sinclair

ZX81 BASIC PROGRAMMING

sinclair

ZX81
BASIC
PROGRAMMING

por Steven Vickers
(Traducido y adaptado por
INVESTRONICA, S.A.)

Primera Edición

Ilustración de la cubierta por John Harris de Young Artists, encargada especialimente por Sinclair Research Limited.

Contenido

Contenido

CAPITULO 1

Instalación del ZX81

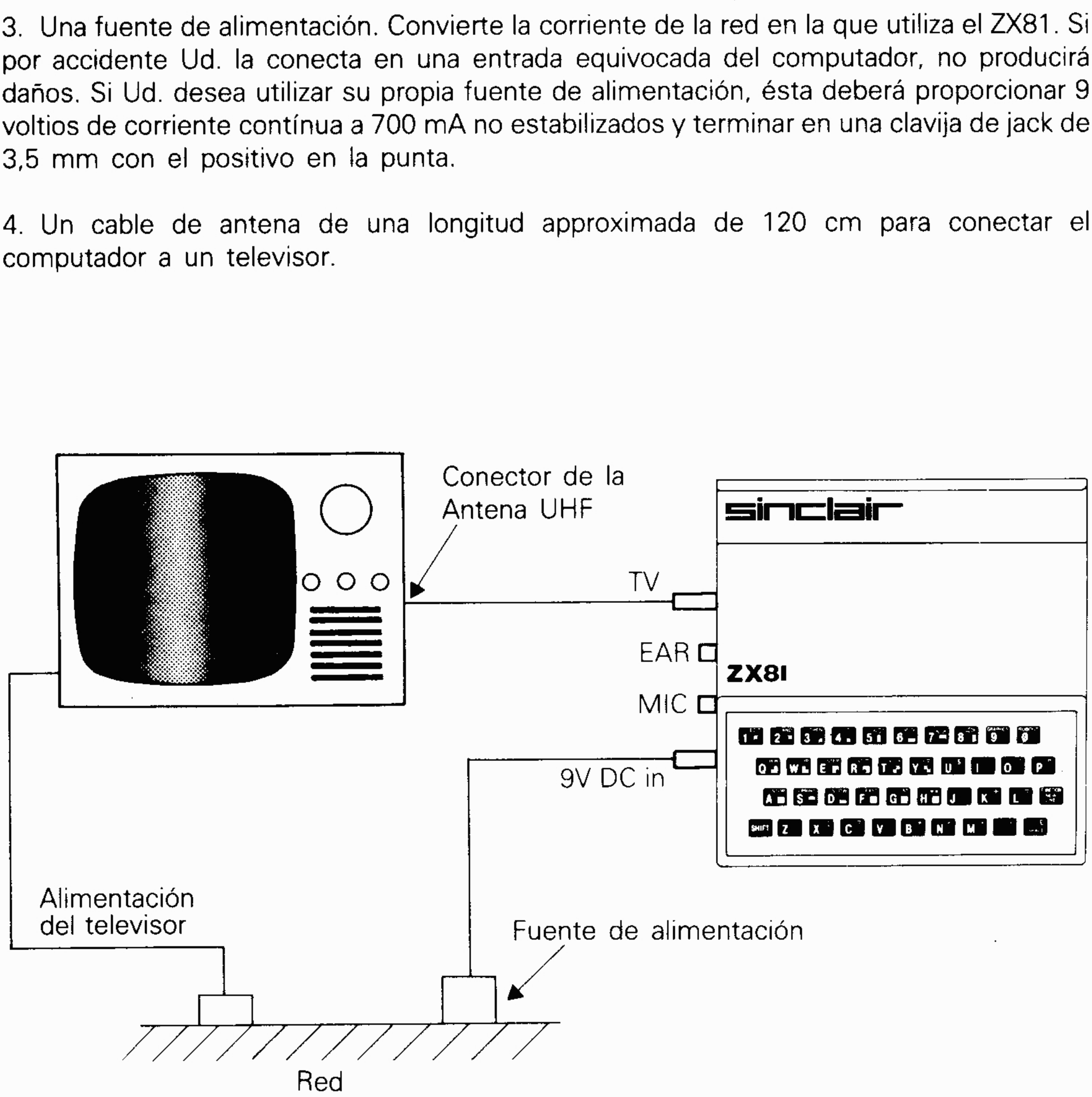
Al desembalar el ZX81, Ud. debe encontrar:

1. Este manual.

2. El computador. Este tiene tres enchufes para jacks (marcados 9V DC IN, EAR y MIC) una toma de antena y una parte de su placa de circuito al descubierto, donde Ud. puede conectar equipo extra. No existen interruptores, para encenderlo solo tiene que enchufarlo a la fuente de alimentación.

3. Una fuente de alimentación. Convierte la corriente de la red en la que utiliza el ZX81. Si por accidente Ud. la conecta en una entrada equivocada del computador, no producirá daños. Si Ud. desea utilizar su propia fuente de alimentación, ésta deberá proporcionar 9 voltios de corriente contínua a 700 mA no estabilizados y terminar en una clavija de jack de 3,5 mm con el positivo en la punta.

4. Un cable de antena de una longitud approximada de 120 cm para conectar el computador a un televisor.



5. Un par de cables de aproximadamente 30 cm de largo con clavijas de jacks en ambos extremos. Sirven para conectar el computador a un magnetofón.

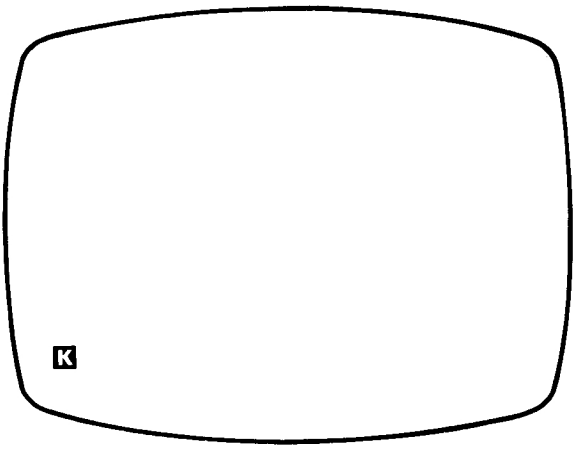
También necesitará un televisor. El ZX81 puede funcionar sin él, ¡pero Ud. no podrá ver lo que está haciendo!. Debe ser un televisor con UHF; si no está preparado para recibir el segundo programa, no sirve.

Mas tarde necesitará un magnetofón a casetes, ya que cuando Ud. desconecta el ZX81, se pierde toda la información almacenada en él, y la única manera de conservarla es registrándola en una casete (en el capítulo 16 verá Ud. como se hace). También puede comprar cintas que han preparado otras personas y así usar sus programas.

Cuando tenga todo reunido (excepto el magnetofón) conéctelo según se indica en la página anterior.

Si su televisor tiene dos enchufes de antena, marcados UHF y VHF, utilice la de UHF.

Conecte la alimentación y encienda el televisor. Ahora necesita sintonizar el televisor. El ZX81 trabaja sobre el canal 36 de UHF y cuando está sintonizado correctamente da una imagen como ésta:



Cuando esté utilizándo el computador, es aconsejable quitar totalmente el volumen.

Si su televisor tiene un control de sintonía de variación contínua, Ud. solo tiene que ajustarlo hasta conseguir ésta imagen. Actualmente muchos televisores tienen un pulsador individual para cada emisora. Elija uno libre y sintonícelo.

Si Ud. se encuentra en apuros con el computador, recuerde que siempre puede ponerlo a cero y recobrar ésta imagen, desenchufando la clavija de "9V DC IN" y volviéndola a enchufar. Esta operación debe constituir un recurso de emergencia, ya que al hacerla perderá toda la información almacenada en el computador.

Nota: La descripción del televisor se aplica a España, donde hay un sistema de UHF que utiliza 625 lineas a 50 imágenes por segundo. Este es el mismo sistema en otros paises

(por ejemplo, la mayoría de los paises de Europa Occidental excepto Francia). En los Estados Unidos de América, se utiliza VHF y 525 lineas con 60 imágenes por segundo.

Ahora que ya tiene instalado el computador, Ud. querrá hacer uso de él. Si Ud. ya conoce el lenguaje BASIC de computador, léase el Apéndice C y utilice el resto del manual solamente para aclarar los puntos oscuros.

Si Ud. no es aun experto, entonces le parte principal del manual se ha escrito para Ud.. No deje de lado los ejercicios, ya que la mayoría de ellos suscitan puntos interesantes que no están recogidos en el texto. Busque en ellos y haga lo que le dicte su imaginación, o lo que parezca encubrir le que Ud. no acaba de entender.

Independientemente de lo que Ud. haga, mantenga el computador en uso. Si Ud. se pregunta: "¿Que hará si yo le ordeno tal y tal cosa?", la respuesta es sencilla: Mándeselo y mire. Siempre que en el manual se le diga a Ud. que escriba alguna cosa, se debe preguntar: "¿Que podría escribir en lugar de eso?" y tratar de encontrar la respuesta. Cuanto mas ponga de su propia cosecha al escribir, major entenderá el ZX81. (Esto se llama enseñanza no programada). Escriba lo que escriba Ud. no puede dañar el computador.

CAPITULO 2

Diga al computador lo que tiene que hacer

Encienda el computador (enchufándolo) y consiga la pantalla blanca con la letra K en blanco sobre negro, como en la figura del capítulo 1. Para conseguir que haga algo, Ud. tiene que darle un mensaje que entienda; por ejemplo, el mensaje:

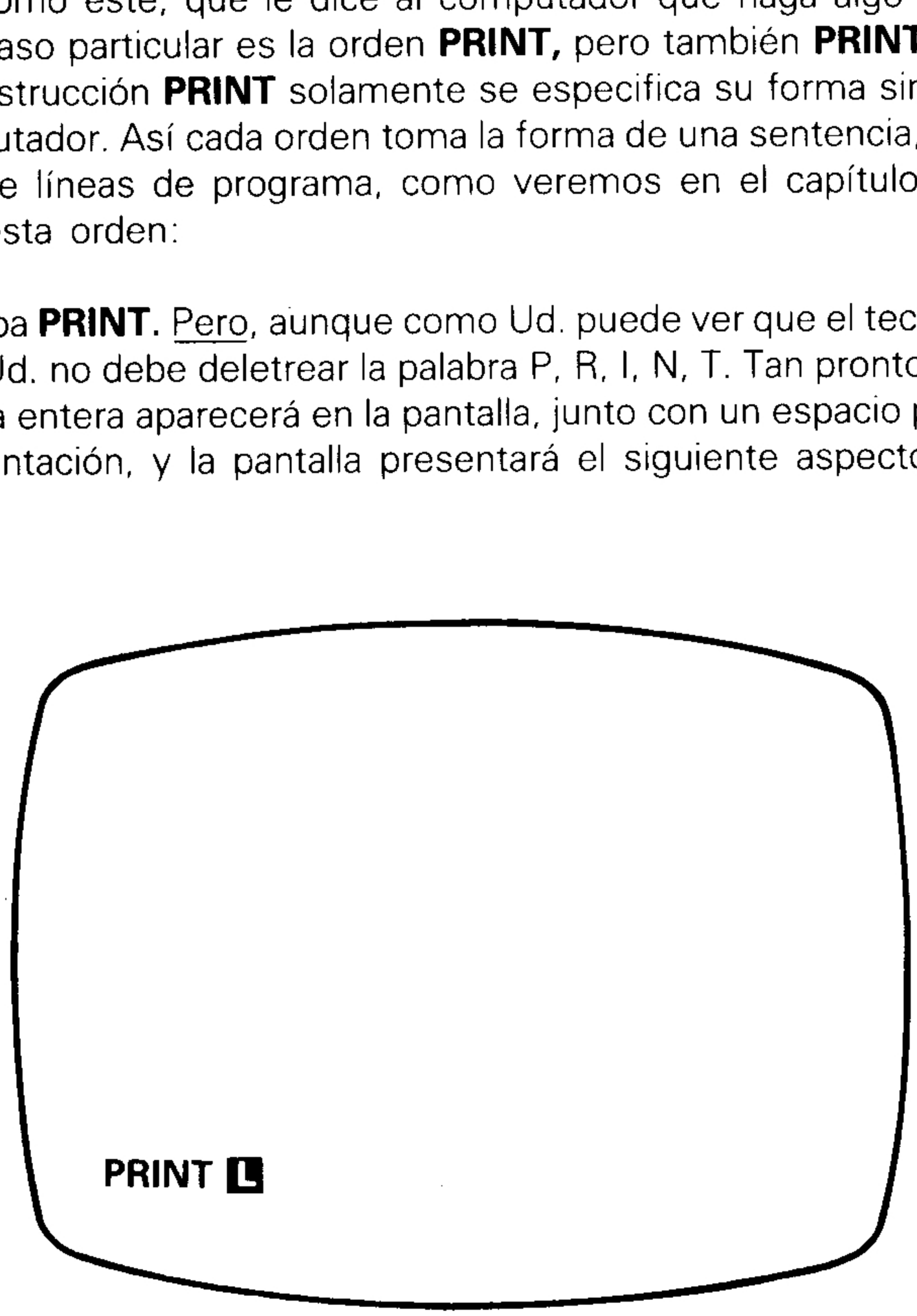
PRINT 2 + 2

le dice que debe realizar la suma 2+2 y presentar la respuesta en la pantalla del televisor.

Un mensaje como éste, que le dice al computador que haga algo en el acto, es una órden; en éste caso particular es la orden **PRINT,** pero también **PRINT** es una sentencia. Marcando una instrucción **PRINT** solamente se especifica su forma sin indicar como va a utilizarla el computador. Así cada orden toma la forma de una sentencia, pero hace algunas otras cosas (hace líneas de programa, como veremos en el capítulo 8).

Para escribir ésta orden:

1. Primero escriba **PRINT.** Pero, aunque como Ud. puede ver que el teclado tiene una tecla para cada letra, Ud. no debe deletrear la palabra P, R, I, N, T. Tan pronto como Ud. pulse la tecla P, la paiabra entera aparecerá en la pantalla, junto con un espacio para dar a las cosas una mejor presentación, y la pantalla presentará el siguiente aspecto:



La razón de ésto es que al principio de cada orden el computador está esperando una palabra-clave, una palabra que especifica que clase de orden es. Las palabras clave están escritas encima de las teclas y puede ver que **"PRINT"** aparece encima de la tecla P, de manera que para poner **"PRINT"** Ud. tiene que pulsar P.

El computador le hace saber a Ud. que espera una palabra-clave mediante la **K** con la que Ud. tuvo que empezar. Casi s pre hay alguna letra blanca sobre fondo negro (imagen en negativo), bien **K** ó **L** (o, como veremos más adelante, **F** ó **G** llamada el cursor.

La **K** significa: "cualquier tecla que Ud. pulse, la interpretaré como una palabra clave". Como Ud. vió, despúes de que pulsó P para **PRINT,** la **K** se cambió por una **L.**

Este sistema de pulsa solamente una tecla para conseguir más de un simbolo se emplea mucho en el ZX81. En el resto del manual aquellas palabras que tienen tecla propria están escritas en **NEGRITA.**

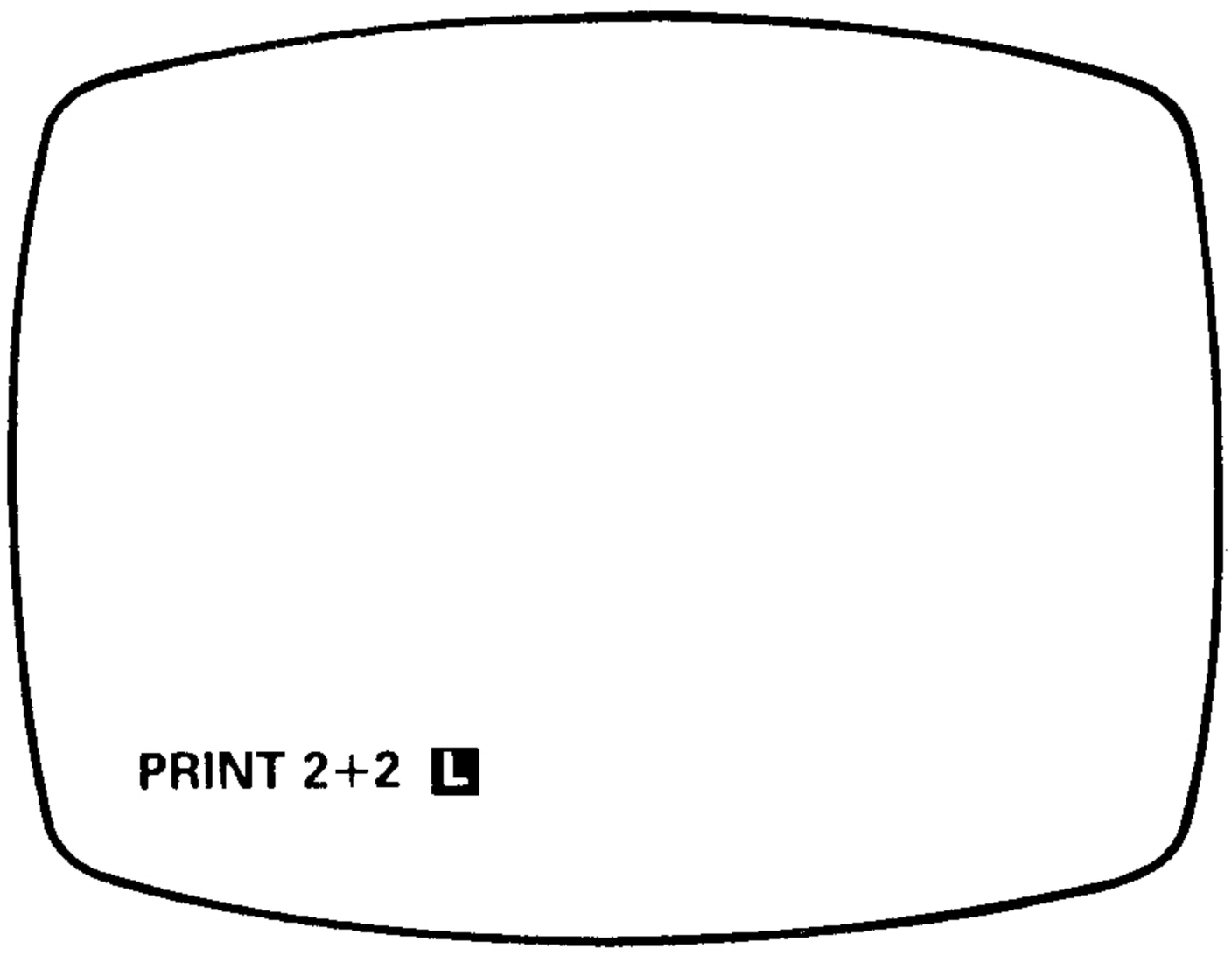
Debe recordar que es inútil tratar de deletrear éstas palabras, porque el computador no le entenderá.

2. Ahora pulse el 2. Esto no debería originar problemas. Debería aparecer el 2 en la pantalla y la **L** avanzará un espacio.

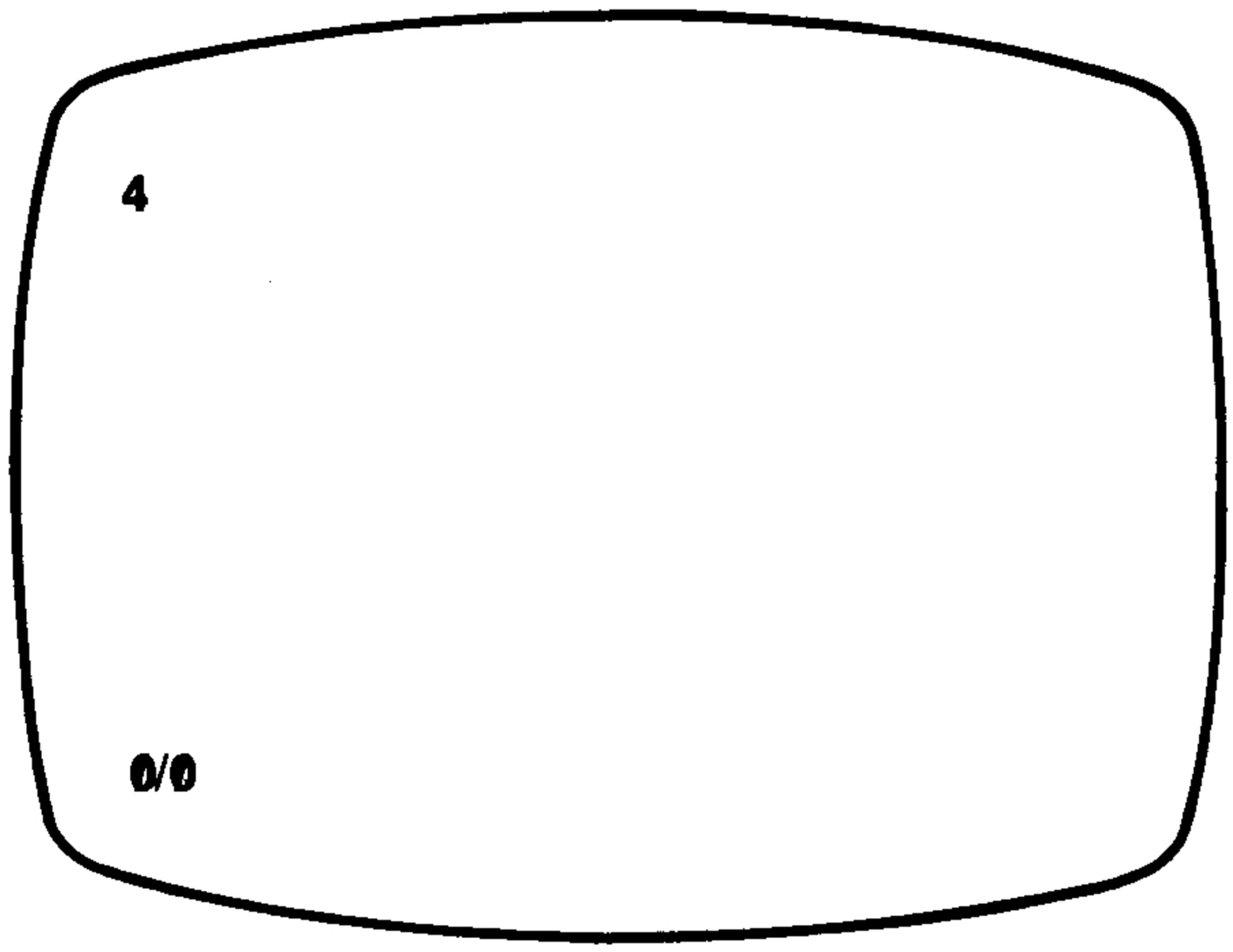
Observe como entre **PRINT** y el 2 se deja automáticamente un espacio para mejor presentación. Esto se hace siempre que es posible, de tal manera que Ud. raramente tendrá que introducir un espacio, no obstante si Ud. introduce un espacio, aparecerá en la pantalla, pero sin afectar al significado del mensaje.

3. Ahora pulse +. Este es un caracter cambiado (estos están marcados en rojo, el color de la palabra **SHIFT** en su tecla, en la esquina superior derecha de cada tecla) lo que significa que para introducir "+" debe mantener pulsada la tecla SHIFT y mientras lo hace pulsar la tecla LIST **K** + LEN.

4. Ahora vuelva a pulsar el 2. La pantalla se verá así:



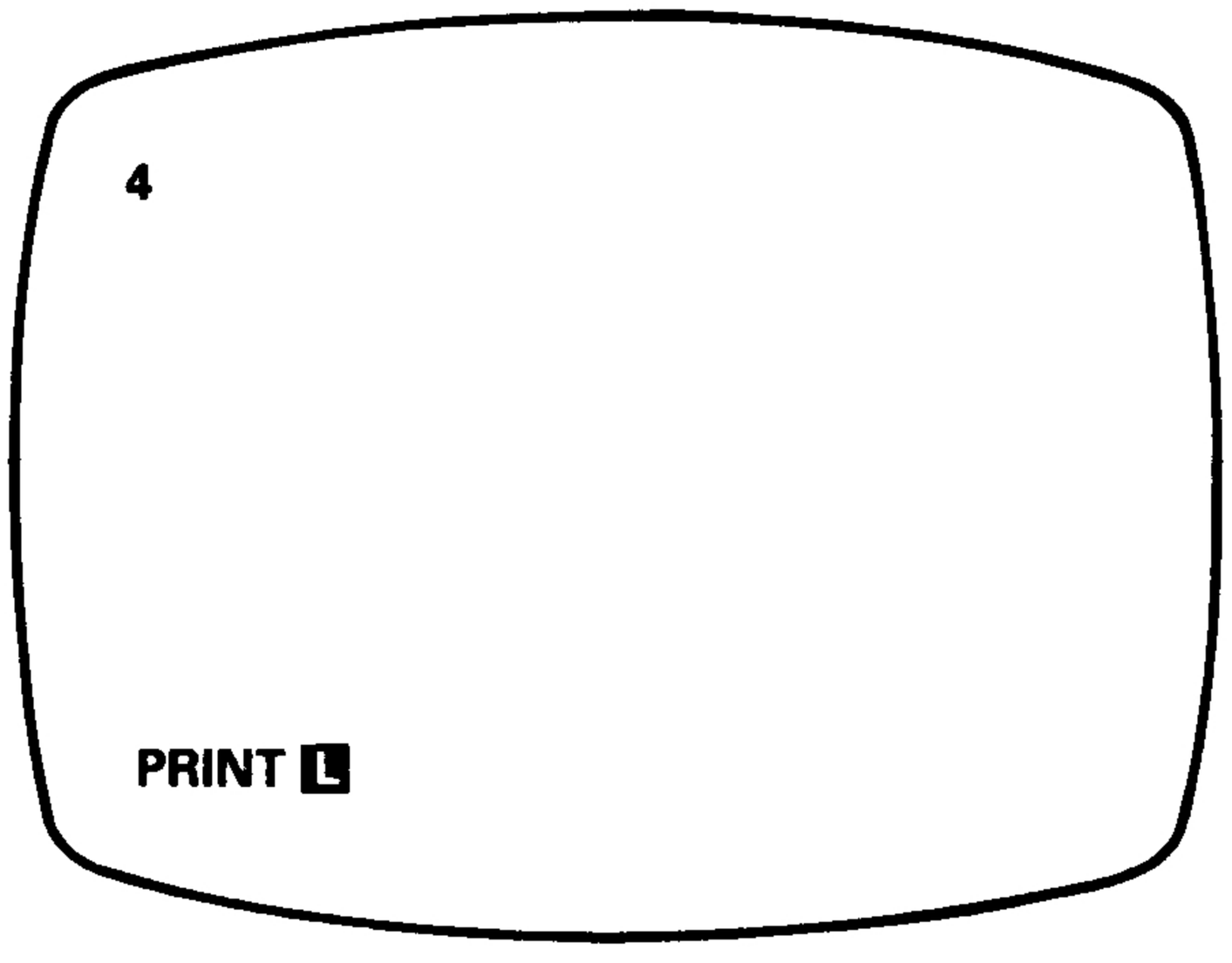
5. Ahora, y Ud. debe recordar siempre ésto, pulse **NEWLINE,** es decir la tecla FUNCTION NEW LINE Esto significa "mensaje terminado", o "de acuerdo computádor, muestranos lo que haces". Ahora el computador leerá el mensaje, estudiará lo que tiene que hacer, y lo hará. En éste caso, la pantalla cambiará a



4 es la respuesta, pero desde luego Ud. no necesita comprarse un computador para calcular ésto.

0/0 (Observe como el cero se escribe con una barra para distinguirlo de la O mayúscula. Esto es bastante corriente en el mundo de los computadores) es el informe mediante el cual el computador le dice a Ud. que tuvo éxito. El primer 0 significa "OK, no hay problemas". (En el apéndice B hay una lista de otros códigos informativos que pueden aparecer, por ejemplo si algo marcha mal). El segundo 0 significa "la última cosa que hice fué la línea 0". Ud. verá mas adelante, cuando escriba prògramas, que a una sentencia se le puede asignar un número y almacenarla en otro sitio para ejecutarla mas tarde, y en éste caso es una linea de programa. Realmente las órdenes no poseen número, pero a efecto de los informes el computador pretende que son línea 0.

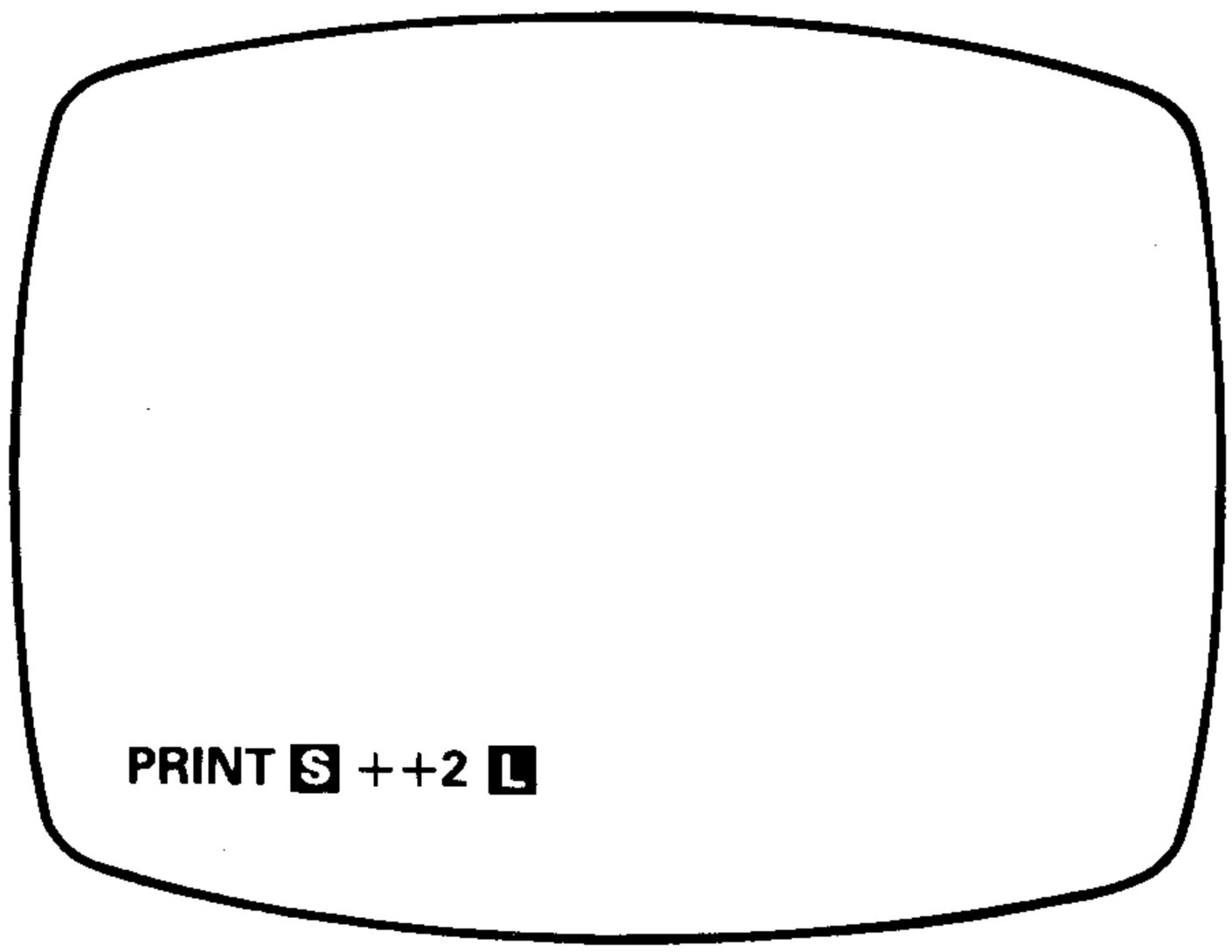
Imagínese que un informe oculta un cursor K y si Ud. pulsa ahora P para **PRINT**, el informe desaparecerá y en la pantalla tendremos:



El cursor también se puede utilizar para corregir errores. Pulse ++2 para obtener.

PRINT ++2 L

en la línea inferior. Cosa bastante incomprensible, y cuando Ud. pulse **NEWLINE** obtendrá:



La S es el indicador de error de sintaxis (la sintaxis es la gramática de los mensajes y dice lo que está y no está permitido) y nos dice que el computador entendió hasta **PRINT**, pero que el resto no era un mensaje correcto.

Desde luego lo que tiene Ud. que hacer es borrar el primer + y sustituirlo por, digamos 3. Primeramente tiene que mover el cursor hasta que se sitúe justo a la derecha del primer +; existen dos teclas, ◊ y ◊ (5 cambiado y 8 cambiado), que mueven el cursor hacia la izquierda y hacia la derecha. Manteniendo pulsada **SHIFT,** pulse la tecla ◊ dos veces. Esto hace que el cursor se mueva dos lugares hacia la izquierda para darnos

PRINT + L +2

Ahora pulse la tecla **RUBOUT** (Ø cambiado) y obtendrá

PRINT L +2

RUBOUT borra el caracter (o palabra-clave) situado inmediatamente a la izquierda del cursor.

Si Ud. ahora pulsa el 3, introducirá éste caracter inmediatamente a la izquierda del cursor dando

PRINT 3 L +2

y pulsando **NEWLINE** da la respuesta (5).

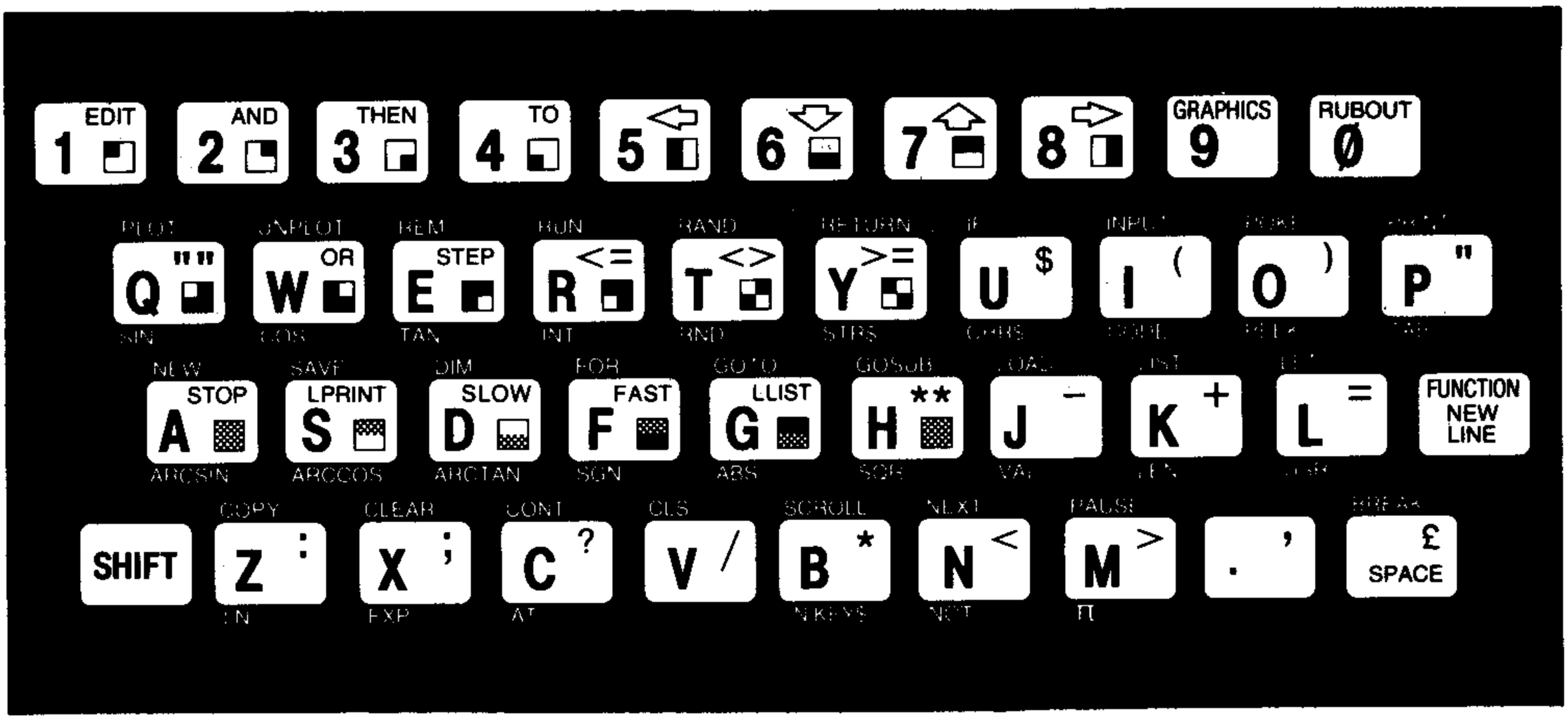
La tecla ⇨ (8 cambiada) trabaja exactamente igual que la tecla ⇦, excepto que el cursor se mueve hacia la derecha en lugar de hacia la izquierda.

Resumen

Este capítulo ha desarrollado como introducir mensajes en el ZX81, explicando:
El sistema de tecla de pulsación única para palabras.
Los cursores **K** y **L** .
Informes.
El indicador de error de sintaxis, **S** .
Como corregir errores utilizando ⇦, ⇨ y **RUBOUT.**

El teclado

He aquí una reproducción del teclado.



Recuerde que para usar **SHIFT,** tiene que mantenerla pulsada al mismo tiempo que pulsa otra tecla. No confundir el dígito 0 con la letra O.

CAPITULO 3

Una lección de Historia

Los mensajes que Ud. escribe estan en un lenguagje de programación llamado BASIC (significa Beginners' All-purpose Symbolic Instruction Code, Código de instrucción por símbolos de uso general para principiantes). Realmente al computador le supone un gran esfuerzo desmenuzar los mensajes BASIC en sus propias operaciones rudimentarias, pero, despues de todo se le paga para que lo haga. Los mensajes BASIC contienen suficientes palabras en inglés (como **PRINT**) como para que resulten bastante fáciles de aprender para alguien que sepa un poco de dicho idioma.

BASIC fué ideado en el Dartmouth College en New Hampshire, USA, en 1964, y desde entonces se ha convertido en el lenguaje de computador mas ampliamente utilizado por principiantes y aficionados. Esto es debido, en gran parte, a que está muy bien adaptado para su uso directo; cuando el usuario escribe algo el computador contesta enseguida. Existen otros lenguajes, tal como ALGOL (de hecho una familia completa de ALGOLs) y PASCAL, con una estructura mucho mas esmerada y mayor potencia que BASIC, pero solamente unos pocos, tal como los relativamente desconocidos APL y POP-2, son tan fáciles de utilizar directamente. Otros que deben ser recordados son FORTRAN, PL 1 y COBOL.

Muchas revistas sobre computadores personales publican programas en BASIC y merecen la pena echarles un vistazo por las ideas. Es casi seguro que Ud. tendrá que adaptarlos ligeramente porque cada computador que utiliza el lenguaje BASIC tiene su propio dialecto, diferente en todos ellos.

CAPITULO 4

El Sinclair ZX81 como una calculadora

Conecte el computador. Ahora puede utilizarlo como una calculadora, siguiendo las directrices del capítulo 2: Pulse **PRINT**, a continuación, todo lo que quiera calcular y después **NEWLINE.** (Normalmente no nos tomaremos la molestia de decircle a Ud. que pulse **NEWLINE**).

Como Ud. ya esperaría, el ZX81 no solo puede sumar, sino que también resta, multiplica (utilizando un asterisco * en lugar del acostumbrado signo "por", lo que constituye una práctica bastante común en computadoras) y divide (utilizando / en lugar de ÷). Practíquelas.

+, –, * y / son operaciones y los números que manejan son sus operandos.

El computador también puede calcular la potencia de un número utilizando la operación ** (H cambiada. No pulse *, B cambiada, dos veces): escriba

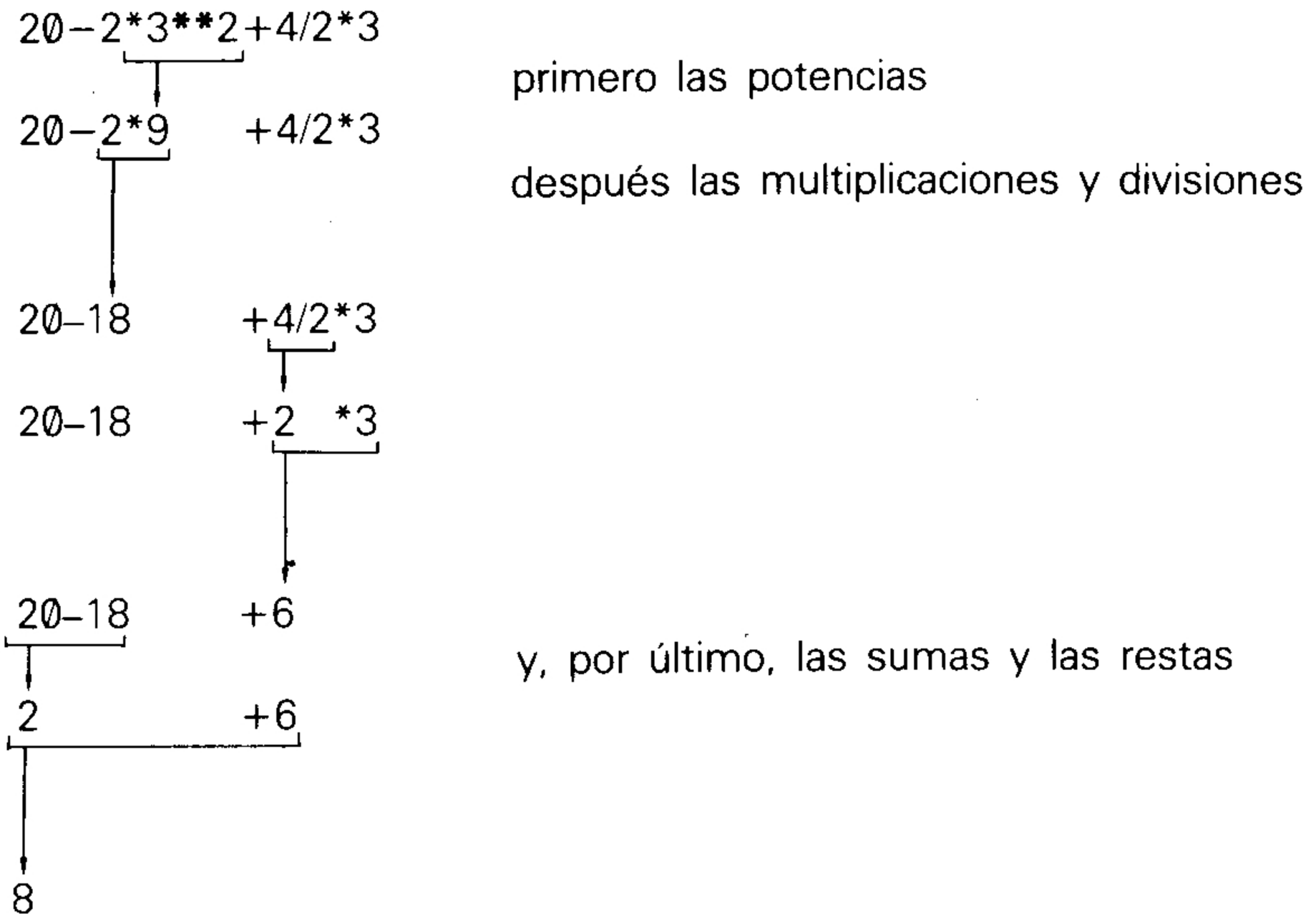
PRINT 2**3 (recuerde la **NEWLINE**)

y Ud. obtendrá 8 como resultado (2 elevado a la potencia 3, o 2^3, o el cubo de 2).

El ZX81 también realiza operaciones combinadas. Por ejemplo

PRINT 20 – 2*3**2 + 4/2*3

da 8 como resultado, pero tiene que calcular todas las etapas para conseguirlo, pues primero calcula todas las potencias (**) ordenadamente de izquierda a derecha, despues todas las multiplicaciones y divisiones (*y /) de nuevo de izquierda a derecha y a continuación las sumas y restas (+ y –) una vez mas de izquierda a derecha. Asi pues, nuestro ejemplo se realiza en las siguientes etapas:



Daremos forma a ésto asignando a cada operación una prioridad, un número entre 1 y 16. Las operaciones de mas alta prioridad se efectúan en primer lugar, y las operaciones con igual prioridad se calculan ordenadamente de izquierda a derecha.

** tiene prioridad 10
* y / tienen prioridad 8
+ y – tienen prioridad 6

Cuando el signo – se utilice para hacer algo negativo, como cuando se escribe –1, entonces tiene prioridad 9. Este es el menos "unario", como opuesto al menos binario en 3 – 1; una operación "unaria" tiene un solo operando, mientras que una binaria tiene dos. Observe que en el ZX81 no puede usar + como una operación unaria.

Este orden es totalmente rígido, pero Ud. puede burlarlo utilizándo paréntesis de manera que todo lo que está entre paréntesis se calcula primero y a continuación se trata como un solo número, por ejemplo

PRINT 3*2+2

da el resultado 6+2 = 8, pero

PRINT 3*(2+2)

da como resultado 3*4 = 12.

Una combinación como ésta se llama una expresión, en éste caso, una expresión aritmética o numérica porque la respuesta es un número. En general, siempre que el computador esté esperando que Ud. le facilite un número, puede darle una expresión en su lugar y él mismo calculará la respuesta.

Ud. puede escribir números con decimales (utilice el punto) y también puede emplear notación científica, como es bastante corriente en las calculadoras de bolsillo. Aquí, despues de un numero ordinario (con o sin decimales), Ud. puede escribir una partè exponencial consistente en la letra E, a continuación + o –, y por último un número sin decimales. Aquí la E significa "*10**" ("multiplicado por 10 elevado a"), por ejemplo

2.34E0 = 2.34 * 10**0 = 2.34
2.34E3 = 2.34 * 10**3 = 2340
2.34E–2 = 2.34 * 10**–2 = 0.0234, etc.

(Trate de hacer esto en el ZX81).

La manera mas fácil de concebir ésto es imaginarse que la parte exponente cambia el punto decimal hacia la derecha si el exponente es positivo y hacia la izquierda si es negativo.

También puede Ud. imprimir mas de una cosa a la vez, separándolas o bien con comas (,) o con puntos y comas (; o X cambiada). Si utiliza una coma, el número siguiente aparecrá

empezando o bien en el margen izquierdo o bien en el centro de la línea, en la colunna nº. 16.

Si Ud. utiliza punto y coma, el siguiente número se representará inmediatamente despues del último. Pruebe

```
PRINT 1; 2; 3; 4; 5; 6; 7; 8; 9; 10
```

y

```
PRINT 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

para ver las diferencias. Si lo desea puede mezclar comas y puntos y comas dentro de una misma sentencia **PRINT**.

Resumen

Sentencias: **PRINT,** con comas y puntos y comas
Operaciones: +, –, *, /, **,
Expresiones, notación científica

Ejercicios

1. Pruebe

```
PRINT 2.34E0
PRINT 2.34E1
PRINT 2.34E2
```

y así sucesivamente hasta

```
PRINT 2.34E15
```

Verá que después de un rato el ZX81 también empieza a utilizar notación científica. Esto es debido a que nunca utiliza mas de 14 espacios para escribir un número. De igual manera, pruebe

```
PRINT 2.34E–1
PRINT 2.34E–2
```

y así sucesivamente.

2. Pruebe

```
PRINT 1,,2,,3,,,4,,,,5
```

Una coma siempre le hace avanzar a Ud. un bit para el próximo número. Ahora pruebe

PRINT 1;;2;;3;;;4;;;;5

¿Porqué una cadena de puntos y comas no difiere de un solo punto y coma?

3. **PRINT** solo proporciona 8 cifras significativas. Pruebe

PRINT 4294967295, 4294967295 – 429E7

Esto prueba que el computador puede retener todos los dígitos de 4294967295, incluso aunque no esté preparado para representarlos todos a la vez.

4. Si Ud. tiene tablas de logaritmos, compruebe ésta regla:
Elevando 10 a una potencia es lo mismo que hallar el antilogaritmo del exponente. Por ejemplo, escriba

PRINT 10**0.3010

y búsque el antilogaritmo de 0.3010. ¿Porqué no son exactamente iguales las respuestas?

5. El ZX81 utiliaza aritmética de coma flotante, lo que signifaca que guarda por separado los dígitos de un número (su mantisa) y la posición de la coma (el exponente). Esto no siempre es exacto, incluso para números enteros. Escriba

PRINT 1E10+1–1E10, 1E10–1E10+1

Los números se mantienen con una precisión de 9 1/2 dígitos, por lo que 1E10 es demasiado grande para ser mantenido en su forma exacta. La imprecisión (realmente de casi 2) es mayor de 1, por lo que los números 1E10 y 1E10+1 le parecen iguales al computador.

Para un ejemplo mas peculiar, escriba

PRINT 5E9+1 – 5E9

En éste caso, la imprecisión en 5E9 es solamente de casi 1, y el 1 que se añade hace que se redondee hasta 2, por lo que al computador los números 5E9+1 y 5E9+2 le parecen iguales.

El número entero mas grande que puede retener totalmente exacto es $2^{32} - 1$ (4,294,967,295).

CAPITULO 5

Funciones

Matemáticamente hablando, una funcion es una regla para obtener un numero (el resultado) a cambio de otro (el argumento u operando) y en realidad es una operación "unaria". El ZX81 tiene algunas incorporadas y sus nombres son las palabras escritas debajo de las teclas. **SQR** por ejemplo, es la función raíz cuadrada que todos conocemos, y

PRINT SQR 9

da 3, la raíz cuadrada de 9. (Para conseguir **SQR**, primero pulse la tecla **FUNCTION, NEWLINE** cambiada. Esto hace que el cursor cambie a **F**. Ahora pulse la tecla **SQR** (H): **SQR** aparece en la pantalla y el cursor vuelve a **L**. Para todas las palabras que están escritas debajo de las teclas se utiliza el mismo procedimiento, ya que casi todas ellas son nombres de funciones).

Pruebe

PRINT SQR 2

Puede comprobar la exactitud de la respuesta mediante

PRINT SQR 2***SQR** 2

que debe dar 2. Observe que las dos **SQRs** se calculan antes de *, y de hecho todas las funciones (excepto una, **NOT**) se calculan antes de las cinco operaciones +, –, , / y ** Nuevamente Ud. puede saltarse ésta regla haciendo uso de los parentesis.

PRINT SQR (2*2)

da 2.

He aqui algunas funciones más (en el apéndice C tiene Ud. una lista completa). Si sus conocimientos matemáticos no son lo suficientemente profundos para entender algunas de ellas, no se preocupe, todavía prodrá seguir utilizando el computador.

SGN Es la función signo (algunas veces se le llama signum para evitar confusiones con las SIN). El resultado es –1, 0 ó +1 según que el argumento sea negativo, cero o positivos

ABS Valor absoluto o módulo, El resultado es el argumento hecho positivo, así

ABS –3.2 = **ABS** 3.2 = 3,2

SIN, **COS**, **TAN**, **ASN** (arcsen), **ACS** (arccos), **ATN** (arctan): Funciones trigonométricas. Trabajan en radianes, no en grados.

LN Logaritmo natural (en base 2.718281828459045 . . ., llamado nº. E)
EXP Función exponencial
SQR Raíz cuadrada
INT Parte entera. Siempre redondea por defecto, así **INT** 3.9 = 3 e **INT** – 3.9 = 4. (un entero es un número entero, incluso negativo).
PI π = 3.141592653589979 . . ., la longitud de la circunferencia en metros de un circulo que tiene un metro de diámetro. **PI** no tiene argumento (En el computador solo hay almacenados diez dígitos y solo se representan ocho en la pantalla).
RND Tampoco **RND** tiene argumento. Representa un número al azar entre 0 (cuyo valor puede tomar) y 1 (que no puede alcanzar).

Empleando la terminología del último capítulo, todas éstas funciones, excepto **PI** y **RND** son operaciones unarias con prioridad 11. (**PI** y **RND** son operaciones nularias, porque no tienen operandos).

Las funciones trigonompetricas y **EXP**, **LN** y **SQR**, generalmente se calculan con exactitud hasta el octavo dígito.

RND Y **RAND**: Ambas están en la misma tecla, pero mientras **RND** es una función, **RAND** es una palabra-clave como **PRINT**. **RAND** se utiliza para controlar la aleatoriedad de **RND**.

RND no es realmente aleatoria, sino que sigue una secuencia de 65536 números. Al estar desordenados parecen aleatorios (**RND** es seudoaleatoria). Puede utilizar **RAND** para que **RND** empiece en un lugar determinado en ésta secuencia pulsando **RAND**, a continuación un número entre 1 y 65535 y despúes **NEWLINE**. No es tan importante conocer donde ésta un número determinado para iniciar **RND**, por cuanto el mismo número despúes de **RAND** hará arrancar a **RND** siempre desde el mismo lugar. Por ejemplo, escriba

RAND 1 (1 **NEWLINE**)

y despúes

PRINT RND

y repítalo varias veces. (Recuerde utilizar **FUNCTION** para conseguir **RND**) La respuesta de **RND** será siempre 0.00022735596, lo que no es precisamente aleatorio.

RAND 0

(u olvídese del 0) actua un poco distinto: juzga donde debe arrancar **RND** en función del tiempo de encendido del televisor y ésto sería autenticamente aleatorio.

Nota: En algunos dialectos de BASIC encontrará siempre el argumento de una función encerrado entre paréntesis. Este no es el caso en ZX81 8K BASIC.

Resumen

Sentencia: **RAND**
Funciones: **SGN, ABS, SIN, COS, TAN, ASN, ACS, ATN, LN, EXP, SQR, INT, PI, RND. FUNCTION**

Ejercicios

1. Para calcular logaritmos vulgares (en base 10), que son los que Ud. buscaba en las tablas, divida el logaritmo natural por **LN** 10. Por ejemplo, para hallar el log 2

PRINT LN 2/**LN** 10

que dá como resultado 0.301003.

Trate de hacer multiplicaciones y divisiones utilizando logaritmos, empleando el ZX81 como tablas de logaritmos de la manera explicada (para antilogaritmos ver el ejercicio no 4 del capítulo 4). Compruebe los resultados utilizando * y / (lo que es mas fácil, rápido, seguro y mucho mas adecuado).

2. **EXP** y **LN** son funciones inversas entre si, en el sentido de que si Ud. aplica una y despues la otra, vuelve a obtener el número original. Por ejemplo:

LN EXP 2 = **EXP LN** 2 = 2

Lo mismo nos ocurre con **SIN** y **ASN, COS** y **ACS** y **TAN** y **ATN.** Ud. puede aprovecharse de esto para comprobar con qué exactitud calcula el computador éstas funciones.

3. π radianes son 180°, y así para convertir grados en radianes divida por 180 y multiplique por π: por ejemplo

PRINT TAN (45/180***PI**)

da tan 45° (1).

Para pasar de radianes a grados, divida por π y multiplique por 180.

4. Intente

PRINT RND

unas cuantas veces para ver como varían las respuestas. ¿Puede Ud. encontrar alguna ley? (Lo dudamos).

¿Como utilizaría **RND** e **INT** para conseguir un número cualquiera entre 1 y 6, que represente la tirada de un dado? (Respuesta: **INT** (**RND** * 6) + 1).

5. Compruebe ésta regla:

Suponga que elige un número entre 1 y 872 y escribe

RAND y su número (y **NEWLINE**)

Entonces el próximo valor de **RND** será:

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

6. (Solo para matemáticos)
   Sea p un número primo alto y a una raiz primitiva del módulo p.
   Entonces si b_i es el residuo de a^i módulo p ($1 \leqslant b_i \leqslant p - 1$), la serie

$$\frac{b_i - 1}{p - 1}$$

es una serie cíclica de p – 1 números distintos de la gama del 0 al 1 (excluyendo el 1). Mediante una elección adecuada, se puede conseguir que parezca totalmente aleatoria.

65537 es un número primo Mersenne, $2^{16} - 1$. Utilice esto y la ley de Gauss de reciprocidad de segundo grado, para demostrar que 75 es una raiz primitiva del módulo 65537.

El ZX81 utiliza p = 65537 y a = 75 y guarda algunos $b_i - 1$ en la memoria. La función **RND** supone sustituir $b_i - 1$ en la memoria por $b_{i+1} - 1$ y producir el resultado $(b_{i+1} - 1)/(p - 1)$. **RAND** n (siendo $1 \leqslant n \leqslant 65535$) hace b_i igual a n + 1.

7. **INT** siempre redondea por defecto. Para redondear al entero mas próximo, añada en primer lugar 0,5. Por ejemplo:

INT (2.9+0.5) = 3 **INT** (2.4+0.5) = 2
INT (–2.9+0.5) = –3 **INT** (–2.4+0.5) = –2

Compare estos resultados con los que obtendría sin añadir 0.5.

8. Pruebe

PRINT PI, PI –3, **PI** –3.1, **PI** –3.14, **PI** –3.141

Esto le indica con que precisión el computador almacena π.

CAPITULO 6

Variables

Ud. estará diciendo: "Mi calculadora de bolsillo puede almacenar un número y recordarlo mas tarde. ¿Puede su ZX81 hacer lo mismo?"

Si. De hecho y hablando literalmente puede almacenar centenares, utilizando la sentencia **LET.** Suponga que los huevos cuestan 58 pesetas la docena y que Ud. quiere recordarlo. Escriba

LET HUEVOS = 58 (y **NEWLINE,** por supuesto)

En primer lugar, el computador ha reservado un sitio en su interior donde Ud. puede almacenar su número, y en segundo lugar, a ese sitio le ha dado el nombre de "HUEVOS" para que Ud. pueda localizarlo mas tarde. Esta combinación de sitio de almacenaje y de nombre se le llama variable. En tercer lugar, ha guardado el número 58 en ese sitio: nosotros decimos que le ha asignado el valor 58 a la variable (cuyo nombre es) HUEVOS. Huevos es una variable numérica, puesto que su valor es un número.

¿Quiere saber cuanto cuestan los huevos? Escriba

PRINT HUEVOS

Si Ud. quiere saber cuanto cuesta media docena de huevos, entonces escriba

PRINT HUEVOS/2

En realidad, si Ud. quisiera conocer el cuadrado del seno del precio de un huevo, puede escribir

PRINT SIN (HUEVOS/12)**2

"¡Que fácil!" debe de estar Ud. pensando, y justo cuando se está preguntando que hará a continuación, entra de repente su vecina diciendo: "¡Dios mio! ¡Los huevos acaban de subir a 61 pesetas la docena!"

Bien. No hay tiempo que perder. Escriba

LET HUEVOS = 61

Esto no reserva ningún espacio de memoria mas, sino que sustituye el viejo valor de 58 por 61. Ahora puede escribir

PRINT HUEVOS

seguro de conseguir el precio mas actualizado disponible.
Ahora escriba

PRINT LECHE

Le aparecerá un informe 2/0 y buscando 2 en el apéndice B, verá que significa "variable no localizada", el computador no tiene ni la mas remota idea de lo que cuesta la leche, porque Ud. no se lo ha dicho. Escriba

LET LECHE = 18.5

y todo estará arreglado.
(Escriba

PRINT LECHE

de nuevo).

Un variable no tiene por qué tener nombre de comestible, Ud. puede utilizar cualquier letra o dígito siempre que el primero sea una letra. También puede haber espacios para facilitar su lectura pero no se contabilizarán como parte del nombre.

Por ejemplo, se pueden utilizar como nombres de variables:

DOS KILOGRAMOS DE MANZANAS PERO NO DELICIOSAS GOLDEN
RADIO 3
RADIO 33
X
K9P

pero éstos no:

3 OSOS (empieza por un dígito)
TALBOT? (? no es una letra ni un dígito)
WHITE ON BLACK (No se admiten caracteres en negativo)
JUAN-JOSE (– no es una letra ni un dígito)

Ahora escriba

CLEAR

y

PRINT HUEVOS

Conseguirá nuevamente un informe 2 (variable no localizada). El efecto de **CLEAR** es limpiar todo el espacio de memoria que había sido reservado para variables con lo que cada

variable está como si no hubiese sido nunca definida. Desenchufando y volviendo a enchufar el computador también ocurre ésto, pero entonces no recuerda nada en absoluto cuando se le vuelve a enchufar.

Las expresiones pueden incluir el nombre de una variable en cualquier parte donde puedan tener un número.

Nota: En algunas versiones de BASIC se puede omitir **LET** y escribir solamente (por ejemplo)

```
HUEVOS = 58
```

Esto no está permitido en el ZX81. En cualquier caso, encontrará bastantes dificultades para escribirlo.

También en algunas versiones, solo se comprueban los dos primeros caracteres de un nombre, y así RADIO 3 y RADIO 33 se contabilizarían como el mismo nombre; y en algunos otros un nombre de variable debe ser una letra seguida de un dígito. Ninguna de éstas restricciones se aplica al ZX81.

Aún mas, en algunas versiones de BASIC, si una variable no ha aparecido todavía a la izquierda de una de una sentencia **LET,** se supone que tiene el valor 0. Como ya vió antes con **PRINT** LECHE, esto no es así en el ZX81.

Resumen

Variables
Sentencias: **LET, CLEAR**

Ejercicios

1. ¿Por qué los nombres de variables tienen que empezar con una letra?

2. Si Ud. no está familiarizado con el cálculo de potencias (**, H cambiada), haga éste ejercicio.

En su nivel mas elemental, "A**B" signifca "A multiplicado por sí mismo B veces", pero naturalmente esto solamente tiene sentido si B es un número entero positivo. Para encontrar una definición que sirva para otros valores de B, tengamos en cuenta la regla

```
A ** (B + C) = A **B * A**C
```

No necesitará mucho para convencerse de que esto funciona cuando B y C son sendos números enteros positivos, pero si decidimos que sirva incluso cuando no lo son, nos vemos obligados a aceptar que

A ** 0 = 1
A ** (–B) = 1/A ** B
A ** (1/B) = la raíz de índice B de A

y

A ** (B * C)= (A ** B) ** C

Si Ud. nunca ha visto nada de esto con anterioridad, no trate de recordarlo todo enseguida; solamente recuerde que

A ** –1 = 1/A

y

A ** (1/2) = la raíz cuadrada de A

y puede que cuando se haya familiarizado con éstas, el resto empiece a tener sentido para Ud.

Experimente con todo esto diciéndole al computador que imprima varias expresiones que contengan **, por ejemplo:

PRINT 3 **(2+0), 3**2*3**0
PRINT 4 **–1, 1/4

3. Escriba

LET E = **EXP** 1

Ahora E tiene el valor 2.718281828 . . ., base de los logarítmos naturales. Pruebe la regla

EXP un número = E ** el número

para varios números.

CAPITULO 7

Cadenas

Una cosa que el ZX81 puede hacer y que las calculadoras de bolsillo no pueden, es tratar textos. Escriba

PRINT "!Oiga, o soy su ZX81!" (" es la P cambiada)

El cálido saludo entrecomillado se llama una cadena (quiere decir una cadena de caracteres) y puede contener todos los caracteres que Ud. quiera excepto las comillas de la cadena, ". No obstante Ud. puede utilizar la llamada representación de comillas, " " (Q cambiada) y se imprimiran como " por medio de una sentencia **PRINT.**

Un error de suritura con cadenas bastante corriente es olvidar una de las comillas, lo que será advertido por el indicador **S** .

Si Ud. está escribiendo números, puede utilizar estas cadenas para explicar lo que significan los números. Por ejemplo, escriba

```
LET HUEVOS = 61
```

y a continuación

```
PRINT "EL PRECIO DE LOS HUEVOS ES"; HUEVOS; "PESETAS LA
      DOCENA".
```

(No se preocupe si sobrepasa el final de la linea).

Esta sentencia exhibe tres cosas (partidas **PRINT**), a saber: la cadena "EL PRECIO DE LOS HUEVOS ES", el número 61 (valor de la variable HUEVOS) y por último "PESETAS LA DOCENA". De hecho Ud. puede PRINT la cantidad de partidas que desee y cualquier mezcla de cadenas y números (o expresiones), observe como en una cadena los espacios forman parte de ella tanto como las letras. No son ignorados ni al final.

Hay montones de cosas que Ud. puede hacer con cadenas.

1. Puede asignarlas a variables. Sin embargo, el nombre de la variable debe ser especial para indcar que su valor es una cadena y no un número: debe ser una sola letra seguida por $ (U cambiada). Por ejemplo, escriba

```
LET A$ = "JAMON DULCE"
```

y

```
PRINT A$
```

2. Puede sumarlas, juntándolas. A esto se le suele llamar concatenación, que significa "encadenar juntas", y eso es exactamente lo que hace. Pruebe

```
PRINT "JAMON" + "N LONCHAS"
```

No se pueden restar, multiplicar ni dividir cadenas, ni elevarlas a potencias.

3. Puede aplicar algunas funciones a cadenas para obtener números, y viceversa.

LEN Se aplica a una cadena, y el resultado es su longitud. Por ejemplo: **LEN** "QUESO" = 5

VAL Se aplica a una cadena, y el resultado es lo que esa cadena da cuando se calcula como una expresión aritmética. Por ejemplo: (si A = 9), **VAL** "1/2 + **SQR**A" = 3.5. Si la cadena a la que se aplica **VAL** contiene variables, se deben seguir dos reglas:

(i) Si la función **VAL** es parte de una expresión mas larga, debe constituir la primera partida, por ejemplo:
10 **LET** X = 7 + **VAL** "Y" debe cambiarse a 10 **LET** X = **VAL** "Y" + 7.

(ii) **VAL** solo puede aparecer en la primera coordenada de una sentencia **PRINT AT, PLOT** o **UNPLOT** (ver capítulos 17 y 18), por ejemplo:

```
10 PLOT 5, VAL "X" debe cambiarse a
10 LET Y = VAL "X"
15 PLOT 5, Y
```

STR$ Se aplica a un número, y el resultado es lo que aparecería en la pantalla si el número fuese representado por una sentencia **PRINT.** Por ejemplo: **STR$** 3.5 = "3.5".

4. Igual que con los números, puede combinarlas para hacer expresiones encadenadas, como

VAL (**STR$ LEN** "123456" + "–4")

que se calcula como

```
VAL (STR$ LEN "123456" + "–4")
VAL (STR$ 6           + "–4")
VAL ("6"              + "–4")
VAL ("6–4"                   )
VAL "6–4"
6–4
2
```

Resumen

Cadenas
Operación: + (para cadenas)
Funciones: **LEN, VAL, STR$**

Ejercicios

1. Escriba

LET A$ = "2+2"

y a continuación

PRINT A$;" = ",**VAL** A$

Trate de cambiar A$ por cosas mas complicadas y que hagan lo mismo, por ejemplo

LET A$ = "**ATN** 1*4"

(La solución debería ser π)

2. La cadena " " sin caracteres se llama cadena vacía o nula. Es la única cadena cuya longitud es 0. Recuerde que los espacios tienen significado y una cadena vacía no es la misma que otra que tenga espacios.

No la confunda con la imagen de comillas, **" "** (una sola señal, Q cambiada). Este es un dispositivo especial para superar el hecho de que Ud. no puede escribir unas comillas ordinarias de cadena en medio de una cadena (¿Por qué no?). Cuando la imagen de comillas aparece en una cadena que tiene sus comillas al final (por ejemplo, en el listado de programa), se representa como dos símbolos de comillas para distinguirla de las comillas ordinarias, pero cuando se representa por medio de una sentencia PRINT, es como un solo símbolo de comillas.

Pruebe

PRINT "X";"";"X","""","""";"";"""

3. Si Ud. disfruta poniendo en apuros a computadores, escriba

PRINT "2+2 = ";2+1

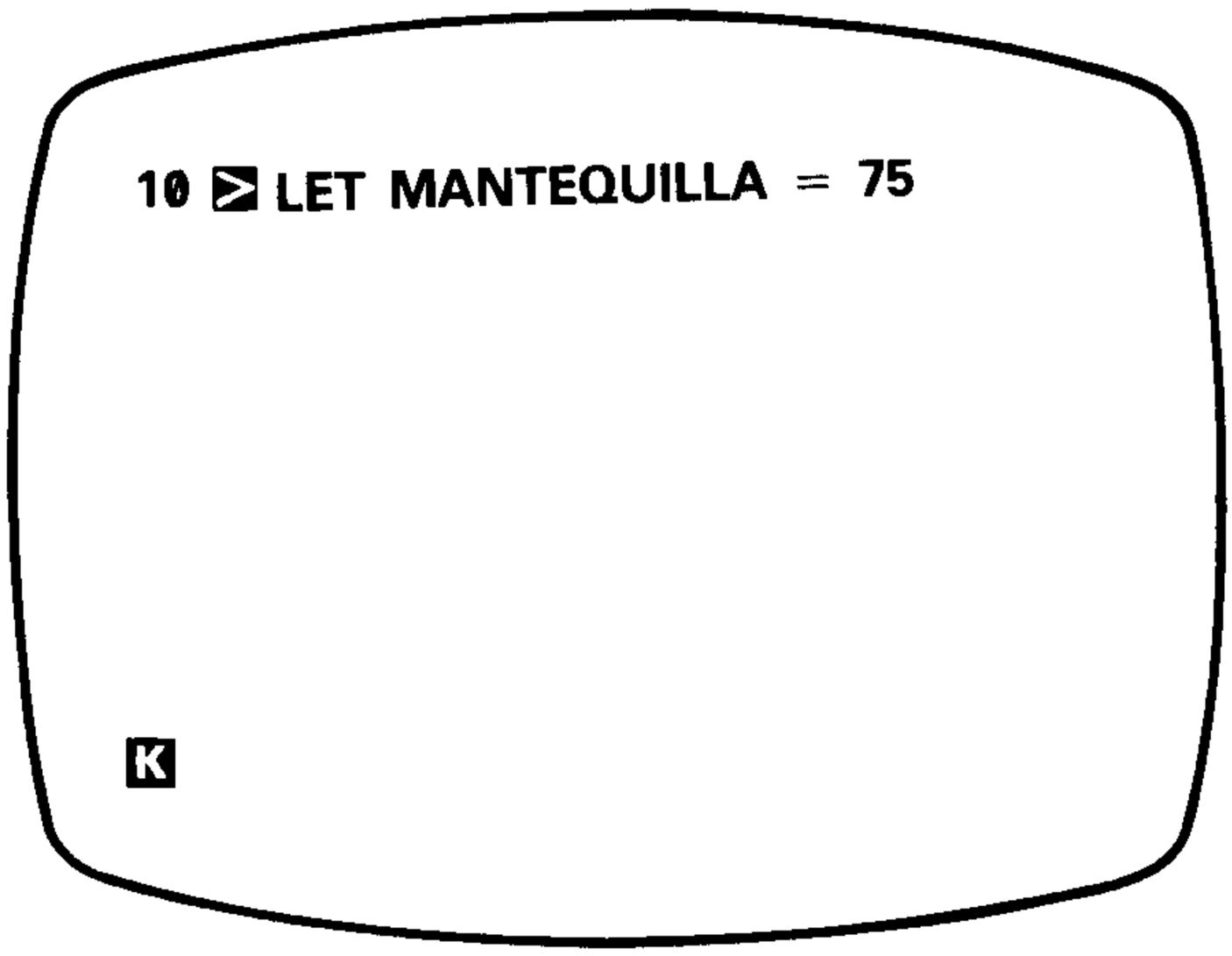
CAPITULO 8

Programando el computador

Y ahora, ¡al fin!, deberá escribir un programa. Desenchufe y vuelva a enchufar el computador para estar seguros de que está inicializado. Ahora escriba

10 **LET** MANTEQUILLA = 75 (y **NEWLINE**)

y la pantalla presentará el siguiente aspecto



Esto es diferente de lo que ocurrió con HUEVOS en el capítu 6; si Ud. escribe

PRINT MANTEQUILLA

verá (por el informe 2) que la variable MANTEQUILLA no ha sido definida. (Pulse nuevamente **NEWLINE** y la pantalla volverá a tomar el aspecto anterior).

Motivado por el hecho de que la sentencia **LET** tiene un número, 10, delante, el computador no la ejecuta inmediateamente, sino que la guarda para mas adelante. 10 es su número de línea y se utiliza bastante para referirse a ella de la misma manera que los nombres se usan para referirse a variables. Un juego de éstas sentencias almacenadas se llama un programa. Ahora escriba

20 **PRINT** MANTEQUILLA

y en la pantalla tendremos

Esto es un listado de su programa. Para tener realizado (o ejecutado o hecho), el programa, pulse

RUN (no se olvide de **NEWLINE**)

y en la esquina superior izquierda de la pantalla aparecera la respuesta, 75. En la esquina inferior izquierda verá el informe 0/20. Como Ud. ya sabe, 0 signifca "OK, no hay problemas", y 20 es el número de la línea en que terminó. Pulse **NEWLINE** y el listado volverá a aparecer.

Observe que las sentencias fueron ejecutadas siguiendo el orden de sus números de línea.

Supongamos ahora que de repente se acuerda que tambien tiene que registrar el precio de la levadura. Escriba

15 **LET** LEVADURA = 40

y lo registra. Esto hubiese sido mucho mas dificil si las dos primeras líneas tuviesen los números 1 y 2 en lugar de 10 y 20 (los números de línea deben ser números enteros entre 1 y 9999), y por eso cuando se escriba por primera vez un programa, es buena costumbre dejar espacios en los números de línea.

Ahora necesitamos cambiar la línea 20 a

20 **PRINT** MANTEQUILLA, LEVADURA

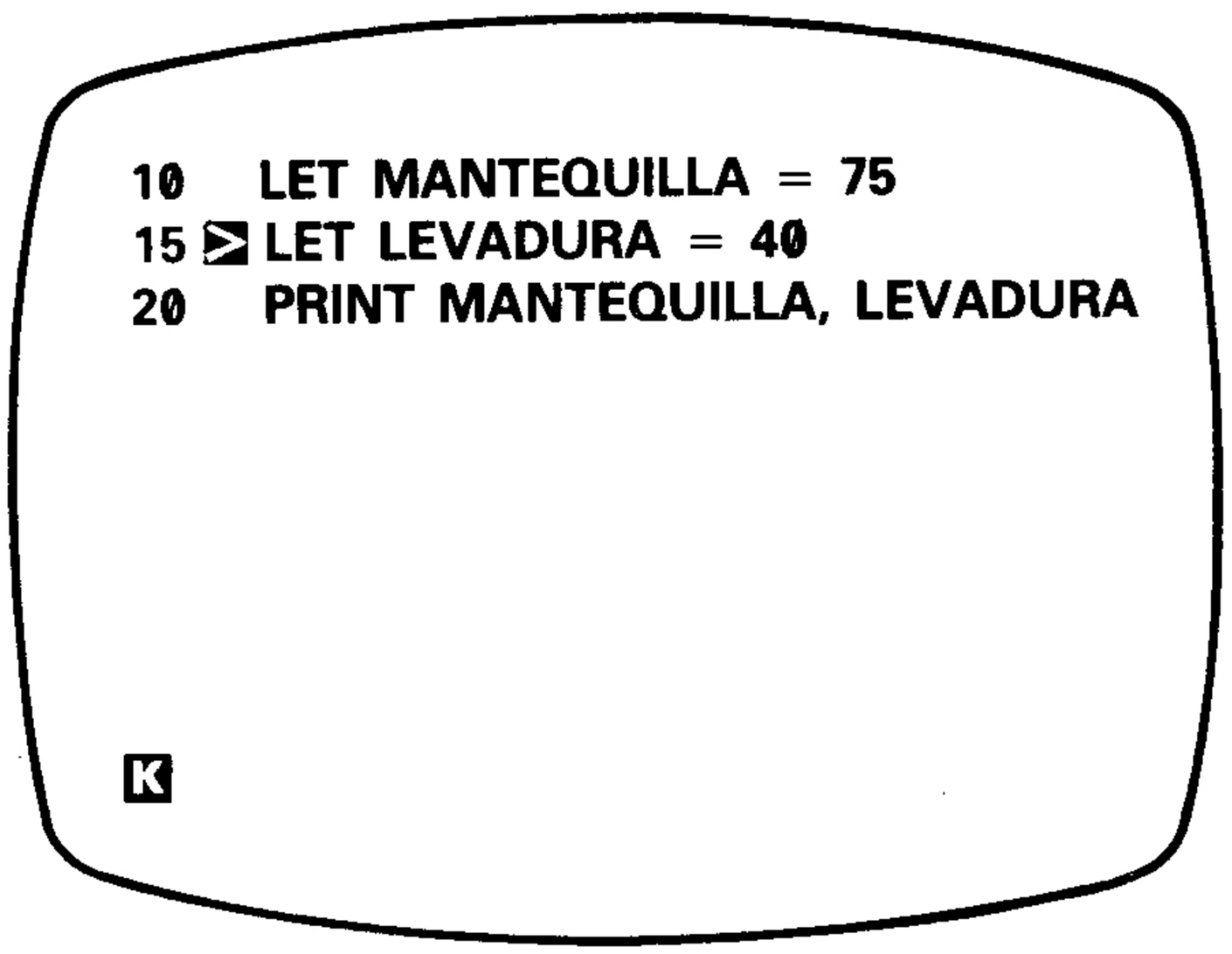
Ud. podría hacerlo volviendo a escribir toda la línea, pero existe un procedimiento para utilizar lo que ya hay. ¿Ve Ud. esa pequeña ▶ en la línea 15? Es el cursor de programa, y la línea a la que apunta es la línea en la que se está escribiendo (línea actual). Pulse la tecla ⇨ (6 cambiada) y se moverá hasta la línea 20. La tecla ⇦ la mueve hacia arriba. Ahora pulse la tecla **EDIT,** y en la parte inferior de la pantalla le aparecerá una copia de la línea 20. Pulse la tecla ⇩ 7 veces hasta que el cursor ▮ llegue al final de la línea, y escriba

,LEVADURA (sin **NEWLINE**)

La línea en la parte inferior de la pantalla dirá:

20 PRINT MANTEQUILLA, LEVADURA ▮

Pulse **NEWLINE** y ésta línea sustituirá a la antigua linea 20. Ahora la pantalla se verá así:



RUN éste programa y aparecerán representados ambos precios.

He aquí un truco muy útil empleando **EDIT,** a usar cuando Ud. quiera borrar totalmente la parte inferior de la pantalla. Pulse **EDIT** y la línea actual será bajada del programa sustituyendo a la que queríamos borrar. Si ahora pulsa **NEWLINE,** la línea regresará al programa, sin introducir ningún cambio y la parte inferior de la pantalla aparcecerá limpio, permaneciendo solo el cursor.

Ahora, en un ataque de distracción, escriba

12 **LET** LEVADURA = 40

Esto irá al programa y Ud. se dará cuenta de su error. Para eliminar ésta línea innecesaria, escriba

12 (con **NEWLINE,** desde luego)

Se sorprenderá al observar que el cursor de programa ha desaparecido. Se lo puede imaginar como escondido entre las líneas 10 y 15, de manera que si Ud. pulsa ⌃ subirá hasta la línea 10, mientras que si pulsa ⌄ bajará hasta la 15.

Por último, escriba

LIST 15

Ahora en la pantalla tendrá

```
15 ▶ LET LEVADURA = 40
20   PRINT MANTEQUILLA, LEVADURA
```

La línea 10 ha desaparecido de la pantalla, pero todavía permanece en el programa, como puede comprobar volviendo a pulsar **NEWLINE** de nuevo. Los únicos efectos de **LIST** 15 son producir un listado que comienza en la línea 15 y colocar el cursor de programa en la línea 15.

LIST

por sí solo hace que el listado empiece al principio del programa.

Resumen

Programas
Edición de programas empleando ⌃, ⌄ y **EDIT**
Sentencias: **RUN, LIST**

Ejercicios

1. Modificar el programa para que aparezcan no solo los dos precios sino tambie5n mensajes en los que se diga lo que es cada uno.

2. Utilice la tecla **EDIT** para cambiar el precio de la mantequilla.

3. Ejecute el programa y a continuación escriba

PRINT MANTEQUILLA, LEVADURA

Las variables están allí todavía, aunque el programa haya terminado.

4. Escriba

12 (y **NEWLINE**)

De nuevo, el cursor de programa estará escondido entre las líneas 10 y 15. Ahora pulse **EDIT** y la línea 15 vendrá a la parte inferior. Cndo el cursor de programa está escondido entre dos líneas, **EDIT** lleva abajo la segunda línea. Pulse **NEWLINE** para limpiar la parte inferior de la pantalla.

Ahora escriba

30

Esta vez, el cursor de programa está escondido despues del final del programa, y si Ud. pulsa **EDIT,** la línea 20 será bajada en la pantalla.

5. Introduzca una sentencia **LIST** en el programa de manera que cuando Ud. lo ejecute, se haga el listado por sí mismo.

CAPITULO 9

Continúa la programación del computador

Escriba

NEW

Esto borrará todos los programas y variables antiguas en el ZX81. (Se parece a **CLEAR,** pero **CLEAR** solo borra las variables).

Ahora escriba con cuidadao éste programa:

```
10 REM ESTE PROGRAMA EXTRAE RAICES CUADRADAS
20 INPUT A
30 PRINT A, SQR A
40 GOTO 20
```

(Ud. mismo deberá introducir todos los espacios de la línea 10).

Ahora ejecútelo. Aparentemente la pantalla se ha vuelto blanca y no ocurre nada, pero mire el cursor en la esquina izquierda: donde Ud. esperaba encontrar una K hay una L , la máquina se ha puesto en estado de entrada (de datos). Esto es debido a la sentencia **INPUT** de la línea 20. La máquina está esperando a que Ud. escriba un número (o incluso una expresión), y no continuará hasta que Ud. lo haga. Despues de eso, tendrá el efecto de

20 **LET** A = . . . todo lo que escribió

Sin embargo . . . ¡Un momento! ¿Que ocurrió con la línea 10? Parece como si el computador la hubiese olvidado completamente. En efecto, así ha sido. **REM** en la línea 10 está para advertir o recordar, y está allí solamente para recordarle a Ud. lo que hace el programa. Una sentencia **REM** consiste en **REM** seguida por todo lo que Ud. quiera, pero el computador la ignorará.

De acuerdo, así pues nosotros estamos en estado de entrada para la línea 20. Escriba algún número, digamos el 4, y acontinuación **NEWLINE.** El 4 y su raiz cuadrada aparecen en la pantalla y Ud. podría pensar que eso era todo. Pues no, parece que espera otro número. Esto es debido a la línea 40, **GOTO** 20, que significa exactamente lo que dice (ir a 20). En lugar de ejecutar el programa y pararse, el computador salta a la línea 20 y empieza de nuevo. Así pues, escriba otro número (digamos el 2; o cualquier valor siempre que sea positivo).

Despues de unos cuantos números mas, se preguntará si la maquina no se cansará de éste juego; no se cansará. La próxima vez que en su insaciabilidad pida otro número, escriba **STOP** (A cambiada) en su lugar; él entenderá la indirecta. El computador nos dará el informe D/20, mire el significado de D en la lista de informes (apéndice B). 20 es la línea en la que estaba esperando alguna entrada cuando Ud. lo paró.

¿De repente se ha acordado Ud que tiene más números de los que quiere conocer su raiz cuadrada? Entonces escriba

CONT

(abreviatura de CONTINUE – continuar) y el computador borrará la pantalla y le pedirá a Ud. otro número.

Para **CONT**, el computador recuerda el nùmero de lìnea en el ùltimo informe que le facilitò a Ud. y que tenìa (el informe) un còdigo distinto de Ø y salta a esa línea. Como el ùltimo informe fùé D/2Ø (y D no es Ø), en nuestro caso **CONT** es idéntico a **GOTO** 2Ø.

Ahora escriba números hasta que la pantalla empiece a llenarse. Cuando esté llena, el computador se detendrá dándo el informe 5/3Ø. 5 significa "pantalla llena" y 3Ø es el número de sentencia **PRINT** que estaba tratando de ejecutar cuando descubrió que no tenía sitio. De nuevo

CONT

borrará la pantalla y continuará; ésta vez, **CONT** significa **GOTO** 3Ø.

Observe que la pantalla se borra, no porque sea una sentencia **CONT,** sino porque es una órden. Todas las órdenes (excepto **COPY,** que aparece en el capítulo 2Ø) borran en primer lugar la pantalla.

Cuando Ud. esté cansado de ésto, detenga el programa usando **STOP** y obtenga el listado pulsando **NEWLINE.**

Mire la sentencia **PRINT** de la línea 3Ø. Cada vez que se imprimen un par de números A y **SQR**A, se hace en una nueva línea y esto es así, porque la sentencia **PRINT** no termina con una coma o un punto y coma. Siempre que sea éste el caso, la siguiente sentencia **PRINT** empieza a imprimirse en una nueva línea. (Así pues, para introducir una línea en blanco, utilice una sentencia **PRINT** en la que no haya escrito nada, solamente **PRINT**).

No obstante, una sentencia **PRINT** puede acabar en una coma o en un punto y coma, y entonces la próxima sentencia **PRINT** empieza a imprimirse como si las dos fuesen una sentencia larga.

Por ejemplo, con comas, sustituya la 3Ø por

3Ø **PRINT** A,

y ejecute el programa para ver como sentencias **PRINT** sucesivas se imprimen en la misma línea pero separadas en dos columnas.

Por otro lado, con puntos y comas, con

3Ø **PRINT** A;

todas se ponen juntas.
Pruebe también

```
30 PRINT A
```

Ahora escriba éstas líneas extra

```
100 REM ESTE PROGRAMA MIDE CADENAS
110 INPUT A$
120 PRINT A$, LEN A$
130 GOTO 110
```

Este es un programa separado del anterior, pero puede almacenar los dos al mismo tiempo. Para ejecutar el nuevo, escriba

```
RUN 100
```

Este programa, en lugar de un número introduce una cadena, la imprime y da su longitud. Escriba

```
PAN          (y NEWLINE, como de costumbre)
```

Como el computador está esperando una cadena, imprime dos comillas de cadena, lo que constituye un recordatorio para Ud. y normalmente le evita el tener que escribirlas. Pero Ud. no tiene que limitarse a encadenar constantes (cadenas explícitas con comillas de apertura y cierre y todos sus caracteres intermedios); el computador puede calcular cualquier expresión encadenada, como puede ser una con variables encadenadas. En éste caso, debe borrar las comillas que el computador ha representado. Inténtelo. Bórrelas (con ◊ y RUBOUT dos veces), y escriba

```
A$
```

Puesto que A$ todavía tiene el valor "PAN", de nuevo la respuesta es PAN 3.
Ahora introduzca de nuevo

```
A$
```

pero ésta vez sin borrar las comillas de cadena. Ahora A$ tiene el valor "A$" y la respuesta es 2.

Sie quiere utilizar **STOP** para la introducción de cadenas, debe retrasar primero el cursor al comienzo de la línea, utilizando ◊.

Ahora volvamos a ese **RUN** 100 que teníamos antes. Ese salto precisamente a la línea 100, ¿no se podría haber dicho **GOTO** 100? En éste caso, ocurre que la respuesta es

afirmativa, pero hay una diferencia. **RUN** 10Ø ante todo borra las variables (como **CLEAR** en el capítulo 6) y despues de eso trabaja exactamente igual como **GOTO** 10Ø. **GOTO** 10Ø no borra nada. Puede muy bien haber ocasiones en que Ud. quiera realizar un programa sin borrar ninguna variable y en ese caso **GOTO** es necesario y **RUN** sería desastroso, por consiguiente es mejor no adquirir el hábito de escribir automáticamente **RUN** para realizar un programa.

Otra diferencia es que se puede escribir **RUN** sin número de línea y empieza en la primera línea del programa. **GOTO** siempre necesita un número de línea.

Estos dos programas se detienen porque Ud. escribió **STOP** en la línea **INPUT;** pero algunas veces, por error, Ud. escribe un programa que no puede detener y no se parará por sí solo. Escriba

```
2ØØ GOTO 2ØØ
RUN 2ØØ
```

Parece que se ha dispuesto para que este corriendo eternamente, a menos que Ud. tire de la clavija; pero hay un remedio menos drástico. Pulse la tecla **SPACE,** que tiene escrito "BREAK" sobre ella. El programa se detendrá, apareciendo D/2ØØ.

Al final cada línea de programa, el computador mira a ver si ésta tecla está pulsada, y si ello es así se detiene. La tecla **BREAK** también se puede utilizar cuando Ud. está en pleno uso del magnetofón o del impresor.

Hasta ahora, Ud. ha visto las sentencias **PRINT, LET, INPUT, RUN, LIST, GOTO, CONT, CLEAR, NEW** y **REM,** y que la mayoría de ellas se pueden emplear o como ordenes o como líneas de programa, lo que es cierto para casi todas las sentencias en BASIC. Realmente la única exceptión es **INPUT,** que no se puede usar como una orden (Ud. conseguirá un informe 8 si lo intenta; la razón es que dentro del computador se utiliza la misma zona para órdenes que para entrada de datos y una orden **INPUT** ocasionaría problemas). Normalmente **RUN, LIST, CONT, CLEAR,** y **NEW** no se utilizan mucho en un programa, pero se pueden utilizar.

Resumen

Sentencias: GOTO, CONT, INPUT, NEW, REM, PRINT
STOP en entrada de datos
BREAK

Ejercicios

1. En el programa de la raiz cuadrada, intente sustituir la línea 4Ø por **GOTO** 5, **GOTO** 1Ø o **GOTO** 15; no debería introducir variaciones apreciales en la realización del programa, En una sentencia **GOTO,** si el numero de línea se refiere a una línea no existente, entonces salta a la siguiente línea más próxima. Lo mismo ocurre con **RUN;** de hecho **RUN** por si sola significa realmente **RUN** Ø.

2. Ejecute el programa de longitud de cadena y cuando le pida una entrada, escriba

X$ (despúes de borrar las comillas)

Desde luego, X$ es una variable no definida y tendrá el informe 2/11Ø.

Si ahora Ud. escribe

LET X$ = "ALGO DISTINTO"

(que tiene su propio informe ØØ) y

CONT

encontrará que puede usar X$ como dato de entrada sin ningún problema.

Lo importante de éste ejercicio es que **CONT** tiene el efecto de **GOTO** 11Ø. No hace caso del informe ØØ porque tiene el código Ø y toma su número de línea del informe anterior 2/11Ø. Se pretende que sea útil. Si un programa se detiene debido a algún error, entonces Ud. puede hacer todo lo necesario para corregirlo y **CONT** le hará seguir trabajando despues.

3. Plantee éste programa:

```
10 INPUT A$
20 PRINT A$; " = "; VAL A$
30 GOTO 10
```

(compárese con el ejercicio 1 del capítulo 7)

Ponga en sentencias PRINT adicionales, lo que el computador anuncia que va a hacer y pregunte con extrema cortesía por la cadena de entrada.

4. Escriba un programa para introducir precios y obtener el correspondiente impuesto de valor añadido (IVA) al 15%. De nuevo, póngalo en sentencias **PRINT** de tal manera que el computador le diga lo que está haciendo. Modifique el programa de tal modo que pueda también introducir el valor del IVA (para permitir el valor cero o futuros presupuestos).

5. Escriba un programa para imprimir una suma contínua de los números que Ud. introduzca (Una sugerencia: Tiene dos variables TOTAL, puesta a Ø para empezar, e ITEM. Introduzca ITEM, añádalo a TOTAL, imprima ambos y empiece de nuevo).

6. Los listados automáticos (aquellos que no son el resultado de una sentencia **LIST**) bien pueden tenerle a Ud. perplejo. Si escribe en un programa con 50 líneas, todo sentencias **REM**,

```
1 REM
2 REM
3 REM
: :
49 REM
50 REM
```

entonces podrá experimentar.

La primera cosa a tener en cuenta es que la línea actual (con ▶) siempre aparecerá en la pantalla, y preferiblemente en el centro.
Escriba

LIST (y **NEWLINE,** por supuesto)

y acontinuación **NEWLINE** de nuevo. En la pantalla deberán aparecer las líneas 1 a 22. Ahora escriba

23 **REM**

y en la pantalla deberá tener las líneas 2 a 23; escriba

48 **REM**

y obtendrá las líneas 27 a 48. (En ambos casos, al escribir una nueva línea, Ud. ha movido el cursor de programa de manera que se ha confeccionado un nuevo listado).

¿Le parecè un poco arbitrario todo esto? En realidad, está tratandó de darle exactamente lo que le pide, aunque, siendo los humanos criaturas imprevisibles, no siempre acierte.

El computador guarda un registro no solo de la línea actual, la que tiene que aparecer en la pantalla, sino también la línea mas alta de la pantalla. Cuando él trata de hacer un listado, lo primero que hace es comparar la línea superior con la línea actual.

Si la línea superior viene después, entonces no es el punto en donde se deba empezar, por lo que utiliza la línea actual como una nueva línea superior y hace su listado.

Por lo demás, primero trata de hacer el listado empezando en la línea superior. Si la línea actual sigue en la pantalla, entonces todo está bién; si la línea actual está justamente fuera al pie de la pantalla, entonces traslada la línea superior un lugar y lo intenta de nuevo; y si la línea actual está alejada del pie de la pantalla, cambia la linea superior para convetirla en la línea anterior a la línea actual.

Experimente a cerca del movimiento de la línea actual, escribiendo

número de línea **REM**

LIST mueve la línea actual pero no la línea superior, por lo que los listados posteriores pueden ser diferentes. Por ejemplo, escriba

LIST

para conseguir el listado **LIST,** y continuación pulse **NEWLINE** otra vez para convertir la línea 0 en la línea superior. Deberá tener las líneas 1 a 22 en la pantalla. Escriba

LIST 22

lo que le dará las líneas 22 a 43; cuando pulse **NEWLINE** de nuevo, volverá a las líneas 1 a 22. Esto tiende a ser mas útil para programas cortos que para largos.

7. ¿Que harían **CONT, CLEAR** y **NEW** en un programa? Puede Ud. pensar siquiera un poco en algunos usos?

CAPITULO 10

Si . . .

Todos los programas que hemos visto hasta aquí han sido facilmente predecibles, funcionan directamente siguiendo al pie de la letra todas las instrucciones, y es posible que lo repitieran de nuevo. Esto no es útil. En la práctica se espera que el computador tome decisiones y actúe en consecuencia; lo hace empleando la sentencia **IF.**

Borre el computador (usando **NEW**), y escriba y ejecute este pequeño programa terriblemente divertido:

```
 10 PRINT "TE CUENTO UN CHISTE?"
 20 INPUT A$
 30 IF A$ = "NO GRACIAS" THEN GOTO 200
 40 PRINT "CUANTO SUMAN UNA VIEJA CANSADA Y UN ABANICAO
    ROTO?"
 50 INPUT SUMA
 60 IF SUMA = 150 THEN GOTO 100
 70 PRINT "NO, 150. LA VIEJA CANSADA SE" "SIENTA, Y EL ABANICO
    NO VIENTA"
 80 STOP
100 PRINT "SI." ,, "TE LO CUENTO OTRA VEZ?"
110 GOTO 20
200 PRINT "CONFORME, ENTONCES, NO LO HARE"
```

Antes de estudiar la sentencia **IF,** fíjese primero en la sentencia **STOP** en la línea 80: una sentencia **STOP** detiene la ejecución del programa, dando un informe 9.

Como Ud. puede ver, una sentencia **IF** toma la forma

IF condición **THEN** sentencia

En el ejemplo las sentencias son sentencias **GOTO,** pero pueden ser de cualquier tipo, incluso mas sentencias **IF.** La condición es algo que está en vias de ser calculado de manera que puede resultar verdadero o falso; si resulta verdadero se ejecuta la sentencia que sigue a **THEN,** de lo contrario se omite.

Las condiciones mas utilizadas comparan dos números o dos cadenas: pueden comprobar si dos números son iguales, o si uno es mayor que otro; pueden comprobar si dos cadenas son iguales o si una va antes que otra en orden alfabético. Se usan las relaciones =, <, >, <=, > = y <>.

=, que hemos usado dos veces en el programa (una para números y otra para cadenas) significa "igual a". No es lo mismo que el = en una sentencia **LET.**

< significa "es menor que", así

1<2
–2<–1
y –3<1

son todas <u>verdad</u>, pero

1<0
y 0<–2

son <u>falsas</u>.

Para ver como funciona esto, permitanos escribir un programa para introducir números y representar el mayor hasta el momento.

```
10 PRINT "NUMERO", "EL MAYOR HASTA EL MOMENTO"
20 INPUT A
30 LET EL MAYOR = A
40 PRINT A, EL MAYOR
50 INPUT A
60 IF EL MAYOR <A THEN LET EL MAYOR = A
70 GOTO 40
```

La parte crucial es la línea 60 que actualiza EL MAYOR si el valor viejo es mas pequeño que el nuevo número introducido A.

> (M cambiada) significa "es mayor que", y es como < pero en sentido contrario. Ud. puede recordar cual es cual, porque el vértice señala al número que se supone que es el mas pequeño.

< = (R cambiada, no lo escriba como < seguido por =) significa "es menor que o igual a", así es como < excepto que es válido incluso si los dos números son iguales: así 2<= 2 vale, pero 2<2 no.

> = (Y cambiada) significa "es mayor que o igual a" y es similar a >.

< > (T cambiada) significa "no es igual a" y significa lo contrario a =.

Estas seis operaciones de relación tienen prioridad 5.

Los matemáticos usualmente escriben los símbolos < =, > = y <> como $\leqslant$, $\geqslant$ y $\neq$. También escriben cosas como "2 < 3 < 4" para decir "2 < 3" y "3 < 4", pero esto no es posible en BASIC.

Estas relaciones se pueden combinar utilizando las <u>operaciones lógicas</u> **AND, OR** y **NOT.**

una relación **AND** otra relación

es verdad siempre que ambas relaciones sean verdad.

una relación **OR** otra

es verdad siempre que una de las dos relaciones sea verdad (o ambas).

NOT relación

es verdad siempre que la relación sea falsa, y es falsa siempre que la relación sea verdad.

Se pueden construir expresiones lógicas con relaciones y **AND, OR** y **NOT** igual que las expresiones numéricas se pueden hacer con números y +, –, etc.; incluso se pueden utilizar paréntesis, si es necesario. **NOT** tiene prioridad 4, **AND** 3 y **OR** 2.

Puesto que (a diferencia de otras funciones) **NOT** tiene una prioridad bastante baja, su argumento no necesita paréntesis a menos que contenga **AND** o **OR**; así **NOT** A = B significa **NOT** (A = B) (o lo que es lo mismo A < > B).

Para ilustrar ésto, borre la memoria del computador y plantee este programa

```
10 INPUT F$
20 INPUT EDAD
30 IF F$ = "S" AND EDAD <18 OR F$ = "M" AND EDAD <14
THEN PRINT
"NO AUTORIZADA";
40 PRINT "PUEDÉ PASAR."
50 GOTO 10
```

Aquí se supone que F$ es la clasificación de una película, S para 18 años o mas, M para 14 años o mas y T O L para todos, y el programa calcula si una persona de una determinada edad está autorizada para ver la película.

Por último, podemos comparar no solamente números sino también cadenas. Hemos visto "=" utilizado en "F$="S"", e incluso puede Ud. utilizar las otras cinco, <, etc.

Pero ¿que significado tiene "menor que" hablando de cadenas? Una cosa que no quiere decir es "mas corta que", así que no cometamos ese error. Nosotros hacemos la definición de que una cadena es menor que otra si está la primera en orden alfabético, por consiguiente

```
"SANCHEZ"       < "SANZ"
"SANZ"          > "SANCHEZ"
"CALVO"         < "CALVO-SOTELO"
"BILLON"        < "MILLON"
"TCHAIKOVSKY"   < "WAGNER"
"DURO"          < "PESETA"
```

todas son válidas. < = significa "es menor que o igual a", y así sucesivamente, igual que para los números.

Nota: En algunos dialectos de BASIC, pero no en el ZX81, la sentencia **IF** puede tener la forma

IF condición **THEN** número de línea

lo que significa lo mismo que

IF condición **THEN GOTO** número de línea

Resumen

Sentencias: **IF, STOP**
Operaciones: =, <, >, < =, > =, <>, **AND, OR**
Función: **NOT**

Ejercicios

1. <> y = son opuestos en el sentido de que **NOT** A = B es lo mismo que A <> B
y
NOT A <> B es lo mismo que A = B

Convénzase Ud, mismo de que > = es opuesto a <, y que < = es opuesto a > para que Ud. pueda librarse siempre de **NOT** al principio de una relación mediante el cambio de esa relación por su opuesta.

También

NOT (una primera expresión lógica **AND** una segunda)

es lo mismo que

NOT (la primera) **OR NOT** (la segunda)

y

NOT (una primera expresión lógica **OR** una segunda)

es lo mismo que

- **NOT** (la primera) **AND NOT** (la segunda).

Utilizando esto, puede manejar **NOTs** por medio de paréntesis hasta que finalmente todas estén aplicadas a relaciones y entonces pueda Ud. librarse de ellos. Así pues, hablando en lógica, **NOT** es innecesario. Aunque puede Ud. considerar que utilizándolo hace un programa mas claro.

2. Algunas veces BASIC puede trabajar por caminos distintos de nuestro idioma. Por ejemplo, considere la frase: "Si A no es igual a B o a C". ¿Podría Ud. escribirla en BASIC? (La respuesta no es

"**IF** A <> B **OR** C" ni "**IF** A <> B **OR** A <> C")

No se preocupe si no entiende los ejercicios 3, 4 y 5, ya que los aspectos estudiados en ellos son bastante refinados.

3. (Para expertos)
Plantee

PRINT 1 = 2, 1 <> 2

lo que ya puede esperar que le de un error de sintaxis. De hecho, por lo que al computador respecta, no existe tal cosa como valor lógico.

(i) =, <, >, <=, >= y <> son todas ellas operaciones binarias con valor numérico de prioridad 5. El resultado es 1 (por verdad) si la relación vale, y Ø (por falso) si no vale.

(ii) En

IF condición **THEN** sentencia

la condición puede ser en realidad cualquier expresión numérica. Si su valor es Ø entonces cuenta como falsa y si es cualquier otro valor como verdad. Así pues la sentencia **IF** significa exactamente lo mismo que

IF condición <> Ø **THEN** sentencia

(iii) **AND, OR** y **NOT** tambien son operaciones con valor numérico.

X **AND** Y tiene el valor { X si Y es no-cero (contando como verdad); Ø si Y es cero (contando como falso)

X **OR** Y tiene el valor { 1 si Y es no-cero; X si Y es cero

y

NOT X tiene el valor { Ø si X es no-cero; 1 si X es cero

A la luz de ésta revelación, léase todo el capítulo de nuevo, asegurándose de que todo funciona.

En las expresiones X **AND** Y, X **OR** Y y **NOT** X, por lo general X e Y toman cada una el valor Ø o 1, como falso o verdad. Prepare las diez combinaciones posibles y compruebe que hacen lo que Ud. espera que hagan **AND, OR** y **NOT.**

4. Prepare éste programa:

```
10 INPUT A
20 INPUT B
30 PRINT (A AND A > = B) + (B AND A < B)
40 GOTO 10
```

Cada vez él imprime el mayor de los dos números A y B ?Por qué? Convenzase Ud. mismo de que puede recordar que

X **AND** Y

significa

"X si Y (en otro caso el resultado es Ø)"

y que

X **OR** Y

significa

"X a no ser que Y (en cuyo caso el resultado es 1)"

Una expresión como ésta en la que se utilizan **AND** y **OR** se llama una expresión condicional. Un ejemplo utilizando **OR** podría ser

LET PRECIO VENTA FINAL = PRECIO DE VENTA * (1.15 **OR** V$ = "ARTICULO SIN IMPUESTO IVA").

En este caso si V$ = "ARTICULO SIN IMPUESTO IVA", el precio de venta final no incluye impuesto y si no se cumple esta condición, este precio-final se gravará con el 15% correspondiente al IVA.

Observe como **AND** tiende a ir con sumas (porque su valor de omisión es Ø), y **OR** tiende a ir con multiplicaciones (porque su valor de omisión es 1).

5. Tambien puede Ud. construir expresiones condicionales con valor de cadena, pero utilizando **AND** solamente.

X$ AND Y$ tiene el valor { X$ si Y es no-cero / si Y es cero

así pues, significa "X$ si Y (en otro caso la cadena vacía)".

Plantee éste programa que introduce dos cadenas y las coloca en orden alfabético.

```
10 INPUT A$
20 INPUT B$
30 IF A$< = B$ THEN GOTO 70
40 LET C$ = A$
50 LET A$ = B$
60 LET B$ = C$
70 PRINT A$;" ";("<" AND A$ < B$) + ("=" AND A$ = B$);" "; B$
80 GOTO 10
```

6. Plantee éste programa

```
10 PRINT "X"
20 STOP
30 PRINT "Y"
```

Cuando Ud. lo ejecute, representará "X" y se detendrá con informe 9/20. Ahora escriba.

CONT

Puede suponer que esto actúa como "**GOTO** 20", solo que el computador se acaba de parar de nuevo sin representar "Y"; pero esto no sería muy útil, por lo que las cosas estan preparadas paradar un informe 9 (sentencia **STOP** ejecutada); el número de línea se incrementa en una unidad para una sentencia **CONT.** Así pues en nuestro ejemplo, "**CONT**" se comporta como "**GOTO** 21" (que como no existen líneas entre 20 y 30 se convierte en "**GOTO** 30").

7. Muchos dialectos de BASIC (pero no el BASIC del ZX81) tienen una sentencia ON, que toma la forma

ON expresión numérica GOTO número de línea, número de línea, . . . número de línea

Aquí se calcula la expresión numérica; supongamos que su valor es n con lo que el efecto sería

GOTO a la línea número n

Por ejemplo,

ON A GOTO 100, 200, 300, 400, 500

Aquí, si A tiene el valor 2, entonces se ejecuta "GOTO 200". En ZX81 BASIC, esto se puede sustituir por

GOTO 100*A

En el caso en que los números de línea no sean centenas exactas como éstas, resuelva como usaría

GOTO una expresión condicional

en su lugar.

CAPITULO 11

El juego de caracteres

Las letras, los dígitos, los signos de puntuación y demás cosas que pueden aparecer en las cadenas se llaman caracteres, y constituyen el alfabeto o juego de caracteres que utiliza el ZX81. La mayoría de estos caracteres son símbolos aislados, pero hay algunos mas, llamados señales, que representan palabras enteras, tales como **PRINT, STOP, ****, etc.

En total hay 256 caracteres, y cada uno tiene un código entre 0 y 255. En el apéndice A hay una lista completa de ellos, Para convertir códigos y caracteres entre si, hay dos funciones **CODE** y **CHR$.**

CODE se aplica a una cadena y da el código del primer caracter de la cadena (o 0 si la cadena está vacía).

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

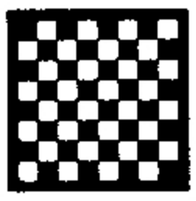
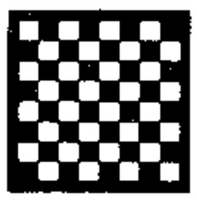
Este programa imprime el juego de caracteres completo.

```
10 LET A = 0
20 PRINT CHR$ A;
30 LET A=A + 1
40 IF A < 256 THEN GOTO 20
```

Al principio puede ver los símbolos '', £, $, y así hasta Z, que estan todos en el teclado y que pueden escribirse cuando Ud. tiene el cursor **L** . Además, Ud. puede ver los mismos caracteres, pero en blanco sobre fondo negro (imagen en negativo); estos tambien se pueden obtener desde el teclado. Si pulsa **GRAPHICS** (9 cambiada) el cursor se cambiará a **G** : esto significa trabajar en gráficos. Si escribe un símbolo aparacerá su imagen en negativo y permanecerá hasta que Ud. pulse nuevamente **GRAPHICS** o **NEWLINE. RUBOUT** tendrá su significado habitual. Tenga cuidado, no vaya a perder el cursor **G** entre todos los caracteres en negativo que Ud. acaba de escribir.

Cuando haya experimentado un poco, tendrá todavía el juego de caracteres al principio; si no es así, vuelva a ejecutar el programa. Justo al principio hay un espacio y diez dibujos en negro, blanco y motitas grises; además de estos hay otros once. Estos son los llamados símbolos gráficos y se utilizan para trazar dibujos. Puede introducirlos mediante el teclado utilizando de nuevo el trabajo en gráficos (excepto para el espacio, que es un símbolo ordinario utilizando el cursor **L** ; el cuadrado negro es la inversa del espacio). Utilice las 20 teclas que tienen símbolos gráficos escritos sobre ellos. Por ejemplo, suponga que quiere el símbolo▚ que está en la tecla T. Pulse **GRAPHICS** para obtener el cursor **G** , y a continuación pulse T cambiada. De la descripción anterior del trabajo en gráficos, Ud. podría esperar conseguir un símbolo en negativo; pero la T cambiada normalmente es <>, una señal, y las señales no tienen inversas por lo que Ud. consigue en su lugar el símbolo ▞.

Aquí están los 22 símbolos gráficos.

Símbolo	*Código*	*Como Obtenerlo*	*Símbolo*	*Código*	*Obtenerlo*
	0	**K** o **L** **SPACE**		128	**G** **SPACE**
	1	**G** 1 cambiado		129	**G** Q cambiada
	2	**G** 2 cambiada		130	**G** W cambiada
	3	**G** 7 cambiada		131	**G** 6 cambiada
	4	**G** 4 cambiada		132	**G** R cambiada
	5	**G** 5 cambiada		133	**G** 8 cambiada
	6	**G** T cambiada		134	**G** Y cambiada
	7	**G** E cambiada		135	**G** 3 cambiada
	8	**G** À cambiada		136	**G** H cambiada
	9	**G** D cambiada		137	**G** G cambiada
	10	**G** S cambiada		138	**G** F cambiada

Veamos de nuevo el juego de caracteres en pantalla. Las señales aparecen bastante claramente separadas en dos bloques: un pequeño grupo de tres (**RND, INKEY$** y **PI**) despues de la Z, y un grupo mas grande (empezando con la imagen de comillas despues de **Z** y continuando con **AT** hasta **COPY**).

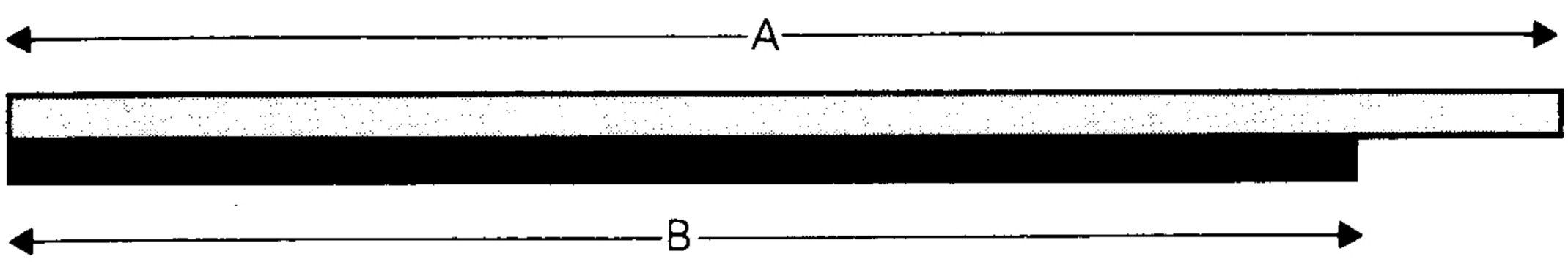
El resto de los caracteres parece ser todo signos de interrogación. Esta es en la realidad la manera en que se obtienen impresos; el auténtico signo de interrogación está entre: y (. De los aparentes, algunos son para caracteres de control como ◊, **EDIT** y **NEWLINE,** y el resto son para caracteres que no tienen ningún significado especial para el ZX81.

Resumen

Funciones: **CODE, CHR$**

Ejercicios

1. Imagínese el espacio del símbolo de una tecla dividido en cuatro cuartos: ⊞. Si cada cuarto puede ser negro o blanco, existen 2*2*2*2 = 16 posibilidades. Búsque todas ellas en el juego de caracteres.
2. Imagínese el espacio de un símbolo dividido en dos horizontalmente ⊟. Si cada mitad puede ser negra, blanca o gris, existen 3*3 = 9 posibilidades. Búsquelas.
3. Los caracteres del ejercicio 2 están destinados a ser utilizados en gráficos de barras horizontales, empleando dos colores, gris y negro. Escriba un programa que introduzca dos números A y B (ambos entre 0 y 32) y dibuje un gráfico de barras para ellos:



Necesitará empezar por imprimir "▓" y cámbielo a "▒" o "▄", según que A sea mayor, igual o menor que B.

¿Que hace su programa si A y B no son números enteros? ¿O si no están dentro del intervalo de 0 a 32? Un buen programa, "amable con el usuario" es la expresión de moda, hará algo sensato y útil.

4. En el teclado, en la A y en la H, hay dos caracteres distintos totalmente grises. Si los mira muy de cerca, verá que el de la H es como un tablero de ajedrez en miniatura, mientras que el de la A es como un tablero de ajedrez visto desde un costado. Imprímalos uno al lado del otro, tocándose, y verá que no casan. con ▒ y ▄ (en S yD), mientras que el de la H es para casar con ▓ y ▄ (en F y G).

5. Ejecute este programa:

```
10 INPUT A
20 PRINT CHR$ A;
30 GOTO 10
```

Si Ud. experimenta con él, encontrará que para **CHR$,** A es redondeado al número entero mas próximo, y si A no está comprendido entre 0 y 255, el programa se detiene dando un informe B.

6. Utilizando los códigos de caracteres, podemos ampliar el concepto de "orden alfabético" para cubrir cadenas que contengan cualquier tipo de caracteres, no solo letras. Si en lugar de pensar en términos del alfabeto de 26 caracteres, utilizamos el alfabeto ampliado de 256 caracteres, en el mismo orden en que estan sus códigos, entonces el principio es exactamente el mismo. Por ejemplo, éstas cadenas están en el orden alfabético del ZX81.

```
"  ZACARIIAS"
"▗"
"(ASPA)"
"123 RESPONDA OTRA VEZ"
"AASVOGEL"
"AARDVARK"
"ZACARIAS"
"AARDVARK"
```

Esta es la regla. En primer lugar compare los primeros caracteres de la dos cadenas. Si son distintos, uno de ellos tendrá su código inferior al otro y la cadena que tenga ese primer caracter es la anterior (mas pequeña) de las dos cadenas. Si son iguales, entonces se comparan los caracteres siguientes. Si en éste proceso una de las cadenas se acaba antes que la otra, entonces esa cadena es la anterior; de otro modo serían iguales.

Escriba de nuevo al programa del ejercicio 5 del capítulo 10 (el que introduce dos cadenas y las imprime ordenadamente) y utilícelo para experimentar.

7. Este programa imprime toda una pantalla de caracteres gráficos blancos y negros al azar:

```
10 LET A=INT (16*RND)
20 IF A>=8 THEN LET A=A + 120
30 PRINT CHR$ A;
40 GOTO 10
```

(¿Como trabaja?)

CAPITULO 12

Bucles

Supongas que Ud. quiere introducir cinco números y sumarlos. Un procedimiento (no lo siga a menos que se vea obligado) es escribir

```
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 no constituye una buena práctica en programación. Puede ser bastante controlable para cinco números, pero se puede Ud. imaginar lo pesado que sería sumar diez números con un programa como éste, y un centenar sería imposible.

Es mucho mejor establecer una variable para contar hasta cinco y detener el programa, como lo siguiente (que Ud. debería escribir):

```
10 LET TOTAL = 0
20 LET CUENTA = 1
30 INPUT A
40 REM CUENTA = NUMERO DE VECES QUE HA SIDO INTRODUCIDO
   HASTA AHORA
50 LET TOTAL = TOTAL + A
60 LET CUENTA = CUENTA + 1
70 IF CUENTA < = 5 THEN GOTO 30
80 PRINT TOTAL
```

Observe que fácil sería cambia la línea 70 para que éste programa sume diez números o incluso un centenar.

Esta manera de calcular es tan útil que hay dos sentencias especiales para hacerlo más facil: la sentencia **FOR** y la sentencia **NEXT** que siempre se usan juntas. Utiliźandolas, el programa que acaba de escribir es exactamente el mismo que

```
10 LET TOTAL = 0
20 FOR C = 1 TO 5
30 INPUT A
40 REM C = NUMERO DE VECES QUE A HA SIDO INTRODUCIDO HASTA
   AHORA
50 LET TOTAL = TOTAL + A
60 NEXT C
80 PRINT TOTAL
```

(Para conseguir éste programa a partir del anterior, Ud. solamente tiene que editar las líneas 20, 40, 60, y 70. **TO** es 4 cambiada).

Observe que hemos cambiado CUENTA por C. La variable de cálculo, o variable de control de un bucle **FOR-NEXT** tiene que tener como nombre una sola letra.

El efecto de éste programa es que C lee los valores 1 (el valor inicial), 2, 3, 4, y 5 (el límite), y para cada uno se ejecutan las líneas 30, 40 y 50, y cuando C ha agotado sus cinco valores, se ejecuta la línea 80.

Una sutileza extra es que la variable de control no tiene que incrementarse en 1 cada vez: Ud. puede cambiar éste 1 por lo que desee utilizando una parte **STEP** en la sentencia **FOR.** La forma más general de una sentencia **FOR** es:

FOR variable de control = valor inicial **TO** límite **STEP** incremento, donde la variable de control es una sola letra, y el valor inicial el límite y el incremento son expresiones numéricas. Así, si Ud. reemplaza la línea 20 del programa por

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

C leerá los valores 1, 2.5 y 4. Note que Ud. no tiene porqué limitarse a números enteros, y también, que el valor de control no tiene que llegar exactamente al límite, él continuará haciendo el bucle mientras sea inferior o igual al límite (si no, vea el ejercicio 4).

Debe de tener cuidado si está ejecutadno dos bucles **FOR-NEXT** juntos, uno dentra del otro. Pruebe éste programa que imprime un juego completo de dominó de 6 puntos.

```
10 FOR M = 0 TO 6
20 FOR N = 0 TO M
30 PRINT M; ":";N;"  ";      } Bucle N
40 NEXT N                    } Bucle M
50 PRINT
60 NEXT M
```

Puede ver que el bucle N está completamente dentro del bucle M, están encajados adecuadamente. Lo que se debe evitar son dos bucles **FOR-NEXT** que se solapen sin que uno esté completamente dentro del otro, como lo siguiente:

```
          10 FOR M = 0 TO 6      }
          20 FOR N = 0 TO M      } Bucle M   }
          30 PRINT M;";";N;" ";  }           }
ERRONEO   40 NEXT M                          } Bucle N
          50 PRINT                           }
          60 NEXT N                          }
```

Dos bucles **FOR-NEXT** deberán estar o el uno dentro del otro o completamente separados.

Otra cosa que se debe evitar es saltar desde fuera en medio de un bucle **FOR-NEXT.** La variable de control solamente se encuantra a gusto cuando se ejecuta su sentencia **FOR,** y si Ud. olvida esto la próxima sentencia **NEXT** desconcertará al computador, y conseguirá un informe 1 o 2 de error (diciéndole que una sentencia **NEXT** no contiene una variable de control reconocida).

Resumen

Sentencias: **FOR, NEXT**
TO, STEP

Ejercicios

1. Vuelva a escribir el programa del capítulo 11 que imprime el juego de caracteres, utilizando un bucle **FOR-NEXT** (Respuesta en el capítulo 13).
2. Una variable de control no tiene solamente un nombre y un valor, como una variable cualquiera, sino tambien un límite, un incremento y un número de línea para volver al bucle (la línea despues de la sentencia **FOR** donde fué iniciado). Cerciórese Ud. en primer lugar de que cuando se ejecuta la sentencia **FOR** está disponible toda ésta información (empleando el valor inicial como el primer valor que toma), y despues, que (utilizando como un ejemplo nuestros segundo o tercer programa), ésta información es suficiente para convertir la línea única

```
NEXT C
```

en las dos líneas

```
LET C = C + 1
IF C< = 5 THEN GOTO 30
```

(Realmente aquí hemos trampeado un poco, pues **GOTO** 30 debería ser **GOTO** 21. Producirá el mismo efecto en nuestro programa.)

3. Ejecute el tercer programa, y escriba

PRINT C

¿Por qué la respuesta es 6 y no 5?
(Contestación: La sentencia **NEXT** de la línea 60 es ejecutada 5 veces y cada vez se añade 1 a C). ¿Que ocurre si pone **STEP** 2 en la línea 20?

4. Cambie el tercer programa de manera que en lugar de sumar automáticamente cinco números, le pida a Ud. que le diga cuantos números quiere sumar. Cuando Ud. ejecute éste programa, ¿Que ocurre si Ud. contesta 0, queriendo decir que no quiere sumar numeros? ¿Por que puede Ud. esperar que esto cause problemas al computador, incluso cuando está claro lo que Ud. quiere decir? (El computador tiene que hacer una búsqueda de la sentencia **NEXT** C, lo que normalmente no es necesario). En la práctica hay que tener cuidado con todo esto.

5. Plantee éste programa, para imprimir los números del 1 al 10 en orden inverso.

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

Conviértalo en un programa que no haga uso de bucles **FOR-NEXT,** de la misma manera que Ud. convertió el programa 3 en el programa 2 (ver ejercicio 2). ¿Por que el incremento negativo lo hace ligeramente distinto?

CAPITULO 13

Lento y rápido

El ZX81 puede funcionar a dos velocidades, LENTO y RAPIDO. Cuando se enchufa, el computador funciona en modo LENTO y puede calcular y representar información en la pantalla simultáneamente. Este funcionamiento es el ideal para imágenes animadas.

Sin embargo, puede ir cuatro veces mas rápido y lo hace olvidándose de la imagen excepto cuando no tiene nada que hacer. Para ver ésta manera de trabajar, pulse

FAST

Ahora, siempre que pulse una tecla, la pantalla parpadeará, ya que el computador detiene la representación de una imagen mientras ejecuta lo que le dice la tecla pulsada.

Escriba un programa, digamos

```
10 FOR N = 0 TO 255
20 PRINT CHR$ N;
30 NEXT N
```

Cuando lo ejecute, toda la pantalla será de un gris indeterminado hasta el fin del programa, cuando la salida de la sentencia **PRINT** apareca en la pantalla.

La imagen tambien está en pantalla durante una sentencia **INPUT,** mientras el computador está esperando a que Ud. introduzca datos. Pruebe éste programa:

```
10 INPUT A
20 PRINT A
30 GOTO 10
```

Para volver al funcionamiento normal (computador y pantalla) pulse

SLOW

Suele ser materia de gusto personal el que Ud. prefiera funcionamiento de computador y pantalla por claridad de presentación o funcionamiento rápido por su velocidad, pero en general Ud. utilizará el funcionamiento rápido cuando:

(i) su programa contenga un montón de cálculos numéricos, especialmente si no hay mucho que imprimir, sin embargo, el tiempo funcionando en computador y pantalla no parece ser tan largo ya que Ud. puede ir viendo la aparición de resultados en la pantalla, o

(ii) está escribiendo un programa largo. Ud. ya se habrá dado cuenta de come se rehace el listado, cada vez que introduce una nueva línea de programa, y esto puede llegar a ser aburrido.

En los programas puede utilizar las sentencias **SLOW** y **FAST** sin ningún problema.

Por ejemplo,

```
10 SLOW
20 FOR N = 1 TO 64
30 PRINT "A";
40 IF N = 32 THEN FAST
50 NEXT N
60 GOTO 10
```

Resumen

Sentencias: **FAST, SLOW**

CAPITULO 14

Subrutinas

Algunas veces partes diferentes de su programa tendrán que hacer tareas bastante similares, y se econtrará Ud. escribiendo las mismas líneas dos veces o mas; sin embargo, no es necesario. Ud. puede escribir las líneas una vez, en la forma conocida como una subrutina y así utilizarlas, o llamarlas, allí donde aparezcan en el programa sin tener que escribirlas de nuevo.

Para hacerlo, utilice las sentencias **GOSUB** (IR a SUBrutina) y **RETURN.**

GOSUB n

donde n es el número de línea de la primera de la subrutina, es exáctamente como **GOTO** n excepto que el computador recuerda el número de línea de la sentencia **GOSUB** de manera que puede regresar a ella despues de hacer la subrutina, Este recuerdo lo realiza poniendo el número de línea (la dirección de retorno) en lo alto de un montón de ellas (la pila de GOSUB).

RETURN

toma el número de línea que está en lo alto de la pila de **GOSUB,** y va a la línea siguiente.

He aquí un primer ejemplo,

```
  10 PRINT "ESTE ES EL PROGRAMA PRINCIPAL",
  20 GOSUB 1000
  30 PRINT "Y DE NUEVO";
  40 GOSUB 1000
  50 PRINT "Y ESO ES TODO."
  60 STOP
1000 REM LA SUBRUTINA EMPIEZA AQUI
1010 PRINT "ESTA ES LA SUBRUTINA,"
1020 RETURN
```

La sentencia **STOP** de la línea 60 es muy importante porque si no el programa continuaría con la subrutina y daría lugar al error 7 cuando llegase a la sentencia **RETURN.**

Para un ejemplo menos trivial, suponga que quiere escribir un programa para manejar libras, chelines y peniques. Aquellas personas con buena memoria recordarán que antes de 1971 una libra estaba dividida en veinte chelines, con lo que un chelín vale 5p (nuevos peniques), y un chelín estaba dividido en doce peniques antíguos; d era la abreviatura de un penique antíguo. Ud. tendrá tres variables L, S, y D (y puede que otras mas L1, S1, D1, etc) y una artimética sumamente fácil. En primer lugar Ud. debe manejar por separado las libras, los chelines y los peniques, por ejemplo, para sumar dos cantidades de dinero, sume los peniques, sume los chelines y sume las libras; para duplicar una cantidad de dinero, duplique los peniques, duplique los chelines y duplique las libras, etc. Cuando se ha hecho todo ésto, ajuste el resultado a su forma correcta de tal manera que los peniques estén

entre 0 y 11, y los chelines entre 0 y 19. Esta última etapa es común a todas las operaciones, por lo que podemos hecerla dentro de una subrutina.

Dejando a un lado de momento la noción de subrutinas, es mejor para Ud. que trate de escribir el programa. Dados los números arbitrarios L, S, y D ¿como los convertiría correctamente en libras, chelines y peniques? Parte del problema es que Ud. empezará a pensar en casos mas y mas raros.

Lo primero que nos acude a la mente, probablemente será algo así como £1 . . 25s . . 17d, que Ud. quiere convertir en £2 . . 6s . . 5d. No es difícil. Pero suponga que tiene números negativos. Una deuda de £1 . . 25s . . 17d, o £–1 . . –25s . . –17d, podría convertirse en £–3 . . 13s . . 7d, lo que es mas bien una manera antígua de expresarlo (como si la gente siempre prestaba a otros solamente libras enteras). ¿Y que ocurre con las fracciones? Si Ud. divide £1 . . 25s . . 17d entre dos, Ud. obtiene £.5 . . 12 . . 5s . . 8.5d y aunque tiene los peniques, 8.5, entre 0 y 11; los chelines, 12.5, entre 0 y 19, ciertamente no está tan bien como £1 . . 3s . . 2.5d. Plantee y calcule sus propias respuestas a todo esto, y utilícelas en un programa de computador, antes de que Ud. lea nada mas.

Aquí hay una solución.

```
1000 REM SUBRUTINA PARA CONVERTIR L, S, D EN LA FORMA
     HABITUAL DE LIBRAS, CHELINES Y PENIQUES
1010 LET D = 240*L + 12*S + D
1020 REM AHORA TODA ESTA EN PENIQUES
10300 LET E = SGN D
1040 LET D = ABS D
1050 REM TRABAJAMOS CON D POSITIVO; CONSERVANDO SU SIGNO
     EN E
1060 LET S = INT (D/12)
1070 LET D = (D–12*S)*E
1080 LET L = INT (S/20)*E
1090 LET S = S*E–20*L
1100 RETURN
```

Siguiendo con lo nuestro, todo esto no tiene mucha utilidad porque no hay un programa que nos de L, S, y D de antemano, ni que haga algo con ellos despues. Escriba éste programa principal y además otra subrutina para imprimir L, S y D.

```
10 INPUT L
20 INPUT S
30 INPUT D
40 GOSUB 2000
45 REM IMPRIME LOS VALORES
50 PRINT
60 PRINT "□□□ =";
70 GOSUB 1000
75 REM LA CONVERSION
80 GOSUB 2000
85 REM IMPRIME LOS VALORES
90 PRINT
100 GOTO 10

2000 REM SUBRUTINA PARA IMPRIMIR L, S Y D
2010 PRINT "£"; L; ". ."; S; "S . ."; D; "D";
2020 RETURN
```

(Recuerde del capítulo 9 que la sentencia vacía **PRINT** de la línea 50 dejará una línea en blanco).

Evidentemente hemos economizado en programa por el empleo de la subrutina de impresión de la línea 2000, y por eso un empleo muy corriente de las subrutinas es ese: para acortar programas. No obstante, la subrutina de conversión hace de hecho el programa mas largo, por un **GOSUB** y un **RETURN**; por lo que la longitud de programa no es la única razón. Utilizadas con habilidad las subrutinas pueden hacer los programas mas fáciles de entender a los que les interesan, los humanos.

El programa principal se simplifica mediante el empleo de sentencias mas eficaces: cada **GOSUB** representa complicar algo el BASIC, pero olvídelo, solo importa el resultado final y por ello es mucho mas fácil aprenderse bien la estructura principal del programa.

Por otro lado, las subrutinas se simplifican por una razón muy diferente, principalmente porque son mas cortas. Ellas todavía utilizan el viejo ajetreo de las sentencias **LET** y **PRINT,** pero como solo tienen que hacer una parte del trabajo total, son mas fáciles de escribir.

La habilidad consiste en elegir el nivel, o niveles, en que escribir las subrutinas. Tienen que ser lo suficientemente importantes como para tener un impacto apreciable en el programa principal, pero lo suficientemente pequeñas para ser notablemente mas fáciles de escribir que un programa completo sin subrutinas. Estos ejemplos (no recomendados) lo aclararán.

Primero,

```
  10 GOSUB 1000
  20 GOTO 10
1000 INPUT L
1010 INPUT S
1020 INPUT D
1030 PRINT " "; L; ". ."; S; "S . ."; D; "D"; TAB 8; "=";
1040 LET D = 240*L + 12*S + D
   :       :
   :       :
2000 RETURN
```

y segundo

```
  10 GOSUB 1010
  20 GOSUB 1020
  30 GOSUB 1030
  40 GOSUB 1040
  50 GOSUB 1050
   :       :
   :       :
  60 GOTO 10
1010 INPUT L
1015 RETURN
1020 INPUT S
1025 RETURN
1030 INPUT D
1035 RETURN
1040 PRINT " "; L; ". ."; S; "S . ."; D; "D"; TAB 8; "=";
1045 RETURN
1050 LET D = 240*L + 12*S + D
1055 RETURN
   :       :
   :       :
```

El primero, con su eficáz subrutina única, y la segunda, o con otras mas triviales, nos muestran extremos bastante opuestos, pero de igual inutilidad.

Afortunadamente, una subrutina puede llamar a otra o incluso a sí misma (una subrutina que se llama a sí misma es recursiva) por lo que no debemos asustarnos de tener varios niveles.

Resumen

Sentencias: **GOSUB, RETURN**

Ejercicios

1. El programa del ejemplo es virtualmente un calculator LSD universal. Como lo utilizaría para:
   (i) convertir libras y nuevos peniques en libras, chelines y peniques.
   (ii) convertir guineas en libras y chelines. (1 guinea = £1 = £1 . . 1s)
   (iii) calcular fracciones de libra. (Por ejemplo, el tercio de una libra, o un marco es 6s . . 8d).

2. Añada dos sentencias al programa:

```
4 LET CONVERSION = 1000
7 LET IMPRIME LSD = 2000
```

y cambie

GOSUB 1000 por **GOSUB** CONVERSION
GOSUB 2000 por **GOSUB** IMPRIME LSD

Esto funciona exactamente como Ud. esperaba; en efecto, el número de línea en una sentencia **GOSUB** (o **GOTO** o **RUN**) puede ser cualquier expresión numérica. No espere que esto sirva para otros computadores que no sean el ZX81, porque no es normal en BASIC.

Esta especie de fruslería puede hacer milagros en la claridad de sus programas.

3. Vuelva a escribir el programa principal del ejemplo para hacer algo completamente diferente, pero utilizando todavía las mismas subrutinas.

4.
```
. . . GOSUB n
. . . RETURN
```

en líneas consecutivas se pueden sustituir por

```
. . . GOTO n
```

¿Por qué?

5. Una subrutina puede tener varios puntos de entrada. Por ejemplo, por la manera en que nuestro programa principal las utiliza, con **GOSUB** 1000 seguida inmediatamente de **GOSUB** 2000, podemos reemplazar nuestras dos subrutinas por una grande que calcule L, S y D y los imprima. Esta tiene dos puntos de entrada: uno al pricipio de subrutina única y otro mas adelante, solo para la parte de impresión.

Haga los reajustes necesarios.

6. Ejecute éste programa:

```
10 GOSUB 20
20 GOSUB 10
```

Las direcciones de retorno son empujadas a manadas sobre el compartimento de **GOSUB,** pero nunca se vuelvan a sacar y casualmente no hay sitio para ninguna mas en el computador. El programa se detiene con error 4 (ver apéndice B).

Puede que tenga dificultades para desenredarlas sin perderlas todas, pero esto le servirá:

(i) Borre las dos sentencias **GOSUB**
(ii) Introduzca dos nuevas líneas

```
11 RETURN
21 RETURN
```

(iii) Pulse

```
RETURN
```

Las direcciones de retorno son desenredadas hasta conseguir error 7.
(iv) Cambie su programa para que no le vuelva a ocurrir lo mismo.

¿Como funciona esto?

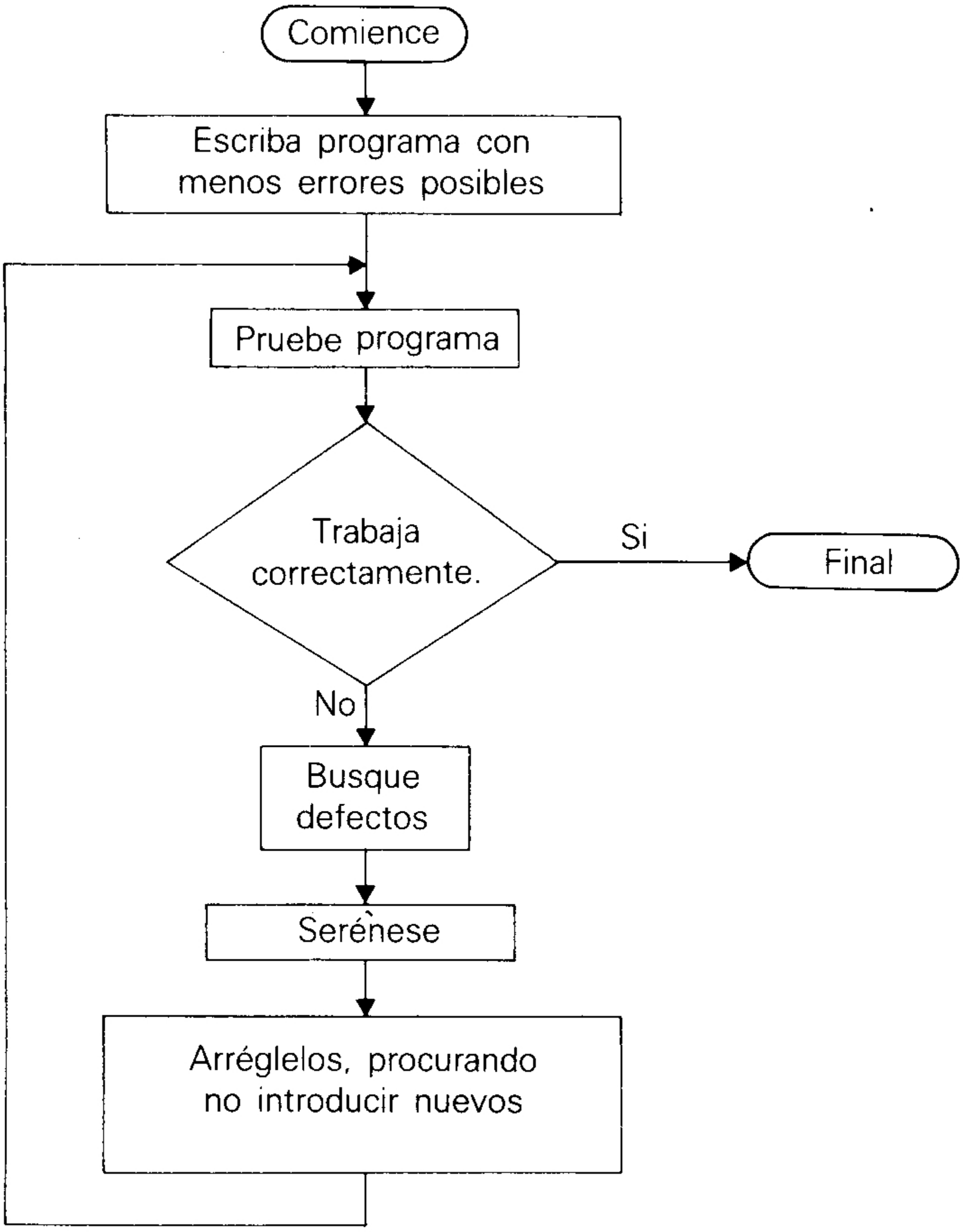
CAPITULO 15

Ejecutando sus programas

En el arte de programar computadores hay algo mas que conocer que hacen las sentencias. Probablemente Ud. ya habrá encontrado que la mayoría de sus programas tienen, lo que familiarmente se conocen como gazapos, cuando se programa por primera vez: es posible que solo se trate de errores de escritura, o puede haber errores en sus ideas a cerca de lo que el programa debería hacer. Puede achacarlos a su inexperiencia, pero se engañaría Ud. mismo.

TODO PROGRAMA ARRANCA CON IMPERFECCIONES.

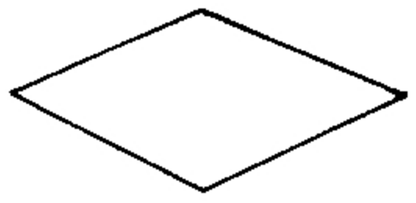
Y muchos programas tambien acaban con defectos. Hay dos corolarios a esto: Primero, Ud. debe probar todos sus programas enseguida; y segundo, no pierda su humor cada vez que no le funcionen. El plan general a seguir se puede ilustrar con un organigrama o diagrama:



La idea es que Ud. siga las flechas de recuadro en recuadro, haciendo lo que dice cada uno. Hemos utilizado diferentes clases de recuadros para diferentes clases de instrucciones:

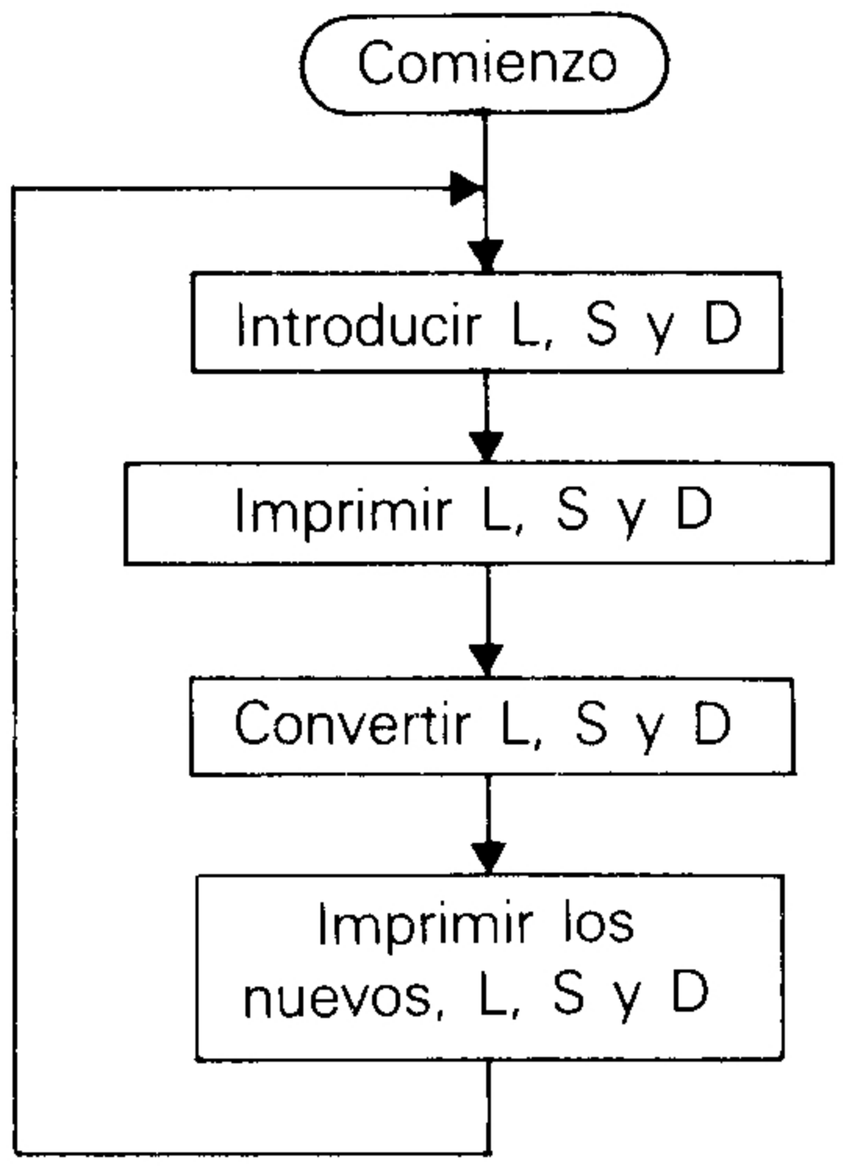
Un recuadro redondeado  es empezar o acabar.

Un recuadro rectangular  es una instrucción sencilla.

Un rombo  le pide a Ud. que tome una decisión antes de continuar.

(Estas formas se utilizan bastante ampliamente, pero de ningún modo el fin del mundo depende de ellas).

Por supuesto, estos diagramas están mal adaptados para describir actividades humanas; pensar a lo largo de líneas rectas fijas como estas, no es bueno para la creatividad o flexibilidad. Sin embargo, para los computadores esto constituye precisamente su trabajo. Son los mejores para describir la estructura a grandes rasgos de los programas, con una subrutina en casi todos los recuadros y así el diagrama para nuestro ejemplo de las libras del último capítulo podría ser:



Conceptualmente, los recuadros rectangulares corresponden a **GOSUBs**; no obstante, en la práctica algunos recuadros, como el anterior al que introduce L, S y D, son trasladados directamente a sentencias BASIC, sin subrutinas.

Cualquier cosa, como diagrams, subrutinas y tambien sentencias **REM,** que haga el

programa mas claro le proporciona a Ud. un mejor entendimiento del mismo; y así está Ud. obligado a escribir con menos defectos. Pero las subrutinas tambien le ayudan a localizar los defectos que de todos modos introducirá, haciendo el programa mas fácil de probar. Encontrará que es mucho mas fácil probar las subrutinas individualmente y asegurarse de que se acoplan correctamente, que probar un programa entero no estructurado.

Asi pués, las subrutinas le ayudan en el recuadro "busque defectos" y éste es el recuadro donde Ud. necesita todo la ayuda que puede conseguir, por cuanto suele ser el más exasperante. Otras sugerencias para descubrir defectos son:

(i) compruebe que no hay errores de escritura. Hagalo siempre.

(ii) Trate de determinar todas las variables que debería haber en cada etapa y, si es posible, expliquelo en sentencias **REM.** Puede comprobar las variables en un punto dado del programa insertando allí una sentencia **PRINT.**

(iii) Si el efecto del programa es hacer que se detenga con un informe de error, utilice esta información tan ampliamente como pueda. Busque el código del informe y determine porque se detiene en la línea en que lo hace. Si es necesario, calcule los valores de las variables.

(iv) Ud. debería poder andar a través de un programa, línea a línea mediante la escritura de sus líneas como órdenes.

(v) Intente ser el computador: ejecute el programa por si mismo empleando lápiz y papel para registrar los valores de las variables.

Una vez que haya localizado las imperfecciones, corregirlas es como escribir el programa original, pero debe probar nuevamente el programa. Es sorprendentemente fácil corregir un defecto e introducir otro.

Ejercicios

1. El diagrama para calcular LSD no tiene recuadro de "Fin". ¿Importa esto? ¿Donde lo pondría Ud. si quisiera hacerlo?

2. Escriba diagramas para los programs de bucles del capítulo 12.

CAPITULO 16

Almacenamiento en cinta magnetica

Como se dijo en el capítulo 1, y Ud. no habrá dudado en experimentar, cuando desconecta el ZX81 pierde todo el programa y las variables que hubiera introducido. La única manera de salvarlo es registrar en una casete todo el contenido de la memoría del computador; y mas tarde podrá volver a introducirlo con lo que el computador quedará prácticamente en la misma situación que tenía cuando se hizo la grabación.

Con el ZX81, Ud. habrá encontrado un par de cables (artículo 5 del capítulo 1) que conectan el ZX81 al magnetofón a casetes. Ud. se debe agenciar su propio magnetofón, y algunos funcionan mejor que otros.

En primer lugar, por lo que al ZX81 concierne, los magnetofones baratos, portátiles, monocanal son al menos tan buenos como los caros estereofónicos, y tambien dan menos problemas. Encontrará muy útil que tenga contador de cinta.

En segundo lugar, el magnetofón debe tener un enchufe de entrada para micrófono y un enchufe de salida para auricular (si no lo hubiera, pruebe con el enchufe para altavoz externo). Deberan ser, preferentemente, enchufes para jacks de 3,5mm (para admitir las clavijas con que están equipados los cables), ya que otros tipos no suelen proporcionar una señal de suficiente potencia para el ZX81.

Ahora, habiendo conseguido un magnetofón adecuado, conéctelo al computador: un cable deberá unir la entrada de micrófono del magnetofón al enchufe marcado "MIC" en el costado del ZX81, y el otro conecta la salida de auricular del magnetofón al enchufe "EAR" en el ZX81. Asegúrese de que no ha cruzado los cables (aunque el ZX81 no sufrirá daños si así ocurre).

Introduzca algún programa en el computador, digamos el programa del juego de caracteres del capítulo 11. Para conservarlo, va a tener que ponerle un nombre al programa, y es una buena idea introducir este nombre en el programa y así aparecerá en los listados, siendo el camino mas fácil con una sentencia REM. Escriba:

```
5 REM "CARACTERES"
```

Ahora, y ésta es justamente una operación estéril, con el fin de que Ud. pueda ver lo que ocurre, escriba

```
SAVE "CARACTERES"
```

y mire el televisor. Durante cinco segundos será de un color grisaceo y a continuación, durante seis segundos aproximadamente, habrá una imagen característica cambiante de barras finas blancas y negras y, por último, la pantalla se quedará en blanco con el informe 0/0. El computador esturo enviando una señal al enchufe "MIC"; pero tambien estuvo enviando la misma señal al televisor, dando lugar a la imagen que vió. La parte gris era un intervalo silencioso, y la parte en blanco y negro era el programa.

Desde luego, lo que Ud. quiere hacer es recoger esa señal en la cinta, asi que esta vez permítanos hacerlo adecuadamente.

Grabando un programa

1. Posicione la cinta en una parte que esté en blanco, o que Ud. haya borrado para volver a grabar.
2. Utilizando un micrófono, grábese Ud. mismo diciendo "caracteres". No es imprescindible, pero le hará mucho mas fácil la posterior búsqueda del programa. Vuelva a conectar el computador al magnetofón.
3. Escriba

SAVE "CARACTERES" (sin **NEWLINE**)

4. Ponga en marcha el magnetofón en grabación.
5. Pulse **NEWLINE.**
6. Mire al televisor como antes. Cuando haya terminado (con el informe Ø/Ø), pare el magentofón.

Para estar seguros de que ha funcionado, debería escuchar la cinta a través del altavoz del magnetofón. (Probablemente tendrá que desenchufar el cable del enchufe del auricular del magnetofón). Rebobine la cinta hasta donde Ud. empezó a grabar y reprodúzcala.

En primer lugar oirá su propia voz diciendo "caracteres".

Despues viene un suave, susurrante zumbido. Esto no es realmente parte de la grabación, sino la parte final de la señal para el televisor (antes de que pulsara **NEWLINE**), que tambien se tuvo que enviar al magnetofón.

Despues vienen cinco segundos de silencio, el principio de la propria señal de grabación. Corresponden al periodo en que la pantalla del televisor estuvo gris.

Despues vienen aproximadamente seis segundos de un zumbido muy chillón de tono alto, que a todo volumen constituye un ruido desagradable. Es la grabación del programa y corresponde a la imagen en blanco y negro de la pantalla del televisor.

Finalmente vuelve el zumbido suave y susurrante.

Si Ud. no oyó los distintos zumbidos y susurros, compruebe si tenía convenientemente conectados el computador y el magnetofón. Con algunos magnetofones ocurre que si la clavija de jack no está bien introducida a fondo, no hace contacto. Trate de introducirla aproximadamente un par de milímetros, algunas veces puede apreciar que queda en una posición mas natural.

Ahora supongamos que el sonido de la grabación es correcto según el oido humano y que Ud. quiere tratar de volver el programa al computador (cargarlo en el computador).

Cargando un programa con nombre

1. Rebobine la cinta hasta el lugar en que Ud. empezó.
2. Asegúrese de que el enchufe "EAR" del computador está conectado al enchufe del auricular del magnetofón.
3. Gire el control de volumen del magnetofón hasta aproximadamente tres cuartos del volumen máximo. Si tiene controles de tono ajústelos de manera que los agudos sean fuertes y los bajos suaves (así sonará chirriante).

4. Escriba
LOAD "CARACTERES" (nuevamente, sin **NEWLINE**)

5. Ponga en marcha el magnetofón en reproducción.
6. Pulse **NEWLINE**.

De nuevo, podrá ver imágenes de la grabación en el televisor, pero ésta vez un poco distintas, aunque siempre con una imagen en blanco y negro. Las dos partes, la silenciosa y el programa, estarán menos diferenciadas, pero podrá apreciar que la parte del programa tiene unas líneas mucho mas anchas, mas definidas. (Haga el ejercicio 1 varias veces).

Transcurridos quince segundos, debería estar cargado y detenerse con informe Ø/Ø. Si no es así, pulse la tecla **BREAK** (espacio), que le permitirá salir de su desgracia.

Lo que tiene mas probabilidades de haber fallado es el nivel de volumen. Este debería ser:

(i) lo suficientemente alto para que la parte de programa sea captada por el computador,
(ii) no tan alto como para que la parte de programa se distorsione (realmente es un caso bastante raro),
y (iii) lo suficientemente bajo como para que la parte silenciosa sea reconocida como un silencio por el computador.

El mejor ajuste consiste en subir el volumen tanto como se pueda sin que la parte silenciosa llegue a ser ruidosa; puede hacer esto mientras escucha la grabación por el altavoz. Si la parte silenciosa es incorregiblemente ruidosa, U. dpuede tener otros problemas:

Algunos magnetofones forman un bucle de realimentación con el ZX81. Esto solo puede ocurrir cuando los dos cables EAR y MIC están conectados a la vez, por lo que la solución es hacer la grabación (**SAVE**) con el cable EAR desconectado.

Algunos magnetofones pueden registrar un zumbido de alimentación. Se soluciona haciéndolos funcionar con pilas.

Algunos magnetofones, especialmente los viejos y antiguos, son ruidosos de por sí. Esto se podría corregir utilizando cinta de major calidad, aunque no debería ser necesario.

Si está sucia, limpie la cabeza de grabación, en el magnetofón a casetes.

Por último, puede tratarse del mismo problema de un perfecto enchufe de las clavijas como vimos anteriormente.

Si Ud. tiene un programa en cinta y no puede recordar su nombre, aún puede cargarlo. (Pruebe a hacerlo con el programa "CARACTERES" que usó antes).

Cargando un programa sin nombre

1. Posicione la cinta al principio de la parte silenciosa.
2. Compruebe todo y ajuste los controles como antes. Puede muy bien encontrarse con que tiene que tener mucho mas cuidado con el nivel de volumen que cuando teníamos un nombre.

3. Escriba

LOAD "" (sin **NEWLINE**)

4. Ponga en marcha el magnetofón en reproducción.
5. Pulse **NEWLINE.**
6. El resto, como antes.

La idea es que si el nombre del programa que Ud. pide que se cargue es la cadena vacía, el computador carga el primer programa que le llegue. Observe que cuando conserva un programa, no puede darle como nombre el de cadena vacía, porque si lo intenta obtendrá un error F.

LOAD y **SAVE** tambien se pueden utilizar en programas. Con **SAVE,** el programa se conserva por sí mismo de tal manera que cuando sea cargado se pondrá inmediatamente en ejecución a partir de la línea siguiente de la sentencia **SAVE.**

Por ejemplo, escriba

```
  5 REM "INUTIL"
 10 PRINT "ESTO ES TODO LO QUE HACE"
 20 STOP
100 SAVE "INUTIL"
110 GOTO 10
```

Conecte el magnetofón, escriba

RUN 100 (sin **NEWLINE**),

ponga enmarcha el magentofón en grabación, y pulse **NEWLINE.** Cuando el programa se haya conservado a sí mismo, continuará ejecutándose. Mas tarde descubrirá que la L de INUTIL en la línea 100 ha cambiado a imagen en negativo, pero eso es algo que no debe preocuparle.

Para cargarlo, rebobine la cinta hasta un poco antes de empezar el programa, escriba

LOAD "INUTIL" (sin **NEWLINE**),

ponga en marcha el magnetofón para repetir la cinta dentro del computador, y pulse **NEWLINE.** Cuando se haya cargado, continuará con la línea 110 y se ejecutará por sí solo sin que Ud. tenga que hacer nada.

Observe como poniendo la sentencia **SAVE** al final del programa, si Ud. quiere ejecutarlo sin que se efectúe la sentencia **SAVE**, solamente tiene que escribir **RUN**, y así no tiene que rodear la sentencia **SAVE**.

No haga una grabación (**SAVE**) desde dentro de una rutina **GOSUB,** pues no funcionará correctamente.

No ponga caracteres en negativo en un nombre de programa. Todo lo que vaya, en el nombre, detrás del caracter en negativo se perderá.

El nombre no deberá tener mas de 127 caracteres.

El nombre en una **LOAD** o **SAVE** no tiene por qué ser una constante de cadena, puede ser cualquier expresion de cadena con valor, como A$ de CHR$ 100.

Resumen

Grabando un programa en cinta
Cargando un programa con nombre desde una cinta
Cargando el primer programa disponible de una cinta
Grabando un programa de tal manera que se cargará y ejecutará por sí mismo
Sentencias: **SAVE, LOAD**

Ejercicios

1. Prepare una cinta con muchos programas cortos, introdúzcala en el computador y escriba

```
LOAD "NO EL NOMBRE DE UN PROGRAMA"
```

En el televisor, podrá ver facilmente la diferencia entre los trechos vacios de la cinta (con una imagen en blanco y negro totalmente desordenada) y los programas (con líneas mas definidas). Ambas imágenes son distintas de las que Ud. veía cuando hacía la grabación. Si Ud. baja el volumen mientras está pasando un programa, podrá ver que la imagen cambia a la forma de espacio vacio ya que la señal es demasiado baja para verse como programa.

2. Confeccione una cinta en la que el primer programa, cuando sea cargado, imprima una minuta (una lista de los otros programas de la cinta), le pida a Ud. que seleccione un programa y lo cargue.

3. Escriba nuevamente el programa "CARACTERES", y escriba

```
LET X = 7
```

y así, aunque no aparece en el programa, el computador contiene ahora una variable X con valor 7. Ahora grabe el programa, desenchufe y vuelva a enchufar el computador (para estar seguros de que no hay trampa), y cargue nuevamente el programa. Escriba

```
PRINT X
```

y obtendrá la respuesta 7. La sentencia **SAVE** grabó no solo el programa, sino tambien todas las variables, incluyendo X.

Si Ud. quiere mantener éstas variables cuando ejecute el programa, debe acordarse de utilizar **GOTO** y no **RUN** (como se dijo en el capítulo 9). Puede evitar tener que acordarse de esto haciendo que el programa se ejecute por sí solo (utilizando **SAVE** como una línea del programa).

4. Escriba un programa muy largo y desconecte momentáneamente la fuente de alimentación. Estas cosas ocurren a veces espontáneamente; no es un defecto sino un apagón. No hay nada que pueda Ud. hacer excepto lamentarse. Si ocurre con mas frecuencia de lo que Ud. pueda soportar, entoces es que probablemente hay algo defectuoso, pero mercería la pena hacer grabaciones parciales en cinta para no perder la totalidad de lo escrito y poder recuperarlo si surge el problema.

CAPITULO 17

Imprimiendo ordenadamente

Recordará que una setencia **PRINT** tiene una lista de partidas, siendo cada una, una expresión (o posiblemente nada en absoluto), que están separadas por comas o puntos y comas. Existen dos clases mas de accesorios **PRINT** que se usan para decir al computador no lo que, sino donde imprimir. Por ejemplo, **PRINT AT** 11, 16; "*" imprime un asterisco en el centro de la pantalla.

AT línea, columna

mueve la posición de **PRINT** (el sitio donde se va a imprimir la siguiente cosa) a la línea y columna especificada. Las líneas están numeradas de 0 (la superior) a 21, y las columnas de 0 (la de la izquierda) a 31.

TAB columna

mueve la posición de **PRINT** a la columna especificada. Permanece en la misma línea, o, si tiene que retroceder, se mueve a la siguiente. Observe que el computador reduce el número de columna al módulo 32 (lo divide por 32 y toma el resto); así **TAB** 33 significa lo mismo que **TAB** 1.

Por ejemplo

PRINT TAB 30; 1; **TAB** 12; "CONTENIDO"; **AT** 0,3; "CAPITULO"; **TAB** 24; "PAGINA"

(Esto es como podría imprimir el encabezamiento de una pagina de Contenido, con el 1 como número de página)

Algunos pequeños detalles:

(i) Estas nuevas partidas es lo mejor terminarlas con punto y coma, como hemos hecho antes. Puede usar comas (o nada, al final de la sentencia), pero esto signìfica que despues de haber establecido cuidadosamente la posición de PRINT, Ud. la mueve de nuevo inmediatamente, lo que no suele ser muy útil.

(ii) Aunque **AT** y **TAB** no son funciones, tiene que pulsar la tecla de funciones (newline cambiada) para conseguirlas.

(iii) Ud. no puede imprimir en las dos líneas mas inferiores (22 y 23) de la pantalla porque están reservadas para órdenes, datos **INPUT,** informes, etc. Cuando nos referimos a "la línea inferior" normalmente queremos decir línea 21.

(iv) Puede utilizar **AT** para colocar la posición **PRINT** allí donde incluso ya hay algo escrito; lo anterior quedará sobreimpreso.

Hay dos sentencias mas, relacionadas con **PRINT,** a saber **CLS** y **SCROLL.**

CLS Borra la pantalla (pero nada mas).

SCROLL nueve toda la imagen una línea hacia arriba (desapareciendo la línea de arriba) y mueve la posición de **PRINT** al principlio de la línea inferior.

Para ver como trabaja esto, ejecute el programa:

```
10 SCROLL
20 INPUT A$
30 PRINT A$
40 GOTO 10
```

Resumen

Partidas **PRINT: AT, TAB**
Sentencias: **CLS, SCROLL**

Ejercicios

1. Pruebe a ejecutar esto:

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

Esto muestra lo que se entiende por reducción de números de **TAB** al módulo 32. Para un ejemplo mas elegante, cambie el 8 de la línea 20 por un 6.

CAPITULO 18

Gráficos

He aquí algunas de las características mas attractivas del ZX81; utiliza lo que se conocen como pixels ("picture elements" – elementos de imagen). La pantalla sobre la que Ud. puede dibujar tiene 22 líneas y 32 columnas, lo que hace 22*32 = 704 posiciones de caracteres, y cada una de ellas contiene 4 pixels, dividida como una rebanada de pastel de Battenburg.

Un pixel se define por dos números, sus coordenadas. El primero, su coordenada x, nos dice a que distancia está transversalmente desde la columna del extremo izquierdo, (X es UNA FILA ¡ Recuérdelo!), y el segundo, su coordenada y, nos dice a que distancia está del pié de la pantalla. Normalmente, éstas coordenadas se escriben como un par de números entre paréntesis, así (0, 0), (63,0), (0,43) y (63,43) son las esquinas: inferior izquierda, inferior derecha, superior izquierda y superior derecha.

La sentencia

PLOT coordenada x, coordenada y

ennegrece el pixel con éstas coordenadas, mientras que la sentencia

UNPLOT coordenada x, coordenada y

lo pone en blanco.

Plantee éste programa moteado:

```
10 PLOT INT (RND * 64), INT (RND * 44)
20 INPUT A$
30 GOTO 10
```

Esto traza un punto al azar cada vez que Ud. pulse newline.

He aquí un programa bastante mas útil. Traza un gráfico de la función **SIN** (una curva sinusoidal) para valores entre 0 y 2 π.

```
10 FOR N = 0 TO 63
20 PLOT N, 22 + 20 * SIN (N/32 * PI)
30 NEXT N
```

Y el siguiente traza un gráfico de **SQR** (parte de una parábola) entre 0 y 4:

```
10 FOR N = 0 TO 63
20 PLOT N, 20 * SQR (N/16)
30 NEXT N
```

Observe que las coordenadas de un pixel son bastante distintas de las de línea y columna en una partida **AT**. Al final del presente capítulo hay un diagrama que le puede resultar útil para trabajar con coordenadas de pixels y números de línea y de columna.

Ejercicios

1. Hay tres diferencias entre los números de una partida **AT** y las coordenadas de un pixel; ¿cuales son?

Suponga que una posición **PRINT** corresponde a **AT** L, C (línea, columna). Demuestre que los cuatro pixels correspondientes a esa posición tienen como coordenadas de las x 2*C o 2*C + 1 y como coordenadas de las y 2* (21–L) o 2* (21–L) + 1. (Mire el diagrama).

2. Haga una rebenada de queso modificando el programa moteado de manera que primero llene la pantalla de negro (un cuadrado negro es un espacio de vídeo invertido), y después trace puntos al azar. Si Ud. tiene solamente 1K de memoria, es decir, la máquina standard sin memoria adicional, se encontrará que se ha quedado sin existencias de espacio, por lo que deberá fijar el programa para que utilice solo una parte de la pantalla.

3. Modifique el programa del gráfico **SIN** para que antes de trazar la curva imprima una linea horizontal a base de guiones "–" como eje de las x, y una línea vertical de "/" como eje de las y.

4. Escriba programas para trazar gráficas de mas funciones, por ejemplo **COS, EXP, LN, ATN, INT,** etc. Para cada una tendrá estudiar:

(i) sobre que intervalo va a tomar Ud. las funciones (correspondiente al intervalo de Ø a 2π en la curva **SIN**).

(ii) en que lugar de la pantalla se coloca el eje de las x (equivaliendo a 22 en la línea 2Ø en el programa de la curva **SIN**).

(iii) la escala del eje de las y en el gráfico (correspondiendo 2Ø en la línea 2Ø en el programa para la curva **SIN**)

Encontrará que **COS** es la mas fácil, exactamente como **SIN.**

5. Ejecute lo siguiente:

```
1Ø PLOT 21, 21
2Ø PRINT "COMILLAS GRUESAS"
3Ø PLOT 54, 21
```

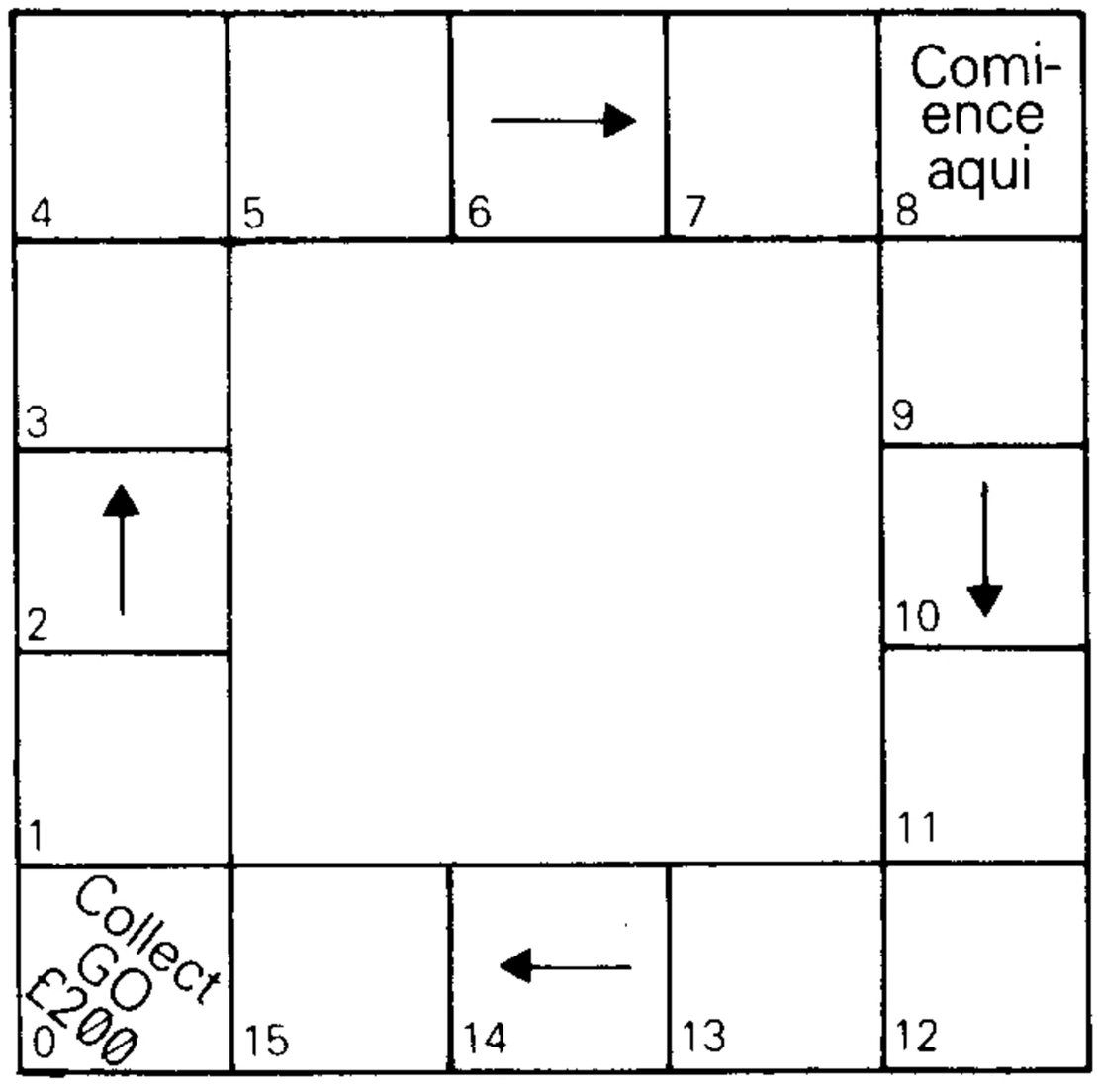
PLOT se mueve sobre la posición **PRINT.** (**UNPLOT** también lo hace).

6. Esta subrutina traza una línea (bastante) recta entre el pixel (A, B) y el pixel (C, D). Empléélo como parte de algún programa principal que le facilite los valores de A, B, C y D.

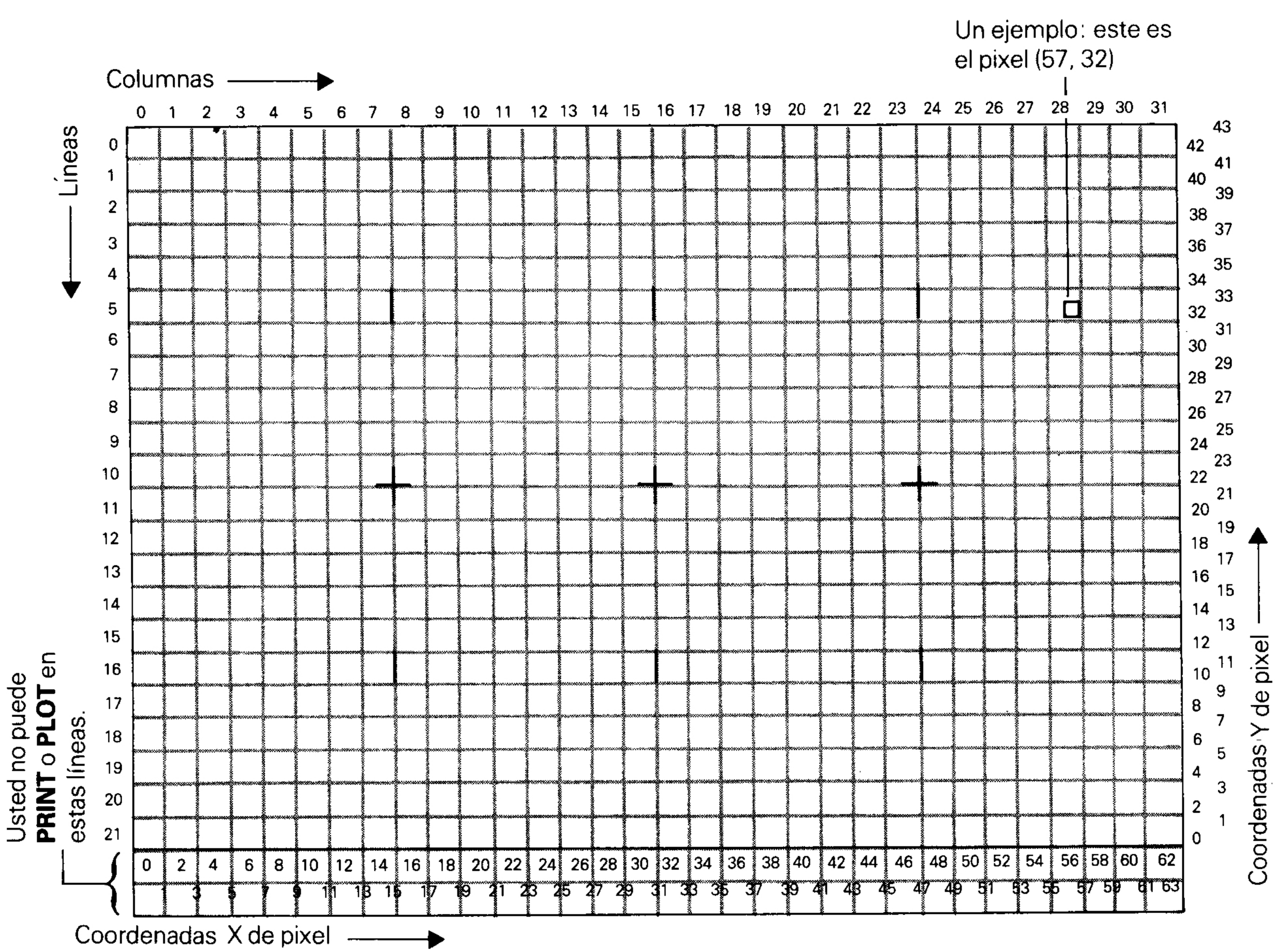
(Si no tiene una placa para ampliación de memoria, probablemente necesitará omitir las sentencias **REM**).

```
1000 LET U = C - A
1005 REM INDICA CUANTOS PASOS EN HORIZONTAL DEBEMOS
AVANZAR
1010 LET V = D - B
1015 REM V INDICA CUANTOS PASOS HACIA ARRIBA
1020 LET D1X = SGN U
1030 LET D1Y = SGN V
1035 REM (D1X, D1Y) ES UN SOLO PASO EN DIAGONAL
1040 LET D2X = SGN U
1050 LET D2Y = 0
1055 REM (D2X, D2Y) ES UN SOLO PASO A LA DERECHA O A LA
IZQUIERDA
1060 LET M = ABS U
1070 LET N = ABS V
1080 IF M>N THEN GOTO 1130
1090 LET D2X = 0
1100 LET D2Y = SGN V
1105 REM AHORA (D2X, D2Y) ES UN SOLO PASO HACIA ARRIBA O
ABAJO
1110 LET M = ABS V
1120 LET N = ABS U
1130 REM M ES EL MAYOR DE ABS U Y ABS V, N ES EL MENOR
1140 LET S = INT (M/2)
1145 REM QUEREMOS IR DE (A, B) A (C, D) EN M PASOS USANDO N
PASOS D2 DE DERECHA A IZQUIERDA O DE ARRIBA A ABAJO, Y M - N
PASOS D2 DE DERECHA A IZQUIERDA O DE ARRIBA A ABAJO, Y M - N
PASOS D1 EN DIAGONAL; DISTRIBUIDOS LO MAS UNIFORMEMENTE
POSIBLE
1150 FOR I = 0 TO M
1160 PLOT A, B
1170 LET S = S + N
1180 IF S<M THEN GOTO 1230
1190 LET S = S - M
1200 LET A = A + D1X
1210 LET B = B + D1Y
1215 REM UN PASO EN DIAGONAL
1220 GOTO 1250
1230 LET A = A + D2X
1240 LET B = B + D2Y
1245 REM UN PASO ARRIBA - ABAJO O DERECHA - IZQUIERDA
1250 NEXT I
1260 RETURN
```

La última parte (líneas 1150 en adelante) mezcla uniformemente los M – N pasos D1 con los N pasos D2. Imagine un tablero de Monopoly con M cuadrados alrededor del borde, numerados del 0 a M –1. El cuadrado en que Ud. está en cualquier momento es el número S, empezando en la esquina opuesta a GO. Cada jugada le nueve a Ud. N cuadrados alrededor del tablero, y sobre la línea recta de la pantalla Ud. da un paso de izquierda – derecha/arriba – abajo (si pasa GO en el tablero), o un paso en diagonal si no lo pasa. Puesto que su recorrido total en el tablero es M * N pasos, o dar la vuelta alrededor N veces, Ud. pasa la casilla GO N veces y uniformemente espaciados en sus M pasos hay N pasos izquierda – derecha/arriba – abajo.



Ajuste el programa de manera que si otro parámetro, E, es 1 la línea se trace en negro (como en el ejemplo), y si es 0 se trace en blanco (empleando **UNPLOT**). Así pues, Ud. puede borrar una línea que acabe de dibujar mediante su inversión.


Un ejemplo: este es el pixel (57, 32)
Columnas
Líneas
Usted no puede **PRINT** o **PLOT** en estas líneas.
Coordenadas X de pixel
Coordenadas Y de pixel

CAPITULO 19

Tiempo y movimiento

Bastante frecuentemente Ud. querrá hacer que el programa pare un tiempo determinado y para ello encontrará útil (especialmente en funcionamiento rápido) la sentencia **PAUSE:**

PAUSE n

detiene el cómputo y mantiene la imagen durante n cuadros de televisión (a 50 cuadros por segundo, o 60 en America). n puede ser hasta 32767, lo que nos dá algo menos de 11 minutos; si n es cualquier número mayor, su significado es "**PAUSE** para siempre".

Una pausa siempre puede interrumpirse pulsando una tecla (observe que un espacio, o £, tambien la interrumpen). Tiene que pulsar la tecla después de que haya empezado la pausa.

Al final de la pausa, la pantalla parpadeará.

Si se utiliza una sentencia **PAUSE** en una programa que se ejecutará en funcionamiento rápido, ésta sentencia deberá ir seguida por **POKE** 16437, 255. Aparentemente, **PAUSE** ejecutará su trabajo sin necesidad de esto, pero lo mas probable es que su programa quede destruido.

Este programa controla el segundero (justamente como si tuviera un punto en el extremo) de un reloj.

```
 5 REM PRIMERO DIBUJAMOS LA ESFERA DEL RELOJ
10 FOR N = 1 to 12
20 PRINT AT 10 - 10* COS (N/6*PI), 10 + 10* SIN (N/6*PI);N
30 NEXT N
35 REM AHORA PONEMOS EN MARCHA EL RELOJ
40 FOR T = 0 TO 10000
45 REM T ES EL TIEMPO EN SEGUNDOS
50 LET A = T/30*PI
60 LET SX = 21 + 18*SIN A
70 LET SY = 22 + 18*COS A
200 PLOT SX, SY
300 PAUSE 42
310 POKE 16437, 255
320 UNPLOT SX, SY
400 NEXT T
```

(Olvídese de las sentencias **REM** a menos que tenga una placa de extensión de memora).

A éste reloj se le acabará la cuerda después de 2¾ horas, debido a la línea 40, pero puede hacer fácilmente que funcione por mas tiempo. Observe como ese tiempo está controlado por la línea 300. Ud. podría haber esperado **PAUSE** 50 para que hiciera tictac una vez por segundo, pero el cálculo también necesita un poco de tiempo y hay que dárselo. Esto se hace mejor por tanteo, comparando el reloj del computador con uno real, y

ajustando la línea 300 hasta que esté en hora. (No podriá conseguir una gran precisión; una variación de un cuadro por segundo es el 2% o media hora al día).

La función **INKEY$** (que no tiene argumento) lee el teclado. Si Ud. está pulsando exactamente una tecla (o la tecla "shift" y otra) el resultado es el caracter que esa tecla proporciona en funcionamiento **L** ; de lo contrario el resultado es la cadena vacía. Los caracteres de control no tienen su efecto acostumbrado, sino que dan resultados como **CHR$** 118 por "newline", ellos se imprimen como "?".

Pruebe el siguiente programa, que trabaja como una máquina de escribir.

```
10 IF INKEY$ <> "" THEN GOTO 10
20 IF INKEY$ = "" THEN GOTO 20
30 PRINT INKEY$;
40 GOTO 10
```

Aquí la línea 10 espera a que Ud, levante su dedo del teclado y la línea 20 a que Ud. pulse una nueva tecla.

Recuerde que a diferencia de **INPUT, INKEY$** no le espera a Ud. Así Ud. no tiene que pulsar "newline" pero por otro lado si no escribe nada en absoluto entonces ha perdido su oportunidad.

Ejercicios

1. ¿Que pasa si omite la línea 10 en el programa de la máquina de escribir?
2. ¿Por qué no puede escribir un espacio o £ en el programa de la máquina de escribir?

He aquí un programa modificado que le proporciona a Ud. un espacio si pulsa el cursor derecho (8 cambiada).

```
10 IF INKEY$ <> "" THEN GOTO 10
20 IF INKEY$ = "" THEN GOTO 20
30 LET A$ = INKEY$
40 IF A$ = CHR$ 115 THEN GOTO 110
90 PRINT A$;
100 GOTO 10
110 PRINT " ";
120 GOTO 10
```

Observe como interpretamos **INKEY$** dentro de A$ en la línea 30. Sería posible olvidarse de esto y sustituir A$ por **INKEY$** en las líneas 40 y 90, pero siempre existiría alguna ocasión en que **INKEY$** cambiaría entre líneas.

Añada algo mas al programa de manera que si Ud. pulsa **NEWLINE** (**CHR$ 118**) obtenga una nueva línea.

3. Otra manera de utilizar **INKEY$** es en combinación con **PAUSE,** como en ésta variante del programa de la máquina de escribir.

```
10 PAUSE 40000
20 POKE 16437, 255
30 PRINT INKEY$;
40 GOTO 10
```

Para hacer éste trabajo, ¿por qué es esencial que una pausa no sea interrumpida si cuando va a iniciarse Ud. ya ha pulsado una tecla?.

Este método tiene la desventaja de que la pantalla parpadea pero en trabajo rápido es la única manera de conseguirlo. Ejecute el programa en funcionamiento rápido, y observe como el computador aprovecha la oportunidad de una pausa para dar la imagen al televisor.

4. El siguiente le volverá a Ud. loco. El computador le dá en pantalla un número, que Ud. (o una víctima inocente) tiene que volver a escribir. Para empezar tiene Ud. un segundo para hacerlo, pero si se equivoca para el siguiente número tendrá mas tiempo, mientras que si lo hace bien tendrá menos para el siguiente. La idea es conseguir ir tan rápido como sea posible, y entonces pulsar Q para ver su puntuación, cuanto mas alta, mejor.

```
 10 LET T = 50
 15 REM T = NUMERO DE CUADROS POR JUGADA – INICIALMENTE 50
POR 1 SEGUNDO
 20 SCROLL
 30 LET A$ = CHR$ INT (RND*10 + CODE "0")
 35 REM A$ ES UN DIGITO AL AZAR
 40 PRINT A$
 45 PAUSE T
 50 POKE 16437,255
 60 LET B$ = INKEY$
 70 IF B$ = "Q" THEN GOTO 200
 80 IF A$ = B$ THEN GOTO 150
 90 PRINT "NO VALE"
100 LET T = T*1.1
110 GOTO 20
150 PRINT "OK"
160 LET T = T*0.9
170 GOTO 20
200 SCROLL
210 PRINT "SU PUNTUACION ES"; INT (500/T)
```

5. (Solamente para aquellos que tengan RAM extra). Empleando la rutina de la línea recta del capítulo 18, modifique el programa del segundero de manera que también nos muestre el minutero y la manecilla de las horas, dibujándolas cada minuto. Haga la de las horas mas corta. Si Ud. quiere mas, arrégledlo de manera que cada cuarto de hora le dé algún tipo de señal.

6. (Para divertirse). Plantee lo siguiente:

```
10 IF INKEY$ = "" THEN GOTO 10
20 PRINT AT 11,14; "¡HUY!"
30 IF INKEY$ <> "" THEN GOTO 30
40 PRINT AT 11,14"      "
50 GOTO 10
```

CAPITULO 20

La impresora del ZX81

Si Ud. tiene una impresora del ZX81, con ella le habrán dado las instrucciones para manejarla. Este capitulo describe las sentencias BASIC necesarias para hacerla funcionar.

Las dos primeras, **LPRINT** y **LLIST,** son justamente como **PRINT** y **LIST,** excepto que utilizan la impresora en lugar del televisor. La L es un accidente histórico. Cuando se inventó el BASIC se empleaba normalmente una máquina de escribir eléctrica en lugar de un televisor y así **PRINT** significa realmente imprimir. Si tenía necesidad de obtener una gran cantidad de datos de salida debía utilizar una impresora de líneas muy rápida conectada al computador, y una sentencia **LPRINT** significando **"PRINT** impresora de línea".

Como ejemplo, pruebe éste programa.

```
10 LPRINT "ESTE PROGRAMA:",,,,
20 LLIST
30 LPRINT ,, "IMPRIME EL JUEGO DE CARACTERES.",,
40 FOR N = 0 TO 255
50 LPRINT CHR$ N;
60 NEXT N
```

La tercera sentencia, **COPY,** imprime una copia de la pantalla del televisor. Por ejemplo, consiga en la pantalla un listado del programa anterior, y escriba

```
COPY
```

Ud. siempre puede detener la impresora cuando está funcionando mediante la pulsación de la tecla **BREAK** (espacio).

Si Ud. ejecuta éstas sentencias sin estar conectada la impresora, solamente perderá la impresión y continuará con la sentencia siguiente. Sin embargo, en ocasiones puede molestar el que esté conectada y cuando esto ocurre la tecla "break" lo parara.

Resumen

Sentencias: **LPRINT, LLIST, COPY**

Nota: Ninguna de éstas sentencias es normal en BASIC, aunque **LPRINT** se utiliza en algunos otros computadores.

Ejercicios

1. Plantee lo siguiente:

```
10 FOR N = 31 TO 0 STEP -1
20 PRINT AT 31 - N, N; CHR$ (CODE "0" + N)
30 NEXT N
```

Verá una figura formada por letras que bajan diagonalmente desde la esquina superior derecha hasta alcanzar el pié de la pantalla, donde el programa se detiene con informe de error 5.

Ahora cambie "AT 31 – N, N" en la línea 20 por "**TAB** N". El programa tendrá exactamente el mismo efecto que antes.

Ahora cambie **PRINT** en la línea 20 por **LPRINT.** Esta vez no habrá error 5, lo que no debería ocurrir con la impresora, y el dibujo continuará diez lineas mas con los dígitos.

Ahora cambie "**TAB** N" por "**AT** 21 – N,N" utilizando todavía **LPRINT.** Esta vez obtendrá Ud. solamente una línea de símbolos. La razón de la diferencia es que la salida para **LPRINT** no se efectúa directamente, sino que en una memoria intermedia se ordena una imagen sobre una sola línea horizontal de lo que el computador imprimirá cuando llegue a ello. La impressión tiene lugar.

(i) cuando la memoria intermedia esté llena,
(ii) Despues de una sentencia **LPRINT** que no termina en coma o en punto y coma,
(iii) cuando una coma o partida **TAB** necesite una nueva línea,
o (iv) al final de un programa, si hay algo sin imprimir.

(iii) explica por qué nuestro programa con **TAB** sigue el camino que sigue. Como en el caso de **AT,** no se hace caso del número de línea y la posición **LPRINT** (como la posición **PRINT,** pero para la impresora en lugar del televisor) se cambia por el número de columna. Una partida **AT** nunca puede originar que se envíe una línea a la impresora. Realmente, el número de línea despues de **AT** no es completamente ignorado; tiene que estar entre – 21 y + 21 o dará lugar a un error. Por ésta razón es mas seguro especificar siempre línea 0. La partida "**AT** 21 – N,N" en la versión final de nuestro programa estaría mucho mejor aunque menos ilustrativo si lo cambiásemos por "**AT** 0,N".

2. Haga un gráfico impreso de **SIN** ejecutando el programa del capítulo 18 y después utilizando **COPY.**

CAPITULO 21

Subcadenas

Dada una cadena, una subcadena de ella consiste en algunos caracteres consecutivos de la misma, tomados en orden. Así "CADENA" es una subcadena de "CADENA MAS GRANDE", pero "CADNA G" y "CAD GRAM" no lo son.

Existen una escritura llamada "slicing" – troceado para describir subrutinas, que se puede aplicar a cualquier expresión que sea cadena. La forma general es

expresión en cadena (comienzo **TO** fin)

así que, por ejemplo,

"ABCDEF" (2 **TO** 5) = "BCDE"

Si Ud. omite el comienzo, se supone que es 1 y si omite el final se supone que es hasta el final de la cadena, Entonces

"ABCDEF" (**TO** 5) = "ABCDEF" (1 **TO** 5) = "ABCDE"
"ABCDEF" (2 **TO**) = "ABCDEF" (2 **TO** 6) = "BCDEF"

y

"ABCDEF" (**TO**) = "ABCDEF" (1 **TO** 6) = "ABCDEF"

(ésta última también la puede escribir como "ABCDEF" (), que es equivalente).

Una forma ligeramente diferente toma la omisión de **TO** y tiene solamente un número:

"ABCDEF" (3) = "ABCDEF" (3 **TO** 3) = "C"

Aunque normalmente tanto la parte inicial como final se refieren a partes existentes de la cadena, ésta regla queda anualada por ésta otra: si el comienzo es mayor que el final, el resultado es la cadena vacía. Así

"ABCDEF" (5 **TO** 7)

da error 3 (error en el subescrito) porque la cadena solamente tiene 6 caracteres y 7 es deamsiado grande, pero

"ABCDEF" (8 **TO** 7) = ""

y

"ABCDEF" (1 **TO** 0) = ""

Ni el comienzo ni el final deben ser negativos, u obtendrá error B.

El siguiente programa hace B$ igual a A$, pero omitiendo todos los espacios que se lleven arrastrando.

```
10 INPUT A$
20 FOR N = LEN A$ TO 1 STEP -1
30 IF A$(N) <> " " THEN GOTO 50
40 NEXT N
50 LET B$ = A$ ( TO N)
60 PRINT " " " "; A$; " " " ", " " " "; B$; " " " "
70 GOTO 10
```

Observe como si A$ es toda espacios, entonces en la línea 50 tenemos N = 0 y A$ (**TO** N) = A$ (1 **TO** 0) = " ".

Para variables de cadena, podemos no solo extraer subcadenas, sino también asignarlas a ellas, Por ejemplo, escriba

```
LET A$ = "SIMON QUIERE UN CARAMELO"
```

y a continuaciòn

```
LET A$ (5 TO 8) = "******"
```

y

```
PRINT A$
```

Observe como al ser la subcadena A$ (5 **TO** 8) solamente de 4 caracteres, solo se han utilizado los cuatro primeros asteriscos. Esta es una característica para asignar a las subcadenas: la subcadena tiene que ser exactamente igual de larga antes que despues. Para estar seguros de que esto se cumple, la cadena que está siendo tratada se corta por la derecha si es demasiado larga, o se completa con espacios si es demasiado corta, esto es lo que se llama una traslación Procrusteana desde que el posadero Procrustes la utilizó para asegurarse de que sus huéspedes se ajustaban a la cama, bién alargándolas sobre un caballete o amputándoles los piés.

Si Ud. ahora intenta

```
LET A$() = "PARA QUE "
```

y

```
PRINT A$;"."
```

verá que ha vuelto a ocurrir lo mismo (ésta vez introduciendo los espacios) porque A$() cuenta como una subcadena.

```
LET A$ = "PARA QUE ,,
```

lo hará convenientemente.

Se puede considerar que el "slicing" tiene prioridad 12, así, por ejemplo,

LEN "ABCDEF" (2 **TO** 5) = **LEN** ("ABCDEF" (2 **TO** 5)) = 4

Las expresiones complicadas de cadenas necesitarán ser encerradas entre paréntesis antes de que puedan ser troceadas. Por ejemplo,

"ABC" + "DEF" (1 **TO** 2) = "ABCDE"
("ABC" + "DEF") (1 **TO** 2) = "AB"

Resumen

Troceado, empleando **TO.** Tenga en cuenta que ésta notación no es normalizada.

Ejercicios

1. Algunos BASICs (no el BASIC del ZX81) tienen funciones llamadas LEFT$, RIGHT$, MID$ y TL$.

LEFT$ (A$,N) da la subcadena de A$ formada por los N primeros caracteres.

RIGHT$ (A$,N) da la subcadena de A$ formada por los caracteres desde el N-simo en adelante.

MID$(A$,N1,N2) da la subcadena de A$ formada por N2 caracteres empezando en el N1-simo.

TL$(A$) da la subcadena de A$ formada por todos sus caracteres excepto el primero.

¿Como podría escribirlas en BASIC de ZX81? ¿Sus respuestas trabajarían con cadenas de longitud 0 o 1?

2. Pruebe ésta secuencia de órdenes:

LET A$ = "X" + "Y"
LET A$(2) = **CHR$** 11 (el caracter de comillas de cadena)
LET A$(4) = **CHR$** 11
PRINT A$

¡Ahora A$ es una cadena con comillas de cadena dentro de ella! De ésta manera no hay nada que le detenga a Ud. haciendo esto si Ud. es suficientemente perseverante, pero está claro que si hubiera escrito originalmente

LET A$ = "X" + "Y"

la parte de la derecha del signo igual se habría tratado como una expresión, dando a A$ el valor "XY".

Ahora escriba

```
LET B$ = "X""" + """Y"
```

Encontrará que aunque A$ y B$ parecen lo mismo cuando se imprimen, no son iguales; pruebe

```
PRINT A$ = B$
```

Mientras que B$ contiene nada mas que caracteres de imagen (con código 192), A$ contiene los caracteres auténticos de comillas de cadena (con código 11).

3. Ejecute éste programa:

```
 10 LET A$ = "LEN""ABCD"""
100 PRINT A$;" = ";VAL A$
```

Fallará porque **VAL** no trata la imagen de comillas "" como unas comillas de cadena.

Introduzca algunas líneas extra entre 10 y 100 para sustituir las imágenes de comillas en A$ por comillas de cadena (para lo que deberá llamar a CHR$ 11), y pruebe de nuevo.

Haga el mismo tipo de modificación en el programa del ejercicio 3 del capítulo 9, y experimente con él.

4. Esta subrutina elimina toda incidencia de la cadena "CARTHAGO" procedente de A$.

```
1000 FOR N = 1 TO LEN A$ - 7
1020 IF A$(N TO N + 7) = "CARTHAGO" THEN LET
A$(N TO N + 7) = "********"
1030 NEXT N
1040 RETURN
```

Escriba un programa que dé a A$ varios valores (por ejemplo, "DELENDA EST CARTHAGO.") y aplique la subrutina.

CAPITULO 22

Conjuntos

Supongamos que tiene una lista de números, por ejemplo el número de multas de circulación que le han puesto cada mes en el actual año. Para guardarlos en el computador Ud. podría establecer una variable sencilla para cada mes, pero encontrará que esto es muy embarazoso. Ud. podría decidirse por llamar a las variables solo 1, solo 2, y así hasta solo 12, pero el programa para imprimir estos doce números será mas bien largo y pesado de escribir.

Sería mucho mas bonito si Ud. pudiese escribir esto:

```
 5 REM ESTE PROGRAMA NO TRABAJARA
10 FOR N = 1 TO 12
20 PRINT SOLO N
30 NEXT N
```

Bien, pero no puede.

Sin embargo, existe un mecanismo por el cual Ud. puede aplicar ésta idea, y es utilizar conjuntos. Un conjunto es un juego de variables, sus elementos, todas con el mismo nombre y que se diferencian solamente por un número (el subíndice) escrito entre paréntesis a continuación del nombre. En nuestro ejemplo el nombre sería A (como en el caso de las variables de control para bucles **FOR-NEXT**, el nombre de un conjunto debe ser una sola letra) y entonces las doce variables serían A(1), A(2), etc, hasta A(12).

Los elementos de un conjunto se llaman variables con subíndice, para distinguirlas de las variables sencillas que ya nos son familiares.

Antes de que haga uso de un conjunto, debe reservar algún espacio dentro del computador, para lo cual utilizará la sentencia **DIM** (de dimensión).

```
DIM A(12)
```

establece un conjunto llamado A con dimensión 12 (es decir, hay 12 variables con suíndice A(1), . . ., A(12)) y asigna los 12 índices hasta el 0. Al mismo tiempo borra cualquier conjunto llamado A que existiera previamente, pero nó una variable sencilla. Pueden coexistir un conjunto y una variable numérica sencilla con el mismo nombre, y no hay posibilidad de confusión entre ellas puesto que la variable de conjunto siempre tiene un subíndice.

El subíndice puede ser cualquier expresión numérica y así puede Ud. escribir.

```
10 FOR N = 1 TO 12
20 PRINT A(N)
30 NEXT N
```

También puede establecer conjuntos con mas de una dimensión. En un conjunto de dos dimensiones Ud. necesita dos números para definir cada elemento, parecido a los números de línea y columna para definir la posición de un caracter sobre la pantalla de televisión, por lo que el conjunto adquiere la forma de una tabla. Opcionalmente, si Ud. se imagina que los

números de línea y columna (dos dimensiones) están sobre una página impresa, tendra una dimensión mas por el número de página. Desde luego, estamos hablando de conjuntos numéricos, por lo que los elementos no son letras como en un libro, sino números. Piense que los elementos de un conjunto tridimensional C están definidos por C (número de página, número de línea, número de columna).

Por ejemplo, para establecar un conjunto bidimensional B con dimensiones 3 y 6, Ud. utiliza la sentencia **DIM**

DIM B(3,6)

Que le proporciona a Ud. 3*6 = 18 variables con subíndice

B(1,1), B(1,2), . . . , B(1,6)
B(2,1), B(2,2), . . . , B(2,6)
B(3,1), B(3,2), . . . , B(3,6)

El mismo principio sirve para cualquier número de dimensiones.

Aunque Ud. pueda tener un número y un conjunto con el mismo nombre, sin embargo no puede tener dos conjuntos con el mismo nombre aunque tengan diferentes números de dimensiones.

También hay conjuntos de cadenas. Las cadenas de un conjunto se diferencian de las simples en que son de una longitud fija y su traslación es siempre procrusteana, otra manera de imaginarselas es como conjuntos (con una dimensión mas) de caracteres sencillos. El nombre de un conjunto de cadenas es una sola letra seguida de $, y un conjunto de cadenas y una variable sencilla de cadena no pueden tener el mismo nombre (a diferencia del caso de los números).

Supongamos entonces, que Ud. quiere un conjunto A$ de cinco cadenas. Debe decidir la longitud de las mismas, supongamos que con diez caracteres en cada una tenemos bastante. Así pues Ud. dice

DIM A$(5,10) (escríbalo)

Lo que establece un conjunto de 5 * 10 caracteres, pero Ud. puede imaginarse que cada fila forma una cadena:

A$(1) =	A$(1,1)	A$(1,2)	. . .	A$(1,10)
A$(2) =	A$(2,1)	A$(2,2)	. . .	A$(2,10)
:	:	:	:	:
A$(5) =	A$(5,1)	A$(5,2)	. . .	A$(5,10)

Si Ud. da el mismo número de subíndices (dos en éste caso) como si fueran dimensiones en una sentencia **DIM,** obtendrá un solo caracter; pero si omite el último, obtendrá una cadena de longitud fija. Así, por ejemplo, A$(2,7) es el 7° caracter de la cadena A$(2); empleando la notación de troceado, podríamos haber escrito esto como A$(2)(7). Ahora escriba

LET A$(2) = "1234567890"

y

PRINT A$(2), A$(2,7)

Obtendrá

1234567890 7

Para el último subíndice (el que Ud. puede olvidar) también puede tener un troceador, por ejemplo

A$(2,4 **TO** 8) = A$(2)(4 **TO** 8) = "45678"

Recuerde:

En un conjunto de cadenas, todas tienen la misma longitud especificada.
La sentencia **DIM** tiene un número mas (el último) para especificar su longitud.
Cuando Ud. consigna una variable con subíndice para un conjunto de cadenas, puede poner un número mas, o un troceador, para adaptarse al número extra de la sentencia **DIM**.

Resumen

Conjuntos (la manera en que el ZX81 maneja conjuntos de cadenas, no esta generalizada en otros Basic.
Sentencias: **DIM**

Ejercicios

1. Establezca un conjunto M$ de doce cadenas en el que M$(N) es el nombre del N-simo mes. (Sugerencia: la sentencia **DIM** será **DIM** M$(12,9).) Prúebelo imprimiendo todas las M$(N) (use un bucle).
Escriba

PRINT "AHORA ES EL MES DE"; M$(5) "CUANDO FLORECEN LAS ROSAS MAS BELLAS"

¿Que puede hacer con todos estos espacios?

2. Ud. puede tener conjuntos de cadenas sin dimensiones. Escriba

DIM A$(10)

y verá que A$ se comporta justamente como una variable de cadena, excepto que siempre tiene una longitud de 10, y su traslacion siempre es procrusteana.

3. READ, DATA y RESTORE; ¿quien las necesita?

La mayoría de los BASIC (pero no el BASIC del ZX81) tienen tres sentencias llamadas READ, DATA y RESTORE.

Una sentencia DATA es una lista de expresiones, es decir, las sentencias DATA en un programa proporcionan una lista de expresiones, la lista DATA.

Las sentencias READ se utilizan para asignar éstas expresiones, una a una, a variables:

```
READ X
```

por ejemplo, asigna la expresión en curso en la lista DATA a la variable X, y se va a la siguiente expresión para la próxima sentencia READ.

```
(RESTORE retrocede al principio de la lista DATA)
```

En teoría, siempre puede sustituir sentencias READ y DATA por sentencias **LET**; sin embargo, una de sus mayores utilizaciones es para poner iniciales a los conjuntos, como en éste programa:

```
 5 REM ESTE PROGRAMA NO TRABAJARA EN ZX81 BASIC
10 DIM M$(12,3)
20 FOR N = 1 TO 12
30 READ M$(n)
40 NEXT N
50 DATA "ENE", "FEB", "MAR", "ABR"
60 DATA "MAY", "JUN", "JUL", "AGO"
70 DATA "SEP", "OCT", "NOV", "DIC"
```

Si solo quiere efecutar éste programa una vez, podría igualmente reemplazar la línea 30 por una sentencia **INPUT** así:

```
10 DIM M$(12,3)
20 FOR N = 1 TO 12
30 INPUT M$(N)
40 NEXT N
```

y no tendrá que escribir nada mas. No obstante, si quiere conservar el programa, por supuesto que no querrá tener que escribir los meses cada vez que lo ejecute.

Le sugerimos que siga éste método:

(i) Inicialice el conjunto usando un programa como el anterior.

(ii) Edite el programa de inicialización. (No utilice **NEW,** ya que quiere conservar el conjunto)

(iii) Escriba el resto del programa, y consérvelo. Así conservará también las variables, incluyendo el conjunto.

(iv) Cuando cargue el programa, cargará también el conjunto.

(v) Cuando ejecute el programa, no utilice **RUN,** que le borra las variables. En su lugar utilice **GOTO** con un número de línea.

Ud. debería ser capáz de utilizar **LOAD** y la técnica de ejecución del capìtulo 16, y su ejercicio 3. Entonces, en la etapa (iii) anterior usará una sentencia **SAVE** en el programa, y no tendrá que tener en cuenta el punto 5.

CAPITULO 23

Cuando el computador está lleno

El ZX81 tiene una capacidad limitada de memoria interna y no es dificil llenarla. Normalmente, la mejor señal de que está ocurriendo esto es un informe 4 de error, pero pueden ocurrir otras cosas y algunas de ellas son bastante extrañas. El comportamiento exacto depende de si Ud. tiene enchufada una placa de extensión de memoria, por lo que primeramente supondremos que no la tiene. Si Ud. la tiene, desenchúfela (habiendo desconectado primero el computador).

El archivo de imagen, esto es, la zòna dentro del computador donde almacena la imagen de televisión, está astutamente diseñado de manera que solamente ocupa el espacio para lo que hemos impreso hasta ahora: una línea en la imagen consiste en 32 caracteres y a continuación un caracter **NEWLINE.** Esto significa que puede sacar algo de la memoria por la impresión de alguna cosa, y el momento mas oportuno es mientras se está haciendo un listado. Escriba

```
NEW
DIM A(150)
10 FOR N = 1 TO 15
20 PRINT N
```

Aquí llega la primera sorpresa: la línea 10 desaparece del listado. El listado se ajusta para incluir la línea actual, 20, y no hay sitio para ambas líneas. Ahora escriba.

```
30 NEXT N
```

De nuevo, solamente hay sitio para la línea 30 en el listado, Escriba

```
40 REM X        (sin NEWLINE)
```

y verá como desaparece la línea 30 y la 40 salta a la parte superior de la pantalla; no ha sido introducida en el programa, todavía tiene el cursor **L** y puede moverlo acá y allá. Todo lo que Ud. ha visto es algún oscuro mecanismo que da prioridad a la mitad inferior de la pantalla sobre la mitad superior. Escriba

```
XXXXXX          (todavía sin NEWLINE)
```

y el cursor desaparecerá, no hay sitio para representarlo. Escriba otra X, sin **NEWLINE,** y una de las Xs desaparecerá. Ahora pulse **NEWLINE** y todo desaparecerá, pero el programa está todavía en el computador, como puede comprobar suprimiendo la línea 10 y utilizando ⌂ y ⌔. Ahora escriba

```
10 FOR N = 1 TO 15
```

otra vez; se moverá hasta la parte superior de la pantalla como hizo la línea 40. Pero cuando pulse **NEWLINE,** no será registrada, aunque no haya mensaje de error ni indicador **S** que

nos diga que algo anda mal. Este es el resultado de no haber sitio para comprobar la sintaxis de una línea, y normalmente solo tiene lugar para líneas que contengan números (que no sean el número de línea del principio).

La única solución es hacer algo de sitio de un modo u otro, pero primero borre la línea 10 que no deberá ir. Pulse **EDIT:** la pantalla se quedará en blanco, porque no hay sitio para continuar la línea.

Cuando **EDIT** no da resultado, a veces es posible hacer sitio escribiendo una serie de espacios hasta que al cursor vuelva a la pantalla.

Pulse **NEWLINE** y conseguirá que vuelva parte del listado. Ahora borre la línea (la cual de todos modos no la quería) escribiendo

```
40      (y NEWLINE)
```

Ahora trate de escribir de nuevo en la línea 10, y no lo conseguirá todavía. Ud. aún debe encontrar algún hueco en alguna parte. Tenga en mente que la razón por la que la línea 10 fué rechazada probablemente era que no había sitio para comprobar la sintaxis de los dos números, 1 y 15: por lo que si Ud. borra la línea 20 en el programa podrá tener sitio para meter la línea 10, y todavía le sobrará para despues reintroducir la 20. Inténtelo. Escriba

```
20
10 FOR N = 1 TO 15
20 PRINT N
```

y el programa es registrado adecuadamente.

Escriba

```
GOTO 10
```

y de nuevo se encontrará con que ésta línea es rechazada porque no se puede comprobar su sintaxis; pero si Ud. lo borra y escribe

```
RUN
```

funcionará. (**RUN** borra el conjunto, dejándole un montón de espacio).

Ahora escriba lo mismo que antes desde **NEW** hasta la línea 30, y a continuación

```
40 REM XXXXXXXXXXXX
```

(12 Xs), que acabará viéndose como 40 RE. Cuando pulse **NEWLINE,** el listado estará formado solamente por la línea 30, y de hecho la línea 40 se ha perdido completamente, porque era demasiado larga para encajar en el programa. El efecto es un poco peor cuando la línea es una versión alargada de una línea que ya está en el programa, pues perderá tanto la antigua línea del programa como la nueva que debería sustituirla.

La solución definitiva a estos problemas de ocupacion, es acoplar un paquete RAM a la parte posterior del computador.

El paquete RAM Sinclair de 16K proporciona al computador una memoría dieciseis veces mayor que la que tiene incorporada.

El comportamiento con el paquete RAM es bastante distinto ya que el archivo de imagen se rellena con espacios para hacer cada línea de 32 caracteres (observe que **SCROLL** trastorna todo esto, vea capítulo 27). Ahora la impresión y confección de listados no hará que el computador sobrecargue la memoria y no verá todos sus listados recortados, y saltando alrededor; pero todavía verá a las líneas detenerse o perderse, y nuevamente la única solución es encontrar mas espacio de reserva.

Si Ud. tiene una placa de extensión de memoria, móntela y examine detenidamente lo escrito en este capítulo, usando

DIM A(3Ø69)

para sustituir a **DIM** A(15Ø).

Para resumir, éstar es una historia embrollada y la moraleja es evitar el conseguir un computador totalmente atascado si es posible. Pero, la segunda moraleja es que las cosas no suelen estar tan mal como parecen.

1. Si el listado empieza a cortarse o las cosas empiezan a saltar alrededor, entonces es que el espacio empieza a ser escaso.
2. Si **NEWLINE** parece no surtir efecto al final de una línea, probablemente no hay sitio para tratar algún número. Borre la línea utilizando **EDIT-NEWLINE** o **RUBOUT.**
3. **NEWLINE** puede malograr toda una línea.

Para todas estas singularidades, el remedio es el mismo. No asustarse, y buscar algún espacio vacío.

La primera cosa a tener en cuenta es **CLEAR.** Si Ud. tiene algunas variables y no le importa perder algunas, es lo que tiene que hacer.

Si esto falla, busque sentencias innecesarias en el programa, tales como sentencias **REM,** y borre algunas de ellas.

Resumen

Cuando la memoria está llena pueden ocurrir cosas extrañas; pero normalmente no suelen ser irremediables.

CAPITULO 24

Contando con los dedos

El próximo capítulo profundiza un poco en el computador, pero antes veamos lo que sirve tambien para describir como calculan los computadores: lo hacen utilizando el sistema binario, lo que significa que solo utilizan dos dedos.

La mayoria de los paises europeos calculan utilizando una configuracion mas o menos regular de decenas, como en España:

veinte, veintiuno, veintidos, . . ., veintinueve
treinta, treinta y uno, treinta y dos . . ., treinta y nueve
cuarenta, cuarenta y uno, cuarenta y dos, . . ., cuarenta y nueve etc.

y esto se hace incluso mas sistemáticamente con los números arábigos que utilizamos. Pero, la única razón de usar diez es que da la casualidad que tenemos diez dedos.

Supongamos ahora unos marcianos que tengan tres dedos mas en cada mano: en lugar de usar nuestro sistema decimal, con el diez como base, ellos utilizan un sistema hexadecimal (o hex, en abreviatura), basado en el dieciseis. Necesitarían seis dígitos hex mas, además de los diez que nosotros usamos, y los escribirían como A, B, C, D, E y F. ¿Y que vien después de F?. Igual que nosotros, con diez dedos escribimos 10 para diez, así ellos con dieciseis, escribirá 10 para dieciseis. Su sistema de numeración empieza así:

Hex	*Castellano*
0	cero
1	uno
2	dos
:	:
:	:
9	nueve

igual que hacemos nosotros, pero continúa así

A	diez
B	once
C	doce
D	trece
E	catorce
F	quince
10	dieciseis
11	diecisiete
:	:
:	:
19	veinticinco
1A	veintiseis

Hex	*Castellano*
1B	veintisiete
:	:
:	:
1F	treinta y uno
20	treinta y dos
21	treinta y tres
:	:
:	:
9E	ciento cincuenta y ocho
9F	ciento cincuenta y nueve
A0	ciento sesenta
A1	ciento sesenta y uno
:	:
:	:
FE	doscientos cincuenta y cuatro
FF	doscientos cincuenta y cinco
100	doscientos cincuenta y seis

Si Ud. está utilizando notación hexadecimal y quiere hacerlo bastante fácil, escriba "h" al final del número y diga "hexadecimal". Por ejemplo, para ciento cincuenta y ocho, escriba "9Eh" y diga "nueve E hexadecimal".

Estará asombrado de que todo esto se haga con computadores. De hecho, los computadores se comportan como si solo tuvieran dos dígitos, representados por una tensión baja, o corte (0), y una tensión alta, o conducción (1). Esto se conoce como sistema binario, y los dos dígitos binarios se llaman bits: así un bit puede ser 0 o 1.

En los diversos sistemas, la cuenta empieza así:

Castellano	*Decimal*	*Hexadecimal*	*Binario*
cero	0	0	0 o 0000
uno	1	1	1 o 0001
dos	2	2	10 o 0010
tres	3	3	11 o 0011
cuatro	4	4	100 o 0100
cinco	5	5	101 o 0101
seis	6	6	110 o 0110
siete	7	7	111 o 0111
ocho	8	8	1000
nueve	9	9	1001
diez	10	A	1010
once	11	B	1011
doce	12	C	1100
trece	13	D	1101
catorce	14	E	1110
quince	15	F	1111
dieciseis	16	10	10000

El punto importante es que dieciseis es dos elevado a la cuarta potencia, lo que hace muy sencilla la conversión entre hexadecimal y binario.

Para convertir hexadecimal a binario, cambie cada dígito hexadecimal por cuatro bits, utilizando la tabla anterior.

Para convertir binario a hexadecimal, divida el número binario en grupos de cuatro bits, empezando por la derecha y a continuación convierta cada grupo en el correspondiente dígito hex.

Por ésta razón, aunque hablando con exactitud los computadores emplean un sistema binario puro, los humanos suelen escribir los números almacenados en un computador utilizando la notación hexadecimal.

Dentro del computador, los bits están en su mayor parte agrupados en juegos de ocho o bytes. Un solo byte puede representar cualquier número entre cero y doscientos cincuenta y cinco (11111111 binario o FF hex), o cualquier caracter del juego de caracteres del ZX81. Este valor se puede escribir con dos dígitos hexadecimales.

Se pueden agrupar dos bytes juntos para formar lo técnicamente se llama una palabra. Una palabra se puede escribir empleando dieciseis bits o cuatro dígitos hex, y representa un número comprendido entre Ø a (en decimal) $2^{16} - 1 = 65535$.

Un byte son siempre ocho bits, pero las palabras varían de computador a computador.

Resumen

Sistemas decimal, hexadecimal y binario.

Bits y bytes (no los confunda) y palabras.

Ejercicios

1. La unidad monetaria marciana es la libra, y está dividida en dieciseis onzas. ?Como podría convertir libras y onzas en onzas y viceversa

   (i) si todos los números están escritos en decimal?

   (ii) si todos los números están escritos en hex?

2. ?Como haría la conversión entre decimal y hex? (Sugerencia: ejercicio 1).

   Escriba programas en el ZX81 para convertir valores numéricos en cadenas dando su representación hexadecimal y viceversa. (Esto es lo que **STR$** y **VAL** hacen con representaciones decimales.)

3. Suponga que la gente de Venus tiene un total de ocho dedos, sin pulgares, ?que utilidad tendría en los computadores su sistema de numeración octal (en base ocho)?

CAPITULO 25

Como trabaja el computador

La fotografía de la página siguiente muestra el interior del ZX81 (pero no lo desarme Ud. mismo porque le puede resultar un asunto espinoso el volver a montarlo).

Como puede ver cada cosa tiene una abreviatura de tres letras.

Las piezas de plástico con montones de patitas metálicas son las maravillosas pastillas de silicio, que están no solo en los relojes digitales sino tambien en el Sinclair ZX81. Dentro de cada pieza de plástico hay un trozo de silicio aproximadamente del tamaño del cursor **K** , unido con hilos a las patillas metálicas.

El cerebro que gobierna el funcionamiento es la pastilla del procesador, frecuentemente llamada la CPU (Central Processor Unit – unidad central de proceso). En este caso particular es un procesador Z80 (actualmente un Z80A, que es mas rápido).

El procesador controla al computador completo, hace los cálculos, estudia que teclas ha pulsado Ud., decide que hacer como resultado y gobierna la imagen del televisor. Sin embargo, a pesar de toda su habilidad, no puede hacer todo esto por si solo. Abandonado asi mismo, no sabe nada a cerca de BASIC, cálculos con coma flotante, o televisores, y tiene que conseguir todas sus instrucciones de otra pastilla, la ROM (Read Only Memory – memoria solo legible). La ROM es justamente una larga lista de instrucciones que constituyen un programa completo de computador que le dice al procesador lo que tiene que hacer bajo todas las circunstancias previsibles. Este programa no está escrito en BASIC, si no en lo que se llama código de máquina Z80, y toma la forma de una larga serie de bytes. (Recuerde que un byte es un número entre 0 y 255). Cada byte tiene una dirección indicando cual es su paradero en la ROM; el primero tiene como dirección 0, el segundo tiene como dirección 1 y asi sucesivamente hasta 8191, lo que hace un total de 8192 = 8*1024 bytes, razon por la cual éste ZX81 BASIC se dice que es un BASIC de 8K. 1K es 1024.

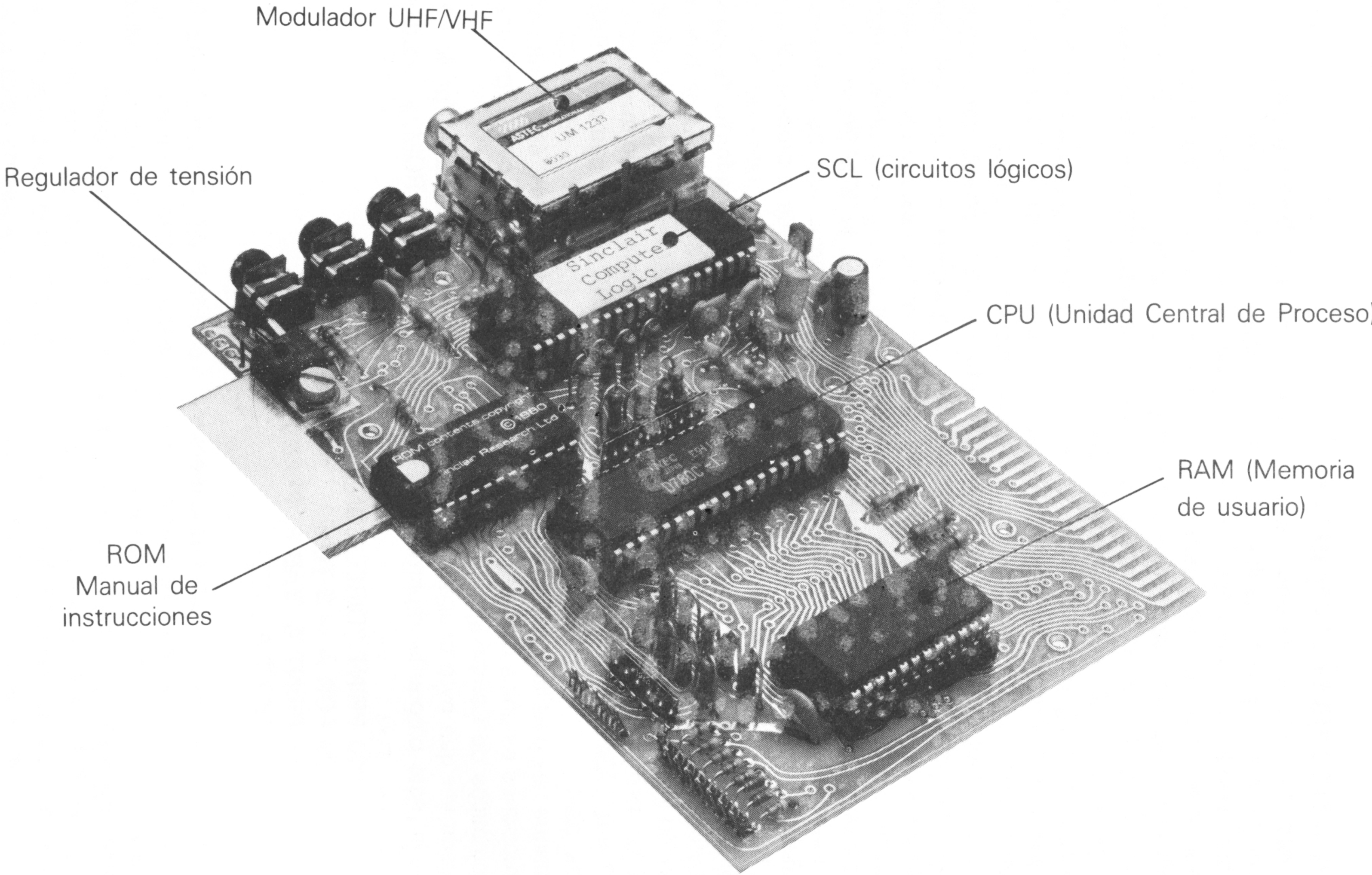
Aunque hay pastillas similares en muchas máquinas diferentes, ésta serie particular de instrucciones es única para el ZX81 y fué escrita especialmente para él por una firma de matemáticos de Cambridge.

Puede ver que byte ocupa una determinada dirección utilizando la función **PEEK.** Por ejemplo, éste programa imprime los 21 primeros bytes de la ROM, y sus direcciones.

```
10 PRINT "DIRECCION"; TAB 8; "BYTE"
20 FOR A = 0 TO 20
30 PRINT A; TAB 8; PEEK A
40 NEXT A
```

La siguiente pastilla a estudiar es la memoria, o pastilla RAM (Random Access Memory – memoria de acceso al azar). Aquí es donde el procesador guarda la información que quiere conservar, su programa BASIC, las variables, la imagen para el televisor y varias notas (las variables del sistema) sobre en que estado se encuentra el computador.

Como la ROM, la memoria está dispuesta en bytes, cada uno con una dirección: el intervalo de direcciones de 16384 a 17407 (o hasta 32767, si Ud. tiene el paquette RAM de 16K). Como en el caso de la ROM, puede encontrar los valores de estos bytes usando


Modulador UHF/VHF
Regulador de tensión
SCL (circuitos lógicos)
CPU (Unidad Central de Proceso)
RAM (Memoria de usuario)
ROM
Manual de instrucciones
Sinclair
Compute
Logic

PEEK, pero con la gran diferencia con relación a la ROM de que puede cambiarlos. Por supuesto, la ROM es fija e inalterable.

Escriba

POKE 17300,57

Lo que hace que el byte con dirección 17300 tenga el valor 57. Si Vd. ahora escribe

PRINT PEEK 17300

Vuelve a obtener su número 57. (Pruebe otros valores, para demostrar su utilidad).

Observe que la dirección tiene que estar en 0 y 65535 y que la mayoría de ellas se refieren a bytes en la ROM o en ninguna parte y por consiguiente no causarán efecto. El valor debe estar comprendido entre – 225 y + 255, y si es negativo se consigue sumandole 256.

La capacidad de acceso a la memoria le da a Vd. un inmenso poder sobre el computador si sabe como usarla; pero los conocimientos necesarios son algo mas de lo que se puede impartir en un manual de presentación como es éste.

La última pastilla grande es la pastilla lógica, o pastilla SCL (Sinclair Computer Logic – Lógica del Computador Sinclair). Esta también fué diseñada y fabricada especialmente para el ZX81 y conecta a las otras pastillas entre sí de tal manera que las obliga a hacer mas de lo que harían por sí solas.

El modulador convierte la salida de televisión del computador en una forma adecuada para el televisor, y el regulador convierte los uniformes, pero no regulados 9 voltios de la fuente de alimentación en 5 voltios regulados.

Resumen

Pastillas: Circuitos
Sentencias: **POKE**
Funciones: **PEEK**

CAPITULO 26

Usando el código de máquina

Este capítulo está escrito para aquellos que entienden el código de máquina Z80, el juego de instrucciones que utiliza la pastilla del procesador Z80. Si Ud. no está en éste caso, pero le gustaría estarlo, existen libros acerca del tema; dos que le pueden servir como preparación son "Programando el Z80" por Rodnay Zaks, publicado por Sybex y "Programando el Z80 y el 8080 con lenguaje Assembly" por Rathe Spracklen, publicado por Hayden.

La máxima autoridad es el "Manual de programación en lenguaje Assembly del Z80" junto con el "Manual técnico del Z80-CPU y del Z80A-CPU", publicados por Zilog, pero son un poco fuertes para ser recomendados a personas no expertas.

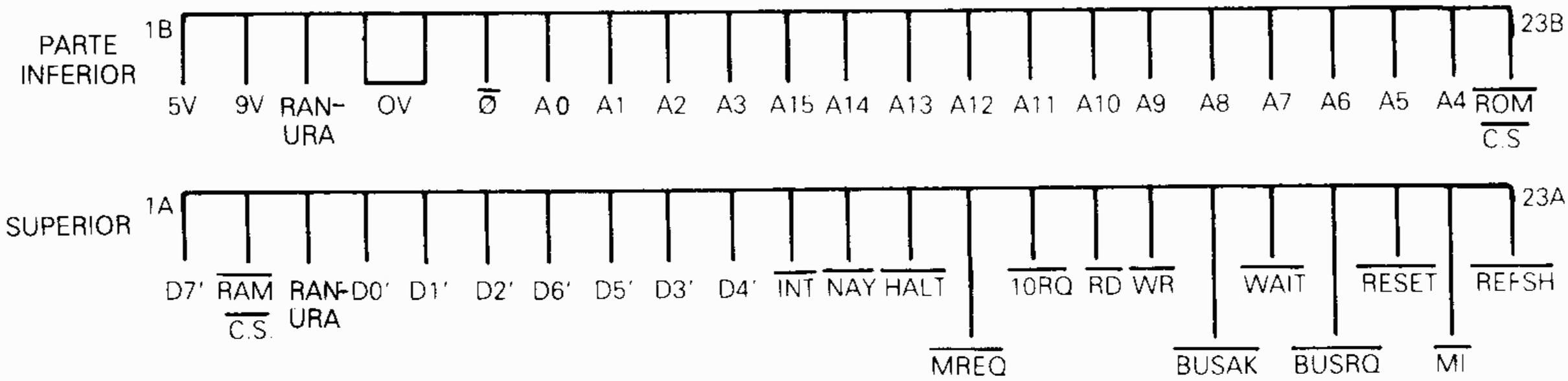
Las rutinas en código de máquina se pueden ejecutar desde dentro de un programa BASIC empleando la función **USR.** El argumento de **USR** es la dirección de inicio de la rutina, y su resultado es un número entero sin signo de dos bytes, el contenido del par de registros bc en retorno. La dirección de retorno al BASIC está amontonada de la manera usual, y así el retorno se efectúa por una instrucción de retorno Z80.

Existen ciertas restricciones sobre las rutinas **USR**:

(i) En retorno, los registros iy e i deben tener los valores 4000 h y 1Eh.

(ii) La rutina de representación visual utiliza los registros a', f', ix y r, por lo que una rutina **USR** no debería utilizarlos si están funcionando el cómputo y la pantalla. (No siempre es seguro que lea el par af'.)

Los Bus de control de datos y direcciones aparecen todas en la parte posterior del ZX81, y así casi todo lo que hace con un Z80 lo puede hacer con un ZX81. Sin embargo, el hardware del ZX81 puede sacar ventaja en funcionamiento, especialmente en cómputo e imagen. Este es un diagrama de las conexiones que aparcen en la parte de atrás.



Un trozo de código de máquina en medio de la memoria corre el riesgo de que le escriban encima por el sistema BASIC. Algunos sitios seguros son:

(i) En una sentencia **REM**; escriba en una sentencia **REM** con suficientes caracteres para que tenga cabida su código de máquina cuando lo busque después. Haga esto en la primera línea del programa, o muévala hasta que ocupe ese lugar. Evite las instrucciones de parada, ya que podrían ser identificadas como final de la sentencia **REM.**

(ii) En una cadena: prepare una cadena de suficiente longitud, y asigne un byte de código

de máquina a cada caracter. Las cadenas siempre están sujetas a moverse a cualquier lugar dentro de la memoria.

En el apéndice A, el juego de caracteres, encontrara estos y las instrucciones Z80 escritas por orden una al lado de la otra y puede serle de utilidad cuando registre códigos.

(iii) En lo alto de la memoria. Cuando se conecta el ZX81, efectúa un chequeo para ver cuanta memoria hay y pone el compartimento de máquina justo en la parte superior por lo que allí no hay sitio para las rutinas **USR.** Almacena la dirección del primer byte no existente (p.e., 17K, o 17408, si Ud. tiene 1K de memoria) en una variable del sistema conocida como RAMTOP, en los dos bytes con direcciones 16388 y 16389. Por otro lado, **NEW** no hace una comprobacion de toda la memoria, sino solamente hasta delante de la direccion en RAMTOP. Asi si Ud. busca la dirección del byte que está dentro de RAMTOP, haciendo **NEW** toda la memoria a partir de ese byte en adelante está fuera del sistema BASIC y lo deja de lado. Por ejemplo, supongamos que tiene 1K de memoria y acaba de conectar el computador.

```
PRINT PEEK 16388 + 256*PEEK 16389
```

le dice a Ud. la dirección (17408) del primer byte inexistente.

Ahora supongamos que tiene una rutina **USR** de 20 bytes de longitud, y quiere cambiar RAMTOP a 17388 = 236 + 256*67 (¿Como solucionaría esto en el computador?), por lo tanto escriba

```
POKE 16388,236
POKE 16389,67
```

y a continuación **NEW.** Los veinte bytes de memoria desde la dirección 17388 hasta la 17407 son ahora suyos para hacer lo que quiera con ellos. Si ahora vuelve a escribir otra vez **NEW,** no afectará a estos veinte bytes.

El principio de la memoria es un buen lugar para las rutinas **USR,** seguro (incluso para **NEW)** e inamovible. Su principal desventaja radica en que no es conservado por **SAVE.**

Resumen

Funciones: **USR**
Sentencias: **NEW**

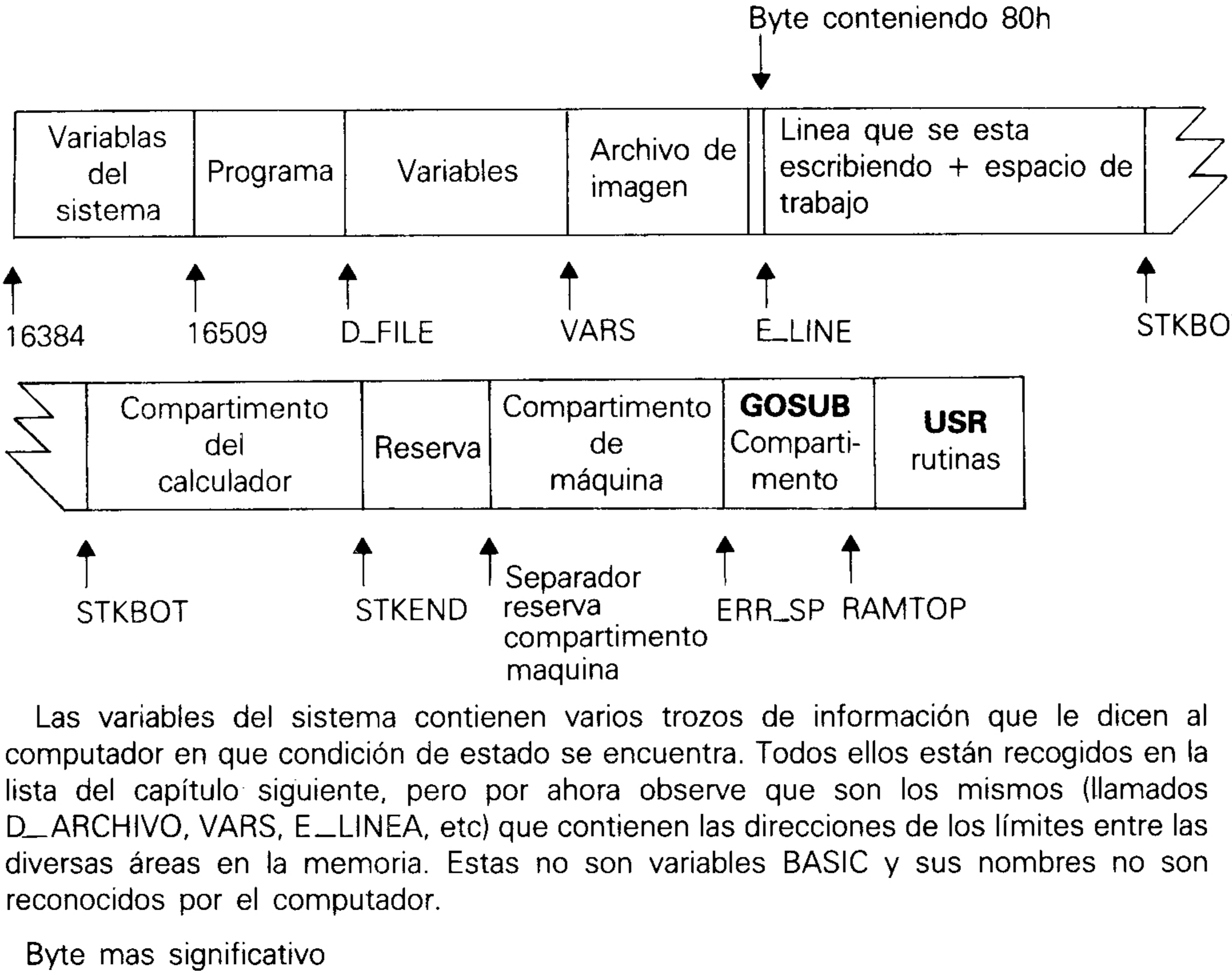
Ejercicios

1. Haga RAMTOP igual a 16700 y a continuación ejecute **NEW**. Obtendrá una idea de lo que ocurre cuando la memoria está llena.

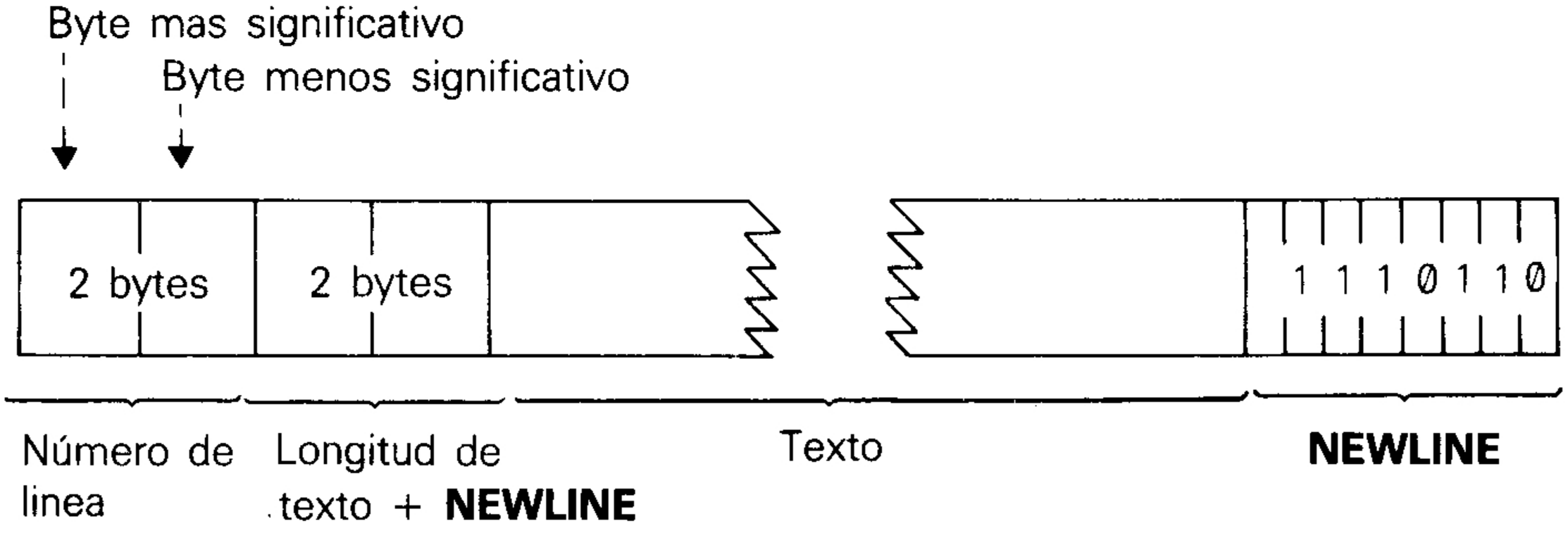
CAPITULO 27

Organización de la memoria

La memoria está dividida en diferentes áreas para almacenar diferentes clases de información. Las áreas tienen el tamaño justo para la información que realmente deben contener, y si Ud. introduce algo mas en un punto determinado (por ejemplo, añadiendo una variable o línea de programa) se hace sitio por desplazamiento hacia arriba de todo lo que hay antes de ese punto. Recíprocamente si Ud. elimina información, entonces se corre todo lo anterior hacia abajo.



Las variables del sistema contienen varios trozos de información que le dicen al computador en que condición de estado se encuentra. Todos ellos están recogidos en la lista del capítulo siguiente, pero por ahora observe que son los mismos (llamados D_ARCHIVO, VARS, E_LINEA, etc) que contienen las direcciones de los límites entre las diversas áreas en la memoria. Estas no son variables BASIC y sus nombres no son reconocidos por el computador.



Observe que en contraste con los otros casos de números de dos bytes en el Z8Ø, aquí el número de línea (y también en una variable de control **FOR-NEXT**) es almacenado con su byte mas significativo en primer lugar: es decir, en el orden en que Ud. los escribe en orden descendente.

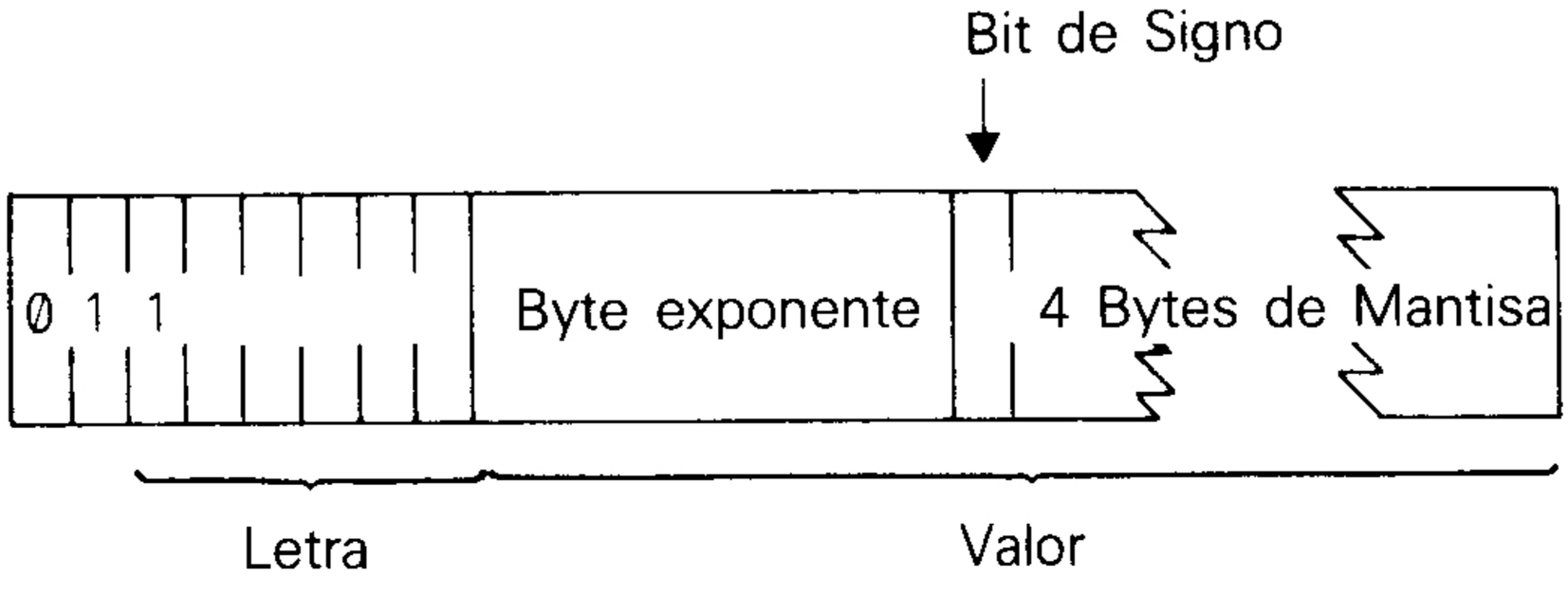
En el programa, una constante numérica está seguida por su forma binaria, usando el caracter **CHR$** 126 seguido de cinco bytes para el número en sí.

El archivo de imagen es la copia en memoria de la imagen del televisor. Empieza con un caracter **NEWLINE** y a continuación tiene veinticuatro líneas de texto, cada una acabada con un **NEWLINE.** El sistema está diseñado de tal manera que una línea de texto no necesita todo el espacio de treinta y dos caracteres: se pueden omitir espacios del final. Esto se utiliza para economizar espacio cuando la memoria es pequeña.

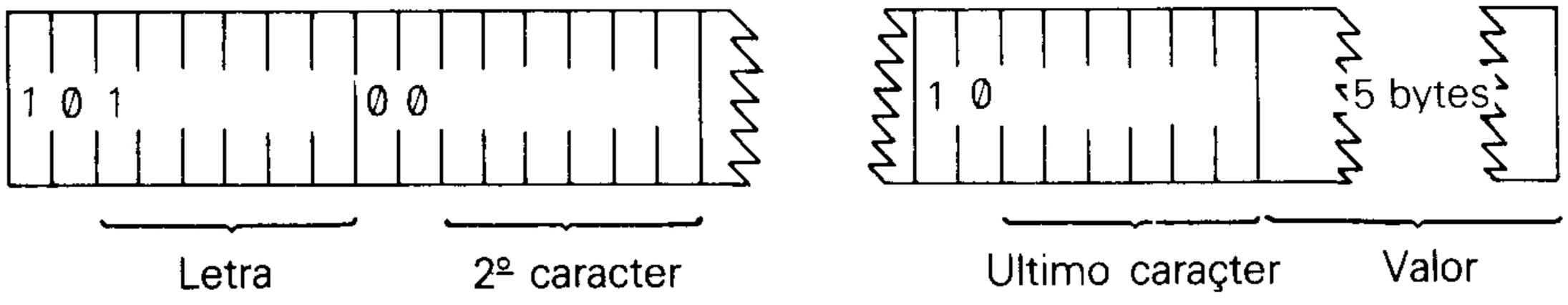
Cuando la cantidad total de memoria (de acuerdo con la variable del sistema RAMTOP) es inferior a 3¼K, una pantalla libre, tal como se encuentra al empezar o despues de **CLS,** está formada justamenpor veinticinco **NEWLINEs.** Cuando la memoria es mayor, una pantalla libre se rellena con 24*32 espacios y en general permanece en todo su tamaño; no obstante, **SCROLL** y ciertas condiciones en que la parte inferior de la pantalla se ensancha a mas de dos líneas, pueden desordenar esto introduciendo líneas cortas al pié de la pantalla.

Las variables tienen diferentes formatos según sus diferentes naturalezas.

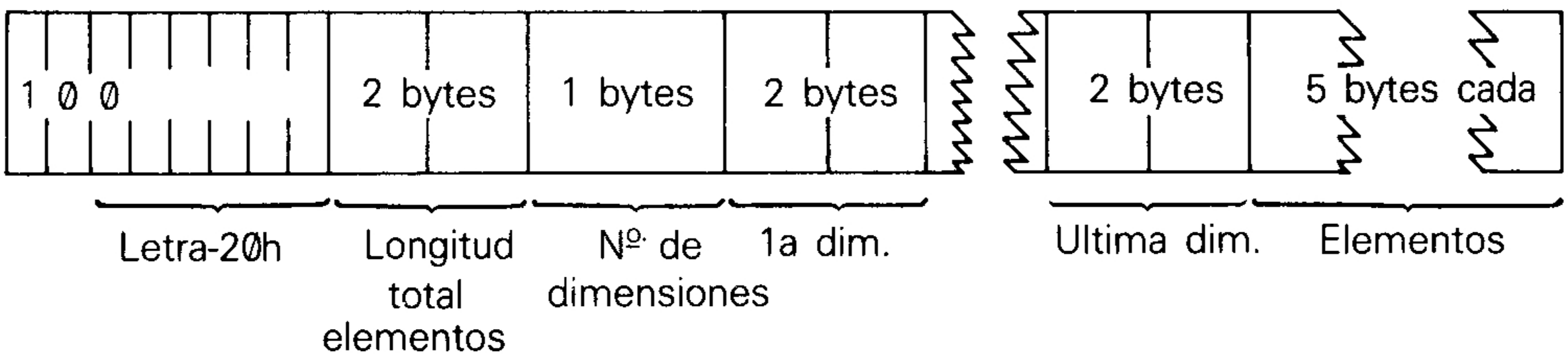
Número cuyo nombre es una sola letra:



Número cuyo nombre es mas largo de una letra:



Conjunto de números:



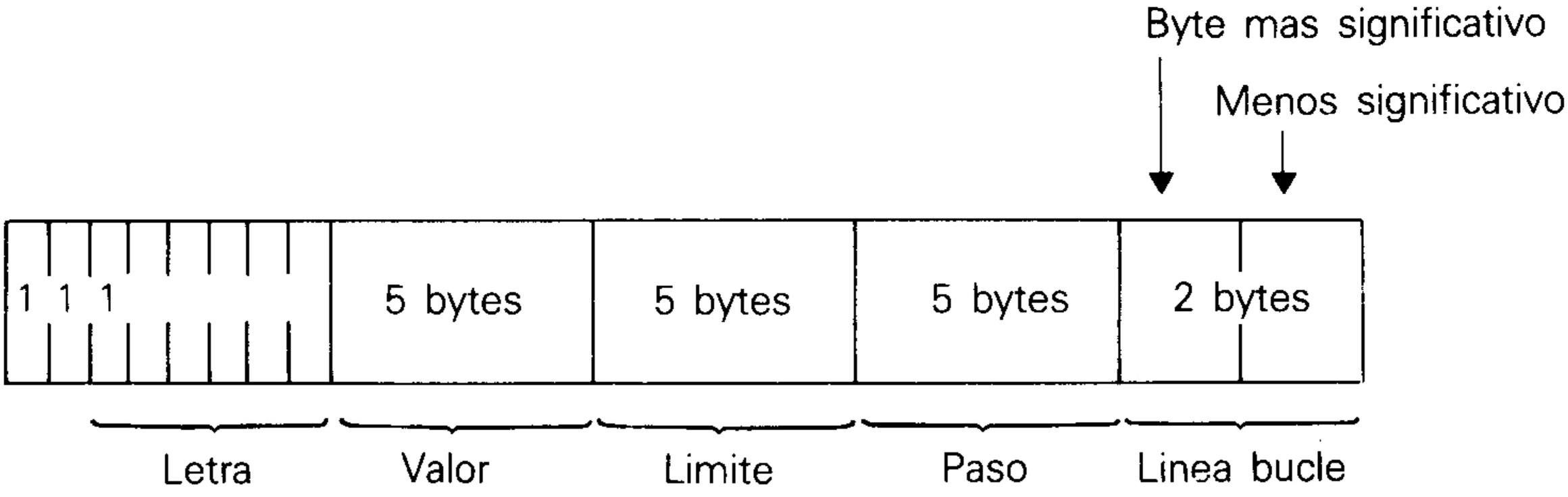
El orden de los elementos es:
primero, los elementos cuyo primer subíndice es 1
despues, los elementos cuyo primer subíndice es 2
despues, los elementos cuyo primer subíndice es 3
y asi sucesivamente, hasta agotar todos los valores posibles del primer subíndice.

Los elementos con un determinado primer subíndice se ordenan de la misma manera utilizando el segundo subíndice y así sucesivamente hasta llegar al último.

Como ejemplo, los elementos del conjunto B de 3 × 6 del capítulo 22 están almacendos en el siguiente orden

B(1,1), B(1,2), B(1,3), B(1,4), B(1,5), B(1,6), B(2,1), B(2,2), . . ., B(2,6), B(3,1), B(3,2), . . ., B(3,6).

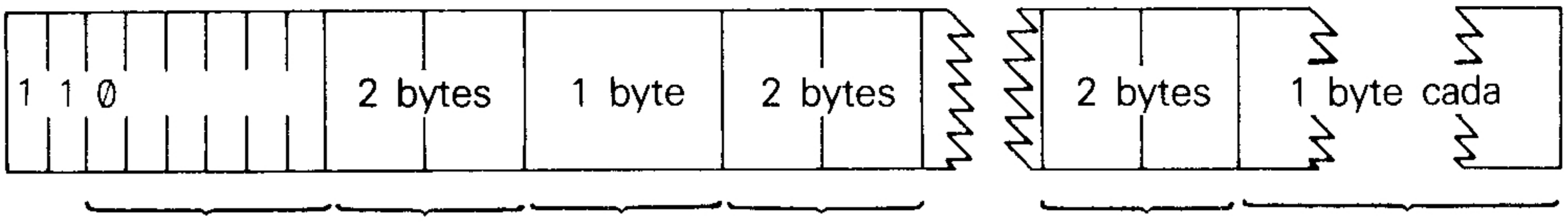
Variable de control de un bucle **FOR-NEXT:**



Cadena:

Letra-20h | Cantidad de Caracteres | Texto de cadena (puede estar vacia)

Conjunto de caracteres:



Letra-20h | Cantidad total de elementos y dims, + 1 para nº de dimensiones | Nº de dims. | 1ª dim. | Ultima dim. | Elementos

La parte que empieza en E_LINE contiene la línea que se está escribiendo (como una orden, una línea de programa, o datos **INPUT**) y además algún espacio para trabajar.

El calculador es la parte del sistema BASIC relacionado con la aritmético, y los números que se están manejando en ella se guardan en su mayor parte en el compartimento del calculador.

La parte de reserva contiene el espacio que no se ha utilizado por el momento.

El compartimento de máquina es el compartimento utilizado por la pastilla Z80 para guardar las direcciones de retorno y otras cosas.

El compartimento **GOSUB** se describió en el capítulo 14.

El espacio para rutinas **USR** tiene que ser anulado por Ud., empleando **NEW** como se describió en el último capítulo.

CAPITULO 28

Las variables del sistema

Los bytes en memoria desde el 16384 hasta el 16508 están reservados para usos específicos por el sistema. Puede hecharles un vistazo para descubrir varias cosas sobre el sistema, y puede ser útil introducirse en algunos de ellos. Aquí estan todos junto con su empleo.

Son las llamadas variables del sistema, y tienen nombres, pero no debe confundirlas con las variables empleadas por el BASIC. El computador no reconocerá los nombres como variables del sistema ya se han dado solamente a efectos nemotécnicos.

Las abreviaturas de la columna 1 tienen el siguiente significado:

X No debe trabajar con esta variable porque el sistema se puede caer
N El trabajo con la variable no tendrá efectos permanentes.
S La variable es conservada por **SAVE.**

El número de la columna 1 es el número de bytes de la variable. Para dos bytes, el primero de ellos es el menos significativo, al revés de lo que Ud. podría esperar. Así para introducir un valor v en una variable de dos bytes en la dirección n, emplee

POKE n, v-256***INT** (v/256)
POKE n + 1, **INT** (v/256)

y para ver éste valor, utilice la expresion

PEEK n + 256***PEEK** (n + 1)

Notas	Direccion	Nombre	Contenido
1	16384	ERR NR	1 menos que el código registrado. Empieza en 255 (por –1), así **PEEK** 16384, si funciona bien, da 255. **POKE** 16384, n puede utilizarse para forzar una detención por error: 0≤n≤14 da uno de los informes habituales, 15≤n≤34 o 99≤n≤127 da un informe no normalizado, y 35≤n≤98 está expuesto a desordenar el archivo de imagen.
X1	16385	FLAGS	Varias banderas para controlar el sistema BASIC.
X2	16386	ERR_SP	Dirección de la primera partida en el compartimento de máquina (despues de **GOSUB** retornos)

Notas	*Dirección*	*Nombre*	*Contenido*
2	16388	RAMTOP	Dirección del primer byte fuera del área del sistema BASIC. Ud. se puede introducir en ella para hacer que **NEW** reserve espacio fuera del área (ver capítulo 26) o para engañar a **CLS** y que establezca un archivo de imagen mínimo (Capítulo 27). La introducción en RAMTOP no tiene efecto hasta que se ejecuta una de éstas dos.
N1	16390	MODE	El cursor K, K, F o G especificado
N2	16391	PPC	El número de línea de la sentencia que se está ejecutando. Trabajar en ella no tiene efectos permanentes excepto en la última línea del programa.
S1	16393	VERSN	0 identifica al ZX81 BASIC en los programas grabados.
S2	16394	E_PPC	Número de la línea actual (con cursor de programa).
SX2	16396	D_FILE	Ver capitulo 27.
S2	16398	DF_CC	Dirección de la posición de **PRINT** en archivo de imagen. Se puede tocar ya que la salida **PRINT** es enviada a otra parte.
SX2	16400	VARS	Ver capítulo 27.
SN2	16402	DEST	Dirección de variable en asignación.
SX2	16404	E_LINE	Ver capítulo 27
SX2	16406	CH_ADD	Dirección del próximo caracter a definir: el caracter posterior al argumento de **PEEK,** o el **NEWLINE** al final de una sentencia **POKE.**
S2	16408	X_PTR	Dirección del caracter que precede al indicador **S** .
SX2	16410	STKBOT	Ver capítulo 27.
SX2	16412	STKEND	
SN1	16414	BERG	Registro b del calculador.
SN2	16415	MEM	Dirección del área utilizada para memoria del calculador. (Generalmente MEMBOT, pero no siempre).
S1	16417	no utilizada	
SX1	16418	DF_SZ	El número de líneas (incluyendo una línea en blanco) en la parte inferior de la pantalla.

Notas	*Dirección*	*Nombre*	*Contenido*
S2	16419	S__TOP	El número de la línea superior del programa en listados automáticos.
SN2	16421	LAST__K	Indica las teclas pulsadas.
SN1	16423		Pone de manifiesto la situación del teclado
SN1	16424	MARGIN	Número de líneas en blanco por encima o por debajo de la imagen: 55 en Europa, 31 en America.
SX2	16425	NXTLIN	Dirección de la próxima línea a ejecutar.
S2	16427	OLDPPC	Número de línea de la que salta **CONT.**
SN1	16429	FLAGX	Varias banderas.
SN2	16430	STRLEN	Longitud de cadena; designación de tipo en asignación.
SN2	16432	T__ADDR	Dirección de la próxima partida en la tabla de sintaxis (muy improbable que le sirva para algo).
S2	16434	SEED	El origen para **RND.** Es la variable que es puesta en orden por **RAND.**
S2	16436	FRAMES	Cuenta los cuadros pasados en el televisor. El bit 15 es 1. Los bits 0 a 14 van disminuyendo por cada cuadro enviado al televisor. Se puede utilizar esto como temporización, pero **PAUSE** también hace uso de ello. **PAUSE** pone a 0 el bit 15, e introduce en los bits 0 a 14 la duración de la pausa. Cuando han sido contados hasta cero, la pausa termina. Si la pausa termina porque se ha pulsa do una tecla, el bit 15 pasa nuevamente al 1.
S1	16438	COORDS	Coordenada x del último punto **PLOT**ted-representado.
S1	16439		Coordenada y del último punto **PLOT**ted-representado.
S1	16440	PR__CC	El byte de dirección menos significativo de la siguiente posición para **LPRINT** a imprimir en (en PRBUFF).
SX1	16441	S__POSN	Número de columna para posicionar **PRINT.**
SX1	16442		Números de línea para posicionar **PRINT**.
S1	16443	CDFLAG	Varias banderas. El bit 7 está en (1) durante el funcionamiento en cómputo y pantalla.
S33	16444	PRBUFF	Memoria intermedia del impresor (el 33° caracter es **NEWLINE**).

Notas	*Dirección*	*Nombre*	*Contenido*
SN3Ø	16477	MEMBOT	Area de memoria del calculador; utilizada para almacenar números que no se pueden colocar convenientemente en el compartimento del calculador.
S2	165Ø7		no utilizado.

Ejercicios

1. Plantee éste programa

```
1Ø FOR N = Ø TO 21
2Ø PRINT PEEK (PEEK 164ØØ + 256*PEEK 164Ø1 + N)
3Ø NEXT N
```

Le dará a Ud. los 22 primeros bytes del área de variables: intente casar la variable de control N con la descripción del capítulo 27.

2. En el programa anterior, cambie la línea 2Ø por

```
2Ø PRINT PEEK (165Ø9 + N)
```

Le dará a Ud. los 22 primeros bytes del área de programa. Encájelos con el programa en sí.

Código	*Caracter*	*Hex*	*Z80 assembler*	*después de CBh*	*después de EDh*
32	4	20	jr nz, DIS	sla b	
33	5	21	ld hl,NN	sla c	
34	6	22	ld (NN), hl	sla d	
35	7	23	inc hl	sla e	
36	8	24	inc h	sla h	
37	9	25	dec h	sla l	
38	A	26	ld, h,N	sla (hl)	
39	B	27	daa	sla a	
40	C	28	jr z,DIS	sra b	
41	D	29	add hl,hl	sra c	
42	E	2A	ld hl,((N)	sra d	
43	F	2B	dec hl	sra e	
44	G	2C	inc l	sra h	
45	H	2D	dec l	sra l	
46	I	2E	ld,l,N	sra (hl)	
47	J	2F	cpl	sra a	
48	K	30	jr nc,DIS		
49	L	31	ld sp.NN		
50	M	32	ld (NN), a		
51	N	33	inc sp		
52	O	34	inc (hl)		
53	P	35	dec (hl)		
54	Q	36	ld (hl),N		
55	R	37	sfc		
56	S	38	jr c,DIS	srl b	
57	T	39	add hl,sp	srl c	
58	U	3A	ld a,(NN)	srl d	
59	V	3B	dec sp	srl e	
60	W	3C	inc a	srl h	
61	X	3D	dec a	srl l	
62	Y	3E	ld a,N	srl (hl)	
63	Z	3F	ccf	srl a	
64	**RND**	40	ld b,b	bit 0,b	in b,(c)
65	**INKEY$**	41	ld b,c	bit o,c	out (c),b
66	**PI**	42	ld b,d	bit 0, d	sbc hl,bc
67		43	ld b,e	bit 0,e	ld (NN), bc
68		44	ld b,h	bit 0,h	neg
69	sin uso	45	ld b,l	bit 0,l	retn
70		46	ld b,(hl)	bit 0,(hl)	im 0
71		47	ld b,a	bit 0,a	ld i,a
72		48	ld c,b	bit 1,b	in c,(c)

Código	*Caracter*	*Hex*	*Z80 assembler*	*después de CBh*	*después de EDh*
73		49	ld c,c	bit 1,c	out (c),c
74		4A	ld c,d	bit 1,d	adc hl,bc
75		4B	ld c,e	bit 1,e	ld bc,(NN)
76		4C	ld c,h	bit 1,h	
77		4D	ld c,l	bit 1,l	reti
78		4E	ld c,(hl)	bit 1,(hl)	
79		4F	ld c,a	bit 1,a	
80		50	ld d,b	bit 2,b	in d,(c)
81		51	ld d,c	bit 2,c	out (c),d
82		52	ld d,d	bit 2,d	sbc hl, de
83		53	ld d,e	bit 2,e	ld (NN),de
84		54	ld d,h	bit 2,h	
85		55	ld d,l	bit 2,l	
86		56	ld d,(hl)	bit 2,(hl)	im 1
87		57	ld d,a	bit 2,a	ld a,i
88		58	ld e,b	bit 3,b	in e,(c)
89		59	ld e,c	bit 3,c	out (c),e
90		5A	ld e,d	bit 3,d	adc hl,de
91		5B	kd e,e	bit 3,e	ld de,(NN)
92	sin uso	5C	ld e,h	bit 3,h	
93		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	
96		60	ld h,b	bit 4,b	in h,(c)
97		61	ld h,c	bit 4,c	out (c),h
98		62	ld h,d	bit 4,d	sbc hl,hl
99		63	ld h,e	bit 4,e	ld (NN),hl
100		64	ld h,h	bit 4,h	
101		65	ld h,l	bit 4,l	
102		66	ld h,(hl)	bit 4,(hl)	
103		67	ld h,a	bit 4,a	rrd
104		68	ld l,b	bit 5,b	in l,(c)
105		69	ld l,c	bit 5,c	out (c),l
106		6A	ld l,d	bit 5,d	adc hl,hl
107		6B	ld l,e	bit 5,e	ld de,(NN)
108		6C	ld l,h	bit 5,h	
109		6D	ld l,l	bit 5,l	
110		6E	ld l,(hl)	bit 5,(hl)	
111		6F	ld l,a	bit 5,a	rld
112	cursor arriba ⌔	70	ld (hl),b	bit 6,b	
113	cursor abajo ⌔	71	ld (hl),c	bit 6,c	

Código	*Caracter*	*Hex*	*Z80 assembler*	*después de CBh*	*después de EDh*
114	cursor izqda. ◊	72	ld (hl),d	bit 6,d	sbc hl,sp
115	cursor drcha. ◊	73	ld (hl),e	bit 6,e	ld (NN),sp
116	**GRAPHICS**	74	ld (hl),h	bit 6,h	
117	**EDIT**	75	ld (hl),l	bit 6,l	
118	**NEWLINE**	76	halt	bit 6,(hl)	
119	**RUBOUT**	77	ld (hl),a	bit 6,a	
120	**K** / **L** mode	78	ld a,b	bit 7,b	in a,(c)
121	**FUNCTION**	79	ld a,c	bit 7,c	out (c),a
122	sin uso	7A	ld a,d	bit 7,d	adc hl,sp
123	sin uso	7B	ld a,e	bit 7,e	ld sp,(NN)
124	sin uso	7C	ld a,h	bit 7,h	
125	sin uso	7D	ld a,l	bit 7,l	
126	número	7E	ld a,(hl)	bit 7,(hl)	
127	cursor	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	inverso"	8B	adc a,e	res 1,e	
140	inverso £	8C	adc a,h	res 1,h	
141	inverso $	8D	adc a,l	res 1,l	
142	inverso :	8E	adc a,(hl)	res 1,(hl)	
143	inverso ?	8F	adc a,a	res 1,a	
144	inverso (	90	sub b	res 2,b	
145	inverso)	91	sub c	res 2,c	
146	inverso >	92	sub d	res 2,d	
147	inverso <	93	sub e	res 2,e	
148	inverso =	94	sub h	res 2,h	
149	inverso +	95	sub l	res 2,1	
150	inverso –	96	sub (hl)	res 2,(hl)	
151	inverso *	97	sub a	res 2,a	
152	inverso /	98	sbc a,b	res 3,b	
153	inverso ;	99	sbc a,c	res 3,c	
154	inverso ,	9A	sbc a,d	res 3,d	

Código	*Caracter*	*Hex*	*Z80 assembler*	*después de CBh*	*después de EDh*
155	inverso .	9B	sbc a,e	res 3,e	
156	inverso 0	9C	sbc a,h	res 3,h	
157	inverso 1	9D	sbc a,l	res 3,l	
158	inverso 2	9E	sbc a,(hl)	res 3,(hl)	
159	inverso 3	9F	sbc a,a	res 3,a	
160	inverso 4	A0	and b	res 4,b	ldi
161	inverso 5	A1	and c	res 4,c	cpi
162	inverso 6	A2	and d	res 4,d	ini
163	inverso 7	A3	and e	res 4,e	outi
164	inverso 8	A4	and h	res 4,h	
165	inverso 9	A5	and l	res 4,1	
166	inverso A	A6	and (hl)	res 4, (hl)	
167	inverso B	A7	and a	res 4,a	
168	inverso C	A8	xor b	res 5,b	ldd
169	inverso D	A9	xor c	res 5,c	cpd
170	inverso E	AA	xor d	res 5,d	ind
171	inverso F	AB	xor e	res 5,e	outd
172	inverso G	AC	xor h	res 5,h	
173	inverso H	AD	xor l	res 5,l	
174	inverso I	AE	xor (hl)	res 5,(hl)	
175	inverso J	AF	xor a	res 5,a	
176	inverso K	B0	or b	res 6,b	ldir
177	inverso L	B1	or c	res 6,c	cpir
178	inverso M	B2	or d	res 6,d	inir
179	inverso N	B3	or e	res 6,e	otir
180	inverso O	B4	or h	res 6,h	
181	inverso P	B5	or l	res 6,l	
182	inverso Q	B6	or (hl)	res 6,(hl)	
183	inverso R	B7	or a	res 6,a	
184	inverso S	B8	cp b	res 7,b	lddr
185	inverso T	B9	cp c	res 7,c	cpdr
186	inverso U	BA	cp d	res 7,d	indr
187	inverso V	BB	cp e	res 7,e	otdr
188	inverso W	BC	cp h	res 7,h	
189	inverso X	BD	cp l	res 7,l	
190	inverso Y	BE	cp (hl)	res 7, (hl)	
191	inverso Z	BF	cp a	res 7,a	
192	" "	C0	ret nz	set 0,b	
193	**AT**	C1	pop bc	set 0,c	
194	**TAB**	C2	jp nz,NN	set 0,d	
195	sin uso	C3	jp NN	set 0,o	

Código	*Caracter*	*Hex*	*Z80 assembler*	*después de CBh*
196	**CODE**	C4	call nz,NN	set 0,h
197	**VAL**	C5	push bc	set 0,l
198	**LEN**	C6	add a,N	set 0,(hl)
199	**SIN**	C7	rst 0	set 0,a
200	**COS**	C8	ret z	set 1,b
201	**TAN**	C9	ret	set 1,c
202	**ASN**	CA	jp z,NN	set l,d
203	**ACS**	CB		set l,e
204	**ATN**	CC	call z,NN	set 1,h
205	**LN**	CD	call NN	set 1,l
206	**EXP**	CE	adc a,N	set 1,(hl)
207	**INT**	CF	rst 8	set 1,a
208	**SQR**	D0	ret nc	set 2,b
209	**SGN**	D1	pop de	set 2,c
210	**ABS**	D2	jp nc,NN	set 2,d
211	**PEEK**	D3	out N,a	set 2,e
212	**USR**	D4	call nc,NN	set 2,h
213	**STR$**	D5	push de	set 2,l
214	**CHR$**	D6	sub N	set 2,(hl)
215	**NOT**	D7	rst 16	set 2,a
216	**	D8	ret c	set 3,b
217	**OR**	D9	exx	set 3,c
218	**AND**	DA	jp c,NN	set 3,d
219	<=	DB	in a,N	set 3,e
220	>=	DC	call c,NN	set 3,h
221	<>	DD	prefija instrucciones usand ix.	set 3,1
222	**THEN**	DE	sbc a,N	set 3,(hl)
223	**TO**	DF	rst 24	set 3,a
224	**STEP**	E0	ret po	set 4,b
225	**LPRINT**	E1	pop hl	set 4,c
226	**LLIST**	E2	jp po,NN	set 4,d
227	**STOP**	E3	ex (spl),hl	set 4,e
228	**SLOW**	E4	call po,NN	set 4,h
229	**FAST**	E5	push hl	set 4,l
230	**NEW**	E6	and N	set 4,(hl)
231	**SCROLL**	E7	rest 32	set 4,a
232	**CONT**	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

Código	*Caracter*	*Hex*	*Z80 assembler*	*después de CBh*
236	**GOTO**	EC	call pe,NN	set 5,h
237	**GOSUB**	ED		set 5,l
238	**INPUT**	EE	xor N	set 5,(hl)
239	**LOAD**	EF	rst 40	sset 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	**RAND**	F9	ld sp,hl	set 7,c
250	**IF**	FA	jp m,NN	set 7,d
251	**CLS**	FB	ei	set 7,e
252	**UNPLOT**	FC	call m,NN	set 7,h
253	**CLEAR**	FD	prefixa instrucciones usando iy	set 7,l
254	**RETURN**	FF	cp N	set 7,(hl)
255	**COPY**	FF	rst 56	set 7,a

Códigos de informacion

La tabla siguiente recoge los códigos de informacion junto con una descripción general y una relación de situaciones en que pueden aparecer. En el apéndice C, en cada sentencia o función se da una descripción mas detallada de lo que significan los mensajes de error.

Código	*Significado*	*Situación*
Ø	Ejecución acertada, o salto a la línea de número mas alto que haya. Un informe con código Ø, no cambia el número de línea usado por **CONT.**	Cualquiera
1	No existe la variable de control (no ha sido establecida por una sentencia **FOR**) pero hay una variable ordinaria con el mismo nombre.	**NEXT**
2	Se ha utilizado una variable no definida. Con una variable simple, se presenta si se usa la variable antes de haber sido asignada a una sentencia **LET** Con una variable con subíndice, se presenta si se usa la variable antes de haber sido dimensionada en una sentencia **DIM**. Con una variable de control, se presenta si se usa la variable antes de que se haya establecido como variable de control en una sentencia FOR, si no hay una variable ordinaria simple con el mismo nombre.	Cualquiera
3	Subíndice fuera del grupo. Si el subíndice está definitivamente fuera del grupo (negativo o mayor que 65535) dará un error B.	Variables con subíndice.
4	No hay suficiente sitio en la memoria. Tenga en cuenta que el numero de línea en el informe (despues de /) puede estar incompleto en la pantalla, por falta de memoria: p.e., 4/Ø puede aparecer como 4/2. Ver capítulo 23. Para **GOSUB** ver ejercicio 6 del capítulo 14.	**LET, INPUT, DIM, PRINT, LIST, PLOT, UNPLOT, FOR, GOSUB.** Algunas veces durante el cálculo de una función.
5	No hay mas sitio en la pantalla. **CONT** hará sitio por borrado de la pantalla.	**PRINT, LIST.**
6	Overflow aritmético: los cálculos han llegado a un número mas grande de $1Ø^{38}$.	Cualquier cálculo.
7	No hay **GOSUB** para una sentencia **RETURN.**	**RETURN**
8	Ud. ha intentado **INPUT** como una orden (no permitido).	**INPUT**

Código	*Significado*	*Situaciones*
9	Sentencia **STOP** ejecutada. **CONT** no intentará volver a ejecutar la sentencia **STOP.**	**STOP**
A	Argumento no válida para ciertas funciones.	**SQR, LN, ASN, ACS**
B	Número entero fuera de escala. Cuando se pide un número entero, el argumento con coma flotante se redondea al entero mas próximo. Si éste se sale de una escala adecuada, surge el error B.	**RUN, RAND, POKE, DIM, GOTO, GOSUB, LIST, LLIST, PAUSE, PLOT, UNPLOT, CHR$, PEEK, USR**
	Para acceso a conjuntos, ver también el informe 3.	Acceso a conjuntos.
C	El texto del argumento (cadena) de **VAL** no forma una expresión numérica válida.	**VAL**
D	(i) Programa interrumpido por **BREAK.**	Al final de cualquier sentencia, o en **LOAD, SAVE, LPRINT, LLIST** o **COPY**
	(ii) La línea **INPUT** empieza con **STOP**.	**INPUT**
E	Sin uso.	
F	El nombre dado al programa es la cadena vacía.	**SAVE**

El ZX81 para aquellos que entienden BASIC

Generalidades

Si Ud. ya sabe BASIC no tendrá muchos problemas para manejar el ZX81; pero hay una o dos peculiaridades en éste sistema.

(i) Las palabras no están deletreadas, sino que tienen teclas propias, como se trató en el capítulo 2 (para palabras-clave y teclas cambiadas) y en el capítulo 5 (para nombres de funciones). En el texto, éstas palabras están impresas en **NEGRITA.**

(ii) El ZX81 carece de READ, DATA y RESTORE (pero vea el ejercicio 3 del capítulo 22 relativo a esto), de funciones definidas por el usuario (FN y DEF; en su lugar puede usar algunas veces **VAL**), y líneas de sentencias múltiples.

(iii) Las facilidades para el manejo de cadenas son amplias pero no estan normalizadas, ver el capítulo 21, y también el 22 (para conjuntos de cadenas).

(iv) El juego de caracteres del ZX81 es completamente particular.

(v) En general la imagen del televisor no está planificada en la memoria.

(vi) Si está acostumbrado a utilizar **PEEK** y **POKE** en otra máquina, recuerde que en el ZX81 todas las direcciones serán distintas.

Velocidad

La máquina trabaja a dos velocidades, llamadas funcionamiento en cómputo e imagen, y funcionamiento rápido.

En cómputo e imagen, la imagen del televisor se produce de manera contínua y el cómputo se efectúa durante las partes en blanco de la parte de arriba y de abajo.

En funcionamiento rápido, la imagen del televisor se corta, y solamente aparece al final del programa, mientras está esperando por datos **INPUT,** o durante una pausa (ver **PAUSE**).

El funcionamiento rápido tiene lugar a una velocidad cuatro veces mayor de lo normal, y debería emplearse en programas con mucho cómputo y poca salida, o cuando escriba programas largos.

La conmutación entre velocidades se efectúa con las sentencias **FAST** y **SLOW** (que ya vimos).

El teclado

Los caracteres del ZX81 incluyen no solo símbolos sencillos (letras, números, etc.) sino también señales compuestas (palabras-clave, nombres de funciones, etc.; aquí están impresas en **NEGRITA**) y todas se introducen el teclado sin necesidad de deletrearlas. Para conseguirlo, algunas teclas tienen hasta cinco significados distintos, dados algunos por cambio de las teclas (es decir, pulsando la tecla **SHIFT** al mismo tiempo que se pulsa la otra tecla) y otros por tener a la máquina en diferentes formas de trabajo.

La forma de trabajo se indica por el cursor, una letra en imagen negativa que indica donde se introducirá el siguiente caracter procedente del teclado.

La forma **K** (para palabras-clave) aparece automáticamente cuando la máquina está a la espera de una orden o línea de programa (en vez de datos **INPUT**), y de su posición en la línea sabe si debe esperar un número de línea o una palabra-clave. Está al principio de la

línea, o justamente después de algunos dígitos al principio de la línea, o justo después de **THEN.** Si no está cambiada, la proxima tecla se interpretará o como una palabra-clave (la mayoría están escritas encima de las teclas), o un dígito.

La forma **L** (para letras) normalmente aparece en todos los demás casos. Si no está cambiada, la siguiente tecla se interpretará como el símbolo principal de esa tecla.

En ambas formas **K** y **L**, las teclas cambiadas se interpretaran como el correspondiente caracter en rojo que hay en la esquina superior derecha de la tecla.
La forma **F** (para funciones) aparece después que se pulsa **FUNCION** (**NEWLINE** cambiada), y permanece solamente mientras se pulsa una tecla. Esa tecla se interpretará como un nombre de función, que aparecen debajo de las teclas.
La forma **G** (por gráficos) aparece después que se ha pulsado **GRAPHICS** (9 cambiada), y permanece hasta que se pulsa nuevamente. Una tecla no cambiada dará la imagen en negativo de su interpretación en forma **L** ; una tecla cambiada hará lo mismo, siempre que sea un simbolo, pero si la tecla cambiada daría en condiciones normales una señal, en forma gráfica proporciona el símbolo gráfico que aparece en la esquina inferior derecha de la tecla.

La pantalla

Tiene 24 líneas, cada una con 32 caracteres de longitud, y está dividida en dos partes. La parte superior tiene como máximo 22 líneas, y en ellas se puede representar o bien un listado o la salida de un programa. La parte inferior, de al menos dos líneas, se utiliza para la introducción de órdenes, líneas de programa y datos **INPUT,** y tambien para la presentación de infórmes.

Entrada por teclado: aparece en la mitad inferior de la pantalla segun se va escribiendo, cada caracter (símbolo sencillo o señal compuesta) se introduce justo antes del cursor. El cursor se puede mover hacia la izquierda con ◊ (5 cambiada) o hacia la derecha con ◊ (8 cambiada). El caracter anterior al cursor se puede borrar con **RUBOUT** (0 cambiada). Nota: se puede borrar toda la línea mediante la pulsación de **EDIT** (1 cambiada) seguida de **NEWLINE;** o, si son datos **INPUT,** solamente con pulsar **EDIT.**

Cuando se pulsa **NEWLINE,** se ejecuta la línea, se introduce en el programa, o se utiliza como datos **INPUT,** segun sea el caso, a menos que contenga un error de sintaxis, en cuyo caso aparece el símbolo **S** justamente delante del error.

Cuando se han introducido las líneas de programa, en la mitad superior de la pantalla aparece un listado. La manera en que se produce el listado es bastante complicada, y se explica con mas detalle en el ejercicio 6 del capítulo 9. La última línea a ser introducida se llama la línea actual y está señalada con el símbolo **>** , pero se puede cambiar empleando las teclas ◊ (6 cambiada) y ◊ (7 cambiada). Si se pulsa **EDIT** (1 cambiada), la línea actual se baja a la parte inferior de la pantalla y se puede modificar.

Cuando se ejecuta una orden o un programa, al principio la pantalla está totalmente en blanco y después la salida se representa en la mitad superior de la pantalla y permanece hasta que se introduce una línea de programa, se pulsa **NEWLINE** con una línea llena, o se

pulsa ⇦ o ⇨. En la parte inferior aparece un informe de la forma m/n en el que m es un código que indica lo que ha ocurrido (ver apéndice B), y n es el número de la última línea ejecutada, o 0 si fué una orden. El informe permanece hasta que se pulsa una tecla (e indica forma K).

En ciertas circunstancias la tecla **SPACE** actua como un **BREAK,** deteniendo el funcionamiento del computador con informe D. Esto se admite.

(i) al final de una sentencia mientras se está ejecutando un programa,

(ii) mientras el computador está esperando un programa grabado en cinta magnetofónica,

o (iii) mientras el computador está utilizando la impresora (o si por accidente se está tratando de utilizarla sin estar conectada).

El BASIC

Los números están almacenados con una precisión de 9 o 10 dígitos. El número mas grande que puede obtener aproximadamente es 10^{38}, y el número positivo mas pequeño es $4 * 10^{-39}$.

En el ZX81 un número se almacena en binario con coma flotante con un byte e de exponente ($1 < e < 255$), y cuatro bytes m de mantisa ($\frac{1}{2} < m < 1$). Esto representa al número $m * 2^{e-128}$.

Como $\frac{1}{2} < m < 1$, el bit mas significativo de la mantisa m es siempre 1. Por consiguiente en el estado actual podemos reemplazarlo por un bit que nos de el signo, 0 para números positivos, 1 para negativos.

El cero tiene una representación especial en la que los cinco bytes son 0.

Las variables numéricas tienen una longitude arbitraria, empezando con una letra y continuando con letras y núméros; todos ellos tienen valor, así por ejemplo LONGNAME y LONGNAMETOO son nombres distintos. Los espacios no se tienen en cuenta.

Las variables de control de bucles **FOR-NEXT** tienen nombres de una sola letra de longitud.

Los conjuntos numéricos tienen como nombre una sola letra, que puede ser la misma que el nombre de una variable simple. Pueden tener varias dimensions de cualquier tamaño. Los subíndíces empiezan en 1.

Las cadenas son completamente flexibles en su longitud. El nombre de una cadena está formado por una sola letra seguida de $.

Los conjuntos de cadenas pueden tener varias dimensiones de cualquier tamaño. El nombre es una sola letra seguida de $ y puede no ser la misma que el nombre de una cadena. Todas las cadenas de un determinado conjunto tienen la misma longitud fija, que se especifica en una dimensión final mas en la sentencia **DIM.** Los subíndices empiezan en 1.

Slicing-troceado: Subcadenas de cadenas que se pueden especificar usando los siguientes formatos.

El formato puede ser

(i) vacio

o

(ii) expresión numérica

o

(iii) expresión numérica opcional **TO** expresión numérica opcional y se utiliza para expresar una subcadena mediante

(a) una expresión de cadena (formato)

o mediante

(b) una variable de conjunto de cadenas (subíndice, . . ., subíndice, formato)

lo que quiere decir lo mismo que

variable de conjunto de cadenas (subíndice, . . ., subíndice) (formato).

En (a), suponga que la expresión de cadena vale S$.

Si el formato está vacío, el resultado es S$ considerada como una subcadena en sí misma.

Si el formato es una expresión numérica de valor m, el resultado es el caracter m-simo de S$ (una subcadena de longitud 1).

Si el formato es el (iii), suponga que la primera expresión numérica tiene el valor m (el valor por defecto es 1), y la segunda, n (el valor por defecto es la longitud de S$).

Si $1 \leqslant m \leqslant n \leqslant$ la longitud de S$, el resultado es la subcadena de S$ que empieza con el caracter M-simo y termina con el n-simo.

Si $0 \leqslant n \leqslant m$, el resultado es la cadena vacía.

Cualquier otra cosa, da error 3 como resultado.

El troceado se realiza antes de que se calculen las funciones u operaciones, a menos que haya parentesis que obliguen a otra cosa.

Las cadenas pueden ser asignadas (ver **LET**).

El argumento de una función no necesita parentesis si es una constante o una variable (posiblemente con subíndices o troceada).

Función	**Tipo de operando (x)**	**Resultado**
ABS	número	Magnitud absoluta
ACS	número	Arco coseno en radianes Error A si x no está entre –1 y +1
AND	operación binaria, operando derecho siempre un número.	
	Operando izq. número:	$A\ \textbf{AND}\ B = \begin{cases} A \text{ si } B \neq 0 \\ 0 \text{ si } B = 0 \end{cases}$
	Operando izq. cadena:	$A\$\ \textbf{AND}\ B = \begin{cases} A\$ \text{ si } B \neq 0 \\ \text{'''' si } B = 0 \end{cases}$
ASN	número	Arco seno en radianes Error A si x no está entre –1 y +1

Función	Tipo de operando	Resultado
ATN	número	Arco tangente en radianes
CHR$	número	El caracter cuyo código es x, redondeado al entero mas próximo. Error B si x no está entre 0 y 255.
CODE	cadena	El código del primer caracter en x (o 0 si x es la cadena vacía).
COS	número (en radianes)	Coseno
EXP	número	e^x
INKEY$	ninguno	Lee el teclado. El resultado es el caracter que representa (en forma L) la tecla pulsada si hay alguna, en caso contrario la cadena vacía.
INT	número	Parte entera (siempre redondeada por defecto)
LEN	cadena	Longitud
LN	número	Logaritmo natural. (En base e). Error A si $x \leq 0$
NOT	número	0 si $x \neq 0$, 1 si $x = 0$ **NOT** tiene prioridad 4
OR	operación binaria, ambos operandos números	A **OR** B = { 1 si B ≠ 0 ; A si B = 0 **OR** tiene prioridad 2
PEEK	número	El valor del byte en memoria cuya dirección es x (redondeado al entero mas próximo). Error B si x no está entre 0 y 65535.
PI	ninguno	π (3.14159265 . . .)
RND	ninguno	El número "y" seudoaleatorio mas próximo en una serie originada tomando las potencias de 75 de módulo 65537, restándoles 1 y diviendo por 65536. $0 \leq y < 1$.
SGN	número	Signo: el signo de x (–1, 0 o +1)
SIN	número (en radianes)	Seno
SQR	número	Raíz cuadrada. Error B si x<0
STR$	número	La cadena de caracteres que aparecería si se imprimiera (printed) x.
TAN	número (en radianes)	Tangente
USR	número	Llama a la subrutina en código de máquina cuya dirección de comienzo es x (redondeado al entero mas próximo). En retorno, el resultado es el contenido del par de registros bc. Error B si x no está entre 0 y 65535.

Función	Tipo de operando	Resultado
VAL	cadena	Valora x (sin sus comillas de limitación) como una expresión numérica. Error C si x contiene error de sintaxis, o da un valor de cadena. Son posibles otros errores dependiendo de la expresión.
–	número	Negación

Son operaciones binarias:

+	Suma (de números), o concatenación (de cadenas)	
–	Sustracción	
*	Multiplicación	
/	División	
**	Potencias. Error B si el operando izquierdo en negativo.	
=	Igual a	Ambos operandos deben ser del mismo tipo. El resultado es un número, 1 si la comparación es válida, Ø si no los es.
>	Mayor que	
<	Menor que	
< =	Menor o igual que	
> =	Mayor o igual que	
<>	Distinto de	

Las funciones y operaciones tienen las siguientes prioridades:

Operación	Prioridad
Poner subíndices y trocear	12
Todas las funciones excepto **NOT** y menos unario	11
**	10
Menos unario	9
*, /	8
+, – (– binario)	6
=, >, <, < =, > =, <>	5
NOT	4
AND	3
OR	2

Sentencias

En la siguiente lista,

∝	representa una sola letra
v	representa una variable
x, y, z,	representan expresiones numéricas

m, n	representan expresiones numéricas redondeadas al entero mas próximo
e	representa una expresión
f	representa una expresión valorado como cadena
s	representa una sentencia

Observe que en todos los sitios se admiten expresiones arbitrarias (excepto para el número de línea al principio de una sentencia).

Todas las sentencias, excepto **INPUT**, se pueden usar como órdenes o en programas (aunque pueden ser mas sensibles en uno que en otro caso).

CLEAR
Borra todas las variables, liberando el espacio que ocupan.

CLS
(Clear Screen) Limpia el archivo de imagen. Ver en el capítulo 27 lo relativo al archivo de imagen.

CONT
Suponga que p/q fué el último informe con un no-cero. Entonces **CONT** tiene el efecto
GOTO q si p ≠ 9
GOTO q + 1 si p = 9 (sentencia **STOP**)

COPY
Envía una copia de lo representado en pantalla a la impresora si está conectada, y si no lo está no hace nada.
A diferencia de otras órdenes, una orden **COPY** no borra en primer lugar la pantalla. Después de **COPY** no debe haber espacios. Informe D si **BREAK** está pulsada.

DIM ∝ (n_1, . . ., n_k)
Suprime todo conjunto con nombre ∝, y establece un conjunto ∝ de números con k dimensiones n_1, . . ., n_k. Inicia todos los valores en 0
Aparece el error 4 si no existe sitio para establecer el conjunto. Un conjunto está indefinido hasta que se fijan sus dimesiones en una sentencia **DIM.**

DIM ∝$ (n_1, . . ., n_k)
Suprime todo conjunto o cadena con nombre ∝$, y establece un conjunto ∝$ de caracteres con k dimensiones n_1, . . ., n_k. Inicia todos los valores en " ". Esto se puede considerar como un conjunto de cadenas de longitud fija n_k, con k – 1 dimensiones n_1, . . ., n_{k-1}.
Aparece el error 4 si no existe sitio para establecer el conjunto. Un conjunto está indefinido hasta que se fijan sus dimensiones en una sentencia **DIM.**

FAST
Pone en marcha la forma rápida, en la cual el archivo de imagen es puesto en pantalla solamente al final del programa, mientras están siendo escritos los datos **INPUT**, o durante una pausa.

FOR ∝ = x **TO** y
FOR ∝ = x **TO** y **STEP** 1

FOR ∝ = x **TO** y **STEP** z
Suprime cualquier variable simple ∝, y esblece una variable de control con valor x, limite y, incremento z y dirección de bucle 1 mas que el número de línea de la sentencia **FOR** (–1 si es una orden). Comprueba si el valor inicial es mayor (si el incremento ⩾ Ø) o menor (si el incremento < Ø) que el límite, y así salta a sentencia **NEXT** ∝ en el comienzo de una línea. Ver **NEXT** ∝.
Puede aparecer error 4 si no hay sitio para la variable de control.

GOSUB n
Registra el número de línea de la sentencia **GOSUB** en un compartimento; después actúa como **GOTO** n.
Puede aparecer error 4 si no hay suficientes **RETURNs**.

GOTO n
Salta a la línea n (o, si no existe, a la línea siguiente a esa).

IF x **THEN** s
Si x es verdad (no-cero) se ejecuta s. La forma "**IF** x **THEN** número de línea" no está permitida.

INPUT v
Se detiene y espera a que el usuario escriba una expresión cuyo valor asigna a v. En forma rápida, el archivo de imagen es representado en pantalla. **INPUT** no se puede usar como orden; si lo intenta le aparecerá error 8.
Si el primer caracter de la línea **INPUT** es **STOP,** el programa se detiene con informe D.

LET v = e
Asigna el valor e a la variable v.
LET no se puede omitir.
Una variable simple permanece no definida hasta que es asignada en una sentencia **LET** o **INPUT.**
Si v es una variable de cadena con subíndice, o una variable de cadena troceada (subcadena), entonces la asignación es Procrusteana: el valor de cadena de e o bien se recorta o bien se rellena con espacios a la derecha, para hacer que tenga la misma longitud que la variable v.

LIST
LIST Ø

LIST N
Hace el listado en el televisor, empezando en la línea n, y hace de n la línea actual. Si el listado es demasiado largo para entrar en la pantalla aparecerá error 4 ó 5; **CONT** hará exactamente lo mismo otra vez.

LLIST
LLIST Ø

LLIST n
Como **LIST,** pero usando la impresora en lugar del televisor.
Si la impresora no está conectada no deberá hacer nada.
Si se pulsa **BREAK** se detiene con informe D.

LOAD f
Busca en la cinta un programa llamado f y lo carga junto con sus variables.
Si f = "", carga el primer programa disponible.

Si está pulsada **BREAK** o se detecta un error en la cinta, entonces

(i) si todavía no se ha leido el programa de la cinta, se detiene con informe D y el programa anterior;

(ii) si ya se ha leido parte de un programa, entonces ejecuta **NEW.**

LPRINT . . . Como **PRINT,** pero usando la impresora en lugar del televisor. A la impresora se le envía una línea de texto

(i) cuando la impresión ocupa mas de una línea de texto,

(ii) después de una sentencia **LPRINT** que no termina en coma o en punto y coma,

(iii) cuando una coma o partida **TAB** pide una nueva línea, o

(iv) al final del programa, si queda algo por imprimir.

En una partida **AT**, solamente tiene algun efecto el número de columna; el número de línea es ignorado (excepto que surjan las mismas condiciones de error que cuando **PRINT** está fuera de intervalo). Una partida **AT** nunca envía una línea de texto a la impresora.

Si la impresora no está presente no se deberá producir nada.

Si **BREAK** está pulsada se detiene con informe D.

NEW Pone nuevamente en marcha el sistema BASIC, suprimiendo programa y variables, y limpia toda la memoria pero sin incluir el byte cuya dirección es RAMTOP en las variables del sistema (bytes 16388 y 16389).

NEXT

(i) Busca la variable de control ∝.

(ii) Añade su incremento a su valor.

(iii) Si el incremento ≥ 0 y el valor $>$ el límite; o
si el incremento < 0 y el valor $<$ el límite,
salta a la línea de bucle.

Si hay una variable simple ∝, error 1.

Si no hay variable de control o simple ∝, error 2.

PAUSE n Detiene el computo y presenta en pantalla el archivo de imagen durante n imágenes (a razón de 50 imagenes por segundo) o hasta que se pulse una tecla.

$0 \leq n \leq 65535$, si no error B. Si $n \geq 32767$ la pausa no está temporizada, y continua hasta que se pulse una tecla.

PLOT m, n Pone en negro el pixel (| m |, | n |); mueve la posición **PRINT** hasta colocarlo a continuación de ese pixel.

$0 \mid m \mid \leq 63$, $0 \mid n \mid \leq 43$, si no error B.

POKE m, n Escribe el valor n correspondiente al byte en memoria con dirección m.

$0 \leq m \leq 65535$, $-255 \leq n \leq 255$, si no error B.

PRINT . . .

Los '. . .' es una sucesión de partidas **PRINT** separadas por comas o puntos y comas, y están escritas en el archivo de imagen para su representación en el televisor. La posición (línea y columna) donde el siguiente caracter se imprimirá se llama la posición de **PRINT.**

Una partida de **PRINT** puede ser

(i) vacía, es decir nada

(ii) una expresión numérica.

En primer lugar, si el valor es negativo se imprime un signo menos. Ahora, sea x el módulo del valor.

Si $x \leqslant 10^{-5}$ o $x \geqslant 10^{13}$, en este caso se imprime utilizando notación científica.

La parte de mantisa tiene hasta ocho dígitos (sin ceros de cola), y la coma decimal (ausente si solo tiene un dígito) está después del primero. La parte exponente es E, seguida de + o –, seguida de uno o dos dígitos.

En todos los demas casos x se imprime en notación decimal ordinaria con hasta seis dígitos significativos, y sin ceros de colar después de la coma decimal. Una coma decimal justamente al principio se escribe siempre precedida por un cero, así por ejemplo .3 y 0.3 son lo mismo.

0 se escribe como un solo dígito 0.

(iii) una expresión de cadena.

Los símbolos en la cadena están ampliados, posiblemente con un espacio antes o después.

El caracter de imagen de comillas se escribe como ''.

Los caracteres no utilizados y los caracteres de control se escriben como ?.

(iv) **AT** m, n

La posición **PRINT** se cambia a la línea | m | (contando desde el principio), columna | n | (contando desde la izquierda).

$0 \leqslant |\, m \,| \leqslant 21$, si no error 5 si | m | = 22 ó 23, en todos los demas casos error B.

$0 \leqslant |\, n \,| \leqslant 31$, si no error B.

(v) **TAB** n

n es el módulo reducido de 32, y así, la posición de **PRINT** se mueve a la columna n, permaneciendo en la misma línea a menos que esto suponga retroceder, en cuyo caso va a la línea siguiente.

$0 \leqslant n \leqslant 255$, si no error B.

Un punto y coma entre dos partidas deja la posición de **PRINT** inalterada, de tal manera que la segunda partida va

inmediatamente después de la primera . Por lo contrario, una coma mueve la posición de **PRINT** por lo menos un lugar, y según esto, aunque muchas veces sea necesario dejarlo en la columna 0 ó 16, dirigiéndose a una nueva línea si es necesario.

Al final de una sentencia **PRINT,** si no termina en una coma o punto y coma, se inicia una nueva línea.

Con tres K o menos de memoria puede aparacer error 4 (fuera de memoria).

Error 5 significa que la pantalla está llena.

En ambos casos, la solución es **CONT,** que limpiará la pantalla y permitirá continuar.

RAND

RAND 0

RAND n

Establece la variable del sistema (llamada SEED) utilizada para generar el valor siguiente de **RND.** Si n = 0, entonces a SEED se la da el valor n; si n = 0 se le da el valor de otra variable del sistema (llamada FRAMES) que cuenta los imágenes que han pasado por el televisor, por lo que es practicamente aleatorio.

Si n no está entre 0 y 65535, aparece error B.

REM . . .

No produce ningún efecto. ". . ." puede ser cualquier secuencia de caracteres excepto **NEWLINE.**

RETURN

Saca el correspondiente número de línea del compartimento **GOSUB,** y salta a la línea siguiente a él.

Cuando no haya ningún número de línea en el compartimento aparecerá error 7. Hay algún error en su programa; los **GOSUBs** no están convenientemente equilibrados por **RETURNS.**

RUN

RUN 0

RUN n

CLEAR, y a continuación **GOTO** n.

SAVE f

Registra el programa y las variables en una cinta magnetofónica, y lo llama f. **SAVE** no se debe utilizar dentro de una rutina **GOSUB.**

Aparece error F si f es la cadena vacia, que no está permitida como nombre de programa.

SCROLL

Recorta una línea de la parte superior del archivo de imagen, haciendo desaparecer la línea superior y proporcionando una línea vacía al pie de la pantalla.

NB la nueva línea está autenticamente vacía con solamente un caracter **NEWLINE** y sin espacios. Ver capítulo 27.

SLOW

Pone al computador en forma de computo e imagen, en la que el archivo de imagen es puesto en pantalla de manera continua, y el computo se realiza durante los huecos en la

parte superior e inferior de la imagen.

STOP Detiene el programa con informe 9. **CONT** continuará con la línea siguiente.

UNPLOT m, n Como **PLOT,** pero pone en blanco un pixel en lugar de ennegrecerlo.

Indice

Este índice incluye las teclas del teclado y como conseguirlas (la forma de trabajo – **K**, **L**, **F** o **G** – y si cambiada o no), y sus códigos.

Generalemente, una entrada se menciona solo una vez por capítulo, por lo que habiéndola encontrado es conveniente que mire el resto del capítulo, incluyendo los ejercicios.

C

Printed at The Leagrave Press Limited, Luton & London

INVESTRONICA
Tomás Bretón
21. Madrid-7. Spain.
Tels. 468 01 00/468 03 00.
Télex: 23399 IYCO E.