

ZX Spectrum +2

sinclair



Manual del Usuario



ZX Spectrum +2

MANUAL DEL USUARIO



Introducción

Sinclair ZX Spectrum +2 Ordenador doméstico de 128K

Una culminación

Es motivo de gran satisfacción poder presentar este nuevo ordenador, el ZX Spectrum +2, en el que culmina una serie que tantos éxitos ha cosechado: el Spectrum original, el Spectrum + y el Spectrum 128.

El nuevo ZX Spectrum +2 es una máquina en la que se combina todo el ingenio de la tecnología Sinclair con la experiencia de Amstrad en materia de integración y fiabilidad.

Compatibilidad de los programas

El +2 puede funcionar con los programas escritos para modelos anteriores de la serie ZX Spectrum. Esto representa que ya hay una inmensa variedad de programas disponibles para el +2; literalmente miles de títulos que cubren todas las aplicaciones imaginables: juegos, utilidades, música, programas científicos y educativos, etc.



Cuando vaya a comprar un programa, asegúrese de que en la carátula lleva impreso el logotipo 'SINCLAIR QUALITY CONTROL'.

Compre solamente programas de las marcas que se hayan adherido a este sistema, cuyo objetivo es garantizar al usuario la compatibilidad de los programas.

Acerca de este libro

Este libro ha sido concebido solamente como guía introductoria a la utilización del +2. Si usted necesita información más detallada o de nivel superior, puede consultar alguno de los numerosos libros que han sido publicados sobre los modelos Spectrum + y Spectrum 128, muchos de los cuales son excelentes y proporcionan toda la información que se puede necesitar acerca de los ordenadores ZX Spectrum y sobre Sinclair BASIC.

Sin embargo, el presente libro será perfectamente adecuado para sus necesidades si lo que usted desea es instalar el ordenador, conectar los periféricos, cargar programas o aprender los fundamentos de la programación en BASIC.

AMSTRAD

CONSUMER ELECTRONICS PLC

© Copyright 1986 AMSTRAD Consumer Electronics plc

El contenido de este manual y el producto en él descrito no pueden ser adaptados ni reproducidos, ni total ni parcialmente, salvo con el permiso escrito de AMSTRAD Consumer Electronics plc ('Amstrad').

El producto descrito en este manual, así como los diseñados para ser utilizados con él, están sujetos a desarrollo y mejoras continuas. Toda la información técnica relativa al producto y su utilización (incluida la que figura en este manual) es suministrada por AMSTRAD de buena fe.

Toda reparación u operación de mantenimiento de este producto debe ser confiada a los distribuidores autorizados de Sinclair. AMSTRAD no puede asumir ninguna responsabilidad derivada del daño o pérdida que se pueda ocasionar como resultado de reparaciones efectuadas por personal no autorizado. El objetivo de este manual no es sino servir de ayuda al usuario en la utilización del producto; por consiguiente, AMSTRAD queda eximido de responsabilidad por el daño o pérdida a que pueda dar lugar la utilización de la información aquí publicada o la incorrecta utilización del producto.

Rogamos a los usuarios que rellenen y envíen las tarjetas de registro/garantía.

AMSOFT agradecerá el envío de comentarios y sugerencias relativos a este manual y al producto en él descrito.

Toda la correspondencia se debe dirigir a:

AMSTRAD ESPAÑA

Aravaca, 22
28040 Madrid
España

Escrito por Ivor Spital y Rupert Goodwins

Incluye extractos de la obra 'ZX Spectrum BASIC programming', de Steven Vickers y Robin Bradbeer.

Publicado por Amstrad

Edición española producida por Vector Ediciones

Traducción: Emilio Benito Santos

Primera edición: 1986

AMSTRAD es marca comercial registrada por AMSTRAD Consumer Electronics plc

Queda estrictamente prohibido utilizar la marca y la palabra AMSTRAD sin la debida autorización
Impreso por Artes Gráficas EMA, S.A.

IMPORTANTE

Por favor, lea las siguientes advertencias:

1. No intente conectar este equipo a una red de distribución de energía eléctrica que no sea de 220–240 V c.c., 50 Hz.
 2. Cuando haya terminado de usar el +2, desconecte *siempre* la fuente de alimentación de la red.
 3. El mantenimiento que pueda hacer el usuario no requiere en ningún caso acceder al interior del equipo. Así pues, **NO ABRA NUNCA LA FUENTE DE ALIMENTACIÓN**, porque en su interior hay **ALTA TENSIÓN**. Confíe todas las reparaciones a personal cualificado.
 4. No obstruya ni cubra los orificios de ventilación.
 5. No utilice ni almacene el equipo a temperaturas demasiado altas ni demasiado bajas, ni en lugares húmedos o polvorientos.
 6. No conecte ni desconecte ningún dispositivo en el zócalo **EXPANSION E/S** del +2 cuando éste esté encendido, pues corre el grave riesgo de dañar tanto el ordenador como el dispositivo externo.
 7. Cuando haya apagado el televisor (o el monitor), no apague el +2 inmediatamente; espere unos segundos.
 8. No apague el ordenador cuando en la memoria del +2 tenga algún programa o datos que desee conservar, pues los perdería inmediatamente. Además, si enciende o apaga cualquier periférico puede provocar la «caída del sistema», lo que representa la pérdida del programa y los datos.
-

Contenido

Capítulo 1: Apertura de la caja	1
Desembalaje	1
Conexión del ordenador a la red	2
Instalación	2
Capítulo 2: Funcionamiento de su +2	5
Encendido	5
Sintonización del televisor	5
Utilización del +2	9
El menú de presentación	9
Capítulo 3: Carga de programas del Spectrum 128	13
Carga de los programas	13
Interrupción del proceso de carga	13
Reinicialización del +2	14
Capítulo 4: Carga de programas del Spectrum 48	15
Carga de los programas	15
Interrupción del proceso de carga	15
Reinicialización del +2	16
Capítulo 5: Introducción a BASIC	17
Capítulo 6: Utilización de 128 BASIC	19
El editor	19
El menú de edición	20
Renumeración de un programa de BASIC	21
Cambio de pantalla	22
Listado por impresora	22
Introducción de un programa	22
Movimiento del cursor	23
Ejecución de un programa	23
Órdenes e instrucciones	24

Capítulo 7: Utilización de 48 BASIC	27
Utilización del +2 como Spectrum de 48K	27
Activación del modo 48 BASIC	28
El teclado en 48 BASIC	28
Introducción de un programa	32
Edición de la línea actual	32
Capítulo 8: Guía completa de programación en BASIC	33
Parte 1: Introducción	33
Parte 2: Conceptos sencillos de programación	39
Parte 3: Decisiones	47
Parte 4: Bucles	49
Parte 5: Subrutinas	55
Parte 6: Datos en los programas	57
Parte 7: Expresiones	61
Parte 8: Cadenas literales	65
Parte 9: Funciones	69
Parte 10: Funciones matemáticas	77
Parte 11: Números aleatorios	83
Parte 12: Matrices	87
Parte 13: Condiciones	91
Parte 14: El juego de caracteres	95
Parte 15: Más sobre PRINT e INPUT	105
Parte 16: Colores	113
Parte 17: Gráficos	121
Parte 18: Movimiento	127
Parte 19: Sonido	131
Parte 20: Operaciones con el magnetófono	141
Parte 21: Operaciones de la impresora	153
Parte 22: Otros periféricos	157
Parte 23: IN y OUT	159
Parte 24: La memoria	163

Parte 25: Variables de sistema	171
Parte 26: Utilización del código de máquina	177
Parte 27: Juego de caracteres del Spectrum	181
Parte 28: Mensajes	189
Parte 29: Información de referencia	199
Parte 30: BASIC	203
Parte 31: Programas de ejemplo	217
Parte 32: Binario y hexadecimal	225
Capítulo 9: Utilización de la calculadora	229
Selección de la calculadora	229
Introducción de números	230
Resultado actual	230
Uso de las funciones matemáticas incorporadas	230
Edición de la pantalla	230
Asignación de variables	231
Salida de la calculadora	231
Capítulo 10: Conexión de periféricos	233
Joystick(s)	233
Monitor	234
Amplificador	235
Impresora	235
Dispositivo serie	235
Dispositivos MIDI	236
Teclado numérico	237
'Interface One' y microunidades	237
Otros dispositivos	237



Apertura de la caja

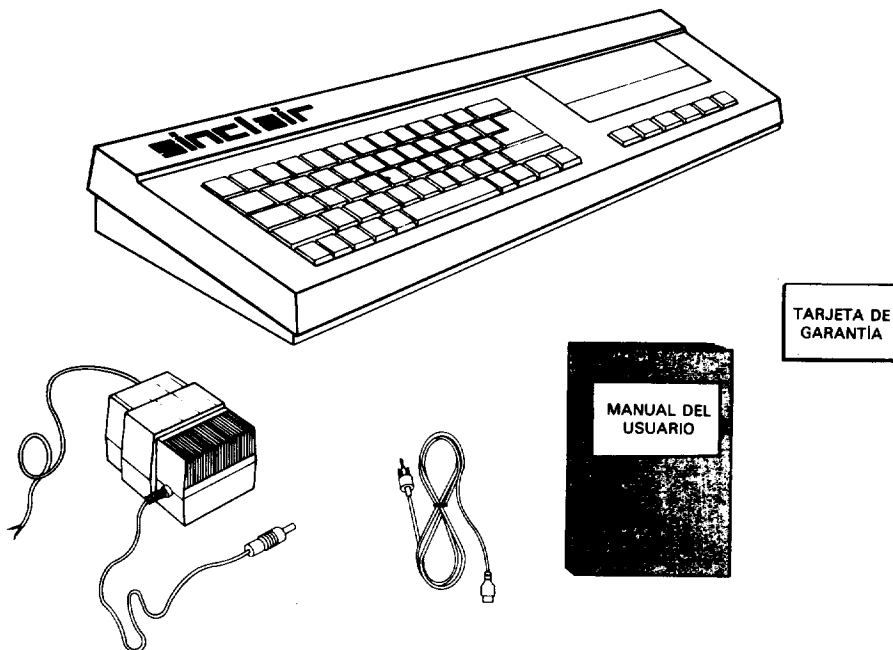
Temas tratados en este capítulo:

Desembalaje
Conexión del ordenador a la red
Instalación

Desembalaje

La caja en que se suministra este ordenador debe contener lo siguiente:

- El ordenador Spectrum +2
- La unidad de alimentación
- El cable de la antena
- Este manual (junto con las tarjetas de registro y garantía)



Conexión del ordenador a la red

El Spectrum +2 sólo se puede conectar a la red de 220–240V c.a., 50Hz.

No extraiga ningún tornillo ni trate de abrir la carcasa de la unidad de alimentación. Respete las advertencias que se dan en la etiqueta de características, que está situada en la cara inferior de la fuente de alimentación:

¡ATENCIÓN! NO EXTRAIGA NINGÚN TORNILLO. CIRCUITOS ACTIVOS EN EL INTERIOR.

Instalación

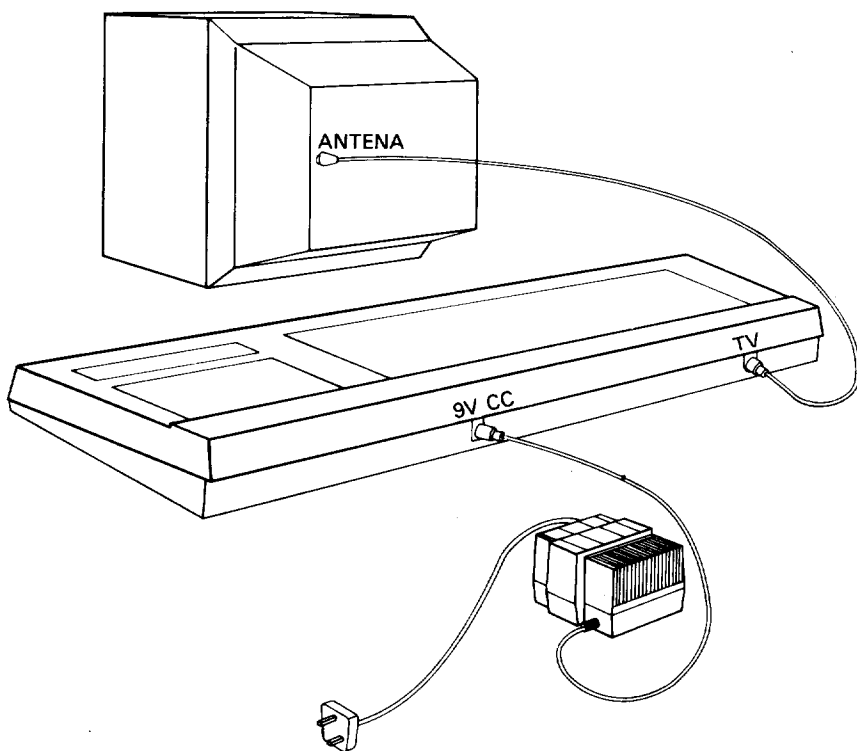
Vamos a describir la instalación del sistema +2 estándar. Todo lo que usted necesita, aparte de los elementos que ya ha desembalado, es un aparato de televisión (con UHF). Puede ser un televisor de color o de blanco y negro, si bien con este último no podrá disfrutar de la plena capacidad de color de su +2.

Si desea conectar a su +2 *algún periférico* (por ejemplo, joysticks, microunidades, monitor, teclado numérico, amplificador de sonido, dispositivo MIDI, impresora, etc.), debe consultar el capítulo 10 ('Conexión de periféricos a su +2').

Coloque el ordenador +2 sobre una superficie plana adecuada, listo para ser conectado al televisor. A continuación, si tiene conectado un cable en el zócalo de antena del televisor, desconéctelo. Tome el cable de la antena suministrado con su +2; introduzca la clavija *más grande* en el zócalo de la antena del televisor, y la *más pequeña* en el zócalo TV de la cara posterior del +2.

Finalmente, de los dos cables que salen de la fuente de alimentación, tome el que termina en la clavija pequeña y conéctelo en el zócalo señalado con **9V DC**, también en la cara posterior del +2.

El sistema +2 se encuentra ahora listo para ser encendido.



Instalación del sistema +2 estándar



Capítulo 2

Funcionamiento de su +2

Temas tratados en este capítulo:

- Encendido
- Sintonización del televisor
- Utilización del +2
- El menú de presentación

Encendido

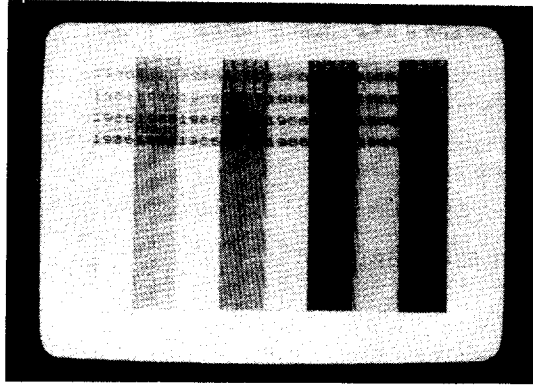
Conecte la clavija de corriente alterna de la fuente de alimentación a la toma de corriente. En este momento se debe encender el piloto rojo que está en el panel principal del +2.

A continuación, encienda el televisor. Seguramente verá una imagen aleatoria, que es la característica 'nieve' (ruido blanco), y oirá un sonido intenso y silbante procedente de los altavoces del televisor. Ajuste el control de volumen del aparato de televisión hasta conseguir un nivel de sonido que no le resulte molesto. El siguiente paso será preparar el +2 para realizar la sintonización.

Preparativos para sintonizar el televisor

El +2 es capaz de generar su propia 'carta de ajuste', mediante la cual se puede sintonizar el televisor con toda precisión. Esa carta de ajuste consiste en dieciséis barras verticales de color (con caracteres de texto sobreimpresos) que aparecen en la pantalla del televisor y un sonido repetitivo que es reproducido por el altavoz. (Si el televisor es de blanco y negro, las barras de color aparecerán en diversos matices de gris). Podrá oír y ver esta señal de comprobación cuando haya terminado de sintonizar el televisor (por el procedimiento que describiremos en la sección siguiente).

Para activar la señal de prueba, pulse la tecla **BREAK** (que se encuentra en el extremo superior derecho del teclado) y, sin soltarla, pulse y suelte el botón **RESET** (que está en la cara izquierda del +2). Mantenga pulsada la tecla **BREAK** durante unos segundos más y luego suéltela. El +2 empezará a generar la 'carta de ajuste' y usted podrá comenzar a sintonizar el televisor, tal y como se explica a continuación.



Sintonización cuando el televisor tiene selector de canales de botonera

Si su aparato de televisión no tiene selector de canales de botonera, pase a la sección siguiente.

Si el selector de canales sí es de ese tipo, pulse uno de los botones para seleccionar un canal que esté libre (es decir, uno que no esté utilizando para recibir programas de televisión o de video). Si el televisor dispone de 'control automático de frecuencia' (AFC o AFT), debe desconectarlo.

Ajuste el mando de sintonía correspondiente al canal elegido hasta que la pantalla muestre la imagen de la figura anterior y el sonido sea lo más limpio posible.

Una vez conseguida la sintonía, ya puede conectar el control automático de frecuencia del televisor.

Finalmente, ajuste los mandos de contraste, color y brillo del televisor para optimizar la legibilidad de los caracteres de texto.

Ahora que ha sintonizado un canal *específicamente* para el +2, cada vez que desee utilizar el ordenador junto con el televisor bastará con que seleccione ese canal.

Si todo ha ido bien, puede pasar directamente a la sección titulada 'Utilización del +2'. Si no, consulte la sección '¿Algún problema?'.

Sintonización manual

Si su televisor no está equipado con selector de canales de botonera, tendrá que utilizar el mando de sintonía manual para ajustarlo al +2.

Después de conectar y encender el +2 y el televisor, active la señal de prueba según se ha explicado en la sección 'Preparativos para sintonizar el televisor'.

Ajuste el mando de sintonía hasta que la pantalla muestre la carta de ajuste y el sonido sea lo más limpio posible.

Finalmente, ajuste los mandos de contraste, color y brillo del televisor para optimizar la legibilidad de los caracteres de texto.

Cada vez que desee utilizar el +2 junto con el televisor, deberá seguir este procedimiento de sintonización manual.

Si todo ha ido bien, puede pasar directamente a la sección titulada 'Utilización del +2'. Si no, consulte la sección siguiente.

¿Algún problema?

Si ha conseguido sintonizar el televisor satisfactoriamente, puede pasar a la siguiente sección.

Si no es capaz de sintonizarlo, la siguiente lista de comprobación puede ayudarle a determinar dónde reside el problema y qué remedio aplicarle.

1. Problema: No se enciende el piloto rojo de alimentación.

- Remedios:
- Compruebe que la fuente de alimentación está conectada al ordenador.
 - Compruebe que la clavija de corriente alterna de la fuente de alimentación se encuentra conectada a la toma de corriente.
 - Compruebe las conexiones dentro de la toma de corriente.

2. Problema: El piloto de alimentación se enciende, pero no es posible sintonizar ninguna señal en el televisor.

- Remedios:
- Compruebe que el televisor está conectado y que funciona correctamente.
 - Compruebe que el televisor es del tipo UHF estándar (para color o para blanco y negro).
 - Compruebe que el cable de antena (suministrado con el ordenador) está bien conectado al zócalo de antena del televisor y al del ordenador.
 - Si el televisor tiene selector de canales por botonera, compruebe que está pulsado el botón correspondiente al canal elegido.

3. Problema: La señal que se consigue en el televisor, procedente del ordenador, es de baja calidad.

-
- Remedios:
- Compruebe que el aparato de televisión está conectado y que funciona correctamente.
 - Compruebe que el cable de antena (suministrado con el ordenador) está bien conectado al zócalo de antena del televisor y al del ordenador.
 - Si el televisor dispone de control automático de frecuencia (AFC), desactívelo.
 - Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.

4. Problema: **Se ha conseguido sintonizar una señal procedente del ordenador, pero no se trata de la carta de ajuste descrita anteriormente.**

- Remedio:
- Asegúrese de que el ordenador está enviando al televisor la señal de prueba; consulte la sección titulada 'Preparativos para sintonizar el televisor'.

5. Problema: **Aparecen las barras de color de la carta de ajuste, pero no se oye ningún sonido (tono repetitivo) procedente de los altavoces.**

- Remedios:
- Compruebe que el control de volumen del televisor no está al mínimo.
 - Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.

6. Problema: **Se puede oír el sonido (tono repetitivo) de la señal de prueba, pero no se ve las barras de color en la pantalla.**

- Remedios:
- Compruebe que los mandos de contraste, color y brillo del televisor no están al mínimo.
 - Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.

7. Problema: **Se ha conseguido sintonizar las barras de color y el sonido de la señal de prueba, pero no es posible leer los caracteres de texto.**

- Remedios:
- Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.
 - Compruebe que los mandos de contraste, color y brillo del televisor han sido ajustados adecuadamente.

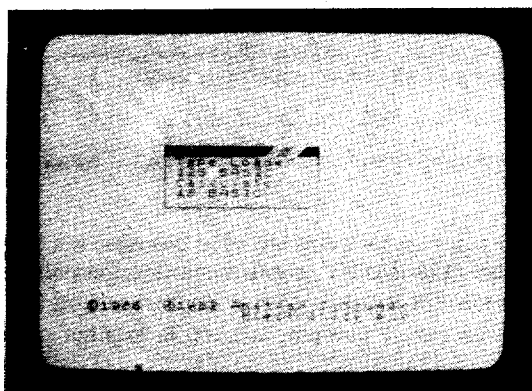
Si no logra identificar la causa de su problema, realice de nuevo el procedimiento completo (desde el principio de este capítulo). Si el problema aún persiste, avise a su distribuidor.

Utilización del +2

El sistema +2 debería estar ahora completamente preparado, con las barras de color de la señal de prueba en la pantalla y el sonido repetitivo en los altavoces del televisor.

Ahora vamos a desconectar la señal de prueba y comenzaremos a utilizar el +2. Pulse y suelte el botón **RESET** (cara izquierda el +2). Desaparecerá de la pantalla la carta de ajuste y en su lugar podrá ver el 'menú de presentación'.

El menú de presentación



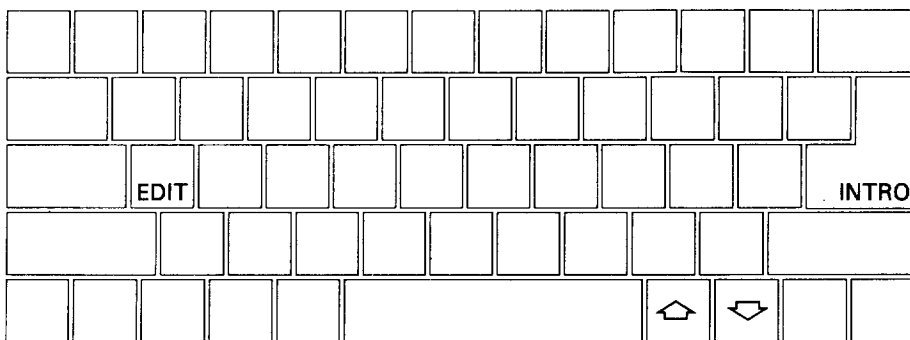
El menú de presentación aparece cada vez que se enciende el +2, y también cada vez que se lo renicializa (pulsando y soltando el botón **RESET**).

Este menú le ofrece las cuatro opciones que puede ver en un recuadro en la pantalla:

- Carg. cinta** Elija esta opción si desea cargar programas escritos para el Spectrum 128.
- 128 BASIC** Elija esta opción si desea utilizar el +2 para programar en BASIC.
- Calculadora** Elija esta opción si desea utilizar el +2 solamente como calculadora.
- 48 BASIC** Elija esta opción si desea cargar programas escritos para el Spectrum 48 (o si quiere utilizar el +2 como si fuese un Spectrum de 48K).

Cómo elegir una opción

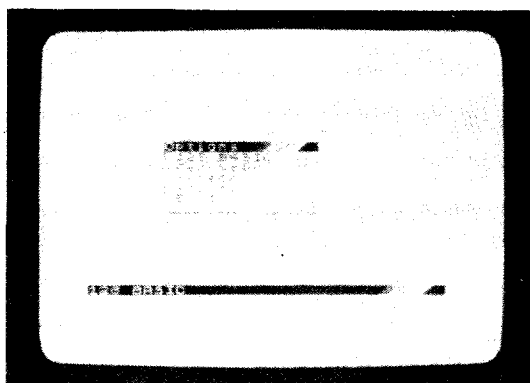
Observe que la opción **Carg. cinta** aparece resaltada por una 'barra'. Esto significa que dicha opción está preseleccionada, o sea, lista para ser seleccionada (la selección aún no ha sido confirmada). A los efectos de este ejemplo, supongamos que usted no quiere seleccionar **Carg. cinta**, sino **128 BASIC**. Esto implica que tiene que mover la barra hasta la línea de la opción **128 BASIC**. Para ello, pulse las teclas del *cursor* (ilustradas en la figura siguiente) hasta que dicha barra llegue a la posición deseada.



Teclas del cursor

Cuando la barra esté sobre '128 BASIC', confirme la elección pulsando la tecla **INTRO**. El +2 seleccionará el modo 128 BASIC. (Verá una barra negra horizontal en la parte inferior de la pantalla y un cursor que parpadea en la esquina superior izquierda.)

No se preocupe si no sabe nada de BASIC, pues todavía no vamos a empezar a programar. Ahora vamos a volver al menú de presentación. Para ello utilizaremos un menú diferente, denominado *menú de edición*. Este menú se invoca pulsando la tecla **EDIT**.



Usando de nuevo las teclas del cursor e **INTRO**, seleccione la opción **Salida** para volver al menú de presentación.

Ahora puede usted seleccionar cualquier opción del menú de presentación. Dependiendo de su selección, consulte alguno de los siguientes capítulos para obtener más información:

Carg. cinta Consulte el capítulo 3.

128 BASIC Consulte los capítulos 5, 6 y 8.

Calculadora Consulte el capítulo 9.

48 BASIC Consulte los capítulos 4, 5, 7 y 8.

Importante. Cuando haya terminado su sesión de trabajo con el +2, *no olvide* desconectar la fuente de alimentación de la toma de corriente.



Carga de programas del Spectrum 128

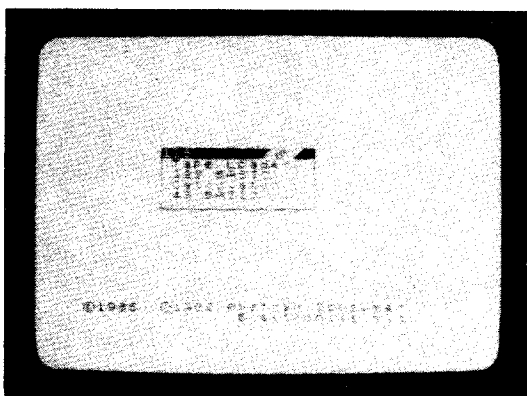
Temas tratados en este capítulo:

Carga de los programas
Interrupción del proceso de carga
Reinicialización del +2

Recomendamos el uso exclusivo de programas que lleven el logotipo 'SINCLAIR QUALITY CONTROL' ('Control de calidad Sinclair'). Para obtener más información, lea la 'Introducción' del principio de este manual.

Para cargar programas escritos para el Spectrum 128 (un juego, un programa de aplicación, etc.) siga estas instrucciones:

1. Instale y encienda el sistema +2 por el procedimiento explicado en el capítulo anterior, de forma tal que aparezca en la pantalla el menú de presentación:



2. Seleccione la opción **Carg. cinta** (cargador de cinta) del menú de presentación. (Si no sabe cómo seleccionar una opción del menú, consulte el capítulo 2.)

-
3. Introduzca la cinta del programa en el magnetófono de datos y asegúrese de que la cinta está rebobinada hasta el principio.
 4. Ponga la cinta en movimiento. Cuando comience la carga, el color del margen parpadeará y aparecerá rayado, indicando que el programa está siendo leído desde la cinta. Si el mando de volumen del televisor no está al mínimo, oirá también un sonido variable de alta frecuencia. Nuevamente se trata de una indicación de que el programa está siendo leído.

Si desea abandonar la carga, mantenga pulsada la tecla **BREAK** hasta que el +2 regrese al menú de presentación.

La mayor parte de las cintas de programas disponibles comercialmente tarda unos pocos minutos en cargar. Inicialmente aparecerá el nombre del programa (**Program: nombre**) en el extremo superior izquierdo de la pantalla, seguido de otros diversos mensajes e imágenes (que diferirán de un programa a otro.)

Cuando el programa esté cargado, detenga la cinta. El programa estará ya listo para ser utilizado.

Cuando termine de usar el programa y desee emplear el +2 para cualquier otra cosa, pulse y suelte el botón **RESET** (que está en el lateral izquierdo del +2). Recuerde siempre que cada vez que se pulsa el botón **RESET** se borra todo lo que hay en la memoria (RAM) del ordenador. Por esta razón, *antes* de pulsar este botón es necesario estar seguros de que no hay nada en la memoria del +2 cuya pérdida sea importante.

Carga de programas del Spectrum 48

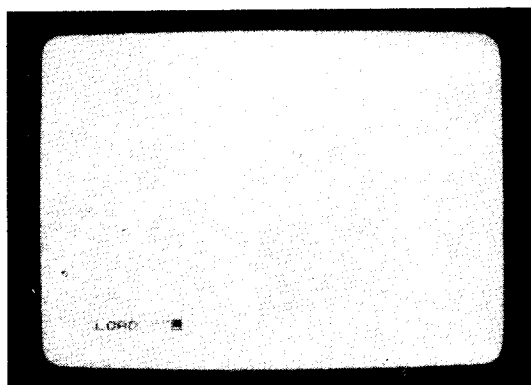
Temas tratados en este capítulo:

Carga de los programas
Interrupción del proceso de carga
Reinicialización del +2

Recomendamos el uso exclusivo de programas que lleven el logotipo 'SINCLAIR QUALITY CONTROL' ('Control de calidad Sinclair'). Para obtener más información, lea la 'Introducción' del principio de este manual.

Para cargar programas escritos para el Spectrum 48 (un juego, un programa de aplicación, etc.) siga estas instrucciones:

1. Instale y encienda el sistema +2 por el procedimiento descrito en el capítulo 2, de forma tal que aparezca en la pantalla el menú de presentación.
2. Seleccione la opción **48 BASIC** del menú de presentación. (Si no sabe cómo seleccionar una opción del menú, consulte el capítulo 2.)
3. El menú de presentación desaparece y en la parte inferior de la pantalla se muestra el mensaje '©1982 Amstrad'. Ahora pulse la tecla **J** una vez y dos veces la tecla **"** (comillas). El aspecto de la pantalla debe ser el siguiente:



Si la imagen que aparece en la pantalla no es igual a la que se muestra en esta figura, quizás haya seleccionado una opción equivocada del menú o no haya pulsado la tecla correcta. En tal caso, pulse y suelte el botón **RESET** (que está en el lateral izquierdo del +2) y repita las etapas 2 y 3.

Cuando vea el mensaje mostrado en la figura anterior, pulse **INTRO**.

4. Introduzca la cinta del programa en el magnetófono de datos y asegúrese de que la cinta está rebobinada hasta el principio.
5. Ponga la cinta en movimiento. Cuando comience la carga, el color del margen parpadeará y aparecerá rayado, indicando que el programa está siendo leído. Si el mando de volumen del televisor no está al mínimo, oirá también un sonido variable de alta frecuencia. Nuevamente se trata de una indicación de que el programa está siendo leído.

Si desea abandonar la carga, mantenga pulsada la tecla **BREAK** hasta que el +2 regrese al modo **48 BASIC**.

La mayor parte de las cintas de programas disponibles comercialmente tarda unos pocos minutos en cargar. Inicialmente aparecerá el nombre del programa (**Program: nombre**) en el extremo superior izquierdo de la pantalla, seguido de otros diversos mensajes e imágenes (que diferirán de un programa a otro.)

Cuando el programa esté cargado, detenga la cinta. El programa estará ya listo para ser utilizado.

Cuando termine de usar el programa y desee emplear el +2 para cualquier otra cosa, pulse y suelte el botón **RESET** (que está en el lateral izquierdo del +2). Recuerde siempre que cada vez que se pulsa el botón **RESET** se borra todo lo que hay en la memoria (RAM) del ordenador. Por esta razón, antes de pulsar este botón es necesario estar seguros de que no hay nada en la memoria del +2 cuya pérdida sea importante.

Introducción a BASIC

El +2 utiliza un lenguaje de ordenador llamado *BASIC* (Beginners All-purpose Symbolic Instruction Code: código de instrucciones simbólicas de propósito general para principiantes). A pesar de que BASIC es, con mucho, el lenguaje más usual para ordenadores domésticos, cada marca y modelo de ordenador suele tener su propio dialecto, y el +2 no es una excepción. El *Spectrum BASIC* ha sido diseñado para que pueda ser fácilmente aprendido y utilizado, aunque en muchos aspectos es diferente de otras versiones de BASIC. En el capítulo 8 le proporcionamos una guía completa de BASIC en el +2. Sin embargo, si usted no está acostumbrado a programar, debería leer primero el capítulo 6 ('Utilización de 128 BASIC'). Incluso si es usted un experimentado usuario de BASIC en otro ordenador, es posible que quiera leer dicho capítulo, que describe el *editor* y otros aspectos exclusivos del +2.

Si está usted acostumbrado al Spectrum de 48K, gran parte de lo que contiene este manual le resultará familiar; de hecho, hay un modo en el que el +2 funciona exactamente igual que el antiguo Spectrum (incluso en lo referente a edición y programación). Este modo sólo tiene interés histórico; de cualquier forma, en el capítulo 7 ('Utilización de 48 BASIC') hemos reunido la información más relevante.



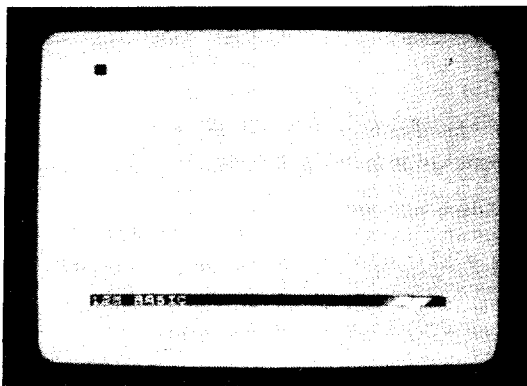
Utilización de 128 BASIC

Temas tratados en este capítulo:

- El editor
- El menú de edición
- Renumeración de un programa de BASIC
- Cambio de pantalla
- Listado por impresora
- Introducción de un programa
- Movimiento del cursor
- Ejecución de un programa
- Órdenes e instrucciones

El +2 dispone de un avanzado *editor* que se utiliza para crear, modificar y ejecutar programas de BASIC de 128K. Para activar el editor, seleccione la opción **128 BASIC** del menú de presentación utilizando las teclas del cursor y la tecla **[INTRO]**. (Si no sabe cómo seleccionar una opción del menú, consulte el capítulo 2.)

En la pantalla se debería ver lo siguiente:



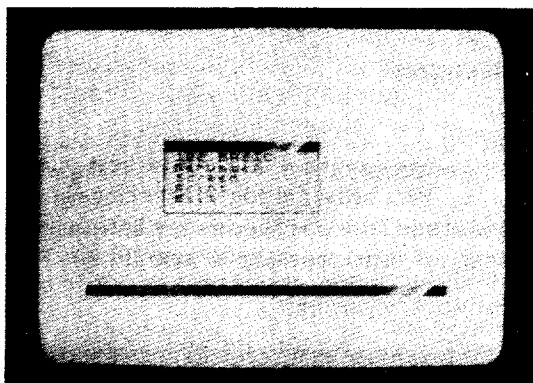
En relación con esta pantalla, hay que tener en cuenta tres cosas:

Primera, que en el extremo superior izquierdo hay un pequeño rectángulo parpadeante, cuyo color alterna entre el blanco y el azul: es el *cursor*. Si pulsa algunas letras en el teclado, éstas aparecerán en la pantalla en la posición del cursor.

Segunda, que en la parte inferior de la pantalla hay una barra negra. Recibe el nombre de *barra de información* porque indica qué parte del software incorporado al +2 está siendo utilizado. En este momento dicha barra indica '128 BASIC', ya que ése es el nombre del editor.

El último punto a tener en cuenta por el momento es la pantalla pequeña, que se encuentra situada entre la barra de información y el extremo inferior de la pantalla y que generalmente está en blanco. Esta pantalla pequeña sólo tiene espacio para dos líneas de texto y suele ser utilizada por el +2 cuando éste detecta un error y necesita escribir un informe para comunicarlo. No obstante, también tiene otras aplicaciones, que describiremos más adelante.

Ahora pulse la tecla **[EDIT]**. Observará que ocurren dos cosas: se borra el cursor y aparece un nuevo menú, llamado *menú de edición*:



Las opciones del menú de edición se seleccionan de la misma manera que las del menú de presentación (utilizando las teclas del cursor y la tecla **[INTRO]**).

Veamos esas opciones una por una.

128 BASIC Esta opción cancela el menú de edición y restituye el cursor. Aunque no le parezca muy útil, permite volver al programa actual, sin deteriorarlo, cuando se pulsa **[EDIT]** accidentalmente.

Renumerar Los programas de BASIC utilizan números de línea para determinar el orden en el que las instrucciones han de ser ejecutadas. Estos números (que pueden ser cualquier entero desde el 1 al 9999) debe usted introducirlos al principio de cada línea de programa que escriba. Al seleccionar la opción

Renumerar, BASIC renumera las líneas de forma que la primera sea la 10 y los números sucesivos vayan aumentando de 10 en 10. Las referencias a números de línea que estén incluidas en las instrucciones del programa (por ejemplo, tras GO TO, GO SUB, LINE, RESTORE, RUN y LIST) resultan correctamente renumeradas.

Si por alguna razón no es posible renumerar, quizá porque no hay ningún programa en la memoria del ordenador, o porque la operación generaría números de línea mayores que 9999, el +2 emite un pitido de tono grave y el menú desaparece.

(Los recién llegados a BASIC deberían pasar ahora al apartado que describe la opción Pantalla.)

Utilizando técnicas avanzadas, es posible renumerar el programa con valores distintos de los que ofrece esta opción del menú (*comienzo*=10 y *salto*=10). Esto es útil, por ejemplo, cuando se quiere renumerar un programa de más de 1000 líneas (las cuales no podrían ser renumeradas de forma válida a intervalos de 10). La siguiente orden sirve para realizar una renumeración más general.

(*Nota*. Si no tiene usted cierta experiencia en la programación del BASIC del Spectrum, seguramente no comprenderá cómo funciona esta orden.)

```
LET com=5: LET salto=2: LET comalt=INT (com/256):  
LET saltoalt=INT (salto/256): POKE 23444,com-256*comalt:  
POKE 23445,comalt: POKE 23446,salto-256*saltoalt:  
POKE 23447,saltoalt
```

Cambiando los valores de las variables *comienzo* y *salto*, la opción Renumerar renumerará con cualquier número de línea (válido) y cualquier intervalo. Dé la orden anterior y después seleccione la opción del menú.

Más adelante, cuando haya usted aprendido a escribir programas de BASIC y a grabarlos en el magnetófono, quizás quiera incorporar lo anterior en un programa corto para utilizarlo en el futuro. Por ejemplo:

```
10 INPUT "Línea inicial", com  
20 INPUT "Intervalo", salto  
30 LET comalt=INT (com/256)  
40 LET saltoalt=INT (salto/256)  
50 POKE 23444,com-256 * comalt  
60 POKE 23445,comalt  
70 POKE 23446,salto-256 * saltoalt  
80 POKE 23447,saltoalt  
90 PRINT "Pulse EDIT y luego seleccione la opción Renumerar"
```

Pantalla Esta opción lleva el cursor a la zona más pequeña (inferior) de la pantalla y permite que las líneas de BASIC sean introducidas y editadas allí. Esto es útil sobre todo cuando se está trabajando con gráficos (véase el capítulo 8, parte 17), ya que lo que se haga en la pantalla inferior no interferirá con la pantalla superior. Para volver a esta última (lo cual se puede hacer en cualquier momento durante la edición), seleccione de nuevo la opción **Pantalla** del menú de edición.

Imprimir Si hay una impresora conectada, esta opción imprimirá un listado del programa actual. Cuando concluya este listado, el menú desaparecerá y volverá el cursor. Si por alguna razón el ordenador no puede imprimir (por ejemplo, porque la impresora se encuentra fuera de línea o está desconectada), pulsando dos veces la tecla **BREAK** se vuelve al editor.

Salida Esta opción conduce de nuevo al menú de presentación (el +2 retiene en la memoria el programa con el que se estuviera trabajando). Si desea volver al programa, seleccione la opción **128 BASIC** del menú de presentación.

Si selecciona la opción **48 BASIC** del menú de presentación (o si apaga o reinicializa con el botón **RESET** el ordenador), perderá cualquier programa que se encuentre en la memoria. (Sin embargo, sí puede utilizar la opción **Calculadora** del menú de presentación sin perder el programa actual.)

Reinicialice el +2 y *seleccione la opción 128 BASIC*. Después escriba la siguiente línea. Según los vaya escribiendo, los caracteres aparecerán en la pantalla (un *carácter* es una letra, un número, un espacio, etc.). El signo *igual* (=) se obtiene pulsando la tecla L en combinación con **SIM**.

Ahora escriba esta línea:

```
10 for f=1 to 255 step 10
```

y a continuación pulse **INTRO**. Suponiendo que lo haya escrito todo correctamente, el +2 habrá reescrito la línea con las palabras FOR y STEP en letras mayúsculas:

```
10 FOR f=1 TO 255 STEP 10
```

El +2 habrá emitido también un pitido breve y llevado el cursor al principio de la siguiente línea.

Si la línea permanece en minúsculas y oye usted un pitido de tono grave, esto indica que ha escrito algo equivocado. Observe también que el color del cursor cambia a rojo cuando se detecta un error, y que el ordenador no acepta la línea mientras usted no la corrija. Utilice las teclas del cursor para llevar el cursor hasta el lugar de la línea que hay que corregir y escriba los caracteres que quiera insertar, o utilice la tecla **BORR** para suprimir los caracteres sobrantes. Cuando haya terminado de corregir la línea, pulse **INTRO**.

Ahora escriba la siguiente línea:

```
20 plot 0,0:draw f,175:plot 255,0:draw -f,175      (pulse [ENTER])
```

[Los dos puntos (:) se obtienen pulsando la tecla de la Z en combinación con [SIMB]; el signo menos (-), con [SIMB] y J.]

No se preocupe por el hecho de que al llegar al borde derecho de la pantalla el texto «invada» la siguiente línea; el ordenador se encarga de pasar de una línea a otra cuando es necesario y de alinear el texto de forma tal que pueda ser leído más fácilmente. A diferencia de lo que ocurre con la máquina de escribir, en el ordenador no hay que hacer nada especial cuando se llega al final de una línea de la pantalla, ya que el +2 lo detecta automáticamente y lleva el cursor al principio de una nueva línea.

La última línea de este programa es:

```
30 next f      (pulse [INTRO])
```

Los números que hemos puesto al principio de cada línea son los números de *línea* y sirven para identificarlas. La línea que usted acaba de escribir es la 30; el cursor debería encontrarse ahora debajo de ella y a su izquierda. Pulse una vez la tecla de 'cursor arriba', [↑]. El cursor sube a la línea 30, pero no en línea recta, como se podría esperar, ya que encima de él no había ningún carácter. Lo que hace es tratar de averiguar qué es lo que usted quiere hacer, y se sitúa en consecuencia. El cursor intenta por todos los medios evitar los espacios en blanco (aunque no le importan los espacios reales que pueda haber entre las palabras de una línea); siempre busca algún carácter al que ir.

Pulse una vez más la tecla [↑]. Ahora lleve el cursor hacia la derecha con la tecla [→], hasta que se encuentre sobre el 1 de DRAW -f,175. ¿Qué cree que ocurrirá con el cursor, dada su aversión por los espacios en blanco, cuando usted trate de llevarlo a la línea siguiente? Pruébelo (con la tecla [↓]). Como era de esperar, el cursor salta hasta el carácter disponible más cercano, que en este caso es el final de la línea 30. Ahora pulse nuevamente la tecla [↑]. Se podría pensar, ya que había caracteres justamente encima, que el cursor se movería directamente hacia arriba; y sin embargo no es así, sino que vuelve a la posición anterior. De nuevo se trata de la «inteligencia» del +2: se da cuenta de que usted no ha movido el cursor por la línea 30 y recuerda la última posición en la que ha estado. Para comprobarlo, desplace el cursor otra vez hacia abajo y luego hacia la izquierda (para ponerlo sobre la f), después hacia la derecha y finalmente hacia arriba. El ordenador piensa que usted ha estado trabajando en la línea 30 y, por lo tanto, no tiene inconveniente en olvidar la última posición del cursor en la línea 20. Por eso el cursor se mueve directamente hacia arriba.

Este tipo de movimiento del cursor, denominado *rastreo*, puede resultar un poco confuso al principio. No obstante, le facilitará la edición de programas una vez que se haya familiarizado con él.

Ahora pulse **[INTRO]**. El ordenador abre una línea en preparación de un nuevo texto. Escriba:

run (pulse **[INTRO]**)

Ocurrirán muchas cosas. Para empezar, la barra de información y las líneas del programa desaparecen de la pantalla, ya que el editor de 128 BASIC se prepara para ceder el control al programa que usted acaba de escribir. A continuación comienza el programa, dibuja un atractivo diseño y termina con el informe:

0 OK, 30:1

No se preocupe por el significado de este informe.

Pulse **[INTRO]**. La pantalla se borrará y reaparecerá la barra de información, así como el listado del programa. Esto dura aproximadamente un segundo, durante el cual el +2 no admitirá ninguna entrada proveniente del teclado, así que no se moleste en escribir nada mientras todo esto ocurre.

Usted ya acaba de realizar la mayor parte de las operaciones necesarias para programar y utilizar un ordenador. En primer lugar, le ha dado al +2 una lista de instrucciones. Las *instrucciones* le dicen al ordenador qué tiene que hacer y cómo debe hacerlo (por ejemplo, 30 NEXT f). Por otra parte, las instrucciones van precedidas de un número de línea y, cuando usted las escribe, el ordenador las almacena, en lugar de obedecerlas inmediatamente. Finalmente, usted le dio al +2 la orden RUN para ejecutar el programa que tenía almacenado en la memoria.

Las *órdenes* son similares a las instrucciones, pero no tienen número de línea y el +2 las obedece inmediatamente, tan pronto como se pulsa **[INTRO]**. En general, cualquier instrucción puede ser utilizada como una orden, y viceversa; depende de las circunstancias. Toda orden o instrucción debe contener al menos una palabra clave. Las *palabras clave* constituyen el vocabulario del ordenador, y muchas de ellas necesitan *parámetros*.

Por ejemplo, en la orden DRAW 40,200, DRAW es la palabra clave, mientras que 40 y 200 son los parámetros (que le dicen al ordenador dónde, exactamente, debe realizar el dibujo). Todo lo que el ordenador haga en BASIC se atendrá a estas reglas.

Ahora pulse **[EDIT]** y seleccione la opción Pantalla. El editor desplaza el programa a la zona inferior de la pantalla y se deshace de la barra de información. Usted sólo puede ver la línea 10 del programa, ya que el resto está escondido fuera de esta zona de la pantalla (compruébelo subiendo y bajando el cursor).

Pulse **[INTRO]** y escriba:

run (pulse **[INTRO]**)

El programa es ejecutado exactamente igual que antes. Pero, esta vez, si pulsa **[INTRO]** al concluir la ejecución, la pantalla no se borrará y usted podrá subir y bajar el listado del programa (utilizando las teclas **[↑]** y **[↓]**) sin alterar la pantalla superior. Quizás

piense que si pulsa ahora **[INTRO]** para obtener el menú de edición se estropeará la imagen dibujada. Sin embargo, el +2 recuerda lo que hay tras el menú de edición y lo recupera cuando oculta el menú.

Para comprobar que el editor está trabajando realmente en la parte inferior de la pantalla, pulse **[INTRO]** y cambie la línea 10 por:

```
10 FOR f=1 TO 255 STEP 7
```

Para ello debe llevar el cursor hasta el final de la línea 10 (justamente a la derecha de STEP 10), pulsar dos veces **[BORR]**, escribir 7 y pulsar **[INTRO]**.

Ahora escriba:

```
go to 10      (pulse [INTRO])
```

Las palabras clave go to ordenan al +2 que no borre la pantalla antes de iniciar el programa. El programa modificado realiza un dibujo ligeramente distinto, superponiéndolo al antiguo. Si lo desea, puede usted continuar editando el programa para añadir cuantos diseños quiera.

Advertencia. Cuando utilice la pantalla inferior, no intente editar instrucciones que tengan una longitud mayor que dos líneas de pantalla, ya que si el editor encuentra una instrucción que tiene su principio o su fin fuera de la pantalla, puede confundirse. (Esto mismo es válido para la pantalla superior, aunque, desde luego, en ese caso no es probable que dicha limitación cause problemas, pues la pantalla es mucho más amplia.

Cuando esté escribiendo, quizá note que las teclas numéricas, al utilizarlas conjuntamente con **[MAYUSC]**, hacen cosas extrañas: **[MAYUSC]** con 5, 6, 7 y 8 mueve el cursor; **[MAYUSC]** con 1 invoca el menú de edición; **[MAYUSC]** con 0 borra un carácter; **[MAYUSC]** con 2 es equivalente a **[BLOQ MAYUSC]**; **[MAYUSC]** con 9 selecciona el modo gráfico. Todas estas funciones están disponibles utilizando las teclas dedicadas a tal fin en el +2, y no hay ninguna razón por la que sea preferible emplear estas alternativas.

Cuando considere que ha comprendido perfectamente cómo funciona el editor, pase al capítulo 8. Experimente con los ejemplos que hemos dado y no tenga miedo de probar algo distinto.

Utilización de 48 BASIC

Temas tratados en este capítulo:

Utilización del +2 como Spectrum de 48K

Activación del modo 48 BASIC

El teclado en 48 BASIC

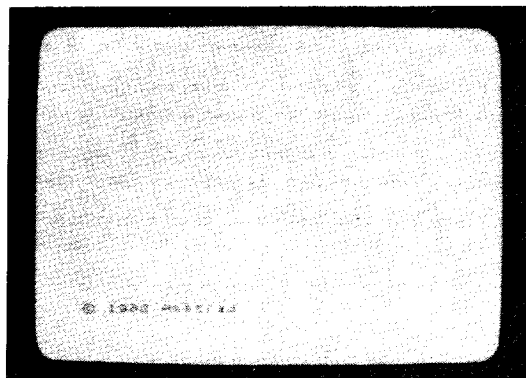
Introducción de un programa

Edición de la línea actual

El +2 tiene la facultad de funcionar exactamente igual que el Spectrum de 48K (o Spectrum +). Esto se logra seleccionando la opción **48 BASIC** en el menú de presentación. En este modo no se puede aprovechar las nuevas características del +2, como son la memoria adicional, el editor de pantalla completa, el sonido en varios canales, los interfaces **RS232/MIDI** y para el **TECLADO NUMÉRICO**. Sin embargo, sí funcionarán los zócalos **JOYSTICK 1** y **JOYSTICK 2**.

El modo 48 BASIC ha sido incluido sólo por razones de compatibilidad: no hay ninguna ventaja en utilizar este modo (en vez de 128 BASIC) para escribir programas, y no lo recomendamos. La siguiente información sólo ha sido incluida a modo de referencia y para quienes estén acostumbrados al Spectrum de 48K y quieran utilizar inmediatamente la máquina sin tener que aprender el manejo del editor de 128 BASIC.

En realidad, hay dos métodos para llevar el +2 al modo 48 BASIC. El primero consiste en seleccionar la opción **48 BASIC** del menú de presentación (si no sabe cómo hacerlo, consulte el capítulo 2). Una vez seleccionada esta opción, verá lo siguiente en la pantalla:



El segundo método permite entrar en el modo 48 BASIC mientras está editando un programa en 128 BASIC. Para ello, estando en el editor de 128 BASIC, escriba:

spectrum (pulse **INTRO**)

El +2 responderá con el mensaje OK y habrá cambiado al modo 48 BASIC conservando el programa actualmente almacenado en la memoria. Una vez en 48 BASIC, ya no hay forma de volver a 128 BASIC, salvo reinicializando el +2 (o apagándolo y volviendo a encenderlo).

La principal diferencia entre las dos versiones de BASIC se encuentra en la forma de introducir y editar programas. En general, los programas de demostración del capítulo 8 funcionarán en ambos modos, pero los relacionados con la música y el 'disco de silicio' solamente lo harán en BASIC 128. Observe también que en 128 BASIC los códigos SPECTRUM y PLAY han reemplazado a los caracteres gráficos definibles por el usuario para las teclas T y U (valores 163 y 164).

Una vez en el modo 48 BASIC, el teclado funciona de la siguiente manera:

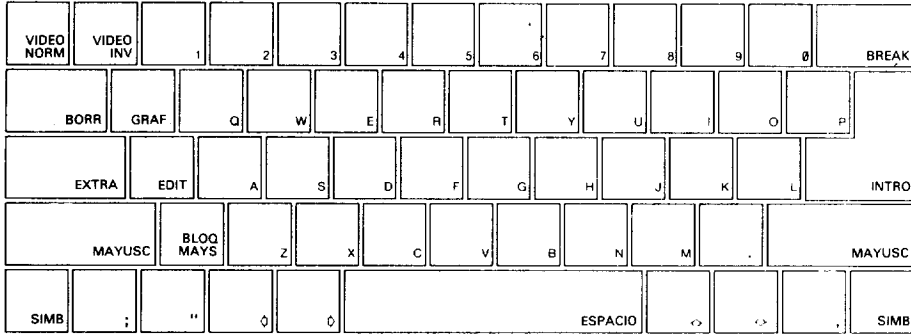
Todos los operadores, órdenes y funciones de BASIC están disponibles *directamente* desde el teclado y no es necesario escribirlos letra a letra. A fin de acomodar todas estas funciones y órdenes, algunas teclas tienen cinco o más significados diferentes, que se obtienen en parte modificando el *estado* de las teclas (es decir, pulsando las teclas en combinación con **MAYUSC** o **SIMB**), y en parte poniendo la máquina en diferentes *modos*. El cursor parpadeante contiene una letra (**K**, **L**, **C**, **E** o **G**) para indicar qué modo está activado en cada momento.

El modo **K** (de *keywords*, 'palabras clave') reemplaza automáticamente al modo **L** (de 'letras') cuando el ordenador está esperando una orden o una línea de programa (en vez de una entrada de datos). Por la posición que ocupa el cursor en la línea, el +2 sabe si debe esperar un número de línea o una palabra clave. El modo **K** se activa al principio de la línea, después del signo de dos puntos (:), salvo cuando éste forma parte de una cadena literal, y tras la palabra clave THEN. Siempre que aparezca el cursor **K**, la siguiente tecla que se pulse será interpretada como palabra clave o como número, según se muestra en la siguiente figura:

VIDEO NORM	VIDEO INV	1	2	3	4	5	6	7	8	9	0	ESPACIO
BORR		PLOT Q	DRAW W	REM E	RUN R	RANDOMIZE T	RETURN Y	IF U	INPUT I	POKE O	PRINT P	
	EDIT	NEW A	SAVE S	DIM D	FOR F	GOTO G	GOSUB H	LOAD J	LIST K	LET L		INTRO
		COPY Z	CLEAR X	CONTINUE C	CLS V	BORDER B	NEXT N	PAUSE M	.			
								ESPACIO	<	<	,	

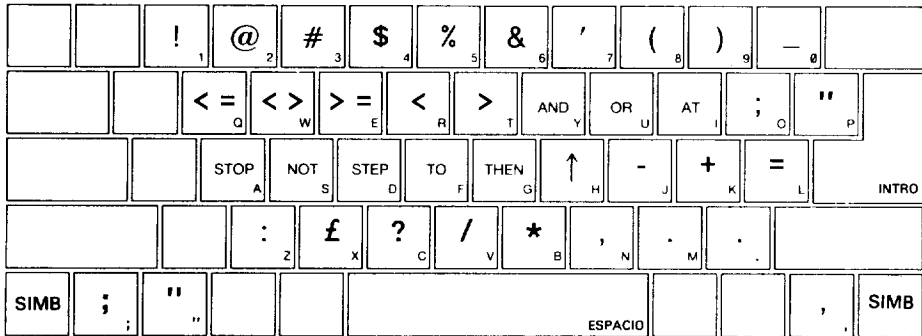
El teclado en el modo **K**

El modo **L** (de 'letras') es el que está activo normalmente (salvo cuando lo está el modo **K**). Siempre que aparezca el cursor **L**, la siguiente tecla pulsada será interpretada de acuerdo con la leyenda que está grabada en la propia tecla; es decir,



El teclado en el modo **L**

En ambos modos, **K** y **L**, la combinación de las diferentes teclas con **SIMB** se interpreta de la siguiente manera:



El teclado con **SIMB** en los modos **K** y **L**

Empleando **MAYUSC** en el modo **L**, las letras minúsculas se convierten en mayúsculas. En el modo **K**, sin embargo, **MAYUSC** no afecta a las palabras clave.

El modo **C** (de capitals, 'mayúsculas') es una variante del modo **L**, en la que todas las letras aparecen como mayúsculas. La tecla **BLOQ MAYS** se utiliza para pasar del modo **L** al **C**, y vice versa.

El modo E (de 'extra') se utiliza para obtener diversos caracteres, principalmente códigos de palabras clave. Este modo se activa pulsando la tecla **EXTRA** y sólo afecta al siguiente carácter (o pulsación de una tecla). Siempre que aparezca el cursor E, la siguiente pulsación será interpretada de esta manera:

		PAPEL AZUL 1	PAPEL ROJO 2	PAPEL MGNTA 3	PAPEL VERDE 4	PAPEL CYAN 5	PAPEL AMRILLO 6	PAPEL BLANCO 7	BRILLO NO 8	BRILLO SI 9	PAPEL NEGRO 0	ESPACIO
		SIN Q	COS W	TAN E	INT R	RND T	STR\$ Y	CHR\$ U	CODE J	PEEK O	TAB P	
EXTRA		READ A	RESTORE S	DATA D	SGN F	ABS G	SQR H	VAL J	LEN K	USR L		INTRO
		LN Z	EXP X	LPRINT C	LLIST V	BIN B	INKEY\$ N	PI M				
												ESPACIO

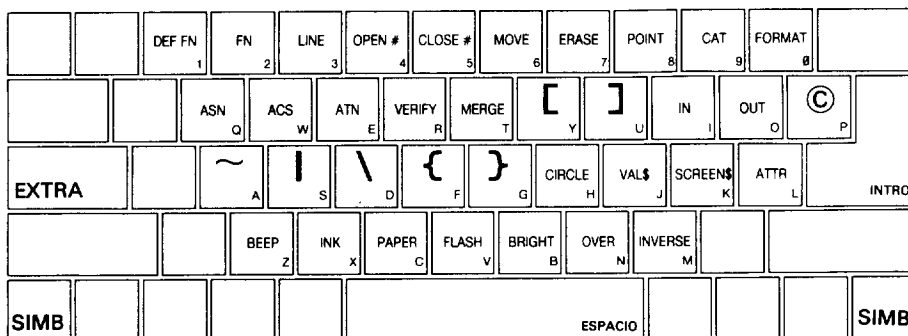
El teclado en el modo E

Al pulsar **MAYUSC** en modo E, la siguiente pulsación será interpretada de esta manera:

		TINTA AZUL 1	TINTA ROJA 2	TINTA MGNTA 3	TINTA VERDE 4	TINTA CYAN 5	TINTA AMRILLO 6	TINTA BLANCA 7	PARP. NO 8	PARP. SI 9	TINTA NEGRA 0	
		ASN Q	ACS W	ATN E	VERIFY R	MERGE T	[Y] U	IN I	OUT O	© P	
EXTRA		~ A	S	\ D	{ F	} G	CIRCLE H	VAL\$ J	SCREEN\$ K	ATTR L		INTRO
MAYUSC		BEEP Z	INK X	PAPER C	FLASH V	BRIGHT B	OVER N	INVERSE M				MAYUSC
												ESPACIO

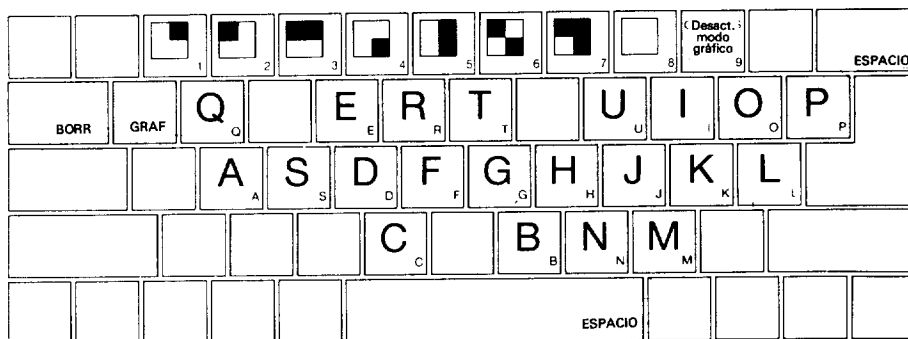
El teclado con **MAYUSC** en el modo E

Al aplicar **SIMB** en modo E, la siguiente pulsación será interpretada de esta manera:



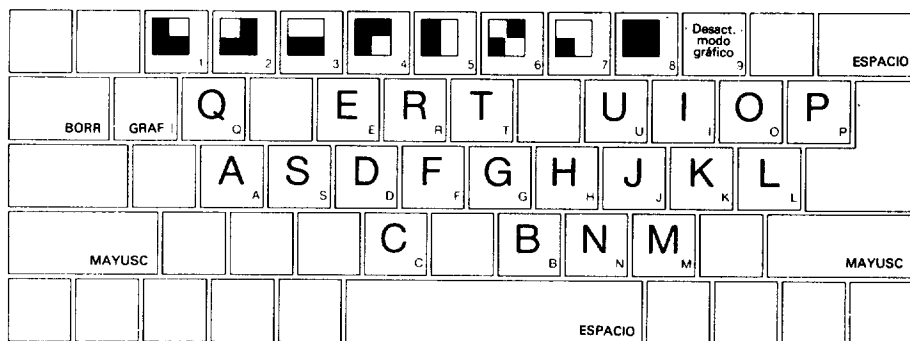
El teclado con **SIMB** en el modo E

El modo G (de 'gráficos') se obtiene pulsando **GRAF** y permanece activado hasta que se vuelve a pulsar la misma tecla (o la del 9). En este modo, cada tecla numérica proporciona un gráfico de mosaico, y cada tecla alfabética (a excepción de V, W, X, Y y Z) da un gráfico definible por el usuario, el cual, mientras no haya sido definido, tendrá forma idéntica al de la letra mayúscula. Siempre que aparezca el cursor G, la siguiente pulsación será interpretada de esta manera:



El teclado en el modo G

Al aplicar **MAYUSC** en modo G, se invierten los gráficos de mosaico (es decir, el color de la tinta pasa a ser el color del papel, y vice versa). En consecuencia, la siguiente pulsación será interpretada de esta manera:



El teclado con **MAYUSC** en el modo G

Si se mantiene pulsada una tecla durante más de 2 o 3 segundos, comenzará a repetirse. Todo lo introducido por el teclado aparece en la mitad inferior de la pantalla según se va escribiendo; cada carácter (símbolo sencillo o código de palabra clave) se inserta justamente donde está el cursor. Éste puede ser desplazado a derecha e izquierda mediante las 'teclas de movimiento del cursor', \rightarrow y \leftarrow , que se encuentran a la izquierda de la barra espaciadora). El carácter que está a la izquierda del cursor se borra con la tecla **BORR**.

Cuando se pulsa **INTRO**, la línea es ejecutada, almacenada como línea de programa o utilizada como entrada de información (para INPUT). No obstante, si la línea contiene un *error de sintaxis*, junto al error aparece un signo de interrogación parpadeante.

Según se va introduciendo líneas de programa, en la mitad superior de la pantalla va apareciendo un listado. La última línea introducida se llama línea actual y está indicada por el símbolo > a la derecha del número de línea. Cualquier línea del programa puede ser seleccionada como línea actual (con objeto de editarla) utilizando las teclas \downarrow y \uparrow (que se encuentran a la derecha de la barra espaciadora). Para editar la línea actual así seleccionada se debe pulsar la tecla **EDIT**. (La edición tiene lugar en la parte inferior de la pantalla.)

Cuando se ejecuta una orden o un programa, el resultado es exhibido en la mitad superior de la pantalla, donde permanece hasta que se pulsa **INTRO** o las teclas de movimiento vertical del cursor, \downarrow y \uparrow . En la parte inferior aparece un informe que consiste en un código (un dígito o una letra) del que hablaremos en el capítulo 8, parte 28. Este informe permanece en la pantalla hasta que se pulsa una tecla y el +2 vuelve al modo K.

Guía completa de programación en BASIC

Parte 1 Introducción

Tanto si ha leído primero el capítulo 6 como si ha venido directamente a éste, debe saber que:

Las *órdenes* son obedecidas inmediatamente.

Las *instrucciones* comienzan con un número de línea y son almacenadas para su uso posterior.

Esta guía de BASIC comienza repitiendo algunas de las cosas tratadas en el capítulo 6, pero de forma más detallada. Al final de algunas secciones hemos incluido ejercicios: le recomendamos que no los ignore, pues muchos de ellos ilustran conceptos que sólo han sido mencionados de pasada en el texto. Écheles un vistazo y trabaje con los que más le interesen, o con los que traten de alguna cuestión que el texto no le haya dejado completamente clara. En cualquier caso, no deje de experimentar con el ordenador. Siempre que se pregunte qué haría el +2 si usted escribiese tal o cual cosa, la respuesta es sencilla: pruebe y lo verá. Recuerde que, escriba lo que escriba, no puede dañar al +2.

El teclado

VIDEO NORM	VIDEO INV		1	2	3	4	5	6	7	8	9	0	BREAK
BORR	GRAF	Q	W	E	R	T	Y	U	I	O	P		
EXTRA	EDIT	A	S	D	F	G	H	J	K	L		INTRO	
MAYUSC	BLOQ MAYS	Z	X	C	V	B	N	M	.			MAYUSC	
SIMB	:	"	◊	◊	ESPACIO				◊	◊	,	SIMB	

Los caracteres utilizados en el +2 no son solamente símbolos simples (letras, dígitos, etc.) sino también símbolos compuestos (palabras clave, nombres de funciones, etc.). Todo debe ser escrito completamente; y en la mayor parte de los casos es indiferente hacerlo en mayúsculas o minúsculas. En el teclado hay tres clases de teclas: las de letras y números (llamadas alfanuméricas), las de símbolos (signos de puntuación) y las teclas de control (tales como **MAYUSC**, **BORR**, etc.).

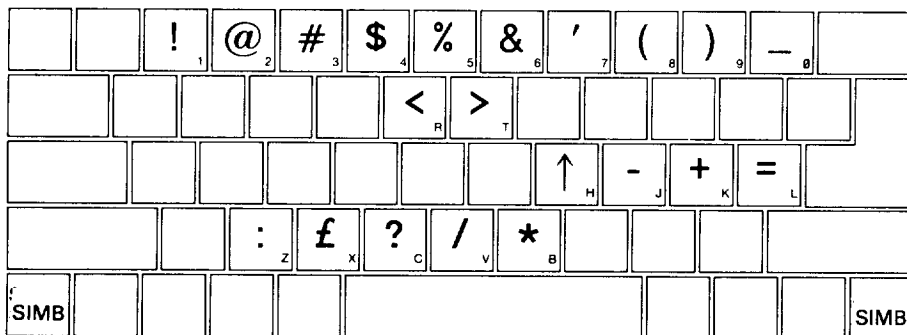
Las teclas alfanuméricas son las más frecuentemente utilizadas en BASIC. Cuando se pulsa una tecla alfabética, en la pantalla aparece una letra minúscula junto con un cuadrado parpadeante (cuyo color alterna entre azul y blanco), llamado *cursor*. Para obtener la mayúscula se debe mantener pulsada la tecla **MAYUSC** al tiempo que se escribe la letra.

Si desea escribir con mayúsculas continuamente, pulse una vez la tecla **BLOQ MAYS**: todas las teclas alfabéticas que pulse en lo sucesivo producirán letras mayúsculas. Para volver a las minúsculas, pulse otra vez **BLOQ MAYS**.

Para escribir los símbolos que aparecen en las teclas alfanuméricas, es decir,

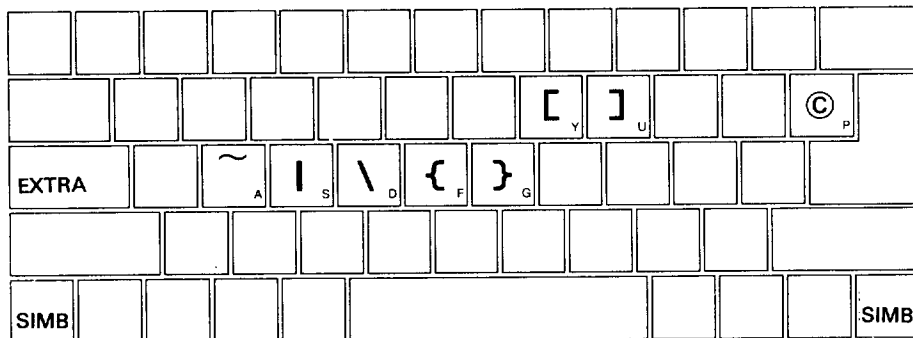
! @ # \$ % & ' () _ < > ↑ - + = : £ ? / *

pulse la tecla correspondiente en combinación con **SIMB** (véase el diagrama siguiente).



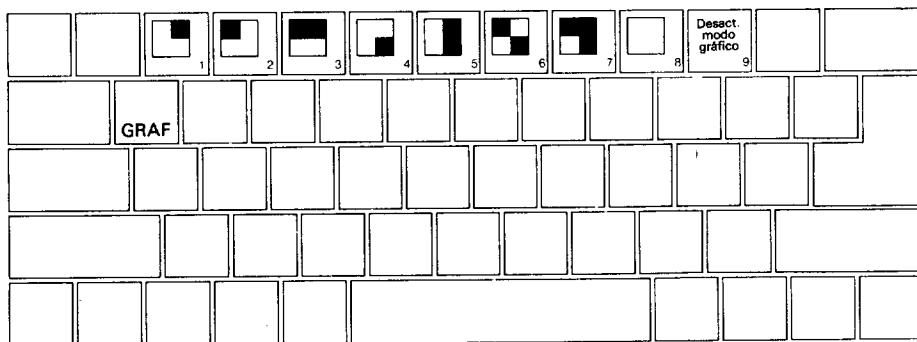
Símbolos disponibles con **SIMB**

Además se puede obtener los símbolos [] © ~ | \ { y } pulsando una vez la tecla **EXTRA** y pulsando luego una tecla alfabética en combinación con **SIMB** (véase el diagrama siguiente).



Símbolos disponibles con **SIMB** en modo **EXTRA**

Para activar el modo de gráficos, pulse una vez la tecla **GRAF**. Los gráficos de mosaico (véase el siguiente diagrama) se obtienen entonces pulsando las teclas numéricas (excepto 9 y 0). Combinado **MAYUSC** con las teclas numéricas citadas se obtiene los mismos gráficos de mosaico, pero con los colores invertidos. Pulsando las teclas alfabéticas (excepto T, U, V, W, X, Y y Z) se obtiene los gráficos definidos por el usuario.



Gráficos de mosaico disponibles en modo **GRAF**

Si tecla se mantiene pulsada una tecla durante más de 2 o 3 segundos, comenzará a repetirse. A medida que se va pulsando teclas, se forma una línea en la pantalla. Dicho sea de paso, por *línea* entendemos una línea de BASIC, la cual puede ocupar varias líneas físicas en la pantalla. La línea puede ser recorrida utilizando las teclas de movi-

miento del cursor: \leftarrow , \rightarrow , \uparrow , \downarrow . Si la parte de la línea hacia la que se mueve el cursor se encuentra fuera de la pantalla, entonces el texto se moverá hacia arriba o hacia abajo para hacerla visible. Cualquier carácter que se escriba será insertado en la posición del cursor. Pulsando **BORR** se borra el carácter que está a la izquierda del cursor. En cuanto se pulsa **INTRO**, o si se intenta sacar el cursor de la línea, el ordenador comprueba si la línea tiene sentido. Si lo tiene, genera un pitido agudo y obedece la línea inmediatamente, o bien la almacena como parte de un programa. Si la línea contiene un error, el ordenador emite un pitido grave y lleva el cursor a la zona en la que piense que se encuentra dicho error (además, el color del cursor cambia a rojo). Es imposible salir de una línea que contenga un error: el +2 siempre volverá a llevar el cursor a esa línea.

La pantalla

La pantalla consta de 24 líneas (de 32 caracteres cada una) y está dividida en dos partes. La más amplia (la superior) tiene a lo sumo 22 líneas y puede mostrar tanto el listado como los resultados del programa. Es la que generalmente se utiliza para editar líneas. Cuando la impresión en la *pantalla superior* ha llegado a su borde inferior, el contenido se desplaza una línea hacia arriba. No obstante, si este desplazamiento implica la pérdida de una línea que aún no se ha tenido la oportunidad de leer, el +2 se detiene y emite el mensaje:

scroll?

Si se pulsa cualquier tecla (excepto N, **BREAK** o la barra espaciadora), el desplazamiento en vertical continúa.

Si se pulsa la tecla N, **BREAK** o la barra espaciadora, el programa se detiene y emite el mensaje:

D BREAK – CONT repeats

La parte más pequeña (inferior) de la pantalla se utiliza para editar programas cortos, introducir datos y órdenes directas (cuando no conviene usar la pantalla superior; por ejemplo, en programas de gráficos) y para exhibir informes.

Introducción de programas

Si el programa que está siendo introducido sobrepasa el tamaño de la pantalla, el +2 intentará presentar el área de mayor interés (generalmente, la última línea introducida junto con las cercanas a ella). No obstante, usted puede hacer que el ordenador muestre otra área del programa especificándola en la orden:

LIST xxx

donde xxx es un número de línea. Esta orden hace que el +2 exhiba la zona del programa especificada.

Cuando se ejecuta una orden o un programa, el resultado se muestra en la pantalla superior, donde permanece cuando el programa termina (hasta que se pulsa una tecla). Si el programa está siendo editado en la pantalla inferior, los resultados exhibidos en la pantalla superior permanecerán en ella hasta que se escriba sobre ellos, hasta que desaparezcan por efecto del desplazamiento en vertical o hasta que se ejecute una orden CLS. La pantalla inferior muestra un informe que consiste en un código (dígito o letra), cuyo significado explicaremos en la parte 28 de este capítulo. Ese informe permanecerá en la pantalla inferior hasta que se pulse una tecla.

Mientras el +2 está ejecutando un programa de BASIC, la tecla **BREAK** es comprobada de vez en cuando; concretamente, al final de cada sentencia, durante la utilización del magnetófono y de la impresora y mientras se interpreta la música. Si el +2 detecta que se ha pulsado **BREAK**, detiene la ejecución del programa y se emite el informe:

D o bien L

y entonces el programa puede ser editado.

Parte 2

Conceptos sencillos de programación

Temas tratados:

Programas
Números de línea
Edición de programas utilizando las teclas del cursor
RUN, LIST
GO TO, CONTINUE, INPUT, NEW, REM, PRINT
Detención de un programa

Escriba las dos primeras líneas de un programa que, cuando esté completo, calculará y exhibirá en la pantalla la suma de dos números:

```
20 print a (pulse INTRO)  
10 let a=10 (pulse INTRO)
```

Observe que la pantalla muestra lo siguiente:

```
10 LET a=10  
■  
20 PRINT a
```

Como hemos visto antes, puesto que estas líneas comenzaban con números, el +2 no las obedeció inmediatamente, sino que las almacenó como líneas de programa. Este ejemplo muestra también que los números de línea rigen el *orden* en que las líneas de programa van a ser ejecutadas; tal y como puede ver en la pantalla, el +2 ordena todas las líneas cada vez que se introduce una nueva.

Observe además que, aunque habíamos escrito cada línea con letras minúsculas, el ordenador ha convertido a mayúsculas las palabras clave (PRINT y LET) en cuanto ha aceptado la línea. De ahora en adelante mostraremos en letras mayúsculas la información que se debe escribir; sin embargo, usted puede continuar escribiéndola en minúsculas.

Hasta ahora sólo hemos introducido un número de los dos que tenemos que sumar, así que escriba:

```
15 LET b=15 (pulse INTRO)
```

Ahora debemos transformar la línea 20 del siguiente modo:

```
20 PRINT a+b
```

Se podría escribir la nueva línea completa, pero es mucho más cómodo situar el cursor justamente detrás de la a y luego escribir:

+b (no pulse todavía **INTRO**)

La línea debería quedar así:

```
20 PRINT a+b
```

Ahora pulse **INTRO**; el cursor se desplaza a la línea de abajo, de forma tal que en la pantalla se ve lo siguiente:

```
10 LET a=10
15 LET b=15
20 PRINT a+b
```

■

Para ejecutar el programa dé la orden:

```
RUN (pulse INTRO)
```

La suma aparecerá en la pantalla.

Ejecute nuevamente el programa y después escriba:

```
PRINT a,b (pulse INTRO)
```

Observe que las variables continúan en la memoria, a pesar de que el programa ya ha terminado.

Si introducimos una línea por error, por ejemplo

```
12 LET b=8
```

para borrarla basta con escribir su número y pulsar **INTRO**:

```
12 (pulse INTRO)
```

La línea 12 desaparece del listado y el cursor se sitúa en el lugar en donde antes estaba la línea.

Ahora escriba:

```
30 (pulse INTRO)
```

El +2 busca la línea 30; como no hay ninguna con ese número, va a parar al final del programa. Por eso el cursor queda situado justamente detrás de la última línea. Si escribimos un número de línea que no existe, el +2 sitúa el cursor en el lugar en que la línea debería estar si realmente existiese. Ésta puede ser una forma útil de moverse en programas largos; pero tenga cuidado porque también puede ser muy peligroso. En efecto, si la línea ya existe antes de que escribamos el número y pulsemos **INTRO**, ciertamente no existirá después.

Para listar un programa en la pantalla basta con escribir:

LIST (pulse **INTRO**)

A veces se prefiere listar solamente desde cierta línea en adelante (en particular cuando se trabaja con programas largos). Para ello se escribe el número de línea adecuado detrás de la orden LIST.

Escriba:

LIST 15 (pulse **INTRO**)

y compruebe el resultado.

Observe cómo pudimos insertar la línea 15 entre las otras dos al escribir el programa anterior. Esto habría sido imposible si sus números hubiesen sido 1 y 2, en vez de 10 y 20. Por esta razón, siempre es conveniente dejar intervalos prudentiales entre los números de línea.

(Tenga en cuenta que todos los números de línea tiene que ser enteros y estar entre 1 y 9999.)

Si en algún momento se da usted cuenta de que no ha dejado suficiente espacio entre los números de línea, puede invocar el menú de edición y reenumerar el programa. Para ello pulse la tecla **EDIT** y seleccione la opción **Renumerar** del menú. El programa queda reenumerado a partir de la línea 10 y con intervalo de 10 unidades entre líneas sucesivas. Pruébelo y observe cómo cambian los números.

Ahora vamos a utilizar la orden NEW de BASIC. Esta orden borra el programa almacenado en el +2, junto con todas sus variables. Es la orden que se debe dar al ordenador cuando se quiere empezar partiendo de cero. Escriba:

NEW

y pulse **INTRO**. De ahora en adelante ya no escribiremos 'pulse **INTRO**' cada vez que usted deba pulsar esta tecla, pero no se olvide de hacerlo.

Con el menú de presentación en la pantalla, active BASIC seleccionando la opción **128 BASIC**.

A continuación transcriba cuidadosamente el siguiente programa, que convierte grados Fahrenheit en grados centígrados:

```
10 REM Conversión de temperatura
20 PRINT "Grados F","Grados C"
30 PRINT
40 INPUT "Introduzca grados F",f
50 PRINT f,(f-32)*5/9
60 GO TO 40
```

Observe que, si escribe toda la línea 10 en minúsculas, BASIC sólo convierte en mayúsculas la palabra 'REM', ya que ésta es la única palabra clave que hay en la línea. Además, a pesar de que nosotros hemos escrito **GO TO** en forma de dos palabras separadas por un espacio, usted puede escribirlas juntas (**GOTO**).

Ahora ejecute el programa. Verá que las cabeceras son escritas en la pantalla superior, como consecuencia de la línea 20. Pero ¿qué ha hecho la línea 10? Parece que el +2 la ha ignorado completamente; en efecto, eso es lo que ha hecho. **La palabra REM** de la línea 10 es en realidad una abreviatura de *remark* ('observación') y su único efecto es permitir que usted haga anotaciones al programa. Una sentencia **REM** consiste en la palabra **REM** seguida de cualquier cosa. El ordenador ignora todo lo que se escriba a la derecha de **REM** hasta el final de la línea.

El ordenador ya ha llegado a la orden **INPUT** de la línea 40 y está esperando que escriba usted un valor para la variable **f** (lo cual se confirma por el hecho de que el cursor está en la pantalla inferior).

Introduzca un número. El +2 muestra el resultado del cálculo y queda a la espera de otro número. Esto se debe a que la instrucción de la línea 60 es **GO TO 40**, que en castellano se puede leer 'ir a 40'; en otras palabras, 'en vez de salir del programa y detenerte, salta hacia atrás, a la línea 40, y continúa a partir de ella'.

Así pues, introduzca otra temperatura, luego otra, Quizá se pregunte usted si la máquina llegará a cansarse de hacer siempre lo mismo; pues no, no se cansa nunca. Para detener el programa debe hacer lo siguiente: en vez de introducir otro número, pulse la tecla **A** en combinación con **[SIMB]**. Esto hace que aparezca la palabra **STOP**; cuando usted pulse **[INTRO]**, el +2 responderá con el informe:

```
H STOP in INPUT, 40:1
```

Este informe indica que el programa se ha detenido en una instrucción **INPUT**, que es la primera (:1) de la línea 40.

Si desea reanudar la ejecución del programa, escriba:

```
CONTINUE
```

y el +2 le pedirá otro número.

Cuando se le da la orden CONTINUE, el +2 recuerda el número de línea incluido en el último informe que ha emitido (siempre que no fuera 0 OK) y salta a esa línea, que en nuestro caso es la 40 (la de la orden INPUT).

Detenga nuevamente el programa y cambie la línea 60 por:

```
60 GO TO 31
```

No habrá una diferencia perceptible en la ejecución del programa, porque si el número de línea de una orden GO TO se refiere a una línea inexistente, el salto se efectúa hasta la siguiente línea *posterior* al número especificado. Análogamente ocurre con la orden RUN (de hecho, RUN equivale a RUN 0).

Ahora introduzca números hasta que la pantalla superior empiece a llenarse. Cuando ya esté llena, el +2 desplazará hacia arriba todo el contenido de la pantalla superior para hacer sitio para nuevas líneas, y el encabezamiento se saldrá de la pantalla.

Si lo desea, ya puede detener el programa, tal y como lo hizo antes y activar el editor pulsando **INTRO**.

Observe la sentencia PRINT de la línea 50. En ella la coma (,) es muy importante.

Las comas sirven para hacer que la impresión comience en el margen izquierdo o en centro de la pantalla, según los casos. Así, en la línea 50 la coma hace que la temperatura centígrada se escriba a partir del centro de la línea.

Por otro lado, punto y coma (;) se utiliza para hacer que el siguiente número o cadena literal se imprima inmediatamente después del precedente.

Otro signo de puntuación que puede usted utilizar en las órdenes PRINT es el apóstrofo ('), el cual hace que lo que se escriba a continuación aparezca al principio de la línea siguiente. Esto mismo ocurre por defecto (es decir, cuando no se especifica otra cosa) al final de cada orden PRINT. Si desea inhibir el salto a la línea siguiente, puede poner una coma o un punto y coma al final de la sentencia PRINT. Para ver cómo funciona, reemplace sucesivamente la línea 50 por cada una de éstas:

```
50 PRINT f,  
50 PRINT f;  
50 PRINT f
```

y ejecute el programa cada vez para observar la diferencia.

La línea que termina en coma lo escribe todo en dos columnas; la línea del punto y coma lo coloca todo junto; la línea que no lleva coma ni punto y coma escribe cada número en una nueva línea (esto mismo se conseguiría con PRINT f').

Recuerde siempre la diferencia entre la coma y el punto y coma en las órdenes PRINT y no los confunda con los dos puntos (:), los cuales se utilizan como separadores entre órdenes incluidas en una misma línea; por ejemplo:

```
PRINT f: GO TO 40
```

Ahora escriba estas líneas adicionales:

```
100 REM Este programa recuerda su nombre
110 INPUT n$
120 PRINT "Hola ";n$;"!"
130 GO TO 110
```

Éste programa es independiente del anterior, pero usted puede guardar ambos en el +2 al mismo tiempo. Para ejecutar el nuevo programa dé la orden:

RUN 100

Puesto que el programa espera que usted introduzca una *cadena literal* (o sea, un carácter o un grupo de caracteres) en vez de un número, escribirá el cursor entre comillas (" ") como recordatorio. Así pues, escriba un nombre y pulse **[INTRO]**.

La próxima vez volverán a aparecer las comillas, pero usted no tiene que utilizarlas si no lo desea. Por ejemplo, pruebe lo siguiente: borre las comillas pulsando dos veces **[←]** y dos veces **[BORR]**; después escriba:

n\$

Al no haber comillas, el +2 sabe que tiene que realizar algún cálculo (en este caso, averiguar el valor de la variable literal n\$, que será el nombre introducido la vez anterior). De este modo, la sentencia INPUT actúa como LET n\$=n\$, así que el valor de n\$ permanece inalterado.

Cuando quiera detener el programa, borre las comillas, pulse A en combinación con **[SIMB]** y finalmente pulse **[INTRO]**.

Ahora examinemos la instrucción RUN 100, que provoca el salto a la línea 100 e inicia la ejecución del programa a partir de ella. Usted podría preguntarse: "¿cuál es la diferencia entre RUN 100 y GO TO 100?" Pues bien, RUN 100 empieza por borrar la pantalla y todas las variables, y luego ya realiza la misma función que GO TO 100. En cambio, GO TO 100 no borra nada. Hay ocasiones en que se necesita ejecutar un programa sin borrar las variables; en tales casos es necesario usar GO TO, porque RUN podría ser desastrosa. Por consiguiente, no conviene que se acostumbre a escribir RUN sistemáticamente para poner en marcha los programas.

Otra diferencia obvia consiste en que se puede escribir RUN sin número de línea (y entonces la ejecución comienza a partir de la primera línea del programa), mientras que GO TO siempre tiene que ir seguida de un número de línea.

Tanto este programa como el de conversión de la temperatura se detienen solamente cuando usted pulsa **[SIMB]** A en la línea de entrada. A veces se escribe por error un programa que no se detiene por sí mismo ni puede ser interrumpido por este procedimiento. Por ejemplo, escriba:

```
200 GO TO 200
RUN 200
```

Aunque la pantalla está en blanco, el programa sí está funcionando, ejecutando la línea 200 una y otra vez. Aparentemente va a continuar así para siempre, a menos que desenchufe el cable o pulse el botón **RESET**. Sin embargo, hay un remedio menos drástico: pulsar la tecla **BREAK**. El programa se detiene y el ordenador emite el informe:

L BREAK into program, 200:1

Cada vez que termina de ejecutar una sentencia, el ordenador comprueba si está pulsada la tecla **BREAK**; si lo está, interrumpe el programa. También se puede usar esta tecla para interrumpir las operaciones con el magnetófono, la impresora y algunos otros periféricos que pueden ser conectados al +2. En estos casos el informe es distinto:

D BREAK – CONT repeats

En esta situación (y también en muchas otras), la instrucción **CONTINUE** repite la sentencia en la que el programa fue interrumpido y continúa directamente con la siguiente.

Ejecute otra vez el programa del 'nombre' y, cuando le pida la entrada, escriba:

n\$ (después de borrar las comillas)

Puesto que n\$ es una variable que no ha sido definida, el ordenador emite el siguiente mensaje de error:

2 Variable not found, 110:1 ('Variable no encontrada')

Defina la variable escribiendo la orden:

1.ET n\$="cara de pez"

(que produce el informe 0 OK, 0:1) y luego escriba:

CONTINUE

Como puede comprobar, ahora ya se puede usar n\$ como dato de entrada sin ningún problema.

En este caso, **CONTINUE** provoca el salto a la orden **INPUT** de la línea 110; ignora el informe producido por la sentencia **LET** porque era OK y salta a la orden aludida en el informe anterior, la 110. Esta característica puede ser muy útil, pues permite «reparar» un programa que se ha detenido a causa de un error y continuar (**CONTINUE**) a partir de donde se produjo la interrupción.

Como dijimos antes, el informe **L BREAK into program** es especial porque, tras él, **CONTINUE** no repite la orden en la que el programa se detuvo.

Ya hemos visto las sentencias PRINT, LET, INPUT, RUN, LIST, GO TO, CONTINUE, NEW y REM. Todas ellas pueden ser utilizadas como órdenes directas o en líneas de programa. (Esto es válido para casi todas las órdenes del Spectrum BASIC; sin embargo, no es frecuente incluir RUN, LIST, CONTINUE y NEW en las líneas de programa.)

Ejercicios

1. Ponga una sentencia LIST en un programa, de forma que el programa se liste a sí mismo al ejecutarlo.
2. Escriba un programa que pida los precios de los artículos y escriba el importe del IVA (12 %). Utilice sentencias PRINT mediante las que el ordenador explique qué va a hacer y pida los precios con extravagante amabilidad. Modifique luego el programa de forma que éste pregunte el porcentaje del impuesto (con lo cual el programa valdrá también para artículos exentos de IVA y para los que estén sujetos a porcentajes diferentes).
3. Elabore un programa que escriba la suma actualizada de los números que el usuario vaya introduciendo. (*Sugerencia:* cree una variable que se llame total, dándole inicialmente el valor 0; utilice otra variable que se llame número; en cada vuelta, asigne valor a la variable número en una instrucción INPUT y súmela a total; escriba los valores de ambas y pase a la siguiente vuelta.)
4. ¿Cuál sería el efecto de CONTINUE y NEW dentro de un programa? ¿Se le ocurre alguna aplicación práctica?

Parte 3

Decisiones

Temas tratados:

```
CLS, IF, STOP  
=, <, >, <=, >=, <>
```

Todos los programas que hemos visto hasta ahora eran bastante predecibles: obedecían sucesivamente las instrucciones y volvían al principio. Esto no es lo más útil que puede hacer el ordenador por nosotros; en la práctica, lo que esperamos de un ordenador es que sea capaz de tomar decisiones y obrar en consecuencia. En BASIC, la instrucción que lleva a cabo la toma de decisiones tiene la siguiente forma: 'IF (si) algo es cierto (o falso) THEN (entonces) haz tal cosa'.

Veamos un ejemplo. Dé la orden NEW para borrar el programa anterior de la memoria del +2, active 128 BASIC y transcriba y ejecute el siguiente programa (que, evidentemente, ha sido ideado para que jueguen dos personas):

```
10 REM Adivinar un número  
20 INPUT "Introduzca un número secreto",a: CLS  
30 INPUT "Adivine el número",b  
40 IF b=a THEN PRINT "Correcto!": STOP  
50 IF b<a THEN PRINT "Ese es demasiado pequeño; vuelva a intentarlo"  
60 IF b>a THEN PRINT "Ese es demasiado grande; vuelva a intentarlo"  
70 GO TO 30
```

Observe que la orden CLS (en la línea 20) significa 'borrar la pantalla' (en inglés, clear screen). Nosotros la hemos incluido en este programa para impedir que la otra persona vea el número secreto una vez que éste ha sido introducido.

Como puede ver, la sentencia IF tiene la forma:

```
IF condición THEN xxx
```

donde xxx representa una orden (o una secuencia de órdenes separadas por signos de dos puntos). La *condición* es algo que el ordenador tiene que evaluar y que puede dar como resultado 'verdadero' o 'falso'. Si resulta 'verdadero', las sentencias del resto de la línea (posteriores a THEN) serán ejecutadas; de lo contrario, el ordenador las ignora y salta a la siguiente instrucción.

Las condiciones más sencillas consisten en la comparación de dos números o dos cadenas. Por ejemplo, se puede comparar dos números para averiguar si son iguales o si

uno es mayor que el otro, o comparar dos cadenas para ver cuál está antes en el orden alfabético. En las comparaciones se utiliza los símbolos =, <, >, <=, >= y <>, que son los denominados *operadores de relación*:

= se lee 'es igual a'
< se lee 'es menor que'
> se lee 'es mayor que'
<= se lee 'es igual o menor que'
>= se lee 'es igual o mayor que'
<> se lee 'es distinto de'

En el programa que acabamos de escribir, la línea 40 compara a con b. Si son iguales, el programa es interrumpido por la orden STOP. El mensaje que aparece en la pantalla inferior es:

9 STOP statement, 40:3

el cual indica que la tercera sentencia (es decir, la orden STOP) de la línea 40 hizo que el programa se detuviera.

La línea 50 averigua si b es menor que a; la línea 60, si b es mayor que a. Si una de estas condiciones es verdadera, el programa escribe el comentario apropiado y continúa en la línea 70, desde la cual salta a la 30, donde se reinicia el proceso.

Para terminar, observe que en algunas versiones de BASIC (no en el Spectrum BASIC) la sentencia IF puede tener la forma:

IF *condición* THEN *número de línea*

En Spectrum BASIC, esto equivale a:

IF *condición* THEN GO TO *número de línea*

Ejercicio

1. Pruebe el siguiente programa:

```
10 LET a=1
20 LET b=1
30 IF a>b THEN PRINT a;" es mayor"
40 IF a<b THEN PRINT b;" es mayor"
```

Antes de ejecutarlo, trate de adivinar qué va a aparecer en la pantalla.

Parte 4

Bucles

Temas tratados:

FOR, NEXT
TO, STEP

Supongamos que usted desea escribir un programa que capte cinco números por el teclado y los sume.

Un método podría ser el siguiente (no copie esto, a menos que quiera hacer prácticas de mecanografía):

```
10 LET total=0
20 INPUT a
30 LET total=total+a
40 INPUT a
50 LET total=total+a
60 INPUT a
70 LET total=total+a
80 INPUT a
90 LET total=total+a
100 INPUT a
110 LET total=total+a
120 PRINT total
```

Este método es pésimo. El programa resulta más o menos manejable cuando se trata de sumar cinco números, pero imagínese que hubiera que sumar diez (o cien).

Lo que vamos a hacer es usar una variable para contar hasta 5 y luego detener el programa (éste sí debe copiarlo):

```
10 LET total=0
20 LET contador=1
30 INPUT a
40 REM contador indica el número de veces que se ha captado el valor de a hasta el momento
50 LET total=total+a
60 LET contador=contador+1
70 IF contador<=5 THEN GO TO 30
80 PRINT total
```

Observe qué fácil sería ahora modificar la línea 70 para que el programa sumara diez números, o incluso cien.

Este método es tan útil que hay dos órdenes especiales para hacerlo más fácil: la orden FOR y la orden NEXT, las cuales siempre han de ser utilizadas conjuntamente. Introduciéndolas, el programa anterior se convierte en:

```
10 LET total=0
20 FOR c=1 TO 5
30 INPUT a
40 REM c es el número de veces que se ha captado el valor de a hasta ahora
50 LET total=total+a
60 NEXT c
80 PRINT total
```

(Para obtener este programa a partir del anterior, basta con editar las líneas 20, 40 y 60 y borrar la línea 70.)

Observe que hemos cambiado contador por c. Esto ha sido necesario porque el nombre de la variable de control de un bucle FOR...NEXT tiene que constar de una sola letra.

El mecanismo de este programa es el siguiente: c va tomando sucesivamente los valores 1 (*valor inicial*), 2, 3, 4 y 5 (*límite*), y para cada uno de ellos el ordenador ejecuta las líneas 30, 40 y 50. Después, cuando c ya ha recorrido sus cinco valores, se ejecuta la línea 80.

Ahora intente resolver el ejercicio 2 del final de esta Parte 4, que hace referencia al programa anterior.

Un refinamiento adicional de la estructura FOR...NEXT consiste en que la variable de control no tiene que crecer necesariamente de uno en uno, sino que usted puede cambiar ese 1 por cualquier otro número sin más que incluir la cláusula STEP (paso) en la orden FOR. La forma más general de una orden FOR es, pues,

FOR variable de control = valor inicial TO límite STEP paso

donde la *variable de control* es una sola letra, y el *valor inicial*, el *límite* y el *paso* son «expresiones» (es decir, cualquier cosa que el +2 pueda evaluar y cuyo valor sea un número: constantes numéricas, sumas de números, variables numéricas, etc.). Pues bien, si reemplazamos la línea 20 del programa por:

```
20 FOR c=1 TO 5 STEP 3/2
```

la variable de control se verá incrementada en 3/2 cada vez que se ejecute el bucle FOR. Observe que podríamos haber puesto STEP 1.5, o haber asignado el valor del paso a una variable (por ejemplo, p) y luego haber especificado STEP p.

Con esta modificación de arriba, la variable c tomará los valores 1, 2.5 y 4. Observe que no hay por qué limitarse a números enteros y que, por otra parte, no es necesario

que la variable de control llegue a coincidir exactamente con el valor del límite: el bucle se reitera mientras el valor de la variable de control sea igual o menor que el del límite.

Intente resolver el ejercicio 3 del final de esta Parte 4, que hace referencia al programa anterior.

Los valores del paso pueden ser negativos en vez de positivos. Pruebe el siguiente programa, que escribe los números del 1 al 10 en orden inverso. (Acuérdese siempre de dar la orden NEW antes de empezar a escribir un programa nuevo.)

```
10 FOR n=10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

Dijimos antes que el programa continúa realizando bucles mientras la variable de control sea igual o menor que el límite, pero eso sólo es válido cuando no se incluye la cláusula STEP o cuando el valor del *paso* es positivo. Si el paso es negativo, la regla es como sigue: el bucle se repite mientras la variable de control sea igual o mayor que el límite.

Intente resolver los ejercicios 4 y 5 del final de esta Parte 4, que hacen referencia al programa anterior.

Hay que tener cuidado cuando se utiliza dos bucles FOR...NEXT anidados (es decir, uno dentro de otro). Pruebe el siguiente programa, que escribe todos los valores posibles de las fichas de dominó:

```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;" ":"n;" ";
40 NEXT n
50 PRINT
60 NEXT m
```

} bucle n

} bucle m

Observe que el bucle *n* está completamente dentro del bucle *m*. Esto significa que el anidamiento es correcto.

Lo que siempre se debe evitar es tener dos bucles FOR...NEXT solapados, como ocurre en el siguiente programa:

```
5 REM Este programa es incorrecto
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;" ":"n;" ";
40 NEXT m
50 PRINT
60 NEXT n
```

} bucle m

} bucle n

En resumen, si un programa contiene dos bucles FOR...NEXT, éstos deben estar, o bien uno dentro del otro, o bien completamente separados.

Otra cosa que hay que evitar es saltar al interior de un bucle FOR...NEXT desde fuera de él. La variable de control sólo se prepara correctamente cuando se ejecuta su sentencia FOR; si el ordenador no ejecuta una sentencia FOR, la sentencia NEXT lo dejará completamente confundido. La consecuencia más probable es el mensaje de error NEXT without FOR ('NEXT sin FOR'), o bien Variable not found ('Variable no encontrada').

Nada impide utilizar un bucle FOR...NEXT en una orden directa. Por ejemplo, pruebe la siguiente:

```
FOR m=0 TO 10: PRINT m: NEXT m
```

Hay una forma (un tanto artificial) de repetir automáticamente una orden directa. (Recuerde que GO TO necesita un número de línea, y eso es lo que una orden directa no puede tener). Por ejemplo,

```
FOR m=0 TO 1 STEP 0: INPUT a: PRINT a: NEXT m
```

El hecho de que el paso sea 0 hace que la orden se repita indefinidamente.

Pero este método no es muy recomendable: si se produce un error se pierde la orden y hay que volver a escribirla; además, CONTINUE no funcionará en este caso.

Ejercicios

1. Asegúrese de que entiende perfectamente que una variable de control no sólo tiene un nombre y un valor, como una variable ordinaria, sino además un límite, un paso y una conexión con la sentencia FOR correspondiente. Por otra parte, cuando se ejecuta la sentencia FOR, toda esta información se encuentra disponible y es suficiente para que la sentencia NEXT sepa cómo tiene que modificar el valor de la variable, si debe o no repetir el bucle y, de ser así, a qué línea tiene que saltar.
2. Ejecute el tercer programa de esta sección y luego dé la orden:

```
PRINT c
```

¿Por qué la respuesta es 6 y no 5?

(*Solución:* la orden NEXT de la línea 60 es ejecutada cinco veces, y en cada una de ellas se suma 1 a c. La última vez c se convierte en 6; por eso la orden NEXT decide no repetir el bucle, sino continuar, ya que c ha sobrepasado el límite.)

¿Qué ocurre si ponemos STEP 2 al final de la línea 20?

-
3. Modifique el tercer programa, de forma que, en lugar de sumar automáticamente los cinco números, pregunte al usuario cuántos números quiere sumar. Al ejecutar este programa, ¿qué ocurriría si usted respondiese con un 0 (lo que significaría que no quiere sumar ningún número)? ¿Cree que esto le causaría algún problema al +2, aun estando claro lo que usted quiere decir?
 4. En la línea 10 del cuarto programa de esta sección, cambie 10 por 100 y ejecute el programa. En la pantalla aparecerán los números del 100 al 79 y luego, en la pantalla inferior, podrá ver el mensaje scroll? ('¿desplazar?'). El +2 le da así la oportunidad de observar los números antes de que sean desplazados hacia arriba. Si pulsa N, BREAK o la barra espaciadora, el programa se detendrá con el mensaje D BREAK - CONT repeats. Si pulsa cualquier otra tecla, el programa escribirá otras 22 líneas y volverá a preguntarle si desea otro desplazamiento vertical.
 5. Borre la línea 30 del cuarto programa. Cuando ejecute la nueva versión, el programa escribirá el primer número y se detendrá con el mensaje 0 OK. Si a continuación escribe:

 NEXT n

el programa ejecutará una vez el bucle y escribirá el número siguiente.

Parte 5

Subrutinas

Temas tratados:

GO SUB, RETURN

A veces nos encontramos con que diferentes partes del programa tienen que realizar funciones bastante similares, y esto fácilmente puede llevarnos a escribir las mismas instrucciones varias veces con diferente número de línea. Afortunadamente, en BASIC disponemos de un recurso que nos permite evitar este despilfarro: podemos escribir las instrucciones una sola vez, en lo que denominamos una *subrutina*, y luego invocar la subrutina siempre que sea necesario.

El control de las subrutinas se realiza con dos sentencias: GO SUB y RETURN. Para invocar una subrutina se da una orden del tipo:

```
GO SUB xxx
```

donde xxx es el número de la primera línea de la subrutina. El mecanismo es similar al de GO TO xxx, con la diferencia de que el ordenador recuerda dónde estaba la sentencia GO SUB y así sabe a dónde tiene que volver cuando termina de ejecutar la subrutina.

(Por si le interesa saberlo, el +2 «recuerda» en qué punto del programa estaba la sentencia GO SUB porque almacena la *dirección de retorno* en la *pila* de GO SUB).

Cuando el ordenador encuentra la orden

```
RETURN
```

sabe que ahí termina la subrutina; entonces recuerda dónde estaba la sentencia GO SUB (tomando de la pila la dirección de retorno) y continúa a partir de la sentencia siguiente.

Como ejemplo, echemos un vistazo de nuevo al programa de adivinar números. Reescribalo de la siguiente forma:

```
10 REM Programa de adivinar números, reorganizado
20 INPUT "Introduzca un número secreto",a:CLS
30 INPUT "Adivine el número",b
40 IF b=a THEN PRINT "Correcto!":STOP
50 IF b<a THEN GO SUB 100
60 IF b>a THEN GO SUB 100
70 GO TO 30
100 PRINT "Vuelva a intentarlo"
110 RETURN
```

La sentencia GO TO 30 de la línea 70, y la sentencia STOP del programa siguiente, son muy importantes; si no fuera por ellas, los programas entrarían en las subrutinas y se produciría un error (7 RETURN without GO SUB, 'RETURN sin GO SUB') al llegar a la sentencia RETURN.

El siguiente programa utiliza una subrutina (líneas 100 a 150) que escribe la tabla de multiplicar correspondiente al *parámetro* n. La orden GO SUB 100 que invoca la subrutina puede estar en cualquier lugar del programa. Cuando el ordenador encuentra la orden RETURN en línea 150 de la subrutina, el control retorna al programa principal, que continúa funcionando a partir de la sentencia siguiente a aquélla en que se encontraba el GO SUB original. Al igual que ocurría con GO TO, se puede escribir GO SUB en una sola palabra: GOSUB.

```
10 REM  Tabla de multiplicar del 2, el 5, el 10 y el 11
20 LET n=2: GO SUB 100
30 LET n=5: GO SUB 100
40 LET n=10: GO SUB 100
50 LET n=11: GO SUB 100
60 STOP
70 REM  Fin del programa principal, comienzo de la subrutina
100 PRINT "Tabla de multiplicar por ";n
110 FOR v=1 TO 9
120 PRINT v;" x ";n;" = ";v*n
130 NEXT v
140 PRINT
150 RETURN
```

Una subrutina puede invocar a otra, o incluso a sí misma. (Una rutina que se llama a sí misma es una *rutina recursiva*).

Parte 6

Datos en los programas

Temas tratados:

READ, DATA, RESTORE

En algunos de los programas anteriores hemos visto que la información (es decir, los datos) puede ser captada por el programa mediante la sentencia INPUT. En ocasiones esto resulta muy tedioso, especialmente cuando buena parte de la información que hay que introducir es la misma en todas las ejecuciones del programa. Se puede ahorrar un tiempo considerable utilizando las sentencias READ, DATA y RESTORE. Veamos un ejemplo:

```
10 READ a,b,c
20 PRINT a,b,c
30 DATA 1,2,3
```

Una sentencia READ consiste en la palabra clave READ seguida de una lista de nombres de variables separados entre sí por comas. Funciona de modo bastante parecido a INPUT, con la ventaja de que, en vez de hacerle a usted escribir los valores que hay que asignar a las variables, el ordenador los lee en la sentencia DATA.

Cada sentencia DATA es una lista de expresiones (numéricas o literales) separadas por comas. Las sentencias DATA pueden estar en cualquier lugar del programa, ya que el +2 las ignora completamente, excepto cuando está ejecutando una sentencia READ. Podemos imaginarnos las expresiones de todas las sentencias DATA del programa como si estuvieran colocadas todas juntas, formando una larga lista: la lista de datos. La primera vez que el +2 lee (con READ) un valor, consulta la primera expresión de la lista de datos; a la siguiente vez, lee la segunda; y así sucesivamente, según va encontrando instrucciones READ, sigue su camino a lo largo de la lista de datos. (Si trata de leer más allá del fin de la lista, se produce un error.)

Observe que es una pérdida de tiempo poner instrucciones DATA en una orden directa, ya que READ no las encontrará. Las sentencias DATA tienen que estar en líneas de programa.

Vamos a ver cómo funciona todo esto en el programa que acaba de transcribir. La línea 10 le pide al ordenador que lea tres datos y los asigne a las variables a, b y c. La línea 20 le pide que escriba (PRINT) los valores de esas variables. La sentencia DATA de la línea 30 proporciona los valores de a, b y c para que los lea la línea 10.

La información contenida en una sentencia DATA puede ser leída por un bucle FOR...NEXT. Escriba lo siguiente:

```
10 FOR n=1 TO 6
20 DATA 2,4,6,8,10,12
30 READ d
40 PRINT d
50 NEXT n
```

Los dos programas anteriores han demostrado que las sentencias DATA pueden estar en cualquier lugar (antes o después de la instrucción READ).

Al ejecutar este último programa, la sentencia READ avanza por la lista de datos en cada pasada por el bucle FOR...NEXT.

Las sentencias READ pueden también asignar valores a variables literales. Por ejemplo,

```
10 READ d$
20 PRINT "La fecha es",d$
30 DATA "20 de diciembre de 1986".
```

No siempre hay que leer las sentencias DATA en orden, de la primera a la última; de hecho, podemos saltar de una sentencia DATA a otra mediante la orden RESTORE. La forma de esta orden es:

```
RESTORE xxx
```

donde xxx es el número de la línea en que se encuentra la sentencia DATA que debe ser leída. La orden RESTORE sin número de línea hace que el *puntero de datos* apunte hacia la primera sentencia DATA del programa.

Transcriba y pruebe el siguiente programa:

```
10 DATA 1,2,3,4,5
20 DATA 6,7,8,9
30 GO SUB 110
40 GO SUB 110
50 GO SUB 110
60 RESTORE 20
70 GO SUB 110
80 RESTORE
90 GO SUB 110
100 STOP
110 READ a,b,c
120 PRINT a'b'c
130 PRINT
140 RETURN
```

La orden GO SUB 110 invoca una subrutina que lee los siguientes tres elementos de DATA y después los escribe (PRINT). Observe cómo la orden RESTORE decide qué elementos son leídos en cada caso.

Borre la línea 60 y ejecute este programa de nuevo para ver qué ocurre.

Parte 7

Expresiones

Temas tratados:

Operaciones: +, -, *, /

Expresiones, notación científica, nombres de variables

Ya ha visto algunas de las formas en las que el +2 realiza cálculos con números. Además de las cuatro operaciones aritméticas, +, -, * y / (recuerde que en BASIC * es el símbolo de la multiplicación y / el de la división), sabe encontrar el valor de una variable, dado su nombre.

La sentencia

```
LET impuesto=suma*15/100
```

es un ejemplo de cómo se puede combinar los cálculos. Una combinación tal como $\text{suma} * 15 / 100$ es lo que se llama *expresión*. Por consiguiente, una expresión es una forma abreviada de decirle al +2 cómo debe realizar varios cálculos, uno a continuación de otro. En nuestro ejemplo, la expresión $\text{suma} * 15 / 100$ significa: 'busca el valor de la variable llamada *sumas*', multiplícalo por 15 y divídelo por 100'.

En la parte 30 de este capítulo daremos la lista completa de las prioridades de las operaciones matemáticas (y lógicas).

En las expresiones que contienen *, /, + o -, la multiplicación y la división son prioritarias con respecto a la suma y la resta (es decir, el ordenador las realiza antes que la suma y la resta). La multiplicación y la división tienen la misma prioridad una que la otra, lo que significa que el +2 las realiza en el orden en que aparezcan en la expresión (de izquierda a derecha). Las siguientes operaciones que realiza el +2 son la suma y la resta; como también tienen el mismo nivel de prioridad, el ordenador las calcula de izquierda a derecha.

Así pues, en la expresión $8 - 12 / 4 + 2 * 2$ la primera operación que se efectúa es la división $12 / 4$, cuyo resultado es 3, por lo que la expresión equivale a $8 - 3 + 2 * 2$.

La siguiente operación que se lleva a cabo es la multiplicación $2 * 2$, que da 4, así que la expresión se convierte entonces en $8 - 3 + 4$.

El siguiente paso es restar $8 - 3$, que da 5; la expresión pasa a ser $5 + 4$. Finalmente, se realiza la suma y el resultado es 9.

Para comprobarlo, dé la orden

```
PRINT 8-12/4+2*2
```

Se puede cambiar la prioridad de los cálculos dentro de una expresión mediante el uso adecuado de los paréntesis. Los cálculos que van entre paréntesis se realizan antes que todos los demás; por lo tanto, si en la expresión anterior queremos que se empiece por calcular la suma $4+2$, basta con escribirla entre paréntesis. Compruébelo con la orden:

```
PRINT 8-12/(4+2)*2
```

El resultado es ahora 4 en lugar de 9.

Las expresiones son útiles por el hecho de que, siempre que el $+2$ esté esperando un número, nosotros podemos darle una expresión en su lugar y él hallará la respuesta.

También se puede «sumar» cadenas (o variables literales) en una expresión. Por ejemplo,

```
10 LET a$="arroz"
20 LET b$="leche"
30 PRINT a$;" con ";b$
```

Ha llegado el momento de que le digamos qué es lo que puede y lo que no puede usar como nombre de variable. Como ya sabe, el nombre de una variable literal tiene que ser una sola letra seguida de \$, y el nombre de la variable de control de los bucles FOR...NEXT también tiene que consistir en una sola letra. Pero los nombres de las variables numéricas ordinarias son mucho más libres; pueden constar de dígitos o letras cualesquiera, siempre que el primer carácter sea una letra. También se puede introducir espacios en los nombres de las variables, para hacerlos más legibles, aunque el $+2$ los ignorará a todos los efectos, salvo en los listados. Además, es indiferente escribir los nombres en mayúsculas o en minúsculas. No obstante, hay algunas restricciones acerca de los nombres de variables: no pueden coincidir con las palabras clave y, en general, si un nombre de variable contiene una palabra clave con espacios a los lados, BASIC no lo aceptará.

He aquí unos cuantos ejemplos de nombres de variable válidos:

```
x
cualquier cosa
t42
este nombre no es conveniente porque es demasiado largo
tobeornottobe
espacios y mayusculas mezclados
EspaciosyMayusculasMezclados
```

(Observe que los dos últimos nombres son considerados iguales y se refieren a la misma variable.)

Los siguientes nombres de variable no son válidos:

pi	(PI es una palabra clave)
2001	(comienza con dígito)
Albany, New York	(contiene NEW entre espacios)
to be or not to be	(TO, OR y NOT son palabras clave de BASIC)
3osos	(comienza con un dígito)
M*A*S*H	(* no es letra ni dígito)
M-30	(- no es letra ni dígito)

Los valores numéricos pueden ser representados por un número y un exponente (en *notación científica*). Pruebe las siguientes órdenes:

```
PRINT 2.34e0  
PRINT 2.34e1  
PRINT 2.34e2
```

y así sucesivamente hasta

```
PRINT 2.34e15
```

PRINT sólo puede escribir los números con ocho cifras significativas. Pruebe la siguiente orden:

```
PRINT 4294967295, 4294967295-429e7
```

Esto demuestra que el ordenador guarda los dígitos de 429467295, aunque no es capaz de mostrarlos todos de una vez.

El +2 trabaja en *aritmética de punto flotante*, lo que significa que guarda por separado los dígitos del número (la *mantisa*) y la posición del punto decimal (el *exponente*). Este método no siempre da resultados exactos, ni siquiera con números enteros. Escriba:

```
PRINT 1e10+1-1e10,1e10-1e10+1
```

Los números se guardan con unos nueve dígitos y medio de precisión, de modo que 1e10 es demasiado grande como para ser almacenado con precisión absoluta. La inexactitud (en este caso cerca de 2) es mayor que 1, y por tanto los números 1e10 y 1e10+1 le parecen iguales al ordenador.

Un ejemplo aún más peculiar es:

```
PRINT 5e9+1-5e9
```

Aquí la inexactitud de 5e9 es sólo de alrededor de 1, y el 1 que debe ser sumado se redondea hasta 2. Los números 5e9+1 y 5e9+2 le parecen iguales al ordenador. El número completo más grande que se puede almacenar con completa exactitud es 4294967294.

La cadena "", que no tiene caracteres, se llama cadena vacía o cadena nula. Recuerde que los espacios son significativos y que una cadena vacía no es lo mismo que una que sólo contenga espacios.

Pruebe

```
PRINT "Leiste "El Pais" ayer?"
```

Al pulsar **INTRO** aparece el cursor parpadeante en rojo, que, como sabemos, indica la presencia de un error en algún lugar de la línea. Cuando el +2 encuentra las comillas al principio de "El Pais", da por supuesto que marcan el final de la cadena "Leiste", y entonces no sabe qué significa El Pais.

Hay un recurso especial para evitar esto: cuando necesite que las comillas formen parte de una cadena, escribálas repetidas. Por ejemplo,

```
PRINT "Leiste ""El Pais"" ayer?"
```

Como puede ver en la pantalla, las comillas sólo aparecen una vez (pero usted tiene que escribirlas dos veces para que sean reconocidas).

Parte 8

Cadenas literales

Temas tratados:

Dissección de cadenas, TO

Dada una cadena, una *subcadena* de ella consiste en varios de sus caracteres tomados secuencialmente. Así, "cadena" es una subcadena de "cadena mayor", pero "c mayor" y "caneda" no lo son.

Existe una notación específica para describir subcadenas que puede ser aplicada a expresiones literales cualesquiera. Su forma general es:

expresión literal (principio TO fin)

Así, por ejemplo

"abcdef"(2 TO 5)

es igual a bcde.

Si se omite el *principio*, BASIC toma por defecto el 1; si se omite el *fin*, se toma la longitud total de la cadena. De este modo:

"abcdef"(TO 5)	es igual a	abcde
"abcdef"(2 TO)	es igual a	bcdef
"abcdef"(TO)	es igual a	abcdef

Esta última expresión se puede escribir también en la forma "abcdef"().

Hay una forma ligeramente diferente en la que se omite el TO y sólo se escribe un número:

"abcdef"(3) equivale a "abcdef"(3 TO 3) que es igual a c

Aunque normalmente el *principio* y el *fin* deben hacer referencia a caracteres que realmente existan en la cadena, esta regla está supeditada a otra: si el *principio* es mayor que el *fin*, el resultado es la cadena vacía. Así,

"abcdef"(5 TO 7)

da el error 3 Subscript wrong ('Subíndice erróneo'), porque la cadena sólo tiene 6 caracteres y el valor 7 es, por consiguiente, demasiado grande; pero

```
"abcdef"(8 TO 7)
```

y

```
"abcdef"(1 TO 0)
```

son iguales a la cadena vacía y, por lo tanto, están permitidas.

El *principio* y el *fin* no deben ser negativos; si lo son se produce el error B integer out of range ('Entero fuera de margen'). El siguiente programa ilustra algunas de estas reglas:

```
10 LET a$="abcdef"
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
```

Escriba NEW cuando haya probado este programa y luego transcriba el siguiente:

```
10 LET a$="1234567890"
20 FOR n=1 TO 10
30 PRINT a$(n TO 10),a$((11-n) TO 10)
40 NEXT n
```

Si la cadena no es constante, sino variable, también podemos asignar valores a sus subcadenas. Por ejemplo, escriba:

```
LET a$="Me gusta mi Sinclair"
```

y después

```
LET a$(13 TO 20)="Amstrad*****"
```

Ahora dé la orden

```
PRINT a$
```

Observe que la subcadena a\$(13 TO 20) sólo tiene ocho caracteres y que por eso en la segunda sentencia de asignación sólo se utiliza los ocho primeros caracteres de Amstrad*****. Los cuatro restantes, ****, son ignorados. Ésta es una característica de la asignación de valores a subcadenas: la subcadena debe tener exactamente la misma longitud antes que después de la asignación. Para ello, si la cadena asignada es demasiado larga, BASIC la recorta por la derecha; si es demasiado corta, la completa con espa-

cios por la derecha. Esto es algo parecido a lo que hacía Procrustes, un posadero que siempre adaptaba los huéspedes al tamaño de la cama: o los estiraba en el potro o les cortaba los pies.

Si ahora hace

```
LET a$()="Hola!"
```

y da la orden

```
PRINT a$;"."
```

comprobará que BASIC ha considerado a\$ como subcadena de a\$ y que ha completado su nuevo valor rellenando con espacios por la derecha.

En cambio,

```
LET a$="Hola!"
```

asigna un valor totalmente nuevo a a\$, sin respetar la longitud anterior.

Las expresiones literales complicadas necesitarán estar entre paréntesis para poder extraer subcadenas de ellas. Por ejemplo,

```
"abc"+"def"(1 TO 2)    es igual a "abcde"  
("abc"+"def")(1 TO 2) es igual a "ab"
```

Ejercicio

1. Diseñe un programa que escriba el día de la semana recurriendo a la extracción de subcadenas. (*Sugerencia.* Forme la cadena con *LunMarMieJueVieSabDom.*)

Parte 9

Funciones

Temas tratados:

DEF
LEN, STR\$, VAL, SGN, ABS, INT, SQR
FN

Para explicar en qué consisten las funciones, tomemos como ejemplo la máquina de hacer zumo. Usted introduce una pieza de fruta por un extremo, pone en marcha la máquina y por el otro lado sale el zumo. De una naranja sale zumo de naranja; de una pera, zumo de pera; y de una manzana, zumo de manzana.

Las funciones son como las máquinas de hacer zumo, aunque hay una diferencia: trabajan con números y cadenas en vez de con frutas. Usted suministra un valor (el *argumento*), lo *procesa* realizando con él algunos cálculos y, finalmente, obtiene otro valor: el *resultado*.

Entrada de fruta	→ Máquina de hacer zumo	→ Zumo
Argumento	→ Función	→ Resultado

Diferentes argumentos producen diferentes resultados; si el argumento es completamente inadecuado, la función no puede procesarlo y BASIC emite un mensaje de error.

Al igual que en la industria puede haber diferentes máquinas para fabricar diferentes productos (una para zumos, otra para manteles, una tercera para guantes de goma, etc.), en el ordenador necesitamos distintas funciones para realizar los diferentes cálculos. Cada función tiene un nombre que la distingue de todas las demás.

Para utilizar una función se escribe su nombre seguido del argumento; BASIC calcula la función cuando evalúa la expresión en la que aquélla interviene.

Por ejemplo, hay una función llamada LEN que da como resultado la longitud de una cadena. El argumento de LEN es la cadena cuya longitud se desea hallar. Por ejemplo, si damos la orden

```
PRINT LEN "Spectrum +2"
```

el ordenador responde con el número 11, es decir, el número de caracteres (incluidos los espacios) de que consta la cadena Spectrum +2.

Si mezclamos funciones y operaciones en una sola expresión, aquéllas son evaluadas antes que éstas. Pero, por otra parte, podemos alterar este orden utilizando paréntesis.

Por ejemplo, las dos expresiones siguientes sólo se distinguen en los paréntesis y, precisamente por ello, los cálculos son realizados en un orden completamente distinto en cada caso (aunque los resultados finales son idénticos):

LEN "Pepe " + LEN "y Manolo"	LEN ("Pepe " + "y Manolo")
↓	↓
5+LEN "y Manolo"	LEN ("Pepe y Manolo")
↓	↓
5+8	LEN "Pepe y Manolo"
↓	↓
13	13

Veamos unas cuantas funciones más:

STR\$ convierte números en cadenas. Su argumento es un número; su resultado es la cadena que aparecería en la pantalla si el número hubiera sido escrito por **PRINT**. Observe que el nombre de la función termina en un signo \$ para indicar que el resultado es una cadena. Por ejemplo,

```
LET a$=STR$ 1e2
```

produce exactamente el mismo efecto que

```
LET a$="100"
```

Otro ejemplo:

```
PRINT LEN STR$ 100.0000
```

da como resultado 3, ya que **STR\$ 100.0000** es igual a la cadena 100, cuya longitud es 3 caracteres.

VAL es la función inversa de **STR\$**: convierte cadenas en números. Por ejemplo,

```
VAL "3.5"
```

produce el número 3.5.

Decimos que **VAL** es la función inversa de **STR\$** porque si tomamos un número cualquiera, le aplicamos **STR\$** y después **VAL**, el resultado final es el número original.

En cambio, si tomamos una cadena, le aplicamos **VAL** y luego **STR\$**, no siempre obtenemos la cadena original.

VAL es una función muy potente, ya que la cadena que le entregamos como argumento no necesariamente ha de tener la forma de un número constante, sino que puede tener la de una expresión numérica. Así, por ejemplo...

```
VAL "2*3"
```

da como resultado el número 6; incluso

```
VAL ("2"+"*3")
```

produce el número 6. En este último caso han intervenido dos procesos. En primer lugar se ha producido la *concatenación* de "2" y "*3" para dar la cadena "2*3"; después VAL ha suprimido las comillas y ha calculado el valor del producto 2*3, que es 6.

Esto puede resultar bastante confuso cuando las expresiones son complicadas. Por ejemplo,

```
PRINT VAL "VAL""VAL""2""""""
```

(Recuerde que dentro de una cadena las comillas deben ser escritas dos veces. Si bucea a más profundidad en las cadenas, resulta que las comillas han de ser cuadruplicadas, o incluso octuplicadas.)

Existe otra función, bastante similar a VAL, aunque probablemente menos útil, llamada VAL\$. Su argumento sigue siendo una cadena, pero su resultado también lo es. Para ver cómo funciona, recuerde la forma en que VAL realiza los cálculos: primero evalúa su argumento hasta obtener una cadena sencilla; después suprime las comillas y todo lo que queda lo evalúa como número. En el caso de VAL\$, el primer paso es el mismo, pero, una vez eliminadas las comillas en el segundo paso, todo lo que queda se evalúa como cadena. Por ejemplo,

```
VAL$ ""Ursula"" es igual a "Ursula"
```

(Observe cómo las comillas proliferan de nuevo.) Haga ahora

```
LET a$="99"
```

y haga que el ordenador escriba todo siguiente: VAL a\$, VAL "a\$", VAL ""a\$"", VAL\$ a\$, VAL\$ "a\$" y VAL\$ ""a\$"". Algunas de ellas funcionarán y otras no; intente explicar todas las respuestas. (Mantenga la cabeza fría.)

SGN es la función 'signo'. Es la primera función que nos encontramos que no tiene nada que ver con las cadenas, ya que tanto su argumento como su resultado son números. El resultado es +1 si el argumento es positivo, 0 si el argumento es 0, y -1 si el argumento es negativo.

ABS es otra función cuyos argumento y resultado son números. Convierte el argumento en un número positivo (que es el resultado) olvidando el signo. Así, por ejemplo,

ABS -3.2

es igual a

ABS 3.2

que, sencillamente, es igual a 3.2.

INT significa 'parte entera', un número entero, posiblemente negativo. Esta función convierte un número fraccionario (con decimales) en un entero desechando los decimales. Así, por ejemplo,

INT 3.9

da 3.

Tenga cuidado al aplicar esta función a números negativos, ya que siempre los redondea hacia abajo. Por ejemplo,

INT -3.1

es igual a -4.

SQR calcula la raíz cuadrada del argumento; es decir, da un resultado que, multiplicado por sí mismo, es el argumento. Por ejemplo,

SQR 4

es igual a 2 porque $2*2=4$.

SQR 0.25

es igual a 0.5 porque $0.5*0.5=0.25$.

SQR 2

es (aproximadamente) igual a 1.4142136 porque $1.4142136*1.4142136=2.0000001$.

Si usted multiplica cualquier número (incluso uno negativo) por sí mismo, el producto siempre es positivo. Esto significa que los números negativos no tienen raíz cuadrada; por consiguiente, si aplicamos SQR a un argumento negativo, obtenemos el mensaje de error A Invalid Argument ('Argumento no válido').

Aparte de la funciones que BASIC nos ofrece, podemos también definir otras. Sus nombres consistirán en las letras FN seguidas de una letra de nuestra elección (si el resul-

tado va a ser un número) o en FN seguidas de otra letra y de \$ (si el resultado va ser una cadena). Estas funciones son mucho más estrictas en lo que se refiere a los paréntesis (el argumento siempre tiene que figurar entre paréntesis).

Una función se define poniendo una sentencia DEF en algún lugar del programa. Por ejemplo, la siguiente sentencia define la función FN c, cuyo resultado es el cuadrado del argumento:

```
10 DEF FN c(x)=x*x: REM el cuadrado de x
```

La letra c que sigue a DEF FN es el nombre que hemos elegido para la función. La x entre paréntesis es el nombre por el cual nos referiremos al argumento de la función. Para esto se puede utilizar una letra cualquiera (pero sólo una), o bien, si el argumento es una cadena, una sola letra seguida de \$.

Tras el signo = viene la verdadera definición de la función. Ésta puede ser cualquier expresión, y puede hacer referencia al argumento utilizando el nombre que le hemos dado (en este caso, x), como si fuera cualquier otra variable.

Una vez ejecutada la sentencia DEF, la función se puede utilizar de la misma manera que las intrínsecas de BASIC: escribiendo su nombre, FN c, seguido por el argumento entre paréntesis. (Recuerde siempre que en las funciones definidas por el usuario el argumento tiene que estar entre paréntesis). Practique unas cuantas veces:

```
PRINT FN c(2)
PRINT FN c(3+4)
PRINT 1+INT FN c (LEN "pollo"/2+3)
```

Decíamos antes que INT siempre redondea hacia abajo. Para redondear al entero más cercano, sume 0.5 al número antes de aplicarle INT; si no quiere tener que hacer esto cada vez que necesite redondear un número, puede encomendarle la tarea a una función creada al efecto:

```
20 DEF FN r(x)=INT (x+0.5): REM da x redondeado al entero más cercano
```

Después puede comprobar que, por ejemplo,

```
FN r(2.9)   es igual a 3      FN r(2.4)   es igual a 2
FN r(-2.9)  es igual a -3     FN r(-2.4)  es igual a -2
```

Compare estas respuestas con las que se obtiene empleando INT en lugar de FN r. Transcriba y ejecute este programa:

```
10 LET x=0: LET y=0: LET a=10
20 DEF FN p(x,y)=a+x*y
30 DEF FN q()=a+x*y
40 PRINT FN p(2,3),FN q()
```

En este programa hay muchas sutilezas. Primero, las funciones pueden tener varios argumentos, e incluso no tener ninguno (lo que no se puede omitir es los paréntesis).

Segundo, las sentencias DEF pueden estar en cualquier lugar del programa. Cuando el ordenador ha ejecutado la línea 10, se salta las líneas 20 y 30 para llegar a la 40. Sin embargo, tienen que estar en *alguna* parte del programa, no en una orden directa.

Tercero, tanto x como y son nombres de variables en el conjunto del programa y además son los nombres de los argumentos de la función FN p. Ésta olvida temporalmente las variables llamadas x e y, pero, puesto que no tiene ningún *argumento* con el nombre de a, sigue recordando la *variable* a. De este modo, cuando FN p(2,3) está siendo evaluada, a tiene el valor 10 porque es la variable, x tiene el valor 2 porque es el primer argumento, e y tiene el valor 3 porque es el segundo argumento. Así que el resultado es $10+2*3$, que es igual a 16. Por otra parte, cuando FN q() está siendo evaluada, no hay argumentos, de modo que a, x e y siguen refiriéndose a las *variables* y, por lo tanto, tienen los valores 10, 0 y 0, respectivamente. La respuesta en este caso es $10+0*0$, que es igual a 10.

Cambie ahora la línea 20 por:

```
20 DEF FN p(x,y)=FN q()
```

Esta vez FN p(2,3) tendrá el valor 10, ya que FN q volverá a las variables x e y y no utilizará los argumentos de FN p.

Algunas versiones de BASIC (aunque no el Spectrum BASIC) tienen las funciones LEFT\$, RIGHT\$, MID\$ y TL\$.

LEFT\$(a\$,n) da la subcadena de a\$ consistente en los n primeros caracteres.

RIGHT\$(a\$,n) da la subcadena de a\$ consistente en los últimos caracteres, a partir del n-ésimo.

MID\$(a\$,n1,n2) da la subcadena de a\$ consistente en los n2 caracteres tomados a partir del n1-ésimo.

TL\$(a\$) da la subcadena de a\$ consistente en todos sus caracteres menos el primero.

Usted puede definir funciones de usuario que produzcan los mismos resultados:

```
10 DEF FN t$(a$)=a$(2 TO ): REM TL$
20 DEF FN l$(a$,n)=a$( TO n): REM LEFT$
```

Compruebe que estas funciones operan con cadenas de longitud 0 o 1. Observe que nuestra FN l\$ tiene dos argumentos; uno es un número y el otro una cadena. Una función puede tener hasta 26 argumentos numéricos (¿por qué 26?) y, al mismo tiempo, hasta 26 argumentos literales.

Ejercicio

1. Utilice la función FN $c(x)=x*x$ para probar SQR. Observará que

FN $c(\text{SQR } x)$

es igual a x si se pone cualquier número positivo en lugar de x , y que

SQR FN $c(x)$

es igual a $\text{ABS } x$ tanto si x es positivo como si es negativo. (¿Por qué aparece ABS aquí?)



Parte 10

Funciones matemáticas

Temas tratados:

↑
PI, EXP, LN, SIN, COS, TAN, ASN, ACS, ATN

En esta sección vamos a ocuparnos de las funciones matemáticas que puede manejar el +2. Es posible que usted nunca haya utilizado ninguna de estas funciones; si se le hace demasiado pesado continuar, no le importe saltarse esta sección. Examinaremos la operación ↑ (elear a una potencia), las funciones EXP y LN y las funciones trigonométricas SIN, COS, TAN, y sus inversas, ASN, ACS y ATN.

↑ y EXP

Elevar un número a una potencia es multiplicar el número por sí mismo cierto número de veces. Esta operación se indica normalmente escribiendo el segundo número (el exponente) a la derecha del primero y un poco más arriba; obviamente, esto sería difícil en un ordenador, por lo que utilizamos el símbolo ↑. Por ejemplo, las potencias de 2 son:

$2\uparrow 1 = 2$
 $2\uparrow 2 = 2 * 2 = 4$ (2 al cuadrado, escrito normalmente 2^2).
 $2\uparrow 3 = 2 * 2 * 2 = 8$ (2 al cubo, escrito normalmente 2^3).
 $2\uparrow 4 = 2 * 2 * 2 * 2 = 16$ (2 a la cuarta, escrito normalmente 2^4).
etcétera.

Así, al nivel más elemental, $a\uparrow b$ significa 'a multiplicado por sí mismo b veces'; pero, evidentemente, esto sólo tiene sentido si b es un número entero positivo. Para encontrar una definición que sirva para otros valores de b , consideremos la regla:

$$a\uparrow(b+c) = a\uparrow b * a\uparrow c$$

(Observe que concedemos a ↑ una prioridad mayor que a * y a /, de forma que, cuando hay varias operaciones en una expresión, las potencias se evalúan antes que los productos y las divisiones.) Cualquiera puede aceptar sin demasiados reparos que esta regla es válida cuando a y b son números enteros positivos; pero si queremos que funcione también cuando no lo sean, nos vemos forzados a aceptar que

$$a \uparrow 0 = 1$$

$$a \uparrow (-b) = 1/a \uparrow b$$

$a \uparrow (1/b)$ = la b -ésima raíz de a , es decir, el número que hay que multiplicar por sí mismo b veces para obtener a

y que

$$a \uparrow (b * c) = (a \uparrow b) \uparrow c$$

Si usted no ha visto nunca antes nada de esto, no intente aprendérselo de memoria a la primera; basta con que recuerde que:

$$a \uparrow (-1) = 1/a$$

y que

$$a \uparrow (1/2) = \text{SQR } a$$

Puede ser, cuando se familiarice con estas fórmulas, que todas las demás empiecen a cobrar sentido.

Experimente probando este programa:

```
10 INPUT a,b,c
20 PRINT a↑(b+c),a↑b*a↑c
30 GO TO 10
```

Por supuesto, si la regla que dimos antes es cierta, en cada pasada por la línea 20 los dos números que escriba el +2 serán iguales. (Observe que, debido al modo en que el ordenador calcula las potencias, el número que está a la izquierda de \uparrow , a en este caso, nunca debe ser negativo.)

Un ejemplo bastante típico de aplicación de esta operación es el interés compuesto. Imagine que invierte una parte de su dinero en una sociedad inmobiliaria que le produce un 15% de interés anual. Después de un año no tendrá solamente el 100% de lo invertido, sino también el 15% de interés que le ha pagado la inmobiliaria, que hace un total del 115% de lo que tenía en un principio. En otras palabras, tiene que multiplicar su dinero por 1.15, y esto es válido cualquiera que sea la cantidad inicial. Cuando transcurra otro año habrá ocurrido lo mismo, por lo que tendrá $1.15 * 1.15$; o, dicho de otra forma, $1.15 \uparrow 2$, es decir, 1.3225 veces la cantidad inicial de dinero. En general, al cabo de y años, tendrá $1.15 \uparrow y$ veces la cantidad inicial.

Si prueba esta orden

```
FOR y=0 TO 100: PRINT y,10*1.15↑y: NEXT y
```

verá que, incluso empezando con 1000 pesetas, la suma se incrementará bastante deprisa y , lo que es más, el ritmo al que aumenta va siendo cada vez mayor. (Pero no se haga

ilusiones: para calcular el incremento real del valor de su dinero tendría que restar el porcentaje de inflación de ese 15% que le da la inmobiliaria.)

Este tipo de comportamiento, en el que al cabo de cierto intervalo de tiempo una cantidad se multiplica por un número fijo, es lo que se llama *crecimiento exponencial*. El valor final se calcula elevando ese número fijo a la potencia dada por el número de unidades de tiempo.

Supongamos que definimos la siguiente función:

10 DEF FN a(x)=a↑x

La variable a habrá sido definida en una sentencia LET (su valor corresponde al tipo de interés, que cambia frecuentemente).

Hay un valor especial de a que hace la función FN a particularmente atractiva para el ojo experimentado de un matemático; este valor es el llamado *número e* . El +2 posee una función, llamada EXP, definida por

EXP x es igual a $e↑x$

Lamentablemente, el número e no es en sí mismo un número demasiado bonito; se trata de un decimal infinito no periódico. Puede ver sus primeros decimales escribiendo la orden:

PRINT EXP 1

ya que $\text{EXP } 1 = e↑1 = e$. Por supuesto, esto es sólo una aproximación. No es posible escribir e con exactitud.

LN

La función inversa de la exponencial es la *función logarítmica*. El *logaritmo* de base a de un número x es la potencia a la que hay que elevar a para obtener el número x ; abreviadamente, $\log_a x$. Así, por definición, $a↑\log_a x = x$; además, $\log_a(a↑x) = x$.

Quizá sepa usted cómo utilizar logaritmos de base 10; son los logaritmos decimales. El +2 dispone de una función, LN, que calcula logaritmos de base e , o sea, logaritmos *neperianos* o *naturales*. Para calcular el logaritmo de un número en cualquier otra base se divide el logaritmo neperiano del número por el logaritmo neperiano de la base; es decir, $\log_a x = \text{LN } x / \text{LN } a$.

PI

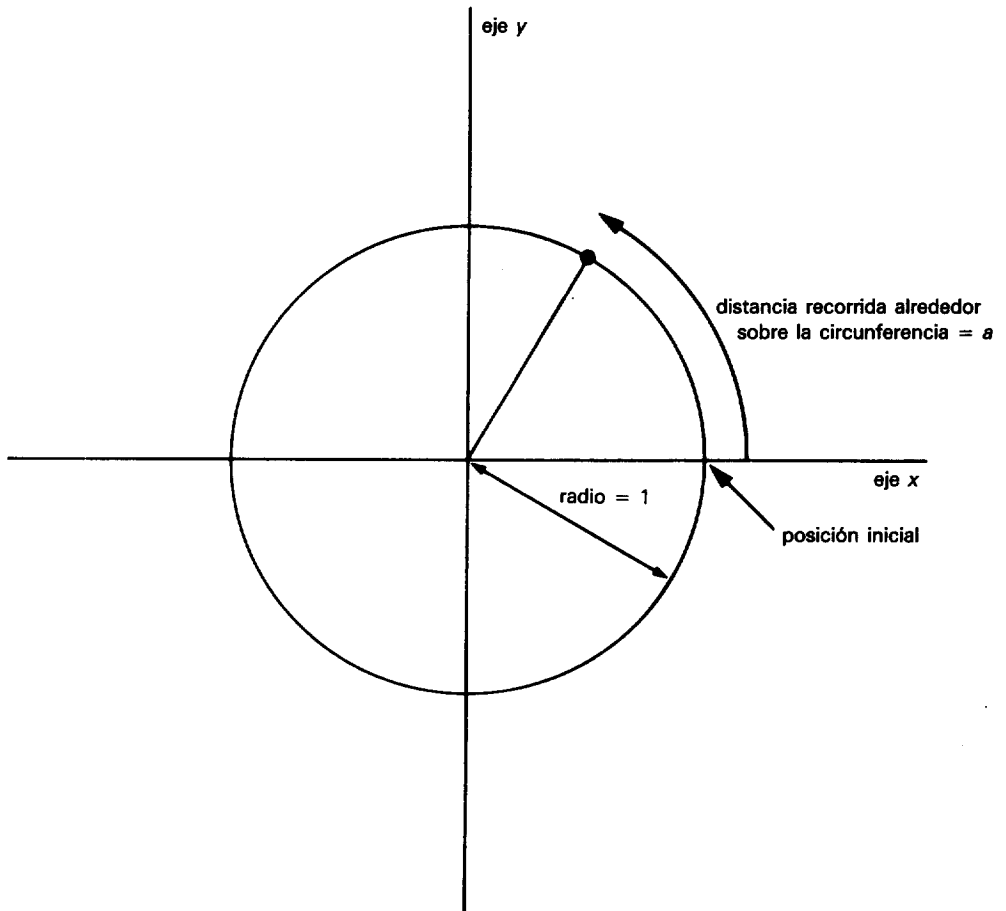
Como es bien sabido, la longitud de la circunferencia se obtiene multiplicado el diámetro por el número π . Este símbolo, π , es una 'p' griega que se lee 'pi'.

Al igual que e , π es un decimal infinito no periódico (empieza por 3.1415927). La palabra PI representa ese número en el +2. Pruebe lo siguiente:

PRINT PI

SIN, COS Y TAN: ASN, ACS Y ATN

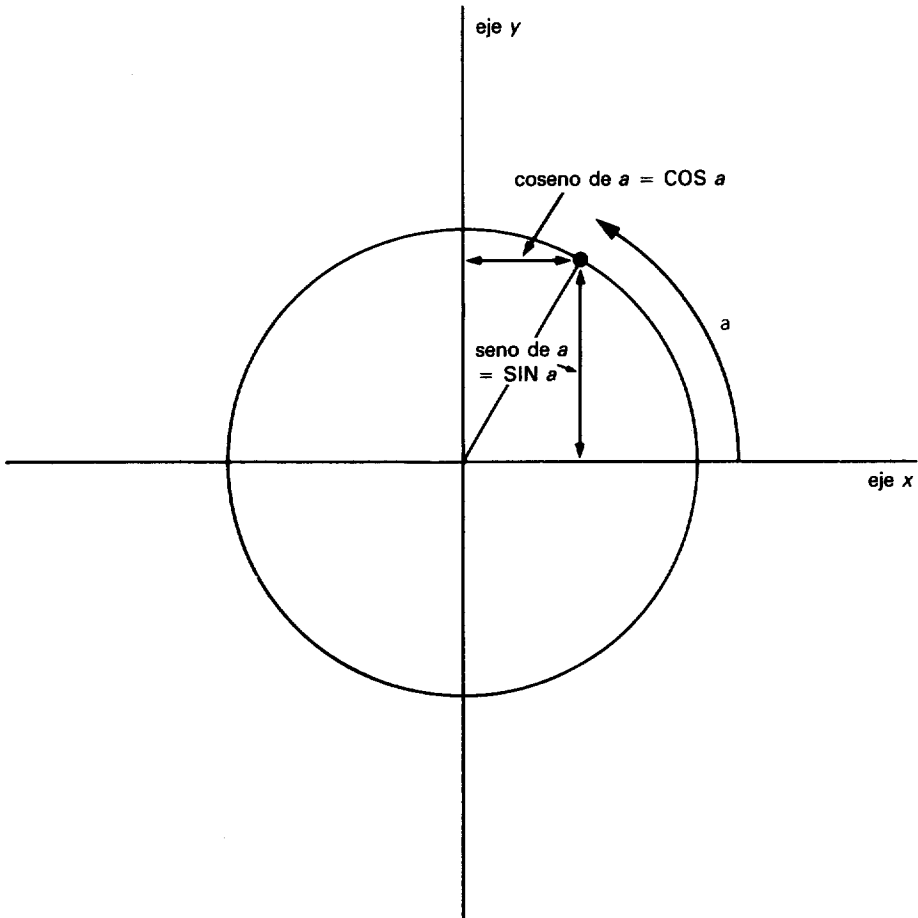
Las funciones *trigonométricas* dan una medida de lo que ocurre cuando un punto se mueve alrededor de una circunferencia. Tomemos una circunferencia de radio 1 y un punto que la recorre. El punto empieza en la posición de 'las tres' y se desplaza en el sentido contrario al de las agujas del reloj.



Hemos dibujados dos rectas, llamadas *ejes*, que pasan por el centro de la circunferencia. La que está en la posición de 'las tres' se llama eje *x*; la que pasa por 'las doce' es el eje *y*.

Para caracterizar la posición del punto basta con decir qué distancia ha recorrido sobre la circunferencia a partir de su posición inicial; vamos a llamar *a* a esta distancia. Sabemos que la longitud de la circunferencia es 2π (porque su radio es 1 y su diámetro, por consiguiente, es 2); así, cuando el punto ha recorrido un cuarto de la longitud de la circunferencia, $a = \pi/2$; cuando ha recorrido la mitad, $a = \pi$; y cuando ha recorrido todo el trecho, $a = 2\pi$.

Dada la distancia recorrida sobre la circunferencia, *a*, puede interesar conocer la distancia a la que está el punto *a la derecha* del eje *y* y la distancia a la que está *por encima* del eje *x*. Estas distancias se llaman, respectivamente, *coseno* y *seno* de *a*, y son las calculadas por las funciones COS y SIN del +2.



Observe que si el punto está a la izquierda del eje y , el coseno es negativo; si el punto está por debajo del eje x , el seno es negativo.

Otra propiedad interesante es que, una vez que a ha crecido hasta 2π , el punto ha llegado a la posición inicial y el seno y el coseno empiezan a tomar otra vez los mismo valores. Es decir, $\text{SIN}(a+2*\text{PI})$ es igual a $\text{SIN } a$ y $\text{COS}(a+2*\text{PI})$ es igual a $\text{COS } a$.

La *tangente* de a se define como el seno dividido por el coseno; la función correspondiente en el +2 se llama TAN.

Algunas veces necesitamos utilizar estas funciones al revés, o sea, averiguar qué valor de a ha producido cierto seno, coseno o tangente. Las funciones encargadas de esto se llaman *arco seno* (ASN en el +2), *arco coseno* (ACS) y *arco tangente* (ATN).

Observe el diagrama anterior y fíjese en el radio que une el punto con el centro. Está claro que la distancia que hemos llamado a (la distancia recorrida por el punto desde la posición inicial) es un modo de medir del ángulo formado por ese radio y el eje x . Cuando $a=\pi/2$, el ángulo tiene 90 grados; cuando $a=\pi$, el ángulo es de 180 grados, y así sucesivamente, hasta que $a=2\pi$ y el ángulo es de 360 grados. Podríamos olvidar completamente los grados y caracterizar el ángulo solamente con a ; decimos entonces que estamos midiendo el ángulo en radianes. Así, $\pi/2$ radianes = 90 grados, etc.

Recuerde siempre que en el +2 las funciones SIN, COS, etc. trabajan siempre con radianes, no con grados. Para convertir grados en radianes se divide por 180 y se multiplica por π . Para convertir radianes en grados se divide por π y se multiplica por 180.

Parte 11

Números aleatorios

Temas tratados:

RANDOMIZE
RND

En esta sección vamos a estudiar las palabras clave RND y RANDOMIZE.

En algunos aspectos, RND es semejante a una función: realiza cálculos y genera un resultado. En cambio, es peculiar en el sentido de que no necesita argumento.

Cada vez que utilizamos RND, su resultado es un nuevo número aleatorio comprendido entre 0 y 1. (Algunas veces puede tomar el valor 0, pero nunca el 1.)

Pruebe lo siguiente:

```
10 PRINT RND
20 GO TO 10
```

¿Puede detectar alguna regla o algún patrón repetitivo en los números? No debería, ya que 'aleatorio' significa 'impredecible', 'sin normas'.

En realidad, RND no es perfectamente aleatoria, ya que los valores que genera están tomados de una secuencia fija que consta de 65536 números. Sin embargo, estos están tan «revueltos» que podemos considerarlos impredecibles a todos los efectos prácticos. Por eso decimos que RND es pseudoaleatoria.

RND da un número (pseudo)aleatorio comprendido entre 0 y 1, pero fácilmente se puede transformar ese intervalo en otro cualquiera. Por ejemplo, $5 * \text{RND}$ da valores comprendidos entre 0 y 5; $1.3 + 0.7 * \text{RND}$ los da en el margen de 1.3 a 2. Cuando necesitemos números enteros podremos utilizar INT (sin olvidar que INT siempre redondea hacia abajo), como en $1 + \text{INT}(\text{RND} * 6)$, expresión que utilizaremos en un programa que simulará el lanzamiento de dados. $\text{RND} * 6$ genera valores que están en el margen de 0 a 6, pero, puesto que realmente nunca llega a 6, $\text{INT}(\text{RND} * 6)$ sólo puede ser 0, 1, 2, 3, 4 o 5.

He aquí el programa:

```
10 REM Programa de lanzamiento de dados
20 CLS
30 FOR n=1 TO 2
40 PRINT 1+INT(RND*6);" ";
50 NEXT n
60 INPUT a$: GO TO 20
```

Pulse **INTRO** cada vez que quiera 'lanzar' los dados.

La sentencia **RANDOMIZE** sirve para hacer que **RND** se ponga en marcha a partir de una posición determinada en su secuencia de números, como demuestra este programa:

```
10 RANDOMIZE 1
20 FOR n=1 TO 5: PRINT RND,: NEXT n
30 PRINT: GO TO 10
```

Tras cada ejecución de **RANDOMIZE 1**, la secuencia de números aleatorios vuelve a arrancar a partir de 0.0022735596. Como parámetro de **RANDOMIZE** se puede poner cualquier número comprendido entre 1 y 65535 para poner en marcha la secuencia **RND** en diferentes posiciones.

Una aplicación de **RANDOMIZE** podría ser la siguiente: supongamos que usted tiene un programa que genera números aleatorios y que no funciona correctamente; puede usar **RANDOMIZE** para hacer que el programa genere siempre los mismos números y así poder estudiar más sistemáticamente su comportamiento.

RANDOMIZE a secas (o **RANDOMIZE 0**) tiene un efecto diferente: elige el punto de partida de forma bastante impredecible. Pruebe el siguiente programa:

```
10 RANDOMIZE
20 PRINT RND: GO TO 10
```

La secuencia que obtiene usted aquí no es muy aleatoria, pero esto tiene fácil explicación. **RANDOMIZE** toma como parámetro el *tiempo* transcurrido desde que se encendió el +2. Como en este programa no transcurre mucho tiempo desde una ejecución de **RANDOMIZE** a la siguiente, **RND** hace más o menos lo mismo todas las veces. Se obtendría una «aleatoriedad» mejor reemplazando **GO TO 10** por **GO TO 20**.

El siguiente programa simula el lanzamiento de monedas y cuenta el número de caras y cruces:

```
10 LET caras=0: LET cruces=0
20 LET moneda=INT (RND*2)
30 IF moneda=0 THEN LET caras=caras+1
40 IF moneda=1 THEN LET cruces=cruces+1
50 PRINT caras;" , ";cruces,
60 IF cruces<>0 THEN PRINT caras/cruces;
70 PRINT: GO TO 20
```

La proporción de caras y cruces debe llegar a ser aproximadamente 1 si se deja que el programa continúe el tiempo suficiente, ya que, a la larga, es de esperar más o menos igual número de caras que de cruces.

Ejercicio

1. Supongamos que usted elige un número comprendido entre 1 y 872 y escribe:

RANDOMIZE *su número*

Compruebe que el siguiente valor generado por RND es

$$(75 * (\textit{su número} + 1) - 1) / 65536$$

Pruébalo varias veces.

Parte 12

Matrices

Temas tratados:

Matrices (observe que el +2 maneja las matrices de forma algo diferente de la estándar)

DIM

Supongamos que tenemos una lista de números (por ejemplo, las notas de diez alumnos de una clase). La forma más obvia de almacenarlos en el +2 sería utilizar las variables $m1, m2, m3, \dots, m10$. Sin embargo, el programa necesario para asignar los valores a esas diez variables sería bastante largo y tedioso de escribir:

```
10 LET m1=75
20 LET m2=44
30 LET m3=90
40 LET m4=38
50 LET m5=55
60 LET m6=64
70 LET m7=70
80 LET m8=12
90 LET m9=75
100 LET m10=60
```

Afortunadamente, existe un mecanismo, denominado *matriz*, que permite guardar todos esos valores con un solo nombre de variable. Una matriz es una variable especial que puede contener varios valores, llamados *elementos*; en esto se distingue de las variables ordinarias, que sólo pueden contener uno. Cada elemento de la matriz se identifica por un número, que es el *índice* (o *subíndice*). El índice se escribe entre paréntesis a continuación del nombre de la matriz. En el ejemplo anterior, el nombre global de los elementos de la matriz podría ser m (el nombre de las variables matriciales siempre tiene que consistir en una sola letra); por consiguiente, los diez elementos serían $m(1), m(2), m(3), \dots, m(10)$.

Los elementos de una matriz reciben también el nombre de *variables indexadas*; las variables que habíamos conocido hasta ahora se llaman variables *sencillas* u *ordinarias*.

Antes de poder utilizar una matriz, es necesario haberle reservado espacio en la memoria del +2; esta operación es lo que se llama *dimensionar* la matriz y se realiza mediante la palabra clave DIM. Por ejemplo, la sentencia

```
DIM m(10)
```

reserva espacio en la memoria para una matriz llamada *m* cuyas dimensiones van a ser diez (es decir, equivaldrá a diez variables indexadas). La sentencia DIM «inicializa» los elementos de la matriz dándoles el valor 0. Además, si ya existía una matriz con ese mismo nombre, la borra. (Pero no afecta a la variable sencilla *m*, si es que existe; una variable matricial puede coexistir con una variable sencilla numérica del mismo nombre, ya que la matriz es siempre distinguible por su subíndice.)

Los subíndices de los elementos de la matriz pueden ser representados por cualquier expresión numérica que produzca un número válido como subíndice. Gracias a esto, las matrices pueden ser procesadas utilizando bucles FOR...NEXT. De ese modo, podemos olvidar el interminable programa que dimos al principio de esta sección y guardar los diez datos mediante el siguiente:

```
10 DIM m(10)
20 FOR n=1 TO 10
30 READ m(n)
40 NEXT n
50 DATA 75,44,90,38,55,64,70,12,75,60
```

(Observe que la sentencia DIM tiene que haber sido ejecutada *antes* de intentar acceder a la matriz.)

Si lo desea, puede usted examinar el contenido de la matriz escribiendo:

```
PRINT m(1)
PRINT m(2)
PRINT m(3)
etcétera
```

También podemos formar matrices con varias dimensiones. En una matriz bidimensional necesitaremos dos números para especificar cada uno de sus elementos, de la misma manera que necesitamos un número de fila y un número de columna para especificar la posición de un carácter en la pantalla. Si imaginamos que los números de línea y de columna (dos dimensiones) describen una página impresa, en la tercera dimensión tendríamos los números de página. Por supuesto, nosotros estamos hablando de matrices numéricas, por lo que los elementos de esa matriz tridimensional no serían caracteres de texto, sino números. Trate de imaginarse los elementos de una matriz tridimensional *v* especificados por *v*(*número de página*, *número de línea*, *número de columna*).

Por ejemplo, para formar una matriz bidimensional con dimensiones 3 y 6, se utiliza la siguiente sentencia DIM:

```
DIM c(3,6)
```

Esa matriz tendrá un total de $3 \cdot 6 = 18$ variables indexadas:

$c(1,1), c(2,2), \dots, c(1,6)$
 $c(2,1), c(2,2), \dots, c(2,6)$
 $c(3,1), c(3,2), \dots, c(3,6)$

La misma idea es válida para cualquier número de dimensiones.

Aunque es posible tener una variable sencilla y una matricial con el mismo nombre, no puede haber dos matrices que se llamen igual, aunque tengan diferente número de dimensiones.

Hasta ahora hemos descritos la *matrices numéricas*. También existen *matrices literales*, cuyos elementos son, naturalmente, cadenas literales. Las cadenas de una matriz se distinguen de las cadenas sencillas en que son de longitud fija y en que la asignación de valores se realiza por el procedimiento de Procrustes (¿lo recuerda?): si el valor es demasiado largo, se lo recorta; si es demasiado corto, se lo complementa con espacios por la derecha.

El nombre de toda matriz literal consiste en una sola letra seguida de \$. A diferencia lo que ocurría con las matrices numéricas, una matriz literal y una variable literal sencilla no pueden tener el mismo nombre.

Supongamos, pues, que queremos una matriz a\$ que pueda albergar cinco cadenas. Debemos decidir qué longitud van a tener las cadenas. Por ejemplo, una matriz para cinco cadenas de diez caracteres cada una se dimensiona con:

DIM a\$(5,10)

Esta sentencia prepara una matriz de $5 \cdot 10$ caracteres, que podemos manejar individualmente o por filas completas:

$a\$(1) = a\$(1,1)a\$(1,2) \dots a\$(1,10)$
 $a\$(2) = a\$(2,1)a\$(2,2) \dots a\$(2,10)$
 $a\$(3) = a\$(3,1)a\$(3,2) \dots a\$(3,10)$
 $a\$(4) = a\$(4,1)a\$(4,2) \dots a\$(4,10)$
 $a\$(5) = a\$(5,1)a\$(5,2) \dots a\$(5,10)$

Si al usar la matriz especificamos el mismo número de subíndices (dos en este caso) que dimensiones hay en la sentencia DIM, obtenemos un solo carácter. Pero si omitimos el último, la matriz da la fila entera (una cadena de longitud fija). Así que, por ejemplo, $a\$(2,7)$ es el séptimo carácter de la cadena $a\$(2)$. Utilizando la notación de disección de cadenas (Parte 8 de este capítulo) también podríamos escribir $a\$(2)(7)$ en lugar de $a\$(2,7)$.

Ahora escriba:

```
LET a$(2)="1234567890"
```

y

```
PRINT a$(2), a$(2,7)
```

El resultado será:

```
1234567890    7
```

El último subíndice (el que se puede omitir), también puede tener la estructura de los parámetros de la disección de cadenas. Por ejemplo,

```
a$(2,4 TO 8)    es igual a    a$(2)(4 TO 8)    que a su vez es igual a    "45678"
```

Recuerde: en una matriz literal, todas las cadenas tienen la misma longitud. La sentencia DIM tiene un parámetro adicional (el último) que especifica esa longitud. Al escribir una variable indexada perteneciente a una matriz literal, podemos incluir un número adicional (o bien un *disector de cadenas*), que corresponderá al último parámetro de la sentencia DIM.

Si al dimensionar una matriz literal sólo especificamos un parámetro, la matriz se comportará como variable sencilla de longitud física. Escriba:

```
DIM a$(10)
```

y podrá comprobar que a\$ funciona igual que una variable literal ordinaria, con la única diferencia de que su longitud siempre será 10 y los valores le serán asignados por el método de Procrustes.

Ejercicio

1. Utilice las sentencias READ y DATA para formar una matriz m\$ en la cual m\$(n) sea el nombre del n-ésimo mes. (*Sugerencia.* Suponiendo que usted escriba 'setiembre' y no 'septiembre', la sentencia DIM necesaria es DIM m\$(12,9).) Compruebe todos los valores de m\$(n) haciendo que los escriba un bucle FOR...NEXT.

Parte 13

Condiciones

Temas tratados:

AND, OR
NOT

Vimos en la Parte 3 de este capítulo que la sentencia IF toma la forma

IF *condición* THEN ...

Las condiciones eran entonces relaciones (construidas a base de =, <, >, <=, >= y <>) que comparaban dos números o dos cadenas. También podemos combinar varias relaciones utilizando los operadores lógicos: AND ('y'), OR ('o') y NOT ('no').

Una relación combinada mediante AND con otra relación da el valor 'verdadero' siempre que ambas sean verdaderas; por ejemplo, podríamos tener una línea del siguiente estilo:

```
IF a$="si" AND x>0 THEN PRINT x
```

en la que x sólo se escribe si a\$ es igual a "si" y x mayor que cero.

Una relación combinada mediante OR con otra relación da el valor 'verdadero' siempre que al menos una de ellas sea verdadera. (El resultado también es verdadero si las dos relaciones son verdaderas.)

La relación NOT invierte el valor lógico: NOT *relación* es 'verdadero' siempre que la *relación* es falsa, y 'falso' cuando la *relación* es verdadera.

Las *expresiones lógicas* consisten en combinaciones de AND, OR y NOT, de la misma forma que las expresiones numéricas son combinaciones de +, -, *, /, etc. También se puede colocarlas entre paréntesis si es necesario. Las operaciones lógicas tienen orden de prioridad, lo mismo que +, -, *, / y ↑. OR es la menos prioritaria; en orden de prioridad ascendente le siguen AND y NOT.

NOT es en realidad una función, con un argumento y un resultado, pero su prioridad es muy inferior a la de las otras funciones. Por lo tanto, su argumento no necesita paréntesis, a menos que sea en sí una operación lógica (construida con AND, OR o ambas). NOT a=b significa lo mismo que NOT (a=b), y, por supuesto, lo mismo que a<>b.

<> es la negación de =, en el sentido de que <> es verdadero si y sólo si = es falso. En otras palabras

<>b es lo mismo que NOT a=b

y además

$\text{NOT } a <> b$ es lo mismo que $a = b$

Si lo piensa, se dará cuenta de que $>=$ y $<=$ son la negación de $<$ y $>$, respectivamente. Por consiguiente, para invertir el valor de una relación se puede escribir NOT delante de ella o sustituirla por su negación.

Por otra parte,

$\text{NOT (primera expresión lógica AND segunda expresión lógica)}$

es lo mismo que

$\text{NOT (primera expresión lógica) OR NOT (segunda expresión lógica)}$

Además

$\text{NOT (primera expresión lógica OR segunda expresión lógica)}$

es lo mismo que

$\text{NOT (primera expresión lógica) AND NOT (segunda expresión lógica)}$

Utilizando estas reglas, la negación de una expresión se puede reducir a una expresión en la que NOT esté aplicada directamente a relaciones. En último extremo, NOT no es imprescindible, aunque en la práctica facilita la programación al hacer más inteligibles las expresiones lógicas.

Lo que resta de esta sección es bastante complicado; si no le interesa mucho este tema, puede omitir su lectura y pasar directamente a la Parte 14.

Pruebe la siguiente orden:

```
PRINT 1=2, 1<>2
```

que aparentemente debería producir un error de sintaxis. En realidad, el ordenador no sabe qué es eso de 'valor lógico'; en lugar de los valores 'verdadero' y 'falso' utiliza números ordinarios, que están sujetos a unas cuantas reglas:

- (i) Cuando el ordenador evalúa las relaciones construidas a base de $=$, $<$, $>$, $<=$, $>=$ y $<>$, obtiene un valor numérico que es 1 cuando la relación es verdadera y 0 cuando la relación es falsa. Por eso la anterior orden PRINT escribió 0 para $1=2$, que es una relación falsa, y 1 para $1<>2$, que es verdadera.

(ii) En la sentencia

IF *condición* THEN ...

la *condición* puede ser en realidad una expresión numérica cualquiera. Si su valor es 0, el ordenador actúa como si se tratase de una expresión lógica con valor 'falso'; si el valor de *condición* es distinto de 0, el ordenador la considera 'verdadera'. Por tanto, la anterior sentencia IF es exactamente equivalente a

IF *condición*<>0 THEN ...

(iii) AND, OR y NOT son también operaciones aplicables a expresiones numéricas cualesquiera:

x AND y vale $\begin{cases} x & \text{si } y \text{ es verdadero (distinto de cero)} \\ 0 & \text{(falso) si } y \text{ es falso (cero)} \end{cases}$

x OR y vale $\begin{cases} 1 & \text{(verdadero) si } y \text{ es verdadero (distinto de cero)} \\ x & \text{si } y \text{ es falso (cero)} \end{cases}$

NOT x vale $\begin{cases} 0 & \text{(falso) si } x \text{ es verdadero (distinto de cero)} \\ 1 & \text{(verdadero) si } x \text{ es falso (cero)} \end{cases}$

(Observe que 'verdadero' significa 'distinto de cero' cuando estamos comprobando un valor dado, pero que significa '1' cuando estamos produciendo un valor nuevo.)

Ahora pruebe este programa:

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a>=b)+(b AND a< b)
40 GO TO 10
```

En cada vuelta, este programa escribe el mayor de los dos números introducidos.

Si se para a pensarlo, se dará cuenta de que

x AND y

se podría leer así:

x si y (de lo contrario, el resultado es 0)

y de que

x OR y

significa lo mismo que

x a no ser que y (en cuyo caso el resultado es 1)

Una expresión en la que se utilice AND u OR de esta forma es lo que se llama *expresión condicional*. Un ejemplo con OR podría ser:

```
LET total=precio sin impuesto*(1.12 OR v$="Exento de IVA")
```

Observe que AND tiende a asociarse con la suma (porque su valor por defecto es 0) y OR con la multiplicación (porque su valor por defecto es 1).

También se puede formar expresiones condicionales cuyo valor sea una cadena literal, pero sólo utilizando AND:

```
x$ AND y vale      x$ si y es distinto de cero  
                  "" (la cadena vacía) si y es cero
```

de modo que x\$ AND y significa 'x\$ si y (de lo contrario, su valor es la cadena vacía)'.

Pruebe este programa, que capta dos cadenas y las coloca en orden alfabético:

```
10 INPUT "Escriba dos cadenas" 'a$,b$  
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$  
30 PRINT a$;" ";(" "<" AND a$<b$)+("=" AND a$=b$);" ";b$  
40 GO TO 10
```

Parte 14

El juego de caracteres

Temas tratados:

CODE, CHR\$
POKE, PEEK
USR
BIN

Las letras, dígitos, espacios, signos de puntuación, etc. que pueden formar parte de las cadenas literales se llaman *caracteres* y componen el *juego de caracteres* del +2. En su mayor parte, estos caracteres son símbolos simples, pero algunos representan palabras completas, tales como PRINT, STOP, <>, etc.

Hay un total de 256 caracteres, cada uno de ellos identificado por un código comprendido entre 0 y 255 (en la Parte 27 de este capítulo damos la lista completa). Para efectuar la conversión entre códigos y caracteres disponemos de dos funciones: CODE y CHR\$.

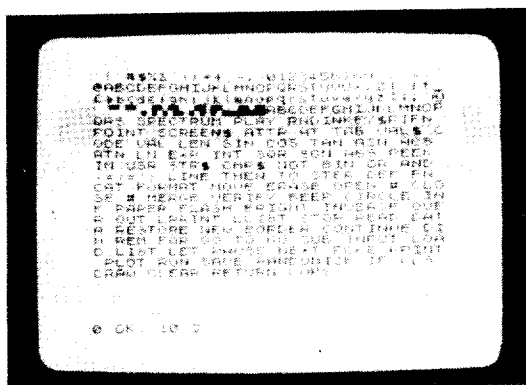
CODE se aplica a una cadena y da el código de su primer carácter (o, si se trata de la cadena vacía, da el número 0).

CHR\$ se aplica a un número y da la cadena de un único carácter cuyo código es ese número.

El siguiente programa escribe el juego de caracteres del +2 (del 32.º en adelante):

```
10 FOR a=32 TO 255: PRINT CHR$ a;: NEXT a
```

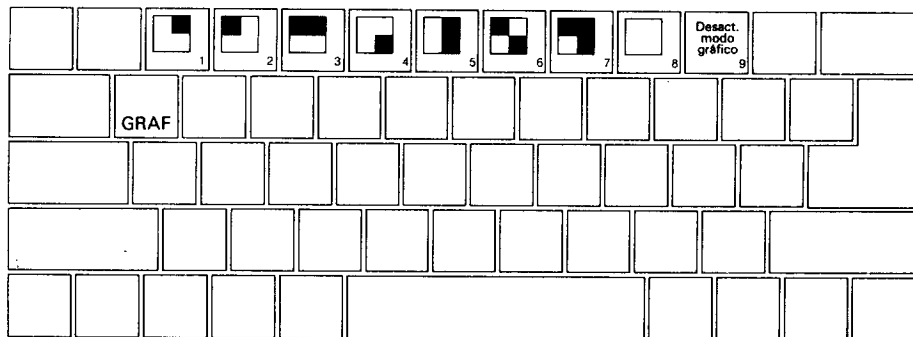
En la pantalla aparecerá lo siguiente...



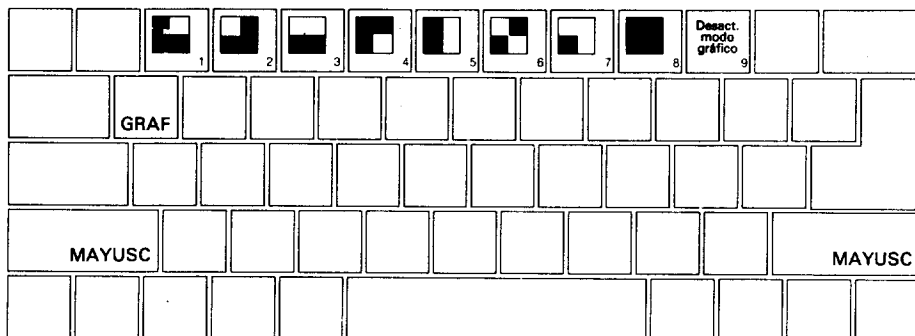
El juego de caracteres

Como puede apreciar, el juego de caracteres consta del espacio, 15 símbolos y signos de puntuación, los diez dígitos, otros siete símbolos, las mayúsculas, otros seis símbolos (entre los que se encuentra la 'Ñ'), las minúsculas y cinco símbolos más (entre los que se encuentra la 'ñ'). Todos ellos (excepto 'Pt', '©', '¡', '¿', 'ñ' y 'Ñ') están tomados de un juego de caracteres estándar, muy conocido y ampliamente utilizado: el juego ASCII (*American Standard Code for Information Interchange*, 'Código americano estándar para el intercambio de la información'). Los caracteres del juego ASCII están identificados por códigos, y esos códigos son los que emplea el +2.

Los demás caracteres no forman parte del juego ASCII, sino que son específicos de los ordenadores ZX Spectrum. Los primeros de ellos son un espacio y 15 combinaciones de cuadrados blancos y negros; éstos son los llamados *símbolos gráficos*, que pueden ser usados para dibujar. El usuario puede introducirlos por el teclado seleccionando el denominado *modo gráfico (modo G)*. Al pulsar la tecla **GRAF** se activa el modo gráfico; los símbolos gráficos se obtienen entonces pulsando las teclas numéricas (del 1 al 8).



















Estando en modo gráfico, al pulsar las teclas numéricas mencionadas en combinación con **MAYUSC** se obtiene los mismos símbolos gráficos, pero 'invertidos' (es decir, convirtiendo lo negro en blanco, y vice versa).



En modo gráfico las teclas del cursor no funcionan como de costumbre, ya que el +2 las interpreta como teclas numéricas combinadas con **MAYUSC** y, por lo tanto, producen símbolos gráficos.

Pulsando la tecla del 9 (o pulsando por segunda vez **GRAF**) se sale del modo gráfico y se vuelve al normal. Pulsando la tecla del 0 se borra el carácter que está a la izquierda del cursor.

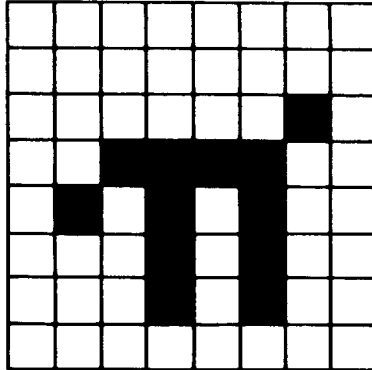
He aquí los 16 símbolos gráficos:

<i>Símbolo</i>	<i>Código</i>	<i>Símbolo</i>	<i>Código</i>
	128		143
	129		142
	130		141
	131		140
	132		139
	133		138
	134		137
	135		136

Los siguientes caracteres del juego del +2 son, aparentemente, una repetición de las mayúsculas de la 'A' a la 'S'. En realidad, se trata de caracteres cuya forma podemos rediseñar (sin embargo, mientras no los rediseñemos, tendrán por defecto la forma de esas letras mayúsculas); se les llama *gráficos definibles por el usuario*. Para introducirlos por el teclado primero se activa el modo gráfico y luego se pulsa una tecla alfabética (de la A a la S).

Para definir un nuevo carácter usted mismo, aplique la siguiente receta (que define un carácter con la forma de la letra griega 'π'):

- (i) Decida qué forma quiere que tenga el carácter. Cada carácter ocupa una retícula de 8×8 puntos, cada uno de los cuales puede estar encendido o apagado (los cuadrados negros representan los puntos que están encendidos).



Cuando un punto está encendido, el +2 lo dibuja con el color de la tinta; cuando el punto está apagado, lo dibuja con el color del papel. (Los términos *tinta* y *papel* están explicados en la Parte 16 de este capítulo.)

Hemos dejado un borde libre alrededor del diseño del carácter porque todas las letras también lo tienen, excepto las minúsculas con descendente, en las que éste llega hasta el extremo inferior.

- (ii) Decida a qué gráfico definible por el usuario quiere asignar la forma de la 'π'. Supongamos que elige el correspondiente a la 'P', de modo que cuando usted pulse P (en modo gráfico) obtenga π.
- (iii) Almacene el nuevo diseño. Cada gráfico definido por el usuario se almacena en forma de una sucesión de ocho números, uno para cada fila. Usted puede especificar estos números en un programa escribiendo la palabra BIN seguida de 8 ceros o unos (0 para el papel, 1 para la tinta). De esta forma, los ocho números que describen nuestro carácter π son:

BIN 00000000 primera fila (superior)
BIN 00000000 segunda fila
BIN 00000010 tercera fila
BIN 00111100 cuarta fila
BIN 01010100 quinta fila
BIN 00010100 sexta fila

BIN 00010100 séptima fila
BIN 00000000 octava fila (inferior)

(Si conoce el sistema de numeración binario, podemos decirle que BIN se utiliza para escribir los números en ese sistema.) Observe este grupo de números binarios con los ojos entornados: quizá incluso pueda ver el carácter π .

Estos ocho números se almacenan en ocho posiciones (bytes) de memoria. Cada una de estas posiciones tiene una *dirección*. La dirección del primer byte (o grupo de ocho dígitos binarios) es `USR "P"` (porque escogimos la 'P' en el apartado (ii)). La dirección del segundo byte es `USR "P"+1`, y así hasta llegar al octavo byte, cuya dirección es `USR "P"+7`.

Aquí, `USR` es una función que convierte un argumento literal en la dirección del primer byte de memoria en que está almacenado el correspondiente gráfico definido por el usuario. El argumento literal debe ser un solo carácter, bien el propio gráfico definido por el usuario, bien la letra correspondiente (mayúscula o minúscula). `USR` tiene otro significado cuando su argumento es un número, pero de eso hablaremos más adelante.

Pues bien, aunque no haya conseguido seguirnos en el proceso que acabamos de explicar, pruebe el siguiente programa, que realiza la definición del carácter:

```
10 FOR n=0 to 7
20 READ fila: POKE USR "P"+n,fila
30 NEXT n
40 DATA BIN 00000000
50 DATA BIN 00000000
60 DATA BIN 00000010
70 DATA BIN 00111100
80 DATA BIN 01010100
90 DATA BIN 00010100
100 DATA BIN 00010100
110 DATA BIN 00000000
```

La sentencia `POKE` almacena directamente un número en una posición de memoria, eludiendo el mecanismo normalmente usado por `BASIC`. Lo inverso de `POKE` es `PEEK`, función que permite leer (sin modificarlo) el contenido de las posiciones de memoria. En la Parte 24 de este capítulo describiremos `PEEK` y `POKE` más detalladamente.

Continuando con el juego de caracteres, los siguientes son las *claves*.

Habrás notado que en el primer programa de esta sección no hemos escrito los 32 primeros caracteres (códigos 0 al 31); son los *caracteres de control*. No producen ningún carácter visible, sino que nos permiten controlar la imagen que se forma en la pantalla y algunas otras funciones del +2.

(Si usted intenta escribir caracteres de control, el +2 muestra un ? en la pantalla para indicar que no los entiende. Los caracteres de control están descritos más detalladamente en la Parte 27 de este capítulo.)

Tres de los códigos que afectan a la imagen de la pantalla son el 6, el 8 y el 13. Vamos a explicarlos.

De los tres, CHR\$ 8 es posiblemente el único que tendrá interés para usted.

CHR\$ 6 inserta espacios exactamente de la misma forma que lo hace la coma en las sentencias PRINT. Por ejemplo,

```
PRINT 1; CHR$ 6;2
```

produce el mismo efecto que

```
PRINT 1,2
```

Obviamente, ésta no es una forma muy clara de usarlo. Una manera más comprensible sería:

```
LET a$="1"+CHR$ 6+"2"  
PRINT a$
```

CHR\$ 8 es el *espacio hacia atrás* o *retroceso del cursor*: desplaza la posición de escritura un lugar hacia la izquierda. Pruebe la siguiente orden:

```
PRINT "1234"; CHR$ 8;"5"
```

cuyo efecto final es escribir

```
1235
```

CHR\$ 13 es *línea nueva*: desplaza la posición de escritura al comienzo de la línea siguiente.

La pantalla reconoce también los códigos 16 al 23, que están explicados en las Partes 15 y 16 de este capítulo. (La lista completa de todos los códigos se encuentra en la Parte 27.)

Teniendo en cuenta todos los caracteres «visibles», podemos ampliar el concepto de 'orden alfanumérico' a cadenas que contengan cualquier carácter, no sólo letras. Para ello debemos considerar un alfabeto ampliado que, en lugar de constar de 26 letras, sea la lista de todos los 256 caracteres, en el mismo orden que sus códigos. Por ejemplo, las siguientes cadenas están, para el +2, en orden alfabético. (Observe que, curiosamente, las letras minúsculas van detrás de todas las mayúsculas, de forma que la 'a' es posterior a la 'Z'; además, los espacios son significativos.)

CHR\$ 3+"ZOOLOGICO"

CHR\$ 8+"AARRR"

"AAAARRR!"

"(Nota entre paréntesis)"

"100"

"129.95 inc. IVA"

"AAARRR"

"AaaRRR"

"Elton John"

"PRINT"

"Zoo"

"[interpolación]"

"aaarr"

"aaass"

"derecho"

"zoo"

"zoología"

La regla para ordenar dos cadenas es la siguiente. Primero se compara el primer carácter de una con el primer carácter de la otra. Si son diferentes, uno de los dos caracteres tendrá un código menor que el otro; la primera cadena en orden «alfabético» es la que empieza por el carácter cuyo código es menor. En cambio, si los dos caracteres son iguales, se pasa al segundo carácter de las dos cadenas y se realiza con ellos la comparación; y así sucesivamente hasta que los caracteres comparados sean diferentes. Si, en este proceso, una de las cadenas termina antes que la otra sin que se haya detectado diferencia entre los caracteres, la cadena más corta es la primera; de lo contrario, las dos cadenas son iguales.

Las relaciones =, <, >, <=, >= y <> son aplicables tanto a cadenas como a números: < significa 'es anterior a' y > 'es posterior a', de modo que, por ejemplo, las relaciones

"AA RRR"<"AAARRR"

"AAARRR">"AAA RRR"

son ambas verdaderas.

<= y >= funcionan de la misma forma que con números, y por lo tanto el valor de la relación

"Esta cadena"<="Esta cadena"

es 'verdadero', mientras que el de

"Esta cadena"<"Esta cadena"

es 'falso'.

Para experimentar con todo esto, ejecute el programa siguiente, que capta dos cadenas y las ordena:

```
10 INPUT "Escriba dos cadenas:",a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";
40 IF a$<b$ THEN PRINT "<"; GO TO 60
50 PRINT "=";
60 PRINT " ";b$
70 GO TO 10
```

Observe que, tanto en este programa como en el del final de la Parte 13, hemos tenido que usar c\$ (línea 20) para poder intercambiar los valores de a\$ y b\$. ¿Comprende por qué con

```
LET a$=b$: LET b$=a$
```

no habríamos obtenido el efecto deseado?

El siguiente programa define gráficos de forma que las siguientes teclas produzcan las piezas del ajedrez:

- B para el alfil
- K para el rey
- R para la torre
- Q para la reina
- P para el peón
- N para el caballo

Piezas del ajedrez:

```
5 LET b=BIN 01111100: LET c=BIN 00111000: LET d=BIN 00010000
10 FOR n=1 TO 6: READ p$: REM 6 piezas
20 FOR f=0 TO 7: REM leer 8 bytes
30 READ a: POKE USR p$+f,a
40 NEXT f
50 NEXT n
100 REM alfil
110 DATA "b",0,d, BIN 00101000, BIN 01000100
120 DATA BIN 01101100,c,d,0
130 REM rey
140 DATA "k",0,d,c,d
150 DATA c, BIN 01000100,c,0
160 REM torre
170 DATA "r",0, BIN 01010100,b,c
180 DATA c,b,b,0
```

```
190 REM reina
200 DATA "q",0, BIN 01010100, BIN 00101000,d
210 DATA BIN 01101100,b,b,0
220 REM peón
230 DATA "p",0,0,d,c
240 DATA c,d,b,0
250 REM caballo
260 DATA "n",0,d,c, BIN 01111000
270 DATA BIN 00011000,c,b,0
```

Observe que en las sentencias DATA anteriores hemos puesto 0 en lugar de BIN 00000000.

Cuando haya ejecutado el programa, puede ver las piezas pulsando **GRAF** y cualquiera de las teclas: B, K, R, Q, P o N.

Ejercicios

1. Imagine el espacio para un símbolo dividido en cuatro cuartos. Si cada cuarto puede ser blanco o negro, hay $2^4=16$ posibilidades. Búsquelas todas en el juego de caracteres.
2. Ejecute este programa:

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

Puede comprobar que CHR\$ redondea el número a al entero más cercano; si a no está en el margen de 0 a 255, el programa se detiene y BASIC emite el mensaje de error B integer out of range ('Entero fuera de intervalo').

3. ¿Cuál de las dos cadenas siguientes es menor?

```
"GATO"
"gato"
```

Parte 15

Más sobre PRINT e INPUT

Temas tratados:

CLS

Elementos de PRINT

PRINT 'nada'

TAB, AT

Separadores de PRINT: , ; '

Elementos de INPUT

LINE

Elementos de PRINT que no empiezan con una letra

Desplazamiento de la pantalla

SCREEN\$

Hemos utilizado la orden PRINT bastantes veces, así que usted ya tendrá una idea bastante buena de cómo funciona. Las expresiones cuyos valores escribimos con PRINT son los elementos de PRINT. Pueden estar separadas entre sí por comas, apóstrofos o signos de punto y coma. Todos estos signos de puntuación reciben el nombre de *separadores* de PRINT. Un elemento de PRINT puede ser también 'nada', y esto explica qué ocurre cuando en PRINT ponemos dos comas seguidas.

Hay otros dos tipos de elementos de PRINT que se emplean para comunicar al +2 *dónde* debe escribir, no *qué* debe escribir. Por ejemplo, la instrucción

```
10 PRINT AT 11,16;"*"
```

escribe un asterisco en el centro de la pantalla. Esto se debe a que

AT fila,columna

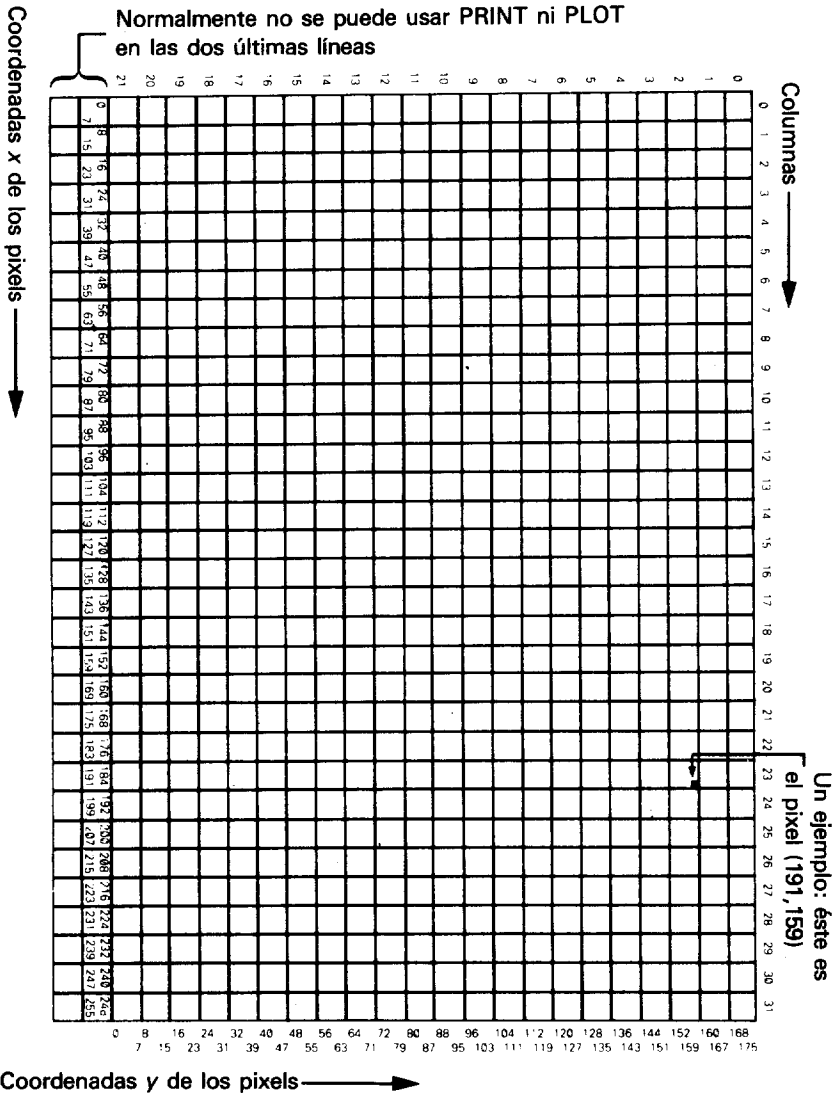
desplaza la *posición de escritura* (o sea, el lugar donde va a aparecer el siguiente elemento que se escriba) a la fila y columna especificadas. Las filas están numeradas de 0 (la más alta) a 21; las columnas están numeradas de 0 (extremo izquierdo) a 31.

SCREEN\$ es la función inversa de PRINT AT, pues «lee» (dentro de ciertos límites) el carácter que se encuentra en la pantalla en la posición especificada. Sus parámetros son los números de fila y columna, con el mismo significado que en PRINT AT, pero puestos entre paréntesis. Por ejemplo, la instrucción

```
20 PRINT AT 0,0; SCREEN$ (11,16)
```

lee el asterisco que escribimos antes en el centro de la pantalla y luego lo reproduce en la posición 0,0 (el extremo superior izquierdo).

Los caracteres de las claves son leídos normalmente (como caracteres sencillos); los espacios son leídos como tales. En cambio, si se intenta leer caracteres definidos por el usuario, caracteres gráficos o líneas dibujadas por PLOT, DRAW o CIRCLE, la función SCREEN\$ da la cadena vacía. Lo mismo ocurre si se ha usado OVER para crear un carácter compuesto. (Las palabras clave PLOT, DRAW, CIRCLE y OVER están descritas en las Partes 16 y 17 de este capítulo.)



La función

TAB *columna*

escribe los espacios necesarios para desplazar la posición de escritura hasta la columna especificada. Intenta no cambiar de línea pero, si ello la obliga a retroceder, prefiere saltar a la línea siguiente. Observe que el +2 toma el número de columna 'módulo 32' (es decir, divide por 32 y toma el resto), de forma que TAB 33 significa lo mismo que TAB 1.

Por ejemplo,

```
PRINT TAB 30; 1; TAB 12; "Contenido"; AT 3,1; "Capitulo"; TAB 25; "Pagina"
```

sería una forma de escribir la cabecera de la primera página de un libro.

Pruebe ahora este programa

```
10 FOR n=0 TO 20
20 PRINT TAB 8*n;n;
30 NEXT n
```

Esto muestra lo que entendemos por 'tomar el número de columna módulo 32'.

Pruebe el programa después de cambiar el 8 de la línea 20 por un 6.

Observe los siguientes detalles:

- (i) Las cláusulas TAB y los elementos de PRINT normalmente deberían terminar en punto y coma. No es que no se pueda poner comas (o nada, al final de la sentencia), pero, después de haber seleccionado tan cuidadosamente la posición de escritura, no nos interesa volver a desplazarla inmediatamente.
- (ii) No se puede escribir en las dos últimas líneas de la pantalla (22 y 23) porque están reservadas para órdenes, captación de datos con INPUT, mensajes de error, informes, etc. Así pues, cuando en lo sucesivo hablemos de 'última línea' normalmente nos estaremos refiriendo a la línea 21.
- (iii) Se puede utilizar AT para situar la posición de escritura en una posición en la que ya haya algo escrito (el nuevo elemento sencillamente reemplazará al antiguo).

Otra sentencia relacionada con PRINT es CLS, cuyo efecto es borrar toda la pantalla.

Cuando al escribir llegamos a la parte inferior de la pantalla, el texto empieza a desplazarse hacia arriba como si se tratase del folio de una máquina de escribir. Para comprobar cómo funciona este mecanismo, seleccione la opción Pantalla en el menú de edición (descrito en el capítulo 6) y luego escriba:

```
CLS: FOR n=1 TO 30: PRINT n: NEXT n
```

Cuando se llena la pantalla, el +2 se detiene y emite el mensaje scroll? ('¿desplazar?') en la parte inferior de la misma. Ahora puede usted observar a su gusto los 22 primeros números. Cuando haya terminado, pulse Y (para yes, 'sí') y el +2 mostrará la siguiente tanda de números. En realidad, da lo mismo pulsar Y que cualquier otra tecla, a excepción de N, la barra espaciadora y **BREAK**. Estas últimas hacen que el programa se detenga y que el ordenador emita el informe D BREAK – CONT repeats.

La sentencia INPUT es capaz de hacer mucho más de lo que le hemos exigido hasta ahora. Ya hemos visto sentencias INPUT similares a

```
INPUT "¿Cuántos años tienes?", edad
```

en la cual el +2 escribe la frase ¿Cuántos años tienes? en la parte inferior de la pantalla y queda a la espera de que el usuario introduzca un número. Sin embargo, una sentencia INPUT puede incluir elementos y separadores, exactamente igual que las sentencias PRINT. ¿Cuántos años tienes? y edad son elementos de INPUT.

Los elementos de INPUT son generalmente iguales a los de PRINT, aunque hay algunas diferencias importantes:

Primero, un elemento adicional obvio de INPUT es la *variable* a la que se asigna valor en la sentencia (en el ejemplo anterior, edad). La regla es que si un elemento INPUT comienza con una letra, BASIC considera que se trata de una variable cuyo valor se va a introducir.

Podría parecer que esto implica que no podremos escribir los valores de las variables como parte de un mensaje inductor. (*Mensaje inductor* es el que se escribe en una sentencia INPUT para indicar al usuario cuál es la naturaleza de los datos que debe introducir.) Sin embargo, esto se resuelve poniendo la variable entre paréntesis. Cualquier expresión que empiece con una letra debe ir entre paréntesis si queremos que INPUT la escriba como parte del mensaje inductor.

Cualquier clase de elemento de PRINT al que no afecten estas reglas es también un elemento de INPUT. Veamos un ejemplo:

```
LET mi edad = INT (RND*100): INPUT ("Tengo ";mi edad;" años.");  
"¿Cuántos años tienes tú?", tu edad
```

Puesto que *mi edad* está entre paréntesis, INPUT escribe su valor. En cambio, *tu edad* no va entre paréntesis, y por eso INPUT entiende que se trata de la variable a la que debe asignar el valor captado.

Todo lo que escribe una sentencia INPUT va a la pantalla inferior, que actúa de forma bastante independiente de la pantalla superior. En particular, sus líneas se numeran a partir de la primera línea de la propia pantalla inferior, aun cuando ésta haya crecido hacia arriba en el monitor (lo cual ocurre cuando hay que escribir muchos datos en respuesta a INPUT). Cualquier cosa que haga la pantalla inferior durante la ejecución de INPUT, su tamaño siempre revertirá a las dos líneas habituales en cuanto se detenga el programa y volvamos al modo de edición.

Para ver cómo funciona AT en una sentencia INPUT, pruebe lo siguiente:

```
10 INPUT "Esta es la línea 1",a$; AT 0,0;"Esta es la línea 0",a$;AT 2,0;"Esta es la
   línea 2",a$; AT 1,0;"Esta sigue siendo la línea1",a$
```

Ejecute este programa (basta con que pulse **INTRO** cada vez que se detenga). Cuando el programa escribe Esta es la línea 2, la pantalla inferior se desplaza hacia arriba para dejarle sitio; pero también la numeración se desplaza hacia arriba, de forma que las líneas del texto conservan los mismos números.

Ahora pruebe esto:

```
10 FOR n=0 TO 19: PRINT AT n,0;: NEXT n
20 INPUT AT 0,0;a$; AT 1,0;a$; AT 2,0;a$; AT 3,0;a$; AT 4,0;a$; AT 5,0;a$;
```

Cuando la pantalla inferior empieza a subir, la pantalla superior permanece inalterada, hasta que la inferior amenaza con escribir en la misma línea que la última sentencia PRINT. Entonces la pantalla superior se desplaza hacia arriba para impedirlo.

Otro refinamiento de la sentencia INPUT que no hemos visto todavía es la opción LINE, que nos ofrece otra forma de captar por el teclado los valores de las variables literales. Si ponemos LINE antes del nombre de la variable literal que va a ser captada, como por ejemplo en

```
INPUT LINE a$
```

el +2 no mostrará las comillas como de costumbre (aunque para sus adentros actuará como si las hubiera escrito). Entonces, si respondemos escribiendo

```
gato
```

INPUT asignará el valor gato a a\$. Puesto que las comillas no aparecen rodeando a la cadena, no podremos borrarlas. Recuerde que no se puede usar LINE cuando la variable es numérica.

Hay un interesante efecto secundario de INPUT. Mientras se está respondiendo a una sentencia INPUT, el antiguo sistema Spectrum de introducción de palabras clave por la pulsación de una sola tecla se comporta de una forma extraña, hasta que lo hacemos entrar en vereda al pulsar **INTRO**. Ejecute este programa:

```
10 INPUT numeros
20 PRINT numeros
30 GO TO 10
```

Introduzca unos cuantos números y verá cómo éstos son reproducidos fielmente en la pantalla. Ahora pulse la tecla **EXTRA** seguida de M. Aparece la palabra PI; si ahora pulsa **INTRO**, el +2 escribe 3.1415927 como por arte de magia. Sin embargo, si usted

escribe las letras PI sin pulsar antes **[EXTRA]** y M, el +2 detiene el programa y emite el informe

2 Variable not found, 10:1 ('Variable no encontrada')

Otra prueba: ejecute el programa; pulse **[EXTRA]** y H; aparece SQR; si ahora introduce 25 y pulsa **[INTRO]**, el +2 responde escribiendo el número 5, que es la raíz cuadrada de 25.

No hay una explicación sencilla para este comportamiento. En todo caso, conviene saber que cosas de éstas pueden suceder si pulsamos ciertas combinaciones de teclas en respuesta a INPUT.

Los caracteres de control **CHR\$ 22** y **CHR\$ 23** producen unos efectos similares a los de **AT** y **TAB**. Siempre que se le pide al +2 que «escriba» uno de ellos, el carácter debe ir seguido por dos caracteres más, que son tratados por el ordenador como parámetros de **AT** o de **TAB**. Normalmente es más cómodo usar explícitamente **AT** y **TAB** que estos dos caracteres, aunque hay situaciones en que pueden ser útiles.

El carácter de control que corresponde a **AT** es **CHR\$ 22**. El primer número que se especifica a continuación es el número de fila; el siguiente, el número de columna. Así,

```
PRINT CHR$ 22+CHR$ 1+CHR$ c;
```

equivale exactamente a

```
PRINT AT 1,c;
```

Esto es así a pesar de que **CHR\$ 1** o **CHR\$ c** tendrían en otra situación significados diferentes (por ejemplo, cuando $c=13$); el **CHR\$ 22** que los precede les cambia el significado.

El carácter de control que corresponde a **TAB** es **CHR\$ 23**; los dos números que le siguen se combinan para dar un número del margen de 0 a 65535, que es el que se toma como parámetro de **TAB**. La sentencia

```
PRINT CHR$ 23+CHR$ a+CHR$ b;
```

es equivalente a

```
PRINT TAB a+256*b;
```

Con **POKE** podemos hacer que el ordenador deje de preguntarnos si queremos desplazar la pantalla ('scroll?'). Para ello podemos escribir

```
POKE 23692,255
```

de vez en cuando. Después de hacer esto, la pantalla tendrá que desplazarse 255 veces antes de que el +2 nos haga la pregunta scroll?. Como ejemplo, pruebe

```
10 FOR n=0 TO 1000
20 PRINT n: POKE 23692,255
30 NEXT n
```

y observe cómo los números se escapan de la pantalla.

Ejercicio

1. Pruebe este programa con algún niño para ver si se sabe las tablas de multiplicar:

```
10 LET m$=""
20 LET a=INT (RND*10)+1: LET b=INT (RND*10)+1
30 INPUT (m$) "¿Cuanto es ";a;" por ";b;"?";c
100 IF c=a*b THEN LET m$="Bien!": GO TO 20
110 LET m$="Mal. Inténtalo otra vez.": GO TO 30
```

Si el niño es un poco despabilado, puede darse cuenta de que no necesita hacer el cálculo él mismo. Por ejemplo, si el +2 le pregunta que cuánto es 2 por 3, todo lo que tiene que hacer es escribir 2*3 literalmente.

Parte 16

Colores

Temas tratados

INK, PAPER, FLASH, BRIGHT, INVERSE, OVER, BORDER

Pruebe este programa

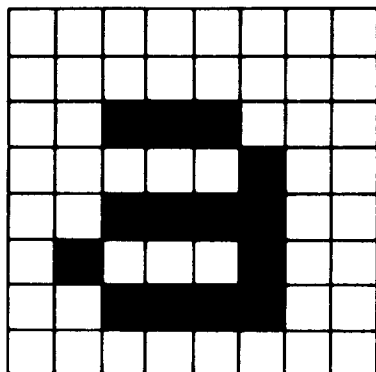
```
10 FOR m=0 TO 1: BRIGHT m
20 FOR n=1 TO 10
30 FOR c=0 TO 7
40 PAPER c: PRINT "  "; REM 4 espacios de colores
50 NEXT c: NEXT n: NEXT m
60 FOR m=0 TO 1: BRIGHT m: PAPER 7
70 FOR c=0 TO 3
80 INK c: PRINT c;" ";
90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;" ";
120 NEXT c: NEXT m
130 PAPER 7: INK 0: BRIGHT 0
```

Esto muestra los ocho colores (incluidos el blanco y el negro) y los dos niveles de brillo que puede producir el +2 en un televisor de color. (Si su aparato es de blanco y negro, lo que se aprecia es diversas intensidades de gris.) Una forma más rápida, y drástica, de obtener el mismo resultado consiste en reinicializar (botón **RESET**) el +2 mientras se tiene pulsada la tecla **BREAK**. Veamos la lista de los colores y sus códigos:

- 0 negro
- 1 azul
- 2 rojo
- 3 magenta
- 4 verde
- 5 cyan
- 6 amarillo
- 7 blanco

En los televisores de blanco y negro, estos números están por orden de luminosidad decreciente. Para usar los colores adecuadamente es necesario entender cómo se forma la imagen en la pantalla.

La imagen está dividida en 768 posiciones (24 filas por 32 columnas), llamadas *celdas*, en las que podemos escribir los caracteres.



Una celda de carácter típica

Cada celda de carácter consiste en una retícula de 8x8 puntos (como la de la figura anterior). Esto debería recordarle a usted los gráficos definidos por el usuario de la Parte 14, donde representábamos con un 0 los puntos blancos y con un 1 los negros.

El carácter tiene dos colores asociados con él: la *tinta*, o color del primer plano, que es el color con el que el +2 dibuja los puntos negros de nuestro cuadrado, y el *papel*, o color del fondo, que es el usado para los puntos blancos. Inicialmente todas las celdas tienen tinta negra y papel blanco, de forma que los caracteres aparecen en negro sobre blanco.

El carácter también tiene un nivel de brillo o luminosidad (normal o extra) asociado, y algo que indica si parpadea o no. El parpadeo consiste en un intercambio continuo de los colores de la tinta y el papel. Toda esta información puede ser codificada en números, de modo que un carácter tiene lo siguiente:

- (i) Una retícula de ceros y unos que definen la forma del carácter (0 para el papel, 1 para la tinta).
- (ii) Un código para el color de la tinta y otro para el del papel, cada uno codificado mediante un número del 0 al 7.
- (iii) Un código para el brillo (0 para el normal, 1 para el extra).
- (iv) Un código para el parpadeo (0 para constante, 1 para parpadeo).

Puesto que cada código de color de tinta y de papel se refiere a una celda de carácter completa, es imposible tener más de dos colores en un bloque dado de 64 puntos. Lo mismo ocurre con los códigos de brillo y parpadeo: se refieren a la célula de carácter completa, no a puntos individuales dentro de ella. Los códigos de color, brillo y parpadeo de un carácter dado son lo que denominamos *atributos* de ese carácter.

Cuando escribimos algo en la pantalla, lo que hacemos es alterar la situación de los puntos de la celda de carácter afectada. Es menos obvio, pero cierto, que también podemos cambiar los atributos de la celda. Mientras no especifiquemos otra cosa, todo se escribe con tinta negra sobre papel blanco (con brillo normal y sin parpadeo); sin embargo, podemos cambiar esta situación utilizando las sentencias INK, PAPER, BRIGHT y FLASH. Seleccione la opción Pantalla del menú de edición para llevar el cursor a la pantalla inferior; dé la siguiente orden:

PAPER 5

y luego escriba (con PRINT) unos cuantos caracteres en la pantalla; aparecerán sobre papel cyan, porque éste es el color que habíamos elegido para el papel (el código del color cyan es el 5).

Las otras instrucciones funcionan de manera similar, así que con

PAPER (número entero entre 0 y 7)
INK (número entero entre 0 y 7)
BRIGHT (número entero entre 0 y 1)
FLASH (número entero entre 0 y 1)

podemos controlar todos los atributos de los caracteres que escribamos a continuación.

Pruebe todas estas instrucciones. Cuando lo haya hecho ya podrá entender cómo funcionaba el programa del principio de esta sección (recuerde que un espacio es un carácter en el que todos los puntos tienen el color del papel).

Hay otros números que podemos usar en estas sentencias y cuyos efectos son menos directos.

El 8 se puede usar en todas las sentencias y significa 'transparente', en el sentido de que respeta el atributo que estuviera antes en vigor. Supongamos, por ejemplo, que damos la orden

PAPER 8

Ninguna posición de carácter puede tener este color de papel, sencillamente porque no existe el color 8; lo que ocurre es que cuando escribamos sobre una posición, el color del papel seguirá siendo el que hubiera antes en ella. Sin embargo, INK 8, BRIGHT 8 y FLASH 8 funcionan del mismo modo que los otros números de atributo.

El número 9 sólo es aplicable a PAPER e INK y significa 'contraste'. Sirve para hacer que el color (de la tinta o del papel) utilizado contraste con el otro, convirtiéndolo en blanco si el otro es un color oscuro (negro, rojo o magenta) o haciéndolo negro si el otro es claro (verde, cyan, amarillo o blanco).

Compruébelo dando estas instrucciones:

```
INK 9: FOR c=0 TO 7: PAPER c: PRINT c: NEXT c
```

Una muestra más impresionante de su potencia es ejecutar el programa del principio para que dibuje las rayas de colores (de nuevo, seleccione la pantalla inferior antes de escribir RUN) y luego hacer lo siguiente:

```
INK 9: PAPER 8: PRINT AT 0,0;: FOR n=1 TO 1000: PRINT n;: NEXT n
```

El color de la tinta contrasta siempre con el color antiguo del papel en todas las celdas de carácter.

Los televisores de color en realidad sólo utilizan tres colores: rojo, verde y azul. Todos los demás colores son mezclas de éstos. Por ejemplo, el magenta se forma mezclando rojo con azul; ésta es la razón por la que su código es 3, ya que este número es la suma de los códigos del rojo (2) y el azul (1).

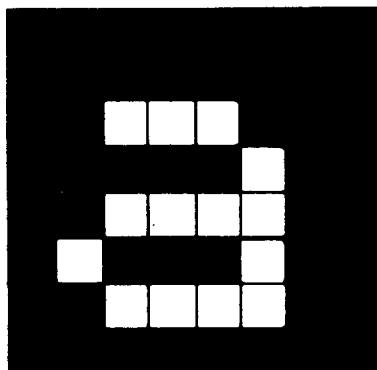
Para comprender cómo se forman los ocho colores, imagine tres focos rectangulares, de colores rojo, verde y azul, que arrojan su luz en la oscuridad sobre tres zonas parcialmente solapadas de un pedazo de papel blanco. Donde las zonas se solapan verá usted mezclas de colores, como muestra el programa siguiente (observe que los cuadrados de tinta sólida se obtienen activando el modo gráfico con la tecla **GRAF** y pulsando luego la tecla del 8 en combinación con **MAYUSC**; para salir del modo gráfico se pulsa la tecla del 9):

```
10 BORDER 0: PAPER 0: INK 7: CLS
20 FOR a=1 TO 6
30 PRINT TAB 6; INK 1; "          " : REM 18 cuadrados de tinta
40 NEXT a
50 LET línea datos=200
60 GO SUB 1000
70 LET línea datos=210
80 GO SUB 1000
90 STOP
200 DATA 2,3,7,5,4
210 DATA 2,2,6,4,4
1000 FOR a=1 TO 6
1010 RESTORE línea datos
1020 FOR b=1 TO 5
1030 READ c: PRINT INK c; "          " : REM 6 cuadrados de tinta
1040 NEXT b: PRINT: NEXT a
1050 RETURN
```

Hay una función, llamada ATTR, que averigua los atributos de una posición dada de la pantalla. Es una función complicada, y por eso la aplazaremos para el final de esta sección.

Disponemos de otras dos sentencias, INVERSE y OVER, que no controlan los atributos, sino la distribución de puntos del carácter que resulta impreso en la pantalla. Llevan como parámetro el número 0, que significa 'activado', o el número 1, que significa

'desactivado'. Así, INVERSE 1 invierte la situación de todos los puntos de la retícula: apaga los encendidos y enciende los apagados. Por ejemplo, la celda de carácter de la letra 'a' se convierte en la que se muestra en la siguiente figura:



Si, como ocurre en el momento de encender el ordenador, tenemos tinta negra y papel blanco, la 'a' aparecerá en blanco sobre negro.

La sentencia

```
OVER 1
```

activa un tipo particular de sobreimpresión. Normalmente, cuando se escribe algo en una posición de la pantalla, el carácter nuevo «tapa» completamente lo que hubiera antes en ella; en cambio, si previamente se da la orden OVER 1, el nuevo carácter se funde con el antiguo. Esto permite escribir caracteres compuestos, por ejemplo, letras subrayadas, como en el programa siguiente. (Reinicialice el +2 y seleccione 128 BASIC. El carácter de subrayado se obtiene con `SIMB` y 0.)

```
10 OVER 1
20 PRINT "w"; CHR$ 8;"_";
```

(Observe que hemos usado el carácter de control CHR\$ 8 para hacer retroceder la posición de escritura antes de escribir el '_'.)

Hay otra forma de usar INK, PAPER, etc. que usted probablemente encontrará más cómoda que la que hemos visto hasta ahora. Podemos utilizarlas como elementos de PRINT seguidas del signo de punto y coma, y harán exactamente lo mismo que si las hubiésemos ejecutado como sentencias independientes, con la única diferencia de que sus efectos son sólo temporales, pues duran solamente hasta el final de la sentencia PRINT que las contiene. Por tanto, si escribimos

```
PRINT PAPER 6;"x";: PRINT "y"
```

sólo la x quedará escrita sobre papel amarillo.

Cuando usamos INK y sus compañeras como sentencias independientes, no afectan al color de la pantalla inferior, en la que se realiza la captación de datos por las sentencias INPUT y donde el +2 muestra los mensajes de error. En esta parte de la pantalla, el color del papel es el del borde, el de la tinta es el 9 (contraste), carece de parpadeo y su brillo es normal. Como color del borde se puede elegir cualquiera de los ocho colores normales (no el 8 ni el 9) mediante la instrucción

BORDER *color*

Cuando se está ejecutando INPUT, los caracteres introducidos aparecen en tinta de contraste sobre papel del mismo color que el borde; pero se puede cambiar el color de los mensajes inductores escritos por el +2 usando para ello INK y PAPER (y las demás) como elementos de INPUT, de la misma forma que se haría en una sentencia PRINT. Sus efectos duran hasta el final de la sentencia o hasta que se termine de introducir los datos, lo que antes ocurra. Pruebe lo siguiente:

```
INPUT FLASH 1; INK 1;"¿Cual es su numero?";n
```

El +2 tiene en gran estima su salud mental: cualquiera que sea la combinación de colores producida por el programa, el editor siempre funciona con tinta negra sobre papel blanco.

Otra forma de cambiar los colores consiste en usar caracteres de control, de forma parecida a como hacíamos con AT y TAB de la Parte 15.

CHR\$ 16	corresponde a	INK
CHR\$ 17	corresponde a	PAPER
CHR\$ 18	corresponde a	FLASH
CHR\$ 19	corresponde a	BRIGHT
CHR\$ 20	corresponde a	INVERSE
CHR\$ 21	corresponde a	OVER

Estos caracteres de control van seguidos de un carácter que especifica un color; así, por ejemplo,

```
PRINT CHR$ 16+CHR$ 9;"elemento"
```

produce el mismo efecto que

```
PRINT INK 9;"elemento"
```

En la práctica, no tiene ningún interés utilizar estos caracteres de control, pues siempre podemos emplear en su lugar las sentencias INK, PAPER, etc. Sin embargo, si usted tiene en cassette algún programa escrito para el antiguo 48 BASIC, puede encontrar tales caracteres de control inmersos en el listado. En general, el editor los ignorará activamente y los suprimirá a la primera oportunidad. No es posible introducirlos en los listados, como se hacía en el antiguo Spectrum de 48K.

La función ATTR tiene la forma

ATTR (*fila,columna*)

Sus dos argumentos son los números de fila y columna que ya sabemos utilizar en la cláusula AT. El resultado de la función es un número que informa sobre los colores y los demás atributos de la posición de carácter especificada. Esta función puede intervenir en expresiones, con la misma libertad que cualquier otra función.

El número resultante es una combinación de cuatro números que se construye de la siguiente forma:

sumar 128 si la celda de carácter está parpadeando; 0 en otro caso
sumar 64 si la celda de carácter es brillante; 0 si es normal
sumar 8 multiplicado por el código del color del papel
sumar 1 multiplicado por el código del color de la tinta

Por ejemplo, si la celda de carácter está parpadeando, tiene un brillo normal, papel amarillo y tinta azul, los cuatro números que debemos sumar son 128, 0, $8*6=48$ y 1, lo que da un total de 177. Compruébelo escribiendo:

```
PRINT AT 0,0; FLASH 1; PAPER 6; INK 1;" ";ATTR (0,0)
```

Ejercicios

1. Pruebe la siguiente orden:

```
PRINT "--"; CHR$ 8; OVER 1;"/";
```

En el lugar de intersección de la barra con el signo menos ha quedado un punto blanco. Ésta es la forma de sobreimpresión en el +2: papel más papel y tinta más tinta dan papel; tinta más papel da tinta. Este sistema tiene la interesante propiedad de que si sobreescribimos la misma cosa dos veces, se restaura el carácter original. Sabido esto, si ahora hacemos

```
PRINT CHR$ 8; OVER 1;"/"
```

¿por qué recuperamos un ‘-’ incólume?

2. Ejecute este programa:

```
10 POKE 225227+RND*704, RND*127  
20 GO TO 10
```

(No se preocupe por cómo funciona el programa.) El programa está cambiando los colores de las células de la pantalla del televisor; RND garantiza que esto ocurre de forma aleatoria. Las rayas diagonales que finalmente llegan a aparecer son la manifestación del hecho de que RND no genera verdaderos números aleatorios, sino pseudoaleatorios.

Parte 17

Gráficos

Temas tratados:

PLOT, DRAW, CIRCLE
pixels

A lo largo de toda esta sección, escriba los programas de ejemplo, las órdenes y la palabra RUN en la pantalla inferior (selecciónela con la opción Pantalla del menú de edición).

En esta sección vamos a aprender a dibujar figuras en el +2. La parte de la pantalla que usted puede utilizar para los dibujos tiene 22 filas y 32 columnas, lo que da un total de $22 \times 32 = 704$ posiciones de carácter. Como recordará de la Parte 16, cada posición de carácter es una retícula de 8×8 puntos, llamados *pixels* (de *picture elements*, 'elementos de imagen').

Un pixel se especifica dando dos números: sus *coordenadas*. El primero, su coordenada x , indica a qué distancia se encuentra el pixel del borde izquierdo de la pantalla. El segundo, su coordenada y , indica la distancia que hay del pixel al borde inferior. Estas coordenadas se escriben generalmente entre paréntesis y separadas por comas, de modo que, por ejemplo, (0,0), (255,0), (0,175) y (255,175) representan, respectivamente, los rincones inferior izquierdo, inferior derecho, superior izquierdo y superior derecho de la pantalla.

Si no consigue recordar cuál es la coordenada x y cuál la coordenada y , quizá le sirva de ayuda asociar mentalmente la forma de la 'y' con la 'v' de 'vertical'.

La sentencia

PLOT *coordenada x*, *coordenada y*

«tiñe» el pixel correspondiente a esas coordenadas, de forma que este programa

```
10 PLOT INT (RND*256), INT (RND*176): INPUT a$: GO TO 10
```

dibuja un punto en una posición aleatoria cada vez que usted pulsa **INTRO**.

Veamos un programa más interesante; traza un gráfico de la función SIN (una onda sinusoidal) para valores del arco comprendidos entre 0 y 2π :

```
10 FOR n=0 TO 255
20 PLOT n,88+80*SIN (n/128*PI)
30 NEXT n
```

El siguiente programa traza un gráfico de SQR (parte de una parábola) entre 0 y 4:

```
10 FOR n=0 TO 255
20 PLOT n,80*SQR (n/64)
30 NEXT n
```

Observe que las coordenadas que describen la posición de los pixels son distintas de las que utilizábamos para dar el número de fila y de columna en la cláusula AT. Quizá le resulte útil el diagrama de la Parte 15 de este capítulo a la hora de elegir las coordenadas de los pixels y los números de fila y columna.

PLOT realiza los dibujos punto a punto. También podemos dibujar rectas, circunferencias y trozos de circunferencia utilizando las sentencias DRAW y CIRCLE.

La sentencia DRAW dibuja una recta; su forma es:

```
DRAW x,y
```

El punto de partida de la recta es el pixel en el que se detuvo la última sentencia PLOT, DRAW o CIRCLE (que es lo que llamamos *posición actual* del cursor gráfico; RUN, CLEAR, CLS y NEW la restablecen en el extremo inferior izquierdo, punto 0,0). El punto final de la recta se encuentra a x pixels a la derecha de la posición actual y a y por arriba. La sentencia DRAW determina por sí misma la longitud y la dirección de la recta, pero no su punto de partida.

Experimente con unas cuantas órdenes PLOT y DRAW; por ejemplo,

```
PLOT 0,100: DRAW 80,-35
PLOT 90,150: DRAW 80,-35
```

Observe que los números de la sentencia DRAW pueden ser negativos, porque representan desplazamientos con respecto a una posición, mientras que los de PLOT tienen que ser positivos, porque representan coordenadas absolutas.

También podemos dibujar en color, aunque hay que tener presente que los colores cubren siempre la totalidad de la celda de un carácter y no pueden ser especificados para pixels individuales. Cuando dibujamos un pixel, éste se tiñe del color de la tinta, y el resto de la celda en que se encuentra el pixel toma el mismo color, como demuestra el siguiente programa:

```
10 BORDER 0: PAPER 0: INK 7: CLS: REM tiñe la pantalla de negro
20 LET x1=0: LET y1=0: REM comienzo de la recta
30 LET c=1: REM para color de tinta, comenzando con azul
40 LET x2=INT (RND*256): LET y2=INT (RND*176): REM final aleatorio de la
   recta
50 DRAW INK c;x2-x1,y2-y1
60 LET x1=x2: LET y1=y2: REM la siguiente recta comienza donde acabó la anterior
70 LET c=c+1: IF c=8 THEN LET c=1: REM otro color
80 GO TO 40
```

Las rectas parecen hacerse más amplias a medida que avanza el programa. Esto se debe a que cada recta cambia los colores de todas las celdas que atraviesa. Observe que podemos introducir las cláusulas PAPER, INK, FLASH, BRIGHT, INVERSE y OVER en las sentencias PLOT y DRAW, lo mismo que hacíamos en PRINT e INPUT. Estas cláusulas van entre la palabra clave y las coordenadas y terminan en coma o en punto y coma.

Una característica adicional de DRAW es que podemos usarla para dibujar fragmentos de circunferencias en vez de rectas. La sentencia necesita entonces un número más para especificar el ángulo del trozo de circunferencia. La forma es

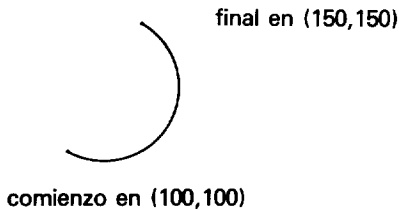
`DRAW x,y,a`

donde *x* e *y* especifican, al igual que antes, el punto final de la línea, y donde *a* especifica el arco (en número de radianes). Si *a* es positivo, el giro se realiza hacia la izquierda; si es negativo, hacia la derecha. Otra forma de entender el significado de *a* es considerar que indica qué fracción de una circunferencia completa se va a dibujar (una circunferencia completa tiene 2π radianes). Por ejemplo, si $a=\pi$, obtendremos una semicircunferencia; si $a=0.5\pi$, un cuarto de circunferencia; y así sucesivamente.

Supongamos que $a=\pi$. Entonces, cualesquiera que sean los valores de *x* e *y*, siempre obtendremos una circunferencia. Pruebe el programa:

```
10 PLOT 100,100: DRAW 50,50,PI
```

que dibuja lo siguiente:



El dibujo comienza en dirección sudeste, pero, en el momento en que se detiene, ya va en dirección noroeste. Entre el rumbo inicial y el final ha girado 180 grados, o π radianes, que es el valor de *a*.

Ejecute el programa varias veces reemplazando PI por otras expresiones; por ejemplo: $-\pi$, $\pi/2$, $3\pi/2$, $\pi/4$, 1, 0, etc.

La última sentencia de esta sección es CIRCLE, que dibuja una circunferencia completa. La circunferencia se especifica dando las coordenadas del centro y el radio. La forma de CIRCLE es:

`CIRCLE coordenada x, coordenada y, radio`

Al igual que sucedía con PLOT y DRAW, al principio de CIRCLE podemos poner cláusulas que controlen el color y los otros atributos del dibujo.

La función POINT averigua si un pixel tiene el color de la tinta o el del papel. Sus dos argumentos son las coordenadas del pixel (que deben ir entre paréntesis). El resultado es 0 si el pixel tiene el color del papel, o 1 si es del color de la tinta. Pruebe lo siguiente:

```
CLS : PRINT POINT (0,0): PLOT 0,0: PRINT POINT (0,0)
```

Escriba:

```
PAPER 7: INK 0
```

e investiguemos ahora cómo funcionan INVERSE y OVER dentro de una sentencia PLOT. Ambas afectan sólo al pixel especificado en PLOT, no al resto de la celda de carácter. Normalmente estas cláusulas están desactivadas (a 0) en las sentencias PLOT, así que sólo es necesario mencionarlas cuando se desee activarlas (ponerlas a 1).

He aquí una lista de las posibilidades:

PLOT;

Ésta es la forma usual. Dibuja un punto con la tinta; es decir, tiñe el pixel con el color de la tinta.

PLOT INVERSE 1;

Dibuja un punto con el 'borrador de tinta'; es decir, hace que el pixel muestre el color del papel.

PLOT OVER 1;

Intercambia el color del pixel con el que tuviera antes, de modo que si antes era el de la tinta lo convierte en el del papel, y vice versa.

PLOT INVERSE 1; OVER 1;

Esto deja el pixel exactamente como estaba antes, pero cambia la posición actual; puede servir, pues, para mover el cursor gráfico.

Para ver otro ejemplo de aplicación de la sentencia OVER, llene la pantalla de texto escrito en negro sobre blanco y después haga

```
PLOT 0,0: DRAW OVER 1;255,175
```

Esto dibuja una recta bastante aceptable, aunque con puntos en blanco cada vez que se tropieza con algo escrito previamente. Ahora dé otra vez exactamente la misma orden: la recta desaparece sin dejar rastro alguno; ésta es la gran ventaja de OVER 1. Si hubiéramos dibujado la recta con

```
PLOT 0,0: DRAW 255,175
```

y la hubiéramos borrado con

```
PLOT 0,0: DRAW INVERSE 1;255,175
```

habríamos borrado también parte del texto.

Ahora pruebe

```
PLOT 0,0: DRAW OVER 1;250,175
```

y trate de borrar con

```
DRAW OVER 1;-250,-175
```

Esto no funciona demasiado bien, y es que los pixels por los que pasa la recta en su camino de vuelta no son exactamente los mismos que los que había recorrido a la ida. Por consiguiente, para borrar una recta hay que recorrerla en el mismo sentido en que se la dibujó.

Una manera de obtener colores especiales es mezclar dos colores normales en un cuadrado, usando para ello un gráfico definible por el usuario. Pruebe el siguiente programa:

```
1000 FOR n=0 TO 6 STEP 2
1010 POKE USR "a"+n, BIN 01010101: POKE USR "a"+n+1, BIN 10101010
1020 NEXT n
```

que genera un gráfico definido por el usuario con el diseño de un tablero de ajedrez. Si ahora escribimos este carácter (pulse **GRAF** y luego **A**) en tinta roja sobre papel amarillo, obtendremos un naranja bastante aceptable.

Ejercicios

1. Experimente con las cláusulas PAPER, INK, FLASH y BRIGHT en una sentencia PLOT. Éstas son las partes que afectan a la totalidad de la celda de carácter en la que se encuentra el pixel. Normalmente, es como si la sentencia PLOT hubiera empezado con

PLOT PAPER 8; FLASH 8; BRIGHT 8; etc.

y sólo se altera el color de la tinta de una celda de carácter cuando se dibuja algo en ella, pero usted puede cambiar esta situación si lo desea.

Tenga cuidado especialmente cuando use colores con INVERSE 1, ya que esta cláusula hace que el pixel muestre el color del papel, y puede cambiar el color de la tinta, el cual pudiera no ser el que usted esperaba.

2. Pruebe a dibujar circunferencias usando SIN y COS con el siguiente programa (si ha leído la Parte 10, trate de entender cómo funciona):

```
10 FOR n=0 TO 2*PI STEP PI/180
20 PLOT 100+80*COS n,87+80*SIN n
30 NEXT n
40 CIRCLE 150,87,80
```

Como puede ver, la sentencia CIRCLE es mucho más rápida, pero menos precisa.

3. Pruebe lo siguiente:

```
CIRCLE 100,87,80: DRAW 50,50
```

De esto se deduce que la sentencia CIRCLE deja el cursor en una posición bastante imprecisa (siempre se trata de algún punto a media altura en el lado derecho de la circunferencia). Normalmente habrá que ejecutar una sentencia PLOT a continuación de una CIRCLE antes de seguir dibujando.

Parte 18

Movimiento

Temas tratados:

PAUSE, INKEY\$, PEEK

Con frecuencia se necesita poder controlar el tiempo que dura la ejecución de un programa. Para este fin se dispone de la sentencia PAUSE.

PAUSE *n*

detiene el programa y mantiene la imagen durante *n* barridos del cuadro (a razón de 50 barridos por segundo en Europa, 60 en EE.UU.). El valor de *n* puede llegar hasta 65535, número que produce una pausa de 22 minutos. Si *n*=0, la pausa es de duración infinita.

La pausa se abandona en cualquier momento pulsando una tecla.

El siguiente programa mueve el segundero de un reloj:

```
10 REM primero dibujamos la esfera del reloj
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
40 NEXT n
50 REM ahora ponemos en marcha el reloj
60 FOR t=0 TO 200000: REM t es el tiempo en segundos
70 LET a=t/30*PI: REM a es el ángulo del segundero en radianes
80 LET sx=80*SIN a: LET sy=80*COS a
200 PLOT 128,88: DRAW OVER 1;sx,sy: REM dibuja el segundero
210 PAUSE 42
220 PLOT 128,88: DRAW OVER 1;sx,sy: REM borra el segundero
400 NEXT t
```

El reloj deja de funcionar al cabo de unas 55.5 horas, debido a la línea 60, pero usted puede darle más cuerda fácilmente. Observe que el control del tiempo lo lleva la línea 210. Parecería natural que PAUSE 50 realizase el batido de segundos, pero también hay que tener en cuenta el tiempo que tarda el ordenador en llevar a cabo las restantes instrucciones del programa. La forma de ajustar el parámetro de PAUSE es comparar el funcionamiento del programa con un cronómetro hasta dar con el valor correcto. (El ajuste no puede ser demasiado perfecto; una diferencia de un barrido más o menos representa un error del 2%, es decir, aproximadamente media hora al día.)

Hay otro método mucho más exacto para medir el tiempo, basado en el contenido de ciertas posiciones de memoria. Los datos almacenados en la memoria se leen con la función PEEK. La Parte 25 de este capítulo explica con detalle el significado exacto de las posiciones de memoria que vamos a leer. La expresión que necesitamos es:

$$(65536 * \text{PEEK } 23674 + 256 * \text{PEEK } 23673 + \text{PEEK } 23672) / 50$$

que da el número de segundos transcurridos desde que se encendió o reinició el ordenador (hasta alrededor de tres días y 21 horas, porque después de ese tiempo vuelve a 0).

El siguiente programa es una versión revisada del programa anterior en la que utilizamos esa expresión:

```
10 REM primero dibujamos la esfera del reloj
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
40 NEXT n
50 DEF FN t()=INT ((65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50):
    REM número de segundos desde la reinicialización
100 REM ahora ponemos en marcha el reloj
110 LET t1=FN t()
120 LET a=t1/30*PI: REM a es el ángulo del segundero en radianes
130 LET sx=72*SIN a: LET sy=72*COS a
140 PLOT 131,91: DRAW OVER 1;sx,sy: REM dibujar segundero
200 LET t=FN t()
210 IF t<=t1 THEN GO TO 200: REM esperar hasta que llegue el momento de
    moverlo
220 PLOT 131,91: DRAW OVER 1;sx,sy: REM borrar segundero
230 LET t1=t: GO TO 120
```

El reloj interno en el que se basa este método tiene una precisión de alrededor del 0.01% (aproximadamente 10 segundos por día), a condición de que lo único que haga el ordenador sea ejecutar el programa. Sin embargo, cuando se ejecuta la sentencia BEEP (descrita en la Parte 19 de este capítulo) o se trabaja con el magnetófono, la impresora o cualquier otro periférico, el reloj interno se detiene temporalmente y por lo tanto se atrasa.

Los números PEEK 23674, PEEK 23673 y PEEK 23672 están almacenados en el ordenador y constituyen un contador de cincuentavos de segundo. Crecen gradualmente de 0 a 255, y, al llegar a 255, vuelven a cero.

El que se incrementa más a menudo es PEEK 23672 (una unidad cada 1/50 segundos). Cuando llega a 255, el siguiente incremento lo lleva a 0, y al mismo tiempo suma 1 a PEEK 23673. Cuando PEEK 23673 pasa de 255 a 0 (cada 256/50 segundos), esto a su vez suma 1 a PEEK 23674. Esta explicación debería ser suficiente para que usted comprenda cómo funciona la expresión anterior.

Suponga que nuestros tres números son 0 (PEEK 23674), 255 (PEEK 23673) y 255 (PEEK 23672). Esto significa que hace unos 21 minutos que se encendió el ordenador. El valor de la expresión será

$$(65536*0+255+255)/50=1310.7$$

Pero hay un peligro oculto: la próxima vez que se incremente el contador en 1/50 segundos, los tres números cambiarán a 1, 0 y 0. Alguna vez ocurrirá esto en el preciso momento en que el ordenador está calculando la expresión; entonces el +2 leerá el número 0 en PEEK 23674, pero los otros dos valores cambiarán a cero antes de que el programa los haya leído. El valor de la expresión será en ese caso:

$$(65536*0+256*0+0)/50=0$$

que, evidentemente, no es correcta.

Una manera sencilla de evitar este problema es evaluar la expresión dos veces seguidas y tomar la respuesta mayor. Si usted observa cuidadosamente el programa anterior, verá que hace esto implícitamente.

Veamos un truco para aplicar esta regla. Defina las funciones:

```
10 DEF FN m(x,y)=(x+y+ABS (x-y))/2: REM el mayor de x e y
20 DEF FN u()=(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50:
  REM tiempo (puede ser incorrecto)
30 DEF FN t()=FN m(FN u(), FN u()): REM tiempo (correcto)
```

Podemos cambiar los tres números del contador para que den el tiempo real en vez del tiempo que lleva el ordenador encendido. Por ejemplo, para poner el contador en hora a las 10 de la mañana, observemos que 10 horas son $10 \times 60 \times 60 \times 50 = 1800000$ cincuentavos de segundo y que

$$1800000=65536*27+256*119+64$$

Por eso debemos escribir los números 27, 119 y 64 en las tres posiciones de memoria, y eso se hace con la siguiente orden:

```
POKE 23674,27: POKE 23673,119: POKE 23672,64
```

En los países en los que la frecuencia de la red de energía eléctrica sea de 60 Hz, el número 50 que aparece en todos estos programas debe ser cambiado por 60.

La función INKEY\$ (que no tiene argumento) lee el teclado. Si en el momento de evaluar INKEY\$ tenemos pulsada una tecla sola (o combinada con MAYUSC), el resultado es el carácter que esa tecla da normalmente; de lo contrario, el resultado es la cadena vacía.

Pruebe este programa, que funciona como una máquina de escribir:

```
10 IF INKEY$<>"" THEN GO TO 10
20 IF INKEY$="" THEN GO TO 20
30 PRINT INKEY$;
40 GO TO 10
```

La línea 10 espera hasta que usted retira sus dedos del teclado; la línea 20 espera hasta que pulsa una nueva tecla.

Recuerde que, a diferencia de INPUT, INKEY\$ no le espera, de modo que no tiene que pulsar **[INTRO]**. Por otra parte, si usted no escribe absolutamente nada, ha perdido su oportunidad.

Ejercicios

1. ¿Qué ocurre si omitimos la línea 10 del programa de la máquina de escribir?
2. Otra forma de usar INKEY\$ consiste en combinarla con PAUSE, como en este segundo programa de máquina escribir:

```
10 PAUSE 0
20 PRINT INKEY$;
30 GO TO 10
```

¿Por qué es esencial, para que esto funcione, que la pausa no acabe si ya se encuentra usted pulsando una tecla cuando comienza?

3. Adapte el programa del segundero de forma que también muestre el minuterero y la manecilla de las horas, redibujándolas una vez por minuto. Si se siente usted ambicioso, haga que produzca algún efecto adicional cada cuarto de hora, quizá el repique de las campanas del 'Big Ben' usando PLAY (orden descrita en la Parte 19 de este capítulo).

Parte 19

Sonido

Temas tratados:

BEEP, PLAY

Como usted ya habrá observado, el +2 puede producir diversos ruidos. Para obtener la mejor calidad de sonido posible, es importante que el televisor esté perfectamente sintonizado (véase capítulo 2). Si en lugar de un televisor está usted usando un monitor (que no reproducirá el sonido del +2), observe que puede tomar del zócalo **SOUND** de la parte posterior del ordenador una señal de sonido, la cual puede ser enviada a los altavoces o auriculares a través de un amplificador de audio. Tenga en cuenta, no obstante que los auriculares *no* pueden ser conectados directamente al ordenador.

Las conexiones del zócalo **SOUND** están descritas en el capítulo 10.

El conocimiento de la terminología musical ayuda a extraer el mayor partido posible de la capacidad sonora del +2.

Nota. En los ejemplos que siguen, es importante que usted escriba las expresiones literales exactamente como nosotros las mostramos, en mayúsculas y minúsculas; así, por ejemplo, "ga" no es lo mismo que "Ga", "gA" o "GA".

Escriba esta orden (de momento no se preocupe por lo que significa):

PLAY "ga"

Suenan dos notas, la segunda ligeramente más alta (aguda) que la primera. La diferencia entre las notas se llama *tono*. Ahora pruebe

PLAY "g\$a"

De nuevo han sonado dos notas; la primera era la misma que en el ejemplo anterior, pero el salto hasta la segunda fue menor. Si no ha notado la diferencia, pruebe otra vez el primer ejemplo y luego el segundo.

La diferencia entre las dos notas del segundo ejemplo es un *semitono*.

Cuando se haya familiarizado con los semitonos, pruebe esto:

PLAY "gD"

Esta diferencia se llama *quinta* y aparece frecuentemente en todos los tipos de música. Con ese ejemplo todavía en sus oídos, escriba:

PLAY "gG"

A pesar de que ahora la diferencia entre las notas ha sido mucho mayor que en el caso de la quinta, de algún modo las notas sonaron bastante más parecidas. Esta diferencia se llama una octava, y es el punto a partir del cual la música empieza a 'repetirse a sí misma'. No se preocupe demasiado por esto; basta con que recuerde cómo suena una octava.

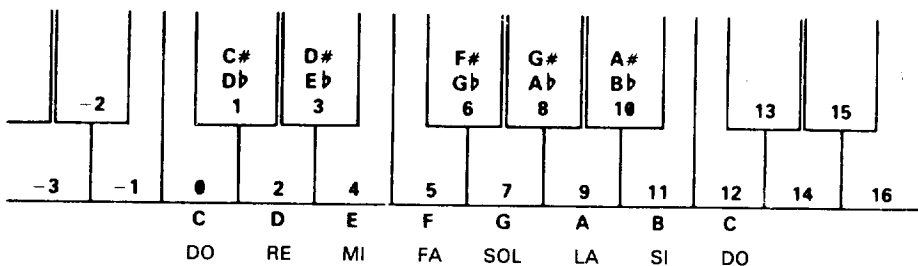
Hay dos formas de producir música y sonidos con el +2. La más elemental es la orden, un poco espartana, BEEP. Su forma es:

BEEP *duración, altura*

donde, como de costumbre, *duración* y *altura* son expresiones numéricas. La duración se da en segundos; la altura, en número de semitonos por encima de la nota DO medio; los números negativos representan las notas por debajo de la DO medio.

NOTA	
Para representar las notas musicales, el +2 utiliza la nomenclatura anglosajona. La equivalencia entre ésta y la española es como sigue:	
C	DO
D	RE
E	MI
F	FA
G	SOL
A	LA
B	SI

El siguiente diagrama muestra los valores del parámetro *altura* para todas las notas de una octava en el piano.



Por consiguiente, para producir la nota LA que está por encima de la DO medio, con duración de medio segundo, debemos dar la orden:

```
BEEP 0.5,9
```

Para interpretar una escala (por ejemplo, DO mayor) se necesita un programa completo (aunque corto):

```
10 FOR f=1 to 8
20 READ nota
30 BEEP 0.5,nota
40 NEXT f
50 DATA 0,2,4,5,7,9,11,12
```

Para obtener notas más altas o más bajas que las mostradas en el diagrama, se debe sumar o restar 12 por cada octava que quiera subir o bajar.

La orden BEEP ha sido incluida más que nada por compatibilidad con modelos más antiguos del Spectrum, aunque también puede ser útil para generar efectos sonoros muy cortos o rápidos. Para cualquier programa nuevo que usted desarrolle, la segunda forma de producir sonido, basada en la orden PLAY, es, con mucho, la preferible.

PLAY es mucho más flexible que BEEP; puede interpretar hasta tres voces en armonía con todo tipo de efectos, y ofrece un sonido de calidad muy superior. Además, es bastante más fácil de usar. Por ejemplo, para interpretar la nota LA posterior a DO medio durante medio segundo, dé la orden

```
PLAY "a"
```

Para interpretar la escala de DO mayor basta con

```
PLAY "cdefgabC"
```

Observe que la última C de este ejemplo es mayúscula. Este hecho indica a la orden PLAY que debe interpretar esta nota una octava más arriba que la representada por la c minúscula. A propósito, una *escala* es el nombre que se da al conjunto de las notas que hay en una octava. La escala del ejemplo anterior se llama la 'escala de Do mayor' porque es el conjunto de notas entre un DO y el siguiente. ¿Por qué 'mayor'? Hay dos tipos básicos de escala: mayor y menor. Esta terminología es sólo una forma abreviada de describir dos conjuntos diferentes. Sólo por si le interesa, la escala de DO menor suena así:

```
PLAY "cd$efg$a$bC"
```

Cuando una nota va precedida de \$, suena un semitono más baja (*bemol*); si el prefijo es #, suena un semitono más alta (*sostenido*). La orden PLAY cubre 9 octavas; se puede especificar la octava deseada escribiendo la letra O mayúscula seguida de un número.

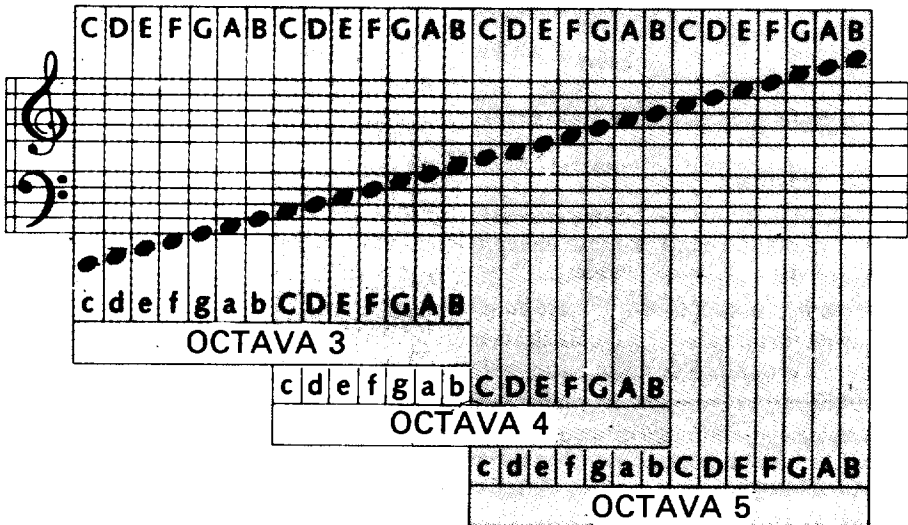
Pruebe este programa:

```
10 LET o$="O5"  
20 LET n$="DECcg"  
30 LET a$=o$+n$  
40 PLAY a$
```

Hay unas cuantas novedades en este programa. En primer lugar, **PLAY** funciona igual de bien con una variable literal que con una constante. En otras palabras, suponiendo que **a\$** ha sido definida con anterioridad, **PLAY a\$** produce el mismo efecto que **PLAY "O5DECcg"**. De hecho, el uso de variables en sentencias **PLAY** tiene algunas otras ventajas, y nosotros las emplearemos de aquí en adelante.

Observe también que la cadena **a\$** se ha formado por concatenación de dos cadenas más pequeñas: **o\$** y **n\$**. Aunque esto no representa gran ventaja cuando las cadenas son así de cortas, el hecho es que **PLAY** puede manejar cadenas de una longitud de muchos miles de notas, y la única forma práctica de crear y editar esas cadenas en **BASIC** es combinar cadenas más pequeñas.

Ahora ejecute el programa anterior. Edite la línea 10 para poner "O7" en lugar de "O5" y ejecútelo de nuevo. Pruébelo también con "O2". Si no especifica número de octava en una cadena determinada, el +2 toma por defecto la octava 5. El siguiente diagrama da las notas y números de octava que corresponden a la 'escala musical bien temperada'.



Hay un solapamiento considerable; por ejemplo, "O3D" es lo mismo que "O4d". Gracias a esto se puede escribir melodías sin tener que especificar demasiados cambios de

escala. Por razones técnicas, algunas de las notas de las octavas más bajas (0 y 1) no son muy exactas, pero el ordenador trata de desafinar lo menos posible.

PLAY puede producir notas de diferente duración. Edite el programa anterior de forma que la línea 10 sea ahora:

```
10 LET o$="2"
```

y ejecútelo. Después pruebe con otros valores de o\$, entre "1" y "9". La longitud de nota se puede especificar en cualquier lugar de la cadena incluyendo un número entre el "1" y el "9"; la nueva duración queda en vigor hasta que PLAY encuentra otro número más adelante. Cada una de estas nueve duraciones de nota tiene un nombre musical específico. La tabla siguiente da los nombres y notaciones musicales:

Número	Nombre de la nota	Símbolo musical
1	simicorchea	
2	semicorchea con puntillo	
3	corchea	
4	corchea con puntillo	
5	negra	
6	negra con puntillo	
7	blanca	
8	blanca con puntillo	
9	redonda	

PLAY también puede producir *tresillos*, que son grupos de tres notas interpretadas en el tiempo de dos. A diferencia de las duraciones de nota sencilla, el número de tresillo sólo afecta a las tres notas siguientes; después de ellas continúa en vigor el anterior número de duración. Los números de tresillo son los siguientes:

Número	Nombre de la nota	Símbolo musical
10	tresillo de semicorcheas	
11	tresillo de corcheas	
12	tresillo de negras	

Además, PLAY sabe callarse a tiempo. Al periodo de tiempo durante el cual no se interpreta ninguna nota se lo llama *silencio*. Un silencio se representa por &; su longitud es la que esté en vigor para las notas. Edite las líneas 10 y 20 y cámbielas por

```
10 LET o$="O4"
20 LET n$="DEC&cg"
```

Dos notas ejecutadas juntas forman una *ligadura*. En PLAY las ligaduras se especifican mediante un signo de subrayado. Por ejemplo, una DO negra y una DO blanca ligadas se representan por 5_7c (el segundo de los dos números especifica la duración para las notas siguientes).

Hay ocasiones en las que se presenta alguna ambigüedad. Supongamos que una pieza musical necesita la octava 6 y una duración de nota de 2; entonces

```
10 LET o$="O62"
```

parece una buena baza, pero no lo es. El ordenador encontrará la O y tratará de leer el número siguiente. Como lo que encuentra es el 62, se detiene y emite el mensaje de error n Out of range ('Fuera del margen'). Para casos como éste disponemos de una 'nota auxiliar', llamada N, que sólo sirve como separador. Así, en la línea 10 deberíamos poner:

```
10 LET o$="O6N2"
```

El volumen es ajustable entre 0 (mínimo) y 15 (máximo) escribiendo el número detrás de la letra V. En la práctica, si no se utiliza un amplificador, sólo resultan útiles los números del 10 al 15, ya que los del 0 al 9 son demasiado suaves. Como ya hemos mencionado, BEEP produce un sonido más intenso que un canal de PLAY; pero si se hace sonar PLAY en los tres canales a volumen 15, la intensidad será igual a la de BEEP.

El manejo de varios canales es muy sencillo; basta con especificar en PLAY varias listas de notas, separadas con comas. Pruebe este nuevo programa:

```
10 LET a$="O4cCcCgGgG"  
20 LET b$="O6CaCe$bd$bD"  
30 PLAY a$,b$
```

En general, no hay diferencia entre los tres canales, y cualquier cadena de notas puede ser reproducida en cualquier canal. La velocidad global de la música, el *tempo*, debe estar en la cadena asignada al canal A (la primera que se especifica tras PLAY), pues de lo contrario será ignorada. Para especificar el tempo en número de notas (negras) por minuto, se escribe la letra T seguida de un número comprendido entre 60 y 240. El valor estándar es 120, que equivale a dos negras por segundo. Modifique el programa anterior de esta forma:

```
5 LET t$="T120"  
10 LET a$=t$+"O4cCcCgGgG"  
20 LET b$="O6CaCe$bd$bD"  
30 PLAY a$,b$
```

y ejecútelo varias veces poniendo en la línea 5 tempos diferentes.

Un fenómeno muy frecuente en la música es la repetición de un grupo de notas. Cualquier parte de una cadena puede ser repetida escribiéndola entre paréntesis. Así, si cambiamos la línea 10 por:

```
10 LET a$=t$+"O4(cC)(gG)"
```

el efecto es el mismo que obteníamos antes. Si ponemos el paréntesis de cerrar sin un paréntesis de abrir anterior que le corresponda, se repite indefinidamente todo el tramo previo de la cadena, desde el principio hasta el paréntesis. Esto es aprovechable en los efectos rítmicos y líneas de bajo. Para demostrarlo, pruebe lo siguiente (tendrá que usar **BREAK** para detener el sonido):

```
PLAY "O4N2cdefgfed)"
```

y después

```
PLAY "O4N2cd(efgf)ed)"
```

Si usted prepara una línea de bajo que se repita indefinidamente y después la utiliza para acompañar una melodía, sería interesante que el acompañamiento terminase al mismo tiempo que la melodía. Afortunadamente, en PLAY hay un mecanismo que nos permite lograrlo: si PLAY se encuentra la letra H en cualquiera de las cadenas que está interpretando, detiene todos los sonidos inmediatamente. Ejecute el siguiente programa (de nuevo tendrá que pulsar **BREAK** para detenerlo):

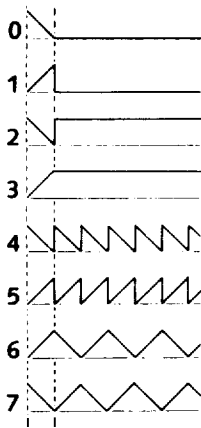
```
10 LET a$="cegbdfaC"  
20 LET b$="O4cC)"  
30 PLAY a$,b$
```

Ahora modifique la línea 10 de la forma siguiente:

```
10 LET a$="cegbdfaCH"
```

y ejecútelo de nuevo.

Hasta aquí sólo hemos usado notas que comienzan y terminan en el mismo nivel de volumen. El +2 puede alterar el volumen de una nota mientras está interpretándola, de forma que, por ejemplo, la nota puede ser intensa al principio y decaer lentamente (como las del piano). Para controlar este efecto se utiliza la letra W (de *waveform*, 'forma de onda') seguida de un número entre 0 y 7, junto con una U para cada canal al que se quiere aplicar el efecto. Si en un canal determinado se ha especificado volumen con V, no responderá a la U. El siguiente diagrama muestra la forma en que varía el volumen a lo largo del tiempo para los diversos valores del parámetro de V.



- 0 caída simple y fin
- 1 ataque simple y fin
- 2 caída simple y sostenimiento
- 3 ataque simple y sostenimiento
- 4 caída repetida
- 5 ataque repetido
- 6 ataque-caída repetidos
- 7 caída-ataque repetidos

El siguiente programa interpreta la misma nota con cada uno de estos valores. Trate de «escuchar» las formas ilustradas en el diagrama.

```
10 LET a$="UX1000W0C&W1C&W2C&W3C&W4C&W5C&W6C&W7C"
20 PLAY a$
```

La U activa los efectos y la W selecciona la forma de onda. La cláusula X1000 establece cuánto tiempo deben durar los efectos (su parámetro debe estar entre 0 y 65535). Si no se incluye la X, el +2 escogerá el valor más largo. Las formas que llegan a estabilizarse (1 a 3 en la tabla anterior) funcionan mejor con valores de X en torno a 1000, mientras que las periódicas (4 a 7) son más eficaces con valores pequeños, tales como 300. Pruebe diversos valores de X en el programa anterior para hacerse una idea de cómo funciona cada uno.

La orden PLAY no está limitada a las notas puramente musicales. Hay también tres generadores de 'ruido blanco' (ruido blanco es, más o menos, el ruido que produce la radio en FM o el televisor cuando no están sintonizados). Cualquiera de los tres canales puede interpretar notas, ruido blanco o una mezcla de ambas cosas. Para seleccionar esta mezcla se especifica la letra M seguida de un número (del 1 al 63). El significado del número es el que se indica en la siguiente tabla.

	Canales de tono			Canales de ruido		
	A	B	C	A	B	C
Número	1	2	4	8	16	32

Para obtener el número que se debe especificar tras M, se toma nota de los números que corresponden a los efectos que se desea activar y luego se los suma. Por ejemplo, si quiéramos ruido en el canal A, tono en el B y ambas cosas en el C, tendríamos que sumar

8, 2, 4 y 32 para obtener 46 (el orden de los canales es el orden en que ponemos las cadenas tras la orden PLAY). Los mejores efectos se obtienen en el canal A; no se prive de experimentar.

A estas alturas ya estará usted escribiendo sinfonías. Sin embargo, cuando las cosas se complican, puede ser difícil recordar qué tramo concreto de una cadena es responsable de cierta parte de la música. Para aliviar este problema, la cadena puede incluir comentarios escritos entre signos de admiración !. Por ejemplo,

```
1098 LET z$=z$+"CDcE3Ge4_6f! fin del compás 75 !egeA"
```

La orden PLAY sencillamente se salta los comentarios y los ignora.

Si usted dispone de un instrumento musical electrónico con MIDI, el +2 puede controlarlo con la orden PLAY. Hasta ocho canales de música pueden ser enviados a los sintetizadores, tambores y secuenciadores. La orden PLAY se construye exactamente como hemos explicado en esta sección, con la única diferencia de que cada cadena debe incluir una Y seguida de un número (entre 1 y 16). Este número controla a qué canal son asignados los datos de la música. Se puede usar hasta ocho cadenas; las tres primeras seguirán siendo interpretadas a través del televisor, como antes, por lo que puede ser conveniente bajar el volumen del aparato. También se puede enviar códigos de programación del MIDI a través de la orden PLAY, usando para ello una Z seguida del número de código. Las velocidades (intensidad sonora) se calculan y envían como 8 veces el valor de V (así, V6 enviará el número 48 como velocidad).

Para concluir esta sección, vamos a dar una lista de los parámetros que pueden ser incluidos en las cadenas de PLAY, junto con los valores que pueden tomar:

Cadena	Función
a-g } A-G }	Especifican el tono de la nota, dentro de la octava actual
§	Especifica que la nota siguiente debe ser convertida en bemol
#	Especifica que la nota que sigue debe ser convertida en sostenido
On	Especifica el número de octava (<i>n</i> entre 0 y 8)
1-12	Especifica la duración de las notas
&	Especifica un silencio
_	Especifica una ligadura
N	Separador entre números
Vn	Especifica el volumen de las notas (<i>n</i> entre 0 y 15)
Wn	Especifica el efecto de variación de volumen (<i>n</i> entre 0 y 7)
U	Introduce el efecto de variación de volumen en una cadena
Xn	Especifica la duración del efecto de variación de volumen (<i>n</i> entre 0 y 65535)

Cadena	Función
<i>Tn</i>	Especifica el tempo de la música (<i>n</i> entre 60 y 240)
()	Los paréntesis especifican que la frase escrita entre ellos debe ser repetida
!!	Los signos de admiración especifican que el comentario escrito entre ellos debe ser ignorado
H	Interrumpe la generación de sonido
<i>Mn</i>	Especifica qué canal o canales, musicales o de ruido, debe usar PLAY (<i>n</i> entre 1 y 63)
<i>Yn</i>	Especifica qué canal MIDI se debe usar (<i>n</i> entre 1 y 16)
<i>Zn</i>	Especifica un código de programación del MIDI (<i>n</i> es el número del código)

Parte 20

Operaciones con el magnetófono

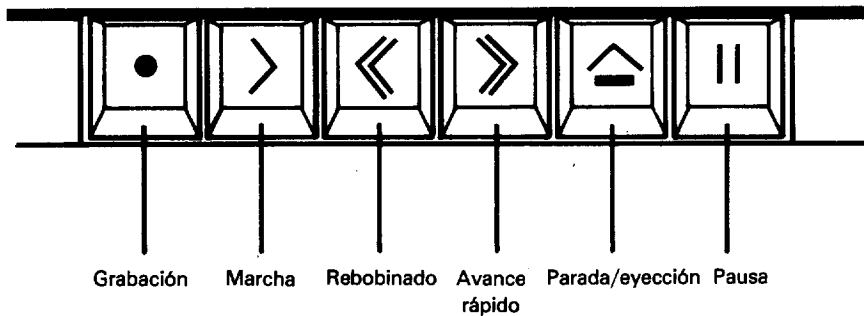
Temas tratados:

LOAD, SAVE, VERIFY, MERGE

El procedimiento básico de uso del magnetófono para cargar programas está explicado en los capítulos 3 y 4.

También podemos usar el magnetófono para almacenar (grabar) programas en una cinta, de forma que podamos volver a cargarlos en el ordenador siempre que deseemos usarlos (de no ser por esto, tendríamos que volver a escribir el programa cada vez que lo necesitásemos).

Antes de nada, familiarícese con las seis teclas de control del magnetófono:



Para estudiar cómo se graba un programa en el magnetófono, empiece por transcribir el programa que vimos al final de la Parte 16 (el que llenaba la pantalla con cuadrados de colores):

```
10 POKE 22527+RND*704, RND*127
20 GO TO 10
```

Éste es el programa que vamos a grabar. Cualquier cinta de cassette estándar servirá, aunque son preferibles las de bajo nivel de ruido.

Escriba lo siguiente:

```
SAVE "cuadrados"
```

"cuadrados" es justamente el nombre que utilizamos para 'etiquetar' el programa en el momento de almacenarlo en la cinta. Los nombres de los programas pueden constar de hasta 10 caracteres.

El ordenador mostrará en la pantalla el siguiente mensaje:

Pulse REC y PLAY, y luego tecla

Primero llevaremos a cabo una grabación simulada para que usted pueda ver lo que ocurrirá cuando más tarde grabemos el programa realmente. Esta vez, pues, no pulse los botones de grabación y marcha, sino sólo una tecla del ordenador (por ejemplo, **INTRO**), y observe el borde de la pantalla. Verá las siguientes pautas de rayas de colores:

Cinco segundos de rayas de color rojo y cyan desplazándose lentamente hacia arriba, seguidas por una cortísima irrupción de rayas azules y amarillas.

Una pausa corta.

Dos segundos de rayas rojas y cyan de nuevo, seguidas por una corta irrupción de rayas azules y amarillas.

Al tiempo que aparecen las rayas en la pantalla, también se puede oír el «sonido» de los datos a través del altavoz del televisor.

Siga probando la orden SAVE anterior (sin poner en marcha el magnetófono) hasta que pueda reconocer esas pautas. Lo que está sucediendo en realidad es que la información está siendo grabada en dos *bloques*, y ambos tienen una «cabecera» (que corresponde a las rayas rojas y cyan) a la que sigue la información propiamente dicha (que corresponde a las rayas azules y amarillas). El primero es un bloque preliminar que contiene el nombre y alguna otra información sobre el programa; el segundo es el programa con sus variables. La pausa entre ellos es sólo un hueco, sin ningún otro significado.

Ahora grabemos realmente el programa en la cinta:

1. Haga avanzar o retroceder la cinta hasta una zona que esté libre o en la que usted haya decidido grabar.
2. Escriba:

SAVE "cuadrados"

3. Obedezca el mensaje Pulse REC y PLAY, y luego tecla.
4. Observe que la pantalla se comporta como vimos antes. Cuando el +2 haya terminado de grabar (informe 0 OK), pulse el botón de parada del magnetófono.

Después de grabar *con éxito* un programa, ya se puede apagar tranquilamente el ordenador, o reinicializarlo, o bien comenzar un programa nuevo (NEW), sabiendo que podrá volver a cargar el programa siempre que lo necesite. No obstante, antes de borrarlo de

la memoria del ordenador, debe comprobar que el proceso de grabación ha funcionado correctamente; puede comparar la señal que ha quedado grabada en la cinta con el programa que todavía está en la memoria usando la orden VERIFY:

1. Rebobine la cinta hasta poco antes del punto en el que inició la grabación del programa.
2. Escriba:

VERIFY "cuadrados"

El borde alternará entre los colores rojo y cyan hasta que el +2 encuentre el programa especificado; después mostrará la misma pauta que apareció cuando se grabó el programa. Durante la pausa entre los bloques, aparecerá el mensaje Program: cuadrados. (Cuando el +2 está buscando algo en la cinta, muestra en la pantalla el nombre de todo lo que encuentra.) Si, después de haber aparecido la pauta, el ordenador emite el mensaje 0 OK, eso quiere decir que el programa está bien grabado, y entonces usted puede saltarse los cinco párrafos siguientes. De lo contrario, ha habido algún problema; siga este procedimiento para averiguar en qué consiste:

Si el nombre del programa no ha aparecido en la pantalla, puede ser que el programa no quedara grabado o que, si quedó bien grabado, el ordenador no haya podido leerlo. Tenemos que averiguar cuál de estas dos cosas ha ocurrido. Para ver si la grabación es correcta, rebobine la cinta hasta antes de donde empezó la grabación del programa y vuelva a pasarla mientras escucha el altavoz del televisor. La cabecera (rojo y cyan) debe producir una nota clara y persistente, de tono alto; la parte de la información (azul y amarillo) producirá un chirrido menos agradable.

Si no oye estos ruidos, es probable que el programa no quedara grabado. Asegúrese de que no está tratando de grabar el programa en la guía de plástico del principio de la cinta. Cuando lo haya hecho, intente otra vez grabar el programa.

Si puede oír los sonidos descritos, entonces la grabación (SAVE) es correcta y el problema está en la lectura.

Quizá haya escrito mal el nombre del programa al grabarlo; en tal caso, cuando el +2 encuentra el programa muestra en pantalla el nombre incorrecto. También puede haber escrito mal el nombre tras la orden VERIFY, en cuyo caso el ordenador ignorará el programa correctamente grabado y continuará buscando el nombre erróneo, provocando destellos de rojo y cyan según avanza.

Si realmente hay un error en la cinta, el +2 mostrará el mensaje R Tape loading error ('Error de carga de la cinta'), lo cual significa en este caso que ha fracasado la operación de verificación. Un pequeño defecto de la cinta misma (que podría ser casi inaudible en una grabación musical) puede causar estragos en un programa de ordenador. Trate de grabar el programa de nuevo, tal vez en una parte diferente de la cinta.

Ahora supongamos que usted ha grabado el programa y lo ha verificado con éxito. La carga del programa en la memoria es similar a la verificación, con la diferencia de que ahora la orden es:

LOAD "cuadrados"

(en lugar de VERIFY "cuadrados").

Puesto que la verificación tuvo éxito, la carga no debería dar ningún problema.

LOAD borra el programa (y las variables) que pudiera haber en la memoria cuando carga el nuevo programa desde la cinta.

Una vez cargado el programa, se puede dar la orden RUN para ejecutarlo.

Como hemos dicho en los capítulos 3 y 4, usted puede comprar programas comerciales pregrabados en cinta. Tales programas deben haber sido escritos especialmente para los ordenadores ZX Spectrum (es decir, para el Spectrum, el Spectrum +, el Spectrum 128 o el Spectrum +2). Cada marca y modelo de ordenador tiene una forma diferente de almacenar programas, así que con este ordenador no es posible leer las cintas grabadas por o para otros.

Si en la misma cara de la cinta hay varios programas, cada uno tendrá un nombre. En la orden LOAD se puede especificar qué programa se desea cargar; por ejemplo, si el que queremos cargar se llama 'helicoptero', podemos escribir:

LOAD "helicoptero"

La orden LOAD "" significa que se debe cargar el primer programa que el ordenador encuentre en la cinta. Esto puede ser muy útil si usted no recuerda el nombre con el que grabó el programa.

La opción Carg. cinta del menú de presentación tiene el mismo efecto que LOAD "", y es mucho más fácil y rápida de usar (basta con encender el ordenador y pulsar **INTRO**).

Como ya hemos dicho, LOAD borra del ordenador el programa y las variables antiguos siempre que se carga los nuevos desde la cinta. No obstante, hay otra orden, MERGE, que es similar a LOAD, pero que sólo borra una línea o una variable antigua si hay una nueva con el mismo número o nombre. Escriba el programa 'datos' de la Parte 11 de este capítulo y grábelo en la cinta con el nombre de datos. Después transcriba y ejecute el siguiente programa:

```
1 PRINT 1
2 PRINT 2
10 PRINT 10
20 LET x=20
```

Rebobine la cinta hasta poco antes de donde comenzó la grabación del programa datos y luego dé la orden

MERGE "datos"

Siga el mismo procedimiento que si estuviera cargando (LOAD) el programa. Si ahora lista (LIST) el programa, verá que las líneas 1 y 2 han sobrevivido y que las líneas 10 y 20 han sido sustituidas por las del programa dados. La variable x también se conserva (compruébelo con PRINT x).

Ya hemos visto las formas más sencillas de las cuatro órdenes que trabajan en colaboración con el magnetófono:

- SAVE • Almacena en la cinta el programa y las variables.
- VERIFY • Compara el programa y las variables leídos en la cinta con los de la memoria del ordenador.
- LOAD • Borra el programa y todas las variables del ordenador y los sustituye por los nuevos que ha leído en la cinta.
- MERGE • Similar a LOAD, pero **no** borra las líneas ni las variables del programa antiguo a menos que tenga que hacerlo (por tener el mismo número o nombre en los dos programas).

En todas estas órdenes la palabra clave va seguida por una cadena. Para la orden SAVE, la cadena consiste en el nombre con el cual se almacena el programa en la cinta, mientras que para las otras tres órdenes la cadena indica al ordenador qué programa buscar. Cuando el ordenador está buscando, muestra en pantalla el nombre de los programas que encuentra en la cinta. Sin embargo, hay un par de peculiaridades en todo esto:

VERIFY, LOAD y MERGE pueden ir seguidas de una cadena vacía "" como nombre de programa; entonces el ordenador no se preocupa por el nombre, sino que toma el primer programa que encuentra.

Hay una variante de SAVE que tiene la forma:

SAVE *cadena* LINE *número*

Un programa grabado con esta orden se almacena de forma tal que, cuando es leído por LOAD (pero no por MERGE), salta automáticamente a la línea especificada por el número dado, ejecutándose por tanto a sí mismo.

Si usted carga un programa que no es de ejecución automática (usando la opción Carga cinta del menú de presentación), después de cargarlo tendrá que seleccionar la opción 128 BASIC para poder ejecutarlo o editarlo.

Hasta aquí, los únicos tipos de información que hemos almacenado en cinta han sido programas junto con sus variables. Hay otros dos tipos más, llamados *matrices* y *bytes*.

Podemos grabar matrices en cinta incluyendo la cláusula DATA en la sentencia SAVE:

SAVE *cadena* DATA *nombre-de-matriz*()

donde *cadena* es el nombre con que la información quedará grabada en la cinta y funciona exactamente igual que cuando grabamos un programa (o simples bytes).

El *nombre-de-matriz* especifica la matriz que usted quiere grabar, así que tiene que ser una sola letra (o una letra seguida de \$). No olvide poner los paréntesis () tras el nombre de la matriz.

Distinga bien los papeles que desempeñan *cadena* y *nombre-de-matriz*. Si, por ejemplo, hacemos:

```
SAVE "Vargas" DATA b()
```

entonces SAVE toma la matriz b del ordenador y la almacena en la cinta con el nombre de Vargas.

Cuando luego demos la orden:

```
VERIFY "Vargas" DATA b()
```

el ordenador buscará una matriz numérica almacenada en la cinta con el nombre de Vargas. Cuando la encuentre, escribirá en la pantalla el mensaje Number array: Vargas ('Matriz numérica') y la comparará con la matriz b que está en la memoria del ordenador.

La orden:

```
LOAD "Vargas" DATA b()
```

encuentra la matriz en la cinta y después (si hay espacio para ella en el ordenador) borra la matriz que pueda existir con el nombre de b y carga la nueva desde la cinta, llamándola b.

No se puede usar MERGE para cargar matrices.

También podemos grabar matrices de caracteres (literales), exactamente de la misma forma que las numéricas. Cuando el ordenador está buscando en la cinta y encuentra una matriz literal, escribe en la pantalla Character array: ('Matriz de caracteres') y a continuación el nombre. Cuando carga una matriz literal, el ordenador borra no sólo la matriz literal, sino también la variable literal que se encuentre en la memoria con el mismo nombre.

El almacenamiento de bytes se aplica a bloques de información cualesquiera, con independencia de la naturaleza de esa información (podría ser una imagen de la pantalla, o quizá algunos gráficos definidos por el usuario, etc.). Este tipo de grabación se realiza incluyendo la cláusula CODE en la sentencia SAVE; por ejemplo,

```
SAVE "imagen" CODE 16384,6912
```

La unidad de almacenamiento en memoria es el *byte* (un número entero comprendido entre 0 y 255); cada byte de la memoria está identificado por una *dirección* (que es un número comprendido entre 0 y 65535). El primer número que se pone después de CODE es la dirección del primer byte que debe ser grabado en la cinta; el segundo número es la cantidad de bytes que deben ser grabados. En nuestro caso, 16384 es la dirección del primer byte del fichero (que contiene la imagen de pantalla); 6912 es la cantidad de bytes que hay en él. Pruebe la anterior orden SAVE. (No tiene por qué grabar los bytes precisamente con el nombre imagen; nosotros hemos elegido este nombre para que nos recuerde cuál es la naturaleza de la información grabada.)

Para cargar esta información se da la orden

```
LOAD "imagen" CODE
```

También podemos poner números detrás de CODE:

```
LOAD nombre CODE comienzo,longitud
```

Aquí *longitud* se utiliza como medida de seguridad. Cuando el ordenador ha encontrado en la cinta los bytes con el *nombre* solicitado, comprueba la *longitud* y, en caso de que haya más, sólo carga los bytes especificados; de esta forma se previene el riesgo de sobreescritura accidental de una zona de la memoria que interese preservar. Además, en tal caso, el ordenador emite el mensaje R Tape loading error.

Si omitimos la *longitud*, el ordenador lee los bytes que encuentra, sin importarle cuántos son.

El parámetro *comienzo* especifica la dirección de memoria en la que debe ser cargado el primer byte, que puede ser diferente de la dirección desde la que fue grabado. No obstante, si ambas direcciones son iguales, se puede omitir el parámetro *comienzo* en la sentencia LOAD.

CODE 16384,6912 es un área de memoria (imagen de pantalla) tan útil, que BASIC dispone de una función especial, SCREEN\$, para representarla. Así, para grabar y cargar la pantalla se puede usar las órdenes:

```
SAVE "imagen" SCREEN$
```

y

```
LOAD "imagen" SCREEN$
```

Éste es uno de los pocos casos en los que VERIFY no funciona. En efecto, puesto que VERIFY escribe los nombres de los ficheros que encuentra en la cinta, altera la imagen de la pantalla, y entonces ésta ya no puede ser igual a la grabada en la cinta.

Cualquier operación que podamos hacer en la cinta con SAVE, LOAD o MERGE puede ser realizada también con el *disco de silicio* (que está incorporado al +2 y que actúa como si fuese una cinta, con un par de órdenes adicionales). La diferencia entre la cinta

y el disco de silicio consiste en que éste tiene una capacidad de almacenamiento de unos 64K y un tiempo de acceso muy pequeño; la desventaja es que el contenido del disco de silicio se pierde cuando se apaga o reinicializa el ordenador (sin embargo, «sobrevive» a la orden NEW). Todas las órdenes se utilizan exactamente igual que con el magnetófono, pero intercalando un signo de admiración ! entre la palabra clave y la cadena asociada. Así, en lugar de grabar en la cinta con

SAVE "cuadrados"

podemos grabar en el disco de silicio con

SAVE ! "cuadrados"

Hay dos órdenes adicionales que pueden ser aplicadas al disco de silicio. La primera es

CAT !

que le da una lista de todos los programas o ficheros de datos almacenados en el disco.

La segunda orden es

ERASE ! "*nombre-de-fichero*"

que borra el programa o fichero de datos especificado.

Quizá la aplicación más obvia del disco de silicio sea el almacenamiento de porciones de un programa de BASIC para mezclarlas (con MERGE !) sucesivamente con un programa más pequeño. Esto hace posible tener en la memoria un programa de cerca de 90K (para hacer esto la estructura del programa tiene que estar bien definida).

Una de las aplicaciones más atractivas del disco de silicio es la *animación* de imágenes. Un programa de BASIC, de por sí relativamente lento, puede definir una serie de imágenes y almacenarlas en la memoria. Éstas pueden ser luego transferidas a la pantalla a gran velocidad. El siguiente programa da una idea de lo que se puede lograr con esta técnica (sin duda, usted puede hacer algo mejor):

```
10 INK 5: PAPER 0: BORDER 0: CLS
20 FOR f=1 TO 10
30 CIRCLE f*20,150,f
40 SAVE ! "balon"+STR$(f) CODE 16384,2048
50 CLS
60 NEXT f
70 FOR f=1 TO 10
80 LOAD ! "balon"+STR$(f) CODE
90 NEXT f
100 BEEP 0.01, 0.01
110 FOR f=9 TO 2 STEP -1
120 LOAD ! "balon"+STR$(f) CODE
```

```

130 NEXT f
140 BEEP 0.01, 0.01
150 GO TO 70
160 REM utilice GO TO 160 para borrar las imágenes del disco
170 FOR f=10 TO 1 STEP -1
180 ERASE ! "balon"+STR$(f)
190 NEXT f

```

En la línea 40, los dos números que siguen a CODE son la dirección de memoria donde empieza la pantalla y la longitud del tercio superior de ésta. Al grabar y cargar solamente ese trozo de la pantalla se mejora la velocidad global. Hemos incluido la rutina que va de la línea 160 a la 190 por si usted quiere detener el programa, modificar la porción que dibuja las circunferencias y grabar las nuevas imágenes. Así pues, para borrar las imágenes del disco de silicio, dé la orden GO TO 160. (Es conveniente borrar los ficheros «hacia atrás», de forma que el último fichero grabado será el primero en ser borrado; esto facilita el trabajo al ordenador y hace que el proceso sea mucho más rápido.)

Para concluir esta sección, vamos a dar un resumen de las cuatro instrucciones de control del magnetófono:

Notas

- El parámetro *nombre* representa una expresión literal y se refiere al nombre con el que la información se graba en la cinta. Debe consistir en caracteres ASCII, de los cuales el ordenador sólo utiliza los 10 primeros.
- Hay cuatro clases de información que pueden ser almacenadas en cinta o en el disco de silicio: programa y variables (juntos), matrices numéricas, matrices literales y bytes.
- Cuando VERIFY, LOAD y MERGE están buscando información en la cinta con un nombre dado y de una clase determinada, el ordenador muestra en la pantalla la *clase* y el *nombre* de toda la información que encuentra. La clase la indica mediante Program: (programa), Number array: (matriz numérica), Character array: (matriz literal) o Bytes: (bytes). Si el *nombre* es la cadena vacía (""), el ordenador toma el primer bloque de información (de la clase correcta) sin tener en cuenta el nombre.
- En lo que sigue, los corchetes [] indican que ! es opcional. Cuando se incluye esta cláusula, la orden se refiere al disco de silicio.

SAVE

1. Programa y variables:

SAVE [!] *nombre* LINE *numero-de-línea*

graba el programa y las variables de tal forma que LOAD implica automáticamente un GO TO *numero-de-línea*.

2. Bytes:

SAVE [!] *nombre* CODE *comienzo*,*longitud*

graba el número de bytes especificado por *longitud*, empezando por la dirección *comienzo*.

Observe que

SAVE [!] *nombre* SCREEN\$

es equivalente a

SAVE [!] *nombre* CODE 16384,6912

y graba la imagen de la pantalla.

3. Matrices:

SAVE [!] *nombre* DATA *letra*()

o

SAVE [!] *nombre* DATA *letra*\$()

graba la matriz numérica cuyo nombre es *letra*, o la matriz literal cuyo nombre es *letra*\$.

VERIFY

1. Programa y variables:

VERIFY *nombre*

lee el programa y las variables grabados en la cinta con el *nombre* especificado y los compara con los que están en la memoria.

2. Bytes:

VERIFY *nombre* CODE *comienzo*,*longitud*

Si los bytes grabados como *nombre* no son más de *longitud*, entonces compara los bytes de la cinta con los de la memoria, empezando por la dirección *comienzo*.

VERIFY *nombre* CODE

compara los bytes grabados en la cinta como *nombre* con los de la memoria, empezando por la dirección *comienzo*.

3. Matrices:

VERIFY *nombre* DATA *letra*()

o

VERIFY *nombre* DATA *letra*\$()

compara la matriz numérica cuyo nombre es *letra*, o la matriz literal cuyo nombre es *letra*\$, con la matriz *letra* o *letra*\$ de la memoria.

LOAD

1. Programa y variables:

LOAD [!] *nombre*

borra el programa y las variables antiguos y carga desde la cinta el programa y las variables grabados con el *nombre* especificado. Si el programa había sido grabado con SAVE *nombre* LINE *número-de-línea*, entonces LOAD ejecuta un GO TO *número-de-línea* automático en cuanto termina de cargar el programa.

Si la carga no tiene éxito, ni el programa ni las variables antiguos son borrados.

2. Bytes:

LOAD [!] *nombre* CODE *comienzo*,*longitud*

Si los bytes grabados como *nombre* no son más de *longitud*, los carga en la memoria desde la cinta, empezando en la dirección *comienzo* y escribiendo sobre lo que hubiese previamente en esa zona de la memoria.

LOAD [!] *nombre* CODE *comienzo*

carga incondicionalmente en la memoria desde la cinta los bytes grabados con el *nombre* especificado, empezando en la dirección *comienzo* y escribiendo sobre lo que hubiese previamente en esa zona de la memoria.

LOAD [!] *nombre* CODE

carga en la memoria desde la cinta los bytes grabados con el *nombre* especificado, empezando en la dirección a partir de la cual fue grabado el primer byte, y escribiendo sobre los bytes que estaban antes en esa zona de la memoria.

3. Matrices:

LOAD [!] *nombre* DATA *letra*()

o

LOAD [!] *nombre* DATA *letra*\$()

borra la matriz numérica que pudiera haber en la memoria con el nombre de *letra*, o la matriz literal llamada *letra*\$, y crea una nueva, con ese nombre, a base de los datos leídos en la cinta.

MERGE

1. Programa y variables:

MERGE [!] *nombre*

mezcla el programa grabado con el *nombre* especificado con el que ya está en la memoria, sobrescribiendo sólo las líneas y las variables del programa antiguo cuyos números o nombres existan también en el nuevo.

2. Bytes:

No es posible.

3. Matrices:

No es posible.

Ejercicio

1. Practique la grabación, carga y mezcla de programas y de ficheros de datos, tanto en cinta como en disco de silicio.

Parte 21

Operaciones de la impresora

Temas tratados:

LPRINT, LLIST, COPY

El +2 tiene incorporada una puerta serie y el software necesario para controlar una impresora. Estos recursos sólo pueden ser utilizados en modo 128 BASIC.

La impresora debe tener un interfaz de tipo serie RS232; además, para que pueda reproducir las imágenes de pantalla, debe tener *modo gráfico de imagen de bits de cuádruple densidad compatible Epson*.

Asegúrese de que tiene el cable correcto para conectar la impresora con el +2; en caso de duda, consulte a su distribuidor.

Para lograr que el +2 y la impresora se comuniquen el uno con la otra, ambos deben tener la misma *velocidad de transmisión*, que es, como su nombre indica, la velocidad a la que se transfieren los datos del ordenador a la impresora. Aunque la velocidad de transmisión pueda ser seleccionada en la impresora, probablemente será más fácil cambiarla, si es necesario, en el ordenador. En algún lugar del manual de la impresora estará especificada la velocidad de transmisión; averigüe cuál es y selecciónela en +2 con la orden:

FORMAT "p"; *velocidad de transmisión*

(No será necesario hacer esto si la impresora funciona normalmente a 9600 baudios, pues ésta es la velocidad que el +2 supone por defecto.)

Una vez preparada la conexión, ya podemos empezar a usar las tres órdenes de BASIC que controlan la impresora. Las dos primeras, LPRINT y LLIST, son análogas a PRINT y LIST, salvo que dirigen la salida a la impresora en lugar de al televisor. La opción Imprimir del menú de edición de 128 BASIC produce el mismo efecto que LLIST, pero la hemos incluido para proporcionar una forma más directa y fácil de obtener un listado.

Pruebe este programa:

```
10 PRINT "Este programa ""
20 LLIST
30 LPRINT ""imprime el juego de caracteres: ""
40 FOR n=32 TO 255
50 LPRINT CHR$ n;
60 NEXT n
```

Es importante observar que LPRINT y LLIST tienen buen cuidado de filtrar todos los códigos de color (y sus parámetros) que encuentran antes de imprimir o listar cualquier cosa. Tales códigos de color son una «reliquia» del Spectrum de 48K; cuando se los incluía en una cadena ajustaban INK (tinta), PAPER (papel), etc. En general las impresoras utilizan esos códigos para cosas completamente diferentes, como activar o desactivar cursiva, subrayado, etc., así que sería muy peligroso enviarles estos códigos de color y esperar que no ocurriese ningún incidente. Como un efecto secundario de esto, es imposible (desde BASIC) controlar cualquier función especial en una impresora que utilice *secuencias de escape* (carácter 27) o códigos de control similares.

La tercera instrucción, COPY, imprime una copia de la pantalla del televisor. Para ver una primera demostración, entre en la pantalla pequeña, dé la orden LIST para tener en la pantalla algo que imprimir y después escriba:

COPY

La orden COPY tarda de 15 a 30 segundos en prepararse para enviar datos a la impresora, así que no se inquiete si no parece que ocurra nada inmediatamente. Finalmente obtendrá otro listado del programa en la impresora, pero esta vez se parecerá bastante a lo que había en la pantalla. Si, en cambio, todo lo que produce COPY en la impresora es un montón de caracteres aleatorios, es probable que la impresora no sea del todo compatible.

En cualquier momento se puede detener la impresión pulsando la tecla **BREAK**. Algunas impresoras tienen lo que se conoce por el nombre de *tampón*, una memoria en la que almacenan el texto antes de imprimirlo. Si su impresora tiene tampón, al pulsar **BREAK** no se detendrá inmediatamente, a pesar de que el +2 dejará de enviarle datos en ese mismo momento.

Si usted intenta ejecutar una orden LLIST, LPRINT o COPY cuando la impresora no está conectada, el +2 se pondrá a esperar pacientemente que la impresora (inexistente) le diga que está preparada. Pulsando **BREAK** volverá a despertar al ordenador.

Pruebe este programa:

```
10 FOR n=31 TO 0 STEP -1
20 PRINT AT 31-n,n; CHR$(CODE "0"+n);
30 NEXT n
```

Verá una serie de caracteres bajando diagonalmente desde el extremo superior derecho hasta el borde inferior de la pantalla, y en ese momento el programa preguntará si quiere desplazar la pantalla.

Ahora cambie el AT 31-n,n de la línea 20 por TAB n. El programa producirá exactamente el mismo efecto que antes.

Ahora cambie PRINT en la línea 20 por LPRINT. Esta vez el ordenador no se para a preguntar scroll?, pues esto sólo ocurre cuando la salida está siendo dirigida a la pantalla.

Ahora vuelva a sustituir TAB n por AT 31-n,n manteniendo LPRINT. Esta vez obtendrá solamente una única línea de símbolos. La causa de esta diferencia es que el resultado de LPRINT no se imprime inmediatamente, sino que se almacena en un tampón hasta que se haya acumulado el equivalente a una línea de impresora, o bien hasta que alguna otra cosa provoque el vaciado del tampón. Así pues, la impresión sólo tiene lugar:

1. Cuando se llena el tampón.
2. Tras una sentencia LPRINT que no acabe en coma ni en punto y coma.
3. Cuando una coma, apóstrofo o cláusula TAB obligue a cambiar de línea.
4. Al final de un programa, si ha dejado algo sin imprimir.
5. En algunas impresoras, cuando se pulsa el botón de 'fuera de línea'.

El punto 3 explica por qué nuestro programa con TAB funciona así. En cuanto a AT, el número de fila es ignorado y la posición de LPRINT (igual que la de PRINT) se desplaza a la columna especificada. Un elemento AT no puede hacer nunca que la línea sea enviada a la impresora.

Ejercicio

1. Imprima un gráfico de la función seno: ejecute el primer programa de la Parte 17 de este capítulo y luego dé la orden COPY.

Parte 22

Otros periféricos

Temas tratados:

- Microunidades ZX
- Redes
- RS232
- Subteclado numérico

Hay muchos periféricos disponibles para conectarlos al +2. En el capítulo 10 le daremos información más amplia sobre su conexión y funcionamiento.

La microunidad ZX es un dispositivo flexible de almacenamiento masivo de datos a alta velocidad. Funcionará no sólo con SAVE, VERIFY, LOAD y MERGE, sino también con PRINT, LIST, INPUT e INKEY\$.

Una *red* es un sistema para interconectar varios ordenadores de forma que puedan no sólo comunicarse, sino también utilizar recursos comunes. Una de las aplicaciones sería, por ejemplo, tener una sola microunidad a disposición de varios ordenadores.

El interfaz RS232 es una conexión estándar que permite unir un ordenador con teclados, impresoras y otros dispositivos, aunque no hayan sido diseñados específicamente para el +2.

El subteclado numérico facilita el uso de las funciones de edición adicionales de 128 BASIC y también la introducción de datos numéricos.

Parte 23

IN y OUT

Temas tratados:

OUT
IN

Como el lector ya sabe, podemos leer la memoria ROM y RAM y escribir en la RAM utilizando la función PEEK y la orden POKE. En realidad, al microprocesador no le importa si la memoria es ROM o RAM: sólo sabe que hay 65536 direcciones de memoria y que puede leer un byte en cada una de ellas (aunque no tenga sentido) y escribir también un byte en cada una (aunque se pierda). Análogamente, hay 65536 *puertas E/S* (puertas de entrada/salida). Estas puertas las utiliza el microprocesador para comunicarse con, por ejemplo, el teclado y la impresora, y también para controlar la memoria adicional y el chip de sonido. Algunas de ellas pueden ser controladas desde BASIC sin problemas mediante la función IN y la orden OUT, pero hay posiciones en las que usted no debe escribir desde BASIC, ya que probablemente provocaría la *caída del sistema*, lo que representaría la pérdida del programa y todos los datos.

IN es una función, como PEEK. Su forma es:

IN *dirección*

Tiene un argumento (la dirección de la puerta) y su resultado es el byte leído en esa puerta.

OUT es una sentencia, como POKE. Su forma es:

OUT *dirección,valor*

que escribe el *valor* dado en la puerta cuya *dirección* es la especificada. La interpretación que se dé a la dirección depende en gran medida del resto del ordenador. Frecuentemente muchas direcciones diferentes significan lo mismo. En el +2 lo más sensato es imaginar la dirección escrita en binario, ya que los bits individuales (cada uno de los cuales puede tener el valor 0 o 1) funcionan independientemente. Una dirección está formada por 16 bits:

A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0

A0 es el bit de las unidades, A1 el bit de los 'doses', A2 el bit de los 'cuatros', y así sucesivamente. Los bits A0, A1, A2, A3 y A4 son los importantes. Normalmente tienen el valor 1, y el hecho de que uno de ellos esté a 0 le encarga al ordenador algo específico.

El ordenador no puede hacer varias cosas al mismo tiempo, así que no debe estar a 0 más que uno de estos bits. Los bits A6 y A7 son ignorados, de modo que, si es usted un mago de la electrónica, puede usarlos como quiera. Las mejores direcciones son los múltiplos de 32 menos 1, de tal manera que los bits A0 a A4 estén todos a 1. Los bits A8, A9 y siguientes se usan a veces para dar información adicional; se los utiliza casi siempre para controlar la memoria adicional y el sonido.

El byte leído o escrito (byte de datos) está formado por ocho bits:

D7, D6, D5, D4, D3, D2, D1, D0

Veamos una lista de las direcciones de puerta utilizadas:

Las siguientes direcciones de entrada leen el teclado y el magnetófono. El teclado se compone de ocho medias filas de cinco teclas, a saber:

IN 65278 lee desde **MAYUSC** hasta V
IN 65022 lee desde A hasta G
IN 64510 lee desde Q hasta T
IN 63486 lee desde 1 hasta 5 (y JOYSTICK 2)
IN 61438 lee desde 0 hasta 6 (y JOYSTICK 1)
IN 57342 lee desde P hasta Y
IN 49150 lee desde **INTRO** hasta H
IN 32766 lee desde 'espacio' hasta B

(Estas direcciones son $254 + 256 * (255 - 2 \uparrow n)$, donde n va de 0 a 7.)

En el byte leído, los bits D0 a D4 representan las cinco teclas de la media fila en cuestión. D0 representa la tecla más externa; D4, la más próxima al centro. El bit es 0 si está pulsada la tecla, y 1 en caso contrario. D6 es controlado por el magnetófono; en ausencia de datos procedentes de la cinta, su valor es aleatorio.

Para JOYSTICK 1, el bit 0 representa disparo; el bit 1, arriba; el bit 2, abajo; el bit 3, derecha; y el bit 4, izquierda. Para JOYSTICK 2, el bit 0 representa izquierda; el bit 1, derecha; el bit 2, abajo; el bit 3, arriba; y el bit 4, disparo. En BASIC éstos se leen como las teclas de números.

La puerta de salida 254 controla el sonido (D4) y la señal de grabación hacia el magnetófono (D3) y establece el color del borde (D2, D1 y D0).

Las puertas 254, 247 y 239 se usan para controlar los dispositivos externos mencionados en la Parte 22.

La puerta 32765 gestiona la memoria adicional. La ejecución de OUT desde BASIC hacia esa puerta causará casi siempre la caída del sistema, con la consiguiente pérdida del programa y los datos. Esta puerta está descrita en la Parte 24 de este capítulo (bajo el título de 'Gestión de la memoria'). Esta puerta es de *sólo escritura*; no se puede determinar el estado actual de la paginación mediante una función IN.

La puerta 49149 controla los registros de datos del chip de sonido. La puerta 65533 en modo de salida escribe una dirección de registro, y en modo de entrada lee un registro. El uso cuidadoso de estos dos registros puede permitir la generación de sonidos mientras BASIC realiza otra tarea; pero hay que tener en cuenta que también controlan el RS232, el subteclado numérico y el MIDI.

Ejecute este programa para ver cómo funciona el teclado:

```
10 FOR n=0 TO 7: REM numero de media fila
20 LET a=254+256*(255-2↑n)
30 PRINT AT 0,0; IN a: GO TO 30
```

y entreténgase pulsando teclas. Cuando acabe con cada media fila, pulse **BREAK** y escriba:

```
NEXT n
```

Los *buses* de control, datos y direcciones están todos a su disposición en la parte trasera del +2, en el zócalo **EXPANSION E/S**, de modo que usted puede hacer con el +2 casi todo lo que hacía con un Z80 (aunque algunas veces el hardware del ordenador puede impedirle algo).

En el capítulo 10 damos un diagrama y una descripción de las patillas del zócalo **EXPANSION E/S**.

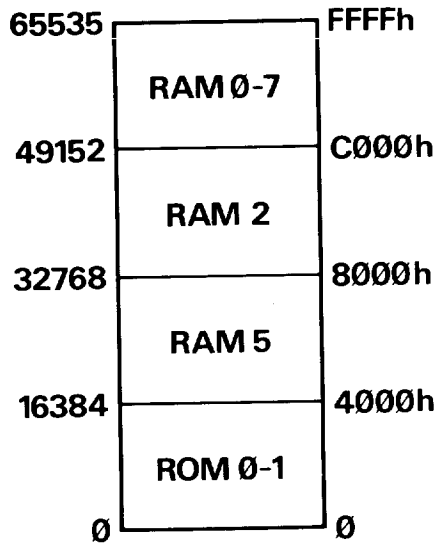
Parte 24

La memoria

Temas tratados:

CLEAR

En el interior del +2 todo se almacena en forma de *bytes*, es decir, números enteros del margen de 0 a 255. Usted puede pensar que ha almacenado el precio de las verduras o la dirección de su amigo Juan, pero, de hecho, toda la información ha sido convertida en colecciones de bytes, y son los bytes lo que el ordenador ve y entiende.



Mapa de memoria del +2

Cada lugar donde puede ser almacenado un byte está caracterizado por una dirección, que es un número entre 0 y FFFFh (una h al final de los dígitos significa que el número es hexadecimal). Esto significa que una dirección puede ser almacenada en forma de dos bytes. Podemos imaginar la memoria como una larga fila de cajas numeradas, cada una de las cuales puede contener un byte. Sin embargo, no todas las cajas son iguales; del 4000h al FFFFh son cajas de RAM, lo que significa que podemos levantar la tapa y alterar el contenido. Pero del 0 al 3FFFh son cajas de ROM, las cuales tienen una tapa de cristal que no podemos abrir; lo único que podemos hacer es ver a través del

cristal lo que se puso en ella cuando se fabricó el ordenador. En el +2 hemos puesto más del doble de cajas que las que caben cómodamente: mientras que el procesador puede acceder directamente a 65536 bytes, el +2 tiene 131072 bytes de RAM y 32768 bytes de ROM, lo que da un total de 163840 bytes (160K). El hardware oculta este exceso de memoria al microprocesador mediante un sistema llamado *paginación*; BASIC y el microprocesador siempre «ven» la memoria como 16K de ROM y 48K de RAM.

Para inspeccionar el contenido de una caja usamos la función PEEK. Su argumento es la dirección de la caja y su resultado es el contenido. Por ejemplo, el siguiente programa escribe los primeros 21 bytes de la ROM junto con sus direcciones:

```
10 PRINT "Dirección"; TAB 10; "Byte"  
20 FOR a=0 TO 20  
30 PRINT a; TAB 10; PEEK a  
40 NEXT a
```

Estos bytes seguramente no tendrán ningún significado para usted, pero el microprocesador los entiende como instrucciones que le dicen qué tiene que hacer.

Para cambiar el contenido de una caja (suponiendo que sea de RAM), usamos la orden POKE. Su forma es:

POKE dirección, contenido

donde *dirección* y *contenido* son expresiones numéricas. Por ejemplo, la orden

```
POKE 31000,57
```

da al byte de la dirección 31000 el nuevo valor 57. Escriba:

```
PRINT PEEK 31000
```

para comprobarlo. (Pruebe con otros valores para ver que no estamos haciendo trampa.) El nuevo valor debe estar entre -255 y +255; si es negativo, BASIC le suma 256.

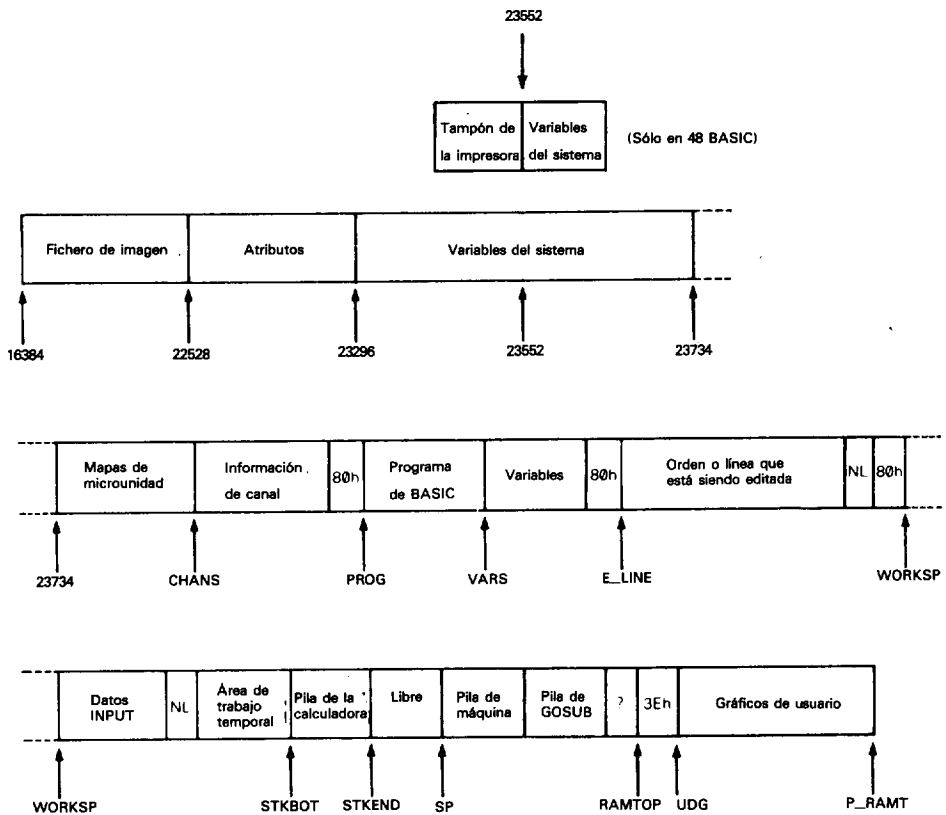
La facultad de modificar el contenido de una posición de memoria nos otorga un inmenso poder sobre el ordenador si sabemos cómo usarla, e inmensas posibilidades de destrucción si no sabemos. Es muy fácil, colocando un valor incorrecto en ciertas direcciones de memoria, perder largos programas que hemos tardado horas en transcribir. Sin embargo, afortunadamente, esto no le hará al ordenador ningún daño permanente.

Ahora veremos más detalladamente cómo se usa la RAM. No se moleste en leer esto si no está realmente interesado.

La memoria está dividida en diferentes áreas (mostradas en el diagrama siguiente), en cada una de las cuales se almacena un tipo de información diferente. El tamaño de cada área es el justo para la información que contiene en el momento; si introducimos algo más en cierta posición (por ejemplo, añadiendo una línea de programa o una variable),

el ordenador le hace hueco desplazando hacia arriba todo lo que haya por encima de esa posición. A la inversa, si borramos información, todo el resto de la memoria se des-
plaza hacia abajo.

El fichero de imagen (memoria de pantalla) almacena los datos necesarios para construir la imagen de la pantalla. Está organizado de una forma más bien curiosa, así que no tiene ningún interés leerlo (PEEK) ni modificarlo (POKE). Cada posición de carácter en la pantalla consiste en una retícula de 8x8 puntos; cada punto puede ser 0 (papel) o 1 (tinta), por lo que podemos almacenar su descripción en 8 bytes (uno para cada fila). Sin embargo, estos ocho bytes no se almacenan uno al lado del otro. Las columnas correspondientes en los 32 caracteres de una línea se almacenan juntas en un bloque de 32 bytes, ya que esto es lo que necesita el rayo de electrones del televisor al barrer la pantalla de izquierda a derecha. Puesto que la imagen completa consiste en 24 líneas de ocho barridos cada una, sería de esperar que los 192 bloques (24x8) de 32 bytes se almacenasen uno a continuación del otro; pues bien, no es así. Primero vienen los bloques superiores de las líneas 0 a 7, después los siguientes bloques de las líneas 0 a 7, y



Mapa de la memoria de BASIC

así hasta los bloques más bajos de esas mismas líneas; después se hace lo mismo con las líneas 8 a 15, y luego con las líneas 16 a 23. El resultado final de todo esto es que, si está usted acostumbrado a un ordenador en el que puede aplicar PEEK y POKE a la memoria de pantalla, más vale que empiece a habituarse a SCREEN\$ y PRINT AT (o PLOT y POINT).

Los *atributos* de cada posición de carácter son los colores y las restantes características; su formato es el que describimos a propósito de la función ATTR y *sí* son almacenados línea por línea en el orden que usted esperaría.

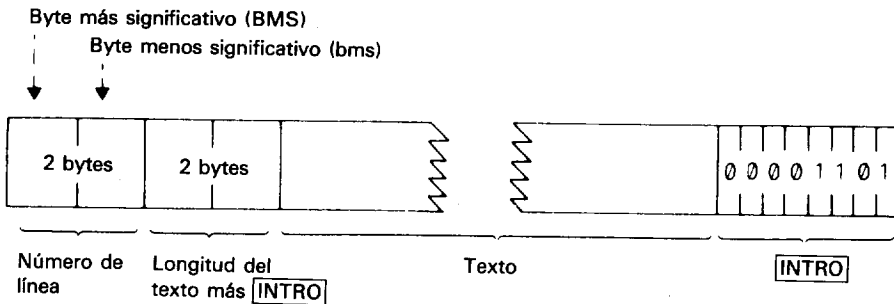
La forma en la que el ordenador organiza sus asuntos no es exactamente la misma en 48 BASIC y en 128 BASIC. El área que constituía el tampón de la impresora en 48 BASIC se utiliza para almacenar ciertas variables adicionales del sistema en 128 BASIC.

Las variables del sistema contienen diversos elementos de información que indican al ordenador en qué de estado se encuentra. La lista completa está en la Parte 26 de este capítulo; por el momento observe que algunas de ellas (CHANS, PROG, VARS, E_LINE, etc.) contienen la dirección de los límites entre las distintas áreas de la memoria. *No* son variables de BASIC y por lo tanto no serán reconocidas por el +2.

Los mapas de microunidad se usan sólo cuando se tiene una microunidad conectada. Normalmente están vacíos.

La información de canal contiene datos sobre los dispositivos de entrada y salida: el teclado (junto con la pantalla inferior), la pantalla superior y la impresora.

Cada línea de programa de BASIC tiene la siguiente forma:

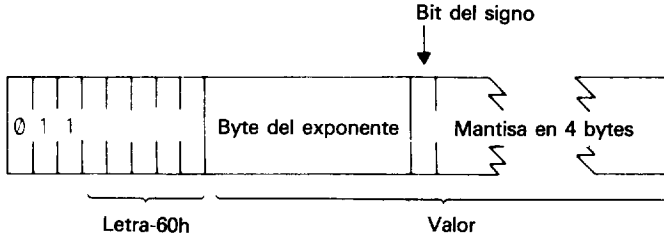


Observe que, en contraste con todos los demás casos de números de dos bytes en el Z80, aquí el número de línea se almacena con su byte más significativo primero, es decir, en el mismo orden en que nosotros los escribimos.

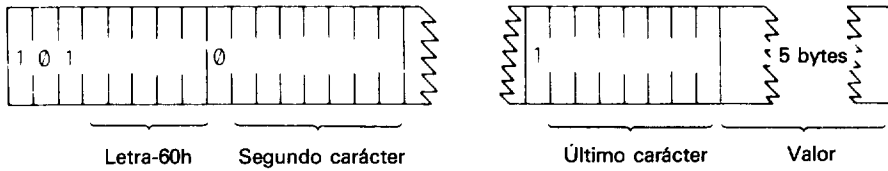
Una constante numérica del programa lleva a continuación su forma binaria, usando el carácter CHR\$ 14 seguido por cinco bytes para el número propiamente dicho.

Las variables tienen formatos diferentes para los distintos tipos. Las letras de los nombres se almacenan como si empezasen en minúscula.

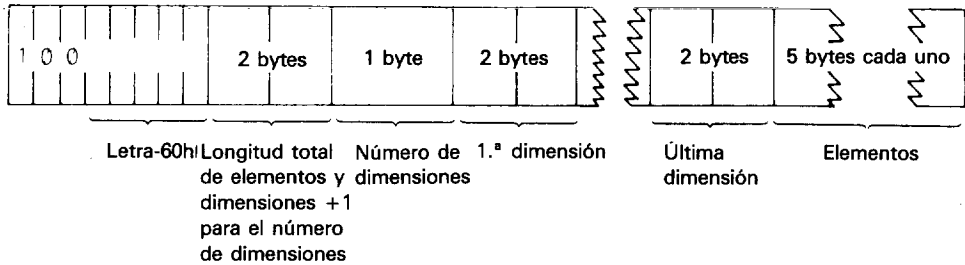
Variable numérica cuyo nombre es una sola letra:



Variable numérica cuyo nombre consiste en varias letras:



Matriz numérica:



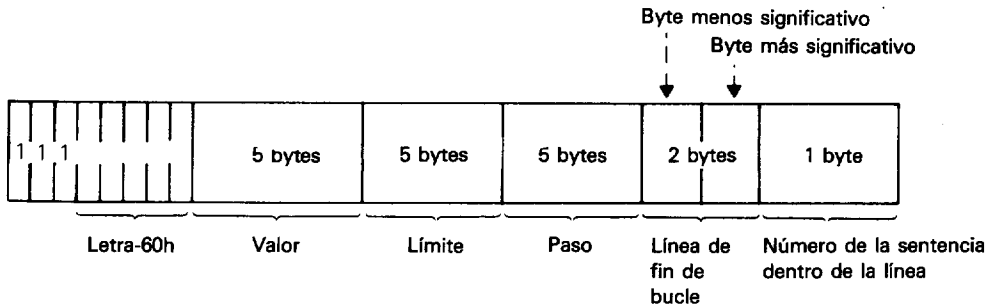
El orden de los elementos es:

- en primer lugar los elementos cuyo primer subíndice es 1
- a continuación los elementos cuyo primer subíndice es 2
- a continuación los elementos cuyo primer subíndice es 3
- ... y así sucesivamente para todos los posibles valores del primer subíndice.

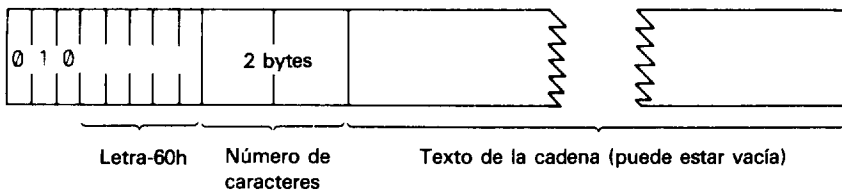
Los elementos con un primer subíndice dado se ordenan de la misma forma con respecto al segundo, y así sucesivamente hasta el último.

Por ejemplo, los elementos de la matriz *c* de 3*6 de la Parte 12 de este capítulo se almacena en el siguiente orden: *c*(1,1) *c*(1,2) *c*(1,3) *c*(1,4) *c*(1,5) *c*(1,6) *c*(2,1) *c*(2,2)...*c*(2,6) *c*(3,1) *c*(3,2)...*c*(3,6).

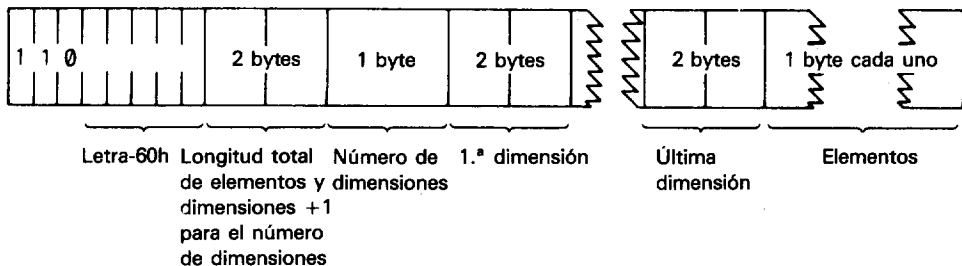
Variable de control para un bucle FOR...NEXT:



Variable literal:



Matriz literal:



La calculadora es la parte de BASIC que se encarga de la aritmética; los números con los que opera se guardan en su mayoría en la pila de la calculadora.

La parte libre que se sigue en el mapa de memoria es el espacio que no ha sido utilizado de momento.

La pila de máquina es la usada por el microprocesador Z80 para guardar direcciones de retorno, etc.

El funcionamiento de la pila de GO SUB está descrito en la Parte 5 de este capítulo.

El byte al que «apunta» RAMTOP tiene la dirección más alta utilizada por BASIC. Incluso NEW, que borra la RAM, no pasa de esa dirección, y por tanto no cambia los gráficos definidos por el usuario. Se puede modificar la dirección RAMTOP especificando su nuevo valor en una sentencia CLEAR:

CLEAR nueva RAMTOP

cuyo efecto es el siguiente:

1. Borra todas las variables.
2. Borra el fichero de imagen.
3. Restaura la posición del cursor gráfico en el extremo inferior izquierdo de la pantalla.
4. Restaura (RESTORE) el puntero de datos.
5. Borra la pila de GO SUB e iguala su base a RAMTOP (suponiendo que ésta se encuentre entre la pila de la calculadora y el final físico de la RAM, pues de lo contrario deja RAMTOP donde estaba).

RUN ejecuta implícitamente una sentencia CLEAR, pero no modifica RAMTOP. Utilizando CLEAR de esta forma se puede, o bien desplazar RAMTOP hacia arriba para dedicar más espacio a BASIC (destruyendo los gráficos de usuario), o bien desplazarla hacia abajo para hacer más grande la zona de RAM que no es borrada por NEW.

Escriba NEW y luego CLEAR 23825 para hacerse una idea de lo que le ocurre a la máquina cuando se queda sin memoria.

Si intenta que el +2 calcule algo (escriba, por ejemplo, PRINT 1 + 1), verá en la pantalla el mensaje 4 Out of memory ('Memoria agotada'). Esto significa que al ordenador no le queda más espacio donde almacenar información. Si se le presenta este mensaje cuando está transcribiendo un programa largo, tendrá que vaciar un poco la memoria (por ejemplo, borrando una línea) para poder controlar el ordenador.

Gestión de la memoria

Ya hemos mencionado antes que en el ordenador hay bastante más memoria que la que el microprocesador puede manejar cómodamente. Éste puede «direccionar» sólo 64K de memoria de una vez, pero la memoria adicional puede ser intercambiada a discreción

con esos 64K. Observe un aparato de televisión; a pesar de que sólo puede ocuparse de un canal a la vez (igual que usted), hay otros canales que pueden ser seleccionados pulsando los botones adecuados. Así, aunque hay más información de la que usted puede asimilar a un tiempo, usted puede elegir la que le interese.

Lo que ocurre en el ordenador es muy parecido. Colocando los bits adecuados en cierta puerta de E/S, la máquina puede elegir qué porciones quiere usar del total de 160K de memoria. BASIC ignora casi permanentemente toda la memoria adicional. En cambio, en los programas de juegos viene muy bien tener una RAM casi triplicada. Vuelva a echar un vistazo al mapa de memoria del principio de esta sección. Los bancos RAM 2 y RAM 5 están siempre en la posición indicada en el diagrama, y sin embargo no hay ninguna razón por la que no puedan estar en la sección conmutable (C000h a FFFFh), si bien esto tampoco tendría ninguna utilidad. Los bancos de RAM son de dos tipos: RAM 4 a RAM 7, que son compartidos (es decir, que comparten el tiempo con los circuitos de video), y RAM 0 a RAM 3, que sólo están al servicio del microprocesador. Cualquier programa de código de máquina que requiera un control muy preciso del tiempo (por ejemplo, los de música o comunicaciones) debe guardar todas las rutinas de temporización en los bancos no compartidos.

El conmutador de hardware está en la dirección de E/S 7FFDh (32765 decimal). El campo de bits para esta dirección es el siguiente:

D0 a D2	Selección de RAM
D3	Selección de pantalla
D4	Selección de ROM
D5	Bloqueo de 48K

D2 a D0 crean un número de tres bits que selecciona qué RAM entra en el hueco comprendido entre C000h y FFFFh. En BASIC normalmente está seleccionado el banco RAM 0; durante la edición de usa RAM 7 para almacenar varios tampones y memorias transitorias. D3 conmuta pantallas; la pantalla 0 está en la RAM 5 (que empieza en 4000h) y es la que utiliza BASIC; la pantalla 1 está en RAM 7 (a partir de C000h) y sólo puede ser usada por programas de código de máquina. Es perfectamente factible preparar una pantalla en RAM 7 y después «desconmutarla»; de esta forma se deja los 48K libres para datos y programa. D4 determina si debe ser colocada en las direcciones 0000h a 3FFFh la ROM 0 (la ROM del editor) o la ROM 1 (la ROM de BASIC). D5 es un dispositivo de seguridad; en cuanto se pone a 1 este bit, el ordenador adopta la configuración del Spectrum de 48K estándar y bloquea todos los circuitos de paginación de memoria. A partir de ese momento no se lo puede volver a convertir en un ordenador de 128K más que apagándolo o pulsando el botón **RESET**; no obstante, el chip de sonido puede seguir siendo controlado por OUT.

Parte 25

Variables de sistema

Temas tratados:

POKE, PEEK

Los bytes de memoria del 23296 al 23733 están reservados para usos específicos del sistema. Hay también unas cuantas rutinas (usadas para mantener en orden la paginación de memoria) y algunas posiciones que contienen *variables de sistema*. Éstas pueden ser examinadas (con PEEK) para conocer diversos aspectos del sistema; también puede tener interés modificar algunas de ellas. En esta sección daremos la lista completa.

Hay gran diferencia, como sería de esperar, entre el área de variables de sistema de 48 BASIC y de 128 BASIC. En el modo de 48 BASIC las rutinas y variables por debajo de 23552 no existen; en su lugar hay un tampón, entre 23296 y 23552, que se usa para controlar la impresora. Ésta era una posición muy popular para pequeñas rutinas de código de máquina en el Spectrum de 48K; si usted prueba cualquiera de estas rutinas en 128 BASIC, provocará la caída del sistema con toda seguridad. Todos los programas antiguos que usan PEEK, POKE y USR deben ser ejecutados, por tanto, en 48 BASIC (no obstante, se los puede introducir en 128 BASIC y luego pasar a 48 BASIC con la orden SPECTRUM).

Las variables de sistema tienen nombres, pero es preciso no confundirlos con las palabras clave y los nombres de variable utilizados en BASIC. El ordenador no reconocerá los nombres como referencias a variables de sistema; nosotros los utilizamos solamente como nemónicos y para el ordenador no significan nada.

Las abreviaturas de la columna 1 de la tabla significan lo siguiente:

X No se debe modificar estas variables porque el sistema podría fallar.

N La modificación de las variables no tendrá efectos duraderos.

R No es una variable, sino el punto de entrada a una rutina.

El número de la columna 1 es el número de bytes de que consta la variable o rutina. En el caso de dos bytes, el primero es el menos significativo, lo contrario de lo que parecería natural. Así, para escribir el valor v en una variable de dos bytes que se encuentra en la dirección n se debe dar las órdenes:

```
POKE n,v-256*INT (v/256)
POKE n+1,INT (v/256)
```

Para leer ese valor se utiliza la expresión:

```
PEEK n+256*PEEK (n+1)
```

Notas	Dirección	Nombre	Contenido
R20	23296	SWAP	Subrutina de paginación.
R9	23316	YOUNGER	Subrutina de paginación.
R18	23325	ONERR	Subrutina de paginación.
R5	23343	PIN	Prerrutina de entrada del RS232.
R22	23348	POUT	Prerrutina de salida de claves del RS232. Puede ser modificada para evitar el filtrado de códigos de control.
R14	23370	POUT2	Prerrutina de salida de caracteres del RS232.
N2	23384	TARGET	Dirección de subrutina en ROM 1.
X2	23386	RETADDR	Dirección de retorno en ROM 0.
X1	23388	BANKM	Copia del último byte enviado al banco.
X1	23389	RAMRST	Instrucción RST8.
N1	23390	RAMERR	Número de error, ROM 1.
2	23391	BAUD	Periodo de bit del RS232 en estados T/26.
N2	23393	SERFL	Indicador de recepción del segundo carácter, y datos.
N1	23395	COL	Columna actual, desde la 1 hasta la anchura.
1	23396	WIDTH	Anchura del papel en columnas.
1	23397	TVPARS	Número de los parámetros por línea esperados por el RS232.
1	23398	FLAGS3	Diversos indicadores.
N10	23399	N STR1	Nombre de fichero.
1	23409	HD 00	Código de tipo de fichero.
2	23410	HD 0B	Longitud del bloque.
2	23412	HD 0D	Comienzo del bloque.
2	23414	HD 0F	Longitud del programa.
2	23416	HD 11	Número de línea.
1	23418	SC 00	Código de tipo de fichero, segundo grupo.
2	23419	SC 08	Longitud del bloque, segundo grupo.
2	23421	SC 0D	Comienzo del bloque, segundo grupo.
2	23423	SC 0F	Longitud del programa, segundo grupo.
X2	23425	OLDSP	SP antiguo cuando se está usando TSTACK.
X2	23427	SFNEXT	Puntero hacia la primera reseña libre del directorio.
X3	23429	SFSPACE	Número de bytes disponibles (17 bits).
N1	23432	ROW01	Indicadores e imagen de la fila 1 del teclado numérico.
N1	23433	ROW23	Imágenes de las filas 2 y 3 del teclado numérico.
N1	23434	ROW45	Imágenes de las filas 4 y 5 del teclado numérico.
X2	23435	SYNRET	Dirección de retorno para ONERR.
5	23437	LASTV	Último valor escrito por la calculadora.
2	23442	RNLINE	Línea que está siendo renumerada.
2	23444	RNFIRST	Número de línea inicial para Renumerar.
2	23446	RNSTEP	Valor del incremento para Renumerar.
N8	23448	STRIP1	Mapa de bits de la banda superior del menú.
N8	23456	STRIP2	Mapa de bits de la banda inferior del editor.

Notas	Dirección	Nombre	Contenido
X	23551	TSTACK	La pila temporal crece desde aquí hacia abajo.
N8	23552	KSTATE	Usada en la lectura del teclado.
N1	23560	LAST K	Última tecla pulsada.
1	23561	REPDEL	Tiempo, en cincuentavos de segundo (sesentavos en EE.UU.), que debe mantenerse pulsada una tecla antes de que se repita. Inicialmente es 35, pero se le puede dar otros valores (con POKE).
1	23562	REPPER	Pausa, en cincuentavos de segundo (sesentavos en EE.UU.), entre las repeticiones sucesivas de una tecla pulsada (inicialmente, 5).
N2	23563	DEFADD	Si se está evaluando una función definida por el usuario, dirección de los argumentos; de lo contrario, 0.
N1	23565	K DATA	Segundo byte de los controles de color introducidos por el teclado.
N2	23566	TVDATA	Almacena bytes de los controles de color, AT y TAB que van al televisor.
X38	23568	STRMS	Direcciones de los canales conectados a las salidas.
2	23606	CHARS	256 menos que la dirección del juego de caracteres (que empieza con el espacio y continúa hasta el símbolo de copyright). Normalmente en ROM, pero lo puede copiar en la RAM y hacer que CHARS apunte hacia él.
1	23608	RASP	Duración del pitido de aviso.
1	23609	PIP	Duración del 'click' del teclado.
1	23610	ERR NR	1 menos que el código del informe. Empieza en 255 (para -1), de forma que PEEK 23610 da 255.
X1	23611	FLAGS	Diversos indicadores que controlan el sistema BASIC.
X1	23612	TVFLAG	Indicadores asociados con el televisor.
X2	23613	ERR SP	Dirección del elemento de la pila de máquina que hay que usar como retorno de error.
N2	23615	LIST SP	Dirección de la dirección de retorno desde un listado automático.
N1	23617	MODE	Especifica cursor K, L, C, E o G.
2	23618	NEWPPC	Línea a la que hay que saltar.
1	23620	NSPPC	Número de sentencia dentro de una línea al que hay que saltar. La modificación de NEWPPC primero y de NSPPC después fuerza el salto a una sentencia específica dentro de una línea.
2	23621	PPC	Número de línea de la sentencia que está siendo ejecutada.
1	23623	SUBPPC	Número, dentro de la línea, de la sentencia que está siendo ejecutada.

Notas	Dirección	Nombre	Contenido
1	23624	BORDCR	Color del borde multiplicado por 8; también contiene los atributos usados normalmente para la parte inferior de la pantalla.
2	23625	EPPC	Número de la línea actual (con el cursor de programa).
X2	23627	VARS	Dirección de variables.
N2	23629	DEST	Dirección de la variable en asignación.
X2	23631	CHANS	Dirección de los datos del canal.
X2	23633	CURCHL	Dirección de la información que está siendo usada para entrada y salida.
X2	23635	PROG	Dirección del programa de BASIC.
X2	23637	NXTLIN	Dirección de la siguiente línea del programa.
X2	23639	DATADD	Dirección del terminador del último elemento de DATA.
X2	23641	E LINE	Dirección de la orden que está siendo escrita en el teclado.
2	23643	K CUR	Dirección del cursor.
X2	23645	CH ADD	Dirección del próximo carácter que debe ser interpretado (el carácter que sigue al argumento de PEEK, o el código de 'línea nueva' al final de una sentencia POKE).
2	23647	X PTR	Dirección del carácter que está detrás del marcador [?].
X2	23649	WORKSP	Dirección de área de trabajo temporal.
X2	23651	STKBOT	Dirección del extremo inferior de la pila de la calculadora.
X2	23653	STKEND	Dirección del comienzo del espacio libre.
N1	23655	BREG	Registro <i>b</i> de la calculadora.
N2	23656	MEM	Dirección del área usada para memoria de la calculadora. (Generalmente MEMBOT, pero no siempre.)
1	23658	FLAGS2	Otros indicadores.
X1	23659	DF SZ	Número de líneas (incluida una en blanco) de que consta la pantalla inferior.
2	23660	STOP	El número de la primera línea en listados automáticos.
2	23662	OLDPPC	Número de la línea a la cual salta CONTINUE.
1	23664	OSPCC	Número de la sentencia, dentro de la línea, a la cual salta CONTINUE.
N1	23665	FLAGX	Diversos indicadores.
N2	23666	STRLEN	Longitud de la cadena que está siendo asignada.
N2	23668	T ADDR	Dirección del siguiente elemento en la tabla de sintaxis (muy improbable que sea útil).
2	23670	SEED	Semilla para RND. Ésta es la variable establecida por RANDOMIZE.

Notas	Dirección	Nombre	Contenido
3	23672	FRAMES	Contador de barridos del cuadro en 3 bytes (primero el byte menos significativo); se incrementa cada 20 ms. (Véase Parte 18 de este capítulo.)
2	23675	UDG	Dirección del primer gráfico definido por el usuario. Usted puede cambiarla, por ejemplo, para ahorrar espacio disponiendo de menos gráficos definibles.
1	23677	COORDS	x-coordenada del último punto dibujado.
1	23678		y-coordenada del último punto dibujado.
1	23679	P POSN	33 menos número de columna de la posición de la cabeza impresora.
1	23680	PR CC	Byte menos significativo de la dirección de la próxima posición en la que debe escribir LPRINT (en el tampón de la impresora).
1	23681		No utilizada.
2	23682	ECHO E	33 menos el número de columna y 24 menos número de fila (pantalla inferior) del final del tampón de entrada.
2	23684	DF CC	Dirección en el fichero de imagen de la posición de escritura.
2	23686	DF CCL	Como DF CC, pero para la pantalla inferior.
X1	23688	S POSN	33 menos número de columna de la posición de escritura.
X1	23689		24 menos número de fila de la posición de escritura.
X2	23690	SPOSNL	Como S POSN, pero para la pantalla inferior.
1	23692	SCR CT	Cuenta los desplazamientos de la pantalla. Es siempre 1 más que el número de desplazamientos que serán realizados antes de emitir la pregunta scroll?. Si usted modifica este número introduciendo un valor mayor que 1, la pantalla se desplazará sin pedirle permiso.
1	23693	ATTR P	Colores permanentes actuales, etc. (establecidos por las instrucciones de color).
1	23694	MASK P	Usada para colores transparentes, etc. Cualquier bit que sea 1 indica que el tributo correspondiente no se toma de ATTRP, sino de lo que ya está en la pantalla.
N1	23695	ATTR T	Colores temporales actuales, etc. (establecidos por cláusulas de color).
N1	23696	MASK T	Como MASK P, pero temporal.
1	23697	P FLAG	Otros indicadores.
N30	23698	MEMBOT	Área de memoria de la calculadora, usada para almacenar números que no conviene introducir en la pila de la calculadora.

Notas	Dirección	Nombre	Contenido
2	23728		No utilizada.
2	23730	RAMTOP	Dirección del último byte del área de sistema de BASIC.
2	23732	P-RAMT	Dirección del último byte de la RAM física.

Ejercicio

1. El siguiente programa muestra 22 bytes del área de variables (de KSCAN en adelante):

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 23627+256*PEEK 23628+n)
30 NEXT N
```

Trate de identificar la variable de control n a la vista de las descripciones anteriores. Ahora cambie la línea 20 por

```
20 PRINT PEEK (23755+n)
```

El programa muestra así los primeros 22 bytes del área de programa. Trate de correlacionarlos con el propio programa.

Parte 26

Utilización del código de máquina

Temas tratados:

USR con argumento numérico

Esta sección está escrita para quienes ya conocen el *código de máquina* del Z80, es decir, el juego de instrucciones a las que responde el microprocesador Z80. Si usted no sabe nada sobre código de máquina y tiene interés en aprenderlo, puede leer alguno de los muchos libros que se han escrito sobre el tema; debería adquirir uno que se titulase "Código de máquina (o lenguaje ensamblador) en el Z80 para el principiante absoluto"; y si en él se menciona el +2 o los otros ordenadores ZX Spectrum, tanto mejor.

Los programas de código de máquina se escriben normalmente en *lenguaje ensamblador*, el cual, aun siendo bastante críptico, con la práctica no resulta muy difícil de comprender. (Puede ver la lista de las instrucciones del lenguaje ensamblador en la Parte 27 de este capítulo.) Sin embargo, para ejecutar en el +2 un programa escrito en lenguaje ensamblador es necesario convertir esas instrucciones en una serie de bytes, que constituyen el programa en código de máquina. Esa conversión puede realizarla el propio ordenador mediante un programa que se llama *ensamblador*. En el +2 no hay ningún ensamblador incorporado, pero usted puede adquirirlo en una cinta. A falta de ensamblador, puede hacer la conversión usted mismo, pero, en la práctica, esto sólo es posible con programas muy cortos.

Tomemos como ejemplo el programa:

```
ld bc, 99  
ret
```

que carga el par de registros bc con el número 99. Al ensamblarlo, este programa se traduce en los cuatro bytes siguientes: 1, 99, 0 (que significan ld bc, 99) y 201 (que es ret). (Si consulta los códigos 1 y 201 en la tabla de la Parte 27, verá que el 1 corresponde a ld bc, NN, donde NN representa cualquier número de dos bytes, y que el 201 corresponde a ret).

Una vez convertido el programa de lenguaje ensamblador a código de máquina, el siguiente paso es introducirlo en el ordenador (un ensamblador probablemente lo haría de forma automática). Hay que empezar por decidir el lugar de la memoria en el que queremos situarlo (lo más conveniente es reservar un espacio entre el área de BASIC y la de los gráficos definidos por el usuario).

La orden

```
CLEAR 65267
```

reserva un espacio de 100 bytes a partir de la dirección 65268. Para introducir el programa de código de máquina, podríamos usar un programa de BASIC de este estilo:

```
10 LET a=65268
20 READ n: POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

(Este programa se detiene y emite el mensaje E Out of DATA ('Datos agotados') cuando termina de escribir en la memoria los cuatro bytes especificados.)

Para ejecutar el programa de código de máquina se utiliza la función USR (pero ahora con argumento numérico; en concreto, la dirección inicial del programa). El resultado de USR es el valor que queda en el par de registros bc al volver del programa en código de máquina, así que si escribimos

```
PRINT USR 65268
```

obtendremos la respuesta 99.

La dirección de retorno a BASIC es 'apilada' de la manera usual, así que el retorno se lleva a cabo mediante una instrucción ret del Z80. No se debe usar los registros iy e i en rutinas de código de máquina que vayan a utilizar el mecanismo de interrupciones de BASIC. Tampoco se debe cargar i con valores comprendidos entre 40h y 7Fh (aunque no se vaya a usar nunca IM 2). Los valores entre C0h y FFh para i también deberían ser eludidos si se va a insertar la memoria compartida (RAM 4 o RAM 7) entre C000h y FFFFh. Esto se debe a una interacción entre el controlador de video y el mecanismo de refresco del Z80, que puede ocasionar la caída del sistema, la corrupción de la pantalla u otros efectos indeseables. Por tanto, sólo se debe dirigir las interrupciones IM 2 hacia direcciones comprendidas entre 8000h y BFFFh, a menos que se esté muy seguro de la estructura del mapa de memoria.

Hay una serie de problemas típicos que se presentan sistemáticamente cuando se programa en código de máquina un sistema de conmutación de bancos como el del +2. Si a usted le ocurre lo mismo, compruebe que su pila no está siendo conmutada durante las interrupciones, y que su rutina de interrupción está siempre donde debe estar (es conveniente inhibir las interrupciones durante las operaciones de paginación). También es recomendable que guarde una copia del registro de bancos actual en algún lugar no paginado de la RAM, ya que la puerta es de sólo escritura. BASIC y el editor usan la variable de sistema BANK M.

El programa de código de máquina se puede grabar con toda facilidad:

```
SAVE "nombre" CODE 65268,4
```

No hay ninguna manera de grabar el programa de forma que se ejecute automáticamente a sí mismo una vez que haya sido cargado. No obstante, esto se remedia empleando un pequeño programa de BASIC:

```
10 LOAD "" CODE 65268,4  
20 PRINT USR 65268
```

que debe ser grabado en la cinta inmediatamente antes que el código de máquina. El procedimiento sería grabar este programa con la orden:

```
SAVE "cargador" LINE 0
```

y luego grabar el código de máquina con:

```
SAVE "codigo m" CODE 65268,4
```

Hecho esto, se puede ejecutar el código de máquina desde BASIC con sólo dar la orden:

```
LOAD "cargador"
```

que carga y ejecuta automáticamente el programa de BASIC, el cual a su vez carga y ejecuta el código de máquina.



Parte 27

Juego de caracteres del Spectrum

Temas tratados:

Códigos de control

Caracteres

Nemónicos de ensamblador en el Z80

La tabla siguiente da la lista completa del juego de caracteres del Spectrum, con sus códigos en versión decimal y hexadecimal. Por otra parte, si se considera esos códigos como instrucciones de código de máquina del Z80, entonces las columnas de la derecha dan los nemónicos correspondientes en lenguaje ensamblador. Tenga en cuenta que algunas instrucciones del Z80 empiezan por CBh o EDh; éstas aparecen en las dos columnas de la derecha.

Cuando un carácter es distinto en las dos versiones de BASIC (48k y 128k), el carácter correspondiente a 48 BASIC se escribe entre paréntesis a continuación del de 128 BASIC.

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
0	no utilizado	00	nop	ob NOP	E
1	no utilizado	01	ld bc,NN	rlc c	
2	no utilizado	02	ld(bc),a	rlc d	
3	no utilizado	03	inc bc	rlc e	
4	no utilizado	04	inc b	rlc h	
5	no utilizado	05	dec b	rlc l	
6	coma de PRINT	06	ld b,N	rlc(hl)	
7	EDIT	07	rlca	rlc a	
8	cursor a la izquierda, ←	08	ex af,af	rrc b	
9	cursor a la derecha, →	09	add hl,bc	rrc c	
10	cursor abajo, ↓	0A	ld a,(bc)	rrc d	
11	cursor arriba, ↑	0B	dec bc	rrc e	
12	BORR	0C	inc c	rrc h	
13	INTRO	0D	dec c	rrc l	
14	número	0E	ld c,N	rrc (hl)	
15	no utilizado	0F	rrca	rrc a	
16	control de INK	10	djnz DIS	rl b	
17	control de PAPER	11	ld de,NN	rl c	
18	control de FLASH	12	ld (de),a	rl d	
19	control de BRIGHT	13	inc de	rl e	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
20	control de INVERSE	14	inc d	rl h	
21	control de OVER	15	dec d	rl l	
22	control de AT	16	ld d,N	rl (hl)	
23	control de TAB	17	rla	rl a	
24	no utilizado	18	jr DIS	rr b	
25	no utilizado	19	add hl,de	rr c	
26	no utilizado	1A	ld a,(de)	rr d	
27	no utilizado	1B	dec de	rr e	
28	no utilizado	1C	inc e	rr h	
29	no utilizado	1D	dec e	rr l	
30	no utilizado	1E	ld e,N	rr (hl)	
31	no utilizado	1F	rra	rr a	
32	espacio	20	jr nz,DIS	sla b	
33	!	21	ld hl,NN	sla c	
34	"	22	ld(NN),hl	sla d	
35	#	23	inc hl	sla e	
36	\$	24	inc h	sla h	
37	%	25	dec h	sla l	
38	&	26	ld h,N	sla (hl)	
39	'	27	daa	sla a	
40	(28	jr z,DIS	sra b	
41)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	-	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	i	5B	ld e,e	bit 3,e	ld de,(NN)
92	Ñ	5C	ld e,h	bit 3,h	
93	z	5D	ld e,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	—	5F	ld e,a	bit 3,a	ld a,r
96	Pt	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l,(c)
105	i	69	ld l,c	bit 5,c	out (C),l
106	j	6A	ld l,d	bit 5,d	adc hl,hl
107	k	6B	ld l,e	bit 5,e	ld hl,(NN)
108	l	6C	ld l,h	bit 5,h	
109	m	6D	ld l,l	bit 5,l	
110	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	70	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld (hl),l	bit 6,l	
118	v	76	halt	bit 6,(hl)	
119	w	77	ld (hl),a	bit 6,a	
120	x	78	ld a,b	bit 7,b	in a,(c)
121	y	79	ld a,c	bit 7,c	out (c),a
122	z	7A	ld a,d	bit 7,d	adc hl,sp
123	{	7B	ld a,e	bit 7,e	ld sp,(NN)
124	ñ	7C	ld a,h	bit 7,h	
125	}	7D	ld a,l	bit 7,l	
126	~	7E	ld a,(hl)	bit 7,(hl)	
127	©	7F	ld a,a	bit 7,a	
128	☐	80	add a,b	res 0,b	
129	▣	81	add a,c	res 0,c	
130	▤	82	add a,d	res 0,d	
131	▥	83	add a,e	res 0,e	
132	▦	84	add a,h	res 0,h	
133	▧	85	add a,l	res 0,l	
134	▨	86	add a,(hl)	res 0,(hl)	
135	▩	87	add a,a	res 0,a	
136	▪	88	adc a,b	res 1,b	
137	▫	89	adc a,c	res 1,c	
138	▬	8A	adc a,d	res 1,d	
139	▭	8B	adc a,e	res 1,e	
140	▮	8C	adc a,h	res 1,h	
141	▯	8D	adc a,l	res 1,l	
142	▰	8E	adc a,(hl)	res 1,(hl)	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
143	■	8F	adc a,a	res 1,a	
144	(a)	90	sub b	res 2,b	
145	(b)	91	sub c	res 2,c	
146	(c)	92	sub d	res 2,d	
147	(d)	93	sub e	res 2,e	
148	(e)	94	sub h	res 2,h	
149	(f)	95	sub l	res 2,l	
150	(g)	96	sub (hl)	res 2,(hl)	
151	(h)	97	sub a	res 2,a	
152	(i)	98	sbc a,b	res 3,b	
153	(j)	99	sbc a,c	res 3,c	
154	(k)	9A	sbc a,d	res 3,d	
155	(l)	9B	sbc a,e	res 3,e	
156	(m)	9C	sbc a,h	res 3,h	
157	(n)	9D	sbc a,l	res 3,l	
158	(o)	9E	sbc a,(hl)	res 3,(hl)	
159	(p)	9F	sbc a,a	res 3,a	
160	(q)	A0	and b	res 4,b	ldi
161	(r)	A1	and c	res 4,c	cpi
162	(s)	A2	and d	res 4,d	ini
163	SPECTRUM (t)	A3	and e	res 4,e	outi
164	PLAY (u) ↓	A4	and h	res 4,h	
165	RND	A5	and l	res 4,l	
166	INKEY\$	A6	and (hl)	res 4,(hl)	
167	PI	A7	and a	res 4,a	
168	FN	A8	xor b	res 5,b	ldd
169	POINT	A9	xor c	res 5,c	cpd
170	SCREEN\$	AA	xor d	res 5,d	ind
171	ATTR	AB	xor e	res 5,e	outd
172	AT	AC	xor h	res 5,h	
173	TAB	AD	xor l	res 5,l	
174	VAL\$	AE	xor (hl)	res 5,(hl)	
175	CODE	AF	xor a	res 5,a	
176	VAL	B0	or b	res 6,b	ldir
177	LEN	B1	or c	res 6,c	cpir
178	SIN	B2	or d	res 6,d	inir
179	COS	B3	or e	res 6,e	otir
180	TAN	B4	or h	res 6,h	
181	ASN	B5	or l	res 6,l	
182	ACS	B6	or (hl)	res 6,(hl)	
183	ATN	B7	or a	res 6,a	

gráficos de
usuario

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
184	LN	B8	cp b	res 7,b	lddr
185	EXP	B9	cp c	res 7,c	cpdr
186	INT	BA	cp d	res 7,d	indr
187	SQR	BB	cp e	res 7,e	otdr
188	SGN	BC	cp h	res 7,h	
189	ABS	BD	cp l	res 7,l	
190	PEEK	BE	cp (hl)	res 7,(hl)	
191	IN	BF	cp a	res 7,a	
192	USR	C0	ret nz	set 0,b	
193	STR\$	C1	pop bc	set 0,c	
194	CHR\$	C2	jp nz,NN	set 0,d	
195	NOT	C3	jp NN	set 0,e	
196	BIN	C4	call nz,NN	set 0,h	
197	OR	C5	push bc	set 0,l	
198	AND	C6	add a,N	set 0,(hl)	
199	<=	C7	rst 0	set 0,a	
200	>=	C8	ret z	set 1,b	
201	<>	C9	ret	set 1,c	
202	LINE	CA	jp z,NN	set 1,d	
203	THEN	CB		set 1,e	
204	TO	CC	call z,NN	set 1,h	
205	STEP	CD	call NN	set 1,l	
206	DEF FN	CE	adc a,N	set 1,(hl)	
207	CAT	CF	rst 8	set 1,a	
208	FORMAT	D0	pop de	set 2,b	
209	MOVE	D1	pop de	set 2,c	
210	ERASE	D2	jp nc,NN	set 2,d	
211	OPEN #	D3	out (N),a	set 2,e	
212	CLOSE #	D4	call nc,NN	set 2,h	
213	MERGE	D5	push de	set 2,l	
214	VERIFY	D6	sub N	set 2,(hl)	
215	BEEP	D7	rst 16	set 2,a	
216	CIRCLE	D8	ret c	set 3,b	
217	INK	D9	exx	set 3,c	
218	PAPER	DA	jp c,NN	set 3,d	
219	FLASH	DB	in a,(N)	set 3,e	
220	BRIGHT	DC	call c,NN	set 3,h	
221	INVERSE	DD	Prefijo de instrucciones que afectan a ix	set 3,l	
222	OVER	DE	sbc a,N	set 3,(hl)	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
223	OUT	DF	rst 24	set 3,a	
224	LPRINT	E0	ret po	set 4,b	
225	LLIST	E1	pop hl	set 4,c	
226	STOP	E2	jp po,NN	set 4,d	
227	READ	E3	ex (sp),hl	set 4,e	
228	DATA	E4	call po,NN	set 4,h	
229	RESTORE	E5	push hl	set 4,l	
230	NEW	E6	and N	set 4,(hl)	
231	BORDER	E7	rst 32	set 4,a	
232	CONTINUE	E8	ret pe	set 5,b	
233	DIM	E9	jp (hl)	set 5,c	
234	REM	EA	jp pe,NN	set 5,d	
235	FOR	EB	ex de,hl	set 5,e	
236	GOTO	EC	call pe,NN	set 5,h	
237	GO SUB	ED		set 5,l	
238	INPUT	EE	xor N	set 5,(hl)	
239	LOAD	EF	rst 40	set 5,a	
240	LIST	F0	ret p	set 6,b	
241	LET	F1	pop af	set 6,c	
242	PAUSE	F2	jp p,NN	set 6,d	
243	NEXT	F3	di	set 6,e	
244	POKE	F4	call p,NN	set 6,h	
245	PRINT	F5	push af	set 6,l	
246	PLOT	F6	or N	set 6,(hl)	
247	RUN	F7	rst 48	set 6,a	
248	SAVE	F8	ret m	set 7,b	
249	RANDOMIZE	F9	ld sp,hl	set 7,c	
250	IF	FA	jp m,NN	set 7,d	
251	CLS	FB	ei	set 7,e	
252	DRAW	FC	call m,NN	set 7,h	
253	CLEAR	FD	Prefijo de instrucciones que afectan a iy	set 7,l	
254	RETURN	FE	cp N	set 7,(hl)	
255	COPY	FF	rst 56	set 7,a	



Parte 28

Mensajes

Temas tratados:

Informes emitidos por la pantalla
Mensajes de error
Mensajes
CONTINUE

En la última línea de la pantalla aparece un mensaje cada vez que el +2 termina de ejecutar alguna orden en BASIC. Su misión es explicar a qué se debe la detención, ya sea por razones naturales o porque ha habido un error.

El informe consiste en un código (número o letra), un breve texto que explica lo que ha ocurrido y el número de línea (y el número de sentencia dentro de esa línea) en la que se ha detenido. Cuando se trata de una orden directa, el número de línea que muestra es el 0. Dentro de una línea, la sentencia 1 es la que se encuentra al principio, la 2 es la que va a continuación del primer signo de dos puntos (o de THEN), etc.

El comportamiento de CONTINUE depende en gran medida de los mensajes. Normalmente CONTINUE va a la línea y sentencia indicadas en el último informe, pero hay excepciones en los mensajes de códigos 0, 9 y D.

La siguiente tabla muestra todos los informes e indica en qué circunstancias pueden aparecer; con esta información se puede consultar la Parte 30. Por ejemplo, el error A Invalid argument (Argumento no válido) puede presentarse en SQR, IN, ACS y ASN, y la descripción que se da de estas palabras clave en la Parte 30 indica exactamente qué argumentos no son válidos para ellas.

Código	Texto y significado	Situación
0	OK Tarea terminada con éxito, o intento de saltar a un número de línea mayor que cualquiera de los existentes. Este mensaje no cambia la línea ni la sentencia a la que salta CONTINUE.	Cualquiera

Código	Texto y significado	Situación
1	<p>NEXT without FOR</p> <p>NEXT sin FOR</p> <p>La variable de control no existe (no ha sido establecida por una sentencia FOR), pero hay una variable ordinaria con el mismo nombre.</p>	NEXT
2	<p>Variable not found</p> <p>Variable no encontrada</p> <p>En el caso de una variable sencilla, ocurre si se intenta utilizarla antes de definirla (con LET, READ o INPUT), o de cargarla desde la cinta, o de definirla en una sentencia FOR. En el caso de una variable indexada, ocurre si se intenta utilizarla antes de dimensionar la matriz (DIM) o de cargarla desde la cinta.</p>	Cualquiera
3	<p>Subscript wrong</p> <p>Subíndice incorrecto</p> <p>Un subíndice es mayor que la dimensión de la matriz, o el número de subíndices no se corresponde con el de dimensiones de la matriz. Si el subíndice es negativo o mayor que 65535, se produce el error B.</p>	Variables indexadas, subcadenas
4	<p>Out of memory</p> <p>Memoria agotada</p> <p>No hay espacio suficiente en la memoria del ordenador para lo que se pretende hacer. Si el ordenador parece ser incapaz de salir de esta situación, tendrá que borrar la línea de órdenes actual pulsando BORR y luego borrar una o dos líneas del programa (con la intención de reintroducirlas más tarde).</p>	LET, INPUT, FOR, DIM, GO SUB, LOAD, MERGE. A veces durante la evaluación de una expresión

Código	Texto y significado	Situación
5	<p>Out of screen</p> <p>Fuera de la pantalla</p> <p>Una sentencia INPUT ha tratado de generar más de 23 líneas en la pantalla inferior. También se produce con PRINT AT 22,xx.</p>	INPUT, PRINT AT
6	<p>Number too big</p> <p>Número demasiado grande</p> <p>Los cálculos han desembocado en un número superior a aproximadamente 10^{38}.</p>	Cualquier cálculo aritmético
7	<p>RETURN without GO SUB</p> <p>RETURN sin GO SUB</p> <p>Hay un RETURN al que no corresponde ningún GO SUB.</p>	RETURN
8	<p>End of file</p> <p>Fin de fichero.</p>	Operaciones con microunidades, etc.
9	<p>STOP statement</p> <p>Sentencia STOP</p> <p>Después de este mensaje, CONTINUE no repetirá la sentencia STOP, sino que continuará a partir de la sentencia siguiente.</p>	STOP
A	<p>Invalid argument</p> <p>Argumento no válido</p> <p>El argumento que se ha puesto en una función no es adecuado.</p>	SQR, LN, ASN,USR (con argumento literal)

Código	Texto y significado	Situación
B	<p>Interger out of range</p> <p>Entero fuera de margen</p> <p>Cuando una sentencia requiere un entero, el argumento de punto flotante es redondeado al entero más próximo. Este error se produce si el resultado de la operación de redondeo es inadecuado.</p> <p>En relación con las matrices, véase también el error 3.</p>	<p>RUN, RANDOMIZE, POKE, DIM, GOTO, GO SUB, LIST, LLIST, PAUSE, PLOT, CHR\$, PEEK,USR (con argumento numérico)</p> <p>Acceso a matrices</p>
C	<p>Nonsense in BASIC</p> <p>Absurdo en BASIC</p> <p>El texto del argumento (literal) no constituye una expresión válida. También se produce cuando el argumento de una función o de una orden es escandalosamente erróneo.</p>	<p>VAL, VAL\$</p>
D	<p>BREAK – CONT repeats</p> <p>BREAK – CONT para repetir</p> <p>Se ha pulsado BREAK durante alguna operación con algún periférico. El comportamiento de CONTINUE tras este informe es normal en el sentido de que repite la sentencia. Compárelo con el informe L.</p>	<p>LOAD, SAVE, VERIFY, MERGE. También cuando el ordenador pregunta scroll? y se responde pulsando N, BREAK o la barra espaciadora</p>
E	<p>Out of DATA</p> <p>Datos agotados</p> <p>Se ha tratado de leer más allá del final de la lista DATA.</p>	<p>READ</p>
F	<p>Invalid file name</p> <p>Nombre de fichero no válido</p> <p>El nombre especificado tras SAVE consta de más de 10 caracteres o es la cadena vacía.</p>	<p>SAVE</p>

Código	Texto y significado	Situación
G	No room for line No queda espacio para la línea No queda espacio en la memoria para acomodar la nueva línea de programa.	Introducción de una línea de programa
H	STOP in INPUT STOP en INPUT Algún dato introducido en INPUT comenzó con STOP. A diferencia del caso del informe 9, después de H la orden CONTINUE se comporta normalmente, repitiendo la sentencia INPUT.	INPUT
I	FOR without NEXT FOR sin NEXT Se ha establecido un bucle que no tiene que ser ejecutado ninguna vez (por ejemplo, FOR n=1 TO 0) y BASIC no ha encontrado el NEXT correspondiente.	FOR
J	Invalid I/O device Dispositivo de E/S no válido	Operaciones con microunidades, etc.
K	Invalid colour Color no válido El número especificado no es correcto.	INK, PAPER, BORDE, FLASH, BRIGHT, INVERSE, OVER; también después de uno de los correspondientes caracteres de control
L	BREAK into program Programa interrumpido Se ha pulsado [BREAK] . La posible pulsación de esta tecla se explora entre cada dos sentencias. Los nú-	Cualquiera

Código	Texto y significado	Situación
	meros de línea y sentencia del informe se refieren a la sentencia anterior a la pulsación de BREAK , pero CONTINUE pasa a la sentencia siguiente, de forma que no repite ninguna.	
M	<p>RAMTOP no good RAMTOP no válido</p> <p>El número especificado para RAMTOP es demasiado grande o demasiado pequeño.</p>	CLEAR ; posiblemente también en RUN
N	<p>Statement lost Sentencia perdida</p> <p>Salto a una sentencia que ya no existe.</p>	RETURN , NEXT , CONTINUE
O	<p>Invalid Stream Canal no válido</p>	Operaciones con microunidades, etc.
P	<p>FN without DEF FN sin DEF</p> <p>Se ha intentado usar una función de usuario que no está definida en el programa.</p>	FN7
Q	<p>Parameter error Error en parámetro</p> <p>Número de argumentos incorrecto, o bien uno de ellos es de tipo incorrecto (cadena en vez de número, o vice versa).</p>	FN

Código	Texto y significado	Situación
R	<p>Tape loading error</p> <p>Error de carga de cinta</p> <p>Se ha encontrado el fichero en la cinta, pero por alguna razón no ha sido posible leerlo o verificarlo.</p>	VERIFY, LOAD, MERGE
a	<p>MERGE error</p> <p>Error en MERGE</p> <p>MERGE ! no ha podido ser ejecutada por alguna razón (tamaño o tipo de fichero incorrectos).</p>	MERGE !
b	<p>Wrong file type</p> <p>Fichero de tipo incorrecto</p> <p>Se ha especificado un fichero de tipo inadecuado para una operación con el disco de silicio; por ejemplo, un fichero CODE en LOAD ! "<i>nombre</i>".</p>	MERGE !, LOAD !
c	<p>CODE error</p> <p>Error en CODE</p> <p>El tamaño del fichero encontrado es tal que se sobrepasaría el límite de la memoria.</p>	LOAD ! <i>fichero</i> CODE
d	<p>Too many brackets</p> <p>Demasiados paréntesis</p> <p>Demasiados paréntesis en una frase repetida en uno de los argumentos.</p>	PLAY
e	<p>File already exists</p> <p>Ya existe el fichero</p> <p>Ya existe en el disco de silicio un fichero con ese mismo nombre.</p>	SAVE !

Código	Texto y significado	Situación
f	<p>Invalid name</p> <p>Nombre no válido</p> <p>El nombre de fichero especificado es la cadena vacía o tiene más de diez caracteres.</p>	ERASE I
h	<p>File does not exist</p> <p>No existe ese fichero</p> <p>En el disco de silicio no hay ningún fichero con ese nombre.</p>	LOAD I, ERASE I
i	<p>Invalid device</p> <p>Dispositivo no válido</p> <p>El nombre del dispositivo que se ha puesto en la orden FORMAT no existe o no corresponde a un dispositivo físico.</p>	FORMAT
j	<p>Invalid baud rate</p> <p>Velocidad de transmisión no válida</p> <p>Se ha especificado velocidad cero para la puerta RS232.</p>	FORMAT
k	<p>Invalid note name</p> <p>Nombre de nota no válido</p> <p>PLAY ha encontrado una nota o una orden que no reconoce, o una orden escrita en minúsculas.</p>	PLAY
l	<p>Number too big</p> <p>Número demasiado grande</p> <p>El parámetro de una orden de PLAY es un orden de magnitud demasiado grande.</p>	PLAY

Código	Texto y significado	Situación
m	Note out of range Nota fuera de margen Una serie de bemoles o sostenidos ha producido una nota que excede el margen del chip de sonido.	PLAY
n	Out of range Fuera de margen El parámetro de una orden es demasiado grande o demasiado pequeño. Si es muy grande, se produce el error l.	PLAY
o	Too many tied notes Demasiadas notas ligadas Se ha intentado hacer una ligadura con demasiadas notas.	PLAY



Parte 29

Información de referencia

Temas tratados:

Hardware

El +2 está diseñado en torno al microprocesador Z80, el cual funciona a una velocidad de 3.54 MHz (3.54 millones de ciclos por segundo).

La memoria del +2 está dividida en 32K de ROM y 128K de RAM y distribuida en páginas de 16K. A las dos páginas de ROM se accede a través de los 16K inferiores (direcciones 0 a 3FFFh) del mapa de la memoria. Las ocho páginas de RAM (0-7) son accesibles a través de los 16K superiores del mapa (direcciones C000h a FFFFh). La página 5 de la RAM también está situada en el margen de 4000h a 7FFFh, y la página 2 en el comprendido entre 8000h y BFFFh.

Físicamente hablando, la ROM es un solo dispositivo de 32K (similar a un 27256) que el sistema trata como si fuesen dos chips de 16K. La RAM está compuesta por dieciséis chips de 64K × 1 bit (4164), algunos de los cuales (bancos RAM4 a RAM7) están compartidos entre los circuitos que producen la imagen de televisión (de los que hablaremos más adelante) y el Z80A. Los otros ocho chips (bancos RAM0 a RAM3), así como la ROM, son de uso exclusivo del Z80A.

La matriz lógica no comprometida (ULA, *Uncommitted Logic Array*) controla la mayor parte de las operaciones de E/S: teclado, magnetófono, pantalla, etc. Convierte los bytes que se encuentran en la memoria en formas y colores para la pantalla y permite al Z80A explorar el teclado y leer y escribir los datos en la cinta.

El sonido de tres canales es producido por el AY-3-8912, un chip de sonido muy popular; este dispositivo controla también las puertas RS232, MIDI y el subteclado numérico. La manera en que trabaja es bastante compleja; se recomienda a quienes se sientan tentados a experimentar que consulten la hoja de datos del AY-3-8912. No obstante, la siguiente información debería ser suficiente para empezar. El chip de sonido contiene dieciséis registros que se seleccionan escribiendo primero el número de registro en la puerta de escritura de direcciones (dirección de E/S FFFDh, 65533 en decimal) y después leyendo el valor del registro (en la misma dirección) o escribiendo en la dirección de escritura de registros de datos (BFFDh, 49149 en decimal). Una vez que ha sido seleccionado un registro, se puede realizar cualquier número de operaciones de lectura o escritura de datos. Sólo habrá que volver a escribir en la puerta de escritura de direcciones cuando se necesite seleccionar otro registro.

La frecuencia de reloj básica de este circuito es 1.7734 MHz (con precisión del 0.01%).

Los registros hacen lo siguiente:

- R0 Ajuste fino del tono, canal A
- R1 Ajuste aproximado del tono, canal A
- R2 Ajuste fino del tono, canal B
- R3 Ajuste aproximado del tono, canal B
- R4 Ajuste fino del tono, canal C
- R5 Ajuste aproximado del tono, canal C

El tono de cada canal es un valor de 12 bits formado combinando los bits D3-D0 del registro de ajuste aproximado y los bits D7-D0 del registro de ajuste fino. La unidad básica del tono es la frecuencia de reloj dividida por 16 (es decir, 110.83 KHz). Como el contador es de 12 bits, se puede generar frecuencias de 27Hz a 110 KHz.

- R6 Control del generador del ruido, D4-D0

El periodo del generador de ruido se toma contando los cinco bits inferiores del registro de ruido cada periodo del reloj de sonido dividido por 16.

- R7 Control del mezclador y de E/S

- D7 No utilizado
- D6 1=puerta de entrada, 0=puerta de salida
- D5 Ruido en el canal C
- D4 Ruido en el canal B
- D3 Ruido en el canal A
- D2 Tono en el canal C
- D1 Tono en el canal B
- D0 Tono en el canal A

Este registro controla la mezcla de ruido y tono para cada canal y la dirección de la puerta de E/S de ocho bits. Un cero en un bit de mezcla indica que la función está activada.

- R8 Control de amplitud del canal A
- R9 Control de amplitud del canal B
- RA Control de amplitud del canal C

- D4 1=utilizar generador de envolvente
0=utilizar el valor de D3-D0 como amplitud
- D3-D0 Amplitud

Estos tres registros controlan la amplitud de cada canal y si ésta debe ser modulada o no por los registros de envolvente.

- RB Ajuste aproximado del periodo de envolvente
- RC Ajuste fino del periodo de envolvente

Los valores de ocho bits de RB y RC se combinan para producir un número de 16 bits que se cuenta en unidades de 256 por el periodo del reloj de sonido. Las frecuencias de envolvente pueden estar entre 0.1Hz y 6KHz.

RD Control de envolventes

D3 Continua

D2 Ataque

D1 Alternada

D0 Sostenida

El diagrama de las formas de envolvente (Parte 19 de este capítulo) da una ilustración gráfica de los posibles estados de este registro.



Parte 30

BASIC

Temas tratados:

- Números
- Variables
- Cadenas
- Funciones
- Resumen de palabras clave
- Operaciones matemáticas

Los números se almacenan en BASIC con precisión de 9 o 10 dígitos. El mayor número que puede manejar BASIC es aproximadamente 10^{38} ; el más pequeño (positivo) es aproximadamente $4 \cdot 10^{-39}$.

La forma de almacenamiento es la *binaria de punto flotante*, con un byte para el exponente (e , $1 \leq e \leq 255$) y cinco bytes para la mantisa (m , $\frac{1}{2} \leq m < 1$), lo que representa el número $m \cdot 2^{e-128}$.

Puesto que $\frac{1}{2} \leq m < 1$, el bit más significativo de la mantisa m siempre es 1. Por lo tanto, podemos utilizarlo como indicador del signo (0 para los números positivos y 1 para los negativos).

Los enteros pequeños tienen una representación especial en la que el primer byte es 0, el segundo es el byte del signo (0 o FFh) y el tercero y el cuarto son el entero propiamente dicho (en complemento a dos) con el byte menos significativo en primer lugar.

Los nombres de las variables numéricas son de longitud arbitraria; su primer carácter siempre es una letra y los siguientes pueden ser letras o dígitos. BASIC permite los espacios, pero los ignora; además convierte internamente todas la letras en minúsculas.

Los nombres de las variables de control de los bucles FOR...NEXT consisten en una sola letra.

Los nombres de las matrices numéricas consisten en una sola letra, que puede coincidir con el nombre de una variable sencilla. Estas matrices pueden tener múltiples dimensiones de tamaño arbitrario. Los subíndices empiezan en el 1.

Las cadenas son de longitud completamente arbitraria. Su nombre consiste en una sola letra seguida de \$.

Las matrices literales pueden tener múltiples dimensiones de tamaño arbitrario. Su nombre consiste en una sola letra seguida de \$ y no puede coincidir con el nombre de

una variable literal sencilla. Todas las cadenas de una matriz dada tienen la misma longitud fija, que está especificada por el último parámetro de la sentencia DIM. Los subíndices empiezan con el 1.

Disección de cadenas. Dada una cadena, se puede especificar una subcadena utilizando los llamados *cortadores*. Los cortadores pueden tener cualquiera de las siguientes formas:

- (i) vacío
- (ii) una expresión numérica
- (iii) *expresión numérica opcional* TO *expresión numérica opcional*

La subcadena se expresa mediante:

(a) *expresión literal*(cortador)

o

(b) *variable de matriz literal*(subíndice, ..., subíndice, cortador)

que es lo mismo que

variable de matriz literal(subíndice, ..., subíndice)(cortador)

Supongamos que la expresión literal, caso (a), es s\$.

En el caso (a), si el cortador es vacío el resultado es la misma expresión literal s\$ (que se considera como subcadena de sí misma).

Si el cortador es una expresión numérica, con valor m , el resultado es el m -ésimo carácter de s\$ (o sea, la subcadena tiene longitud 1).

Si el cortador tiene la forma (iii), supongamos que la primera expresión numérica tiene el valor m (el valor por defecto es 1) y la segunda el valor n (el valor por defecto es la longitud de la expresión literal). Si $1 \leq m \leq n \leq \text{LEN } s\$$, el resultado es la subcadena de s\$ que comienza en el m -ésimo carácter y acaba en el n -ésimo.

Si $0 \leq n < m$, el resultado es la cadena vacía. En cualquier otro caso se produce el error 3.

BASIC realiza la disección de las cadenas antes de evaluar las funciones y operaciones, a menos que los paréntesis impongan lo contrario.

A las subcadenas se les puede asignar valor (véase LET). Si una cadena debe contener comillas, al especificarla se debe poner el signo de comillas (") repetido.

Funciones

El argumento de una función no necesita paréntesis si es una constante o una variable (opcionalmente, subindexada o producto de una disección).

Función	Tipo de argumento	Resultado
ABS	Número	Valor absoluto.
ACS	Número	Arco coseno en radianes. Error A si el argumento no está entre -1 y $+1$.
AND	Operación binaria. El operando de la derecha siempre es un número. Si el de la izquierda es un número: Si el de la izquierda es una cadena:	$a \text{ AND } b \text{ es } \begin{cases} a & \text{si } b <> 0 \\ 0 & \text{si } b = 0 \end{cases}$ $a\$ \text{ AND } b \text{ es } \begin{cases} a\$ & \text{si } b <> 0 \\ "" & \text{si } b = 0 \end{cases}$
ASN	Número	Arco seno en radianes. Error A si el argumento no está entre -1 y $+1$.
ATN	Número	Arco tangente en radianes.
ATTR	Dos argumentos, x e y , ambos numéricos, escritos entre paréntesis	Un número cuya forma binaria codifica los atributos de la posición de carácter que está en la línea x , columna y , de la pantalla. El bit 7 (el más significativo) es 1 para parpadeo, 0 para fijo. El bit 6 es 1 para brillo intenso, 0 para normal. Los bits 5 al 3 dan el color del papel. Los bits 2 al 0 dan el color de la tinta. Error B a menos que $0 \leq x \leq 23$ y $0 \leq y \leq 31$.
BIN		No es realmente una función, sino una notación alternativa para números: BIN seguido de una secuencia de ceros y unos es el número cuya representación binaria es la especificada por esa secuencia.
CHR\$	Número	El carácter cuyo código es el argumento (redondeado al entero más próximo).

Función	Tipo de argumento	Resultado
CODE	Cadena	Código del primer carácter de la cadena (o 0, si se trata de la cadena vacía).
COS	Número (en radianes)	Coseno del argumento.
EXP	Número	Número <i>e</i> elevado al argumento.
FN		FN, seguida de una letra, invoca una función definida por el usuario (véase DEF). Los argumentos tienen que estar entre paréntesis. (Hay que poner los paréntesis aunque no haya argumentos.)
IN	Número	Resultado de leer (a nivel del microprocesador) la puerta de entrada especificada por el argumento <i>x</i> ($0 \leq x \leq \text{FFFFh}$). Carga el par de registros bc con <i>x</i> y ejecuta la instrucción de lenguaje ensamblador in a,(c).
INKEY\$	Ninguno	Lee el teclado. El resultado es el carácter que representa la tecla pulsada, si está pulsada una y sólo una; de lo contrario, el resultado es la cadena vacía.
INT	Número	Parte entera (redondeado hacia abajo).
LEN	Cadena	Longitud del argumento.
LN	Número	Logaritmo neperiano o natural (de base <i>e</i>). Error A si $x \leq 0$.
NOT	Número	0 si $x < > 0$, 1 si $x = 0$. NOT tiene prioridad de nivel 4.
OR	Operación binaria. Ambos operandos son numéricos	$a \text{ OR } b \text{ es } \begin{cases} 1 & \text{si } b < > 0 \\ a & \text{si } b = 0 \end{cases}$ <p>OR tiene prioridad de nivel 2.</p>
PEEK	Número	El valor del byte que está en la posición de memoria cuya dirección es el argumento (redondeado al entero más cercano). Error B si el argumento no está entre 0 y 65535.
PI	Ninguno	Número π (3.1415927...).
POINT	Dos argumentos, <i>x</i> e <i>y</i> , ambos numéricos, escritos entre paréntesis	1 si el pixel especificado por (<i>x</i> , <i>y</i>) tiene el color de la tinta; 0 si tiene el del papel. Error B a menos que $0 \leq x \leq 255$ y $0 \leq y \leq 175$.

Función	Tipo de argumento	Resultado
RND	Ninguno	El siguiente número pseudoaleatorio de una sucesión que se genera tomando las potencias de 75 módulo 65537, restando 1 y dividiendo por 65536. $0 \leq y < 1$.
SCREEN\$	Dos argumentos, x e y , ambos numéricos, escritos entre paréntesis	El carácter que está (normal o invertido) en la posición de la pantalla especificada por la fila x , columna y . Da la cadena vacía si no se reconoce el carácter. Error B a menos que $0 \leq x \leq 23$ y $0 \leq y \leq 31$.
SGN	Número	Signo del número. Da -1 para los negativos, 0 para el 0 y $+1$ para los positivos.
SIN	Número (en radianes)	Seno del argumento.
SQR	Número	Raíz cuadrada. Error A si $x < 0$.
STR\$	Número	La cadena de caracteres que aparecería en la pantalla si se escribiera el número.
TAN	Número (en radianes)	Tangente del argumento.
USR	Número	Invoca a la subrutina de código de máquina cuya dirección inicial es el argumento. Al retornar, el resultado es el contenido del par de registros bc.
	Cadena	Dirección de la descripción de la forma del gráfico de usuario correspondiente al argumento. Error A si el argumento no es una sola letra, de la a a la u, o un gráfico de usuario.
VAL	Cadena	Evalúa el argumento (tras suprimir las comillas) como si fuera una expresión numérica. Error C si el argumento no es válido como expresión numérica. Pueden darse otros errores, dependiendo de la forma de la expresión.
VAL\$	Cadena	Evalúa el argumento (tras suprimir las comillas de sus extremos) y da la expresión literal resultante. Error C si el argumento contiene un error de sintaxis o da un valor numérico. Pueden darse otros errores, dependiendo de la expresión.
-	Número	Negación.

Las siguientes son operaciones binarias:

- + Suma de números o concatenación de cadenas
 - Resta
 - * Multiplicación
 - / División
 - ↑ Elevación a una potencia. Error B si el operando de la izquierda es negativo
 - = Igual que
 - > Mayor que
 - < Menor que
 - <= Igual o menor que
 - >= Igual o mayor que
 - <> Distinto de
- } Ambos operandos deben ser del mismo tipo. El resultado es el número 1 si la proposición es 'verdadera'; 0 si no lo es.

Las funciones y operaciones tienen las siguientes prioridades:

Operación	Nivel de prioridad
Subíndices y disección de cadenas	12
Todas las funciones excepto NOT y negación	11
↑	10
Negación (signo menos usado para negar)	9
*, /	8
=, - (signo menos usado para restar)	6
=, >, <, <=, >=, <>	5
NOT	4
AND	3
OR	2

Sentencias

Notación utilizada en esta lista:

- a* Representa una sola letra.
- v* Representa una variable.
- x, y, z* Representan expresiones numéricas.
- m, n* Representan expresiones numéricas que BASIC redondea al entero más próximo.
- e* Representa una expresión.
- f* Representa una expresión cuyo valor es una cadena.
- s* Representa una secuencia de sentencias separadas por signos de dos puntos.
- c* Representa una secuencia de cláusulas de color, cada una de las cuales termina en una coma o en un signo de punto y coma. Una cláusula de color tiene la forma de una sentencia PAPER, INK, FLASH, BRIGHT, INVERSE u OVER.

Observe que en todo lugar se puede poner expresiones en lugar de constantes (excepto para el número de línea al principio de una sentencia).

Todas las sentencias, a excepción de INPUT, DEF FN y DATA, pueden ser usadas como órdenes directas o en líneas de programa (aunque no tendrán la misma utilidad en ambos casos). Tanto las órdenes directas como las líneas de programa pueden constar de varias sentencias separadas por signos de dos puntos. No hay restricción alguna sobre en qué lugar de la línea puede aparecer una sentencia concreta; véase, no obstante, IF y REM.

BEEP x,y	Hace sonar una nota a través del altavoz del televisor durante x segundos a una altura de y semitonos por encima de la nota DO medio.
BORDER m	Establece el color del borde de la pantalla superior y también el color del papel para la pantalla inferior.
BRIGHT n	Establece el brillo de los caracteres que se escriba en lo sucesivo; $n=0$ para normal, 1 para brillo y 8 para transparente. Error K si n no es 0, 1 ni 8.
CAT	Sólo funciona con microunidades, etc.
CAT I	Da una lista de los ficheros que hay actualmente en el disco de silicio.
CIRCLE x,y,z	Dibuja un arco de circunferencia, con centro en (x,y) y radio z .
CLEAR	Borra todas las variables, liberando así el espacio que ocupaban. Ejecuta RESTORE y CLS; reajusta la posición del cursor gráfico en el extremo inferior izquierdo y borra la pila de GO SUB.
CLEAR n	Igual que CLEAR, pero cambia la variable de sistema RAMTOP a n (si ello es posible) y coloca en esa dirección la nueva pila de GO SUB.
CLOSE #	Sólo funciona con microunidades, etc.
CLS	Borra el fichero de imagen (memoria de pantalla).
CONTINUE	Reanuda la ejecución del programa a partir de la sentencia en que éste se había detenido emitiendo un informe distinto del 0. Si el mensaje fue 9 o L, continúa a partir de la sentencia siguiente; de lo contrario, repite la sentencia en la que se produjo el error. Si el último informe se produjo en una línea de órdenes, CONTINUE intentará completar dicha línea y entrará en un bucle si el error fue en 0:1, generará el mensaje 0 si fue en 0:2, o bien el mensaje N si tuvo lugar en 0:3 o posterior.

COPY	Envía a la impresora (si está conectada) una copia de las 22 líneas superiores de la pantalla en formato de mapa de bit Epson de densidad cuádruple; de lo contrario, no hace nada. Mensaje D si se pulsa BREAK .
DATA e_1, e_2, e_3, \dots	Define un tramo de la lista de datos. Debe estar en un programa; de lo contrario, no produce efecto.
DEF FN $a(a_1, \dots, a_k) = e$	Definición de una función de usuario. Debe estar en un programa; de lo contrario, no produce efecto. a y a_1 a a_k son, cada una, una sola letra (o una letra seguida de \$ para especificar un resultado o un argumento literal). Toma la forma DEF FN $a() = e$ si no hay argumentos.
DIM $a(n_1, \dots, n_k)$	Borra la matriz a (si existe) y forma la matriz numérica a con k dimensiones, n_1, \dots, n_k . Inicializa todos los valores a 0.
DIM $a\$(n_1, \dots, n_k)$	Borra la matriz o cadena $a\$($ (si existen) y forma una matriz de caracteres $a\$($ con k dimensiones, n_1, \dots, n_k . Inicializa todos los valores a "". Se puede considerar esta matriz como matriz de cadenas de longitud fija n_k , con $k-1$ dimensiones (n_1, \dots, n_{k-1}). Una matriz está indefinida hasta que se la dimensiona con DIM. Error 4 si no queda espacio para la matriz en la memoria.
DRAW x, y	Equivale a DRAW $x, y, 0$.
DRAW x, y, z	Dibuja una línea (recta o arco de circunferencia) desde la posición actual del cursor gráfico desplazándose x unidades en horizontal e y unidades en vertical en relación con el punto de partida, y al mismo tiempo girando un ángulo z . Error B si la línea se sale de la pantalla.
ERASE	Sólo funciona con microunidades, etc.
ERASE I f	Borra un fichero del disco de silicio.
FLASH n	Establece si los caracteres deben ser parpadeantes o fijos; $n=0$ para fijos, $n=1$ para parpadeo, $n=8$ para que no haya cambios.
FOR $a=x$ TO y	Borra la variable ordinaria a (si existe); define la variable de control a con valor inicial x , límite y , paso z y con una dirección de retorno del bucle que hace referencia a la sentencia que sigue a FOR. Comprueba si el valor inicial es mayor (si $z \geq 0$) o menor (si $z < 0$) que el límite; en caso afirmativo, salta a la sentencia NEXT a , dando el error I si ésta no existe. Véase NEXT. Error 4 si no queda espacio para la variable de control en la memoria.
FOR $a=x$ TO y STEP 1	
FOR $a=x$ TO y STEP z	

FORMAT <i>f,n</i>	Establece en <i>n</i> baudios la velocidad de transmisión para el dispositivo <i>f</i> . Son dispositivos válidos "P" y "P" (la puerta RS232) y velocidades válidas las comprendidas entre 75 y 19200.
GO SUB <i>n</i>	Deposita el número de línea <i>n</i> en la pila; después actúa como GO TO <i>n</i> . Error 4 si no hay suficientes sentencias RETURN.
GO TO <i>n</i>	Salta a la línea de número <i>n</i> (o, si no existe, a la primera línea posterior a ese número).
IF <i>x</i> THEN <i>s</i>	Si <i>x</i> es 'verdadero' (distinto de cero), ejecuta <i>s</i> . Observe que <i>s</i> comprende todas las sentencias hasta el final de la línea. La forma IF <i>x</i> THEN <i>número-de-línea</i> no está permitida.
INK <i>n</i>	Establece el color de la tinta (primer plano) para los caracteres que se escriba en lo sucesivo; <i>n</i> está en el margen de 0 a 7 para un color, <i>n</i> =8 para transparentes y <i>n</i> =9 para contraste. Error K a menos que $0 \leq n \leq 9$.
INPUT ...	'...' es una secuencia de elementos de INPUT, separados, al igual que en una sentencia PRINT, por comas, signos de punto y coma o apóstrofes. Un elemento de INPUT puede ser cualquiera de los siguientes: <ul style="list-style-type: none"> (i) Cualquier elemento de PRINT que no empiece por una letra. (ii) Un nombre de variable. (iii) LINE seguido de un nombre de variable de tipo literal. Los elementos de PRINT y los separadores del apartado (i) son tratados exactamente igual que en PRINT, con la excepción de que la escritura se dirige a la pantalla inferior. Para (ii), el ordenador se detiene y espera la introducción de una expresión por el teclado; el valor de esa expresión es asignado a la variable. Los caracteres escritos son reproducidos en la pantalla en la forma habitual; los errores de sintaxis producen un signo de interrogación en negativo y parpadeante. En el caso de expresiones de tipo literal, el tampón de entrada se inicializa de forma que contenga dos comillas (que pueden ser borradas, si es necesario). Si el primer carácter de la entrada es STOP (SIMB A), el programa se detiene y emite el error H. (iii) es igual que (ii), con la diferencia de que la entrada es tratada como constante literal sin comillas y el mecanismo de STOP no funciona (en su lugar, para detener la sentencia hay que pulsar la tecla ↓).

INVERSE <i>n</i>	<p>Controla la inversión de los caracteres que se escriba en lo sucesivo. Si $n=0$, los caracteres aparecerán en video normal, es decir, color de la tinta sobre color del papel. Si $n=1$, los caracteres aparecen en video inverso, esto es, color del papel sobre color de la tinta. Error K (véase la Parte 28 de este capítulo) si n no es 0 ni 1.</p> <p>En 48 BASIC, la pulsación de la tecla <u>VIDEO INV</u> es equivalente a INVERSE 1; la pulsación de <u>VIDEO NORM</u> equivale a INVERSE 0.</p>
LET $v=e$	<p>Asigna el valor de e a la variable v. No se puede omitir la palabra 'LET'. Una variable sencilla está indefinida mientras no se le asigne valor en una sentencia LET, READ o INPUT. Si v es una variable literal indexada o una variable literal obtenida por disección de cadenas (subcadena), la asignación es «procrustea» (de longitud fija); es decir, el valor de e es truncado o rellenado con espacios por la derecha, con el fin de darle la longitud que se ha especificado para v.</p>
LIST	<p>Equivale a LIST 0.</p>
LIST <i>n</i>	<p>Lista el programa en la pantalla superior, comenzando en la primera línea cuyo número es, como mínimo, n; convierte la línea n en línea actual.</p>
LLIST	<p>Equivale a LLIST 0.</p>
LLIST <i>n</i>	<p>Como LIST, pero dirigiendo el listado hacia la impresora.</p>
LOAD <i>f</i>	<p>Carga el programa y las variables.</p>
LOAD <i>f</i> DATA ()	<p>Carga una matriz numérica.</p>
LOAD <i>f</i> DATA \$()	<p>Carga una matriz literal.</p>
LOAD <i>f</i> CODE m,n	<p>Carga (como máximo) n bytes, comenzando en la dirección m.</p>
LOAD <i>f</i> CODE m	<p>Carga bytes comenzando en la dirección m.</p>
LOAD <i>f</i> CODE	<p>Carga bytes en la dirección desde la cual fueron grabados.</p>
LOAD <i>f</i> SCREEN\$	<p>Equivale a LOAD <i>f</i> CODE 16384,6912.</p>
LOAD !	<p>Como LOAD (véase más arriba las opciones), pero usa el disco de silicio.</p>
LPRINT ...	<p>Como PRINT, pero dirigiendo la salida hacia la impresora.</p>
MERGE <i>f</i>	<p>Como LOAD <i>f</i>, pero no borra las líneas ni las variables del programa antiguo, salvo las que tengan el mismo número o nombre que en el programa nuevo.</p>
MERGE ! <i>f</i>	<p>Como MERGE <i>f</i>, pero usa el disco de silicio.</p>
MOVE f_1,f_2	<p>Sólo funciona con microunidades, etc.</p>

NEW	Inicializa el sistema BASIC, borrando todos los programas y variables y toda la memoria a la que BASIC tiene acceso (hasta la dirección dada por la variable de sistema RAMTOP). Conserva variables de sistema UDG, P-RAMT, RASP y PIP. Devuelve el control al menú de presentación, pero no afecta al disco de silicio.
NEXT <i>a</i>	<ul style="list-style-type: none"> (i) Busca la variable de control <i>a</i>. (ii) Añade a su valor el valor del paso especificado en FOR. (iii) Si el paso es igual o mayor que 0 y el valor de <i>a</i> es mayor que el límite, o si el paso es menor que 0 y el valor de <i>a</i> es menor que el límite, salta a la sentencia de retorno del bucle. <p>Error 2 si no existe la variable <i>a</i>.</p> <p>Error 1 si la variable <i>a</i> no es la misma que la especificada en FOR.</p>
OPEN #	Sólo funciona con microunidades, etc.
OUT <i>m,n</i>	Envía el byte <i>n</i> a la puerta <i>m</i> . (Carga el par de registros bc con <i>m</i> y el registro a con <i>n</i> y ejecuta la instrucción out (c),a.)
	Error B a menos que $0 \leq m \leq 65535$ y $-255 \leq n \leq 255$.
OVER <i>n</i>	Controla la sobreimpresión para los caracteres que se escriba en lo sucesivo.
	Si $n=0$, los caracteres sustituyen a los que hubiera previamente en esa posición.
	Si $n=1$, los caracteres nuevos se mezclan con los antiguos para dar el color de la tinta dondequiera que uno de ellos (pero no ambos) tuviese dicho color, y el color del papel donde ambos fuesen papel o ambos tinta.
	Error K a menos que <i>n</i> sea 0 o 1.
PAPER <i>n</i>	Como INK, pero para controlar el color del papel (fondo).
PAUSE <i>n</i>	Detiene el programa y muestra la imagen durante <i>n</i> barridos del cuadro (a 50 barridos por segundo; 60 por segundo en EE.UU.), o hasta que se pulsa una tecla. Si $n=0$ la pausa no se cronometra, sino que dura hasta que se pulsa una tecla.
	Error B a menos que $0 \leq n \leq 65535$.
PLAY <i>f,f,f,f,...</i>	Interpreta hasta ocho cadenas (véase la Parte 19 de este capítulo) y las ejecuta simultáneamente. Las primeras tres cadenas son ejecutadas a través del altavoz del televisor y (opcionalmente) a través de la puerta de MIDI; todas las siguientes van a la puerta MIDI.

PLOT <i>c;m n</i>	<p>Dibuja un punto de tinta (supeditado a OVER y a INVERSE) en el pixel (<i>m,n</i>); establece en este punto la nueva 'posición actual' del cursor gráfico.</p> <p>A no ser que los elementos de color <i>c</i> especifiquen otra cosa, el color de la tinta de la posición de carácter que contiene al pixel se cambia por el color de tinta permanente actual, y los otros (color del papel, parpadeo y brillo) permanecen como estaban.</p> <p>Error B a menos que $0 \leq m \leq 255$ y $0 \leq n \leq 175$.</p>
POKE <i>m,n</i>	<p>Escribe el valor <i>n</i> en la posición de memoria cuya dirección es <i>m</i>.</p> <p>Error B a menos que $0 \leq m \leq 65535$ y $-255 \leq n \leq 255$.</p>
PRINT ...	<p>'...' es una secuencia de elementos de PRINT, separados entre sí por comas, signos de punto y coma o apóstrofes. La sentencia los escribe en el fichero de imagen (memoria de pantalla) para su salida por la pantalla.</p> <p>Un punto y coma entre dos elementos no produce ningún efecto (sirve sólo para separar los elementos); una coma produce el carácter de control de la 'coma'; un apóstrofo produce el 'retorno del carro' (equivalente a la pulsación de la tecla INTRO), que PRINT emite por defecto si la sentencia no termina en punto y coma, coma o apóstrofo).</p> <p>Un elemento de PRINT puede ser:</p> <ul style="list-style-type: none"> (i) Vacío; es decir, nada. (ii) Una expresión numérica. <p>Primero se escribe un signo menos si el valor es negativo. Supongamos que <i>x</i> es el valor absoluto de la expresión. Si $x \leq 10^{-5}$ o $x \geq 10^{13}$, el número se escribe en notación científica. La mantisa tiene hasta ocho dígitos (sin ceros por la derecha); el punto decimal (ausente si sólo hay un dígito) va detrás del primero. La parte del exponente es E, seguido de + o -, a continuación, de uno o dos dígitos. Si <i>x</i> no está en el margen mencionado, el número se escribe en notación decimal con hasta diez dígitos significativos y sin ceros por la derecha después del punto decimal.</p> (iii) Una expresión literal. <p>Las claves contenidas en la cadena son interpretadas, posiblemente con un espacio antes o después. Los caracteres de control producen su efecto de control. Los caracteres irreconocibles se convierten en ?.</p>

	(iv) AT m,n .
	Emite un carácter de control AT seguido por un byte para m (número de fila) y otro para n (número de columna).
	(v) TAB n .
	Emite un carácter de control TAB seguido por dos bytes para n (primero el byte menos significativo), que es tope de tabulación.
	(vi) Un elemento de color, que toma la forma de una sentencia PAPER, INK, FLASH, BRIGHT, INVERSE u OVER.
RANDOMIZE	Equivale a RANDOMIZE 0.
RANDOMIZE n	Establece la variable de sistema (llamada SEED) que se va a usar para generar el próximo valor de RND. Si $n < > 0$, se le da a SEED el valor n . Si $n = 0$, a SEED se le da el valor de otra variable de sistema, FRAMES, que lleva la cuenta de los barridos del cuadro exhibidos hasta ahora en la pantalla y que, por consiguiente, debería ser bastante aleatoria. Error B a menos que $0 \leq n \leq 65535$.
READ v_1, v_2, \dots, v_k	Asigna valores a las variables leyéndolos en las sucesivas expresiones de la lista de datos. Error C si una expresión es de tipo inadecuado para la variable a la que debería ser asignada. Error E si se ha agotado la lista de datos cuando todavía quedaban variables a las que asignar valor.
REM ...	No produce ningún efecto. '...' puede ser cualquier secuencia de caracteres que termine en 'retorno del carro' (INTRO). No se ejecutará ninguna sentencia que esté detrás de la palabra 'REM' en la misma línea; los signos de dos puntos <i>no</i> serán considerados separadores de sentencias.
RESTORE	Equivale a RESTORE 0.
RESTORE n	Dirige el «puntero» de datos hacia la primera sentencia DATA de la línea n . Si no existe esa línea (o si no contiene ninguna sentencia DATA), el puntero se dirige hacia la primera sentencia DATA posterior a la línea n . La siguiente sentencia READ empezará a leer a partir de ese punto.
RETURN	Lee en la pila de GO SUB la referencia a una sentencia y salta a la línea siguiente a ella. Error 7 cuando en la pila no hay referencia a ninguna sentencia. (Esto quiere decir que probablemente hay algún error en el programa; asegúrese de que todos los GO SUB están emparejados con un RETURN.)

RUN	Equivale a RUN 0.
RUN <i>n</i>	Ejecuta CLEAR y después GO TO <i>n</i> .
SAVE <i>f</i>	Graba el programa y las variables en un fichero al que da el nombre <i>f</i> . Error F si el nombre está vacío o consta de más de diez caracteres. Véase la Parte 20 de este capítulo.
SAVE <i>f</i> LINE <i>m</i>	Graba el programa y las variables de forma tal que, cuando más tarde se lo cargue, BASIC realizará un salto automático a la línea <i>m</i> .
SAVE <i>f</i> DATA ()	Graba la matriz numérica.
SAVE <i>f</i> DATA \$()	Graba la matriz de caracteres.
SAVE <i>f</i> CODE <i>m,n</i>	Graba <i>n</i> bytes a partir de la dirección <i>m</i> .
SAVE <i>f</i> SCREEN\$	Equivale a SAVE <i>f</i> CODE 16384,6912. Graba la imagen actual de la pantalla.
SAVE ! <i>f</i>	Como SAVE, pero sobre el disco de silicio. Véase Parte 20 de este capítulo.
SPECTRUM	Pasa de 128 BASIC a 48 BASIC, conservando el programa que pueda haber en la RAM. De 48 BASIC no se puede volver a 128 BASIC.
STOP	Detiene el programa y emite el informe 9. CONTINUE reanudará la ejecución del programa a partir de la sentencia siguiente.
VERIFY	Como LOAD, pero sin cargar en la RAM la información leída en la cinta, sino sólo comparándola con la que hay actualmente en la RAM. Error R si la comparación detecta alguna diferencia entre lo grabado en la cinta y lo almacenado en la RAM.

Parte 31

Programas de ejemplo

Programas:

Días
I Ching
Animales
Bandera
Hangman

En esta sección vamos a dar algunos programas de ejemplo que pueden serle de interés. Si desea ejecutar los programas más de una vez, no olvide grabarlos (con SAVE) permanentemente en una cinta, o temporalmente en el disco de silicio.

Días

El primero de estos programas pide una fecha (de este siglo), y después da el día de la semana correspondiente.

```
10 REM Convertir fecha en día de la semana
20 DIM d$(7,9): REM Días de la semana
30 FOR n=1 TO 7: READ d$(n): NEXT n
40 DIM m(12): REM Días que tiene cada mes
50 FOR n=1 TO 12: READ m(n): NEXT n
100 REM Captar fecha
110 INPUT "Dia? ";día
120 INPUT "Mes? ";mes
130 INPUT "Año (sólo siglo XX? ";año
140 IF año<1901 THEN PRINT "El siglo XX empieza en el 1901": GO TO 100
150 IF año>2000 THEN PRINT "El siglo XX termina en el 2000": GO TO 100
160 IF mes<1 THEN GO TO 210
170 IF mes>12 THEN GO TO 210
180 IF año/4-INT(año/4)=0 THEN LET m(2)=29: REM Bisiesto
190 IF día > m(mes) THEN PRINT "Ese mes solo tiene ";m(mes);" días.": GO TO 500
200 IF día>0 THEN GO TO 300
210 PRINT "No trates de engañarme. Dame una fecha real."
220 GO TO 500
300 REM Convertir fecha en número de días desde el principio del siglo
310 LET a=año-1901
320 LET b=365*a+INT(a/4): REM Número de días hasta el principio del año
```

```

330 FOR n=1 TO mes-1: REM Sumar meses anteriores
340 LET b=+m(n): NEXT n
350 LET b=b+dia
400 REM Convertir a día de la semana
410 LET b=b-7* INT (b/7)+1
420 PRINT dia;" / ";mes;" / ";año
430 PRINT" is ";d$(b)
500 LET m(2)=28: REM Restaurar febrero
510 INPUT "¿Otra vez? ",a$
520 IF a$="n" THEN GO TO 540
530 IF a$<> "N" THEN GO TO 100
1000 REM Días de la semana
1010 DATA "lunes", "martes", "miercoles"
1020 DATA "jueves", "viernes", "sabado", "domingo"
1100 REM Días de los meses
1110 DATA 31, 28, 31, 30, 31, 30
1120 DATA 31, 31, 30, 31, 30, 31

```

I Ching

El siguiente programa lanza monedas para el 'I Ching'. Las formas que produce están «tumbadas», pero los resultados son aceptables:

```

5 RANDOMIZE
10 FOR m=1 TO 6: REM Seis lanzamientos
20 LET c=0: REM Inicializar total a 0
30 FOR n=1 TO 3: REM Para 3 monedas
40 LET c=c+2+INT (2*RND)
50 NEXT n
60 PRINT " ";
70 FOR n=1 TO 2: REM Primero para el hexagrama, segundo para los cambios
80 PRINT " --- ";
90 IF c=7 THEN PRINT "- ";
100 IF c=8 THEN PRINT " ";
110 IF c=6 THEN PRINT "X";: LET c=7
120 IF c=9 THEN PRINT "0";: LET c=8
130 PRINT " --- ";
140 NEXT n
150 PRINT
160 INPUT a$
170 NEXT m: NEW

```

Cuando termine de copiar el programa, ejecútelo (RUN) y después pulse cinco veces **INTRO** para obtener los dos hexagramas. Búsquelos en el 'Libro chino de los cambios'. El texto le describirá una situación y las acciones apropiadas para ella, y usted

debe meditar profundamente para encontrar la analogía entre esto y su propia vida. Pulse **INTRO** por una sexta vez y el programa se borrará a sí mismo (para que usted no lo utilice frívolamente).

Muchas personas piensan que los textos están más acertados que si se debieran a mera casualidad; éste puede o no ser el caso con su +2. *En general, los ordenadores son criaturas bastante ateas.*

Animales

El siguiente programa le permitirá enseñarle zoología al +2. Usted piensa en un animal y el ordenador trata de adivinar cuál es haciéndole preguntas que pueden ser contestadas con 'sí' o 'no'. Si la máquina no ha oído hablar nunca de su animal, entonces le pide que escriba una pregunta que pueda utilizar para distinguirlo de los que ya conoce.

```
5 REM Animales
10 LET nq=100: REM Número de preguntas y de animales
15 DIM q$(nq,50): DIM a(nq,2): DIM r$(1)
20 LET qf=8
30 FOR n=1 TO qf/2-1
40 READ q$(n): READ a(n,1): READ a(n,2)
50 NEXT n
60 FOR n=n TO qf-1
70 READ q$(n): NEXT n

100 REM Empezar el juego
110 PRINT "Piensa en un animal.", "Pulsa una tecla para continuar."
120 PAUSE 0
130 LET c=1: REM Empezar con la primera pregunta
140 IF a(c,1)=0 THEN GO TO 300
150 LET p=q$(c): GO SUB 910
160 PRINT "?": GO SUB 1000
170 LET i=1: IF r$="s" THEN GO TO 210
180 IF r$="S" THEN GO TO 210
190 LET i=2: IF e$="n" THEN GO TO 210
200 IF e$<>"N" THEN GO TO 150
210 LET c=a(c,i): GO TO 140

300 REM animal
310 PRINT "¿Estas pensando en"
320 LET p=q$(c): GO SUB 900: PRINT "?"
330 GO SUB 1000
340 IF r$="s" THEN GO TO 400
350 IF r$="S" THEN GO TO 400
360 IF r$="n" THEN GO TO 500
370 IF r$="N" THEN GO TO 500
380 PRINT "¡Contestame correctamente cuando"; "te pregunto!": GO TO 300
```

```

400 REM Acertado
410 PRINT "¡Lo sabia!": GO TO 800
500 REM Otro animal
510 IF qf>nq-1 THEN PRINT "No dudo que tu animal sera muy","interesante, pero
    no me queda","memoria para el.": GO TO 800
520 LET a$(qf)=q$(c): REM Retirar animal antiguo
530 PRINT "Entonces ¿que es?": INPUT q$(qf+10)
540 PRINT "Dime con que pregunta distin-","guirias entre"
550 LET p$=q$(qf): GO SUB 900: PRINT " y"
560 LET p$=q$(qf+1): GO SUB 900: PRINT " "
570 INPUT s$: LET b=LEN s$
580 IF s$(b)="?" THEN LET b=b-1
590 LET q$(c)=s$(TO b): REM Insertar pregunta
600 PRINT "¿Cual es la respuesta para"
610 LET p$=q$(qf+1): GO SUB 900: PRINT "?"
620 GO SUB 1000
630 LET i=1: LET io=2: REM Respuestas para animales nuevo y antiguo
640 IF r$="s" THEN GO TO 700
650 IF r$="S" THEN GO TO 700
660 LET i=2: LET io=1
670 IF r$="n" THEN GO TO 700
680 IF r$="N" THEN GO TO 700
690 PRINT "¡Eso no vale!": GO TO 600
700 REM Actualizar respuestas
710 LET a(c,i)=qf+1: LET a(c,io)=qf
720 LET qf=qf+2: REM Siguiente animal libre
730 PRINT "Eso no lo sabia."
800 REM Otra vez?
810 PRINT "¿Quieres jugar otra vez?": GO SUB 1000
820 IF r$="s" THEN GO TO 100
830 IF r$="S" THEN GO TO 100
900 REM Escribir sin espacios por la derecha
905 PRINT " ";
910 FOR n=50 TO 1 STEP -1
920 IF p$(n)<>" " THEN GO TO 940
930 NEXT n
940 PRINT p$(TO n);: RETURN
1000 REM Captar respuesta
1010 INPUT r$: IF r$="" THEN RETURN
1020 LET r$=r$(1): RETURN
2000 REM Animales iniciales
2010 DATA "¿Vive en el mar",4,2
2020 DATA "¿Vive en la tierra",3,5
2030 DATA "¿Come hormigas",6,7
2040 DATA "una ballena", "un buitre", "un oso", "una hormiga"

```

Bandera

Este programa dibuja la bandera británica:

```
5 REM Bandera británica
10 LET r=2: LET w=7: LET b=1
20 BORDER 0: PAPER b: INK w: CLS
30 REM Parte inferior de la pantalla en negro
40 INVERSE 1
50 FOR n=40 TO 0 STEP -8
60 PLOT PAPER 0:7,n: DRAW PAPER 0;241,0
70 NEXT n: INVERSE 0
100 REM Dibujar zonas blancas
105 REM San Jorge
110 FOR n=0 TO 7
120 PLOT 104+n,175: DRAW 0,-35
130 PLOT 151-n,175: DRAW 0,-35
140 PLOT 151-n,48: DRAW 0,35
150 PLOT 104+n,48: DRAW 0,35
160 NEXT n
200 FOR n=0 TO 11
210 PLOT 0,139-n: DRAW 111,0
220 PLOT 255,139-n: DRAW -111,0
230 PLOT 255,84+n: DRAW -111,0
240 PLOT 0,84+n: DRAW 111,0
250 NEXT n
300 REM San Andrés
310 FOR n=0 TO 35
320 PLOT 1+2*n,175-n: DRAW 32,0
330 PLOT 224-2*n,175-n: DRAW 16,0
340 PLOT 254-2*n,48+n: DRAW -32,0
350 PLOT 17+2*n,48+n: DRAW 16,0
360 NEXT n
370 FOR n=0 TO 19
380 PLOT 185+2*n,140+n: DRAW 32,0
390 PLOT 200+2*n,83-n: DRAW 16,0
400 PLOT 39-2*n,83-n: DRAW 32,0
410 PLOT 54-2*n,140+n: DRAW -16,0
420 NEXT n
425 REM Completar
430 FOR n=0 TO 15
440 PLOT 255,160+n: DRAW 2*n-30,0
450 PLOT 0,63-n: DRAW 31-2*n,0
460 NEXT n
470 FOR n=0 TO 7
480 PLOT 0,160+n: DRAW 14-2*n,0
```

```

485 PLOT 255,63-n: DRAW 2*n-15,0
490 NEXT n
500 REM Bandas rojas
510 INVERSE 1
520 REM San Jorge
530 FOR n=96 TO 120 STEP 8
540 PLOT PAPER r;7,n: DRAW PAPER r;241,0
550 NEXT n
560 FOR n=112 TO 136 STEP 8
570 PLOT PAPER r;n,168: DRAW PAPER r;0,-113
580 NEXT n
600 REM San Patricio
610 PLOT PAPER r;170,140: DRAW PAPER r;70,35
620 PLOT PAPER r;179,140: DRAW PAPER r;70,35
630 PLOT PAPER r;199,83: DRAW PAPER r;56,-28
640 PLOT PAPER r;184,83: DRAW PAPER r;70,-35
650 PLOT PAPER r;86,83: DRAW PAPER r;-70,-35
660 PLOT PAPER r;72,83: DRAW PAPER r;-70,-35
670 PLOT PAPER r;56,140: DRAW PAPER r;-56,28
680 PLOT PAPER r;71,140: DRAW PAPER r;-70,35
690 INVERSE 0: PAPER 0: INK 7

```

Ahora puede tratar de dibujar la bandera española o la de su comunidad autónoma. En general las banderas tricolores son bastante fáciles, aunque algunos colores (por ejemplo, el naranja) pueden presentar dificultades. Cuando termine de dibujar una bandera puede grabarla en el disco de silicio con la orden `SAVE ! "bandera1" SCREEN$,` después dibujar otra bandera diferente y almacenarla con nombre diferente, etc. En el disco de silicio hay espacio para unas diez pantallas diferentes.

Hangman

A continuación le ofrecemos una versión de este clásico juego. Por si no lo conoce, podemos decirle que un jugador introduce una palabra y el otro trata de adivinarla:

```

5 REM Hangman
10 REM Preparar pantalla
20 INK 0: PAPER 7: CLS
30 LET x=240: GO SUB 1000: REM Dibujar hombre
40 PLOT 238,128: DRAW 4,0: REM Boca
100 REM Preparar palabra
110 INPUT p$: REM Palabra que hay que adivinar
120 LET b=LEN p$: LET v$=""
130 FOR n=2 TO b: LET v$=v$+" "
140 NEXT n: REM v$=palabra adivinada hasta ahora
150 LET c=0: LET d=0: REM Contador de intentos y aciertos

```

```

160 FOR n=0 TO b-1
170 PRINT AT 20,n;"-";
180 NEXT n: REM Escribir guiones en vez de letras
200 INPUT "Adivine una letra: ";g$
210 IF g$="" THEN GO TO 200
220 LET g$=g$(1): REM Sólo primera letra
230 PRINT AT 0,c;g$
240 LET c=c+1: LET u$=v$
250 FOR n=1 TO b: REM Actualizar palabra
260 IF p$(n)=g$ THEN LET v$(n)=g$
270 NEXT n
280 PRINT AT 19,0;v$
290 IF v$=p$ THEN GO TO 500: REM Palabra acertada
300 IF v$<>u$ THEN GO TO 200: REM Letra acertada
400 REM Dibujar siguiente trozo del patíbulo
410 IF d=8 THEN GO TO 600: REM Colgarlo
420 LET d=d+1
430 READ x0,y0,x,y
440 PLOT x0,y0: DRAW x,y
450 GO TO 200
500 REM Liberarlo
510 OVER 1: REM Borrar hombre
520 LET x=240: GO SUB 1000
530 PLOT 238,128: DRAW 4,0: REM Boca
540 OVER 0: REM Redibujar hombre
550 LET x=146: GO SUB 1000
560 PLOT 143,129: DRAW 6,0: PI/2: REM Sonrisa
570 GO TO 800
600 REM Colgar hombre
610 OVER 1: REM Borrar suelo
620 PLOT 255,65: DRAW -48,0
630 DRAW 0,-48: REM Abrir trampilla
640 PLOT 238,128: DRAW 4,0: REM Borrar boca
650 REM Mover miembros
655 REM Brazos
660 PLOT 255,117: DRAW -15,-15: DRAW -15,15
670 OVER 0
680 PLOT 236,81: DRAW 4,21: DRAW 4,-21
690 OVER 1: REM Piernas
700 PLOT 255,66: DRAW -15,15: DRAW -15,-15
710 OVER 0
720 PLOT 236,60: DRAW 4,21: DRAW 4,-21
730 PLOT 237,127: DRAW 6,0,-PI/2: REM Triste
740 PRINT AT 19,0;p$
800 INPUT "¿Otra vez? ";a$
810 IF a$="" THEN GO TO 850

```

```
820 LET a$=a$(1)
830 IF a$="n" THEN STOP
840 IF a$(1)="N" THEN STOP
850 RESTORE: GO TO 5
1000 REM Dibujar hombre en la columna x
1010 REM Cabeza
1020 CIRCLE x,132,8
1030 PLOT x+4,134: PLOT x-4,134: PLOT x,131
1040 REM Cuerpo
1050 PLOT x,123: DRAW 0,-20
1055 PLOT x,101: DRAW 0,-19
1060 REM Piernas
1070 PLOT x-15,66: DRAW 15,15: DRAW 15,-15
1080 REM Brazos
1090 PLOT x-15,117: DRAW 15,-15: DRAW 15,15
1100 RETURN
2000 DATA 120,65,135,0,184,65,0,91
2010 DATA 168,65,16,184,81,16,-16
2020 DATA 184,156,68,0,184,140,16,16
2030 DATA 204,156,-20,-20,240,156,0,-16
```

Parte 32

Binario y hexadecimal

Temas tratados:

Sistemas de numeración
Bits y bytes

Esta sección describe la forma en la que los ordenadores manejan internamente los números, utilizando el sistema binario.

En la mayor parte de los idiomas europeos, los nombres de los números reflejan una relación fundamental con el número diez. En castellano, por ejemplo, solamente son excepcionales, en este sentido, el 'once', el 'doce', el 'trece', el 'catorce' y el 'quince'. A partir de éste la regularidad es manifiesta:

... dieciséis, diecisiete, dieciocho, diecinueve
veinte, veintiuno, veintidós, ..., veintinueve
treinta, treinta y uno, treinta y dos, ..., treinta y nueve
cuarenta, cuarenta y uno, cuarenta y dos, ..., cuarenta y nueve
...

Esto facilita el uso sistemático de los números. La razón por la que el número diez desempeña este papel fundamental es el hecho de que tenemos diez dedos en las manos. Este sistema de numeración se llama *decimal*.

Los ordenadores, en lugar del sistema decimal (de *base diez*), utilizan el *hexadecimal* (abreviadamente, *hex*), cuya base es el número dieciséis. Ya que en nuestro sistema numérico decimal sólo tenemos diez dígitos, necesitamos otros seis para representar los números en hexadecimal. Las seis cifras adicionales son A, B, C, D, E y F. ¿Y qué viene después de la F? Bien, cuando se nos acaban los dígitos en el sistema decimal, volvemos a empezar por el 0 poniéndole un 1 delante (10); lo mismo ocurre en el sistema hexadecimal: el número que sigue a F es 10 (que es el 16 decimal), el siguiente es 11 (17 decimal), etc.

Veamos una tabla que da la equivalencia entre los dos sistemas:

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4

Decimal	Hexadecimal
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
	...
25	19
26	1A
27	1B
	...
31	1F
32	20
33	21
	...
158	9E
159	9F
160	A0
161	A1
	...
255	FF
256	100
etcétera.	

Si un número está escrito en notación hexadecimal, se puede evidenciar este hecho escribiendo una 'h' como sufijo. Por ejemplo, el número 256 decimal es 100h en hexadecimal.

Puede usted estar preguntándose qué tiene que ver todo esto con los ordenadores. De hecho, los ordenadores se comportan como si sólo tuvieran dos dígitos, representados por una tensión eléctrica baja ('apagado', 0) y una tensión alta ('encendido', 1). Este sistema de numeración en que sólo se dispone de dos dígitos se denomina *binario*. Los dos dígitos se llaman *bits* (de *binary digit*, dígito *binario*). Así pues, un bit es un estado que puede tener dos valores: 0 o 1.

La siguiente tabla da las versiones decimal, hexadecimal y binaria de los primeros números:

Decimal	Hex	Binario
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	10000
17	11	10001
etcétera.		

Es habitual rellenar los números binarios con ceros, de forma que siempre contengan al menos cuatro bits; por ejemplo, 0000, 0001, 0010, 0011 (que representan los números 0 al 3 en decimal).

La conversión entre binario y hexadecimal es muy fácil (consultando la tabla anterior):

Para convertir un número binario en hexadecimal, divida el binario en grupos de cuatro bits (comenzando por la derecha del número) y convierta cada grupo en su correspondiente dígito hexadecimal. Finalmente, una los dígitos hexadecimales para formar el número hex completo. Por ejemplo, para pasar el número binario 10110100 a hexadecimal, convierta el primer grupo (por la derecha) de cuatro bits (0100) en el hexadecimal 4; después convierta el siguiente grupo (1011) en el hexadecimal B, únalos y obtendrá el número hexadecimal completo, B4h. Si el número binario tiene más de ocho bits, puede continuar convirtiendo cada grupo de cuatro bits en un dígito hex. Por ejemplo, el número binario 11101011110000 corresponde al 3AF0h.

Para convertir un número hexadecimal en binario, transforme cada dígito hex en cuatro bits y a luego una los bits para formar un número binario completo. Por ejemplo, para pasar F3h a binario, convierta primero a 3 en su correspondiente 0011 binario (recuerde que debe rellenar con ceros por la izquierda para que el número binario conste de cuatro

bits); después transforme F en su correspondiente 1111 binario, únalos y tendrá el número binario completo, 11110011.

A pesar de que los ordenadores usan un sistema binario puro, los humanos a menudo escribimos los números que vamos a suministrar al ordenador utilizando la notación hexadecimal; después de todo, el número 3AF0h, por ejemplo, es más fácil de leer y recordar que su equivalente, 0011101011110000, en notación binaria de dieciséis bits.

Los bits que se encuentran dentro del ordenador están agrupados normalmente en bloques de ocho, es decir en *bytes*. Un byte puede representar cualquier número decimal del 0 al 255 (11111111 binario o FFh).

Se puede agrupar dos bytes para formar lo que se llama técnicamente una *palabra*. Una palabra se puede expresar mediante dieciséis bits o cuatro dígitos hex y representa un número decimal del 0 al 65535 (1111111111111111 binario o FFFFh).

Un byte se compone *siempre* de ocho bits, pero la longitud de las palabras varía de un ordenador a otro.

La notación BIN usada en la Parte 14 de este capítulo proporciona un medio de introducir números binarios en el +2. Por ejemplo, BIN 10 representa el 4 decimal, BIN 111 representa el 7 decimal, BIN 11111111 representa el 255 decimal, etc.

En esta notación, después de BIN sólo podemos poner ceros y unos, de forma que el número sólo puede ser un entero no negativo. Por ejemplo, no se puede usar BIN -11 para representar el -3 decimal; lo que sí podemos hacer es escribir -BIN 11. El número tampoco puede ser mayor que el decimal 65535; es decir, no puede constar de más de dieciséis bits. Si rellenamos un número binario con ceros por la izquierda, por ejemplo, BIN 00000001, BIN los ignora y trata el número como si fuera BIN 1.

Utilización de la calculadora

Temas tratados en este capítulo:

Selección de la calculadora

Introducción de números

Resultado actual

Uso de las funciones matemáticas incorporadas

Edición de la pantalla

Asignación de variables

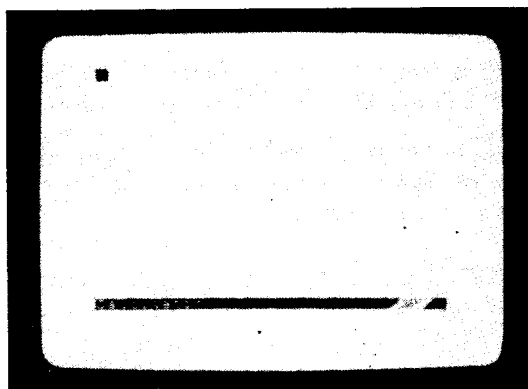
Salida de la calculadora

El +2 puede ser usado también como calculadora, con todas las funciones habituales de las calculadoras más potentes.

Para utilizar la calculadora, abra el menú de presentación y seleccione la opción Calculadora. (Si no sabe cómo seleccionar una opción del menú, consulte el capítulo 2.)

La calculadora puede ser seleccionada en cuanto se enciende el +2. O bien, si ya se está trabajando con 128 BASIC, se puede elegir la opción Salida del menú de edición para volver al menú de presentación y entonces seleccionar la opción Calculadora. El programa de BASIC con el que se estuviera trabajando al realizar esta selección será conservado en la memoria y reestablecido cuando se abandone esta opción.

Cuando se ha seleccionado la opción Calculadora la pantalla cambia a:



y la calculadora del +2 queda preparada para aceptar instrucciones. Escriba:

6+4

Tan pronto como pulse **INTRO**, aparecerá la respuesta: 10. (Observe que no se pulsa la tecla **=** como se haría en una calculadora corriente.)

Verá que el cursor está situado a la derecha de la respuesta, que es un *resultado actual* (como en una calculadora normal). Esto significa que usted puede sencillamente escribir la siguiente operación para que sea aplicada al 'resultado actual' (sin tener que escribirlo de nuevo). Así, con el cursor todavía a la derecha del 10, escriba:

/5

y obtendrá la respuesta: 2. Ahora escriba:

*PI

La calculadora responde con 6.2831853. El +2 ha utilizado la función intrínseca π ; todo lo que usted ha tenido que hacer es escribir PI. Esto es válido para *todas* las funciones matemáticas del +2. Para comprobarlo, escriba:

*ATN 60

y obtendrá 9.7648943. También es posible 'editar' el contenido de la pantalla. Desplace el cursor (con la tecla **←**) hasta el principio de la línea y escriba INT, de forma que la línea quede así:

INT 9.7648943

Cuando pulse **INTRO**, obtendrá la respuesta: 9. Esto también demuestra que el +2 puede escribir el valor de una expresión aunque ello no implique la realización de ningún cálculo. Como ejemplo, pulse **INTRO** y después escriba:

1E6

y obtendrá el valor de esa expresión. Fíjese en que antes de escribir 1E6 ha sido necesario pulsar **INTRO** para indicarle al ordenador que íbamos a empezar un cálculo nuevo.

Una característica muy útil de la calculadora del +2 es que permite asignar valores a variables para usarlos en cálculos posteriores. Esto requiere el empleo de la sentencia LET (como en BASIC). Escriba lo siguiente:

LET x=10

(Es necesario pulsar **INTRO** *dos veces* para que el +2 acepte la asignación de la variable.) Ahora, para comprobar que el +2 reconoce la variable x, escriba lo siguiente:

x+90

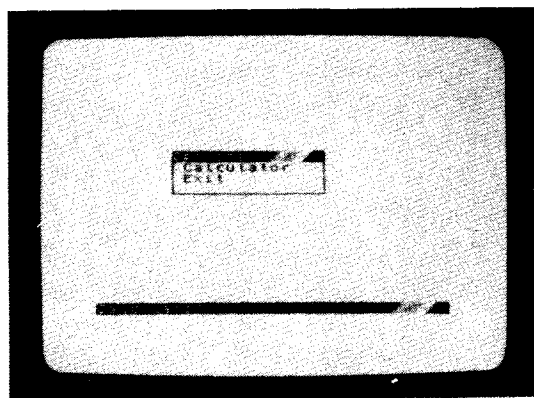
y a continuación:

+x*x

Si está usted usando la calculadora mientras tiene almacenado en la memoria un programa de BASIC, debe elegir las variables para la calculadora de forma que no entren en conflicto con las utilizadas por el programa.

No se puede usar las palabras clave de BASIC como nombres de variables.

Cuando termine de trabajar con la calculadora, pulse la tecla **EDIT**. La pantalla cambiará a:



Seleccione la opción Salida para volver al menú de presentación. Si estaba trabajando en un programa de 128 BASIC antes de empezar a usar la calculadora, puede volver al programa seleccionando la opción 128 BASIC. (Si desea seguir con la calculadora, seleccione la opción Calculadora.)



Conexión de periféricos

Temas tratados en este capítulo:

- Joystick(s)
- Monitor
- Amplificador
- Impresora
- Dispositivos serie
- Dispositivo MIDI
- Teclado numérico
- 'Interface One' y microunidades
- Otros dispositivos

El +2 puede ser conectado a una amplia gama de dispositivos adicionales (*periféricos*), tales como joysticks, monitor, amplificador, etc. En este capítulo vamos a dar toda la información necesaria para conectarlos.

Joysticks

Con el +2 sólo se puede utilizar los joysticks Sinclair **SJS1**. Cualquier otro tipo de joystick (por ejemplo, Atari) *no* operará directamente, ya que su clavija de conexión está cableada de forma diferente.

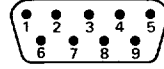
En el lateral izquierdo del +2 se encuentran los dos zócalos para joystick. En general, con los programas de juego sólo se usa el **JOYSTICK1**.

Si un programa determinado le permite elegir el tipo de joystick, seleccione la opción 'Interface Two' (o 'Sinclair'), ya que los circuitos de joystick del +2 están diseñados para funcionar exactamente igual que el Interface Two.

Es importante no conectar ni desconectar el joystick cuando el +2 está encendido.

Zócalos JOYSTICK1 y JOYSTICK2

Patilla	Función
1	no utilizada
2	masa
3	no utilizada
4	disparo
5	arriba
6	derecha
7	izquierda
8	masa
9	abajo



Monitor

Además de un televisor (o en su lugar), el +2 puede utilizar un monitor monocromático o de color. Si el monitor que usted desea usar no está anunciado como compatible con el Sinclair +2 (o el Spectrum 128), es muy probable que tenga que adquirir un cable especial para él; en tal caso, consulte a su distribuidor.

Observe que si el monitor no acepta la señal *BRIGHT*, sólo podrá mostrar 8 de los 16 colores disponibles.

Zócalo RGB



Patilla	Señal	Nivel
1	PAL compuesta	1.2V pico a pico/75 ohms
2	0 V	—
3	brillo	TTL
4	sincr. compuesta	TTL
5	sincr. vertical	TTL
6	verde	TTL
7	rojo	TTL
8	azul	TTL

Al utilizar un monitor se ha de prever qué se desea hacer con el sonido. Si el monitor tiene entrada de audio, se la puede conectar al zócalo **SONIDO** de la parte posterior del +2; si el monitor no admite sonido, se necesitará un amplificador externo (véase el apartado siguiente).

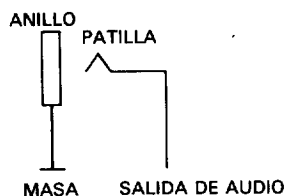
Amplificador

El +2 normalmente reproduce el sonido a través del aparato de televisión al que está conectado. Sin embargo, si se está usando un monitor o si se desea grabar o amplificar el sonido, se puede tomar la señal de audio del zócalo **SONIDO** de la parte posterior del +2. (Es una clavija monoaural de 3.5 mm; la señal es de 200 mV pico a pico sobre impedancia de 5 k Ω .) Cuando se utiliza un amplificador, conviene tener en cuenta que las señales de 'carga' y 'grabación' del magnetófono también son enviadas a la salida **SONIDO**; por consiguiente, se debe bajar el mando de volumen del amplificador al mínimo mientras se efectúa estas operaciones.

Otro detalle que se debe observar es que el nivel de sonido producido por BEEP es equivalente al de los tres canales de PLAY funcionando al mismo tiempo. En la práctica, esto significa que BEEP sonará bastante más alto que PLAY, lo cual puede causar problemas si los niveles de sonido son críticos.

No se debe conectar ni desconectar nada en el zócalo **SONIDO** mientras el +2 esté encendido.

Zócalo **SOUND**:



Impresora (y otros dispositivos serie)

El +2 puede ser utilizado con la mayor parte de las impresoras 'serie' que cumplan la norma RS232. Recomendamos a los usuarios no expertos que se abstengan de experimentar con las conexiones del interfaz. Pida a su distribuidor el cable necesario para interconectar el ordenador y la impresora; atégase a las instrucciones del fabricante de la impresora en lo referente a conexión y funcionamiento.

La impresora se conecta en el zócalo **RS232/MIDI** que está en la parte posterior del +2.

Cualquier otro dispositivo serie necesitará también un cable de tipo 'serie para Spectrum +2'; pídaselo así a su distribuidor. Por si desea preparar usted mismo el cable, las conexiones son las siguientes:

Patilla	Función
1	GND
2	TXD
3	RXD
4	DTR
5	CTS
6	+12 V

Zócalo RS232:

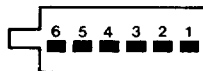


Dispositivo MIDI

Aunque la puerta para el **MIDI** (*Musical Instrument Digital Interface*, 'interfaz digital para instrumentos musicales') comparte con el RS232 el mismo zócalo, requiere un cable diferente (que usted puede conseguir de su distribuidor). El otro extremo del cable debe ser conectado al zócalo **MIDI IN** del sintetizador. El +2 no está preparado para recibir datos del MIDI, sino que sólo puede actuar como generador. La puerta MIDI no requiere ninguna preparación (salvo las órdenes incluidas en **PLAY** para encenderlo).

El empleo del interfaz MIDI no afecta a la velocidad de transmisión del RS232.

Zócalo MIDI:



Patilla	Función
1	Retorno
2	no utilizada
3	no utilizada
4	no utilizada
5	Salida de datos
6	no utilizada

Teclado numérico

El teclado numérico permite el acceso a una amplia gama de recursos de edición, tales como 'desplazamiento por página', 'borrar palabras' y 'borrar hasta el final de la línea'. También puede ser empleado teclado de tipo 'calculadora'.

El teclado numérico se conecta en el zócalo **TECLADO** de la parte posterior del +2.

Interface One y microunidades

El +2 puede funcionar con el 'Interface One' y con microunidades. Con estos dispositivos se suministra instrucciones completas.

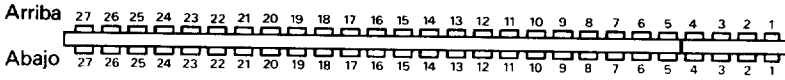
El 'Interface One' y las microunidades se conectan en el zócalo **EXPANSION E/S** de la parte posterior del +2.

Otros dispositivos

El +2 puede ser conectado a una gama muy amplia de periféricos a través del zócalo **EXPANSION E/S** de la parte posterior del aparato. Aunque este zócalo es muy parecido al del antiguo Spectrum de 48K, no se puede garantizar que un dispositivo que funcione correctamente con éste lo haga igual con el +2. Así pues, antes de comprar cualquier dispositivo de expansión, debe usted cerciorarse de que funciona con el +2.

¡ATENCIÓN! Es muy peligroso conectar o desconectar cualquier dispositivo en el zócalo **EXPANSION E/S** estando el +2 encendido. Si lo hace, es probable que estropee tanto el ordenador como el dispositivo.

Zócalo EXPANSION E/S:



Patilla	Fila superior	Fila inferior
1	A15	A14
2	A13	A12
3	D7	+5 V
4	no utilizada	+9 V
5	D0	0 V
6	D1	0 V
7	D2	CK
8	D6	A0
9	D5	A2
10	D3	A2
11	D4	A3
12	<u>INT</u>	IORQGE
13	<u>NMI</u>	0 V
14	<u>HALT</u>	no utilizada
15	<u>MREQ</u>	no utilizada
16	<u>IORQ</u>	no utilizada
17	<u>RD</u>	no utilizada
18	<u>WR</u>	<u>BUSRQ</u>
19	<u>-5 V</u>	RESET
20	<u>WAIT</u>	A7
21	+12 V	A6
22	-12 V	A5
23	<u>M1</u>	<u>A4</u>
24	<u>RFSH</u>	ROMCS
25	A8	BUSACK
26	A10	A9
27	no utilizada	A11

Índice

A

ABS 72, 205
ACS 82, 205
Altavoces 131
Amplificador 131, 235
AND 91, 205
Anidamiento 51
Animación 148
Apóstrofo 43, 214
ASN 82, 205
AT 105, 107, 155, 173, 215
ATN 82, 205
Atributos 114, 116, 165
ATTR 116, 119, 205
Auriculares 131

B

BASIC 17, 19, 27, 33, 176, 231
BEEP 132, 209, 235
BIN 98, 99, 205, 228
Binario 99, 225
Bit 226
BLOQ MAYS, tecla 29, 34
BORDER 118, 209
BORR, tecla 22
BREAK, tecla 36, 154, 192
BRIGHT 115, 209
Brillo 114
Bucles 49
Byte 99, 163, 227

C

Cable de antena 1, 2
Cadena vacía 64, 106, 129, 145, 204
Calculadora 229
Caracteres 95, 99, 181
 de control 99, 181
Carga de programas 13, 15, 142, 143
Carta de ajuste 5
CAT 148, 209
CHR\$ 95, 110, 205

Cinta de cassette 141, 145, 146
CIRCLE 123, 209
Circunferencias 79, 123
CLEAR 169, 177, 209
CLOSE 209
CLS 47, 107, 209
CODE 95, 146, 179, 195, 206
Código de máquina 177
Color 5, 113, 125, 173, 175, 214
Coma 43, 214
Comillas 44, 64, 65
Coordenada(s) 106, 121
 x 106, 121
 y 106, 121
COPY 154, 210
CONTINUE 43, 45, 174, 187, 209
Contraste 115
COS 80, 206
Cursor 19, 28, 32, 36, 64, 174

D

DATA 57, 146, 173, 192, 209
Decimal 225
DEF 73, 194, 210
Desembalaje 1
Detención de un programa 42, 45, 48
DIM 87, 210
Dimensionar 87, 210
Disco de silicio 28, 148
Disección 89, 204
Dos puntos 43, 214
DRAW 122, 210

E

Edición 19, 23, 25, 32, 36
EDIT, tecla 10, 20, 41, 231
Eje x 81
Eje y 81
Encendido 5
Ensamblador 177, 181
ERASE 148, 210
Error de sintaxis 32

EXP 77, 206
EXPANSION E/S, zócalo 161, 237, 238
Exponentes 63, 77
Expresiones literales 44, 61, 62, 89, 101, 168
 aritméticas 61, 72, 77, 208
 lógicas 91
 matemáticas 61, 72, 74, 208
 numéricas 63, 70, 167, 203
EXTRA, tecla 30, 35

F

FLASH 115, 210
FN 73, 194, 206
FOR 50, 187, 210
FORMAT 153, 196, 211
Funciones 69, 72, 205
 logarímicamente 79
 trigonométricas 80

G

GO SUB 55, 192, 211
GO TO 25, 42, 44, 211
Grabación de un programa 141, 144
Grados 82
Gráficos 96, 121
 definibles por el usuario 31, 35, 97, 102, 125
GRAF, tecla 31, 35, 96
Hardware 164, 199
Hexadecimal 225

I

IF 47, 91, 211
Impresora 22, 153, 235
IN 159, 206
INK 115, 211
INKEY\$ 129, 206
INPUT 42, 107, 211
Instalación 2, 3
Instrucciones 33
INT 72, 206

Interface
 one 237
 two 233
INTRO, tecla 41, 130
INVERSE 116, 123, 212

J

Joysticks 160, 233
 zócalo 233, 234

L

LEFT\$ 74
LEN 69, 206
LET 40, 61, 212, 230
LINE 109, 145, 179
LIST 36, 39, 212
Listado 22, 41
LLIST 153
LN 79, 206
LOAD 144-147, 151, 179
LPRINT 153, 175

M

Magnetófono 14, 16, 141
Matrices 87, 146, 167, 168, 203
MAYUSC, tecla 28-30, 32, 129
Mensajes 189
 de error 189
Menús 9-11, 13, 15
Memoria 159, 163, 169, 199
MERGE 145, 148, 152, 195
Microprocesador Z80 177, 181, 199
Microunidades 157, 166, 237
MIDI 139, 161, 213, 236
MID\$ 74
Modo
 C 29
 E 30
 G 31
 K 28, 32
 L 29
MOVE 212

Movimiento 127
Música 131, 137

N

NEW 41, 169, 213
NEXT 50, 187, 213
NOT 91, 206
Números aleatorios 83
 de línea 20, 23, 39

P

OPEN 213
Operadores de relación 48, 208
OR 91, 206
Órdenes 33
OUT 159, 213
OVER 116, 123, 213

P

Palabras clave 39, 63, 231
Pantalla 7, 22, 24, 32, 37, 106, 147
PAPER 115, 213
Paréntesis 62, 67, 73, 146, 205
Parpadeo 114
PAUSE 127, 213
PEEK 99, 128, 164, 171, 206
Periféricos 157, 233
PI 79, 123, 206
Pila 55, 169
Pixel 106, 114, 121, 124
PLAY 28, 131, 139, 195, 213, 235
PLOT 121, 214
POINT 124, 206
POKE 99, 110, 164, 171, 214
PRINT 39, 43, 63, 214
Procesador 177
Programas 13, 15, 142
Pseudoaleatorios 84
Puertas E/S 159
Punto y coma 43, 214

R

Radianes 82
Raices 77
Raíz cuadrada 72
RAM 159, 163, 169, 170, 199
RAMTOP 169, 176, 194, 209
RANDOMIZE 83, 84, 174, 215
READ 57, 192, 215
Redes 157
Redondeo de números 72, 73
Reinicialización 14
REM 42, 215
Renumerar 21, 41, 172
RESET, botón 9, 14
RESTORE 57, 215
RETURN 55, 191, 215
RGB, zócalo 234
RIGHT\$ 74
RND 83, 174, 207
ROM 159, 163, 170, 199
RS232 153, 157, 161, 235
RUN 24, 43, 44, 144, 216

S

SAVE 141, 144, 145, 146, 149, 179
SCREEN\$ 105, 147, 207
Scroll 36, 108, 111, 175
SGN 71, 207
Signo 72
SIMB, tecla 28, 34
SIN 80, 121, 207
Sintonización TV 5
Sobreimpresión 117, 119
Sonido 131, 200, 235
SOUND, zócalo 131, 235
SPECTRUM 28, 171, 216
SQR 72, 207
STEP 50, 210
STR\$ 70, 207
STOP 48, 191, 216
Subcadena 65, 204
Subíndice 87, 190

Subrutina 55

Subteclado numérico 157, 161, 237

T

TAB 107, 155, 173, 215

Tampón 154

TAN 82, 207

Teclado 28, 33, 129, 160

THEN 47, 91, 211

TL\$ 74

TO 50, 65, 210

TV 2, 5, 6, 7, 113, 116

U

Unidad de alimentación 1, 2

USR 99, 125, 171, 179, 207

V

VAL 70, 207

VAL\$ 71, 207

Variable(s) 62, 74, 87, 108, 145, 171,
190

de control 50, 168

de sistema 166, 169

Velocidad de transmisión 153

VERIFY 141-147, 150, 151, 216

VIDEO INV, tecla 212

VIDEO NORM, tecla 212

Z

Zócalo

TV 2

9 V DC 2







GRUPO INDESCOMP

AMSTRAD

ESPAÑA

Aravaca, 22. 28040 MADRID. Tel. 459 30 01, Télex 47660 INSC E, Fax 459 22 92
Delegación en Cataluña: Tarragona, 110. 08015 BARCELONA, Tel. 325 10 58