VIFI ANAYA

Joe Pritchard

Rutinas en Lenguaje Máquina para Amstrad

CPC 464-664-6128

RUTINAS EN LENGUAJE MAQUINA PARA AMSTRAD

Rutinas en lenguaje máquina para Amstrad

Joe Pritchard

ANAYA MULTIMEDIA

MICROINFORMATICA

Título de la obra original:
READY MADE MACHINE LANGUAGE ROUTINES
FOR THE AMSTRAD CPC 464/CPC 664

Traducción: Rafael Sarmiento

Diseño de colección: Antonio Lax

Diseño de cubierta: Narcís Fernández

Edición publicada por acuerdo con
Melbourne House (Publishers) Ltd., Londres

Villafranca, 22. 28028 Madrid
Depósito legal: M. 12.129-1986
ISBN: 84-7614-080-0
Printed in Spain
Imprime: Anzos, S. A. - Fuenlabrada (Madrid)

Indice

Prólogo del traductor

De grandes piedras se hacen los castillos, pero los detalles que los embellecen no son más que guijarros. Un programa en BASIC puede llegar a ser todo lo robusto e "inteligente" que se quiera, pero tal vez no impresione tanto como la velocidad o posibilidades que conlleva el código máquina. No hay nada como juntar ambos lenguajes para obtener programas de primera calidad.

Este libro no pretende enseñarte nada que todavía ignores; sin embargo, es muy posible que te aclare muchos de los secretos de tu Amstrad. Tampoco está escrito para que aprendas las instrucciones del microprocesador Z-80 y su manejo, sino más bien para que conozcas las subrutinas y parte del sistema operativo de tu ordenador. De todas formas estoy seguro que algo te va a enseñar y que algo vas a aprender.

No importa en absoluto que carezcas de conocimientos sobre el lenguaje máquina. La labor de este libro es llevarte más allá de los límites del BASIC y acercarte todo lo posible a las entrañas de tu Amstrad. Por eso el lenguaje máquina no es la meta del libro, sino tan sólo un medio. Difícilmente se encontraba hasta hace poco un libro para el Amstrad que no fuera de inicialización o aprendizaje. La poca literatura que trataba seriamente del tema estaba en lengua extranjera. Es más, allá por el año 1984 en el que adquirí mi Amstrad, no había un solo libro que me enseñara cuáles eran las auténticas posibilidades del aparato.

Se hacía necesario un libro así. Presentando individualmente cada una

de las posibilidades del ordenador, y obteniéndolas con la máxima velocidad. Dejando a la imaginación del usuario la combinación de cuantas rutinas quiera. Además, no está orientado hacia una aplicación específica, por lo que su utilidad es grande frente a todas las posibles exigencias. *Rutinas en lenguaje máquina para Amstrad* es verdaderamente un libro que no se puede soltar una vez abierto. Permite al usuario conocer más a fondo su ordenador, a la vez que resuelve multitud de problemas que resultan inviables desde el BASIC. Y refiriéndome a ti en concreto, observarás la cantidad de aplicaciones que se te ocurren según vayas probando una a una las rutinas aquí presentadas. Espero sinceramente que encuentres en este libro la solución a tus dudas y problemas con el ordenador.

Notas sobre programas ensambladores

Existen algunas diferencias entre los distintos programas ensambladores de código máquina. Unos ofrecen más facilidades, como el editado por Hisoft, y otros son más pequeños, pero igualmente válidos, como el de Honey Fold.

La principal diferencia está en los directivos del ensamblador. Un directivo es una seudo-instrucción que sólo es entendida y efectuada por el propio ensamblador, y que no afecta para nada al programa en código máquina. Los listados de este libro fueron realizados con el programa DEVPAC de Amsoft, y los directivos que puedes encontrar en los listados deben ser interpretados del siguiente modo:

ORG dirección:	Indica al ensamblador la "dirección" de la memoria donde debe almacenar el código máquina. Suele insertarse al comienzo de los programas.
DEFW número:	Reserva 2 bytes de la memoria, guardando en ellos el byte bajo y alto del código binario equivalente al número. Un directivo análogo puede ser WORD.
DEFB número:	Reserva un byte de la memoria, guardando en él el código binario del número que le acompa-

ña. El número debe estar en la gama 0-255. El directivo equivalente a éste, para otros ensambladores es: BYTE.

DEFS número: Aumenta el valor del puntero el número de posiciones de memoria que indique "número". Simplemente reserva bytes de memoria sin darles ningún valor concreto.

DEFM "expresión": Almacena consecutivamente en la memoria los códigos ASCII de cada uno de los caracteres que contenga la "expresión". Sirve para guardar mensajes y texto en general. Otro directivo equivalente es TEXT.

Si no dispones de ningún programa ensamblador, podrás aprender en el capítulo 2 cómo guardar los bytes correspondientes a cada rutina en la memoria. Esos bytes vienen expresados en código hexadecimal en cada una de las rutinas.

Introducción

Una de las más desalentadoras perspectivas que se presentan ante el programador de código máquina es el diseño de pequeñas rutinas para ordenador, tales como escribir cadenas de caracteres, guardar bloques de datos en el cassette o sistema de almacenamiento, o incluso leer caracteres del teclado. Estas rutinas aparecen a menudo en todo tipo de programas, pero el trabajo inicial de escribirlas puede resultar tedioso. Afortunadamente las rutinas presentadas en este libro resuelven el problema. Todas ellas han sido probadas en el Amstrad CPC 464 con sistema de cassette; pero funcionan igualmente bien en el modelo 664 que incorpora unidad de disco, aunque en algunos casos habrá que hacer pequeñas modificaciones. Las rutinas de pantalla han sido diseñadas para su uso sin ventanas definidas; pero si prefieres definirlas también funcionarán sin problemas. En general las rutinas trabajarán correctamente en los tres modos de pantalla, y algunas resultarán de gran utilidad para el programador de BASIC.

Los programas fueron listados utilizando el ensamblador DEVPAC de Amsoft. Quisiera expresar mi agradecimiento a los editores que me propusieron llevar a cabo esta obra y también reconocer la ayuda de Amsoft en la preparación de este libro. Finalmente, me gustaría dedicárselo a mi madre y a mi padre por todos los años de tolerancia especial que me han demostrado.

También quiero agradecer la colaboración de algunos gatos del barrio, que demostraron la posibilidad de programar un microordenador incluso ante el ataque de un gato.

JOE PRITCHARD

1

Lenguaje máquina en el Amstrad

Este libro te ofrecerá una amplia variedad de rutinas en código máquina listas para funcionar; sin embargo, para obtener mayor provecho a la hora de programar, te vendrán bien algunos conocimientos sobre programación en código máquina en el Amstrad y unas cuantas notas sobre la estructura del sistema operativo.

Las rutinas en este libro

Por supuesto, lo primero que hay que hacer es introducir los programas en el ordenador. El mejor método para conseguirlo es usar un programa ensamblador, como el Amstrad DEVPAC publicado por Amsoft. Alternativamente podemos usar la orden POKE para introducir los bytes directamente en memoria. Estos bytes son los que conforman el programa en código máquina. En este libro verás que todos los programas aparecen listados de la siguiente forma:

Dirección en memoria	Código hexadecimal	Listado en ensamblador
9D0B	2A0000	LD HL,0000
9D0B	CDC0BB	CALL #BBC0
9D0E	C9	RET

de modo que puedas usar cualquiera de los dos métodos. El listado en ensamblador puede ser tecleado en un programa ensamblador y los bytes en hexadecimal pueden ser almacenados en memoria mediante POKE por distintas vías, de las cuales la más sencilla es un programa BASIC de este tipo:

```
10 FOR I=0 TO N : REM N=numero de bytes
20 READ A$
30 POKE (direccion+I),VAL("&"+A$)
40 REM  direccion, es donde el codigo se
   almacenara en la memoria.
50 NEXT
60 DATA DD,6E,00,DD,66,01,7E,DD,6E,02,DD
   ,66,03,86,77,C9
```

La sentencia DATA contiene las representaciones hexadecimales de los bytes que conforman el programa. Es obvio que este método es muy útil para combinar rutinas en lenguaje máquina con programas en BASIC, función que se realiza a menudo. El hecho de que los programas se listen con bytes hexadecimales permitirá usar las rutinas del libro a aquellos programadores sin acceso a un ensamblador.

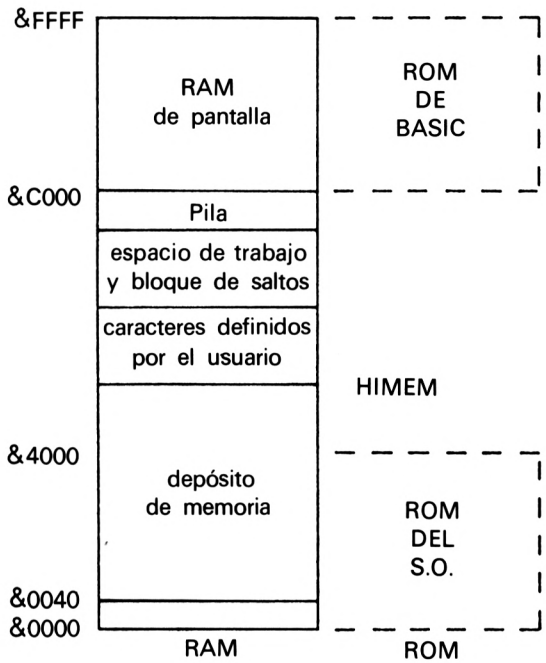
Muchas de las rutinas se escribirán de forma que funcionen en cualquier dirección de memoria disponible por el usuario sin alteración: a estos programas los llamaremos "relocalizables". Otras en cambio requieren una dirección concreta en la memoria para poder funcionar. Tales rutinas serán "no relocalizables", y habrá que realizar algunos cambios para conseguir que funcionen en dirección distinta a la asignada. De cualquier modo detallaré todo aquello que sea necesario para que realices esos cambios. Así podrás seleccionar entre las rutinas listadas en este libro aquellas que precises para una aplicación concreta y también cargarlas en memoria donde quieras.

Sin embargo, ¿en qué posición de la memoria del ordenador puede almacenarse el código máquina? Si ya has realizado pequeños programas en código máquina en el Amstrad, quizá prefieras pasar por alto esta sección y reincorporarte en el estudio de la rutina CALL. Pero si quieres refrescar conocimientos sobre el tema, allá vamos.

Empleo de la memoria en el Amstrad

La zona denominada "depósito de memoria" es el área de memoria utilizada para almacenar nuestros programas en BASIC y sus variables. Todos los programas que escribamos en código máquina deben ser protegidos contra superposiciones accidentales del programa en BASIC, y de-

ben ser preservados de los destrozos del resto del sistema operativo, que usa varias áreas de memoria como espacio de trabajo. La forma de crear esta área protegida de memoria es bastante simple: "capturamos" parte del depósito de memoria y vedamos la entrada al BASIC.



Mapa simplificado de la memoria del Amstrad

Al último byte de RAM disponible para el BASIC se le asigna un nombre especial —HIMEM— y la orden PRINT HIMEM devolverá siempre la dirección del último byte de memoria disponible para BASIC en cada momento. Poniendo en marcha el sistema, la dirección es 43903 o &AB7F. En estas circunstancias el byte de la dirección 43904 es parte de la definición del carácter número 240, que es el primer carácter definible por el usuario de que dispone el programador al conectar el ordenador.

Nos "adueñamos" de parte de la memoria disponible normalmente por el BASIC decrementando en la memoria el valor de HIMEM hacia la dirección &0000. Esto tiene por efecto la creación de un espacio entre HIMEM y el primer byte de la zona de caracteres definibles por el usuario. Desplazamos el valor de HIMEM usando la orden MEMORY. Por ejemplo:

```
MEMORY 39999
```

localizará a HIMEM en la dirección 39999, permitiéndonos usar el espacio situado entre las direcciones 40000 y 43903, ambos inclusive, para las

rutinas en lenguaje máquina. Esto es suficiente para la mayoría de las aplicaciones. Habrá que tener en cuenta un par de cuestiones sobre el uso de la orden MEMORY. La primera es el uso aconsejable de SYMBOL AFTER para reservar espacio a tantos caracteres definidos por el usuario como requiera tu programa, antes de usar MEMORY para variar el valor de HIMEM. La segunda es que SYMBOL AFTER no funcionará correctamente mientras algún fichero del sistema de cassette permanezca abierto. Esto es así porque la apertura de un fichero produce una variación en el valor de HIMEM de forma que SYMBOL AFTER no lo hallará donde esperaba.

Por eso, usando MEMORY, podemos reservar una zona a salvo para nuestros programas en código máquina en la que almacenar mediante POKE los bytes que constituyen los programas, tal y como mencioné anteriormente en este capítulo. Ahora hemos de conseguir que actúe el código máquina. Esto lo logramos a través de la sentencia CALL.

Acceso a los programas en código máquina

Los programadores del Amstrad tenéis suerte al contar con la sentencia CALL; la mayoría de los ordenadores caseros tienen una escasa comunicación entre el BASIC y el lenguaje máquina. La sentencia CALL del Amstrad está muy mal documentada en la *Guía del usuario del Amstrad CPC 464*, pero en realidad es una orden muy versátil y polifacética. Como la usaremos continuamente a lo largo del libro para llamar a nuestras rutinas, vamos a examinarla con cierto detalle.

Puedes utilizar la instrucción de dos formas distintas. He aquí un ejemplo de ambas formas de operar:

```
CALL  &BD19

CALL  40000,A%,B%,C%
```

La primera de ellas simplemente provoca la ejecución de la rutina en lenguaje máquina localizada en la dirección &BD19 de la memoria RAM del Amstrad. La segunda sentencia produce la ejecución de la rutina de la dirección 40000, pero además hace que la rutina de lenguaje máquina disponga del valor actual de las variables A%, B% y C% del BASIC. Comprobarás que esto es muy útil. Las variables A%, B% y C% se denominan PARAMETROS de esta llamada CALL. Ahora que conocemos este uso más técnico de la sentencia, podemos conseguir una interacción directa de los números y cadenas de variables entre el BASIC y nuestras rutinas en código máquina.

Así pues, vamos a profundizar en la llamada CALL y sus parámetros. En este libro consideraremos únicamente las rutinas que usan números enteros y cadenas; la experiencia ha demostrado que la mayoría de las

aplicaciones que trabajan con una matemática compleja y números reales se ejecuta mejor en BASIC. Esto se debe en parte a que la realización de los programas en código máquina lleva más tiempo, y generalmente no son tan eficientes como las rutinas presentes en la ROM de BASIC para el cálculo de operaciones aritméticas complejas.

Se distinguen tres amplias clases de parámetros que pueden ser transferidos como parte de una sentencia CALL, y una sentencia puede contener tantos parámetros como puedas incluir en una línea (limitada a 255 caracteres). Claro está que los parámetros de una determinada sentencia CALL pueden pertenecer a cualquiera de las tres clases. Los tipos de parámetros son:

1. Un número, como 100, 2 ó 1000; un nombre de una variable entera, como A%; o una expresión que evaluada dé como resultado un entero. El valor transferido debe pertenecer al intervalo (0, 65535), aunque, como veremos más tarde, hay un par de puntos a tratar con precaución. Si una variable entera actúa como parámetro, como A%, y al ser incluida en la sentencia CALL no se le asignó un valor previo, entonces la variable será considerada como portadora del valor 0. Un ejemplo de este tipo de sentencia CALL sería

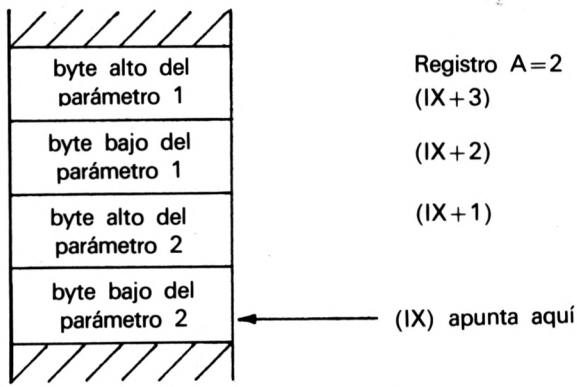
   ```
   CALL  40000,A%
   ```

   donde 40000 es la dirección de la rutina llamada, y A% es el parámetro.
2. El nombre de una variable entera precedido del carácter "@" como @A%. Este método de transferir un parámetro numérico a una rutina de código máquina también permite una transferencia de información en los dos sentidos, hacia y desde el programa en lenguaje máquina, como pronto vamos a ver.
3. Una variable literal precedido por "@". Esta es la única forma de transferir cadenas de caracteres a rutinas de código máquina. No podemos pasar cadenas constantes como "José" a una rutina, sino sólo las variables.

Ahora, ¿cómo podemos acceder a los parámetros transferidos a los programas en código máquina? Al comienzo de la rutina en la dirección llamada, dos de los registros de la CPU han sido modificados por el intérprete de BASIC para ayudarte a acceder a los parámetros transferidos. El registro A contiene el número de estos parámetros, y el registro índice IX apunta a un área de memoria llamada bloque de parámetros, que utiliza dos bytes para cada parámetro transferido a la rutina de código máquina. IX apunta a los parámetros del modo indicado en la figura; el contenido exacto de cada dos bytes de entrada dependerá del tipo de parámetro.

```
CALL  40000,para1,para2
```

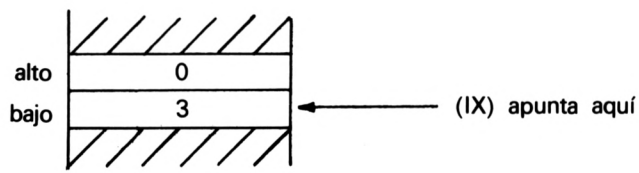
generará:



Examinaremos ahora el contenido de lo que cabe encontrar en cada bloque de parámetros de entrada para cada tipo de parámetro.

Números o variables enteras

Aquí se incluyen elementos como A% o 5, y el bloque de parámetros de entrada contendrá una representación del número en binario de dos bytes. Así, para CALL 40000,3 el bloque de entrada sería:



Una pequeña cuestión a tener en cuenta: en las dos llamadas

```
CALL  40000,-1
CALL  40000,65535
```

se entenderá como bloque de parámetros de entrada &FFFF, ya que ésta es la representación en complemento a dos de −1; que es el sistema que utiliza el Amstrad para almacenar internamente los enteros negativos. Así, si intentamos transferir un valor como 65535 a una rutina en código máquina dentro de una variable entera, como

```
A%=65535:CALL  40000,A%
```

se producirá un error de desbordamiento, porque las variables enteras sólo pueden tomar valores dentro del intervalo −32768 a 32767. Resulta bastante fácil introducir en un registro o un par de registros los valores contenidos en el bloque de parámetros.

```
LD L,(IX+0) ; Byte bajo.
LD H,(IX+1) ; Byte alto.
```

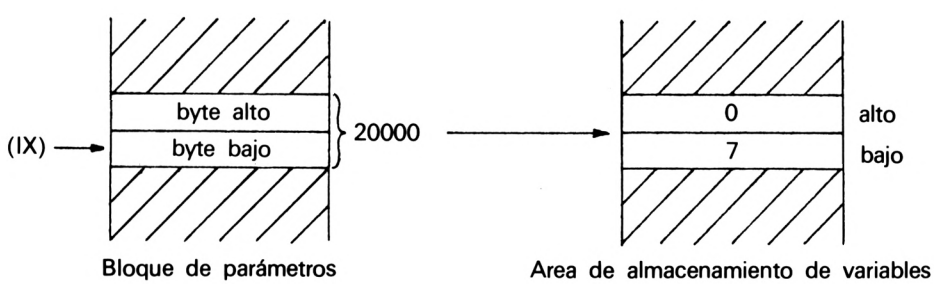
Variables precedidas de @

Hasta aquí, la comunicación entre el BASIC y el lenguaje máquina ha sido tratada en un solo sentido: hemos aprendido a transferir valores desde el BASIC al código máquina, pero no a la inversa. La utilización de @ nos permitirá hacerlo. El bloque de parámetros de entrada para un parámetro de este tipo no será ahora el valor de la variable, como en el caso de A% o 5, sino la dirección de la variable; esto es, el lugar donde se ha almacenado la variable en la memoria de BASIC.

Por ejemplo:

```
CALL  40000,@H%
```

generará un bloque de parámetros que contendrá la dirección en RAM de la variable H%. En cuanto al argumento, supongamos que los dos bytes del bloque de parámetros de entrada contienen el valor 20000. Esto indicaría que el byte bajo de H% está almacenado en la dirección 20000 y que el byte alto de H% está almacenado en la dirección 20001. Esquemáticamente, esto se puede mostrar así:



La utilidad de todo esto, como verás, es que, alterando los valores contenidos en los dos bytes utilizados para almacenar el valor de H%, estaremós alterando su valor desde nuestro programa en lenguaje máquina. Así en este ejemplo que asigna el valor 7 a H%, cargaríamos 7 en la posición 2000 y 0 en la posición 20001. Como pequeña demostración,

examina el siguiente programa, al que accederíamos mediante CALL 40000,@A%,n.

```
40000
DD7E00      LD   A,(IX+0) ; Introduce el valor de n en el registro A.
DD6E02      LD   L,(IX+2) ; Toma la direccion de la variable entera
DD6603      LD   H,(IX+3) ; y la almacena en el par HL .
77          LD   (HL),A ; Guarda en esa direccion el valor que
C9          RET  ; contenia A y regresa al BASIC.
```

A% tendría que ser inicializada a 0 antes de hacer la llamada (CALL). Como el programa afecta únicamente al byte bajo de la variable en la memoria, n tendrá que tener un valor comprendido entre 0 y 255. Por supuesto que cualquier nombre de variable entera puede ser usado y no sólo A%. Así:

```
jose%=0:CALL 40000,@jose%,6
paco%=0:CALL 40000,@paco%,67
```

son ambas expresiones correctas. En el primer caso, jose% tomará el valor 6 después de la orden CALL, y en el segundo caso, paco% contendrá el valor 67. n puede ser también una variable, siempre que no tenga el prefijo @.

Una última cuestión sobre el uso de @. Este prefijo permite al usuario acceder a la dirección de cualquier variable determinada, siempre y cuando la variable en cuestión haya sido previamente utilizada, incluso si se ha inicializado con el valor 0. Si intentas hacer una llamada mediante la orden CALL, usando @ con una variable aún no inicializada, el Amstrad te responderá con un mensaje de error. La explicación es obvia: si una variable aún no ha sido utilizada, el intérprete de BASIC no tendrá ninguna dirección en la que pueda almacenar el bloque de parámetros. Es decir, aún no conoce esa dirección. Cada vez que inicialices una variable, el BASIC creará un espacio en memoria para cada una, asignándole a esa dirección el nombre de la variable que hayas creado. Este espacio será preservado, ya estés trabajando en código máquina o en BASIC.

Transferencia de cadenas

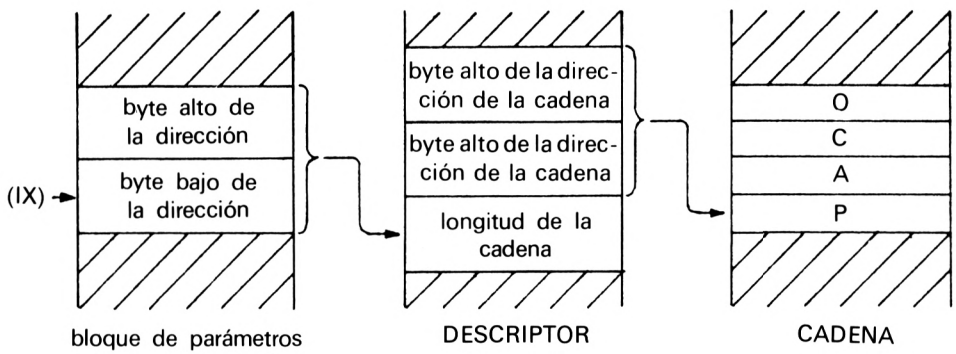
Aunque toda constante numérica, como "1" ó "4000", puede ser transferida a los programas en código máquina, aún no podemos pasar cadenas constantes como "Hola" o "Amstrad" a una rutina en código máquina. Para transferir cadenas literales debes asignarlas a variables literales como M$, PEZ$, etc. Como en el caso anterior, estas variables deben estar precedidas por el carácter @.

De nuevo habrá que inicializar la variable literal antes de ser utilizada en una sentencia CALL, permitiendo así al intérprete de BASIC trabajar

en la memoria, allí donde se encuentre almacenada la cadena. Una sentencia que contiene una variable literal es, por ejemplo:

```
A$="PACO":CALL 40000,@A$
```

Los dos bytes del bloque de parámetros de entrada para este tipo de parámetros serán interpretados de la siguiente forma. Estos apuntan hacia un área de la memoria llamada BLOQUE DESCRIPTOR de la cadena, el cual nos suministra información útil sobre la cadena literal.



El descriptor nos señala la longitud de la cadena y su paradero en la memoria.

Es cierto que también se puede usar la orden POKE para alojar en ciertas posiciones de memoria los valores de la cadena, y después hacer que tus rutinas de código máquina accedan a ellas. Y la orden PEEK puede ser utilizada para extraer desde el BASIC los valores almacenados por el lenguaje máquina. Sin embargo, CALL nos proporciona un método elegante y eficiente para todos los casos, incluso los más sencillos, de ahí su uso frecuente en este libro.

A continuación, y antes de meternos con el uso de las llamadas al sistema operativo de la ROM, veremos algunas indicaciones sobre la utilización de las rutinas en lenguaje máquina del Amstrad.

1. Utilización de los registros alternativos:

Bajo ninguna circunstancia deberás usar los registros alternativos del Z-80. Todos ellos son utilizados por el sistema operativo con diferentes propósitos, y dado que el SO puede requerir alguno de ellos en cualquier momento, podría ser fatal intentar alterar el contenido de cualquiera de estos registros.

2. Superposición de la RAM sobre la ROM:

Gracias a una inteligente artimaña del diseñador, la ROM que contiene al sistema operativo ha sido superpuesta a la RAM que puede ser utiliza-

da para almacenar el BASIC o los programas en código máquina. Asimismo, la ROM que contiene el BASIC está superpuesta a la memoria de pantalla. En circunstancias normales, la RAM será accesible al programador; si quieres leer directamente en la ROM tendrás que hacerlo con cierta habilidad.

3. Estado de los registros en la CPU:

A menudo las rutinas de la ROM utilizadas tienden a dejar algunos registros alterados, y en muchos casos el registro de *flags* también se altera. Por ello, si quieres conservar algún registro, deberás almacenarlo en la pila mediante la orden PUSH antes de llamar a las rutinas de la ROM.

4. La orden CALL:

Al trabajar con rutinas en código máquina se suele aconsejar la preservación de los registros antes de retornar al BASIC desde una rutina llamada mediante la orden CALL. Sin embargo, he comprobado que la máquina recupera todo muy bien sin necesidad de todo esto, cuando se utiliza la orden CALL.

Utilización de llamadas a la ROM

Utilizaremos rutinas de la ROM del Amstrad para aquellas funciones más comunes, tales como escribir en la pantalla o leer un carácter del teclado. Todas estas rutinas son accesibles llamando a direcciones concretas de la memoria RAM, mediante el uso de la instrucción CALL del Z-80 que manejamos desde los programas en código máquina. Al conectar o reinicializar el aparato, el sistema operativo se encarga de preparar estas direcciones. Todas estas rutinas se encuentran en un área de la memoria llamada BLOQUE DE SALTOS, y todas las llamadas a las rutinas de la ROM deberán pasar previamente por el mismo, debido a las razones que paso a exponer.

La primera razón, que examinaremos después con más detalle, es que la ROM se encuentra normalmente inhibida, y, por ello, para usar cualquier rutina de la ROM, debemos primero habilitarla. Esto lo realiza automáticamente el bloque de saltos con cada rutina en particular. La segunda es que las direcciones de este bloque de saltos están garantizadas por Amstrad, sin importar cuántas versiones diferentes haya del sistema operativo. Así, una llamada a la dirección &BB5A siempre cederá el control a la rutina del SO que imprime un carácter en pantalla, sea cual sea la versión del SO que gobierne el aparato. Esto convierte al Amstrad en una máquina a "prueba del futuro", asegurando que las posibles alteraciones del sistema operativo no invalidarán programas ya escritos en las primeras versiones del SO.

La tercera y última razón es que podemos alterar el bloque de saltos para una rutina en concreto del sistema operativo, y variar con ello el comportamiento del mismo en determinados casos.

El bloque de saltos

Ocupa un área bastante grande de la RAM, justo encima del depósito de memoria, y permite acceder a todas las rutinas importantes del sistema operativo y a algunas de las rutinas de la ROM que contiene el BASIC. A continuación se muestra una entrada típica del bloque de saltos.

```
DIRECCION EN MEMORIA        VALOR
&BB18                       &CF
&BB19                       &56        Byte bajo de la direccion.
&BB1A                       &9B        Byte alto de la direccion.
```

&CF es un código un tanto especial en el Amstrad; descrito con sencillez, &CF especifica un salto a una rutina contenida en la ROM en la dirección cuyo código viene dado por los dos bytes que le acompañan. Los dos bytes de dirección, en este ejemplo &9B56, son codificados del siguiente modo:

Bit 15. Este bit de la dirección controla la ROM del BASIC. Si se encuentra activado ("1"), entonces la ROM de BASIC permanecerá inhibida. Si se encuentra en el estado 0, entonces la ROM de BASIC es habilitada.

Bit 14. Este bit de la dirección controla la ROM del sistema operativo. Igual que antes, si su estado es "1", la ROM del SO estará inhibida, y si su estado es 0, la ROM del SO podrá ser utilizada.

Los bits 0 a 13 de la dirección especifican la dirección de la rutina contenida en la ROM seleccionada por los dos bits de mayor peso antes mencionados. Concretando el ejemplo, escribamos el valor de &9B56 en binario.

```
1 0 0 1   1 0 1 1   0 1 0 1   0 1 1 0

Bit 15                          Bit 0
```

En este caso, el bit 15 está puesto a 1, por lo que inhibirá la ROM que contiene el BASIC. El bit 14 está puesto a 0, lo que quiere decir que la rutina se encuentra en la ROM del sistema operativo, que permanece habilitada gracias a este bit. La dirección contenida en los bits 0 al 13 es &1B56, siendo precisamente la dirección de la rutina contenida en el SO que será llamada normalmente con una orden CALL dirigida al bloque de

saltos en &BB18. El byte &CF no es el código de una instrucción propiamente dicha del Z-80; más bien es una "seudo-operación" implementada por Amstrad para esta particular aplicación.

Veremos más adelante cómo alterar el bloque de saltos para aumentar o sustituir las funciones normales del sistema operativo. Si haces esto, el &C3, código de JP, remplazará a &CF, y los dos bytes siguientes compondrán la dirección en RAM de la rutina que hayas escrito. Así, una entrada en el bloque de saltos como

```
&C3
&00
&A0
```

provocará un salto a la rutina en la dirección &A000. La rutina podrá terminar entonces de dos formas diferentes: la primera es con una instrucción RET que hará que la rutina que has añadido sustituya totalmente a la función normal del sistema operativo; la segunda es terminar tu programa con el contenido original del bloque de saltos, comenzando con el byte &CF. Observa que parchear de esta forma el sistema operativo puede causar problemas con posteriores versiones del mismo. En este último caso, primero se ejecutará tu rutina y posteriormente el control pasará a la rutina del SO.

¡Y esto es todo! El resto del libro contiene aquello que te ha animado a adquirirlo: rutinas en código máquina preparadas y documentadas para el Amstrad. Espero que experimentes a fondo con las rutinas; posiblemente las desarrolles y perfecciones consiguiendo incluso mayor versatilidad.

2
Rutinas de salida de textos

En BASIC la salida de textos es bastante fácil, pues contamos con la orden PRINT que realiza todo el trabajo. Pero esto no es así en código máquina, por lo que tendremos que encadenar varias rutinas para conseguir algo tan corriente como escribir en la pantalla mensajes y números. Veremos algunas de ellas en este capítulo.

La primera se llama ECADEN, que significa Escribe CADENa, y nos permite escribir mensajes en la pantalla.

ECADEN (Escribe CADENa)

Escribe una cadena de caracteres en el cauce *(stream)* seleccionado y en la posición de pantalla especificada. El color será el asignado a los textos. Los códigos de control se imprimen en pantalla, pero no tienen efecto. La rutina es relocalizable.

Requisitos de entrada:	B será la coordenada X. C será la coordenada Y. IX apunta a la cadena en memoria. La cadena debe terminar con un CHR$(0).
Condiciones de salida:	Todo alterado.
Longitud:	37 bytes.

```
                    10 ;          ** ECADEN **
2518  DDE5          20          PUSH IX
251A  CD54BB        30          CALL #BB54;    Permite la salida de caracteres
                    35 ;                       por el cauce ( pantalla )
251D  3E1F          40          LD   A,#1F
251F  CD5ABB        50          CALL #BB5A;    Posiciona el cursor de textos
2522  78            60          LD   A,B;      mediante CHR$(31)
2523  CD5ABB        70          CALL #BB5A
2526  79            80          LD   A,C
2527  CD5ABB        90          CALL #BB5A
252A  DDE1          100         POP  IX
252C  DD7E00        110 BUCLE   LD   A,(IX+0);Extrae caracter
252F  FE00          120         CP   00;       Es cero ?
2531  C8            130         RET  Z;        Si asi es, termina
2532  DDE5          140         PUSH IX
2534  CD5DBB        150         CALL #BB5D;    Imprime el caracter
2537  DDE1          160         POP  IX
2539  DD23          170         INC  IX;       Apunta hacia el siguiente caracter
253B  18EF          180         JR   BUCLE;    Repite el ciclo
```

Comentarios

Si una frase o un mensaje es demasiado largo y se sale de la línea en que se inició, continuará en la siguiente línea.

El mejor modo de ver este programa en funcionamiento es teclear el programa de muestra que se lista a continuación. Se añaden algunas instrucciones en código máquina, pero sólo para inicializar los registros antes de llamar a la rutina ECADEN. Estas instrucciones adicionales son:

```
LD  IX,#9C40     ; Direccion de la cadena
LD  B,nn         ; Coordenada X introducida con POKE
LD  C,nn         ; Coordenada Y introducida con POKE
```

Cuando se llama a la rutina ECADEN los valores de los registros B y C deben ser apropiados para el modo de pantalla que se esté utilizando en ese momento. El programa en BASIC es:

```
10 REM Demostracion de ECADEN
20 MEMORY 39999
30 WINDOW #1,1,20,20,25
40 CLS
50 GOSUB 1000
60 LOCATE #1,1,1
70 INPUT #1,"Posicion de X";x
80 INPUT #1,"Posicion de Y";y
90 INPUT #1,"Cadena :";a$
100 FOR I=1 TO LEN(a$)
110 POKE (39999+I),ASC(MID$(a$,I))
120 NEXT
130 POKE (39999+I),0: REM  Finaliza la
    cadena en memoria
140 POKE 40205,x:POKE 40207,y  : REM
```

```
    Introduce las coordenadas X e Y
150 CALL 40200
160 CLS #1
170 GOTO 60
1000 REM Rutina que efectua el POKE del
     codigo maquina
1010 FOR I=0 TO 44
1020 READ a$:POKE (40200+I),VAL("&"+a$)
1030 NEXT
1040 RETURN
1050 DATA DD,21,40,9C,06,00,0E,00  :
     REM Inicializacion de los registros
1060 DATA DD,E5,CD,54,BB,3E,1F,CD,5A,BB,
     78,CD,5A,BB,79,CD,5A,BB,DD,E1,DD,7E
     ,00,FE,00,C8,DD,E5,CD,5D,BB,DD,E1,
     DD,23,18,EF
```

La ejecución de este programa te permitirá situar en la pantalla una cadena desde el código máquina.

Te habrás dado cuenta de que en la rutina antes expuesta usamos un código de control, en este caso el código ASCII 31, para situar el texto en la pantalla. La guía del usuario de Amstrad muestra los distintos códigos de control disponibles, y explica sus efectos. Examinaremos ahora una rutina corta que nos permite usar estos distintos códigos de control desde nuestros programas, de forma que podamos aprovechar a fondo las facilidades que ofrece el ordenador Amstrad.

Por cierto que la lista podrás encontrarla en las páginas 2 a 5 del capítulo 9 de la guía del usuario. Una rápida ojeada al mismo te revelará muchos códigos de utilidad. Además también verás que cada código de control tiene un carácter imprimible asociado al mismo, siendo este carácter el que aparece en la pantalla con la rutina ECADEN. La siguiente rutina que estudiaremos, ECONTR, no imprime nada en la pantalla, sino que EJECUTA los códigos de control; de esta forma, al transferir CHR$(7) a ECONTR, se generará un pitido en vez de imprimirse el carácter asignado a este código que sería un pequeño "invasor del espacio".

ECONTR (Envía código CONTRol)

Envía una cadena de códigos de control a la VDU de textos. La rutina es relocalizable.

Requisitos de entrada: B contiene el número de caracteres.
IX apunta a un bloque de memoria que contiene los códigos. El código del carácter que se encuen-

tre en la dirección a la que apunta IX será el primero en ser enviado.

Condiciones de salida: AF, BC e IX son alterados.

Longitud: 14 bytes.

```
                    10 ;       ** ECONTR **
23B1  CD54BB        20         CALL #BB54 ; Activa la VDU de texto
23B4  DD7E00        30 BUCLE   LD   A,(IX+0)
23B7  CD5ABB        40         CALL #BB5A ; Envia el codigo de control
23BA  DD23          50         INC  IX
23BC  10F6          60         DJNZ BUCLE ; Todo realizado ?
23BE  C9            70         RET  ; Si.
```

A modo de demostración en cuanto a su uso, el siguiente ejemplo muestra cómo usar ECONTR para definir un carácter —, equivalente en código máquina a la orden SYMBOL. En la memoria se ha dispuesto un bloque de bytes que contiene los datos para el carácter. Así, para SYMBOL 240, 1, 3, 7, 15, 31, 63, 127, 255, tendríamos que colocar el siguiente bloque:

```
n+9    255   ultimo byte de la definicion
       127
       63
       31
       15
       7
       3
       1
n+1    240   caracter a definir
n+0    25    codigo de control de SYMBOL
```

ECONTR

```
10 MEMORY 39999
20 PRINT CHR$(240)
30 GOSUB 90
40 RESTORE 150
50 FOR I=0 TO 9: READ A:POKE (40000+I),A
   : NEXT
60 CALL 40200
70 PRINT CHR$(240)
80 END
90 FOR I=0 TO 19
100 READ A$: POKE (40200+I),VAL("&"+A$)

110 NEXT
120 DATA 06,0A,DD,21,40,9C
130 DATA CD,54,BB,DD,7E,00,CD,5A,BB,DD,
```

```
        23,10,F5,C9
140 RETURN
150 DATA 25,240,1,3,7,15,31,63,127,255
```

Supongamos que n=41000.

El código de máquina para llamar a ECONTR será simplemente

```
LD IX,41000
LD B,10
CALL ECONTR
RET
```

Si examinas la lista de los códigos de control aprenderás a usar ECONTR para cambiar el color de la pluma y del papel del texto, y tendrás la oportunidad de cambiar el color usando ECONTR en combinación con ECADEN.

Hasta aquí sólo hemos visto cómo situar una cadena de textos en la posición del cursor de texto. Pero, ¿qué ocurre con el código máquina equivalente a la orden TAG del BASIC? GCADEN te permite situar el texto en el cursor de gráficos, y también especificar los puntos de pantalla que median entre los caracteres que constituyen la cadena.

GCAD2 es una versión ampliada de GCADEN que te permite usar una orden CALL ampliada para poder utilizar esta rutina desde el BASIC con mayor facilidad.

GCADEN (CADENa en Gráficos)

Imprime una cadena en el color asignado a gráficos y en las coordenadas X e Y especificadas. El espacio entre caracteres escritos también puede predeterminarse. Esta rutina es relocalizable.

Requisitos de entrada: BC contiene el número de puntos de pantalla entre caracteres.
HL coordenada Y.
DE coordenada X.
IX apunta a la cadena en memoria.

La cadena debe finalizar con CHR$(0).

Condiciones de salida: Todos los registros alterados.

Longitud: 37 bytes.

```
                10 ;       ** GCADEN **
2702  C5        20 GCADEN PUSH BC;    Guarda el valor de los
                30 ;                  registros en la pila
```

```
2703  D5          40        PUSH DE
2704  E5          50        PUSH HL
2705  CDC0BB      60        CALL #BBC0; Posiciona el cursor de texto
                  70 ;                  en las coordenadas X,Y
2708  E1          80        POP  HL;    Vuelve a cargar los registros
2709  D1          90        POP  DE
270A  C1         100        POP  BC
270B  DD7E00     110        LD   A,(IX+0); Examina el caracter
270E  FE00       120        CP   00;    Si es cero, termina
2710  C8         130        RET  Z
2711  C5         140        PUSH BC
2712  D5         150        PUSH DE
2713  E5         160        PUSH HL
2714  CDFCBB     170        CALL #BBFC; Imprime el caracter en donde se
                  180 ;                 encuentre el cursor de graficos
2717  E1         190        POP  HL
2718  D1         200        POP  DE
2719  C1         210        POP  BC
271A  E5         220        PUSH HL;    Guarda en la pila la coordenada Y
271B  D5         230        PUSH DE;    Almacena la coordenada X
                  240 ;                 en el registro HL
271C  E1         250        POP  HL
271D  AF         260        XOR  A;     Pone a cero el acarreo
271E  ED4A       270        ADC  HL,BC; Actualiza la coordenada X
2720  E5         280        PUSH HL;    Retorno de X
2721  D1         290        POP  DE;    al registro DE
2722  E1         300        POP  HL;    Rescata de la pila el valor de Y
2723  DD23       310        INC  IX;    Apunta a la direccion
                  320 ;                 del siguiente caracter
2725  18DB       330        JR   GCADEN;Vuelve a comenzar
```

```
GCADEN

10 MODE 1
20 MEMORY 39999
30 CLS
40 GOSUB 1000
50 LOCATE 1,20
60 INPUT "CADENA :",A$
70 FOR I=1 TO LEN(A$)
80 POKE (39999+I),ASC(MID$(A$,I))
90 NEXT
100 POKE (39999+I),0
110 CALL 40200,
120 GOTO 50
1000 FOR I=0 TO 49
1010 READ A$
1020 POKE (40200+I),VAL("&"+A$)
1030 NEXT
1040 RETURN
1050 DATA DD,21,40,9C,11,C8,0,21,C8,0,1,
     10,00
1060 DATA C5,D5,E5,CD,C0,BB,E1,D1,C1,DD,
     7E,00,FE,00,C8,C5,D5,E5,CD,FC,BB,E1
1070 DATA D1,C1,E5,D5,E1,AF,ED,4A,E5,D1,
     E1,DD,23,18
1080 DATA DB
```

Comentarios

En esta rutina, todo texto que exceda del borde derecho de la pantalla se perderá. El valor de BC puede variarse según tus necesidades, pero los valores siguientes te ofrecerán un buen punto de partida:

```
Modo 0      BC=32
Modo 1      BC=16
Modo 2      BC=8
```

Es obvio que los valores almacenados en DE y HL como coordenadas X e Y también dependen del modo de pantalla que se esté utilizando. El programa que a continuación expongo, GCAD2, es un método que te permite experimentar más fácilmente con esta rutina, pues contiene las instrucciones precisas para obtener los parámetros desde una sentencia CALL, en BASIC.

GCAD2 (CADena en Gráficos 2)

A esta rutina se accede mediante

```
CALL direccion,x%,y%,b%,@a$
```

donde x% es la coordenada X, y% es la coordenada Y, b% es el espacio entre caracteres, y a$ representa la cadena que se va a imprimir. Por supuesto, los tres caracteres numéricos también pueden ser constantes; "dirección" es la dirección donde está almacenada la rutina.

En vez de repetir el listado de GCADEN, listaré las instrucciones extra que se precisan para aislar los parámetros, y luego proporcionaré un programa en BASIC para probar la rutina. GCAD2 tiene 71 bytes de longitud, incluyendo GCADEN.

```
                    10 ;       ** GCAD2 **
259D  FE04          20         CP   4;          Comprueba que hay 4 parametros
259F  C0            30         RET  NZ;         Si no los hay, regresa al
                    40 ;                        programa principal
25A0  DD6E00        50         LD   L,(IX+0);   Toma la direccion de
25A3  DD6601        60         LD   H,(IX+1);   la cadena del descriptor
25A6  23            70         INC  HL;
25A7  4E            80         LD   C,(HL)
25A8  23            90         INC  HL
25A9  46           100         LD   B,(HL);     Almacena esa direccion en BC
25AA  C5           110         PUSH BC
25AB  DD4E02       120         LD   C,(IX+2)
25AE  DD4603       130         LD   B,(IX+3);   Ahora el par BC guarda
                   140 ;                        el espaciamiento de caracteres
25B1  DD6E04       150         LD   L,(IX+4)
25B4  DD6605       160         LD   H,(IX+5);   Coordenada Y en el par
                   170 ;                        de registros HL
25B7  DD5E06       180         LD   E,(IX+6)
25BA  DD5607       190         LD   D,(IX+7);   Coordenada X en el par DE
25BD  DDE1         200         POP  IX;         Direccion de la cadena en IX
```

El resto del programa es igual que GCADEN. Observa que, al igual que en GCADEN, la cadena de caracteres que se va a escribir debe terminar con CHR$(0). El programa en BASIC expuesto a continuación demuestra el funcionamiento de GCAD2.

```
10 MODE 2
20 MEMORY 39999
30 GOSUB 1000: REM Inserta en memoria el
   codigo maquina
40 LOCATE 1,22
50 INPUT "Cadena ";A$
60 A$=A$+CHR$(0):REM Incluye el caracter
   final
70 INPUT "Coordenada X :",X%
80 INPUT "Coordenada Y :",Y%
90 INPUT "Espacio entre caracteres :";B%
100 CALL 40200,X%,Y%,B%,@A$:REM
      Llama a la rutina
110 GOTO 40
1000 REM Introduccion de los bytes
     mediante POKE . Observa  que  si
     cambias la direccion, tambien has
     de cambiarla en la linea 100
1010 FOR I=0 TO 70
1020 READ A$: POKE (40200+I),VAL("&"+A$)
1030 NEXT
1040 DATA FE,04,C0,DD,6E,00,DD,66,01,23,
     4E,23,46,C5,DD,4E,02,DD,46,03,DD,6E
     ,04,DD,66,05,DD,5E,06,DD,56,07,DD,
     E1
1050 DATA C5,D5,E5,CD,C0,BB,E1,D1,C1,DD,
     7E,00,FE,00,C8,C5,D5,E5,CD,FC,BB,E1
     ,D1,C1,E5,D5,E1,AF,ED,4A,E5,D1,E1,
     DD,23,18,DB
1060 RETURN
```

Aunque veremos con más detalle las operaciones con gráficos en el capítulo siguiente, no estará de más aquí un pequeño receso para explicar cómo puede cambiarse el color de la pluma de gráficos. Lo menciono aquí porque el texto escrito a través de GCADEN o GCAD2 se imprimirá en el color actual de los gráficos. Una rutina de ROM llamada GRAPLUMA te permitirá fijar el color de la pluma de gráficos.

GRAPLUMA (PLUMA de GRAficos)

Cambia el color de la pluma de gráficos.

Requisitos de entrada: A es el color de pluma elegido.

Condiciones de salida: AF alterado.

No tienes más que acceder a la dirección &BBDE mediante CALL. Así,

```
LD      A,1
CALL    #BBDE
RET
```

fijará el color de gráficos con la PLUMA 1.

Impresión de números

Es muy fácil imprimir números en BASIC; no hay más que usar PRINT. Sin embargo, no existe una orden similar en código máquina, por lo que tienes que escribir tu propia rutina para resolver el problema. Las rutinas que escriben el contenido de los registros de la CPU bajo forma numérica pueden resultar muy útiles, tanto para un programa de utilidad como para exponer un simple marcador en un programa de juegos. Concluiremos este capítulo sobre salida de textos examinando una variedad de rutinas que realizan las siguientes funciones:

1. Escribir el contenido del registro A como número binario, hexadecimal o decimal.
2. Escribir el contenido del par de registros HL como número binario, hexadecimal o decimal.

Vamos a comenzar con la impresión de números en binario.

EBINA (Escribe el registro A en BINario)

Escribe el contenido del registro A como un número binario de ocho dígitos en la pantalla, en la posición y el color actual del texto. La rutina es relocalizable.

Requisitos de entrada: A contiene el número que se va a escribir.

Condiciones de salida: AF, BC alterados.

Longitud: 26 bytes.

```
                10 ;       ** EBINA **
256D  4F        20 CBINA   LD   C,A    ; Guarda una copia de A
256E  0608      30         LD   B,8    ; Ocho bits en el registro A
2570  CB21      40 BUCLE   SLA  C      ; Desplaza un bit a la izquierda
                50 ;                     quedando en el acarreo el bit
                60 ;                     de mayor peso
2572  3809      70         JR   C,CERO
2574  3E30      80         LD   A,#30 ; Es el codigo ASCII para '0'
2576  C5        90         PUSH BC
2577  CD5ABB   100         CALL #BB5A ; Imprime un '0'
257A  C1       110         POP  BC
257B  1807     120         JR   OUT
257D  3E31     130 CERO    LD   A,#31 ; Codigo ASCII para '1'
257F  C5       140         PUSH BC
2580  CD5ABB   150         CALL #BB5A ; Lo imprime
2583  C1       160         POP  BC
2584  10EA     170 OUT     DJNZ BUCLE ; Se han desplazado todos los bits ?
2586  C9       180         RET  ;       Al terminar, regresa al programa
                190 ;                     principal
```

Puedes probar esta rutina con el siguiente programa en BASIC:

```
10 MEMORY 39999
20 FOR I=0 TO 27
30 READ A$
40 POKE (40200+I),VAL("&"+A$)
50 NEXT:CLS
60 INPUT "Valor";A
70 POKE 40201,A
80 CALL 40200
90 PRINT:GOTO 60
100 DATA 3E,FF,4F,06,08,CB,21,38,09,3E,
    30,C5,CD,5A,BB,C1,18,07,3E,31,C5,CD,
    5A,BB,C1,10,EA,C9
110 DATA 06,10,CB,25,CB,15,38,0B,3E,30,
    C5,E5,CD,5A,BB,E1,C1,18,09,3E,31,C5,
    E5,CD,5A,BB,E1,C1,10,EA,C9
```

Una aplicación muy útil de esta rutina es la visión que te da del estado de los distintos indicadores o *flags* en el registro de *flags*. Es obvio que el registro F debe ser copiado en el registro A antes de poder hacer esto, pero ello no conlleva excesiva complicación.

La siguiente rutina escribe el contenido de HL de modo similar.

EBINHL (Escribe HL en BINario)

Esta rutina escribe el contenido del par HL como un número binario de dieciséis dígitos. El número se imprime en el color y en la posición actual del cursor de textos. Esta rutina es relocalizable.

Requisitos de entrada: HL contiene el número.

Condiciones de salida: AF, BC y HL alterados.

Longitud: 31 bytes.

```
250E  0610        10 EBINHL LD   B,16  ; Es un numero de 16 digitos
2510  CB25        20 BUCLE  SLA  L ;     Desplaza los 16 bits un bit
                  30 ;                  a la izquierda
2512  CB14        40        RL   H     ; Bit de mayor peso en C
2514  380B        50        JR   C,CERO
2516  3E30        60        LD   A,#30 ; Codigo ASCII para 0
2518  C5          70        PUSH BC
2519  E5          80        PUSH HL
251A  CD5ABB      90        CALL #BB5A ; Lo escribe en pantalla
251D  E1         100        POP  HL
251E  C1         110        POP  BC
251F  1809       120        JR   SALIDA
2521  3E31       130 CERO   LD   A,#31 ; ASCII de 1
2523  C5         140        PUSH BC
2524  E5         150        PUSH HL
2525  CD5ABB     160        CALL #BB5A ; Escribe el 1
2528  E1         170        POP  HL
2529  C1         180        POP  BC
252A  10E4       190 SALIDA DJNZ BUCLE ; Si aun quedan digitos,
                  200 ;                  volver otra vez
252C  C9         210        RET
```

El programa en BASIC equivalente a esta rutina sería:

```
10 MEMORY 39999
20 FOR I=0 TO 33
30 READ A$
40 POKE (40200+I),VAL("&"+A$)
50 NEXT:CLS
60 INPUT "Valor";A
70 alto=INT(A/256)
80 bajo=A-(alto*256)
90 POKE 40201,bajo
100 POKE 40202,alto
110 CALL 40200
120 PRINT:GOTO 60
130 DATA 21,00,00
140 DATA 06,10,CB,25,CB,14,38,0B,3E,30,
    C5,E5,CD,5A,BB,E1,C1,18,09,3E,31,C5,
    E5,CD,5A,BB,E1,C1,10,E4,C9
```

De todos modos se suele optar por la impresión del contenido del registro en modo decimal o hexadecimal. Las dos rutinas que a continuación expongo tratan de la escritura de números en representación hexadecimal.

EHEXA (Escribe A en HEXadecimal)

Escribe el contenido del registro A como un número hexadecimal de dos dígitos, en la posición y color actuales del cursor de texto. La rutina es relocalizable.

Requisitos de entrada: A contiene el número.

Condiciones de salida: AF, BC alterados.

Longitud: 41 bytes.

```
                    10 ;       ** EHEXA **
269C  0600          20 EHEXA   LD    B,0    ; B es usado como flag
269E  4F            30         LD    C,A
269F  CB1F          40         RR    A      ; Las siguientes instrucciones trasladan
26A1  CB1F          50         RR    A      ; los 4 bits de mayor peso
26A3  CB1F          60         RR    A      ; a la zona de menor peso
26A5  CB1F          70         RR    A      ; del registro A
26A7  E60F          80 ESCBAJ  AND   #0F    ; Hace una mascara de esta zona
26A9  FE0A          90         CP    #0A    ; Si es mayor o igual que 10
26AB  2008         100         JR    NZ,AaF; salta.
26AD  C630         110         ADD   A,#30 ; Convierte el numero de A
                   120 ;                     en un caracter entre 0 y 9
26AF  C5           130         PUSH  BC
26B0  CD5ABB       140         CALL  #BB5A ; Escribe el digito
26B3  1806         150         JR    SALIDA
26B5  C637         160 AaF    ADD   A,#37 ; Convierte el contenido de A
                   170 ;                     en un caracter entre A y F
26B7  C5           180         PUSH  BC
26B8  CD5ABB       190         CALL  #BB5A ; Lo escribe
26BB  C1           200 SALIDA POP   BC
26BC  78           210         LD    A,B    ; Examina el flag, si es 1
26BD  FE01         220         CP    1      ; todos los digitos han sido escritos
26BF  C8           230         RET   Z      ; Vuelve al comienzo
26C0  79           240         LD    A,C    ; con el segundo digito preparado
26C1  0601         250         LD    B,1    ; Activa el flag B con 1
26C3  18E2         260         JR    ESCBAJ
```

Puedes introducirlo en la memoria mediante la orden POKE de la siguiente forma:

```
10 REM      ** EHEXA **
20 MEMORY 39999
30 FOR I=0 TO 42
40 READ A$
50 POKE (40200+I),VAL("&"+A$)
60 NEXT:CLS
70 INPUT "Valor para el registro A ";A
80 POKE 40201,A
90 CALL 40200
100 PRINT
110 GOTO 70
120 DATA 3E,00
130 DATA 06,00,4F,CB,1F,CB,1F,CB,1F,CB,
    1F,E6,0F,FE,0A,30,08,C6,30,C5,CD,5A,
```

```
BB,18,06,C6,37,C5,CD,5A,BB,C1,78,FE,
01,C8,79,06,01,18,E2
```

En el cuerpo de esta rutina se hallan las dos instrucciones de adición que transforman los valores de 0 a 9 en sus correspondientes códigos, y los dígitos de A a F en sus correspondientes códigos ASCII.

EHEXHL (Escribe HL en HEXadecimal)

Escribe el contenido del par de registros HL como un número hexadecimal de cuatro dígitos. El número se escribe en la posición y el color actual del cursor de texto. El código es no relocalizable, y los bytes que doy a continuación deben cargarse en la dirección 41000. Sin embargo, consulta los comentarios para ver cómo puedes alterar el código.

Requisitos de entrada: HL contiene el número.

Condiciones de salida: AF, BC alterados.

Longitud: 51 bytes.

```
243F  7C          10 EHEXHL LD   A,H
2440  CD4824      20        CALL EHEXA
2443  7D          30        LD   A,L
2444  CD4824      40        CALL EHEXA
2447  C9          50        RET
2448  0600        60 EHEXA  LD   B,0
244A  4F          70        LD   C,A
244B  CB1F        80        RR   A
244D  CB1F        90        RR   A
244F  CB1F       100        RR   A
2451  CB1F       110        RR   A
2453  E60F       120 ESCBAJ AND  #0F
2455  FE0A       130        CP   #0A
2457  3008       140        JR   NC,AaF
2459  C630       150        ADD  A,#30
245B  C5         160        PUSH BC
245C  CD5ABB     170        CALL #BB5A
245F  1806       180        JR   SALIDA
2461  C637       190 AaF    ADD  A,#37
2463  C5         200        PUSH BC
2464  CD5ABB     210        CALL #BB5A
2467  C1         220 SALIDA POP  BC
2468  78         230        LD   A,B
2469  FE01       240        CP   1
246B  C8         250        RET  Z
246C  79         260        LD   A,C
246D  0601       270        LD   B,1
246F  18E2       280        JR   ESCBAJ
```

El programa BASIC que puedes utilizar para introducir la rutina en la memoria del ordenador es:

```
10 REM      ** EHEXHL **
20 MEMORY 39999
30 FOR I=0 TO 6
40 READ A$
50 POKE (40200+I),VAL("&"+A$)
60 NEXT:CLS
70 FOR I=0 TO 49
80 READ A$
90 POKE (41000+I),VAL("&"+A$)
100 NEXT
110 INPUT "Valor del registro HL ";A
120 POKE 40202,INT(A/256)
130 POKE 40201,(A-((INT(A/256))*256))
140 CALL 40200
150 PRINT
160 GOTO 110
170 DATA 21,00,00,CD,28,A0,C9
180 DATA 7C,CD,31,A0,7D,CD,31,A0,C9,06,
    00,4F,CB,1F,CB,1F,CB,1F,CB,1F,E6,0F,
    FE,0A,30,08,C6,30,C5,CD,5A,BB,18,06,
    C6,37,C5,CD,5A,BB,C1,78,FE,01,C8,79,
    06,01,18,E2
```

Comentarios

Si eres observador, habrás advertido que EHEXHL equivale simplemente a dos llamadas a EHEXA. Aquí comienzan los problemas con la relocalización del código; al haber diseñado parte del programa dentro de una subrutina, su dirección dependerá de aquella donde el programa esté almacenado en memoria. Si EXEXHL se carga en la dirección N, entonces EHEXA se hallará en la dirección (N+9). Así, los bytes (N+2) y (N+6) deberán alterarse de forma que contengan el byte bajo de esta dirección; y los bytes (N+3) y (N+7) deberán modificarse hasta contener el byte alto de la dirección de EHEXA.

La última rutina de impresión de números que veremos escribe el contenido de los registros en decimal. Antes de verla, convendría que echaras un vistazo al algoritmo utilizado.

Es un simple caso de sucesivas restas de las potencias de 10 del número a imprimirse, hasta que el resultado sea negativo. Entonces añadiremos al resultado la potencia de 10, reconvirtiéndolo en un número positivo. El número de veces que determinada potencia de 10 haya sido sustraída será el dígito para esta potencia de 10 en concreto. Entonces se escribe el dígito. El proceso se repite con la siguiente potencia de 10 inmediatamente inferior, y así sucesivamente, hasta que se sustraigan las unidades, quedando como número tratado el valor 0. Así, para un registro de ocho bits, tendremos que sustraer, por orden, las centenas, las decenas

y, finalmente, las unidades. Estas rutinas son útiles para muchas aplicaciones como tablas de puntuación, e impresión de números de línea para juegos en código máquina, impresión de números de línea para rutinas de utilidad en código máquina, etc.

EDECA (Escribe A en DECimal)

Escribe el contenido del registro A en la posición y color actuales del cursor de texto, como un número decimal de tres dígitos. La rutina es no relocalizable, pero aun así consulta los Comentarios. Los siguientes bytes funcionan correctamente en la dirección 41060.

Requisitos de entrada: A contiene el número.

Condiciones de salida: AF, BC, DE resultan alterados.

Longitud: 30 bytes.

```
2512  1664        10 EDECA  LD   D,100
2514  CD1E25      20        CALL EDEC      ; Escribe las centenas
2517  160A        30        LD   D,10
2519  CD1E25      40        CALL EDEC      ; Escribe las decenas
251C  1601        50        LD   D,1
251E  0E00        60 EDEC   LD   C,0       ; C actua como contador
2520  92          70 BUCLE  SUB  D         ; Resta la potencia de diez;
2521  3803        80        JR   C,SALIDA ; si el resultado es negativo, salta
2523  0C          90        INC  C
2524  18FA       100        JR   BUCLE
2526  82         110 SALIDA ADD  A,D       ; Restaura a A como positivo
2527  F5         120        PUSH AF
2528  79         130        LD   A,C
2529  C630       140        ADD  A,#30     ; Convierte al contador en un digito
252B  CD5ABB     150        CALL #BB5A     ; y lo escribe por pantalla
252E  F1         160        POP  AF
252F  C9         170        RET  ;           Terminado el ciclo
```

Y su equivalente en BASIC:

```
10 REM      ** EDECA **
20 MEMORY 39999
30 FOR I=0 TO 5
40 READ A$
50 POKE (40200+I),VAL("&"+A$)
60 NEXT:CLS
70 FOR I=0 TO 29
80 READ A$
90 POKE (41060+I),VAL("&"+A$)
100 NEXT
110 INPUT "Valor del registro A ";A
120 POKE 40201,A
130 CALL 40200
```

```
140 PRINT
150 GOTO 110
160 DATA 3E,00,CD,64,A0,C9
170 DATA 16,64,CD,70,A0,16,0A,CD,70,A0,
    16,01,0E,00,92,38,03,0C,18,FA,82,F5,
    79,C6,30,CD,5A,BB,F1,C9
```

Comentarios

Si la rutina anterior se almacena en otra dirección que no sea 41060, digamos dirección N, entonces la subrutina PDEC se situará en la dirección (N+12). Las posiciones (N+3) y (N+8) deben contener el byte bajo de la dirección (N+12); y las posiciones (N+4) y (N+9) deben contener el byte alto de la dirección (N+12).

EDECHL (Escribe HL en DECimal)

Escribe el contenido del par de registros HL bajo la forma de un número decimal de cinco dígitos, en el color y la posición actual del cursor de texto. La rutina es no relocalizable. Los siguientes bytes funcionarán correctamente en la dirección 41200, pero me remito a los comentarios para mayores detalles sobre el funcionamiento de la rutina en otras direcciones.

Requisitos de entrada: HL contiene el número.

Condiciones de salida: AF, HL, DE son alterados.

Longitud: 46 bytes.

```
25B2  111027      10 EDECHL LD   DE,10000
25B5  CDCD25      20        CALL EDECH   ; Escribe las decenas de millar
25B8  11E803      30        LD   DE,1000
25BB  CDCD25      40        CALL EDECH   ; Escribe las unidades de millar
25BE  116400      50        LD   DE,100
25C1  CDCD25      60        CALL EDECH   ; Escribe las centenas
25C4  110A00      70        LD   DE,10
25C7  CDCD25      80        CALL EDECH   ; Escribe las decenas
25CA  110100      90        LD   DE,1
25CD  AF         100 EDECH  XOR  A       ; Inicializa el contador
25CE  37         110 BUCLE  SCF
25CF  3F         120        CCF  ;         Pone a 0 el flag de acarreo
25D0  ED52       130        SBC  HL,DE   ; Realiza la sustraccion
25D2  3803       140        JR   C,SALIDA; hasta que sea negativa
25D4  3C         150        INC  A
25D5  18F7       160        JR   BUCLE
25D7  19         170 SALIDA ADD  HL,DE   ; Lo restaura haciendolo positivo
25D8  C630       180        ADD  A,#30   ; Convierte el contador en un
                 190 ;                     digito ASCII y . . .
25DA  E5         200        PUSH HL
25DB  CD5ABB     210        CALL #BB5A ; lo escribe
25DE  E1         220        POP  HL
25DF  C9         230        RET
```

```
10 REM      ** EDECHL **
20 MEMORY 39999
30 FOR I=0 TO 6
40 READ A$
50 POKE (40200+I),VAL("&"+A$)
60 NEXT:CLS
70 FOR I=0 TO 45
80 READ A$
90 POKE (41200+I),VAL("&"+A$)
100 NEXT
110 INPUT "Valor para el registro HL ";A
120 POKE 40201,(A-(INT(A/256)*256))
130 POKE 40202,INT(A/256)
140 CALL 40200
150 PRINT
160 GOTO 110
170 DATA 21,00,00,CD,F0,A0,C9
180 DATA 11,10,27,CD,0B,A1,11,E8,03,CD,
    0B,A1,11,64,00,CD,0B,A1,11,0A,00,CD,
    0B,A1,11,01,00,AF,37,3F,ED,52,38,03,
    3C,18,F7,19,C6,30,E5,CD,5A,BB,E1,C9
```

Comentarios

En este caso, la subrutina EDECH se encuentra en la dirección &A10B. Se hallará siempre en la dirección (N+27), siendo N la dirección donde se almacenó la totalidad del programa. Por tanto, si almacenas la rutina en una dirección distinta a la dada, deberás alterar las llamadas a la subrutina que se encuentran al comienzo de esta rutina para que apunten hacia la nueva dirección de EDECH.

Esto completa el capítulo sobre salida de textos, aunque veremos en posteriores capítulos otras rutinas relacionadas con la manipulación de textos. Ahora vamos a ver algunas operaciones de gráficos, incluyendo una variedad de rutinas que aumentan las órdenes usuales de BASIC para manejo de gráficos.

3
Rutinas de gráficos

En este capítulo veremos varias rutinas que aprovechan las facilidades ofrecidas por Amstrad para la producción de gráficos. También me ocuparé de algunas rutinas de manipulación de textos, como las que escriben en pantalla grandes caracteres, y, dado su interesante funcionamiento, de aquellas que invierten y escriben los caracteres boca abajo en la pantalla. Además, se incluyen una rutina de propósito general para dibujar composiciones gráficas y cuadros en la pantalla a partir de una lista de datos, y varias rutinas más de utilidad para los gráficos.

Comenzaremos con algunas rutinas que escriben grandes caracteres en pantalla, muy útiles para títulos o programas de demostración. Las dos primeras no utilizan ninguna rutina propia para gráficos; sin embargo, presentan algunas interesantes rutinas de la ROM.

CGRAND (Carácter GRANDe)

Esta escribe en la pantalla un carácter enormemente grande de ocho caracteres de ancho y de ocho de altura. La rutina trabajará en cualquier modo de pantalla y el carácter aumentado se escribirá en la posición actual del cursor de textos, y con el color de la tinta asignada a textos. Examina los comentarios posteriores para los detalles de la relocalización.

Requisitos de entrada: Trabajando desde el BASIC, llamaremos a esta rutina mediante CALL dirección, n; donde n es el código ASCII del carácter que ha de ser escrito.
Si hacemos la llamada desde otra rutina de código máquina, entonces A contiene el valor 1 y el registro IX apunta al código del carácter a escribir.

Condiciones de salida: Todos los registros son alterados. La rutina termina cuando el carácter ha sido escrito o cuando detecta que se ha transferido un número erróneo de parámetros.

Longitud: 80 bytes.

```
                10 ;       ** CGRAND **
9D08            20         ORG  40200
9D08  FE01      30         CP   1
9D0A  C0        40         RET  NZ          ; Si no hay un parametro
                50 ;                          regresa al programa BASIC
9D0B  CD06B9    60         CALL #B906       ; Rutina de la ROM
9D0E  F5        70         PUSH AF          ; Salva los registros de estado
9D0F  DD7E00    80         LD   A,(IX)      ; Carga el caracter a imprimir
9D12  CDA5BB    90         CALL #BBA5       ; Extrae la direccion patron;
9D15  DD21409C 100         LD   IX,40000    ; que es donde se almacena
                110 ;                         el modelo del caracter
9D19  0608      120        LD   B,8         ; 8 bytes a desplazar
9D1B  7E        130 BUCLE  LD   A,(HL)      ; HL contiene la direccion
                140 ;                         del modelo
9D1C  DD7700    150        LD   (IX),A
9D1F  23        160        INC  HL
9D20  DD23      170        INC  IX
9D22  10F7      180        DJNZ BUCLE       ; Transfiere los 8 bytes
                190 ;                         del modelo
9D24  F1        200        POP  AF
9D25  CD0CB9    210        CALL #B90C       ; Restaura los registros de estado
9D28  DD21409C  220        LD   IX,40000
9D2C  1608      230        LD   D,8         ; Estos son los 8 bytes
                240 ;                         que definen el caracter
9D2E  DD7E00    250 BUCLE1 LD   A,(IX)      ; Extrae el byte al que apunta IX
9D31  4F        260        LD   C,A
9D32  0608      270        LD   B,8
9D34  CB21      280 BUCLE2 SLA  C           ; Salta de bit en bit
                290 ;                         para cada byte
9D36  3807      300        JR   C,CERO
9D38  3E20      310        LD   A,32        ; Escribe un espacio si
                320 ;                         el bit es '0'
9D3A  CD5ABB    330        CALL #BB5A
9D3D  1805      340        JR   SALIDA
9D3F  3EFF      350 CERO   LD   A,255       ; Escribe CHR$(255)
                360 ;                         si el bit es '1'
9D41  CD5ABB    370        CALL #BB5A
9D44  10EE      380 SALIDA DJNZ BUCLE2
9D46  DD23      390        INC  IX
9D48  3E0A      400        LD   A,10        ; Baja una linea
9D4A  CD5ABB    410        CALL #BB5A
9D4D  0608      420        LD   B,8
9D4F  3E08      430 BUCLE3 LD   A,8
9D51  CD5ABB    440        CALL #BB5A
9D54  10F9      450        DJNZ BUCLE3      ; Envia 8 veces el
                460 ;                         caracter CHR$(8)
```

```
9D56  15        470       DEC  D
9D57  20D5      480       JR   NZ,BUCLE1; Si todos los bytes de la
                490 ;                    definicion han sido
9D59  C9        500       RET  ;          realizados ... Final.
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 SYMBOL 255,255,255,255,255,255,255,
   255,255
20 CLS
30 LOCATE 1,1
40 INPUT " Cadena: ",A$
50 LOCATE 10,10
60 FOR I=1 TO LEN(A$)
70 LOCATE I*8+1,10
80 CALL 40200,ASC(MID$(A$,I,1))
90 NEXT
100 CALL &BB06:GOTO 20
110 REM Cambia la linea 10 por :
120 REM SYMBOL 255,40,40,40,40....
130 REM Hay muchas combinaciones.
```

Comentarios

Normalmente, podemos llamar a esta rutina mediante una sentencia en BASIC como ésta:

```
y=10:A$="cadena":FOR X=1 TO LEN(A$)
LOCATE X*8+1,y
CALL 40200,ASC(MID$(A$,X,1))
NEXT X
```

Obviamente, he supuesto que el código máquina ha sido ensamblado a partir de la dirección 40200. Los bytes del programa anteriormente listado son correctos únicamente para esta dirección. Mira los comentarios que hay después de CARVAR para los detalles sobre la relocalización de estas rutinas.

CDOBLE (Carácter DOBLE)

Esta rutina presenta en la pantalla un carácter con una altura doble de lo normal. Al igual que la anterior, ésta funcionará en cualquier modo de pantalla y se escribirá con el color de la tinta de textos y en la posición que tenga el cursor de textos en ese momento.

Requisitos de entrada: Trabajando con el BASIC, podremos acceder a ella con la sentencia CALL dirección, n; siendo n el código ASCII del carácter que ha de escribirse. Partiendo de otra rutina en código máquina, podremos llamar a CDOBLE siempre que el registro IX apunte al carácter que ha de ser escrito y que el registro A contenga el valor 1.

Condiciones de salida: Todos los registros son alterados; la rutina devolverá el control cuando el carácter haya sido escrito o cuando detecte que un número erróneo de parámetros ha sido transferido desde la sentencia CALL.

Longitud: 73 bytes.

```
9D08            10        ORG  40200
9D08  FE01      20        CP   1
9D0A  C0        30        RET  NZ       ; Si no hay ningun parametro
                40 ;                       regresa al programa BASIC
9D0B  CD06B9    50        CALL #B906     ; Activa la ROM
9D0E  F5        60        PUSH AF        ; Salva el registro de estado
                70 ;                       de la ROM
9D0F  DD7E00    80        LD   A,(IX)    ; Extrae el caracter que
      .         90 ;                       tiene que escribir
9D12  CDA5BB    100       CALL #BBA5     ; Obtiene la direccion modelo
9D15  DD21409C  110       LD   IX,40000; que es donde almacenaremos
                120 ;                      la matriz del caracter
9D19  0608      130       LD   B,8       ; 8 bytes para desplazar
9D1B  7E        140 BUCLE1 LD  A,(HL)
9D1C  DD7700    150       LD   (IX),A    ; Cada byte es copiado dos veces
9D1F  DD23      160       INC  IX        ; para lograr asi un
9D21  DD7700    170       LD   (IX),A    ; byte de longitud doble
9D24  DD23      180       INC  IX        ; en la definicion de la altura
9D26  23        190       INC  HL
9D27  10F2      200       DJNZ BUCLE1
9D29  F1        210       POP  AF
9D2A  CD0CB9    220       CALL #B90C     ; Restaura el estado de la ROM
9D2D  3EFE      230       LD   A,254     ; Define a CHR$(254) como
9D2F  21409C    240       LD   HL,40000; parte superior del caracter
9D32  CDA8BB    250       CALL #BBA8
9D35  3EFF      260       LD   A,255     ; Define a CHR$(255) como
9D37  21479C    270       LD   HL,40007; la mitad inferior del caracter
9D3A  CDA8BB    280       CALL #BBA8
9D3D  3EFE      290       LD   A,254     ; Escribe la mitad superior
9D3F  CD5ABB    300       CALL #BB5A
9D42  3E0A      310       LD   A,10      ; Baja una linea
9D44  CD5ABB    320       CALL #BB5A
9D47  3E08      330       LD   A,8       ; Retrocede un espacio
9D49  CD5ABB    340       CALL #BB5A
9D4C  3EFF      350       LD   A,255     ; Escribe la mitad inferior
9D4E  CD5ABB    360       CALL #BB5A
9D51  C9        370       RET  ;         Regreso al BASIC
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 REM Rutina CDOBLE
20 CLS
30 INPUT A$:CLS
```

```
40 Y=10 : X1=3 : FOR I=1 TO LEN(A$)
50 LOCATE X1+I-1,Y
60 CALL 40200,ASC(MID$(A$,I,1))
70 NEXT
```

Comentarios

He dicho antes que esta rutina trabaja en todos los modos de pantalla, pero será particularmente efectiva en el modo 0, donde la doble anchura del texto facilitará la lectura haciendo más legible el texto. Si deseas ver el programa en acción, ensambla el código máquina a partir de la dirección 40200 y prueba también con las siguientes líneas de BASIC:

```
10 Y=10 : X=1 :REM Posicion de pantalla
20 MODE 2
30 FOR I=X TO LEN(A$)
40 LOCATE X+I-1,Y
50 CALL 40200,ASC(MID$(A$,I,1))
60 NEXT
```

Las notas sobre la relocalización de esta rutina se darán también después de la descripción de CARVAR.

Veamos ahora esta última rutina que escribe grandes caracteres. Examinando el listado comprobarás que hace uso de operaciones con gráficos.

CARVAR (CARácter VARiable)

Esta rutina es muy versátil para escribir caracteres de tamaño variable a voluntad. El ancho del carácter se mantiene constante a seis espacios, permitiendo con ello mayor legibilidad en cualquier modo de pantalla. El carácter a escribir se localizará en la posición actual del cursor de gráficos y tendrá el color asignado para los gráficos.

Requisitos de entrada: Si partimos desde BASIC, usaremos CALL dirección,n,m. En este caso, n es el código ASCII del carácter que ha de escribirse, y m es un número entero que especifica su altura relativa.
Si accedemos a esta rutina desde un programa en código máquina, A contendrá el valor 2 y el registro IX apuntará hacia un bloque de memoria.
(IX+0) contiene la altura y (IX+2) contendrá el código ASCII del carácter.

Condiciones de salida: Todos los registros se devuelven alterados. La rutina finaliza cuando escribe el carácter o cuando detecta un número incorrecto de parámetros.

Longitud: 238 bytes.

```
                    10 ;       ** CARVAR **
9D08                20         ORG  40200
9D08  FE02          30         CP   2
9D0A  C0            40         RET  NZ          ; Si no hay dos parametros
                    50 ;                          retorna
9D0B  DD7E00        60         LD   A,(IX)
9D0E  32F19D        70         LD   (ALTURA),A; Recoge el valor de la altura
9D11  CD06B9        80         CALL #B906       ;    Acceso a la memoria ROM
9D14  F5            90         PUSH AF          ; Guarda el estado de la ROM
9D15  DD7E02       100         LD   A,(IX+2)    ; Caracter para escribir
9D18  CDA5BB       110         CALL #BBA5       ; Obtiene la direccion del
9D1B  DD21409C     120         LD   IX,40000    ; modelo del caracter
9D1F  0608         130         LD   B,8         ; y almacena los 8 bytes del modelo
9D21  7E           140 BUCLE   LD   A,(HL)      ; en el espacio de trabajo
                    150 ;                         mediante incrementos de HL
9D22  DD7700       160         LD   (IX),A
9D25  23           170         INC  HL
9D26  DD23         180         INC  IX
9D28  10F7         190         DJNZ BUCLE       ; Transferidos los 8 bytes
9D2A  F1           200         POP  AF
9D2B  CD0CB9       210         CALL #B90C       ; Restaura el estado de la ROM
9D2E  DD21409C     220         LD   IX,40000
9D32  1608         230         LD   D,8         ; 8 bytes de definicion
                    240 ;                         del caracter
9D34  DD7E00       250 BUCLE1  LD   A,(IX)      ; Obtiene el byte al que
                    260 ;                         apunta IX
9D37  4F           270         LD   C,A
9D38  0608         280         LD   B,8
9D3A  CB21         290 BUCLE2  SLA  C           ; Salta de bit en bit, dentro
                    300 ;                         de cada byte
9D3C  CD8C9D       310         CALL CURSOR
9D3F  3805         320         JR   C,CERO
9D41  CDDC9D       330         CALL VERTS       ; El bit es '0'
9D44  1803         340         JR   SALIDA
9D46  CDC79D       350 CERO    CALL VERT        ; El bit es '1'
9D49  10EF         360 SALIDA  DJNZ BUCLE2
9D4B  DD23         370         INC  IX
9D4D  CD749D       380         CALL SALH
9D50  15           390         DEC  D
9D51  20E1         400         JR   NZ,BUCLE1 ; Si se ha dado cuenta de
                    410 ;                         todos los bits de la definicion...
9D53  C9           420         RET  ; Final
9D54  E5           430 ESCRIB  PUSH HL
9D55  D5           440         PUSH DE
9D56  C5           450         PUSH BC
9D57  110600       460         LD   DE,6
9D5A  210000       470         LD   HL,0
9D5D  CDF9BB       480         CALL #BBF9       ; Dibuja una corta linea horizontal
                    490 ;                         El trazo es relativo
9D60  C1           500         POP  BC
9D61  D1           510         POP  DE
9D62  E1           520         POP  HL
9D63  C9           530         RET
9D64  E5           540 ESCRIS  PUSH HL
9D65  D5           550         PUSH DE
9D66  C5           560         PUSH BC
9D67  110600       570         LD   DE,6
9D6A  210000       580         LD   HL,0
9D6D  CDC3BB       590         CALL #BBC3       ; Se mueve a lo largo de la linea
                    600 ;                         El movimiento es relativo
9D70  C1           610         POP  BC
```

```
9D71  D1          620         POP  DE
9D72  E1          630         POP  HL
9D73  C9          640         RET
9D74  C5          650 SALH    PUSH BC
9D75  E5          660         PUSH HL
9D76  D5          670         PUSH DE
9D77  210000      680         LD   HL,0       ; Ahora trabaja verticalmente
9D7A  3AF19D      690         LD   A,(ALTURA);Movimiento relativo dependiente
                  700 ;                       del parametro ALTURA
9D7D  47          710         LD   B,A
9D7E  2B          720 DBUCLE  DEC  HL         ; Asi, la siguiente linea del
9D7F  2B          730         DEC  HL         ; caracter se dibujara con un
9D80  10FC        740         DJNZ DBUCLE     ; movimiento relativo de 8*6 puntos
9D82  11D0FF      750         LD   DE,-48
9D85  CDC3BB      760         CALL #BBC3
9D88  D1          770         POP  DE
9D89  E1          780         POP  HL
9D8A  C1          790         POP  BC
9D8B  C9          800         RET
9D8C  C5          810 CURSOR  PUSH BC
9D8D  E5          820         PUSH HL
9D8E  D5          830         PUSH DE
9D8F  F5          840         PUSH AF
9D90  CDC6BB      850         CALL #BBC6      ; Posiciona el cursor de graficos
9D93  0606        860         LD   B,6
9D95  13          870 CBUCLE  INC  DE         ; Suma 6 a la coord. X
9D96  10FD        880         DJNZ CBUCLE
9D98  ED53F29D    890         LD   (TEMPX),DE; Lo almacena
9D9C  22F49D      900         LD   (TEMPY),HL; Almacena el bit de Y
9D9F  F1          910         POP  AF
9DA0  D1          920         POP  DE
9DA1  E1          930         POP  HL
9DA2  C1          940         POP  BC
9DA3  C9          950         RET
9DA4  C5          960 RESTAU  PUSH BC
9DA5  D5          970         PUSH DE
9DA6  E5          980         PUSH HL
9DA7  11FAFF      990         LD   DE,-6      ; Movimiento relativo de
                 1000 ;                       6 puntos a la izquierda
9DAA  21FEFF     1010         LD   HL,-2      ; y de 2 hacia abajo
9DAD  CDC3BB     1020         CALL #BBC3
9DB0  E1         1030         POP  HL
9DB1  D1         1040         POP  DE
9DB2  C1         1050         POP  BC
9DB3  C9         1060         RET
9DB4  F5         1070 NEXT    PUSH AF
9DB5  C5         1080         PUSH BC
9DB6  D5         1090         PUSH DE
9DB7  E5         1100         PUSH HL
9DB8  ED5BF29D   1110         LD   DE,(TEMPX)
9DBC  2AF49D     1120         LD   HL,(TEMPY)
9DBF  CDC0BB     1130         CALL #BBC0      ; Movimiento absoluto al
                 1140 ;                       siguiente (NEXT) punto, con lo
                 1150 ;                       cual la siguiente parte de la
                 1160 ;                       linea del caracter debe ser
                 1170 ;                       punteada
9DC2  E1         1180         POP  HL
9DC3  D1         1190         POP  DE
9DC4  C1         1200         POP  BC
9DC5  F1         1210         POP  AF
9DC6  C9         1220         RET
9DC7  C5         1230 VERT    PUSH BC         ; Dibuja varias lineas
9DC8  CD8C9D     1240         CALL CURSOR     ; horizontales, una debajo
9DCB  3AF19D     1250         LD   A,(ALTURA);de la otra y con
9DCE  47         1260         LD   B,A        ; altura variable
9DCF  CD549D     1270 VBUCLE  CALL ESCRIB
9DD2  CDA49D     1280         CALL RESTAU
9DD5  10F8       1290         DJNZ VBUCLE
9DD7  CDB49D     1300         CALL NEXT
9DDA  C1         1310         POP  BC
```

```
9DDB  C9        1320         RET
9DDC  C5        1330 VERTS   PUSH BC          ; Por ser el vertice,
9DDD  CD8C9D    1340         CALL CURSOR      ; no se dibujan lineas
9DE0  3AF19D    1350         LD   A,(ALTURA)
9DE3  47        1360         LD   B,A
9DE4  CD649D    1370 VSBUC   CALL ESCRIS
9DE7  CDA49D    1380         CALL RESTAU
9DEA  10F8      1390         DJNZ VSBUC
9DEC  CDB49D    1400         CALL NEXT
9DEF  C1        1410         POP  BC
9DF0  C9        1420         RET
9DF1  00        1430 ALTURA  DEFB 0
9DF2  0000      1440 TEMPX   DEFW 00
9DF4  0000      1450 TEMPY   DEFW 00
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 CLS
20 INPUT "Cadena: ",A$
30 INPUT "Altura: ",A%
40 MOVE 100,100
50 FOR I=1 TO LEN(A$)
60 MOVE I*80+10,100
70 CALL 40200,ASC(MID$(A$,I,1)),A%
80 NEXT
90 IF INKEY$="" THEN 90
100 CLS
110 GOTO 20
```

Comentarios

Debido a la cantidad de subrutinas que maneja, esta rutina no es fácilmente relocalizable. Sin embargo, si te sientes dispuesto a ello, lee las siguientes advertencias y tenlas muy en cuenta a la hora de cambiar la dirección donde se aloja el programa:

1. La dirección donde se encuentra el espacio de trabajo, normalmente en la 40000, tendrá que ser cambiada si la nueva dirección del programa es tal que se superpone al espacio de trabajo.
2. Obviamente tendrás que modificar, conforme a la nueva relocalización del programa, los bytes que forman las direcciones de las distintas subrutinas del programa y que pertenecen a las instrucciones CALL.
3. Me remito a las notas del prólogo para comentar algunos directivos del ensamblador.

Los bytes que forman el programa antes listado están preparados para trabajar en la dirección 40200. Tal vez quieras probar también su utilidad

con el siguiente programa en BASIC. Se supone que el código máquina se encuentra ya en el ordenador y en la dirección correcta.

```
10 MODE 2
20 INPUT "Cadena: ",A$
30 INPUT "Altura: ",A%
40 MOVE 100,100:REM Posicion de comienzo
50 FOR I=1 TO LEN(A$)
60 MOVE I*50+10,100:REM Se mueve a la
   posicion del siguiente caracter.
70 CALL 40200,ASC(MID$(A$,I,1)),A%
80 NEXT
90 IF INKEY$="" THEN 90 :REM Pulsa para
   continuar.
100 CLS
110 GOTO 20
```

La separación entre los caracteres escritos se puede ajustar variando las constantes de la sentencia MOVE de la línea 60. El carácter se escribe en pantalla con su esquina superior izquierda en la posición que ocupa el cursor de gráficos.

Dedicaremos unas líneas para examinar algunas rutinas de la ROM, que hemos utilizado para la confección del programa CARVAR. También vamos a ver algunas limitaciones respecto al lugar en donde pueden ser relocalizadas las tres rutinas anteriores.

Rutinas de ROM utilizadas

&B906: Esta nos permite acceder a la ROM más baja en vez de disponer de la RAM que normalmente ocupa esta zona del mapa de memoria. La ROM baja ocupa las posiciones de memoria comprendidas entre &0000 y &4000; necesitamos tenerla disponible para poder copiar las definiciones de los caracteres que se guardan en la ROM y almacenarlas en el espacio de trabajo, que en estos programas empieza en la dirección 40000. Al volver de esta rutina, el registro A contiene lo que se conoce como estado de la ROM, mediante el cual podremos, al finalizar este tramo de la rutina, cerrar el acceso a la ROM y disponer nuevamente de la RAM.

&B90C: La rutina de esta dirección precisa disponer del byte de estado que fue generado por el sistema operativo cuando accedimos anteriormente a la ROM. Su función es devolver a la ROM el estado en que se encontraba originalmente. Cuando hayamos terminado de copiar la definición de los caracteres desde la ROM y la almacenemos en la RAM, usaremos esta rutina para que las cosas vuelvan a su estado normal.

La utilización de estas dos rutinas para acceder a la ROM explica el especial cuidado que debe ponerse al relocalizar el código máquina. Piensa un momento en ello; si inhibimos la zona RAM donde está almacenado nuestro código máquina, la ejecución del programa se romperá al activar la ROM. Por esta razón, sólo podemos situar estos programas en una zona de memoria que no se vea afectada al perder acceso a las direcciones de RAM ocupadas en ese instante por la ROM. Ninguna dirección entre &4000 y &BFFF será afectada cuando accedamos a la ROM; así, puedes relocalizar tu programa en cualquier dirección dentro de esta área, pero nunca almacenar en la zona comprendida entre &0000 y &3FFF ni el programa ni la zona de trabajo.

Las otras direcciones de ROM usadas por estas rutinas son:

&BBA5: Esta rutina, llamada TXT GET MATRIX por Amsoft, nos permite conocer la dirección del primero de los ocho bytes que definen el modelo del carácter que se escribe en pantalla. Para llamar a esta rutina, el registro A debe contener el código ASCII del carácter. Al regresar de la rutina, el par de registros HL contiene la dirección.

&BBA8: CDOBLE la utiliza para definir la mitad superior e inferior del carácter. En este caso, al llamar a esta rutina, HL contiene la dirección del primero de los ocho bytes consecutivos que conforman la definición, y el registro A contiene el código ASCII del carácter que tenemos que definir.

Un estudio más detallado de las rutinas de la ROM puede encontrarse en el manual de *firmware* del Amstrad y en la obra *Programación avanzada del Amstrad.*

Las dos siguientes rutinas que vamos a ver carecen de utilidad inmediata, siendo, sin embargo, bastante interesantes. Tanto ESPEJOV como ESPEJOH pueden ser relocalizadas en cualquier región de la memoria entre &4000 y &BFFF. De ahora en adelante, esta área útil de la memoria será denominada "depósito de memoria" *(memory pool).*

ESPEJOV (ESPEJO Vertical)

Esta rutina escribe un carácter boca abajo. El carácter corresponde al código ASCII que se transfiere al llamar a la rutina. Actúa como si un espejo hubiera sido situado al pie del carácter. La rutina es relocalizable en cualquier dirección del depósito de memoria. El carácter se escribirá en la posición y color actuales del cursor de textos.

Requisitos de entrada: Al igual que en rutinas precedentes, ésta puede estar disponible mediante la sentencia CALL dirección, n, siendo n el código ASCII del carácter

que nos interesa. Si partimos de otro programa en código máquina, entonces IX deberá apuntar al código del carácter, y A=1.

Condiciones de salida: Todos los registros son alterados.

Longitud: 50 bytes.

```
9D08                10        ORG  40200
9D08  FE01          20        CP   1
9D0A  C0            30        RET  NZ        ; Si no hay un solo parametro,
                    40 ;                       retorno al BASIC
9D0B  CD06B9        50        CALL #B906     ; Activa el acceso a la ROM
9D0E  F5            60        PUSH AF        ; Guarda el byte de estado de la ROM
9D0F  DD7E00        70        LD   A,(IX)
9D12  CDA5BB        80        CALL #BBA5     ; Obtiene la direccion en ROM
                    90 ;                       donde se guarda el caracter patron
9D15  DD21409C      100       LD   IX,40000; Direccion donde se almacena
9D19  110700        110       LD   DE,7      ; el modelo del caracter
9D1C  19            120       ADD  HL,DE
9D1D  0608          130       LD   B,8
9D1F  7E            140 BUCLE LD   A,(HL)    ; HL contiene la direccion del
                    150 ;                      ultimo byte del modelo
9D20  DD7700        160       LD   (IX),A
9D23  2B            170       DEC  HL
9D24  DD23          180       INC  IX
9D26  10F7          190       DJNZ BUCLE     ; Transfiere los 8 bytes del
                    200 ;                      caracter patron, empezando
                    210 ;                      por el ultimo de ellos
9D28  F1            220       POP  AF
9D29  CD0CB9        230       CALL #B90C     ; Restaura el estado de la ROM
9D2C  21409C        240       LD   HL,40000; Redefine el caracter CHR$(255)
9D2F  3EFF          250       LD   A,255
9D31  CDA8BB        260       CALL #BBA8
9D34  3EFF          270       LD   A,255     ; Escribe CHR$(255)
9D36  CD5ABB        280       CALL #BB5A
9D39  C9            290       RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 CLS
20 INPUT "Cadena:  ",A$
30 LOCATE 10,10 : PRINT A$
40 LOCATE 10,11
50 FOR I=1 TO LEN(A$)
60 CALL 40200,ASC(MID$(A$,I,1))
70 NEXT
80 PRINT
90 INPUT A$
100 GOTO 10
```

Comentarios

La rutina puede ser relocalizada en cualquier zona del depósito de memoria, pero recuerda que tienes que alterar la dirección del espacio de

trabajo (desde 40000), cuando almacenes el programa en este espacio de la memoria en concreto. Para ver la acción de esta rutina, prueba también con el siguiente programa BASIC:

```
10 MODE 1
20 INPUT "Cadena:  ",A$
30 LOCATE 10,10 : PRINT A$
40 LOCATE 10,11:REM Justo bajo la cadena
50 FOR I=1 TO LEN(A$)
60 CALL 40200,ASC(MID$(A$,I,1))
70 REM Se supone que el programa ya se
   encuentra almacenado en la direccion
   40200 de la memoria.
80 NEXT
90 PRINT
100 INPUT "Pulsa para continuar",a$
110 GOTO 10
```

ESPEJOH (ESPEJO Horizontal)

Esta rutina es similar a la anterior, con la diferencia de que escribe en pantalla un reflejo del perfil del carácter. Es decir, el carácter se reflejará en un espejo situado en su lado derecho. El carácter se escribirá en la posición y color actuales del cursor de textos.

Requisitos de entrada: Es necesario un único parámetro que acompañe a la orden CALL si trabajamos con un programa BASIC. El parámetro contiene el código del carácter a escribirse.
Si partimos de código máquina, IX deberá apuntar al código ASCII del carácter, y A=1.

Condiciones de salida: Todos los registros son alterados.

Longitud: 56 bytes.

```
                10 ;      ** ESPEJOH **
9D08            20        ORG   40200
9D08  FE01      30        CP    1
9D0A  C0        40        RET   NZ        ; Si no hay un solo parametro,
                50 ;                        regresa al BASIC
9D0B  CD06B9    60        CALL  #B906     ; Activa la ROM
9D0E  F5        70        PUSH  AF        ; Guarda el byte de estado de la ROM
9D0F  DD7E00    80        LD    A,(IX)
9D12  CDA5BB    90        CALL  #BBA5     ; Obtiene la direccion de la
                100 ;                       matriz del caracter
9D15  DD21409C  110       LD    IX,40000; Direccion donde se almacenara
                120 ;                       el modelo del caracter
9D19  0608      130       LD    B,8
```

```
9D1B  7E        140 BUCLE  LD   A,(HL)  ; HL contiene la direccion del
                150 ;                     primer byte de la matriz
9D1C  C5        160        PUSH BC
9D1D  0608      170        LD   B,8     ; 8 bits por cada byte
9D1F  4F        180        LD   C,A     ; Extrae cada byte de la definicion
9D20  CB11      190 IBUCLE RL   C       ; Transfiere el bit de la
9D22  1F        200        RRA  ;         izquierda del registro C
                210 ;                     al registro A
9D23  10FB      220        DJNZ IBUCLE
9D25  C1        230        POP  BC      ; Recupera BC
9D26  DD7700    240        LD   (IX),A  ; Transfiere el byte modificado
                250 ;                     de la definicion
9D29  23        260        INC  HL
9D2A  DD23      270        INC  IX
9D2C  10ED      280        DJNZ BUCLE   ; Se almacenan los 8 bytes de
                290 ;                     la matriz modelo
9D2E  F1        300        POP  AF
9D2F  CD0CB9    310        CALL #B90C   ; Restaura la ROM en su estado normal
9D32  21409C    320        LD   HL,40000; Redefine el caracter CHR$(255)
9D35  3EFF      330        LD   A,255
9D37  CDABBB    340        CALL #BBA8
9D3A  3EFF      350        LD   A,255   ; Escribe CHR$(255)
9D3C  CD5ABB    360        CALL #BB5A
9D3F  C9        370        RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 CLS
20 LINE INPUT "Cadena : ",A$
30 LOCATE 10,10
40 PRINT A$
50 LOCATE 10+LEN(A$),10
60 FOR I=LEN(A$) TO 1 STEP -1
70 CALL 40200,ASC(MID$(A$,I,1))
80 NEXT
90 CALL &BB06
100 GOTO 10
```

Comentarios

También esta rutina es relocalizable en el depósito de memoria, siempre que cambiemos la dirección del espacio de trabajo si fuera necesario.

La rutina puede probarse también con el mismo programa dado para ESPEJOV.

Ya hemos visto suficiente sobre caracteres. Vamos a ver ahora cómo realizar dibujos y gráficos reales. Con el sistema operativo del Amstrad resulta realmente fácil el uso de operaciones con gráficos desde rutinas de código máquina. Existen rutinas que dibujan puntos, trazan líneas, desplazan el cursor de gráficos, etc. La primera rutina de gráficos que vamos a

describir es una versión gráfica de la rutina ECADEN. La rutina acepta una tabla de instrucciones y coordenadas de gráficos, y opera con ellas para dibujar una figura con la forma que la tabla especifique. Dentro de la rutina, esta tabla es denominada FIGURA, ya que lo que hace es definir una figura. Como cada uno pensaréis en una figura distinta, éste será el mejor método para introduciros en los gráficos en código máquina por su generalidad. Claro está que, si queremos trabajar con un solo tipo de figuras, será más rápido escribir una rutina que realice este trabajo específico; pero con el tiempo comprobarás la gran versatilidad y expansión que puede alcanzar esta rutina.

FIGRAF (FIguras GRAFicas)

Se trata de una rutina de propósito general que dibuja en la pantalla gráficos de figuras en los tres modos. Los datos de las figuras a dibujar se encuentran en una "Tabla de Figuras" cuya estructura se describe en los Comentarios que hay después del programa.

Requisitos de entrada: Ninguno. El programa inicializa los registros por sí solo; pero consulta los Comentarios posteriores.

Condiciones de salida: Todos los registros son alterados.

Longitud: 99 bytes.

```
                    5 ;       ** FIGRAF **
9D08                10        ORG   40200
9D08  DD216B9D      20        LD    IX,FIGURA ; Direccion de los datos
9D0C  DD7E00        30 BUCLE  LD    A,(IX)    ; Primer byte de los 5 que formaran
                    40 ;                        un codigo de operacion
9D0F  FE00          50        CP    0
9D11  C8            60        RET   Z
9D12  FE01          70        CP    1
9D14  CC3B9D        80        CALL  Z,MOV     ; CALL cuando sea apropiado
9D17  FE02          90        CP    2
9D19  CC449D       100        CALL  Z,DIBUJA
9D1C  FE03         110        CP    3
9D1E  CC2A9D       120        CALL  Z,COLOR
9D21  FE04         130        CP    4
9D23  CC4D9D       140        CALL  Z,PUNTO
9D26  DD23         150        INC   IX        ; Incremento de IX para que
                   160 ;                        apunte hacia el siguiente codigo
                   170 ;                        de operacion
9D28  18E2         180        JR    BUCLE
9D2A  F5           190 COLOR  PUSH  AF        ; Guarda AF hasta la salida;
                   200 ;                        el registro A todavia conserva
                   210 ;                        el codigo de operacion
9D2B  DD23         220        INC   IX
9D2D  DD7E00       230        LD    A,(IX)    ; Asigna el color
9D30  CDDEBB       240        CALL  #BBDE     ; Impone esa tinta para los graficos
9D33  DD23         250        INC   IX
9D35  DD23         260        INC   IX
9D37  DD23         270        INC   IX        ; Actualiza el valor de IX
9D39  F1           280        POP   AF
9D3A  C9           290        RET
9D3B  F5           300 MOV    PUSH  AF
```

```
9D3C  CD569D     310         CALL  COORDS
9D3F  CDC0BB     320         CALL  #BBC0     ; Movimiento absoluto
9D42  F1         330         POP   AF
9D43  C9         340         RET
9D44  F5         350 DIBUJA  PUSH  AF
9D45  CD569D     360         CALL  COORDS
9D48  CDF6BB     370         CALL  #BBF6     ; Traza una linea
9D4B  F1         380         POP   AF
9D4C  C9         390         RET
9D4D  F5         400 PUNTO   PUSH  AF
9D4E  CD569D     410         CALL  COORDS
9D51  CDEABB     420         CALL  #BBEA     ; Puntea en una posicion absoluta
9D54  F1         430         POP   AF
9D55  C9         440         RET
9D56  DD23       450 COORDS  INC   IX         ; Rutina que carga las coordenadas
                 460 ;                         X e Y en los registros DE y HL
                 470 ;                         para su uso por las rutinas de ROM
9D58  DD5E00     480         LD    E,(IX)
9D5B  DD23       490         INC   IX
9D5D  DD5600     500         LD    D,(IX)    ; Coordenada X en DE
9D60  DD23       510         INC   IX
9D62  DD6E00     520         LD    L,(IX)
9D65  DD23       530         INC   IX
9D67  DD6600     540         LD    H,(IX)    ; Coordenada Y en HL
9D6A  C9         550         RET
9D6B  02         560 FIGURA  DEFB  2          ; A partir de aqui son datos
9D6C  C800       570         DEFW  200
9D6E  0000       580         DEFW  0
9D70  03         590         DEFB  3
9D71  0300       600         DEFW  3
9D73  0000       610         DEFW  0
9D75  02         620         DEFB  2
9D76  C800       630         DEFW  200
9D78  C800       640         DEFW  200
9D7A  03         650         DEFB  3
9D7B  0400       660         DEFW  4
9D7D  0000       670         DEFW  0
9D7F  02         680         DEFB  2
9D80  0000       690         DEFW  0
9D82  C800       700         DEFW  200
9D84  03         710         DEFB  3
9D85  0500       720         DEFW  5
9D87  0000       730         DEFW  0
9D89  02         740         DEFB  2
9D8A  0000       750         DEFW  0
9D8C  0000       760         DEFW  0
9D8E  00         770         DEFB  0 ; Final
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 0
20 CALL 40200
30 FOR T=1 TO 1000:NEXT:MODE 2
40 CALL 40200
```

Comentarios

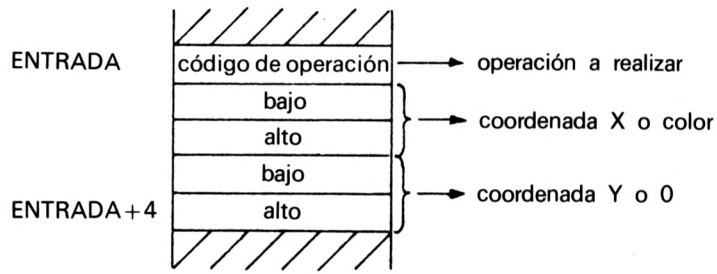
La dirección de la Tabla de Figuras se carga al comienzo del programa en el registro IX. Esto se puede alterar, de forma que cuando llames al programa puedas suministrar la dirección de tu Tabla de Figuras al mismo. La rutina es relocalizable en el depósito de memoria teniendo en cuenta, por supuesto, que hay que cambiar las direcciones que contienen

las instrucciones CALL de la rutina, refiriéndome con ésta a las llamadas a subrutinas del programa y no a las que se hacen al sistema operativo del ordenador.

La rutina comenzará consultando las instrucciones almacenadas en la Tabla de Figuras. Las operaciones se realizarán con el color y posición del cursor de gráficos en el estado en que se encontraba antes de entrar en la rutina. Por eso, si al llamar a la rutina tienes alguna duda sobre ese estado, tendrás que asegurarte de que el primer par de instrucciones de la Tabla de Figuras inicializan apropiadamente el color y la posición del cursor de gráficos.

La Tabla de Figuras

Consiste en unas series de cinco bytes con el formato de la siguiente figura.



La introducción de cinco bytes se repite en cada operación realizada por FIGRAF. Los códigos de operación que entiende esta rutina son los que se muestran en la siguiente tabla:

Tabla de códigos de operación de FIGRAF

Código de operación	Operación
0	final
1	movimiento absoluto
2	trazo absoluto
3	color
4	punteo absoluto

Cualquier secuencia de operaciones terminará con una entrada de cinco bytes puestos a 0, indicando con ello el final del grupo de instrucciones. Para las operaciones de mover el cursor, dibujar o puntear, las coordenadas X e Y estarán almacenadas en dos pares de bytes con el byte bajo en primer lugar. Para color —que especificará cuál de los colores

disponibles se debe usar a continuación— el byte bajo de la coordenada X se usa para guardar el código del color. Este código coincide con la pluma de gráficos que quieras utilizar, y no con el número asignado a cada color. En el caso de color, todas las demás entradas estarán a cero.

Es fácil añadir más códigos de operación a la rutina; y si, por ejemplo, quisieras incluir instrucciones de movimientos o trazos de líneas relativos, te serán de ayuda las siguientes recomendaciones.

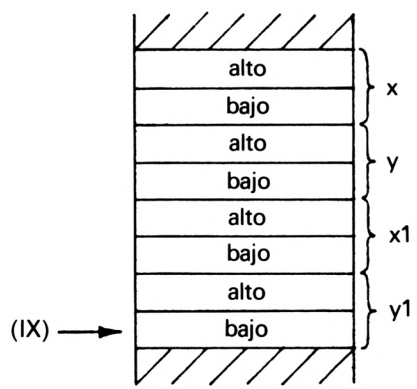
1. El registro AF siempre tiene que preservarse al entrar en la rutina del nuevo código de operación. Esto nos garantiza que el registro A pueda ser devuelto a su estado original al finalizar la rutina. Esto es importante, sobre todo si el programa puede llamar a una segunda subrutina antes de actualizar el registro IX y antes de buscar un nuevo código de operación.
2. Probablemente sea una buena idea dejar el 0 como último código de operación, a menos que quieras alterar el programa.

Voy a mostrarte ahora algunas rutinas dedicadas a la construcción de figuras, tales como triángulos y rectángulos.

TRIANG (TRIANGulo)

Esta rutina dibuja un triángulo, cuyos vértices son la última posición que ocupó el cursor de gráficos antes de llamar a la rutina, y los dos puntos transferidos como parámetros a la rutina en código máquina. El color del triángulo es el definido para los gráficos.

Requisitos de entrada: Si utilizamos una sentencia CALL, la forma correcta será CALL dirección,x,y,x1,y1, siendo los puntos x,y y x1,y1 los vértices del triángulo.
Si partimos de otra rutina en código máquina, entonces el registro IX apunta hacia un bloque de parámetros como el de la figura. Además A=4.



Condiciones de salida: Todos los registros son alterados.

Longitud: 44 bytes.

```
                    10 ;       ** TRIANG **
9D08                20         ORG  40200
9D08  FE04          30         CP   4      ; Tiene que haber 4 parametros
9D0A  C0            40         RET  NZ
9D0B  CDC6BB        50         CALL #BBC6 ; Obtiene la posicion actual del cursor
9D0E  D5            60         PUSH DE    ; Guarda el valor de X
9D0F  E5            70         PUSH HL    ; Guarda el valor de Y
9D10  CD249D        80         CALL TRAZA ; Linea hasta el punto X1,Y1
9D13  DD23          90         INC  IX    ; Actualiza IX para que apunte
                   100 ;                     hacia los parametros X e Y
9D15  DD23         110         INC  IX
9D17  DD23         120         INC  IX
9D19  DD23         130         INC  IX
9D1B  CD249D       140         CALL TRAZA ; Linea hasta el punto X,Y
9D1E  E1           150         POP  HL    ; Rescata los valores originales de X e Y
9D1F  D1           160         POP  DE
9D20  CDF6BB       170         CALL #BBF6 ; Ultimo trazo hasta ese punto
9D23  C9           180         RET
9D24  DD6E00       190 TRAZA   LD   L,(IX)
9D27  DD6601       200         LD   H,(IX+1)
9D2A  DD5E02       210         LD   E,(IX+2)
9D2D  DD5603       220         LD   D,(IX+3)
9D30  CDF6BB       230         CALL #BBF6
9D33  C9           240         RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 REM Rutina que dibuja triangulos.
30 CLS
40 INPUT " Primer punto X,Y ",X,Y
50 INPUT " Segundo punto X1,Y1 ",X1,Y1
60 CALL 40200,X,Y,X1,Y1
70 LOCATE 1,1
80 GOTO 40
```

Comentarios

El programa es relocalizable a lo largo de todo el depósito de memoria. Cuando el triángulo se dibuje, el cursor de gráficos se encontrará en la posición que ocupaba antes de que la llamada a la rutina fuera realizada.

CAJA

Esta rutina dibuja figuras cuadradas o rectangulares en la pantalla. Las figuras pueden ser rellenadas o no, a voluntad. Tanto el contorno como el relleno tendrán el color asignado a los gráficos. La rutina es relocalizable, pero poniendo cuidado en las direcciones de las subrutinas usadas.

Requisitos de entrada: Para la sentencia CALL, usaremos el modelo: CALL dirección,x,y, longitud, altura,(n). El parámetro n es opcional, y si está presente se llenará la caja dibujada. Su valor carece de importancia. Para aclararte sobre los demás parámetros observa la figura 1.

Si partimos de otra rutina en código máquina, IX deberá apuntar hacia uno de los bloques de parámetros mostrados en la figura 2.

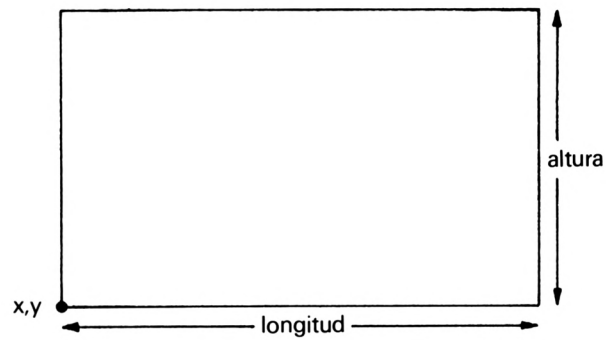


Figura 1

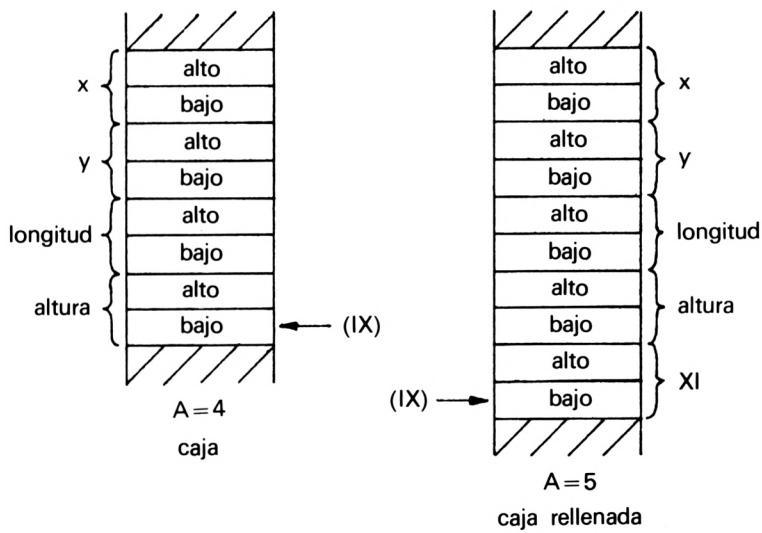


Figura 2

Condiciones de salida: Todos los registros son alterados.

Longitud: 149 bytes.

```
                10 ;       ** CAJA **
9D08            20         ORG  40200
9D08  FE04      30         CP   4
9D0A  2805      40         JR   Z,VACIA
9D0C  FE05      50         CP   5          ; Si hay 5 parametros, la caja sera
                60 ;                         rellenada
9D0E  2848      70         JR   Z,LLENA
9D10  C9        80         RET
9D11  CD489D    90 VACIA   CALL MOV
9D14  DD5E02   100         LD   E,(IX+2)
9D17  DD5603   110         LD   D,(IX+3)
9D1A  210000   120         LD   HL,0
9D1D  CDF9BB   130         CALL #BBF9      ; Dibuja el borde inferior
9D20  110000   140         LD   DE,0
9D23  DD6E00   150         LD   L,(IX)
9D26  DD6601   160         LD   H,(IX+1)
9D29  CDF9BB   170         CALL #BBF9      ; Dibuja el borde derecho
9D2C  CD489D   180         CALL MOV
9D2F  110000   190         LD   DE,0
9D32  DD6E00   200         LD   L,(IX)
9D35  DD6601   210         LD   H,(IX+1)
9D38  CDF9BB   220         CALL #BBF9      ; Dibuja el borde izquierdo
9D3B  210000   230         LD   HL,0
9D3E  DD5E02   240         LD   E,(IX+2)
9D41  DD5603   250         LD   D,(IX+3)
9D44  CDF9BB   260         CALL #BBF9      ; Dibuja el borde superior
9D47  C9       270         RET
9D48  DD6E04   280 MOV     LD   L,(IX+4)
9D4B  DD6605   290         LD   H,(IX+5)
9D4E  DD5E06   300         LD   E,(IX+6)
9D51  DD5607   310         LD   D,(IX+7)
9D54  CDC0BB   320         CALL #BBC0      ; Se mueve a la posicion inicial
9D57  C9       330         RET  ;            de la caja
9D58  CDCCBB   340 LLENA   CALL #BBCC
9D5B  D5       350         PUSH DE
9D5C  E5       360         PUSH HL         ; Guarda las coordenadas del origen
                370 ;                        del cursor de graficos
9D5D  DD6E06   380         LD   L,(IX+6)
9D60  DD6607   390         LD   H,(IX+7)
9D63  DD5E08   400         LD   E,(IX+8)
9D66  DD5609   410         LD   D,(IX+9)
9D69  CDC9BB   420         CALL #BBC9      ; Mueve el origen de coordenadas
                430 ;                        hasta la posicion de la caja
9D6C  DD6E02   440         LD   L,(IX+2)
9D6F  DD6603   450         LD   H,(IX+3); Extrae el numero
9D72  E5       460         PUSH HL         ; de lineas necesarias para
9D73  C1       470         POP  BC         ; rellenar la caja y lo pone en BC
9D74  210000   480         LD   HL,0
9D77  DD5E04   490 BUCLE   LD   E,(IX+4); Traza una serie de lineas
9D7A  DD5605   500         LD   D,(IX+5); de longitud L, que rellenaran
9D7D  C5       510         PUSH BC         ; la caja
9D7E  E5       520         PUSH HL
9D7F  210000   530         LD   HL,0
9D82  CDF9BB   540         CALL #BBF9
9D85  E1       550         POP  HL
9D86  23       560         INC  HL
9D87  E5       570         PUSH HL
9D88  110000   580         LD   DE,00
9D8B  CDC0BB   590         CALL #BBC0
9D8E  E1       600         POP  HL
9D8F  C1       610         POP  BC
9D90  0B       620         DEC  BC
9D91  79       630         LD   A,C
9D92  B0       640         OR   B
9D93  FE00     650         CP   0
```

```
9D95  20E0      660       JR   NZ,BUCLE; Si no estan todas, empieza
9D97  E1        670       POP  HL       ; otra vez; cuando termina,
9D98  D1        680       POP  DE       ; restaura el origen de graficos
9D99  CDC9BB    690       CALL #BBC9
9D9C  C9        700       RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 REM Rutina que dibuja y rellena
   cajas cuadradas o rectangulares
20 CALL 40200,100,100,400,200
30 CALL 40200,150,150,300,100
40 CALL 40200,200,170,200,60,1
50 END
```

Comentarios

Existen algunos puntos interesantes de esta rutina, que merece la pena comentar. El primero es el empleo del número de parámetros transferidos a la rutina, para especificar cuál de las dos opciones, "lleno" o "vacío", ha de realizarse. Esta indicación se consigue fácilmente gracias a que el número de parámetros transferidos se almacena en el registro A al entrar en el código máquina mediante la orden CALL. De ahí que el valor del parámetro "n" no tenga ninguna importancia; únicamente su presencia provoca que la rutina actúe de una u otra forma.

Otros puntos de interés sobre esta rutina son las subrutinas de la ROM utilizadas, y el método empleado para rellenar las cajas cuando es necesario.

&BBCC: Esta rutina nos permite conocer la posición actual del origen de coordenadas para las operaciones de gráficos. Esto es, el punto usado como 0,0 en dichas operaciones. Al regreso, HL contiene la coordenada Y, y el par DE contiene la coordenada X. Esta información nos es necesaria porque, cuando dibujamos el relleno de las cajas, alteramos la posición del origen, y, además, es bueno volver las cosas a su estado original antes de volver al BASIC.

&BBC9: Esta nos permite definir el origen de coordenadas de los gráficos en un determinado punto x,y. El registro DE contiene la coordenada X y el registro HL la coordenada Y. Después de cumplir estos requisitos, podemos llamar a la rutina. Este programa la utiliza para restaurar el origen de gráficos en su posición original, justo antes de terminar la rutina.

Aunque en la memoria ROM del Amstrad hay una rutina especial para rellenar áreas de pantallas con un color, esta rutina requiere que las

coordenadas de los puntos se conviertan en direcciones de la memoria de pantalla. Así que decidí tomar un camino más fácil y opté por dibujar simplemente líneas que rellenaran el área de la caja. No es tan rápido como la rutina interna de llenado, pero es más sencillo de entender y de llevar a cabo.

Como una posible extensión de este problema puedes intentar que el parámetro "n" especifique el color con el que se dibuje el relleno de una caja. El color se puede activar mediante la rutina GRAPLUMA que expuse brevemente en el capítulo 2.

Dibujando círculos

La rapidez al dibujar un círculo es una exigencia muy común al programar. Sin embargo, se presentan varios problemas al usar el código máquina para este fin. El trabajo con números reales para el cálculo de senos y cosenos en las rutinas que dibujan círculos, se complica en gran medida. Esto puede originar que las rutinas que escribamos para calcular estos valores se hagan realmente largas y resulten apenas más rápidas que el BASIC. Por eso, quiero presentar aquí un par de rutinas en BASIC para dibujar círculos, igualmente útiles y de propósito general. El tercero es un programa híbrido que emplea BASIC en combinación con código máquina, y que se puede emplear en casos especiales.

CIRCULO 1

Esta es una rutina directa para dibujar círculos y la mayoría de los polígonos, dándole esta última función especial utilidad. En el programa, cualquier valor de "n", a excepción de 0, 1 y 2, dará una figura poligonal. Observa que hay valores como n=5, n=10 o n=15, que dan figuras con un tramo sin dibujar. Los valores de "n" superiores a 25 darán un círculo más o menos razonable.

```
10 REM      Circulo 1
20 REM Dibuja poligonos
30 REM Los valores de N mayores que 25
40 REM generaran un circulo
50 REM Cuanto mayor sea N , mas tiempo
60 REM llevara el dibujo, pero mejor
70 REM definido estara el circulo
80 MODE 2
90 INPUT "Numero de lados  ",N
100 XC=200  : REM Centro del circulo
110 YC=200
120 CSALTO=6.26/N :REM Incremento
```

```
130 CFIN=6.28
140 CRAD=100 : REM Radio de la figura
150 MOVE XC+CRAD,YC
160 FOR I=0 TO CFIN STEP CSALTO
170 DRAW XC+CRAD*COS(I),YC+CRAD*SIN(I)
180 REM Realizacion de la figura
190 NEXT
200 END
```

CIRCULO 2

Esta rutina es más rápida que la anterior, pero no tan versátil, ya que no puede dibujar otros polígonos. De hecho, el programa CIRCULO1 es digno de un pequeño experimento. La velocidad extra de este programa se debe a que el tiempo consumido en el cálculo de senos y cosenos se hace una sola vez. Los demás valores trigonométricos que se requieren son calculados a partir de los valores iniciales.

```
10 REM Circulo 2
20 MODE 2
30 REM Dibuja los circulos mas rapido
40 REM pero no dibuja poligonos.
50 CRAD=100 : REM Radio del circulo.
60 CX=320 : REM Centro del circulo
70 CY=200
80 C=COS(3.14/25) : S=SIN(3.14/25)
90 CVIEJO=1 : SVIEJO=0
100 MOVE CX+CRAD*CVIEJO,CY+CRAD*SVIEJO
110 FOR I=1 TO 50
120 CNUEVO=CVIEJO*C-SVIEJO*S
130 SNUEVO=SVIEJO*C+CVIEJO*S
140 DRAW CX+CRAD*CNUEVO,CY+CRAD*SNUEVO
150 CVIEJO=CNUEVO : SVIEJO=SNUEVO
160 NEXT
```

CIRCULO 3

Llegamos ahora a un método un tanto híbrido de dibujar círculos, que usa eficazmente una variación del programa FIGRAF que vimos al comienzo de este capítulo. Una rutina en BASIC calcula las coordenadas del círculo que se quiere dibujar, y almacena los valores así obtenidos en una tabla desde donde el código máquina accede a las coordenadas y las dibuja en la pantalla cuando se precisa. La desventaja de este método es

que sólo es útil para un radio en particular y una determinada posición en la pantalla. Para dibujar diferentes círculos es necesario redefinir los valores de la tabla usados por el programa, volviendo a recorrer la parte BASIC del programa con los nuevos parámetros.

```
9D08              10        ORG  40200
9D08  DD216C9D    20        LD   IX,40300
9D0C  DD5E00      30        LD   E,(IX)
9D0F  DD5601      40        LD   D,(IX+1)
9D12  DD6E02      50        LD   L,(IX+2)
9D15  DD6603      60        LD   H,(IX+3)
9D18  CDC0BB      70        CALL #BBC0
9D1B  DD23        80        INC  IX
9D1D  DD23        90        INC  IX
9D1F  DD23       100        INC  IX
9D21  DD23       110        INC  IX
9D23  0632       120        LD   B,50
9D25  DD5E00     130 BUCLE  LD   E,(IX)
9D28  DD23       140        INC  IX
9D2A  DD5600     150        LD   D,(IX)
9D2D  DD23       160        INC  IX
9D2F  DD6E00     170        LD   L,(IX)
9D32  DD23       180        INC  IX
9D34  DD6600     190        LD   H,(IX)
9D37  DD23       200        INC  IX
9D39  C5         210        PUSH BC
9D3A  CDF6BB     220        CALL #BBF6
9D3D  C1         230        POP  BC
9D3E  10E5       240        DJNZ BUCLE
9D40  C9         250        RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 2
20 REM        Circulo 3
30 REM Dibuja circulos rapidamente.
40 REM La posicion del circulo es fija
50 REM y el radio es fijo.
60 CRAD=100 : REM Radio del circulo.
70 CX=100 : REM Centro del circulo.
80 CY=100
90 C=COS(3.14/25) : S=SIN(3.14/25)
100 CVIEJO=1 : SVIEJO=0
110 COORDS=40300
120 XCOORD=CX+CRAD*CVIEJO : GOSUB 250
130 XCOORD=CY+CRAD*SVIEJO : GOSUB 250
140 FOR I=1 TO 50
150 CNUEVO=CVIEJO*C-SVIEJO*S
160 SNUEVO=SVIEJO*C+CVIEJO*S
170 XCOORD=CX+CRAD*CNUEVO : GOSUB 250
180 XCOORD=CY+CRAD*SNUEVO : GOSUB 250
190 CVIEJO=CNUEVO : SVIEJO=SNUEVO
200 NEXT
210 CLS
220 INPUT"Pulsa ENTER para dibujar.",a$
```

```
230 CALL 40200
240 END
250 A$=HEX$(XCOORD)
260 A$=RIGHT$("0000"+A$,4)
270 BB=VAL("&"+RIGHT$(A$,2))
280 BA=VAL("&"+LEFT$(A$,2))
290 POKE COORDS,BB : COORDS=COORDS+1
300 POKE COORDS,BA : COORDS=COORDS+1
310 RETURN
```

El listado del lenguaje ensamblador de este programa precisa que las coordenadas para el círculo se almacenen en memoria a partir de la dirección 40300. La rutina de dibujo del círculo calcula las primeras coordenadas X e Y, y las sigue utilizando para moverse a partir de ellas hasta un punto que diste un radio del centro del círculo. El siguiente programa almacenará la información de las coordenadas y llamará a la rutina que dibuja el círculo. El programa espera encontrar a esta rutina en la dirección 40200, con los datos en la dirección 40300.

```
10 CRAD=100 : REM Radio del circulo.
20 CX=100 : REM Centro del circulo.
30 CY=100
40 C=COS(3.14/25) : S=SIN(3.14/25)
50 CVIEJO=1 : SVIEJO=0
60  COORDS=40300 : REM Direccion de
                          la tabla.
70 XCOORD=CX+CRAD*CVIEJO : GOSUB 250
80 XCOORD=CY+CRAD*SVIEJO : GOSUB 250
90  FOR I=1 TO 50
100 CNUEVO=CVIEJO*C-SVIEJO*S
110 SNUEVO=SVIEJO*C+CVIEJO*S
120 XCOORD=CX+CRAD*CNUEVO : GOSUB 250
130 XCOORD=CY+CRAD*SNUEVO : GOSUB 250
140 CVIEJO=CNUEVO : SVIEJO=SNUEVO
150 NEXT
160 CLS
170 INPUT"Pulsa ENTER para dibujar.",a
180 CALL 40200 : REM Presumiendo que la
190 REM rutina esta en esta direccion.
200 END
210 CLS
220 INPUT"Pulsa ENTER para dibujar.",a$
230 CALL 40200
240 END
250 REM Subrutina que se estudia en los
260 REM comentarios posteriores
270 A$=HEX$(XCOORD)
```

```
280 A$=RIGHT$("0000"+A$,4)
290 BB=VAL("&"+RIGHT$(A$,2))
300 BA=VAL("&"+LEFT$(A$,2))
310 POKE COORDS,BB
320 COORDS=COORDS+1
330 POKE COORDS,BA
340 COORDS=COORDS+1
350 RETURN
```

La subrutina de la línea 200 es muy útil para almacenar en memoria números decimales con el formato "primero el byte bajo" que el Z80 espera encontrar en todos sus datos. En este caso se usa para almacenar en memoria la información sobre las coordenadas, de forma que pueda ser utilizada por la rutina en código máquina para el dibujo. Probablemente puedas ver en el listado el funcionamiento exacto de esta rutina: convierte los números en una cadena de caracteres que representan al número en hexadecimal. Después utiliza las órdenes RIGHT$ y LEFT$ para extraer los bytes bajo y alto de cada número.

Con esto se completa este capítulo de rutinas de gráficos. Sin embargo, muchas de las rutinas mencionadas en los dos capítulos siguientes están orientadas gráficamente. Por eso ¡no te asustes si todavía no has encontrado exactamente lo que quieres!

4
Desplazamiento de pantalla

Antes de nada, ¿qué es un desplazamiento? Pues es el hecho de trasladar toda una pantalla o parte de la misma, de una vez, conservando la imagen expuesta. Esto nos permite hacer cosas muy interesantes; la pantalla puede moverse hacia arriba o hacia abajo, de un lado a otro, y también podemos conseguir que una simple palabra o un símbolo gráfico atraviesen la pantalla. Existen rutinas en el *firmware* que realizan simples desplazamientos verticales y horizontales, y en este capítulo veremos algunas rutinas que las utilizan, así como otras que para desplazar trabajarán accediendo directamente a la memoria de pantalla.

Hay dos tipos principales de desplazamiento: el desplazamiento *software*, que se realiza sin ayuda del *hardware* encargado de generar todo lo relativo a la pantalla, y el desplazamiento *hardware*, en el que el *hardware* que genera la imagen de la pantalla se manipula de algún modo.

Empezaremos con una rutina muy simple, que desplaza toda la pantalla hacia arriba o hacia abajo.

VDESPLZ (DESPLaZamiento Vertical)

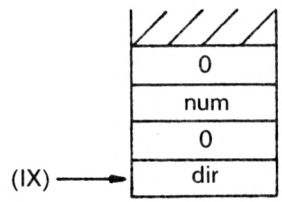
Es una rutina que desplaza la totalidad de la pantalla hacia arriba o hacia abajo un número dado de líneas de caracteres, en cualquier modo de pantalla. La rutina es relocalizable, y las líneas de texto que se desplazan

más allá del margen superior o inferior de la pantalla se pierden definitivamente. Las líneas insertadas que sustituyen a las otras ya perdidas se rellenan con el color asignado al papel.

Requisitos de entrada: Desde el BASIC accederemos mediante CALL dirección, num, dir, donde num es el número de líneas que quieres desplazar, y dir puede ser 0 o 1. Si dir=1, la pantalla se desplazará hacia arriba, y si dir=0, lo hará hacia abajo.
Desde código máquina, A=2 y IX apunta a un bloque de parámetros; num deberá estar comprendido en el intervalo (0 a 255), aunque los valores grandes simplemente limpiarán la pantalla.

Condiciones de salida: Todos los registros alterados.

Longitud: 22 bytes.



Bloque de parámetros de VDESPLZ

```
                    10 ;       ** VDESPLZ **
9D08                20         ORG  40200
9D08  CD99BB        30         CALL #BB99    ; Obtiene el color del papel de textos
9D0B  CD2CBC        40         CALL #BC2C    ; Codifica el color para mas tarde
9D0E  DD4602        50         LD   B,(IX+2); Numero de desplazamientos
9D11  F5            60 BUCLE   PUSH AF       ; Preserva los registros
9D12  C5            70         PUSH BC
9D13  DD4600        80         LD   B,(IX+0); Hacia arriba o hacia abajo?
9D16  CD4DBC        90         CALL #BC4D    ; Realiza el desplazamiento
9D19  C1           100         POP  BC
9D1A  F1           110         POP  AF
9D1B  10F4         120         DJNZ BUCLE    ; Repite hasta que se hagan todos
9D1D  C9           130         RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 2
20 LOCATE 1,10
30 PRINT ".....HOLA"
40 PRINT ".....QUE TAL"
50 PRINT ".....POR"
60 PRINT ".....AHI ?"
```

```
70 CALL 40200,1,dir
80 a$=INKEY$ : IF a$="" GOTO 80
90 a=ASC(a$)
100 IF a=241 THEN dir=0
110 IF a=240 THEN dir=1
120 GOTO 70
```

Comentarios

Aquí se utiliza una rutina muy útil del *firmware* para desplazar toda la pantalla, a la que se accede llamando a la dirección &BC4D. Al entrar, si B=0, la pantalla se desplaza una línea hacia abajo, y si B=1 entonces se moverá una línea hacia arriba. El siguiente programa en BASIC también demuestra el funcionamiento del anterior código máquina.

```
10 MODE 2
20 FOR X=1 TO 80 STEP 10
30 FOR Y=1 TO 25
40 LOCATE X,Y
50 PRINT Y
60 NEXT : NEXT
70 CALL 40200,1,dir
80 REM Suponiendo al codigo en 40200
90 a$=INKEY$ : IF a$="" GOTO 90
100 a=ASC(a$)
110 IF a=241 THEN dir=0 : REM abajo
120 IF a=240 THEN dir=1 : REM arriba
130 GOTO 70
```

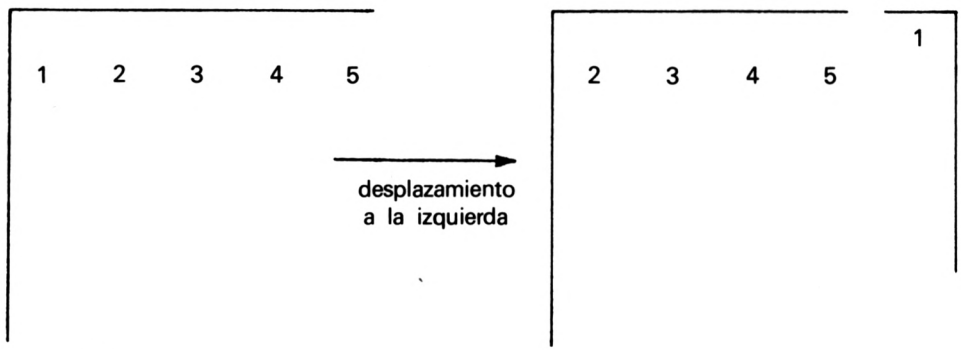
Pulsando las teclas que dirigen el cursor hacia arriba o hacia abajo conseguiremos mover la pantalla. El texto desplazado fuera de la pantalla se pierde definitivamente.

Pero sería bastante más útil poder desplazar la pantalla hacia los lados. Esto conlleva algo más de trabajo, pues tenemos que producir una rutina ligeramente distinta para cada modo de pantalla. Antes de examinar estas rutinas con detalle, daré unas notas de carácter general.

Desplazamiento lateral

Todas estas rutinas desplazan la mayor parte de la pantalla, pero no toda. Las líneas 0 y 24 de la pantalla se utilizan como espacio de trabajo por las rutinas y, por ello, no conviene utilizarlas cuando trabajes con dichas rutinas. A continuación te explico la razón.

Estamos manejando una mezcla de desplazamientos *software* y *hardware* para conseguir los resultados deseados. Los desplazamientos laterales mediante *hardware* resultan sencillos alterando lo que se conoce como offset de pantalla. No dispongo de espacio en este libro para entrar en detalles, por lo que me remito al manual *firmware* o a cualquier otro trabajo similar.

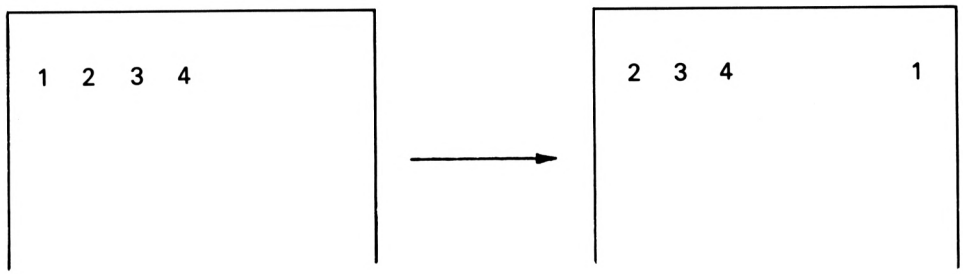


Puedes ver en la figura anterior de qué forma afecta a la imagen un desplazamiento lateral con *hardware*. El primer carácter de la línea se traslada al borde opuesto de la pantalla y el resto de los caracteres se mueven un determinado espacio hacia la izquierda. Tal y como está diseñado el *hardware* del Amstrad, éste valor depende del offset de pantalla de la siguiente manera.

Incrementando el offset de pantalla conseguimos un desplazamiento a la derecha, y disminuyéndolo obtendremos un desplazamiento a la izquierda. El valor actual del offset de pantalla puede obtenerse usando una rutina del *firmware*, como enseguida veremos. Una segunda rutina puede ser empleada para volver a escribir el offset de pantalla modificado en los circuitos de video. En estos tres programas, el offset en modo 1 y 2 se modifica en 2 unidades cada vez. En modo 1 esto provoca el desplazamiento de un carácter. En modo 2 se desplazarán dos caracteres. Así, para desplazar un carácter de la pantalla en modo 2 hacia la izquierda sólo tienes que dividir por dos el offset. En modo 0 el offset de pantalla se modifica en cuatro unidades cada vez que se requiere un desplazamiento. Con ello la pantalla se traslada un carácter. La razón de que se requieran distintos offsets para cada modo de pantalla a la hora de desplazar la misma se debe a la distinta distribución de la memoria del vídeo para cada modo de pantalla.

Sin embargo, si eres observador, habrás advertido que el desplazamiento que os mostré más arriba no era un verdadero desplazamiento lateral; el carácter desplazado en la pantalla se hallaba una línea más arriba o más abajo, según el sentido del desplazamiento, de aquella en la que originalmente estaba. Por ejemplo, de un desplazamiento hacia la derecha

con este método, resultaría que un carácter se saldría del borde derecho de la pantalla, reapareciendo en el borde izquierdo, pero una línea más abajo. Para un verdadero desplazamiento lateral se requiere que el material desbordado por un lado de la pantalla reaparezca por el otro lado, pero en la misma línea, de la siguiente manera:



Esto puede hacerse usando la rutina SWSCROLL presentada en el *firmware* del Amstrad. Con ella conseguimos desplazar un área en particular de la pantalla, hacia arriba o hacia abajo el espacio de un carácter. Antes de examinar la rutina de desplazamiento, veremos brevemente las rutinas del *firmware* que vamos a utilizar.

&BC0B: Esta rutina devuelve el valor actual del offset de pantalla al par de registros HL. Entonces podrá modificarse y reenviarse al circuito de vídeo.

&BC05: Esta rutina te permite dar al offset de pantalla un valor a tu elección. A la entrada, el par de registros HL deberá contener el valor del offset deseado.

&BC50: Esta rutina, llamada SWSCROLL, te permite desplazar verticalmente una línea dada de la pantalla. A la entrada, B=0 para un desplazamiento hacia abajo, o 1 para un movimiento hacia arriba. H contiene el borde izquierdo del área a desplazar, L la fila superior, D el borde derecho y E la fila inferior. Todos ellos se representan en términos de espacio de caracteres, empezando en 0,0, que es la esquina superior izquierda de la pantalla. El registro A contiene el código de la tinta que se empleará para rellenar la línea que se desplaza. En nuestras rutinas utilizaremos el color del papel de texto para rellenar la línea desplazada.

&BB99: Al regreso de esta rutina el registro A contiene el color actual del papel para el texto. Antes de usar este valor devuelto por la rutina SWSCROLL debemos codificarlo llamando a la rutina de la dirección &BC2C.

Con un examen de los listados para estas tres rutinas de desplazamiento lateral comprobaremos que utilizamos la rutina SWSCROLL para

"alinear" las columnas de la imagen que han sido expulsadas por un lado de la pantalla y reintegradas por el lado opuesto.

LDESP2 (DESPlazamiento Lateral 2)

Desplaza lateralmente dos espacios una pantalla en modo 2. Las líneas 0 y 24 de la pantalla no son desplazadas en el sentido propio de la palabra. La rutina puede relocalizarse siempre que la dirección del espacio de trabajo se altere de forma que no coincida con el programa. Los bytes proporcionados funcionan únicamente en la dirección 40200.

Requisitos de entrada: Desde BASIC: CALL dirección, dir, donde dir = 1 para un desplazamiento a la izquierda y dir = 0 para un desplazamiento a la derecha. Desde código máquina, A = 1 y IX apunta a un solo byte que contiene a dir.

Condiciones de salida: Todos los registros alterados.

Longitud: 90 bytes.

```
                    10 ;      ** LDESP2 **
9D08                20        ORG  40200
9D08  F3            30        DI   ;          Prohibe interrupciones
9D09  CD99BB        40        CALL #BB99
9D0C  CD2CBC        50        CALL #BC2C
9D0F  325B9D        60        LD   (PAPEL),A; Guarda el color del papel
9D12  DD7E00        70        LD   A,(IX)   ; Decide si es a izquierda o derecha
9D15  FE00          80        CP   0
9D17  2820          90        JR   Z,OTRO
9D19  264E         100        LD   H,78
9D1B  2E00         110        LD   L,0
9D1D  164F         120        LD   D,79
9D1F  1E18         130        LD   E,24
9D21  0600         140        LD   B,0
9D23  C5           150        PUSH BC       ; Preparado para desplazar,
9D24  E5           160        PUSH HL       ; guarda los registros
9D25  D5           170        PUSH DE       ; en la pila
9D26  CD0BBC       180        CALL #BC0B    ; Obtiene el offset
9D29  23           190        INC  HL
9D2A  23           200        INC  HL       ; Lo actualiza
9D2B  CD05BC       210        CALL #BC05    ; y vuelve a activarlo
9D2E  3A5B9D       220        LD   A,(PAPEL)
9D31  D1           230        POP  DE
9D32  E1           240        POP  HL
9D33  C1           250        POP  BC
9D34  CD50BC       260        CALL #BC50    ; Desplaza una columna,
                   270 ;                      el espacio de un caracter
9D37  FB           280        EI   ;          Permite las interrupciones
9D38  C9           290        RET
9D39  2600         300 OTRO   LD   H,0       ; Prepara los registros para
9D3B  2E00         310        LD   L,0       ; el desplazamiento de columnas
9D3D  1601         320        LD   D,1       ; al final de la rutina
9D3F  1E18         330        LD   E,24
9D41  0601         340        LD   B,1
9D43  3A5B9D       350        LD   A,(PAPEL)
9D46  F5           360        PUSH AF
9D47  C5           370        PUSH BC
```

```
9D48  D5          380        PUSH DE
9D49  E5          390        PUSH HL
9D4A  CD0BBC      400        CALL #BC0B    ; Obtiene el offset
9D4D  2B          410        DEC  HL
9D4E  2B          420        DEC  HL       ; Altera el offset y lo
9D4F  CD05BC      430        CALL #BC05    ; envia al 6845
9D52  E1          440        POP  HL       ; (controlador de video)
9D53  D1          450        POP  DE
9D54  C1          460        POP  BC
9D55  F1          470        POP  AF
9D56  CD50BC      480        CALL #BC50    ; Desplaza la columna de la
                  490 ;                      izquierda, una linea
                  500 ;                      hasta el otro extremo
9D59  FB          510        EI
9D5A  C9          520        RET
9D5B  00          530 PAPEL  DEFB 0
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 REM Desplazamiento a izquierda y
   derecha en el modo 2
20 MODE 2
30 ORIGIN 0,32 :REM Origen de graficos
   de la ventana a desplazar
40 WINDOW 1,80,2,24 : REM Ventana de
   texto que se va a desplazar.
50 MOVE 0,350:DRAW 640,350
60 MOVE 0,150:DRAW 640,150
70 LOCATE 1,13
80 PRINT "cuando  esta  frase  termine
   dire      que"
90 CALL &BD19: REM Espera a cada barrido
100 a$=INKEY$:IF a$="" THEN 100
110 IF ASC(a$)=243 THEN CALL 40200,0
120 REM Teclas de desplazamiento
130 IF ASC(a$)=242 THEN CALL 40200,1
140 GOTO 90
```

Comentarios

Las líneas superior e inferior de la imagen no se desplazan propiamente hablando, siendo éste el resultado del modo en que la alteración del offset afecta al material expulsado por un lado de la pantalla. Son sólo las 23 líneas centrales de la imagen las que se desplazan.

En esta rutina se inhiben las interrupciones para conseguir mayor velocidad. Hablando de obtener más velocidad, observa cómo son inicializados los registros para acceder a SWSCROLL nada más comenzar la rutina. Esto reduce el número de instrucciones que deben ejecutarse entre la alteración del offset de pantalla y la llamada a SWSCROLL para limpiar el borde de la pantalla.

LDESP1 y LDESP0 son rutinas similares, pero han sido diseñadas para usarse en modo 1 y 0, respectivamente. La rutina en BASIC que doy a continuación puede usarse para probar el funcionamiento de todas estas rutinas.

```
10 MODE 2
20 ORIGIN 0,32 :REM No utilizar la linea
   inferior de la pantalla.
30 WINDOW 1,80,2,24 : REM Ventana de
   texto que se va a desplazar.
40 MOVE 0,10:DRAW 400,100
50 DRAW 150,50 : DRAW 200,50
60 DRAW 300,60 : DRAW 400,100
70 DRAW 450,20 : DRAW 500,10
80 DRAW 600,150 : DRAW 640,10
90 CALL &BD19: REM Espera al barrido.
100 a$=INKEY$:IF a$="" THEN 100
110 IF ASC(a$)=243 THEN CALL 40200,0
120 REM Teclas de desplazamiento
130 IF ASC(a$)=242 THEN CALL 40200,1
140 GOTO 90
```

Pulsando las teclas de desplazamiento lateral del cursor se conseguirá que la figura de la pantalla se desplace correlativamente.

LDESP0 (DESPlazamiento Lateral 0)

Esta rutina realiza un desplazamiento lateral de un carácter en el modo 0 de pantalla.

Requisitos de entrada: Igual que para LDESP2.

Condiciones de salida: Igual que para LDESP2.

Longitud: 96 bytes.

```
                10 ;      ** LDESP0 **
9D08            20        ORG  40200
9D08  F3        30        DI   ;          Prohibe interrupciones
9D09  CD99BB    40        CALL #BB99
9D0C  CD2CBC    50        CALL #BC2C
9D0F  32679D    60        LD   (PAPEL),A; Guarda el color del papel
9D12  DD7E00    70        LD   A,(IX)   ; Decide si es a izquierda o derecha
9D15  FE00      80        CP   0
9D17  2825      90        JR   Z,OTRO
9D19  2613     100        LD   H,19
9D1B  2E00     110        LD   L,0
9D1D  1613     120        LD   D,19
9D1F  1E18     130        LD   E,24
```

```
9D21  0600      140        LD   B,0
9D23  C5        150        PUSH BC        ; Preparado para desplazar,
9D24  E5        160        PUSH HL        ; guarda los registros
9D25  D5        170        PUSH DE        ; en la pila
9D26  CD0BBC    180        CALL #BC0B     ; Obtiene el offset
9D29  110400    190        LD   DE,4
9D2C  19        200        ADD  HL,DE     ; Lo actualiza
9D2D  3A679D    210        LD   A,(PAPEL); Obtiene el color del papel
9D30  CD05BC    220        CALL #BC05     ; Guarda el offset en el 6845
9D33  3A679D    230        LD   A,(PAPEL); (controlador de video)
9D36  D1        240        POP  DE
9D37  E1        250        POP  HL
9D38  C1        260        POP  BC
9D39  CD50BC    270        CALL #BC50     ; Desplaza una columna,
                280 ;                       el espacio de un caracter
9D3C  FB        290        EI   ;         Permite las interrupciones
9D3D  C9        300        RET
9D3E  2600      310 OTRO   LD   H,0        ; Prepara los registros para
9D40  2E00      320        LD   L,0        ; el desplazamiento de columnas
9D42  1600      330        LD   D,0        ; al final de la rutina
9D44  1E18      340        LD   E,24
9D46  0601      350        LD   B,1
9D48  3A679D    360        LD   A,(PAPEL)
9D4B  F5        370        PUSH AF
9D4C  C5        380        PUSH BC
9D4D  D5        390        PUSH DE
9D4E  E5        400        PUSH HL
9D4F  CD0BBC    410        CALL #BC0B     ; Obtiene el offset
9D52  AF        420        XOR  A
9D53  110400    430        LD   DE,4      ; Altera el offset y lo
9D56  ED52      440        SBC  HL,DE
9D58  3A679D    450        LD   A,(PAPEL)
9D5B  CD05BC    460        CALL #BC05     ; envia al controlador de video
9D5E  E1        470        POP  HL
9D5F  D1        480        POP  DE
9D60  C1        490        POP  BC
9D61  F1        500        POP  AF
9D62  CD50BC    510        CALL #BC50     ; Desplaza la columna de la
                520 ;                       izquierda, una linea
                530 ;                       hasta el otro extremo
9D65  FB        540        EI
9D66  C9        550        RET
9D67  00        560 PAPEL  DEFB 0
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 0:REM Desplazamiento a izquierda
   y derecha en el modo 0
20 ORIGIN 0,32 :REM Origen de graficos
   de la ventana a desplazar.
30 WINDOW 1,20,2,24 : REM Ventana de
   texto que se va a desplazar.
40 MOVE 0,10:DRAW 400,100
50 DRAW 150,50 : DRAW 200,50
60 DRAW 300,60 : DRAW 400,100
70 DRAW 450,20 : DRAW 500,10
80 DRAW 600,150 : DRAW 640,10
90 CALL &BD19: REM Espera al barrido.
100 a$=INKEY$:IF a$="" THEN 100
110 IF ASC(a$)=243 THEN CALL 40200,0
120 REM Teclas de desplazamiento
```

```
130 IF ASC(a$)=242 THEN CALL 40200,1
140 GOTO 90
```

LDESP1 (DESPlazamiento Lateral 1)

Esta rutina realiza un desplazamiento lateral de un espacio en el modo 1.

Requisitos de entrada: Como para LDESP2.

Condiciones de salida: Como para LDESP2.

Longitud: 96 bytes.

```
                10 ;       ** LDESP1 **
9D08            20         ORG  40200
9D08  F3        30         DI   ;          Prohibe interrupciones
9D09  CD99BB    40         CALL #BB99
9D0C  CD2CBC    50         CALL #BC2C
9D0F  32679D    60         LD   (PAPEL),A; Guarda el color del papel
9D12  DD7E00    70         LD   A,(IX)    ; Decide si es a izquierda o derecha
9D15  FE00      80         CP   0
9D17  2825      90         JR   Z,OTRO
9D19  2627     100         LD   H,39
9D1B  2E00     110         LD   L,0
9D1D  1627     120         LD   D,39
9D1F  1E18     130         LD   E,24
9D21  0600     140         LD   B,0
9D23  C5       150         PUSH BC         ; Preparado para desplazar,
9D24  E5       160         PUSH HL         ; guarda los registros
9D25  D5       170         PUSH DE         ; en la pila
9D26  CD0BBC   180         CALL #BC0B      ; Obtiene el offset
9D29  110200   190         LD   DE,2
9D2C  19       200         ADD  HL,DE      ; Lo actualiza
9D2D  3A679D   210         LD   A,(PAPEL); Obtiene el color del papel
9D30  CD05BC   220         CALL #BC05      ; Guarda el offset en el 6845
9D33  3A679D   230         LD   A,(PAPEL); (controlador de video)
9D36  D1       240         POP  DE
9D37  E1       250         POP  HL
9D38  C1       260         POP  BC
9D39  CD50BC   270         CALL #BC50      ; Desplaza una columna,
               280 ;                        el espacio de un caracter
9D3C  FB       290         EI   ;          Permite las interrupciones
9D3D  C9       300         RET
9D3E  2600     310 OTRO    LD   H,0        ; Prepara los registros para
9D40  2E00     320         LD   L,0        ; el desplazamiento de columnas
9D42  1600     330         LD   D,0        ; al final de la rutina
9D44  1E18     340         LD   E,24
9D46  0601     350         LD   B,1
9D48  3A679D   360         LD   A,(PAPEL)
9D4B  F5       370         PUSH AF
9D4C  C5       380         PUSH BC
9D4D  D5       390         PUSH DE
9D4E  E5       400         PUSH HL
9D4F  CD0BBC   410         CALL #BC0B      ; Obtiene el offset
9D52  AF       420         XOR  A
9D53  110200   430         LD   DE,2       ; Altera el offset y lo
9D56  ED52     440         SBC  HL,DE
9D58  3A679D   450         LD   A,(PAPEL)
9D5B  CD05BC   460         CALL #BC05      ; envia al controlador de video
9D5E  E1       470         POP  HL
9D5F  D1       480         POP  DE
9D60  C1       490         POP  BC
```

```
9D61  F1        500        POP  AF
9D62  CD50BC    510        CALL #BC50    ; Desplaza la columna de la
                520 ;                      izquierda, una linea
                530 ;                      hasta el otro extremo
9D65  FB        540        EI
9D66  C9        550        RET
9D67  00        560 PAPEL  DEFB 0
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1:REM Desplazamiento a izquierda
   y derecha en el modo 1
20 ORIGIN 0,32 :REM Origen de graficos
   de la ventana a desplazar.
30 WINDOW 1,40,2,24 : REM Ventana de
   texto que se va a desplazar.
40 MOVE 0,10:DRAW 400,100
50 DRAW 150,50 : DRAW 200,50
60 DRAW 300,60 : DRAW 400,100
70 DRAW 450,20 : DRAW 500,10
80 DRAW 600,150 : DRAW 640,10
90 CALL &BD19: REM Espera al barrido.
100 a$=INKEY$:IF a$="" THEN 100
110 IF ASC(a$)=243 THEN CALL 40200,0
120 REM Teclas de desplazamiento
130 IF ASC(a$)=242 THEN CALL 40200,1
140 GOTO 90
```

Probablemente adviertas que es posible incorporar todas estas rutinas en un único programa, en el que la rutina compruebe el modo de pantalla actual y actualice los registros apropiadamente para las rutinas SWSCROLL y para el offset de pantalla.

Acabamos de ver cómo desplazar toda la pantalla hacia arriba, hacia abajo y lateralmente. Usando la rutina del *firmware* localizada en la dirección &BC50, también podremos desplazar hacia arriba o hacia abajo un área dada de la pantalla, que llamaré ventana de desplazamiento. Las dos rutinas que vamos a examinar a continuación desplazan un pequeño área de la pantalla lateralmente, una versión horizontal de SWSCROLL. El material expulsado por un lado de la ventana de desplazamiento se pierde. Sin embargo, para realizar estos movimientos, hay que examinar brevemente la forma en que la memoria de video del Amstrad ha sido dispuesta, ya que tendremos que elaborar un método de acceso directo a la memoria de pantalla para realizar el resto de los desplazamientos de este capítulo. El resto de las rutinas de desplazamiento continuarán trabajando con el espacio de un carácter como unidad básica de movimiento. Así que vamos con la disposición de pantalla.

Cada carácter de la pantalla, en cualquier modo de pantalla, está formado verticalmente por ocho líneas de pantalla. La pantalla en su totalidad tiene una altura de 25 filas, lo que nos da un resultado de 200 líneas de pantalla para todos los modos. Todos los modos de

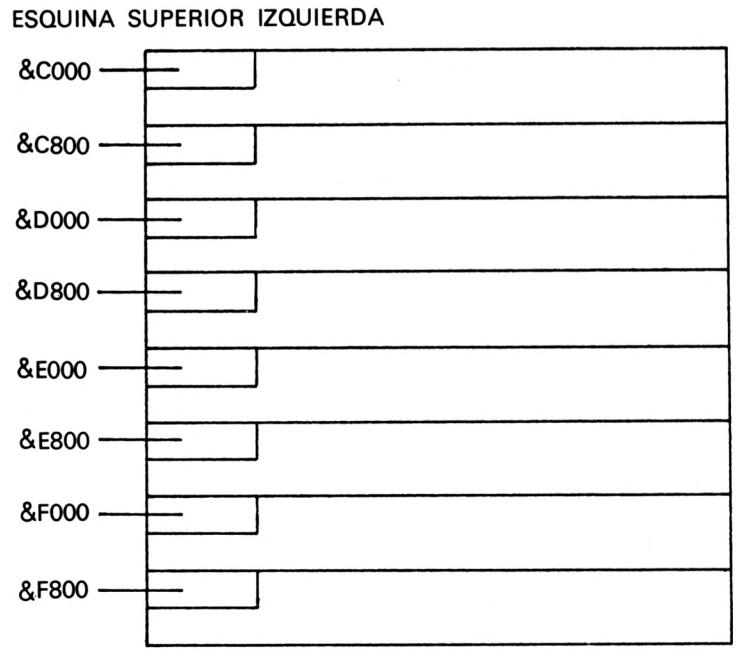
pantalla utilizan 16384 bytes de memoria, empezando en la dirección &C000 y terminando en la dirección &FFFF. Las imágenes para todos los modos de pantalla tienen un ancho de 80 bytes. Esto es, una línea de pantalla de izquierda a derecha, necesita 80 bytes de memoria de pantalla para definir su contenido. Esto explica la necesidad de 16384 bytes para la memoria de pantalla (200 líneas de pantalla con 80 bytes cada una).

La disposición exacta de la memoria de pantalla en relación con los cuadros de caracteres, etc., dependerá del modo de pantalla en uso.

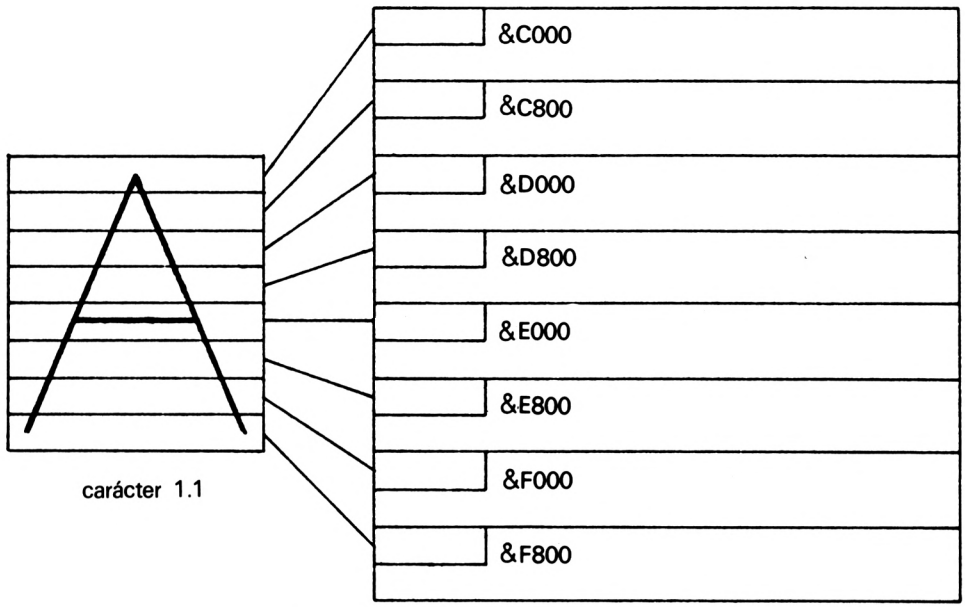
Modo 2

Es el modo de pantalla de uso más sencillo. Al igual que la memoria de pantalla en todos los demás modos, la memoria de video se dispone en ocho bloques de memoria de 2K, que tras un cambio de modo se organizan de la siguiente forma.

En modo 2, cada carácter tiene un ancho de un byte; por esta razón la pantalla tiene 80 columnas en este modo. Cada bit dentro de cada byte corresponde a un punto de pantalla. Así, la posición superior de la pantalla, 1,1, tras un cambio de modo, se define por el contenido de &C000. El carácter consta de ocho líneas de pantalla, que se toman de los otros siete bloques de memoria de 2K. En este caso, la fila 2 del carácter se define por el byte en &C800, la fila 3 por &D000, la fila 4 por &D800, y así sucesivamente.



Disposición de la RAM de pantalla

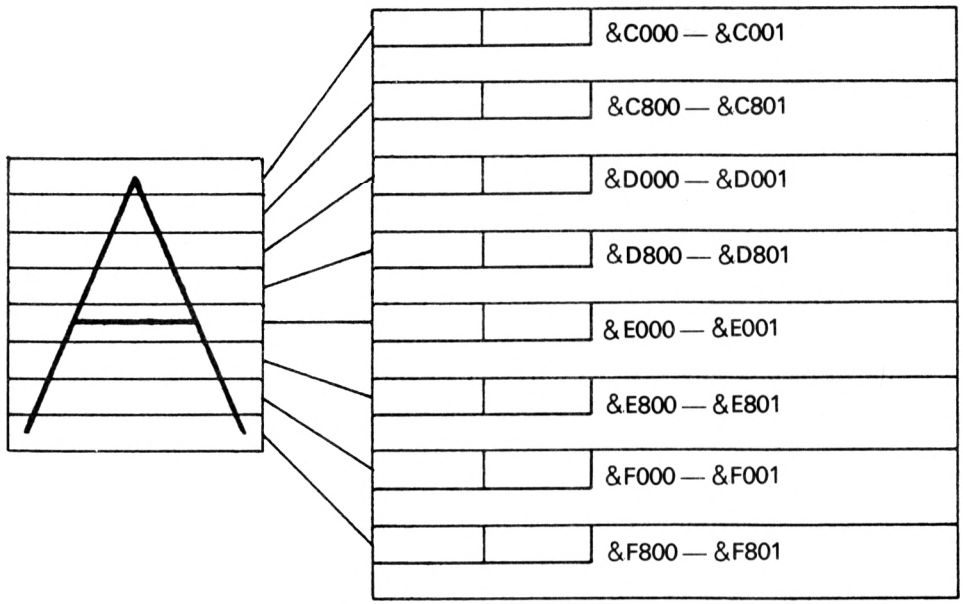


Cuadro de un carácter en modo 2

Si hay gráficos en la pantalla, las posiciones de los caracteres correspondientes tendrán alterados ciertos bytes de acuerdo con los gráficos dibujados. Una vez conocida la dirección dentro de la memoria de pantalla de la fila superior del carácter en cuestión, podemos obtener los otros 7 bytes que conforman la definición de este cuadro del carácter, añadiendo repetidamente 2048 a la primera dirección hasta obtener las direcciones de las otras siete líneas de pantalla. En modo 2 no existe información en cuanto a color; si un punto de pantalla va a ser expuesto con el color principal, el correspondiente bit en el byte apropiado se coloca a 1. Si se va a exponer con el color de fondo, el bit se pondrá a 0. La relación entre las filas de una misma posición de carácter, estando separadas entre sí por 2048 bytes, es la misma en todos los modos de pantalla.

Modo 1

Aquí se complica un poco el asunto, ya que en este modo existe la posibilidad de exponer más de un color. Cada cuadro de carácter tiene un ancho de 2 bytes, lo que explica que la pantalla en modo 1 tenga una anchura de 40 columnas. El cuadro de un carácter en modo 1 se define en la forma de la figura. Al igual que antes, después de un cambio de modo, veremos qué definen los bytes en la posición 1,1 de la pantalla.

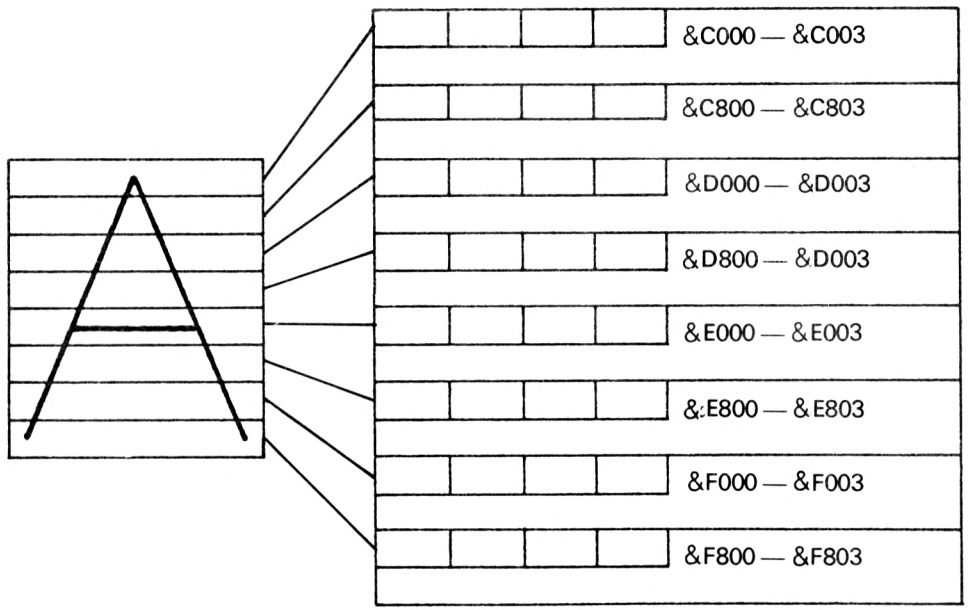


Uno de los bytes que forman el carácter define el color de los puntos de pantalla, así como el estado en que se encuentren.

El otro byte de la memoria de pantalla, por tanto, define el estado de una línea de pantalla que tiene por anchura la mitad de un carácter. Con lo cual el cuadro del carácter es definido por 16 bytes de memoria de video.

Modo 0

Dado que existen dieciséis colores disponibles en este modo de pantalla, cada carácter requiere mayor número de bytes para definirlo; en éste el ancho es de cuatro bytes. Esto nos da las veinte columnas de la pantalla de texto del modo 0. Inmediatamente después de una orden modo 0 el mapa de la memoria de video de la posición de pantalla 1,1 queda como se indica en la página siguiente.



Disposición de pantalla en modo 0

Según esto, el modo 0 requiere 32 bytes de memoria de video para definir cada carácter.

Bueno, ya sabemos que cada línea de pantalla, dentro de un mismo carácter, queda separada de la siguiente por 2048 bytes. Ahora necesitamos algún medio para hallar en la memoria de video la dirección de la línea superior del carácter que nos interese.

Afortunadamente, los simpáticos chicos de Amsoft han resuelto este problema proporcionándonos una rutina *firmware* que realiza este trabajo. Esto se consigue llamando a la dirección &BC1A, y la posición del carácter se transmite a la rutina en el par de registros HL. El registro H contiene la posición X y el registro L la posición Y. Tanto X como Y toman valores de 0 en adelante, siendo 0,0 la esquina superior izquierda de la pantalla.

La rutina devuelve una dirección en el par de registros HL. Para el modo 2, ésta será la dirección de la posición de la fila superior del carácter que nos interese. Para los modos 0 y 1 será la dirección del byte más a la izquierda de la fila superior. Así la dirección del primer byte de la segunda fila será la dirección (HL + 2048). Adentrándonos en las rutinas de bucle que enseguida veremos, están las instrucciones de movimiento de bloques LDIR y LDDR de la CPU Z-80. Para aquellos que no estén muy versados en estas instrucciones, voy a dar una rápida descripción de las mismas y de su utilización.

LDIR y LDDR

Un método para transferir bytes a través de la memoria del ordenador puede ser repetir un determinado número de veces un lazo de instrucciones como las que a continuación exponemos.

```
LD A,(HL)
LD (DE),A
INC HL
INC DE
```

Esta secuencia es muy directa, y realiza la transferencia de forma sencilla. El problema surge cuando hay que transferir muchos datos, pues esta sentencia, cuando se repite a menudo, requiere demasiado tiempo. Sin embargo, el Z-80 tiene implementadas instrucciones de transferencia de bloques. La LDIR se ejecuta con la siguiente estructura:

```
LD HL,direccion fuente
LD DE,direccion de destino
LD BC,numero de bytes
LDIR
```

La dirección en el registro HL es la dirección desde la cual los bytes serán transferidos y DE es la dirección donde serán almacenados. El registro BC contiene el número de bytes a transferir. El primer byte transferido irá a la dirección DE, el segundo a la dirección DE+1 y así sucesivamente. Tanto HL como DE son incrementados en cada transferencia individual. Una vez que la instrucción LDIR comienza a ejecutarse, no parará hasta que todos los bytes sean transferidos. Esto es mucho más rápido que realizar el mismo trabajo con instrucciones individuales del Z-80, así que cada vez que tengas un montón de datos que copiar deberías usar esta instrucción o la LDDR. LDDR hace lo mismo, con la diferencia de que los registros HL y DE son decrementados en cada transferencia individual.

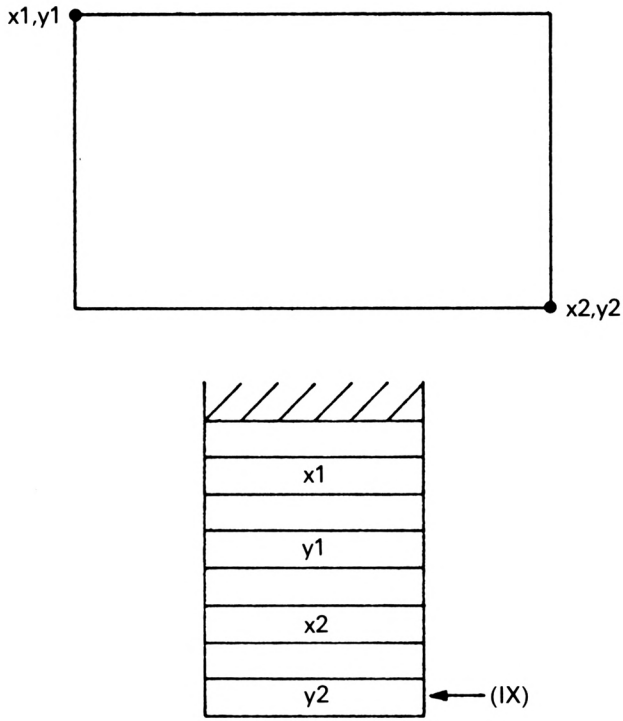
Los listados de estas rutinas de bucle están bien documentados y podrás seguir su evolución examinando los listados y las notas antes indicadas sobre la disposición de la memoria de pantalla, para los diferentes modos de pantalla.

IZDES (DESplazamiento a la IZquierda)

Esta rutina, en cualquier modo, desplazará un área dada de pantalla, el espacio de un carácter a la izquierda.

Cualquier carácter desplazado fuera del borde izquierdo de la ventana de trabajo se pierde. La rutina es, pues, análoga a la SWSCROLL perteneciente al *firmware*.

Requisitos de entrada: Desde BASIC: CALL dirección,x1,y1,x2,y2.
Las coordenadas van desde 1,1, siendo ésta la esquina superior izquierda de la pantalla.
Desde código máquina A=4 y IX apunta a un bloque de parámetros como el mostrado a continuación.



Bloque de parámetros para IZDES

Condiciones de salida: Todo alterado.

Longitud: 169 bytes.

```
                    10 ;        ** IZDES **
9D08                20          ORG  40200
9D08  FE04          30          CP   4
9D0A  C0            40          RET  NZ          ; Si no hay 4 parametros, retorno
9D0B  CD11BC        50          CALL #BC11       ; Obtiene el modo de pantalla
9D0E  32B09D        60          LD   (MODO),A
9D11  CD959D        70          CALL ANCHO       ; Numero de caracteres a mover
9D14  32AE9D        80          LD   (CAR),A
9D17  CD5F9D        90          CALL MODOC       ; Ajusta este valor para que
9D1A  32AE9D       100          LD   (CAR),A     ; encaje en el modo actual
9D1D  AF           110          XOR  A
```

```
9D1E  32AF9D      120         LD   (CAR+1),A ; Pone a cero el byte alto
9D21  DD6606      130         LD   H,(IX+6)  ; Impone las coordenadas de
9D24  DD6E04      140         LD   L,(IX+4)  ; la esquina superior izquierda
9D27  CD899D      150         CALL ALTURA    ; Numero de lineas?
9D2A  E5          160 OBUCLE  PUSH HL
9D2B  C5          170         PUSH BC
9D2C  CD389D      180         CALL MOVIM     ; Realiza el desplazamiento
9D2F  C1          190         POP  BC
9D30  E1          200         POP  HL
9D31  CDA09D      210         CALL PONESP    ; Rellena la columna de la
                  220 ;                        derecha
9D34  2C          230         INC  L         ; Desplaza la siguiente linea
9D35  10F3        240         DJNZ OBUCLE
9D37  C9          250         RET
9D38  2D          260 MOVIM   DEC  L
9D39  25          270         DEC  H         ; Convierte coordenadas fisicas
9D3A  CD1ABC      280         CALL #BC1A     ; en posiciones de pantalla
9D3D  0608        290         LD   B,8       ; 8 Bytes a desplazar
9D3F  E5          300         PUSH HL
9D40  D1          310         POP  DE
9D41  23          320         INC  HL        ; Movimiento del caracter
9D42  CD799D      330         CALL AJUSTE    ; de byte en byte
9D45  C5          340 BUCLE   PUSH BC
9D46  D5          350         PUSH DE
9D47  E5          360         PUSH HL
9D48  ED4BAE9D    370         LD   BC,(CAR) ; Numero de bytes horizontales
9D4C  EDB0        380         LDIR ;          Movimiento de bloques
9D4E  010008      390         LD   BC,2048  ; El siguiente esta 2048 mas alla
9D51  E1          400         POP  HL
9D52  09          410         ADD  HL,BC    ; Direccion de la proxima linea
9D53  D1          420         POP  DE
9D54  E5          430         PUSH HL
9D55  D5          440         PUSH DE
9D56  E1          450         POP  HL
9D57  09          460         ADD  HL,BC
9D58  E5          470         PUSH HL
9D59  D1          480         POP  DE        ; Proxima direccion de destino
9D5A  E1          490         POP  HL
9D5B  C1          500         POP  BC
9D5C  10E7        510         DJNZ BUCLE     ; Hecho para 8 bytes
9D5E  C9          520         RET
9D5F  3AB09D      530 MODOC   LD   A,(MODO) ; Ajuste del numero de bytes
9D62  FE02        540         CP   2         ; horizontales a desplazar ( Modo )
9D64  2004        550         JR   NZ,MOD1
9D66  3AAE9D      560         LD   A,(CAR)
9D69  C9          570         RET
9D6A  FE00        580 MOD1    CP   0
9D6C  2805        590         JR   Z,MODO
9D6E  3AAE9D      600         LD   A,(CAR)
9D71  87          610         ADD  A,A       ; Modo 1; ancho de 2 bytes
9D72  C9          620         RET
9D73  3AAE9D      630 MODO    LD   A,(CAR)
9D76  87          640         ADD  A,A
9D77  87          650         ADD  A,A       ; Modo 0; ancho de 4 bytes
9D78  C9          660         RET
9D79  3AB09D      670 AJUSTE  LD   A,(MODO) ; Ajuste de la direccion
9D7C  FE02        680         CP   2 ;        de cada linea, a la anchura del
9D7E  C8          690         RET  Z ;        caracter
9D7F  FE01        700         CP   1
9D81  2002        710         JR   NZ,MOVO
9D83  23          720         INC  HL
9D84  C9          730         RET
9D85  23          740 MOVO    INC  HL
9D86  23          750         INC  HL
9D87  23          760         INC  HL
9D88  C9          770         RET
9D89  E5          780 ALTURA  PUSH HL        ; Calcula las lineas que
9D8A  DD7E00      790         LD   A,(IX)    ; hay que desplazar
9D8D  DD6604      800         LD   H,(IX+4)
9D90  94          810         SUB  H
```

```
9D91  3C        820        INC  A
9D92  47        830        LD   B,A
9D93  E1        840        POP  HL
9D94  C9        850        RET
9D95  DD7E02    860 ANCHO  LD   A,(IX+2) ; Numero de caracteres
9D98  E5        870        PUSH HL        ; a desplazar horizontalmente
9D99  DD6606    880        LD   H,(IX+6)
9D9C  94        890        SUB  H
9D9D  3C        900        INC  A
9D9E  E1        910        POP  HL
9D9F  C9        920        RET
9DA0  E5        930 PONESP PUSH HL        ; Rellena la columna de
9DA1  DD6602    940        LD   H,(IX+2) ; la derecha con un espacio
9DA4  CD75BB    950        CALL #BB75
9DA7  3E20      960        LD   A,32
9DA9  CD5ABB    970        CALL #BB5A
9DAC  E1        980        POP  HL
9DAD  C9        990        RET
9DAE  0000     1000 CAR    DEFW 00
9DB0  00       1010 MODO   DEFB 0
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 CLS : FOR I=1 TO 10
30 PRINT "12345678901234567890123456789"
40 NEXT
50 CALL 40200,2,2,10,10
60 FOR H=0 TO 200 : NEXT : GOTO 50
```

Comentarios

Esta rutina no está protegida contra parámetros incoherentes, así que no intentes trabajar con coordenadas ridículamente grandes, o atribuyendo a y2 un valor inferior a y1. ¡Quedas advertido!

El programa no es relocalizable con facilidad y los bytes utilizados son para la dirección 40200. Observa cómo usamos una rutina del *firmware* para recuperar el modo de pantalla en uso. La rutina llamada en &BC11, devuelve un valor en el registro A que corresponde al modo de pantalla. La rutina entonces mueve los bytes de memoria según el modo de pantalla.

DESPDER (DESPlazamiento a la DERecha)

Esta rutina desplaza un área definida en la pantalla el espacio de un carácter hacia la derecha. Al igual que antes, cualquier cosa expulsada por el borde derecho de la ventana se pierde. La rutina funcionará en cualquier modo de pantalla.

Requisitos de entrada: Los mismos que para IZDES.

Condiciones de salida: Los mismos que para IZDES.

Longitud: 138 bytes.

```
                    10 ;       ** DESPDER **
9E98                20         ORG  40600
9E98  FE04          30         CP   4
9E9A  C0            40         RET  NZ          ; Si no hay 4 parametros, retorno
9E9B  CD11BC        50         CALL #BC11       ; Obtiene el modo de pantalla
9E9E  324E9F        60         LD   (MODO),A
9EA1  CD339F        70         CALL ANCHO       ; Numero de caracteres a mover
9EA4  324C9F        80         LD   (CAR),A
9EA7  CDFD9E        90         CALL MODOC       ; Ajusta este valor para que
9EAA  324C9F       100         LD   (CAR),A     ; encaje en el modo actual
9EAD  AF           110         XOR  A
9EAE  324D9F       120         LD   (CAR+1),A ; Pone a cero el byte alto
9EB1  DD6602       130         LD   H,(IX+2)  ; Impone las coordenadas de
9EB4  DD6E04       140         LD   L,(IX+4)  ; la esquina superior derecha
9EB7  CD279F       150         CALL ALTURA      ; Numero de lineas?
9EBA  E5           160 OBUCLE  PUSH HL
9EBB  C5           170         PUSH BC
9EBC  CDC89E       180         CALL MOVIM       ; Realiza el desplazamiento
9EBF  C1           190         POP  BC
9EC0  E1           200         POP  HL
9EC1  CD3E9F       210         CALL PONESP      ; Rellena la columna de la
                   220 ;                          derecha
9EC4  2C           230         INC  L           ; Desplaza la siguiente linea
9EC5  10F3         240         DJNZ OBUCLE
9EC7  C9           250         RET
9EC8  2D           260 MOVIM   DEC  L
9EC9  25           270         DEC  H           ; Convierte coordenadas fisicas
9ECA  CD1ABC       280         CALL #BC1A       ; en posiciones de pantalla
9ECD  3A4E9F       290         LD   A,(MODO)    ; Ajusta las operaciones
9ED0  FE02         300         CP   2           ; al modo actual de pantalla
9ED2  2807         310         JR   Z,OK
9ED4  FE01         320         CP   1
9ED6  2802         330         JR   Z,OK2
9ED8  23           340         INC  HL
9ED9  23           350         INC  HL
9EDA  23           360 OK2     INC  HL
9EDB  0608         370 OK      LD   B,8         ; 8 Bytes a desplazar
9EDD  E5           380         PUSH HL
9EDE  D1           390         POP  DE
9EDF  2B           400         DEC  HL          ; Movimiento del caracter
9EE0  CD179F       410         CALL AJUSTE      ; de byte en byte
9EE3  C5           420 BUCLE   PUSH BC
9EE4  D5           430         PUSH DE
9EE5  E5           440         PUSH HL
9EE6  ED4B4C9F     450         LD   BC,(CAR)    ; Numero de bytes horizontales
9EEA  EDB8         460         LDDR ;             Movimiento de bloques
9EEC  010008       470         LD   BC,2048     ; El siguiente esta 2048 mas alla
9EEF  E1           480         POP  HL
9EF0  09           490         ADD  HL,BC       ; Direccion de la proxima linea
                   500 ;                              a trasladar
9EF1  D1           510         POP  DE
9EF2  E5           520         PUSH HL
9EF3  D5           530         PUSH DE
9EF4  E1           540         POP  HL
9EF5  09           550         ADD  HL,BC
9EF6  E5           560         PUSH HL
9EF7  D1           570         POP  DE          ; Proxima direccion de destino
9EF8  E1           580         POP  HL
9EF9  C1           590         POP  BC
9EFA  10E7         600         DJNZ BUCLE       ; Hecho para 8 bytes
9EFC  C9           610         RET
9EFD  3A4E9F       620 MODOC   LD   A,(MODO)    ; Ajuste del numero de bytes
```

```
9F00  FE02      630         CP   2          ; horizontales a desplazar ( Modo )
9F02  2004      640         JR   NZ,MOD1
9F04  3A4C9F    650         LD   A,(CAR)
9F07  C9        660         RET
9F08  FE00      670 MOD1    CP   0
9F0A  2805      680         JR   Z,MODO
9F0C  3A4C9F    690         LD   A,(CAR)
9F0F  87        700         ADD  A,A        ; Modo 1; ancho de 2 bytes
9F10  C9        710         RET
9F11  3A4C9F    720 MODO    LD   A,(CAR)
9F14  87        730         ADD  A,A
9F15  87        740         ADD  A,A        ; Modo 0; ancho de 4 bytes
9F16  C9        750         RET
9F17  3A4E9F    760 AJUSTE  LD   A,(MODO)   ; Ajuste de la direccion
9F1A  FE02      770         CP   2          ; de cada linea, a la anchura del
9F1C  C8        780         RET  Z          ; caracter
9F1D  FE01      790         CP   1
9F1F  2002      800         JR   NZ,MOVO
9F21  2B        810         DEC  HL
9F22  C9        820         RET
9F23  2B        830 MOVO    DEC  HL
9F24  2B        840         DEC  HL
9F25  2B        850         DEC  HL
9F26  C9        860         RET
9F27  E5        870 ALTURA  PUSH HL         ; Calcula las lineas que
9F28  DD7E00    880         LD   A,(IX)     ; hay que desplazar
9F2B  DD6604    890         LD   H,(IX+4)
9F2E  94        900         SUB  H
9F2F  3C        910         INC  A
9F30  47        920         LD   B,A
9F31  E1        930         POP  HL
9F32  C9        940         RET
9F33  DD7E02    950 ANCHO   LD   A,(IX+2)   ; Numero de caracteres
9F36  E5        960         PUSH HL         ; a desplazar horizontalmente
9F37  DD6606    970         LD   H,(IX+6)
9F3A  94        980         SUB  H
9F3B  3C        990         INC  A
9F3C  E1       1000         POP  HL
9F3D  C9       1010         RET
9F3E  E5       1020 PONESP  PUSH HL         ; Rellena la columna de
9F3F  DD6606   1030         LD   H,(IX+6)   ; la derecha con un espacio
9F42  CD75BB   1040         CALL #BB75
9F45  3E20     1050         LD   A,32
9F47  CD5ABB   1060         CALL #BB5A
9F4A  E1       1070         POP  HL
9F4B  C9       1080         RET
9F4C  0000     1090 CAR     DEFW 00
9F4E  00       1100 MODO    DEFB 0
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 CLS : FOR I=1 TO 10
30 PRINT "12345678901234567890123456789"
40 NEXT
50 CALL 40600,2,2,10,10
60 FOR H=0 TO 200 : NEXT : GOTO 50
```

Comentarios

Como ocurría con la IZDES, no se detecta ningún error en los parámetros introducidos en la rutina. La rutina no es fácilmente relocalizable y los bytes del listado son para la dirección 40600.

Tanto para IZDES como para DESPDER, la parte más lenta es la rutina PONESP que rellena la columna que ha sido desplazada fuera de la ventana. Esta área del programa utiliza una rutina del *firmware*. Podría sustituirse por una sección de código que escribiese directamente ceros en las direcciones apropiadas de la memoria de pantalla.

El siguiente programa de BASIC probará las rutinas IZDES y DESPDER, según las direcciones utilizadas.

```
10 MODE 1
20 REM direccion=40200 para IZDES
30 REM direccion=40600 para DESPDER
40 direccion=40200
50 CLS : FOR I=1 TO 10
60 PRINT "12345678901234567890123456789"
70 NEXT
80 CALL direccion,2,2,10,10
90 FOR H=0 TO 200 : NEXT :REM retardo
100 GOTO 80
```

El problema principal en las rutinas dadas hasta aquí, es que el material expulsado por un borde de la ventana se pierde definitivamente. Las tres últimas rutinas de desplazamiento que vamos a ver tratan esta cuestión, dando una versión de SWSCROLL que recupera el material que ha sido expulsado de la ventana, y versiones de la IZDES y de la DESPDER que realizan trabajos similares. Estas rutinas funcionarán en cualquier modo de pantalla y desplazarán, al igual que las rutinas anteriores, tanto textos como gráficos.

DESPIZR (DESPlazamiento a la IZquierda Recuperando)

Esta rutina desplaza un área definida de la pantalla el espacio de un carácter a la izquierda. Todo lo que excede del borde izquierdo de la pantalla reaparece por el borde derecho. Esto puede resultar muy aparente para desplazar la página de presentación de un programa, encabezamientos, etc., especialmente bajo el control de las órdenes AFTER o EVERY del BASIC.

Requisitos de entrada: Igual que para IZDES.

Condiciones de salida: Todos los registros son alterados.

Longitud: 385 bytes.

```
                    10 ;       ** DESPIZR **
A21C                20         ORG  41500
A21C  FE04          30         CP   4
A21E  C0            40         RET  NZ          ; Si no hay 4 parametros, retorno
A21F  CD11BC        50         CALL #BC11       ; Obtiene el modo de pantalla
A222  3274A3        60         LD   (MODO),A
A225  CDACA2        70         CALL ANCHO       ; Numero de caracteres a mover
A228  3271A3        80         LD   (CAR),A
A22B  CD76A2        90         CALL MODOC       ; Ajusta este valor para que
A22E  3271A3       100         LD   (CAR),A     ; encaje en el modo actual
A231  AF           110         XOR  A
A232  3272A3       120         LD   (CAR+1),A ; Pone a cero el byte alto
A235  DD6606       130         LD   H,(IX+6)    ; Impone las coordenadas de
A238  DD6E04       140         LD   L,(IX+4)    ; la esquina superior derecha
A23B  CDA0A2       150         CALL ALTURA      ; Numero de lineas?
A23E  E5           160 OBUCLE  PUSH HL
A23F  C5           170         PUSH BC
A240  CD06A3       180         CALL OBCAR       ; Obtiene el caracter
A243  CD4FA2       190         CALL MOVIM       ; Realiza el desplazamiento
A246  C1           200         POP  BC
A247  E1           210         POP  HL
A248  CDB7A2       220         CALL PONESP      ; Rellena la columna de la
                    230 ;                          derecha con el caracter
A24B  2C           240         INC  L           ; Desplaza la siguiente linea
A24C  10F0         250         DJNZ OBUCLE
A24E  C9           260         RET
A24F  2D           270 MOVIM   DEC  L
A250  25           280         DEC  H           ; Convierte coordenadas fisicas
A251  CD1ABC       290         CALL #BC1A       ; en posiciones de pantalla
A254  0608         300         LD   B,8         ; 8 Bytes a desplazar
A256  E5           310         PUSH HL
A257  D1           320         POP  DE
A258  23           330         INC  HL          ; Movimiento del caracter
A259  CD90A2       340         CALL AJUSTE      ; de byte en byte
A25C  C5           350 BUCLE   PUSH BC
A25D  D5           360         PUSH DE
A25E  E5           370         PUSH HL
A25F  ED4B71A3     380         LD   BC,(CAR)    ; Numero de bytes horizontales
A263  EDB0         390         LDIR ;             Movimiento de bloques
A265  010008       400         LD   BC,2048     ; El siguiente esta 2048 mas alla
A268  E1           410         POP  HL
A269  09           420         ADD  HL,BC       ; Direccion de la proxima linea
                    430 ;                          a trasladar
A26A  D1           440         POP  DE
A26B  E5           450         PUSH HL
A26C  D5           460         PUSH DE
A26D  E1           470         POP  HL
A26E  09           480         ADD  HL,BC
A26F  E5           490         PUSH HL
A270  D1           500         POP  DE          ; Proxima direccion de destino
A271  E1           510         POP  HL
A272  C1           520         POP  BC
A273  10E7         530         DJNZ BUCLE       ; Hecho para 8 bytes
A275  C9           540         RET
A276  3A74A3       550 MODOC   LD   A,(MODO)     ; Ajuste del numero de bytes
A279  FE02         560         CP   2           ; horizontales a desplazar ( Modo )
A27B  2004         570         JR   NZ,MOD1
A27D  3A71A3       580         LD   A,(CAR)
A280  C9           590         RET
A281  FE00         600 MOD1    CP   0
A283  2805         610         JR   Z,MODO
A285  3A71A3       620         LD   A,(CAR)
A288  87           630         ADD  A,A         ; Modo 1; ancho de 2 bytes
A289  C9           640         RET
A28A  3A71A3       650 MODO    LD   A,(CAR)
A28D  87           660         ADD  A,A
A28E  87           670         ADD  A,A         ; Modo 0; ancho de 4 bytes
A28F  C9           680         RET
A290  3A74A3       690 AJUSTE  LD   A,(MODO)     ; Ajuste de la direccion
A293  FE02         700         CP   2           ; de cada linea, a la anchura del
```

```
A295  C8        710         RET  Z          ; caracter
A296  FE01      720         CP   1
A298  2002      730         JR   NZ,MOVO
A29A  23        740         INC  HL
A29B  C9        750         RET
A29C  23        760 MOVO    INC  HL
A29D  23        770         INC  HL
A29E  23        780         INC  HL
A29F  C9        790         RET
A2A0  E5        800 ALTURA  PUSH HL         ; Calcula las lineas que
A2A1  DD7E00    810         LD   A,(IX)     ; hay que desplazar
A2A4  DD6604    820         LD   H,(IX+4)
A2A7  94        830         SUB  H
A2A8  3C        840         INC  A
A2A9  47        850         LD   B,A
A2AA  E1        860         POP  HL
A2AB  C9        870         RET
A2AC  DD7E02    880 ANCHO   LD   A,(IX+2)   ; Numero de caracteres
A2AF  E5        890         PUSH HL         ; a desplazar horizontalmente
A2B0  DD6606    900         LD   H,(IX+6)
A2B3  94        910         SUB  H
A2B4  3C        920         INC  A
A2B5  E1        930         POP  HL
A2B6  C9        940         RET
A2B7  E5        950 PONESP  PUSH HL
A2B8  D5        960         PUSH DE
A2B9  C5        970         PUSH BC
A2BA  DDE5      980         PUSH IX
A2BC  DD6602    990         LD   H,(IX+2)   ; Extrae la columna a rellenar
A2BF  25       1000         DEC  H
A2C0  2D       1010         DEC  L          ; Fila y columna fisicas
A2C1  3A74A3   1020         LD   A,(MODO)   ; Selecciona la rutina para
A2C4  FE01     1030         CP   1          ; el modo actual de pantalla
A2C6  2816     1040         JR   Z,PMOD1
A2C8  FE00     1050         CP   0
A2CA  2822     1060         JR   Z,PMODO
A2CC  CD3AA3   1070         CALL INIC       ; Rutina para el modo 2
A2CF  DD7E00   1080 CHPL    LD   A,(IX)     ; Extrae de la memoria
A2D2  77       1090         LD   (HL),A     ; Lo pone en la memoria de pantalla
A2D3  DD23     1100         INC  IX         ; Siguiente posicion de la memoria
A2D5  19       1110         ADD  HL,DE      ; Byte de la RAM de video
A2D6  10F7     1120         DJNZ CHPL       ; Repite esto para 8 bytes
A2D8  DDE1     1130 POK    POP  IX
A2DA  C1       1140         POP  BC
A2DB  D1       1150         POP  DE
A2DC  E1       1160         POP  HL
A2DD  C9       1170         RET
A2DE  CD3AA3   1180 PMOD1   CALL INIC       ; Rutina para el modo 1
A2E1  CD69A3   1190 P1L     CALL PONHL      ; 2 bytes cada caracter
A2E4  CD69A3   1200         CALL PONHL      ; Almacena 2 bytes en la memoria
A2E7  2B       1210         DEC  HL         ; de pantalla
A2E8  2B       1220         DEC  HL
A2E9  19       1230         ADD  HL,DE      ; Direccion del siguiente byte
A2EA  10F5     1240         DJNZ P1L        ;   de memoria
A2EC  18EA     1250         JR   POK
A2EE  CD3AA3   1260 PMODO   CALL INIC       ; Modo 0; 4 bytes
A2F1  CD69A3   1270 POL     CALL PONHL      ; Extrae los 4 bytes
A2F4  CD69A3   1280         CALL PONHL      ; que forman el caracter
A2F7  CD69A3   1290         CALL PONHL
A2FA  CD69A3   1300         CALL PONHL
A2FD  2B       1310         DEC  HL
A2FE  2B       1320         DEC  HL
A2FF  2B       1330         DEC  HL
A300  2B       1340         DEC  HL
A301  19       1350         ADD  HL,DE      ; Siguiente caracter de la fila
A302  10ED     1360         DJNZ POL
A304  18D2     1370         JR   POK
A306  E5       1380 OBCAR   PUSH HL
A307  C5       1390         PUSH BC
A308  D5       1400         PUSH DE
```

```
A309  DDE5      1410         PUSH IX
A30B  25        1420         DEC  H
A30C  2D        1430         DEC  L
A30D  3A74A3    1440         LD   A,(MODO)   ; Chequea el modo actual,
A310  FE01      1450         CP   1          ; saltando a la rutina preparada
A312  2816      1460         JR   Z,GMOD1    ; para ese modo
A314  FE00      1470         CP   0
A316  2831      1480         JR   Z,GMODO
A318  CD3AA3    1490         CALL INIC       ; Modo 2; 1 byte
A31B  7E        1500 CHGL    LD   A,(HL)
A31C  DD7700    1510         LD   (IX),A
A31F  DD23      1520         INC  IX
A321  19        1530         ADD  HL,DE
A322  10F7      1540         DJNZ CHGL
A324  DDE1      1550 CAROK   POP  IX         ; Restaura los registros
A326  D1        1560         POP  DE
A327  C1        1570         POP  BC
A328  E1        1580         POP  HL
A329  C9        1590         RET
A32A  CD3AA3    1600 GMOD1   CALL INIC       ; Rutina para el modo 1
A32D  CD61A3    1610 M1L     CALL SHL        ; Guarda 2 bytes de la memoria
A330  CD61A3    1620         CALL SHL        ; de pantalla por cada caracter
A333  2B        1630         DEC  HL         ; de la linea
A334  2B        1640         DEC  HL
A335  19        1650         ADD  HL,DE      ; Siguiente direccion de
                1660 ;                         la RAM de video
A336  10F5      1670         DJNZ M1L
A338  18EA      1680         JR   CAROK
A33A  CD1ABC    1690 INIC    CALL #BC1A      ; Rutina que obtiene la
A33D  DD2175A3  1700         LD   IX,TEMP    ; direccion del primer caracter,
A341  0608      1710         LD   B,8        ; introduciendolo en HL
A343  11000B    1720         LD   DE,2048
A346  C9        1730         RET
A347  10E4      1740         DJNZ M1L
A349  CD3AA3    1750 GMODO   CALL INIC       ; Rutina para el modo 0
A34C  CD61A3    1760 MOL     CALL SHL        ; 4 bytes de anchura para
A34F  CD61A3    1770         CALL SHL        ; cada caracter
A352  CD61A3    1780         CALL SHL
A355  CD61A3    1790         CALL SHL
A358  2B        1800         DEC  HL
A359  2B        1810         DEC  HL
A35A  2B        1820         DEC  HL
A35B  2B        1830         DEC  HL
A35C  19        1840         ADD  HL,DE
A35D  10ED      1850         DJNZ MOL
A35F  18C3      1860         JR   CAROK
A361  7E        1870 SHL     LD   A,(HL)     ; Transfiere un byte desde
A362  DD7700    1880         LD   (IX),A     ; la RAM de pantalla, a un almacen
A365  23        1890         INC  HL         ; de la memoria
A366  DD23      1900         INC  IX
A368  C9        1910         RET
A369  DD7E00    1920 PONHL   LD   A,(IX)     ; Transfiere un byte desde
A36C  77        1930         LD   (HL),A     ; el almacen de memoria, a la
A36D  23        1940         INC  HL         ; memoria RAM de pantalla
A36E  DD23      1950         INC  IX
A370  C9        1960         RET
A371  0000      1970 CAR     DEFW 00
A373  00        1980 CAR2    DEFB 0
A374  00        1990 MODO    DEFB 0
A375            2000 TEMP    DEFS 40
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 CLS : FOR I=1 TO 10
30 PRINT "12345678901234567890123456789"
```

```
    40 NEXT
    50 PEN 2:LOCATE 2,4 : PRINT "..HOLA..."
    60 CALL 41500,2,2,10,10
    70 FOR H=0 TO 200 : NEXT : GOTO 60
```

Comentarios

La rutina no es fácilmente relocalizable, y los bytes del listado están preparados para la dirección 41500. Como para el resto de las rutinas de desplazamiento, la velocidad de movimiento depende del tamaño de la ventana que se va a desplazar. Aun así, la rutina es bastante rápida, incluso para ventanas grandes.

DSPDERR (DeSPlazamiento a la DERecha Recuperando)

Esta rutina desplaza un área definida de la pantalla el espacio de un carácter hacia la derecha. Todo el material de la pantalla que exceda del borde derecho de la ventana reaparece por el borde izquierdo.

Requisitos de entrada: Como para DESPDER.

Condiciones de salida: Todos los registros son alterados.

Longitud: 399 bytes.

```
                    10 ;       ** DSPDERR **
9D08                20         ORG  40200
9D08  FE04          30         CP   4
9D0A  C0            40         RET  NZ          ; Si no hay 4 parametros, retorno
9D0B  CD11BC        50         CALL #BC11       ; Obtiene el modo de pantalla
9D0E  326E9E        60         LD   (MODO),A
9D11  CDA69D        70         CALL ANCHO       ; Numero de caracteres a mover
9D14  326B9E        80         LD   (CAR),A
9D17  CD709D        90         CALL MODOC       ; Ajusta este valor para que
9D1A  326B9E       100         LD   (CAR),A     ; encaje en el modo actual
9D1D  AF           110         XOR  A
9D1E  326C9E       120         LD   (CAR+1),A ; Pone a cero el byte alto
9D21  DD6602       130         LD   H,(IX+2)  ; Extrae las coordenadas de
9D24  DD6E04       140         LD   L,(IX+4)  ; la esquina superior derecha
9D27  CD9A9D       150         CALL ALTURA      ; Numero de lineas?
9D2A  E5           160 OBUCLE  PUSH HL
9D2B  C5           170         PUSH BC
9D2C  CD009E       180         CALL OBCAR       ; Obtiene el caracter
9D2F  CD3B9D       190         CALL MOVIM       ; Realiza el desplazamiento
9D32  C1           200         POP  BC
9D33  E1           210         POP  HL
9D34  CDB19D       220         CALL PONESP      ; Rellena la columna de la
                   230 ;                          derecha con el caracter
9D37  2C           240         INC  L           ; Desplaza la siguiente linea
9D38  10F0         250         DJNZ OBUCLE
9D3A  C9           260         RET
9D3B  2D           270 MOVIM   DEC  L
9D3C  25           280         DEC  H           ; Convierte coordenadas fisicas
9D3D  CD1ABC       290         CALL #BC1A       ; en posiciones de pantalla
9D40  3A6E9E       300         LD   A,(MODO)    ; Ajuste en concordancia con el
```

```
9D43  FE02        310          CP   2          ; modo actual de pantalla
9D45  2807        320          JR   Z,OK
9D47  FE01        330          CP   1
9D49  2802        340          JR   Z,OK2
9D4B  23          350          INC  HL
9D4C  23          360          INC  HL
9D4D  23          370 OK2      INC  HL
9D4E  0608        380 OK       LD   B,8        ; 8 bytes a desplazar
9D50  E5          390          PUSH HL
9D51  D1          400          POP  DE         ; Direccion a desplazar, en DE
9D52  2B          410          DEC  HL         ; Obtiene la direccion desde la
9D53  CD8A9D      420          CALL AJUSTE     ; que se desplaza el byte del
                  430 ;                          caracter
9D56  C5          440 BUCLE    PUSH BC
9D57  D5          450          PUSH DE
9D58  E5          460          PUSH HL
9D59  ED4B6B9E    470          LD   BC,(CAR)   ; Numero de bytes horizontales
9D5D  EDB8        480          LDDR ;            Movimiento de bloques
9D5F  010008      490          LD   BC,2048    ; El siguiente esta 2048 mas alla
9D62  E1          500          POP  HL
9D63  09          510          ADD  HL,BC      ; Direccion de la proxima linea
                  520 ;                          a trasladar
9D64  D1          530          POP  DE
9D65  E5          540          PUSH HL
9D66  D5          550          PUSH DE
9D67  E1          560          POP  HL
31CB  09          570          ADD  HL,BC
31CC  E5          580          PUSH HL
31CD  D1          590          POP  DE         ; Proxima direccion de destino
31CE  E1          600          POP  HL
31CF  C1          610          POP  BC
31D0  10FE        620          DJNZ BUCLE      ; Hecho para 8 bytes
31D2  C9          630          RET
31D3  3AD132      640 MODOC    LD   A,(MODO)   ; Ajuste del numero de bytes
31D6  FE02        650          CP   2          ; horizontales a desplazar ( Modo )
31D8  2004        660          JR   NZ,MOD1
31DA  3ACE32      670          LD   A,(CAR)
31DD  C9          680          RET
31DE  FE00        690 MOD1     CP   0
31E0  2805        700          JR   Z,MODO
31E2  3ACE32      710          LD   A,(CAR)
31E5  87          720          ADD  A,A        ; Modo 1; ancho de 2 bytes
31E6  C9          730          RET
31E7  3ACE32      740 MODO     LD   A,(CAR)
31EA  87          750          ADD  A,A
31EB  87          760          ADD  A,A        ; Modo 0; ancho de 4 bytes
31EC  C9          770          RET
31ED  3AD132      780 AJUSTE   LD   A,(MODO)   ; Ajuste de la direccion
31F0  FE02        790          CP   2          ; de cada linea, a la anchura del
31F2  C8          800          RET  Z          ; caracter
31F3  FE01        810          CP   1
31F5  2002        820          JR   NZ,MOVO
31F7  2B          830          DEC  HL
31F8  C9          840          RET
31F9  2B          850 MOVO     DEC  HL
31FA  2B          860          DEC  HL
31FB  2B          870          DEC  HL
31FC  C9          880          RET
31FD  E5          890 ALTURA   PUSH HL         ; Calcula las lineas que
31FE  DD7E00      900          LD   A,(IX)     ; hay que desplazar
3201  DD6604      910          LD   H,(IX+4)
3204  94          920          SUB  H
3205  3C          930          INC  A
3206  47          940          LD   B,A
3207  E1          950          POP  HL
3208  C9          960          RET
3209  DD7E02      970 ANCHO    LD   A,(IX+2)   ; Numero de caracteres
320C  E5          980          PUSH HL         ; a desplazar horizontalmente
320D  DD6606      990          LD   H,(IX+6)
3210  94         1000          SUB  H
```

```
3211  3C        1010         INC   A
3212  E1        1020         POP   HL
3213  C9        1030         RET
3214  E5        1040 PONESP  PUSH  HL
3215  D5        1050         PUSH  DE
3216  C5        1060         PUSH  BC
3217  DDE5      1070         PUSH  IX
3219  DD6606    1080         LD    H,(IX+6)   ; Extrae la columna a rellenar
321C  25        1090         DEC   H
321D  2D        1100         DEC   L          ; Fila y columna fisicas
321E  3AD132    1110         LD    A,(MODO)   ; Selecciona la rutina para
3221  FE01      1120         CP    1          ; el modo actual de pantalla
2CD7  2816      1130         JR    Z,PMOD1
2CD9  FE00      1140         CP    0
2CDB  2822      1150         JR    Z,PMODO
2CDD  CD4B2D    1160         CALL  INIC       ; Rutina para el modo 2
2CE0  DD7E00    1170 CHPL    LD    A,(IX)     ; Extrae de la memoria
2CE3  77        1180         LD    (HL),A     ; Lo pone en la memoria de pantalla
2CE4  DD23      1190         INC   IX         ; Siguiente posicion de la memoria
2CE6  19        1200         ADD   HL,DE      ; Byte de la RAM de video
2CE7  10F7      1210         DJNZ  CHPL       ; Repite esto para 8 bytes
2CE9  DDE1      1220 POK     POP   IX
2CEB  C1        1230         POP   BC
2CEC  D1        1240         POP   DE
2CED  E1        1250         POP   HL
2CEE  C9        1260         RET
2CEF  CD4B2D    1270 PMOD1   CALL  INIC       ; Rutina para el modo 1
2CF2  CD7A2D    1280 P1L     CALL  PONHL      ; 2 bytes cada caracter
2CF5  CD7A2D    1290         CALL  PONHL      ; Almacena 2 bytes en la memoria
2CF8  2B        1300         DEC   HL         ; de pantalla
2CF9  2B        1310         DEC   HL
2CFA  19        1320         ADD   HL,DE      ; Direccion del siguiente byte
2CFB  10F5      1330         DJNZ  P1L        ; de memoria
2CFD  18EA      1340         JR    POK
2CFF  CD4B2D    1350 PMODO   CALL  INIC       ; Modo 0; 4 bytes
2D02  CD7A2D    1360 POL     CALL  PONHL      ; Extrae los 4 bytes
2D05  CD7A2D    1370         CALL  PONHL      ; que forman el caracter
2D08  CD7A2D    1380         CALL  PONHL
2D0B  CD7A2D    1390         CALL  PONHL
2D0E  2B        1400         DEC   HL
2D0F  2B        1410         DEC   HL
2D10  2B        1420         DEC   HL
2D11  2B        1430         DEC   HL
2D12  19        1440         ADD   HL,DE      ; Siguiente caracter de la fila
2D13  10ED      1450         DJNZ  POL
2D15  18D2      1460         JR    POK
2D17  E5        1470 OBCAR   PUSH  HL
2D18  C5        1480         PUSH  BC
2D19  D5        1490         PUSH  DE
2D1A  DDE5      1500         PUSH  IX
2D1C  25        1510         DEC   H
2D1D  2D        1520         DEC   L
2D1E  3A852D    1530         LD    A,(MODO)   ; Chequea el modo actual,
2D21  FE01      1540         CP    1          ; saltando a la rutina preparada
2D23  2816      1550         JR    Z,GMOD1    ; para ese modo
2D25  FE00      1560         CP    0
2D27  2831      1570         JR    Z,GMODO
2D29  CD4B2D    1580         CALL  INIC       ; Modo 2; 1 byte
2D2C  7E        1590 CHGL    LD    A,(HL)
2D2D  DD7700    1600         LD    (IX),A
2D30  DD23      1610         INC   IX
2D32  19        1620         ADD   HL,DE
2D33  10F7      1630         DJNZ  CHGL
2D35  DDE1      1640 CAROK   POP   IX         ; Restaura los registros
2D37  D1        1650         POP   DE
2D38  C1        1660         POP   BC
2D39  E1        1670         POP   HL
2D3A  C9        1680         RET
2D3B  CD4B2D    1690 GMOD1   CALL  INIC       ; Rutina para el modo 1
2D3E  CD722D    1700 M1L     CALL  SHL        ; Guarda 2 bytes de la memoria
```

```
2D41  CD722D    1710         CALL SHL       ; de pantalla por cada caracter
2D44  2B        1720         DEC  HL        ; de la linea
2D45  2B        1730         DEC  HL
2D46  19        1740         ADD  HL,DE     ; Siguiente direccion de
                1750 ;                        la RAM de video
2D47  10F5      1760         DJNZ MIL
2D49  18EA      1770         JR   CAROK
2D4B  CD1ABC    1780 INIC    CALL #BC1A     ; Rutina que obtiene la
2D4E  DD21862D  1790         LD   IX,TEMP   ; direccion del primer caracter,
2D52  0608      1800         LD   B,8       ; introduciendolo en HL
2D54  110008    1810         LD   DE,2048
2D57  C9        1820         RET
2D58  10E4      1830         DJNZ MIL
2D5A  CD4B2D    1840 GMODO   CALL INIC      ; Rutina para el modo 0
2D5D  CD722D    1850 MOL     CALL SHL       ; 4 bytes de anchura para
2D60  CD722D    1860         CALL SHL       ; cada caracter
2D63  CD722D    1870         CALL SHL
2D66  CD722D    1880         CALL SHL
2D69  2B        1890         DEC  HL
2D6A  2B        1900         DEC  HL
2D6B  2B        1910         DEC  HL
2D6C  2B        1920         DEC  HL
2D6D  19        1930         ADD  HL,DE
2D6E  10ED      1940         DJNZ MOL
2D70  18C3      1950         JR   CAROK
2D72  7E        1960 SHL     LD   A,(HL)    ; Transfiere un byte desde
2D73  DD7700    1970         LD   (IX),A    ; la RAM de pantalla, a un almacen
2D76  23        1980         INC  HL        ; de la memoria
2D77  DD23      1990         INC  IX
2D79  C9        2000         RET
2D7A  DD7E00    2010 PONHL   LD   A,(IX)    ; Transfiere un byte desde
2D7D  77        2020         LD   (HL),A    ; el almacen de memoria, a la
2D7E  23        2030         INC  HL        ; memoria RAM de pantalla
2D7F  DD23      2040         INC  IX
2D81  C9        2050         RET
2D82  0000      2060 CAR     DEFW 00
2D84  00        2070 CAR2    DEFB 0
2D85  00        2080 MODO    DEFB 0
2D86            2090 TEMP    DEFS 40
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 CLS : FOR I=1 TO 10
30 PRINT "12345678901234567890123456789"
40 NEXT
50 PEN 3:LOCATE 2,4 : PRINT "..HOLA..."
60 CALL 40200,2,2,10,10
70 FOR H=0 TO 200 : NEXT : GOTO 60
```

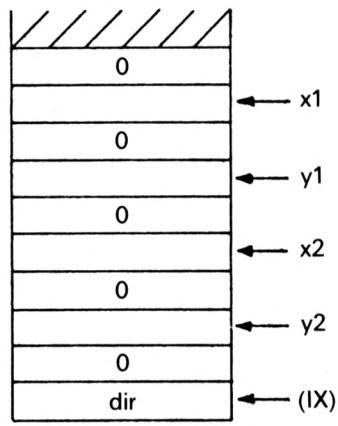
Comentarios

Esta rutina tampoco es relocalizable. Los bytes del listado están preparados para la dirección 42000.

Tanto DESPIZR como DSPDERR pueden probarse con el programa en BASIC que se listó al hablar de IZDES dentro de este mismo capítulo.

DESPVR (DESPlazamiento Vertical Recuperando)

Esta rutina desplaza un área de la pantalla hacia arriba o hacia abajo. El material desbordado por el borde superior o inferior de la ventana reaparece por el borde opuesto. La rutina funciona en cualquier modo de pantalla.



Bloque de parámetros de DESPVR

Requisitos de entrada: Desde BASIC, CALL dirección,x1,y1,x2,y2,dir, donde:
x1 = borde izquierdo del área
y1 = borde superior del área
x2 = borde derecho del área
y2 = borde inferior del área
dir = 0 desplaza hacia abajo
dir = 1 desplaza hacia arriba
Las coordenadas van desde 1,1, siendo ésta el cuadro del carácter situado en la esquina superior izquierda de la pantalla. Si la rutina es llamada desde código máquina IX apunta a un bloque de parámetros adecuado y A debe contener el valor 5.

Condiciones de salida: Todos los registros son alterados.

Longitud: 914 bytes, incluyendo la memoria temporal o *buffer.*

```
                10 ;      ** DESPVR **
A028            20        ORG  41000
A028  FE05      30        CP   5
A02A  C0        40        RET  NZ          ; Retorna si no hay 5 parametros
A02B  CD99BB    50        CALL #BB99
```

```
A02E  CD2CBC      60         CALL #BC2C
A031  32B9A3      70         LD   (PAPEL),A  ; Guarda el papel para textos
A034  DD7E00      80         LD   A,(IX)
A037  FE00        90         CP   0          ; Decide si es hacia arriba o
A039  2826       100         JR   Z,ABAJO    ; hacia abajo
A03B  CD87A0     110 ARRIBA CALL ANCH        ; Anchura de la ventana
A03E  DD6608     120         LD   H,(IX+8)
A041  DD6E06     130         LD   L,(IX+6)
A044  3E01       140         LD   A,1
A046  32FBA0     150         LD   (DIR),A
A049  CD97A0     160         CALL LINEA      ; Almacena la linea superior
A04C  CDDFA0     170         CALL DESPLZ     ; Desplaza 1 hacia arriba
A04F  CD87A0     180         CALL ANCH
A052  DD6608     190         LD   H,(IX+8)
A055  DD6E02     200         LD   L,(IX+2)
A058  3E00       210         LD   A,0
A05A  32FBA0     220         LD   (DIR),A
A05D  CD97A0     230         CALL LINEA      ; Escribe la linea en la parte
A060  C9         240         RET  ;            inferior
A061  CD87A0     250 ABAJO  CALL ANCH
A064  DD6608     260         LD   H,(IX+8)
A067  DD6E02     270         LD   L,(IX+2)
A06A  3E01       280         LD   A,1
A06C  32FBA0     290         LD   (DIR),A
A06F  CD97A0     300         CALL LINEA      ; Almacena la linea inferior
A072  CDDFA0     310         CALL DESPLZ     ; Desplaza 1 hacia abajo
A075  CD87A0     320         CALL ANCH
A078  DD6608     330         LD   H,(IX+8)
A07B  DD6E06     340         LD   L,(IX+6)
A07E  3E00       350         LD   A,0        ; Se prepara para escribir
A080  32FBA0     360         LD   (DIR),A
A083  CD97A0     370         CALL LINEA      ; Escribe la linea almacenada
A086  C9         380         RET  ;            en la linea superior
                 390 ;                         de la ventana
2AEE  E5         400 ANCH   PUSH HL
2AEF  DD7E04     410         LD   A,(IX+4)
2AF2  DD6608     420         LD   H,(IX+8)
2AF5  94         430         SUB  H
2AF6  47         440         LD   B,A
2AF7  3C         450         INC  A
2AF8  32632B     460         LD   (ANCHO),A
2AFB  04         470         INC  B          ; Ancho en el registro B
2AFC  E1         480         POP  HL
2AFD  C9         490         RET
2AFE  C5         500 LINEA  PUSH BC
2AFF  D5         510         PUSH DE
2B00  E5         520         PUSH HL
2B01  DDE5       530         PUSH IX         ; Preserva los registros
2B03  DD21642B   540         LD   IX,BUFFER
2B07  0608       550         LD   B,8
2B09  C5         560         PUSH BC
2B0A  25         570         DEC  H
2B0B  2D         580         DEC  L
2B0C  CD1ABC     590         CALL #BC1A      ; Conoce la direccion del caracter
2B0F  ED43602B   600         LD   (NUEVOA),BC; Ancho del caracter en NUEVOA
2B13  110008     610         LD   DE,2048    ; Actualiza DE
2B16  C1         620         POP  BC         ; Recupera el numero de lineas, (8)
2B17  C5         630 OBUCLE PUSH BC         ; Lo guarda otra vez
2B18  3A632B     640         LD   A,(ANCHO)  ; Conoce el numero de caracteres
2B1B  47         650         LD   B,A
2B1C  E5         660         PUSH HL         ; Guarda la direccion de comienzo
                 670 ;                         de la linea
2B1D  C5         680 BUCLE1 PUSH BC         ; Guarda el numero de caracteres
2B1E  ED4B602B   690         LD   BC,(NUEVOA); Conoce la anchura en bytes
                 700 ;                         del caracter
2B22  3A622B     710 BUCLE  LD   A,(DIR)    ; Extrae el byte de la memoria
                 720 ;                         de pantalla,
2B25  FE00       730         CP   0          ; y lo introduce en el almacen
                 740 ;                          temporal de memoria
2B27  2817       750         JR   Z,PONESP   ; dependiendo del valor de DIR
```

```
2B29  7E        760         LD   A,(HL)
2B2A  DD7700    770         LD   (IX),A
2B2D  DD23      780 CAROK   INC  IX          ; Siguiente byte...
2B2F  23        790         INC  HL          ; Escribe una linea del caracter
2B30  10F0      800         DJNZ BUCLE
2B32  C1        810         POP  BC
2B33  10E8      820         DJNZ BUCLE1      ; Realizara todo esto para
                830 ;                          todos los caracteres
                840 ;                        de la ventana
2B35  E1        850         POP  HL
2B36  19        860         ADD  HL,DE       ; Obtiene la direccion en RAM de
                870 ;                          pantalla, de la linea
2B37  C1        880         POP  BC
2B38  10DD      890         DJNZ DBUCLE      ; Repite para todo el caracter
2B3A  DDE1      900         POP  IX
2B3C  E1        910         POP  HL
2B3D  D1        920         POP  DE
2B3E  C1        930         POP  BC          ; Restaura los registros
2B3F  C9        940         RET
2B40  DD7E00    950 PONESP  LD   A,(IX)
2B43  77        960         LD   (HL),A
2B44  18E7      970         JR   CAROK
2B46  DD6608    980 DESPLZ  LD   H,(IX+8)    ; Desplaza un area
2B49  DD6E06    990         LD   L,(IX+6)
2B4C  DD5604   1000         LD   D,(IX+4)
2B4F  DD5E02   1010         LD   E,(IX+2)
2B52  DD4600   1020         LD   B,(IX)
2B55  3A202E   1030         LD   A,(PAPEL)
2B58  25       1040         DEC  H
2B59  2D       1050         DEC  L
2B5A  15       1060         DEC  D
2B5B  1D       1070         DEC  E
2B5C  CD50BC   1080         CALL #BC50
2B5F  C9       1090         RET
2B60  0000     1100 NUEVOA DEFW 00
2B62  00       1110 DIR    DEFB 0
2B63  00       1120 ANCHO  DEFB 0
2B64           1130 BUFFER DEFS 700
2E20  00       1140 PAPEL  DEFB 0
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 direccion=40200
30 CLS : FOR I=1 TO 10
40 PRINT "12345678901234567890123456 78"
50 NEXT
60 PEN 3:LOCATE 2,3 : PRINT "** HOLA *"
70 G$=INKEY$ : IF G$="" THEN 70
80 IF ASC(G$)=240 THEN dir=1
90 IF ASC(G$)=241 THEN dir=0
100 CALL 41000,2,2,19,6,dir
110 GOTO 70
```

Comentarios

La rutina es no relocalizable, debido al empleo extenso de subrutinas. Los bytes proporcionados son para la dirección 41000.

Las tres últimas rutinas utilizan temporalmente un área de la memoria. Esto se utiliza de la siguiente forma. Para la DESPIZR y la DSPDERR cada línea de pantalla de la ventana desplazada se mueve hacia el lado requerido. Sin embargo, antes de desplazarse, el área que se va a perder por el desplazamiento es copiada temporalmente en el área de memoria antes indicado. Esta sólo contendrá la cantidad de memoria necesaria para definir un cuadro de carácter, por lo que el máximo será de 32 bytes para un cuadro de carácter en modo 0. Cuando finaliza el desplazamiento, copiamos el contenido del área de memoria temporal y se escribe en la línea del lado opuesto de la ventana que acaba de ser desplazada. Como sólo estamos copiando el contenido de la memoria de pantalla, la información en cuanto al color y los posibles gráficos se mantiene, así como los datos del carácter. En la DESPVR, debemos retener toda una fila de pantalla. Esto nos dará un máximo de (80*8) bytes, teniendo en cuenta que alguien querrá desplazar la pantalla en su totalidad. Esto implica 80 bytes de ancho y son necesarios 8 bytes para cada fila de pantalla en términos de líneas de pantalla. Cada vez que haces un desplazamiento, la línea de pantalla que de otro modo se perdería, se copia en esta área de memoria temporal o *buffer*. Así, después de un desplazamiento, el contenido del *buffer* se copia en las áreas de memoria de pantalla apropiadas, para restablecer la imagen en el otro extremo de la ventana.

5
Más rutinas de pantalla

Este capítulo presenta un variado surtido de rutinas para la manipulación de la pantalla. Comenzaremos viendo unos cuantos métodos para limpiar la pantalla.

Borrando la pantalla

La forma más fácil de hacerlo desde código máquina es llamar a la rutina situada en la dirección &BC14. Esta colocará toda la pantalla con la tinta 0; justamente lo que hace CLS. Sin embargo, el cursor de texto no retornará a la esquina superior izquierda de la pantalla. Alternativamente, el equivalente más directo a CLS es:

```
LD    A,12
CALL  #BB5A
```

que limpiará la ventana de texto y localizará el cursor en la esquina superior izquierda de la ventana. La orden CLG, por supuesto, también sirve para limpiar la pantalla, pero la llamada a &BC14 realiza este trabajo igualmente bien.

Sin embargo, todos estos métodos de limpieza de la pantalla son demasiado repentinos y en alguna ocasión es útil tener una rutina que

desvanezca gradualmente la imagen de la pantalla, en vez de borrarla súbitamente. Por ejemplo, podrías usar una rutina que difumine poco a poco la página de presentación de un programa que hayas escrito, manteniendo tu nombre expuesto el mayor tiempo posible. Para ello, aquí tienes un par de rutinas que te ofrecen esta posibilidad.

DCLS

Esta rutina desvanece la pantalla hasta que toda ella esté con la tinta 0. Es relocalizable.

Requisitos de entrada: Desde BASIC o código máquina; CALL dirección. No es necesario ningún parámetro.

Condiciones de salida: Los registros AF, HL y DE son alterados.

Longitud: 18 bytes.

```
                10 ;       ** DCLS **
9D08            20         ORG  40200
9D08  1EFE      30         LD   E,254    ; Mascara inicial
9D0A  2100C0    40 BUCLE0  LD   HL,#C000; Comienzo de la memoria de pantalla
9D0D  7B        50 BUCLE1  LD   A,E      ; Carga la mascara en A
9D0E  A6        60         AND  (HL)     ; Enmascara un byte de la RAM
9D0F  77        70         LD   (HL),A   ; de pantalla, y lo devuelve
                80 ;                       a su posicion de origen
9D10  23        90         INC  HL       ; Siguiente byte de la RAM
9D11  7D       100         LD   A,L
9D12  B4       110         OR   H        ; Es la direccion 0000?
9D13  20F8     120         JR   NZ,BUCLE1; Si no, vuelve otra vez
9D15  CB13     130         RL   E        ; Rotacion de la mascara
9D17  38F1     140         JR   C,BUCLE0; Repetir mientras 'C' sea 1
9D19  C9       150         RET
```

Pruébalo con:

```
10 MODE 1:FOR T=1 TO 25
20 PRINT STRING$(39,"*"):NEXT:CALL 40200
```

Comentarios

Esta rutina trabaja desplazando repetidamente un 0 a través de un byte que inicialmente tiene todos sus bits a 1. Este byte sirve como máscara. Cada byte de la memoria de pantalla es enmascarado con el byte máscara mediante la operación AND y devuelto otra vez a la memoria de pantalla. Para dar un poco de velocidad al asunto, el final de la memoria de pantalla se busca chequeando el par de registros HL, hasta que conten-

gan el valor cero. La memoria de pantalla termina en la dirección &FFFF y si incrementamos HL en el instante en que contenga este valor, el nuevo contenido será 0000. Observa también la forma en que el programa detecta si los 8 bits del byte han sido puestos a cero. Esperamos hasta que el *flag* C contenga un cero, indicando con ello que el 0 que rotaba por el byte ha terminado sus desplazamientos y que ya ha pasado por todos los bits del byte.

CLSDESPLZ (CLS DESPLaZando)

Esta rutina limpia la pantalla desplazando su contenido 25 líneas hacia arriba o hacia abajo. La rutina es relocalizable.

Requisitos de entrada: Desde el BASIC usaremos la sentencia CALL dirección, desp. Donde desp = 0 indica que queremos un desplazamiento hacia abajo. Si lo que queremos es un desplazamiento hacia arriba, lo indicaremos con desp = 1.
Desde código máquina, IX contiene la dirección del byte de memoria que contiene el valor desp, que especifica qué tipo de desplazamiento se requiere. A = 1.

Condiciones de salida: Todos los registros alterados.

Longitud: 48 bytes, incluyendo el almacenamiento temporal.

```
                    10 ;       ** CLSDESPLZ **
9D08                20         ORG  40200
9D08  DD7E00        30         LD   A,(IX)
9D0B  32369D        40         LD   (DIR),A
9D0E  CD99BB        50         CALL #BB99
9D11  CD2CBC        60         CALL #BC2C      ; Obtiene la tinta para el papel
9D14  32379D        70         LD   (PAPEL),A  ; de textos y lo almacena codificado
9D17  2600          80         LD   H,0        ; Borde izquierdo,
9D19  2E00          90         LD   L,0        ; fila superior,
9D1B  1E18         100         LD   E,24       ; y linea inferior a desplazar
9D1D  CD17BC       110         CALL #BC17      ; Obtiene la ultima columna
9D20  50           120         LD   D,B
9D21  0619         130         LD   B,25
9D23  C5           140 BUCLE   PUSH BC
9D24  D5           150         PUSH DE
9D25  E5           160         PUSH HL
9D26  3A369D       170         LD   A,(DIR)
9D29  47           180         LD   B,A
9D2A  3A379D       190         LD   A,(PAPEL)
9D2D  CD50BC       200         CALL #BC50
9D30  E1           210         POP  HL
9D31  D1           220         POP  DE
9D32  C1           230         POP  BC
9D33  10EE         240         DJNZ BUCLE
9D35  C9           250         RET
9D36  00           260 DIR     DEFB 0
9D37  00           270 PAPEL   DEFB 0
```

Pruébalo con esto:

```
10 FOR T=1 TO 25:PRINT STRING$(39,"*")
20 NEXT:CALL 40200,1
```

Comentarios

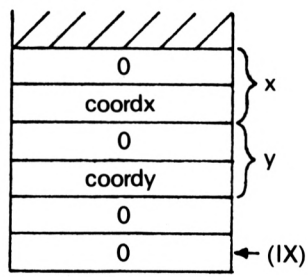
Los bytes hexadecimales listados son para la dirección 40200. Cuando relocalices la rutina, no olvides cambiar la dirección de la memoria temporal utilizada. En este programa, sólo se maneja una nueva rutina de la ROM, la llamada a &BC17. En el anterior capítulo vimos la rutina de desplazamiento. Esta rutina en &BC17 nos devuelve la última columna y fila de pantalla disponibles en el modo actual de pantalla. Es denominada SCR CHAR LIMITS por AMSOFT, y a su regreso el registro B y el registro C contienen, respectivamente, la última columna de pantalla y la última fila disponible de la pantalla. Utilizamos esta información para darle a la rutina el número correcto de columnas a desplazar para cada modo de pantalla.

En la programación de rutinas de juegos, es a menudo útil detectar si el espacio de un carácter está o no ocupado por otra cosa antes de colocar otro carácter en ese espacio. Las siguientes dos rutinas están diseñadas para ayudarte en estas situaciones.

RCARAC (Reconoce CARACteres)

Esta rutina devolverá el código ASCII de cualquier carácter identificable en una posición específica de la pantalla.

Requisitos de entrada: Desde BASIC, CALL dirección,x,y,@caract%, donde x es la coordenada X, y es la coordenada Y, caract% es la variable que contendrá, al retornar, el resultado de la llamada.
Desde el código máquina, el registro IX apuntará a un bloque de parámetros de 6 bytes como el de la figura. A = 3.



Bloque de parámetros para RCARAC

Condiciones de salida: En BASIC, caract% contendrá el código ASCII si el carácter fue reconocido. En caso de que el carácter no fuera identificado, la variable contendrá el valor 256.
Desde código máquina, la posición (IX) contendrá el código ASCII de un carácter "legal" y (IX+1)=0. Para un carácter no reconocido por el sistema, entonces (IX+0)=0 y (IX+1)=1. Alternativamente, C=1 (*flag* de acarreo activado) y A=código ASCII de un carácter legal; en caso contrario, A=0 y el *flag* de acarreo "C" puesto a cero.

Longitud: 31 bytes.

```
9D08            10        ORG  40200
9D08  FE03      20        CP   3
9D0A  C0        30        RET  NZ        ; Retorno para un numero
                40 ;                       erroneo de parametros
9D0B  DD6604    50        LD   H,(IX+4)
9D0E  DD6E02    60        LD   L,(IX+2)
9D11  CD75BB    70        CALL #BB75     ; Posiciona el cursor
9D14  CD60BB    80        CALL #BB60     ; Extrae el caracter
9D17  DD6E00    90        LD   L,(IX+0)
9D1A  DD6601   100        LD   H,(IX+1)
9D1D  3005     110        JR   NC,NOCAR ; Si el caracter no es conocido...
9D1F  77       120        LD   (HL),A    ; Carga el codigo ASCII en
9D20  23       130        INC  HL        ; la variable
9D21  3600     140        LD   (HL),0
9D23  C9       150        RET
9D24  3600     160 NOCAR  LD   (HL),0    ; Si el caracter no es conocido,
9D26  23       170        INC  HL        ; viene aqui
9D27  3601     180        LD   (HL),1
9D29  C9       190        RET
```

Pruébalo con esto:

```
10 LOCATE 5,10:PRINT "UN-CARACTER"
20 F%=0:CALL 40200,10,10,@F%
30 PRINT F%
```

Comentarios

La rutina debe llamarse en compañía de tres parámetros; de otro modo, se provocará un regreso inmediato al BASIC. Recuerda que, utilizando variables con el prefijo @, la variable tiene que haber sido declarada previamente. La causa más típica de obtener 256 al retorno es que una orden de gráficos como PLOT o DRAW haya dejado un punto o una línea en ese espacio de carácter en particular, alterando con ello la imagen.

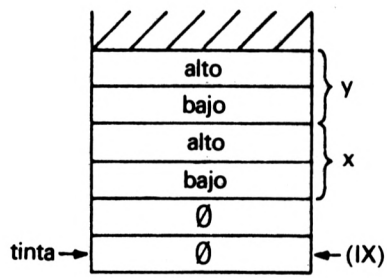
Una segunda causa para que la rutina devuelva el 256 es que los colores del papel y la pluma hayan sido cambiados después de que la imagen fuera impresa en pantalla. La forma de evitar esto último es chequear la posición del carácter con todas las combinaciones de papel y pluma. Sin embargo, como lo que normalmente buscamos es la presencia o ausencia de algo, la rutina sigue siendo muy útil.

Una rutina similar es denominada CPUNTO, pero ésta nos devuelve el color de la tinta encontrado en un punto determinado de la pantalla.

CPUNTO (Color del PUNTO)

Devuelve al usuario el color de la tinta de un punto *(pixel)* específico. La rutina es relocalizable.

Requisitos de entrada: Desde BASIC, usaremos CALL dirección,x, y, @J%, donde x,y son las coordenadas del punto y J% es la variable que contendrá el valor de la tinta al regreso de la rutina. Si partimos desde el código máquina, IX apuntará a un bloque de parámetros como el de la figura. El registro A = 3.



Bloque de parámetros de CPUNTO

Condiciones de salida: Todos los registros son alterados. Si se transfiere un número erróneo de parámetros, se regresa inmediatamente al BASIC o al programa en código máquina que hizo la llamada.

Longitud: 29 bytes.

```
                10 ;      ** CPUNTO **
9D08            20        ORG  40200
9D08  FE03      30        CP   3
9D0A  C0        40        RET  NZ       ; Si no hay 3 parametros, retorno
9D0B  DD6E02    50        LD   L,(IX+2)
```

```
9D0E  DD6603      60        LD   H,(IX+3)
9D11  DD5E04      70        LD   E,(IX+4)
9D14  DD5605      80        LD   D,(IX+5)
9D17  CDF0BB      90        CALL #BBF0   ; Obtiene la tinta de HL y DE
9D1A  DD6E00     100        LD   L,(IX+0)
9D1D  DD6601     110        LD   H,(IX+1)
9D20  77         120        LD   (HL),A  ; Almacena el valor de la tinta
                 130 ;                      en la variable
9D21  23         140        INC  HL
9D22  3600       150        LD   (HL),0
9D24  C9         160        RET
```

Pruébalo con esto:

```
10 F%=0
20 CALL 40200,80,390,,@F%
30 PRINT F%
```

Comentarios

Tanto en esta rutina como en RCARAC, si lo único que pretendes es usar las rutinas desde código máquina, entonces será más fácil llamar directamente a la rutina de la ROM sin poner el resultado en la variable apropiada. Las dos rutinas de ROM utilizadas son las siguientes:

&BB60: Esta fue discutida en el capítulo 4.

&BBF0: Esta rutina es accesible cuando DE contiene la coordenada X del punto en cuestión y HL contiene la coordenada Y. A la salida de la rutina, el registro A contendrá el color de la tinta.

La siguiente rutina es otra de las "decorativas". Invierte la pantalla complementando todos los bits de la memoria de pantalla que estén a 1, poniéndolos a 0, y viceversa.

PINVIERT (INVIERTe Pantalla)

Cambia la pantalla complementando cada byte de la memoria de pantalla. Sirve para los tres modos. La rutina es relocalizable.

Requisitos de entrada: CALL dirección.

Condiciones de salida: Son alterados los registros HL y AF.

Longitud: 12 bytes.

```
                    10 ;       ** PINVERT **
9D08                20         ORG  40200
9D08  2100C0        30         LD   HL,#C000
9D0B  7E            40 BUCLE   LD   A,(HL)    ; Extrae un byte de la pantalla
9D0C  2F            50         CPL  ;           Complementa el byte
9D0D  77            60         LD   (HL),A    ; Vuelve a almacenarlo en pantalla
9D0E  23            70         INC  HL        ; Siguiente byte
9D0F  7D            80         LD   A,L
9D10  B4            90         OR   H
9D11  20F8         100         JR   NZ,BUCLE ; Repite si HL no es cero
9D13  C9           110         RET
```

Pruébalo con esto:

```
10 FOR T=1 TO 25:PRINT STRING$(39,"*")
20 NEXT:CALL 40200
```

Comentarios

Cada byte de la memoria de pantalla desde &C000 hasta &FFFF es complementado. Esto significa simplemente que cada 0 es convertido en un 1 y que cada 1 es reemplazado por un 0. Por ejemplo, el byte 10101010 al ser complementado se convierte en 01010101. El cambio de color que se produce puede ser realmente interesante, en especial en el modo 0 con varios colores a la vez en pantalla. Si hacemos una segunda llamada a la rutina, la pantalla volverá a su estado original.

La siguiente rutina es sencilla, pero con un fuerte uso de la memoria. Puede ser extremadamente útil cuando necesitemos rápidas alteraciones de la imagen que se presenta en la pantalla. Vamos a utilizar 16K de memoria, comenzando desde la posición 26000 en decimal, como una "pantalla" temporal que puede contener una copia de la propia pantalla. Podemos dibujar una imagen en la pantalla, copiarla en esta segunda pantalla, y después dibujar una segunda imagen. Cuando deseemos visualizar en pantalla el contenido de la segunda pantalla, transferiremos simplemente el contenido de la RAM desde la posición 26000 a la memoria de pantalla.

SEGUNDAP (SEGUNDA Pantalla)

Esta rutina permite el uso de un área de la memoria como una segunda "pantalla". Esta zona de memoria puede contener una copia de la imagen actual de la pantalla. La pantalla así salvada en la RAM puede ser restaurada posteriormente de una forma casi instantánea.

Requisitos de entrada: Desde el BASIC, usaremos CALL dirección, n; donde n especifica el sentido de la transferencia de datos: n = 0 significa que la transferencia se hace desde la pantalla a la RAM y n = 1 que la transferencia se hace desde la RAM a la memoria de pantalla. Partiendo del código máquina, IX apunta hacia un byte de memoria que contendrá el valor de n. El registro A = 1.

Condiciones de salida: Todos los registros alterados.

Longitud: 21 bytes, más el área de memoria comprendida entre 26000 y 42384, ambos inclusive.

```
                10 ;       ** SEGUNDAP **
61A8            20         ORG  25000
61A8  FE01      30         CP   1
61AA  C0        40         RET  NZ        ; Retorna ante un numero erroneo
                50 ;                        de parametros
61AB  DD7E00    60         LD   A,(IX)
61AE  FE00      70         CP   0
61B0  280C      80         JR   Z,ABAJO   ; Si es 0, de pantalla a memoria
61B2  1100C0    90         LD   DE,#C000 ; Transfiere de memoria a pantalla
61B5  219065   100         LD   HL,26000
61B8  010040   110         LD   BC,#4000
61BB  EDB0     120         LDIR
61BD  C9       130         RET
61BE  119065   140 ABAJO   LD   DE,26000
61C1  2100C0   150         LD   HL,#C000
61C4  010040   160         LD   BC,#4000
61C7  EDB0     170         LDIR
61C9  C9       180         RET
```

Pruébalo con esto:

```
10 CALL 25000,0
20 CLS: INPUT"Pulsa ENTER para restaurar
   la pantalla",a$
30 CALL 25000,1
40 GOTO 40:REM Se evitan desplazamientos
```

Comentarios

Advertirás que el área de memoria utilizada para el almacenamiento temporal puede tener una localización un tanto caprichosa. La razón de esto es sencilla. El área de la segunda pantalla necesita tener una longitud de 16384 bytes, y si usáramos posiciones de memoria más altas, terminaríamos escribiendo sobre el bloque de rutinas de salto del *firmware*. El resultado de esto, como podrás suponer, es un desastroso accidente del sistema. Además nuestra rutina en código máquina deberá estar o bien

entre el final de nuestra "pantalla" y del bloque de saltos del *firmware*, o bien en una dirección inferior a 26000. A pesar de todas estas contrariedades, la rutina es relocalizable.

Otros puntos a observar son los siguientes:

1. Sólo se obtendrán resultados coherentes si el modo de pantalla presente en el momento de restaurar la segunda pantalla coincide con el modo que se estaba usando en el momento de almacenar la pantalla.
2. Una imagen debe ser guardada antes de que ocurra ningún desplazamiento. Cuando queramos "cargar" la segunda pantalla desde la RAM deberá hacerse inmediatamente después de un cambio de modo. Las operaciones de desplazamiento alteran la forma en que la memoria de pantalla se corresponde con posiciones particulares de pantalla.

SEGUNDAP tiene el inconveniente de destruir la imagen que había en pantalla, al realizar la transferencia desde nuestra segunda pantalla a la pantalla principal. La siguiente rutina evita este problema mediante un intercambio de los contenidos de las dos pantallas.

PCANJE (CANJE de Pantallas)

Esta rutina canjea las dos "pantallas" poniendo así en la memoria la imagen de la pantalla, empezando desde la posición 26000 y presentando por pantalla el contenido de la RAM desde la posición 26000. La rutina es relocalizable dentro de los límites impuestos por las notas de SEGUNDAP.

Requisitos de entrada: CALL dirección.

Condiciones de salida: Todos los registros son alterados.

Longitud: 18 bytes, más la memoria comprendida entre las posiciones 26000 y 42384, ambas inclusive.

```
                10 ;       ** PCANJE **
61A8            20         ORG  25000
61A8  119065    30         LD   DE,26000 ; Inicializacion de registros
61AB  2100C0    40         LD   HL,#C000
61AE  4E        50 BUCLE   LD   C,(HL)   ; Toma un byte de la pantalla
61AF  1A        60         LD   A,(DE)   ; Toma otro del almacen temporal
61B0  77        70         LD   (HL),A   ; Siguiente intercambio de bytes
61B1  79        80         LD   A,C
61B2  12        90         LD   (DE),A
61B3  23       100         INC  HL       ; Apunta hacia los siguientes bytes
61B4  13       110         INC  DE
61B5  7D       120         LD   A,L      ; HL es cero cuando toda la
61B6  B4       130         OR   H        ; pantalla sea barrida
61B7  20F5     140         JR   NZ,BUCLE
61B9  C9       150         RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 CALL 25000
20 CLS : DRAW 100,100 : DRAW 100,200 :
   DRAW 0,300
30 FOR I=0 TO 200:NEXT
40 CALL 25000
50 GOTO 30
```

Comentarios

El intercambio de las dos pantallas ofrece un tipo de desvanecimiento como el que se obtiene con un proyector de diapositivas. El desvanecimiento puede hacerse más lento si insertamos un bucle de retardo en el código máquina antes listado. El siguiente programa BASIC demuestra también el funcionamiento de la rutina en código máquina, suponiendo que ésta ya se encuentra en la dirección 25000.

```
10 CLS
20 FOR K=1 TO 20
30 LOCATE 15,K:PRINT "! HOLA AQUI !"
40 NEXT
50 CALL 25000
60 CLS : DRAW 100,100 : DRAW 100,200 :
   DRAW 0,300
70 FOR I=0 TO 300:NEXT:REM retardo
80 CALL 25000
90 GOTO 70
```

Sería interesante sacar un pleno rendimiento de esta rutina en tiempo real. Probablemente alguna vez hayas visto impresionantes dibujos en la pantalla de un ordenador. Lo normal es programar estos dibujos punto a punto, bien mediante funciones o bien asignando a cada punto de la pantalla un estado determinado. El tiempo de procesamiento de estos dibujos puede llegar a ser extremadamente largo, y su programa ocupa fácilmente la totalidad de la memoria. Existe un segundo método para dibujos artísticos o de precisión, que está muy alejado de todo lo referente a números y complicadas ecuaciones. En este caso, el usuario puede ver el dibujo según lo va imaginando y componiendo. Con este método, una especie de pluma puede ser desplazada a voluntad por la pantalla marcando las líneas o puntos que se deseen. Pueden ser incluidos comentarios o cadenas de caracteres en cualquier posición de la pantalla. Una vez finalizado el trabajo, se puede recurrir a una rutina como PCANJE para almacenar en memoria el contenido de la pantalla. Seguidamente se puede

guardar el contenido del área de memoria de la segunda pantalla, en la unidad de almacenamiento (disco o cassette). Este es el tipo de dibujos que se pueden realizar con la siguiente rutina.

ACCION

Permite al usuario rellenar la pantalla a voluntad, tanto con texto como con gráficos. Se dispone de todos los colores que permita cada modo de pantalla. La velocidad de desplazamiento del cursor de gráficos puede ser determinada en el momento de acceder a la rutina. Es relocalizable. La rutina está preparada para trabajar en el modo 1, pero es fácilmente alterable para utilizarla en los demás modos.

Requisitos de entrada: Desde el BASIC, llamaremos a la rutina mediante CALL40000, vel; donde vel es la velocidad que se quiere dar al cursor de gráficos. El valor de vel se refiere al número de barridos de pantalla que se efectuarán entre cada desplazamiento del cursor. Como sabes, el ordenador envía información a la pantalla 50 veces por segundo. Así, un valor de vel = 50 hará que transcurra un segundo completo entre cada movimiento del cursor. Partiendo desde código máquina, IX apunta a un bloque de la memoria en donde está almacenado un valor que representa al retardo. A = 1.

Condiciones de salida: El color será el último utilizado en la rutina. Todos los registros son alterados.

Longitud: 418 bytes.

```
                    10 ;      ** ACCION **
9C40                20        ORG  40000
9C40  FE01          30        CP   1
9C42  C0            40        RET  NZ
9C43  DD7E00        50        LD   A,(IX)
9C46  32E29D        60        LD   (RET),A
9C49  CDCCBB        70        CALL #BBCC       ; Obtiene las coordenadas
                    80 ;                         del origen
9C4C  ED53DD9D      90        LD   (TEMPX),DE  ; de graficos, y las almacena
9C50  22DF9D       100        LD   (TEMPY),HL  ; temporalmente
9C53  110000       110        LD   DE,0
9C56  218F01       120        LD   HL,399
9C59  CDC9BB       130        CALL #BBC9       ; Impone el nuevo origen de coord.
9C5C  3E01         140        LD   A,1
9C5E  CDDEBB       150        CALL #BBDE       ; Fija la tinta de graficos
9C61  3E17         160        LD   A,23        ; Modo de graficos 'XOR'
9C63  CD5ABB       170        CALL #BB5A
9C66  3E01         180        LD   A,1
9C68  CD5ABB       190        CALL #BB5A
9C6B  110000       200        LD   DE,0
9C6E  210000       210        LD   HL,0
9C71  E5           220        PUSH HL
```

```
9C72  D5        230         PUSH DE          ; Guarda la posicion del punto
9C73  CDEABB    240         CALL #BBEA       ; Imprime el punto-guia
                250 ;                          en el origen
9C76  3AE29D    260 TECLAS  LD   A,(RET)
9C79  47        270         LD   B,A
9C7A  CD19BD    280 RETARD  CALL #BD19
9C7D  10FB      290         DJNZ RETARD
9C7F  3E00      300         LD   A,0         ; Comienza a explorar el teclado
9C81  CD1EBB    310         CALL #BB1E       ; Tecla 0; desplazamiento
9C84  203B      320         JR   NZ,ARRIBA   ; hacia arriba
9C86  3E02      330         LD   A,2
9C88  CD1EBB    340         CALL #BB1E
9C8B  2079      350         JR   NZ,ABAJO    ; Tecla 2, abajo
9C8D  3E01      360         LD   A,1
9C8F  CD1EBB    370         CALL #BB1E
9C92  C24B9D    380         JP   NZ,DER      ; Tecla 1, derecha
9C95  3E08      390         LD   A,8
9C97  CD1EBB    400         CALL #BB1E
9C9A  C2669D    410         JP   NZ,IZ       ; Tecla 8, izquierda
9C9D  3E3A      420         LD   A,58
9C9F  CD1EBB    430         CALL #BB1E
9CA2  C2A99D    440         JP   NZ,ESCRBE   ; Tecla 58 (E), escribir
9CA5  3E22      450         LD   A,34
9CA7  CD1EBB    460         CALL #BB1E
9CAA  C2CD9D    470         JP   NZ,ORIGEN   ; Tecla 34, (O), origen
9CAD  3E3E      480         LD   A,62
9CAF  CD1EBB    490         CALL #BB1E
9CB2  C2819D    500         JP   NZ,CCOLOR   ; Tecla 62, (C), cambio de color
9CB5  3E35      510         LD   A,53
9CB7  CD1EBB    520         CALL #BB1E
9CBA  C29C9D    530         JP   NZ,FINAL    ; Tecla (F), final
9CBD  CA769C    540         JP   Z,TECLAS
9CC0  C9        550         RET
9CC1  3E01      560 ARRIBA  LD   A,1
9CC3  CD1EBB    570         CALL #BB1E       ; Comprueba si es una diagonal
9CC6  CACF9C    580         JP   Z,U1        ; a la derecha
9CC9  010200    590         LD   BC,2
9CCC  C5        600         PUSH BC
9CCD  1812      610         JR   U2
9CCF  3E08      620 U1      LD   A,8
9CD1  CD1EBB    630         CALL #BB1E       ; Comprueba si es una diagonal
9CD4  CADD9C    640         JP   Z,U11       ; a la izquierda
9CD7  01FEFF    650         LD   BC,-2
9CDA  C5        660         PUSH BC
9CDB  1804      670         JR   U2
9CDD  010000    680 U11     LD   BC,0
9CE0  C5        690         PUSH BC
9CE1  3E2F      700 U2      LD   A,47
9CE3  CD1EBB    710         CALL #BB1E       ; El punto no se borra si la
9CE6  C2F29C    720         JP   NZ,U3       ; barra esta pulsada
9CE9  C1        730         POP  BC
9CEA  D1        740         POP  DE
9CEB  E1        750         POP  HL          ; Recupera las coordenadas
9CEC  E5        760         PUSH HL          ; del punto-guia
9CED  D5        770         PUSH DE
9CEE  C5        780         PUSH BC
9CEF  CDEABB    790         CALL #BBEA       ; Lo elimina de la pantalla
9CF2  C1        800 U3      POP  BC
9CF3  D1        810         POP  DE
9CF4  E1        820         POP  HL          ; Recupera las coordenadas
9CF5  23        830         INC  HL
9CF6  23        840         INC  HL          ; Actualiza la coordenada Y
9CF7  E5        850         PUSH HL          ; guardandola despues
9CF8  62        860         LD   H,D
9CF9  6B        870         LD   L,E
9CFA  09        880         ADD  HL,BC       ; Actualiza y guarda la
9CFB  54        890         LD   D,H
9CFC  5D        900         LD   E,L
9CFD  E1        910         POP  HL
9CFE  E5        920         PUSH HL
```

```
9CFF  D5          930        PUSH DE          ; coordenada X
9D00  CDEABB      940        CALL #BBEA       ; Imprime el nuevo punto
9D03  C3769C      950        JP   TECLAS
9D06  3E01        960 ABAJO  LD   A,1
9D08  CD1EBB      970        CALL #BB1E       ; Comprueba si es diagonal
9D0B  CA149D      980        JP   Z,A1        ; hacia la derecha
9D0E  010200      990        LD   BC,2
9D11  C5         1000        PUSH BC
9D12  1812       1010        JR   A2
9D14  3E08       1020 A1     LD   A,8
9D16  CD1EBB     1030        CALL #BB1E       ; Comprueba si es diagonal
9D19  CA229D     1040        JP   Z,A11       ; a la izquierda
9D1C  01FEFF     1050        LD   BC,-2
9D1F  C5         1060        PUSH BC
9D20  1804       1070        JR   A2
9D22  010000     1080 A11    LD   BC,0        ; No hay variacion horizontal
9D25  C5         1090        PUSH BC
9D26  3E2F       1100 A2     LD   A,47
9D28  CD1EBB     1110        CALL #BB1E       ; El punto no se borra si
9D2B  C2379D     1120        JP   NZ,A3       ; la barra esta pulsada
9D2E  C1         1130        POP  BC
9D2F  D1         1140        POP  DE
9D30  E1         1150        POP  HL          ; Recupera las coordenadas
9D31  E5         1160        PUSH HL
9D32  D5         1170        PUSH DE          ; y las guarda de nuevo
9D33  C5         1180        PUSH BC
9D34  CDEABB     1190        CALL #BBEA       ; Borra el punto de la pantalla
9D37  C1         1200 A3     POP  BC
9D38  D1         1210        POP  DE
9D39  E1         1220        POP  HL          ; Recupera los registros
9D3A  2B         1230        DEC  HL
9D3B  2B         1240        DEC  HL          ; Un punto hacia abajo
9D3C  E5         1250        PUSH HL          ; Nueva coordenada Y
9D3D  62         1260        LD   H,D
9D3E  6B         1270        LD   L,E
9D3F  09         1280        ADD  HL,BC       ; Variacion horizontal
9D40  54         1290        LD   D,H
9D41  5D         1300        LD   E,L
9D42  E1         1310        POP  HL
9D43  E5         1320        PUSH HL
9D44  D5         1330        PUSH DE          ; Nueva coordenada X
9D45  CDEABB     1340        CALL #BBEA
9D48  C3769C     1350        JP   TECLAS
9D4B  3E2F       1360 DER    LD   A,47
9D4D  CD1EBB     1370        CALL #BB1E       ; Comprueba si se pulsa la barra
9D50  C25A9D     1380        JP   NZ,D1
9D53  D1         1390        POP  DE
9D54  E1         1400        POP  HL
9D55  E5         1410        PUSH HL
9D56  D5         1420        PUSH DE          ; Recupera las coordenadas
                 1430 ;                        del punto
9D57  CDEABB     1440        CALL #BBEA       ; Lo elimina
9D5A  D1         1450 D1     POP  DE
9D5B  E1         1460        POP  HL
9D5C  13         1470        INC  DE          ; Ahora incrementa X en dos
9D5D  13         1480        INC  DE
9D5E  E5         1490        PUSH HL
9D5F  D5         1500        PUSH DE
9D60  CDEABB     1510        CALL #BBEA       ; Imprime el nuevo punto
9D63  C3769C     1520        JP   TECLAS
9D66  3E2F       1530 IZ     LD   A,47
9D68  CD1EBB     1540        CALL #BB1E
9D6B  C2759D     1550        JP   NZ,I1
9D6E  D1         1560        POP  DE          ; Si la barra no esta pulsada,
9D6F  E1         1570        POP  HL          ; se borra el punto anterior
9D70  E5         1580        PUSH HL
9D71  D5         1590        PUSH DE
9D72  CDEABB     1600        CALL #BBEA
9D75  D1         1610 I1     POP  DE
9D76  E1         1620        POP  HL
```

```
9D77  1B        1630          DEC  DE          ; Se decrementa la coordenada X
9D78  1B        1640          DEC  DE
9D79  E5        1650          PUSH HL
9D7A  D5        1660          PUSH DE
9D7B  CDEABB    1670          CALL #BBEA       ; Se imprime el nuevo punto
9D7E  C3769C    1680          JP   TECLAS
9D81  D1        1690 CCOLOR   POP  DE
9D82  E1        1700          POP  HL          ; Recupera la posicion;
9D83  E5        1710          PUSH HL
9D84  D5        1720          PUSH DE          ; borra el punto y lo vuelve
9D85  CDEABB    1730          CALL #BBEA       ; a imprimir en el nuevo color
9D88  3AE19D    1740          LD   A,(COLOR)
9D8B  FE0D      1750          CP   13
9D8D  2803      1760          JR   Z,TOPE      ; Si es la tinta 3, vuelve
9D8F  3C        1770          INC  A           ; a comenzar por la 0
                1780 ;                          En otro caso
9D90  1802      1790          JR   OTRA        ; incrementa el numero de tinta
9D92  3E00      1800 TOPE     LD   A,0
9D94  32E19D    1810 OTRA     LD   (COLOR),A   ; Almacena el nuevo color
9D97  CDDEBB    1820          CALL #BBDE       ; Lo impone para graficos
9D9A  181A      1830          JR   PPUNT
9D9C  ED5BDD9D  1840 FINAL    LD   DE,(TEMPX)
9DA0  2ADF9D    1850          LD   HL,(TEMPY)
9DA3  CDC9BB    1860          CALL #BBC9       ; Devuelve el origen
                1870 ;                          de coordenadas
                1880 ;                            a su posicion inicial
9DA6  D1        1890          POP  DE
9DA7  E1        1900          POP  HL
9DA8  C9        1910          RET  ;             Regresa al BASIC
9DA9  CD09BB    1920 ESCRBE   CALL #BB09       ; Limpia el buffer del teclado
9DAC  38FB      1930          JR   C,ESCRBE    ; Hay mas caracteres?
9DAE  CD18BB    1940          CALL #BB18       ; Espera un caracter
9DB1  FE0D      1950          CP   13          ; Si se trata de (ENTER),
9DB3  C2C09D    1960          JP   NZ,PUNTO    ; sale de este sub-modo,
9DB6  D1        1970 PPUNT    POP  DE
9DB7  E1        1980          POP  HL
9DB8  E5        1990          PUSH HL
9DB9  D5        2000          PUSH DE          ; recuperando la nueva
9DBA  CDEABB    2010          CALL #BBEA       ; posicion del punto-guia
9DBD  C3769C    2020          JP   TECLAS
9DC0  CDFCBB    2030 PUNTO    CALL #BBFC       ; Escribe el caracter
9DC3  D1        2040          POP  DE
9DC4  0610      2050          LD   B,16
9DC6  13        2060 BUCLE    INC  DE          ; Incrementa X el ancho de
                2070 ;                           un caracter
9DC7  10FD      2080          DJNZ BUCLE
9DC9  D5        2090          PUSH DE          ; Lo guarda
9DCA  C3A99D    2100          JP   ESCRBE
9DCD  D1        2110 ORIGEN   POP  DE
9DCE  E1        2120          POP  HL
9DCF  110000    2130          LD   DE,0
9DD2  210000    2140          LD   HL,0        ; Pone a cero las coordenadas
9DD5  E5        2150          PUSH HL
9DD6  D5        2160          PUSH DE
9DD7  CDEABB    2170          CALL #BBEA       ; e imprime el punto en el origen
9DDA  C3769C    2180          JP   TECLAS
9DDD  0000      2190 TEMPX    DEFW 0
9DDF  0000      2200 TEMPY    DEFW 0
9DE1  01        2210 COLOR    DEFB 1
9DE2  01        2220 RET      DEFB 1
```

Comentarios

La rutina no borra la pantalla al empezar; por lo que, si quieres hacer un dibujo que ocupe toda la pantalla, tendrás que borrarla antes de hacer la llamada. Una vez llamada, el cursor de gráficos aparece en la esquina

superior izquierda de la pantalla. Para desplazarlo, utiliza las teclas con "flechitas" que se usan normalmente para mover el cursor del texto. Cuando quieras trazar una línea, pulsa la barra espaciadora a la vez. El color se cambia pulsando la tecla "C". Pulsando la tecla "O" el cursor vuelve al origen de la pantalla. Para insertar texto o caracteres sueltos, pulsa la tecla "E" seguida de la cadena que quieras insertar y finaliza la operación pulsando ENTER. La rutina terminará cuando pulses la letra "F". En ese momento sería interesante que guardaras el contenido de la pantalla primero en una segunda pantalla y después en la unidad de almacenamiento.

Como ya dije antes, la rutina aquí listada funcionará correctamente en el modo 1. Para poder utilizarla en los otros modos, tendrás que alterar la rutina de la siguiente forma:

En las líneas 590, 650, 990, 1050, cambiar el 2 por un 4 para el modo 0, y por un 1 para el modo 2.

En las líneas 1470, 1480, 1630, 1640, introducir dos incrementos más de DE para el modo 0, o suprimir uno de los que hay, para el modo 2.

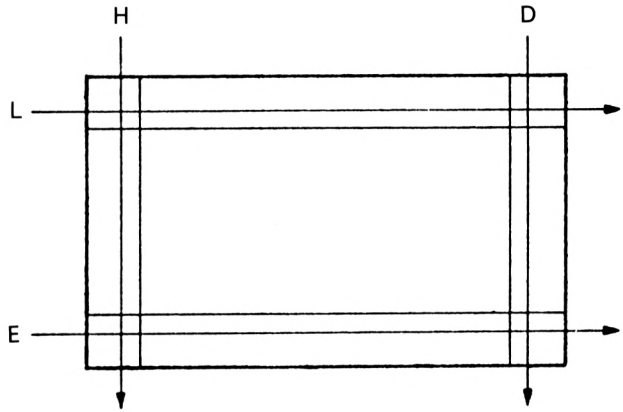
Por último, la línea 2050 indica el ancho de los caracteres para cada modo; por tanto, para el modo 2 el registro B debe contener 8; en el modo 1, B=16, y en el modo 0 B=32.

Puedes modificar la rutina de forma que admita trabajar con los diferentes modos de pantalla. Si quieres interaccionar esta rutina con SEGUNDAP, recuerda que debes relocalizarla en una dirección inferior a 26000.

Rutinas de llenado

Recordarás cómo en el capítulo 3 listé una rutina que dibuja rectángulos en la pantalla, con la opción de que estuvieran vacíos o rellenados. Hay una gran variedad de formas para que un área de la pantalla pueda ser rellenada con un color. Más adelante en esta sección veremos una rutina de propósito general para rellenar con líneas horizontales, la cual puede ser la base para las rutinas de llenado de propósito general. Sin embargo, antes de ver esto, puede ser interesante un breve estudio de las rutinas de llenado residentes en el sistema, y de las que podemos disponer.

SCR FILL BOX, localizada en &BC44, rellenará un área de pantalla con un color específico. Sin embargo, hay algunas limitaciones en la resolución del área rellenada, tales como que los límites se especifican en términos de espacios de caracteres. Para acceder a la rutina, A contiene el código del color, que puede ser obtenido del número de tinta normal mediante la rutina de la dirección &BC2C. Esto fue detallado en el capítulo 4. Los registros HL y DE especifican el área que tiene que rellenarse según la siguiente figura.



SCR FLOOD BOX nos permite una mayor resolución, pero es un poco más difícil de utilizar. Se accede mediante una llamada a la dirección &BC47 y rellenará el área especificada con el color de la tinta cuyo valor codificado está en el registro C. Los límites del área a rellenar están dados en términos de direcciones de la RAM de pantalla. Normalmente esto implica que haya que convertir la posición de un punto o de un carácter en una dirección de la memoria de pantalla. Existen rutinas en la ROM que nos permiten hacer esto tal y como vimos en el capítulo 4. Sin embargo, no entraré en más detalles por ahora. Es suficiente decir que el par HL contiene la dirección de la esquina superior izquierda del área que ha de rellenarse; el registro D contiene el ancho del área en bytes y el registro E contiene la altura del área expresada en líneas de pantalla.

En todos los modos la pantalla tiene una altura de 200 líneas y un ancho de 80 bytes. Este hecho explica la presencia de los 16384 bytes de memoria de pantalla (80*200). En el modo 0, cada carácter tiene 4 bytes de ancho; en el modo 1, la anchura es de 2 bytes, y en el modo 2, cada carácter ocupa sólo el ancho de un byte. Con lo cual, para rellenar toda la pantalla con un color determinado, al cual suponemos almacenado en C, ejecutaremos una rutina en código máquina como la siguiente:

```
LD     HL,#C000 ; Extremo superior izquierdo de la pantalla.
LD     D,80
LD     E,200
CALL   #BC47
RET
```

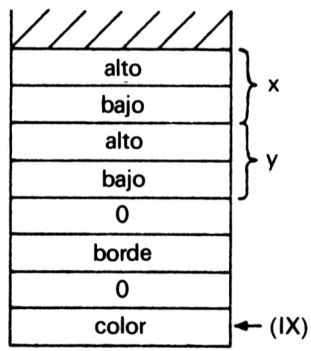
Sin embargo, todas estas rutinas son apenas inteligentes, ya que tenemos que especificar los bordes del área a rellenar, pudiendo llegar a ser muy tedioso este trabajo. La complejidad puede ser elevada si lo que tratamos de rellenar son figuras no rectangulares. Necesitamos, pues, de una rutina que rellene un contorno dibujado con un determinado color, con la tinta del color que especifiquemos. Cada línea de pantalla con una

figura puede ser entonces rellenada. Después de examinar el código máquina, veremos cómo puede emplearse para rellenar figuras tales como triángulos o círculos.

LINLLENA (LINea LLENA)

Esta rutina llena una simple línea de pantalla entre dos puntos que actúan como "borde". El color de la línea puede ser determinado por el usuario. El color de los dos puntos también es especificado por el usuario. Si deseas relocalizar la rutina, entonces es necesario alterar la dirección de "DERCHA". Los bytes que se especifican en el listado son para la dirección 40200.

Requisitos de entrada: Desde BASIC, usaremos CALL dirección,x,y, borde, color; donde x,y es la posición desde donde comienza el rellenado; borde es el color de la tinta que marca el límite del rellenado; color es la tinta con la que queremos que se trace la línea.
Partiendo desde código máquina, IX apunta hacia un bloque de parámetros como el de la figura, registro A=4.



Condiciones de salida: Si el número de parámetros es incorrecto, se producirá un retorno inmediato al BASIC. Todos los registros son alterados.

Longitud: 91 bytes.

```
                    10 ;      ** LINLLENA **
9D08                20        ORG   40200
9D08  FE04          30        CP    4
9D0A  C0            40        RET   NZ
9D0B  DD6E04        50        LD    L,(IX+4)
9D0E  DD6605        60        LD    H,(IX+5)
9D11  DD5E06        70        LD    E,(IX+6)
```

```
9D14  DD5607      80         LD   D,(IX+7)
9D17  E5          90         PUSH HL
9D18  D5         100         PUSH DE
9D19  E5         110 BUCLE   PUSH HL          ; Comienza a rastrear
                 120 ;                          hacia la derecha
9D1A  D5         130         PUSH DE
9D1B  CDF0BB     140         CALL #BBF0        ; Conoce la tinta del punto
9D1E  D1         150         POP  DE
9D1F  E1         160         POP  HL
9D20  DDBE02     170         CP   (IX+2)       ; Es el color del borde?
9D23  2808       180         JR   Z,SALD       ; Si asi es, SALD
9D25  3E03       190         LD   A,3          ; Si D=3, entonces estamos fuera
9D27  BA         200         CP   D            ; de la pantalla; saltamos a SALD
9D28  2803       210         JR   Z,SALD
9D2A  13         220         INC  DE           ; Siguiente punto
9D2B  18EC       230         JR   BUCLE
9D2D  1B         240 SALD    DEC  DE           ; Retrocede un punto
9D2E  ED53619D   250         LD   (DERCHA),DE; Guarda la posicion
9D32  D1         260         POP  DE
9D33  E1         270         POP  HL
9D34  E5         280 BUCLE2  PUSH HL           ; Ahora rastrea hacia la izquierda
9D35  D5         290         PUSH DE
9D36  CDF0BB     300         CALL #BBF0        ; Obtiene el color del punto
9D39  D1         310         POP  DE
9D3A  E1         320         POP  HL
9D3B  DDBE02     330         CP   (IX+2)       ; Es el borde?
9D3E  2807       340         JR   Z,SALI
9D40  1B         350         DEC  DE           ; Siguiente punto
9D41  7B         360         LD   A,E          ; Si DE=0, entonces estamos fuera
9D42  B2         370         OR   D            ; de la pantalla
9D43  2802       380         JR   Z,SALI
9D45  18ED       390         JR   BUCLE2
9D47  13         400 SALI    INC  DE           ; Siguiente punto
9D48  DD6E04     410         LD   L,(IX+4)     ; Coordenada Y en HL
9D4B  DD6605     420         LD   H,(IX+5)
9D4E  E5         430         PUSH HL           ; La guarda
9D4F  CDC0BB     440         CALL #BBC0
9D52  DD7E00     450         LD   A,(IX+0)     ; Extrae la tinta para el relleno
9D55  CDDEBB     460         CALL #BBDE        ; de la linea, y la activa como
                 470 ;                          color de los graficos
9D58  E1         480         POP  HL
9D59  ED5B619D   490         LD   DE,(DERCHA); Coordenada X en DE
9D5D  CDF6BB     500         CALL #BBF6        ; Traza la linea
9D60  C9         510         RET  ;            Terminado
9D61  0000       520 DERCHA  DEFW 00
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

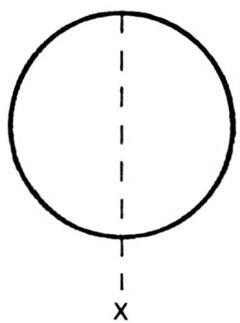
```
10 CLS:MOVE 100,0
20 DRAW 100,600
30 MOVE 200,0
40 DRAW 200,600
50 CALL 40200,150,200,1,1
```

Comentarios

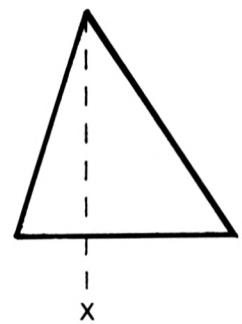
El método operativo de la rutina es muy sencillo, y se hace uso de las rutinas del *firmware* del Amstrad para permitir su utilización en todos los modos de pantalla. Primero busca hacia la derecha hasta que la coorde-

nada X sea mayor que 750 o hasta que se encuentre un punto con el color especificado como "borde". En este último caso, la posición X es almacenada en una variable de dos bytes llamada "DERCHA". La rutina devolverá la posición X a su valor inicial. La búsqueda comienza entonces hacia la izquierda, hasta que se encuentre un punto del borde o hasta que la coordenada X sea igual a 0. La línea se rellenará con el color requerido mediante una llamada a la rutina que dibuja líneas. El color que fue seleccionado para el relleno será ahora el color de la pluma de gráficos.

Por ahora ya podemos rellenar una línea de la pantalla. ¿Qué pasa con las figuras reales? Bien, eso es muy fácil. Elegimos una coordenada X que corresponda a la mayor coordenada Y, asociada a una figura determinada. Esto significa que la operación de rellenado de una figura compleja puede que se tenga que llevar a cabo en varios pasos. Para aclararlo todo, fíjate en los dos dibujos siguientes.



Círculo



Triángulo

El llenado se lleva a cabo con un bucle FOR...NEXT, que va variando la coordenada Y desde el valor más bajo, hasta su valor máximo. El siguiente programa en BASIC demuestra esta acción. Supongo que la rutina en código máquina ya está ensamblada en la dirección 40200:

```
10 MODE 1 :REM Fija el modo de pantalla
20 PLOT 0,0,1:REM Se localiza en 0,0
30 REM Pluma 1
40 DRAW 100,100:DRAW 200,0:DRAW 0,0
50 REM Dibuja un triangulo
60 FOR Y=1 TO 99:REM Bucle la coord. Y
70 CALL 40200,100,Y,1,2
80 NEXT
```

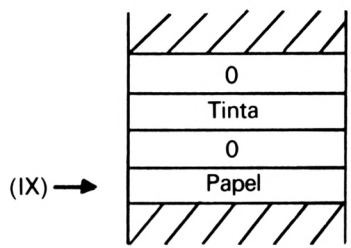
Esta rutina presenta algunos inconvenientes, pero aun así es bastante útil. El principal problema es la velocidad, por eso conviene usarla sólo

para rellenar áreas o figuras pequeñas. También puede usarse en combinación con otras rutinas para rellenar el grueso de una figura, dejando las áreas secundarias a la anterior rutina.

GPLUMA (PLUMA de Gráficos)

Esta sencilla rutina especifica el color a usar para el papel y la pluma de gráficos. Es relocalizable. El papel de gráficos elegido sólo entrará en acción tras la ejecución de la orden CLG.

Requisitos de entrada: Desde BASIC, usaremos CALL dirección, pluma, papel; siendo "pluma" el color de la tinta que debe usarse para la pluma de graficos, y "papel" es el color que se usará para el papel de gráficos. Partiendo del código máquina, IX apunta hacia un bloque de parámetros como el mostrado en la figura, A = 2.



Bloque de parámetros de GPLUMA

Condiciones de salida: Todos los registros son alterados.

Longitud: 16 bytes.

```
                10 ;       ** GPLUMA **
9D08            20         ORG  40200
9D08  FE02      30         CP   2          ; Si no hay 2 parametros,
9D0A  C0        40         RET  NZ         ; retorno
9D0B  DD7E00    50         LD   A,(IX+0)
9D0E  CDE4BB    60         CALL #BBE4      ; Cambia el papel de graficos
9D11  DD7E02    70         LD   A,(IX+2)
9D14  CDDEBB    80         CALL #BBDE      ; Cambia la tinta de graficos
9D17  C9        90         RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1 : PLOT 0,0,1
20 DRAW 100,100 : CALL 40200,2,3
30 DRAW 100,400
```

Comentarios

El color de la pluma de gráficos entra en acción inmediatamente y el papel lo hará después de la orden CLG.

Vamos a ver ahora cómo mover imágenes a través de la pantalla, utilizando rutinas en código máquina. El resto de este capítulo estará dedicado al movimiento de caracteres a través de la pantalla y en el modo de pantalla que tú elijas. Haremos uso de las rutinas residentes en el *firmware* del Amstrad, para escribir caracteres en pantalla, ya que todavía pueden dar buenos resultados. Después de ver brevemente la técnica empleada, presentaré un programa que mueve un carácter por pantalla. Esto provee al Amstrad de un sencillo "seudo-duendecillo" (SPRITE de otros sistemas) para movimientos con gráficos, de propósito general. Terminaré el capítulo con un estudio de los caracteres multicolores, y de cómo pueden ser desplazados por la pantalla.

Desplazando caracteres

Las principales operaciones que deben llevarse a cabo en los desplazamientos de caracteres a través de la pantalla, son las siguientes:

1. El carácter debe desaparecer en la pantalla de la antigua posición.
2. Las coordenadas X e Y del carácter deben ser actualizadas a la nueva posición.
3. El carácter debe escribirse en la nueva posición.

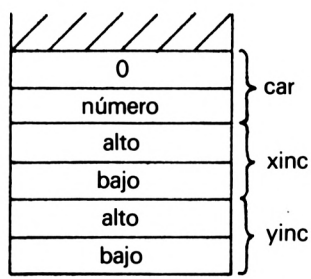
La suavidad del movimiento resultante depende de dos factores. El primero es la magnitud del incremento que damos a las coordenadas X e Y entre una posición y la posterior. El segundo factor es la frecuencia con que se altera la posición del carácter. Se pueden obtener movimientos delicados si el carácter se traslada uno o dos puntos de pantalla cada vez. En cambio, si la variación es de carácter a carácter, el movimiento parecerá brusco y sin continuidad. Igualmente, una rápida actualización de la posición del carácter nos ofrecerá suavidad en el desplazamiento. La tarea de eliminar el carácter de la posición anterior puede tener una solución sencilla si reescribimos un espacio en blanco en esa posición anterior. Sin embargo, esta acción puede causar problemas si el carácter se está desplazando sobre un fondo que contiene otra imagen. Al escribir un espacio en blanco, desaparecerán tanto el carácter como el fondo. Por este motivo, no utilizaremos esta técnica. Un segundo método es manejar el modo XOR para los gráficos de pantalla. Sin entrar en detalles, esto causaría la desaparición de una imagen de la pantalla si volvemos a escribir por segunda vez la misma imagen y en el mismo sitio. A pesar de todo se presentan algunos inconvenientes. Aunque este método deja intacto el

fondo de la pantalla, pueden producirse algunos cambios de color del fondo, mientras el carácter se mueva por esa zona. Sin embargo, probablemente éste sea el método más sencillo para conseguir que las cosas funcionen aceptablemente. Claro está que siempre puedes volver a dibujar el fondo después de cada desplazamiento, pero esto haría que las cosas marcharan demasiado lentas. Por ello vamos a ver un programa que desplaza caracteres de un solo color, utilizando el modo XOR.

DESPCAR (DESPlaza CARácter)

La rutina desplaza un determinado carácter definido por el usuario, o un carácter habitual, desde una posición de la pantalla a otra. El movimiento es instantáneo. En el interior del programa se encuentra una tabla de información denominada Tabla de Figuras, que contiene la información de cada carácter que quieras desplazar durante el programa. La rutina se puede relocalizar, cuidando que la dirección de "TEMP" y de la Tabla de Figuras sea también alterada.

Requisitos de entrada: Partiendo de un programa BASIC, usaremos CALL dirección, car, Xinc, Yinc; siendo "car" el número asignado al carácter dentro de la Tabla de Figuras. Este "car" se refiere al carácter que se quiere desplazar. Xinc es el incremento que se da a la coordenada X de la posición del carácter; Yinc es el incremento que deseamos darle a la coordenada Y de la posición del carácter. Estos incrementos pueden ser tanto positivos como negativos. El valor de "car" debe ser mayor que 0. Si accedemos desde código máquina, entonces A=3 y IX apuntando a un bloque de parámetros como el de la figura. También deberá estar presente, como pronto veremos, una Tabla de figuras.



Bloque de parámetros de DESPCAR

Condiciones de salida: Todos los registros son alterados. El color de la pluma de gráficos será el del carácter dibujado. El modo de gráficos quedará en "absoluto" o "forzado", y el cursor permanecerá en la última posición del carácter.

Longitud: 127 bytes sin incluir la Tabla de Figuras.

```
                    10 ;       ** DESPCAR **
A410                20         ORG  42000
A410  FE03          30         CP   3
A412  C0            40         RET  NZ
A413  DD4604        50         LD   B,(IX+4)
A416  DDE5          60         PUSH IX
A418  DD2191A4      70         LD   IX,TABLA ; Comienzo de la Tabla de Figuras
A41C  DD23          80 BUCLE   INC  IX       ;  Obtiene la definicion
A41E  DD23          90         INC  IX
A420  DD23         100         INC  IX
A422  DD23         110         INC  IX
A424  DD23         120         INC  IX
A426  DD23         130         INC  IX
A428  10F2         140         DJNZ BUCLE
A42A  DD228FA4     150         LD   (TEMP),IX; Almacena la definicion
A42E  DD5E00       160         LD   E,(IX)
A431  DD5601       170         LD   D,(IX+1) ; Obtiene la coordenada X
A434  DD6E02       180         LD   L,(IX+2) ; Obtiene la coordenada Y
A437  DD6603       190         LD   H,(IX+3)
A43A  E5           200         PUSH HL
A43B  D5           210         PUSH DE
A43C  CDC0BB       220         CALL #BBC0    ; Se mueve a la posicion
A43F  DD7E05       230         LD   A,(IX+5) ; Impone el color
A442  CDDEBB       240         CALL #BBDE
A445  3E01         250         LD   A,1      ; Activa el modo de graficos XOR
A447  CD59BC       260         CALL #BC59 ;
A44A  DD7E04       270         LD   A,(IX+4)
A44D  CDFCBB       280         CALL #BBFC    ; Imprime el caracter
A450  D1           290         POP  DE
A451  E1           300         POP  HL
A452  DDE1         310         POP  IX       ; Recupera IX
A454  E5           320         PUSH HL
A455  D5           330         PUSH DE
A456  E1           340         POP  HL
A457  DD4E02       350         LD   C,(IX+2) ; Suma el incremento a la
A45A  DD4603       360         LD   B,(IX+3) ; coordenada X
A45D  09           370         ADD  HL,BC
A45E  E5           380         PUSH HL
A45F  D1           390         POP  DE       ; La introduce en DE
A460  DDE5         400         PUSH IX
A462  DD2A8FA4     410         LD   IX,(TEMP)
A466  DD7300       420         LD   (IX),E   ; Lo almacena en el
                    430 ;                      registro temporal
A469  DD7201       440         LD   (IX+1),D
A46C  DDE1         450         POP  IX
A46E  DD4E00       460         LD   C,(IX+0) ; Suma el incremento a la
A471  DD4601       470         LD   B,(IX+1) ; coordenada Y...
A474  E1           480         POP  HL
A475  09           490         ADD  HL,BC
A476  DD2A8FA4     500         LD   IX,(TEMP)
A47A  DD7502       510         LD   (IX+2),L ; y la almacena en el
                    520 ;                      registro temporal
A47D  DD7403       530         LD   (IX+3),H
A480  CDC0BB       540         CALL #BBC0    ; Se mueve a la nueva posicion
A483  DD7E04       550         LD   A,(IX+4) ; Imprime el caracter
A486  CDFCBB       560         CALL #BBFC
A489  3E00         570         LD   A,0      ; Impone el modo 'absoluto'
A48B  CD59BC       580         CALL #BC59    ; para los graficos...
A48E  C9           590         RET  ;        y finaliza
```

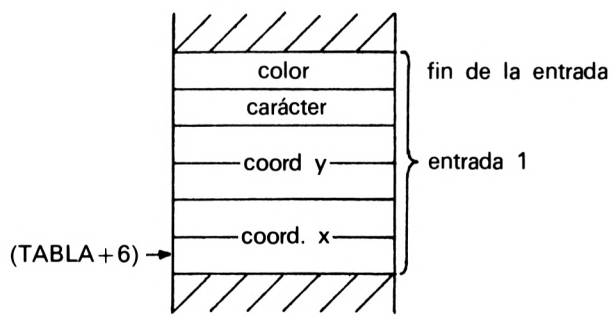
```
A48F  0000        600 TEMP   DEFW 00
A491  0000        610 TABLA  DEFW 00        ; Tabla de Figuras
A493  0000        620        DEFW 00        ; Esta es 'postiza'
A495  0000        630        DEFW 00
A497  6400        640        DEFW 100
A499  6400        650        DEFW 100
A49B  EA          660        DEFB 234
A49C  01          670        DEFB 1
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 J=1
30 MOVE 100,100
40 PRINT CHR$(23)+CHR$(1)
50 TAG : PRINT CHR$(234); : TAGOFF
60 FOR I=1 TO 300
70 CALL 42000,1,J,0 : NEXT
```

Comentarios

El programa está asistido por la presencia de la Tabla de Figuras, la cual contiene toda la información del carácter que el programador va a desplazar mediante esta rutina. Esta Tabla tine una entrada de 6 bytes para cada carácter, y la llamada con CALL a esta rutina especificará con el parámetro "car" la entrada de la Tabla de Figuras, que contiene los detalles del carácter a trasladar. Uno de estos datos de la entrada es el código ASCII del propio carácter. Una típica entrada de la Tabla es la del programa antes listado. En la siguiente figura se muestra un examen más detallado de lo que contiene una de las entradas de la Tabla:



Entrada de la Tabla de Figuras: DESPCAR

Antes de acceder a cualquiera de las rutinas en código máquina tendremos que preparar la entrada de cada carácter. La introducción de los valores apropiados en la Tabla de Figuras se efectuará con la orden

POKE en la dirección adecuada. Recuerda que los seis primeros bytes de la Tabla, que corresponden a la entrada 0, estarán en blanco. La Tabla contiene las posiciones X e Y iniciales del carácter, el código ASCII del carácter y el color con el que será escrito en pantalla. Los bytes que representan a X e Y serán actualizados cada vez que se cambie la posición del carácter. Esta forma de almacenar la información de los caracteres a trasladar, hace que esta rutina sea de la mayor utilidad general posible.

El programa utiliza el modo XOR para eliminar el carácter de su sitio y redibujarlo en otro. La última posición es adquirida añadiendo el incremento de X (Xinc) a la actual coordenada X y añadiendo, por otra parte, el incremento Y (Yinc) al valor actual de la coordenada Y. Esta posición actualizada se almacena entonces en la Tabla de Figuras. La rutina trabaja perfectamente durante todo el tiempo, excepto con la primera aparición del carácter sobre la pantalla, ya que justo en la primera posición ocupada por el carácter quedará una sombra del mismo. Si piensas un poco, descubrirás la respuesta a este problema.

Cuando comienza a ejecutarse la rutina, primero hace desaparecer el carácter de la última posición, mediante el modo XOR, y después actualiza la posición. Sin embargo, si no había nada que eliminar en aquella posición, en la pantalla quedará abandonada la imagen del carácter. Todas las restantes operaciones funcionarán perfectamente; pero no este primer detalle. Podemos solucionarlo si utilizamos una línea de BASIC como la de abajo, para posicionar inicialmente a cada carácter, justo en el lugar que le corresponde al principio.

```
PLOT 1000,1000,1:TAG:PRINT CHR$(23)+
CHR$(1);:MOVE inicioX,inicioY
PRINT CHR$(caracter);:TAGOFF
```

La sentencia PLOT 1000,1000,1 impone que el color de la pluma de gráficos sea 1, y CHR$(23) seguido de CHR$(1) activa el modo de gráficos XOR. Si tenías 100,100 en la Tabla de Figuras como punto de partida para un carácter dado, 1 como su color y 254 como código ASCII del carácter, entonces la línea en BASIC anterior tomará la forma:

```
PLOT 1000,1000,1:PRINT CHR$(23)+CHR$(1);:TAG:
MOVE 100,100:PRINT CHR$(254);:TAGOFF
```

El siguiente programa de demostración funcionará siempre que el código máquina de la rutina, incluyendo la Tabla de Figuras, se encuentre en la dirección 42000.

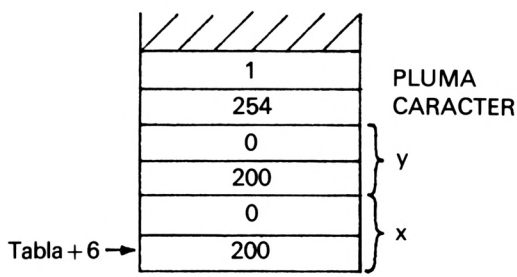
```
10 MODE 1 : PRINT CHR$(23)+CHR$(1);
20 PLOT 1000,1000,1:TAG
30 MOVE 100,100:PRINT CHR$(234);:TAGOFF"
40 Yinc=1 : Xinc=1
```

```
50 FOR i=1 TO 50:NEXT:REM retardo
60 CALL 42000,1,Xinc,Yinc
70 GOTO 50
```

Si deseas volver a ejecutar este programa, entonces tendrás que restaurar las coordenadas actuales X e Y de la Tabla a sus valores iniciales, ya que durante la ejecución del programa ambas coordenadas han sido modificadas. Para volver a inicializar la Tabla, utiliza la orden POKE.

Mientras utilizas esta rutina pueden suceder dos cosas: la primera es que, si intentaras mover el carácter demasiado rápido, éste parecerá que se bambolea, que tiembla. Esto se debe a la interacción entre la frecuencia del movimiento y la velocidad con que la imagen de la pantalla es renovada por el ordenador. La solución es sencilla; reduce la velocidad mediante un bucle de retardo en BASIC o código máquina, o bien mediante una orden CALL &BD19, que esperará hasta que se produzca un nuevo barrido de la pantalla, antes de proseguir con el programa. La segunda cuestión es que valores pequeños de Xinc y de Yinc ofrecerán un movimiento suave, pero también lento. Finalmente, una alteración del color en el que se dibuja el carácter, después de que éste haya comenzado su movimiento, puede provocar algunos efectos secundarios debido a la combinación del modo XOR con diferentes colores.

Antes de terminar con esta rutina, veamos más detenidamente la estructura de la Tabla de Figuras. Como ya dije, cada entrada tiene una longitud de 6 bytes; los 6 primeros bytes de la Tabla deben estar puestos a cero. Llamaremos entrada 0 a estos 6 primeros bytes; la segunda entrada será la entrada 1, y así sucesivamente. El parámetro "car" de la sentencia CALL es utilizado para indicar cuál de las entradas de la Tabla de Figuras es la que deseas usar. Supongamos que queremos que la entrada 1 corresponda al carácter 254 en el color 1 y con posición inicial en el punto 200,200 de la pantalla. La entrada comenzará en la dirección (TABLA+6), donde TABLA es la dirección inicial de la Tabla de Figuras dentro de la memoria. La entrada completa así determinada sería:



Por ello, cuando queramos poner este carácter en la pantalla por primera vez, usaremos una sentencia BASIC que escriba CHR$(254) en la posición 200,200 con la pluma 1 de gráficos.

En la Tabla de Figuras pueden introducirse numerosas entradas; puede escribirse una subrutina que se encargue de mover todos los caracteres a la vez. Por ejemplo, podríamos usar:

```
100 REM    Rutina de movimiento de
                caracteres
110 numero=6 :REM 6 caracteres definidos
    por el usuario.
120 FOR car=1 TO numero
130 CALL 42000,car,Xinc,Yinc
140 NEXT car:RETURN
```

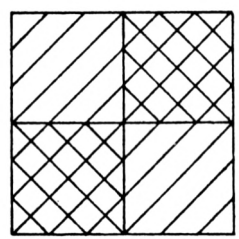
Utilizando esto en combinación con la sentencia EVERY del BASIC podemos conseguir que el movimiento de los caracteres a través de la pantalla se produzca a intervalos determinados. Los diferentes valores de Xinc y de Yinc son alterados mientras tanto en el núcleo principal del programa.

Los efectos de Xinc y de Yinc son inmediatos. Los valores positivos de Xinc provocan un desplazamiento hacia la derecha, mientras que los valores negativos de Xinc provocan un movimiento hacia la izquierda. Por otra parte los valores positivos de Yinc causan un desplazamiento hacia arriba por la pantalla, mientras que los Yinc negativos provocarán que el movimiento sea hacia abajo.

Veamos ahora una rutina similar, para trasladar caracteres de dos colores. Todos los principios que vamos a discutir son también aplicables a caracteres que tengan más de dos colores. El primer asunto a tratar aquí es cómo podemos dibujar en pantalla caracteres que tengan más de un color. Esto también puede serte útil en otras rutinas.

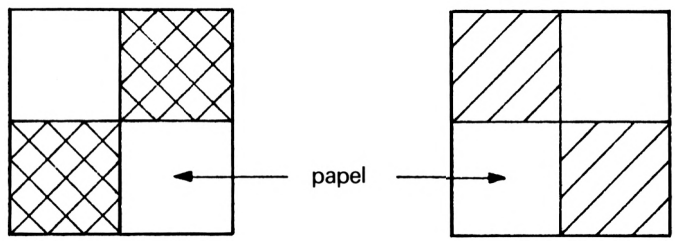
Caracteres multicolores

La solución se presenta en el uso del modo de gráficos XOR. Imagina que queremos construir un carácter de dos colores como el de la siguiente figura:



Típico carácter de dos colores

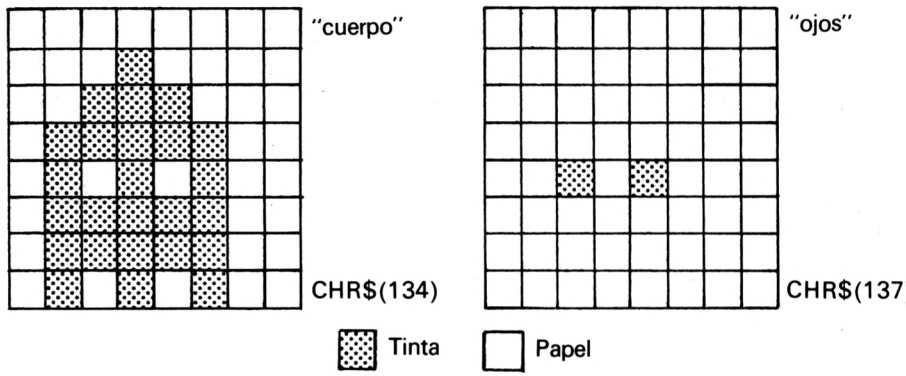
Dos de los cuadrados están a un color y los dos restantes están en otro color. Desde el BASIC un carácter así puede dibujarse en pantalla utilizando, como enseguida veremos, la orden TAG. Lo primero es definir un carácter de usuario para cada uno de los colores de fondo que posee la composición final. Así, para un carácter de dos colores tendremos que definir dos caracteres definibles por el usuario. Observa que no debe haber áreas solapadas. Si esto ocurre cuando estamos usando el modo XOR, entonces pueden aparecer algunos efectos secundarios de color. En el ejemplo del carácter arriba definido debemos definir dos caracteres como éstos:



Afortunadamente, éstos están disponibles entre las definiciones de caracteres del Amstrad, como CHR$(134) Y CHR$(137). Todo lo que hacemos ahora es utilizar TAG para escribir el carácter en las mismas coordenadas gráficas para ambos componentes. Con esto se superpondrán las dos imágenes. Si dibujamos cada uno de ellos de un color diferente, entonces obtendremos nuestro carácter bicolor. Esto es lo que efectúa el siguiente programa en BASIC.

```
10 MODE 1
20 GOSUB 100
30 FOR I=0 TO 300 : NEXT
40 GOSUB 100
50 GOTO 30
100 REM Subrutina que imprime el carac.
110 TAG
120 PLOT 1000,1000,1 : REM  color del
    primer caracter.
130 MOVE 100,100 : REM Se localiza.
140 PRINT CHR$(134);:REM Lo escribe.
150 TAGOFF:PRINT CHR$(23)+CHR$(1);:REM
    Fija el modo XOR de pantalla.
160 TAG
170 PLOT 1000,1000,2 : REM  color del
    segundo caracter.
180 MOVE 100,100 : REM Se localiza.
190 PRINT CHR$(137);:REM Lo escribe.
200 RETURN
```

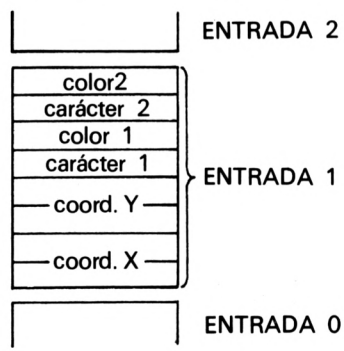
Ejecuta el programa y verás que la figura de dos colores se dibuja y elimina repetidamente. Tal vez te gustaría experimentar con otros caracteres. Si estás interesado en juegos, los siguientes dos caracteres pueden serte de interés.



El cuerpo de esta bestia puede ser dibujado de amarillo y los ojos en rojo. Recuerda otra vez que no debería haber solapamientos. Si los hay, pueden obtenerse algunos efectos interesantes con el color y tal vez estés interesado en conseguir algunos de estos efectos.

Veremos ahora un programa que nos permite desplazar caracteres de dos o más colores a través de la pantalla. Aunque la rutina venga preparada para caracteres de dos colores, los Comentarios que vienen a continuación te permitirán modificar el programa si así lo deseas.

DESPCAR2, que es como llamé al programa en un derroche de imaginación, es muy similar a DESPCAR, tal y como mostrará un estudio del listado. La diferencia comienza en la Tabla de Figuras, donde ahora hay un color y un carácter para cada una de las "máscaras" que componen el carácter completo. Así, una rutina bicolor tendrá dos entradas para color y carácter; una para cada una de las partes de la imagen global. Por ello, para un carácter de dos colores, la Tabla de Figuras presentará para cada una de las entradas una disposición así:



Entrada de la Tabla de Figuras para dos colores

Dentro del propio programa, la principal diferencia es la presencia de una segunda rutina para escribir un carácter tanto en la etapa de eliminación como en la de redibujar el carácter, que se usa con el segundo carácter coloreado. Obviamente, para un carácter con un tercer color, tendrá que haber una tercera operación de eliminado y redibujado que se encargue de este tercer carácter. Además, una rutina que desplace caracteres de tres colores necesitará dos bytes extra en la entrada de la Tabla de Figuras para definir este tercer carácter y su color.

DESPCAR2 (DESPlaza CARacteres a 2 colores)

Esta rutina es utilizada para trasladar por pantalla caracteres bicolores en cualquiera de los modos de pantalla. Consulta los comentarios posteriores para los detalles de manejo.

Requisitos de entrada: Desde BASIC usaremos CALL dirección, car, Xinc, Yinc; donde car, Xinc y Yinc tienen el mismo significado que en DESPCAR.
Partiendo de otras rutinas en código máquina, A=3 y IX contiene la dirección de un bloque de parámetros con estructura idéntica a la utilizada para DESPCAR.

Condiciones de salida: Todos los registros son alterados. El color de la pluma de gráficos será el último que se utilizó para dibujar el carácter. El modo de gráficos será el "absoluto" o "forzado" y el cursor de gráficos se encontrará en la posición a donde fue trasladado el carácter.

Longitud: 179 bytes, sin incluir la Tabla de Figuras.

```
                10 ;       ** DESPCAR2 **
A410            20         ORG  42000
A410  FE03      30         CP   3
A412  C0        40         RET  NZ
A413  DD4604    50         LD   B,(IX+4)
A416  DDE5      60         PUSH IX
A418  DD21C1A4  70         LD   IX,TABLA ; Comienzo de la Tabla de Figuras
A41C  DD23      80 BUCLE   INC  IX        ; Obtiene la definicion
A41E  DD23      90         INC  IX
A420  DD23     100         INC  IX
A422  DD23     110         INC  IX
A424  DD23     120         INC  IX
A426  DD23     130         INC  IX
A428  DD23     140         INC  IX
A42A  DD23     150         INC  IX
A42C  10EE     160         DJNZ BUCLE
A42E  DD22BFA4 170         LD   (TEMP),IX; Almacena la definicion
A432  DD5E00   180         LD   E,(IX)
A435  DD5601   190         LD   D,(IX+1) ; Obtiene la coordenada X
A438  DD6E02   200         LD   L,(IX+2) ; Obtiene la coordenada Y
A43B  DD6603   210         LD   H,(IX+3)
A43E  E5       220         PUSH HL
```

```
A43F  D5        230        PUSH DE
A440  CDC0BB    240        CALL #BBC0     ; Se mueve a la posicion
A443  DD7E05    250        LD   A,(IX+5)  ; Impone el primer color
A446  CDDEBB    260        CALL #BBDE
A449  3E01      270        LD   A,1       ; Activa el modo de graficos XOR
A44B  CD59BC    280        CALL #BC59 ;
A44E  DD7E04    290        LD   A,(IX+4)
A451  CDFCBB    300        CALL #BBFC     ; Imprime el primer caracter
A454  DD7E07    310        LD   A,(IX+7)
A457  CDDEBB    320        CALL #BBDE     ; Obtiene el segundo color
A45A  D1        330        POP  DE
A45B  E1        340        POP  HL
A45C  E5        350        PUSH HL
A45D  D5        360        PUSH DE
A45E  CDC0BB    370        CALL #BBC0     ; Se posiciona otra vez
A461  DD7E06    380        LD   A,(IX+6)
A464  CDFCBB    390        CALL #BBFC     ; Imprime el segundo caracter
A467  D1        400        POP  DE
A468  E1        410        POP  HL
A469  DDE1      420        POP  IX        ; Recupera IX
A46B  E5        430        PUSH HL
A46C  D5        440        PUSH DE
A46D  E1        450        POP  HL
A46E  DD4E02    460        LD   C,(IX+2)  ; Suma el incremento a la
A471  DD4603    470        LD   B,(IX+3)  ; coordenada X
A474  09        480        ADD  HL,BC
A475  E5        490        PUSH HL
A476  D1        500        POP  DE        ; La introduce en DE
A477  DDE5      510        PUSH IX
A479  DD2ABFA4  520        LD   IX,(TEMP)
A47D  DD7300    530        LD   (IX),E    ; Lo almacena en el
              540 ;                         registro temporal
A480  DD7201    550        LD   (IX+1),D
A483  DDE1      560        POP  IX
A485  DD4E00    570        LD   C,(IX+0)  ; Suma el incremento a la
A488  DD4601    580        LD   B,(IX+1)  ; coordenada Y...
A48B  E1        590        POP  HL
A48C  09        600        ADD  HL,BC
A48D  DD2ABFA4  610        LD   IX,(TEMP)
A491  DD7502    620        LD   (IX+2),L  ; y la almacena en el
              630 ;                         registro temporal
A494  DD7403    640        LD   (IX+3),H
A497  E5        650        PUSH HL
A498  D5        660        PUSH DE
A499  CDC0BB    670        CALL #BBC0     ; Se mueve a la nueva posicion
A49C  DD7E05    680        LD   A,(IX+5)  ; Primer color
A49F  CDDEBB    690        CALL #BBDE
A4A2  DD7E04    700        LD   A,(IX+4)  ; Imprime el primer caracter
A4A5  CDFCBB    710        CALL #BBFC
A4A8  D1        720        POP  DE
A4A9  E1        730        POP  HL
A4AA  CDC0BB    740        CALL #BBC0
A4AD  DD7E07    750        LD   A,(IX+7)  ; Segundo color
A4B0  CDDEBB    760        CALL #BBDE
A4B3  DD7E06    770        LD   A,(IX+6)  ; Imprime el segundo caracter
A4B6  CDFCBB    780        CALL #BBFC
A4B9  3E00      790        LD   A,0       ; Impone el modo 'absoluto'
A4BB  CD59BC    800        CALL #BC59     ; para los graficos...
A4BE  C9        810        RET  ;           y finaliza
A4BF  0000      820 TEMP   DEFW 00
A4C1  0000      830 TABLA  DEFW 00        ; Tabla de Figuras
A4C3  0000      840        DEFW 00        ; Esta es 'postiza'
A4C5  0000      850        DEFW 00        ; (Se puede cambiar)
A4C7  0000      860        DEFW 00
A4C9  6400      870        DEFW 100
A4CB  6400      880        DEFW 100
A4CD  86        890        DEFB 134       ; Primer caracter
A4CE  01        900        DEFB 1         ; Color del primer caracter
A4CF  89        910        DEFB 137       ; Segundo caracter
A4D0  02        920        DEFB 2         ; Color del segundo caracter
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 MODE 1
20 CLS
30 J=1
40 PLOT 1000,1000,1 : MOVE 100,100
50 PRINT CHR$(23)+CHR$(1);
60 TAG : PRINT CHR$(134);
70 PLOT 100,100,2 : PRINT CHR$(137);
80 TAGOFF
90 FOR I=1 TO 300 : CALL 42000,1,J,0
100 CALL &BD19
110 NEXT
```

Comentarios

Con cuidado, la rutina puede ser relocalizable. Será necesario alterar la dirección dada por los bytes del programa, los cuales están preparados para funcionar en la dirección 42000. Para la Tabla de Figuras y la variable TEMP ocurre lo mismo. Al igual que con DESPCAR, todo carácter tiene que ser colocado en la posición de comienzo para su primera aparición en pantalla. Mediante el uso de TAG desde el BASIC, podemos evitar que aparezca abandonada la "sombra" del carácter desde el momento en que éste varía su posición respecto a la inicial. También necesitaremos cargar de información la Tabla de Figuras para cada entrada, mediante la orden POKE. El siguiente programa BASIC también demuestra el funcionamiento del código máquina de DESPCAR2. Supongo que el código máquina ya está almacenado en la dirección 42000 y que la misma Tabla de Figuras dada en el listado de la rutina ya ha sido incluida. Advierte que, al igual que en DESPCAR, la primera entrada de la Tabla de Figuras, entrada 0, tiene todos sus bytes puestos a cero.

```
10 MODE 1
20 Yinc=1
30 Xinc=1
40 PLOT 1000,1000,1 : REM  color del
   primer caracter.
50 PRINT CHR$(23)+CHR$(1);:REM Fija el
   modo XOR de pantalla.
60 MOVE 100,100 : REM Se localiza.
70 TAG:PRINT CHR$(134);:REM escribe el
   primer caracter.
80 PLOT 1000,1000,2 : REM  color del
   segundo caracter.
90 MOVE 100,100 : REM Se localiza.
```

```
100 PRINT CHR$(137);:REM escribe el
    segundo caracter.
110 TAGOFF
120 FOR I=0 TO 300
130 CALL 42000,1,Xinc,Yinc
140 FOR J=0 TO 20
150 NEXT J : REM Retardo.
160 NEXT I
170 END
```

Si deseas volver a ejecutar esta rutina, tendrás que introducir otra vez los valores iniciales en la Tabla de Figuras. De hecho, se puede escribir un sencillo programa en código máquina que tome como parámetros los diferentes trozos de información necesarios para la Tabla de Figuras y los almacene en el lugar correcto. Te dejo a ti esta tarea de escritura.

El bucle de retardo de las líneas 140-150 del anterior programa puede requerir una alteración dependiendo del resto del programa y del tiempo que le lleve desplazar un número variable de caracteres. Cuantas más rutinas de escritura de caracteres haya, el tiempo que lleva desplazar caracteres multicolores será mayor que el trabajo de mover caracteres de un único color.

Fijémonos en la siguiente subrutina, que es un ejemplo de subrutina en BASIC para actualizar una entrada de la Tabla de Figuras del programa DESPCAR2. Se supone que la variable "tabla" contiene la dirección del primer byte de la Tabla de Figuras.

```
1000 entrada=tabla+(n*8)
1010 REM n=numero de entrada
1020 POKE entrada,100:REM coordenada X
1030 POKE (entrada+1),0
1040 POKE (entrada+2),100 : REM
     coordenada Y
1050 POKE (entrada+3),0
1060 POKE (entrada+4),134 : REM primer
     caracter
1070 POKE (entrada+5),1 : REM  primer
     color
1080 POKE (entrada+6),137 : REM segundo
     caracter
1090 POKE (entrada+7),2 : REM  segundo
     color
```

Tanto en DESPCAR como en DESPCAR2, sólo es necesario escribir el carácter en la posición inicial cuando éste vaya a aparecer por primera vez en la pantalla. Después de esto, para trasladar el carácter a una

posición de pantalla deseada, simplemente tendremos que ajustar los valores de Xinc y de Yinc.

Con esto se completa este capítulo de rutinas de manipulación de gráficos. Continuaremos ahora viendo las rutinas de código máquina que interaccionan con el teclado. Se incluyen rutinas de utilidad para la entrada de cadenas de caracteres, para el manejo de juegos y otras aplicaciones.

6
Operaciones de teclado

En el "Locomotive BASIC" contamos, afortunadamente, con una muy sofisticada gama de órdenes que nos permiten realizar cosas como determinar el intervalo de repetición de las teclas pulsadas, cambiar el carácter asignado a las teclas, etc. Además, tenemos la orden KEY que nos permite ligar cadenas de caracteres a teclas determinadas. Y, cómo no, contamos con órdenes tan útiles como INPUT, INKEY e INKEY$. Sin embargo, para rutinas en código máquina, lo que nos interesa normalmente es saber si una tecla ha sido pulsada; esto es extremadamente fácil desde nuestros propios programas, ya que disponemos de las excelentes facilidades del *firmware*. En este capítulo te proporcionaré una variedad de rutinas que podrás usar en programas tanto en código máquina como en BASIC. Así que sin más rodeos vamos allá.

NORESET

Esta rutina altera el comportamiento del ordenador frente a la secuencia de teclas SHIFT-CTRL-ESC. En vez de inicializar el ordenador totalmente, se ignora por completo durante la ejecución del programa y sólo genera el mensaje *break* volviendo al modo interactivo. Análogamente, ESC queda completamente inhibido en la ejecución de un programa. Incluso proporciona un mayor grado de protección en la ejecución de un programa. La rutina es relocalizable.

Requisitos de entrada: Desde BASIC, CALL dirección,n, donde n=0 inhibe la inicialización del aparato y n=1 provoca que SHIFT-CTRL-ESC generen una inicialización completa.
Desde código máquina, IX apunta a un solo byte que contiene "n" y A=1.

Condiciones de salida: AF alterado.

Longitud: 22 bytes.

```
                    10 ;        ** NORESET **
9D08                20          ORG  40200
9D08  FE01          30          CP   1
9D0A  C0            40          RET  NZ
9D0B  DD7E00        50          LD   A,(IX)
9D0E  FE00          60          CP   0
9D10  2806          70          JR   Z,INHIBE
9D12  3EC3          80          LD   A,195
9D14  32EEBD        90          LD   (#BDEE),A
9D17  C9           100          RET
9D18  3EC9         110 INHIBE   LD   A,201
9D1A  32EEBD       120          LD   (#BDEE),A
9D1D  C9           130          RET
```

Pruébalo con esto:

```
CALL 40200,1
FOR J=1 TO 3000:NEXT
CALL 40200,0
```

Comentarios

Esta rutina, como ya he dicho, inhibe totalmente a ESC durante la ejecución de un programa, por lo que si realizas una orden

```
CALL 40200,0
```

y luego pasas a un bucle continuo... ¡lo llevas claro! Esta línea sólo se ejecutaría si tuvieras un programa trabajando. Funciona alterando el primer byte de una de las entradas del bloque de saltos para interpretar el código como una instrucción RET. Cuando se accede al bloque de saltos después de introducir cualquiera de las secuencias de teclas anteriores, se realiza un regreso inmediato. Para restablecer el comportamiento normal, el byte antiguo, que es 195, se repone como primer byte en la entrada del bloque de saltos.

GET

La mayoría de los ordenadores cuentan con una función llamada GET, cuyo papel es conseguir que la ejecución del programa cese hasta que se

pulse una tecla. La función devuelve como resultado el código ASCII de la tecla que ha sido pulsada. Así

```
G=GET
```

devolverá el código ASCII en la variable G. El BASIC del Amstrad no dispone de esta función, y normalmente podemos simularla utilizando un par de líneas de BASIC tales como:

```
10 G$=INKEY$:IF G$="" THEN GOTO 10
20 G=ASC(G$)
30 RETURN
```

He aquí una rutina en código máquina que realiza la función GET sin necesidad de las líneas de BASIC anteriores.

Requisitos de entrada: Desde BASIC CALLdirección,@G%, donde G% es una variable previamente definida en donde se introducirá el código ASCII de la tecla pulsada. Desde código máquina, es mejor llamar directamente a la rutina del *firmware.*

Condiciones de salida: AF, HL son alterados.

Longitud: 18 bytes.

```
                10 ;      ** GET **
9D08            20        ORG  40200
9D08  FE01      30        CP   1
9D0A  C0        40        RET  NZ        ; Retorno en caso de haber un numero
                50 ;                       erroneo de parametros
9D0B  DD6E00    60        LD   L,(IX)
9D0E  DD6601    70        LD   H,(IX+1) ; Obtiene la direccion de CAR%
9D11  CD18BB    80        CALL #BB18
9D14  77        90        LD   (HL),A
9D15  AF       100        XOR  A         ; Pone el registro A a cero
9D16  23       110        INC  HL
9D17  3600     120        LD   (HL),0
9D19  C9       130        RET
```

Pruébalo con esto:

```
10 car%=0
20 CALL 40200,@car%
30 PRINT car%
40 GOTO 20
```

Comentarios

La variable usada para contener el código ASCII devuelto debe haber sido inicializada previamente de algún modo, como se hace normalmente con las variables prefijadas con@, incluso si le has dado el valor cero. Con respecto a la rutina del *firmware*, el código ASCII se devuelve en el registro A con el *flag* C puesto a 1.

El programador en código máquina dispone de otras rutinas de teclado, pero no requieren inicialización, por lo que puedes llamarlas directamente desde tus rutinas en código máquina. No ofrecen nuevas ventajas al programador de BASIC.

LECTURA DE TECLAS

Esta rutina, accesible llamando a &BB1B, devuelve un código si está pulsada una tecla en el mismo instante en que se llama a la rutina. No esperará a que se pulse una tecla. Es por esto más útil en programas de juegos donde no se requieren pausas. Si una tecla estaba pulsada, al volver de la rutina el indicador C estará puesto a 1 y A contendrá el código de la tecla. De otro modo C estará puesto a 0.

CHEQUEO DE TECLA

Esto tiene algún parecido con la función BASIC INKEY(n), ya que examina si una cierta tecla fue pulsada en el mismo instante en que la rutina fue llamada. A la salida, Z=0 si la tecla se pulsó, y Z=1 si la tecla en cuestión no se pulsó. La rutina está localizada en la dirección &BB1E, conteniendo el registro A el número de la tecla a examinar.

Puede ser útil a veces, al programar, para obtener información sobre la situación actual de las teclas SHIFT, CTRL, CAPS LOCK y SHIFT LOCK. Esto te permite detectar secuencias de teclas superfluas, como SHIFT-CTRL-ENTER, si fueses propenso a ello. La rutina se llama ESTADO.

ESTADO

Devuelve el estado actual de las teclas Shift, CTRL, Shift Lock y Caps Lock. La rutina es relocalizable.

Requisitos de entrada: Desde BASIC, CALL dirección,@estado%.
Desde código máquina, IX apunta a un bloque de parámetros con A=1. El bloque de parámetros es un bloque de 2 bytes que contiene (prime-

ro el byte bajo) la dirección de la posición donde quieras almacenar el byte de estado.

Condiciones de salida: Todos los registros son alterados.

Longitud: 31 bytes.

```
                    10 ;      ** ESTADO **
9D08                20        ORG  40200
9D08  FE01          30        CP   1
9D0A  C0            40        RET  NZ
9D0B  DD6E00        50        LD   L,(IX+0)
9D0E  DD6601        60        LD   H,(IX+1)
9D11  E5            70        PUSH HL
9D12  CD1EBB        80        CALL #BB1E ; Obtiene el estado de SHIFT/CTRL
9D15  C5            90        PUSH BC    ; Guarda el registro BC
9D16  CD21BB       100        CALL #BB21 ; Obtiene el estado de CAPS/LOCK
9D19  7D           110        LD   A,L
9D1A  E680         120        AND  128   ; Compara el estado de  CAPS/LOCK
9D1C  6F           130        LD   L,A   ; con 0 y con 128,
                   140 ;                   guardando el resultado
9D1D  7C           150        LD   A,H   ; Compara el estado de SHIFT/CTRL
9D1E  E601         160        AND  1     ; con 0 y con 1,
                   170 ;                   sumando el resultado a L
9D20  85           180        ADD  A, L  ; para obtener el valor
                   190 ;                   de la comparacion
9D21  C1           200        POP  BC    ; Recupera BC
9D22  E1           210        POP  HL
9D23  71           220        LD   (HL),C; Guarda los estados obtenidos
9D24  23           230        INC  HL    ; en la variable ESTADO%
9D25  77           240        LD   (HL),A
9D26  C9           250        RET
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 estado%=0
20 CALL 40200,@estado%
30 PRINT HEX$(estado%)
40 INPUT A$
50 GOTO 20
```

Comentarios

El valor almacenado en la posición para el byte de estado o el valor devuelto al BASIC en la variable estado%, deberá ser decodificado para conocer el estado de las diferentes teclas. Es mejor considerarlo bajo un formato hexadecimal de 4 dígitos, que puedes obtener en BASIC usando

```
estado%=0:CALL 40200,@estado%
estado$=HEX$(estado%):PRINT estado$
```

El estado puede ser interpretado mediante la siguiente tabla:

	CAPS LOCK	SHIFT LOCK	SHIFT	CTRL
ON	&0100	&8000	&0020	&0080
OFF	&0000	&0000	&0000	&0000

Un par de ejemplos para aclarar el uso de esta tabla. Si la tecla SHIFT LOCK estuviera pulsada y CTRL también, al mismo tiempo, el valor del estado sería:

```
&8000+&0080
=&8080
```

Análogamente, si un valor de &A0 fuera devuelto, un examen de la tabla revelaría que para obtener este valor a partir de los datos en la tabla, se requeriría la pulsación simultánea de las teclas SHIFT y CTRL (&20+&80). La rutina devuelve el estado de LOCK que estaba presente desde la última vez en que el ordenador estuvo esperando algún tipo de entrada.

Probablemente habrás advertido programando tu Amstrad que las teclas pulsadas durante largos bucles de programas son almacenadas en el *buffer* del teclado y aparecen con la siguiente sentencia INPUT o INKEY una vez finalizado el bucle. Compruébalo con esto. Una vez que hayas pulsado "ENTER" pulsa una secuencia cualquiera de las teclas y fíjate cómo aparecen en el momento en que el programa llega a INPUT.

```
FOR I=0 TO 3000:NEXT:INPUT a$
```

Esto puede dar lugar a confusiones si no lo tienes en cuenta, y si el programa está esperando a que pulses una tecla antes de continuar, estas pulsaciones de tecla almacenadas pueden provocar el que el programa continúe sin esperar más. Muchos aparatos tienen un método mediante el cual tales caracteres pueden ser retirados de la memoria del teclado; este proceso se llama "despejar el *buffer* del teclado". Retirará todos los códigos de teclas almacenadas en ese momento, y así, si se usa inmediatamente antes de una orden del tipo "GET", asegura que el ordenador no fracase en su tarea por descuido. La rutina siguiente realiza este trabajo y se llama LIMPBUF.

LIMPBUF (LIMPia el *BUFfer*)

Despeja la memoria del teclado.

Requisitos de entrada: CALL dirección, tanto desde BASIC como desde código máquina.

Condiciones de salida: AF alterado.

Longitud: 6 bytes.

```
                  10 ;       ** LIMPBUF **
9D08              20         ORG  40200
9D08  CD09BB      30 BUCLE   CALL #BB09    ; Obtiene un caracter del registro
                  40 ;                     de memoria para el teclado
9D0B  38FB        50         JR   C,BUCLE ; Si C=1 , mas caracteres?
9D0D  C9          60         RET
```

Comentarios

Para ver la rutina en acción prueba lo siguiente. La rutina es relocalizable, pero para este caso supongo que los bytes están en la dirección 402000.

```
10 FOR I=0 TO 3000:NEXT I
30 INPUT a$
```

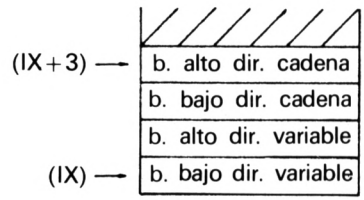
Ejecutando esto y pulsando algunas teclas mientras el ordenador está en la línea 10, introducirá algunos códigos de teclas en la memoria, que serán mostrados cuando la sentencia INPUT se ejecute. Repite la ejecución ahora, pero con un CALL 40200 en la línea 20. Las teclas extra pulsadas ahora serán borradas y los únicos caracteres que aparecerán en la sentencia INPUT serán aquellos introducidos una vez ejecutada la línea 20.

ESPERAT (ESPERA Tecla)

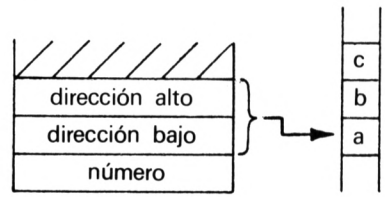
Esta rutina acepta como parámetros una cadena de caracteres y una variable precedida de @. La rutina espera hasta que el programa detecte la pulsación de una de las teclas cuyo carácter se haya especificado en la cadena de parámetros. La variable con @ devuelve la posición dentro de la cadena de parámetros del carácter que ha sido pulsado. La rutina es relocalizable.

Requisitos de entrada: Desde BASIC, CALL dirección,@a$,@p%, donde a$ ha sido previamente inicialiada con los caracteres que la rutina debe esperar, p% debe inicializarse a 0 antes de hacer la llamada.
Desde código máquina, las cosas son algo más complicadas. IX apunta a un bloque de paráme-

tros y A=2. En el bloque de parámetros, "cadena" es la dirección de un bloque descriptor como el mostrado más abajo, "dir. variable" es la dirección de la posición de un solo byte que recogerá, una vez ejecutado el código, la posición de la tecla dentro del bloque de datos de los códigos ASCII aceptables a los que se apunta en el bloque descriptor; "número" es el número de códigos ASCII almacenados en la tabla de códigos ASCII posibles apuntados por "dirección".



Bloque de parámetros de ESPERAT



Bloque descriptor de ESPERAT

Condiciones de salida: Todos los registros están alterados. Se produce una salida inmediata si se introduce un número equivocado de parámetros o si A no es igual a 2.

Longitud: 63 bytes.

```
                    10 ;       ** ESPERAT **
9D08                20         ORG   40200
9D08  FE02          30         CP    2
9D0A  C0            40         RET   NZ
9D0B  DD6E00        50         LD    L,(IX)
9D0E  DD6601        60         LD    H,(IX+1)
9D11  E5            70         PUSH HL           ; Guarda en la pila la direccion
                    80 ;                           de las variables
9D12  DD6E02        90         LD    L,(IX+2)    ; Extrae la direccion del bloque
9D15  DD6603       100         LD    H,(IX+3)    ; descriptor de la cadena
9D18  7E           110         LD    A,(HL)      ; Numero de caracteres de la caden
9D19  23           120         INC   HL          ; Introduce la direccion de la
9D1A  4E           130         LD    C,(HL)      ; cadena, en el
9D1B  23           140         INC   HL          ; registro BC
9D1C  46           150         LD    B,(HL)
9D1D  C5           160         PUSH BC
9D1E  E1           170         POP   HL          ; Tambien lo hace en HL
```

```
9D1F  32449D     180         LD   (TEMP),A    ; Almacena la longitud de
                 190 ;                          la cadena buscada
9D22  22459D     200         LD   (TEMP2),HL ; Almacena la direccion
                 210 ;                          de la cadena
9D25  3A449D     220 BUCLE2  LD   A,(TEMP)    ; Introduce la longitud en B
9D28  47         230         LD   B,A
9D29  2A459D     240         LD   HL,(TEMP2) ; Recupera la direccion
9D2C  C5         250         PUSH BC
9D2D  E5         260         PUSH HL
9D2E  CD18BB     270         CALL #BB18      ; Espera algun caracter
9D31  E1         280         POP  HL
9D32  C1         290         POP  BC
9D33  1E01       300         LD   E,1        ; Inicializa un contador
9D35  BE         310 BUCLE   CP   (HL)       ; Compara el caracter con
9D36  2806       320         JR   Z,VISTO    ; la tabla; salta si lo encuentra
9D38  1C         330         INC  E          ; Actualiza el contador
9D39  23         340         INC  HL         ; Mira todos los caracteres
9D3A  10F9       350         DJNZ BUCLE      ; hasta que termine la tabla
9D3C  18E7       360         JR   BUCLE2     ; Si no coincide ninguno,
                 370 ;                          vuelve a empezar otra vez
9D3E  E1         380 VISTO   POP  HL         ; Recupera la direccion
                 390 ;                          de la variable
9D3F  73         400         LD   (HL),E     ; Pone el valor en ella
9D40  23         410         INC  HL
9D41  3600       420         LD   (HL),0
9D43  C9         430         RET  ;            Vuelve al BASIC
9D44  00         440 TEMP    DEFB 0
9D45  0000       450 TEMP2   DEFW 00
```

Pruébalo con esto.

```
10 A$="abcdef" : p%=0
20 CALL 40200,@A$,@p%:PRINT p%;:GOTO 20
```

Comentarios

La rutina es muy útil para múltiples aplicaciones. Por ejemplo, examinando una contestación del usuario a una pregunta que requiere tan sólo un sí o un no, como respuesta. Las únicas teclas de respuesta con algún interés serán "S", "N", "s" y "n". Así, desde BASIC podemos limitarnos a ejecutar la línea

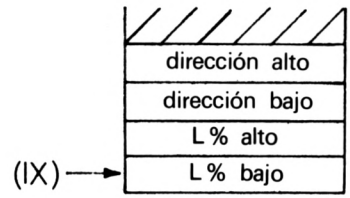
```
A$="SsNn":P%=0:CALL 40200,@A$,@P%
```

La rutina sólo volverá al BASIC cuando una respuesta conveniente haya sido recibida desde el teclado, en este caso "S", "N", "s" o "n". p% contendrá entonces un valor que corresponde a la tecla pulsada. Así, si "S" ha sido pulsada, p%=1. O bien, si "n" ha sido pulsada, p%=4.

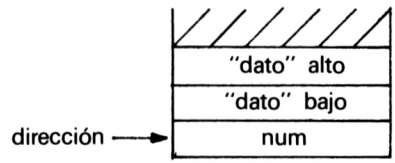
La siguiente rutina que veremos es una simple rutina "de introducción" que permite al usuario teclear una cadena de caracteres terminados por la tecla ENTER. La rutina se llama INTROCAD.

INTROCAD (INTROduce CADena)

Acepta una cadena de caracteres desde el teclado y los almacena bien en una variable literal, bien en un bloque de memoria.



Bloque de parámetros de INTROCAD



Bloque descriptor de INTROCAD

Requisitos de entrada: Desde BASIC, CALL dirección,@A$,L%, donde L% es el máximo número de caracteres que el usuario puede teclear. A$ es la variable literal que ha sido previamente inicializada para ser más larga que L%.
Desde código máquina, IX apunta a un bloque de parámetros, "dirección" es la dirección de un bloque descriptor. En el bloque descriptor, num es el número de bytes que han sido colocados aparte para la cadena introducida. Deberá ser mayor que L%. "dato" es la dirección en la memoria donde la cadena introducida se va a situar.

Condiciones de salida: Todos los registros son alterados. La rutina termina cuando:

1. Se pulsa la tecla ENTER.
2. El usuario teclea un número de caracteres mayor que L%. La cadena devuelta en este caso serán los L% primeros caracteres.

En ambos casos, se producirá un pequeño pitido. Los caracteres introducidos pueden ser entonces accesibles bien en una variable literal, bien en el área "dato" de la memoria.

Longitud: 137 bytes.

```
                    10 ;      ** INTROCAD **
A028                20        ORG   41000
A028  FE02          30        CP    2
A02A  C0            40        RET   NZ
A02B  DD7E00        50        LD    A,(IX)
A02E  DD6E02        60        LD    L,(IX+2)
A031  DD6603        70        LD    H,(IX+3)
A034  BE            80        CP    (HL)
```

```
A035  7E          90        LD   A,(HL)
A036  32B0A0      100       LD   (TEMP),A
A039  3801        110       JR   C,BIEN      ; Unicamente continuaremos
                  120 ;                         si la longitud de la cadena
                  130 ;                         es mayor que la especificada
                  140 ;                         por la sentencia CALL
A03B  C9          150       RET
A03C  23          160 BIEN  INC  HL
A03D  4E          170       LD   C,(HL)
A03E  23          180       INC  HL
A03F  46          190       LD   B,(HL)
A040  C5          200       PUSH BC          ; Obtiene la direccion de
A041  DD7E00      210       LD   A,(IX)      ; la longitud propuesta
A044  47          220       LD   B,A         ; para la cadena
A045  DDE1        230       POP  IX
A047  DD22AEA0    240       LD   (TEMPIX),IX
A04B  DDE5        250       PUSH IX
A04D  C5          260       PUSH BC
A04E  3AB0A0      270       LD   A,(TEMP)
A051  47          280       LD   B,A
A052  DD360020    290 LIMPIA LD  (IX),32
A056  DD23        300       INC  IX
A058  10F8        310       DJNZ LIMPIA      ; Quita los espacios de la cadena
A05A  C1          320       POP  BC
A05B  DDE1        330       POP  IX
A05D  C5          340 BUCLE PUSH BC          ; Espera un caracter
A05E  CD18BB      350       CALL #BB18
A061  FE7F        360       CP   127
A063  2818        370       JR   Z,BORRA
A065  FE0D        380       CP   13
A067  280D        390       JR   Z,FIN       ; Salta si se ha pulsado ENTER
A069  DD7700      400       LD   (IX),A
A06C  CD5ABB      410       CALL #BB5A       ; Escribe el caracter en
                  420 ;                         la pantalla
A06F  DD23        430       INC  IX          ; Siguiente posicion a llenar
A071  C1          440       POP  BC          ; Ahora mira si excede de la
                  450 ;                         longitud maxima
A072  10E9        460 BUCLE1 DJNZ BUCLE
A074  1801        470       JR   FIN2        ; Salta sin recuperar BC
A076  C1          480 FIN   POP  BC          ; Finaliza la pila
A077  3E07        490 FIN2  LD   A,7
A079  CD5ABB      500       CALL #BB5A       ; Emite un pitido
A07C  C9          510       RET
A07D  2AAEA0      520 BORRA LD   HL,(TEMPIX);Vendra aqui cuando
A080  AF          530       XOR  A           ; se pulse DEL
A081  DDE5        540       PUSH IX
A083  D1          550       POP  DE
A084  ED52        560       SBC  HL,DE       ; Si no hay nada en la cadena...
A086  2007        570       JR   NZ,ATRAS
A088  3E07        580       LD   A,7
A08A  CD5ABB      590       CALL #BB5A
A08D  181A        600       JR   NADA        ; pita y salta a NADA
A08F  3E08        610 ATRAS LD   A,8
A091  CD5ABB      620       CALL #BB5A
A094  3E20        630       LD   A,32
A096  CD5ABB      640       CALL #BB5A
A099  3E08        650       LD   A,8
A09B  CD5ABB      660       CALL #BB5A       ; Retrocede y escribe un espacio
A09E  C1          670       POP  BC          ; Ajusta el contador de caracteres
A09F  04          680       INC  B
A0A0  04          690       INC  B
A0A1  DD2B        700       DEC  IX          ; Ajusta la posicion de la cadena
A0A3  DD360020    710       LD   (IX),32     ; Espacio
A0A7  18C9        720       JR   BUCLE1      ; Vuelve
A0A9  C1          730 NADA  POP  BC
A0AA  04          740       INC  B
A0AB  04          750       INC  B
A0AC  18C4        760       JR   BUCLE1      ; Vuelve al bucle principal
A0AE  0000        770 TEMPIX DEFW 00
A0B0  00          780 TEMP  DEFB 0
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 A$=STRING$(40," ")
20 CALL 41000,@A$,35
30 PRINT
40 PRINT A$
```

Comentarios

La rutina finaliza inmediatamente si el número de parámetros no es correcto o si A no contiene el valor 2. Si el valor de L% es superior a la longitud de la cadena (desde BASIC) o mayor que la cantidad de espacio "dato" disponible (desde código máquina), también se produce un inmediato regreso. De otro modo, una vez llamada la rutina puedes teclear caracteres hasta que pulses la tecla ENTER o el número de caracteres tecleados supere L%. Los caracteres se imprimirán en la pantalla y también se almacenarán en el área de la cadena. Será posible borrar, pero la rutina NO te permitirá borrar más allá de donde comenzó la cadena. Sonará un pitido si lo intentas.

Prueba también la siguiente rutina de demostración en BASIC. Los bytes listados son para la dirección 41000 y supongo que el código máquina ya está almacenado en esta dirección.

```
100 MODE 2
110 P%=10:REM Numero de caracteres.
120 A$=STRING$(12," ")
130 CALL 41000,@A$,P%
140 PRINT:PRINT
150 PRINT A$
```

Si ejecutas esto, introduces algunos caracteres y luego listas el programa, advertirás que la definición de la cadena en la línea 120 ha sido modificada para recoger los caracteres que se han tecleado. Hablaré más de ello en el capítulo 9, cuando discutamos la estructura del programa en BASIC de Amstrad con más detalle.

Aunque las rutinas ROM existen en el Amstrad para distintas aplicaciones, como alterar los intervalos de repetición de las teclas, etc., no me propongo entrar aquí en la explicación de ninguna rutina. Normalmente es más sencillo utilizar equivalentes en BASIC. Así que aquí terminamos nuestro estudio de las rutinas de teclado.

7
Rutinas de sonido

El Amstrad es capaz de generar algunos efectos de sonido impresionantes y no hay duda que serás consciente de ello si ya has intentado usar las extensas facilidades de sonido del ordenador desde el BASIC. Muchas de estas facilidades también están disponibles desde el código máquina por medio de las rutinas del *firmware*. Estas permiten al programador de código máquina el acceso a la cola de espera, a las envolventes de tono, envolventes de volumen, etc. Todas ellas están cuidadosamente documentadas en el manual técnico de *firmware* de Amsoft, y no hay motivo para repetir aquí dicha información.

En lugar de eso, quiero dirigir la atención directamente hacia la programación del generador programable de sonido. Dirás que, ¿para qué molestarse si podemos conseguir los efectos mediante las rutinas del *firmware*? Bien, el empleo de las rutinas del *firmware* requiere, a menudo, complicadas tablas de datos que han de ser almacenadas en memoria antes de acceder a dichas rutinas. El proceso de inicialización no es tan largo de realizar cuando usamos directamente el GPS. Si bien el GPS no puede hacer todo lo que está disponible desde el BASIC, sí que es capaz de generar tonos, ruido y tiene algunas envolventes de volumen accesibles con facilidad; todo ello lo convierte en un dispositivo bastante versátil. Las envolventes de tono y demás, disponibles desde el BASIC, son producidas por el GPS con una pequeña ayuda de la CPU. La ventaja de usar

directamente el GPS es que para los efectos de sonido a menudo deseados en programas de juegos, etc., es más conveniente el acceso directo al GPS que el inicializar las diferentes tablas de datos que son requeridas por las rutinas del *firmware*. Claro está que para algunos efectos sonoros necesitaremos utilizar rutinas del *firmware*, pero aun así encontrarás sorprendente la cantidad de cosas que puedes obtener del GPS por sí solo.

¡Pita!

El efecto sonoro más simple que se puede obtener es el generado por el siguiente fragmento de código máquina.

```
LD      A,7
CALL    #BB5A
RET
```

Este "pitido" es útil para indicar que algo ha ocurrido o que se ha detectado un error, o para cosas así.

El generador programable de sonido

El GPS es un instrumento muy versátil y relativamente sencillo de usar, aunque a primera vista parezca un poco apabullante. Es un dispositivo de tres voces; esto es, es capaz de generar tres sonidos diferentes a un mismo tiempo. También posee unas cuantas envolventes de volumen, implementadas en su *hardware*. El control del volumen producido por cada canal se puede gobernar desde el exterior. Si no estás seguro sobre el significado de la palabra envolvente, entonces te sugiero que consultes el manual del usuario de Amstrad y leas el capítulo 6. La generación y la amplitud de los tonos emitidos son controlables individualmente para cada voz. La pastilla también es capaz de generar ruido blanco. Una colección de quince registros insertados en el GPS son los encargados de llevar a cabo todo esto. Estos registros pueden ser leídos o escritos de una forma similar a los de la CPU. El GPS también es capaz de realizar operaciones de entrada/salida, pero no vamos a profundizar en eso aquí. Aunque el sistema operativo del Amstrad accede a esta pastilla mediante el *chip* PPI (interfaz periférico programable), no podemos escribir directamente en los registros de la PPI. Bueno, en realidad, sí podemos, pero no sería aconsejable debido a la complejidad de las operaciones de E/S del Amstrad y la posibilidad de olvidar algún detalle mientras lo preparamos. En cambio, Amstrad nos ofrece una rutina muy útil que nos permite escribir valores en un determinado registro del GPS sin peligro alguno de dejar las cosas a medio hacer.

La rutina MC REGISTRO DE SONIDO, del *firmware*

Esta rutina se encuentra en la dirección &BD34; para poder llamarla, el registro C contendrá un valor entre 0 y 255 y el registro A contendrá el número de registro del GPS en donde quieras escribir el valor. A la salida, los registros AF y BC estarán alterados. La siguiente rutina, REGISTRO, puede utilizarse desde el BASIC para escribir en valores en los registros del GPS.

Registro

Una rutina que permite al programador acceder directamente y desde el BASIC a los registros del GPS.

Requisitos de entrada: Partiendo desde BASIC, usaremos CALL dirección, reg, valor; siendo reg el número de registro, y valor el valor que queremos escribir en el registro.
Desde el código máquina, la rutina de la ROM puede ser llamada directamente.

Condiciones de salida: Irrelevante.

Longitud: 12 bytes.

```
9D08                10 REGIST ORG  40200
9D08  DD7E02        20        LD   A,(IX+2)
9D0B  DD4E00        30        LD   C,(IX+0)
9D0E  CD34BD        40        CALL #BD34
9D11  C9            50        RET
```

Comentarios

Es esencial para esta rutina que adquieras algunos conocimientos sobre los registros del generador programable de sonido antes de usarla. Veamos esto ahora mismo.

Registros del GPS

Esta sección es algo así como un "cursillo intensivo y apresurado" sobre los registros del GPS. Si lo que realmente quieres es estar versado en el dispositivo, entonces consulta la hoja de aplicaciones del AY-3-8912 de la General Instruments. Sin embargo, para propósitos de programación, los datos aquí presentados serán probablemente los más adecuados. Los registros del GPS están numerados, con afán de originalidad, del 0 al

15. Los registros 14 y 15 no son utilizados para la generación de sonidos, sino que están enredados con la parte de E/S del GPS. Como todos los registros, si no sabes exactamente qué es lo que estás haciendo, ¡NO LOS TOQUES!

Registros 0 al 5

Estos registros controlan la producción de las notas emitidas en los canales 1, 2 ó 3. La emisión del canal 1 es controlada por los registros 0 y 1, la del canal 2 por los registros 2 y 3, y la del canal 3 por los registros 4 y 5.

Los registros se componen de 12 bits efectivos; los 8 bits más bajos están en el registro de número más bajo de cada par, y los 4 bits más altos están contenidos en el registro con número mayor de los que componen el par. El registro con los ocho bits más bajos, esto es, el registro 0 para el canal 1, es denominado registro de control fino del tono, y el registro con los cuatro bits más altos se denomina registro de control grueso del tono. La razón para esto es obvia; un simple incremento en el registro de control grueso tiene un efecto notable en la frecuencia del tono emitido, mientras que un incremento en el registro de control fino es apenas discernible en el tono producido. Por eso, el registro 0 es el registro de control fino del tono para el canal 1, y el Registro 1 es el registro de control grueso de tono para el mismo canal.

Cuanto más grande sea el número que contengan estos registros, más grave será el tono generado en ese canal en particular.

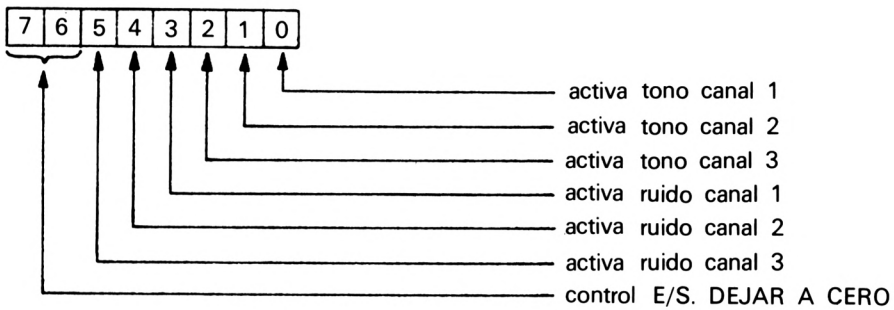
Registros 8 al 10

Estos son los llamados registros de control de amplitud del GPS. Existe uno para cada canal y cada registro tiene 5 bits. Estos 5 bits les permiten contener un valor entre 0 y 31. Sin embargo, si el bit 4 está a 1, el canal correspondiente se comportará de forma especial como enseguida veremos. Normalmente, el volumen del sonido emitido en cada canal dependerá del valor contenido en los bits 0 a 3. El silencio se consigue con el valor 0 y el sonido más fuerte se obtendrá cuando el valor sea 15. Si el bit 4 está puesto a 1, entonces la amplitud del sonido generado en ese canal se controla por una de las envolventes de volumen propias del GPS, en vez de estar controlado por los 4 bits más bajos del registro de control de amplitud. Después examinaremos con gran detalle estas envolventes de volumen. El registro 8 controla el volumen del canal 1, el registro 9 controla el canal 2 y el registro 10 controla la amplitud del canal 3.

Pero no es suficiente indicar una amplitud y activar un determinado canal para que el GPS reproduzca una nota seleccionada. La generación de un sonido en un determinado canal depende del estado en que se

encuentren ciertos bits del registro 7. Este es el registro de control del GPS.

Registro 7



Los bits 6 y 7 de este registro conciernen únicamente al control de las entradas/salidas del GPS. En el Amstrad, aparecen de alguna manera encargados del teclado. Por ejemplo, si en un descuido dejas el bit 6 puesto a 1, el teclado permanecerá totalmente muerto y aislado del ordenador. Si estás en modo interactivo, la única forma de salir de esa situación es apagar el ordenador. Si esto tiene lugar en el transcurso de un programa, puedes tener preparadas algunas instrucciones para desactivar el bit; pero sugiero que lo mejor es dejar los bits 6 y 7 permanentemente a 0.

Bits 0 a 2

Estos 3 bits controlan la generación de tonos en los canales 1, 2 y 3. Si un bit permanece desactivado (a cero), entonces se generará un tono en el canal correspondiente, asumiendo que el registro de control de amplitud de ese canal contiene un valor apropiado. De esta forma, para que el canal 1 emita una nota, tendrán que llevarse previamente a cabo los siguientes pasos:

1. Los registros 0 y 1 contendrán el valor correspondiente al tono elegido.
2. Se guardará en el registro 8 el volumen adecuado.
3. El registro 7 deberá contener &X00111110.

Cuando un bit de un canal dado está puesto a cero de esta forma, se dice que el canal referido está ACTIVADO. Si el bit está puesto a 1, no se

emitirá ningún tono y se dirá que el canal está DESACTIVADO para producir tonos.

Si ponemos los 3 bits a cero, conseguiremos que los tres canales emitan tonos simultáneamente.

Bits 3 a 5

Estos son los denominados bits de activación de ruido, y son los responsables de controlar si aparecerá o no ruido blanco en uno cualquiera o en los tres canales. Dentro de un momento hablaremos de ruido. Al igual que antes, si uno de estos bits está puesto a 0 provocará que aparezca ruido blanco en un canal determinado. El volumen del ruido coincidirá con el que indique el registro de control de amplitud (registro 8 al 10) de ese canal. El ruido de un canal permanecerá desactivado si su bit correspondiente está puesto a 1.

Es posible emitir a un mismo tiempo y en un mismo canal una nota y un ruido. Esto se consigue sencillamente poniendo a 0 los bits de tono y ruido para ese canal. Sin embargo, no es posible tratar por separado el volumen del ruido y el del tono, ya que el registro de control de amplitud es el mismo para ambos. Así pues, los ruidos y tonos que se produzcan simultáneamente en un canal serán emitidos a un mismo volumen. Por supuesto, el ruido también puede ser generado bajo el control de una envolvente de amplitud, del mismo modo que se hace para un tono.

Generación de ruido

El generador programable de sonido es capaz de producir ruido blanco en una amplia banda de "frecuencia", si es que este término se puede aplicar al ruido. Un ruido bajo en frecuencias producirá un sonido "impetuoso", mientras que valores elevados producirán una especie de "siseo". La producción de ruidos emitidos por los tres canales es controlada por un solo registro.

Registro 6

Este es el registro de control de ruido producido. Se trata de un registro de 5 bits, ofreciendo la posibilidad de valores entre 0 y 31 para el ruido a generarse. El valor 31 dará el ruido más bajo en frecuencias, mientras que 0 dará el ruido más alto. Como ya mencioné, sólo disponemos de un único registro de control de ruido para los tres canales; por eso el ruido emitido por cada canal no puede controlarse individualmente.

Finalmente, consideremos las envolventes que residen en el GPS. Estas son todas las envolventes de amplitud y pueden ser de gran utilidad. El

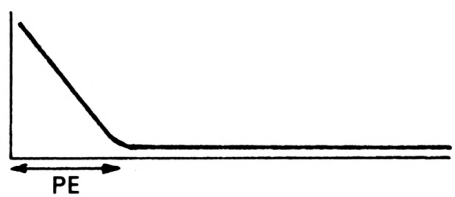
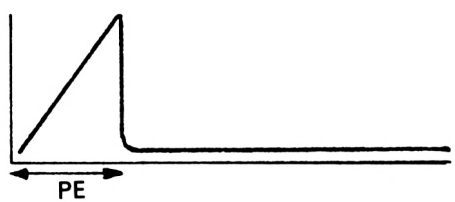
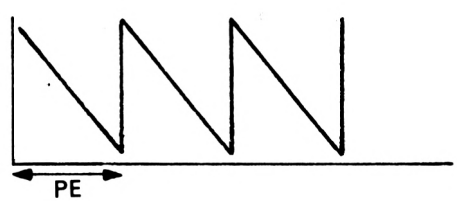
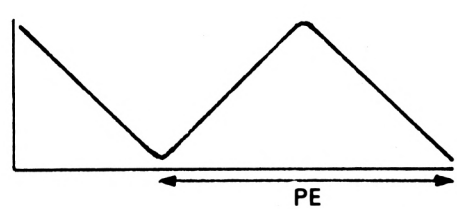
resto de los registros del GPS se refieren todos ellos al control de envolventes y a las E/S.

Envolventes

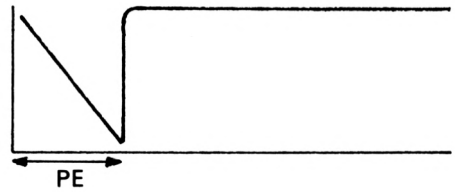
Existen dos parámetros que describen las envolventes de amplitud que ofrece el GPS. Estos son el número de envolvente, que describe "la forma" de la envolvente, y el período de envolvente, que es una valoración de la cantidad de tiempo que llevará emitir el sonido.

Registro 13

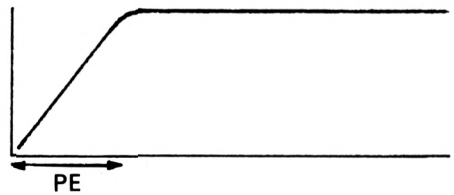
El contenido de este registro especifica la forma de la envolvente que se aplicará a cualquiera de los sonidos con envolvente que se emitan.

Contenido del registro 13	Forma de la envolvente
1	
4	
8	
10	

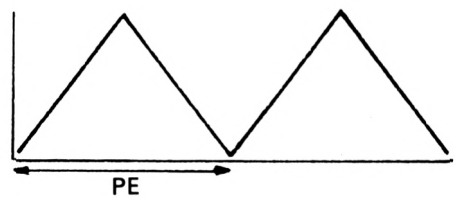
11



13



14



PE indica el período de la envolvente, y su duración está controlada por el valor contenido en los dos registros del período de envolvente, R11 y R12. Juntando ambos registros, se forma un valor de 16 bits, del cual R11 es el byte bajo y R12 es el byte alto. Cuanto más alto sea el valor de estos registros, mayor será la duración de la envolvente. Cuanto mayor sea el período de la envolvente, será más lento el cambio de amplitud durante la emisión de la nota.

El que una envolvente se aplique a un tono emitido, a un canal determinado, o a un ruido o lo que sea, dependerá del estado del bit 4 del registro de control de amplitud del canal correspondiente. Todos los canales que tengan este bit puesto a 1 emitirán tonos o ruido con la misma envolvente. Esto se debe al hecho de que solamente hay un registro de forma de la envolvente y otro registro del período de envolvente.

Técnicas de sonido

Es fácil convertirse en maestro de las técnicas básicas de empleo directo del GPS. Por lo general, necesitarás hacer algunos ensayos y cometer algunos errores antes de que comiencen a salir buenos efectos sonoros del altavoz de tu Amstrad. Si quieres dejar desactivados uno o más canales, es cierto que puedes poner a cero el registro de control de amplitud apropiado. Probablemente no sea ésta la mejor manera de hacer las cosas, aunque sólo sea porque esto elimina a la vez tanto el tono como

el ruido del canal. En general es mejor utilizar el registro 7 para controlar la generación de sonido. El registro 7 puede contener &X00111111, mientras que los demás registros permanecen inicializados; de esta forma los tonos y ruidos de todos los canales quedan desactivados. Después, cuando quieras producir un sonido, no tienes más que poner a 0 los bits apropiados del registro 7 para activar ruido, tono o ambos, en el canal deseado.

Mientras se está generando un sonido, puedes alterar el contenido de los registros. De esta forma se puede desvanecer gradualmente un sonido, bajo control de la CPU, o puedes alterar su tono poco a poco. De hecho, ésta es la forma en que el sistema operativo del Amstrad genera sus envolventes de tono y de volumen: alterando los registros del GPS bajo control de las interrupciones. Debe observarse, sin embargo, que una vez que un canal comienza a emitir sonido continuará haciéndolo hasta que el registro 7 sea convenientemente alterado. Esto es de gran utilidad, especialmente trabajando con las envolventes del GPS, ya que la CPU puede olvidarse momentáneamente del generador de sonido, y hacer algo más importante.

Me gustaría terminar este capítulo con unos cuantos ejemplos de efectos de sonido. Estos están producidos por las propias envolventes del GPS y sirven para mostrar parte de lo que hay disponible. Si deseas producir efectos más exóticos, puedes elaborar sencillos bucles con rutinas en código máquina que actualicen los registros del GPS mientras se está emitiendo otro sonido. Sin embargo, creo que estos ejemplos te darán un punto de partida. Para escucharlos es suficiente inicializar los registros apropiados del GPS con los valores dados, siendo el registro 7 el último en cargar.

Explosión

Registro	Valor
6	30
8	16
11	255
12	30
13	1
7	&X00110111

Locomotora de vapor

Registro	Valor
6	30
8	16
11	255
12	0
13	14
7	&X00110111

Tal vez te guste más si intentas cambiar el tipo de ruido, alterando el contenido del registro 6.

Rayo laser

Registro	Valor
0	255
1	0
8	16
11	100
12	0
13	10
7	&X00111110

¡Boing!

Registro	Valor
0	0
1	9
8	16
11	191
12	10
13	1
7	&X00111110

Disparo

Registro	Valor
6	30
8	16
11	255
12	2
13	1
7	&X00110111

En todas estas rutinas se entiende que los registros no mencionados están todos puestos a cero.

8

Rutinas de manipulación del cassette

Existe una variedad de rutinas en el Amstrad que permiten al programador de código máquina acceder con mayor facilidad a las rutinas encargadas de las operaciones de cassette. Con todas las facilidades que ofrece el BASIC podrías preguntarte para qué molestarse en acceder a esas rutinas desde programas en código máquina. Bueno, es muy útil para cuando estés escribiendo programas que requieran grabar datos en la cinta. Además, no te vendrán mal algunos conocimientos sobre las rutinas de cassette para producir formatos individualizados de cassette para protección en *software* o para ficheros especiales de datos a los que sólo puedan acceder determinados programas.

De cualquier modo, empezaremos con un par de llamadas *firmware* bastante útiles que puedes usar sin inicializar los registros.

Control del motor

Una característica algo irritante del sistema de cassette del Amstrad es el modo en que la tecla PLAY se inhibe, excepto en operaciones de entrada, salida, o de catalogación. Esto significa que para hallar una parte en blanco en la cinta es necesario catalogar la cinta mediante la orden CAT, impidiéndote realizar cualquier otra cosa mientras el cassette avanza.

Pueden usarse dos rutinas ROM para controlar el motor del cassette; CALL &BC6E pondrá en marcha el motor y CALL &BC71 lo desconectará. Una vez conectado el motor, habrá una pequeña pausa antes de que aparezca la expresión "*Ready*". ¡No te asustes! Se trata sólo del sistema operativo asegurándose de que la cinta ha alcanzado una velocidad uniforme de avance antes de devolver el control.

La rutina llamada en la dirección &BC9B también es bastante útil, ya que es el código máquina que realiza la función equivalente a la orden CAT del BASIC. Si utilizas esta última llamada, DE debe apuntar a un área de memoria de 2048 bytes que la rutina *firmware* pueda usar como espacio de trabajo.

CAT

No debe confundirse con la orden de BASIC del mismo nombre; esta rutina en código máquina cargará la cabecera de un fichero de la cinta. Una vez en memoria, los distintos detalles sobre el fichero, como, por ejemplo, la longitud, dirección de comienzo, etc., están disponibles para consulta.

Requisitos de entrada: Desde BASIC o código máquina, simplemente llama a la rutina mediante la orden CALL. Debe haber un espacio temporal de memoria disponible. (Véase Comentarios.)

Condiciones de salida: Todos los registros son alterados.

Longitud: 12 bytes más 64 bytes para la reserva de memoria.

```
                    10 ;      ** CABECERA **
9D08                20        ORG  40200
9D08  3E2C          30        LD   A,#2C     ; Codigo de sincronismo esperado
9D0A  21409C        40        LD   HL,40000  ; Registro de memoria en 40000
9D0D  114000        50        LD   DE,64     ; Numero de bytes a cargar
9D10  CDA1BC        60        CALL #BCA1     ; Los carga
9D13  C9            70        RET  ;           Terminado
```

Ensámblalo y pruébalo con el siguiente programa BASIC.

```
10 CLS
20 CALL 40200
30 nombre$=" " : buffer=40000
40 FOR I=buffer TO buffer+15
50 nombre$=nombre$+CHR$(PEEK(I)):NEXT
60 inicio=PEEK(buffer+21)+256*PEEK(buffe
   r+22)
```

```
70 longitud=PEEK(buffer+24)+256*PEEK(buf
   fer+25)
80 PRINT nombre$ : PRINT "Longitud :....
   ..... ";longitud
90 PRINT "Comienzo en direccion :      "
   ;inicio
100 PRINT:PRINT
110 GOTO 20
```

Comentarios

Esta rutina es relocalizable, siempre que la dirección de la zona temporal de memoria se altere si es necesario. En esta rutina, HL se usa para transferir la dirección de la zona temporal de memoria a la rutina de *firmware*. En la rutina listada se ha usado 40000 como dirección de esta memoria temporal. DE contiene el número de bytes que la rutina de *firmware* va a cargar, y que será siempre 64 para el encabezamiento.

El registro A debe contener un valor llamado byte de sincronismo, a la entrada de la rutina del *firmware*, que diferencia efectivamente el encabezamiento de un bloque de datos dentro de un fichero. El encabezamiento para un bloque de datos es &16 y para un bloque normal de encabezamiento es &2C. Si el byte de sincronismo de un bloque de información que esté en la cinta no corresponde al byte de sincronismo del registro A, tal información será ignorada.

Cuando la rutina *firmware* ha recibido el encabezamiento, regresaremos al BASIC. Sin embargo, también puede ocurrir que la vuelta al BASIC se produzca por una condición errónea de cualquier tipo. Aquí no hemos utilizado este tipo de información pero tal vez quieras extender la rutina anterior añadiendo mensajes de error, así que vamos allá.

Si la operación de carga ha dado resultado, el indicador de acarreo se pondrá a 1. De otro modo, C=0. En este caso el registro A contiene un número de error.

A=0: ESC ha sido pulsada para terminar la operación.

A=1: Error de sobrecarga. Esto ocurre si hay más datos disponibles en el cassette de los especificados en el registro DE cuando la rutina de &BCA1 ha sido llamada.

A=2: Indica un error PRC. La prueba de redundancia cíclica es el sistema mediante el cual el sistema operativo puede determinar si se ha cometido un error en la lectura actual de un byte de datos desde el cassette. Si ocurre esto, generalmente indica que uno o más de los bytes leídos desde el cassette están alterados.

Una vez que tenemos el bloque de encabezamiento en la memoria, nos queda por hacer su decodificación. En esta rutina sólo buscamos el nombre, longitud total y dirección de comienzo del fichero. El resto de los bytes del encabezamiento no nos interesan aquí. Sin embargo, si eres curioso, el manual del *firmware* te proporcionará los detalles que precises; también podrás encontrarlos en *Programación avanzada del Amstrad* publicado por esta casa.

Nombre del fichero: Está almacenado en los 16 primeros bytes del encabezamiento, y ésta es la razón de que el nombre de un fichero para ficheros de cassette sólo pueda tener una longitud de 16 caracteres. Si el nombre es más corto de 16 letras, entonces se completan los 16 caracteres rellenándose con códigos nulos (CHRS(0)).

Dirección de comienzo: Se almacena en los bytes 21 y 22 del encabezamiento. El bloque se numera de 0 en adelante. Esta dirección desde la que los datos fueron grabados, se almacena en el formato usual del Z-80 con el byte bajo en primer lugar. Sin embargo, hay un punto a considerar. Esta sólo es la dirección de comienzo de ese bloque de datos en particular y, por tanto, sólo será la dirección de comienzo de todo el fichero si es la del primer bloque de datos. Para bloques posteriores ésta será la dirección de comienzo de ese bloque de datos en particular. El número del bloque, que puede ser de utilidad para estos casos, se almacena en el byte 16 del encabezamiento.

Longitud del fichero: Es la longitud total del fichero grabado. Se almacena en los bytes 24 y 25 del encabezamiento, de nuevo con el byte bajo almacenado en primer lugar.

El siguiente programa en BASIC utiliza la rutina CAT para conseguir información más detallada a partir de las entradas del encabezamiento mencionadas anteriormente. Yo lo encuentro particularmente útil para revisar mis programas en código máquina, que tiendo a almacenar como ficheros que contienen únicamente bytes. De paso te diré que esta rutina no está diseñada para facilitarte el acceso a los programas ajenos. Es ilegal, así que no lo hagas. He considerado que parte del programa en código máquina está en la dirección 40200 y que la zona temporal de memoria para los datos introducidos está en la dirección 40000. El programa, según se ejecuta, espera a los distintos encabezamientos y escribe algunos detalles en la pantalla. Pulsando ESC terminará el programa.

```
10 buffer=40000
20 CALL 40200
30 nombre$=" "
40 FOR I=buffer TO buffer+15
```

```
50 nombre$=nombre$+CHR$(PEEK(I))
55 NEXT I
60 inicio=PEEK(buffer+21)+256*PEEK(buffe
   r+22)
70 longitud=PEEK(buffer+24)+256*PEEK(buf
   fer+25)
80 PRINT nombre$
85 PRINT "Longitud : ";longitud
90 PRINT "Direccion inicial :";inicio
100 PRINT:PRINT
110 GOTO 20
```

Cuando tengas detalles acerca del resto del encabezamiento, podrás ampliar fácilmente esta rutina. Sin embargo, ésta te servirá para muchas aplicaciones.

ESCRIBCASS (ESCRIBe en CASSette)

Es una rutina para escribir un fichero de datos con nombre en la cinta. Es realmente una versión en código máquina de la orden SAVE, aplicada a ficheros en binario, por lo que no daré requisitos de entrada en BASIC. También daré un par de rutinas lectoras, que te permitan guardar y cargar ficheros de datos desde código máquina.

Requisitos de entrada: IX apunta a un bloque de encabezamiento de 64 bytes, como se muestra en los Comentarios de esta rutina.

Condiciones de salida: Todos los registros son alterados.

Longitud: 37 bytes.

```
                    10 ;       ** ESCRIBCASS **
9D08                20         ORG  40200
9D08  DD212D9D      30         LD   IX,CABEZ
9D0C  DDE5          40         PUSH IX
9D0E  212D9D        50         LD   HL,CABEZ
9D11  114000        60         LD   DE,64
9D14  3E2C          70         LD   A,#2C
9D16  CD9EBC        80         CALL #BC9E
9D19  DDE1          90         POP  IX
9D1B  DD6E15       100         LD   L,(IX+21)
9D1E  DD6616       110         LD   H,(IX+22)
9D21  DD5E13       120         LD   E,(IX+19)
9D24  DD5614       130         LD   D,(IX+20)
9D27  3E16         140         LD   A,#16
9D29  CD9EBC       150         CALL #BC9E
9D2C  C9           160         RET
9D2D  54455354     170 CABEZ   DEFM "TEST"
9D31               180         DEFS 12
9D3D  01           190         DEFB 1
9D3E               200         DEFS 2
9D40  E803         210         DEFW 1000
9D42  6901         220         DEFW 361
9D44  00           230         DEFB 0
```

Pruébalo con esto:

```
CALL 40200
```

Comentarios

El bloque de encabezamiento mencionado es una versión simplificada del utilizado por el sistema operativo cuando lleva a cabo operaciones de grabación y carga en BASIC. Es importante que observes que, aunque los ficheros guardados por ESCRIBCASS aparecerán con la orden CAT del BASIC, no serán cargados si usas la orden LOAD del BASIC. El fichero grabado por ESCRIBCASS consiste en un encabezamiento seguido de un bloque de datos. La estructura de un bloque de encabezamiento deberá ser de la forma siguiente:

```
Bytes 0 a 15   NOMBRE DEL FICHERO+CHR$(0)'s hasta completar 15
Bytes 19 y 20  LONGITUD DEL FICHERO, primero el byte bajo.
Bytes 21 y 22  DIRECCION DE COMIENZO, primero el byte bajo.
Bytes 24 y 25  LONGITUD DEL FICHERO, primero el byte bajo.
```

El resto de los bytes del encabezamiento deberá ponerse a cero. Como ejemplo más detallado, examina el siguiente bloque de encabezamiento, que guardará en la cinta un bloque de datos de 1000 bytes de longitud, comenzando en la dirección 361, y con nombre TEST para el fichero.

```
CABECERA    DEFM    "TEST"
            DEFS    12
            DEFB    0
            DEFB    0
            DEFB    0
            DEFW    1000
            DEFW    361
            DEFW    00
            DEFW    1000
```

El resto del bloque se pone a cero. Advierte que, una vez que se llama a la rutina ESCRIBCASS, el fichero se graba inmediatamente en la cinta, sin los habituales bytes de sincronismo. Estos podrían añadirse fácilmente si fueran necesarios para una aplicación en particular. La única nueva rutina ROM utilizada es CAS WRITE, accesible en la dirección &BC9E. A la entrada, HL contiene la dirección de los datos que se van a escribir, DE la longitud y A el byte de sincronismo. La usamos dos veces, una vez para meter el encabezamiento en la cinta y otra para meter los datos en la cinta.

Una vez hayamos escrito un fichero en la cinta, es conveniente poder leerlo de nuevo. Te muestro dos rutinas para ello, LEECASS y, haciendo alardes de imaginación, LEECASS2. Esta última es sólo una versión más sencilla de usar que LEECASS.

LEECASS (LEE del CASSette)

Esta rutina lee en la cinta los ficheros escritos con ESCRIBCASS.

Requisitos de entrada: Véase Comentarios.

Condiciones de salida: Todos los registros son alterados.

Longitud: 54 bytes, excluyendo las tablas para NOMBRE y CABEZA.

```
                    10 ;       ** LEECASS **
9D08                20         ORG  40200
9D08  114000        30 CARGA   LD   DE,64      ; Numero de bytes del encabezamiento
9D0B  21509D        40         LD   HL,CABEZ    ; Los pone aqui
9D0E  3E2C          50         LD   A,#2C       ; Byte para un sincronismo correcto
9D10  CDA1BC        60         CALL #BCA1       ; Carga la cabecera
9D13  21509D        70         LD   HL,CABEZ    ; DE y HL apuntan hacia el
9D16  113E9D        80         LD   DE,NOMBRE   ; nombre del fichero y hacia
                    90 ;                          el nombre deseado
9D19  1A           100 BUCLE   LD   A,(DE)      ; Examina cada caracter
9D1A  BE           110         CP   (HL)
9D1B  20EB         120         JR   NZ,CARGA    ; Si no es el mismo,
                   130 ;                          carga el siguiente encabezamiento
9D1D  23           140         INC  HL
9D1E  13           150         INC  DE
9D1F  1A           160         LD   A,(DE)
9D20  FE00         170         CP   0           ; Cero indica el final del nombre
9D22  20F5         180         JR   NZ,BUCLE    ; Si no es el final, pasa al
                   190 ;                          siguiente caracter
9D24  DD213E9D     200         LD   IX,NOMBRE   ; Extrae la direccion de carga
9D28  DD6E10       210         LD   L,(IX+16)
9D2B  DD6611       220         LD   H,(IX+17)
9D2E  DD21509D     230         LD   IX,CABEZ    ; Extrae la longitud de la cabecera
9D32  DD5E18       240         LD   E,(IX+24)
9D35  DD5619       250         LD   D,(IX+25)
9D38  3E16         260         LD   A,#16       ; Byte de sincronismo
9D3A  CDA1BC       270         CALL #BCA1
9D3D  C9           280         RET
9D3E  54455354     290 NOMBRE  DEFM "TEST"
9D42               300         DEFS 12
9D4E  409C         310         DEFW 40000       ; Direccion de carga
9D50               320 CABEZ   DEFS 64          ; 64 ceros
```

Comentarios

Esta rutina requiere la presencia de dos tablas de datos, que he llamado NOMBRE y CABEZA. La tabla CABEZA no es más que un bloque de 64 bytes en los que puede cargarse un encabezamiento. NOMBRE, sin embargo, debe ser organizado por el programador. Como sugiere su nombre, contiene el nombre del fichero que quieras leer. También recoge la dirección en la que se va a cargar el fichero. Tiene una longitud de 18 bytes.

Bytes 0 a 15: Contienen el nombre del fichero, rellenándose hasta completar los 16 bytes con CHR$(0).

Bytes 16 y 17: La dirección de carga del fichero se almacena en estas dos posiciones, con el byte bajo en primer término.

Dada su sencillez, esta rutina plantea una serie de inconvenientes. Surgen en cuanto a la imposibilidad de interrumpir la búsqueda de un nombre de fichero, por la ausencia de mensajes mientras la búsqueda continúa, o cuando se descubre un error en la longitud de un fichero, como casos principales. Vamos a ver primero este último problema.

El número de bytes cargados por esta rutina se toma del encabezamiento cuyo nombre coincide con el del fichero que nos interesa. Puede surgir un problema en cuanto que los bytes cargados en el espacio designado como inicial para la dirección de carga pueden superponerse a programas u otros datos que en principio no se suponían allí. Desde luego esto no ocurrirá si tienes cuidado al diseñar los programas, pero puede producirse. Si te interesa evitar este posible accidente, puedes seguir las siguientes instrucciones:

1. Designa una dirección concreta en la memoria como término del área en el que puedan cargarse los datos leídos en la cinta.
2. Cuando se lea el encabezamiento de un fichero, añade su longitud a la que contiene la tabla NOMBRE en la entrada "dirección de carga". Si el resultado excede de la dirección mencionada en 1, no lo cargues.

LEECASS2 no incluye esta característica, pero podrás añadirla fácilmente utilizando los detalles que te he dado.

LEECASS2

Versión modificada de LEECASS.

Requisitos de entrada: Véase comentarios.

Condiciones de salida: Todos los registros son alterados.

Longitud: 215 bytes, excluyendo NOMBRE y CABEZA.

```
                    10 ;       ** LEECASS2 **
9D08                20         ORG  40200
9D08  DD21C99D      30         LD   IX,BUSCA
9D0C  CD619D        40         CALL ESCR
9D0F  DD21B39D      50 CARGA   LD   IX,OK
9D13  CD619D        60         CALL ESCR
9D16  CD18BB        70         CALL #BB18
9D19  FE73          80         CP   115
9D1B  2803          90         JR   Z,SI
9D1D  FE53         100         CP   83
9D1F  C0           110         RET  NZ
9D20  114000       120 SI      LD   DE,64      ; Numero de bytes del encabezamiento
9D23  21F39D       130         LD   HL,CABEZ   ; Los pone aqui
9D26  3E2C         140         LD   A,#2C      ; Byte para un sincronismo correcto
```

```
9D28  CDA1BC      150         CALL #BCA1      ; Carga la cabecera
9D2B  3056        160         JR   NC,ERR     ; Acarreo a cero = error
9D2D  21F39D      170         LD   HL,CABEZ   ; DE y HL apuntan hacia el
9D30  11E19D      180         LD   DE,NOMBRE  ; nombre del fichero y hacia
                  190 ;                         el nombre deseado
9D33  1A          200 BUCLE   LD   A,(DE)     ; Examina cada caracter
9D34  BE          210         CP   (HL)
9D35  20D8        220         JR   NZ,CARGA   ; Si no es el mismo,
                  230 ;                         carga el siguiente encabezamiento
9D37  23          240         INC  HL
9D38  13          250         INC  DE
9D39  1A          260         LD   A,(DE)
9D3A  FE00        270         CP   0          ; Cero indica el final del nombre
9D3C  20F5        280         JR   NZ,BUCLE   ; Si no es el final, pasa al
                  290 ;                         siguiente caracter
9D3E  DD21D59D    300         LD   IX,ENTRA
9D42  CD619D      310         CALL ESCR
9D45  DD21E19D    320         LD   IX,NOMBRE  ; Extrae la direccion de carga
9D49  DD6E10      330         LD   L,(IX+16)
9D4C  DD6611      340         LD   H,(IX+17)
9D4F  DD21F39D    350         LD   IX,CABEZ   ; Extrae la longitud de la cabecera
9D53  DD5E18      360         LD   E,(IX+24)
9D56  DD5619      370         LD   D,(IX+25)
9D59  3E16        380         LD   A,#16      ; Byte de sincronismo
9D5B  CDA1BC      390         CALL #BCA1
9D5E  3023        400         JR   NC,ERR
9D60  C9          410         RET
9D61  DDE5        420 ESCR    PUSH IX
9D63  CD54BB      430         CALL #BB54
9D66  3E0D        440         LD   A,13
9D68  CD5ABB      450         CALL #BB5A
9D6B  3E0A        460         LD   A,10
9D6D  CD5ABB      470         CALL #BB5A
9D70  DDE1        480         POP  IX
9D72  DD7E00      490 BUCLES  LD   A,(IX)
9D75  FE00        500         CP   0
9D77  C8          510         RET  Z
9D78  DDE5        520         PUSH IX
9D7A  CD5DBB      530         CALL #BB5D
9D7D  DDE1        540         POP  IX
9D7F  DD23        550         INC  IX
9D81  18EF        560         JR   BUCLES
9D83  DD21909D    570 ERR     LD   IX,ERROR
9D87  CD619D      580         CALL ESCR
9D8A  3E07        590         LD   A,7
9D8C  CD5ABB      600         CALL #BB5A
9D8F  C9          610         RET
9D90  54656E67    620 ERROR   DEFM "Tengo que informarte que hay error"
9DB2  00          630         DEFB 0
9DB3  436F6E74    640 OK      DEFM "Continua la busqueda?"
9DC8  00          650         DEFB 0
9DC9  42757363    660 BUSCA   DEFM "Buscando..."
9DD4  00          670         DEFB 0
9DD5  43617267    680 ENTRA   DEFM "Cargando..."
9DE0  00          690         DEFB 0
9DE1  54455354    700 NOMBRE  DEFM "TEST"
9DE5              710         DEFS 12
9DF1  409C        720         DEFW 40000      ; Direccion de carga
9DF3              730 CABEZ   DEFS 64         ; 64 ceros
```

Comentarios

Como antes, NOMBRE contiene el nombre del fichero que quieras leer y su dirección de carga. CABEZA es un bloque de 64 bytes para el almacenamiento temporal de los encabezamientos leídos en la cinta. Ob-

servarás (espero) que hemos usado una variación del programa ECADEN para escribir mensajes. Al llamar a la rutina aparecerán los mensajes:

```
"Buscando....."
"Continua la busqueda ?"
```

En ese momento, contestando cualquier cosa menos "S" o "s", la rutina terminará. Esta pregunta se formulará cada vez que se lea un encabezamiento que no sea el del archivo que te interese. Los errores de lectura también se señalan mediante mensajes apropiados.

La última rutina para manejo de cassette que solemos necesitar es una función de comprobación. Esto lo realiza VERIFICA.

VERIFICA

Esta rutina contrasta un fichero de bytes de la cinta con un área de memoria del ordenador. Ello te permitirá comprobar que un área de memoria ha sido correctamente grabada antes de almacenar otra cosa en ella.

Requisitos de entrada: Véase Comentarios.

Condiciones de salida: Todo alterado.

Longitud: 221 bytes, excluyendo NOMBRE y CABEZA.

```
                    10 ;       ** VERIFICA **
9D08                20         ORG   40200
9D08  DD21C09D      30         LD    IX,BUSCA
9D0C  CD619D        40         CALL  ESCR
9D0F  DD21AA9D      50 CARGA   LD    IX,OK
9D13  CD619D        60         CALL  ESCR
9D16  CD18BB        70         CALL  #BB18
9D19  FE73          80         CP    115
9D1B  2803          90         JR    Z,SI
9D1D  FE53         100         CP    83
9D1F  C0           110         RET   NZ
9D20  114000       120 SI      LD    DE,64       ; Numero de bytes del encabezamiento
9D23  21ED9D       130         LD    HL,CABEZ    ; Los pone aqui
9D26  3E2C         140         LD    A,#2C       ; Byte para un sincronismo correcto
9D28  CDA1BC       150         CALL  #BCA1       ; Carga la cabecera
9D2B  3056         160         JR    NC,ERR      ; Acarreo a cero = error
9D2D  21ED9D       170         LD    HL,CABEZ    ; DE y HL apuntan hacia el
9D30  11DB9D       180         LD    DE,NOMBRE   ; nombre del fichero y hacia
                   190 ;                           el nombre deseado
9D33  1A           200 BUCLE   LD    A,(DE)      ; Examina cada caracter
9D34  BE           210         CP    (HL)
9D35  20D8         220         JR    NZ,CARGA    ; Si no es el mismo,
                   230 ;                           carga el siguiente encabezamiento
9D37  23           240         INC   HL
9D38  13           250         INC   DE
9D39  1A           260         LD    A,(DE)
9D3A  FE00         270         CP    0           ; Cero indica el final del nombre
9D3C  20F5         280         JR    NZ,BUCLE    ; Si no es el final, pasa al
                   290 ;                           siguiente caracter
9D3E  DD21CC9D     300         LD    IX,ENTRA
```

```
9D42  CD619D      310         CALL ESCR
9D45  DD21DB9D    320         LD   IX,NOMBRE ; Extrae la direccion de carga
9D49  DD6E10      330         LD   L,(IX+16)
9D4C  DD6611      340         LD   H,(IX+17)
9D4F  DD21ED9D    350         LD   IX,CABEZ  ; Extrae la longitud de la cabecera
9D53  DD5E18      360         LD   E,(IX+24)
9D56  DD5619      370         LD   D,(IX+25)
9D59  3E16        380         LD   A,#16     ; Byte de sincronismo
9D5B  CDA4BC      390         CALL #BCA4
9D5E  3023        400         JR   NC,ERR
9D60  C9          410         RET
9D61  DDE5        420 ESCR    PUSH IX
9D63  CD54BB      430         CALL #BB54
9D66  3E0D        440         LD   A,13
9D68  CD5ABB      450         CALL #BB5A
9D6B  3E0A        460         LD   A,10
9D6D  CD5ABB      470         CALL #BB5A
9D70  DDE1        480         POP  IX
9D72  DD7E00      490 BUCLES  LD   A,(IX)
9D75  FE00        500         CP   0
9D77  C8          510         RET  Z
9D78  DDE5        520         PUSH IX
9D7A  CD5DBB      530         CALL #BB5D
9D7D  DDE1        540         POP  IX
9D7F  DD23        550         INC  IX
9D81  18EF        560         JR   BUCLES
9D83  DD21909D    570 ERR     LD   IX,ERROR
9D87  CD619D      580         CALL ESCR
9D8A  3E07        590         LD   A,7
9D8C  CD5ABB      600         CALL #BB5A
9D8F  C9          610         RET
9D90  4572726F    620 ERROR   DEFM "Error en la verificacion."
9DA9  00          630         DEFB 0
9DAA  436F6E74    640 OK      DEFM "Continua la busqueda?"
9DBF  00          650         DEFB 0
9DC0  42757363    660 BUSCA   DEFM "Buscando..."
9DCB  00          670         DEFB 0
9DCC  56657269    680 ENTRA   DEFM "Verificando..."
9DDA  00          690         DEFB 0
9DDB  54455354    700 NOMBRE  DEFM "TEST"
9DDF              710         DEFS 12
9DEB  409C        720         DEFW 40000     ; Direccion de carga
9DED              730 CABEZ   DEFS 64        ; 64 ceros
```

Comentarios

NOMBRE debería constituirse como la rutina LEECASS, con la diferencia de que los bytes 16 y 17 de NOMBRE deberán contener ahora no la dirección de carga, sino la dirección desde la que los bytes fueron grabados. Si se detecta alguna diferencia entre el área de memoria y el fichero de la cinta, se produce una salida inmediata de la rutina, con un mensaje de error para indicar el hecho. La rutina sólo verifica el bloque de datos.

Con esto completamos el capítulo sobre rutinas de manipulación de cassette. Para mayor información sobre las rutinas *firmware*, me remito al manual técnico de *firmware* de Amstrad, que es realmente útil.

9
BASIC y código máquina

En este capítulo veremos un par de rutinas diseñadas para ayudar al programador de BASIC, y algunas notas detalladas sobre la ampliación del sistema residente, el método para añadir órdenes del BASIC del Amstrad. Sin embargo, empezaremos con un par de rutinas que realmente no pueden integrarse en ningún otro capítulo de este libro.

TIEMPO

La variable del sistema, TIME, una vez calculada, contendrá el espacio de tiempo transcurrido desde que el ordenador se conectó. Esto no incluye los períodos de tiempo en que las interrupciones fueron prohibidas desde el interior de las rutinas en código máquina, así como los períodos de operaciones de cassette. La variable TIME es un número de 4 bytes, que se incrementa 300 veces por segundo. Es, por tanto, muy útil para aplicaciones de medida de tiempo. Una característica, sin embargo, que echo en falta es un método para alterar la variable TIME en cualquier momento. Este tipo de orden es muy común en otros aparatos, como el microordenador BBC (¡cómo oso nombrar tales palabras en estas páginas!) y es muy útil cuando el reloj se está utilizando para cronometrar sucesos breves en un programa en ejecución. Normalmente tenemos que ejecutar una línea como

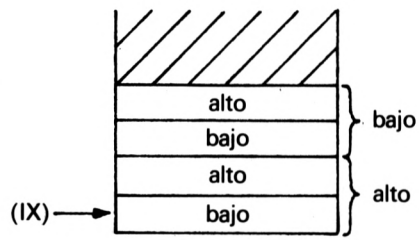
```
T=TIME:GOSUB 1000:PRINT TIME-T
```

mientras que, si pudiéramos poner TIME a cero, sólo diríamos:

```
CALL tiempo-a-cero:GOSUB 1000:PRINT TIME
```

Con esto se ve más claro lo que intentamos realizar. Esto lo hace la siguiente rutina, TIEMPO.

Requisitos de entrada: Desde BASIC, CALL dirección, bajo, alto, donde bajo son los 16 bits inferiores dentro del valor total de 32 bits y alto son los 16 bits superiores del valor total de 32 bits.
Desde otra rutina en código máquina, IX apunta a un bloque de parámetros y A contiene el valor 2.



Bloque de parámetros de TIEMPO

Condiciones de salida: Todos los registros son alterados. La rutina finalizará si se introduce en la misma un número erróneo de parámetros.

Longitud: 98 bytes.

```
                10 ;       ** TIEMPO **
9D08            20         ORG  40200
9D08  FE00      30         CP   0
9D0A  2814      40         JR   Z,CERO ; Si no hay parametros, salta a cero
9D0C  FE02      50         CP   2
9D0E  203C      60         JR   NZ,ERR ; Si no hay dos parametros, error
9D10  DD5E00    70         LD   E,(IX+0)
9D13  DD5601    80         LD   D,(IX+1) ; Parte superior de TIME
9D16  DD6E02    90         LD   L,(IX+2)
9D19  DD6603   100         LD   H,(IX+3) ; Parte inferior de TIME
9D1C  CD10BD   110         CALL #BD10 ; Fija el valor de TIME
9D1F  C9       120         RET
9D20  210000   130 CERO    LD   HL,0
9D23  110000   140         LD   DE,0
9D26  CD10BD   150         CALL #BD10 ; Pone a cero la variable TIME
9D29  C9       160         RET
9D2A  DDE5     170 ESCR    PUSH IX
9D2C  CD54BB   180         CALL #BB54
9D2F  3E0D     190         LD   A,13
9D31  CD5ABB   200         CALL #BB5A
9D34  3E0A     210         LD   A,10
9D36  CD5ABB   220         CALL #BB5A
9D39  DDE1     230         POP  IX
```

```
9D3B  DD7E00      240 BUCLE  LD   A,(IX)
9D3E  FE00        250        CP   0
9D40  C8          260        RET  Z
9D41  DDE5        270        PUSH IX
9D43  CD5DBB      280        CALL #BB5D
9D46  DDE1        290        POP  IX
9D48  DD23        300        INC  IX
9D4A  18EF        310        JR   BUCLE
9D4C  DD21599D    320 ERR    LD   IX,ERROR
9D50  CD2A9D      330        CALL ESCR
9D53  3E07        340        LD   A,7
9D55  CD5ABB      350        CALL #BB5A
9D58  C9          360        RET
9D59  4E756D65    370 ERROR  DEFM "Numero erroneo de parametros."
9D76              380        DEFS 1
```

Comentarios

El hecho de que TIME sea una variable de 32 bits hace que sea más bien difícil introducir este valor mediante una sentencia CALL estándar. Por eso el nuevo valor TIME se introduce en dos partes en la rutina en código máquina. Así, para colocar TIME en 100, deberemos ejecutar una orden como:

```
CALL direccion, 100,0
```

con "bajo" en primer lugar.

Los bytes del listado son para la dirección 41000, pero la rutina puede relocalizarse sin mucho problema. Yo suelo poner la variable TIME a cero, por lo que he decidido hacer de ello un caso especial. CALL dirección, por sí misma, hará que la variable TIME se coloque a cero. Proporcionaré algunas notas acerca de los valores que deben transferirse como parámetros. El valor de TIME para cualquier combinación de "alto" y "bajo" será:

```
TIME=bajo + (65536 * alto)
```

Así, la orden

```
CALL direccion, 0,2
```

colocará TIME a (2*65536)+0, que es 131072. Desde luego, al asignar a la variable TIME un valor muy alto, conseguiremos llegar al tope de capacidad y la variable comenzará otra vez con el valor 0.

Restaurando

Todo programa mal hecho se parece un poquito a mí; son necesarias un montón de correcciones después de intentar ejecutarlo, suponiendo que no hayas bloqueado el sistema. Por ejemplo, un problema corriente es

crear accidentalmente una combinación ilegible de TINTA y PAPEL, o un sonido interminable.

Bueno, pues hay algunas rutinas en el *firmware* que pueden llamarse para restaurar determinadas cosas un poco después de estos incidentes. Son las siguientes:

```
&BB4E   Manipulacion de la pantalla de texto.
&BB00   Teclado.
&BBBA   Manipulacion de la pantalla de graficos.
&BC02   Reinicializa la pantalla.
&BC65   Manipulacion del cassette.
&BCA7   Manipulacion del sonido.
&BD37   Restaura el bloque de saltos.
```

De todas ellas, &BC02, &BCA7 y &BD37 parecen ser las de mayor utilidad. &BC02 es extremadamente útil, puesto que coloca los colores en sus estados originales. Es muy útil cuando uno se las ha ingeniado para revolver la paleta del ordenador ¡estropeándolo todo! &BD37 te resultará útil sólo si realizas algún trabajo más especializado que conlleve la alteración de las entradas del bloque de saltos. Esta rutina las devuelve a sus condiciones iniciales. Llamando a &BCA7 obtendrás la eliminación de un sonido que no quiere callar. También puede utilizarse ejecutando programas para "limpiar" la memoria de sonido y parar así todo sonido que se esté produciendo en ese momento.

LEEROM (LEE de la ROM)

Como he mencionado antes en este libro, la RAM se superpone a los dos bloques de 16K de la ROM. POKE y PEEK operan ambos en posiciones de la RAM sólo en esta memoria. Sin embargo, a veces es interesante echar un vistazo a la ROM para ver los mensajes de error que aún no hayas descubierto, las estructuras de la tabla de órdenes, e incluso para ver cómo los programadores profesionales han diseñado una parte de código en particular. Esta rutina lista el contenido de las posiciones de la ROM en bloques de 20 bytes. Originalmente fue escrita para utilizarse con el modo 2 de pantalla. Las rutinas del *firmware* nos permiten el acceso a las ROM superior e inferior.

Bloque de parámetros de LEEROM

Requisitos de entrada: Desde BASIC, CALL dirección, comienzo, donde comienzo es la dirección de la primera posición cuyo contenido se va a listar. Desde código máquina, IX apunta a un bloque de parámetros de 2 bytes, y A = 1.

Condiciones de salida: Todos los registros son alterados.

Longitud: 161 bytes.

```
                    10 ;       ** LEEROM **
9D08                20         ORG  40200
9D08  0614          30         LD   B,20          ; Escribir 20 bytes
9D0A  DD6E00        40         LD   L,(IX+0)
9D0D  DD6601        50         LD   H,(IX+1)      ; Extrae la direccion de comienzo
9D10  C5            60 OBUCLE  PUSH BC
9D11  E5            70         PUSH HL            ; Guarda los registros
9D12  CD06B9        80         CALL #B906         ; Activa la ROM inferior
9D15  F5            90         PUSH AF            ; Guarda el registro
                   100 ;                           con el estado de la ROM
9D16  CD00B9       110         CALL #B900         ; Activa la ROM superior
9D19  7E           120         LD   A,(HL)
9D1A  32A89D       130         LD   (ALMACN),A    ; Toma un byte y lo almacena
9D1D  F1           140         POP  AF            ; Recupera el byte de estado
                   150 ;                           de la ROM,
9D1E  CD0CB9       160         CALL #B90C         ; y la devuelve a la normalidad
9D21  E1           170         POP  HL
9D22  E5           180         PUSH HL
9D23  CD909D       190         CALL EHEXHL        ; Escribe la direccion
                   200 ;                           en HEXadecimal
9D26  DD21999D     210         LD   IX,SPACIO     ; Deja un espacio
9D2A  CD4F9D       220         CALL ESCR
9D2D  3AA89D       230         LD   A,(ALMACN)    ; Toma el byte leido,
9D30  CD679D       240         CALL EHEXA         ; y lo escribe en hexadecimal
9D33  DD21999D     250         LD   IX,SPACIO
9D37  CD4F9D       260         CALL ESCR          ; Deja otro espacio
9D3A  3AA89D       270         LD   A,(ALMACN)    ; Ahora lo escribe como un caracter
9D3D  CD5DBB       280         CALL #BB5D         ; incluyendo codigos de control
9D40  3E0D         290         LD   A,13
9D42  CD5ABB       300         CALL #BB5A
9D45  3E0A         310         LD   A,10
9D47  CD5ABB       320         CALL #BB5A         ; Comienza en la siguiente linea
9D4A  E1           330         POP  HL
9D4B  23           340         INC  HL            ; Siguiente byte
9D4C  C1           350         POP  BC
9D4D  10C1         360         DJNZ OBUCLE        ; Han sido 20 bytes?
9D4F  DDE5         370 ESCR    PUSH IX
9D51  CD54BB       380         CALL #BB54
9D54  DDE1         390         POP  IX
9D56  DD7E00       400 BUCLE   LD   A,(IX)
9D59  FE00         410         CP   0
9D5B  C8           420         RET  Z
9D5C  DDE5         430         PUSH IX
9D5E  CD5DBB       440         CALL #BB5D
9D61  DDE1         450         POP  IX
9D63  DD23         460         INC  IX
9D65  18EF         470         JR   BUCLE
9D67  0600         480 EHEXA   LD   B,0            ; Estas rutinas han sido
9D69  4F           490         LD   C,A           ; anteriormente comentadas
9D6A  CB1F         500         RR   A
9D6C  CB1F         510         RR   A
9D6E  CB1F         520         RR   A
9D70  CB1F         530         RR   A
9D72  E60F         540 SCRIBO  AND  #0F
9D74  FE0A         550         CP   #0A
9D76  3008         560         JR   NC,PORC
```

```
9D78  C630        570         ADD  A,#30
9D7A  C5          580         PUSH BC
9D7B  CD5ABB      590         CALL #BB5A
9D7E  1806        600         JR   SALIDA
9D80  C637        610 PORC    ADD  A,#37
9D82  C5          620         PUSH BC
9D83  CD5ABB      630         CALL #BB5A
9D86  C1          640 SALIDA  POP  BC
9D87  78          650         LD   A,B
9D88  FE01        660         CP   1
9D8A  C8          670         RET  Z
9D8B  79          680         LD   A,C
9D8C  0601        690         LD   B,1
9D8E  18E2        700         JR   SCRIBO
9D90  7C          710 EHEXHL  LD   A,H
9D91  CD679D      720         CALL EHEXA
9D94  7D          730         LD   A,L
9D95  CD679D      740         CALL EHEXA
9D98  C9          750         RET
9D99  2E2E2E2E    760 SPACIO  DEFM "............."
9DA7              770         DEFS 1
9DA8  00          780 ALMACN  DEFB 0
```

Comentarios

Existen aquí tres importantes rutinas ROM, dos de las cuales ya hemos visto antes, al examinar la rutina CARVAR en el capítulo 3. La rutina en &B900 pagina la ROM superior que contiene el intérprete de BASIC, y devuelve el estado de la ROM en el registro A.

Estrictamente hablando, estas tres rutinas no son rutinas *firmware* porque están en la RAM. Una segunda idea explicará el porqué es así. Hay un pequeño problema al tener las rutinas de paginación dentro de las ROM correspondientes. Si necesitas llamar una rutina de paginación y la ROM en cuestión no está paginada, ¡vas a tener problemas!

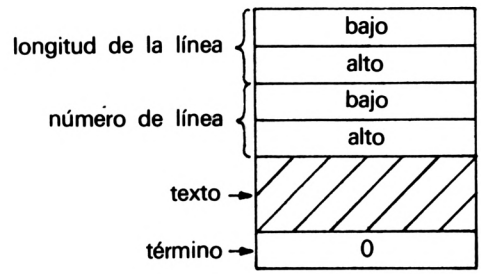
La rutina, cuando es llamada con una dirección de comienzo, escribe el carácter cuyo código ASCII está en la dirección examinada, incluso si el código ASCII leído en la dirección es el de un código de control. Podemos hacer esto usando la rutina &BB5D en vez de la &BB5A. Dado que LEEROM utiliza muchas subrutinas, es muy difícil relocalizarla sin la ayuda de un pequeño programa ensamblador. Los bytes del listado están preparados para la dirección 40200. Si decides relocalizar el programa, es importante conservarlo íntegramente dentro del área "depósito de memoria" de la RAM. De otro modo, cuando se paginen las ROM para tener acceso a su contenido, ¡parte de tu programa quedará inaccesible! No es una situación muy deseable.

La rutina puede, por supuesto, usarse para leer el contenido de la RAM que no esté superpuesta por la ROM.

La siguiente rutina se llama ENCUENTRA y se usa para escudriñar a través de un programa en BASIC y hallar textos o nombres de variables. Luego escribe el número de las líneas en donde se ha encontrado la cadena particular buscada. Antes de ver esta rutina, vamos a dar un vistazo al modo de almacenar en la memoria el texto de los programas BASIC y las posibles aplicaciones.

Estructura de una línea en BASIC

Una típica línea en BASIC es de la forma:



La longitud de la línea es el número de bytes de la línea completa, incluyendo los bytes de longitud de línea, de número de línea, y byte de término. El byte de término "0" actúa separando las líneas unas de otras. El final de un programa puede localizarse buscando a través del texto del programa una entrada de longitud de línea puesta a cero y una entrada de número de línea también a cero.

La alteración de las longitudes y números de línea, introduciéndolos en las posiciones adecuadas de la memoria mediante la orden POKE, puede resultar bastante interesante; sin embargo, también puede parecerte que pierdes un programa al introducir el número de línea con POKE. El programa sigue ahí, pero debes proporcionar la correcta longitud de línea. Un punto interesante es la posibilidad de reproducir un programa ilegible, y ejecutarlo aun así. Aunque eso ya es desviarse un poco del tema. El texto del programa en BASIC comienza en la dirección 368, con el byte bajo de la longitud de línea de la primera línea del programa.

Hay que tener en cuenta que, a diferencia de otros aparatos, la primera línea con una sentencia REM no es el lugar más apropiado para guardar el código máquina, principalmente debido a que estaría en el área de la RAM superpuesta por la ROM. Ello aumentaría los problemas si la ROM fuera a ser paginada.

En el texto actual de la línea, las palabras clave del BASIC y las funciones se almacenan como bytes individuales con valores entre 128 y 255, denominados PALABRA CLAVE *(TOKENS)*. Por ejemplo, ABS toma el valor 255, AFTER es 128 y REM es 197. El texto tras una señal REM, o una cadena entre comillas (" ") asignada a una sentencia PRINT, se almacena como una secuencia de códigos ASCII. Todas las asignaciones en una línea de programa, como:

```
100 a$="paco"
```

son bastante interesantes; si la dirección de a$ se introduce en una rutina en código máquina, usando CALL dirección,@a$, el valor dado como

dirección estará en el cuerpo del programa, en la línea 100. La cuestión de interés es que, si modificas la cadena dentro de la rutina en código máquina, la línea 100 se modificará. Puedes ver esto en acción en el programa INTROCAD. La dirección de cualquier variable puede obtenerse sólo con teclear:

```
PRINT  @nombre-variable
```

No tienes que incluirlo como parte de la sentencia CALL @; puede usarse independientemente. Sin embargo, informaciones tales como la dirección de variables son de mínimo uso en programas prácticos, dado que CALL te soluciona todos los detalles.

Lo que realmente nos interesa respecto a ENCUENTRA es el modo en que los nombres de las variables actuales son almacenados en las líneas del programa. Los nombres de variables se almacenan en ASCII, añadiendo 128 al código ASCII del último carácter. Así, para nombres de una sola letra, la letra tiene 128 añadido a su código ASCII. Cuando hablo de último carácter me refiero a la última letra del nombre de la variable, NO al "identificador del tipo", si es que existe. Así el nombre de la variable "paco" se almacena así:

```
p     112
a      97
c      99
o     239    (111+128)
```

Para lo que nos concierne, esto es lo único que necesitamos saber sobre almacenamiento de nombres de variables. Probablemente encontrarás un serio problema a la hora de buscar nombres de variables en programas BASIC para Amstrad; el último carácter, con el valor 128 añadido a su código ASCII, tiene ahora un código en el mismo intervalo que las PALABRAS CLAVE BASIC. Esto puede acarrear problemas, y de hecho los produce a veces con ENCUENTRA, en particular cuando el nombre de la variable que buscamos tiene sólo una letra. La rutina puede encontrar ocasionalmente, buscando nombres de variable con una sola letra, líneas cuyo contenido es una determinada PALABRA CLAVE de BASIC en lugar de la variable. Sin embargo, con nombres más largos no hay problemas.

ENCUENTRA

Una rutina para encontrar textos o nombres de variables en el cuerpo de un programa BASIC. Los números de línea que se refieren a la posición del texto buscado o al nombre de la variable se imprimen en pantalla.

Requisitos de entrada: Desde BASIC: CALL dirección,@a$, modo, donde "modo" define si la búsqueda es en torno a un nombre de variable (con el último carácter adecuadamente modificado) cuando modo es =0, o en torno a un texto normal cuando modo =1. Cuando modo =1 el último carácter de la cadena buscada en a$ no es modificado, por lo que los nombres de variables no son localizados. Me remito a los Comentarios para más detalles.

Condiciones de salida: Irrelevantes.

Longitud: 334 bytes.

```
                    10 ;       ** ENCUENTRA **
A410                20         ORG  42000
A410  FE02          30         CP   2
A412  205A          40         JR   NZ,PARAMS
A414  DD6E02        50         LD   L,(IX+2)
A417  DD6603        60         LD   H,(IX+3)
A41A  7E            70         LD   A,(HL)
A41B  3264A5        80         LD   (LONG),A    ; Descriptor de anchura
A41E  23            90         INC  HL
A41F  4E           100         LD   C,(HL)
A420  23           110         INC  HL
A421  46           120         LD   B,(HL)
A422  ED4362A5     130         LD   (CADENA),BC; Direccion de la cadena
A426  DD7E00       140         LD   A,(IX+0)
A429  FE01         150         CP   1
A42B  2817         160         JR   Z,OK        ; Si se trata de texto normal,
                   170 ;                          comienza directamente
A42D  3A64A5       180         LD   A,(LONG)    ; En caso contrario, modifica
A430  5F           190         LD   E,A         ; el ultimo caracter del nombre
A431  1600         200         LD   D,0         ; de la variable sumandole 128
A433  1B           210         DEC  DE          ; y almacenandolo despues
A434  2A62A5       220         LD   HL,(CADENA)
A437  19           230         ADD  HL,DE
A438  7E           240         LD   A,(HL)
A439  C680         250         ADD  A,128
A43B  77           260         LD   (HL),A      ; en la memoria
A43C  3A64A5       270         LD   A,(LONG)
A43F  FE01         280         CP   1           ; Si el nombre de la variable
A441  CCC8A4       290         CALL Z,PROBLS    ; lo compone un solo caracter,
A444  DD217001     300 OK      LD   IX,368      ; se escribe un mensaje
                   310 ;                          al inicio del programa
A448  DD5E00       320 BUCLE   LD   E,(IX+0)
A44B  DD5601       330         LD   D,(IX+1)    ; DE = longitud de la linea
A44E  DD6E02       340         LD   L,(IX+2)    ; HL = numero de linea
A451  DD6603       350         LD   H,(IX+3)
A454  2260A5       360         LD   (LINEA),HL
A457  7D           370         LD   A,L
A458  B4           380         OR   H
A459  2875         390         JR   Z,FINAL     ; Terminaremos cuando
                   400 ;                          lleguemos a la linea cero
A45B  CD1BBB       410         CALL #BB1B
A45E  3870         420         JR   C,FINAL     ; Si se pulsa una tecla
                   430 ;                          tambien finalizara
A460  CD70A4       440         CALL BLINEA      ; Examina la linea
A463  DDE5         450         PUSH IX
A465  E1           460         POP  HL
A466  1B           470         DEC  DE
A467  19           480         ADD  HL,DE       ; Actualiza HL para que apunte
A468  23           490         INC  HL          ; al comienzo de la siguiente linea
A469  E5           500         PUSH HL          ;  Lo transfiere a IX
```

```
A46A  DDE1        510         POP   IX
A46C  18DA        520         JR    BUCLE       ; Vuelve otra vez
A46E  1850        530 PARAMS  JR    PARAM2      ; Se necesita para un
                  540 ;                           salto relativo
A470  DDE5        550 BLINEA  PUSH  IX
A472  E1          560         POP   HL          ; Comienzo de la linea en HL
A473  DDE5        570         PUSH  IX
A475  D5          580         PUSH  DE
A476  C1          590         POP   BC          ; Longitud en BC
A477  0B          600         DEC   BC
A478  0B          610         DEC   BC
A479  0B          620         DEC   BC          ; Reduce la linea en cuatro bytes
A47A  0B          630         DEC   BC          ; de forma que solo
                  640 ;                           se examine el texto
A47B  D5          650         PUSH  DE
A47C  110400      660         LD    DE,04
A47F  19          670         ADD   HL,DE       ; Ahora HL apunta hacia
                  680 ;                           el comienzo del texto
A480  ED5B62A5    690 BUCLE1  LD    DE,(CADENA);Direccion de la cadena en DE
A484  C5          700         PUSH  BC          ; Guarda la longitud de la linea
A485  1A          710         LD    A,(DE)
A486  BE          720         CP    (HL)
A487  280B        730         JR    Z,SI        ; Si hay dos caracteres, adelante
A489  C1          740 NOBIEN  POP   BC          ; Decrementa el contador,
A48A  0B          750         DEC   BC          ; de forma que apunte
A48B  23          760         INC   HL          ; hacia el siguiente caracter
A48C  78          770         LD    A,B
A48D  B1          780         OR    C
A48E  20F0        790         JR    NZ,BUCLE1   ; Continuara hasta que finalice
                  800 ;                           la linea
A490  D1          810         POP   DE
A491  DDE1        820         POP   IX          ; Restaura los registros
A493  C9          830         RET
A494  3A64A5      840 SI      LD    A,(LONG)
A497  47          850         LD    B,A
A498  1A          860 BUCLE2  LD    A,(DE)      ; Examina el resto para ver
A499  BE          870         CP    (HL)        ; si puede continuar
A49A  20ED        880         JR    NZ,NOBIEN   ; En caso negativo, regresa
A49C  23          890         INC   HL
A49D  13          900         INC   DE
A49E  10F8        910         DJNZ  BUCLE2
A4A0  CDA5A4      920         CALL  VISTO       ; Salta para escribir
                  930 ;                           lo que encuentra
A4A3  18E4        940         JR    NOBIEN
A4A5  3E0A        950 VISTO   LD    A,10
A4A7  CD5ABB      960         CALL  #BB5A
A4AA  3E0D        970         LD    A,13
A4AC  CD5ABB      980         CALL  #BB5A       ; Retorno de carro
A4AF  DDE5        990         PUSH  IX
A4B1  E5         1000         PUSH  HL
A4B2  D5         1010         PUSH  DE
A4B3  C5         1020         PUSH  BC
A4B4  2A60A5     1030         LD    HL,(LINEA)
A4B7  CDF8A4     1040         CALL  PDECHL
A4BA  C1         1050         POP   BC
A4BB  D1         1060         POP   DE
A4BC  E1         1070         POP   HL
A4BD  DDE1       1080         POP   IX
A4BF  C9         1090         RET
A4C0  DD214BA5   1100 PARAM2  LD    IX,PARA
A4C4  CDD1A4     1110         CALL  ESCR
A4C7  C9         1120         RET
A4C8  DD2126A5   1130 PROBLS  LD    IX,TEXTO1
A4CC  CDD1A4     1140         CALL  ESCR
A4CF  C9         1150         RET
A4D0  C9         1160 FINAL   RET
A4D1  DDE5       1170 ESCR    PUSH  IX        ; Rutinas anteriormente documentadas
A4D3  CD54BB     1180         CALL  #BB54
A4D6  3E0A       1190         LD    A,10
A4D8  CD5ABB     1200         CALL  #BB5A
```

```
A4DB  3E0D      1210          LD   A,13
A4DD  CD5ABB    1220          CALL #BB5A
A4E0  3E07      1230          LD   A,7
A4E2  CD5ABB    1240          CALL #BB5A
A4E5  DDE1      1250          POP  IX
A4E7  DD7E00    1260 BUCLES   LD   A,(IX)
A4EA  FE00      1270          CP   0
A4EC  C8        1280          RET  Z
A4ED  DDE5      1290          PUSH IX
A4EF  CD5DBB    1300          CALL #BB5D
A4F2  DDE1      1310          POP  IX
A4F4  DD23      1320          INC  IX
A4F6  18EF      1330          JR   BUCLES
A4F8  111027    1340 PDECHL   LD   DE,10000
A4FB  CD13A5    1350          CALL PDECH
A4FE  11E803    1360          LD   DE,1000
A501  CD13A5    1370          CALL PDECH
A504  116400    1380          LD   DE,100
A507  CD13A5    1390          CALL PDECH
A50A  110A00    1400          LD   DE,10
A50D  CD13A5    1410          CALL PDECH
A510  110100    1420          LD   DE,1
A513  AF        1430 PDECH    XOR  A
A514  37        1440 BUCLE4   SCF
A515  3F        1450          CCF
A516  ED52      1460          SBC  HL,DE
A518  3803      1470          JR   C,PDSAL
A51A  3C        1480          INC  A
A51B  18F7      1490          JR   BUCLE4
A51D  19        1500 PDSAL    ADD  HL,DE
A51E  C630      1510          ADD  A,#30
A520  E5        1520          PUSH HL
A521  CD5ABB    1530          CALL #BB5A
A524  E1        1540          POP  HL
A525  C9        1550          RET
A526  50756564  1560 TEXTO1   DEFM "Pueden aparecer resultados dispares!"
A54A            1570          DEFS 1
A54B  50617261  1580 PARA      DEFM "Parametros erroneos."
A55F            1590          DEFS 1
A560  0000      1600 LINEA     DEFW 0000
A562  0000      1610 CADENA    DEFW 0000
A564  00        1620 LONG      DEFB 00
```

Comentarios

Dado el extenso uso de subrutinas, este programa puede relocalizarse pero con dificultad. Todas las referencias a las direcciones de las subrutinas necesitan alterarse. Se han usado rutinas como EDECHL y ECADEN, que hemos visto anteriormente en este libro. Los bytes dados en el listado son para la dirección 42000. El programa es relativamente sencillo de manejar.

```
a$="paco" : CALL 42000,@a$,0
```

buscará el nombre de la variable "paco". No pongas el identificador del tipo cuando busques algo similar a "paco%" o "paco$". Quítalo sin más. ENCUENTRA propondrá todas las posibilidades del nombre "paco" de la variable, independientemente del tipo. Si intentas buscar el nombre de una variable de una sola letra, el programa te avisará de los extraños

resultados que puedes obtener. Durante una búsqueda, si quieres terminar, no tienes más que pulsar una tecla cualquiera. Con ello terminará la búsqueda.

```
a$="paco" : CALL 42000,@a$,1
```

buscará un fragmento de texto que contenga "paco" en su interior. Esto podría ser una sentencia PRINT o REM, o podría formar parte de un nombre de una variable, como, por ejemplo, "pacotilla"; "paco" está dentro, y será detectado por ENCUENTRA. Utilizando la rutina con modo=0, volverá al BASIC con el último carácter de a$ alterado, al habérsele añadido 128 a su código ASCII. Esto no ocurre cuando modo=1.

Hemos visto antes en este capítulo cómo el final de un programa en BASIC se indica mediante la aparición de la longitud de línea y del número de línea puestos a cero. La siguiente rutina, LONGITUD, aprovecha esta circunstancia para dar la longitud en bytes de un programa en BASIC.

LONGITUD

Escribe la longitud de un programa en BASIC.

Requisitos de entrada: CALL dirección.

Condiciones de salida: Irrelevantes.

Longitud: 93 bytes.

```
                    10 ;       ** LONGITUD **
A410                20         ORG  42000
A410  DD217001      30 OK      LD   IX,368    ; Comienzo del programa
A414  010000        40         LD   BC,0
A417  DD5E00        50 BUCLE   LD   E,(IX+0)
A41A  DD5601        60         LD   D,(IX+1) ; DE = longitud de la linea
A41D  DD6E02        70         LD   L,(IX+2) ; HL = numero de linea
A420  DD6603        80         LD   H,(IX+3)
A423  7D            90         LD   A,L
A424  B4           100         OR   H
A425  2812         110         JR   Z,FINAL  ; Si es cero, hemos terminado
A427  E5           120         PUSH HL
A428  C5           130         PUSH BC
A429  E1           140         POP  HL       ; El contenido de BC pasa a HL
A42A  19           150         ADD  HL,DE    ; Suma a HL la longitud de la linea
A42B  E5           160         PUSH HL
A42C  C1           170         POP  BC       ; Se actualiza BC
A42D  E1           180         POP  HL
A42E  DDE5         190         PUSH IX
A430  E1           200         POP  HL
A431  1B           210         DEC  DE
A432  19           220         ADD  HL,DE    ; Actualiza HL para que apunte
A433  23           230         INC  HL       ; al comienzo de la siguiente linea
A434  E5           240         PUSH HL       ;  Lo transfiere a IX
A435  DDE1         250         POP  IX
A437  18DE         260         JR   BUCLE    ; Vuelve otra vez
```

```
A439  C5        270 FINAL   PUSH BC
A43A  E1        280         POP  HL        ; BC pasa a HL
A43B  CD3FA4    290         CALL PDECHL    ; Lo escribe
A43E  C9        300         RET
A43F  111027    310 PDECHL  LD   DE,10000
A442  CD5AA4    320         CALL PDECH
A445  11E803    330         LD   DE,1000
A448  CD5AA4    340         CALL PDECH
A44B  116400    350         LD   DE,100
A44E  CD5AA4    360         CALL PDECH
A451  110A00    370         LD   DE,10
A454  CD5AA4    380         CALL PDECH
A457  110100    390         LD   DE,1
A45A  AF        400 PDECH   XOR  A
A45B  37        410 BUCLE4  SCF
A45C  3F        420         CCF
A45D  ED52      430         SBC  HL,DE
A45F  3803      440         JR   C,PDSAL
A461  3C        450         INC  A
A462  18F7      460         JR   BUCLE4
A464  19        470 PDSAL   ADD  HL,DE
A465  C630      480         ADD  A,#30
A467  E5        490         PUSH HL
A468  CD5ABB    500         CALL #BB5A
A46B  E1        510         POP  HL
A46C  C9        520         RET
```

Comentarios

La rutina es también algo complicada de relocalizar dado el uso de subrutinas. Los bytes del listado son para la dirección 42000. Para usar esta rutina, sólo tienes que llamarla cuando lo necesites (mediante CALL). La longitud del programa se escribirá en pantalla, y si no hay ningún programa en el ordenador, aparecerá un valor igual a cero.

Extensiones del sistema residente

El programa ensamblador originalmente utilizado en este libro para producir los listados estaba en un *chip* de ROM, y se invocaba tecleando la orden

```
|ASSEMBLE
```

La línea vertical, obtenida desde el teclado mediante SHIFT @, notifica al intérprete de BASIC que el texto siguiente debe considerarse como una orden extra, y que los detalles para el procesamiento de la orden los hallará en la RAM o en la ROM. Esta facilidad que nos permite añadir instrucciones como ésta a la estructura de las órdenes BASIC es lo que se llama extensión del sistema residente o RSX para abreviar. Fundamentalmente es un modo de llamar a las rutinas en código máquina por su nombre en vez de tener que recordar su dirección. Es clara su utilidad cuando en un programa quieras acceder a varias rutinas diferentes desde BASIC. Los parámetros pueden introducirse en rutinas RSX de modo

prácticamente idéntico a como se introducen en rutinas en código máquina a las que se accede mediante la orden CALL. Las órdenes RSX pueden transferir desde código máquina al BASIC valores de las variables utilizando la función "@"; una línea como:

```
|GET,@caracter%
```

podría, por ejemplo, esperar la pulsación de una tecla y devolver el código ASCII de la tecla pulsada como variable "carácter%". Una línea del tipo:

```
caracter% = |GET
```

no es posible. Por ello, las órdenes RSX no son verdaderas extensiones del BASIC, sino que se asemejan en mayor grado a las denominadas rutinas CALL. Sin embargo, ello no quita para que resulten de suma utilidad.

El intérprete de BASIC debe conocer la presencia de las órdenes RSX, y es una suerte que no tengamos más que llamar a una rutina del sistema operativo del *firmware* para ello. Para mostrar su funcionamiento añadiremos un par de órdenes RSX, |CLS y |BARRE.

|CLS borra la pantalla y restaura el color de encendido.

|BARRE inicia un proceso de interrupción hasta que el siguiente barrido de la pantalla haya concluido. Esto es útil para programas de gráficos, pues te ayuda a reducir los parpadeos.

Ninguna de estas órdenes acepta parámetros, pero enseguida añadiré una orden que lo consigue.

En el cuerpo del sistema RSX se encuentran dos tablas, la tabla de saltos y la tabla de nombres. Sólo te daré aquí la información necesaria para permitirte usar el sistema. Si quieres profundizar más, deberás consultar el manual técnico del *firmware* de Amsoft.

La tabla de saltos

Contiene las direcciones a las que las distintas órdenes RSX añadidas pasan el control. Las direcciones se almacenan como parte de una instrucción JP del Z-80.

```
COMIENZO-TABLA   DEFW nombres ; Direccion de la tabla de nombres.
                 JP   rutina1 ; Primera rutina.
                 JP   rutina2 ; Segunda rutina.
                 JP   rutina3 ; y asi sucesivamente...
```

La tabla de nombres la veremos enseguida. Se necesita una entrada a la tabla de saltos por cada orden RSX que añadas, y el orden de aparición es el mismo en que aparecen en la tabla de nombres RSX.

La tabla de nombres

Contiene los nombres actuales de las órdenes RSX que quieras añadir. La dirección inicial de esta tabla se almacena, con el byte bajo en primer lugar, en los dos primeros bytes de la tabla de saltos.

```
nombres     DEFM  nombre-de-la-orden1 ; nombre de rutina1  .
            DEFM  nombre-de-la-orden2 ; nombre de rutina2  .
            DEFM  nombre-de-la-orden3 ; nombre de rutina3  .
            DEFB  0                   ; Final de la tabla.
```

Hay un punto importante sobre esta tabla a tener en cuenta. Y es que el último carácter del nombre de cada orden tiene el valor 128 añadido a su código ASCII. Y otra cuestión adicional a considerar es el orden de las entradas; al encontrar "nombre de la orden 1" se producirá un salto a la "rutina 1" y así sucesivamente. Finalmente, todas las entradas a la tabla de nombres deberán estar en mayúsculas. Todas las órdenes RSX introducidas en el aparato se convierten en mayúsculas por el intérprete de BASIC, por lo que también debemos colocar los nombres de órdenes de la tabla en mayúsculas, recordando que el último carácter debe ser modificado. El intérprete requiere también 4 bytes para el espacio de trabajo, cuya ubicación en el depósito de memoria es indiferente. Una vez inicializadas, las tablas son "activadas" y la orden se añade a la estructura de órdenes, mediante una llamada a la dirección &BCD1 con la dirección del espacio de trabajo en HL y la dirección del comienzo de la tabla de saltos en BC. Esta llamada no deberá hacerse más de una vez para una misma tabla, pues de lo contrario parece ser que pueden producirse bloqueos.

Puedes añadir también otras tablas de órdenes, si quieres, una tras otra. ¡Es evidente que tendrás que procurar no poner el mismo nombre a dos o más órdenes RSX!

RSX1

Esta rutina añade dos órdenes RSX, |CLS y |BARRE.

Requisitos de entrada: CALL dirección, donde dirección es la dirección de la rutina activada. Sólo deberá llamarse una vez.

Condiciones de salida: Irrelevantes.

Longitud: 44 bytes, incluidas las tablas RSX.

```
                    10 ;     ** RSX1 **
9D08                20       ORG  40200
9D08  011F9D        30       LD   BC,TABLA
9D0B  21309D        40       LD   HL,TRABJO
9D0E  CDD1BC        50       CALL #BCD1  ; Inicializa la tabla
9D11  C9            60       RET
```

```
9D12  CD19BD      70 BARRE  CALL #BD19  ; Rutina que espera hasta el
                  80 ;                    siguiente barrido de pantalla
9D15  C9          90        RET
9D16  CD02BC     100 CLS    CALL #BC02  ; Rutina para CLS
9D19  3E0C       110        LD   A,12
9D1B  CD5ABB     120        CALL #BB5A
9D1E  C9         130        RET
9D1F  279D       140 TABLA  DEFW NOMBRS ; Direccion de la tabla de nombres
9D21  C3129D     150        JP   BARRE  ; Salto a barre
9D24  C3169D     160        JP   CLS    ; Salto a CLS
9D27  42415252   170 NOMBRS DEFM "BARR" ; BARRE con el ultimo caracter
                  180 ;                   modificado
9D2B  C5         190        DEFB 197
9D2C  434C       200        DEFM "CL"   ; CLS con el ultimo caracter modificado
9D2E  D3         210        DEFB 211
9D2F  00         220        DEFB 0      ; Termino
9D30             230 TRABJO DEFS 4      ; Espacio de trabajo
```

Comentarios

Una vez inicializadas las tablas mediante la rutina adecuada, las dos órdenes |BARRE y |CLS estarán en condiciones de usarse.

La introducción de parámetros en las rutinas RSX es fácil. Es prácticamente idéntica a la introducción de rutinas mediante la orden CALL. Al comenzar una de las rutinas RSX, el registro IX apunta a un bloque de parámetros y A contiene el número de parámetros transferidos con la orden RSX. La colocación de los parámetros en el bloque de parámetros es la misma que para la sentencia CALL. La siguiente rutina, |PAUSA, n, demuestra todo esto.

PAUSA

Una orden RSX que produce un retardo de unos n/50 segundos. Realiza esta función mediante repetidas esperas del siguiente barrido de la pantalla, siendo éste un proceso que se produce cada cincuentavo de segundo. El retardo también puede finalizar al pulsar una tecla.

Requisitos de entrada: |PAUSA, n, donde n es el retardo en 1/50 de segundo entre 1 y 65535.

Condiciones de salida: Irrelevantes.

Longitud: 105 bytes.

```
                  10 ;       ** PAUSA **
9D08              20         ORG  40200
9D08  01509D      30         LD   BC,TABLA
9D0B  215B9D      40         LD   HL,TRABJO
9D0E  CDD1BC      50         CALL #BCD1      ; Inicializa la tabla
9D11  C9          60         RET
9D12  FE01        70 PAUSA   CP   1
9D14  2018        80         JR   NZ,ERROR   ; Error si hay mas de un parametro
9D16  DD4E00      90         LD   C,(IX)
```

```
9D19  DD4601      100         LD   B,(IX+1)  ; Duracion de la pausa en BC
9D1C  C5          110 BUCLE   PUSH BC
9D1D  CD19BD      120         CALL #BD19     ; Pausa
9D20  CD1BBB      130         CALL #BB1B     ; Alguna tecla apretada?
9D23  3807        140         JR   C,FINAL   ; Si asi es, salida
9D25  C1          150         POP  BC        ; En caso contrario...
9D26  0B          160         DEC  BC        ; decrementa BC y
9D27  78          170         LD   A,B
9D28  B1          180         OR   C
9D29  20F1        190         JR   NZ,BUCLE  ; si todavia no es cero, repite
9D2B  C9          200         RET  ;           Todo realizado, final
9D2C  C1          210 FINAL   POP  BC        ; Vendra aqui si se pulsa una tecla
9D2D  C9          220         RET
9D2E  DD215F9D    230 ERROR   LD   IX,TEXTO1 ; Escribe un mensaje de error
9D32  3E07        240         LD   A,7
9D34  CD5ABB      250         CALL #BB5A
9D37  DD7E00      260 BUCLE1  LD   A,(IX)
9D3A  FE00        270         CP   0
9D3C  2807        280         JR   Z,HECHO
9D3E  CD5ABB      290         CALL #BB5A
9D41  DD23        300         INC  IX
9D43  18F2        310         JR   BUCLE1
9D45  3E0D        320 HECHO   LD   A,13
9D47  CD5ABB      330         CALL #BB5A
9D4A  3E0A        340         LD   A,10
9D4C  CD5ABB      350         CALL #BB5A
9D4F  C9          360         RET
9D50  559D        370 TABLA   DEFW NOMBRS    ; Direccion de la tabla de nombres
9D52  C3129D      380         JP   PAUSA
9D55  50415553    390 NOMBRS  DEFM "PAUS"    ; PAUSA con el ultimo caracter
                  400 ;                        modificado
9D59  C5          410         DEFB 197
9D5A  00          420         DEFB 0         ; Termino
9D5B              430 TRABJO  DEFS 4         ; Espacio de trabajo
9D5F  50617261    440 TEXTO1  DEFM "Parametros erroneos."
9D73              450         DEFS 1
```

Comentarios

Una instrucción como

```
|PAUSA,50
```

producirá un retardo de un segundo una vez que la orden haya sido activada accediendo a la rutina. La pulsación de una tecla durante este tiempo también provocará el término de la rutina. Prueba con:

```
P=TIME:|PAUSA,50:PRINT TIME-P
```

que transforma la duración de una pausa en 1/300 segundos.

APENDICES

1. Efectos de los códigos de control

Código	EFECTO
0	Sin efecto.
1	Necesita un parámetro comprendido entre 0 y 255. Expone el símbolo que corresponde al valor del parámetro. Permite mostrar los símbolos asociados con los caracteres de la gama 0 a 31, en vez de que sean tratados como códigos de control.
2	Vuelve invisible el cursor de texto.
3	Vuelve visible el cursor de texto.
4	Un parámetro que será el modo de pantalla. Así PRINT CHR$(4)+CHR$(1) activará el modo 1.
5	Un parámetro entre 0 y 255. El parámetro es el código ASCII del carácter que quieres que se exponga en la posición del cursor de gráficos.
6	Activa la pantalla de texto.
7	Produce un pitido.
8	Retrocede el cursor una posición.
9	Avanza el cursor una posición.
10	Mueve el cursor una línea hacia abajo.
11	Mueve el cursor una línea hacia arriba.
12	Limpia la ventana de texto.
13	Desplaza el cursor al borde izquierdo de la línea en que se encuentre.

Código	EFECTO
14	Un parámetro, que es considerado como el número de tinta del papel.
15	Un parámetro, que es considerado como el número de tinta de la pluma.
16	Borra el carácter que se encuentre en la misma posición del cursor de texto. (Igual que CLR.)
17	Limpia la ventana desde la posición actual del cursor hasta el borde izquierdo.
18	Limpia la ventana desde la posición actual del cursor hasta el borde derecho.
19	Limpia la ventana desde el principio hasta la posición actual del cursor.
20	Limpia la ventana desde la posición del cursor hasta el final de la ventana.

Los códigos de control 16-20 limpian la posición actual del cursor, así como el resto. El espacio borrado se rellena con el color asignado al papel de textos.

21	Desactiva la pantalla de texto.
22	Un parámetro; 1 activa la opción de transparencia y 0 la desactiva.
23	Un parámetro que fija el modo de gráficos. 1 modo XOR 2 modo AND 3 modo OR 0 modo absoluto o forzado.
24	Intercambia la tinta del papel y pluma.
25	9 parámetros. Es el equivalente a la orden SYMBOL. El primer parámetro es el código ASCII del símbolo a definir. El resto de parámetros son las definiciones de cada una de las líneas del carácter. He descubierto que este código provoca a veces ciertos problemas.
26	Equivale a la orden WINDOW. Tiene 4 parámetros. Los primeros 2 parámetros especifican los bordes izquierdo y derecho de la ventana. No importa el orden en que introduzcas estos parámetros, ya que el menor es siempre considerado como borde izquierdo. Los siguientes 2 parámetros son las filas superior e inferior de la ventana. Se considerará al valor más pequeño como borde superior y al otro como borde inferior.
27	Sin efecto.
28	3 parámetros. Elige como tinta una pareja de colores. El primer parámetro es el número de la tinta; los 2 segundos indican los colores.

Código	EFECTO
29	Dos parámetros. Equivale a la orden BORDER. Los dos parámetros especifican los dos colores.
30	Desplaza el cursor a la esquina superior izquierda de la ventana.
31	Dos parámetros. Equivale a la orden LOCATE. Sitúa el cursor de texto en la posición X,Y, donde X es el primer parámetro e Y es el segundo.

Todos estos códigos de control pueden ser transferidos a la rutina PCONTR o a la que permanece en la dirección &BB5A. Observa que la rutina del *firmware* situada en &BB5D no actúa con estos códigos de control, sino que escribe el símbolo asociado a ellos.

2. Instrucciones y códigos operativos

d = desplazamiento - nn = número 0-65535 - n = número 0-255

CODIGO OBJETO	INSTRUCCION FUENTE		CODIGO OBJETO	INSTRUCCION FUENTE		CODIGO OBJETO	INSTRUCCION FUENTE	
8E	ADC	A,(HL)	E620	AND	n	CB63	BIT	4,E
DD8E05	ADC	A,(IX+d)	CB46	BIT	0,(HL)	CB64	BIT	4,H
FD8E05	ADC	A,(IY+d)	DDCB0546	BIT	0,(IX+d)	CB65	BIT	4,L
8F	ADC	A,A	FDCB0546	BIT	0,(IY+d)	CB6E	BIT	5,(HL)
88	ADC	A,B	CB47	BIT	0,A	DDCB056E	BIT	5,(IX+d)
89	ADC	A,C	CB40	BIT	0,B	FDCB056E	BIT	5,(IY+d)
8A	ADC	A,D	CB41	BIT	0,C	CB6F	BIT	5,A
8B	ADC	A,E	CB42	BIT	0,D	CB68	BIT	5,B
8C	ADC	A,H	CB43	BIT	0,E	CB69	BIT	5,C
8D	ADC	A,L	CB44	BIT	0,H	CB6A	BIT	5,D
CE20	ADC	A,n	CB45	BIT	0,L	CB6B	BIT	5,E
ED4A	ADC	HL,BC	CB4E	BIT	1,(HL)	CB6C	BIT	5,H
ED5A	ADC	HL,DE	DDCB054E	BIT	1,(IX+d)	CB6D	BIT	5,L
ED6A	ADC	HL,HL	FDCB054E	BIT	1,(IY+d)	CB76	BIT	6,(HL)
ED7A	ADC	HL,SP	CB4F	BIT	1,A	DDCB0576	BIT	6,(IX+d)
86	ADD	A,(HL)	CB48	BIT	1,B	FDCB0576	BIT	6,(IY+d)
DD8605	ADD	A,(IX+d)	CB49	BIT	1,C	CB77	BIT	6,A
FD8605	ADD	A,(IY+d)	CB4A	BIT	1,D	CB70	BIT	6,B
87	ADD	A,A	CB4B	BIT	1,E	CB71	BIT	6,C
80	ADD	A,B	CB4C	BIT	1,H	CB72	BIT	6,D
81	ADD	A,C	CB4D	BIT	1,L	CB73	BIT	6,E
82	ADD	A,D	CB56	BIT	2,(HL)	CB74	BIT	6,H
83	ADD	A,E	DDCB0556	BIT	2,(IX+d)	CB75	BIT	6,L
84	ADD	A,H	FDCB0556	BIT	2,(IY+d)	CB7E	BIT	7,(HL)
85	ADD	A,L	CB57	BIT	2,A	DDCB057E	BIT	7,(IX+d)
C620	ADD	A,n	CB50	BIT	2,B	FDCB057E	BIT	7,(IY+d)
09	ADD	HL,BC	CB51	BIT	2,C	CB7F	BIT	7,A
19	ADD	HL,DE	CB52	BIT	2,D	CB78	BIT	7,B
29	ADD	HL,HL	CB53	BIT	2,E	CB79	BIT	7,C
39	ADD	HL,SP	CB54	BIT	2,H	CB7A	BIT	7,D
DD09	ADD	IX,BC	CB55	BIT	2,L	CB7B	BIT	7,E
DD19	ADD	IX,DE	CB5E	BIT	3,(HL)	CB7C	BIT	7,H
DD29	ADD	IX,IX	DDCB055E	BIT	3,(IX+d)	CB7D	BIT	7,L
DD39	ADD	IX,SP	FDCB055E	BIT	3,(IY+d)	DC8405	CALL	C,nn
FD09	ADD	IY,BC	CB5F	BIT	3,A	FC8405	CALL	M,nn
FD19	ADD	IY,DE	CB58	BIT	3,B	D48405	CALL	NC,nn
FD29	ADD	IY,IY	CB59	BIT	3,C	C48405	CALL	NZ,nn
FD39	ADD	IY,SP	CB5A	BIT	3,D	F48405	CALL	P,nn
A6	AND	(HL)	CB5B	BIT	3,E	EC8405	CALL	PE,nn
DDA605	AND	(IX+d)	CB5C	BIT	3,H	E48405	CALL	PO,nn
FDA605	AND	(IY+d)	CB5D	BIT	3,L	CC8405	CALL	Z,nn
A7	AND	A	CB66	BIT	4,(HL)	CD8405	CALL	nn
A0	AND	B	DDCB0566	BIT	4,(IX+d)	3F	CCF	
A1	AND	C	FDCB0566	BIT	4,(IY+d)	BE	CP	(HL)
A2	AND	D	CB67	BIT	4,A	DDBE05	CP	(IX+d)
A3	AND	E	CB60	BIT	4,B	FDBE05	CP	(IY+d)
A4	AND	H	CB61	BIT	4,C	BF	CP	A
A5	AND	L	CB62	BIT	4,D	B8	CP	B

CODIGO OBJETO	INSTRUCCION FUENTE	
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
FE20	CP	n
EDA9	CPD	
EDB9	CPDR	
EDB1	CPIR	
EDA1	CPI	
2F	CPL	
27	DAA	
35	DEC	(HL)
DD3505	DEC	(IX+d)
FD3505	DEC	(IY+d)
3D	DEC	A
05	DEC	B
0B	DEC	BC
0D	DEC	C
15	DEC	D
1B	DEC	DE
1D	DEC	E
25	DEC	H
2B	DEC	HL
DD2B	DEC	IX
FD2B	DEC	IY
2D	DEC	L
3B	DEC	SP
F3	DI	
102E	DJNZ	e
FB	EI	
E3	EX	(SP),HL
DDE3	EX	(SP),IX
FDE3	EX	(SP),IY
08	EX	AF,AF'
EB	EX	DE,HL
D9	EXX	
76	HALT	
ED46	IM	0
ED56	IM	1
ED5E	IM	2
ED78	IN	A,(C)
ED40	IN	B,(C)
ED48	IN	C,(C)
ED50	IN	D,(C)
ED58	IN	E,(C)
ED60	IN	H,(C)
ED68	IN	L,(C)
34	INC	(HL)
DD3405	INC	(IX+d)
FD3405	INC	(IY+d)
3C	INC	A
04	INC	B
03	INC	BC
0C	INC	C
14	INC	D
13	INC	DE
1C	INC	E
24	INC	H
23	INC	HL
DD23	INC	IX
FD23	INC	IY

CODIGO OBJETO	INSTRUCCION FUENTE	
2C	INC	L
33	INC	SP
DB20	IN	A,(n)
EDAA	IND	
EDBA	INDR	
EDA2	INI	
EDB2	INIR	
C38405	JP	nn
E9	JP	(HL)
DDE9	JP	(IX)
FDE9	JP	(IY)
DA8405	JP	C,nn
FA8405	JP	M,nn
D28405	JP	NC,nn
C28405	JP	NZ,nn
F28405	JP	P,nn
EA8405	JP	PE,nn
E28405	JP	PO,nn
CA8405	JP	Z,nn
382E	JR	C,e
302E	JR	NC,e
202E	JR	NZ,e
282E	JR	Z,e
182E	JR	e
02	LD	(BC),A
12	LD	(DE),A
77	LD	(HL),A
70	LD	(HL),B
71	LD	(HL),C
72	LD	(HL),D
73	LD	(HL),E
74	LD	(HL),H
75	LD	(HL),L
3620	LD	(HL),n
DD7705	LD	(IX+d),A
DD7005	LD	(IX+d),B
DD7105	LD	(IX+d),C
DD7205	LD	(IX+d),D
DD7305	LD	(IX+d),E
DD7405	LD	(IX+d),H
DD7505	LD	(IX+d),L
DD360520	LD	(IX+d),n
FD7705	LD	(IY+d),A
FD7005	LD	(IY+d),B
FD7105	LD	(IY+d),C
FD7205	LD	(IY+d),D
FD7305	LD	(IY+d),E
FD7405	LD	(IY+d),H
FD7505	LD	(IY+d),L
FD360520	LD	(IY+d),n
328405	LD	(nn),A
ED438405	LD	(nn),BC
ED538405	LD	(nn),DE
228405	LD	(nn),HL
DD228405	LD	(nn),IX
FD228405	LD	(nn),IY
ED738405	LD	(nn),SP
0A	LD	A,(BC)
1A	LD	A,(DE)
7E	LD	A,(HL)
DD7E05	LD	A,(IX+d)
FD7E05	LD	A,(IY+d)

CODIGO OBJETO	INSTRUCCION FUENTE	
3A8405	LD	A,(nn)
7F	LD	A,A
78	LD	A,B
79	LD	A,C
7A	LD	A,D
7B	LD	A,E
7C	LD	A,H
ED57	LD	A,I
7D	LD	A,L
3E20	LD	A,n
ED5F	LD	A,R
46	LD	B,(HL)
DD4605	LD	B,(IX+d)
FD4605	LD	B,(IY+d)
47	LD	B,A
40	LD	B,B
41	LD	B,C
42	LD	B,D
43	LD	B,E
44	LD	B,H
45	LD	B,L
0620	LD	B,n
ED4B8405	LD	BC,(nn)
018405	LD	BC,nn
4E	LD	C,(HL)
DD4E05	LD	C,(IX+d)
FD4E05	LD	C,(IY+d)
4F	LD	C,A
48	LD	C,B
49	LD	C,C
4A	LD	C,D
4B	LD	C,E
4C	LD	C,H
4D	LD	C,L
0E20	LD	C,n
56	LD	D,(HL)
DD5605	LD	D,(IX+d)
FD5605	LD	D,(IY+d)
57	LD	D,A
50	LD	D,B
51	LD	D,C
52	LD	D,D
53	LD	D,E
54	LD	D,H
55	LD	D,L
1620	LD	D,n
ED5B8405	LD	DE,(nn)
118405	LD	DE,nn
5E	LD	E,(HL)
DD5E05	LD	E,(IX+d)
FD5E05	LD	E,(IY+d)
5F	LD	E,A
58	LD	E,B
59	LD	E,C
5A	LD	E,D
5B	LD	E,E
5C	LD	E,H
5D	LD	E,L
1E20	LD	E,n
66	LD	H,(HL)
DD6605	LD	H,(IX+d)
FD6605	LD	H,(IY+d)

CODIGO OBJETO	INSTRUCCION FUENTE	
67	LD	H,A
60	LD	H,B
61	LD	H,C
62	LD	H,D
63	LD	H,E
64	LD	H,H
65	LD	H,L
2620	LD	H,n
2A8405	LD	HL,(nn)
218405	LD	HL,nn
ED47	LD	I,A
DD2A8405	LD	IX,(nn)
DD218405	LD	IX,nn
FD2A8405	LD	IY,(nn)
FD218405	LD	IY,nn
6E	LD	L,(HL)
DD6E05	LD	L,(IX+d)
FD6E05	LD	L,(IY+d)
6F	LD	L,A
68	LD	L,B
69	LD	L,C
6A	LD	L,D
6B	LD	L,E
6C	LD	L,H
6D	LD	L,L
2E20	LD	L,n
ED4F	LD	R,A
ED7B8405	LD	SP,(nn)
F9	LD	SP,HL
DDF9	LD	SP,IX
FDF9	LD	SP,IY
318405	LD	SP,nn
EDA8	LDD	
EDB8	LDDR	
EDA0	LDI	
EDB0	LDIR	
ED44	NEG	
00	NOP	
B6	OR	(HL)
DDB605	OR	(IX+d)
FDB605	OR	(IY+d)
B7	OR	A
B0	OR	B
B1	OR	C
B2	OR	D
B3	OR	E
B4	OR	H
B5	OR	L
F620	OR	n
ED8B	OTDR	
EDB3	OTIR	
ED79	OUT	(C),A
ED41	OUT	(C),B
ED49	OUT	(C),C
ED51	OUT	(C),D
ED59	OUT	(C),E
ED61	OUT	(C),H
ED69	OUT	(C),L
D320	OUT	(n),A
EDAB	OUTD	
EDA3	OUTI	
F1	POP	AF
C1	POP	BC

CODIGO OBJETO	INSTRUCCION FUENTE	
D1	POP	DE
E1	POP	HL
DDE1	POP	IX
FDE1	POP	IY
F5	PUSH	AF
C5	PUSH	BC
D5	PUSH	DE
E5	PUSH	HL
DDE5	PUSH	IX
FDE5	PUSH	IY
CB86	RES	0,(HL)
DDCB0586	RES	0,(IX+d)
FDCB0586	RES	0,(IY+d)
CB87	RES	0,A
CB80	RES	0,B
CB81	RES	0,C
CB82	RES	0,D
CB83	RES	0,E
CB84	RES	0,H
CB85	RES	0,L
CB8E	RES	1,(HL)
DDCB058E	RES	1,(IX+d)
FDCB058E	RES	1,(IY+d)
CB8F	RES	1,A
CB88	RES	1,B
CB89	RES	1,C
CB8A	RES	1,D
CB8B	RES	1,E
CB8C	RES	1,H
CB8D	RES	1,L
CB96	RES	2,(HL)
DDCB0596	RES	2,(IX+d)
FDCB0596	RES	2,(IY+d)
CB97	RES	2,A
CB90	RES	2,B
CB91	RES	2,C
CB92	RES	2,D
CB93	RES	2,E
CB94	RES	2,H
CB95	RES	2,L
CB9E	RES	3,(HL)
DDCB059E	RES	3,(IX+d)
FDCB059E	RES	3,(IY+d)
CB9F	RES	3,A
CB98	RES	3,B
CB99	RES	3,C
CB9A	RES	3,D
CB9B	RES	3,E
CB9C	RES	3,H
CB9D	RES	3,L
CBA6	RES	4,(HL)
DDCB05A6	RES	4,(IX+d)
FDCB05A6	RES	4,(IY+d)
CBA7	RES	4,A
CBA0	RES	4,B
CBA1	RES	4,C
CBA2	RES	4,D
CBA3	RES	4,E
CBA4	RES	4,H
CBA5	RES	4,L
CBAE	RES	5,(HL)

CODIGO OBJETO	INSTRUCCION FUENTE	
DDCB05AE	RES	5,(IX+d)
FDCB05AE	RES	5,(IY+d)
CBAF	RES	5,A
CBA8	RES	5,B
CBA9	RES	5,C
CBAA	RES	5,D
CBAB	RES	5,E
CBAC	RES	5,H
CBAD	RES	5,L
CBB6	RES	6,(HL)
DDCB05B6	RES	6,(IX+d)
FDCB05B6	RES	6,(IY+d)
CBB7	RES	6,A
CBB0	RES	6,B
CBB1	RES	6,C
CBB2	RES	6,D
CBB3	RES	6,E
CBB4	RES	6,H
CBB5	RES	6,L
CBBE	RES	7,(HL)
DDCB05BE	RES	7,(IX+d)
FDCB05BE	RES	7,(IY+d)
CBBF	RES	7,A
CBB8	RES	7,B
CBB9	RES	7,C
CBBA	RES	7,D
CBBB	RES	7,E
CBBC	RES	7,H
CBBD	RES	7,L
C9	RET	
D8	RET	C
F8	RET	M
D0	RET	NC
C0	RET	NZ
F0	RET	P
E8	RET	PE
E0	RET	PO
C8	RET	Z
ED4D	RETI	
ED45	RETN	
CB16	RL	(HL)
DDCB0516	RL	(IX+d)
FDCB0516	RL	(IY+d)
CB17	RL	A
CB10	RL	B
CB11	RL	C
CB12	RL	D
CB13	RL	E
CB14	RL	H
CB15	RL	L
17	RLA	
CB06	RLC	(HL)
DDCB0506	RLC	(IX+d)
FDCB0506	RLC	(IY+d)
CB07	RLC	A
CB00	RLC	B
CB01	RLC	C
CB02	RLC	D
CB03	RLC	E
CB04	RLC	H
CB05	RLC	L

CODIGO OBJETO	INSTRUCCION FUENTE	
07	RLCA	
ED6F	RLD	
CB1E	RR	(HL)
DDCB051E	RR	(IX+d)
FDCB051E	RR	(IY+d)
CB1F	RR	A
CB18	RR	B
CB19	RR	C
CB1A	RR	D
CB1B	RR	E
CB1C	RR	H
CB1D	RR	L
1F	RRA	
CB0E	RRC	(HL)
DDCB050E	RRC	(IX+d)
FDCB050E	RRC	(IY+d)
CB0F	RRC	A
CB08	RRC	B
CB09	RRC	C
CB0A	RRC	D
CB0B	RRC	E
CB0C	RRC	H
CB0D	RRC	L
OF	RRCA	
ED67	RRD	
C7	RST	00H
CF	RST	08H
D7	RST	10H
DF	RST	18H
E7	RST	20H
EF	RST	28H
F7	RST	30H
FF	RST	38H
DE20	SBC	A,n
9E	SBC	A,(HL)
DD9E05	SBC	A,(IX+d)
FD9E05	SBC	A,(IY+d)
9F	SBC	A,A
98	SBC	A,B
99	SBC	A,C
9A	SBC	A,D
9B	SBC	A,E
9C	SBC	A,H
9D	SBC	A,L
ED42	SBC	HL,BC
ED52	SBC	HL,DE
ED62	SBC	HL,HL
ED72	SBC	HL,SP
37	SCF	
CBC6	SET	0,(HL)
DDCB05C6	SET	0,(IX+d)
FDCB05C6	SET	0,(IY+d)
CBC7	SET	0,A
CBC0	SET	0,B
CBC1	SET	0,C
CBC2	SET	0,D
CBC3	SET	0,E
CBC4	SET	0,H
CBC5	SET	0,L
CBCE	SET	1,(HL)
DDCB05CE	SET	1,(IX+d)
FDCB05CE	SET	1,(IY+d)
CBCF	SET	1,A
CBC8	SET	1,B
CBC9	SET	1,C
CBCA	SET	1,D
CBCB	SET	1,E
CBCC	SET	1,H
CBCD	SET	1,L
CBD6	SET	2,(HL)
DDCB05D6	SET	2,(IX+d)
FDCB05D6	SET	2,(IY+d)
CBD7	SET	2,A
CBD0	SET	2,B
CBD1	SET	2,C
CBD2	SET	2,D
CBD3	SET	2,E
CBD4	SET	2,H
CBD5	SET	2,L
CBD8	SET	3,B
CBDE	SET	3,(HL)
DDCB05DE	SET	3,(IX+d)
FDCB05DE	SET	3,(IY+d)
CBDF	SET	3,A
CBD9	SET	3,C
CBDA	SET	3,D
CBDB	SET	3,E
CBDC	SET	3,H
CBDD	SET	3,L
CBE6	SET	4,(HL)
DDCB05E6	SET	4,(IX+d)
FDCB05E6	SET	4,(IY+d)
CBE7	SET	4,A
CBE0	SET	4,B
CBE1	SET	4,C
CBE2	SET	4,D
CBE3	SET	4,E
CBE4	SET	4,H
CBE5	SET	4,L
CBEE	SET	5,(HL)
DDCB05EE	SET	5,(IX+d)
FDCB05EE	SET	5,(IY+d)
CBEF	SET	5,A
CBE8	SET	5,B
CBE9	SET	5,C
CBEA	SET	5,D
CBEB	SET	5,E
CBEC	SET	5,H
CBED	SET	5,L
CBF6	SET	6,(HL)
DDCB05F6	SET	6,(IX+d)
FDCB05F6	SET	6,(IY+d)
CBF7	SET	6,A
CBF0	SET	6,B
CBF1	SET	6,C
CBF2	SET	6,D
CBF3	SET	6,E
CBF4	SET	6,H
CBF5	SET	6,L
CBFE	SET	7,(HL)
DDCB05FE	SET	7,(IX+d)
FDCB05FE	SET	7,(IY+d)
CBFF	SET	7,A
CBF8	SET	7,B
CBF9	SET	7,C
CBFA	SET	7,D
CBFB	SET	7,E
CBFC	SET	7,H
CBFD	SET	7,L
CB26	SLA	(HL)
DDCB0526	SLA	(IX+d)
FDCB0526	SLA	(IY+d)
CB27	SLA	A
CB20	SLA	B
CB21	SLA	C
CB22	SLA	D
CB23	SLA	E
CB24	SLA	H
CB25	SLA	L
CB2E	SRA	(HL)
DDCB052E	SRA	(IX+d)
FDCB052E	SRA	(IY+d)
CB2F	SRA	A
CB28	SRA	B
CB29	SRA	C
CB2A	SRA	D
CB2B	SRA	E
CB2C	SRA	H
CB2D	SRA	L
CB3E	SRL	(HL)
DDCB053E	SRL	(IX+d)
FDCB053E	SRL	(IY+d)
CB3F	SRL	A
CB38	SRL	B
CB39	SRL	C
CB3A	SRL	D
CB3B	SRL	E
CB3C	SRL	H
CB3D	SRL	L
96	SUB	(HL)
DD9605	SUB	(IX+d)
FD9605	SUB	(IY+d)
97	SUB	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
D620	SUB	n
AE	XOR	(HL)
DDAE05	XOR	(IX+d)
FDAE05	XOR	(IY+d)
AF	XOR	A
A8	XOR	B
A9	XOR	C
AA	XOR	D
AB	XOR	E
AC	XOR	H
AD	XOR	L
EE20	XOR	n

(Cortesía de Zilog Inc.)

3. Efecto de las instrucciones sobre los *flags*

INSTRUCCION	C	Z	P/V	S	N	H	RESULTADOS
ADC HL, SS	#	#	V	#	0	X	Suma de 16 bits con acarreo.
ADX s, ADD s	#	#	V	#	0	#	Suma de 8 bits; suma con acarreo.
ADD DD, SS	#	—	—	—	0	X	Suma de 16 bits.
AND s	0	#	P	#	0	1	Operaciones lógicas.
BIT b, s	—	#	X	X	0	1	El estado del bit b de la dirección S es copiado en el *flag* Z.
CCF	#	—	—	—	0	X	Complementa el acarreo.
CPD, CPDR; CPI; CPIR	—	#	#	X	1	X	Instrucciones de búsqueda de bloques. Z=1 si A=(HL); en otro caso, Z=0. P/V=1 si BC≠0; en otro caso, P/V=0.
CP s	#	#	V	#	1	#	Compara el acumulador. (Registro A.)
CPL	—	—	—	—	1	1	Complementa el acumulador.
DAA	#	#	P	#	—	#	Convierte a BCD al acumulador.
DEC s	—	#	V	#	1	#	Decremento para 8 bits.
IN r, (C)	—	#	P	#	0	0	Entrada con direccionamiento indirecto.
INC s	—	#	V	#	0	#	Incremento para 8 bits.
IND; INI	—	#	X	X	1	X	Entrada de bloques: Z=0 si B≠0; en caso contrario, Z=1.
INDR INIR	—	1	X	X	1	X	Entrada de bloques: Z=0 si B≠0; en caso contrario, Z=1.
LD A,I; LD A,R	—	#	IFF	#	0	0	IFF: El contenido del *flip-flop* que habilita las interrupciones es copiado en el indicador P/V.
LDD, LDI	—	X	#	X	0	0	Instrucciones de transferencia de
LDDR; LDIR	—	X	0	X	0	0	bloques. P/V=1 si BC≠0; en caso contrario, P/V=0.
NEG	#	#	V	#	1	#	Niega el acumulador.
OR s	0	#	P	#	0	0	OR lógica al acumulador.
OTDR; OTIR	—	1	X	X	1	X	Salida de bloques: Z=0 si B≠0; de otra forma, Z=1.
OUTD; OUTI	—	#	X	X	1	X	Salida de bloques: Z=0 si B≠0; de otra forma, Z=1.
RLA; RLCA; RRA; RRCA	#	—	—	—	0	0	Rotación del acumulador.
RLD; RRD	—	#	P	#	0	0	Rotación de bits a izquierda y derecha.

RLS; RLCs; RRs; RRCs; SLAs; SRAs; SRLs	#	#	P	#	0	0	Rotación y desplazamiento de la dirección s.
SBC HL, SS	#	#	V	#	1	X	Diferencia con acarreo para 16 bits.
SCF	1	—	—	—	0	0	Activa el acarreo.
SBC s; SUB s	#	#	V	#	1	#	Diferencia con acarreo para 8 bits.
XOR s	0	#	P	#	0	#	OR-exclusivos del acumulador.

Símbolo	OPERACION
C	Indicador o *flag* de acarreo. C = 1 si la operación produce un acarreo del bit más significativo del operando o del resultando.
Z	Indicador de cero. Z = 1 si el resultado de la operación es cero.
S	Indicador de signo. S = 1 si el bit más significativo del resultado es uno, esto es, si es un número negativo.
P/V	*Flag* de paridad o de desbordamiento. El mismo indicador es compartido para la paridad (P) y para el desbordamiento (V). El *flag* se verá afectado como paridad con operaciones lógicas, mientras que las operaciones aritméticas lo afectan como indicador de desbordamiento del resultado. Cuando P/V contiene la paridad, P/V = 1 si el resultado de la operación es par; P/V = 0 si el resultado es impar. Cuando P/V contiene el desbordamiento, P/V = 1 si el resultado de la operación provoca un desbordamiento.
H	Indicador de medio acarreo. H = 1 si la operación de suma o resta provoca un acarreo del bit 4 del acumulador.
N	*Flag* de suma/resta. N = 1 si la operación anterior fue resta. H y N son utilizados conjuntamente con la instrucción (DAA) de ajuste a modo BCD, para obtener resultados apropiados a este formato BCD. Los operandos de la suma o resta también deben estar en el formato BCD.
#	El *flag* queda afectado de acuerdo con el resultado de la operación.
—	El indicador no es afectado por la operación.
0	El indicador es colocado a cero (=0).
1	El indicador es colocado a uno (=1).
X	Resultado desconocido para el indicador.
V	El indicador P/V es afectado de acuerdo con el desbordamiento del resultado de la operación.
r	Cualquiera de los registros de la CPU: A,B,C,D,E,H,L.
s	Cualquier posición de 8 bits para todos los modos de direccionamiento permitidos por ese tipo de instrucción.
SS	Cualquier posición de 16 bits para todos los modos de direccionamiento permitidos por esa instrucción.
R	Registro de refresco.
n	Número de 8 bits en la gama 0-255.
nn	Número de 16 bits en la gama 0-65535.

Indice alfabético

ANAYA MULTIMEDIA

Colección «MICROINFORMATICA»

Angell, I. O. y Jones, B. J.: DISEÑO DE GRAFICOS Y VIDEOJUEGOS (incluye cassette).
Beechhold, Henry F.: EL LIBRO DEL HARDWARE. No destape su ordenador personal... sin leer antes este libro.
Birmingham Educational Computing Centre: INTRODUCCION A LA TECNOLOGIA DE LA INFORMACION. PREINFORMATICA.
Bishop, Peter: PROGRAMACION AVANZADA EN BASIC.
Brown, Peter: PASCAL A PARTIR DEL BASIC.
Cavalcoli, Aldo: EL ORDENADOR PERSONAL: COMO ELEGIRLO Y UTILIZARLO.
Coccione, L. y Winter, G.: LOS ORDENADORES NO MUERDEN.
Dachslager, H., Hayashi, M. y Zucker, R.: PROGRAMACION EN BASIC: UN METODO PRACTICO.
Dewhirst, J. y Tennison, R.: TU PRIMER LIBRO DEL ZX SPECTRUM.
D'Opazo, J. y Grupo GOLEM: PROGRAMACION EN LOGO.
Durst, J.: «SPRITES» Y GRAFICOS EN LENGUAJE MAQUINA (ZX SPECTRUM).
Galende Domínguez, F.; Sánchez López, A.; Alfaraz López, M. y Sánchez García, J. A.: COMETAS EN TU MICRO: EL HALLEY. Cálculos de órbitas y parámetros de cometas en BASIC.
Gavin, Maurice: ASTRONOMIA: EL UNIVERSO EN TU ORDENADOR.
Gibbons, John P.: PROGRAMACION AVANZADA DEL COMMODORE 64. Ampliación del BASIC y rutinas gráficas.
Greenwood, Gareth: CODIGOS Y CLAVES SECRETAS. Criptografía en BASIC.
Hammond, R.: EL ORDENADOR Y TUS HIJOS.
Hartnell, Tim: EL LIBRO GIGANTE DE LOS JUEGOS PARA ORDENADOR.
Hartnell, Tim: INTELIGENCIA ARTIFICIAL: CONCEPTOS Y PROGRAMAS.
Hartnell, Tim: EL LIBRO GIGANTE DE LOS JUEGOS PARA ZX SPECTRUM.
Hartnell, Tim, y otros: EL LIBRO GIGANTE DE LOS JUEGOS PARA DRAGON.
Hartnell, Tim: EL SUPERLIBRO DE LOS JUEGOS PARA ORDENADOR.
Heller, R. S. y Martin, C. D.: BITS Y BYTES: INICIACION A LA INFORMATICA.
Hollerbach, Lew: MICROINFORMATICA: CONCEPTOS BASICOS.
Hurley, R.: JUEGOS GRAFICOS DE AVENTURA PARA ZX SPECTRUM.
Johnson, David: DESCUBRE LAS MATEMATICAS CON TU MICRO.
Johnston, J.: MICROS: TAMAÑOS, FORMAS Y SABORES.
Johnston, J.: MICROS: BIPS, PITIDOS Y LUCES.
Johnston, J.: MICROS: MENUS, BUCLES Y RATONES.
Kosniowski, Czes: MATEMATICAS DIVERTIDAS EN BASIC.
Kramer, S.: PROGRAMACION AVANZADA DEL ZX SPECTRUM.
Lacey, Andrew: LIBRO GIGANTE DE LOS JUEGOS PARA MSX.
Núñez, Agustín: PROGRAMACION DEL INTERFACE 1 Y MICRODRIVE.
Otero, M. A.; Pueyo, M. A. y Cajaraville, J. A.: PRIMEROS PASOS EN LOGO. El mundo de la tortuga Fan. Libro del profesor. Libro del alumno.
O'Shea, T. y Self, J.: ENSEÑANZA Y APRENDIZAJE CON ORDENADORES. Inteligencia artificial en educación.
Pentiraro, Egidio: EL ORDENADOR EN EL AULA.
Pritchard, Joe: DESCUBRE TU MSX. Programación y aplicaciones.
Pritchard, Joe: LENGUAJE MAQUINA MSX. Introducción y conceptos avanzados.
Rosso, Vincenzo de: COMO SE PROGRAMAN LOS ORDENADORES.
Sato, T.; Mapstone, P. y Muriel, I.: MSX: GUIA DEL PROGRAMADOR Y MANUAL DE REFERENCIA.
Servello, Fausto: ¿QUE ES LA TELEMATICA?
Snover, S. L. y Spikell, M. A.: JUEGOS MATEMATICOS DE INGENIO EN BASIC.
Thomasson, Don: PROGRAMACION AVANZADA DEL AMSTRAD. Entradas y salidas de la ROM.
Webb, David: LENGUAJE MAQUINA AVANZADO PARA ZX SPECTRUM.
Zaks, Rodnay: EL LIBRO DEL BASIC.

¡Dale a tu AMSTRAD CPC 464/664/6128 toda la potencia del lenguaje máquina sin esfuerzo!: RUTINAS EN LENGUAJE MAQUINA PARA AMSTRAD te facilita el acceso a las entrañas de tu microordenador.

Si estás empezando a profundizar en la programación de tu AMSTRAD, encontrarás en el libro multitud de rutinas máquina que producen efectos absolutamente sorprendentes y con las que podrás desarrollar fácilmente multitud de aplicaciones y programas de calidad profesional.

Si eres un programador habitual en código máquina, RUTINAS EN LENGUAJE MAQUINA PARA AMSTRAD reducirá a la décima parte el tiempo medio de desarrollo de programas, ya que encontrarás resueltas todas las rutinas habituales de manipulación de cadenas, operaciones de teclado, pantalla, etc.

A lo largo del libro encontrarás, además, rutinas e información exhaustiva sobre:

— Manipulación de caracteres y composición de gráficos.
— Rutinas *firmware* y *software* de desplazamiento de pantalla.
— Técnicas gráficas avanzadas: canje de pantallas, rellenado, caracteres multicolores.
— Programación del GPS, registros de sonido del *firmware*, efectos de sonido.
— Rutinas de manipulación del cassette.
— Ampliación del sistema residente.

RUTINAS EN CODIGO MAQUINA PARA AMSTRAD te llevará de la mano más allá de los límites del BASIC a los dominios del sistema operativo de tu ordenador, donde conseguirás la máxima rapidez y eficiencia para tus programas.

ANAYA MULTIMEDIA

Rutinas en Lenguaje Máquina para Amstrad

Joe Pritchard

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/