto* o el *relativo*. Si empleamos el método absoluto, el cursor se desplaza a los puntos especificados con respecto a su posición inicial. Las entradas empleadas para el posicionamiento del cursor se indican en un cuadro aparte.

La VDU de gráficos permite el trazado de pixels de tres modos: individualizados, formando caracteres o formando líneas. Todas las rutinas de trazados tienen la opción de trazado de puntos absolutos o la de puntos relacionados con la posición actual del cursor de gráficos. El cursor de gráficos se para en el último punto trazado, salvo cuando se trazan caracteres donde la posición vertical (ordenada) no cambia y la horizontal (abscisa) se mueve hasta el inicio del carácter siguiente. El cuadro mencionado detalla las entradas que se emplean para este fin.

Por *paquete de pantalla* se entiende una sección del firmware que proporciona el acceso a la pantalla en su nivel inferior. Maneja el direccionamiento de la pantalla, la implementación de la paleta de colores, la codificación y la decodificación de la tinta, la determinación del modo y el desfile de la pantalla, entre otras cosas.

La pantalla es un bloque de 16 K de RAM que puede ser ubicado en uno cualquiera de los cuatro bloques cercanos de 16 K en la memoria. Sólo pueden utilizarse los bloques que comienzan en $4000 y $C000, dado que los otros dos bloques contienen áreas del firmware que podrían quedar machacadas. Se puede conectar la pantalla entre dos bloques útiles por medio de la entrada SCR_SET_BASE; esta técnica puede servir para preparar una pantalla mientras se visualiza otra y hacer como si la actualización de la pantalla fuera instantánea.

Cada byte de memoria empleado para la pantalla contiene información sobre ocho posiciones físicas en pantalla. Éstas pueden corresponder a dos, cuatro u ocho pixels según sea el modo seleccionado por medio de SCR_SET_MODE.

Las asignaciones de pixels se ilustran en un diagrama que muestra cómo se determina cada pixel en un byte a través de los bits de los diferentes modos. Aunque al principio pueda parecer algo difícil, el sistema está de hecho diseñado de modo que, haciendo rotar un byte, se obtengan los bits necesarios para el pixel siguiente a la izquierda o a la derecha. Así, para poner un pixel dentro de un byte, se puede generar una máscara para establecer el pixel extremo izquierda y rotarla después hasta colocarla en la posición adecuada.

El camino más rápido para escribir en pantalla es el acceso a la memoria de pantalla, preferible al empleo de las VDU de textos o de gráficos. El paquete de pantalla proporciona varias rutinas para facilitar esta tarea.

Finalmente nuestro programa *Trazado rápido de un punto* es un ejemplo de listado que emplea estas entradas para trazar un punto físico en la pantalla. Obsérvese que no se puede usar junto con la pantalla de gráficos si se ha establecido una ventana.

La rutina pregunta antes que nada la dirección del byte que contiene información de la posición de pantalla especificada, y posteriormente pide la máscara para poner todos los pixels en ese byte con la tinta especificada. La máscara se manipula ahora para restablecer la posición requerida con tinta cero y volverla a poner en una nueva tinta.

El comPLOT de los 128 K

El Amstrad 6128 presenta unas sutiles mejoras de la instrucción PLOT según se implementaba en el CPC 464. El firmware en ambos modelos proporciona una rutina para seleccionar uno de los cuatro modos diferentes de trazado. Éstos son:

Modo	Actuación
0	Normal – pone nueva tinta al pixel
1	XOR – pone (nueva tinta XOR la existente) al pixel
2	AND – pone (nueva tinta AND la existente) al pixel
3	OR – pone (nueva tinta OR la existente) al pixel

El modo XOR es muy útil, dado que permite trazar líneas y borrarlas sin perturbar el color de fondo si se desea. Por ej., si el color de fondo es el rojo (*paper* 3), el trazado de una línea en blanco (*pen* 4) producirá una línea en magenta (*pen* 7). Ésta puede ser borrada trazándola por segunda vez en *pen* 4, restaurando al mismo tiempo el color primitivo del fondo. Este modo de trazado es muy valioso en las rutinas de manejo de sprites, que necesitan desplazar un fondo sin perturbarlo. En el 6128, los modos de trazado se especifican añadiendo un parámetro más a la instrucción PLOT, pero esta facilidad no está prevista en el BASIC 1.0 del CPC 464. Todo lo que se necesita es una llamada a la rutina del firmware en &BC59, con el número del modo deseado en A

Tabulador

Este programa desplaza el cursor de texto respecto de la ventana activa hasta la columna siguiente, tabulada por medio de TAB. No hay condiciones de entrada y la rutina guarda todos los registros

```
tab:            equ     15      ; columna de parada tabul.
get.window:     equ     #bb69   ; TXT_GET_WINDOW
get.cursor:     equ     #bb78   ; TXT_GET_CURSOR
get.column:     equ     #bb6f   ; TXT_SET_COLUMN
;
                push    af
                push    hl
                push    de
                call    get.window      ; entra tamaño ventana
                inc     d               ; columna rt logica
                inc     d               ; limite col. derecha
                call    get.cursor      ; toma pos. cursor
                ld      a, 1            ; inicio linea
;
loop:           add     tab             ; toma col. tabul. siguiente
                cp      h
                jp      p, loop
;
                ld      h, a            ; lo guarda
                add     tab             ; espacio para nueva col?
                cp      d               ; comprueba límite
                jr      nc, setcol      ; no, obliga nueva linea
                ld      a, h            ; recoge nueva col.
setcol:
                call    set.column      ; establece nueva col.
                pop     de
                pop     hl
                pop     af
                ret
```

Rotación de caracteres

Este programa imprime un carácter tras rotar la matriz de caracteres hacia una de las cuatro esquinas. Primero se halla la dirección de la matriz de caracteres y después se emplea para redefinir el carácter 248, que normalmente es la primera entrada de la tabla de caracteres definidos por el usuario. Si este carácter no es redefinible, no ocurre nada

```
;
; Entry:            IX+0 = Caracter por visualizar
;                   IX+2 = Esquina escogida 1 ... 4
                      1 = arriba
                      2 = atrás
                      3 = abajo
                      4 = normal
;
; Exist:
;                   Arrastre Verd – se imprime caracter
;                   Arrastre Falso = No se imprime carácter
;                   alterados AF, BC, DE, HL
;
txt.get.matrix:     equ     #bba5
txt.set.matrix:     equ     #bba8
txt.output:         equ     #bb5a
;
                    org     8000h
;
start:              ld      a, (ix+0)       ; lee caracter
                    call    txt.get.matrix  ; toma dir. matriz
;
; Copia matriz en caracter 128 definible usuario
; después busca su dirección
;
                ld      a, 248              ; caract. por redefinir
                call    txt.set.matrix      ; redefine matriz
                ret     nc                  ; nada si desactivado
;
                ld      a, 248              ; caracter para rotar
                call    txt.get.matrix      ; halla matriz
;
; Rota la matriz caracter en sentido antihorario
; en mov. de 90 grados
;
                ld      (toprow), hl        ; guarda dirección
                ld      b, (ix+2)           ; lee esquina
;
lop3:           ld      d, 8                ; bucle para 8 filas
;
lop2:           ld      e, 8                ; bucle para 8 bits
;
lop1:           rlc     (hl)                ; bit sig. en c
                rla                         ; guarda nueva fila
                inc     hl                  ; va a fila sig.
                dec     e
                jr      nz, lop1            ; busca fila sig.
;
                push    af                  ; nueva fila en pila
                dec     d
                ld      hl, (toprow)        ; busca bit sig.
                jr      nz, lop2
;
; Bucle para sacar de pila nueva matriz
;
                ld      e, 8
lop4:           pop     af                  ; toma fila sig.
                ld      (hl), a             ; va a matriz
                inc     h                   ; va a fila sig.
                dec     e
                jr      nz, lop4
;
                ld      hl, (toprow)
                djnz    lop3                ; de nuevo, si es neces.
;
; Por ultimo visualiza caracter en pantalla
;
                ld      a, 248              ; carac. por imprimir
                call    txt.output
                ret                         ; concluido
;
toprow:         defw    0                   ; inicio matriz
```

Texto en espiral
El programa de rotación de caracteres permite imprimir un texto verticalmente ¡y hasta de revés! La pantalla que mostramos da una idea de ello, aunque los efectos más brillantes se reservan para el usuario que emplee gráficos definidos por él

Trazado rápido de un punto

Esta rutina opera en todos los modos de pantalla y emplea el firmware para determinar una posición de memoria que contenga información del punto donde se ha de trazar el pixel

```
; Entry:        HL contiene la abscisa Y fisica
;               (0-199)
;               DE contiene la ordenada X fisica
;
;               (0-639)
;               A contiene la tinta asignada al punto
;
; Exit:         Se guardan todos los registros
;
dotpos:         equ     #bc1d       ; SCR_DOT_POSITION
encode:         equ     #bc2c       ; SCR_INK_ENCODE
;
plotter:
                push de             ; guarda los registros
                push hl
                push bc
                push af
;
                call dotpos         ; habla byte de memoria
                pop af              ; repone la tinta en a
                push af
;
; HL contiene ahora la dirección del byte
; y C contiene la máscara necesaria para poner
; solo el pixel deseado
;
                call encode         ; halla la masc. tinta
;
; La mascara para poner todos los pixels con tinta nueva esta en a
;
                and     c           ; solo quiero este pixel
;
                ld      b, a        ; guarda numero tinta nueva
                ld      d, (hl)     ; toma posicion actual
                ld      a, c        ; toma mascara pixel
                camp    a           ; la invierte
                and     d           ; pone el pixel a tinta 0
;
                or      b           ; enmascara nuevo color
                ld      (hl), a     ; y lo almacena
;
                pop     af          ; restaura los registros
                pop     bc          ;
                pop     hl
                pop     de
;
                ret
;
```

Cree usted su propia música

Visualización notable
La visualización es apenas una de las muchas características atractivas de «The music system», valiéndose de iconos y menús diseñados con claridad para ayudar al músico. Esta instantánea está tomada de la opción «Keyboard»

"The music system" es un paquete de alta calidad que se diferencia nítidamente del software musical producido hasta ahora

The music system, de Island Software, posee un juego de cinco opciones, a cada una de las cuales se puede acceder fácilmente desde disco. Por el contrario, la carga desde cassette se complica por el hecho de que hay que manejar dos cintas. Pero con sus ventanas, iconos y gráficos de estilo Macintosh, el sistema es a la vez amable y muy atractivo.

El primero de los cinco programas, y el que se suele usar primero, es el *Editor*, que la empresa describe como el "procesador de notas de un músico". Éste hace que el proceso de escribir o modificar música resulte excepcionalmente sencillo. La pantalla visualiza pentagramas de bajo y soprano, en los que se escriben las notas; usted cambia el valor de las notas, mientras las líneas de los compases se insertan automáticamente según una clave de tiempo preseleccionada. Cuando ha compuesto una melodía, o cuando simplemente quiere escuchar unos pocos compases, pulsa la tecla Tab, que inicializará la reproducción de audio acompañada por una visualización que va desfilando por la pantalla.

Una de las principales cualidades de este sistema reside en que es a la vez adecuado para el recién iniciado y una gran ayuda para el músico ya experimentado. El principiante puede ver y oír exactamente lo que está sucediendo en cualquier etapa de la composición, y los errores se detectan fácilmente, haciendo que la corrección sea rápida y eficaz.

The song and sound library (Biblioteca de canciones y sonido) ofrece una gama de ritmos de apoyo para usar durante la composición, así como numerosas melodías muy conocidas que se pueden tocar y alterar, incluyendo la *Sonata para piano en Mi Bemol* de Mozart y *El vuelo del moscardón*. El *tempo* se puede regular desde un pausado paso de 30 compases por minuto hasta la vertiginosa velocidad de 200 compases por minuto, que ofrecería algunos resultados electrizantes si se aplicara a una fuga de Bach. Si esa velocidad le parece un tanto extravagante, siempre puede poner un ritmo suave después de la *Sonata* de Mozart.

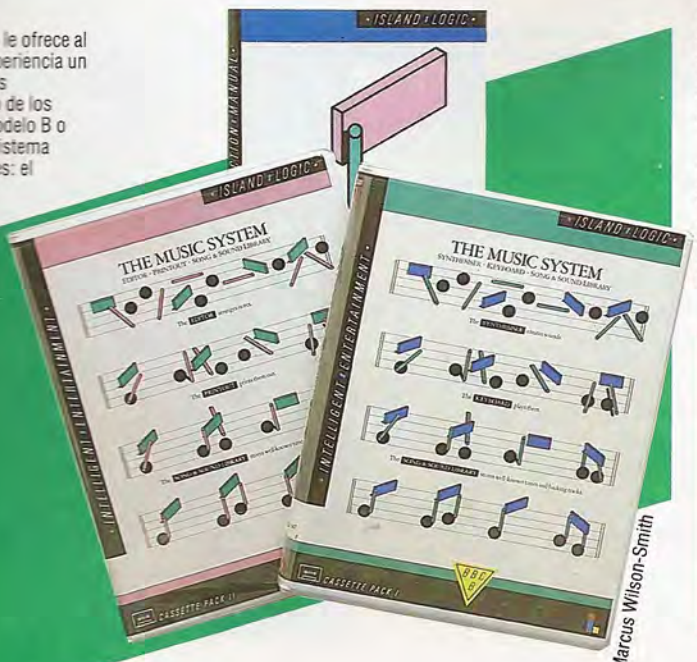
La opción *Synthesiser* (Sintetizador), que se utiliza junto con las opciones *Editor* y *Keyboard* (Teclado), puede crear cualquier sonido que usted desee. Cada sonido se define estableciendo los parámetros de una envolvente, con un máximo de 30 envolventes. La versión en disco también ofrece una opción de 15 efectos de percusión distintos. Al igual que con los otros módulos, la edición es simple. Cada parámetro posee un propio icono en pantalla, y se pueden escuchar los sonidos a medida que se van entrando. Es fácil visualizar gráficos de frecuencia y amplitud mientras los sonidos de *Synthesiser* se pueden guardar en dos archivos y después cargar en el *Editor* o el *Keyboard*.

La opción *Keyboard* visualiza las dos octavas centrales de un teclado de piano completo, actuando el grupo QWERTY del teclado del micro como las teclas blancas, y las teclas numéricas como las negras. Encima del teclado hay una ventana que contiene los iconos de envolvente y de volumen.

Una de las características más notables de este módulo es su simulación de una grabadora de cinta de cuatro pistas. Las pistas de percusión y apoyo se pueden utilizar como fondo y componer música sobre ellas, con lo que usted puede experimentar con su música con gran facilidad.

The music system constituye un programa complejo e incluye dos exhaustivos manuales que es necesario estudiar para sacar el máximo provecho del paquete. Sin embargo, una vez cargado y en plena ejecución, las ventanas e iconos demuestran instantáneamente la sencillez y la eficacia del sistema.

¡Musica maestro!
«The music system» le ofrece al músico con o sin experiencia un control total sobre las facilidades de sonido de los ordenadores BBC Modelo B o Commodore 64. El sistema viene en dos paquetes: el primero contiene los módulos del sintetizador y el teclado; el segundo incluye las facilidades de editor y de impresión



Marcus Wilson-Smith

The music system: Para el BBC Micro Modelo B y el Commodore 64. Cassette Pack 1: consta de Synthesiser, Keyboard, Song & Sound Library; Cassette Pack 2: consta de Editor, Printout, Song & Sound Library
Editado por: Island Logic Limited, 22 St Peter's Square, London W6 9NW, Gran Bretaña
Formato: Cassette
Palancas de mando: No se necesitan

Cortesía de Mecca Bookmakers

Apuestas seguras

Los programas escritos para pronosticar ganadores en las carreras de caballos deben considerar factores tales como las estadísticas y la ley de las probabilidades

Atractivas visualizaciones
Las visualizaciones de información por ordenador mejoradas constituyen apenas una de las muchas aplicaciones a las que las cadenas de apuestas británicas más importantes están destinando la nueva tecnología. Entre otros usos se incluye la monitorización de la actividad de los clientes, la productividad del personal y el cálculo de las posibilidades de opciones de apuestas especiales

Los corredores de apuestas casi podrían considerarse como unos intermediarios que proporcionan un servicio, ofreciendo a los apostadores opiniones diferentes sobre los méritos de los caballos participantes en una carrera dada para que respalden sus apreciaciones con dinero contante y sonante. Las posibilidades que ofrecen los corredores de apuestas reflejan el peso del dinero apostado: cuantas más personas respalden un caballo, más dinero hay para él y, en consecuencia, "menor" es su precio. Además, los corredores intentan calibrar las posibilidades de modo que, en general, estén seguros de quedarse con un porcentaje del total, cualquiera que sea el caballo ganador. De ese modo, en realidad los apostadores pagan los servicios del corredor.

En Gran Bretaña, el mercado de apuestas en realidad cobra fuerza *in situ* en los diez minutos, más o menos, de frenética actividad que preceden a la carrera. Las sesiones de apuestas se transmiten a las agencias de todo el país, de modo que millones de apostadores que no están presentes en el hipódromo puedan hacer su apuesta.

A pesar de los obstáculos para la introducción de la nueva tecnología en la pista, algunos corredores exteriores a las pistas (en particular las grandes cadenas) están confiando cada vez más en los ordenadores para ganar en eficacia y rentabilidad. La firma Ladbrokes, por ejemplo, utiliza en gran medida ordenadores en su sección de apuestas a crédito. Todas las apuestas de los clientes a crédito se registran en micros Cromemco conectados a un disco rígido. Al terminar el día, los datos se transfieren a cinta y se envían a un ordenador central IBM al cual ya se le han proporcionado los resultados de todas las carreras de la jornada. Todas las apuestas, ya sean ganadoras directas, *each-way* o las diversas combinaciones y acumuladores, se determinan, entonces, de forma automática. Las cuentas de los clientes se actualizan semanalmente, y se libran cheques o se realizan requerimientos de pago. Además, el ordenador central se utiliza para las tareas estándares de procesamiento de datos propias de cualquier gran negocio.

En la medida en que las firmas más importantes llegan a considerarse a sí mismas no tanto como corredores sino más bien como una suerte de industria del ocio, existe una creciente tendencia a la utilización de ordenadores. Mecca Bookmakers, por ejemplo, confía en que la reducción del costo de la potencia de los ordenadores allanará el camino hacia un análisis de sus negocios mucho más sofisticado. Se podrían utilizar terminales situados en sucursales seleccionadas para realizar una profunda investigación de las preferencias de los apostadores, haciendo posible la identificación, por ejemplo, de la clase más popular de apuestas, de las facilidades que a la gente le gustaría que se les proporcionara, o del tipo de administración adecuado para incrementar el negocio.

Mecca piensa que la investigación de mercados

asistida por ordenador irá adquiriendo mayor importancia cuando la nueva legislación británica permita, finalmente, que las agencias de apuestas se conviertan en lugares más agradables y acogedores para el público. La empresa ya está utilizando su ordenador central para mejorar las visualizaciones gráficas de información sobre carreras de caballos y otros eventos deportivos, transmitiéndolas en forma espectacular y colorida a oficinas seleccionadas de la cadena. Además, los ordenadores están ayudando a monitorizar el rendimiento y la rentabilidad de las agencias individuales y del grupo como un todo, asegurando que la dirección sepa claramente si está alcanzando o no sus objetivos.

En la actualidad, la mayoría de los corredores exteriores a las pistas ofrecen la posibilidad de predecir los dos o tres primeros corredores por el orden correcto. Estas apuestas se conocen, respectivamente, como el Computer Straight Forecast (CSF: pronóstico directo por ordenador). En ambos casos, el pago se calcula por ordenador de acuerdo con complejas fórmulas previamente establecidas. Cuando se iniciaron los pronósticos de carreras hípicas, algunos corredores se limitaban a multiplicar las posibilidades con el fin de calcular las apuestas ganadoras. No obstante, enseguida se comprendió que su dependencia de los caprichos de las posibilidades de las carreras los dejaba con márgenes inadecuados en el campo de las apuestas pronosticadas, en especial cuando, como sucede frecuentemente, los caballos favoritos terminan primero y segundo.

La fórmula de pago

En Gran Bretaña se ha desarrollado una fórmula para determinar el pago de cada combinación de posibilidades de modo que a los corredores se les garantizaba un margen del 20 %. En 1977 la fórmula se informatizó y nació el CSF, seguido en 1981 por el Tricast. Se introducen modificaciones al programa original (escrito en BASIC) en respuesta a los cambios que se producen en los patrones de apuestas o a los rendimientos del precio de partida.

La alteración más sonada tuvo lugar tras lo que se conoció como el "asunto de Little Owl". En enero de 1982, una carrera de vallas de tres caballos involucraba a un gran favorito (Little Owl), con 11-4 *on* (es decir, 4-11), un segundo favorito con 5-2 y un *rank outsider* con 66-1. Por algún motivo, Little Owl fue frenado recién comenzada la carrera, dejando que el segundo favorito (5-2) llegara a la meta delante del *outsider*. Para algunos corredores de apuestas constituyó un auténtico descalabro. Otros insinuaron veladamente que habían sido víctimas de una maniobra fraudulenta. La adición de una corrección (el "factor armónico") en el programa original aseguró que en competidores que produjeran resultados sorpresa las posibilidades de los *rank outsiders* se redujeran, mientras que las de caballos que gozaran de mayor aceptación se ampliaran ligeramente.

En consecuencia, el ordenador proporciona a los grandes corredores de apuestas exteriores a las pistas una poderosa herramienta para analizar su negocio, identificar áreas de vulnerabilidad y proteger y mejorar sus márgenes, de suma importancia.

Por cuanto concierne al apostador, hay un hecho innegable. Ningún programa diseñado para usar

Desde el hipódromo

Los corredores de apuestas de los hipódromos británicos han desarrollado un complejo sistema de interacción y de comunicación diseñado para satisfacer las demandas del mercado y ponerse a cubierto de las contingencias financieras. Los corredores se dividen en dos categorías: primero están los de mayores vuelos (o *corredores de barandilla*, así llamados debido a la posición que ocupan, junto a las barandillas que separan del público el recinto para socios), quienes sólo aceptan apuestas de clientes acreditados y originan el mercado. Luego están los de la zona del público (*Tattersalls*), quienes aceptan apuestas en efectivo de los espectadores.

En el lapso de unos pocos minutos agitados, antes de la carrera, el mercado de apuestas se convierte en un centro de gran actividad, durante el cual grandes sumas de dinero cambian de mano en miles de transacciones separadas. Durante este período, los corredores se comunican entre sí a través de un *hombre tictac*, quien utilizando sus manos transmite mensajes entre las partes mediante un complicado código de señales. Una de las funciones del *hombre tictac* es facilitar el proceso de "marcar" apuestas. Si un corredor toma conciencia de que se ha puesto a sí mismo en posición de posibles pérdidas con un determinado caballo, "cargará la responsabilidad", colocando, a través del *hombre tictac*, una apuesta sobre ese mismo caballo con otro corredor. Así, si el caballo gana, podrá recuperar parte de sus pérdidas.

Parece poco probable que el ordenador llegue alguna vez a participar en prácticas tan tradicionales, por la sencilla razón de que todo ocurre muy rápido. Comparativamente, sería fácil escribir un programa que simulara las actividades del corredor de apuestas, pero la entrada de los datos sería excesivamente lenta como para hacer frente al torbellino final de apuestas que tiene lugar para cada carrera. Parece ser, entonces, que el ordenador ha hallado en el *hombre tictac* la horma de su zapato

con un micro personal va a convertir al usuario en millonario de la noche a la mañana ni a sacar del negocio al corredor de apuestas. Al saber esto, la mayoría de los posibles apostadores por ordenador podrían sentirse desanimados; pero el micro puede ayudar al apostador juicioso en numerosos sentidos que merecen ser examinados.

Si bien es cierto que por lo general los apostadores deben perder (el margen del corredor y el impuesto sobre las apuestas velan por ello), sería erróneo suponer que todos ellos pierden. Los apostadores más perspicaces (aquellos que poseen una habilidad notoria para leer el formulario y sopesar las posibilidades) ganan, y sus ganancias provienen de quienes son menos capaces. El apostador juicioso dedica considerable tiempo al análisis de los rendimientos de los caballos con el objeto de establecer una imagen clara de sus méritos relativos.

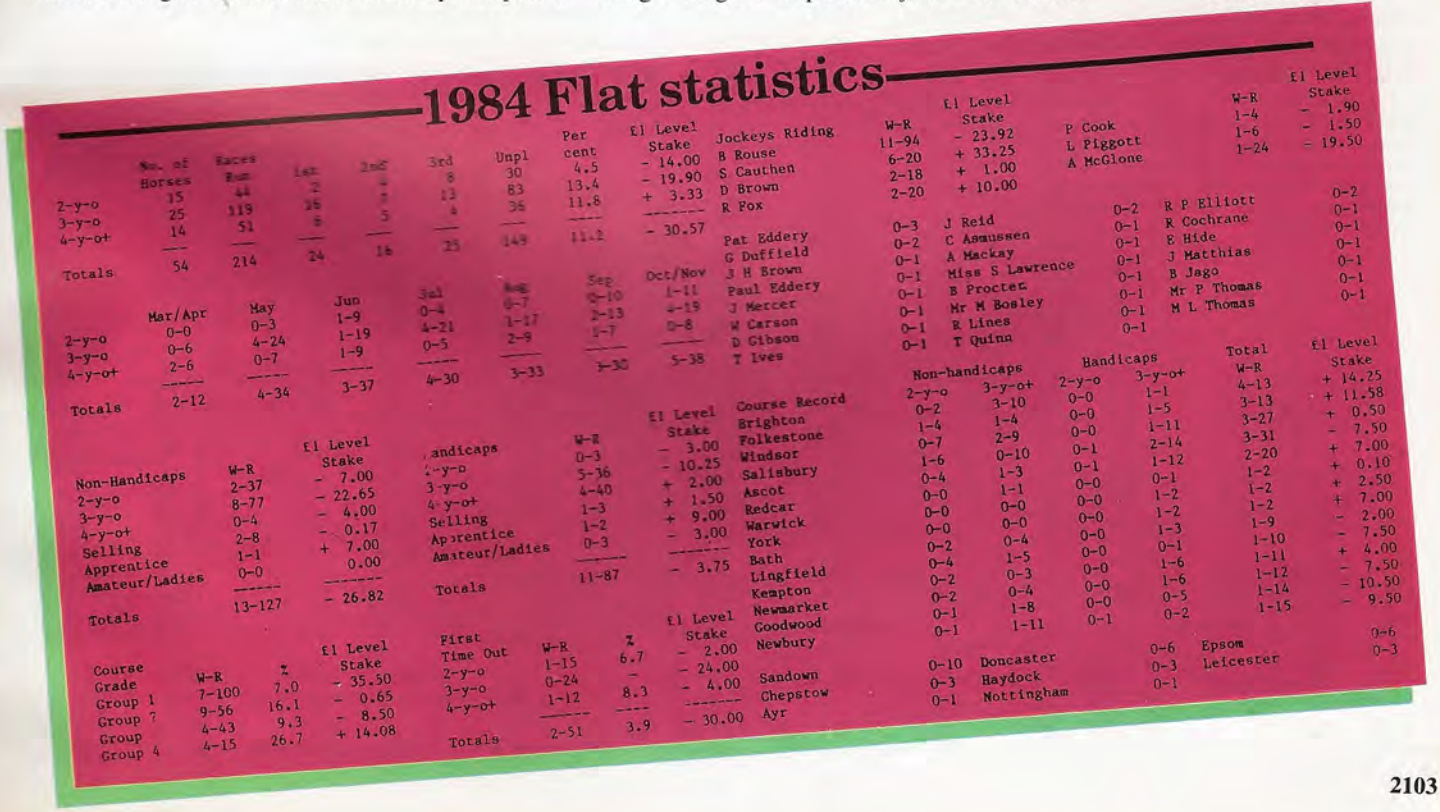
Si el campo se puede reducir a dos o tres con posibilidades evidentes, entonces vale la pena considerar la carrera. Descubrir un caballo con unas *chances* sobresalientes es difícil, pero ocasionalmente viable. Luego se deben tener en cuenta otros factores, tales como la idoneidad del caballo, las condiciones de la carrera (distancia, *going*, jockey, etcétera) y, lo que es vital, la apuesta. Lo que le den por su dinero, por supuesto, es crucial: hallar un caballo con una *chance* obvia con buenas posibilidades. Esto, sin embargo, no es fácil, puesto que, al fin y al cabo, si las *chances* de un caballo son absolutamente evidentes, aun el apostador más incompetente aprovechará la oportunidad, lo que sin duda redundará en que el caballo salga con posibilidades restringidas.

La tarea del apostador de éxito consiste, por consiguiente, en hallar un caballo con *chances* abrumadoras que no atraiga un gran flujo de dinero y que, en consecuencia, es probable que salga con mayores posibilidades que las que debiera. Identificar tales oportunidades de apuesta lleva su tiempo y sólo se consigue raramente, de modo que el principio de una selectividad rigurosa es vital para la estrategia del apostador exitoso.

Simplemente, no hay tiempo para evaluar el formulario en cada una de las seis carreras, como mínimo, de cada reunión hípica. Dado que los apostadores de agencias de apuestas tienen acceso entre dos y cinco reuniones hípicas por tarde, apenas sorprende que la inmensa mayoría opere sobre una consideración del formulario que es poco mejor que una adivinanza lisa y llana. Por supuesto, en ocasiones este enfoque funciona, pero la tendencia general se orienta inexorablemente a la acumulación de pérdidas.

Es aquí donde entran en escena los programas para el apostador. La virtud de estos programas es que imponen al apostador un nivel de selectividad y, por tanto, impiden el enfoque indisciplinado, que es la perdición de tanta gente. Los programas hacen esto debido a lo que algunos considerarían uno de sus principales inconvenientes: la lentitud en la entrada de datos, que puede requerir alrededor de media hora para cada carrera hasta haber entrado todas las variables relevantes. Claro está que en principio el apostador sólo debe considerar aquellas carreras que ofrezcan las mayores posibilidades. Una estrategia recomendable es apostar generalmente en carreras de diez corredores o menos, con lo que se reduce el tiempo de entrada y, a la vez, se aumentan las posibilidades de ganar.

Es improbable, sin embargo, que el apostador por ordenador pueda hacer peligrar los intereses de los corredores de apuestas. El hecho es que tales programas sólo resultan atractivos al apostador juicioso y selectivo, desde cuyo punto de vista probablemente sea mejor que este estado de cosas continúe. Cualquier cambio radical en los patrones de apuestas será rápidamente identificado por la comunidad de corredores de apuestas, y las fuerzas del mercado inevitablemente se combinarán para asegurar que, suceda lo que suceda, el "hombre del talego" se gane su porcentaje al final del día.



1984 Flat statistics

	No. of Horses	Races Run	1st	2nd	3rd	Unpl	Per cent	£1 Level Stake
2-y-o	15	44	2	4	8	30	4.5	- 14.00
3-y-o	25	119	16	7	13	83	13.4	- 19.90
4-y-o+	14	51	6	5	4	36	11.8	+ 3.33
Totals	54	214	24	16	25	149	11.2	- 30.57

Jockeys Riding	W-R	£1 Level Stake
B Rouse	11-94	- 23.92
S Cauthen	6-20	+ 33.25
D Brown	2-18	+ 1.00
R Fox	2-20	+ 10.00
Pat Eddery		
G Duffield		
J H Brown		
Paul Eddery		
J Mercer		
W Carson		
D Gibson		
T Ives		

	W-R	£1 Level Stake
P Cook	1-4	- 1.90
L Piggott	1-6	- 1.50
A McGlone	1-24	- 19.50

	Mar/Apr	May	Jun	Jul	Aug	Sep	Oct/Nov
2-y-o	0-0	0-3	1-9	0-4	0-7	0-10	1-11
3-y-o	0-6	4-24	1-19	4-21	1-17	2-13	4-19
4-y-o+	2-6	0-7	1-9	0-5	2-9	1-7	0-8
Totals	2-12	4-34	3-37	4-30	3-33	3-30	5-38

0-3	J Reid	0-2	R P Elliott
0-2	C Asmussen	0-1	R Cochrane
0-1	A Mackay	0-1	E Hide
0-1	Miss S Lawrence	0-1	J Matthias
0-1	B Procter	0-1	B Jago
0-1	Mr M Bosley	0-1	Mr P Thomas
0-1	R Lines	0-1	M L Thomas
0-1	T Quinn		

	W-R	£1 Level Stake
Non-Handicaps	2-37	- 7.00
2-y-o	8-77	- 22.65
3-y-o	0-4	- 4.00
4-y-o+	2-8	- 0.17
Selling	1-1	+ 7.00
Apprentice	0-0	0.00
Amateur/Ladies		
Totals	13-127	- 26.82

Handicaps	W-R
2-y-o	0-3
3-y-o	5-36
4-y-o+	4-40
Selling	1-3
Apprentice	1-2
Amateur/Ladies	0-3
Totals	11-87

Course Record	£1 Level Stake
Brighton	- 3.00
Folkestone	- 10.25
Windsor	+ 2.00
Salisbury	+ 1.50
Ascot	+ 9.00
Redcar	- 3.00
Warwick	
York	- 3.75
Bath	
Lingfield	
Kempton	
Newmarket	
Goodwood	
Newbury	
Sandown	
Chepstow	
Ayr	

Non-handicaps 2-y-o	Non-handicaps 3-y-o+	Handicaps 2-y-o	Handicaps 3-y-o+
0-2	3-10	0-0	1-1
1-4	1-4	0-0	1-5
0-7	2-9	0-0	1-11
1-6	0-10	0-1	2-14
0-4	1-3	0-1	1-12
0-0	1-1	0-0	0-1
0-0	0-0	0-0	1-2
0-0	0-0	0-0	1-2
0-2	0-4	0-0	1-3
0-4	1-5	0-0	0-1
0-2	0-3	0-0	1-6
0-2	0-4	0-0	1-6
0-1	1-8	0-0	0-5
0-1	1-11	0-1	0-2

	W-R	£1 Level Stake
Total	4-13	+ 14.25
	3-13	+ 11.58
	3-27	+ 0.50
	3-31	- 7.50
	2-20	+ 7.00
	1-2	+ 0.10
	1-2	+ 2.50
	1-2	+ 7.00
	1-9	- 2.00
	1-10	- 7.50
	1-11	+ 4.00
	1-12	- 7.50
	1-14	- 10.50
	1-15	- 9.50

Course Grade	W-R	%	£1 Level Stake
Group 1	7-100	7.0	- 35.50
Group 2	9-56	16.1	- 0.65
Group 3	4-43	9.3	- 8.50
Group 4	4-15	26.7	+ 14.08

First Time Out	W-R	%	£1 Level Stake
2-y-o	1-15	6.7	- 2.00
3-y-o	0-24	-	- 24.00
4-y-o+	1-12	8.3	- 4.00
Totals	2-51	3.9	- 30.00

0-10	Doncaster	0-6	Epsom
0-3	Haydock	0-3	Leicester
0-1	Nottingham	0-1	

Posibilidades razonables
Esta tabla es una muestra típica de las que publica frecuentemente el periódico británico *The Sporting Life*. Ofrece un análisis del rendimiento de un entrenador determinado (en este caso, David Elsworth) durante el año que se indica. Las cifras se desglosan bajo varios subtítulos que pueden interesar al apostador. Sin el ordenador, tales análisis serían imposibles desde el punto de vista del consumo de tiempo, y sólo se han convertido en realidad desde hace pocos años

Hemos llegado a un punto en el que estamos en condiciones de añadir elementos de la "trama". Examinaremos las últimas decisiones que es necesario adoptar en nuestro programa del manipulador de personajes y esbozaremos las estructuras arborescentes implicadas.

La trama se cierra

Hasta ahora hemos proporcionado a nuestros personajes los medios para que manipulen objetos. Lo que necesitamos ahora es añadir las rutinas restantes, que esbozamos en el capítulo anterior. Es en este punto donde se hacen evidentes las ventajas de adoptar un diseño modular para nuestro programa.

El diagrama de flujo de la ilustración representa el proceso completo de toma de decisiones que lleva a cabo el manipulador de personajes. Puede ver que incorpora tanto el árbol de manipulación de objetos, que ya hemos visto en acción, como módulos para manejar la trama, la "conciencia de objetos" y la interacción con otros personajes.

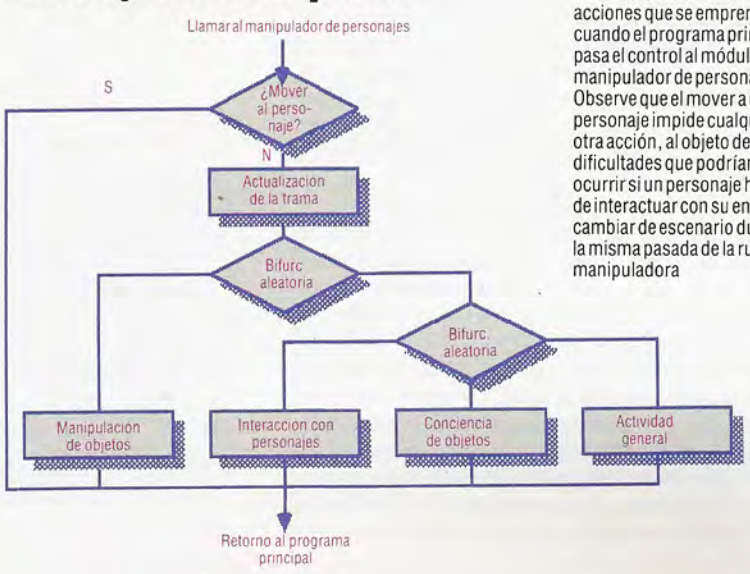
Los rudimentos de la trama ya los hemos analizado: en algún momento durante el curso del juego un infortunado personaje comerá una empanada de carne del Dog and Bucket y morirá por intoxicación. El juego terminará cuando otro personaje haya reunido la información necesaria para demostrar la culpabilidad del barman Fred, cuya costumbre de elaborar las empanadas con alimento para gatos ha dado lugar a este desventurado giro de los acontecimientos. Para resolver el misterio, un personaje debe hallar una lata de alimento para gatos, ver a la víctima y sumar dos más dos.

A los fines de la programación, la información necesaria se almacena en cuatro banderas principales. La primera de ellas ya la hemos visto: la variable numérica g. Ésta se actualiza en la línea 5220 y almacena temporalmente el número del personaje que está comiendo la empanada de carne para que las rutinas de la trama lo procesen. Otras dos banderas registran el fallecimiento de un personaje y si algún personaje ha detectado o no a la víctima. La cuarta bandera asume la forma de una matriz DIMensionada para la cantidad de personajes, con cada elemento inicializado en cero. Cuando un personaje descubre a la víctima, el elemento correspondiente se establece en 255. En el listado completo, que ofreceremos en el próximo capítulo, podremos ver todo esto en acción.

El árbol de la trama comprueba primero si el personaje ya ha muerto y, de ser así, la bandera de la "muerte" se establecerá en el número del personaje. Dado que el programa sólo admite una víctima, en este punto la rutina de la trama retornará sin emprender ninguna otra acción.

Sin embargo, si la bandera de la "muerte" no está establecida, la rutina prosigue comprobando si el personaje ha comido o no un bocado de la empanada. Si el valor de g concuerda con el número de personaje actual, entonces comprobamos si se ha producido alguna muerte (en cuyo caso la bandera de la "muerte"

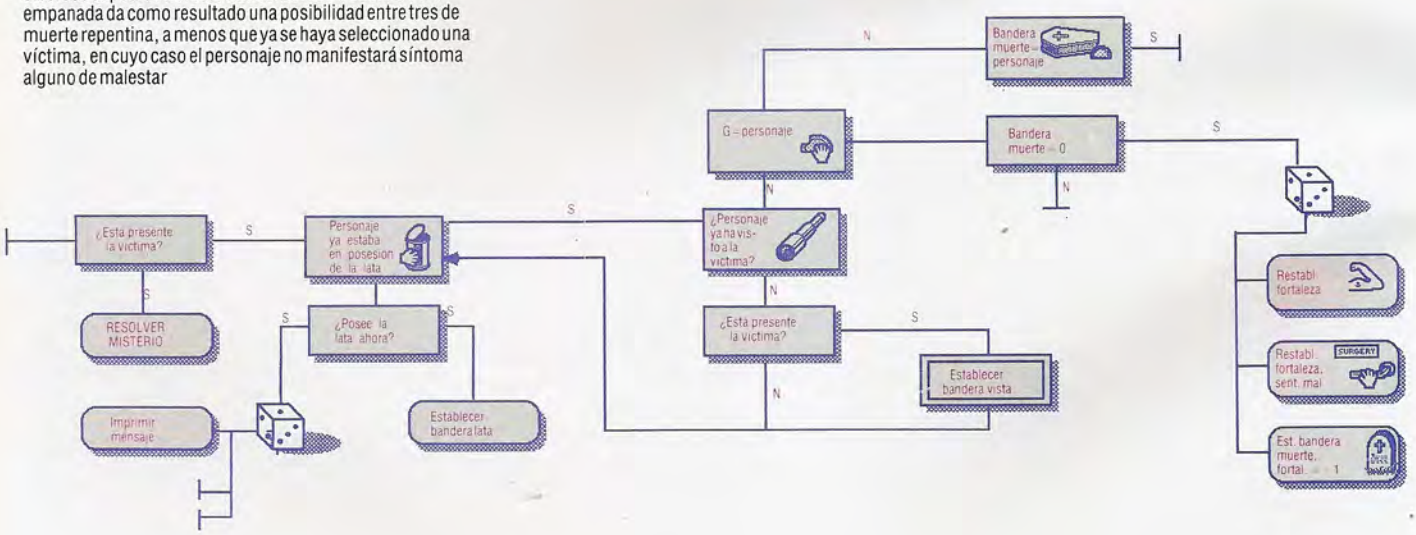
Manejando el reparto



Nuestro diagrama de flujo muestra las diferentes acciones que se emprenden cuando el programa principal pasa el control al módulo manipulador de personajes. Observe que el mover a un personaje impide cualquier otra acción, al objeto de evitar dificultades que podrían ocurrir si un personaje hubiera de interactuar con su entorno y cambiar de escenario durante la misma pasada de la rutina manipuladora

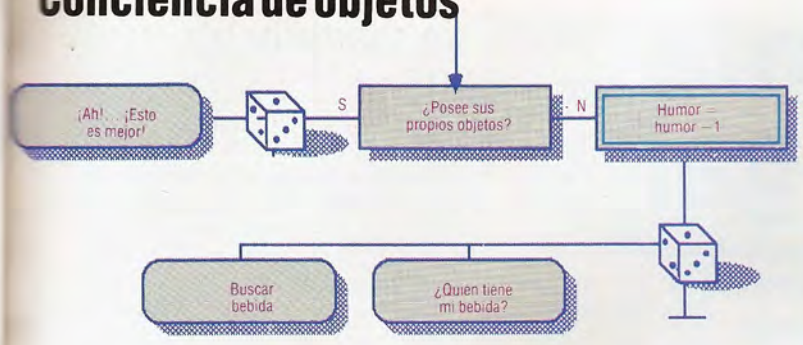
Esquema de la trama

Vemos aquí el flujo de control que se requiere para manejar los diversos aspectos de la trama. Comerse un bocado de la empanada da como resultado una posibilidad entre tres de muerte repentina, a menos que ya se haya seleccionado una víctima, en cuyo caso el personaje no manifestará síntoma alguno de malestar



Ian McKinnell

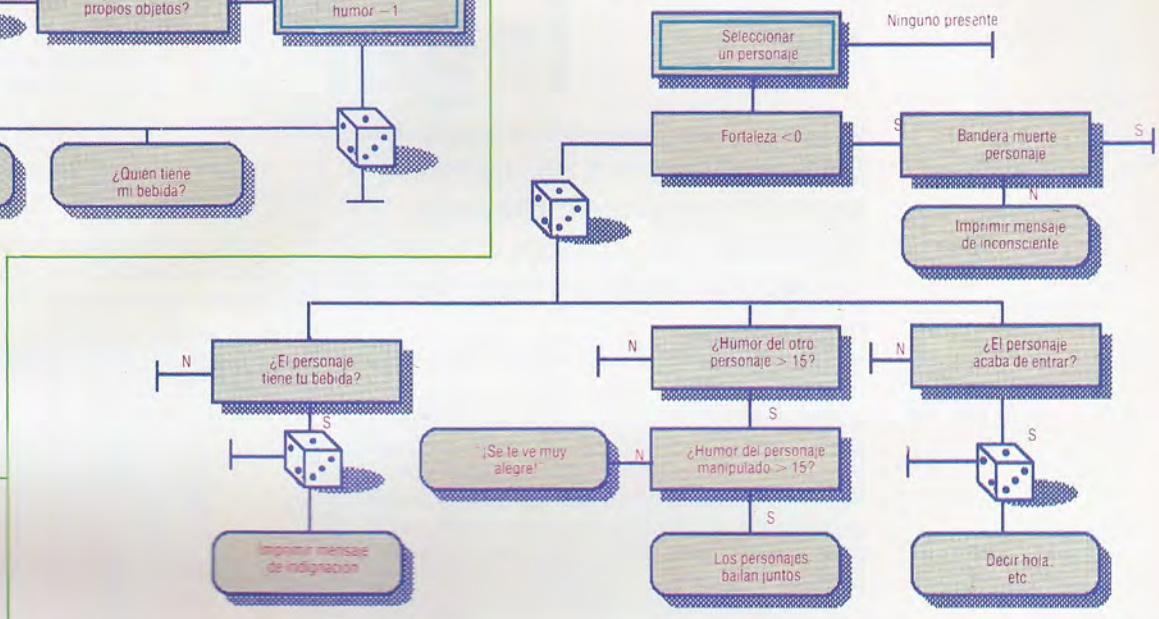
Conciencia de objetos



Aquí añadimos a las rutinas que proporcionaba el árbol de manipulación de objetos algunas otras orientadas a éstos. En especial, el humor del personaje se reduce en uno si no se halla en posesión de su propia bebida

Interacción de personajes

Nuestros personajes necesitan tener cierta conciencia del humor y las acciones de cada uno de los demás. Este árbol les proporciona algunas oportunidades de expresarse por sí mismos



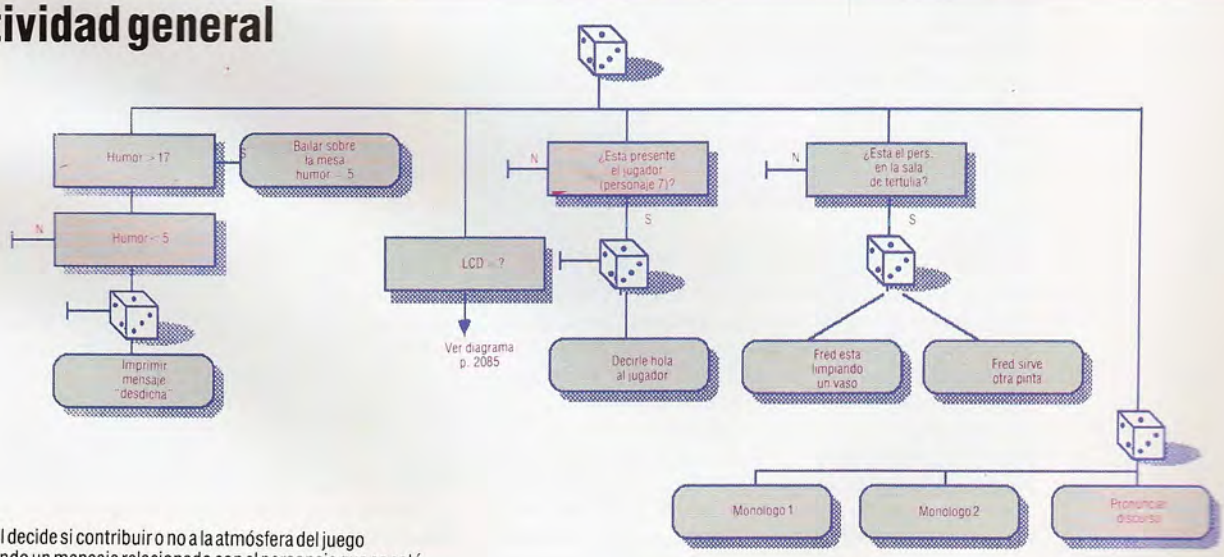
será mayor que cero). Si nadie ha fallecido, el programa se bifurca al azar a una de tres conclusiones. En las dos primeras, el personaje no presentará ningún síntoma de enfermedad o bien sufrirá un repentino dolor de estómago. En ambos casos, la fortaleza del personaje, que la línea 5230 había reducido en 10 puntos, se restablecerá a su nivel anterior.

En el tercer caso, la fortaleza del personaje se reducirá aún más (de ser necesario), a −1, y se establecerá la bandera de la muerte en el número del personaje. Si el personaje no está comiendo la empanada, se llevan a cabo comprobaciones para ver si se han satisfecho o no otras condiciones de la trama. Usted podrá ver exactamente qué sucede recorriendo los distintos caminos a través del árbol. Observe el uso de un nudo aleatorio que se bifurca en tres direcciones para dar una posibilidad entre tres de que se imprima un mensaje, y los dos nudos terminales de la esquina inferior izquierda del árbol, que vuelven a saltar dentro del árbol para permitir más comprobaciones.

Sugerencias

A estas alturas usted ya no tendrá dificultad alguna para "leer" los diagramas de estructuras arborescentes y ver los procesos implicados y las condiciones que se comprueban. Pruebe de seguir el recorrido de los otros árboles que se muestran y decida por sí mismo cómo se podrían entrar en nuestro listado. Quizá le interese considerar las implicaciones de recorrer árboles con diferente número de bifurcaciones desde distintos nudos.

Actividad general



Este árbol decide si contribuir o no a la atmósfera del juego imprimiendo un mensaje relacionado con el personaje que se está manipulando

Caroline Clayton

Perfectamente legal

Desarrollaremos un método sencillo para comprobar la legalidad y la sintaxis de una fórmula entrada en nuestra hoja electrónica

Anteriormente vimos un procedimiento para escribir fórmulas de hoja electrónica denominado *notación polaca inversa*. Por cuanto concierne a nuestro programa de hoja electrónica, la conversión de series de fórmulas escritas "normalmente" (infijos) a notación polaca inversa ofrece dos ventajas. Comprobar la legalidad de las expresiones en polaca inversa es más fácil que comprobar la de infijos y, una vez comprobada, también son más fáciles de evaluar. Veremos de forma detallada cómo se pue-

Aquí mostramos cómo se puede comprobar la legalidad de una serie en polaca inversa.

Al digitar programas, la mayoría de los programadores de BASIC se habrán encontrado con el mensaje de error SYNTAX ERROR en numerosas ocasiones. Las causas más comunes de los errores de sintaxis son los errores de digitación (pulsar una tecla equivocada) y saltarse información vital. Cuando un usuario de hoja electrónica entra una fórmula en una celda, se pueden plantear los mismos problemas. Por ejemplo, podría faltar un paréntesis o no haber suficientes operandos para las operaciones a realizar (pruebe de añadir un número en un espacio en blanco). Existe un método simple para comprobar una serie en notación polaca inversa y asegurar que estén presentes los componentes correctos y por un orden sensato, lo que se ha dado en llamar una serie polaca "bien formada".

La comprobación de que una serie polaca inversa está bien formada se presta idealmente para una aplicación de ordenador. A los elementos que componen una fórmula se les asignan valores así:

- Los operadores binarios (p. ej.,+,−) toman el valor −1
- Los operadores unarios (p. ej., &) toman el valor 0
- Los operandos (p. ej., A2,27) toman el valor 1

Luego se trabaja con la serie polaca inversa con un elemento por vez, de derecha a izquierda; se va llevando un total para los valores de tipos de elementos reseñados arriba. Si cada subtotal es menor o igual que cero y el total final es uno, entonces la expresión es legal. Si algún subtotal es mayor que cero o el total final no es uno, entonces la expresión es incorrecta. El ejemplo ilustra el procedimiento.

Una vez que la serie de infijos, I$, se ha convertido a su equivalente en polaca inversa, P$, la subrutina de la línea 4700 puede realizar una comprobación para asegurar que P$ esté bien formada. Esta rutina va trabajando a través de P$, desde la *izquierda* de a un elemento por vez, colocando los valores de tipos de elementos adecuados (−1, 0 o 1) en una pila, ST(). Habiendo terminado con P$, en la línea 4800 el programa pasa a procesar la pila. Se toma de la pila, ST(), cada tipo de elemento de a uno por vez, y se suma a un total. Si en algún momento durante este proceso (antes de que se saque de la pila el elemento final) al subtotal se hace superior a cero, la rutina se aborta con un mensaje ERROR ANTES DE TERMINAR. Tras procesar la pila por completo, el total final debe ser 1 si la expresión está bien formada. De no ser así, se genera un mensaje de error ERROR AL TERMINAR.

Dado que los operandos de la expresión pueden tomar la forma de nombres de celdas, como A2 o

Cómo se comprueba la legalidad

Para comprobar si el proceso de una serie polaca está bien formado, tomemos un ejemplo sencillo. Si partimos de una fórmula para una celda como ésta:

(A2+A3)*(B2−B3)

entonces la versión en polaca inversa es:

A2A3+B2B3−*

Esta lista refleja cómo se puede explorar empezando desde la derecha, llevando subtotales de los valores de tipos de elementos a medida que avanza:

Elemento	Valor tipo de elem.	Subtotal
*	−1	−1
−	−1	−2
B3	1	−1
B2	1	0
+	−1	−1
A3	1	0
A2	1	1

Al explorar una serie polaca inversa desde la derecha, siempre hay que encontrar un operador antes de los operandos con los que se relaciona. Por tanto, el subtotal debe ser negativo o, si se han encontrado ambos operandos, cero. La excepción la constituye el total final (que debe ser uno), porque en una serie siempre hay un operando más que operadores binarios.

Si por alguna razón el operador * final faltara en el ejemplo anterior, entonces el método detectaría inmediatamente el error:

Elemento	Valor tipo de elem.	Subtotal
−	−1	−1
B3	1	0
B2	1	1**ERROR**
+		
A3		
A2		

Probando las rutinas

En esta etapa se pueden probar las rutinas de conversión a notación polaca inversa y de comprobación de legalidad introduciendo en el programa una sencilla alteración. Digite el listado que ofrecemos aquí y luego cambie la línea 4010 de modo que rece:

```
4010 LET I$="tu serie de infijos"
```

Después entre las siguientes instrucciones en modalidad inmediata:

```
GOSUB 3000:GOSUB 4000:PRINT P$
```

Sinclair Spectrum:

```
4700>REM ********************************
4701 REM * COMPROBAR POLACA INVERSA      *
4702 REM * BIEN FORMADA                  *
4703 REM ********************************
4705 GO SUB 4900
4710 LET P=0: LET SP=1: LET L1=1: LET US""
4720 LET P=P+1: IF P > LEN (PS) THEN GO TO
     4800
4730 LET T$=P$(P)
4740 IF T$="+" OR T$="-" OR T$="*" THEN
     LET S(SP)=-1: LET G$(SP)= T$: LET
     SP=SP+1: GO TO 4720
4745 IF T$="/" OR T$="^" THEN LET S(SP)=-1:
     LET G$(SP)=T$: LET SP= SP+1: GO TO 4720
4747 IF L1=LP THEN GO TO 4720
4750 LET U$=E$(LI)
4751 FOR Z=1 TO LEN(U$)
4752 IF U$(Z)=" " THEN GO TO 4754
4753 NEXT Z
4754 LET U$=U$(1 TO Z-1)
4760 LET U$<> P$(P TO P-1+LEN (U$)) THEN
     LET U$="": GO TO 4720
4770 LET S(SP)=1: LET G$(SP)=U$
4775 LET SP=SP+1
4780 IF L1<LP THEN LET L1=L1+1
4785 LET U$""
4790 GO TO 4720
4800 REM *** PROCESAR PILA *****
4810 LET P=S(SP)
4820 FOR C=SP-1 TO 1 STEP -1
4830 LET P=P+ S(C)
4840 IF P> 0 AND C<>1 THEN PRINT AT
     20,0;"ERROR ANTES DE TERMINAR":
     RETURN
4850 NEXT C
4860 IF P<> 1 THEN PRINT " ERROR AL
     TERMINAR ": RETURN
4870 LET CP=SP-1: RETURN
4900 REM *** HACER LISTA DE OPERANDOS *****
4905 LET P=0: LET LP=1: LET T$="": LET U$=""
4910 LET P=P+1: IF P>LEN (I$)THEN GO
     TO 4995
4920 LET T$=I$(P)
4930 IF T$="+" OR T$="-" OR T$="*" OR
     T$="&" THEN GO TO 4960
4940 IF T$="/" OR T$="^" OR T$="(" OR T$=")"
     THEN GO TO 4960
4950 LET U$=U$+T$: TO TO 4910
4960 IF U$="" THEN GO TO 4910
4970 LET E$(LP)=U$: LET LP=LP+1
4980 LET U$=""
4990 GO TO 4910
4995 IF U$="" THEN RETURN
4997 LET E$(LP)=U$: LET LP=LP+1
4998 RETURN
```

Complementos al BASIC

BBC Micro:
Introduzca las siguientes modificaciones en la versión para el Commodore 64:

```
4840 IF P>1 AND C<>1 THEN PRINT
     TAB(0,22);"ERROR ANTES DE
     TERMINAR ":RETURN
4860 IF P<>1 THEN PRINT TAB(0,22);"
     ERROR AL TERMINAR ":RETURN
```

Amstrad CPC 464/664:
Introduzca las siguientes modificaciones en la versión para el Commodore 64:

```
4840 IF P>1 AND C<>1 THEN LOCATE 1,22:
     PRINT " ERROR ANTES DE
     TERMINAR ":RETURN
4860 IF P<>1 THEN LOCATE 1,22:PRINT"
     ERROR AL TERMINAR ":RETURN
```

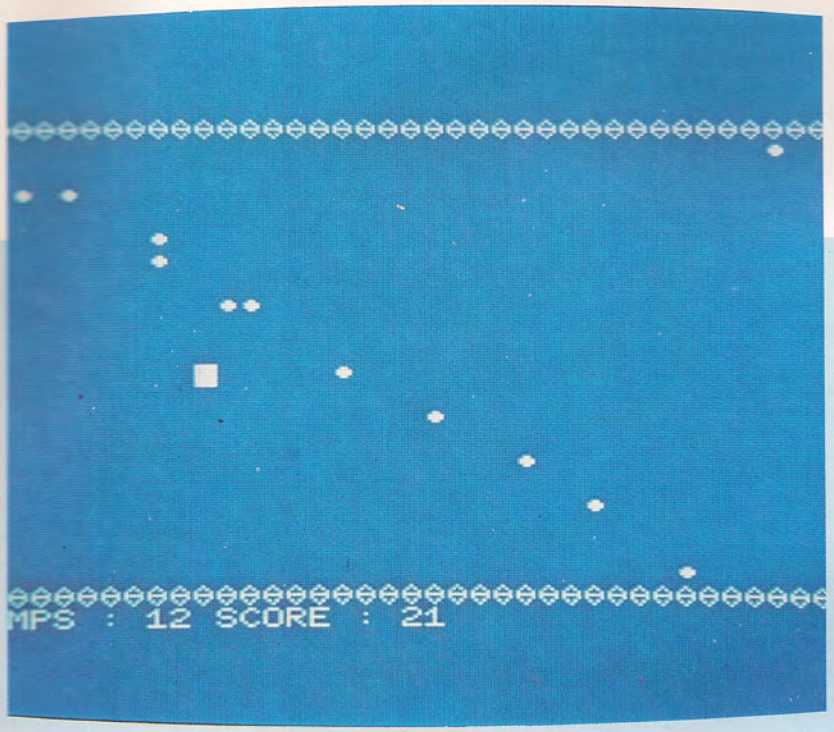
Commodore 64:

```
4700 REM*****************************************
4701 REM * COMPROBAR SERIE POLACA                *
4702 REM * INVERSA BIEN FORMADA                  *
4703 REM *****************************************
4705 GOSUB 4900
4710 P=0: SP=1: L1=1: TE$=""
4720 P=P+: IF P> LEN(P$)THEN 4800
4730 T$=MID$(PS,P,1)
4740 IF T$="+" OR T$="-" OR T$="*" THEN
     ST(SP)=-1:G$(SP)=T$:SP=SP + 1: GOTO 4720
4745 IF T$="/" OR T$"^" THEN
     ST(SP)=-1:G$(SP)=T$:SP=SP+1:GOTO 4720
4746 IF T$="&" THEN ST(SP)=0:G$(SP)=T$:SP=SP+:GOTO
     4720
4747 IF L1=LP THEN 4720
4750 LET TE$=E$(L1)
4760 IF TE$<>MID$(P$,P,LEN(TE$)) THEN TE$="":GOTO
     4720
4770 LET ST(SP)=1:LET G$(SP)TE$
4775 LET SP=SP+1
4780 LET L1=L1-(L1<LP):TE$=""
4790 GOTO 4720
4800 REM *** PROCESAR PILA ***
4810 LET P=ST(SP)
4820 FOR C=SP-1 TO 1 STEP -1
4830 LET P=P+ST(C)
4840 IF P> 0 AND C <> 1 THEN GOSUB 1950:PRINT" ERROR
     ANTES DE TERMINAR ":RETURN
4850 NEXT C
4860 IF P<> 1 THE GOSUB 1950:PRINT " ERROR AL
     TERMINAR ": RETURN
4870 LET CP=SP-1:RETURN
4900 REM ** HACER LISTA DE OPERANDOS
     DE I$ **
4905 LET P=0:LET LP=1:LET T$="":LET TE$=""
4910 FOR K=1 T020:LET E$(K)="":NEXT K
4915 LET P=P+:IF P>LEN(I$) THEN 4995
4920 LET T$=MID$(I$,P,1)
4930 IF T$="+" OR T$="-" OR T$="*" OR T$="&" THEN
     4960
4940 IF T$="/" OR T$="^" OR T$="(" IR T$=")"
     THEN 4960
4950 LET TE$=TE$+T$:GOTO 4915
4960 IF TE$="" THEN 4915
4870 LET E$(LP)=TE$:LP=LP+1
4980 LET TE$=""
4990 GOTO 4915
4995 IF TE$="" THEN RETURN
4996 LET E$(LP)=TE$:LET LP=LP+1
4997 RETURN
```

L14, es importante aislarlos antes de comprobar la legalidad. Puesto que es más fácil identificarlos mientras la expresión está en su forma de infijos (porque todos los operandos están separados por operadores), la subrutina de la línea 4900 trabaja con la versión de infijos de la fórmula, I$, almacenando los nombres de operandos que vaya encontrando en E$( ).

Recogedor

Es posible que usted ya conozca este juego, que propone una curiosa forma de utilizar su ordenador. La versión que presentamos está destinada a los ordenadores MSX



Usted ha de intentar recoger lo más rápidamente posible las migajas que recubren el mantel. Dispone de 30 segundos para conseguir su limpieza total. Las migajas están representadas por puntos blancos. Las teclas de control del cursor, permitirán dirigir su recogedor.

```
10 REM ·····················
20 REM *      RECOGEDOR      *
30 REM ·····················
40 KEY OFF
50 GOSUB 260
60 LOCATE 0,23,0
70 PRINT "TIEMPO:";INT(Z);
   "PUNTUACION: ";S;
80 IF Z<1 THEN 580
90 K=STICK(0)
100 D1=(K=7) - (K=3)
110 D2=(K=1) - (K=5)
120 IF D1<>0 THEN DX=D1:DY=0
130 IF D2<>0 THEN DY=D2:DX=0
140 XP=PX+DX
150 YP=PY+DY
160 CR=VPEEK (XP+YP*40)
170 IF CR=J OR CR=B THEN XP=PX:YP=PY
180 IF CR=M THEN S=S+1:BEEP:X=X+1
190 VPOKE PX+PY*40,N
200 VPOKE XP+YP*40,J
210 PX=XP
220 PY=YP
230 Z=Z-.1
240 IF X=NM THEN 720
250 GOTO 60
260 SCREEN 0,0
270 COLOR 15,4,5
280 DEFINT A-Y
290 B=233
300 M=249
310 N=32
320 S=0
330 NM=10
340 X=0
350 FOR PX=0 TO 39
360 VPOKE PX+40,B
370 VPOKE 880+PX,B
380 NEXT PX
390 FOR PY=2 TO 21
400 VPOKE PY*40,B
410 VPOKE PY*40+39,B
420 NEXT PY
430 FOR I=1 TO NM
440 PX=INT (RND(-TIME)*37)+1
450 PY=INT (RND(-TIME)*21)+2
460 IF VPEEK(PX+PY*40)<>32 THEN 440
470 VPOKE PX+PY*40,M
480 NEXT I
490 PX=INT (RND(-TIME)*37)+1
500 PY=INT (RND(-TIME)*21)+2
510 IF VPEEK(PX+PY*40)<>32 THEN 490
520 VPOKE PX+PY*40,J
530 Z=30
540 DX=0
550 DY=0
560 J=219
570 RETURN
580 FOR I=1 TO 500
590 NEXT I
600 IF INKEY$<>""THEN 600
610 IF S>R THEN R=S
620 LOCATE 13,10
630 PRINT "RECORD:";R;
640 LOCATE 13,16
650 PRINT "OTRA?";
660 D$=INKEY$
670 IF D$="" THEN 660
680 IF D$<>"N" AND D$<>"n" THEN 50
690 CLS
700 LOCATE 0,0,1
710 END
720 NM=NM+1
730 VPOKE XP+YP*40,N
740 GOSUB 340
750 GOTO 60
```

Gestor eficiente

La firma Amstrad confía en que su flamante micro CPC 6128 se introduzca con buen pie en el importante mercado de gestión

Chris Stevens

Un año después del lanzamiento del CPC 464, Amstrad ha captado el 25% del mercado de ordenadores personales de Gran Bretaña. El éxito de esta máquina basada en cassette se basó en el mismo concepto que los productos de *hi-fi* de Amstrad: empaquetar juntos todos los componentes del sistema, proporcionando las facilidades que desean la mayoría de los usuarios y vendiendo la unidad a un precio competitivo. El siguiente micro de Amstrad fue una versión mejorada del CPC 464, que incluía una unidad de disco integral y un BASIC ligeramente ampliado. El CPC 664 y la unidad de disco accesoria DDI-1 para el 464 configuraban CP/M y una versión del Dr LOGO de Digital Research. Pero apenas unos meses después Amstrad lanzaba el CPC 6128, una máquina similar en muchos sentidos al 664, pero con 128 K de RAM. El ordenador 6128 posee un aspecto más serio. Ha sido despojado de las teclas de control de color, que se han reemplazado por un elegante teclado gris uniforme. En los CPC 464/664 se suministraba un relleno numérico separado y un racimo de teclas para el cursor. En el 6128 estas teclas se han integrado en un único banco, haciendo que la anchura general de la nueva máquina sea unos 7 cm menor que la del 464. También se ha reducido la altura de la carcasa, de modo que ahora las teclas descansan a una altura que hace más cómoda la digitación. Además, las teclas poseen menor recorrido, proporcionando al digitar un tacto mucho más seguro.

Al igual que en el CPC 664, en la parte posterior de la máquina se proporcionan puertas para segunda unidad de disco, ampliación e impresora Centronics; pero las puertas para cassette, palanca de mando y audio se han desplazado hacia el lado izquierdo de la carcasa, mientras que los mandos de alimentación y de volumen se han trasladado del lado derecho a la parte posterior. La unidad de disco es la de 3 pulgadas de una sola cara utilizada en el CPC 664, pero el mecanismo se ha alojado en una carcasa mucho más fina y tapado con gráficos de referencia de números para las teclas y colores de pantalla.

Con el CPC 6128 se suministran dos discos. El primero de ellos incluye en una cara una versión mejorada de CP/M, denominada CP/M Plus. La misma está apoyada con varias utilidades de programación, incluyendo un ensamblador, desensamblador y utilidades de manejo de discos, además de las utilidades CP/M habituales, como PIP y SUBMIT. El segundo disco tiene una versión de 48 K del Dr LOGO, que representa una notable mejora con respecto a la versión empaquetada del CPC 664 y del DDI-1, con más de 50 instrucciones adicionales y primitivas.

En la cara superior del segundo disco también se incluyen el programa GSX de Digital Research y una facilidad HELP (ayuda). GSX fue el precursor del entorno GEM de DR, aunque no se hizo muy popular entre las empresas de software. La idea del GSX es que proporciona una "interface para gráficos de dispositivo transparente", lo que significa que las rutinas gráficas escritas utilizando el GSX en una máquina basada en Z80 o 8086 se ejecutará bajo GSX en cualquier otra máquina de base similar.

Si bien la capacidad de direccionamiento de un procesador de ocho bits es de 64 K, el 6128, basado en el Z80A de ocho bits, de algún modo se las arregla para doblar esta cantidad. Ello es posible en virtud del proceso que se conoce como conmutación de bancos, en el cual se pueden "conectar" y "desconectar" distintas áreas de ROM y RAM de los 64 K de espacio de direccionamiento del procesador.

La utilidad BANKMAN del disco de sistemas añade instrucciones adicionales al BASIC, permitiéndole manipular la memoria extra. Los 128 K de RAM están acomodados en dos secciones de 64 K, pero los programas en BASIC sólo pueden ocupar una de estas secciones, impidiendo que en el 6128 se escri-

Tras los pasos del éxito
El ordenador CPC 6128 de Amstrad sigue las huellas de las celebradas máquinas CPC 464 y 664, manteniendo la compatibilidad de software con sus predecesoras e incrementando al mismo tiempo su memoria a 128 Kbytes. La versión mejorada de CP/M que viene con el 6128 permite ejecutar paquetes de gestión CP/M clásicos

CP/M Plus y mucho más

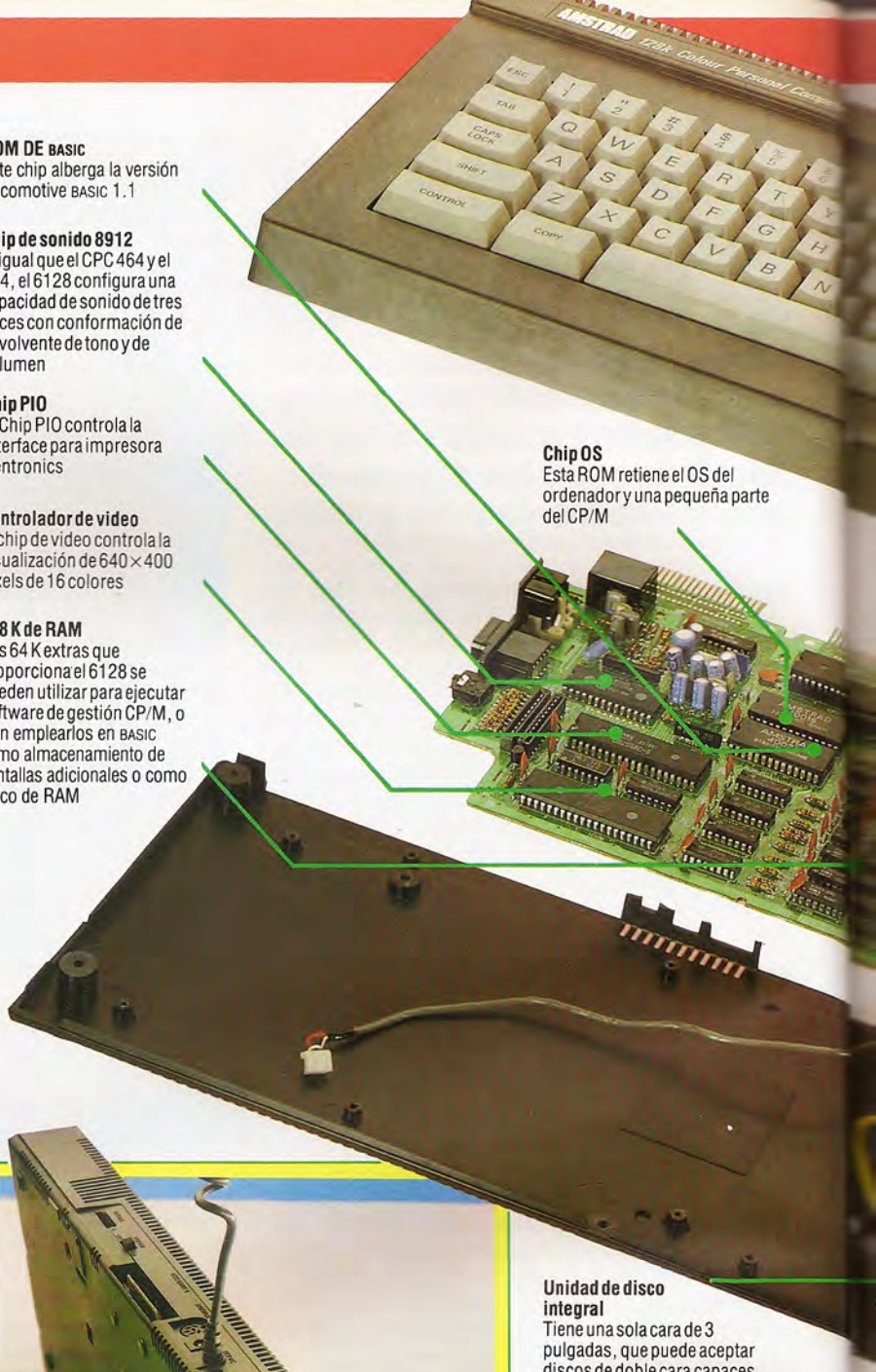
El CP/M Plus, la nueva versión de CP/M que se suministra con el CPC 6128, posee varias ventajas sobre el CP/M 2.2 utilizado en el CPC 664 y el 464 con unidad de disco DDI-1. El CP/M Plus emplea la memoria extra del 6128 para dejar libres 61,5 K de RAM para programas de aplicaciones una vez instalado. Esto es entre 3 y 4 K más de lo que se requiere para programas CP/M estándares y, por lo tanto, permite aprovechar toda una variedad de software de gestión CP/M disponible, incluyendo *SuperCalc*, *dBase II* y *WordStar*. Muchos paquetes CP/M están escritos para unidades de disco gemelas o de doble cara, e intentar acceder a una unidad que no esté presente da por resultado un error. El CP/M Plus se ha modificado para sortear este problema visualizando los mensajes de acción correctiva, de modo que el disco adecuado se pueda insertar de forma manual si así se requiriera. El CP/M Plus también incluye una facilidad para instalar juegos de caracteres de idiomas extranjeros y otra para actuar como terminal VT52 para un miniordenador u ordenador central

Espacio para trabajar

La versión mejorada del Dr LOGO suministrada con el CPC 6128 incluye instrucciones que estaban excluidas en la implementación original del CPC 664. Se le han incorporado instrucciones adicionales para procesamiento de listas, aritmética, gráficos, edición y disco, para hacer de ella una versión mucho más completa y útil. El tamaño global del espacio de trabajo ha pasado de los 2 105 nudos del CPC 664 a 3 753. Son de agradecer las características para acceder a una impresora y puertas E/S; a través de estas últimas es posible controlar tortugas móviles desde LOGO. Se incluyen instrucciones mejoradas de manipulación de disco que permiten cambiar el nombre de los archivos de un disco y seleccionar unidades alternativas desde el propio lenguaje. Instrucciones mejoradas de edición y depuración ofrecen la posibilidad de cargar archivos directamente desde disco en el editor de pantalla del LOGO, y listar en el espacio de trabajo variables globales y procedimientos

ban programas más amplios de los que podrían escribirse en el CPC 464 o 664. No obstante, la sección de 64 K que no se utiliza para programas se puede emplear como zona de almacenamiento a la que se puede acceder desde BASIC. Las nuevas instrucciones permiten utilizar esta memoria extra ya sea para almacenar visualizaciones en pantalla adicionales, que se pueden pasar a la zona de pantalla normal, o bien actuar como un sistema de archivo de registros.

Debido a que la visualización en pantalla exige 16 K, los 64 K de la sección extra pueden retener cuatro pantallas adicionales, numeradas del 2 al 5. La pantalla 1, la estándar, se retiene en la otra sección de 64 K utilizada por el BASIC. La instrucción SCREENCOPY,A,B copia la pantalla B en la zona de 16 K de la pantalla A, escribiendo sobre el contenido original, mientras que SCREENSWAP,A,B intercambia los contenidos de las dos zonas de pantalla.



ROM DE BASIC
Este chip alberga la versión Locomotive BASIC 1.1

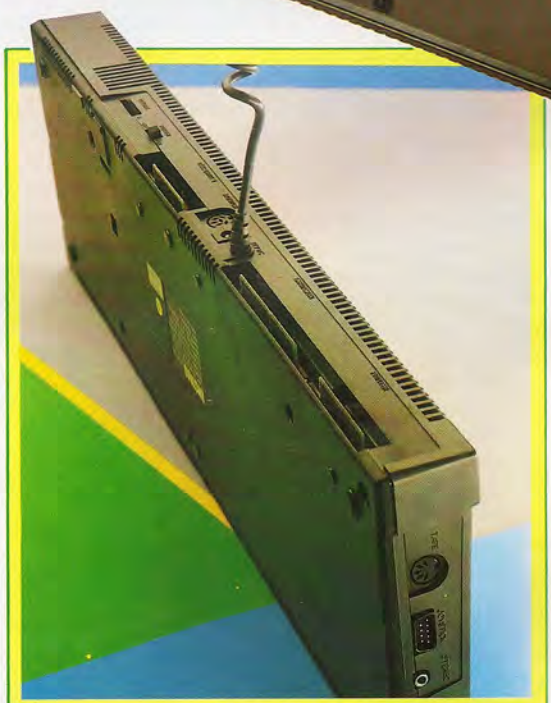
Chip de sonido 8912
Al igual que el CPC 464 y el 664, el 6128 configura una capacidad de sonido de tres voces con conformación de envolvente de tono y de volumen

Chip PIO
El Chip PIO controla la interface para impresora Centronics

Controlador de video
El chip de video controla la visualización de 640×400 pixels de 16 colores

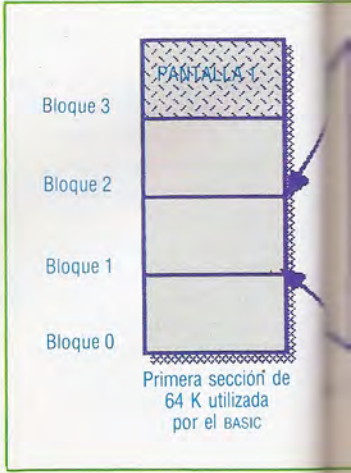
128 K de RAM
Los 64 K extras que proporciona el 6128 se pueden utilizar para ejecutar software de gestión CP/M, o bien emplearlos en BASIC como almacenamiento de pantallas adicionales o como disco de RAM

Chip OS
Esta ROM retiene el OS del ordenador y una pequeña parte del CP/M



Accesos para puertas
El 6128 configura puertas para impresora Centronics, ampliación, segunda unidad de disco, palanca de mando, sonido estéreo y cassette. También hay presentes conexiones para 5 V, 12 V y pantalla, y se acoplan a la pantalla proporcionada

Unidad de disco integral
Tiene una sola cara de 3 pulgadas, que puede aceptar discos de doble cara capaces de almacenar 170 K por cada cara, aunque a la cara inferior sólo se puede acceder dando la vuelta al mismo.

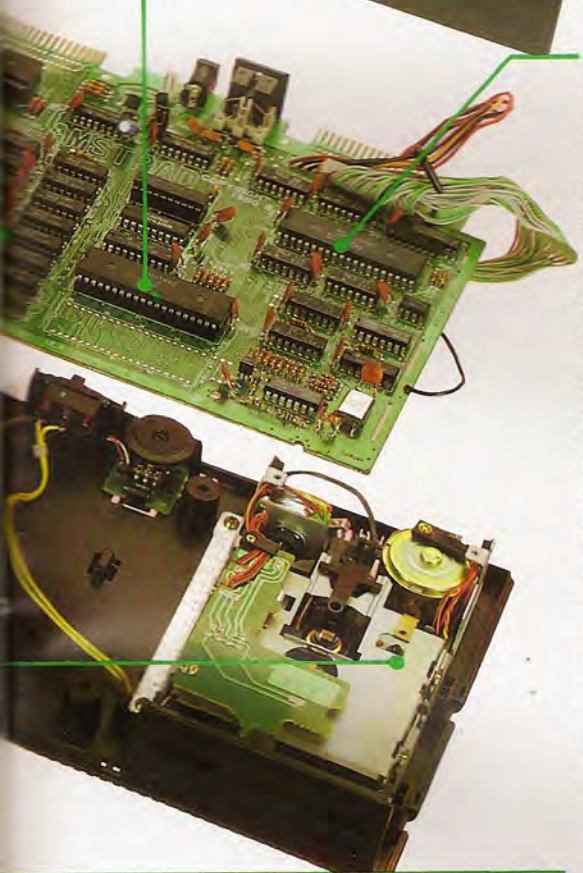


Primera sección de 64 K utilizada por el BASIC

ULA
Este chip construido a medida sincroniza las operaciones internas, incluyendo la conmutación de bancos de memoria y la temporización de pantalla

Teclas de función y del cursor
Para reducir las dimensiones globales de la unidad, Amstrad ha trasladado las teclas de función y del cursor más cerca del teclado principal. Ahora la tecla Return está rodeada por otras, con lo que aumentan las posibilidades de cometer errores al digitar

Controlador de disco flexible
Este chip maneja la unidad de disco integrada y la segunda unidad opcional



Un intercambio de pantalla puede emplear medio segundo, pero es posible intercambiar o copiar una 64ava parte de la pantalla por vez. Es probable que los escritores de juegos aprovechen cabalmente estas facilidades para cambios rápidos en pequeñas porciones de la pantalla.

La sección de 64 K extra también se puede utilizar como un archivo, en cuyo caso la memoria extra se dispone en varios registros donde se pueden almacenar series en BASIC: el disco de RAM, así llamado en razón de su similitud con la estructura de un archivo en disco. Todos los registros del archivo deben tener la misma longitud, que se establece mediante la sentencia IBANKOPEN,n, donde n es la longitud de cada registro, de entre 2 y 255 caracteres. IBANKREAD e IBANKWRITE permiten pasar los datos en serie desde o hacia el disco de RAM, y con estas instrucciones se puede pasar un número de registro opcional que especifica el registro a utilizar. Si no se especifica ningún número de registro, el BASIC comenzará a escribir o a leer empleando el último registro especificado (o el primer registro) e incrementará automáticamente un señalador listo para ocuparse del siguiente registro. En otras palabras, se puede crear y utilizar tanto un archivo de acceso aleatorio como uno secuencial.

El CPC 6128 es una máquina compacta y estilizada que ofrece una buena relación precio-prestaciones. La principal crítica que se le hizo al CPC 664 era que, si bien se suministraba con CP/M, no tenía memoria suficiente como para considerarlo un ordenador serio para uso en pequeña gestión. La ampliación de la RAM a 128 K, junto con una versión de CP/M mejorada capaz de utilizar la memoria extra, le permiten al 6128 ejecutar muchos paquetes CP/M estándares. Asimismo, la nueva máquina también es compatible, desde el punto de vista del software, con el CPC 464 y el 664 y, en consecuencia, dispone de numerosos paquetes elaborados a su medida. Con estos atributos, el CPC 6128 representa una propuesta muy atractiva para el usuario que desee una máquina de juegos y una máquina de gestión, todo en una.

Cambio de dirección

El BASIC suministrado con el 6128 no puede acceder a la totalidad de los 128 K de RAM, y sólo puede trabajar con programas comprendidos en la primera sección de 64 K. Los otros 64 K se pueden utilizar como un disco de RAM o para almacenar cuatro visualizaciones alternativas. Ya que el procesador Z80A sólo puede direccionar 64 K por vez, la memoria extra se divide en bloques de 16 K, de modo que cualquier bloque se puede conmutar temporalmente entre las direcciones &4000 y &7FFF

AMSTRAD CPC 6128

MEMORIA

128 K de RAM, 48 K de ROM. Sólo hay 64 K disponibles para programas en BASIC, pero los otros 64 K se pueden utilizar como disco de RAM o para almacenar hasta cuatro visualizaciones de pantalla alternativas

CPU

Z80A trabajando a 4MHz

DISCO

Una unidad de tres pulgadas de una sola cara, con una puerta para una segunda unidad

INTERFACES

Puertas para bus de ampliación, interface para segunda unidad de disco flexible, cassette, estéreo, palanca de mando e impresora Centronics, entradas de 12 V y 5 V, conector para pantalla

SOFTWARE SUMINISTRADO

Dos discos que contienen CP/M Plus, utilidades de programación, Dr LOGO mejorado y GSX. Asimismo, se proporcionan CP/M 2.2 y Dr LOGO reducido por razones de compatibilidad con el 664 y el 464

DOCUMENTACION

El manual de instrucciones para el usuario cubre la programación en BASIC y las ampliaciones de gestión de memoria al sistema. Las secciones dedicadas al Dr LOGO y al CP/M cubren los aspectos básicos de operación, pero Amstrad ofrece manuales más detallados sobre estos temas

VENTAJAS

El CPC 6128 representa el límite de la tecnología del ordenador de 8 bits y su relación precio-prestaciones es excelente. Su capacidad para ejecutar paquetes CP/M estándares, junto con su excelente BASIC y gráficos, lo hacen igualmente útil para aplicaciones de gestión y de juegos

DESVENTAJAS

Amstrad ha vuelto a utilizar unidades de 3 pulgadas en vez de las unidades de 3½ pulgadas que emplean muchas máquinas de pequeña gestión. Debido a que las unidades son de una sola cara y que algunos programas utilizan más de una cara de un disco, es probable que los usuarios hayan de ocupar cierto margen de tiempo en manipular discos

Adaptación musical

Adaptaremos la interface para que pueda ser utilizada con la gama Amstrad CPC

El diseño de hardware de la interface para el Amstrad es básicamente el mismo que el de la versión para el BBC Micro y el Commodore 64. Las principales diferencias provienen del hecho de que la máquina Amstrad se basa en el procesador Z80 en vez del 65XX que utilizan las máquinas del diseño original. En consecuencia, el bus del Z80 no es compatible con el dispositivo ACIA MC6850 empleado en la interface.

El circuito del bus de la interface requiere un componente adicional: un IC que contenga tres puertas NAND de tres entradas. También se requieren las dos líneas *select* del chip ACIA, CS0 y CS1 (en el circuito original estaban conectadas directamente a +5 V). La línea *enable* del chip ACIA es básicamente lo inverso de la línea $\overline{\text{IORQ}}$ (*I/O request*: solicitud de E/S) del Z80, pero está sincronizado por M1 para impedir el acceso E/S durante la secuencia de conocimiento de interrupción del Z80, en el curso de la cual $\overline{\text{IORQ}}$ y $\overline{\text{M1}}$ se ponen bajos simultáneamente. También está sincronizado por la dirección A7.

La puerta para ampliación del Amstrad no proporciona una línea de decodificación idónea para la conexión directa a dispositivos E/S externos y, por tanto, la decodificación se ha de hacer externamente. El direccionamiento E/S en la gama CPC utiliza la totalidad de los 16 bits de direcciones (durante la mayoría de las instrucciones E/S del Z80, los contenidos del registro B salen en los ocho bits de direcciones superiores). El bus de direcciones dispone para su uso las direcciones desde &F800 a &FBFF, mientras que los ocho bits inferiores han de estar entre &E0 y &FE para los periféricos del usuario.

Esto no es tan complicado como parece, porque todas las direcciones E/S internas tienen el bit de direcciones A10 en +5 V. Por consiguiente, para decodificar nuestra gama de direcciones necesitamos detectar un nivel bajo en A10, y niveles altos en A5-A7. Esto se efectúa conectando A10 a CS2, y A5 y A6 a CS0 y CS1 respectivamente, mientras que la dirección A7 se utiliza para sincronizar la señal *enable* (E), impidiéndole ponerse activa a menos que A7 se ponga alto.

Lectura y escritura

A diferencia del procesador 6502, el Z80 no proporciona una única señal de lectura/escritura, de modo que nuestro diseño trata a los registros de lectura y escritura del 6850 como direcciones diferentes, uniendo la línea R/W a la línea de direcciones A9. La línea *select* del registro interno se conecta a A8, lo que da por resultado que a los registros ACIA se les asignen las direcciones E/S, como refleja la tabla de ubicación de registros ACIA (ver p. 2114).

Es importante observar que, debido a que la acción de leer o escribir en el ACIA está controlada por una línea de direcciones, no es posible escribir en las direcciones de registros de lectura solamente (&FAE0, &FBE0). Esto dará por resultado que tanto el Z80 como el ACIA intenten simultáneamente colocar un byte de datos en el bus del sistema. Es todavía más peligroso intentar leer las direcciones de sólo escritura (&F8E0, &F9E0). Ello daría por resultado que se efectuara una escritura en el ACIA con el bus de datos en un estado flotante impredecible.

La placa se debe comprobar del mismo modo que se comprobó la versión anterior. Aquí presentamos nuevamente el procedimiento, con las regulaciones adecuadas para el Amstrad.

1. Conectar un cable DIN de 5 patillas estándar entre los conectores IN y OUT de la placa.
2. Inicializar el ACIA con la siguiente instrucción:

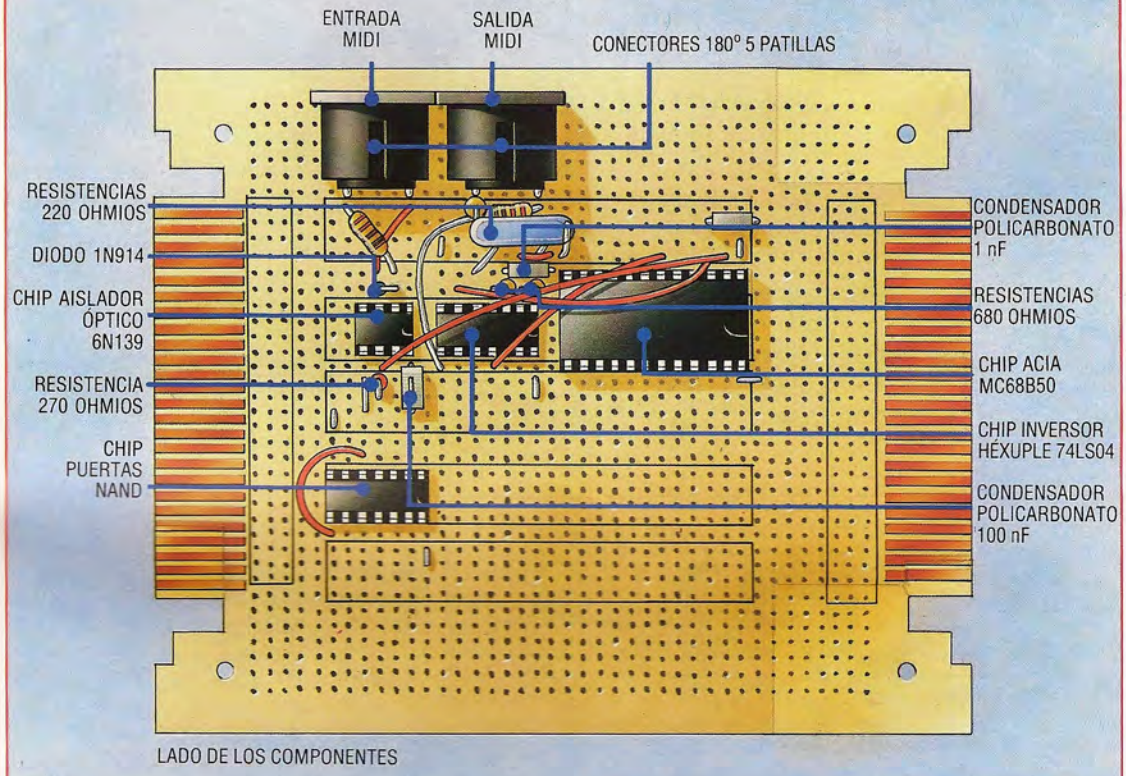
```
OUT &F8E0,3
```

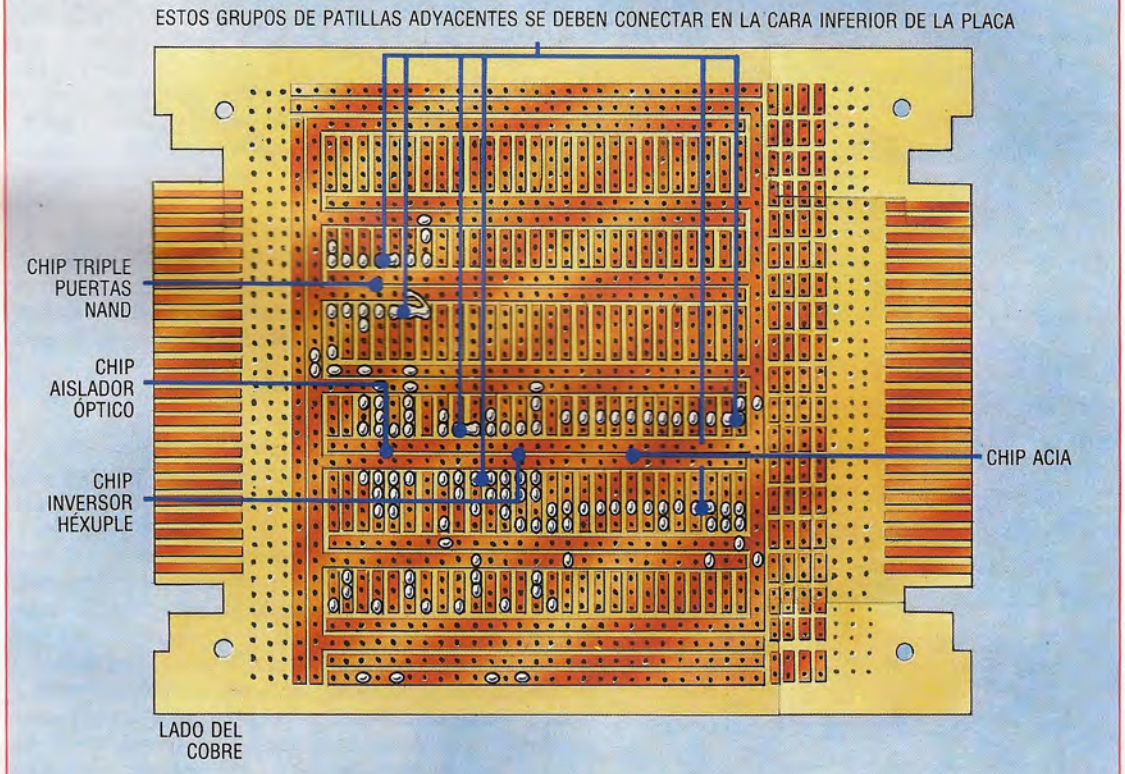
Lista de componentes

N.º	Artículo
1	condensador policarbonato 1 nF
2	condensador policarbonato 100 nF
1	resistencia 270 ohmios
3	resistencia 220 ohmios
2	resistencia 680 ohmios
2	conectores para montar en la placa, DIN de 5 patillas y 180°
1	chip ACIA MC68B50
1	chip aislador óptico 6N139
1	inversor héxuple TTL 74LS04
1	puerta NAND triple de tres entradas 74LS10N
1	zócalo DIL 24 patillas
2	zócalo DIL 14 patillas
1	zócalo DIL 8 patillas
1	cristal 2.0 MHz
1	diodo 1N914
2	conectores de borde de placa de 50 vías
30 cm	cable plano de 50 vías

La placa DIP que se utiliza para montar los componentes se puede conseguir en casi todas las sucursales Tandy, bajo el número de componente 276-164.

Disposición de los componentes



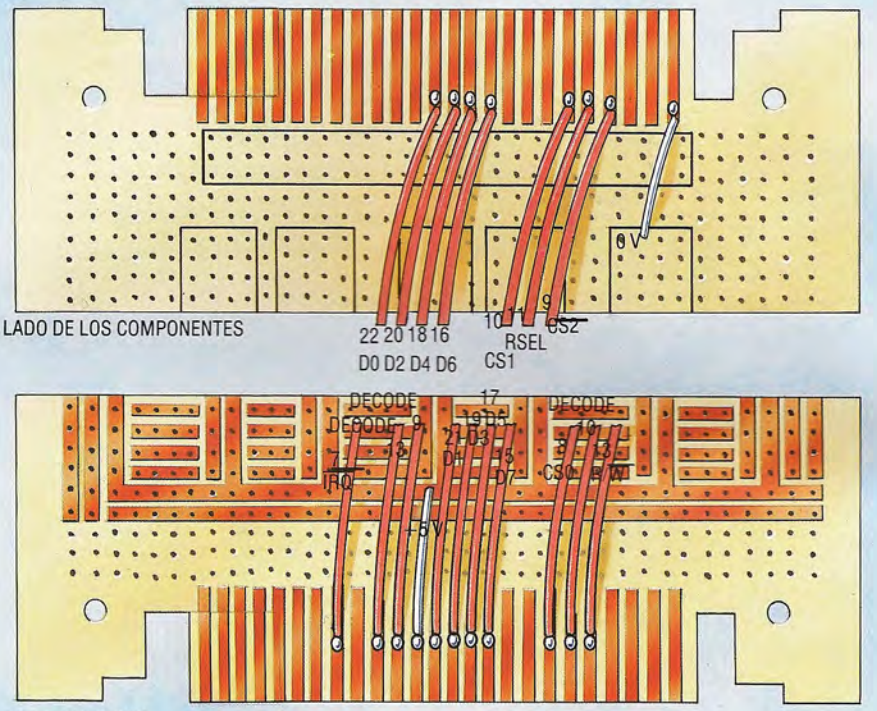


Kevin Jones

Detalles de montaje
Los componentes principales de la interface MIDI deben montarse en la placa especial. Comience por montar los componentes pasivos: las resistencias, los condensadores, los conectores DIL y DIN. Realice las conexiones necesarias en la placa utilizando cable de arrollar aislado y monte el cristal. Asegúrese de que el diodo quede montado de modo que el extremo marcado con una franja de color quede arriba, y no olvide la pequeña unión debajo del condensador situado más cerca del chip inversor héxuple

Conexiones del borde
El extremo izquierdo de la placa se utiliza para aceptar señales desde el bus de ampliación Amstrad. Estos diagramas muestran las conexiones de bus adecuadas con el chip ACIA, el chip decodificador 74LS10 y las alimentaciones. Cuando se hayan realizado todas estas conexiones con cable de arrollar aislado se pueden empujar suavemente los cuatro ICs en sus zócalos, teniendo cuidado de respetar la orientación correcta de las muescas. Por último, construya un cable de 50 vías utilizando dos conectores de borde de 50 vías y un trozo de 30 cm de cable plano. Presione un conector en la placa y el otro en la puerta para ampliación de su Amstrad, asegurándose de que el ordenador esté apagado. Ahora la placa está terminada y lista para probarla

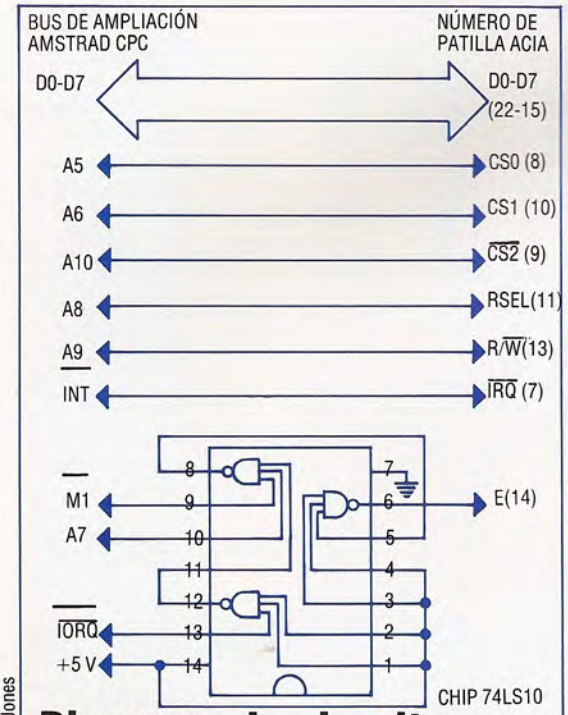
Conexiones del borde



Ubicación de registros ACIA

Dirección	Registro	
&F8E0	Control	(SÓLO ESCRITURA)
&F9E0	Transmisión datos	(SÓLO ESCRITURA)
&FAE0	Estado	(SÓLO LECTURA)

Decodificación Amstrad
Nuestro diseño de circuito original de la interface MIDI estaba pensado para máquinas basadas en el 65XX. Para usar la interface con una máquina basada en el Z80, como cualquiera de la gama Amstrad CPC, se han de decodificar las señales del bus del ordenador antes de la conexión con el chip ACIA. Este diagrama de circuitos muestra la lógica de decodificación necesaria que se obtiene en nuestro diseño corregido utilizando un solo chip triple de puertas NAND de tres entradas



Kevin Jones

Diagrama de circuitos

3. Establecer la velocidad del reloj, formato de palabras en serie e inhabilitar interrupciones recibir-/transmitir digitando:

```
OUT &F8E0,&16
```

4. Leer el registro de estado del ACIA mediante:

```
PRINT INP(&FAE0)
```

El valor correcto debe ser dos, que indica que la conexión del bus está funcionando correctamente.
5. Enviar un byte en serie desde la salida a la entrada digitando:

```
OUT &F9E0,x
```

donde x puede ser cualquier entero entre cero y 255.
6. Verificar la recepción correcta del byte volviendo a leer el registro de estado:

```
PRINT INP(&FAE0)
```

Esto habrá de devolver el valor tres.
7. Verificar, asimismo, que el byte sea el valor esperado, leyendo el registro de datos de recepción:

```
PRINT INP(&FBE0)
```

Esto devolverá el mismo valor x que se utilizó en el paso 5. Los pasos del 4 al 7 se pueden repetir tantas veces como sea necesario utilizando distintos valores de x.

Si alguna de estas pruebas falla, o si el ordenador se queda colgado con la placa conectada, compruebe el cableado con un tester, en caso de que disponga de uno. Si aun así los problemas persisten, remítase a la p. 1848, donde analizamos algunos de los problemas más probables.

En el próximo y último capítulo de este proyecto crearemos software para nuestra interface.

La dimensión FORTRAN

Veamos de qué manera el FORTRAN emplea las estructuras de matriz

La única estructura de datos de que dispone el programador de FORTRAN es la matriz y, al igual que en el BASIC, debe declararse antes de utilizarla; dado que el FORTRAN es un lenguaje compilado, esto ha de hacerse justo al comienzo del programa. Esto se realiza por medio de una sentencia DIMENSION, que funciona de la misma manera que una sentencia DIM del BASIC.

Utilizando los tipos de datos por defecto, en los cuales toda variable cuyo nombre empiece con I, J, K, L, M o N es de enteros y todas las demás son reales, una sentencia como:

```
DIMENSION A(20),I(50)
```

reservará espacio de memoria para una matriz de hasta 20 números reales y otra matriz de hasta 50 números enteros. Si se quiere pasar por alto las tipologías por defecto con sentencias INTEGER, REAL, DOUBLE PRECISION, COMPLEX o LOGICAL, entonces se debe dar uno de éstos además de la sentencia DIMENSION. Sin embargo, se puede hacer la declaración de la matriz mientras se efectúa la declaración del tipo:

```
INTEGER ARR1(20)
REAL NUMS(30)
```

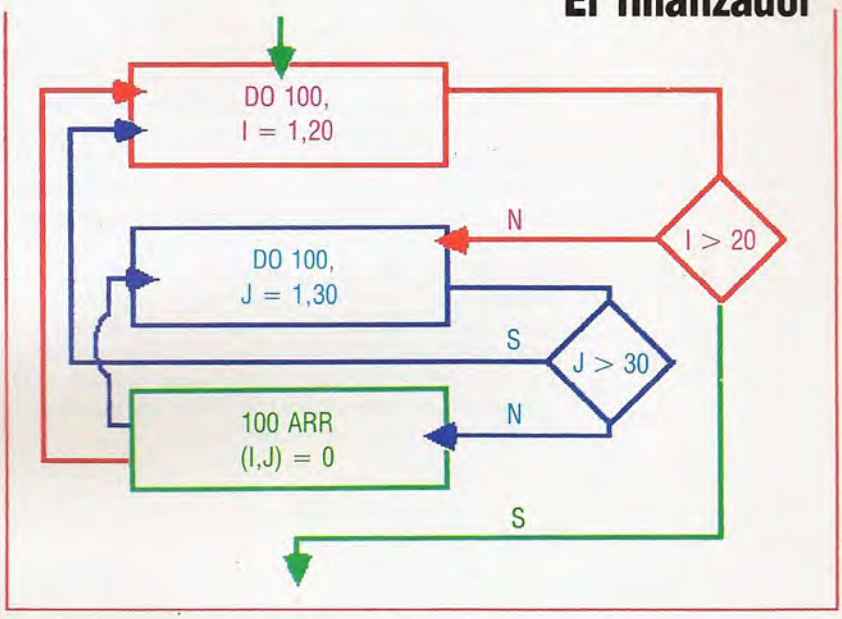
Esto reserva espacio para una matriz de 20 enteros y una para 30 números reales.

Son factibles matrices bidimensionales y algunas versiones permiten tres o más dimensiones. Las matrices bidimensionales se declaran de la forma normal. He aquí dos ejemplos:

```
DIMENSION ARR(20,30)
INTEGER ARR(20,30)
```

Para indexar la matriz se puede utilizar cualquier constante o variable de enteros, como ARR(3,4) o INTARR(I). Aunque en FORTRAN IV no existe nada que se parezca a una serie de caracteres, esta deficiencia, junto con otras cosas, se palió en el FORTRAN 77, que veremos con mayor profundidad en el próximo capítulo. Los caracteres se almacenan como enteros en variables de enteros comunes; en el último capítulo vimos cómo se podían almacenar uno o más caracteres en cualquier variable de enteros. En FORTRAN, lo más cercano a una serie de caracteres es una matriz de enteros, pero esto no es realmente satisfactorio para quien esté acostumbrado a las amplias facilidades de manipulación de series que ofrecen casi todos los BASIC.

El finalizador



Caroline Clayton

En FORTRAN, la principal facilidad de bucle y la única estructura de control (aparte de las sentencias IF y GOTO) es el bucle DO, que efectivamente opera en el mismo estilo que un bucle FOR...NEXT de BASIC. El bucle DO asume la forma:

```
DO  Sn(var. entera)= val.inicial,
    val.final, val.paso
.....
Sn  (última sentencia del bloque a repetir)
```

Sn es el número de la última sentencia del bloque que se ha de repetir. Como contador del bucle (var. entera), se puede utilizar cualquier variable de enteros, y se especifican los valores de comienzo, final y paso. En BASIC, el bucle DO sería así:

```
FOR  var. entera=val.inicial TO val.final STEP
     val.paso
.....
NEXT var. entera
```

Para hacer que el valor de una matriz sea cero, por ejemplo, entraríamos:

```
      DO 100 I = 1,20,1
100 ARR(I)=0
```

cuando el valor del paso de una matriz es 0, entonces se puede omitir, como en:

```
      DO 100 I = 1,20
100   ARR(I)=0
```

La sentencia que se numera para marcar el final del bucle puede ser cualquier sentencia ejecutable, con la excepción de una IF, una GOTO u otra DO.

Código compacto
El FORTRAN carece de alguna sentencia clara de fin del bucle, y más de un bucle puede compartir la misma sentencia de terminación. Esto puede dar origen a una codificación muy compacta, como vemos en este ejemplo de flujo de programa en el que dos bucles comparten el mismo finalizador. El equivalente de BASIC, aunque menos compacto, es mucho más fácil de leer y, por tanto, de depurar:

```
100 FOR I=1 TO 20
110 FOR J=1 TO 30
120 LET A (I,J)=0
130 NEXT J
140 NEXT I
```

El hecho de que no haya ninguna sentencia clara de final del bucle a veces resulta útil si se quiere producir un código más compacto, pero también puede conducir a algunos errores garrafales, así como a un código que resulte absolutamente incomprensible incluso al propio autor, en especial cuando hay dos o más bucles anidados uno dentro de otro. Por ejemplo, esta inicialización de una matriz bidimensional es perfectamente legal:

```
      DO 100 I=1,20
      DO 100 J=1,30
100   ARR (I,J)=0
```

(Observe que en ambos bucles se ha utilizado como finalizador la misma sentencia.)

Para superar las restricciones sobre la sentencia final de un bucle DO y obtener un código más claro, con frecuencia se emplea la sentencia "ficticia" CONTINUE a modo de finalizador. Se trata de una sentencia de FORTRAN ejecutable y legal que no hace absolutamente nada y se puede utilizar en cualquier lugar de un programa donde se necesite. La convención normal exige terminar los bucles DO con su propio y exclusivo CONTINUE, de modo que el ejemplo anterior se convertiría en:

```
      DO101 I=1,20
        DO100 J=1,30
          ARR(I,J)=0
100     CONTINUE
101     CONTINUE
```

Algunos programadores llegan aún más lejos e insisten en que todas las transferencias de control, como los DO, IF y GOTO, han de terminar con un CONTINUE, lo que ciertamente ayuda a hacer que los programas en FORTRAN resulten más comprensibles.

Como lenguaje matemático, cabe esperar que el FORTRAN proporcione algunas facilidades adicionales para la manipulación de matrices de números, particularmente en entrada/salida. Toda matriz uni o bidimensional se puede leer o escribir en una única sentencia READ o WRITE, donde el FORMAT correspondiente da la posición de cada línea o registro. Por ejemplo:

```
      DIMENSION IARR1(20)
      .....
      WRITE(1,10)IARR1
```

produciría la salida de todo el contenido de IARR1 (20 valores de entero). Además:

```
10  FORMAT (20I4)
```

produciría 20 enteros de cuatro dígitos a lo largo de una línea;

```
10 FORMAT (I4)
```

daría uno por línea;

```
10 FORMAT (5I4)
```

daría cuatro líneas de cinco números, y:

```
10 FORMAT (20A1)
```

daría una serie de 20 caracteres en una línea.

Existe un problema cuando se entran o se sacan matrices bidimensionales de este modo: el FORTRAN, a diferencia de casi todos los otros lenguajes, almacena las matrices bidimensionales por orden de columna en lugar de por orden de fila. Si usted no tiene cuidado, se encontrará trabajando con una versión transpuesta de lo que realmente quería. Para sortear este problema, y para situaciones en las que no está manejando la totalidad de una matriz, se proporciona una forma especial de DO denominada *bucle* DO *implícito*, que se escribe con el nombre de la variable en la sentencia READ o WRITE. Por ejemplo, para rellenar IARR2(10,20) con 10 series de 20 caracteres, se podría usar:

```
      DO 100 I=1,10
      READ (1,10)(IARR2 (I,J),J=1,20)
10    FORMAT (20A1)
100   CONTINUE
```

El FORTRAN posee una auténtica facilidad, aunque algo limitada, de subrutinas. Éstas son completamente independientes y suelen emplear sólo variables locales, y por lo general pueden añadirse al final de un programa principal o escribirlas y compilarlas por separado. Es corriente utilizar los mismos números de línea nuevamente en una subrutina, y en la mayoría de los casos los mismos nombres de variables, sin riesgo de confusión. El paso de parámetros es sólo por referencia; en otras palabras, la dirección del parámetro se pasa a la subrutina cuando se llama, de modo que cualquier cambio de valor provocado por la subrutina afectará a la variable correspondiente en el programa de llamada. Las variables globales, a las que se puede hacer referencia en la subrutina y en el programa principal, se pueden declarar, si así se requiriera, mediante una sentencia COMMON en ambos módulos del programa. Se pueden establecer varios bloques COMMON y asignarles un nombre, pero por lo general basta un solo bloque sin nombre.

Las variables designadas en las sentencias COMMON de dos módulos deben coincidir en el uso total de memoria, pero no tienen que coincidir en tipo, si bien no se recomienda esto como mecanismo para cambiar el tipo. Por ejemplo, un programa principal podría contener:

```
      COMMON I1,I2,I3
```

donde la subrutina podría contener:

```
      COMMON I(3)
```

El uso de memoria es el mismo en ambos casos; pero en el primero, las tres variables de enteros se mantienen distintas, mientras que en el segundo se alude a ellas como elementos de una matriz.

Un subrutina típica podría tener la forma:

```
      CALL MYSUB (X,Y,I,J)
```

donde la subrutina comenzaría con una sentencia:

```
      SUBROUTINE MYSUB (A,B,L,M)
```

(Observe que los parámetros dentro de los paréntesis deben coincidir en número y tipo.)

El control retorna desde la subrutina al programa de llamada mediante una sentencia RETURN (de hecho, en una subrutina puede haber más de un RETURN). Uno de éstos se debe ejecutar independientemente del camino de control que se siga a través de la subrutina. La sentencia final de un subprograma de subrutinas debe ser un END, aunque ésta es más una directriz para el compilador que una sentencia ejecutable.

Una facilidad interesante del paso de parámetros es que, en cierto sentido, es posible cambiar el tamaño de una matriz dinámicamente en tiempo de ejecución. En un lenguaje interpretado como el BASIC esto se hace fácilmente, pero es muy raro en un lenguaje compilado, en el cual el espacio de almacenamiento se debe asignar en tiempo de compilación en vez de en tiempo de ejecución.

Además de las subrutinas, el FORTRAN proporciona una facilidad FUNCTION para los subprogramas, que devuelve un único valor y que en muchos sentidos es similar a la del PASCAL. Un subprograma de función se escribe por separado, de modo similar a una subrutina, pero empezando con una sentencia FUNCTION. Antes de que se ejecute un RETURN se debe efectuar una asignación al nombre de función. El tipo de valor por defecto que devuelve la función está determinado por el nombre de función utilizando la convención habitual. De ser necesario se podría obviar este requisito, aunque ello no es recomendable a menos que lo requiera una función empleada por varios programas diferentes. Las funciones se pueden utilizar en el programa principal del mismo modo que las funciones de biblioteca (simplemente usando su nombre, con una lista de parámetros) en cualquier expresión en la que sea válido usar una variable de ese tipo.

Hallando las raíces

```
C     UN PROGRAMA EN FORTRAN PARA LEER
C     CONJUNTOS DE 3 VALORES A,B,C QUE REPR.
C     LOS COEFICIENTES DE UNA ECUACION DE 2°
C     GRADO A*X**2+B*X+ C=0 USANDO LA
C     FORMULA USUAL
      FUNCTION DISCR(A,B,C)
      DISC=B*B-4.0*A*C
      RETURN
      END
      SUBRUT.RESOLVER(A,B,C,X1,X2,RAIZN)
      D=DISCR(A,B,C)
C     OBSERVE QUE AQUI SE PODRIA UTILIZAR
C     UN IF ARITMETICO PERO NO ES ACONSEJABLE
      IF(D.GE.0.0)GOTO 100
C     D ES MENOR QUE 0 POR TANTO R. COMPL.
      RAIZN=0
      X1=0.0-B/(2.0*A)
      X2=SQR(ABS(D))
      GOTO 300
100   IF(D.GT.0.0)GOTO 200
C     D ES CERO DE MODO QUE RAICES IGUALES
      RAIZN=1
      X1=0.0-B/(2.0*A)
      X2=X1
      GOTO 300
C     D ES POSITIVO POR TANTO DOS RAICES
      RAIZN=2
200   X1=(0.0-B+ SQRT(D))/2.0*A
      X2=(0.0-B-SQURT(D))/2.0*A
300   RETURN
      END
C     PRIMERO LEER NUM. CONJUNTOS VALORES
      READ(2,10)VALN
C     IMPRIMIR TITULOS
      WRITE(3,11)
      DO 500 I=1, VALN
C     LEER TRES VALORES Y COMPROBAR
C     SI ACEPTABLES
100     READ(1,12)A,B,C
        IF (A.EQ.0.0)GOTO 500
C     LOS VALORES SON ADECUADOS DE MODO QUE
      LLAMAR SUBRUT.
        CALL SOLVE(A,B,C,X1,X2,RAIZN) IF(RAIZN.GT.0)GOTO 200
C     RAICES COMPLEJAS
        WRITE(1,13)A,B,C,X1,X2
        GOTO 500
200   IF(RAIZN.GT.1)GOTO 300
C     RAICES IGUALES
        WRITE(1,14)A,B,C,X1,X2
        GOTO 500
C     DOS RAICES
300   WRITE(1,15)A,B,C,X1,X2
        GOTO 500
C     VALORES NO VALIDOS PARA A,B Y C
400     WRITE (1,16)A,B,C
500   CONTINUE
      STOP
C     SENTENCIAS FORMAT
10    FORMAT(I4)
11    FORMAT(1H,8X,1HA,8X,1HB, 8X,
      1HC,8X,4NTIPO,8X,2HX1,8X,2HX2)
12    FORMAT(1H,3F7.2)
13    FORMAT(1H,4X,3(F7.2,2X),4X,7H
      COMPLEJA,X,F7.2,3X,F7.2)
14    FORMAT(1H,4X,3,(F7.2,2X),4X,5
      HIGUAL3X,F7.2,3X,F7.2)
15    FORMAT(1H,4X,3,(F7.2,2X),4X,7H
      DESIGUAL,X,F7.2,3X,F7.2)
16    FORMAT(1H,4X,3,(F7.2,2X),4X,7HNO
      VALIDA)
      END
```

Cálculo de media

```
C UNA FUNCION EN FORTRAN PARA
C CALCULAR LA MEDIA (REAL) DE UN
C CONJUNTO DE N NUMEROS ENTEROS
  EN UNA MATRIZ DE ENTEROS IARR
  FUNCION PROM(IARR,N)
  DIMENSION IARR(N)
  SUM=0
  DO 100 I=1,N
    SUM=SUM+FLOAT(IARR(I))
100 CONTINUE
  PROM=SUM/FLOAT(N)
  RETURN
  END
```

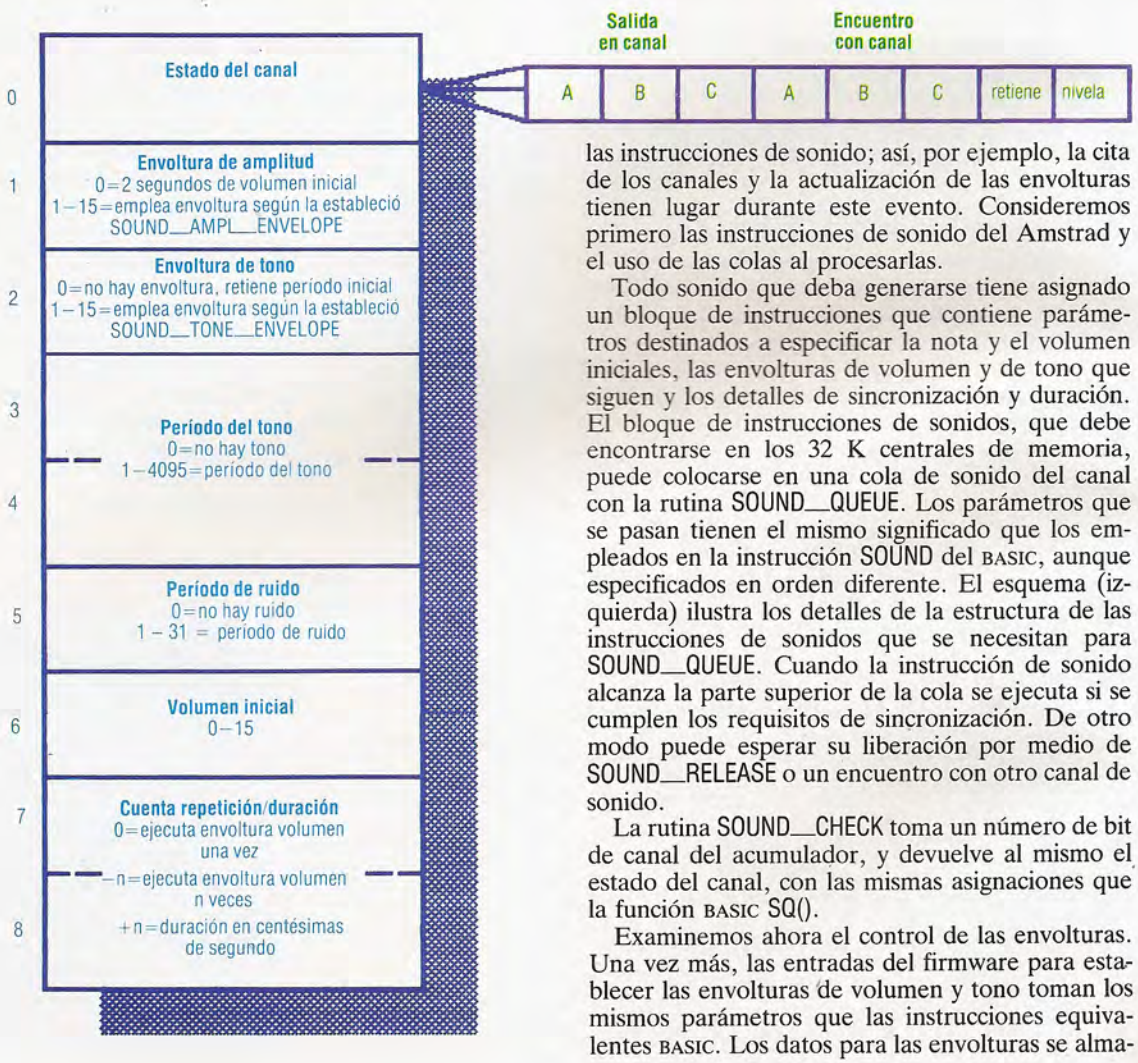
Perfecto pinchadiscos

El OS de los ordenadores Amstrad presenta facilidades sonoras que equivalen a las instrucciones en BASIC

El diseño hardware de los Amstrad emplea el conocido generador de sonidos programable AY-3-8912 de General Instruments (llamado brevemente *8912*), que no sólo sirve para generar sonidos sino que también puede rastrear el teclado mediante la puerta de siete bits del chip.

El 8912 soporta tres canales de sonido independientes (A,B,C) y un generador seudoaleatorio de sonido, que puede programarse para que aparezca en cualquier canal. El chip ofrece las facilidades básicas para producir un sonido y para cambiar el volumen del sonido, mediante envolturas de sonido predefinidas. Las salidas del generador de sonidos se mezclan para producir una salida stereo: el canal A es el de la izquierda, el canal B el de la derecha, y el canal C es una mezcla equitativa de los dos. Estas facilidades han sido ampliadas por el sistema operativo, que proporciona el control software del sonido y del volumen a un tiempo con métodos de sincronización de canales sonoros.

Las instrucciones BASIC tales como ENV, ENT, RELEASE, SOUND y SQ se sirven directamente del sistema operativo para obtener sonidos. De hecho los parámetros que se pasan y los conceptos correspondientes son casi idénticos. El sistema operativo controla el sonido por medio de un evento de interrupción especial que se produce un centenar de veces por segundo. Este evento está destinado a procesar las instrucciones de sonido; así, por ejemplo, la cita de los canales y la actualización de las envolturas tienen lugar durante este evento. Consideremos primero las instrucciones de sonido del Amstrad y el uso de las colas al procesarlas.



Bloque de instrucciones sonoras
Una instrucción de sonido consiste en un bloque de datos que contiene varios parámetros, semejantes a los que se usan en la instrucción SOUND del BASIC. La dirección de este bloque se pasa en HL hasta SOUND_QUEUE, que después realiza la oportuna operación. Si la cola está llena, la instrucción se ignora y el flag de arrastre se hace falso

Todo sonido que deba generarse tiene asignado un bloque de instrucciones que contiene parámetros destinados a especificar la nota y el volumen iniciales, las envolturas de volumen y de tono que siguen y los detalles de sincronización y duración. El bloque de instrucciones de sonidos, que debe encontrarse en los 32 K centrales de memoria, puede colocarse en una cola de sonido del canal con la rutina SOUND_QUEUE. Los parámetros que se pasan tienen el mismo significado que los empleados en la instrucción SOUND del BASIC, aunque especificados en orden diferente. El esquema (izquierda) ilustra los detalles de la estructura de las instrucciones de sonidos que se necesitan para SOUND_QUEUE. Cuando la instrucción de sonido alcanza la parte superior de la cola se ejecuta si se cumplen los requisitos de sincronización. De otro modo puede esperar su liberación por medio de SOUND_RELEASE o un encuentro con otro canal de sonido.

La rutina SOUND_CHECK toma un número de bit de canal del acumulador, y devuelve al mismo el estado del canal, con las mismas asignaciones que la función BASIC SQ().

Examinemos ahora el control de las envolturas. Una vez más, las entradas del firmware para establecer las envolturas de volumen y tono toman los mismos parámetros que las instrucciones equivalentes BASIC. Los datos para las envolturas se alma-

cenan como una tabla en la forma que se pasan a las rutinas, y posteriormente se procesan cada vez que es necesaria la envoltura. La dirección del bloque de datos asociado con una envoltura puede determinarse en cualquier momento por medio de SOUND_T_ADDRESS, o con SOUND_A_ADDRESS, por lo que es posible alterar una envoltura sin llamar al firmware parcheando directamente el bloque de datos.

El formato del bloque de datos es el que mostramos en el diagrama de formato de datos de envoltura (derecha); se puede parchear cualquier sección con sólo calcular el desplazamiento necesario en el bloque de datos. Si se adopta este método, hay que tener cuidado, ya que el firmware puede acceder simultáneamente al área de datos mientras es parcheada. Aunque no se van a producir grandes desastres, es muy posible que el sonido producido no sea como se ha previsto.

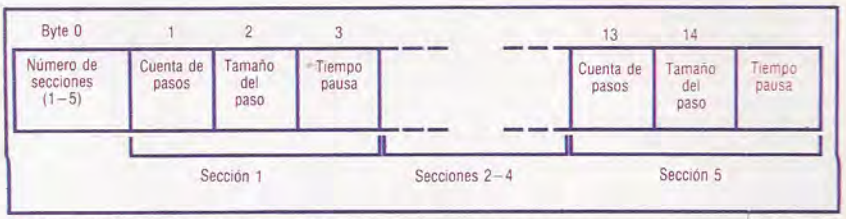
El firmware proporciona una rutina especialmente útil, que es la SOUND_ARM_EVENT. Ésta establece un evento que ha de lanzarse cuando la cola de los sonidos para un siguiente canal especificado se encuentra vacía. De hecho, la rutina lleva un evento especial a la interrupción que procesa el sonido, que sucede cada centésima de segundo. Este evento comprueba la cola de sonidos del canal en cuestión y, si hay espacio, entonces es lanzado el bloque de eventos pasado a la rutina SOUND_ARM_EVENT. Este evento lanzado se procesa después en la manera especificada en su clase de evento.

El bloque de eventos que se pasa debe ser inicializado primero empleando KL_INIT_EVENT, por lo que puede tener cualquier clase o prioridad. Sin embargo, dado que la interrupción de sonido sucede cada centésima de segundo, es aconsejable establecer el evento según el tipo asíncrono normal.

Este evento puede ser usado para generar música continua de fondo a un programa sin que el programador tenga que preocuparse de proporcionar continuamente datos al gestor del sonido. En su lugar, la rutina de eventos debe leer la siguiente dirección del programa del sonido desde el área predefinida de datos y después llamar la entrada SOUND_QUEUE. Se puede disponer la rutina de tal modo que vuelva circularmente al inicio de los datos cuando llegue al extremo final con el fin de generar un efecto musical continuo.

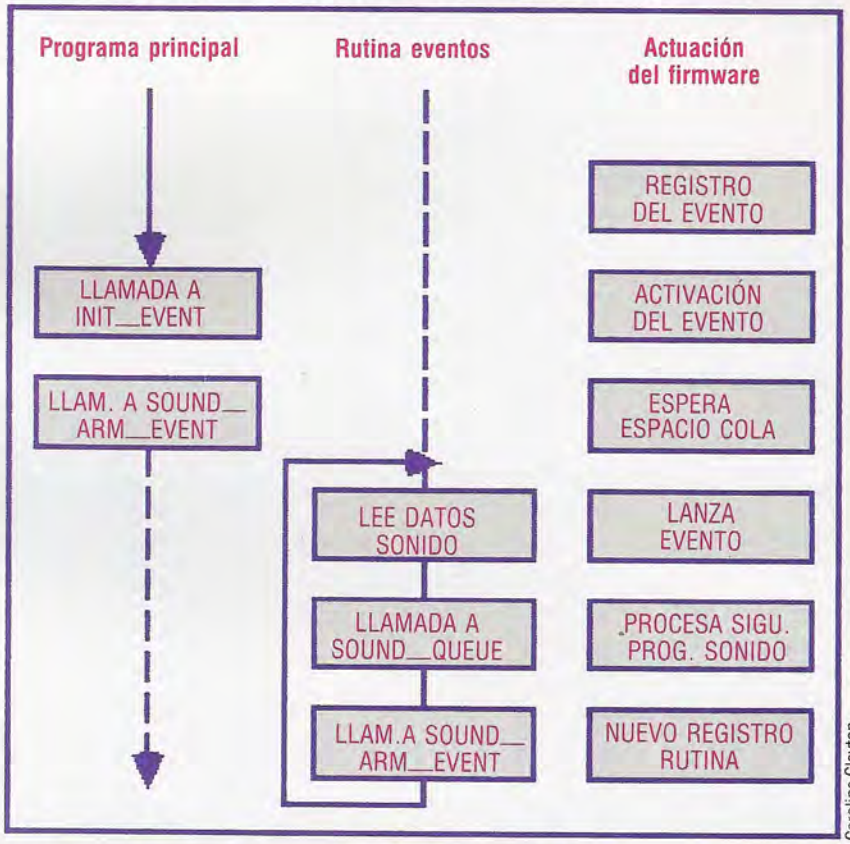
La entrada SOUND_QUEUE desactiva el evento de sonido, de modo que antes que concluya la rutina debe registrarse llamando a SOUND_ARM_EVENT, que pasa su propia dirección del bloque de eventos. El diagrama de flujo muestra cómo interaccionan la rutina de eventos y la rutina de cola de sonidos para generar el efecto musical continuo.

La generación de sonidos puede detenerse en cualquier momento mediante la entrada SOUND_HOLD; ésta suspende toda envoltura actualmente en uso y desactiva el evento de sonido (si está activado). El sonido puede volverse a iniciar llamando SOUND_CONTINUE, SOUND_QUEUE o bien SOUND_RELEASE. De estas rutinas la última se da para iniciar cualquier programa de sonido que esté señalizado (*flag*) para ser retenido individualmente. Se llama SOUND_HOLD siempre que se detecta una parada (*break*) desde el teclado, para asegurarse de que no se permite continuar ninguna generación de sonidos espuria desde dentro de un programa.



▲ **Bloque de datos de envoltura** Toda envoltura puede tener hasta cinco secciones, cada una de las cuales se especifica mediante tres bytes que determinan la cuenta de los pasos, el tamaño de los pasos y el tiempo de pausa. Si se emplean envolturas de volumen del hardware, el primer byte del bloque retiene el número de la envoltura (entre el 128 y el 143) y los bytes dos y tres retienen el tiempo de pausa

▼ **Evento notable** El empleo de SOUND_ARM_EVENT permite al programador generar música controlada por eventos en los Amstrad CPC que funcionará como fondo mientras continúa ejecutándose el programa principal en primer plano. El esquema muestra las operaciones efectuadas por el programa principal, la rutina de eventos y el firmware



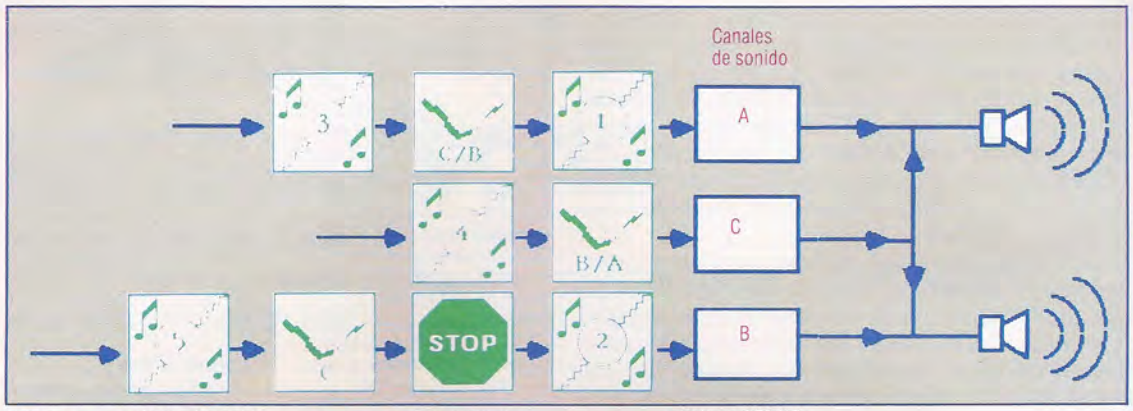
Caroline Clayton

Entradas del gestor de sonidos

OBCA7H	SOUND_RESET	Restablece gestor sonidos
OBCAAH	SOUND_QUEUE	Añade un sonido a una de las colas
OBCADH	SOUND_CHECK	Da el estado de una cola de sonido
OBCB0H	SOUND_ARM_EVENT	Permite el lanzamiento de un evento cuando la cola de sonido está vacía
OBCB3H	SOUND_RELEASE	Libera los sonidos contenidos en una cola
OBCB6H	SOUND_HOLD	Detiene todo sonido en todas las colas
OBCB9H	SOUND_CONTINUE	Reanuda los sonidos detenidos
OBCBCH	SOUND_AMPL_ENVELOPE	Inicializa una de las envolturas de volumen del software
OBCBFH	SOUND_TONE_ENVELOPE	Inicializa una de las envolturas de tono del software
OBCC2H	SOUND_A_ADDRESS	Da la dir. de una envolt. de volumen
OBCC5H	SOUND_T_ADDRESS	Da la dirección de una envolt. de tono

Los detalles completos de las direcciones de entrada y salida de estas rutinas se encuentran en la publicación de Amsoft *SOFT 158*, el manual de especificaciones del firmware del Amstrad

Encuentro con el canal
En la situación que aquí se representa se están ejecutando los sonidos 1 y 2. Cuando 2 concluye, la instrucción siguiente en la cola para el canal B se retiene a la espera de SOUND_RELEASE. Puesto que la instrucción en la cola para C requiere un encuentro con los canales A y B, es retenida a la espera de las últimas instrucciones sobre estas colas. Mientras tanto, 1 termina de sonar y el canal A se encuentra con C, pero todavía espera su encuentro con B. Tan pronto como la cola para B es liberada se hace efectivo el encuentro, y se oirán simultáneamente los sonidos 3, 4 y 5



Sonido y visión
La edición de envolturas de sonido puede ser un asunto complejo, sobre todo si usted no está seguro de cómo se produce exactamente el efecto que desea. Nuestro *Editor de envolturas* le permite especificar parámetros a su elección, y visualiza después la forma de la envoltura, mostrándole la "figura" del sonido que ha creado

Una llamada BASIC

La instrucción CALL del BASIC no está del todo documentada en los manuales del Amstrad. Permite pasar parámetros a rutinas en código máquina en un "bloque de parámetros". Los parámetros que da el usuario se almacenan por orden; el primer parámetro se retiene en la dirección del bloque más baja (la primera). Esta dirección viene apuntada por IX a la entrada a la rutina llamada (CALL). Los valores numéricos pueden ser pasados directamente, mientras no pueden serlo las variables. En contrapartida, la dirección de la variable (o, en el caso de una variable de cadena, la dirección del bloque descriptor de la cadena) es pasada empleando el símbolo @. Así, p.ej., si deseamos llamar (CALL) una rutina en &A000, pasándole el valor 85 y volviendo al BASIC con un resultado almacenado en las variables result% y answer$, escribiríamos:

```
CALL &A000,85,@ result%,@ answer$
```

Al entrar en la rutina, IX apuntará a un bloque en la RAM que contiene primero el valor 85 almacenado en forma de entero sin signo de 16 bits, después dos bytes que contienen la dirección de result% y finalmente la dirección del bloque descriptor de la variable en cadena para answer$. El bloque descriptor de la cadena comprende tres bytes que contienen el valor de la longitud de la cadena, y su dirección. Las rutinas del usuario deberán evitar la alteración del contenido de este bloque y el procesamiento de cadenas habrá de efectuarse con cuidado para no tener problemas con el firmware.

Editor de envolturas

```
50 MEMORY &6FFF: DIM envelope(15)
60 done=0: routine=&7000: buffer=routine+&100: nk=1
70 WHILE -1
80 MODE 2: LOCATE 1,20: INPUT "Inicio volumen (0-
   15)";sv%:I F sv%>15 OR sv%<0 THEN 80
90 LOCATE 1,21:INPUT "Inicio tono";tt:,IF tt<0 OR tt>4095
   THEN 90
100 LOCATE 1,22:INPUT "Escala horizontal (1-10)";sc%: IF
   sc%>10 OR sc%<1 THEN 100
110 LOCATE 1,23:INPUT "Envoltura volumen o tono
   (V/T)";senv$:IF (UPPER$(senv$)<>"V")AND (UPPER$)
   (senv$)<> "T") THEN GOTO 110
120 W=0: IF UPPER$(senv$)="V" THEN pk=&BC: W=1:
   ELSE pk=& BF
130 length%=0: soundx=0: IF W=1 THEN
   soundy=INT(sv%*9.7) ELSE soundy=68
140 FOR x=1 TO 15
150  envelope(x)=o
160 NEXT
170 GOSUB 270
180 LOCATE 1,20: PRINT CHR$(18)
190 FOR count=1 TO 5
200 GOSUB 360: nk=ABS(nk-1): GOSUB 540: IF W=1 THEN
   GOSUB 880 ELSE GOSUB 890
210 NEXT
220 WHILE NOT done
220 LOCATE 5,20: INPUT "Edit (y/n)";ed$
240 IF UPPER$(ed$)="N" THEN done=-1 ELSE GOSUB 790
250 WEND
260 WEND
270 REM inicializa pantalla
280 MODE 1:WINDOW # 1,1,39,1,2: GOSUB 1000
290 MOVE 20,380: DRAW 20,210,3: DRAW 600,210,3
300 MOVE 21,soundy+211
310 POKE routine, &3E: POKE routine+1,1
320 POKE routine+2,&21: POKE routine+3, buffer-
   (INT(buffer/256))*256: POKE routine+4, INT(buffer/256)
330 POKE routine+5,&C3: POKE routine+6,pk: POKE
   routine+7,&BC
340 POKE routine+8,&C9
350 RETURN
360 REM toma datos sección
370 LOCATE 1,20: PRINT CHR$(18);: LOCATE 5,20: PRINT
   "Sección";count
380 LOCATE 1,22: PRINT CHR$(18);: LOCATE 5,22: INPUT
   "Contador pasos"; skip%: IF W=0 THEN GOTO 400 ELSE IF
   skip%<0 OR skip%>127 THEN 380
390 GOTO 410
400 IF skip%<0 OR skip%>239 THEN GOTO 380
410 P$=STR$(SKIP%): GOSUB 900
420 LOCATE 1,23: PRINT CHR$(18);: LOCATE 5,23: INPUT
   "Tamaño pasos";size%: IF W=0 THEN GOTO 440 ELSE IF
   ABS(size%)>15 THEN 420
430 GOTO 450
440 IF ABS(size%)>127 THEN 420
450 P$=STR$(size%): GOSUB 900
460 ss=size%: IF size%<0 THEN size%=256-ABS(size%)
470 LOCATE 1,24: PRINT CHR$(18);: LOCATE 5,24: INPUT
   "Tiempo pausa"; pause%: IFpause%>255 THEN 470
480 P$=STR$(pause%): GOSUB 900
490 envelope (<count-1)*3+1)=skip%: envelope
   ((count-1)*3+2)=size%: envelope
   ((count-1)*3+3=pause%
500 FOR x=20 TO 24
510  LOCATE 1,x: PRINT CHR$(18);
520 NEXT
530 RETURN
540 REM inicialización envoltura
550 POKE fuffer,count
560 CLS #1: GOSUB 1000
570 FOR x=0 TO count-1
580 POKE buffer+(x*3+1),envelope(x*3+1):
   skip%=envelope(x*3+1)
590 p$=STR$(envelope(x*3+1)): GOSUB 900
600 POKE buffer+(x*3+2),envelope(x*3+2):
   size%=envelope(x*3+2)
610 pp=envelope(x*3+2): IF pp>127 THEN pp=pp-256
620 p$=STR$ (pp): GOSUB 900
630 POKE buffer+(x*3+3),envelope(x*3+3):
   pause%=envelope(x*3+3)
640 p$=STR$(envelope(x*3+3)): GOSUB 900
650 NEXT x
660 length%=length%+skip%*pause%
670 CALL routine
680 IF W=0 THEN GOSUB 910: RETURN
690 off%=size%*10: IF size%>127 THEN
   off%=(size%-256)*10
700 FOR x=1 TO skip%
710  soundy=(soundy+off%) MOD 150: IF soundy<0 THEN
   soundy=150-soundy
720 FOR c=1 TO (10/51)*pause%+2: soundx=soundx
   + (sc%)
730  DRAW soundx+21,soundy+211,2+nk
740  NEXT
750 NEXT
760 DRAW soundx+21,soundy+211,2+nk
770 RETURN
780 REM dibuja gráfico otra vez
790 LOCATE 5,20: PRINT CHR$(18);: INPUT "Seccion";sec%:
   IF sec%<1 OR sec%>5 THEN 790
800 count=sec%: GOSUB 360
810 CLS: soundx=0: soundy=INT(sv%*9.7): length%=0:
   GOSUB 270
820 FOR section=1 TO 5
830  count=section: GOSUB 540
840 NEXT
850 GOSUB 870
860 RETURN
870 REM sound
880 SOUND 1,tt,length%,sv%,1: RETURN
890 SOUND 1,tt,length%,sv%,,1: RETURN
900 PRINT #1,MID$(P$,(2-ABS(VAL(p$)<0))),",";:RETURN
910 FOR x=1 TO skip%
920  soundy=soundy-((15/127)*ss): IF soundy<0 THEN
   soundy=0
930 IF soundy>189 THEN soundy=189
940 FOR c=1 TO (7/51)*pause%+2: soundx=soundx+sc%
950  DRAW soundx+21,soundy+211,2+nk
960  NEXT
970 NEXT
980 DRAW soundx+21,soundy+211,2+nk
990 RETURN
1000 IF W=1 THEN PRINT #1, "ENV 1, "; ELSE PRINT #1,
   "ENT 1 , ";
1010 RETURN
```

Imagen por números

El Print-Technik pone en nuestras manos la más avanzada tecnología en el campo de la digitalización de la imagen

Marcus Wilson-Smith

Un digitalizador de video toma una señal proveniente de cualquier fuente de video (por lo general, de una cámara, pero también de un disco láser o una grabadora de video) y convierte las tensiones analógicas de la señal en un gran conjunto de números que representan la imagen. Estos números se pueden almacenar en la memoria de un ordenador, visualizar en gráficos de alta resolución, tratar mediante software, volcar a una impresora, etc.

Los digitalizadores de video pueden tener múltiples usos, desde aplicaciones serias, como sistemas de alarmas, a ideas divertidas como pueden ser "imágenes por ordenador" y distintivos de personas. Print-Technik cita el ejemplo de un peluquero que digitaliza imágenes de sus clientes. Utilizando un lápiz óptico y un paquete para gráficos, puede mostrarle al cliente varios estilos de peinado para ayudarles a escoger el adecuado.

La unidad de Print-Technik, sin embargo, no parece tener capacidad para realizar todo esto. Se trata de una pequeña caja que se introduce en la puerta para el usuario del Commodore 64. Un conector de video estándar permite la conexión con su fuente de video; no hay ningún otro cable, ya que la unidad es alimentada por el propio ordenador. Los únicos otros "controles" son tres pequeños tornillos regulables que varían el brillo, el contraste y la anchura de la imagen digitalizada.

La unidad se controla mediante un paquete de software que se suministra en disco. Éste es un programa funcional y bien construido, pero hace muy poco en cuanto a explotar las auténticas posibilidades de la unidad. El programa presenta un menú amable del cual se seleccionan las opciones desplazando un puntero en forma de mano mediante las teclas del cursor y pulsando RETURN. Se puede optar por digitalizar la señal, ver la imagen que esté actualmente en la memoria, guardarla en disco o imprimirla. Se admiten distintas impresoras, incluyendo las unidades 801 y 1525 de Commodore. Se puede cargar un programa especial para impresoras llamado "16 colores" para imprimir imágenes en hasta 16 colores utilizando una de las cinco principales impresoras matriciales en color. Las imágenes en pantalla se limitan a cuatro colores, pero el digitalizador puede resolver hasta 256 tonalidades.

Como cabría esperar, el software está diseñado para ser empleado conjuntamente con otros programas. Una opción LIGHTPEN llama al propio software para gráficos con lápiz óptico de Print-Technik, si bien el mismo no se suministra como estándar. Lo que es más importante, la imagen de la memoria se puede guardar en disco en un formato apto para dos conocidos paquetes artísticos para el Commodore 64: *Koala-pad* y *Paintmagic*. Los artistas y diseñadores que utilicen estos programas encontrarán que el digitalizador de video representa una valiosa adición a su gama de herramientas. Todas las rutinas del software para guardar y cargar imágenes usan cargadores rápidos para mejorar el rendimiento de la unidad de disco 1541.

En la actualidad, la digitalización de una imagen emplea cuatro segundos, lo que limita las aplicaciones potenciales para la unidad y la hace más difícil de utilizar. El sistema de menor precio que puede formarse se basaría en una cámara de televisión de circuito cerrado similar a las que se utilizan para vigilancia. Estas cámaras no poseen pantalla/visor interno. Por consiguiente, es casi preceptivo tener disponible una pantalla de video normal de modo que el operador pueda centrar el sujeto y enfocar la cámara visualmente antes de desconectarla de la pantalla y conectarla al digitalizador, lista ya para captar la imagen. De modo que, si bien el sistema funciona sin monitor externo, se convierte en una cuestión de ensayo y error, reduciendo en gran medida la calidad de los resultados.

Los cuatro segundos necesarios para producir una imagen también hacen difícil captar una imagen en movimiento. Para obtener imágenes de per-

Técnicas visuales
El digitalizador de video Print-Technik para el Commodore 64 capta imágenes de una cámara o grabadora de video y las digitaliza, permitiendo visualizarlas en la pantalla en cuatro tonalidades, almacenarlas en disco o imprimirlas. Una facilidad particularmente interesante permite guardar las imágenes digitalizadas en disco en formatos que utilizan otros paquetes para gráficos, como el *Koala-pad*. El paquete incluye un pequeño cartucho que se enchufa en la puerta para el usuario y varios paquetes de calidad en disco para soportar el hardware

DIGITALIZADOR DE VIDEO PRINT-TECHNIK

DIMENSIONES

80 × 65 × 20 mm

CONTROLES EXTERNOS

Brillo, contraste y anchura de imagen

RESOLUCION

256 × 256 pixels en 256 tonalidades. Normalmente las imágenes se presentan en formato de 160 × 200 pixels utilizando cuatro tonalidades

VELOCIDAD

Cuatro segundos por imagen

SOFTWARE

Mediante el software estándar las imágenes se pueden digitalizar, guardar en formato *Koala-pad* o *Printmagic* de 256 × 256, imprimir y volver a colorear. Además, los programas de demostración proporcionan un sistema de alarma y una exhibición de diapositivas, y permiten utilizar imágenes digitalizadas en los propios programas en BASIC

DOCUMENTACION

Un folleto de cuatro páginas no logra cubrir los aspectos más técnicos de la unidad. No obstante, el funcionamiento básico es lo suficientemente sencillo como para que esto no constituya un problema

VENTAJAS

La unidad es compacta, su precio es asequible y trabaja muy bien. El software que se proporciona es adecuado para la mayoría de las aplicaciones simples

DESVENTAJAS

La lenta respuesta significa que la unidad se debe utilizar con una pantalla externa. Los límites de la pantalla de gráficos del Commodore 64 (cuatro colores a 160 × 200 pixels) significan que el paquete estándar no explota cabalmente las capacidades

sonas de calidad, el sujeto ha de permanecer quieto durante los cuatro segundos completos, aunque se pueden crear algunos curiosos efectos mediante el movimiento cuando el digitalizador está en funcionamiento. Hay dos soluciones para este problema. La ideal es utilizar un *captador de cuadro* de video, que congelará la acción electrónicamente. La alternativa es ¡tomar una fotografía y digitalizarla!

Con la fuente correctamente regulada, los únicos otros ajustes que se han de hacer a la imagen captada se efectúan mediante los tres pequeños tornillos de la unidad digitalizadora. Éstos se entregan regulados en niveles óptimos, pero pueden alterarse para satisfacer condiciones extrañas o inusuales. El brillo y el contraste se explican por sí mismos, pero han de regularse mediante ensayo y error, dado que no existe forma inmediata de ver los resultados. El tercer tornillo varía el ancho de la imagen, comprimiendo o ensanchando la imagen resultante. Se puede utilizar para dar las proporciones correctas a una imagen en la pantalla de gráficos del ordenador o bien para producir efectos especiales. Un rostro delgado, por ejemplo, se puede transformar mágicamente con sólo dar vuelta al tornillo.

La imagen digitalizada es de 256 × 256 pixels y, por tanto, en la pantalla de gráficos en color de 160 × 200 del Commodore 64 sólo se puede ver parcialmente. El software de Print-Technik permite explorar la imagen utilizando las teclas del cursor. La imagen se presenta normalmente en cuatro tonos (blanco, gris claro, gris oscuro y negro), aunque el digitalizador trabaja con 256 tonalidades. Puede seleccionarse cualquiera de cuatro colores con las teclas de función empleando el programa suministrado.

También se proporcionan numerosos programas alternativos de muestra, de los cuales el más sofisticado es *Alarm*. Éste permite utilizar la cámara de video conjuntamente con el digitalizador a modo de sereno electrónico. El ordenador digitaliza continuamente la señal, tomando una instantánea de lo que esté enfocando la cámara aproximadamente cada cinco segundos.

Cuando la nueva imagen está lista, se compara con la imagen captada cinco segundos antes y, si es diferente en más de un número preestablecido de lugares, el Commodore 64 hará sonar una alarma. Este programa funcionó muy bien cuando usamos el sistema para vigilar una taza de café. Cuando la cantidad de diferencias que hacen que se dispare la alarma es baja (alrededor de 200), la alarma suena aun cuando alguien se limite a proyectar una ligera sombra sobre la zona vigilada. Obviamente, si estuviera vigilando algunas joyas de valor, el límite de diferencias se habría de establecer más alto, para que la alarma no se disparara por la mera sombra del dueño o cuando disminuyera la luz diurna.

Este programa da una impresión muy favorable de la sensibilidad de la unidad de Print-Technik. Con un Commodore 64 conectado a un verdadero sistema de alarma, habría una posibilidad entre cinco que de un ladrón no fuera detectado, ¡siempre que la escena se pudiera restablecer en el intervalo de un segundo!

Print-Technik también incluye un sencillo programa de proyección de diapositivas que permite exhibir en sucesión en la pantalla imágenes guardadas en formato *Koala-pad*. Por último, hay una rutina que permite presentar imágenes en formato *Koala-pad* desde los propios programas en BASIC usando una llamada SYS.

El digitalizador de Print-Technik transforma al Commodore 64 en una poderosa herramienta para producir imágenes gráficas de gran calidad. El hardware es compacto, produce resultados sorprendentemente buenos, y el software suministrado es amplio, aunque no está tan bien terminado como algunos programas. No obstante, cualquier aplicación práctica de la unidad requiere otros equipos: una cámara de video, una pantalla de video y un programa artístico razonable como el *Paintmagic* o el *Koala-pad*. El atractivo de la unidad se ve limitado, en consecuencia, a quienes la destinen a un uso serio como herramienta para producir gráficos interesantes y detallados o para emplear las aplicaciones más interactivas que hemos descrito.

Posibilidades de impresión
Las imágenes digitalizadas se pueden volcar a una gama de impresoras. Las impresoras MPS 801 y 1525 de Commodore producen en papel una imagen monocromática de cuatro tonalidades, pero con una impresora en color se puede imprimir la imagen en hasta 16 colores

MIDI IN

Finalizamos el proceso de adaptación de la interface MIDI a la gama Amstrad CPC

En nuestro proyecto inicial de una MIDI para los micros Commodore 64 y BBC desarrollamos un programa que permitía que un ordenador actuara como un simple grabador en tiempo real. Debido a que este programa se simplificó para demostrar los principios de la MIDI, los datos entrantes eran objeto de un procesamiento mínimo antes de su almacenamiento. Este procesamiento comprobaba los mensajes del sistema, desechando los que se encontraran y calculando los bytes de temporización a insertar entre los bytes MIDI antes de almacenar los datos en la memoria. Puesto que el intervalo mínimo entre dos bytes MIDI consecutivos es de 320 µs, el bucle de procesamiento para cada byte entrante se mantuvo mucho más corto que este período de tiempo crítico, para asegurar que se llevara a cabo todo el procesamiento y que el programa estuviera en condiciones de recibir el siguiente byte antes de que llegara éste.

Un programa más útil habría llevado a cabo un procesamiento mucho más intensivo de los datos entrantes. Si no fuera por el límite de tiempo de 320 µs impuesto sobre el procesamiento, se podría disponer de facilidades tales como grabación y reproducción simultáneas, visualización en pantalla y superposición de comentarios de canales múltiples. Afortunadamente existe un modo, que se implementa comúnmente en sistemas de ordenador, de "escuchar" los bytes entrantes mientras simultáneamente se procesan datos que ya se han recibido.

Para demostrar los principios de la escritura de un programa MIDI que posea rutinas de procesamiento largas (de más de 320 µs) veamos un programa para leer datos MIDI entrantes y visualizarlos en la pantalla en tiempo real. Los datos se visualizarán en notación hexadecimal, comenzando cada mensaje MIDI en una nueva línea. Para hacer esto hay que convertir cada byte MIDI recibido en dos dígitos hexadecimales en código ASCII, enviarlos a la pantalla seguidos por dos caracteres de espacio para separarlos del siguiente byte y, si el byte es el último del mensaje MIDI, enviar un retorno de carro. Las cabeceras del sistema operativo, como la exploración del teclado, son importantes en el contexto de una velocidad de transmisión de datos de 320 µs y, por tanto, es probable que una rutina de procesamiento de este tipo emplee más tiempo que el período crítico entre la recepción de cada byte MIDI sucesivo.

Para no perder datos cuando éstos entran, podríamos usar una interrupción regular en nuestro bucle principal de procesamiento que explorara el registro de estado ACIA y transfiriera todo dato recibido recientemente desde el registro de recepción de datos ACIA a una zona de tamponamiento en la memoria del ordenador. El programa principal podría entonces leer datos de la parte frontal del tampón en un momento adecuado sin que se perdiera ningún dato. Lamentablemente, el intervalo entre interrupciones "de sondeo" necesitaría ser de menos de 320 µs y, como tal, constituye una cabecera de programación inaceptable.

La alternativa a la estructura de interrupciones periódicas es aprovechar la capacidad del ACIA de generar su propia señal de interrupción cada vez que el registro de recepción de datos esté lleno. Por supuesto, necesitamos interceptar la rutina de servicio de interrupciones del Amstrad para distinguir las interrupciones ACIA de otras fuentes de interrupción. Si se detectara una interrupción ACIA, habría que transferir el contenido del registro de recepción de datos.

El programa que vemos aquí utiliza las rutinas de firmware del Amstrad, de las que se ofrece una documentación completa en el manual de firmware y que se puede obtener de Amstrad (SOFT 158). El salto a la rutina para manipular la interrupción ACIA se halla en la posición &003B. La propia rutina de manipulación de interrupción comunica con el programa procesador principal, activando rápidamente un evento, descrito por el bloque de eventos situado en la dirección etiquetada rxevbk.

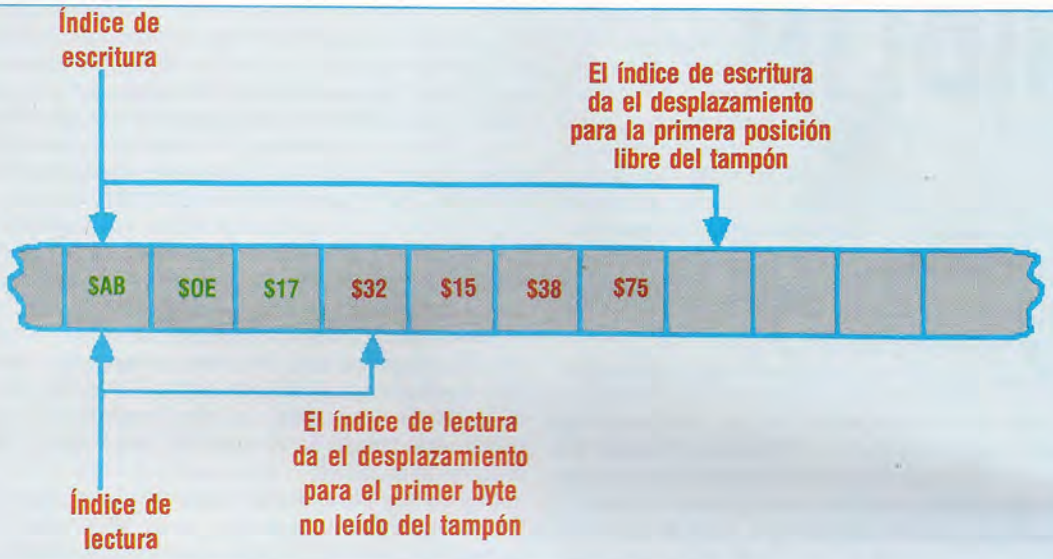
La zona de tampón está definida como 128 bytes de memoria comenzando desde la dirección rxevbk+9. La rutina comprueba primero si hay algún evento destacable esperando a ser procesado. De no ser así, el tampón está vacío y se inicializan los índices de lectura y escritura. La rutina pasa luego a comprobar si ha habido extralimitación del

Pausa musical
El sistema operativo Amstrad posee un método de proporcionar interrupciones que se conoce como *manipulación de eventos*. Se puede utilizar un bloque de eventos para manipular interrupciones generadas por el chip ACIA de la interface MIDI cada vez que su registro de recepción de datos esté lleno. Los seis primeros bytes manipulan funciones del sistema, pero los bytes 7 y 8, y los 128 siguientes, se han preparado en nuestro programa para actuar como tampón de entrada de datos

Caroline Clayton

Punteros del tampón

Señalando el camino
Los punteros del principio y el final de la cola del tampón indican los puntos en los cuales se suprimen o insertan datos. Durante la operación, los punteros se irán moviendo progresivamente por la zona de tampón a medida que se vayan suprimiendo datos y añadiendo otros nuevos. Cada vez que el tampón quede vacío (lo que se indicaría cuando los índices READ [lectura] y WRITE [escritura] tuvieran el mismo valor) los índices se restablecen al comienzo del tampón



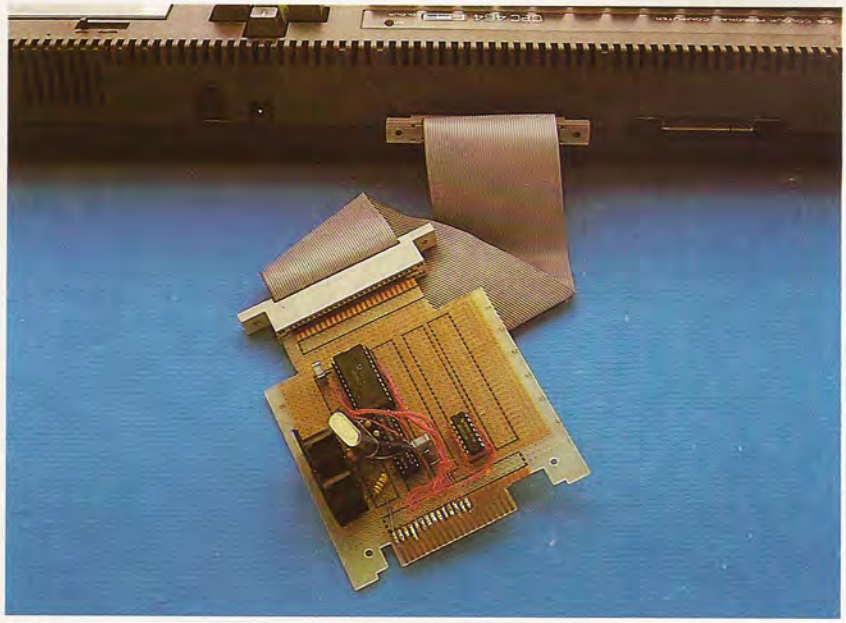
receptor: la llegada de otro byte al ACIA antes de que se haya leído el anterior. La rutina de interrupción debe borrar la fuente de interrupción, ya que de lo contrario se repetiría a sí misma cada vez que se saliera de la rutina manipuladora y se reactivarían las interrupciones, haciendo que la máquina quedara "bloqueada".

La señal de interrupción del ACIA se puede desactivar leyendo el registro de datos ACIA. La rutina calcula luego la dirección de la zona de tampón en la que colocará el byte de datos recibido desde el índice almacenado en el bloque de eventos en rxevbk+8. Este índice es, simplemente, el desplazamiento del final actual de la zona de tampón desde su encabezamiento. Si el tampón no está lleno (tal como sucedería si el índice fuera menor que 128), el byte de datos se escribe en el tampón y se activa rápidamente el evento.

El programa principal simplemente inicializa los registros ACIA y el bloque de eventos, tras lo cual da entrada a un bucle que toma caracteres de la parte de adelante del tampón y los procesa de la forma descrita anteriormente. La dirección de la posición que representa la parte de adelante del tampón se halla a partir del índice de lectura, en la dirección rxevbk+7 dentro del bloque de eventos. Este índice se retiene nuevamente como un desplazamiento desde el comienzo del tampón.

El programa sale si se pulsa cualquier tecla del teclado del ordenador, si el tampón está lleno o si el programa pierde algún byte de datos debido a la extralimitación del receptor.

Conectado en interface
Nuestra placa MIDI rediseñada se conecta a la puerta de ampliación del Amstrad a través de un cable plano de 50 vías y conectores de borde. Utilizando zócalos DIN estándares de 5 patillas, se pueden conectar hasta 16 dispositivos de equipo MIDI y controlarlos mediante el software del ordenador



Los problemas del programa

El programa que ofrecemos está diseñado para demostrar algunos de los principios de la utilización de interrupciones no periódicas con el sistema operativo Amstrad. Es interesante observar algunas de las dificultades del empleo de tal método. Si usted ejecuta el programa y envía datos desde un teclado MIDI, encontrará que ocasionalmente el programa terminará debido a una extralimitación del ACIA, es decir, que se reciba un segundo byte antes de que el byte anterior haya sido borrado del registro de recepción de datos. Esta particularidad obedece a la forma en que el sistema operativo manipula las interrupciones.

La respuesta de interrupciones del Amstrad utiliza el registro alternativo del Z80, establecido temporalmente para almacenar el estado de los registros CPU cada vez que se produce una interrupción. Puesto que sólo está disponible un conjunto de registros alternativos, no se toleran interrupciones de nivel múltiple. Esto significa, a su vez, que las interrupciones se deben inhabilitar a través de la rutina de servicio de interrupciones. Además, debido a que la rutina de servicio incluye varias llamadas para soportar la estructura de eventos del firmware, las interrupciones se podrían inhabilitar durante más del período mínimo de 430 µs crucial, lo que probablemente provocaría la pérdida de uno o más bytes.

Una solución correcta a este problema de extralimitación supondría la sustitución del código manipulador de interrupciones por nuestro propio código introducido en el flujo. Éste, efectivamente, ignoraría las interrupciones periódicas y limitaría seriamente el uso que pudiéramos hacer de las funciones del sistema en firmware.

Cargador hexa:

```
20 org=&408E:loadptr=0
30 WHILE bandera=0
40 GOSUB 1000:IF bandera=1 THEN 60:REM salir
50 POKE org + loadptr,entrada:loadptr=loadptr + 1
60 WEND
70 END
1000 REM **** s/r toma un byte hexa ****
1020 bandera=0 LINE INOUT entrada$
1025 IF entrada$="x" THEN bandera=1:RETURN
1030 IF LEN(entrada$) <> 2 THEN PRINT"error":
     GOTO 1020
1050 entrada=VAL("&"+ entrada$):RETURN
```

Si no se posee ensamblador, el programa en código máquina se puede cargar utilizando el cargador hexa en BASIC que ofrecemos aquí. Los bytes desde la columna izquierda se deben entrar de uno en uno con el cargador hexa en ejecución, y luego guardarlos en disco o en cinta usando la instrucción SAVE'NOMBREARCHIVO',B, &4000,&199. Observe que cada byte debe ir seguido por un retorno de carro; de modo que, p. ej., los tres bytes de la segunda línea, 113B00, se deben entrar como 11 <CR> 3B <CR> 00 <CR>

Programa MIDI Amstrad

```
                org     #4000
klev:           defs    2
rxevbk:         defs    140             ;bloque eventos+lectura/escritura
cont:           equ     #f8e0
txreg:          equ     #f9e0
stat:           equ     #fae0
rxreg:          equ     #fbe0
;direcciones de llamada os
kmrdch:         equ     #bb09
kmarbr:         equ     #bb45          ;prepara rupturas
txtout:         equ     #bb5a
klinev:         equ     #bcef
kleven:         equ     #bcf2
kldisa:         equ     #bd0a          ;anula evento asincr
klpoll:         equ     #b921          ;lleva evento retornos
klnext:         equ     #bcfb          ;toma siguiente evento sincr
kldosy:         equ     #bcfe          ;hace evento sincr
kldone:         equ     #bd01          ;evento sinc hecho
klsres:         equ     #bcf5          ;limpia cola eventos
start:                                 ;parchea rutina interrupción

F3              di
113B00          ld      de,#003b
211441          ld      h1,jpnist
010300          ld      bc,3
EDB0            ldir
FB              ei
        ;inicializa evento recibir
210240          ld      hl,rxevbk
115941          ld      de,rxevrt      ;dirección rutina evento
0601            ld      b,1
CDEFBC          call    klinev
3600            ld      (hl),0         ;inicializa lectura/escritura
23              inc     hl
3600            ld      (h1),0
        ;guardar rutina dir. evento kl
ED5BF3BC        ld      de,(kleven+1)
CBBA            res     7,d
CBB2            res     6,d            ;suprimir bits seleccion rom
ED530040        ld      (klev),de
        ;inicializar 6850
01E0F8          ld      bc,cont
3E03            ld      a,3
ED79            out     (c),a
3E96            ld      a,#96
ED79            out     (c),a
        espera:
CDFBBC          call    klnext
DC0941          call    c,dosync       ;hacer evento si espera
210A40          ld      hl,rxevbk+8
CB7E            bit     7,(hl)
2005            jr      nz,waitl       ;salir si tampón lleno
CD09BB          call    kmrdch
30EE            jr      nc,wait        ;saltar si ninguna tecla pulsada
        espera 1:
CB76            bit     6,(hl)
280C            jr      z,wait2        ;saltar mensaje si no extralim.
21F340          ld      hl,string
0616            ld      b,strlen
7E      nexc:   ld      a,(hl)
CD5ABB          call    txtout
23              inc     hl
10F9            djnz    nexc
        espera 2:
CDF5BC          call    klsres
F3              di
3EC9            ld      a,#c9          ;retiene instrucción
323B00          ld      (# 003b),a
3E16            ld      a,# 16
ED79            out     (c),a          ;desactiva interrupciones 6850
FB              ei
C9              ret                    ;salida a editor
0A0D    string: defb    #0A,#0d
41434941        defm    "error
        extralim. ACIA"
0A0D            defb    #0a,#0D
                strlen: equ $-string
        dosync:                        ;efectúa evento sincr
E5              push    hl             ;guarda dirección evento
F5              push    af             ;guarda prioridad anterior
CDFEBC          call    kldosy         ;efectúa evento
F1              pop     af
E1              pop     hl
CD01BD          call    kldone
C9              ret
C31741  jpinst: jp      rcvint
        rcvint:                        ;rutina interrupción rcv
3A0440          ld      a,(rxevbk+2)
B7              or      a              ;comprobar cuenta=0
2006            jr      nz,rcvl
320940          ld      (rxevbk+7),a   ;inicializar índices
320A40          ld      (rxevbk+8),a
C5      rcv1:   push    bc             ;guarda estado rom antiguo
210A40          ld      h1,rxevbk+8    ;escribe dirección índice
CB91            res     2,c            ;habilita rom inferior
ED49            out     (c),c
01E0FA          ld      bc,stat
ED78            in      a,(c)
CB6F            bit     5,a
280A            jr      z,rcv15
36FF            ld      (h1),#ff       ;establece bandera extralim.
3E16            ld      a,#16
05              dec     b
05              dec     b
ED79            out     (c),a
1817            jr      rcv2
04      rcv15:  inc     b
ED78            in      a,(c)
34              inc     (hl)           ;incrementa índice
5E              ld      e,(hl)
CB7B            bit     7,e            ;comprueba si tampón lleno
200E            jr      nz,rcv2        ;sale si lleno
1600            ld      d,0
19              add     hl,de          ;toma dirección escritura
77              ld      (hl),a
210240          ld      hl,rxevbk
ED5B0040        ld      de,(klev)
CD1600          call    #0016
C1      rcv2:   pop     bc
ED49            out     (c),c
C9              ret
        rxevrt:                        ;rutina evento rx hl=par
210940          ld      hl,rxevbk+7
34              inc     (hl)           ;incrementa índice
5E              ld      e,(hl)
1C              inc     e
1600            ld      d,0
19              add     hl,de          ;apunta a siguiente dato lectura
7E              ld      a,(hl)
FEF8            cp      # f8
D0              ret     nc
B7              or      a
F27641          jp      p,evrt1
47              ld      b,a
3E0A            ld      a,#0a
CD5ABB          call    txtout
3E0D            ld      a,#0d
CD5ABB          call    txtout
78              ld      a,b
        evrt1:
CD7A41          call    hexpr          ;imprime car
C9              ret
                imphex:                ;impresión hexadecimal
0602            ld      b,2
                imphex1:
3E00            ld      a,0
ED6F            rld                    ;toma dígito mayor
CD9141          call    digasc         ;convierte a ascii
CD5ABB          call    txtout         ;imprime
10F4            djnz    hexprl
3E20            ld      a," "
CD5ABB          call    toxtout
CD5ABB          call    txtout
C9              ret
        digasc:                        ;dígito hexa a ascii
C630            add     a,"0"
FE3A            cp      ":"
D8              ret     c
C607            add     a,"A"-":"
C9              ret
```

Preestreno sorpresa

Antes de realizar el listado final nos ocuparemos de los procedimientos de clasificación

La utilización de árboles con estructuras internas irregulares (p. ej., tres bifurcaciones desde un nudo, dos desde otro, cuatro desde un tercero, etc.) plantea problemas especiales. La dificultad principal es saber si hemos llegado o no a un nudo terminal. Como recordará, cuando entramos el árbol de manipulación de objetos pudimos, en virtud de su coherencia interna, numerar los nudos por un orden especial: primero los nudos de elección; luego los nudos que saltaban fuera del árbol y después otra vez a él; en tercer lugar, los nudos terminales que dan por resultado una acción o un mensaje; y, por último, los nudos terminales que se limitaban a retornar sin que se emprendiera ninguna acción. Cuando seleccionábamos a lo largo del árbol (en las líneas 5030-5090), simplemente saltábamos a una rutina determinada por el número del nudo. Lamentablemente, si observa los árboles de las páginas 2104 y 2105 e intenta aplicarles este sistema, verá que no es posible.

Lo que se precisa es un sistema infalible para clasificar un árbol que se pueda aplicar a *cualquier* árbol, con independencia de su estructura interna.

Ahora vamos a resolver este problema y a entrar en el programa un árbol del trama. Veamos cómo lo hacemos...

En primer lugar, necesitamos inicializar las variables para la trama que aún no se hayan utilizado. Suprima la línea 190 de su programa y añada ésta:

```
190 DIM t(5,25,4),k(3,30),c(25),s(6),
h(6):z=0
```

Observará que ahora hemos aplicado la matriz t para hacer frente a nuestros nuevos árboles, y que también hemos añadido algo de espacio en la matriz c para las nuevas condiciones que comprobaremos. Las dos matrices s y h registran, respectivamente, si un personaje ha visto o no un cadáver o manipulado la lata vacía de alimento para gatos. La variable z indica si se ha producido alguna muerte, y se establecerá cuando sea adecuado para indicar el número de personaje de la víctima.

Nosotros programaremos nuestro árbol del siguiente modo. Los cuatro elementos de dimensiones inferior de la matriz t se utilizarán para retener información acerca de cada nudo del árbol, tal como se indica en la tabla de tipos de nudos. Luego, al clasificar a lo largo del árbol, será muy sencillo comprobar el tipo de nudo de cada uno y saltar a la rutina apropiada. Los datos de nudos se almacenan en el listado 1, que es el que debe entrar primero. Después entre las dos líneas siguientes para leer estos datos en la matriz t:

```
230 REM arbol de la trama
240 FOR n=1 TO 22: FOR s=1 TO 4: READ t(2,n,s):
    NEXT s: READ a$: NEXT n
```

Observe los espacios en blanco en las líneas 6270 y 6280. Éstos se añaden para facilitar la lectura, de modo que usted pueda ver con tan sólo una mirada cada grupo de valores de nudos. Los espacios en blanco se leen en la "variable de desperdicios" a$ en la línea 240.

Ahora necesitamos una rutina para "clasificar" este árbol, y saltar a las rutinas correspondientes. Ésta está retenida en las líneas 5400-5550 (listado 2), que es el que usted ha de entrar ahora. El proceso es muy directo. La línea 5470 comprueba el tipo de nudo y le suma uno para generar un número entre 1 y 7, que entonces da como resultado un salto a una de las líneas indicadas. Un nudo de elección simple con sólo dos bifurcaciones (tipo 0) saltará a la 5480, donde se utiliza el valor de la condición retenida en t(2,numero de nudo,2) para decidir si saltar al nudo retenido en t(2,numero de nudo,3) o a t(2,numero de nudo,4).

Cada uno de los restantes tipos de nudo saltan a sus propias líneas y extraen de la matriz t los parámetros correspondientes. Observe que los tipos de nudo 1 y 2 se vectorizan mediante un bloque de salto retenido en las líneas 4500-4570. Esto es para evitar tener que obstruir la rutina de clasificación de árboles con gran cantidad de números de línea distintos cuando vayamos añadiéndole árboles al programa.

Si se remite al diagrama del árbol de la trama, verá que es necesario comprobar algunas condiciones nuevas que todavía no hemos almacenado en la matriz c. Para añadirlas, entre la siguiente línea:

```
2450 c(13)=ABS(z=c): c(14)=ABS(g=C):
     c(15)=ABS(z=0): c(16)=ABS(s(c)=255):
     c(17)=ABS(FNc(z,2)=FNc(c,2)):
     c(18)=ABS(h(c)=255): c(19)=ABS(i=2)
```

Ahora entre los listados 3, 4 y 5. Ya estamos casi listos para probar esta parte del programa. Todo lo que se requiere es variar primero el bucle de control de la línea de modo que rece así:

```
500 REM
510 REM prueba bucle programa
520 REM
530 GOSUB 2100:GOSUB 2150:GOSUB 2240:PRINT:
    PRINT: GOSUB 1000: GOTO 530
```

Probabilidades

Nudo	Tipo	Datos retenidos en t(n.° árbol, n.° nudo,1-4) 1	2	3	4
0	Nudo de elección	0	n.° de condición	nudo al cual saltar si FALSO	VERD.
1	Nudos GOSUB (salir del árbol y después retornar)	1	0	nudo al cual retornar	n.° de rutina
2	Nudo de acción (nudo terminal, salta a rutina)	2	0	n.° de rutina	n.° mensaje, si hubiera
3	Nudo de mensaje (nudo terminal, imprime mensaje)	3	0	0	n.° de mensaje
4	Nudo terminal (salir del árbol, no emprender ninguna acción)	4	0	0	0
5	Nudo aleatorio (generar desplazamiento aleatorio y sumarlo al nudo base)	5	0	nudo base	desplazam. máximo
6	Múltiple opción (sumar valor de condición a nudo base)	6	condición	nudo base	0

Ahora suprima las líneas 540-820, que entramos cuando probamos nuestro árbol de manipulación de objetos, y digite el listado 6. Este listado comprueba las banderas "manipular" y "mover", inicializa condiciones y llama a cada árbol de a uno en las líneas 1200 y 1210. Ahora ya puede ejecutar el programa. En esta etapa se permite a los personajes desplazarse por las dependencias del *pub*, manipular objetos, morirse, e incluso incluso resolver el misterio de las empanadas envenenadas.

Listado 1

Estas líneas retienen los datos para los nudos. Se han incluido espacios ficticios para facilitar la legibilidad.

```
6240 REM
6250 REM datos arbol trama
6260 REM
6270 DATA 0,13,2,22," ",0,14,5,3," ",0,15,21,4," ",
     5,0,18,3," ",0,16,6,7," ",0,17,7,11,"
     ",0,18,9,8," ",0,16,12,13," ",0,19,10,17,"
     ",5,0,14,3," ",1,0,7,1," ",4,0,0,0," ",2,0,1,0,"
     ",2,0,5,1," ",4,0,0,0," ",4,0,0,0," ",2,0,2,4,"
     ",2,0,3,2," ",2,0,4,3

6280 DATA " ",2,0,4,0," ",4,0,0,0," ",4,0,0,0," "
```

Listado 2

Estas líneas clasifican a lo largo de tres árboles, comprobando el tipo de cada nudo y bifurcándose en consecuencia

```
5440 REM
5450 REM clasifica arboles
5460 n=1
5470 ON (t(t,n,1)+1)GOTO 5480,5490,5500,
     5520,5530,5540,5550
5480 k=c(t(t,n,2))+1: n=t(t,n,2+k): GOTO 5470
5490 GOSUB 4530: n=t(t,n,3): GOTO 5470
5500 GOSUB 4570
5510 RETURN
5520 GOSUB 4680: GOSUB 4630: GOSUB 4720
5530 RETURN
5540 a=t(t,n,4): GOSUB 4350: n=t(t,n,3)+ v:GOTO
     5470
5550 k=c(t(t,n,2)): n=t(t,n,3)+k:GOTO 5470
```

Puestos de acción
Las líneas 2810-2920 se encargan de trasladar a los personajes de una habitación a otra. Las líneas 3000-3110 forman la primera parte de la tabla de acciones. Esta tabla contiene las rutinas llamadas por los nudos tipo 2 mediante el bloque de saltos de las líneas 4510-4570. Las rutinas para los "nudos GOSUB" (tipo 1) empiezan en la línea 3900

Listado 3

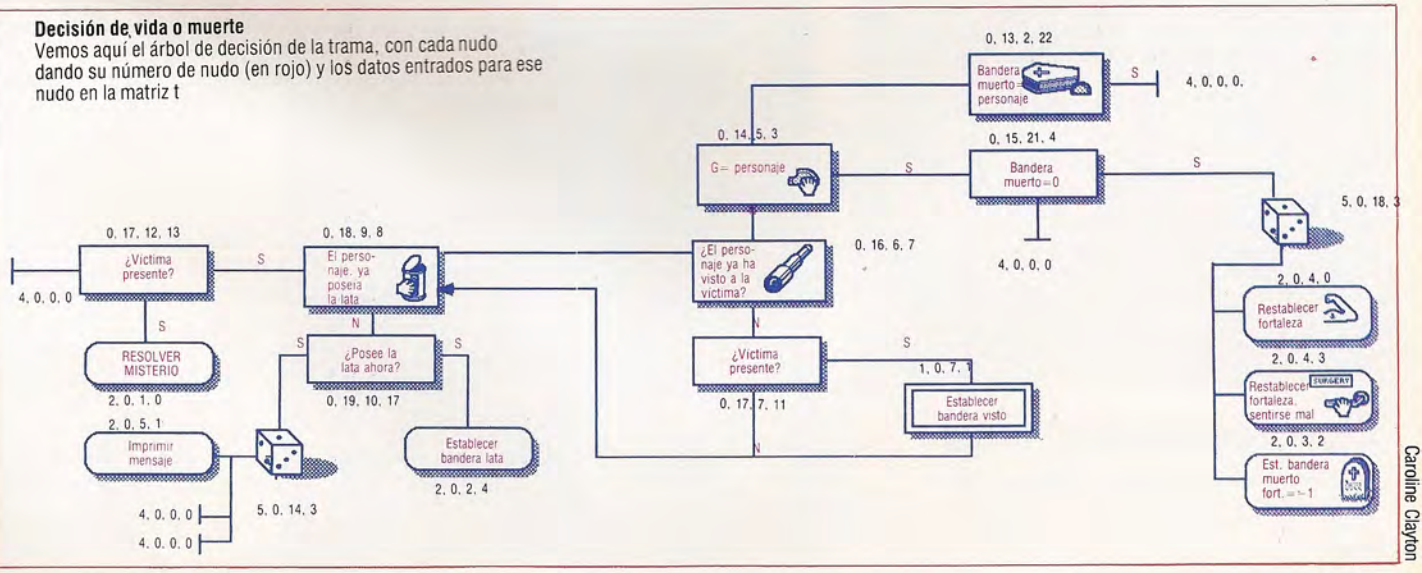
```
2810 REM
2820 REM desplaza un personaje
2830 REM
2840 IF FNc(c,4)<1 THEN RETURN: REM demasiado debil
     para moverse
2850 y=0: f=0: FOR w=2 TO 5: IF I$(VAL(c$(c,2)),w)="0"
     THEN GOTO 2910
2860 GOSUB 4180: IF q=1 THEN f=1: GOTO 2880
2870 GOTO 2910
2880 IF FNc(c,2)=r THEN PRINT c$(c,1);" sale de la
     habitación...";: y=1
2890 c$(c,2)=1$(VAL(c$(c,2)),w): w=5: IF FNc(c,2)=r
     THEN PRINT c$(c,1); " entra en la habitación...";
     : y=1
2900 IF y=1 THEN y=c: GOSUB 2250: c=y: PRINT:
     PRINT: REM actualizar mensaje personajes
     presentes
2910 NEXT w: IF f=0 GOTO 2850
2920 RETURN
3000 REM
3010 REM tabla de acciones
3020 REM
```

Rutinas útiles
Esta parte del programa añade algunas rutinas de bajo nivel nuevas, para generar números aleatorios variables e imprimir mensajes

Listado 4

```
4270 REM
4280 REM imprimir su
4290 REM
4330 REM rutina numeros aleatorios variables
4340 REM
4350 v=INT(RND(2)* a):RETURN
4360 REM
4370 REM imprimir el
4380 REM
4510 REM bloque de saltos
4520 REM
4530 REM
4540 ON t(t,n,4) GOSUB
4550 RETURN
```

Decisión de vida o muerte
Vemos aquí el árbol de decisión de la trama, con cada nudo dando su número de nudo (en rojo) y los datos entrados para ese nudo en la matriz t



Caroline Clayton

```
3030 FOR n=1 TO 3:GOSUB 4090: NEXT n: m$=c$(c,1)+"
     vuelve a mirar el cuerpo y de pronto comprende la
     espantosa verdad. "+CHR$(34)+ "La empanada esta
     elaborada con alimento para gatos"+CHR$ (34)+ "
     ":GOSUB 4630
3040 GOSUB 4390: m$=m$+ "grita, y en una loca carrera
     los clientes reunidos se arremolinan sobre la barra y
     atacan a Fred el barman, quien les suplica en vano que
     tengan piedad de su miserable vida.": GOSUB
     4630:GOSUB 4720
3050 FOR n=1 TO 2000:NEXT n: GOSUB 4720: m$="...y al
     dia siguiente a Fred el barman no se lo ve por ninguna
     parte. No obstante, hay muchisimas empanadas de
     extraña forma para alimentar a la siempre famelica
     clientela del Dog and Bucket...": GOSUB 4630:GOSUB
     4720: END
3060 h(c)=255:GOSUB 4680: m$=c$(c,1)+ m$: GOSUB
     4630: GOSUB 4720: RETURN
3070 z=c: GOSUB 4680: n$=c$(c,1)+m$:GOSUB
     4300:m$=n$+m$+ "estomago, e inmediatamente
     muere. Los otros personajes estan demasiado
     concentrados en sus bebidas como para darse
     cuenta...": GOSUB 4630: GOSUB 4720: g=0:
     c$(c,4)="-1": RETURN
3080 c$(c,4)="10": g=0: IF t(t,n,4) > 0 THEN GOSUB
     4680:m$=c$(c,1)+m$:GOSUB 4630: GOSUB
     4720
3090 RETURN
3100 a=20: GOSUB 4350: IF v<>5 THEN RETURN
3110 GOSUB 4680: GOSUB 4630: GOSUB 4720:
     RETURN
3900 REM
3910 REM gosub tabla
3920 REM
3930 s(c)=255: m$=c$(c,1)+ "se arrodilla junto al cuerpo
     postrado de "+c$(z,1)+". La horrible verdad toma
     cuerpo lentamente, pero los demas parecen estar
     demasiado borrachos como para prestarle ninguna
     atencion inmediata...": GOSUB 4630: GOSUB 4720:
     RETURN
```

```
4560 REM nudos de accion
4570 ON t(t,n,3) GOSUB 3030,3060,3070,3080,3100:
     RETURN
4600 REM
4610 REM imprimir mensajes si el jugador esta presente
4620 REM
4630 IF FNc(c,2)=r THEN PRINT m$;
4640 m$="": RETURN
4650 REM
4660 REM seleccionar un mensaje de la sentencia de
     datos
4670 REM
4680 RESTORE 7030: FOR m=1 TO t(t,n,4):READ m$: NEXT
     m: RETURN
4690 REM
4700 REM imprimir una linea en blanco
4710 REM
4720 IN FNc(c,2)=r THEN PRINT
4730 RETURN
```

Listado 5

Recibir el mensaje

La línea 4680 emplea líneas DATA para recuperar mensajes. Lamentablemente, los ordenadores Commodore no pueden RESTORE a un número de línea, de modo que se ha de utilizar una técnica distinta; remítase a los correspondientes complementos que ofreceremos en el próximo capítulo

```
7000 REM
7010 REM datos mensajes
7020 REM
7030 DATA "Un extraño olor invade el aire... ¿podria ser el
     aroma de Catty-Kit A La Carte?", "de pronto se
     desploma sobre el suelo, apretandose el estomago",
     "parece muy enfermo, y advierte a los demas que no
     toquen la empanada."
7040 DATA "examina atentamente la lata, y su semblante
     adquiere un aire pensativo."
```

Complementos al BASIC

Tal como está impreso, el listado se ejecutará sin ninguna modificación en la gama de ordenadores Amstrad. En el próximo capítulo incluiremos complementos para otras máquinas

Listado 6

Dirigiendo el reparto

El manipulador de personajes procesa a cada personaje de uno en uno. Estas líneas comprueban las banderas «mover» y «manipular» y llaman a las rutinas adecuadas para trasladar personajes o clasificar árboles. Se han incluido algunas sentencias REM para facilitar la lectura

```
1000 REM
1010 REM manipulador personajes
1020 REM
1030 REM comprueba si se ha pulsado alguna tecla
1040 GOSUB 4260: IF i$<> "" THEN GOSUB 2040: RETURN
1050     REM procesa cada personaje por turno
1060 FOR c=1 TO 6
1070     REM comprueba 'bandera manipular'
1080 IF FNc(c,10)>0 THEN c$(c,10)=FNm$(c$(c,10),1):
     GOTO 1500
1090     REM bandera=0 para restaurar bandera y procesar
         personaje
1100 RESTORE: FOR n=1 TO c*10+c-1: READ c$(c,10):
     NEXT n
1110 IF FNc(c,10)=0 THEN GOTO 1500: REM valor por
     defecto=0 por tanto no procesa
1120     REM comprueba bandera mover
1130 IF FNc(c,11)> 0 THEN c$(c,11)=FNm$(c$(c,11),1):
     GOTO 1190
1140     REM bandera mover=0 por tanto restaura bandera y
         mueve personaje
1150 RESTORE: FOR n=1 TO c*11: READ c$(c,11): NEXT n
1160 IF c$(c,11)="0" THEN GOTO 1180
1170 GOSUB 2840: GOTO 1500
1180     REM clasificar a lo largo de los arboles
1190 GOSUB 2400: REM inicializa condiciones
1200 t=2:GOSUB 5460: REM arbol trama
1210 IF FNc(c,4) >0 THEN GOSUB 5000: REM arbol
     manipulacion objetos
1500 NEXT c: GOTO 1030: REM hacer siguiente personaje -
     cuando todos terminados volver a comprobar si tecla
     pulsada/hacerlo otra vez
```

Los designios del azar

Es imprescindible recurrir al cálculo de probabilidades para dar un enfoque científico a los juegos de azar. Con este fin, implementaremos la regla de Bayes

Las apuestas se basan totalmente en las probabilidades y, sin embargo, la mayor parte de los apostadores tienen una idea sumamente vaga acerca de la teoría de las probabilidades. Ésta es, posiblemente la razón por la cual continúan perdiendo dinero. El tema de las probabilidades está lleno de paradojas y trampas ocultas, de modo que no es sorprendente que nos resulte difícil enunciar una teoría que sea precisa acerca de la incertidumbre. El apostador puede llegar muy lejos con la ayuda de un micro, un poco de autodisciplina y cierto conocimiento de la teoría elemental de las probabilidades.

Lo primero que debe saber el apostador bien informado es cómo convertir las posibilidades en probabilidades y viceversa, a pesar de lo cual es sorprendente el escaso número de ellos que así lo hacen. Existen dos clases de posibilidades: las a favor (F) y las en contra (C). Los corredores de apuestas suelen hacer explícitas las posibilidades de un evento de resultado, como de que un caballo gane una carrera, excepto cuando el evento es lo que se dice "seguro". El cuadro se complica por la práctica usual de citar un par de enteros tales como 6 a 4 para definir las posibilidades, cuando 3 a 2 parece mucho más directo.

Es una buena idea simplificar las cosas, reduciendo las variedades de posibilidades a una media uniforme. Esto nos será de ayuda cuando lleguemos a los cálculos informatizados. El primer paso para convertir las posibilidades en probabilidades es expresar las posibilidades con un único número. Esto se puede hacer utilizando una de las fórmulas consignadas abajo, donde la c (contra) y f (favor) minúsculas son los dos números citados por el corredor de apuestas.

$$F=f/c$$
$$C=c/f$$

De modo que si las posibilidades se dan como 100 a 30 en contra, c=100 y f=30. Por tanto, las posibilidades a favor (F) son 30/100, lo que es igual a 0,30, y las posibilidades en contra son 100/30=3,3333. Ahora hemos reducido las posibilidades a una escala común: 7 a 2 es 3,5 a 1, 100 a 30 es 3,3333 a 1, 11 a 4 es 2,75 a 1, y así sucesivamente. Con ello ya se facilitan las comparaciones.

El siguiente paso es transformar las posibilidades en probabilidades, con una de estas fórmulas:

$$P=F/(F+1)$$
$$P=1/(C+1)$$

Éstas nos dan las probabilidades de un evento cuyas posibilidades son F a 1 a favor o C a 1 en contra. De modo que 100/30 o 3,3333 a 1 dan por resultado unas probabi-

Teoría de las probabilidades

El estudio serio de las probabilidades matemáticas comenzó, como cabría esperar, con una serie de apuestas perdidas. El escritor francés Antoine Gombaud, caballero de Méré (1607-1685), había logrado ganar mucho dinero apostando repetidas veces que él podía, en cuatro tiros de un dado, sacar al menos un seis. Lamentablemente, su éxito con esta apuesta fue origen de una clara retracción de apostadores. Por lo tanto, en 1654 pasó a apostar que en 24 tiradas de un par de dados arrojaría al menos una vez un doble seis. En aquel entonces, el único método de determinar las probabilidades en tales casos consistía en lanzar los dados tantas veces como fuera posible y registrar el resultado. Esto no sólo es tedioso, sino también inexacto, de modo que cuando la nueva apuesta del caballero comenzó a amenazarlo con la bancarrota, le escribió al matemático Blaise Pascal recabando su ayuda. De esta forma las probabilidades comenzaron a estudiarse seriamente. Las posibilidades de sacar un seis en cuatro tiradas son de 14 a 13 a favor. Aplicando la fórmula presentada en este capítulo, ¿podría decir por qué la segunda apuesta le hizo perder dinero al caballero?

Imagen digitalizada perteneciente a "How to take a chance", de Darrell Huff

lidades de 1/4,3333 = 0,2308. Las probabilidades (P) siempre se expresan dando por sentado que el evento se produzca verdaderamente (como que un caballo gane). Si usted desea las probabilidades de que el evento *no* se produzca (Q), puede usar la fórmula:

```
Q=1-P
```

Recuerde que P + Q=1, las probabilidades de que un evento se produzca más las probabilidades de que ese evento no tenga lugar deben sumar 1. Observe también que las posibilidades elevadas (en contra) están asociadas con eventos improbables y con escasas probabilidades (a favor).

Las posibilidades de los corredores de apuestas son menores que las auténticas posibilidades; por decirlo en otras palabras, hacen que los eventos en cuestión parezcan más probables de lo que son en realidad. Las razones de esto se hacen evidentes cuando analizamos los hechos desde el punto de vista del corredor de apuestas. Éste ha de ganarse la vida y cubrir sus gastos. En consecuencia, crea lo que podría denominarse un registro de apuestas "compensatorio". Podemos ilustrar esto con una carrera de caballos imaginaria:

Corredor	Apuestas	Posib.	Riesgo
Apricot	10 000 pts.	5-6	18 333 pts.
Olivetti	7 500 pts.	13-8	19 687 pts.
Alice	2 000 pts.	8-1	18 000 pts.
Apple	500 pts.	33-1	17 000 pts.
	20 000 pts.		

En este caso, se han apostado 20 000 pts, la mitad de esta cifra a *Apricot* y el resto tal como se indica. Las posibilidades se han escogido como para mantener el riesgo del corredor de apuestas por debajo del total apostado, es decir, 20 000 pts, suceda lo que suceda. En consecuencia, si gana *Alice*, los apostadores que lo respaldaron por la cantidad de 2 000 pts, recuperarán sus apuestas, más ocho veces lo que apostaron (16 000 pts) en concepto de beneficios, lo que totalizan 18 000 pts. Esto deja un margen de beneficio de 2 000 pts (el 10 %) para el corredor de apuestas. En un registro de apuestas imparcial, *Apricot* sería dinero a la par y *Alice* par-

Probabilidades posibles

Trabajemos las posibilidades de la conversión de probabilidades. Abajo vemos las posibilidades citadas en un cupón de apuestas para un partido de fútbol entre equipos ingleses: Leicester contra Everton.

15/8 Leicester 5/2 Everton 1/1

Esto indica que las posibilidades en contra de que el equipo local (Leicester) gane son de 15 a 8; las posibilidades de un empate son de 5 a 2, y las posibilidades de que el Everton gane fuera de casa son de 1 a 1, o a la par. (Este partido se jugó en el verano de 1985: Everton se anticipó en el marcador, pero al final ganó el Leicester por tres goles a uno, ¡de modo que los perdedores no siempre pierden!) En primer lugar, normalicemos las posibilidades: C=c/f.

Local	15/8	= 1,875
Empate	5/2	= 2,50
Visitante	1/1	= 1,0

Expresémoslas en probabilidades: P=1/(C+1).

Local	1/2,875	= 0,3478
Empate	1/3,5	= 0,2857
Visitante	1/2	= 0,5000
Total		= 1,1335

Observen que estas probabilidades suman más de uno: 1,1335, para ser exactos. Esto viola las leyes de las probabilidades, pero no es ilegal. Significa que en este partido el margen de los corredores de apuestas es del 13,35 %. Si recuerda que las posibilidades elevadas (en contra) se traducen en escasas posibilidades (a favor), verá que los corredores de apuestas reducen las posibilidades para obtener su porcentaje. Recuerde que las posibilidades de los corredores tienden a ser menores que las verdaderas posibilidades. Para ganar dinero tendrá que encontrar casos en los que no se hayan reducido demasiado.

Para ahorrarle parte del trabajo con lápiz y papel (y con la mente también), nuestro listado elimina la labor monótona de estos cálculos. También calcula el margen de los corredores de apuestas. Veamos dos muestras, una con el partido que mencionamos arriba y la segunda con el Derby de 1985 (con precios de partida) que se corrió en Epsom

```
 10 REM ******************************
 15 REM * CALCULOS DE PROBABILIDADES *
 20 REM ******************************
100 PRINT"Calculador de probabilidades:"
110 PRINT "Por favor de las posibilidades como dos numeros;"
120 PRINT "por ej. 3.1 para 3 a 1 etc."
130 PRINT "Use 0,0 para terminar una entrada."'
140 at%=&0102040A: REM formato
150 N=0
160 TP=0: TA=0
190 REM -- Bucle principal:
200 REPEAT
202   @%=4
210   N=N+1
220   PRINT "Corredor ";N;" su nombre es";
222   INPUT NOMBRE$
225   PRINT "Las posibilidades en contra son ";
230   INPUT C,F
240   IF C< =0 AND F<=0 THEN GOTO 310: REM salir
250   P=F/(F+C)
260   C=C/F
270   TP=TP+P: REM prob.totales
280   TC=TC+C: REM pos. totales
290   PRINT "Para el corredor no. ";N;", ",NOMBRE$
295   @%=at%
296   PRINT "las probabilidades son: ";P
300   PRINT
310   UNTIL C< =0 OR F< =0
320 N=N-1
322 @ %=at%
330 PRINT
333 PRINT "El importe en juego es ";100/TP;"%"
335 PRINT "Margen de ganancia de corredores:
    ";100*(TP-1);"%"
360 IF TP<1 THEN PRINT "Estás bromeando!";CHR$7
999 END
> RUN
```

tiría con 9 a 1. Pero ¿a quién le interesaría llevar un registro de apuestas imparcial?

Una particularidad de nuestro registro de apuestas imaginario que responde a la realidad es que a los *outsiders* se les dan posibilidades relativamente escasas. Ello significa que al corredor de apuestas por lo general le va mejor cuando gana un *long shot*, de modo que tenga cuidado al respaldar "caballos oscuros". Es probable que usted esté subvencionando opciones más unánimes. Esto se debe a que los apostadores se muestran reacios a ir muy por debajo del dinero a la par, de modo que los corredores se ven limitados en cuanto a las posibilidades que pueden ofrecer para los favoritos claros. Y lo compensan a costa de los que "no ofrecen esperanzas".

De cuando en cuando los corredores de apuestas no consiguen cubrir sus riesgos, ya sea porque reaccionen con demasiada lentitud a los cambios repentinos en las apuestas, o bien porque abren con cotizaciones no realistas, aunque tales situaciones son raras. No se quede con la impresión de que puede ganar a los corredores, porque no puede hacerlo. Lo que sí puede lograr es ganar algún dinero a sus compañeros apostadores, con el corredor actuando como intermediario y quedándose con su "tajada".

Jugando a las quinielas
Los apostadores rellenan las quinielas de muchas maneras, pero uno de los métodos más populares es el de 8 resultados fijos sobre 10, en el que el apostante coloca cruces junto a diez partidos. Conocidos los resultados, se toman las ocho mejores combinaciones para maximizar la cantidad de aciertos. Los métodos para seleccionar los partidos varían mucho, pero la posibilidad de haber escogido al azar ocho empates es sumamente remota

```
Calculador de probabilidades:
Por favor de las posibilidades como dos numeros;
por ej. 3,1 para 3 a 1 etc.
Use 0,0 para terminar una entrada.

Corredor 1 su nombre es ?local
Las posibilidades en contra son ?15,8
Para el corredor no. 1, local
la probabilidad es de: 0,3478

Corredor 2 su nombre es ?empate
Las posibilidades en contra son ?5,2
Para el corredor no. 2, empate
la probabilidad es de: 0,2857

Corredor 3 su nombre es ?visitante
Las posibilidades en contra son ?1,1
Para el corredor no. 3, visitante
la probabilidad es de: 0,5000

Corredor 4 su nombre es?
Las posibilidades en contra son ?0,0

El importe en juego es 88,2192%
Margen de ganancia de los corredores:
13,3540%
>
>
>RUN
```

```
Calculador de probabilidades:
Por favor de las posibilidades como dos numeros;
por ej. 3,1 para 3 a 1 etc.
Use 0,0 para terminar una entrada.

Corredor 1 su nombre es ?Slip Anchor
Las posibilidades en contra son ?9,4
Para el corredor no. 1, Slip Anchor
la probabilidad es de: 0,3077

Corredor 2 su nombre es ?Law Society
Las posibilidades en contra son ?5,1
Para el corredor no. 2, Law Society
la probabilidad es de: 0,1667

Corredor 3 su nombre es ?Damister
Las posibilidades en contra son ?16,1
Para el corredor no. 3, Damister
la probabilidad es de: 0,0588

Corredor 4 su nombre es ?Supreme Leader
Las posibilidades en contra son ?10,1
Para el corredor no. 4, Supreme Leader
la probabilidad es de: 0,0909
```

```
Corredor 5 su nombre es ?Lanfranco
Las posibilidades en contra son ?14,1
Para el corredor no. 4, Lanfranco
la probabilidad es de: 0,0667

Corredor 6 su nombre es ?Reach
Las posibilidades en contra son ?33,1
Para el corredor no. 6, Reach
la probabilidad es de: 0,0294

Corredor 7 su nombre es ?Theatrical
Las posibilidades en contra son ?10,1
Para el corredor no. 7, Theatrical
la probabilidad es de: 0,0909

Corredor 8 su nombre es ?Phardante
Las posibilidades en contra son ?40,1
Para el corredor no. 8, Phardante
la probabilidad es de: 0,0244

Corredor 9 su nombre es ?Royal Harmony
Las posibilidades en contra son ?40,1
Para el corredor no. 9, Royal Harmony
la probabilidad es de: 0,0244

Corredor 10 su nombre es ?Snow Plant
Las posibilidades en contra son ?100,1
Para el corredor no. 10, Snow Plant
la probabilidad es de: 0,0099

Corredor 11 su nombre es ?Petowski
Las posibilidades en contra son ?33,1
Para el corredor no. 11, Petowski
la probabilidad es de: 0,0294

Corredor 12 su nombre es ?Seurat
Las posibilidades en contra son ?33,1
Para el corredor no. 12, Seurat
la probabilidad es de: 0,0294

Corredor 13 su nombre es ?Shadeed
Las posibilidades en contra son ?7,2
Para el corredor no. 13, Shadeed
la probabilidad es de: 0,2222

Corredor 14 su nombre es ?Main Reason
Las posibilidades en contra son ?200,1
Para el corredor no. 14, Main Reason
la probabilidad es de: 0,0050

Corredor 15 su nombre es ?
Las posibilidades en contra son ?0,0

El importe en juego es 86,5215%
Margen de ganancia de los corredores:
15,5781%
>
```

Dentro de celdas

Habiendo escrito las rutinas de manipulación de gráficos y de procesamiento de fórmulas para nuestro programa de hoja electrónica, estamos en condiciones de añadir las rutinas que permiten entrar datos y fórmulas en la hoja electrónica y efectuar los cálculos que sean pertinentes

El código que manipula la entrada de fórmulas se halla comprendido entre las líneas 2000 y 2050. Esta sección de código utilizaba una instrucción INPUT de BASIC para tomar una fórmula del usuario. Esta fórmula, por supuesto, está en forma de infijos e inicialmente se almacena en D$. La rutina pasa luego a almacenar la fórmula en la matriz F$(). En esta etapa se borra la entrada correspondiente en la matriz que retiene la versión en notación polaca inversa, PS$().

Las fórmulas para las celdas de A1 a A15 se retienen en F$(1) a F$(15); las celdas de B1 a B15 se retienen en F$(16) a F$(30), y así sucesivamente. El elemento correcto de F$() se puede hallar en cualquier momento a partir de las coordenadas del cursor de la hoja electrónica, usando la fórmula (Y-1)*15+X.

La siguiente rutina, entre las líneas 2100 y 2250, se encarga de entrar los datos en una celda de la hoja. Esta subrutina se llama desde la rutina de exploración del teclado de la línea 1170 si se pulsa una tecla numérica. La rutina imprime un mensaje para indicar que usted está entrando datos en la celda actual y luego toma los datos carácter por carácter en la línea 2120. Si la tecla pulsada es una de las teclas del cursor o la barra espaciadora, entonces la rutina se termina y los datos numéricos que se hayan reunido hasta entonces se colocan en una matriz M(,). Las líneas 2149-2170 comprueban que los datos entrados sean válidos y, si lo fueran, añade el número a E$. Una limitación de nuestro programa de hoja electrónica es que cada celda puede aceptar un máximo de sólo cinco caracteres; por ello se comprueba la entrada extra en la línea 2160 y se imprime un mensaje en la línea 2220.

La rutina de cálculo empieza en la línea 2300 y actúa a modo de bucle de control para las rutinas de traducción a polaca inversa desarrolladas en los dos últimos capítulos, antes de evaluar verdaderamente las fórmulas, en la subrutina de la línea 2370.

La primera rutina va trabajando con la matriz de fórmulas de infijos, F$() y la correspondiente matriz de notación polaca inversa, PS$, traduciendo todas las fórmulas recién entradas a polaca inversa, llamando a la subrutina de la línea 4000. Si en el elemento actual que se está comprobando hay una entrada, entonces se llama a la rutina de evaluación.

La rutina de evaluación se vale de la comprobación de legalidad polaca inversa de la línea 4700, que deposita los elementos de la serie polaca inversa con la que está trabajando en una matriz, GS(). Estos elementos son ya sea operadores (como + y −) o bien operandos (A1, B5, 3, etc.). La rutina de evaluación trabaja luego sobre los elementos de la serie polaca inversa de acuerdo a las siguientes condiciones:

- Si el elemento es una constante (diferenciada de la dirección de celda por el hecho de que el carácter de la izquierda será numérico y no alfabético), entonces se coloca el valor en la matriz C() y su posición en C() se coloca en una pila, ST().
- Si el elemento es una dirección de celda, entonces se halla su valor a partir de la matriz M(,) y se coloca en C(). Su posición se coloca en la pila en C().
- Si el elemento es un operador, entonces se lleva a cabo la operación sobre dos (o, en el caso de un menos unario, uno) operandos previamente apilados. De hecho, la pila retiene las posiciones de los operandos de C() y, por tanto, los números que se toman de la pila se utilizan para localizar los elementos correctos en C(). El resultado se coloca en C() y se suprimen de la pila los dos operandos. Luego se coloca en la pila la posición del resultado en C().

Tras haber procesado la serie completa, el resultado de la evaluación de la serie estará retenido en el último elemento de C() y se podrá transferir al elemento apropiado de M(,), la matriz que retiene todos los valores de celdas.

Complementos al BASIC

Amstrad CPC 464/664:
Introduzca las siguientes modificaciones en la versión para el Commodore 64:

```
2010 LOCATE 1,22
2020 PRINT "NUEVA FORMULA          ";
     CHR$(11)
2103 LOCATE 1,22
2120 A$="":WHILE A$="":A$=INKEY$:WEND
2130 IF A$=CHR$(243) OR A$=CHR$(242) OR
     A$=CHR$(241) OR A$=CHR$(240) THEN
     2250
2180 LOCATE H(X+1-H1)-1,V(Y-V1+1)
2190 PRINT CHR$(24);SPACE$((5);CHR$(11)
2200 LOCATE H(X+1-H1)+4-LEN(E$),V(Y-V1
     +1):PRINT E$;CHR$(24)
2220 LOCATE 1,22
2305 LOCATE 1,22
2315 Q$=CHR$(J+64)+MID$(STR$(I),2,2):
     LOCATE 1,1:PRINT "CELDA:";Q$;""
```

BBC Micro:
Introduzca las siguientes modificaciones en la versión para el Commodore 64:

```
2010 PRINT TAB(0,22);
2103 PRINT TAB(0,22);
2120 A$=get$
2130 IF A$=CHR$(136) OR A$=CHR$(137) OR
     A$=CHR$(138) OR A$=CHR$(139) THEN
     2250
2180 COLOUR 2:COLOUR 129
2190 PRINT TAB(H(X+1-H1)-1,V(Y-V1+1)
     -1);" "
2200 PRINT TAB(H(X+1-H1)+4-LEN(E$),
     V(Y-V1+1)-1);E$
2205 COLOUR 1:COLOUR 128
2220 PRINT TAB(0,22);
2230 I$=GET$
2240 PRINT TAB(0,22);
2305 PRINT TAB(0,22);
2315 Q$=CHR$(J+64)+MID$(STR$(I),2,2):
     PRINT TAB(0,0);"CELDA:";Q$;""
```

Rutinas de entrada y cálculo

Commodore 64:

```
1170 IF A$>="0" AND A$=<"9" THEN GOSUB 2100:REM
     ENTRAR DATOS NUMERICOS
2000 REM *** RUTINA ENTRAR FORMULA ***
2010 GOSUB 1950:REM MOVER CURSOR A LINEA DE
     ENTRADA
2020 PRINT "NUEVA FORMULA:   "CU$
2030 INPUT "NUEVA FORMULA:";D$
2040 LET F$((Y-1)*15+X)=D$:LET PS$((Y-1)*15+X)=""
2050 GOSUB 1900:RETURN
2100 REM *****ENTRAR DATOS EN LA CELDA *****
2103 GOSUB 1950:REM MOVER CURSOR A LA LINEA DE
     ENTRADA
2104 PRINT "                              ENTRANDO -DATOS
2105 LET P=0: LET E$=""
2110 IF A$<>"" THEN 2150
2120 GET A$:IF A$="" THEN 2120
2130 IF A$=CHR$(29) OR A$=CHR$(157) OR A$=CHR$(17)
     OR A$=CHR$(145) THEN 2250
2135 IF A$=" " THEN 2250
2140 IF A$="."THEN 2160
2150 IF A$< "0" OR A$> "9" THEN 2120
2160 LET P=P+1:IF P>5 THEN 2220
2170 LET E$=E$+A$
2180 PRINT CO$;:FOR C=1 TO V(Y+1-H1)-1:PRINT
     CD$;:NEXT C
2190 PRINT TAB(H(X+1-H1)-1);CHR$(18); "                ";
     CU$
2200 PRINT TAB(H(X+1-H1)+4-LEN(E$));CHR$(18);E$
2210 GOTO 2120
2220 GOSUB 1950:REM MOVER CURSOR HASTA LINEA DE
     ENTRADA
2225 PRINT "                    ERROR-ENTRADA IGNORADA"
2230 GET I$:IF I$="" THEN 2230
2240 PRINT CO$;:FOR K=1 TO 22:PRINT CD$;:NEXT K
2245 PRINT "                                    ":GOTO
     2120
2250 LET M(Y,X)=VAL(E$):GOSUB 1900:RETURN
2300 REM ***** CALCULA LA HOJA *****
2305 GOSUB 1950:REM MOVER CURSOR A LINEA DE
     ENTRADA
2306 PRINT"          CALCULANDO-ESPERE POR FAVOR "
2310 FOR J=1 TO 15:FOR I=1 TO 15
2315 Q$=CHR$(J+64)+MID$(STR$(I),2,2):PRINT
     CO$;CD$;"CELDA:" ;Q$;" "
2320 LET P$=PS$((J-1)*15+I)
2325 IF P$<>"" THEN GOSUB 4700:GOSUB 2370:GOTO
     2350
2330 LET C$=F$((J-1)*15+I)
2335 IF C$="" THEN 2350
2340 GOSUB 4000:GOSUB 2370:REM  DECODIFICA
     FORMULA
2350 NEXT I,J:GOSUB 1700:RETURN
2370 REM ***** EVALUA FORMULA *****
2375 SP=0:FOR K=1 TO 20:ST(SP)=0:NEXT K
2380 FOR K=1 TO CP
2390 LET T$=G$(K)
2400 IF T$="+" THEN 2500
2405 IF T$="-" THEN 2510
2410 IF T$="*" THEN 2520
2415 IF T$="/" THEN 2530
2420 IF T$="^" THEN 2540
2425 IF T$="&" THEN 2550
2430 IF T$="" THEN 2450
2432 TE$=MID$(T$,1,1)
2435 IF (TE$> ="0" AND TE$<="9") OR TE$="."
     THEN C(K)=VAL(T$):GOTO 2445
2440 LET C(K)=M(ASC(MID$(T$,1,1))-64,VAL(MID$
     (T$,2,2)))
2445 SP=SP+1:ST(SP)=K
2450 NEXT K
2460 LET M(J,I)=C(K-1):RETURN
2500 C(K)=C(ST(SP)-1))+C(ST(SP)):ST(SP)=0:SP-1:
     ST(SP)=K:GOTO 2450
2510 C(K)=C(ST(SP)-1))-C(ST(SP)):ST(SP)=0:SP=
     SP-1:ST(SP)=K:GOTO 2450
2520 C(K)=C(ST(SP-1))*C(ST(SP)):ST(SP)=0:SP=
     P-1:ST(SP)=K:GOTO 2450
2530 C(K)=C(ST(SP-1))/C(ST(SP)):ST(SP)=0:SP=
     P-1:ST(SP)=K:GOTO 2450
2540 C(K)=C(ST(SP-1))^C(ST(SP)):ST(SP)=0:SP=
     P-1:ST(SP)=K:GOTO 2450
2550 C(K)=0-C(ST(SP)):ST(SP):=K:GOTO 2450
3010 DIM H(5),V(8),ST(20),ST$(20),E$(20),G$(20),
     C(20)
```

Sinclair Spectrum:

```
2000 REM ********************
2001 REM * ENTRAR FORMULAS *
2002 REM ********************
2010 PRINT AT 18,0;" ENTRAR NUEVA
     FORMULA "
2020 INPUT LINE D$
2030 LET F$((Y-1)*15+X,1 TO LEN D$)=D$:
     LETH$((Y-1)*15+X,1 TO )=""
2050 GO SUB 1900
2060 RETURN
2100 REM ****************************
2101 REM * ENTRAR DATOS EN LA CELDA *
2102 REM ****************************
2105 PRINT AT 18,0;"     ENTRANDO-DATOS "
2110 LET P=0: LET B$=""
2120 LET A$=INKEY$: IF A$="" THEN GO TO
     2120
2130 IF A$=CHR$ (13) THEN GO SUB 1650: GO
     TO 2250
2140 IF A$="." THEN GO TO 2160
2150 IF A$< "0" OR A$>"9" THEN GO TO 2120
2160 LET P=P+1: IF P> 5 THEN GO TO 2220
2170 LET B$=B$+ A$
2180 PRINT AT V(Y+1-V1),H(X+1-H1);" "
2190 PRINT AT V(Y+1-V1),H(X+1-H1)+
     5-LEN B$;B$
2210 GO TO 2120
2220 PRINT AT 18,0;" ERROR-ENTRADA
     IGNORADA "
2230 LET I$=INKEY$: IF I$="" THEN GO TO
     2230
2240 PRINT AT 18,0;"                ":GO TO 2110
2250 LET M(Y,X)=VAL (B$): GO SUB 1900
2260 RETURN
2300 REM ********************
2301 REM * CALCULAR LA HOJA *
2302 REM ********************
2305 PRINT AT 18,0;"  CALCULANDO-ESPERE
     POR FAVOR"
2310 FOR J=1 TO 15: FOR I=1 TO 15
2315 PRINT AT 0,0;"CELDA:";CHR$
     (J+64);STR$ (I); ""
2320 LET P$=H$((J-1)*15+I,1 TO)
2325 IF P$(1)<>" "THEN GO SUB 2355: GO
     SUB 4700: GO SUB 2370: GO TO 2350
2330 LET C$=F$((J-1)*15+I,1 TO)
2335 IF C$(1)=" " THEN GO TO 2350
2340 GO SUB 4000: GO SUB 2370
2350 NEXT I: NEXT J: GO SUB 1700: RETURN
2355 FOR Z=1 TO LEN P$: IF P$(Z)= "" THEN
     GO TO 2357
2356 NEXT Z
2357 LET P$=P$(1 TO Z-1):RETURN
2370 REM ************************
2371 REM * DECODIFICAR FORMULA *
2372 REM ************************
2375 LET SP=0: DIM S(20)
2380 FOR K=1 TO CP
2390 LET T$=G$(K)
2400 IF T$(1)="+" THEN GO TO 2500
2405 IF T$(1)="-" THEN GO TO 2510
2410 IF T$(1)="*" THEN GO TO 2520
2415 IF T$(1)="/" THEN GO TO 2530
2420 IF T$(1)="^" THEN GO TO 2540
2425 IF T$(1)="&" THEN GO TO 2550
2430 IF T$(1)="" THEN GO TO 2450
2432 LET U$=T$(1)
2435 IF (U$>="0" AND U$<="9") OR
     U$="." THEN LET C(K)=VAL (T$): GO TO
     2445
2440 LET C(K)=M(CODE (T$( TO 1))-64,VAL
     (T$(2 TO )))
2445 LET SP=SP+1: LET S(SP)=K
2450 NEXT K
2460 LET M(J,I)=C(K-1):RETURN
2500 LET C(K)=C(S(SP-1))+C(S(SP)): LET
     S(SP)=0: LET SP=SP-1: LET S(SP)=K:
     GO TO 2450
2510 LET C(K)=C(S(SP-1))-C(S(SP)): LET
     S(SP)=0; LET SP=SP-1: LET S(SP)=K:
     GO TO 2450
2520 LET C(K)=C(S(SP-1))*C(S(SP)): LET
     S(SP)=0: LET SP=SP-1: LET S(SP)= K:
     GO TO 2450
2530 LET C(K)=C(S(SP-1))/C(S(SP)): LET
     S(SP)=0: LET SP=SP-1: LET S(SP)=K:
     GO TO 2450
2540 LET C(K)=C(S(SP-1))^C(S(SP)): LET
     S(SP)=0: LET SP=SP-1: LET S(SP)=K:
     GO TO 2450
2550 LET C(K)=0-C(S(SP)): LET S(SP)=K: GO
     TO 2450
3010 >DIM H(4):DIM V(7):DIM S(20):DIM
     S$(20,5): DIM E$(20,5):DIM G$(20,5):
     DIM C(20)
```

Nueva savia para un viejo lenguaje

El FORTRAN ha experimentado numerosos cambios desde que fuera introducido a fines de los años cincuenta

Son dos las principales críticas que se han formulado respecto al FORTRAN como lenguaje de programación moderno: su falta de estructuras de control adecuadas, que hace difícil implementar programas de forma que resulten fáciles de comprender y corregir, y las facilidades muy restringidas para manipulación de caracteres. Con el correr de los años se han probado numerosas formas de superar estas deficiencias.

Un enfoque relativamente reciente consiste en utilizar un "preprocesador", una especie de compilador de nivel más alto. Los programas escritos en una versión ampliada de FORTRAN se ejecutan entonces con el preprocesador, que cambia todas las configuraciones adicionales a equivalentes en FORTRAN estándar. Esto ofrece la ventaja de poder escribir programas de una manera correctamente estructurada manteniendo, al mismo tiempo, una versión estándar y, en consecuencia, compatible. Los mejores ejemplos conocidos de preprocesadores de FORTRAN son el WATFOR y el RATFOR.

Cuando salió el FORTRAN 77, incorporó nuevas características en forma similar a los populares preprocesadores. Por este motivo el lenguaje conserva cierto grado de coherencia, si bien muchos compiladores para micros se basan en el FORTRAN IV con ampliaciones, en vez de en el auténtico FORTRAN 77.

El problema de proporcionar estructuras de control adecuadas es que el FORTRAN no posee una estructura de bloques como el ALGOL, el PASCAL o el C, y no existe ninguna forma de producir una sentencia compuesta encerrando entre paréntesis varias sentencias (con un begin...end, p.ej.). La única forma de aislar un bloque es extraerlo como una subrutina separada o bien rodearlo de GOTOs. Se ha introducido una nueva estructura de control, si bien no encaja muy bien en el resto del lenguaje. Se trata de IF...THEN...ELSE...ENDIF:

```
IF (expresión logica) THEN
        .....
        sentencias
        .....
ELSE
        .....
        sentencias
        .....
ENDIF
```

Las expresiones lógicas se forman exactamente de la misma forma que en FORTRAN estándar y, como es habitual, la parte ELSE se puede omitir.

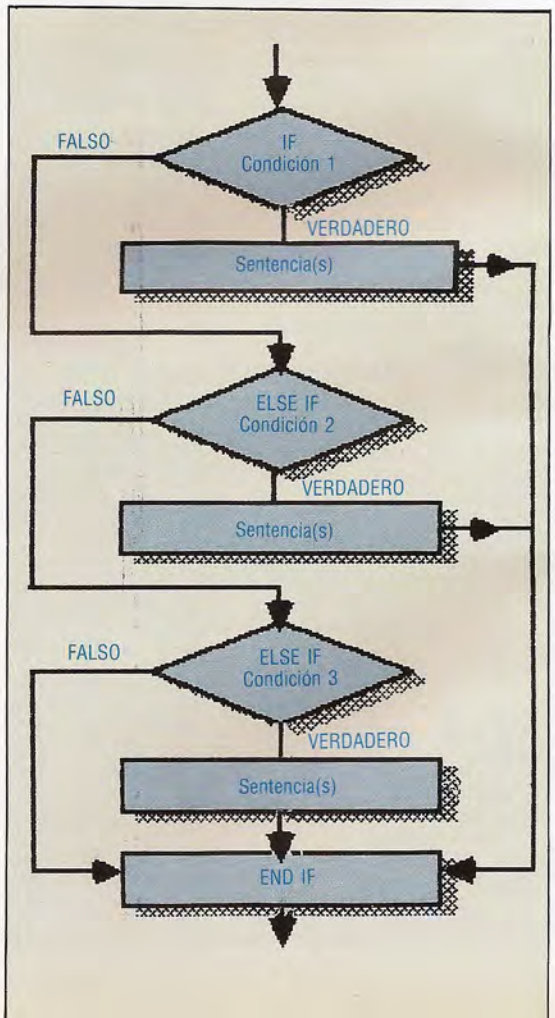
Nido de IFs
La mayoría de las versiones de BASIC admiten sentencias IF anidadas, pero el FORTRAN lo hace aún mejor al implementar la construcción ELSEIF, que permite anidar sentencias IF en cualquier profundidad a lo largo de varias líneas de programa. La construcción acaba con una sentencia ENDIF

Una forma muy útil de esta sentencia, que otros lenguajes harían bien en copiar, es una variante para abordar los IFs anidados:

```
IF (expresión logica) THEN
        .....
        sentencias
        .....
ELSEIF
        .....
        sentencias
        .....
ELSEIF
        .....
ELSE
        .....
ENDIF
```

en donde puede haber tantos ELSEIF como se requiera.

La segunda mejora importante ha sido la introducción de un tipo de datos de carácter. Las series de caracteres se declaran al principio del programa con las otras declaraciones en una de dos formas:



```
CHARACTER*9                    CH1,CH2
CHARACTER                      CH3*7,CH4*16
```

Observe que se ha de dar una longitud máxima para la serie como *n, donde n es un entero. Ésta se puede aplicar ya sea a la palabra clave CHARACTER propiamente dicha, si todas las series tienen la misma longitud, o bien individualmente a cada serie nombrada. Estas declaraciones dan dos series de longitud 9, una de longitud 7 y una de longitud 16.

Se pueden declarar matrices de series del modo normal:

```
CHARACTER *4 CHARS(50)
```

que declara una matriz de 50 series de cuatro caracteres. A las variables en serie se les pueden asignar en serie, como en:

```
CH1'ABCDEFGHI'
```

Si la serie asignada es demasiado corta, se añadirán espacios en blanco; si es demasiado larga, se truncará lo que sobre.

Las series se pueden comparar utilizando los operadores de relación normales, como .GT., .LT., etc., pero también hay dos operadores nuevos para efectuar funciones de series. El operador subserie da acceso a cualquier secuencia de caracteres de la serie, como en:

```
CH1 (3:5)
```

Esto da la subserie de CH1 que comienza en el tercer carácter y termina en el quinto, o sea que:

```
CH1='ABCDEFGHI'
CH2=CH1 (3:5)
```

Mike Clowes

Descifrando el código

```
C     PROGRAMA PARA DECODIFICAR
C     MENSAJES SECRETOS OBSERVE
C     LA PRESENCIA DE UNA
C     SENTENCIA 'PROGRAM' EN FORTRAN 77
      INTEGER CONTADOR, PTRIN,
      PTROUT ARCHIVO LOGICO
      CHARACTER*10 NUM,CAR
      CHARACTER*30 IN,OUT
      DATA CONTADOR /0/
C
C     OBSERVE SENTENCIA 'DATA' PARA
C     INICIALIZAR UNA VARIABLE
C
C     SI ARCHIVO SECRETO NO EXISTE
C     IMPRIMIR MENSAJE ERROR
      INQUIERE (ARCHIVO
      'SECRETO',EXIST=ARCHIVO)
      IF(.NOT.ARCHIVO)THEN
        WRITE(1,10)
        STOP
      ENDIF
C
C     ABRIR ARCHIVOS
C
      OPEN(UNIT=12,ARCHIVO'MENSAJE',
        ESTADO='NUEVO'
C
C     ESPECIFICAR CLAVE DECODIFICADORA
C
      NUM='0123456789'
      CAR='MW3 0D$%^T'
C
C     LEER Y DESCIFRAR MENSAJE SECRETO
C
100   READ(11,20,END=1000)IN
        OUT=''
        PTROUT=1
        DO 200 I=1,30
          PTRIN=INDEX(CAR,IN(I:I))
C
C     LA FUNCION ESTANDAR 'INDEX'
C     DETERMINA LA POSICION
C     DE LA SUBSERIE IN(I:I) EN SERIE CAR
        IF(PTRIN.NE.0)THEN
          OUT(PTROUT:PTROUT)=NUM
          (PTRIN:PTRIN)
          PTROUT=PTROUT+1
        ENDIF
200   CONTINUE
      WRITE(12,20)OUT
C
C     OBSERVE EL USO DEL MISMO FORMATO
C     PARA ENTRADA Y SALIDA
C
      CUENTA=CUENTA+1
      GOTO 100
C
C     CERRAR
C     SE UTILIZA'FINARCHIVO' PARA
C     ESCRIBIR FINAL DEL
C     MARCADOR DE ARCHIVO
1000  FINARCHIVO(UNIT=12)
      STOP
C
C     FORMATOS
10    FORMAT(1H,'EL ARCHIVO SECRETO NO
      EXISTE')
20    FORMAT(A)
C
C     OBSERVE QUE NO ES NECESARIO
C     ESPECIFICAR LA CANTIDAD
C     DE CARACTERES A ENTRAR
C
      END
```

Elemental, querido Watson
Este programa en FORTRAN lee datos desde un archivo denominado SECRETO y utiliza una clave de decodificación para descifrar texto codificado

Daría por resultado que CH2 contuviera 'CDE', o estrictamente 'CDE '. Observe que la segunda cifra dada es la posición del carácter final y no la longitud, lo que le resultaría más familiar al programador de BASIC.

El operador de concatenación, //, se utiliza para unir series entre sí, como en:

```
CHARACTER CH1*4,CH2*4,CH3*8
CH1='ABCD'
CH2='WXYZ'
CH3=CH1//CH2
```

que dejaría a CH3 conteniendo 'ABCDWXYZ'.

Archivos de datos

Un tema que aún nos queda por ver es el empleo de archivos de datos. Para sacar el máximo partido del FORTRAN se requiere un sistema con unidades de disco y la mayoría de los FORTRAN incluyen facilidades para el acceso a archivos tanto secuencial como directo. La verdadera fuente o destino de una operación de E/S se determina mediante el identificador de dispositivo en la sentencia READ o WRITE. Algunos números, por lo general los menores, se reservan para los dispositivos de E/S estándares tales como el teclado, la pantalla y la impresora; sin embargo, el programador dispondrá de una gama de números.

La sentencia OPEN se utiliza para asignarle un identificador a un archivo en disco.

Toma la forma:

```
OPEN(UNIT=expresion de enteros, FILE=
nombre del archivo, STATUS=estado)
```

El valor entero para la unidad (UNIT) es el número que se habrá de utilizar en subsiguientes sentencias READ o WRITE para acceder al archivo, y el nombre del archivo se atendrá a las convenciones del OS. El estado puede ser uno de varios valores; por ejemplo, OLD para un archivo de entrada que ya exista o NEW para un archivo de salida que esté en fase de creación.

En la sentencia OPEN hay varias cláusulas opcionales si se requiere algo que no sea un archivo secuencial directo. Éstas son :ACCESS=, para determinar acceso secuencial o directo; FORM=, para determinar si el archivo está formateado o sin formatear; IOSTAT=, que proporciona un medio de recuperación de errores si por algún motivo no se puede realizar la asignación de archivo correcta; y RECL=, para especificar una longitud de registro.

Una sentencia CLOSE similar podría ser:

```
CLOSE(UNIT=expresión de
enteros,STATUS=estado)
```

Ésta se puede utilizar para cerrar un archivo abierto, pero no siempre es necesario, puesto que cuando el programa termina los archivos se cierran automáticamente.

Además, existen otras muchas sentencias para manipulación de archivos que no hemos mencionado. Quizá la más útil de ellas sea INQUIRE, que le permite al programador determinar el estado actual y la actividad de cualquier archivo.

Hay, asimismo, numerosas adiciones útiles a las sentencias READ y WRITE, para usar cuando se efectúan operaciones de E/S con archivos.

Por ejemplo:

```
READ(7,10,REC=I)A,B,C
```

leería el primer registro de un archivo que se hubiera abierto para acceso directo.

Otro ejemplo:

```
READ(7,11,END=1000)A,B,C
```

haría que el control se transfiriera a la sentencia número 1000 cuando se llegara al final de un archivo secuencial.

Por último, veamos a los compiladores de FORTRAN propiamente dichos. Lo primero a destacar es que, por lo general, son rápidos y eficaces en operación y producen un código veloz, compacto y eficiente, lo que representa una ventaja fundamental, en especial para aplicaciones en tiempo real. Quizá esto sea lo que cabría esperar de un lenguaje que ha estado en servicio durante tanto tiempo y que es de un nivel lo bastante bajo para empezar con él.

La detección de errores no es, sin embargo, un punto fuerte de los compiladores de FORTRAN. Pueden aceptar bastante tranquilamente algunos errores garrafales que jamás pasarían, pongamos por caso, por un compilador de PASCAL. Los espacios, p. ej., no son significativos en una sentencia de FORTRAN, de modo que con frecuencia un compilador suprimirá todos los espacios de cada línea como primera tarea.

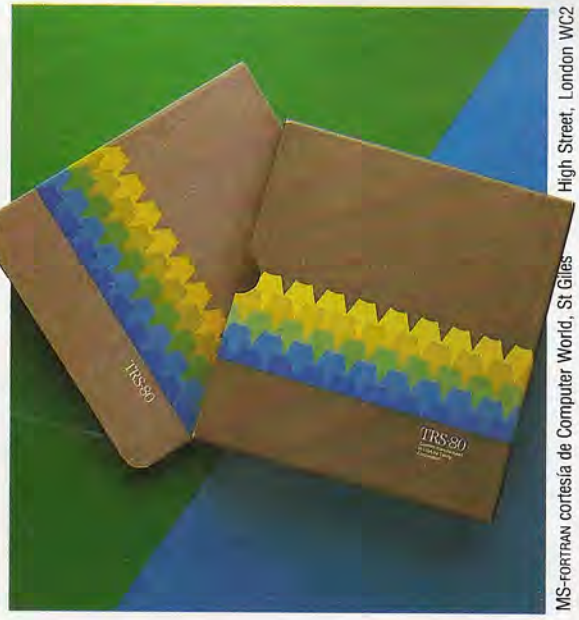
Un ejemplo de ello es la sentencia DO:

```
DO 100 I=1,100
```

Ésta establece un bucle a repetir 100 veces. No obstante, un punto en lugar de la coma, tras suprimir los espacios, lo deja a usted con:

```
DO100I=1.100
```

que es una asignación perfectamente legítima para una variable llamada DO100I. Se comenta que este simple error ha sido el origen de un desastre sumamente oneroso en el programa espacial norteamericano. El FORTRAN ha sido el lenguaje estándar para muchos trabajos de defensa, y errores como éste, al producirse en (entre otras cosas) sistemas de alerta nuclear, fueron factores fundamentales a la hora de decidir el desarrollo del ADA como lenguaje de recambio.



MS-FORTRAN cortesía de Computer World, St Giles High Street, London WC2

FORTRAN para micros
Los compiladores FORTRAN tienden a ser bastante caros y normalmente sólo los hay disponibles para máquinas de gestión; el paquete de la fotografía se ejecuta en el Tandy 2000. Además, el limitado espacio de memoria a menudo hace que la implementación del FORTRAN en micros sea difícil o incluso imposible.

No obstante, con la creciente disponibilidad de CP/M en ordenadores personales tales como Commodore 128, Amstrad CPC y otros, muchos usuarios encontrarán que existe una versión de FORTRAN para su máquina

Queridos residentes

Veamos cómo, gracias a las extensiones residentes de sistema, se pueden añadir nuevas instrucciones al BASIC Locomotive

En el corazón del OS del Amstrad encontramos el núcleo. Esta sección del firmware es la que se encarga del mantenimiento interno del ordenador, como el procesamiento de eventos e interrupciones y la gestión de la memoria.

Muchas entradas al núcleo se realizan a través de las direcciones que están entre &BCC8 y &BD10, pero el núcleo está provisto de su propio bloque, dividido en una sección superior y otra inferior, que se hallan en &B900-&B921 y &0000-&003B, respectivamente. Pero a diferencia del área del bloque de saltos del sistema central, estas posiciones no podrán ser parcheadas por el usuario. Algunas de las entradas son de un uso frecuente, y en la tabla *Direcciones útiles del núcleo* se da una breve descripción de las más interesantes.

Una característica muy poderosa del núcleo es su capacidad para procesar instrucciones externas. Estas instrucciones están en las ROM o en la RAM. En caso de que estén en la RAM, estas instrucciones son conocidas como *extensiones residentes del sistema* (RSX). Una instrucción externa puede ser una rutina de cualquier tamaño o función. Por ejemplo, instrucciones como |DIR y |ERA y otras que incluye el AMSDOS son rutinas que tratan los archivos del disco. Adicionalmente, todo el lenguaje del BASIC es considerado por el sistema como una RSX (pruebe a teclear |BASIC para ver qué pasa).

Las instrucciones externas pueden establecerse de dos maneras. Primera, en el momento de la puesta en marcha, en que se cargan las instrucciones desde la ROM. Segunda, bajo el control de programa, desde donde se pueden cargar estas instrucciones ya sea desde la ROM, ya sea desde la RAM. En ambos casos una instrucción externa está reseñada en una tabla de instrucciones, cuyo formato se muestra en el diagrama. Los nombres de instrucciones pueden tener hasta 16 caracteres, con el bit 7 en uno. Los caracteres pueden ser cualquier código de siete bits, pero si la instrucción ha de llamarse desde el BASIC habremos de cuidar de que los caracteres empleados sean accesibles desde el teclado.

Si las instrucciones están en forma de una RSX (es decir, cargadas en la RAM) entonces el proceso efectivo de *registro* (aviso al firmware de su presencia) se realiza mediante la rutina de núcleo KL_LOG_EXT. Esta rutina es llamada a través de &BCD1, con la dirección de la tabla de instrucciones en BC. Por su parte, HL debe contener la dirección de un espacio de trabajo de cuatro bytes que el núcleo pueda usar, y ambas direcciones deben encontrarse en los 32 K centrales de la RAM.

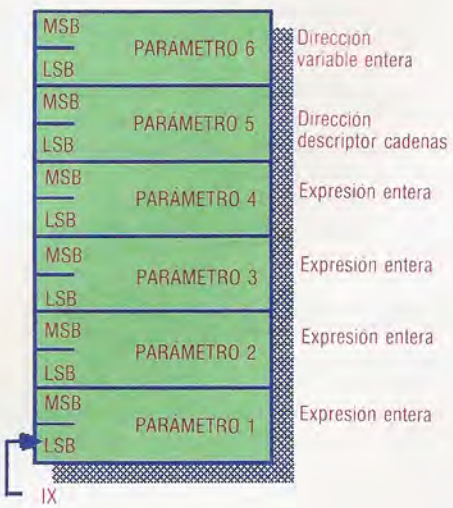
En los casos en que las instrucciones son almacenadas en la ROM, lo que se presenta al firmware es la ROM misma y no la tabla de instrucciones. La ROM debe también adecuarse a un formato particular, como se muestra en la figura tercera. Las ROM pueden ser designadas como ROM de primer plano, de fondo o de extensión. Se emplean ROM de primer plano para albergar programas que, cuando son entrados, toman el control del firmware. Por ejemplo, la ROM del BASIC es de este tipo, como lo sería la ROM de cualquier otro lenguaje, como el PASCAL.

Una ROM de fondo contiene las instrucciones externas que pueden estar o no registradas por el programa de primer plano. Finalmente, una ROM de extensión sirve para albergar el código para un programa de primer plano o una rutina de instrucción externa que pueda no ajustarse a la ROM existente.

Cada ROM se direcciona en el intervalo 0-251, aunque cada tipo tiene sus condiciones restrictivas particulares. Las ROM de primer plano pueden ocupar una dirección cualquiera, siempre que todas las direcciones que quedan por debajo de la escogida sean ocupadas por otras ROM del tipo que sea, dado que el núcleo busca las ROM ascendiendo desde la dirección 0 hasta que encuentra el primer espacio vacío. Las ROM de fondo deben estar en las direcciones 1-7 en el 464, y 1-15 en el 664 y 6128. Las ROM de extensión pueden ocupar cualquier dirección.

Por ejemplo, si el sistema tiene ya el FORTH en la ROM 0 (por defecto) y la ROM de fondo en la ROM 1, entonces una segunda ROM de primer plano deberá instalarse en la dirección 2 de ROM

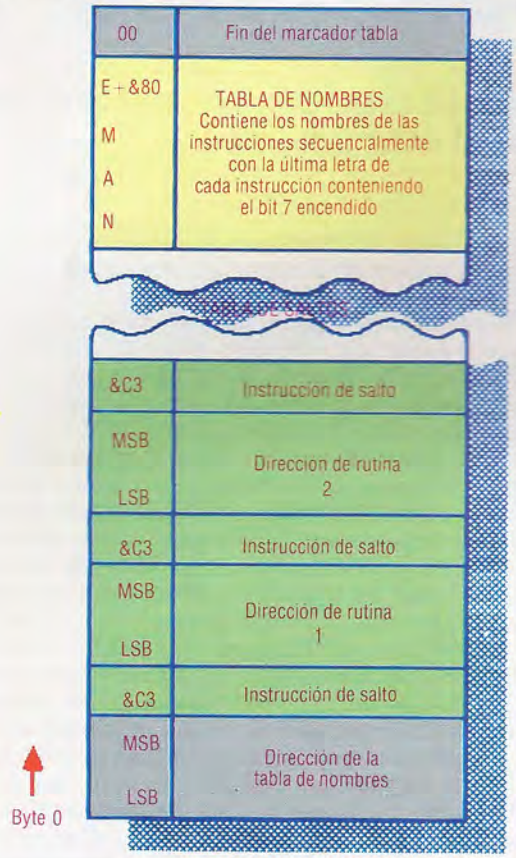
Tabla de parámetros



Proceso de entrada
A la entrada en una rutina de instrucción llamada desde el BASIC, todo parámetro pasado por el usuario es almacenado en un bloque de parámetros. La dirección de base del bloque queda apuntada por IX y un registro A contiene el número de parámetros que se ha pasado. Esta figura muestra la estructura del bloque establecida después de ejecutar | COMMAND, 5,24.6, a%,x,@b$,@x%

Caroline Clayton

Instrucciones externas



Establecimiento de la tabla
Las instrucciones externas son referenciadas a través de una tabla de instrucciones. Una tabla puede contener datos para varias instrucciones diferentes agrupadas en un todo y el formato de dicha tabla es el que mostramos aquí

para asegurar la continuidad. Sin embargo, podría introducirse una segunda ROM de fondo en cualquier dirección entre 3 y 7 (15 en los 664/6128) e instalarse una extensión a cualquier dirección.

Todas las ROM de fondo son registradas al poner en funcionamiento la máquina. Las ROM de fondo pueden ser inicializadas bien individualmente mediante KL __INIT__BACK, bien conjuntamente mediante KL __ROM__WALK. La ROM del BASIC inicializa, por otro lado, todas las ROM de fondo cuando es introducida.

Cualquier ROM de extensión ocupa la misma área de memoria que la ROM del BASIC (de &C000 a &FFFF).

Antes de que se pueda entrar una ROM, primero se debe desactivar la ROM del BASIC y activar la ROM adecuada. El núcleo proporciona varias rutinas para conectar y desconectar ambas ROM, la superior y la inferior, lo cual exime al programador de pensar en conectarlas o desconectarlas directamente.

Posición de las instrucciones

Cuando las instrucciones externas han sido registradas se puede acceder directamente a ellas desde el BASIC, como veremos más adelante, o bien a través del núcleo mediante KL__FIND__COMMAND. Esta entrada toma el nombre de una instrucción y busca las RSX y ROM de fondo para su acoplamiento. Si se encuentra la instrucción, entonces la dirección de la entrada de la rutina se da en HL, conteniendo el registro C a su vez el número de la ROM de fondo (que puede ser ignorado si la rutina de instrucción está en la RAM). Si no se encuentra ninguna instrucción, la rutina retorna con el arrastre falso. Esta rutina es llamada en &BCD4, con HL conteniendo la dirección del nombre de la instrucción, que ha de acabar con un carácter cuyo bit 7 esté a uno.

Las instrucciones externas pueden ser entradas simplemente desde el BASIC antecediendo el nombre de la instrucción con el símbolo | (@ con *shift*). El BASIC permite también pasar parámetros a la rutina separándolos con comas, por Ejemplo:

```
|INSTRUCCION, 1,2,3,x,y*3,@$,@X%
```

El firmware pasa entonces el control a la rutina de instrucción, con el número de parámetros retenidos en el registro A e IX apuntando a la tabla que contiene la información de dichos parámetros. Esta tabla contiene dos bytes por cada parámetro, almacenados en el formato siguiente:

Números enteros/ expresiones	– enteros con signo de 16 bits
Números reales	– enteros sin signo puestos a 16 bits

Las direcciones de variables numéricas o de cadenas pueden ser pasadas también empleando el símbolo @, dejando las variables enteras una dirección de dos bytes donde se almacenan, y las variables de cadena la dirección del bloque descriptor de cadenas. Por desgracia, no es posible determinar si la entrada en la tabla es la dirección de un descriptor de cadenas o de un entero absoluto: es cosa de la rutina de instrucción decidir el tipo de parámetro que espera.

A la entrada a la rutina de instrucción, IX apunta al LSB (bit menos significativo) del *último* parámetro especificado (en el ejemplo anterior, sería la dirección de X%). El nuevo incremento de IX indexará, por tanto, el LSB del penúltimo parámetro pasado. El diagrama (p. 2137) ilustra cómo se indexa la tabla en el ejemplo dado.

A menudo es preferible mirar al comienzo el primer parámetro especificado y trabajar a través de ellos según van entrando, antes que hacerlo en orden inverso. El primer listado proporciona una breve utilidad que modifica IX para que apunte al

Direc. útiles del núcleo

&BCC8 RESET	Restaura el núcleo: limpia evento, colas, etc.
&BCCB ROM WALK	Inicializa todas las ROM de fondo
&BCD1 LOG EXT	Registra una tabla de instrucciones RSX
&BCD4 FIND COMMAND	Posiciona la dir. de RSX, ROM fondo y primer plano
&B90F ROM SELECT	Selecciona una determinada ROM superior
&B912 CURR SELECTION	Comprueba qué ROM sup. es la seleccionada

LSB del primer parámetro, en vez del último. Obsérvese que en este caso es necesario, para mirar al MSB (bit más significativo) del primer parámetro, incrementar una vez el IX, mientras que para mirar al MSB del segundo parámetro será decrementado una vez. De nuevo habrá de decrementarse para mirar al LSB del segundo parámetro, y así sucesivamente.

Las instrucciones RSX son relativamente sencillas para que las emplee un programador en lenguaje máquina, y presentan la gran ventaja de que son fáciles para el usuario comparadas con la necesidad, en otras máquinas, de emplear instrucciones como CALL o SYS. Por esta razón, la instrucción CALL en los ordenadores Amstrad CPC adquiere menor relevancia, y el esfuerzo suplementario que se requiere para implementar las extensiones residentes de sistema compensa con creces. Como ejemplo, damos aquí un listado para una RSX que permite llamar las rutinas del firmware mencionadas en estos capítulos directamente desde el BASIC, con los registros establecidos según se requieren.

ROM de ampliación

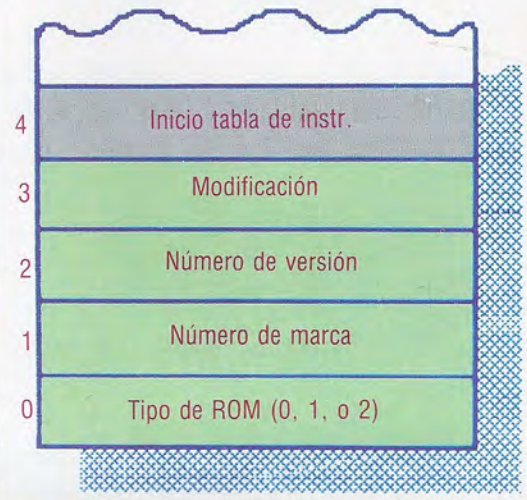


Tabla de la ROM
La tabla muestra un esquema de los bytes empleados por el sistema operativo para reconocer y "registrar" las ROM individuales de ampliación. Una interrogación posterior del sistema operativo posibilitará el retorno de las direcciones para las rutinas de instrucción definidas en la tabla de instrucciones externas de la ROM

Pasar el control

Puntero de parámetros

Esta breve rutina altera IX al entrar a una rutina de instrucción externa para que apunte al primer parámetro especificado

```
                ;entry:   A contiene el número de parámetros
                ;         IX contiene la dirección de la tabla
                ;exit:    IX contiene          la dirección
                ;         de la 1.ª            entrada en tabla
                ;         DE se altera,        los restantes regs.
                ;         se conservan
                ;
5F              ld   e,a                ;copia núm. para E
1D              dec  e                  ;num. de pars.-1
CB23            sla  e                  ; *2
1600            ld   d,0                ;toma desp. en DE
DD19            add  ix,de              ;lo suma para inic.
                ; IX apunta ahora al LSB de la 1.ª entrada
                ; en la tabla de parámetros
```

Llamada del firmware

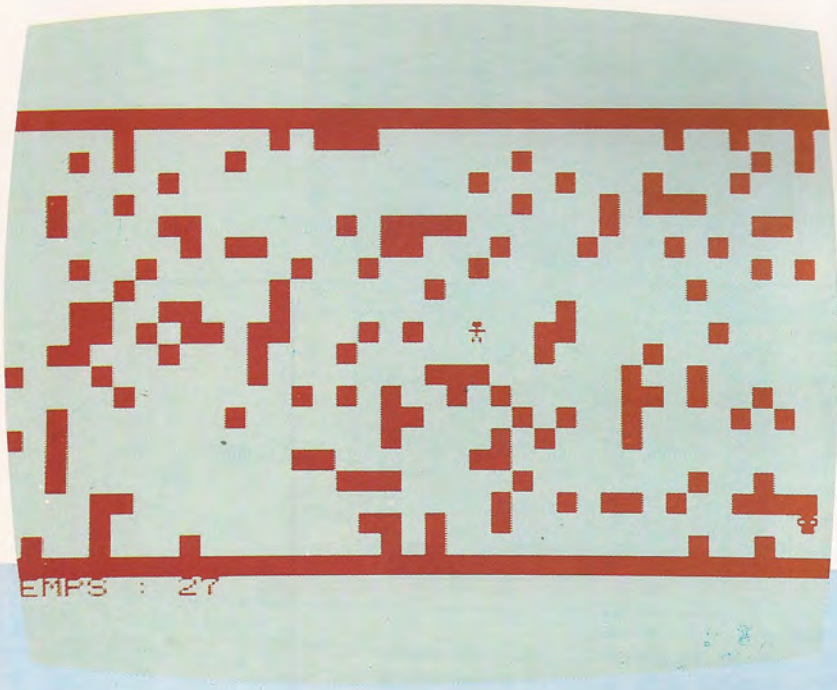
Este listado proporciona una RSX que permite la llamada del firmware directamente desde el BASIC, con los registros establecidos a voluntad. La sintaxis de la instrucción es la siguiente:

|FIRMCALL,<direccion entrada>,
[,<A> <HL>[,<BC>[,DE]]]]

```
                k1_log:   equ  #bcd1
01902C          logon:    ld   bc,commands      ;apunta a tabla
219E2C                    ld   hl,scratch       ;y cuad. apuntes
CDD1BC                    call kl_log           ;registra instr.
C9                        ret                   ;y eso es todo
952C            comman:   defw name             ;apunta al nombre
C3A22C                    jp   firmcall         ;punto de entrada
4649524D        name:     defm "FIRMCAL"        ;pone a 1 el bit 7
                                                del...
CC                        defb "L"+#80          ;...ult. byte del n.
00                        defb 0
                scratc:   defs 4                ;res. area datos
                ; IFIRMCALL,     entry, a, hl, bc, de
5F              firmca:   ld   e,a              ;copia numero
                                                para E
1D                        dec  e                ;num. paramtr-1
CB23                      sla  e                ;*2
1600                      ld   d,0              ;toma desp. en DE
DD19                      add  ix,de            ;lo añade a inicio
DDE5                      push ix
E1                        pop  hl
FE06                      cp   6                ;hasta 5 pars.
D0                        ret  nc               ;muchos, no seguir
B7                        or   a                ;hay parametros?
C8                        ret  z                ;no, pues no
                                                seguir
47                        ld   b,a              ;guardar cont. en B
23                        inc  hl               ;toma MSB de dir.
7E                        ld   a,(hl)           ;lo lee
32E62C                    ld   (entry+1),a      ;lo guarda
2B                        dec  hl               ;LSB
7E                        ld   a,(hl)
32E52C                    ld   (entry),a
2B                        dec  hl               ;MSB de AF
2B                        dec  hl               ;LSB de AF
05                        dec  b                ;bastante?
2822                      jr   z,call           ;si, realizar llam.
7E                        ld   a,(hl)           ;si no, leer en A
05                        dec  b                ;bastante?
281E                      jr   z,call           ;si, realizar llam.
2B                        dec  hl               ;MSB de HL
56                        ld   d,(hl)
2B                        dec  hl
5E                        ld   e,(hl)
EB                        ex   de,hl            ;toma en HL
05                        dec  b                ;bastante?
2816                      jr   z,call           ;si, realizar llam.
EB                        ex   de,hl            ;retomar punt.
2B                        dec  hl               ;MSB de BC
F5                        push af               ;guarda A de mom.
7E                        ld   a,(hl)
2B                        dec  hl               ;LSB de BC
4E                        ld   c,(hl)
05                        dec  b                ;bastante
47                        ld   b,a              ;establece BC en
                                                todo caso
EB                        ex   de,hl            ;y HL
2003                      jr   nz,miss          ;toma el ultimo
F1                        pop  af               ;restaura la pila
1808                      jr   call             ;y hace la llamada
F1              miss:     pop  af
EB                        ex   de,hl            ;retoma la tabla
D5                        push de               ;guarda valor HL
2B                        dec  hl               ;MSB de DE
56                        ld   d,(hl)
2B                        dec  hl               ;LSB de DE
5E                        ld   e,(hl)
E1                        pop  hl               ;retoma HL
C3              call:     defb #c3              ;instr. de salto
0000            entry:    defw 0                ;direccion de salto
                ;se ha parcheado la dirección del salto
                ;a la rutina
```

Persecución

El ladrón, representado por un as de piques, ha escapado llevándose el botín... Se trata de un tema muy visitado por la informática lúdica. El listado es para los micros MSX

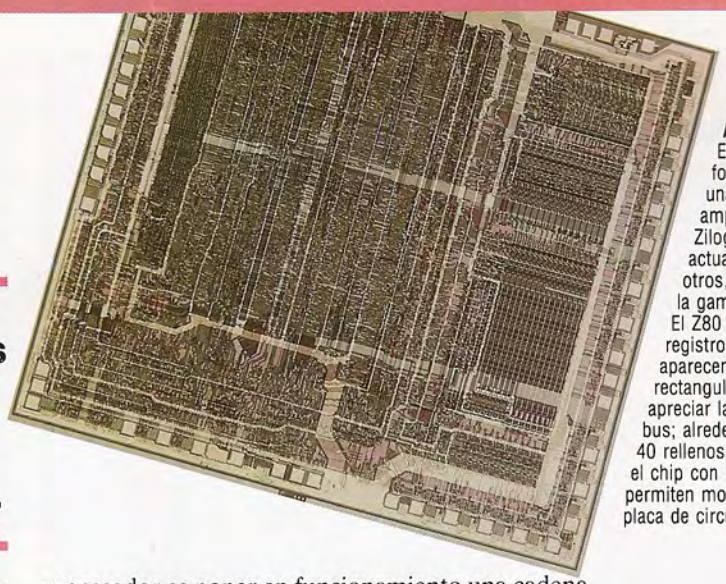


El amigo de lo ajeno se esconde en la ciudad, y usted tiene 50 minutos para encontrarlo y detenerlo. Atención, ¡no se precipite! En efecto, si se echa sobre él sin pensar, el fugitivo tiene todas las posibilidades de escapársele. La mejor manera de atraparlo es alcanzándolo de lado. (Es el método más efectivo a condición de no fallar.) Si no se siente lo bastante seguro de sí mismo, atáquelo de frente, lo cual es más fácil pero mucho menos eficaz, ya que no es tan discreto. Otro consejo: no intente perseguirlo; no daría resultado, pues él es mucho más rápido que usted. Ha de observar sus movimientos como si fuera un detective. Cuando vea que da la vuelta, acérquese sigilosamente y sorpréndalo en el momento justo. Pero recuerde: ¡el tiempo va pasando!

```
10 REM **************
20 REM * PERSECUCION *
30 REM **************
40 KEY OFF
50 WIDTH 39
60 GOSUB 1060
70 S=0
80 N=32
90 V=128
100 P=129
110 GOSUB 670
120 ST=STICK (0)
130 DX=(ST=7)-(ST=3)
140 DY=(ST=1)-(ST=5)
150 Z=Z-.2
160 LOCATE 0,24,0
170 PRINT "TIEMPO :";INT (Z+1);
180 IF Z<0 THEN 370
190 PX=PX+DX
200 PY=PY+DY
210 C=VPEEK (PX+PY*40)
220 IF C=V THEN 1010
230 IF C<>32 THEN PX=XP:PY=YP
240 VPOKE XP+YP*40,N
250 VPOKE PX+PY*40,P
260 YP=PY
270 XP=PX
280 VX=VX+CX
290 VY=VY+CY
300 IF VPEEK (VX+VY*40)<>N THEN GOSUB 510
310 IF VPEEK (VX+VY*40)<>N THEN 280
320 VPOKE XV+YV*40,N
330 VPOKE VX+VY*40,V
340 XV=VX
350 YV=VY
360 GOTO 120
370 IF INKEY$<>"" THEN 370
380 FOR I=1 TO 1000
390 NEXT I
400 LOCATE 10,6
410 PRINT "TIEMPO TRANSCURRIDO";
420 LOCATE 10,10
430 PRINT "PUNTOS :";S;
440 LOCATE 10,18,1
450 PRINT "OTRA ?";
460 D$=INKEY$
470 IF D$="" THEN 460
480 IF D$<>"N" AND D$<>"n" THEN RUN
490 CLS
500 END
510 DT=DT+1
520 GOSUB 620
530 IF VPEEK(XV+CX+(YV+CY)*40)=N THEN VX
    =XV+CX:VY=YV+CY:RETURN
540 DT=DT-2
550 GOSUB 620
560 IF VPEEK (XV+CX+(YV+CY)*40)=N THEN VX
    =XV+CX:VY=YV+CY:RETURN
570 DT=DT-1
580 GOSUB 620
590 VX=XV+CX
600 VY=YV+CY
610 RETURN
620 IF DT> 4 THEN DT=DT-4
630 IF DT< 1 HEN DT=DT+4
640 CX=(DT=1)-(DT=3)
650 CY=(DT=2)-(DT=4)
660 RETURN
670 CLS
680 FOR VX=0 TO 39
690 VPOKE VX+40,219
700 VPOKE VX+880,219
710 NEXT VX
720 FOR VY=2 TO 22
730 VPOKE VY*40,219
740 VPOKE VY*40+39,219
750 NEXT VY
760 FOR VX=1 TO 150
770 GOSUB 970
780 VPOKE PX+PY*40,219
790 NEXT VX
800 GOSUB 970
810 VX=PX
820 VY=PY
830 VPOKE VX+VY*40,V
840 XV=VX
850 YV=VY
860 GOSUB 970
870 VPOKE PX+PY*40,P
880 XP=PX
890 YP=PY
900 Z=50
910 CX=0
920 CY=0
930 DX=0
940 DY=0
950 DT=0
960 RETURN
970 PX=INT (RND(1)*37)+1
980 PY=INT (RND(1)*21)+2
990 IF VPEEK (PX+PY*40)<>N THEN 970
1000 RETURN
1010 FOR I=1 TO 5
1020 PLAY "DC"
1030 NEXT I
1040 S=S+1
1050 GOTO 110
1060 SCREEN 0,0
1070 COLOR 6,14
1080 CLS
1090 DEFINT A-Y
1100 PX=RND(-TIME)
1110 RESTORE
1120 FOR I=0 TO 15
1130 READ A
1140 VPOKE 3072+I,A*4
1150 NEXT I
1160 RETURN
1170 DATA 30,63,45,63,30,30,30,0
1180 DATA 28,28,8,62,8,8,20,20
```

Recorrer el circuito

Iniciamos una serie dedicada a analizar los componentes de las placas de circuito impreso de diversos micros. En primer lugar, centraremos nuestra atención en el microprocesador



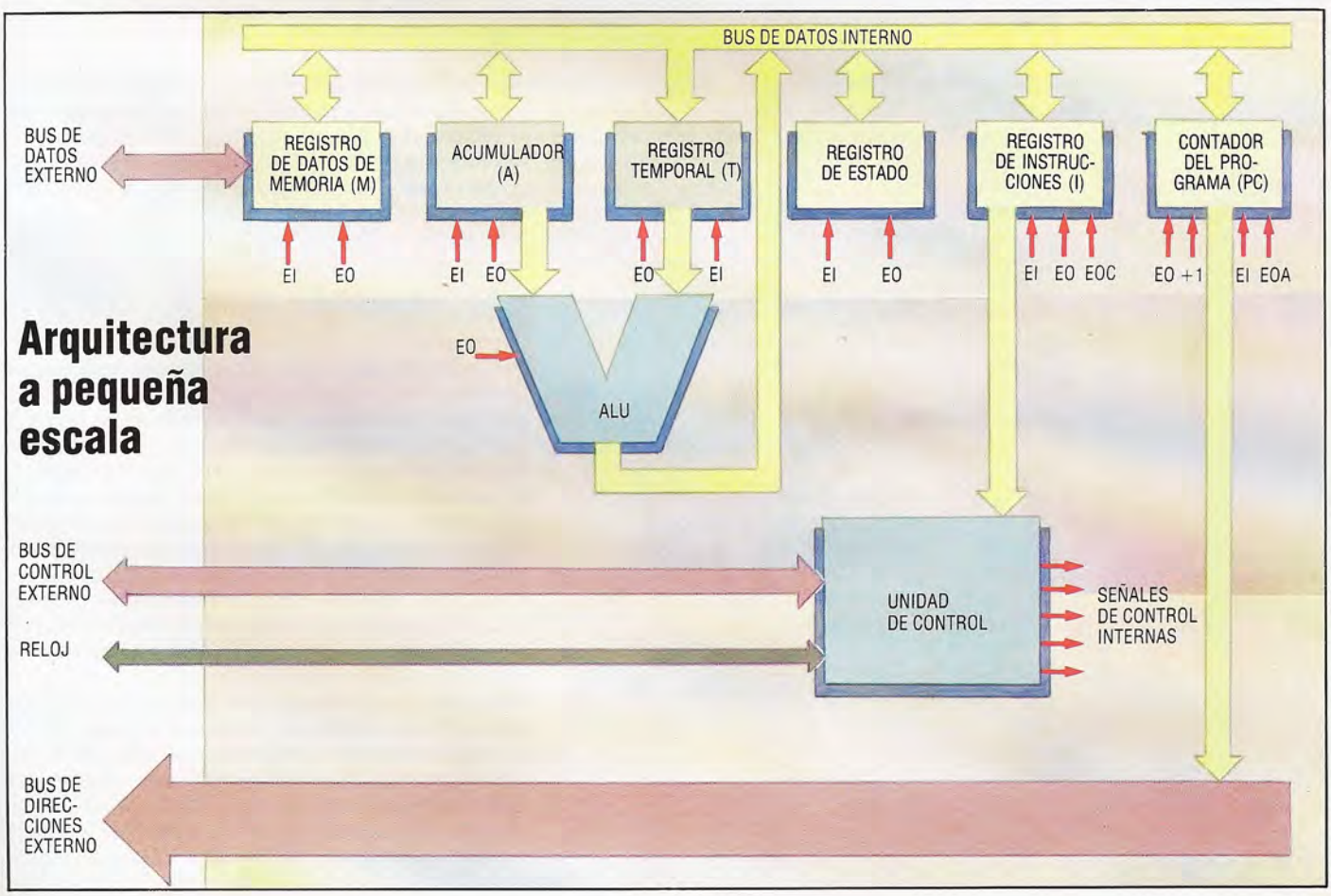
Ampliación
Esta fotomicrografía es una imagen muy ampliada de un chip Zilog Z80, utilizado actualmente, entre otros, en el Spectrum y la gama Amstrad CPC. El Z80 posee más de 25 registros en el chip, que aparecen como formatos rectangulares. Se pueden apreciar las conexiones del bus; alrededor del perímetro, 40 rellenos de conexión unen el chip con las patillas que permiten montarlo en una placa de circuito impreso

Mirando una fotomicrografía de un microprocesador típico percibimos cuadrículas geométricas, que son los registros internos del procesador, varias pistas rectangulares alrededor de la parte exterior, donde se realizan las conexiones externas, y zonas de sistemas de circuitos lógicos entremezclados.

También es posible discernir grupos de líneas que son los enlaces de comunicación interna entre los diversos componentes del chip. Es posible apreciar que la arquitectura de un procesador está determinada por las restricciones de un trazado bidimensional y por la cantidad física de componentes que se pueden incluir en un chip.

Lo que en realidad hacen las instrucciones del procesador es poner en funcionamiento una cadena de "microeventos" electrónicos, y éste es el punto en donde se traza la línea divisoria entre software y hardware. Vamos a ilustrar esto tomando el ejemplo de una instrucción sencilla: sumar un número al valor del acumulador y volver a colocar el resultado en el acumulador. El diagrama (abajo) muestra un ejemplo clásico de arquitectura de procesador de una forma simplificada. Todos los registros internos están unidos mediante un bus de datos interno y cada registro tiene conectadas líneas de control de modo que todas las operaciones de transferencia de datos entre registros y el bus de datos están sincronizadas. Estas líneas de control se unen en una uni-

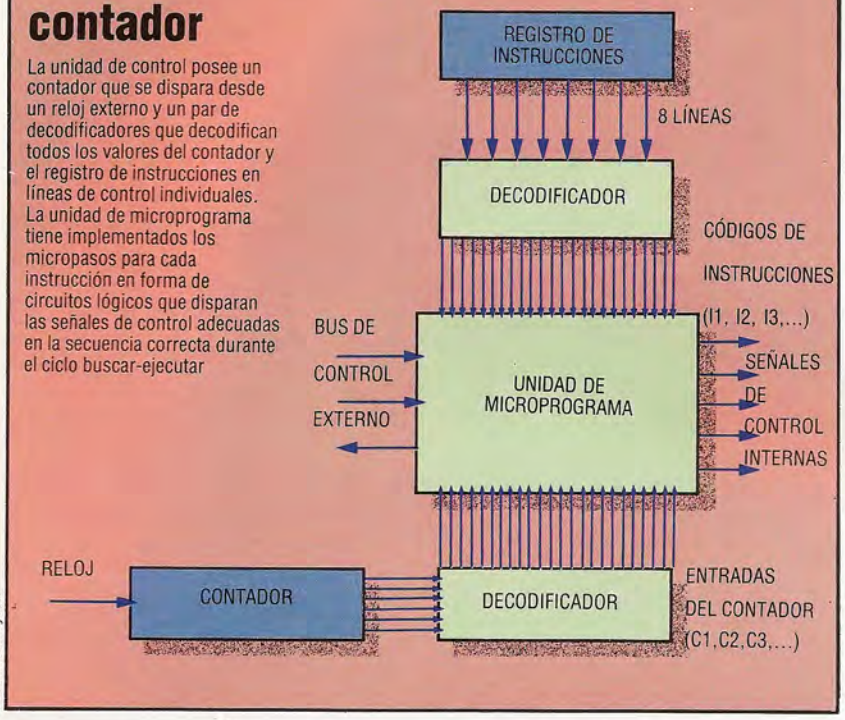
El procesador
El diagrama muestra la arquitectura típica de un procesador de ocho bits. Los componentes en el chip se comunican a través de un bus de datos interno y señales de control internas, pero se mantienen en contacto con la memoria y otros dispositivos mediante señales de control y buses de datos y direcciones externos



Paul Bryant

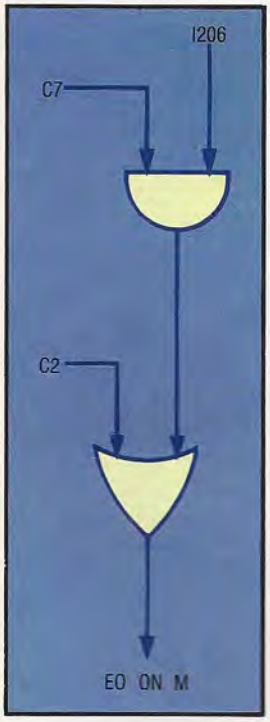
Por el contador

La unidad de control posee un contador que se dispara desde un reloj externo y un par de decodificadores que decodifican todos los valores del contador y el registro de instrucciones en líneas de control individuales. La unidad de microprograma tiene implementados los micropasos para cada instrucción en forma de circuitos lógicos que disparan las señales de control adecuadas en la secuencia correcta durante el ciclo buscar-ejecutar



Caroline Clayton

Circuito de micropasos
Para nuestro ejemplo de instrucción ADD, es necesario establecer, en el paso 2 y el paso 7 del ciclo buscar-ejecutar, una señal de control determinada, la habilitación de salida en el registro de datos de memoria. Este circuito lógico simple combina la señal 206 del decodificador de instrucciones con las dos señales de contador decodificadas adecuadas para disparar "EO on M"



dad de control central, de la que luego hablaremos con mayor detalle.

La instrucción ADD (suma) inmediata que mencionábamos anteriormente cobra vida en forma de un par de bytes en un programa más grande almacenado en la memoria del ordenador. El primer byte contiene la instrucción y el segundo, el número a sumar. Para obedecer esta instrucción, el procesador primero debe traer de la memoria el byte de instrucción y después ejecutarlo: el ciclo buscar-ejecutar. El procesador sabe dónde obtener la instrucción, dado que su dirección estará retenida en el contador del programa (PC: *program counter*). Al describir el ciclo buscar-ejecutar, nos interesa particularmente la parte relativa a cómo gestiona la unidad de control las diversas transferencias entre registros. Esto se realiza enviándole señales *enable* (de habilitación) al registro correcto en el momento adecuado. El sistema que hemos utilizado para diferenciar estas señales es el siguiente:

EI: habilitar entrada a registro desde bus de datos interno
EO: habilitar salida desde un registro hacia el bus de datos
EOC: habilitar salida a la unidad de control
EOA: habilitar salida al bus de direcciones

El ciclo "buscar" comprende estos pasos:

Paso	Señales control	Acción
1	EOA on PC	Copiar el cont. del PC en el bus de direcciones
2	+1 on PC	Incrementar el PC
3	EO on M EI on I	Copiar el contenido del registro de datos de memoria en el registro de instrucciones
4	EOC on I	Copiar el contenido del registro de instrucciones en la unidad de control

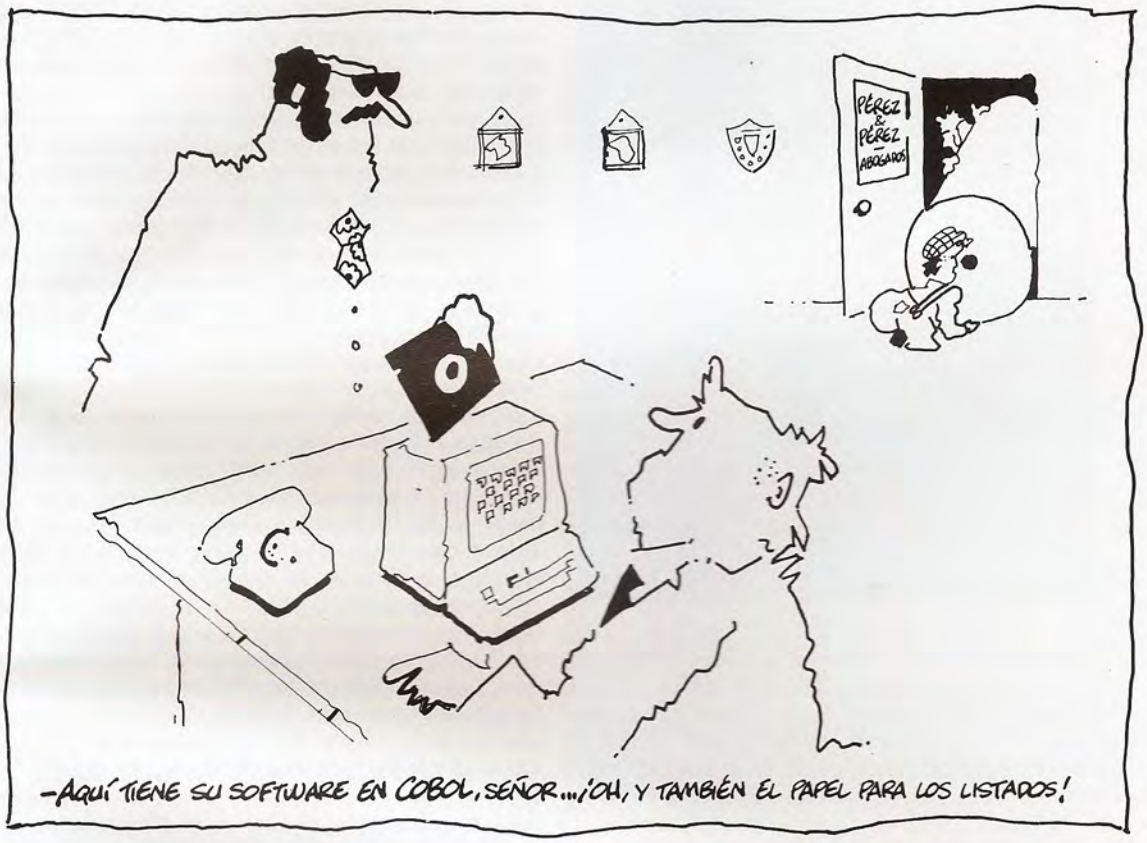
Al concluir estos cuatro pasos, en la unidad de control hay la instrucción ADD, y el contador del programa apunta al byte que contiene el número a sumar. Antes de analizar la forma en que se obedece la instrucción, veamos primero la *unidad de control*. Consta de un registro en el cual se coloca la instrucción actual, un decodificador que decodifica los 8 bits de que consta el byte de instrucción en 256 líneas separadas, una para cada posible valor que puede retener el byte. (Si no se utiliza la totalidad de los 256 posibles códigos para códigos de instrucción, la cantidad de líneas decodificadas puede ser menor.) Estas líneas decodificadas se amplian a la unidad de microprograma, junto con la salida decodificada desde un contador. En la unidad de microprograma se halla la verdadera función de cada código de instrucción, en forma de circuito lógico, y las salidas desde estos circuitos lógicos forman las diversas señales de habilitación de registros. Veamos ahora la parte "ejecutar" del ciclo:

Paso	Señales control	Acción
5	EOA on PC EI on M	Traer el número a sumar a M
6	+1 on PC	Incrementar el PC listo para la sig. instrucción
7	EO on M EI on T	Copiar el núm. en reg. temporal ALU
8	selec. función ADD EO on T EO on A	Efectuar la suma
9	EO on ALU EI on A	Copiar el resultado en el acumulador

Mientras que la parte "buscar" del ciclo es la misma en todos los casos, la parte "ejecutar" está diseñada a medida para llevar a cabo una determinada tarea: en este caso, para tomar el siguiente byte del programa y sumárselo al valor en el acumulador. Cada código de instrucción posee su propio microprograma, implementado como una serie de puertas lógicas. Si observamos una señal de habilitación determinada (la que habilita la salida desde el registro de datos de memoria [EO on M]), podemos ver cómo se implementa la anterior secuencia "ejecutar". Es necesario establecer EO on M en el paso 3 y el paso 7 del ciclo buscar-ejecutar. Supongamos que el código para nuestra instrucción ADD inmediata es 11001110, 206 en decimal. Simplemente aplicando impulsos de apertura a la línea C7 del contador de pasos con la línea 206 desde el decodificador de instrucciones y relacionando el resultado con C3 mediante una función OR, obtenemos una señal de habilitación de salida para el registro de datos de memoria. Por supuesto, muchas instrucciones diferentes podrían necesitar enviar señales de habilitación de salida a este registro, pero es tan sólo cuestión de relacionarlas entre sí mediante una OR.

Los fabricantes de procesadores utilizan máquinas especiales, denominadas ordenadores microprogramables, para programar los micropasos de una instrucción desde software. Éstas a menudo se utilizan para diseñar y desarrollar procesadores nuevos, en los que se puede desarrollar el conjunto de instrucciones antes de confiarlo al hardware en la unidad de control.

Líder comercial

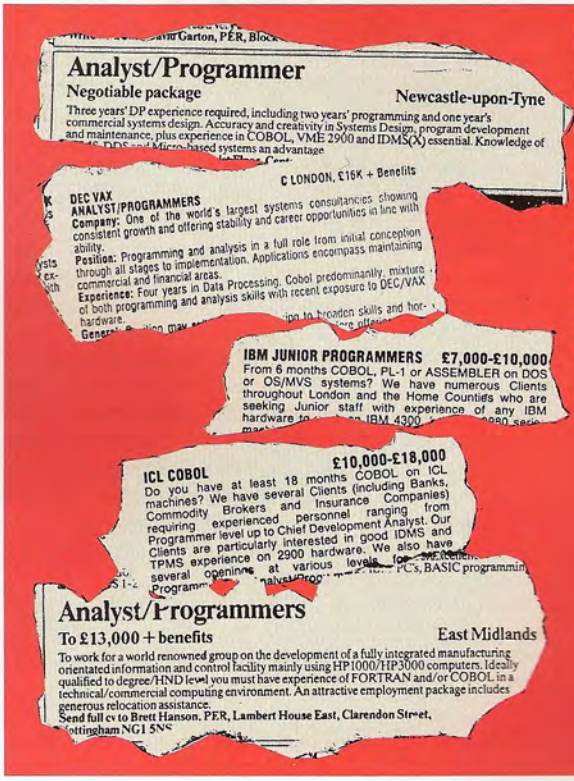


Mike Clowes

El COBOL es el lenguaje informático de uso más generalizado en las aplicaciones comerciales y de gestión

El COBOL (*Common Business Oriented Language*) es con mucho el lenguaje de programación más ambpliamente utilizado en todo el mundo, y si bien el BASIC tal vez sea el más conocido, es probable que las verdaderas líneas de código escrita sen COBOL superen a la de todos los otros lenguajes juntos. Esto no se debe sólo a que el COBOL sea prolijo, necesitando de muchas líneas aun para un programa simple, sino a que ha sido el principal lenguaje (casi el único) para uso comercial y de gestión.

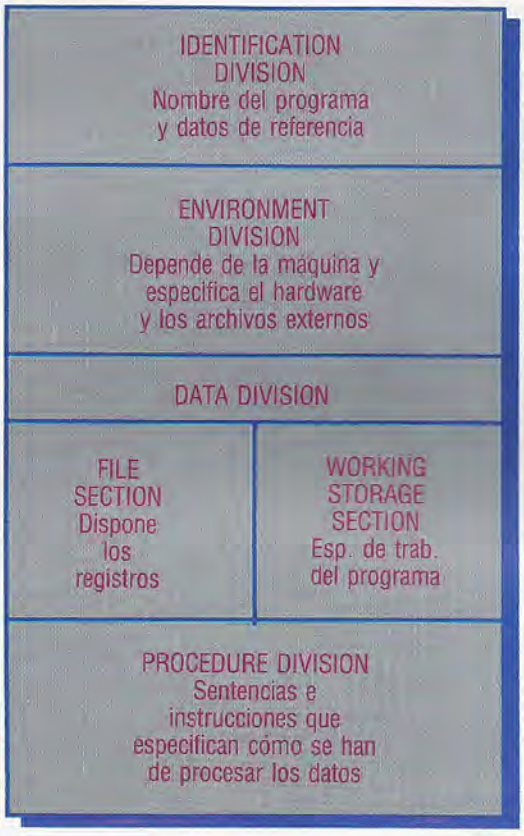
El COBOL es un lenguaje muy estandarizado. De hecho, existe un cuerpo permanente, el comité CODASYL, que supervisa su uso y desarrollo. Este elevado nivel de normalización es importante por varios motivos. Es vital que una organización comercial sea capaz de transferir su software, lo que fácilmente podría ser uno de sus más valiosos bienes, de un ordenador a otro sin ningún cambio sustancial, o mejor aún, sin tener que introducir ningún cambio



Profesión: programador

A pesar de su antigüedad, tanto el COBOL como el FORTRAN siguen siendo ampliamente utilizados por las empresas de todo el mundo. Con frecuencia se imparten cursos sobre estos y otros lenguajes con carácter local, que pueden ser un medio para alcanzar un empleo seguro y disfrutar de buenos salarios

Estructura dividida
Los programadores familiarizados sólo con el BASIC es posible que opinen que la estructura de un programa en COBOL es un tanto extraña. No obstante, este lenguaje cumple a la perfección los requerimientos exigibles en cuanto a legibilidad, adaptabilidad y facilidad de revisión y actualización



Caroline Clayton

en absoluto. Muchas empresas que operan con un gran ordenador central no desearán perder un valioso tiempo de esa máquina y esperarán que el desarrollo de programas se realice en una máquina completamente diferente, con lo que, nuevamente, resulta vital que los programas se puedan transferir con facilidad.

Otro factor es que los programas comerciales son bastante distintos de los programas en BASIC que hemos escrito en nuestros micros personales. Son muy largos, constando a menudo de decenas de miles de líneas de código, y se tienden a escribir en equipo en vez de por un único programador, situación que no es ideal para producir un código libre de errores. Se espera que los programas tengan una vida útil de varios años, durante los cuales se eliminarán los errores y se efectuarán correcciones, ampliaciones y mejoras. Todas estas revisiones las llevarán a cabo uno o más programadores, quienes, muy probablemente, no serán los mismos que escribieron el programa original.

Estas restricciones (y el hecho de que los programas se escriban de acuerdo a unas especificaciones rígidas) pueden hacer de la programación en COBOL una de las tareas más tediosas que existen. No obstante, existe muchísimo interés en el lenguaje en sí mismo y no hay ninguno mejor que él para tareas que requieran tratamiento de archivos complejos.

El aspecto y el funcionamiento de un programa en COBOL es muy diferente al de otros lenguajes, si bien encubiertos por su prolijidad podemos encontrar algunos de los mismos conceptos que subyacen en todos los lenguajes procedurales. Los programas constan de cuatro divisiones:

Identification division (división de identificación): es básicamente de índole documental, y proporciona un espacio para el nombre de un programa, el nombre del programador, fechas de escritura y compilación y comentarios introductorios.

Environment division (división de entorno): está concebida como la parte dependiente de la máquina y contiene detalles relativos al ordenador con el cual se escribió y ejecutó el programa. Muchos de los detalles que se incluyen aquí están ahora en manos de los sistemas operativos modernos, de modo que actualmente esta división se utiliza fundamentalmente para especificar archivos externos utilizados en el programa.

Data division (división de datos): contiene detalle sobre todas las zonas de datos utilizadas por el programa. En cierto sentido, es como la parte de declaraciones de un programa en PASCAL, donde se introducen los nombres de las variables, aunque es algo peligroso usar el concepto de "variable" en su acepción normal tratándose de datos de COBOL. En la división de datos hay dos secciones principales: la sección de archivos, que contiene esencialmente los trazados de registros para archivos externos, y la sección de trabajo-almacenamiento, para los datos utilizados sólo dentro de un programa.

Procedure division (división de procedimientos): aquí se especifican las operaciones y procedimientos a llevar a cabo sobre los datos de la división de datos. Si fuera necesario, la división de procedimientos se podría dividir en dos secciones, y cada sección o la división entera está compuesta de párrafos con nombre compuestos de oraciones. Una oración puede corresponder a una o más "sentencias" o "instrucciones" que a su vez se pueden subdividir en cláusulas para modificar o calificar la verdadera operación que se esté efectuando. La estructura de la división de procedimientos está claramente diseñada para parecerse lo más posible al inglés común, lo que, en unión con largos identificadores de hasta treinta caracteres, puede dar lugar a programas que resulten comprensibles no sólo a los programadores sino también a los no programadores. Sin embargo, escribir un "código espagueti" incomprensible en COBOL es tan fácil como en cualquier otro lenguaje.

Centraremos nuestra atención en la *data division* (división de datos), puesto que definir y designar las zonas de datos es una parte muy importante en un programa en COBOL. En relación a los datos, el punto de vista del COBOL consiste en considerar la división de datos como una larga serie de caracteres que se pueden dividir y subdividir a voluntad. Lo mejor es pensar en ella como si se tratara de un bloque de memoria. A las diversas divisiones que se efectúan en este bloque de datos se les asigna un nombre y se les otorga un número de nivel para indicar la estructura básica de los datos.

Los números de nivel normales van de 01 a 49, con algunos adicionales. Una sección de la división de datos se definirá en el nivel 01. Esta sección se puede luego subdividir otorgándoles a las subsecciones números de nivel superior (02, p. ej.), si bien los números no necesitan ser consecutivos. Una de estas subsecciones se puede volver a subdividir mediante el empleo de un número de nivel aún mayor, y así sucesivamente.

Los nombres de datos pueden tener hasta 30 caracteres de longitud, usando solamente caracteres alfabéticos y numéricos y guiones. El COBOL posee

un gran vocabulario de palabras reservadas que no se pueden utilizar para nombres de datos. No obstante, son pocas las que poseen guiones, de modo que no es difícil superar este problema.

Es necesario distinguir entre dos tipos de datos: elementales y grupales. Un dato elemental es aquel que no está subdividido; en la división de procedimientos se puede aludir tanto a datos elementales como a grupales. Cada dato elemental ha de tener un USAGE y un PICTURE. Hay dos USAGE principales: COMPUTACION y DISPLAY.

Como puede suponer, COMPUTATIONAL se utiliza para datos numéricos destinadosa cálculos y hará que el número se almacene en forma binaria en vez de una serie de caracteres, realizando, en consecuencia, los cálculos aritméticos con mayor rapidez. No obstante, un dato no tiene que ser COMPUTATIONAL para poderse utilizar para cálculos. DISPLAY es el empleo pordefecto normal, que simplemente significa que el dato se alamacena como una serie de caracteres. Cada uno de estos dos USAGEs puede ser objeto de otros refinamientos, según el sistema que se utilice; por ejemplo, COMPUTATIONAL-3 para un número de punto flotante. La cláusula USAGE es opcional; si se omite, entonces se da por sentado DISPLAY.

La cláusula PICTURE define el contenido esperado de cada posición de carácter en un dato elemental. Los principales símbolos utilizados con: 9 para indicar un carácter numérico; A para un carácter alfabético; X para un carácter alfanumérico; y V para indicar la posición de un punto decimal en un dato numérico. Normalmente el punto decimal no se almacena, pero el COBOL lleva el registro de su posición para que los cálculos se puedan realizar correctamente.

Una S indica que un dato numérico llega signo. Si se omite, entonces se toma automáticamente el valor absoluto de cualquier número almacenado. Hay opciones en cuanto a almacenar el signo como un carácter separado o no. (Observe que PIC es la forma abreviada de PICTURE, y que se puede utilizar cualquiera de las dos.) Por último, además de un PICTURE y un USAGE, a cualquier dato se le puede asignar un valor inicial o especificarlo como una matriz.

Al igual que el FORTRAN, el COBOL se diseñó originalmente para usar con tarjetas perforadas. La longitud de cada línea no puede ser mayor de 72 caracteres y, de ser necesario, se debe utilizar una línea de continuación. El trazado está especificado estrictamente, pero muchos sistemas permitirán un trazado más libre si el programa no está destinado para otra máquina. Las seis primeras columnas se utilizan para números de línea. El COBOL no requiere números de línea, pero se pueden incluir para ayudar a la depuración. La columna 7 se usa para marcar una línea de continuación; un asterisco en esta posición indica un comentario. Las columnas de las 8 a la 11 son la zona A. Aquí empiezan los nombres de división, sección y párrafos, así como los datos de nivel 01. Otros datos y sentencias de la división de procedimientosempiezan en la zona B: columnas de la 12 a la 72. La mayoría de las declaraciones de datos constituyen una oración y, por tanto, deben terminar con un punto. Si en un dato se necesitan caracteres a los que realmente no se alude en la división de procedimientos, entonces se utiliza la palabra reservada FILLER.

Detalles personales

Definición de datos para un registro de un archivo de personal:

```
01 EMPLEADO-REGISTRO.
   02 EMPLEADO-NOMBRE.
      03 EMPLEADO-INITIAL PIC A.
      03 FILLER PIC X.
      03 EMPLEADO-APELLIDO PIC X(15).
   02 EMPLEADO-DEPARTAMENTO PIC XXX.
   02 EMPLEADO-SALARIO-CATEGORIA PIC 9.
   02 EMPLEADO-FECHA-NACIMIENTO.
      03 DIA-NACIMIENTO PIC 99.
      03 MES-NACIMIENTO PIC 99.
      03 AÑO-NACIMIENTO PIC 99.
```

Registro de personal
Vemos los tipos de datos utilizados para mantener un registro de información personal sobre los empleados. Observe el uso de PIC para especificar los tipos de datos esperados: 9 indica un número, A un caracter alfabético y X un carácter alfanumérico

Definición de datos para una línea de cabecera (*heading*) destinada a producir un informe a partir del archivo anterior:

```
s  01HEADING-LINE.
      02 HEADING-1 PIC X(4) VALUE "NOMBRE".
      02 FILLER PIC X(20)VALUE SPACES.
      02 HEADING-2 PIC X(10) VALUE
         "DEPARTAMENTO".
      02 FILLER PIC X(3) VALUE SPACES.
      02 HEADING-3 PICX(9) VALUE "SAL
         CATEG".
      02 FILLER PIC X(10) VALUE SPACES.
      02 HEADING-4 PIC X(3) VALUE "FECHA
         NACIM".
```

Diseño del trazado
Aquí se define el trazado del informe a generar. A los datos se les pueden asignar valores iniciales. HEADING-2, por ejemplo, consta de 10 caracteres alfanuméricos y se le ha asignado el valor DEPARTAMENTO

Definición de datos para una línea de salida destinada a producir un informe a partir del archivo de personal:

```
01OUTPUT-LINE.
   02 OUTPUT-NOMBRE.
      03 OUTPUT-INITIAL PIC A.
      03 FILLER PIC X VALUE".".
      03 OUTPUT-APELLIDO PIC X(15).
   02 FILLER PIC X(7) VALUE SPACES.
   02 OUTPUT-DEPARTAMENTO PIC XXX.
   02 FILLER PIC X(10) VALUE SAPCES.
   02 OUTPUT-SALARIO-CATEGORIA PIC 9.
   02 FILLER PIC X(18) VALUE SPACES.
   02 OUTPUT-FECHA-NACIMIENTO.
      03 OUTPUT-DIA PIC 99.
      03 FILLER PIC X VALUE"/".
      03 OUTPUT;MES PIC 99.
      03 FILLER PIC X VALUE"/".
      03 OUTPUT-AÑO PIC 99.
```

Rellenando espacio
Estas líneas especifican la línea de salida desde el archivo de personal. En los datos se pueden incluir caracteres aun cuando no se aluda a ellos en la división de procedimientos; esto se hace utilizando la palabra reservada FILLER, que se usa aquí y en el listado anterior para insertar espacios

Definición de datos para una matriz unidimensional de veinte números de 3 dígitos y una matriz unidimensional de 10 * 20 números de 3 dígitos (con signo y un lugar decimal):

```
01 DATA-TABLE-1.
   02 DATA-TABLE-ENTRY PIC S99V9 USAGE
      COMPUTATIONAL OCCURS 20 TIMES.
01 DATA-TABLE-2.
   02 DATA-TABLE-ROW OCCURS 10 TIMES.
      03 DATA-TABLE-ENTRY-2 PIC S99V9
         USAGE
         COMPUTATIONAL OCCURS 20 TIMES.
```

Hecho para ser veloz
Declarar un dato como COMPUTATIONAL, como hace esta sección del programa, significa que los valores se almacenan en formato binario en lugar de en caracteres. Ello hace que los ulteriores cálculos que utilicen el dato sean mucho más rápidos. No obstante, los datos no han de definirse necesariamente de esta manera si usted quiere efectuar operaciones aritméticas con ellos, pero los cálculos son datos retenidos como caracteres serán más lentos

Facilidades adicionales

Añadiremos algunas nuevas funciones a nuestro programa de hoja electrónica

Cuando se prepara una hoja electrónica para realizar una serie de cálculos, con frecuencia las fórmulas a entrar en cada celda de una columna o fila determinadas son similares. Por ejemplo, quizá quisiéramos sumar cada elemento de la primera columna con cada elemento de la segunda columna y colocar los resultados en la tercera. para llevar a cabo esta sencilla tarea, sería necesario entrar una fórmula en cada celda de la tercera columna. Sería preciso programar la primera celda de esta columna, A3, con fórmula A1 + A2; sería necesario programar la siguiente celda de la tercera columna, B3, con la fórmula B1 + B2, y así sucesivamente. Si bien las fórmulas no son idénticas, son similares, de modo que para evitar la tediosa tarea de tener que entrar manualmente las fórmulas en todas las celdas de la tercera columna, sería de gran utilidad contar con una facilidad que entrara automáticamente fórmulas adecuadamente modificadas en una fila o columna entera (o en parte de ella). Este cometido lo lleva a cabo la función Reproducir (*replicate*).

Las funciones Borrar (*clear*), Almacenar (*store*) y Recuperar (*retrieve*) actúan sobre la totalidad de la hoja y, como sus nombres indican, permiten que el usuario borre los datos de las celdas de la hoja completa, almacene los datos para toda la hoja en la memoria (en lugar de en disco o cinta) y los recupere ulteriormente.

Otra función útil para moverse por la hoja es la función Tab. Si bien es posible trasladarse a cualquier parte de la hoja mediante el cursor, ello puede ser bastante lento, en especial porque la hoja se ha de reescribir cada vez que la ventana de pantalla desfila por la hoja. La función Tab permite que el usuario salte hasta cualquier celda simplemente entrando el nombre de la celda.

La función "Reproducir"

Las subrutinas que manipulan la función Reproducir están ubicadas entre las líneas 5400 y 5995. La primera rutina toma la fórmula a reproducir y la descompone en los elementos que la constituyen. Esta rutina en realidad es llamada por la rutina Reproducir principal y trabaja con una fórmula tomada de la matriz de fórmulas de infijos F$() y almacenada en C$. La fórmula se descompone en sus operadores, nombre de celdas y constantes, y se coloca en la matriz E$(). La rutina va explorando C$ carácter por carácter, y si el actual no es un operador se le añade una serie temporal, T$. Si el carácter *es* un operador, entonces se coloca T$ en el siguiente elemento libre de E$() (ya que encontrar un operador significa que en T$ hay almacenado un operando completo). El operador que se acaba de encontrar también se añade a E$(). El proceso continúa hasta haber explorado la fórmula en su totalidad.

La siguiente sección de código, entre las líneas 5500 y 5650, contiene la rutina "entrar y verificar". Ésta permite que el usuario dirija la función Reproducir para que actúe sobre un grupo de celdas determinado y está escrita para aceptar instrucciones con el formato A1(B1-F1. Esto significa: tomar la fórmula de la celda A1 y reproducirla en la columna 1 entre B1 y F1. Por el contrario, para reproducir a lo largo de una fila, se podría entrar una instrucción como ésta: A1(A2-A10. La primera sección de la rutina de entrada, entre las líneas 5500 y 5520, comprueba que la entrada posea una sintaxis correcta. A continuación, se divide la entrada para extraer los tres nombres de celda sobre las que es necesario trabajar. Estos nombres se colocan en seguida en R1$, R2$ y R3$.

La siguiente sección, en la línea 5700, es el verdadero punto de entrada a la rutina Reproducir, siendo su función la de decidir si la instrucción implica que se ha de reproducir una fila o una columna y saltar a la rutina para reproducir una fila o una columna adecuada. Se puede decidir si se ha de reproducir una fila o una columna examinando los tres nombres de celdas entrados como parte de la instrucción Reproducir. Cada nombre de celda se compone de una letra seguida por un número. Si cada uno de los tres nombres de celda comienzan con la misma letra, como en A1(A2-A10, luego está claro que la reproducción de fórmula debe realizarse a lo largo de una fila, en este caso, la fila A. Sin embargo, si la parte numérica de cada nombre de celda es la misma, por ejemplo A3(B3-H3, entonces se ha de reproducir una columna. Un efecto colateral de la comprobación de fila o de columna es que las entradas ilegales, tales como A1(B1-F2, se detectan automáticamente.

En las líneas 5800 a 5895 encontramos el código que reproduce una fórmula a lo largo de una columna. En primer lugar, se llama a la rutina que analizábamos primero y que descompone la fórmula a reproducir. En la línea 5820, un bucle FOR-...NEXT utiliza los valores ASCII de los primeros caracteres de R2$ y R3$ (los nombres de las celdas entre las cuales se ha de reproducir la fórmula) como sus variables de control. Entonces comienza, en la línea 5830, la tarea de construir la nueva fórmula para cada celda.

En esta etapa es interesante analizar las limitaciones de la función Reproducir. Cuando usted reproduce una fórmula a lo largo de una columna, la función sólo trabajará adecuadamente si cada nombre de celda dentro de la fórmula a reproducir posee la misma letra de fila. Ello significa que una fórmula como A1+A2*A3 se reproducirá con todo éxito, no así se tratara de A1+A2*B2. Sin embargo, ésta no es una restricción seria, porque la mayoría de las aplicaciones que se valen de la función Reproducir emplean fórmulas que trabajan sobre elementos de una sola fila.

"Borrar", "Almacenar", "Recuperar" y "Tab"

El código para todas estas cortas rutinas se da entre las líneas 5000 y 5395. Borrar la hoja es una cues-

tión muy simple. La matriz (M(,), utilizada para retener los datos de celda para toda la hoja, se establece en cero mediante un par de bucles FOR...NEXT anidados, y en virtud de la llamada a la subrutina de la línea 1700 se reimprime la hoja de la pantalla.

Para almacenar y recuperar la hoja de datos se emplea una segunda matriz, N(,). Las rutinas almacenar y recuperar transfieren el contenido de M(,) a N(,), o viceversa.

La rutina Tab empieza en la línea 5200 y acepta como entrada un nombre de celda. Antes de reescribir la hoja, es necesario volver a calcular los límites horizontales y verticales de la ventana de pantalla. Además de reimprimir los datos de la hoja, se han de imprimir nuevas etiquetas de columna y fila para dar cabida al hecho de que la ventana de la hojua puede haberse modificado a consecuencia del desplazamiento a una nueva celda.

Complementos al BASIC

BBC Micro:

Introduzca estos cambios en la versión para el C64 (asegúrese de que las sentencias PRINT de las líneas 5220 y 5502 contengan espacios suficientes para borrar una línea completa):

```
5210 PRINT TAB(0,22);
5220 PRINT "NUEVA
     CELDA:                "
5501 PRINT TAB(0,22;
5502 PRINT
     "REPRODUCIR:          "
```

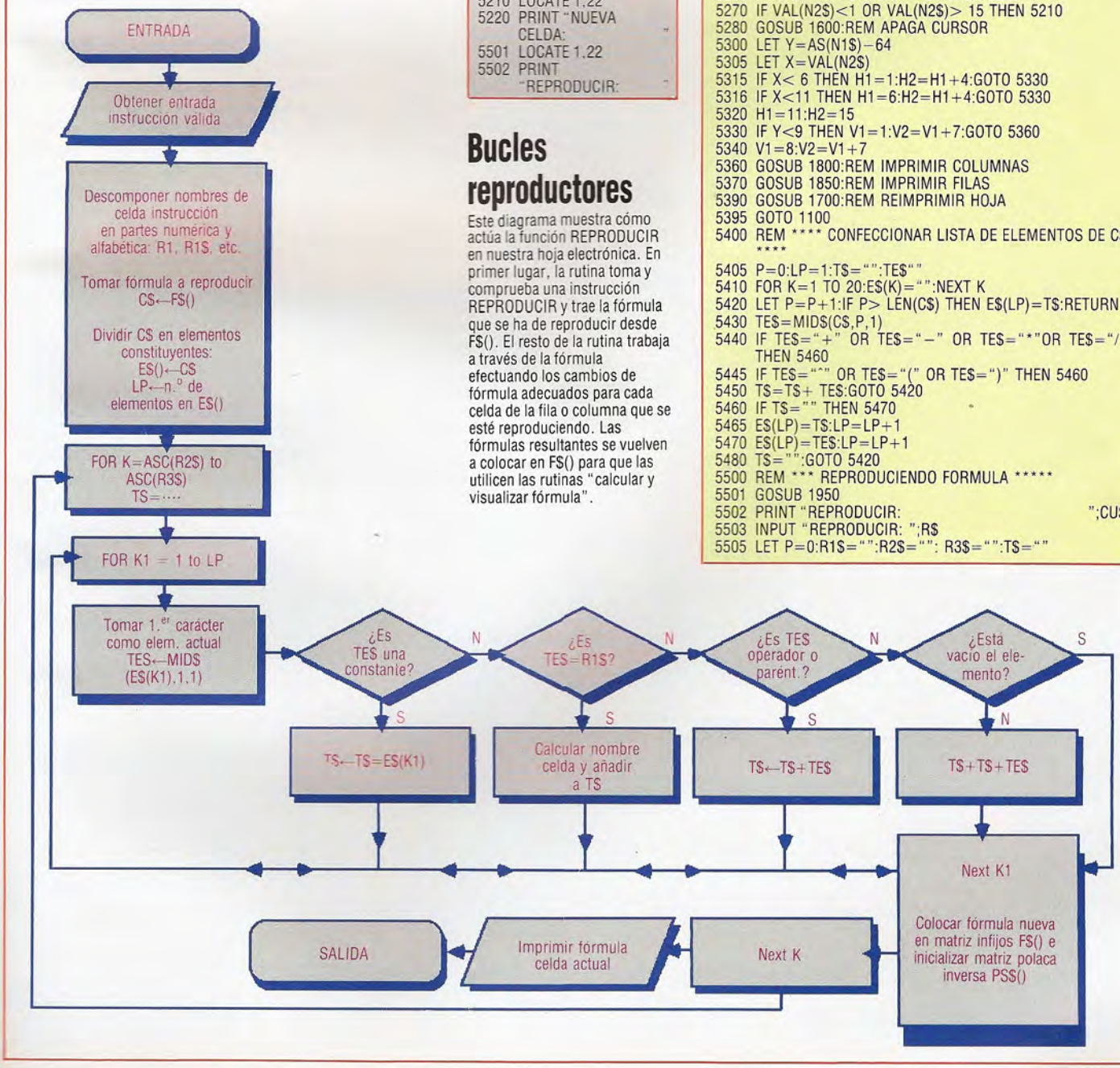
Amstrad CPC 464/664:

Introduzca estos cambios en la versión para el C64 (asegúrese de que las sentencias PRINT de las líneas 5220 y 5502 contengan espacio suficiente para borrar una línea entera)

```
5210 LOCATE 1,22
5220 PRINT "NUEVA
     CELDA:                "
5501 LOCATE 1,22
5502 PRINT
     "REPRODUCIR:          "
```

Bucles reproductores

Este diagrama muestra cómo actúa la función REPRODUCIR en nuestra hoja electrónica. En primer lugar, la rutina toma y comprueba una instrucción REPRODUCIR y trae la fórmula que se ha de reproducir desde FS(). El resto de la rutina trabaja a través de la fórmula efectuando los cambios de fórmula adecuados para cada celda de la fila o columna que se esté reproduciendo. Las fórmulas resultantes se vuelven a colocar en FS() para que las utilicen las rutinas "calcular y visualizar fórmula".



Caroline Clayton

Facilidades adicionales

Commodore 64:

```
2320 >LET P$=H$((J-1)*)15+ I,1TO ):LET
     I$=F$((J-1)*15+I,1TO)
5000 REM **** BORRAR MATRIZ ****
5010 FOR I=1 TO 15:FOR J=1 TO 15:M(I,J)=0:NEXT J,I:
     GOSUB 1700:RETURN
5100 REM **** TOMAR HOJA ANTERIOR ****
5110 FOR I=1 TO 15:FOR J=1 TO 15
5120 M(I,J)= N (I,J):NEXT J,I
5130 GOSUB 1700:RETURN REM IMPRIMIR DATOS
5150 REM ***** ALMACENAR HOJA ACTUAL ****
5160 FOR I=1 TO 15:FOR J=1 TO 15
5170 N(I,J)=M(I,J):NEXT J,I
5180 GOSUB 1700:RETURN:REM IMPRIMIR DATOS
5200 REM **** RUTINA GOTO CELDA ****
5210 GOSUB 1950:REM MOVER HASTA LINEA ENTRADA
5220 PRINT "NUEVA CELDA:                    ";CU$
5230 INPUT "NUEVA CELDA:",NC$
5240 LET NC$=MID$(NC$,1,3)
5250 LET N1$=MID$(NC$,1,1):LET N2$=MID$(NC$,2,2)
5260 IF N1$< "A" OR N1$> "0" THEN 5210
5270 IF VAL(N2$)<1 OR VAL(N2$)> 15 THEN 5210
5280 GOSUB 1600:REM APAGA CURSOR
5300 LET Y=AS(N1$)-64
5305 LET X=VAL(N2$)
5315 IF X< 6 THEN H1=1:H2=H1+4:GOTO 5330
5316 IF X<11 THEN H1=6:H2=H1+4:GOTO 5330
5320 H1=11:H2=15
5330 IF Y<9 THEN V1=1:V2=V1+7:GOTO 5360
5340 V1=8:V2=V1+7
5360 GOSUB 1800:REM IMPRIMIR COLUMNAS
5370 GOSUB 1850:REM IMPRIMIR FILAS
5390 GOSUB 1700:REM REIMPRIMIR HOJA
5395 GOTO 1100
5400 REM **** CONFECCIONAR LISTA DE ELEMENTOS DE C$
     ****
5405 P=0:LP=1:T$="":TE$""
5410 FOR K=1 TO 20:E$(K)="":NEXT K
5420 LET P=P+1:IF P> LEN(C$) THEN E$(LP)=T$:RETURN
5430 TE$=MID$(C$,P,1)
5440 IF TE$="+" OR TE$="-" OR TE$="*"OR TE$="/"
     THEN 5460
5445 IF TE$="^" OR TE$="(" OR TE$=")" THEN 5460
5450 T$=T$+ TE$:GOTO 5420
5460 IF T$="" THEN 5470
5465 E$(LP)=T$:LP=LP+1
5470 E$(LP)=TE$:LP=LP+1
5480 T$="":GOTO 5420
5500 REM *** REPRODUCIENDO FORMULA *****
5501 GOSUB 1950
5502 PRINT "REPRODUCIR:                     ";CU$
5503 INPUT "REPRODUCIR: ";R$
5505 LET P=0:R1$="":R2$="": R3$="":T$=""
```

```
5510 IF MID$(R$,3,1)<>"(" AND MID$(R$,4,1)<>"(" THEN
     5500
5515 IF MID$(R$,6,1)<> "-" AND MID$(R$,7,1)<> "-" AND
     MID$(R$,8,1)<> "-" THEN 5500
5517 IF R$="" THEN 5500
5520 LET P=P+1
5530 LET T$=MID$(R$,P,1)
5540 IF T$"(" THEN P=P+1:GOTO 5570
5550 LET R1$=R1S+T$
5560 GOTO 5520
5570 LET T$=MID$(R$,P,1)
5580 IF T$="-" THEN P=P+1:GOTO 5610
5590 LET R2$=R2$+T$
5600 LET P=P+1:GOTO 5570
5610 LET T$=MID$(RS,P,1)
5620 LET R3$=R3$+T$
5630 LET P=P+1
5640 IF P<= LEN (R$) THEN 5610
5650 RETURN
5700 REM ** DECIDIR COLUMNA O FILA ***
5730 GOSUB 5500:REM COMPROBAR SI ENTRADA
     VALIDA
5765 R1=VAL(MID$(R1S,2)):R1$=MIDS(R1S,1,1)
5770 R2=VAL(MID$(R2S,2)):R2$=MID$(R2S,1,1)
5775 R3=VAL(MID$(R3S,2)):R3$=MID$(R3S,1,1)
5780 IF R1=R2 AND R1=R3 THEN 5800
5790 IF MID$(R1S,1,1)=MID$(R2S,1,1)
     AND MIDS(R1S,1,1)=MID$(R3S,1,1) THEN 5900
5795 GOTO 5700
5800 REM **** REPRODUCIR COLUMNA ****
5805 LET C$=FS((ASC(R1$)-65)* 15+R1)
5810 GOSUB 5400:REM CONFECCIONAR LISTA DE
     ELEMENTOS
5820 FOR K=ASC (R2$) TO ASC(R3$):T$=""
5830 FOR K1=1 TO LP
5840 LET TE$=MID$(ES(K1),1,1)
5845 IF TE$> ="0" AND TE$< ="9" OR TE$="." THEN
     T$=T$+E$(K1):GOTO 5880
5850 IF TE$=R1$ THEN T$=T$+ CHR$(K) +
     MID$(E$(K1),2):GOTO 5880
5860 IF TE$="+" OR TE$="-" OR TE$="*" THEN
     T$=T$+TE$:GOTO 5880
5865 IF TE$="/" OR TE$="^" OR TE$="(" OR TE$=")" THEN
     T$=T$+ TE$:GOTO 5880
5870 IF TE$<>"" THEN T$=T$+ TE$
5880 NEXT K1
5890 FS((K-65)*15+R1)=T$:PS$((K-65)*15+R1)=""
5895 NEXT K:GOSUB 1900:RETURN
5900 REM **** REPRODUCIR FILA ****
5905 LET C$=FS((ASC(R1$)-65)*15+R1)
5910 GOSUB 5400:REM CONFECCIONAR LISTA DE
     ELEMENTOS
5920 FOR K=R2 TO R3:T$=""
5930 FOR K1=1 TO LP
5940 LET TE$=MIDS(ES(K1),1,1)
5945 IF (TE$> ="0" AND TE$ <="9") OR TE$="." THEN
     T$=T$+ES(K1):GOTO 5980
5950 IF TE$>="A" AND TE$<="0"
     THEN T$=T$+T$+MID$(STR$(K),2):GOTO 5980
5960 IF TE$="+" OR TE$="-" OR TE$="*" THEN
     T$=T$+TE$:GOTO 5980
5965 IF TE$="/" OR TE$="^" OR TE$="(" OR TE$=")" THEN
     T$=T$+TE$:GOTO 5980
5970 IF TE$<>"" THEN T$=T$+TE$
5980 NEXT K1
5990 F$((ASC(R1$)-65)*15+K)=T$:PS$((ASC(R1$)-65
     *15+K)=""
5995 NEXT K: GOSUB 1900:RETURN
```

Spectrum:

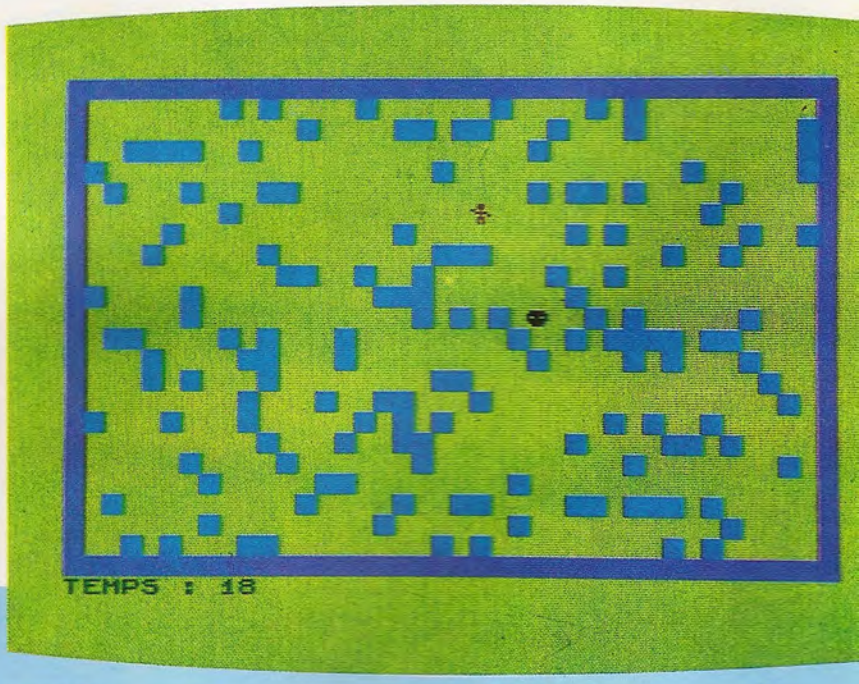
```
5000 > REM *** BORRAR MATRIZ ***
5010 DIM M(15,15):GO SUB 1700: RETURN
5100 REM *** TOMAR HOJA ANTERIOR ****
5110 FOR I=1 TO 15: FOR J=1 TO 15
5120 LET M(I,J)=N(I,J)
5130 NEXT J: NEXT I
5140 GO SUB 1700: RETURN
5150 REM ALMACENAR HOJA ACTUAL EN
     MEMORIA
5160 FOR I=1 TO 15: FOR J=1 TO 15
5170 LET N(I,J)=M(I,J): NEXT J: NEXT I
5180 GO SUB 1700: RETURN
5200 REM **** RUTINA GOTO CELDA ****
5210 PRINT AT 18.0;"          ENTRE NUEVA
     CELDA"
5220 INPUT LINE N$
5225 IF LEN (N$) < 3 THEN LET N$=N$+ " "
5230 LET N$=N$(1 TO 3):LET O$=N$(2 TO 3)
5240 LET N$=N$(1)
5250 IF N$<"A" OR N$> "0" THEN GO TO 5210
5260 IF VAL (O$)<1 OR VAL (O$)> 15 THEN GO TO
     5210
5280 GO SUB 1600: REM APAGAR CURSOR
5300 LET Y=CODE (N$)-64
5305 LET X=VAL (O$)
5310 LET VL=V2: LET HL=H2
5315 IF X< H1 THEN GO TO 5330
5316 IF X< H2 THEN GO TO 5335
5320 IF X>=12 THEN LET H1=12:LET H2=15: GO
     TO 5340
5330 LET H1=X: LET H2=H1+3
5335 IF Y< V1 THEN GO TO 5350
5336 IF Y< V2 THEN GO TO 5360
5340 IF Y> =9 THEN LET V1=9: LET V2=
     V1+6: GO TO 5360
5350 LET V1=Y: LET V2=Y+6
5360 GO SUB 1800: REM IMPRIMIR COLUMNAS
5370 GO SUB 1850: REM IMPRIMIR FILAS
5380 IF V2=VL AND H2=HL THEN GO SUB 1650:
     GO TO 1100
5390 GO SUB 1700
5395 GO TO 1100
5400 REM **** CONFECCIONAR LISTA DE
     ELEMENTOS DE I$ ***
5405 LET P=0: LET LP=1: LET T$="":LET U$=""
5410 DIM E$(20,5)
5420 LET P=P+1: IF P> LEN (C$) THEN LET
     E$(LP)=T$: RETURN
5430 LET U$=C$(P)
5440 IF U$="+" OR U$="-" OR U$="*" OR
     U$="/" THEN GO TO 5460
5445 IF U$="^" OR U$="(" OR U$=")" THEN GO
     TO 5460
5450 LET T$=T$+U$: GO TO 5420
5460 LET E$(LP)=T$: LET LP=LP+1
5470 LET E$(LP)=U$: LET LP=LP+1
5480 LET T$="": GO TO 5420
5500 REM *** REPRODUCIENDO ***
5505 LET P=0: LET X$="": LET Y$"": LET Z$="":
     LET T$=""
5510 IF R$(3)<>"(" AND R$(4)<>"(" THEN GO
     TO 5660
5515 IF R$(6)<> "-" AND R$(7)<>"-" AND
     R$(8)<>"-" THEN GO TO 5660
5520 LET P=P+1
5530 LET T$=R$(P)
5540 IF T$="("THEN LET P=P+1:GO TO
     5570
5550 LET X$=X$+T$
5560 GO TO 5520
5570 LET T$=R$(P)
5580 IF T$="-" THEN LET P=P+1:GO TO
     5610
5590 LET Y$=Y$+T$
5600 LET P=P+1: GO TO 5570
5610 LET T$=R$(P)
5620 LET Z$=Z$+T$
5630 LET P=P+1
5640 IF P< = LEN (R$) THEN GO TO 5610
5650 RETURN
5670 PRINT AT 18,0;"ERROR EN SERIE
     REPRODUCIR"
5680 RETURN
5700 REM **** DECIDIR COLUMNA O FILA ****
5710 PRINT AT 18,0;"REPRODUCIR:"
5720 INPUT LINE R$
5730 GO SUB 5500
5765 LET R1=VAL (X$(2TO )): LET X$=X$(1)
5770 LET R2=VAL (Y$(2 TO )): KET Y$=Y$(1)
5775 LET R3=VAL (Z$(2 TO )): LET Z$=Z$(1)
5780 IF R1=R2 AND R1=R3 THEN GO TO 5800
5790 IF X$=Y$ AND X$=Z$ THEN GO TO 5900
5795 GO TO 5700
5800 REM **** REPRODUCIR COLUMNA *****
5805 LET C$=F$((CODE (X$)-65)*15+R1)
5810 GO SUB 5400: REM ELABORAR LISTA DE
     ELEMENTOS
5820 FOR K=CODE (Y$) TO CODE (Z$): LET T$=""
5830 FOR J= TO LP
5840 LET U$=E$(J) (TO 1)
5850 IF U$=X$ THEN LET T$=T$+ CHR$ (K)+
     E$(J,2 TO ): GO TO 5800
5860 IF U$="+ " OR U$="-" OR U$="*" THEN
     LET T$=T$ +U$: GO TO 588
5865 IF U$="/" OR U$="^" OR U$="(" OR
     U$=")" THEN LET T$=T$+U$: GO TO 5880
5870 IF U$<>"" THEN LET T$=T$+U$
5880 NEXT J
5882 LET U$="": FOR i=1 TO LEN (T$): IF T$ (i TO
     i)<>" " THEN LET U$= U$+T$(i TO i)
5885 NEXT i
5890 LET F$((K-65)*15+R1)=U$: LET
     H$((K-65)* 15+R1)=""
5895 NEXT K: GO SUB 1900: RETURN
5900 REM **** REPRODUCIR FILA ****
5905 LET C$=F$((CODE (X$)-65)*15+R1)
5910 GO SUB 5400: REM CONFECCIONAR LISTA DE
     ELEMENTOS
5920 FOR K=R2 TO R3: LET T$=""
5930 FOR J=1 TO LP
5940 LET U$=E$(J,1 TO 1)
5950 IF U$> ="A" AND U$< = "0" THEN LET
     T$=T$+U$+STR$ (K): GO TO 5980
5960 IF U$="/" OR U$="^" OR U$="(" OR
     U$=")" THEN LET T$=T$+U$: GO TO 5980
5970 IF U$<>"" THEN LET T$=T$+U$
5980 NEXT J
5990 LET F$((CODE(X$)-65)*15+K)=T$: LET
     H$((CODE (X$)-65)*15+K)=""
5995 NEXT K: GO SUB 1900: RETURN
```

Persecución

El ladrón, representado por una máscara negra, ha escapado llevándose el botín... He aquí otra vez este conocido tema de la informática recreativa escrito en BASIC para el MO5 de Thomson

Se esconde en la ciudad, y usted tiene treinta minutos para encontrarlo y detenerlo. Atención, ¡no se precipite! Efectivamente, si se echa sobre él sin pensar, tiene todas las posibilidades de escapársele. La mejor manera de cogerlo es alcanzarlo de lado. (Es el método más eficaz a condición de no fallar). Si no se siente lo bastante seguro de sí mismo, atáquelo de frente, lo cual es más fácil pero mucho menos eficaz, ya que no es tan discreto. *Otro consejo*: no intente perseguirlo; esto no le daría resultado, pues él es mucho más rápido que usted. Ha de observar sus movimientos, como un detective. Cuando vea que da la vuelta, acérquese sin hacer ruido y sorpréndalo en el momento justo. Pero recuerde: ¡el tiempo va pasando!

Para desplazarse utilice la palanca de mando (*joystick*) o las teclas de control del cursor.




```
10 REM **************
20 REM * PERSECUCION *
30 REM **************
40 CLEAR ,,2
50 GOSUB 1330
60 S=0
70 N$=CHR$(32)
80 V$=GR$(1)
90 P$=GR$(0)
100 GOSUB 830
110 ON JK GOTO 180
120 D$=INKEY$
130 DH=(D$=F1$) - (D$=F2$)
140 DV= (D$=F3$) - (D$=F4$)
150 IF DH<> THEN DX=DH:DY=0
160 IF DV<>0 THEN DY=DV:DX=0
170 GOTO 210
180 ST=STICK(0)
190 DX=(ST=7) -(ST=3)
200 DY=(ST=1) - (ST=5)
210 Z=Z-0.2
220 LOCATE 0,24
230 COLOR 0
240 PRINT "TIEMPO :";INT (Z+1);
250 IF Z<0 THEN 500
260 PX=PX+DX
270 PY=PY+DY
280 C=POINT(PX*8+4,PY*8+4)
290 IF C=0 THEN 1280
300 IF C<>-4 THEN PX=XP:PY=YP
310 LOCATE XP,YP
320 PRINT N$;
330 LOCATE PX,PY
340 COLOR 1
350 PRINT P$;
360 YP=PY
370 XP=PX
380 VX=VX+CX
390 VY=VY+CY
400 IF SCREEN(VX,VY)<>32 THEN GOSUB 670
410 IF SCREEN(VX,VY)<>32 THEN 380
420 LOCATE XV,YV
430 COLOR 0
440 PRINT N$;
450 LOCATE VX,VY
460 PRINT V$;
470 XV=VX
480 YV=VY
490 GOTO 110
500 IF INKEY$<>"" THEN 500
510 IF R<S THEN R=S
520 COLOR 0
530 LOCATE 10,6
540 PRINT "TIEMPO TRANSCURRIDO";
550 LOCATE 10,10
560 PRINT "PUNTOS :";S;
570 LOCATE 10,14
580 PRINT "PUNTUACION :";R;
590 LOCATE 10,18
600 PRINT "OTRA ?";
610 D$=INKEY$
620 IF D$="" THEN 610
630 IF D$<>"N" THEN RUN
640 SCREEN 4,6,6
650 CLS
660 END
670 DT=DT+1
680 GOSUB 780
690 IF SCREEN(XV+CX,YV+CY)=32 THEN VX=XV
    +CX:VY=YV+CY:RETURN
700 DT=DT-2
710 GOSUB 780
720 IF SCREEN(XV-CX,YV+CY)=32 THEN VX=XV
    +CX:VY=YV+CY:RETURN
730 DT=DT-1
740 GOSUB 780
750 VX=XV+CX
760 VY=YV+CY
770 RETURN
780 IF DT>4 THEN DT=DT-4
790 IF DT<1 THEN DT=DT+4
800 CX=(DT=1)-(DT=3)
810 CY=(DT=2)-(DT=4)
820 RETURN
830 CLS
840 COLOR 5
850 FOR VX=0 TO 39
860 LOCATE VX,0
870 PRINT CHR$(127);
880 LOCATE VX,23
890 PRINT CHR$(127);
900 NEXT VX
910 FOR VY=1 TO 22
920 LOCATE 0,VY
930 PRINT CHR$(127);
940 LOCATE 39,VY
950 PRINT CHR$(127);
960 NEXT VY
970 COLOR 4
980 FOR VX=1 TO 150
990 GOSUB 1240
1000 LOCATE PX,PY
1010 PRINT CHR$(127);
1020 NEXT VX
1030 GOSUB 1240
1040 VX=PX
1050 VY=PY
1060 COLOR 0
1070 LOCATE VX,VY
1080 PRINT V$;
1090 XV=VX
1100 YV=VY
1110 GOSUB 1240
1120 COLOR 1
1130 LOCATE PX,PY
1140 PRINT P$;
1150 XP=PX
1160 YP=PY
1170 Z=30
1180 CX=0
1190 CY=0
1200 DX=0
1210 DY=0
1220 DT=0
1230 RETURN
1240 PX=INT (RND*38)+1
1250 PY=INT (RND*22)+1
1260 IF SCREEN (PX,PY) <> 32 THEN 1240
1270 RETURN
1280 FOR I=1 TO 5
1290 PLAY "T15REDO"
1300 NEXT I
1310 S=S+1
1320 GOTO 100
1330 SCREEN 4,3,3
1340 CLS
1350 DEFINT A-Y
1360 DEFGRS (0)=28,28,200,62,9,28,20,20
1370 DEFGRS (1)=0,126,255,153,255,126,60,0
1380 F1$=CHR$(8)
1390 F2$=CHR$(9)
1400 F3$=CHR$(11)
1410 F4$=CHR$(10)
1420 ATTRB 1,1
1430 LOCATE 10,10,0
1440 PRINT "JOYSTICK ?";
1450 D$=INKEY$
1460 PX=RND
1470 IF D$="" THEN 1450
1480 IF D$="O" THEN JK=1
1490 ATTRB 0,0
1500 RETURN
```

La joya de un sistema

En esta nueva serie estudiaremos los sistemas WIMP, centrando nuestra atención en los "lenguajes orientados hacia el objeto"

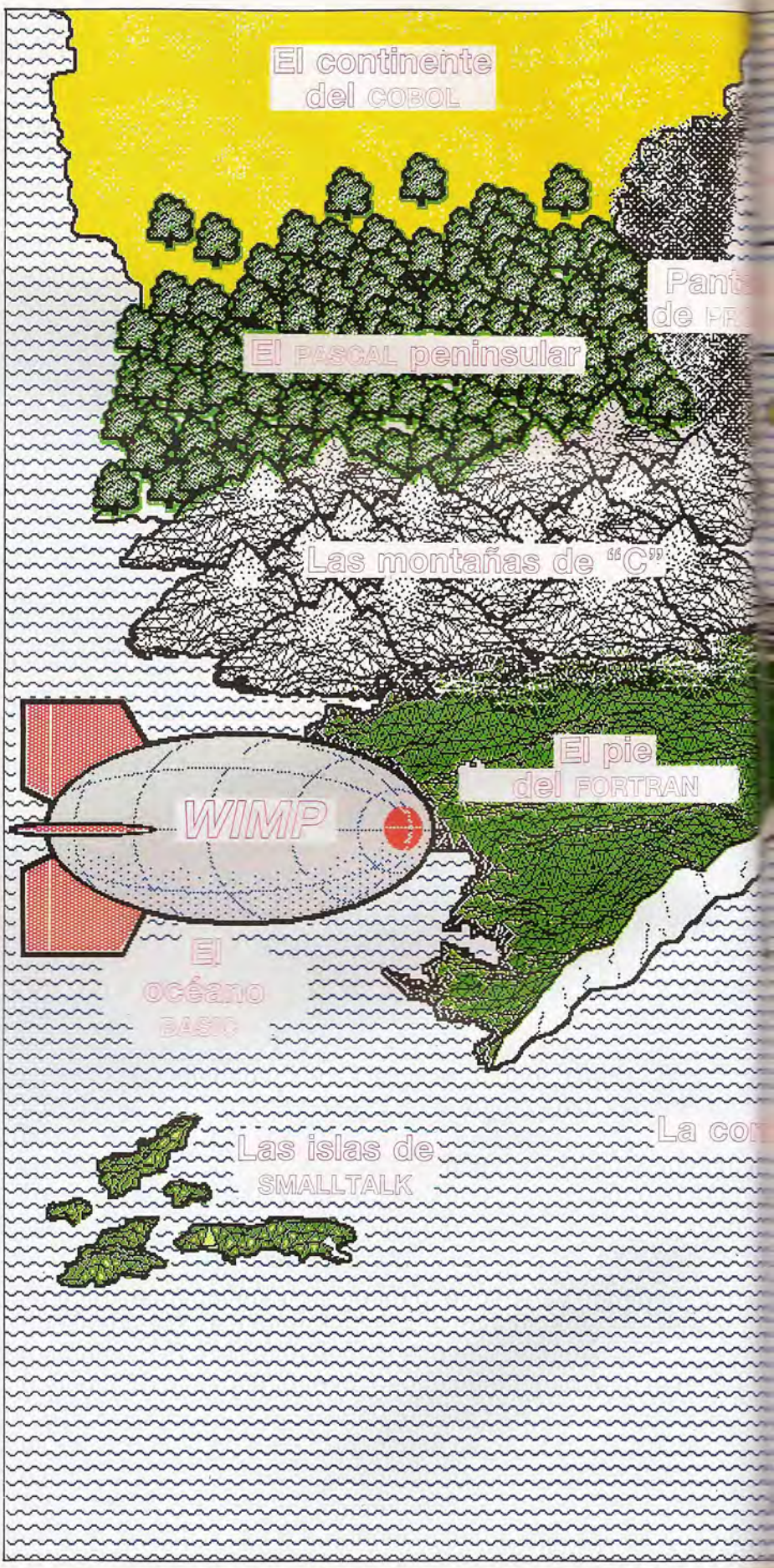
Al recién iniciado en informática se le suele introducir en el uso de los ordenadores personales mediante programas que se ejecutan, o bien se "cargan", al poner en marcha la máquina. En el primer caso, las máquinas se proporcionan con una versión de BASIC residente, con lo que a menudo los usuarios ingenuos tienen la impresión de que los ordenadores son poco más que "máquinas de BASIC" en la que las instrucciones se entran en modalidad directa (es decir, sin números de línea de programa).

Un intérprete de BASIC residente, como el que acabamos de describir, puede ofrecer un entorno más amable que el segundo tipo (el *programa de sistema*), que no es un entorno de lenguaje empaquetado sino que se conoce como el monitor, o sistema operativo (OS). Ejemplos típicos de sistemas operativos estándares son CP/M, MS-DOS, UNIX, etcétera. Estos sistemas tienden a ser bastante hostiles y se remontan a los días en que se esperaba que los usuarios de ordenadores fueran expertos en los intrincados detalles de su máquina.

Como sabrán la mayoría de los lectores, las máquinas como el Apple Macintosh han modificado de forma radical nuestras expectativas acerca de lo que podemos esperar de ellas y especialmente nuestra interacción con las mismas. Tales sistemas tienden a basarse en el uso de ventanas, iconos y ratones, y por este motivo han llegado a conocerse como sistemas WIMP (*windows, icons and mice*). La aparición de la tecnología WIMP es apenas un aspecto de un campo de investigación que es relativamente poco conocido (al cual se suele aludir como *programación orientada hacia el objeto*) y el trabajo que se está realizando en este campo tiene enormes implicaciones de cara al futuro. Es esta investigación, y los cambios a los que ha dado lugar, lo que examinaremos a lo largo de esta serie.

El lenguaje más "objetivo" que se ha desarrollado hasta ahora es, sin lugar a dudas, el SMALLTALK, producto de un equipo de investigadores del Centro de Investigación de Xerox en Palo Alto (PARC). En consecuencia, comenzaremos nuestra serie con un análisis de los antecedentes de SMALLTALK y un proyecto asociado con el mismo: el Dynabook de Alan Kay.

Quizá resulte extraño que una empresa que obtiene la mayor parte de sus beneficios a partir del papel se mostrara interesada en el concepto de un entorno de oficina sin papel: la "oficina electrónica". Sin embargo, esto fue efectivamente así en el caso de la Xerox Corporation de Estados Unidos a



La geografía de los lenguajes de programación ha visto con el correr de los años el desarrollo de muchos entornos diferentes, cada uno de ellos con sus propias características y peligros ocultos para el viajero distraído. Pero en un lejano archipiélago abandonado, denominado SMALLTALK, que el mundo descubriera hace muy poco tiempo, se ha desarrollado una modalidad de transporte que brilla sobre este enterno hostil. La nave llamada *Wimp*, navega a través de un cielo despejado, arrojando sobre cada rincón de la tierra y del mar su sombra, que es recibida con satisfacción

Ian McKinnell

comienzos de los años setenta. Su previsión, al anticiparse a las futuras tendencias, ha venido a significar que Xerox, tal vez en mayor medida que ninguna otra empresa, fuera responsable de los orígenes de lo que ahora conocemos como sistemas WIMP.

Xerox comprendió que con la intensa competencia de IBM, DEC y otras compañías, su tardía entrada en el campo de la informática sería un serio inconveniente a menos que pudiera captar una parte del mercado de fabricantes de ordenadores bien establecidos. Dieron el arriesgado paso de crear el PARC, dando a su equipo de investigadores y técnicos un extenso informe y un gran presupuesto para investigar en el campo de la automatización de oficinas, la inteligencia artificial (AI), la interface a la medida del hombre (MMI) y los sistemas de ordenador en general.

El punto de partida para el SMALLTALK fue el futurista Dynabook, o sistema de referencia para todos. Éste era la invención de Alan Kay, entonces estudiante de investigación, quien imaginó una época en la que todo el mundo tendría un pequeño ordenador autoalimentado (del tamaño aproximado de un libro) que permitiría a cada usuario el acceso dinámico a una enciclopedia completa para consulta de conocimientos.

El Dynabook, portátil y de altas prestaciones, tendría una visualización en alta resolución, dispositivos de entrada y salida para comunicaciones tanto visuales como de audio, y conexiones de radio a una red de satélites con una base de datos compartida. Lamentablemente, los problemas técnicos de la miniaturización, el procesamiento masivo y las fuentes de memoria requeridos para implementar la idea de Kay, dieron como resultado la dispersión del equipo de investigación Dynabook. Sin embargo, muchas de las ideas Dynabook han tenido una influencia intensa y duradera en la investigación y en el desarrollo, que ahora se han abierto camino en las máquinas disponibles actualmente.

Uno de los conceptos clave que surgieron durante las primeras etapas de desarrollo fue el del *centro de trabajo* personal. Hasta entonces la informática había significado compartir los recursos de una gran instalación central o, más recientemente, varios miniordenadores. El procesamiento de los datos era muy indirecto, con el procesamiento por lotes (*batch*), siendo la norma en grandes sistemas.

La diferencia más obvia entre este entorno y la filosofía PARC fue el tiempo de respuesta. Los sistemas convencionales suponían una espera de los resultados que en ocasiones era de varias horas. En 1972, la inmediatez implícita en el concepto Dynabook era bastante revolucionaria. Además, conceptos tales como el uso de un dispositivo señalador con aspecto de ratón, de iconos y ventanas, significó un replanteamiento radical de la forma de comunicar los datos hacia y desde el procesador central y la visualización. Ello condujo a la implementación en el firmware de potentes núcleos de gráficos exclusivos, y también al extraño lenguaje que los investigadores del PARC bautizaron como SMALLTALK.

El núcleo de la investigación en SMALLTALK giraba alrededor de la modelación de un sistema de hardware y software completo basado en la comunicación entre "objetos" con ciertas propiedades definidas.

Todos estamos familiarizados con varios objetos distintos que forman parte de cualquier sistema de ordenador: el teclado, la VDU y la impresora son ejemplos obvios. Cada uno de estos dispositivos de hardware posee propiedades fijas, y sólo comprende cómo realizar determinadas tareas. No tiene sentido pedirle al teclado que imprima un documento, por ejemplo. Cada objeto, a su modo, retiene una cierta cantidad de datos (información sobre sí mismo) y el conocimiento requerido para llevar a cabo una limitada cantidad de tareas correctamente. La pantalla VDU parece "saber" cómo visualizar los datos que se le envían, y ocuparse de sus propias tareas domésticas, tales como volver a enviar el cursor al lado izquierdo de la pantalla cuando el texto intenta continuar más allá del final de la línea.

En realidad, este objeto es una compleja red de interacción de hardware y software, en parte electrónica, en parte sistema operativo u otro software. Al considerar a cada uno de los componentes de un sistema de ordenador como un objeto, y al definir sus propiedades y los algoritmos necesarios para hacer que se conduzcan correctamente, un ingeniero de software puede crear un sistema basado en una red de objetos que se comuniquen entre sí. En vez de la tradicional técnica de lenguaje estructurado de llamar a un procedimiento por su nombre para procesar datos (pasándolos como sus parámetros), los "objetos" de programa se describen según ciertas clases o categorías, y contienen los métodos para procesar tipos de datos específicos. Éstos se suministran como argumentos de un "mensaje". Todo lo que hace el mensaje es pasar los datos y decirle al objeto receptor qué método aplicar: al que envía el mensaje no le concierne en absoluto cómo se lleva a cabo exactamente el método.

SMALLTALK es la implementación más completa del enfoque orientado hacia el objeto, pero requiere grandes recursos de memoria y potencia de procesamiento. Más recientemente, ha habido un interés considerable por el MODULA-2, en especial en Estados Unidos. Este lenguaje europeo es un derivado del PASCAL y fue diseñado por el creador de este lenguaje, el profesor Niklaus Wirth, para programar grandes sistemas en los que un equipo de programadores debía poder implementar distintos "módulos" de forma independiente (de ahí su nombre). Los datos, tipos de datos y procedimientos se pueden mantener separados en cualquier módulo a menos que se "importen" o "exporten" explícitamente para ser utilizados por otros módulos. Esto representa una analogía completa con la idea del SMALLTALK, con la excepción de que el concepto de pasar mensajes y argumentos de datos se sustituye por las llamadas a procedimientos y listas de parámetros de estilo PASCAL, más convencionales. Esto proporciona una forma simple y segura de construir grandes sistemas sin los peligrosos efectos colaterales a los que puede dar lugar el acceso ilimitado a todos los objetos del sistema.

Aunque todavía es muy reciente la aparición de máquinas como el Apple Macintosh, los consumidores ya están dando por sentadas la inmediatez y amabilidad de los sistemas operativos WIMP. SMALLTALK fue responsable del nacimiento de tales sistemas, y su implementación tuvo importantes implicaciones en el desarrollo que ha experimentado el hardware desde entonces.

La fórmula del éxito

Ahora nos corresponde examinar la regla de Bayes

Una época de diletantismo
En los siglos XVI y XVII, el ocio y el aprendizaje tendían a entremezclarse de modo tal que en nuestra época de especialización podría llegar a considerarse carente de toda seriedad. Así, no es sorprendente hallar que las apuestas y las matemáticas estuvieran unidas tan frecuentemente. El reverendo Bayes, que vemos en la ilustración, desarrolló una ecuación para el cálculo de posibilidades, y otros casi contemporáneos suyos (incluyendo a D'Alembert, Pascal y Fermat) se sintieron igualmente atraídos por los enigmas, potencialmente rentables, de las probabilidades

Tras haber analizado en el capítulo anterior la conversión de posibilidades a probabilidades, nos concentraremos ahora en la aplicación de cifras de probabilidades, que es donde el apostador puede valerse de la ayuda del micro.

Usted puede usar su ordenador para calcular las probabilidades que representan las posibilidades citadas, y los corredores de apuestas pueden utilizar esto para elaborar un buen registro de apuestas. Pero la mayoría de los micros se utilizan para detectar ganadores, y la única forma racional de hacerlo es evaluar el formulario. El problema es que hay demasiada información, lo que implica:

- Seleccionar qué evidencia es importante.
- Combinar evidencias desiguales.

Es aquí donde muestra su valía un teorema de estadística: la regla de Bayes. El reverendo Bayes era un clérigo del s. XVIII que produjo esta ecuación:

$$P(H|E)=P(E|H)\times P(H)/P(E)$$

que desde el punto de vista de las posibilidades se puede expresar así:

$$O(H|E)=O(H)\times LR(H|E)$$

Éstas pueden parecer complicadas, pero en realidad son bastante directas. En la primera ecuación, P(H|E) es la probabilidad condicionada, P, de una hipótesis, H, dada (|) alguna evidencia, E.

El O(H|E) de la segunda ecuación es el mismo que el P(H|E), pero expresado como posibilidades (a favor). La segunda ecuación se puede leer como: las posibilidades de una hipótesis (H) dada una evidencia (E) son iguales a las posibilidades anteriores (antes de conocer E) de esa hipótesis, multiplicadas por la razón de probabilidad para esa evidencia. Enseguida analizaremos las razones de probabilidad, pero primero simplifiquemos un poco las cosas examinando un ejemplo paso a paso.

Pronósticos en el fútbol

Partiendo con algunos datos de muestra, supongamos que usted tiene registros de partidos de fútbol ya disputados y desea pronosticar los ganadores locales. Hay que destacar dos puntos: primero, cuando el pronóstico de la prensa deportiva pronostica un ganador visitante, es muy raro que el encuentro acabe con un triunfo del visitante (aunque podría terminar en empate). En segundo lugar, si la posición del equipo local (antes del partido) figura entre los nueve primeros de la tabla, por lo general el partido termina con un triunfo local. Con una muestra de 131 partidos, estas observaciones se pueden resumir como dos tablas de frecuencia:

Evidencia	Resultados	
	Ganados local	Ganados visit.
Prensa pronostica triunfo visitante	3	23
Prensa no lo pronostica	60	45
Totales	63	68

Evidencia	Resultados		
	Ganados local	Ganados visit.	Totales
Equipo local entre los nueve primeros	31	25	56
Equipo local en 9.ª posición o peor	32	43	75
Totales	63	68	131

Ahora podemos utilizar estas dos tablas de frecuencia para calcular las "razones de probabilidad" que mencionábamos antes. Por lo general la razón de probabilidad se define del siguiente modo:

$$LR(H|E)=P(E|H)/P(E|\text{not-}H)$$

Para tablas de eventualidades de dos por dos como las de arriba, podemos calcular las razones de probabilidad a partir de una tabla como la siguiente:

Evidencia	Resultados		
	Hipótesis	No hipótesis	Totales
	a	b	(a+b)
	c	d	(c+d)
Totales	(a+c)	(b+d)	(a+b+c+d)

P(E|H)= cantidad de resultados que sustentan resultados Evidencia/Total a favor de la hipótesis; es decir, P(E|H)=a/(a+c). De modo similar, P(E|H)=b/(b+d). Nuestra razón de probabilidad LR (H|E)=a/(a+c)/b/(b+d), con un poco de malabarismo se convierte en:

$$LR(H|E)=(a\ X\ (b+d)/b\ X\ (a+c))$$

Para nuestras dos tablas los resultados son éstos:

Evidencia	LR(H|E) (evidencia triunfo local)
Prensa pronostica triunfo visitante	0,1408
Prensa no pronostica triunfo visitante	1,4392
Evidencia	**LR(H|E) (evidencia gana local)**
Equipo local entre los nueve primeros	1,3384
Equipo local en 9.ª posición o peor	0,8032

Para quienes prefieren las palabras a los números, el último par de números está diciendo que si usted sabe que el equipo local está entre los nueve primeros clasificados de su tabla, las posibilidades a favor de que gane son 1,3384 veces de lo que serían si usted no supiera ese hecho. Pero si el puesto que ocupa en la tabla es el décimo u otro inferior, las posibilidades son sólo 0,8032 veces de lo que habrían sido a favor de un ganador local.

Para ver cómo se utilizan estas razones, vamos a suponer que la primera evidencia es falsa (Pronóstico <> Visitante) y que la segunda es verdadera (Poslocal <10). Comenzamos con las anteriores posibilidades a favor del local, que, dado que hay 63 locales y 68 no locales (empates o visitantes), es 63/68 = 0,9265. Esto corresponde a una probabilidad de 0,4809.

A continuación, multiplicamos las razones de probabilidad adecuadas. La primera evidencia era falsa, de modo que usamos 1,4392, y la segunda era verdadera, por lo que usamos 1,3384. Por tanto:

Posib. ulteriores $=0{,}9265\times1{,}4392\times1{,}3384$
$=1{,}7846$

Podemos volver a convertir a probabilidades con nuestra fórmula P=F/(1+F), de modo que:

Prob. ulteriores $=1{,}7846/2{,}7846$
$=0{,}6409$

Esta cifra dice que hay alrededor de un 64 % de posibilidades de un triunfo local, dado que el pronóstico del diario no es un visitante y que el equipo local está situado en el puesto noveno (o uno mejor) de la tabla. Ambas evidencias eran favorables, y hemos elevado las probabilidades anteriores desde un 48 % a un 64 %. Si un corredor ofrece 6 a 4 o más, vale la pena aceptar la apuesta (pero debido al impuesto sobre las apuestas, usted aquí no puede permitirse aceptar menos que a la par).

La ventaja de la regla de Bayes es que proporciona una base racional para sopesar y combinar distintas evidencias. También es fácil de calcular: una vez que tiene las razones de probabilidad, sólo es cuestión de multiplicar. Además, dado que el resultado es en posibilidades, que se pueden convertir fácilmente a probabilidades, la regla hace muy simple ver si una apuesta es una buena inversión (a diferencia de otros sistemas que producen "números mágicos" o estimaciones de fortaleza).

La trampa es que el método que vemos aquí probablemente producirá estimaciones exageradas si las evidencias están correlacionadas. Si el experto del diario, por ejemplo, toma en cuenta la posición en la tabla para hacer su pronóstico (como es muy probable) entonces las dos evidencias no son auténticamente independientes. Al multiplicarlas, las estamos tratando como si lo fueran.

Así y todo, la regla de Bayes es muy recomendable como hilo para coser entre sí diversos indicadores de un programa de pronósticos. Si usted escribe un programa bayesiano de pronósticos, la mejor forma de minimizar el problema de la evidencia correlacionada es probar uno por uno los nuevos datos, y añadir uno nuevo sólo si mejora el rendimiento global del sistema. Si al incluir una variable plausible ésta hace que el sistema funcione menos bien, es probable que, en efecto, usted esté midiendo dos veces la misma cosa.

La comprobación de los métodos que emplee en una muestra de datos pasados constituye el corazón del enfoque científico a las apuestas. Lamentablemente, esto implica cierto trabajo preliminar, que la mayoría de la gente tiende a saltarse, prefiriendo, en cambio, basarse en la fe ciega. Un poco de sentido común, junto con ciertos principios estadísticos básicos, pueden ayudarlo a inclinar las posibilidades a su favor. El ordenador puede ser de ayuda en este proceso; pero una cosa que no puede darle es moderación. Esta es esencial si ha de mantenerse fiel a sus planes, para enfocar la cuestión científicamente y no apostar hasta estar seguro y preparado.

Interesante bibliografía

He aquí tres obras relacionadas con las apuestas y la informática. *Winning at the races using your computer* (Ganar a las carreras utilizando su ordenador), de Paul Worden, ha sido editada por Interface Publications. Las dos obras de Frank George, *Horse racing with the Commodore 64* (Carreras de caballos con el C64) y *Football pools with the Commodore 64* (Quinielas con el C64) son ediciones Collins

Kevin Jones

He aquí el listado completo de nuestro programa de personajes interactivos, que se ejecutará sin modificación alguna en la gama de ordenadores Amstrad. También incluimos complementos para las máquinas BBC Micro, Spectrum y Commodore 64

Este listado completo incorpora los restantes árboles de decisión que cubren la conciencia de objetos, la actividad general y la interacción de los personajes. El listado está impreso en dos colores: las líneas impresas en negro ya se han publicado, las líneas en verde son las nuevas adiciones.

Hay uno o dos casos de líneas que, habiendo sido publicadas anteriormente, se han modificado para dar cabida a nuestros árboles nuevos; estas líneas están impresas en color rojo.

Cuando ejecute (RUN) el programa, puede entrar el editor de caracteres en cualquier momento pulsando la tecla cero. De lo contrario, pulsando uno, dos o tres se trasladará al jugador a la sala de tertulia, al salón y a la cocina, respectivamente.

Listado del "Dog and Bucket"

Inicialización

Estas líneas preparan las variables y leen datos en las tres matrices diferentes

```
10 REM *bienvenido al Dog an Bucket*
20 REM
30 REM .....inicializar.....
40 REM
45 GOSUB 4030: REM limpia la pantalla
50 DIM 1$(3,5),b$(12,4),c$(7,11),d$(11)
60 r=1
70 PRINT "Valores por defecto (s/n)?": GOSUB
   4110: IF i$="s" OR i$="S" GOTO 90
80 GOSUB 2350:GOTO 100
90 FOR n=1 TO 7: READ c$(n,1): FOR d=2 TO 11:
   READ c$(n,d): NEXT d: NEXT n
100 FOR n=1 TO 3: READ l$(n,1): FOR e=2 TO 5: READ
   l$(n,e): NEXT e: NEXT n
110 FOR n=1 TO 12: READ b$(n,1): FOR e=2 TO 4:
   READ b$(n,d): NEXT d: NEXT n
120 FOR n=2 TO 11: READ d$(n): NEXT n
130 DEF FNb(y,z)=VAL(b$(y,z))
140 DEF FNc(y,z)=VAL(c$(y,z))
150 DEF FNm$(c$,d)=STR$((VAL(c$))-d)
160 DEF FNi$=b$(VAL(c$(c,3)),1)
180 REM prepara arboles
190 DIM t(5,40,4),k(3,30),c(35),s(6),h(6):z=0
200 REM arbol objetos
210 FOR n=1 TO 21: REM 21 nudos elección
220 READ k(1,n),t(1,n,2),t(1,n,1): NEXT n
230 REM arbol trama
240 FOR n=1 TO 22: FOR s=1 TO 4: READ t(2,n,s):
   NEXT s: READ a$: NEXT n
250 REM arbol interaccion personajes
260 FOR n=1 TO 22:FOR s=1 TO 4: READ t(3,n,s):
   NEXT s: READ a$: NEXT n
270 REM arbol actividad general
280 FOR n=1 TO 39: FOR s=1 TO 4: READ t(4,n,s):
   NEXT s: READ a$: NEXT n
290 REM arbol conciencia objeto
300 FOR n=1 TO 13: FOR s=1 TO 4: READ t(5,n,s):
   NEXT s: READ a$: NEXT n
500 REM
510 REM prueba bucle programa
520 REM
530 GOSUB 2100: GOSUB 2150: GOSUB 2240: PRINT:
   PRINT: GOSUB 1000: GOTO 530
1000 REM
1010 REM manipulador personajes
1020 REM
1030 REM comprobar si se ha pulsado tecla
1040 GOSUB 4260: IF i$<> "" THEN GOSUB 2040:
   RETURN
1050 REM procesar cada personaje uno por uno
1060 FOR c=1 TO 6: IF c$(c,9)="7" THEN
   c$(c,9)="0"
1070 IF z=c THEN GOTO 1500
1080 IF FNc(c,10)> 0 THEN
   c$(c,10)=FNm$(c$(c,10),1):GOTO 1500
```

Manipulador de personajes

Las líneas 1000 a 1500 comprueban cada personaje uno por uno y deciden si se deben o no procesar o desplazar desde un escenario a otro

```
1090 REM bandera=0 de modo que
   restablecer bandera y procesar
   personaje
1100 RESTORE: FOR n=1 TO c*10+c-1:
   READ c$(c,10): NEXT n
1110 IF FNc(c,10)=0 THEN GOTO 1500:
   REM valor defecto=0 de modo que no
   procesar
1120       REM comprueba bandera mover
1130 IF FNc(c,11)> 0 THEN c$(c,11)=
   FNm$(c$(c,11),1):GOTO 1190
1140       REM mover bandera=0 de modo
   que poner a cero bandera y trasladar
   personaje
1150 RESTORE: FOR n=1 TO c*11: READ
   c$(c,11): NEXT n
1160 IF c$(c,11)="0" THEN GOTO 1180
1170 GOSUB 2840: GOTO 1500
1180       REM recorrer los arboles
```

Complementos al BASIC

Algunos complementos ya se han publicado en pp. 1988 y 2085, y los lectores habrán de entrarlos cuando sea conveniente. Además, se deben insertar los siguientes complementos:

Spectrum:

```
4350 v=INT(RND*a):
     RETURN
4540 RESTORE 9920: FOR
     e=1 TO t(t,n,4): READ
     h: NEXT e: GOSUB h
4570 RESTORE 9930: FOR
     e=1 TO t(t,n,3): READ
     h: NEXTe: GOSUB h:
     RETURN
5470 RESTORE 9910: FOR
     e=1 TO (t(t,n,1)+1:
     READ h: NEXT e:
     GOTO h
9910 DATA 5480,5490,
     5500,5520,5530,5540,
     5550
9920 DATA 3930,3940,3995
9930 DATA 3030,3060,
     3070,3080,3100,3120,
     3140,3150,3160,3170,
     3180,3190,3200,3210,
     3220,3230,3240,3250,
     3270
```

BBC Micro:

```
4350 v=RND(a): RETURN
```

Commodore 64:

Suprimir de la línea 7000 en adelante e introducir las siguientes modificaciones:

```
 190 DIM t(5,40,4), k(3,30),
     c(35), s(6), h(6),
     i$(23)
 310 FOR n=1 TO 23: READ
     i$(n): NEXT n
4680 m$=i$(t(t,n,4)):
     RETURN
```

```
1190 GOSUB 2400: REM inicializar
     condiciones
1200 FOR t=2 TO 5: GOSUB 2430: GOSUB
     5460: NEXT t
1210 IF FNc(c,4)> 0 THEN GOSUB 5000:
     REM arbol manipulacion objetos
1500 NEXT c: GOTO 1030: REM hacer
     siguiente personaje – cuando todos
     terminados, comprobar pulsación
     tecla/hacerlo otra vez
```

Rutinas de alto nivel

Estas rutinas comprueban personajes y objetos, imprimen sus detalles en la pantalla y llevan a cabo otras funciones de fines generales

```
2000 REM
2010 REM rutina entrada
2020 REM
2030 GOSUB 4110
2040 IF (ASC(i$)< 48) OR (ASC(i$)> 51)
     THEN RETURN
2050 IF i$="0" THEN GOSUB 2320:
     RETURN
2060 r=VAL(i$): c$(7,2)=i$: RETURN
2070 REM
2080 REM impresion escenario
2090 REM
2100 PRINT l$(r,1)
2110 RETURN
2120 REM
2130 REM imprimir objetos visibles
2140 REM
2150 PRINT "Ves: ";
2160 p=0: FOR b=1 TO 12: IF
     VAL(b$(b,2))<>r GOTO 2190
2170 p=p+1: IF p>1 THEN PRINT ", ";
2180 PRINT b$(b,1);
2190 NEXT b
2200 IF p=0 THEN PRINT "nada";
2210 PRINT: RETURN
2220 REM
2230 REM imprime personajes visibles
2240 REM
2250 p=0: FOR c=1 TO 6: IF
     VAL(c$(c,2))<> r GOTO 2290
2260 p=p+1: IF p=1 THEN PRINT "Estas
     en compañia de: ";: GOTO 2280
2270 PRINT ", ";
2280 PRINT c$(c,1);
2290 NEXT c
2300 IF p=0 THEN PRINT "Aqui no hay
     nadie".
2310 RETURN
2320 REM
2330 REM subrutina inicializar personajes
2340 REM
2350 PRINT: h=c: RESTORE: FOR c=1 TO
     7: READ c$(c,1): GOSUB 4070: PRINT
     c$ (c,1);" – ";: PRINT "Establecer este
     personaje?": GOSUB 4110
2360 IF (i$ <> "s") and (I$<>"S") THEN
     FOR n=2 TO 11: READ n$: NEXT n:
     GOTO 2390
2370 FOR d=2 TO 11: READ s$: PRINT
     d$(d);: INPUT i$: IF i$<>"" THEN c$
     (c,d)=i$
2380 NEXT d
2390 NEXT c: c=h: RETURN
2400 REM
2410 REM condiciones
2420 REM
2430 h=FNc(c,8): i=FNc(c,3) :j=FNc(c,6):
     c(1)=ABS(i> o): c(2) = ABS((FNb(j,2)
     = FNc(c,2)) AND (q=1)): c(3) =
     ABS(b$(i,3) = "s"): c(4)=ABS(i=j:
     c(5) = ABS(b$(i,4)= "s")
2440 c(6) = ABS(i=3): c(7) =
     ABS(FNc(c,5)> 5): c(8) =
     ABS(FNc(c,5)> 2): c(9) =
     ABS(VAL(c$(c,9)) = 1): c(10) =
     ABS(FNc(x,3) = 0):c(11)=ABS(FNc)
     (h,2) = FNc(c,2)): c(12) = 255
2450 c(13) = ABS(z = c): c(14) =
     ABS(g=c): c(15) = ABS(z = 0): c(16)
     = ABS(s(c)=255): c(17) =
     ABS(FNc(z,2)=FNc(c,2)): c(18) =
     ABS(h(c) = 255): c(19) = ABS(i = 2)
2460 c(20) = ABS(FNc(x,4)<0):
     c(21)=ABS(z = x): c(22) =
     ABS(FNc(x,3) = FNc(c,6)): c(23) =
     ABS(FNc(x,5)>15): c(24) =
     ABS(FNc(x,9)=7): c(25) =
     ABS(FNc(c,5)>15): c(26) =
     ABS(FNc(c,5)>17)
2470 c(27) = FNc(c,9): c(28) =
     ABS(FNc(c,2) = r): c(29) =
     ABS(FNc(c,2) = 1):
     c(30)=ABS(FNc(c,5)< 5)
2500 RETURN
2510 REM
2520 REM comprueba escenario en busca
     objetos
2530 REM
2540 f=0: REM establecer a cero 'bandera
     hallado'
2550 FOR b=1 TO 12
2560 IF FNb(b,2)=FNc(c,2) THEN
     f=1:b=12
2570 NEXT b
2580 IF f=1 THEN n=3: GOTO 2600
2590 n=39
2600 RETURN
2610 REM
2620 REM comprueba presencia del
     propietario del objeto: si esta presente,
     establece x al numero del personaje:
     salta al arbol
2630 REM
2640 f=0:x=0
2650 FOR m=1 TO 6
2660 IF (FNc(m,2)=FNc(c,2)) AND
     (FNc(m,6)=FNc(c,3)) THEN f=1:
     x=m: GOSUB 2430: m=6
2670 NEXT m
2680 IF f=1 THEN n=15: GOTO 2700
2690 n=39
2700 RETURN
2710 REM
2720 REM selecciona objeto al azar en
     escenario del personaje
2730 REM
2740 b=0
2750 FOR s=1 TO 12
2760 IF FNb(s,2)<> FNc(c,2) THEN GOTO
     2780
2770 GOSUB 4180: IF q=1 THEN b=s:
     s=12
2780 NEXT s
2790 IF b=0 THEN GOTO 2750
2800 RETURN
2810 REM
2820 REM desplaza un personaje
2830 REM
2840 IF FNc(c,4)<1 THEN RETURN: REM
     demasiado debil para moverse
2850 y=0: f=0: FOR w=2 TO 5: IF
     l$(VAL(c$(c,2)),w)="0" THEN GOTO
     2910
2860 GOSUB 4180: IF q=1 THEN f=1:
     c$(c,9)="7": GOTO 2880
2870 GOTO 2910
2880 IF FNc(c,2)=r THEN PRINT c$(c,1);"
     sale de la habitacion...";: y=1
2890 c$(c,2)=1$(VAL(c$(c,2)),w): w=5: IF
     FNc(c,2)=r THEN PRINT c$(c,1);"
     entra en la habitacion...";: y=1
2900 IF y=1 THEN y=c: GOSUB 2250:
     c=y:PRINT: PRINT: REM actualiza
     mensaje personajes presentes
2910 NEXT w: IF f=0 GOTO 2850
2920 RETURN
```

Tablas de acción

Las líneas 3000 a 3999 retienen las rutinas que se llaman durante la ejecución de los distintos árboles de decisión

```
3000 REM
3010 REM tabla de accion
3020 REM
3030 FOR n=1 TO 3: GOSUB 4090: NEXT
     n:m$=c$(c,1)+" vuelve a mirar el
     cuerpo y de pronto comprende la
     espantosa verdad. "+ CHR$(34)+ "La
     empanada está elaborada con alimento
     para gatos"+CHR$(34)+
     " ": GOSUB 4630
3040 GOSUB 4390: m$=m$+ "grita, y en
     una loca carrera las personas reunidas
     se arremolinan sobre la barra y atacan a
     Fred el barman, quien les suplica en
     vano que tengan piedad de su
     miserable vida.": GOSUB 4630: GOSUB
     4720
3050 FOR n=1 TO 2000: NEXT n:GOSUB
     4720: m$="...y al dia siguiente a Fred
     el barman no se le ve por ninguna
     parte. No obstante, hay muchisimas
     empanadas de extraña forma para
     alimentar a la siempre famelica clientela
     del Dog and Bucket...": GOSUB 4630:
     GOSUB 4720: END
3060 h(c)=255: GOSUB 4680:
     m$=c$(c,1)+ m$: GOSUB 4630:
     GOSUB 4720: RETURN
3070 z=c: GOSUB 4680: m$=c$(c,1)+m$:
     GOSUB 4300: m$=m$+m$+
     "estomago, e inmediatamente muere.
     Los otros personajes estan demasiado
     concentrados en sus bebidas como
     para darse cuenta...": GOSUB 4630:
     GOSUB 4720: g=0: c$(c,4)="-1":
     RETURN
3080 c$(c,4)="10": g=0:IF f(t,n,4)>0
     THEN GOSUB 4680: m$=c$(c,1)+m$:
     GOSUB 4630: GOSUB 4720
3090 RETURN
3100 a=20: GOSUB 4350:IF w<> 5 THEN
     RETURN
3110 GOSUB 4680: GOSUB 4630: GOSUB
     4720: RETURN
3120 a=2: GOSUB 4350: IF w<> 0 THEN
     RETURN
3130 m$=c$(c,1)+ "intenta
     reanimar"+c$(x,1)+ "sin
```

```
conseguirlo...": GOSUB 4630: GOSUB
4720: RETURN
3140 GOSUB 4680: m$ = CHR$(34)
+m$+CHR$(34)+" dice"+c$(c,1)+"
a"+c$(x,1): GOSUB 4630: GOSUB
4720: RETURN
3150 GOSUB 4680: m$=CHR$(34)
+m$+CHR$(34)+ ", "+c$(c,1)+"
pregunta" +c$(x,1): GOSUB 4630:
GOSUB 4720: RETURN
3160 GOSUB 4680: m$=c$(c,1)+" y
"+c$(x,1)+m$:GOSUB 4630: GOSUB
4720: c$(c,5)="10": RETURN
3170 GOSUB 4680: m$ =
CHR$(34)+m$+CHR$(34) +
"dice"+c$(c,1)+ "a "+c$
(x,1):GOSUB 4630: GOSUB 4720:
RETURN
3180 GOSUB 4680: m$=c$(c,1)+m$:
GOSUB 4630: GOSUB 4720:
c$(c,5)+"5": RETURN
3190 GOSUB 4680: m$=c$(c,1)+m$:
GOSUB 4630: GOSUB 4720: RETURN
3200 GOSUB 4680: m$ = CHR$(34)
+m$+CHR$(34)+" dice"+c$(c,1)+"
afligi-
damente...": GOSUB 4630: GOSUB
4720: c$(c,5)="10": RETURN
3210 GOSUB 4680: m$=CHR$(34)
+m$+CHR$(34)+" dice "+c$(c,1):
GOSUB 4630: GOSUB 4720: c$
(c,9)="3": RETURN
3220 x=FNc(c,8): m$=c$(c,1)+" y"
+c$(x,1)+ " estan enfrascados en una
conversacion.":GOSUB 4630: GOSUB
4720: x=0: RETURN
3230 x=FNc(c,3): IF x>0 THEN GOSUB
4680: m$=c$(c,1)+ "examina
ebriamente "+ b$(x,1)+ m$: GOSUB
4630: GOSUB 4720: GOSUB 4220: x=0
3235 RETURN
3240 GOSUB 4680: m$ = CHR$(34)
+m$+CHR$(34)+" se
queja"+c$(c,1): c$(c,5)="4": GOSUB
4630: GOSUB 4720: GOSUB 4220:
RETURN
3250 GOSUB 4680: m$ =
CHR$(34)+m$+CHR$(34) + "grita
"+ c$(c,1)+", mirando la ": GOSUB
4630: IF FNc(c,3)=3 THEN
m$="empanada.": GOSUB 4630:
GOSUB 4720: GOSUB 4220: RETURN
3260 m$="bocadillo.": GOSUB 4630:
GOSUB 4720: GOSUB 4220: RETURN
3270 GOSUB 4680: m$=c$(c,1)+m$:
GOSUB 4630: GOSUB 4630:
m$="bebida": GOSUB 4630: GOSUB
4720: RETURN
3900 REM
3910 REM gosub tabla
3920 REM
3930 s(c)=255: m$=c$(c,1)+" se arrodilla
junto al cuerpo postrado de "+
c$(z,1)+". La horrible verdad toma
cuerpo lentamente, pero los demas
parecen estar demasiado borrachos
como para prestarle ninguna atencion
inmediata...": GOSUB 4630: GOSUB
4720: RETURN
3940 f=0: x=0: FOR y=1 TO 6
3950 IF y=c THEN GOTO 3970
3960 IF FNc(y,2)=FNc(c,2) THEN f=1:
GOSUB 4180:IF q=1 THEN x=y:
GOSUB 2430: y=6
3970 NEXT y: IF f=0 THEN t(3,1,4)=2:
RETURN
3980 IF x=0 THEN GOTO 3940
3990 RETURN
3995 c$(c,5)=FNm$(c$(c,5),1):
RETURN
```

Rutinas de bajo nivel

Sacan datos a la pantalla y se ocupan de otras funciones de bajo nivel, tales como hacer sonar un BEEP y tomar caracteres del teclado

```
4000 REM
4010 REM subrutinas del sistema de bajo
nivel
4020 REM
4030 REM limpia la pantalla
4040 REM
4050 CLS: RETURN
4060 REM
4070 REM beep
4080 REM
4090 PRINT CHR$(7);: RETURN
4100 REM
4110 REM tomar un caracter del teclado
4120 REM
4130 i$=INKEY$: IF i$="" GOTO 4130
4140 RETURN
4150 REM
4160 REM rutina numero aleatorio
4170 REM
4180 q=INT(RND(1)*2)+1: RETURN
4190 REM
4200 REM poner a cero codigos personajes
4210 REM
4220 c$(c,8)="0": c$(c,9)="0": RETURN
4230 REM
4240 REM comprobar si tecla pulsada
4250 REM
4260 i$=INKEY$: RETURN
4320 REM
4330 REM rutina numero aleatorio variable
4340 REM
4350 v=INT(RND(2)*a): RETURN
4360 REM
4370 REM imprimir el/ella
4380 REM
4390 IF c$(c,7)="m" THEN m$="ella ":
RETURN
4400 m$="él ": RETURN
4500 REM
4510 REM bloque de saltos
4520 REM
4530 REM nudos reentrantes
4540 ON t(t,n,4) GOSUB 3930,3940,3995
4550 RETURN
4560 REM nudos de accion
4570 ON t(t,n,3) GOSUB 3030, 3060, 3070,
3080, 3100, 3120, 3140, 3150, 3160,
3170, 3180, 3190, 3200, 3210, 3220,
3230, 3240, 3250, 3270: RETURN
4600 REM
4610 REM imprimir mensajes si el jugador
esta presente
4620 REM
4630 IF FNc(c,2)=r THEN PRINT m$;
4640 m$="": RETURN
4650 REM
4660 REM selecciona un mensaje de la
sentencia data
4670 REM
4680 RESTORE 7030: FOR m=1 TO t(t,n,4):
READ m$: NEXT m:RETURN
4690 REM
4700 REM imprime una linea en blanco
4710 REM
4720 IF FNc(c,2)=r THEN PRINT: PRINT
4730 RETURN
```

Recorrido del árbol

Las líneas 5000 a 5999 clasifican los distintos árboles. Aquí también se incluyen las rutinas para el árbol de manipulación de objetos

```
5000 REM rutinas arbol objetos
5010 p=0: REM poner a cero bandera
imprimir
5020 IF FNc(c,2)=r THEN p=1
5030 n=1: REM empezar en el nudo 1
5040 IF n> 21 GOTO 5070
5050 k=c(k(1,n))+1: IF k(1,n)=12 THEN
GOSUB 4180: k=q
5060 n=t(1,n,k): GOTO 5040
5070 IF n>=24 GOTO 5090
5080 ON (n-21) GOSUB 2540,2640: GOTO
5040
5090 ON (n-23) GOTO
5100,5130,5160,5180,
5210,5240,5260,5270,5280,5300,
5310,5330,5340,5360,5370,5430
5100 GOSUB 2740: c$(c,3)=STR$(b)
5110 IF p=1 THEN PRINT c$(c,1);"
recoge";b$(b,1): PRINT
5120 b$(b,2)="0": c$(c,9)="4": RETURN
5130 c$(c,3)=c$(c,6)
5140 IF p=1 THEN PRINT c$(c,1);"
recoge";FNi$: PRINT
5150 b$(VAL(c$(c,3)),2)="0":
c$(c,9)="4": RETURN
5160 IF p=1 THEN PRINT c$(c,1);" bebe un
sorbo de";FNi$: PRINT
5170 c$(c,4)=FNm$(c$(c,4),-1): RETURN
5180 GOSUB 4180: IF (p=1) AND (q=1)
THEN PRINT c$(c,1);" se esta
comiendo el bocadillo.": PRINT
5190 c$(c,4)=FNm$(c$(c,4),-2):c$
(c,9)="6": GOSUB 4180: IF q=
1 THEN GOSUB 4220
5200 RETURN
5210 IF p=1 THEN PRINT c$(c,1);" muerde
un bocado de la empanada, gruñe y la
tira al suelo.": PRINT
5220 g=c:REM establece bandera empanada
comida
5230 c$(c,3)="0":
c$(c,4)=FNm$(c$(c,4),10):
b$(3,2)=c$(c,2): RETURN
5240 IF p=1 THEN PRINT c$(c,1);"
deja";FNi$: PRINT
5250 b$(VAL(c$(c,3)),2)=c$(c,2):
c$(c,3)="0": RETURN
5260 c$(c,5)=FNm$(c$(c,5),-1): RETURN
5270 GOSUB 5240: RETURN
5280 IF p=1 THEN PRINT c$(c,1);"
tira";b$(VAL(c$(c,3)),1);" a ";c$ (x,1):
PRINT
5290 c$(x,4)=FNm$(x,4),1):
b$(VAL(c$(c,3)),2)=c$(c,2):
```

```
      c$(x,8)=STR$(c): c$(x,9)="5":
      c$(c,3)="0": RETURN
5300  GOSUB 4220: RETURN
5310  IF p=1 THEN PRINT "Creo que tengo
      tu bebida, dice ";c$(c,1);" a ";c$(x,1):
      PRINT
5320  c$(c,8)=STR$(x): c$(c,9)="2":
      RETURN
5330  c$(c,4)=FNm$(c$(c,4),2): RETURN
5340  IF p=1 THEN PRINT c$(c,1);"le da
      ";FNi$;" a ";c$(x,1): PRINT
5350  c$(x,3)=c$(c,3): c$(c,3)="0":
      c$(c,8)=STR$(c): c$(c,9)="1":
      RETURN
5360  GOSUB 4220: RETURN
5370  IF p=0 GOTO 5420
5380  IF p=1 THEN PRINT c$(c,1);" con voz
      ebria le agradece ";c$(VAL(c$
      (c,8)),1);" que le haya devuelto su
      bebida ";
5430  RETURN
5440  REM
5450  REM clasifica arboles
5460  n=1
5470  ON (t(t,n,1)+1) GOTO
      5480,5490,5500,
      5520,5530,5540,5550
5480  k=c(t(t,n,2))+1: n=t(t,n,2+ k):
      GOTO 5470
5490  GOSUB 4530: n=t(t,n,3): GOTO 5470
5500  GOSUB 4570
5510  RETURN
5520  GOSUB 4680: GOSUB 4630: GOSUB
      4720
5530  RETURN
5540  a=t(t,n,4): GOSUB 4350:
      n=t(t,n,3)+v: GOTO 5470
5550  k=c(t(t,n,2)): n=t(t,n,3)+k: GOTO
      5470
```

Almacén de datos

De la línea 6000 en adelante se retienen los datos para las diversas matrices, así como los mensajes utilizados por los módulos del programa

```
6000  REM
6010  REM datos personajes
6020  REM
6030  DATA "Luis Cubas","2","7","10",
      "10","7","v","0","0","1","4",
      "Lola Fiestas","1","8","30","10",
      "8","m","0","0","1","5","Pepe
      Viñas","1","9","8","10","9",
      "v","0","0","1","6","Mari
      Tapas","2","0","20","10","10",
      "m","0","0","1","5"
6040  DATA "Javi Salado","2","11","10",
      "6","11","v","0","0","1","6",
      "Gina Fizz","1","12","15","6",
      "12","m","0","0","1","5","tu",
      "1","0","255","255","0","v",
      "0","0","0","0"
6050  REM
6060  REM datos escenarios
6070  REM
6080  DATA "estas en la sala de tertulia del
      Dog and Bucket. En un rincón hay un
      grupo de dudosos personajes jugando
      al domino. Detras del mostrador esta
      Fred el barman, con su habitual talante
      festivo. Hay salidas al este.","0","0",
      "2","0"
6090  DATA "He aqui el salon del Dog and
      Bucket, al cual le vendría muy bien una
      redecoración completa. El suelo parece
      haber sido regado regularmente con
      cerveza. Hay puertas al oeste y al sur.",
      "0","3","0","1"
6100  DATA "¡Aj! Esta es la cocina, donde se
      preparan las famosas empanadas de
      carne del Dog and Bucket, para una
      clientela siempre famelica. Puedes
      observar algunas latas vacias de
      alimento para gatos, lo cual no deja de
      ser extraño, ya que no hay ningun
      gato.","2","0","0","0"
6110  REM
6120  REM datos objetos (para b$(12,4))
6130  REM
6140  DATA "un vaso de cerveza","2","n",
      "s"," una lata vacia de alimento para
      gatos","3","n","n"," una
      empanada de carne Dog and Bucket",
      "1","s","n","una banqueta de bar",
      "2","n","n","un cenicero","1",
      "n","n"
6150  DATA " un bocadillo de jamon rancio",
      "2","s","n"," una pinta de cerveza
      amarga","0","n","s"," una crema
      de menta","0","n","s","un whisky
      con soda","0","n","s"," un vodka
      puro","2","n","s"," una pinta de
      cerveza añeja","0","n","s"," una
      ginebra con ginger ale","0","n","s"
6160  REM
6170  REM datos atributos personajes (para
      d$(11))
6180  REM
6190  DATA "Escenario","Fortaleza",
      "Inventario","Humor","Objeto
      propio","Sexo","Ultimo pers (Ich)",
      "Codigo ultima instruccion (Icd)",
      "Frecuencia de manipulación",
      "Frecuencia de movimiento"
6200  REM
6210  REM datos arbol objetos
6220  REM
6230  DATA 1,2,22,12,5,4,2,7,6,3,9,8,4,11,
      10,12,39,24,12,6,25,5,12,39,
      6,13,27,12, 23,29,12,30,14, 12,26,
      39,12,28,39,12,31, 17,7,18,
      16,8,39,19,9,21,
      39,10,36,20,12,33,32,12,
      35,34,11,38,37
6240  REM
6250  REM datos arbol trama
6260  REM
6270  DATA 0,13,2,22," ",0,14,5,3,"
      ",0,15,21,4," ",5,0,18,3,"
      ",0,16,6,7," ",0,17,7,11,"
      ",0,18,9,8," ",0,17,12,13,",
      "0,19,10,17," ",5,0,14,3,"
      ",1,0,7,1," ",4,0,0,0," ",2,0,1,0,"
      ",2,0,5,1," ",4,0,0,0," ",4,0,0,0,"
      ",2,0,2,4," ",2,0,3,2," ",2,0,4,3
6280  DATA " ",2,0,4,0," " 4,0,0,0,"
      ",4,0,0,0," "
6300  REM arbol interaccion
6320  DATA 1,0,3,2," ",4,0,0,0,"
      ",0,20,6,4," ",0,21,7,5," ",4,0,0,0,"
      ",5,0,8,3," ",2,0,6,0," "0,22,11,14,"
      ",0,23,12,17," ",0,24,13,18,"
      ",4,0,0,0," ",4,0,0,0," ",4,0,0,0,"
      ",5,0,15,2," ",4,0,0,0," ",2,0,7,5,"
      ",0,25,22,21," "
6330  DATA 5,0,19,2," ",4,0,0,0,"
      ",2,0,8,6," ",2,0,9,7," ",2,0,10,8,"
      "
6340  REM arbol actividad general
6350  DATA 5,0,2,5," ",0,26,11,7,"
      ",6,27,24,7," ",0,28,8,12,"
      ",0,29,9,15," ",5,0,21,3,"
      ",2,0,11,9," ",4,0,0,0," ",4,0,0,0,"
      ",4,0,0,0," ",0,30,10,18,"
      ",5,0,13,2" ",2,0,12,10," ",4,0,0,0,"
      ",5,0,16,2," ",3,0,0,11,"
      ",3,0,0,12,""
6360  DATA 5,0,19,2," ",4,0,0,0,"
      ",2,0,13,13," ",2,0,14,14,"
      ",2,0,14,15," ",2,0,14,16,"
      ",4,0,0,0," ",0,11,31,32,"
      ",4,0,0,0," ",5,0,33,3,"
      ",0,11,36,37," ",5,0,38,2,"
      ",2,0,14,17," "
6370  DATA 4,0,0,0," ",2,0,15,0,"
      ",4,0,0,0," ",2,0,16,18," ",4,0,0,0,"
      ",4,0,0,0," ",2,0,17,19," ",4,0,0,0,"
      ",2,0,18,20," "
6380  REM arbol conciencia objeto
6390  DATA 0,4,8,2," ",5,0,3,5,"
      ",4,0,0,0," ",4,0,0,0," ",4,0,0,0,"
      ",4,0,0,0," ",2,0,14,21," ",1,0,9,3,"
      ",5,0,10,4," ",2,0,19,22,"
      ",2,0,14,23," ",4,0,0,0," ",4,0,0,0,"
      "

7000  REM
7010  REM datos mensajes
7020  REM
7030  DATA "Un extraño color invade el
      aire...¿podria ser el aroma de Catty-Kit
      A La Carte? "," de pronto se desploma
      sobre el suelo apretandose el estomago
      "," parece muy enfermo, y advierte a
      los demas que no toquen la
      empanada."
7040  DATA " examina atentamente la lata, y
      su semblante adquiere un aire
      pensativo. "," ¿Que estas haciendo
      con mi bebida? "," ¿Donde has
      estado?"
7050  DATA ", en un arrebato propio de
      borrachos, saltan sobre la mesa y se
      ponen a bailar."
7060  DATA "Pareces alegre","trepa a gatas
      sobre la barra, intenta bailar y vuelve a
      caer."
7070  DATA ", con el estupor de la
      borrachera, de pronto levanta la vista,
      se asoma por la pantalla de la VDU y te
      mira a ti...","Fred el barman sirve otra
      pinta...","Fred
      esta ocupado lavando vasos"
7080  DATA "Es una vida de perros..."
      ,"No he venido a enterrar a
      Cesar...","Deja que te cuente la
      historia de mi vida...","Barman,
      lleneme la copa!"
7090  DATA "Hola a todos..."," como si
      ocultara el secreto de la vida.","Para
      que has hecho eso!!","Creo que me
      voy a poner enfermo!"
7100  DATA "Ahh...Este trago esta
      sublime..."," busca
      desesperadamente","¿Quien tiene mi
      bebida?"
```

Cambio de tecla

Estudiaremos las distintas maneras de realizar cambios en el teclado del Amstrad desde su sistema operativo

El sistema operativo del Amstrad tiene una sección, denominada *gestor de teclas*, que controla todos los accesos al teclado y a la barra de mando. La auténtica fuerza del gestor de teclas reside en el hecho de que el teclado es puro *soft*. Esto significa que el código obtenido de cada tecla física es definible por el usuario. Esto facilita la implantación de programas tales como el CP/M, dado que podemos adaptar las funciones de las teclas según convenga para que den los códigos deseados en lugar de modificar el programa para que acepte los códigos producidos por omisión.

Hay dos tipos de código obtenible al pulsar una tecla: un carácter único ASCII o uno indicativo (*token*) de ampliación. El indicativo de ampliación puede ser considerado como un flag que sirve para obtener una cadena de caracteres, y no un simple carácter, a través de la simple pulsación de una tecla.

El teclado es revisado cada cincuentavo de segundo en el reloj para uso general. Un mapa de estado de la tecla sirve para registrar qué teclas han sido pulsadas en la última revisión. Este mapa se emplea para asegurarse de que las teclas no sean leídas más de una vez y para determinar si se ha de repetir un carácter o no. También se dispone de un buffer que almacena los códigos que se obtienen de las teclas en el momento en que fueron detectadas; tales códigos pueden ser interpretados bien como caracteres ASCII, bien como indicativos de ampliación.

El gestor de teclas

Se puede acceder al gestor de teclas en varios niveles distintos. El nivel más bajo permite al usuario comprobar si se está pulsando una tecla física en ese momento.

El segundo nivel presenta el carácter único asignado a la tecla. Este carácter puede ser bien un valor ASCII, bien un indicativo de ampliación. Si se trata de un indicativo de ampliación, entonces es el usuario el que debe determinar la interpretación que se le ha de dar. Este nivel tiene en cuenta el estado actual de las teclas Shift y Control a la hora de determinar qué carácter ha de obtenerse.

El nivel más alto da simplemente el carácter siguiente que se ha pulsado en el teclado. Este carácter puede corresponder tanto a una pulsación de tecla única como puede ser el carácter siguiente de un indicativo de ampliación.

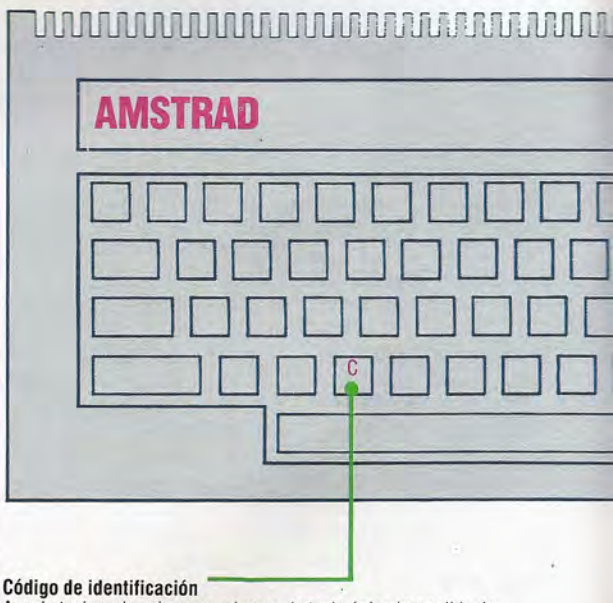
Asociadas a cada tecla existen tres tablas de traducción que determinan el código obtenido cuando se detecta una tecla pulsada en sus estados normal, *shift* o *control*. Cada una de estas entradas puede ser determinada y establecida individualmente, empleando las entradas detalladas en el diagrama. Obsérvese que los caracteres que van desde &E0 a &FC tienen un significado especial bajo BASIC, por lo que si un programa asigna una tecla para obtener uno de estos indicativos, se tendrá que reasignarlos antes de volver al BASIC.

El listado primero muestra cómo se puede establecer un conjunto de traducciones al entrar en la rutina, al mismo tiempo que se guardan sus estados iniciales.

Indicativos de ampliación

Los códigos del &80 al &9F son los denominados *indicativos de ampliación (expansion tokens)*. Cuando son encontrados se almacenan en el buffer a menos que no sean extraídos por medio de KM_READ_CHAR o KM_WAIT_CHAR. En este caso el gestor de teclas trata de determinar qué cadena se asigna a ese indicativo de ampliación e inserta los caracteres en secuencia dentro del buffer hasta el final de la cadena.

Gestión del teclado



Código de identificación
A cada tecla se le asigna un número de tecla único imposible de alterar por el firmware. El código ASCII que se obtiene mediante una tecla se determina a través de tres tablas, que proporcionan el código de la tecla en su estado normal, de *shift* o de *control*

Caroline Clayton

Las cadenas de ampliación se almacenan en un buffer, que puede ser de cualquier tamaño según el número de cadenas requerido. Cada cadena puede componerse de un máximo de 255 caracteres siempre que haya espacio suficiente en el buffer, el cual debe hallarse en los 32 K centrales de la RAM. El tamaño del buffer y su posición se establecen por medio de KM__EXP__BUFFER.

Cada indicativo de ampliación puede tener una cadena asignada mediante KM__SET__EXPAND. Los caracteres dentro de una cadena no son comprobados por el firmware, de modo que es posible obtener caracteres tales como &FC (el indicativo para Break) para una rutina. Las cadenas asignadas a un indicativo de ampliación sólo pueden ser examinadas, carácter tras carácter, mediante la entrada KM__GET__EXPAND.

Cada tecla en el teclado tiene un número asociado a ella. Este número es único y no puede alterarse por medio del firmware, obteniéndose así un modo absoluto de referencia a la tecla. Los números de tecla se detallan en el Apéndice III del manual de firmware y también se proporcionan con mucho esmero en los cartuchos de las unidades de disco del CPC 664 y del 6128.

La entrada KM__TEST__KEY es la entrada de nivel más bajo dentro del gestor de teclas. Permite al usuario averiguar si una determinada tecla ha sido mantenida en el último rastreo del teclado, a excepción de las teclas Control y Shift. Esta entrada es especialmente útil cuando se comprueba una sola tecla; por ejemplo, cuando el teclado ha sido sondeado periódicamente para que la tecla Escape dé a entender que ha de concluirse el programa.

También se proporciona una entrada exclusivamente para las palancas de mando: KM__GET__JOYSTICK. Esta entrada lee los estados de la palanca de mando en el mapa de estado de las teclas y es el método más rápido de determinar su estado actual. Por ello es especialmente apropiado para los programadores de juegos, dado que las palancas de mando deben ser leídas en todo momento con independencia de que se hayan alterado o no sus posiciones.

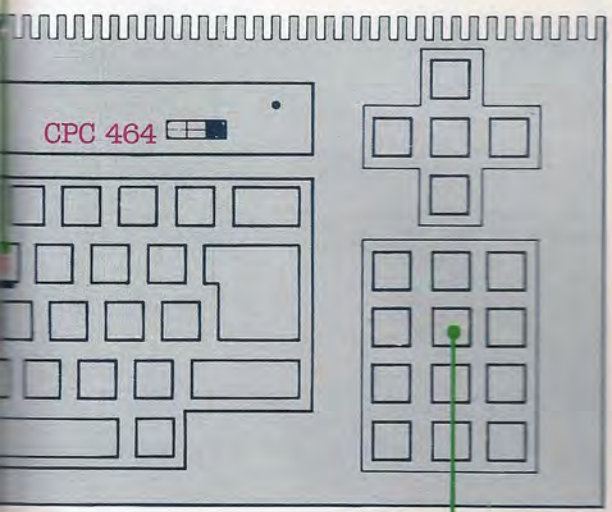
Hay dos entradas del gestor de teclas en el segundo nivel, KM__READ__KEY y KM__WAIT__KEY. La entrada KM__WAIT__KEY se emplea para dar la siguiente pulsación de tecla. Si existe una entrada en el buffer del teclado, entonces se traduce bien en un carácter, bien en un indicativo de ampliación y se retorna inmediatamente; en otro caso, la rutina espera la siguiente pulsación de tecla y da su valor traducido. KM__READ__KEY también comprueba el carácter en el buffer de teclado, y si sólo hay uno, éste se traduce; de otro modo, la rutina retorna inmediatamente, indicando mediante un flag que el buffer está vacío. KM__WAIT__KEY encuentra un uso apropiado cuando un programa no puede continuar hasta que haya recibido un input del usuario (p. ej., cuando se requiere una opción de menú). KM__READ__KEY deberá usarse cuando se sondea el teclado respecto a la pulsación de una tecla.

El tercer nivel tiene dos entradas correspondientes: KM__READ__CHAR y KM__WAIT__CHAR. La única diferencia entre éstas y las del segundo nivel está en que sólo pueden obtenerse caracteres. Si se encuentra un indicativo de ampliación, la cadena de ampliación correspondiente es extraída carácter a carácter antes de ser examinada la entrada siguiente en el buffer del teclado.

Direcciones útiles

&BB27	KM−SET−TRANSLATE	est. una entr. normal en la tabla de trad. de teclas
&BB2A	KM−GET−TRANSLATE	lee la entr. normal actual en la tabla de trad. de teclas
&BB2D	KM−SET−SHIFT	establece una entr. en tabla de trad. para una tecla con *shift*
&BB30	KM−GET−SHIFT	lee la entr. en la tabla de trad. para una tecla con *shift*
&BB33	KM−SET−CONTROL	est. una entr. en la tabla de trad. para la tecla de control pulsada
&BB36	KM−GET−CONTROL	lee la entr. en la tabla de trad. para la tecla de control pulsada

...ay sensible
...firmware mantiene un mapa de estado de las teclas que ...egistra las teclas pulsadas en el momento de la interrupción del ...oj para uso general (cada cincuentavo de segundo). Los ...ores de la tecla se almacenan en un buffer y el firmware ...rmite bien comprobar la pulsación de una tecla o bien esperar ...a pulsación



Indicativos de ampliación
Los códigos ASCII desde el &80 al &9F son tratados por el firmware como *indicativos de ampliación*. Por omisión, estos códigos se asignan a las teclas en el cuaderno numérico de teclas. A cualquiera de estos indicativos se les pueden asignar cadenas de ampliación. Éstas pueden tener hasta 255 caracteres de longitud y contener códigos de control

Prioridad de carácter

El gestor de teclas tiene, además del buffer de teclado, un buffer de un solo byte conocido por *prio-*

ridad de carácter (*putback character*). Este buffer es consultado antes que el buffer principal y si contiene un carácter, se le da prioridad. Esto permite colocar un carácter en cabeza dentro del buffer que puede ser utilizado, por ejemplo, para señalar como flag que ha ocurrido un determinado evento. El gestor de teclas sobreescribe este carácter prioritario cuando se detecta un Break, ya que el indicativo de Break se inserta en calidad de carácter prioritario.

El diagrama muestra los distintos niveles en los que puede obtenerse un carácter desde el gestor de teclas.

Cada tecla del teclado tiene asociados tres parámetros de repetición. El primero determina si se puede o no repetir esa tecla; el segundo especifica la demora entre el momento de detectar la pulsación de la tecla y el de permitir su repetición; el tercero es el intervalo de cada repetición. Las demoras son especificadas en múltiplos de rastreos de teclado y por ello corresponden a un intervalo de un cincuentavo de segundo.

Las determinaciones actuales de demoras pueden ser realizadas por KM__GET__DELAY y pueden alterarse mediante KM__SET__DELAY. Las demoras son válidas para todo el teclado, por lo que no es posible definir valores específicos para las distintas teclas.

El parámetro que determina si una tecla pueda repetirse o no es leído y variado mediante KM__GET__REPEAT y KM__SET__REPEAT.

Interrupciones

La tecla Escape es tratada por el gestor de teclas de modo distinto a las demás teclas. Cuando es pulsada, se llama la dirección de KM__TEST__BREAK. Esta rutina comprueba si también se pulsaron las teclas Control y Chift, en cuyo caso se realiza una restauración; de otro modo, la rutina retorna. Parcheando esta dirección con la instrucción RET es posible, entonces, evitar que dentro de un programa se produzca una restauración.

Hay también un evento que se proporciona para tratar las interrupciones. Este evento puede ser inicializado activándolo o desactivándolo a través del firmware.

Códigos especiales

&80–&9F	Indicativos de ampliacion. &80F corresponde a la cadena de ampliacion 0, y &9F a la cadena de ampliacion 32
&E0–&FC	Codigos control cursor BASIC
&FD	Conmutador mayusculas *on/off* Cambia el estado actual de la tecla mayusculas
&FF	Conmutador de Shift *on/off* Cambia el estado actual de la tecla Shift
&FF	Indicativo de ignorar Este caracter sera ignorado si se retorna desde el gestor de teclas

Traducción de teclas

Este listado proporciona dos rutinas para permitir que un programa pueda configurar el teclado de modo que provea un conjunto predefinido de códigos durante un programa y devolverlo a su estado inicial a la salida. La primera rutina, Setkeys, se llamará al entrar en el programa. Ésta rastrea una tabla que contiene una lista de números de tecla y la traducción que debe tener. El estado actual de las teclas se lee y almacena e la tabla antes de asignar un nuevo valor a una tecla. La segunda rutina lee las traducciones originales en la tabla volviendo a asignarlas a las teclas.

La tabla se establece aquí para configurar las teclas Cursor y Escape para ser utilizadas en un proceso de textos, pero puede ser ajustada a aplicaciones individuales

```
                ;SETKEYS
                get__tr:  equ  #BB2A
                set__tr:  equ  #BB27
                tend:     equ  #FE
21B329          setkey:   ld   hl,tnormal        ;establece tabla trad norm
7E              setlp:    ld   a,(hl)            ;toma numero tecla
FEFE                      cp   tend              ;fin de tabla?
C8                        ret  z
23                        inc  hl
46                        ld   b,(hl)            ;toma nueva trad
23                        inc  hl
EB                        ex   de,hl             ;guarda hl
F5                        push af                ;guarda numero tecla
CD2ABB                    call get__trans        ;lee trad actual
EB                        ex   de,hl
77                        ld   (hl),a            ;guarda trad antigua
23                        inc  hl
EB                        ex   de,hl
F1                        pop  af
CD27BB                    call set__trans        ;guarda nueva tecla
EB                        ex   de,hl
18E9                      jr   setlp             ;continua
                ;
                ;GETKEYS
                ;
21B329          getkey:   ld   hl,tnormal        ;restaura tabla norm
7E              getlp:    ld   a,(hl)            ;toma numero tecla
FEFE                      cp   tend
C8                        ret  z
23                        inc  hl
23                        inc  hl                ;apunta a area guardado
46                        ld   b,(hl)            ;toma trad antigua
EB                        ex   de,hl
CD27BB                    call set__trans        ;restaura la antigua
EB                        ex   de,hl
23                        inc  hl
18F1                      jr   getlp
                ;
                ;tablas traduccion teclas – almacenamiento – teclas – traduccion
                ;
421B00          tnorma:   defb 66,27,0           ;ESCAPE
000B00                    defb 0,11,0            ;CURSOR UP
020A00                    defb 2,10,0            ;CURSOR DOWN
080800                    defb 8,8,0             ;CURSOR BACK
010900                    defb 1,9,0             ;CURSOR FORWARD
FE                        defb tend              ;end
```

Por omisión, el evento está desactivado, de modo que se habrá de crear una rutina especial que puede llamarse cada vez que se detecta una pulsación de la tecla Break.

Si se devuelve el control después de llamar KM__TEST__BREAK, el gestor del teclado comprueba si el evento de *break* está desactivado. Si lo está, en el buffer se inserta el carácter &EF, y se dispara el evento de *break*. El puntero permite que el evento se nivele con el buffer del teclado hasta el punto donde se encontró el *break*.

Con este análisis del gestor de teclas damos por concluida nuestra serie sobre el sistema operativo del Amstrad CPC.