49
185 pts.
IVA INCLUIDO

RUN

Enciclopedia Práctica del Spectrum

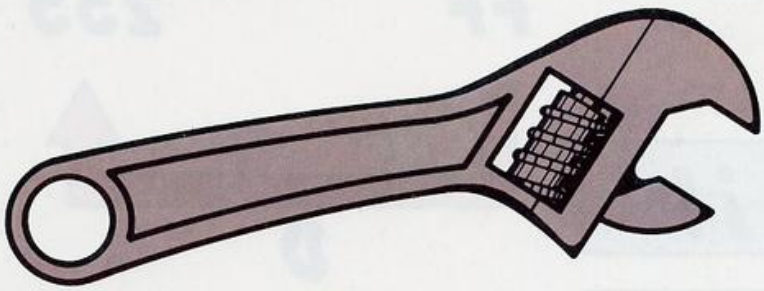
Nueva Lente/Ingelek

AYUDAS A LA PROGRAMACION

Dentro del vasto terreno del *software* podemos diferenciar varios campos en donde agrupar los programas. Uno de estos campos es el destinado a los programas de ayuda a la programación, o herramientas, que nos sirven para extraer mayor partido al ordenador y depurar nuestros programas.

En este breve muestrario de los ávidos programadores veremos, primeramente, un desensamblador, para obtener el listado en ensamblador y en código máquina de un programa en este lenguaje. El siguiente es un compilador, adecuado a los que desean velocidad en sus programas y no se atreven con el lenguaje máquina. ZXED es un programa que servirá de gran ayuda para los programadores al tener posibilidad de renumeración, saber la memoria libre... Para sacar más partido a su SPECTRUM está BETA-BASIC, que incorpora nuevos comandos, dando mayor potencia a la máquina. Por último, tenemos al «mini-lenguaje» DLAN que nos ofrecerá una sugerente presentación en la pantalla de nuestros textos.

Profundicemos ahora en estos programas que pueden ofrecer mucho en ayuda del programador asiduo.



Los programas herramientas, o de ayuda, sirven para depurar nuestros programas y sacar más partido a nuestro "micro".

EL DESENSAMBLADOR SPDE

Ya hemos hablado en esta misma sección de lo que es un programa ensamblador y su utilidad. Sin embargo, también disponemos de la herramienta de programación contraria: un programa desensamblador.

Este tipo de programas nos sirve para obtener el listado en lenguaje máquina y en ensamblador de un programa en código máquina. Así, los programas en este lenguaje dejan de ser una caja misteriosa al intentar descubrir cómo realizan sus funciones.

Para trabajar con SPDE debemos primeramente cargar el programa a desensamblar, y posteriormente el desensamblador en sí. Lo primero que nos aparece es el menú de opciones. Para ver las instrucciones (incluidas en el programa) basta pulsar I.

El comando **A** es el que nos ofrece el listado del programa en cuatro columnas. La primera de ellas es la dirección de memoria en donde se encuentra almacenando el comando, la segunda muestra el contenido de dicha dirección. Las otras dos se refieren al lenguaje ensamblador, hallándose en la tercera el nemónico (el comando) y en la cuarta los parámetros correspondientes a la tercera columna.

Cuando el listado no quepa en una pantalla (como ocurrirá la mayor parte de las veces) pode-

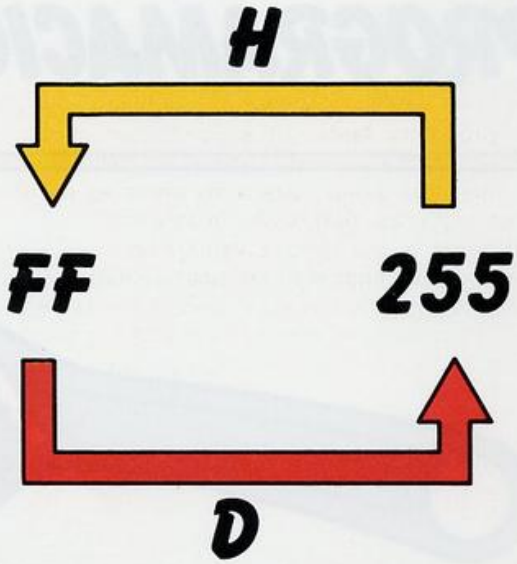
Con un desensamblador sabremos el listado de un programa en código máquina.

¡!

COMPILER no compila programas con números decimales, cadenas alfanuméricas ni tablas; otros comandos han sido transformados y las variables deben ir siempre en mayúsculas.

*

Un compilador es un programa que sirve para traducir nuestros programas BASIC a código máquina y obtener así una mayor velocidad de ejecución.



En DLAN, **D** *transforma un número decimal en hexadecimal y* **H** *nos hace lo contrario.*

mos irlo viendo si pulsamos las teclas del 1 al 9, apareciendo el número de líneas indicadas. Con **B** podremos ver el listado que se encuentra en las páginas anteriores, ya pasadas. Si pulsamos otras teclas diferentes avanzaremos, viendo nuevas páginas de listado.

Siempre que trabajamos en ensamblador o en lenguaje máquina tenemos necesidad de manejar números en hexadecimal obligándonos a realizar operaciones continuamente para hallar el equivalente. Con SPDE podemos efectuar la traducción directamente de hexadecimal a decimal y viceversa. **D**, tras introducir un número en decimal, nos devuelve su equivalente en hexadecimal, al contrario de lo que ocurre con **H**.

Si queremos introducir caracteres en una dirección de memoria pulsaremos **C** tras lo cual nos pedirá el texto que queremos incorporar. La misma función tiene **S**, con la única diferencia que no se introduce texto sino un número hexadecimal. La contraria a **C** es **F** que interpreta todas las órdenes como caracteres ASCII imprimiéndoles al lado.

Con **M** movemos bloques del programa al lugar deseado. Para ello SPDE nos pide qué dirección debemos copiar y la longitud, el número de líneas que queremos desplazar.

Una vez realizadas todas las operaciones deseadas, podemos ejecutar el programa de código máquina con el comando **E**, sin embargo no se nos ofrece la posibilidad de grabarlo en cinta; sí la tenemos para el desensamblador. Esta rara posibilidad se nos ofrece por si queremos eliminar las órdenes que contienen las instrucciones dentro del programa (se hace escribiendo el número de orden correspondiente tras cortar el programa con **CAPS SHIFT** + **SPACE**), algo aconsejable una vez se haya adquirido práctica con él, puesto que así disponemos de más memoria libre para desensamblar un programa.

Una deficiencia que presenta es la ausencia de una salida del listado por impresora obligándonos a trabajar con la pantalla con las limitaciones que ésto presenta.

EL COMPILADOR

Posiblemente alguna vez nos hayamos desesperado al observar la tremenda lentitud que tiene el BASIC en la ejecución de algunos programas largos y complicados. Es evidente que la única solución que le queda es recurrir al código máquina, con lo intrincado que resulta este camino. ¿No sería posible, entonces, que una vez que tengamos el programa BASIC hecho nos lo tradujera a código máquina y se ejecutase como tal? Es como si tuviéramos un programa compilador para BASIC (al igual que existen para otros lenguajes, como el PASCAL). Precisamente es lo que presentamos ahora, un programa que compilará (traducirá) nuestro programa BASIC al lenguaje má-

Hay comandos que COMPILER no puede traducir y otros que los transforma.

¡!

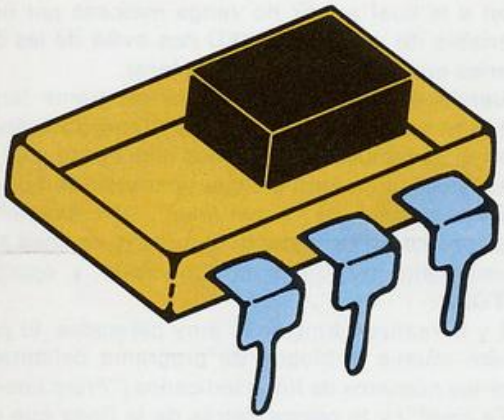
Con ZXED podemos, entre otras cosas, renumerar un programa y hacer que aparezca el número de la línea que estamos introduciendo de forma automática.

*

DLAN es un "mini-lenguaje" de programación, con grandes posibilidades gráficas, que permite una serie de opciones para presentación de textos en la pantalla.

*

La diferencia entre mover un bloque de programa y copiarlo está en que lo primero elimina el bloque en el lugar original y lo traslada al indicado; lo segundo, se mantiene el bloque en el lugar original copiándose donde fue indicado.



Podemos saber directamente la memoria que nos queda libre en la máquina.

quina y lo ejecutará como tal. El compilador que nos ocupa lo distribuye INVESTRONICA y es el COMPILER de *SOFTEK*.
Para trabajar con el COMPILER primeramente le debemos cargar, y una vez realizado podemos empezar ya a programar en BASIC. Antes de la compilación hay que comprobar muy bien el programa para evitar cometer alguna equivocación ya que el compilador tiene muy pocos mensajes de error. De hecho, es aconsejable grabar el programa en BASIC y después compilarlo.
Aunque suponga una maravilla tener un programa de este tipo, puesto que nos evita aprender el código máquina, también tiene sus limitaciones. Así, COMPILER no es capaz de compilar programas que incluyan números decimales, y por tanto no es aconsejable el uso de funciones que puedan producirlos, como es **SIN, COS, TAN, SQR, ↑**. Tampoco acepta cadenas alfanuméricas ni tablas (**DIM**). Además, muchos comandos serán interpretados de forma diferente a la usual, algunos para reforzar la función y otros para disminuirla. Así, no es necesario definir una variable antes de usarla, pero los **GO TO** y **GO SUB** no aceptan una variable ni una expresión. Sólo mencionar otro pequeño detalle: las variables deben ir siempre en mayúsculas. Antes de trabajar con el compilador es aconsejable leerse las instrucciones para conocer estas pequeñas diferencias.
Para entender mejor las ventajas que supone tener un compilador, vamos a teclear el siguiente programa:

```
10 FOR N=1 TO 100
20 PRINT AT 1,1;N
30 NEXT N
```

Una vez tecleado y comprobado que no hay ningún error y que cumple todas las condiciones que exige COMPILER podemos ya compilarlo tecleando **RANDOMIZE USR 49152**. Si no hay ningún problema nos aparecerá el mensaje:

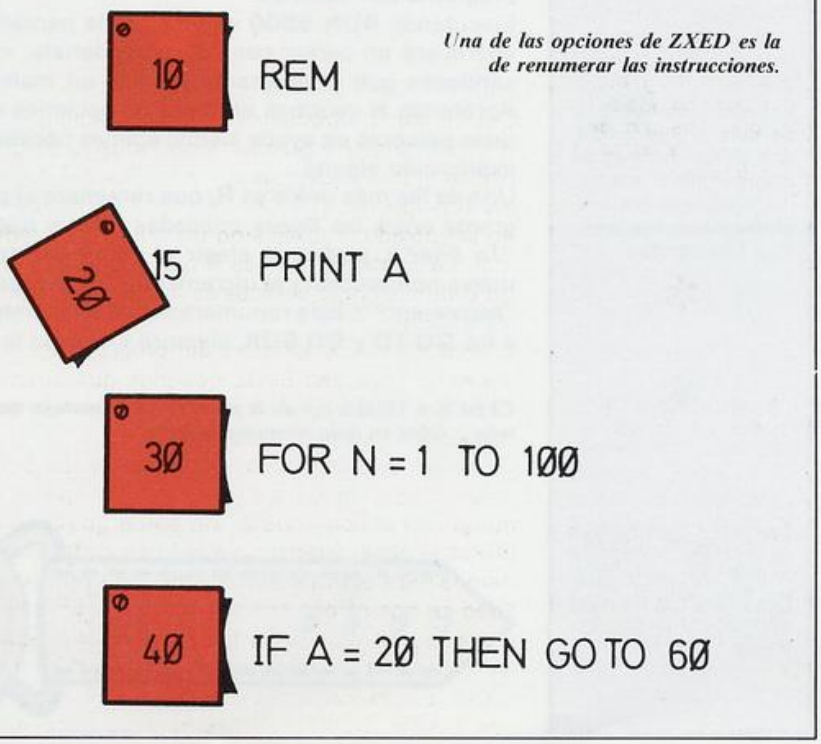
1>PROPERTY OF SOFTEK
ALL COMERCIAL RIGHTS RESERVED

Si lo ha ejecutado antes de compilarlo verá que el programa tarda 19 segundos en terminar. Pruébelo una vez compilado, haga **RUN** normalmente, y el tiempo que tarda ahora es de unos tres segundos. Sorprendente, ¿verdad?
Creo que queda clara la ventaja de un compilador, aunque tenga algunos inconvenientes. De todas maneras hay que estar atentos al mercado informático para las nuevas versiones y ampliaciones, como se asegura en las instrucciones del COMPILER.

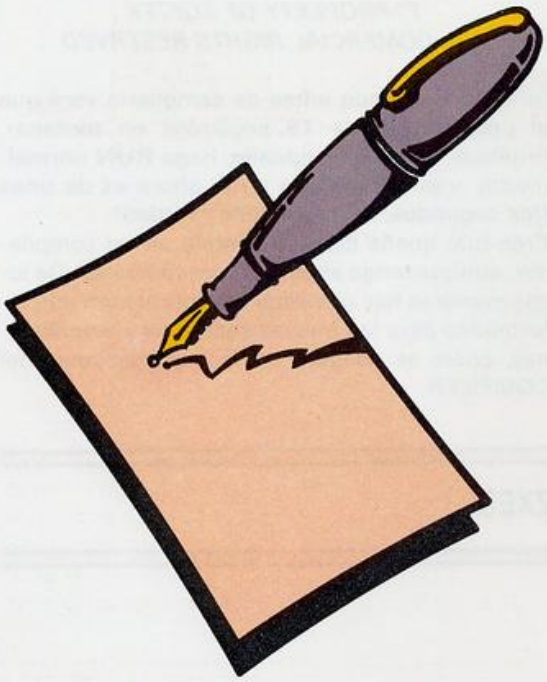
ZXED

Con el compilador no necesitamos saber código máquina, puesto que él mismo se encarga de traducir nuestro programa BASIC.

Los aficionados a realizar sus propios programas quizá encuentren que el SPECTRUM no les da ninguna facilidad para realizar reformas según se va escribiendo, pero todo se complica más cuando el programa que tenemos entre manos es muy largo y necesitamos, por ejemplo, renumerar las líneas o cambiar de nombre a alguna variable.
Esta deficiencia la viene a suplir ZXED, un com-



Una de las opciones de ZXED es la de renumerar las instrucciones.



ZXED nos permite copiar fragmentos de programa en el punto donde indiquemos.

pletísimo programa de ayuda a la programación, que nos sacará de más de un apuro a la hora del tecleo.

Para empezar a trabajar con él lo cargaremos con **LOAD**, como un programa normal. Pero si el programa a depurar lo tenemos en cinta, entonces se cargará primero ZXED con **LOAD** y después el programa con **MERGE**.

Ejecutando **RUN 9900** se limpiará la pantalla y aparecerá un cursor con "T" parpadenate, indicándonos que el programa ya está en marcha. Apretando H veremos el menú de opciones con unas palabras de ayuda siendo apenas necesaria explicación alguna.

Una de las más útiles es **R**, que renumera el programa entre las líneas indicadas *("From line?", "To line?")*, pudiendo elegir el comienzo de la nueva numeración y el incremento *("New base?", "Increment?")*. Esta renumeración afecta también a los **GO TO** y **GO SUB**, siempre y cuando la orden a la cual acudir no venga indicada por una variable. Si así fuera, ZXED nos avisa de las órdenes que no ha podido renumerar.

Cuando estamos tecleando un programa largo podemos evitarnos escribir el número de línea con el comando **S**, el cual nos pide desde cual línea y el incremento en que aparecerá la numeración automática *("From line?", "Increment?")*. Para que deje de aparecer el número de línea automáticamente basta con borrarlo y escribir **STOP**.

M y **C** realizan funciones muy parecidas. El primero mueve el bloque de programa delimitado por los números de línea indicados *("From line?", "To line?")* y lo coloca detrás de la línea que digamos *("Insert after?")*, eliminándole de su situación original. **C** copia un bloque de programa con las mismas condiciones que **M**, aquí, como es lógico, el bloque original no es eliminado. En ambos casos, una vez realizada la función nos indica dónde ha sido insertado el bloque en cuestión *("...To...Inserted")*, haciéndolo siempre con números de línea seguidos, por tanto es necesario después renumerar el programa.

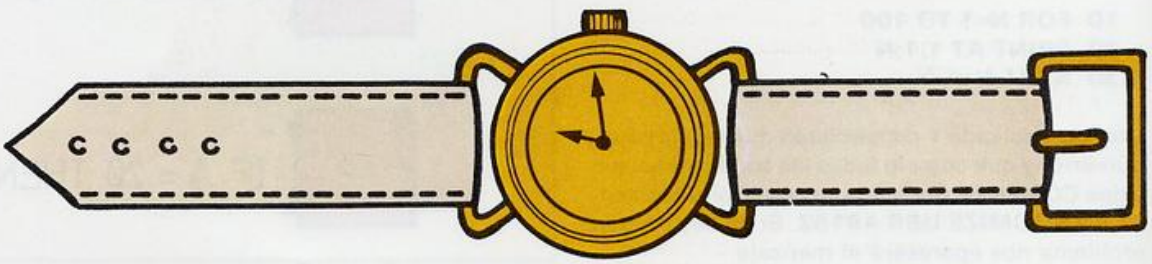
Ya nos habremos percatado de lo fastidioso que resulta, al realizar una serie de cambios en un programa, ir borrando un conjunto de líneas que sobran una por una. Con ZXED tenemos la facilidad de borrar todo un bloque de una manera muy sencilla. Para ello tenemos **D**, que nos pide la primera línea a borrar *("From line?")* y la última *("To line?")*.

Podemos saber rápidamente la memoria que nos queda libre para programar con sólo pulsar **B**, proporcionando el resultado en *bytes*. Igualmente podemos saber en qué dirección de memoria se encuentra almacenada una línea con pulsar **L**.

Otras dos opciones muy parecidas son **A** y **P**. Aquella cambia un carácter por el que nosotros le digamos *("...To be...")* dentro del margen de líneas que indiquemos, y el segundo comando nos escribe en las líneas que mencionemos el carácter deseado. Con **A**, una vez realizado el cambio nos escribe los números de orden en donde se hizo. Si no deseamos ver dichos números basta con pulsar **V** apareciendo el mensaje *Verify Off*.

Por último, cuando deseemos grabar en cinta el programa que estamos realizando es necesario previamente borrar el ZXED eliminando las líneas 9900 a 9914 una a una, puesto que no permite el autoborrado con **D**.

CLOCK *y* **TIME$** *nos da la posibilidad de manejar un reloj y saber en todo momento la hora.*



¡!

BETA-BASIC es una ampliación al lenguaje BASIC del SPECTRUM, que aumenta la potencia de programación al poder manejar comandos que aparecen en ordenadores muy superiores.

*

Los programas de ayuda a la programación nos sirven para sacar más partido al ordenador y depurar los programas que nos estemos fabricando.

*

Tenemos posibilidad de manejar ventanas, definir su color interior y elegir entre ocho tipos de contornos distintos.

AAAAA

¡!

DLAN es un programa que nos permite presentar textos de muy diferentes formas.

AMPLIACION DEL BASIC

Para los expertos programadores quizás el BASIC del SPECTRUM se quede pequeño al faltarle una serie de instrucciones que en ciertas ocasiones pueden ser necesarias. Esto sólo tiene dos soluciones: la primera es la de cambiar de máquina, evidentemente la mejor pero la más costosa. La otra es la de obtener una ampliación del lenguaje BASIC para nuestra máquina. Esto último es lo que nos ofrece VENTAMATIC con el BETA-BASIC.

Este popular programa, del que ya han surgido múltiples versiones, nos da la posibilidad de tener unos comandos nuevos incorporados, y algunos de los existentes transformados, en nuestro ordenador, ofreciéndonos unas posibilidades que no tienen otros aparatos de precio más elevado. Ocupa unos 9 K., quedando por tanto 32 K. libres para la programación.

Comentaremos aquí algunos de los muchos comandos y novedades que incorpora:

— **BREAK**: No es un comando. Es el **BREAK** normal del SPECTRUM pero aumentada su potencia al poder cortar un programa en código máquina.

— **CLOCK, TIME$**: Controlan un reloj interno, que nosotros ponemos en hora, y que tiene posibilidad de alarma. Cada vez que lo requiramos podemos visualizarla.

— **ELSE**: Es el complemento a **IF...THEN**. Cada vez que una condición se cumple se ejecuta lo que se encuentra a continuación de **THEN**, si no se cumple, saltará y el programa seguirá después de **ELSE**.

— **RENUM**: Renumera los números de línea del programa con que estemos trabajando o del bloque de programa indicado. Afecta también a **GO TO, GO SUB, RUN...**

— **AUTO**: Nos va dando, automáticamente, el número de la línea que estamos introduciendo.

— **BIN$, DEC, HEX$**: Cambia un número al sistema de base que nosotros indiquemos. El primero, convierte un número en decimal a binario. El segundo, nos da como resultado un decimal tras introducir un número hexadecimal, y el tercero, transforma un decimal en hexadecimal.

— **MOD**: Da el resto de la división (es la función módulo).

Estos, y más comandos que incorpora el BETA-BASIC, son cargados por medio de software en la RAM, por tanto al desconectar la alimentación de la máquina desaparecerá dicho programa. Igualmente, para ejecutar un programa en el que se hayan utilizado sentencias del BASIC ampliado será necesario primero cargar este programa.

DLAN

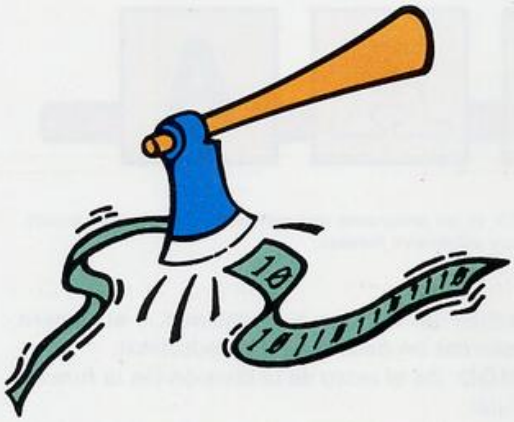
Si alguna vez hemos tenido la necesidad de realizar una presentación en pantalla para nuestros

DLAN permite trabajar con 11 tipos de letras, imprimiendo el texto de izquierda a derecha (con =) y de derecha a izquierda (con <).

*

ELSE es el complemento a **IF...THEN**. Si la condición se cumple se ejecuta lo que está a continuación de **THEN**, en caso contrario, el programa seguirá después del **ELSE**.

Un programa compilado se ejecuta más rápidamente que el mismo en BASIC.



El **BREAK** *del BETA-BASIC corta también los programas del código máquina.*

¡!

Un desensamblador sirve para obtener el listado en lenguaje ensamblador y en lenguaje máquina de un programa de este tipo.

*

Siempre que queramos ejecutar un programa que tenga comandos del BETA-BASIC será necesario cargar primero éste y luego el programa.

*

En el SPDE, **A** enseña el listado del programa en cuatro columnas: la dirección de memoria, contenido de la memoria, nemónicos y parámetros del nemónico.

textos verdaderamente sugerente, nos habremos visto en la necesidad de recurrir al código máquina. Sin embargo, ahora ya no es necesario, puesto que disponemos de DLAN *(Display Languaje*, en inglés), un lenguaje de programación que dispone de unos comandos para facilitarnos esta tarea.
Los comandos de DLAN deben ir escritos en instrucciones **REM** del BASIC y separados entre sí por ";", además de otro al final de la línea. Otra peculiaridad es la de no tener mensajes de error, por lo tanto, ante un parámetro incorrecto el programa lo ejecutará de una manera extraña, debiéndose repasar todo para averiguar donde se ha cometido el error.
Una de las facilidades que nos ofrece es la de manejar ventanas. Con ello podemos simular que tenemos una «mini-pantalla» y cuando se produzca un *scroll* el texto no se saldrá de la ventana definida.

```
10 REM W06090032;
```

Hemos definido una ventana, con **W**, que empezará en la línea 6 ("06") con 9 ("09") de alto. La anchura vendrá definida porque empezará en la columna 0 ("00") con 32 de ancho.

Si queremos hacer un reborde en el límite de la ventana, tenemos ocho donde elegir (de 1 a 8). Para ello ejecutaremos:

```
20 REM E8;
```

El color del interior lo elegiremos con **C** y el número del color correspondiente.

```
30 REM C4; F*;
```

En este caso tendremos una ventana de color verde. El comando que se encuentra a continuación (**F**) la llena del carácter escrito seguido a él, en este caso es "*".
Para escribir un texto disponemos de dos comandos. El primero es = que escribe lo que se encuentra a continuación de izquierda a derecha. Pero antes de imprimir es necesario especificar qué tipo de letra vamos a utilizar. Para ello disponemos de 11 tipos (del 1 al 9 más A y B). El comando que utilizamos es **T**.

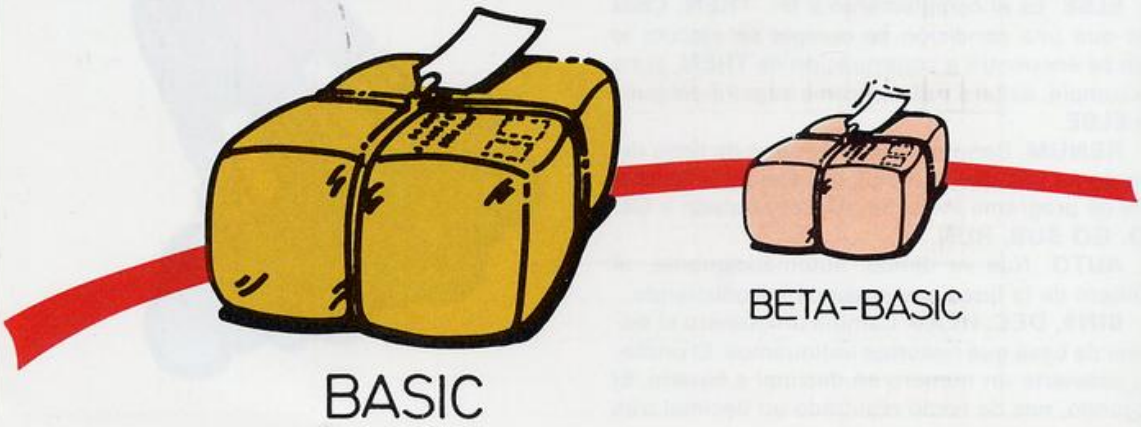
```
40 REM T3;=BIENVENIDOS A;
```

En esta ocasión hemos elegido el tipo de letra 3 para escribir el texto "BIENVENIDOS A" (observemos que no es necesario entrecomillar el texto).
El segundo comando de impresión es < haciendo que el texto que se encuentre a continuación se imprima de derecha a izquierda.

```
50 REM T2;<DLAN;
```

En esta ocasión se imprime el texto "DLAN" con el tipo de letra 2.
Ahora es aconsejable probar el ejemplo aquí presentado como una muestra de lo que se puede hacer. Sin embargo, ofrece mucho más que es necesario conocer para obtener unos óptimos resultados con este lenguaje.

Con BETA-BASIC tenemos una ampliación al BASIC de SINCLAIR al tener nuevos comandos.



BIFURCACIONES

Durante los capítulos precedentes han ido apareciendo tablas resumidas de los diferentes grupos de instrucciones, donde entre otros datos de interés encontrábamos el efecto que determinada instrucción obraba sobre los indicadores F.
Pues bien, nuestro objetivo durante las próximas líneas será describir las instrucciones que modifican la secuencia de un programa. En ocasiones necesitaremos que ésta se rompa tan solo cuando determinada condición se cumpla, y lo «averiguaremos» analizando el contenido del bit especificado en el código de operación de la instrucción, dentro del registro F.

EL PROGRAM COUNTER (PC)

Por lo que hasta ahora sabemos, un programa no es otra cosa que una serie de bytes almacenados en la memoria de nuestro ordenador, que la CPU interpreta al objeto de llevar a cabo una determinada tarea.

En todo momento debe conocer la dirección a donde dirigirse y recavar la información, pues de otra manera el caos sería inevitable y la ejecución de cualquier rutina por sencilla que fuera, sería algo más que misión imposible.
Con la saludable intención de no armarse un fenomenal lío, el Z 80 conserva en su registro PC (contador de programa) la situación de la próxima instrucción a procesar y repite cíclicamente los siguientes pasos:
1) Lee el contenido del registro PC.
2) Se posiciona en la dirección de memoria indicada por éste.
3) Recoge allí el byte almacenado y averigua si es necesario leer octetos adicionales (recuerda que el formato de las instrucciones del Z 80 varía de 1 a 4 bytes).
4) Le añade al contador de programa PC el número de bytes de la instrucción actual, es decir, antes de procesarla, PC ya está señalando a la siguiente.
5) Finalmente, implementa la instrucción y vuelve al paso 1.
Evidentemente, siguiendo este sistema, los programas sólo podrían ejecutarse secuencialmente, es decir, recorriendo un byte tras otro hasta «chocar» con el final de la memoria. Pero tengamos en cuenta que aunque parezca que el ordenador no está haciendo nada mientras está conectado y no ejecutamos programa alguno, en el peor de los casos estará inmerso en alguna ruti-

¡!

Si el código máquina está todavía almacenado a partir de la dirección 23296 introduce como comando directo **GO TO 9996**. En caso contrario, **GO TO 9995** para recuperarlo del casete.

*

Señalemos cual es la primera línea a eliminar y cual la última. Tras ello, comprobaremos que estas han desaparecido de nuestro programa.

MNEMONICO	CODIGO MAQUINA	REGISTRO F								N.º Bytes	CICLOS		NOTAS
		7	6	5	4	3	2	1	0		MAQ.	RELOJ	
		S	Z		H		P/v	N	C				
CALL nn	1 1 0 0 1 1 0 1												ciclos si cc es falsa
	n n n n n n n n												
	n n n n n n n n	•	•	•	•	•	•	•	•	3	5	17	A/B
CALL cc,nn	1 1 x cc x 1 0 0												ciclos si cc es verdadera
	n n n n n n n n												
	n n n n n n n n	•	•	•	•	•	•	•	•	3	3/5	10/17	
RET	1 1 0 0 1 0 0 1	•	•	•	•	•	•	•	•	1	3	10	
RET cc	1 1 x cc x 0 0 0	•	•	•	•	•	•	•	•	1	1/3	5/11	

MNEMONICO	CODIDO MAQUINA	REGISTRO F								N.º Bytes	CICLOS		NOTAS
		7	6	5	4	3	2	1	0		MAQ.	RELOJ	
		S	Z		H		P/v	N	C				
RST p	1 1 x t x 1 1 1	•	•	•	•	•	•	•	•	1	3	11	

t	P	LLAMADA A LA RUTINA
000	00 H	INICIALIZACION DEL SISTEMA
001	08 H	GESTION DE ERRORES
010	10 H	IMPRESION DE UN CARACTER
011	18 H	RECOGIDA DE UN CARACTER
100	20 H	RECOGIDA DEL SIGUIENTE CARACTER
101	28 H	ENTRADA AL CALCULADOR
110	30 H	CREACION DE ESPACIO LIBRE
111	38 H	EXAMEN DEL TECLADO

¡!

Si la carga fue correcta procede a grabar el código máquina en casete mediante la orden **SAVE "C/M" CODE 23296,18.**

*

A continuación, tecleemos el programa DELETE cuidadosamente, respetando los números de instrucción asignados y grabémoslo con **SAVE "DELETE".**

*

Ahora ya podemos comenzar a trabajar con la rutina de borrado de líneas de programa. Supongamos que tenemos un programa en la memoria del cual deseas eliminar cierto bloque de instrucciones.

na de la ROM como, por ejemplo, la de análisis del teclado, en espera de alguna pulsación repitiéndola una y otra vez.
La clave está, por tanto, en el registro PC, y si pudiéramos modificar a nuestro antojo su contenido, sería posible alterar la secuencia de un programa de la misma manera que en BASIC lo hacemos mediante el comando **GO TO.**
Pero entre las instrucciones de carga de registros vistas anteriormente, no disponemos de ninguna del tipo LD PC,nn o algo parecido. Entonces, ¿son posibles las bifurcaciones en C/M? La respuesta es sí, y son las instrucciones de salto (*jump*, en inglés), las encargadas de llevarlas a cabo.

CAMBIOS DE SECUENCIA

Las instrucciones de salto provocan que el control del programa sea transferido a una dirección de memoria especificada en el propio código de operación. La manera de realizar la bifurcación depende del tipo de instrucción elegido, es decir, se pueden ejecutar saltos absolutos, relativos o por direccionamiento indirecto.
Tanto los absolutos como los relativos es posible efectuarlos de manera incondicional o tan solo cuando una determinada condición se verifique (saltos condicionales), mientras que los indirectos siempre son incondicionales.

SALTOS INCONDICIONALES

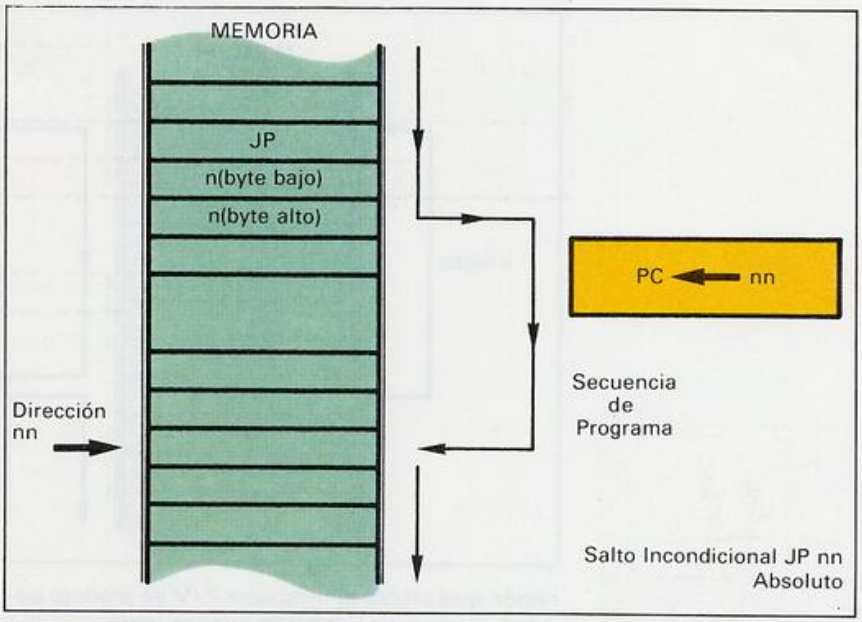
Cuando la CPU accede a la memoria y lee una instrucción de salto incondicional, tras decodificar el primer octeto, reconoce que se trata de una de este tipo. A continuación, cuando el salto es absoluto (JP nn), escruta los dos bytes siguientes donde encuentra la dirección a la cual saltar. Finalmente, carga en el registro PC este valor y continúa con la ejecución del programa a partir de esta última dirección.
El salto, aunque también incondicional, puede efectuarse de forma relativa, es decir, avanzando o retrocediendo en la memoria un determinado número de posiciones a partir de la señalada por el contador de programa PC.
En estos casos, la instrucción tan solo precisa de dos bytes, uno para el código de operación y otro para indicar el desplazamiento, el cual será un número en complemento a 2, y por tanto, en el rango de −128 a +127.
El mecanismo seguido por el Z 80 al implementarla se describe en la figura, y como podemos comprobar es de capital importancia tener presente que el desplazamiento se añade al PC suponiendo que éste se habría actualizado normalmente, como si la instrucción no hubiera sido de salto.
Dada esta circunstancia, los saltos permitidos al procesar una instrucción JR e, donde e es el des-

plazamiento, están comprendidos entre −126 y +129 posiciones de memoria, a partir de la ocupada por el primer byte de la instrucción de salto relativo.

Una de las cualidades de las instrucciones de salto relativo es que son reubicables, es decir, actúan de la misma forma sea cual sea la zona de memoria donde hayamos almacenado nuestro programa en C/M, mientras que con las absolutas, generalmente, no ocurre lo mismo y nos acarrearán problemas de adaptación.

Sin embargo, con ellas sólo se pueden efectuar desplazamientos en los márgenes señalados, mientras que los saltos absolutos pueden recorrer toda la memoria a costa, por supuesto, de ocupar un byte más.

Finalmente, tres instrucciones JP (HL), JP (IX) y JP (IY) se sirven del contenido de los registros indicados para calcular la posición de memoria a la que debe apuntar el PC. Obviamente en todas ellas, el desplazamiento se efectúa de forma absoluta.



LOS SALTOS CONDICIONALES

En el grupo anterior, cuando el microprocesador «tropezaba» en la memoria con una instrucción de salto, fuera cual fuera el estado del registro de indicadores F, SIEMPRE transfería el control del programa a la posición de memoria especificada en la instrucción.

Ahora analizaremos un nuevo tipo de instrucciones, las cuales ejecutan o no el salto (absoluta o relativamente, como antes) basándose en el estado de alguna de las banderas presentes en el registro F.

Para los indicadores de signo S, cero Z, paridad/desbordamiento P/V y arraste C es posible plantear la instrucción de salto absoluto para que en función de su contenido el programa bifurque o no.

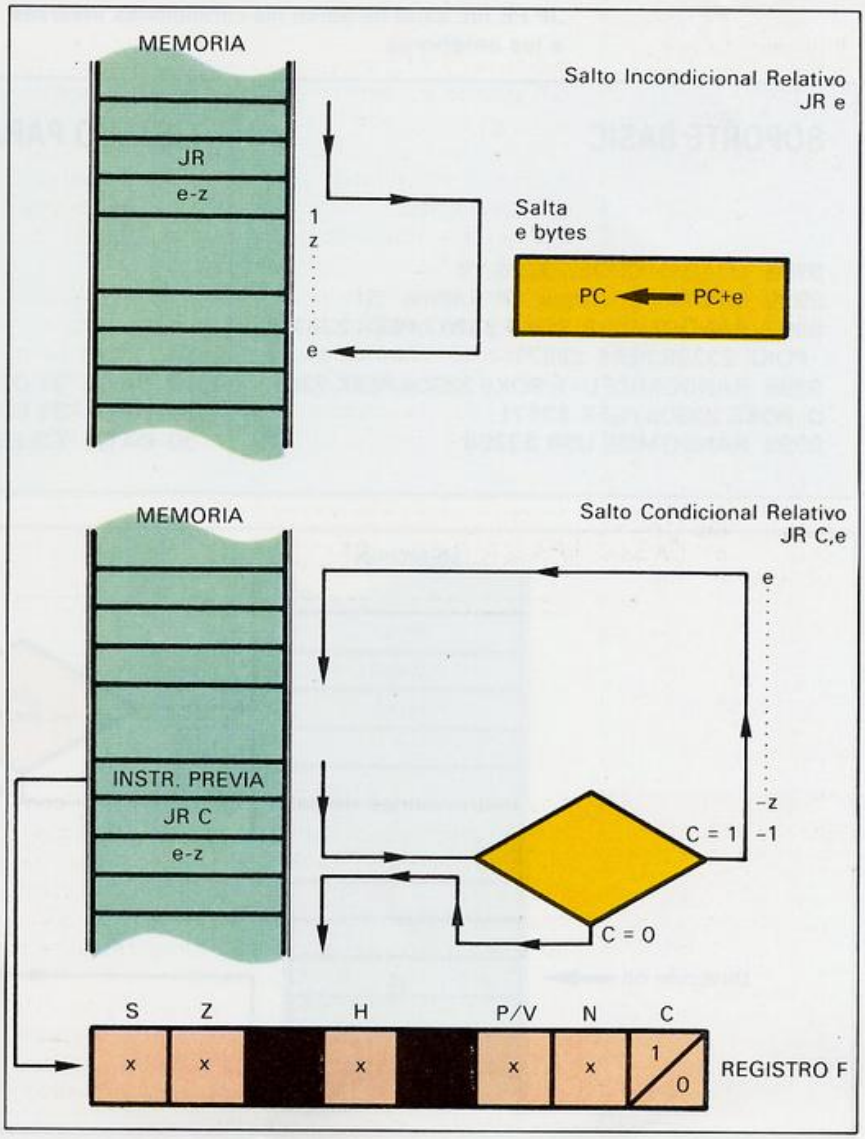
JP NZ,nn: producirá salto a la dirección nn cuando el indicador de cero esté a 0.

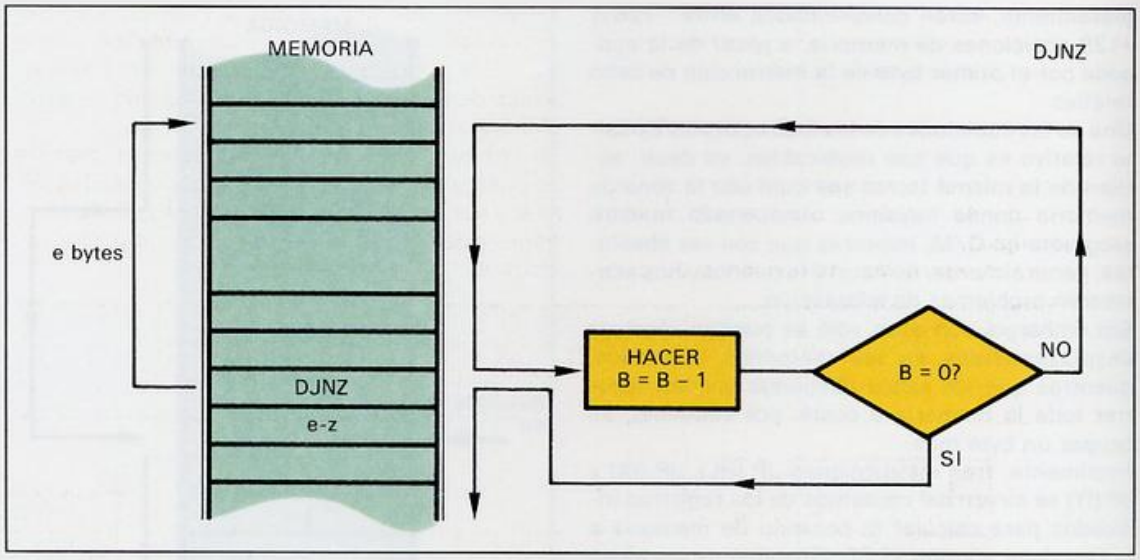
JP Z,nn: salta si la bandera de cero está alzada. Aparentemente, existe un contrasentido en estas instrucciones, pero recordemos que el indicador de cero se sitúa a 1 cuando el resultado de la operación previa fue 0.

JP NC,nn: bifurca en el caso de comprobar que la bandera de acarreo está bajada.

JP C,nn: es la condición contraria a la anterior.

JP PO,nn: el salto es efectivo si en la última ope-





ración que afectó al indicador P/V se produjo paridad impar o no hubo sobrepasamiento.
JP PE,nn: salta de darse las condiciones inversas a las anteriores.
JP P,nn: tras escrutar el indicador de signo S, si éste contiene un 1 se produce el salto.
JP M,nn: es la opuesta de la anterior.
Los saltos condicionales relativos solamente

SOPORTE BASIC

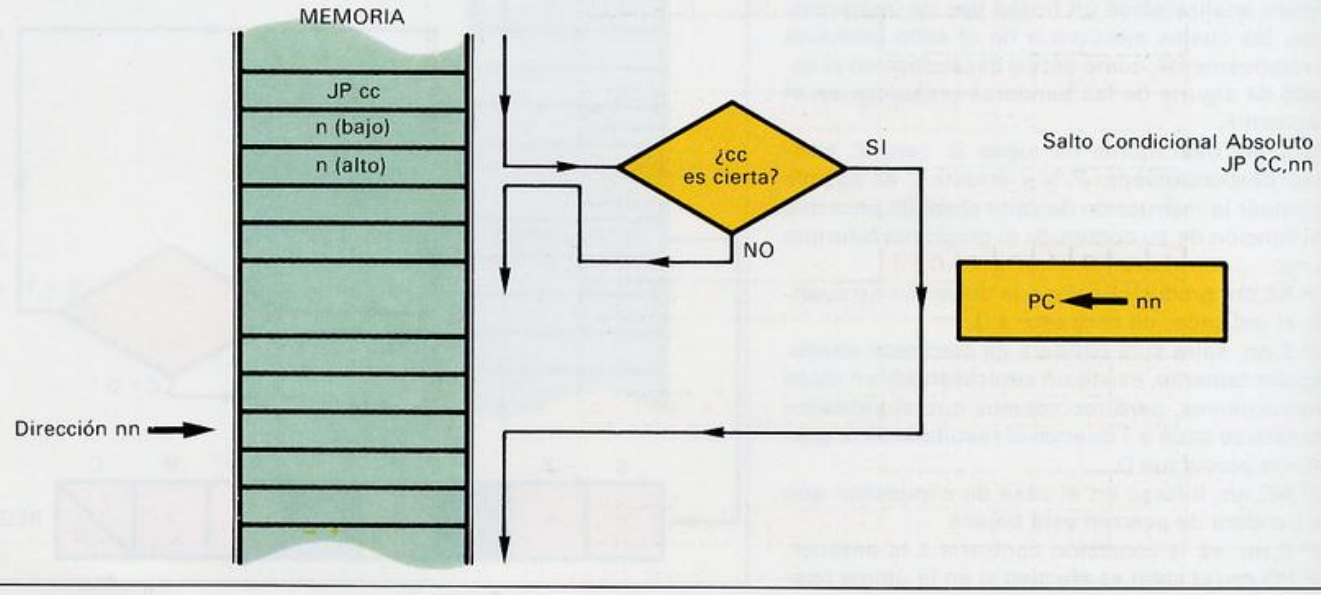
```
9995 LOAD ""CODE 23296,18
9996 INPUT "1a.- Linea ";P,"Ultima ";U
9997 RANDOMIZE P: POKE 23297,PEEK 23670:
  POKE 23298,PEEK 23671
9998 RANDOMIZE U+1: POKE 23304,PEEK 2367
0: POKE 23305,PEEK 23671
9999 RANDOMIZE USR 23296
```

LISTADO PARA CARGADOR

```
10 DATA "21 00 00 CD 6E 19 E5",602
20 DATA "21 00 00 CD 6E 19 D1",582
30 DATA "CD E5 10 C9 **",660
```

LISTADO ENSAMBLADOR

```
LD HL, PRIMERA
CALL 196E
PUSH HL
LD HL, ULTIMA+1
CALL 196E
POP DE
CALL 19E5
RET
```



pueden efectuarse dependiendo del estado de dos de los indicadores: el de acarreo y el de cero. El formato de estas instrucciones es el siguiente:

JR C,e JR NC,e JR Z,e JR NZ,e

donde como antes, e representa el byte de desplazamiento a sumar al PC, caso de cumplirse la condición fijada.
Para terminar, entre las instrucciones de salto existe una de funcionamiento especial (DJNZ,e) la cual utiliza el registro B como condición. Cuando el Z 80 la encuentra decrementa en 1 el contenido de B, comprueba si éste es 0 y de no serlo efectua el salto relativo.
En la figura puedes comprobar como su manejo es una buena técnica para efectuar repetidas pasadas por un grupo de instrucciones, durante el número de veces con que hayamos cargado el registro B.

MNEMONICO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	N.º Bytes	CICLOS MAQ.	CICLOS RELOJ
JP nn	1 1 0 0 0 0 1 1											
	n n n n n n n n											
	n n n n n n n n	•	•	•	•	•	•	•	•	3	3	10
JP cc,nn	1 1 x cc x 0 1 0											
	n n n n n n n n											
	n n n n n n n n	•	•	•	•	•	•	•	•	3	3	10
JR e	0 0 0 1 1 0 0 0											
	x e z x	•	•	•	•	•	•	•	•	2	3	12
JR C,e	0 0 1 1 1 0 0 0											
	x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12
JR NC,e	0 0 1 1 0 0 0 0											
	x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12
JR Z,e	0 0 1 0 1 0 0 0											
	x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12
JR NZ,e	0 0 1 0 0 0 0 0											
	x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12
JP (HL)	1 1 1 0 1 0 0 1	•	•	•	•	•	•	•	•	1	1	4
JP (IX)	1 1 0 1 1 1 0 1											
	1 1 1 0 1 0 0 1	•	•	•	•	•	•	•	•	2	2	8
JP (IY)	1 1 1 1 1 1 0 1											
	1 1 1 0 1 0 0 1	•	•	•	•	•	•	•	•	2	2	8
DJNZ,e	0 0 0 1 0 0 0 0											
	x e z x	•	•	•	•	•	•	•	•	2	2/3	8/13

NOTAS

CONDICION		CC
NZ	No cero	000
Z	Cero	001
NC	No arrastre	010
C	Arrastre	011
PO	Pondad impar	100
PE	Pondad par	101
P	Signo positivo	110
M	Signo negativo	111

• Indicador no afectado

A/B
A: ciclos si NO se ha alcanzado la condición
B: ciclos si se ha alcanzado la condición

e-z proporciona la dirección efectiva de PC+e, puesto que, PC es incrementado en 2 antes de la suma del desplazamiento e.

¡!

Para cargar la rutina DELETE en la memoria, utilicemos el cargador hexadecimal del capítulo 42, al que has de añadir las tres líneas **DATA**.

*

Efectua **RUN**, y cuando seas interrogado sobre «dirección de ubicación» contesta 23296.

LLAMADAS Y RETORNOS

Las llamadas (CALL) son instrucciones que modifican la ejecución secuencial de un programa, pero a diferencia de los saltos toman la precaución de anotar en el *stack* la dirección de memoria de la siguiente instrucción almacenada en la memoria tras ella.

Como podemos comprobar en la tabla correspondiente del capítulo destinado a la presentación del conjunto de ellas, (Saltos Y Llamadas) es factible ejecutarla incondicionalmente o sujeta a alguna restricción de las comentadas anteriormente.

Su funcionamiento es parecido al que en BASIC conseguimos mediante la sentencia **GO SUB**, es decir, cuando el Z 80 la encuentra, carga en el registro PC el contenido de los dos bytes siguientes determinando de esta manera la posición de memoria a donde transferir el control del programa, al objeto de ejecutar allí una determinada subrutina.

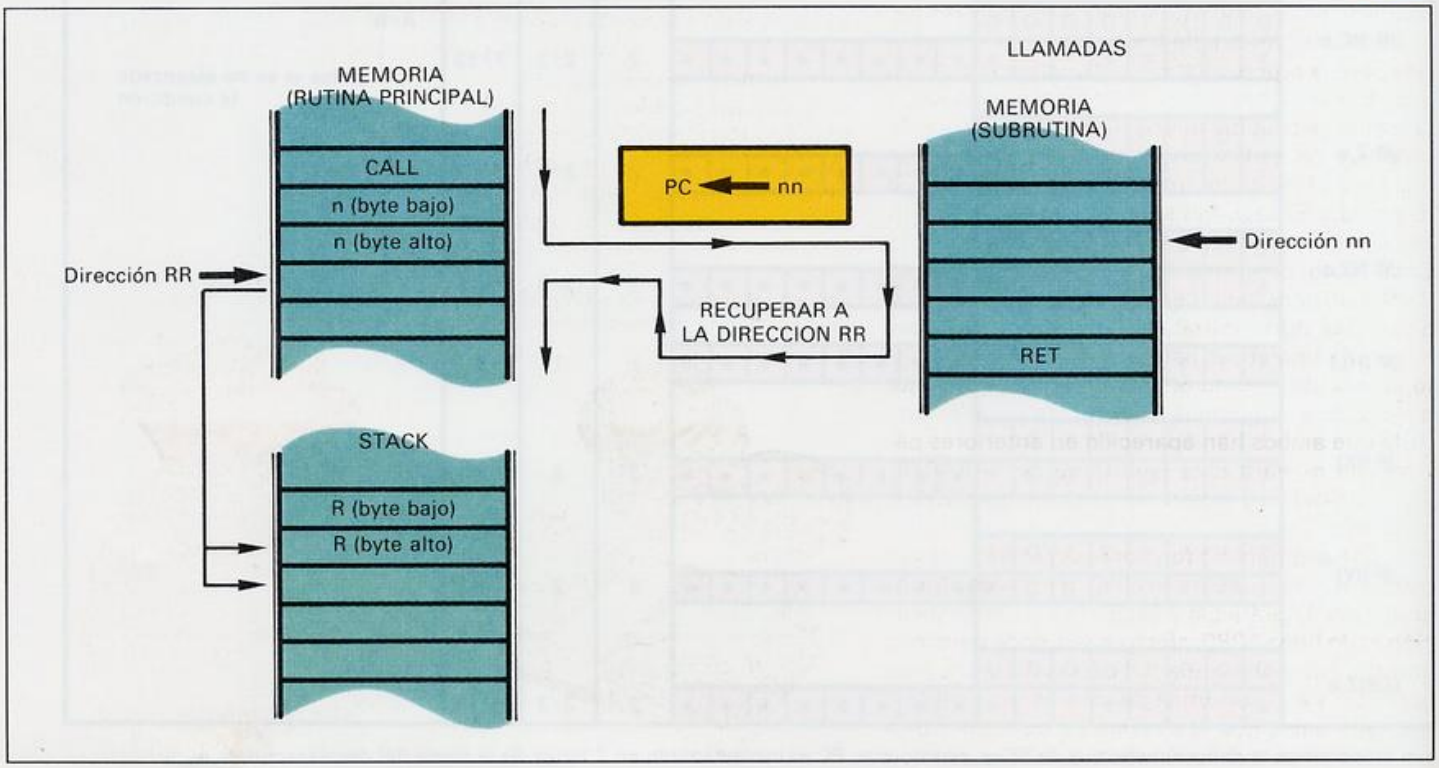
Pero antes de comenzar a procesar las instrucciones almacenadas a partir de dicha dirección, anota en el *stack* la posición de la instrucción siguiente a la llamada CALL. Tras ello, la subrutina es procesada hasta que aparece una instrucción de retorno incondicional RET o una condicionada RET cc, devolviéndose entonces el control a la rutina principal si se da la situación cc estipulada (mecanismo similar al RETURN del BASIC).

En estas circunstancias el microprocesador carga el PC con los dos bytes de lo alto del *stack* y regresa a la dirección especificada por éstos, los cuales, siempre que no manipulemos indebidamente la pila, han de ser los correspondientes a la siguiente posición de memoria tras la instrucción CALL.

Todas las instrucciones de llamada precisan de tres bytes para ser implementadas. Sin embargo, existen otro tipo de llamadas incondicionales de un byte, cuyo mnemónico es RST denominadas instrucciones de ReSTart.

Estas transfieren el control a la rutina almacenada a partir de la dirección especificada en su código de operación. En el caso del Spectrum, éstas se corresponden con ciertas subrutinas de interés incluidas en la ROM, cuyo cometido podemos encontrarlo en la tabla adjunta.

Entre las posibilidades de las instrucciones de llamada se encuentra el aprovechar las utilidades programadas en la ROM del sistema. Hemos preparado una pequeña rutina DELETE, la cual con tan solo 18 bytes elimina, instantáneamente, el bloque de líneas de programa BASIC que nosotros le indiquemos. Por supuesto, no está protegida frente al mal uso y siempre debemos indicar en primer lugar el número de línea inferior.



CALIMOCHO

Calimocho es un juego divertido y ante todo muy entretenido. Con este simpático programa aprenderemos a controlar nuestra codicia. Ya sabéis: «La avaricia rompe el saco». ¡Suerte y... a controlarse!

EL PROGRAMA

Calimocho es un programa en el que ha predominado sobre todo la estética y el color. La utilización de la subrutina de caracteres gigantes de PSION COMPUTERS, junto con el uso alternativo de dos generadores de caracteres (uno el estándar del Spectrum, y otro introducido por nosotros en R.A.M.) han hecho posible tal «perfección» plástica.

Nuestro programa CALIMOCHO consta de dos subprogramas bien diferenciados. El primero genera los 20 gráficos de usuario que son necesarios para la representación del contenido de cada una de las seis caras de que consta un dado, y presenta en pantalla una tabla de puntuación para que tengamos una idea de los puntos que podemos obtener dentro del normal desarrollo de la partida. El segundo subprograma, a todas luces el más importante, es el juego en sí.

Como este programa posee una subrutina en C/M y un generador de caracteres, aconsejamos que antes de la introducción del BASIC tecleemos cada uno de los listados que componen tanto la subrutina como el generador, o en el mejor de los casos carguemos tanto la una como el otro, dado que ambos han aparecido en anteriores páginas de nuestra obra (subrutina de CARACTERES GIGANTES y generador de caracteres ROTULACION).

Para que el programa funcione correctamente debemos introducir instrucción a instrucción el subprograma 1 (gráficos) y grabarlo con autoejecución en la línea 1080, siempre y cuando hayamos grabado con anterioridad el código máquina perteneciente a la subrutina de caracteres gigantes. Así evitaremos que el sistema se «cuelgue». Una vez hecho esto, realizaremos similar proceso con el segundo subprograma, grabándolo con autoejecución en la línea 2360 y a continuación el juego de caracteres ROTULACION a partir de la dirección 51656.

EL CALIMOCHO

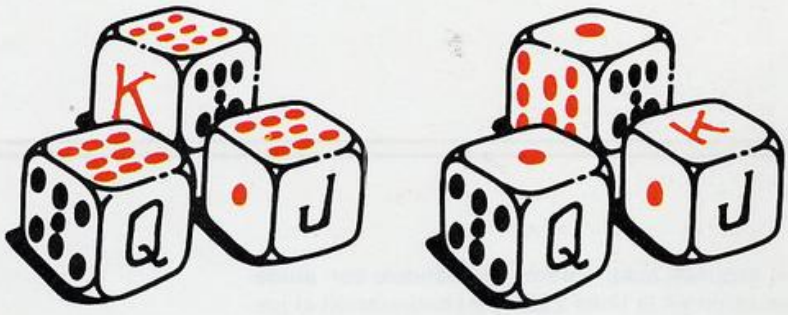
Las instrucciones para jugar al CALIMOCHO son las siguientes: se juega con tres dados de poker. En cada tirada, el Spectrum genera 3 valores aleatorios, (uno por cada dado) cada uno de los cuales se corresponderá con una representación en pantalla de los dados.

Sólo puntúan los «pitos» y los reyes (K), los primeros tienen un valor de 100 puntos mientras que los segundos valen 50. Tres «pitos» puntúan 1000 puntos, y tres reyes (KKK) 500.

Un jugador podrá plantarse cuando lo desee, pero si insiste en su afán de aumentar sus puntos personales, y en esa tirada no consigue puntuación, pierde los puntos parciales acumulados, pasando el turno al jugador siguiente.

De todas las jugadas que se pueden dar, la peor es el CALIMOCHO (3 rojos) es decir sacar en cada uno de los tres dados los ocho puntitos rojos. El Calimocho implica para el jugador que tenga la «fortuna» de conseguirlo, ¡empezar desde cero!

Con este simpático programa aprenderemos a controlar nuestra codicia; tengamos en cuenta que la avaricia rompe el saco.



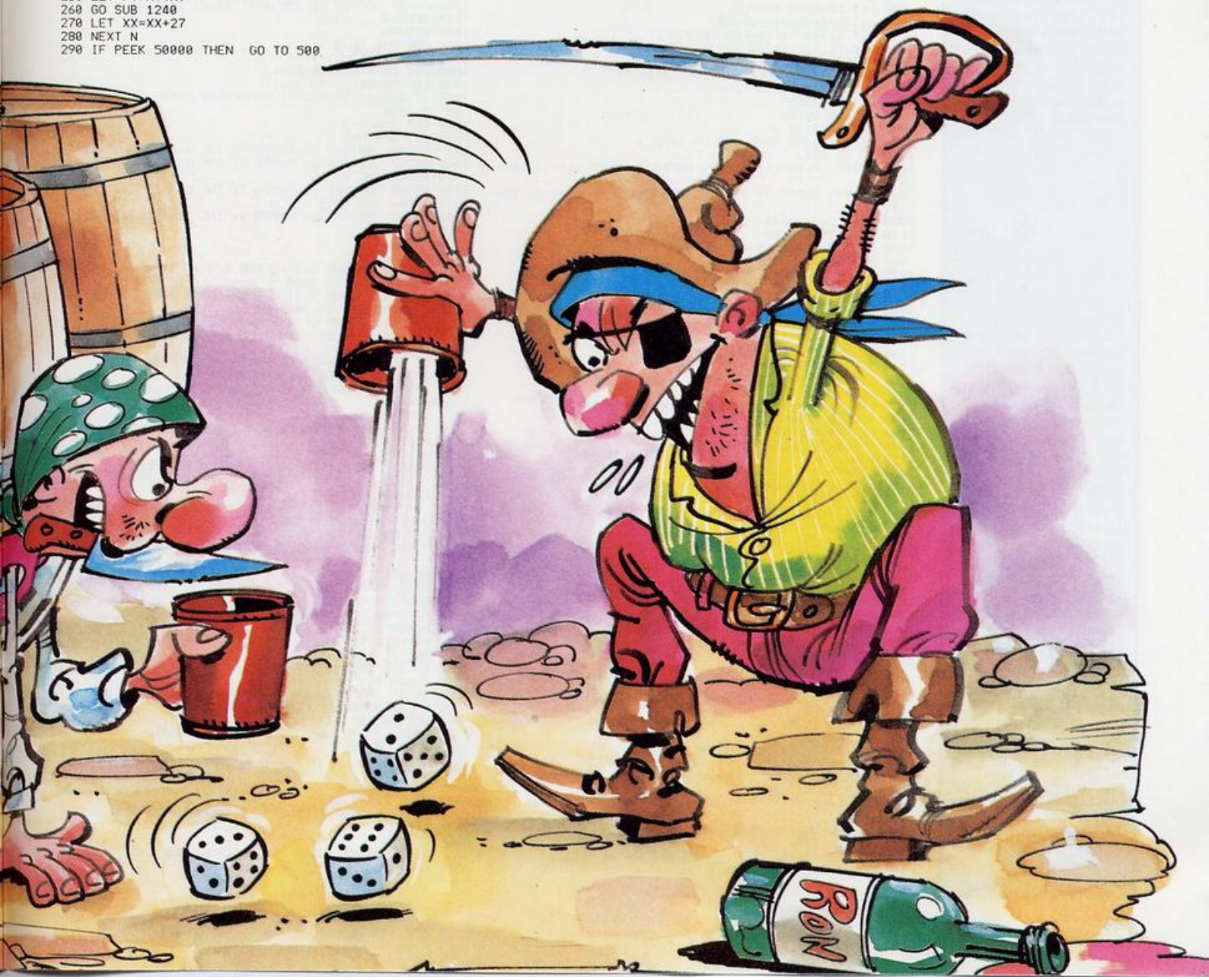
En el CALIMOCHO utilizaremos tres dados de poker. La peor jugada es el CALIMOCHO: sacar en cada uno de los tres dados los ocho puntitos rojos, lo cual equivale a empezar desde cero.

SUBPROGRAMA 1

```
   1 REM ************************          * J.M.MAYORA
L SERRANO *          ************************    *
 CALIMOCHO  (C) 1985 *          ************************
*
   5 REM * PROGRAMA PARA 48 K. *          ************
************
  10 LET YY=50
  20 LET FILA=4
  30 DATA 3,15,63,63,127,127,255,255
  40 DATA 192,240,252,252,254,254,255,255
  50 DATA 255,255,127,127,63,63,15,3
  60 DATA 255,255,254,254,252,252,240,192
  70 DATA 3,3,3,0,0,0,0,0
  80 DATA 254,254,254,112,112,112,112,112
  90 DATA 0,112,112,112,120,63,63,31
 100 DATA 112,112,112,112,240,224,224,192
 110 DATA 15,63,63,120,112,112,112,112
 120 DATA 192,240,240,120,56,56,56,56
 130 DATA 119,115,113,120,63,63,15,0
 140 DATA 184,216,232,240,120,184,188,60
 150 DATA 97,243,243,97,0,0,12,30
 160 DATA 134,207,207,134,0,0,48,120
 170 DATA 30,12,0,0,97,243,243,97
 180 DATA 120,48,0,0,134,207,207,134
 190 DATA 252,252,252,112,112,113,115,119
 200 DATA 127,127,127,60,248,224,192,128
 210 DATA 127,127,124,112,112,252,252,252
 220 DATA 192,224,240,120,60,127,127,127
 230 DATA 6,12,"100 Puntos.",145,6,13,"50 Puntos.",15
3
 240 DATA 12,5,"1000 Puntos.",137,12,6,"500 Puntos.",
145
 250 DATA 12,7,"CALIMOCHO.",153
 260 REM [INV.]  GRAFICOS DE USUARIO [TRUE]
 270 LET A$="ABCDEFGHIJKLMNOPQRST"
 280 FOR G=1 TO LEN A$
 290 FOR F=0 TO 7
 300 READ A
 310 POKE USR A$(G)+F,A
 320 NEXT F
 330 NEXT G
 340 PAPER 0: BORDER 0: INK 9
 350 CLS
 360 LET C=3
 370 LET P$="CALIMOCHO"
 380 LET YY=0
 390 LET XS=3: LET YS=3
 400 INK 1: PAPER 7
 410 GO SUB 970
 420 LET P$="TABLERO DE PUNTUACIONES"
 430 LET YY=25
 440 LET XS=1: LET YS=2
 450 INK 5: PAPER 0
 460 GO SUB 970
 470 BRIGHT 1: INK 2
 480 PRINT AT 6,C;"AB"
 490 PRINT AT 7,C;"CD"
 500 GO SUB 650
 510 PRINT AT 9,C;"QR"
 520 PRINT AT 10,C;"ST"
 530 GO SUB 650
 540 PRINT AT 12,C;"AB AB AB"
 550 PRINT AT 13,C;"CD CD CD"
 560 GO SUB 650
 570 PRINT AT 15,C;"QR QR QR"
 580 PRINT AT 16,C;"ST ST ST"
 590 GO SUB 650
 600 PRINT AT 18,C;"MN MN MN"
 610 PRINT AT 19,C;"OP OP OP"
 620 GO SUB 650
 630 GO TO 710
 640 REM [INV.]  PRESENTACION [TRUE]
 650 BEEP .2,40: INK 6
 660 READ COL,P,P$,XX
 670 GO SUB 900
 680 GO SUB 980
 690 INK 2
 700 RETURN
 710 FOR G=0 TO 1
 720 FOR F=2 TO 7
 730 INK F
 740 BRIGHT 0
 750 GO SUB 980
 760 NEXT F
 770 NEXT G
 780 FOR G=1 TO 3
 790 FOR N=1 TO 10
 800 BEEP .02,N+29
 810 NEXT N
 820 NEXT G
 830 BEEP .5,40
 840 CLS
 850 LET P$="Pulsa  PLAY  en el cassette"
 860 LET yy=80
 870 GO SUB 970
 880 LOAD "CALIMOCHO2"
 890 REM [INV.] RUTINA 1 [TRUE]
 900 LET YY=YY+24: LET FILA=FILA+3
 910 FOR Q=0 TO P-1
 920 PRINT  INK 9;AT FILA,COL+Q;"."
 930 BEEP .02,10
 940 NEXT Q
 950 RETURN
 960 REM [INV.]SBR. CARACT. GIGANTES[TRUE]
 970 LET xx=(256-8*xs*LEN p$)/2
 980 LET i=23306
 990 POKE i,xx: POKE i+1,yy: POKE i+2,xs: POKE i+3,ys
: POKE i+4,8
1000 LET i=i+4: LET ω=LEN p$
1010 FOR q=1 TO ω
1020 POKE i+q,CODE p$(q)
1030 NEXT q
1040 POKE i+ω+1,255
1050 LET ω=USR 64869
1060 RETURN
1070 REM [INV.] AUTO-EJEC. OBLIGATORIA[TRUE]
1080 CLEAR 64868
1090 LOAD "LIT"CODE 64869
1100 RUN
```

SUBPROGRAMA 2

```
  1 REM ************************          * J.M.MAYORA
L SERRANO *          ************************          *
 CALIMOCHO  (C) 1985 *          ************************
*
  5 REM * PROGRAMA PARA 48 K. *          ************
************
 10 PAPER 0: BORDER 0: CLS
 20 GO SUB 1330
 30 LET C$=""
 40 LET D$=""
 50 LET FIN=0
 60 POKE 23658,8
 70 DIM P(3)
 80 DIM T(3)
 90 DIM A(3)
100 DIM D(3)
110 LET PUNTOS=0
120 LET TINTA=3
130 PAPER 0: BORDER 1: INK 9
140 CLS
150 POKE 23606,200: POKE 23607,200
160 GO SUB 1590
170 LET A$="CALIMOCHO"
180 LET XS=3: LET YS=6
190 FOR F=0 TO 1
200 LET YY=8*F*14+8: LET XX=8
210 FOR N=1 TO LEN A$
220 INK TINTA
230 IF TINTA=6 THEN  LET TINTA=2
240 LET TINTA=TINTA+1
250 LET P$=A$(N)
260 GO SUB 1240
270 LET XX=XX+27
280 NEXT N
290 IF PEEK 50000 THEN  GO TO 500
300 LET YS=4
310 NEXT F
320 POKE 23606,0: POKE 23607,60
330 LET P$="## R U N ##"
340 LET YY=80: LET XS=2: LET YS=2
350 GO SUB 1230
360 LET P$="Pulsa una tecla"
370 LET YY=180: LET XS=2: LET YS=1
380 PAPER 1
390 GO SUB 1230
400 POKE 50000,1
410 PRINT  PAPER 0; OVER 1; INK TINTA;AT 10,5;"
420 PRINT  PAPER 0; OVER 1; INK TINTA;AT 11,5;"
430 IF TINTA=7 THEN  LET TINTA=0
440 LET TINTA=TINTA+1
450 IF INKEY$="" THEN  GO TO 410
460 LET P$="
470 GO SUB 1230: PAPER 0
480 GO SUB 1620
490 GO SUB 1700
500 GO SUB 1660
510 POKE 23606,0: POKE 23607,60
520 FOR N=1 TO 3
530 PRINT AT 9,10*N-9; PAPER N+3; INK 9;"  MOCHO ";N
;" "
540 PRINT AT 11,10*N-6;"0000"
550 NEXT N
560 LET A$="MNMNEFIJQRAB"
570 LET B$="QPOPGHKLSTCD"
580 PRINT  INK 9; PAPER 0;AT 18,6;"PUNT. PARCIALES=
```



Para que el programa funcione correctamente debemos introducir instrucción a instrucción el subprograma 1 (gráficos) y grabarlo con autoejecución en la línea 1080, siempre y cuando hayamos grabado con anterioridad el código máquina perteneciente a la subrutina de caracteres gigantes. Una vez hecho esto, realizaremos similar proceso con el segundo subprograma, grabándolo con autoejecución en la línea 2360.

Como este programa posee una subrutina en C/M (CARACTERES GIGANTES) y un generador de caracteres (ROTULACION), aconsejamos que antes de la introducción del BASIC tecleemos cada uno de los listados que componen tanto la subrutina como el generador, o en el mejor de los casos carguemos tanto la una como el otro, dado que ambos han aparecido en anteriores páginas de nuestra obra.

```
 590 PRINT #0; INK 6; PAPER 1;" ~1~- JUGADA     ~P~-
PLANTARSE"
 600 POKE 23606,200: POKE 23607,200
 610 PRINT  INK 2; BRIGHT 1;AT 14,25;"-TOPE-"
 620 PRINT  INK 6;AT 16,26;TP
 630 POKE 23606,0: POKE 23607,60
 640 FOR T=1 TO 3
 650 GO SUB 1790
 660 GO SUB 1740
 670 PRINT  FLASH 1; OVER 1; PAPER T+3;AT 9,10*T-9;"
     "
 680 LET K$=INKEY$
 690 IF K$="1" THEN  BEEP .05,40: GO TO 730
 700 IF K$="P" THEN  BEEP .5,40: GO TO 860
 710 RANDOMIZE
 720 BEEP .01,30: GO TO 680
 730 GO SUB 1790
 740 FOR D=1 TO 3
 750 LET D(D)=INT (RND*6)+1
 760 NEXT D
 770 GO SUB 1860
 780 GO SUB 1990
 790 IF FIN THEN  LET FIN=0: GO SUB 2170
 800 IF CAM=3 THEN  LET CAM=0: LET PUNTOS=0
 810 IF PUNTOS=0 THEN  BEEP .3,40: BEEP .6,20: GO SUB
2080: GO SUB 2120: GO SUB 2150: GO TO 830
 820 GO TO 680
 830 NEXT T
 840 IF T>=3 THEN  GO TO 640
 850 STOP
 860 FOR B=30 TO 45
 870 BEEP .03,B
 880 NEXT B
 890 LET E$=STR$ PUNTOS
 900 LET P(T)=P(T)+PUNTOS
 910 LET C1=INT ((10-LEN E$)/2)
 920 LET C1=C1+10*T-9
 930 PRINT  PAPER 0;;AT 11,10*T-9;"
 940 PRINT  PAPER 0; INK 9;AT 11,C1;P(T)
 950 LET PUNTOS=0
 960 GO SUB 2080
 970 GO SUB 2150
 980 GO SUB 2120
 990 IF P(T)>=TP THEN  GO TO 1010
1000 GO TO 830
1010 PRINT  FLASH 1; INK 2; PAPER 7; OVER 1;AT 9,10*T
-9;"
1020 PRINT  FLASH 1; INK 7; PAPER 2; OVER 1;AT 11,10*
T-9;"
1030 POKE 23606,200
1040 POKE 23607,200
1050 LET P$="GANADOR-MOCHO "+STR$ T
1060 BRIGHT 1: PAPER 0
1070 FOR T=7 TO 3 STEP -1
1080 INK T
1090 GO SUB 1230
1100 NEXT T
1110 POKE 23606,0
1120 POKE 23607,60
1130 BRIGHT 0
1140 INK 5: PAPER 1
1150 LET YY=177: LET XS=1
1160 LET YS=2
1170 LET P$="-PULSA UNA TECLA PARA EMPEZAR-"
1180 GO SUB 1230
1190 LET K$=INKEY$
1200 IF K$="" THEN  GO TO 1190
1210 RUN
1220 STOP
1230 LET xx=(256-8*xs*LEN p$)/2
1240 LET i=23306
1250 POKE i,xx: POKE i+1,yx: POKE i+2,xs: POKE i+3,ys
: POKE i+4,8
1260 LET i=i+4: LET w=LEN p$
1270 FOR q=1 TO w
1280 POKE i+q,CODE p$(q)
1290 NEXT q
1300 POKE i+w+1,255
1310 LET w=USR 64869
1320 RETURN
1330 INK 7: POKE 23606,0: POKE 23607,60
1340 LET P$="TIENES QUE INTRODUCIR EL TOPE"
1350 LET YY=50
1360 LET XS=1: LET YS=2
1370 GO SUB 1230
1380 LET P$="NUMERICO AL QUE SE VA A AJUSTAR"
1390 LET YY=70
1400 GO SUB 1230
1410 LET YY=90
1420 LET P$="ESTA PARTIDA DE CALIMOCHO."
1430 GO SUB 1230
1440 POKE 23606,200
1450 POKE 23607,200
1460 LET YY=110
1470 INK 5
1480 LET P$="MAX.= 5000       MIN.= 50000"
1490 GO SUB 1230
1500 POKE 23606,0: POKE 23607,60
1510 INPUT "          TOPE= ";TP
1520 IF TP<5000 OR TP>50000 THEN  GO TO 1510
1530 LET P$="TOPE-PARTIDA= "+STR$ TP+" PUNTOS."
1540 INK 6
1550 LET YY=140
1560 GO SUB 1230
1570 FOR N=1 TO 300: NEXT N
1580 RETURN
1590 DRAW 255,0: DRAW 0,175
1600 DRAW -255,0: DRAW 0,-175
1610 RETURN
1620 FOR N=10 TO 20
1630 PRINT  PAPER 0;AT N,1;"
     "
1640 NEXT N
1650 RETURN
1660 INK 9
1670 PLOT 5,168: DRAW 245,0
1680 DRAW 0,-57
1690 DRAW -245,0: DRAW 0,57
1700 FOR N=1 TO 6
1710 PRINT AT N,1; PAPER 0; OVER 1; INK N+1;"
     "
1720 NEXT N
1730 RETURN
1740 LET P$=" Juega MOCHO "+STR$ T+" "
1750 PAPER T+3: INK 1
1760 LET YS=2: LET XS=2: LET YY=152:
1770 GO SUB 1230
1780 RETURN
1790 FOR Q=8 TO 20 STEP 6
1800 PRINT  PAPER 7; INK 6;AT 13,Q;"4337"
1810 PRINT  PAPER 7; INK 6;AT 14,Q;"5  5"
1820 PRINT  PAPER 7; INK 6;AT 15,Q;"5  5"
1830 PRINT  PAPER 7; INK 6;AT 16,Q;"1332"
1840 NEXT Q
1850 RETURN
1860 FOR D=1 TO 3
1870 LET D1=2*D(D)-1
1880 LET C$=A$(D1 TO D1+1)
1890 LET D$=B$(D1 TO D1+1)
1900 IF D(D)=1 OR D(D)=3 OR D(D)=4 THEN  LET TINTA=0:
 GO TO 1920
1910 LET TINTA=2
1920 LET Q=6*D+2
1930 PRINT  BRIGHT 1; PAPER 7; INK 6;AT 13,Q;"4337"
1940 PRINT  BRIGHT 1; PAPER 7; INK 6;AT 14,Q;"5"; INK
 TINTA;C$; INK 6;"5"
1950 PRINT  BRIGHT 1; PAPER 7; INK 6;AT 15,Q;"5"; INK
 TINTA;D$; INK 6;"5"
1960 PRINT  BRIGHT 1; PAPER 7; INK 6;AT 16,Q;"1332"
1970 NEXT D
1980 RETURN
1990 LET CAM=0
2000 IF D(1)=5 AND D(2)=5 AND D(3)=5 THEN  LET PUNTOS
=PUNTOS+500: GO TO 2080
2010 IF D(1)=6 AND D(2)=6 AND D(3)=6 THEN  LET PUNTOS
=PUNTOS+1000: GO TO 2080
2020 IF D(1)=2 AND D(2)=2 AND D(3)=2 THEN  LET P(T)=0
: LET PUNTOS=0: LET FIN=1: GO TO 2080
2030 FOR D=1 TO 3
2040 IF D(D)=5 THEN  LET PUNTOS=PUNTOS+50
2050 IF D(D)=6 THEN  LET PUNTOS=PUNTOS+100
2060 IF D(D)=1 OR D(D)=2 OR D(D)=3 OR D(D)=4 THEN  LE
T CAM=CAM+1
2070 NEXT D
2080 PRINT  INK 9; PAPER 0;AT 18,6;"PUNT. PARCIALES=
";PUNTOS;"
     "
2090 RETURN
2100 PRINT  OVER 1; FLASH 0; PAPER T+3; INK 9;AT 9,10
*T-9;"
2110 RETURN
2120 LET P$="                "
2130 PAPER 0: GO SUB 1230
2140 RETURN
2150 PRINT  FLASH 0; OVER 1; PAPER T+3;AT 9,10*T-9;"
     "
2160 RETURN
2170 POKE 23606,200
2180 POKE 23607,200
2190 LET P$="-  CALIMOCHO  -"
2200 FLASH 1
2210 PAPER 1
2220 INK 7
2230 GO SUB 1230
2240 FLASH 0
2250 PRINT  PAPER 0;AT 11,10*T-9;"  "; FLASH 1; INK 2
; PAPER 6;" 0000 "; FLASH 0; PAPER 0;"
     "
2260 FOR V=1 TO 15
2270 FOR S=35 TO 40
2280 BEEP .01,S
2290 NEXT S
2300 NEXT V
2310 POKE 23606,0
2320 POKE 23607,60
2330 PRINT  PAPER 0; INK 9;AT 11,10*T-9;"   0000   "
2340 INK 9
2350 RETURN
2360 CLEAR 49999: LOAD "ROTULACION"CODE 51656,768
2370 RUN
```