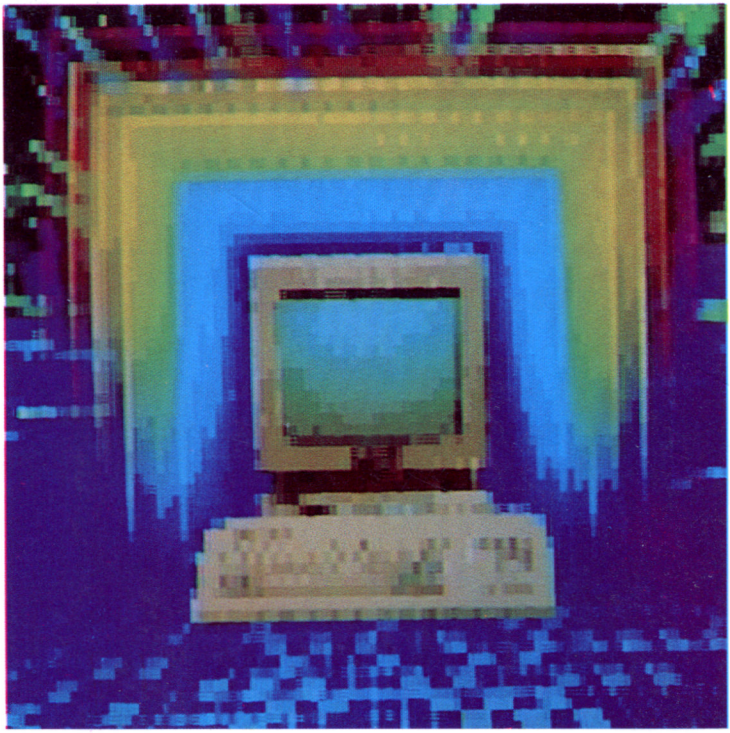
GRAN BIBLIOTECA

AMSTRAD



CÓDIGO MÁQUINA AVANZADO

EL LENGUAJE DEL ORDENADOR

EDICIONES INGELEK

GRAN BIBLIOTECA AMSTRAD

17

CÓDIGO MÁQUINA AVANZADO

Director editor:
Antonio M.ª Ferrer Abelló

Director de producción:
Vicente Robles

Director de la obra:
Fernando López Martínez

Redactor técnico:
Fernando López Martínez

Colaboradores:
L-H Servicios Informáticos
Pilar Manzanera Amaro

Diseño:
Bravo/Lofish

Maquetación:
Carlos González Amezúa

Dibujos:
José Ochoa

Fotografía:
Grupo Gálata

ISBN del tomo: 84-7708-062-3.
ISBN de la obra: 84-7708-004-6.
Fotocomposición: Andueza, S. A.
Imprime: Eurosur, S. A.
Depósito Legal: M. 19.826-1987.
Precio en Canarias, Ceuta y Melilla: 435 ptas.

Septiembre 1987

CÓDIGO MÁQUINA AVANZADO

INTRODUCCIÓN

ada programador encuentra en el código máquina un atractivo diferente: desde su rapidez hasta el refugio ideal para perfeccionados algoritmos que desea permanezcan en el anonimato. En todo caso, siempre es bienvenida cualquier información sobre este tema, aunque sea en forma de pequeñas «chispas», que nos permiten arrancar a nuestro ordenador un efecto curioso, algo que el simple usuario de BASIC desconoce.

Nos adentraremos ahora en el complejo mundo del código máquina, continuando la ardua tarea emprendida en el décimo volumen de esta GRAN BIBLIOTECA AMSTRAD. Desde la generación de las RSX (extensiones del sistema residentes), tan «temidas» como útiles, hasta una completa descripción de las rutinas del firmware.

Lamentablemente, la enorme carencia de información, tanto dentro como fuera de nuestras fronteras, sobre la disposición interna del PCW 8256/8512, nos hace relegar este modelo al ostracismo, dejando reducida la información sobre el mismo a aquella vertida en el número anterior de código máquina dentro de esta biblioteca, aplicable en tanto en cuanto se trata la programación general del microprocesador Z-80, común a las familias CPC y PCW.

En concreto, la información expuesta en este libro toma como base el CPC 464, dada la gran compatibilidad existente con él por parte del resto de los modelos de su gama. En realidad, como tendremos ocasión de comprobar, el sistema de acceso al firmware de los CPC, se estructura de manera que las alteraciones entre unos modelos y otros de la misma familia no afecten en la práctica al usuario, a excepción de la zona referida concretamente al BASIC, la cual no será objeto de tratamiento en este libro.

Así pues, la diferencia esencial entre unos modelos y otros estriba en la unidad de disco que soportan los sistemas 664 y 6128; el comentario de las rutinas de control de este periférico las podremos encontrar en el volumen dedicado específicamente a la unidad de disco, dentro de esta misma colección (LA UNIDAD DE DISCO. Una enciclopedia en 3 pulgadas).

COMENZANDO DESDE EL BASIC

os CPC son máquinas que en el momento de encenderlas se encuentran bajo el control del intérprete BASIC, así pues, cualquier intento de acceder al código máquina, pasa previamente de forma ineludible por este lenguaje. Por tanto, este primer capítulo lo dedicaremos a la relación entre el BASIC y el código máquina, tanto para la llamada a las subrutinas como para el intercambio de datos entre ambos, fase ésta de vital importancia, y que afortunadamente en los equipos Amstrad ha sido elegantemente tratada, a diferencia de la mayoría de los microordenadores presentes en el mercado, que adolecen de una casi completa «falta de comunicación» entre ambos lenguajes.

CALL: UN COMANDO SIN EXPLOTAR

Si nos dejamos guiar por las sucintas explicaciones que del comando **CALL** se nos ofrecen en el manual de consulta de nuestro Amstrad («Permite invocar desde BASIC una rutina escrita en código máquina»), pensaremos que sus posibilidades se reducen simplemente a indicar tras él una dirección de memoria, y el programa almacenado allí se ejecutará sin necesidad de nada más.

Esta es una forma de utilizarlo. Sin embargo, existe otra todavía más potente y versátil que nos permite establecer transferencias de información entre BASIC y código máquina, consistente en añadir una serie de parámetros adicionales:

CALL dirección, param1, param2, param3, ...

Comencemos por determinar a qué nos referimos bajo la denominación «parámetro». Pues bien, en principio podría tratarse de cualquier valor en el margen de 0 a 65535, de una variable entera (Z%, por ejemplo), o de una expresión, la cual una vez evaluada produzca un resultado entero. Son llamadas del tipo:

CALL dir, valor o CALL dir, variable%

donde valor, como hemos indicado, debe ser un número entero.

Si al efectuar una llamada a una rutina en código máquina empleamos una variable que no estuviera definida previamente, entonces se consideraría que su valor, por defecto, es 0.

Por el sistema anterior sólo conseguimos transferencias de información en una dirección: de BASIC a C/M. Por supuesto, mucho más interesante será, en muchos casos, poder establecerla en ambos sentidos. La manera de conseguirlo es anteponer al nombre de la variable entera el símbolo @ (enseguida comprobaremos su efecto). Su sintaxis sería:

var%=valor entero:CALL dir,@var%

También podemos enviar cadenas literales, aunque a diferencia de las expresiones enteras sólo es posible hacerlo si están previamente almacenadas en una variable alfanumérica, es decir, no nos está permitido efectuar llamadas del tipo CALL dir, «expresión literal». Esta siempre debe ir precedida del signo @, dentro de la instrucción CALL, por ejemplo:

Z$="TU MICRO":CALL dir,@Z$

Ahora el problema reside en recoger los parámetros transferidos al programa en C/M. Nada más fácil. Cuando se invoca una subrutina mediante **CALL,** el intérprete de BASIC se encarga de modificar el contenido de algunos registros, de forma que sea fácil acceder a ellos.

De esta manera, el acumulador (A) contiene el número de parámetros, mientras que el registro índice IX apunta a un área denomi-

nada «zona de parámetros», la cual emplea dos bytes para cada uno de los que fueron enviados, y cuyo significado depende del tipo de información manejada en la orden **CALL.**

LA ZONA DE PARÁMETROS

Observemos la figura. Como podemos comprobar, el par IX señala siempre al byte bajo del último parámetro enviado; si éste fue un número constante o una variable entera, allí precisamente está almacenado su valor.

En el caso de variables enteras precedidas del signo @, lo que encontramos en la zona de parámetros es la dirección de memoria en la cual BASIC almacenó su valor. Por ejemplo, la siguiente rutina analiza si la impresora está lista para imprimir, modificando el contenido de z% a un 1, si no está dispuesta a comenzar la impresión. La instrucción de llamada sería **z%=0:CALL 30000,@z%.**

```
ORG 30000
CALL #BD2E; MC BUSY PRINTER (FIRMWARE CPC)
RET NC
LD L,(IX+0)
LD H,(IX+1)
LD (HL),#01
RET
```

Cuando se trata de variables literales, interpretar la zona de parámetros es algo más complicado, pero no más difícil. Los dos bytes de esta última señalan hacia otra zona de memoria que podríamos denominar área de identificación de la cadena. En ella, el primer octeto contiene la longitud del literal transferido y los dos siguientes, en el orden bajo-alto, conforman otra dirección de la RAM donde finalmente el sistema operativo situó su contenido.

Vamos a desarrollar una pequeña rutina, la cual demuestre la forma de trabajar de la función **CALL,** a la vez de servirnos de ayuda para ilustrar el cálculo del tiempo de ejecución de una rutina en concreto. Se trata, por ejemplo, de realizar en código máquina un programa similar a lo que haría la instrucción **X%=45** de BASIC. Almacenaremos la rutina a partir de la posición 30000 y la ejecutaremos mediante **CALL 30000,45,@X%.**

En lenguaje ensamblador, todas las instrucciones de carga, como pudimos ver en la tabla correspondiente de los apéndices del volumen PRINCIPIOS EN CODIGO MAQUINA, tienen código mnemónico

LD. Sea, pues, nuestro programa almacenado a partir de la dirección 30000, el siguiente:

```
30000  DD 6E 00   LD L,(IX+0)
30003  DD 66 01   LD H,(IX+1)
30006  DD 7E 02   LD A,(IX+2)
30009  77         LD (HL),A
30010  C9         RET
```

Aunque a estas alturas no es estrictamente necesario, haremos una breve descripción del desarrollo de la rutina. Las dos primeras instrucciones almacenan en el par HL la dirección de memoria donde se encuentra el valor de la variable X%. La tercera instrucción recoge el número 45 en el acumulador, y la cuarta se encarga de transferirlo a la posición en la que el sistema operativo espera encontrar el valor para X%. Finalmente, mediante la instrucción RET se devuelve el control desde el C/M al BASIC.

Para almacenarla en la memoria del ordenador, podemos utilizar el cargador de código máquina que figura en este capítulo. Carguemos éste y añadamos la siguiente línea:

```
1000 DATA "DD6E00DD6601DD7E0277
C900000000000000000",52C
```

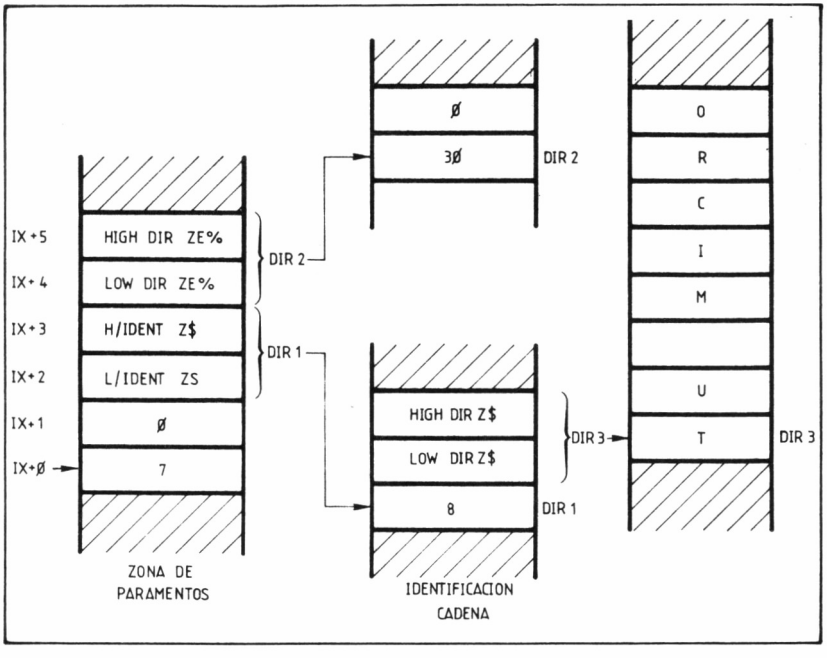
Ejecutemos el programa mediante RUN contestando a la pregunta sobre dirección de ubicación con 30000. Si el cargador indica algún error, repasemos los datos anteriores, hata que todo vaya bien.

Comprobemos antes de realizar la llamada al código máquina el valor de X% mediante PRINT X%. Normalmente debe ser 0, aunque podría ser cualquier otro si es que antes ha sido modificado. Tecleemos entonces, como comando directo CALL 30000,45,@X%. Tras ello, si volvemos a escribir PRINT X%, la respuesta del ordenador debe ser 45.

CARGADOR DE C/M

Hemos preparado un pequeño programa, el cual actuará de cargador hexadecimal para todas las rutinas en C/M que podamos crear gracias a la información presente en este libro.

La elección del sistema hexadecimal nos ha parecido la más adecuada, dada la extensa bibliografía existente que lo utiliza, en especial, como veremos, la literatura dedicada a la descripción de las ruti-



nas almacenadas en la ROM. Además, cuando nos enfrentamos con la ardua labor de teclear un programa en código máquina sin la ayuda de un ensamblador, este sistema ofrece frente al decimal la ventaja adicional de un ahorro considerable de pulsaciones.

Los códigos hexadecimales correspondientes a las instrucciones que configuren los programas deberán estar incluidos en las sentencias DATA que situaremos al final del cargador, y efectuando RUN, quedarán almacenados en la memoria de nuestro ordenador para su posterior ejecución. Dichas líneas deberán integrarse por dos bloques: uno con 20 bytes hexadecimales (completados con ceros si no llegaran a tal cantidad) de dos dígitos, seguidos sin espacio ni indicador de base alguno, en forma de cadena (entre comillas) y un último bloque de DATA, separado del primero por una coma, cuyo valor será la suma de los veinte bytes que constituyen el bloque anterior (también en hexadecimal). Gracias a este último dato, en caso de cometer algún error, se nos indicará la sentencia donde ocurrió, para modificarla, mediante un mensaje intermitente que aparecerá en la pantalla.

Al comienzo se nos preguntará sobre la posición de memoria donde deseamos introducir la rutina en C/M. En principio, toda la RAM

podría utilizarse, pero para evitar posibles conflictos con las Variables del Sistema y otras zonas delicadas de éste, elegiremos posiciones de memoria cercanas al RAMTOP (final del área de memoria destinada al BASIC) donde ejecutar nuestros programas, salvo que se indique lo contrario.

Una vez almacenado el C/M es posible grabarlo utilizando el comando BASIC **SAVE "Nombre",B, dirección de inicio, longitud,** siendo dirección de inicio la elegida por nosotros previamente, y longitud el número de bytes ocupados por la rutina (al finalizar la ejecución del programa cargador se nos informará también de este extremo en la pantalla).

A final del capítulo, encontraremos el listado del programa cargador. Tecleémoslo cuidadosamente y grabémoslo antes de ponerlo en funcionamiento. Aunque los poseedores de un ensamblador encontrarán más cómoda su utilización, la presencia de este cargador hará posible el uso de las rutinas a los que carezcan de tal aplicación.

Código Máquina Dentro Del Basic

La gran mayoría de los programadores no profesionales utilizan la codificación mixta BASIC-C/M en sus programas, aprovechando las condiciones de indudable comodidad que les reporta el lenguaje de alto nivel y la rapidez del de bajo. Es por ello, que en algunas ocasiones, fundamentalmente en el supuesto de implementación de rutinas de escasa longitud, se encontrará de utilidad poder cargar el C/M al mismo tiempo que el propio BASIC, es decir, dentro de su área de texto, para no tener que recurrir posteriormente a la carga de la rutina como bloque de memoria independiente.

A tal fin, existen dos sistemas posibles. El primero de ellos (quizá el más burdo y evidente), consiste en incluir el código máquina en líneas DATA, que serán volcadas en el lugar que deseemos de la memoria en el momento de ejecutar el programa. Las características de este método son la necesidad de un tiempo para la ubicación de la rutina en la memoria y la accesibilidad de ésta para un posible análisis o modificación desde el propio editor de BASIC.

El segundo sistema consiste en utilizar una línea REM para albergar el código máquina al comienzo del propio programa BASIC. De esta manera, la rutina queda automáticamente cargada al finalizar la lectura del programa, y además permanece oculta o al menos desfigurada, ante los ojos de un posible «usurpador de materia gris». Si opta-

mos por este último método, deberemos seguir los pasos que a continuación exponemos:

En primer lugar, hemos de atenernos a la norma de comenzar nuestro programa BASIC con una línea de comentario (REM), que nos servirá de almacén de la rutina. El motivo de elegir precisamente este comando es bien simple: el intérprete ignorará todo lo que se encuentre a partir del REM hasta el final de la línea, motivo por el cual podremos despreocuparnos de la sintaxis que siga nuestra subrutina, en lo referente a su tratamiento por el BASIC. En este sentido, no tiene importancia el número que se asigne a la línea nodriza: no tiene porqué ser la 1 o la 10; será necesario y suficiente que sea la primera del programa en cuestión.

El siguiente paso es situar a continuación del REM, como mínimo, tantos caracteres como bytes ocupe la rutina. Es indiferente el carácter o caracteres que utilicemos como relleno o si el número de éstos rebasa la longitud de la rutina, pero es absolutamente imprescindible que al menos sean tantos como ocupa nuestro código máquina.

Como habremos apreciado, aquí salta la primera limitación al sistema: no podremos implementar de esta forma ninguna rutina con una longitud superior a los 251 bytes, dado que el editor de pantalla no nos va a permitir generar una línea de longitud superior a los 255 bytes, y al menos cuatro de ellos estarán forzosamente ocupados por el número de línea y la palabra clave (1REM).

Finalmente, volcaremos la rutina a partir de la dirección 374 (incluida). Más adelante, cuando pasemos revista a la organización de la memoria en el área de texto BASIC, comprenderemos el motivo de efectuar la carga desde esta dirección.

En cuanto al sistema para realizar el volcado de la rutina, ya queda a gusto del consumidor: desde el simple POKE en comando directo, hasta la transferencia desde DATAs con borrado posterior de las líneas cargadoras, serán suficiente para esta tarea.

Como para muestra sirve un botón, emplearemos este método en la rutina que pasa parametros a la variable X%, cuya finalidad discutimos anteriormente:

```
10 REM**********
20 FOR I=0 TO 10
30 READ A$
40 POKE 374+I,VAL("&"+A$)
50 NEXT
60 DATA DD,6E,00,DD,66,01,DD,7E,02,77,C9
70 DELETE 20-
```

LISTADO CARGADOR

```
   10 REM CARGADOR GENERAL DE CODIGO
MAQUINA
   20 I=0:MODE 1
   30 INPUT "DIRECCION DE UBICACION";ADD
   40 MEMORY ADD-1
   50 READ LIN$,SUM$
   60 IF LIN$="*" THEN END
   70 IF LEN(LIN$)<>40 THEN PRINT"ERROR EN LA
      LINEA "1000+I*10:STOP
   80 TOTAL=0
   90 FOR J=1 TO 39 STEP 2
  100 BYTE=VAL("&"+MID$(LIN$,J,2))
  110 POKE ADD+20*I+(J+1)/2-1,BYTE
  120 TOTAL=TOTAL+BYTE
  130 NEXT
  140 IF TOTAL<>VAL("&"+SUM$) THEN PRINT
      "ERROR EN LA LINEA "1000+I*10
  150 I=I+1
  160 GOTO 50
10000 DATA *,*
```

EXTENSIONES DEL SISTEMA RESIDENTES

En el capítulo anterior hemos podido ver cómo gracias al comando CALL, BASIC facilitaba el acceso al código máquina. No obstante, con este mismo fin se pone a nuestra disposición una herramienta muchísimo más potente, dado que incorpora las características CALL, añadiendo además la comodidad de uso. Se trata de las extensiones del sistema residentes *(Resident System eXtensions,* RSX), cuya misión es posibilitar el acceso a rutinas en código máquina propias del usuario, implementadas en RAM, como si se tratara de instrucciones BASIC.

RSX

Lo dicho anteriormente acerca de la completa falta de información sobre el comando CALL podemos suscribirlo en el caso de las RSX.

Quizá éste sea uno de los puntos más interesantes del sistema operativo de nuestro Amstrad, pues como hemos dicho, permite la generación de nuevos comandos con los cuales ampliar el Locomotive BASIC. Analicemos su fundamento y manera de incorporar nuevas órdenes.

En la dirección #BCD1 del firmware se encuentra la subrutina conocida como KL LOG EXT, la cual habilita la creación de extensiones del sistema almacenadas en la RAM. Las condiciones de entrada para su acceso son las siguientes:

— BC ha de contener la dirección de la tabla de nuevos comandos.
— HL la dirección de un área de trabajo de longitud constante (4 bytes).

Supongamos que deseamos crear dos nuevos comandos que en nuestro ejemplo llamaremos COMUNO y COMDOS. La estructura de la rutina en código máquina a diseñar para ello sería:

```
            * Entrada RSX *
            LD BC, TSALTOS
            LD HL, ATRABAJO
            CALL #BCD1; Llamada a KL LOG EXT
            RET

            * Tabla de saltos *
TSALTOS:    DEFW COMANDOS; Dirección de la tabla de
              nombres
            JP COMUNO
            JP COMDOS

            * Tabla de nombres  *
COMANDOS:   DEFM "COMUN"
            DEFB 207; Código "0"+128
            DEFM "COMDO"
            DEFB 211; Código "S"+128
            DEFB 0; Siempre 0 al final de la tabla

            * Area de trabajo *
ATRABAJO:   DEFW 0
            DEFW 0
```

A partir de las direcciones COMUNO y COMDOS estarían ubicadas las rutinas que gestionan cada una de las nuevas órdenes. Llamarlas desde BASIC no es otra cosa que ejecutar las instrucciones |COMUNO o |COMDOS, tras activar el programa que los habilita, el cual deberá almacenarse en las 32 K centrales de la RAM.

Es importante hacer notar dos hechos: en primer lugar, y desde la óptica de la sintaxis BASIC, los RSX, para ser correctamente identifi-

cados, deben ir precedidos siempre por la barra vertical (|). En segundo lugar, y esta vez en lo concerniente a la generación de RSX, es importante destacar que en la zona que hemos denominado tabla de nombres (etiqueta de ensamblador COMANDOS), la denominación de cada orden deberá tener alzado el bit 7 de su última letra (código de carácter más 128); asimismo, dicha tabla debe finalizar con un cero.

Las órdenes externas ofrecen también la posibilidad de intercambios de información entre BASIC y C/M, incorporando parámetros separados por coma (,) tras ellas. La interpretación de éstos es del todo análoga a la explicada anteriormente para la instrucción CALL.

UN EJEMPLO: SISTEMA DE PARTICIÓN DE MEMORIA

El programa que finaliza este capítulo introduce como extensión del sistema residente varios nuevos comandos, que permitirán al usuario disponer de hasta nueve programas BASIC simultáneamente en la memoria, con sus propias variables y conservando incluso el puntero de lectura de sus sentencias DATA. Para invocarlos deberemos precederlos por el signo barra vertical (|).

Para la gestión de esta utilidad, SPM genera cuatro nuevos comandos, siguiendo el sistema explicado anteriormente.

El primero de ellos es |**DEFINE,** y lógicamente permite asignar la zona de memoria en la cual deseamos que se sitúe un área determinada. Su sintaxis se completa mediante dos parámetros, que indicarán respectivamente, la primera dirección de memoria a partir de la cual se define el área, y el número de área que se asignará a la misma.

El primero de los parámetros habrá de estar, por tanto, comprendido entre &170, dirección en la cual se ubica habitualmente el texto BASIC, y &A300 que es el punto a partir del cual se ha emplazado la propia rutina SPM. En el caso de no respetar estos márgenes, el sistema nos informará de ello emitiendo el mensaje DIRECCION FUERA DE MARGENES. Por otra parte, la rutina comprobará igualmente si la nueva zona definida interfiere con alguna ya presente para evitar la degeneración de ésta. En lo referente al parámetro AREA, debe ser un número entre 1 y 9, puesto que de no ser así la rutina rechazará la definición emitiendo el mensaje AREA FUERA DE RANGO.

Ni las áreas, ni las direcciones de las mismas tienen porqué estar definidas en orden, sino que basta con que nos atengamos a los márgenes anteriormente descritos. No obstante, lo que sí debemos tener muy en cuenta es que un área una vez definida puede llegar a crecer hasta introducirse en la siguiente, hecho que no puede ser depurado por la rutina, con la consiguiente degeneración de la zona posterior. Así pues, es conveniente que dejemos un espacio suficiente entre las áreas definidas.

El siguiente comando nos permite conmutar entre áreas, indicando cuál deseamos utilizar como parámetro de |**AREA.** Así, por ejemplo, si queremos pasar a la zona 3 deberemos efectuar |**AREA,3.** Lógicamente, el área señalada como parámetro deberá haber sido definida previamente mediante el comando |**DEFINE;** de no ser así, la rutina emitirá el mensaje AREA NO DEFINIDA, del mismo modo que al intentar redefinir una zona ya existente se emite el de AREA YA DEFINIDA.

El comando |**BORRA** tiene una doble utilidad: cuando es empleado desde la misma zona que se desea borrar, su efecto será el de un NEW del área correspondiente, permitiéndonos volver a escribir o cargar cualquier nuevo programa; sin embargo, si la acción de borrado se ejecuta desde otra zona, esta pierde incluso su definición, quedando libre para su nueva utilización en el comando |**DEFINE.** Su sintaxis, como era de esperar, es |**BORRA,área,** donde el parámetro área se depura con el mensaje habitual: AREA FUERA DE RANGO, siempre que no se halle comprendido entre 1 y 9.

Por último, el comando |**LIMITE** nos permitirá conocer la dirección de comienzo y de fin de una zona cualquiera, lo cual nos será de gran utilidad antes del comando |**DEFINE,** para evitar la acción de sobreescritura de otras áreas que anteriormente hemos mencionado. Este es el comando con más parámetros (3), pero en ningún caso debemos preocuparnos por la omisión de alguno de ellos, ni por supuesto por añadirlos en exceso, puesto que en todas las órdenes SPM, la rutina se encarga de depurar el número de éstos, emitiendo el mensaje ERROR DE PARAMETROS, en caso de defecto o exceso del número de ellos.

La sintaxis de este último comando es **LIMITE,@var1%, @var2%,área;** donde var1% es la variable en la cual queremos que se nos devuelva el límite inferior de "área", var2% la del límite superior y área la zona cuyos límites se examinan. Es muy importante definir previamente las variables con cualquier valor, puesto que de no ser así el sistema operativo emitirá el mensaje *Improper*

argument, al no conseguir encontrar la dirección de las variables implicadas.

Asimismo, es de vital importancia no omitir el símbolo @ ante las variables, puesto que el significado de la expresión cambia absolutamente. Tengamos en cuenta que la rutina está esperando una dirección de variable en la cual depositar un valor (@var%), y si omitimos el símbolo @ lo que va a entender como dicha dirección es el valor de la variable, con efectos realmente impredecibles, pero en todo caso no muy halagüeños.

Es importante saber que los datos sobre los límites de un área no se actualizan hasta que se ejecuta algún comando |**AREA,** por lo que si deseamos obtener información veraz sobre las dimensiones de la propia zona en la cual nos encontramos, hemos de ejecutar anteriormente dicho comando, especificando como parámetro la misma área en que nos encontramos.

Para la adopción de este sistema de partición de memoria, ejecutemos el listado BASIC adjunto, del cual los usuarios del modelo 664 ó 6128 habrán suprimido las líneas 200, 210, 220 y 1380 en adelante. Al finalizar la ejecución, la rutina queda lista para su uso, emitiendo el mensaje TU MICRO SPM ACTIVADO.

```
LISTADO BASIC SISTEMA DE PARTICION DE MEMORIA

10 '----------------------------------------
20 '-       FERNANDO LOPEZ MARTINEZ        -
30 '-     CARLOS DE LA OSSA VILLACANAS     -
40 '-  S.P.M.   (c)GRAN BIBLIOTECA AMSTRAD -
50 '----------------------------------------
60 e=0:MODE 1
70 LOCATE 9,12:PRINT"UN MOMENTO, POR FAVOR."
80 MEMORY &A2FF
90 FOR i=0 TO 37
100 READ lin$,sum$
110 IF LEN(lin$)<>40 THEN e=1:PRINT"ERROR EN LA LINEA "1000+I*10:STOP
120 total=0
130 FOR j=1 TO 39 STEP 2
140 byte=VAL("&"+MID$(lin$,j,2))
150 POKE &A300+20*i+(j+1)/2-1,byte
160 total=total+byte
170 NEXT
180 IF total<>VAL("&"+sum$) THEN e=1:PRINT"ERROR EN LA LINEA "1000+I*10
190 NEXT
200 REM LINEAS 210 Y 220 SOLO PARA USUARIOS 464/472
210 READ dir$,alto$,bajo$:dir=VAL("&"+dir$):bajo=VAL("&"+bajo$):alto=VAL("&"+alt
o$):POKE dir,bajo:POKE dir+1,alto
220 base=0:FOR i=0 TO 1:FOR j=0 TO 11:READ dir$,alto$,bajo$:dir=base+VAL("&"+dir
$):bajo=VAL("&"+bajo$):alto=VAL("&"+alto$):POKE dir,bajo:POKE dir+1,alto:NEXT:RE
STORE 1400:base=61:NEXT
230 IF e<>1 THEN CALL &A300:REM ACTIVA SPM
240 END
1000 DATA 3A55A33CC03255A3012DA32151A3CDD1BCCD91A5,99B
1010 DATA 5455204D4943524F2053504D204143544956144,56F
1020 DATA 4F0D0AFFC93BA3C357A3C3DBA3C384A4C3AAA444,B4A
1030 DATA 4546494EC5415245C1424F5252C14C494D4954C5,7BA
1040 DATA 0000000000FF00FE02C2DFA4CD9DA556235E7AB3,857
1050 DATA C2FAA4060978C5DD4E02DD4603CDB4A57CB52818,996
1060 DATA D5E5D1C5E1ED52380EE1C5D1ED523808C13E30B0,B5B
1070 DATA C312A5D1C110D62A5EAED05E02DD5603D5EBED52,A9A
1080 DATA D1D23AA5217001ED52D23AA5CDEBA5DD7E003CCD,AC5
1090 DATA CBA52B3E030606722B732BB820031B1B1B10F4CD,620
```

```
1100 DATA 91A5444546494E49444120FFDD7E00CDDEA5C93D,93A
1110 DATA C2DFA4CD9DA556235E7AB3CA5DA52BE53A56A3A7,B0E
1120 DATA 283ECDCBA53A64AE77233A65AE77233A17AE7723,809
1130 DATA 3A18AE77233A66AE77233A67AE77233A68AE7723,755
1140 DATA 3A69AE77233A6AAE77233A6BAE77233A6CAE7723,7B2
1150 DATA 3A6DAE77E17E3264AE237E3265AE237E3217AE23,810
1160 DATA 7E3218AE237E3266AE237E3267AE237E3268AE23,751
1170 DATA 7E3269AE237E326AAE237E326BAE237E326CAE23,7AE
1180 DATA 7E326DAECD91A5434F4E45435441444120FFDD7E,8CA
1190 DATA 003256A3CDDEA5C93DC2DFA4CD9DA55E23562BCD,AA4
1200 DATA EBA53600233600CD91A5424F525241444120FFDD,819
1210 DATA 7E00CDDEA5C9FE03C2DFA4CD9DA57BCDB4A5E5C1,D33
1220 DATA DD6E04DD6605712370DD6E02DD6603732372CD91,894
1230 DATA A5455841 4D494E41444120FFDD7E00CDDEA5C9CD,98D
1240 DATA 91A54552524F52204445205041524144D4554524F,634
1250 DATA 53070D0AFFC9CD91A5415245412059412044454 6,6FE
1260 DATA 494E494441070D0AFFC9F5CD91A5494E54455246,80B
1270 DATA 4552454E43494120434F4E20415245412 0FFF1CD,70D
1280 DATA 5ABBCD91A5070D0AFFC9CD91A5444952454343 49,8F4
1290 DATA 4F4E20465545524120444520 4D415247454E4553,54B
1300 DATA 070D0AFFC9CD91A5415245412 04E4F2044454649,6F7
1310 DATA 4E494441070D0AFFC9E1CD91A5415245412 04655,7BA
1320 DATA 455241204445205241 4E474F070D0AFFC9E17E23,680
1330 DATA E53CC83DCD5ABB18F4DD5E00DD5601B210800ED,8B4
1340 DATA 52DA75A513D5C17BCDCBA5C9DDE5CDCBA5E5DDE1,E12
1350 DATA DD6E00DD6601DD5E0ADD560BDDE1C9C5D521F7A5,AF0
1360 DATA 110C0047AF1802ED5A10FCD1C1C9063080CD5ABB,873
1370 DATA CD91A50D0AFFC9EB3600233600233600 3EBC900,78C
1380 REM RESTO SOLO PARA USUARIOS 464/472
1390 DATA A394,AE,7B
1400 DATA A3F6,AE,81,A3FB,AE,82,A400,AE,30,A405,AE,31,A40A,AE,83,A40F,AE,84
1410 DATA A414,AE,85,A419,AE,86,A41E,AE,87,A423,AE,88,A428,AE,89,A42D,AE,90
```

```
  10 ****************************************
  20 *      FERNANDO LOPEZ MARTINEZ         *
  30 *               &                      *
  40 *    CARLOS DE LA OSSA VILLACANAS      *
  50 * S.P.M. (c) GRAN BIBLIOTECA AMSTRAD  *
  60 ****************************************
  70
  80         ORG  #A300                    ;DIRECCION DE CARGA DE LA RUTINA #A300
  90
 100 ***** ENTRADA RSX *****
 110
 120 ENTRAD: LD   A,(BANDER)               ;A=BANDERA SPM CONECTADO
 130         INC  A                        ;COMPRUEBA SI SPM CONECTADO
 140         RET  NZ                       ;RETORNA SI SPM CONECTADO
 150         LD   (BANDER),A               ;BANDERA SPM=CONECTADO
 160         LD   BC,SALTOS                ;BC=DIRECC.SALTOS RSX
 170         LD   HL,TRABAJ                ;HL=DIRECC.ESPACIO TRABAJO RSX
 180         CALL #BCD1                    ;IMPLEMENTA SPM COMO RSX
 190         CALL MENSAJ                   ;EMITE MENSAJE DEFM Y DEFB
 200         DEFM "TU MICRO SPM ACTIVADO"
 210         DEFB 13,10,255                ;RETORNO CARRO,LINE FEED,FIN MENSAJE
 220         RET                           ;SPM ACTIVADO
 230
 240 ***** TABLA DE SALTOS *****
 250
 260 SALTOS: DEFW SINTAX                   ;DIRECC.TABLA DE NOMBRES
 270         JP   DEFINE
 280         JP   AREA
 290         JP   BORRA
 300         JP   LIMITE
 310
 320 ***** TABLA DE SINTAXIS *****
 330
 340 SINTAX: DEFM "DEFIN"
 350         DEFB 197                      ;CODIGO "E" + 128
 360         DEFM "ARE"
 370         DEFB 193                      ;CODIGO "A" + 128
 380         DEFM "BORR"
 390         DEFB 193
 400         DEFM "LIMIT"
 410         DEFB 197
 420         DEFB 0                        ;MARCA FINAL DE TABLA DE NOMBRES
 430
 440 ***** AREA DE TRABAJO RSX *****
```

```
450
460 TRABAJ: DEFW 0                      ;4 BYTES RESERVADOS PARA RSX
470         DEFW 0
480
490 ***** BANDERA CONEXION SPM *****
500
510 BANDER: DEFB 255                    ;255=NO ACTIVADO/0=ACTIVADO
520 ACTUAL: DEFB 0                      ;AREA SPM ACTIVADA
530
540 ***** DEFINICION AREA SPM *****
550
560 DEFINE: CP   2                      ;COMPRUEBA LLEGADA DE 2 PARAMETROS
570         JP   NZ,ERRPAR              ;SALTA A ERROR PARAMETROS
580         CALL TESTA                  ;COMPRUEBA PARAMETRO AREA
590         LD   D,(HL)                 ;HL CONTIENE AL VOLVER DE TESTA
600         INC  HL                     ;LA DIRECC.DE COMIENZO DEL AREA DE
610         LD   E,(HL)                 ;VARIABLES DEL SUBSISTEMA A
620         LD   A,D                    ;DE=CONTENIDO DE LA DIRECC.HL
630         OR   E                      ;COMPRUEBA SI DE=0
640         JP   NZ,ERRYDE              ;SI DE<>0 SALTA ERROR AREA YA DEFINIDA
650         LD   B,9                    ;NUMERO DE AREAS A COMPROBAR
660 LOOP1:  LD   A,B                    ;CARGA A CON AREA A COMPROBAR
670         PUSH BC                     ;SALVA INDICE BUCLE COMPROBACION
680         LD   C,(IX+2)               ;BC=DIRECC.AREA A DEFINIR
690         LD   B,(IX+3)
700         CALL LIMA                   ;AVERIGUA LIMITES AREA A DEFINIR
710         LD   A,H                    ;LIMA DEVUELVE HL=LIM.INF./DE=LIM.SUP.
720         OR   L                      ;COMPRUEBA HL=0 (AREA NO DEFINIDA)
730         JR   Z,FINL1                ;SALTO A FIN COMPROBACION AREA
740         PUSH DE                     ;SALVA LIM.SUP. AREA
750         PUSH HL                     ;HACE DE=HL A TRAVES DEL STACK
760         POP  DE                     ;EN DE QUEDA LIM.INF.
770         PUSH BC                     ;HACE HL=BC A TRAVES DEL STACK
780         POP  HL                     ;EN HL QUEDA DIRECC.AREA
790         SBC  HL,DE                  ;COMPRUEBA DIRECC.AREA<LIM.INF.
800         JR   C,POP1                 ;SI DIRECC.AREA<LIM.INF. -> AREA O.K.
810         POP  HL                     ;RECUPERA LIM.SUP. EN HL
820         PUSH BC                     ;HACE DE=BC A TRAVES DEL STACK
830         POP  DE                     ;EN DE QUEDA DIRECC.AREA
840         SBC  HL,DE                  ;COMPRUEBA DIRECC.AREA>LIM.SUP.
850         JR   C,FINL1                ;SI DIRECC.AREA>LIM.SUP. -> AREA O.K.
860         POP  BC                     ;LIMPIA STACK
870         LD   A,48                   ;PASA NUMERO DE AREA (A)
880         ADD  A,B                    ;A CODIGO ASCII (A+48)
890         JP   ERRINT                 ;SALTO A ERROR INTERFERENCIA
900 POP1:   POP  DE                     ;LIMPIA DEL STACK EL LIM.SUP.
910 FINL1:  POP  BC                     ;RECUPERA INDICE BUCLE COMPROBACION
920         DJNZ LOOP1                  ;SALTO A COMPROBACION AREAS RESTANTES
930         LD   HL,(#AE5E)             ;HL=HIMEM
940         LD   E,(IX+2)               ;DE=DIRECC.AREA
950         LD   D,(IX+3)
960         PUSH DE                     ;SALVA DIRECC.AREA
970         EX   DE,HL                  ;CAMBIA DE POR HL
980         SBC  HL,DE                  ;COMPRUEBA DIRECC.AREA>=HIMEM
990         POP  DE                     ;RECUPERA DIRECC.AREA EN DE
1000        JP   NC,ERRADD              ;SI DIRECC.>=HIMEM -> SALTO ERROR DIR.
1010        LD   HL,#170                ;HL=INIC.AREA BASIC POR DEFECTO
1020        SBC  HL,DE                  ;COMPRUEBA SI DIRECC.AREA<=INIC.BASIC
1030        JP   NC,ERRADD              ;SI DIRECC.<=BASIC -> SALTO ERROR DIR
1040        CALL CEROS                  ;AREA O.K. BORRA BASIC NUEVO AREA
1050        LD   A,(IX+0)               ;A=NUMERO DE AREA DEFINIDA
1060        INC  A                      ;BUSCA INICIO VAR.SUBSISTEMA A+1
1070        CALL VARSCA
1080        DEC  HL                     ;HL=HL-1
1090        LD   A,3                    ;CUANDO B=3 -> DE=DE-3
1100        LD   B,6                    ;INDICE BUCLE INIC.VAR.SUBSISTEMA
1110 LOOP2: LD   (HL),D                 ;INICIALIZA LAS 3 ULTIMAS VAR. A DE
1120        DEC  HL                     ;Y LAS 3 PRIMERAS A DE-3
1130        LD   (HL),E                 ;COMO EL BUCLE DJNZ ES DECREMENTAL
1140        DEC  HL                     ;LAS VAR.DEL SUBSISTEMA SE INICIALIZAN
1150        CP   B                      ;DE FINAL A PRINCIPIO DEL AREA DE VARS.
1160        JR   NZ,FINL2               ;LAS 3 PRIMERAS=INIC.AREA BASIC
1170        DEC  DE                     ;LAS 3 ULTIMAS=IN.AREA VARIABLES BASIC
1180        DEC  DE
1190        DEC  DE
1200 FINL2: DJNZ LOOP2                  ;FIN BUCLE INICIALIZACION VARS.
1210        CALL MENSAJE                ;DEFINICION COMPLETADA
1220        DEFM "DEFINIDA "
1230        DEFB 255
```

```
1240          LD   A,(IX+0)
1250          CALL NUMERO
1260          RET                          ;RETORNO COMANDO DEFINE
1270
1280 ***** CONMUTACION AREA SPM *****
1290
1300 AREA:    DEC  A                       ;COMPRUEBA LLEGADA DE UN PARAMETRO
1310          JP   NZ,ERRPAR               ;SALTA A ERROR DE PARAMETROS
1320          CALL TESTA                   ;COMPRUEBA PARAMETRO AREA
1330          LD   D,(HL)                  ;HL CONTIENE AL VOLVER DE TESTA
1340          INC  HL                      ;LA DIRECCION DE COMIENZO DEL AREA DE
1350          LD   E,(HL)                  ;VARIABLES DEL SUBSISTEMA A
1360          LD   A,D                     ;DE=CONTENIDO DE LA DIRECCION HL
1370          OR   E                       ;COMPRUEBA SI DE=0
1380          JP   Z,ERRNDE                ;SI DE=0 SALTA A ERROR AREA NO DEFINIDA
1390          DEC  HL                      ;HL APUNTA A AREA VARS.SUBSISTEMA
1400          PUSH HL                      ;SALVA HL
1410          LD   A,(ACTUAL)              ;A=AREA ACTIVADA ACTUALMENTE
1420          AND  A                       ;COMPRUEBA SI NO HAY AREA ACTIVADA
1430          JR   Z,SUBAPR                ;SI NO AREA SALTA A SUBAPR
1440          CALL VARSCA                  ;VUELCA VARS SISTEMA EN SUBSISTEMA ACT
1450          LD   A,(#AE64)               ;COMIENZO DEL AREA BASIC (BYTE BAJO)
1460          LD   (HL),A
1470          INC  HL
1480          LD   A,(#AE65)               ;COMIENZO DEL AREA BASIC (BYTE ALTO)
1490          LD   (HL),A
1500          INC  HL
1510          LD   A,(#AE17)               ;PUNTERO RESTORE/DATA
1520          LD   (HL),A
1530          INC  HL
1540          LD   A,(#AE18)
1550          LD   (HL),A
1560          INC  HL
1570          LD   A,(#AE66)               ;FINAL DEL AREA BASIC
1580          LD   (HL),A
1590          INC  HL
1600          LD   A,(#AE67)
1610          LD   (HL),A
1620          INC  HL
1630          LD   A,(#AE68)               ;VARIABLES BASIC
1640          LD   (HL),A
1650          INC  HL
1660          LD   A,(#AE69)
1670          LD   (HL),A
1680          INC  HL
1690          LD   A,(#AE6A)               ;VARIABLES SIMPLES
1700          LD   (HL),A
1710          INC  HL
1720          LD   A,(#AE6B)
1730          LD   (HL),A
1740          INC  HL
1750          LD   A,(#AE6C)               ;ARRAYS
1760          LD   (HL),A
1770          INC  HL
1780          LD   A,(#AE6D)
1790          LD   (HL),A
1800 SUBAPR:  POP  HL                      ;VUELCA VARS.SUBSISTEMA A SISTEMA
1810          LD   A,(HL)
1820          LD   (#AE64),A
1830          INC  HL
1840          LD   A,(HL)
1850          LD   (#AE65),A
1860          INC  HL
1870          LD   A,(HL)
1880          LD   (#AE17),A
1890          INC  HL
1900          LD   A,(HL)
1910          LD   (#AE18),A
1920          INC  HL
1930          LD   A,(HL)
1940          LD   (#AE66),A
1950          INC  HL
1960          LD   A,(HL)
1970          LD   (#AE67),A
1980          INC  HL
1990          LD   A,(HL)
2000          LD   (#AE68),A
2010          INC  HL
2020          LD   A,(HL)
```

```
2030          LD   (#AE69),A
2040          INC  HL
2050          LD   A,(HL)
2060          LD   (#AE6A),A
2070          INC  HL
2080          LD   A,(HL)
2090          LD   (#AE6B),A
2100          INC  HL
2110          LD   A,(HL)
2120          LD   (#AE6C),A
2130          INC  HL
2140          LD   A,(HL)
2150          LD   (#AE6D),A
2160          CALL MENSAJ                ;CONEXION DE AREA COMPLETADA
2170          DEFM "CONECTADA "
2180          DEFB 255
2190          LD   A,(IX+0)
2200          LD   (ACTUAL),A            ;AREA ACTUAL=AREA A CONECTAR
2210          CALL NUMERO
2220          RET
2230
2240 ***** BORRADO AREA *****
2250
2260 BORRA:   DEC  A                     ;COMPRUEBA LLEGADA DE UN PARAMETRO
2270          JP   NZ,ERRPAR             ;SALTA A ERROR PARAMETROS
2280          CALL TESTA                 ;COMPRUEBA PARAMETRO AREA
2290          LD   E,(HL)
2300          INC  HL
2310          LD   D,(HL)
2320          DEC  HL
2330          CALL CEROS                 ;BORRA AREA
2340          LD   (HL),0                ;BORRA DEFINICION EN VARS SUBSISTEMA
2350          INC  HL
2360          LD   (HL),0
2370          CALL MENSAJ                ;BORRADO DE AREA COMPLETADO
2380          DEFM "BORRADA "
2390          DEFB 255
2400          LD   A,(IX+0)
2410          CALL NUMERO
2420          RET
2430
2440 ***** AVERIGUA LIMITE     AREA *****
2450
2460 LIMITE:  CP   3                     ;COMPRUEBA LLEGADA DE 3 PARAMETROS
2470          JP   NZ,ERRPAR
2480          CALL TESTA
2490          LD   A,E                   ;AVERIGUA LIMITES AREA A
2500          CALL LIMA                  ;LIMA DEVUELVE HL=LIM.INF./DE=LIM.SUP
2510          PUSH HL                    ;HACE BC=HL A TRAVES DEL STACK
2520          POP  BC                    ;BC=LIM.INF.
2530          LD   L,(IX+4)              ;HL=DIRECC.VARIABLE BASIC QUE
2540          LD   H,(IX+5)              ;ALMACENARA EL LIM.INF.
2550          LD   (HL),C                ;DEPOSITA LIM.INF. EN VAR.BASIC
2560          INC  HL
2570          LD   (HL),B
2580          LD   L,(IX+2)              ;HL=DIRECC.VARIABLE BASIC QUE
2590          LD   H,(IX+3)              ;ALMACENARA EL LIM.SUP.
2600          LD   (HL),E                ;DEPOSITA LIM.SUP. EN VAR.BASIC
2610          INC  HL
2620          LD   (HL),D
2630          CALL MENSAJ                ;OBTENCION DE LIMITES COMPLETADA
2640          DEFM "EXAMINADA "
2650          DEFB 255
2660          LD   A,(IX+0)
2670          CALL NUMERO
2680          RET
2690
2700 ***** ERRORES SPM *****
2710
2720 ERRPAR:  CALL MENSAJ                ;ERROR DE PARAMETROS
2730          DEFM "ERROR DE PARAMETROS";MENSAJE A EMITIR
2740          DEFB 7,13,10,255
2750          RET
2760 ERRYDE:  CALL MENSAJ
2770          DEFM "AREA YA DEFINIDA"
2780          DEFB 7,13,10,255           ;BELL,CR,LF Y FIN DE MENSAJE
2790          RET
2800 ERRINT:  PUSH AF                    ;ERROR INTERFERENCIA CON AREA
2810          CALL MENSAJ
```

```
2820          DEFM "INTERFERENCIA CON AREA "
2830          DEFB 255
2840          POP  AF
2850          CALL #BB5A
2860          CALL MENSAJ
2870          DEFB 7,13,10,255
2880          RET
2890 ERRADD:  CALL MENSAJE                ;ERROR DE DIRECCION
2900          DEFM "DIRECCION FUERA DE MARGENES"
2910          DEFB 7,13,10,255
2920          RET
2930 ERRNDE:  CALL MENSAJ                 ;ERROR DE AREA NO DEFINIDA
2940          DEFM "AREA NO DEFINIDA"
2950          DEFB 7,13,10,255
2960          RET
2970 ERRARE:  POP  HL                     ;ERROR DE AREA
2980          CALL MENSAJ
2990          DEFM "AREA FUERA DE RANGO"
3000          DEFB 7,13,10,255
3010          RET
3020
3030 ***** SUBRUTINAS SPM *****
3040
3050 MENSAJ:  POP  HL                     ;HL=DIRECCION DEL MENSAJE A EMITIR
3060          LD   A,(HL)                 ;A=CARACTER A IMPRIMIR
3070          INC  HL                     ;INCREMENTA PUNTERO
3080          PUSH HL                     ;SALVA DIRECCION DE RETORNO
3090          INC  A                      ;COMPRUEBA FIN DE MENSAJE (A=255)
3100          RET  Z                      ;RETORNA SI MENSAJE TERMINADO
3110          DEC  A                      ;A=CARACTER A IMPRIMIR
3120          CALL #BB5A                  ;IMPRIME EL CARACTER
3130          JR   MENSAJ                 ;CONTINUA LECTURA MENSAJE
3140 TESTA:   LD   E,(IX+0)               ;DE=AREA
3150          LD   D,(IX+1)
3160          DEC  DE                     ;DE=DE-1 PARA DISCRIMINAR AREA=0
3170          LD   HL,8                   ;MAXIMO AREA=9
3180          SBC  HL,DE                  ;COMPRUEBA AREA EN RANGO 1-9
3190          JP   C,ERRARE               ;SALTA A ERROR DE AREA
3200          INC  DE                     ;RETORNA DE A VALOR AREA
3210          PUSH DE                     ;HACE BC=DE A TRAVES DEL STACK
3220          POP  BC
3230          LD   A,E
3240          CALL VARSCA
3250          RET
3260 LIMA:    PUSH IX                     ;HL=LIMITE INFERIOR AREA A
3270          CALL VARSCA                 ;DE=LIMITE SUPERIOR AREA A
3280          PUSH HL
3290          POP  IX
3300          LD   L,(IX+0)
3310          LD   H,(IX+1)
3320          LD   E,(IX+10)
3330          LD   D,(IX+11)
3340          POP  IX
3350          RET
3360 VARSCA:  PUSH BC                     ;SALVA BC
3370          PUSH DE                     ;SALVA DE
3380          LD   HL,AREA1               ;HL=INICIO AREA 1
3390          LD   DE,12                  ;CADA AREA OCUPA 12 BYTES
3400          LD   B,A                    ;NUMERO DE AREA A EXAMINAR
3410          XOR  A                      ;PONE A CERO LA BANDERA DE CARRY
3420          JR   ENDVAR                 ;HACE HL=HL+12 POR CADA AREA EXAMINADA
3430 LOOP3:   ADC  HL,DE                  ;HASTA LLEGAR AL AREA A EXAMINAR
3440 ENDVAR:  DJNZ LOOP3                  ;EN HL QUEDA EL COMIENZO DEL
3450          POP  DE                     ;AREA DE VARIABLES DEL SUBSISTEMA A
3460          POP  BC                     ;RECUPERA DE Y BC
3470          RET
3480 NUMERO:  LD   B,48                   ;PASA EL CODIGO DEL ACUMULADOR
3490          ADD  A,B                    ;A CODIGO ASCII (A=A+48)
3500          CALL #BB5A                  ;IMPRIME EL NUMERO DE AREA
3510          CALL MENSAJE                ;IMPLICADA EN EL COMANDO
3520          DEFB 13,10,255              ;DESDE EL CUAL SE HACE
3530          RET                         ;LA LLAMADA A LA SUBRUTINA
3540 CEROS:   EX   DE,HL                  ;INTERCAMBIA DE Y HL
3550          LD   (HL),0
3560          INC  HL                     ;DEPOSITA TRES CEROS SEGUIDOS
3570          LD   (HL),0                 ;A PARTIR DE HL
3580          INC  HL                     ;SE UTILIZA PARA BORRAR UN AREA
3590          LD   (HL),0
3600          INC  HL
```

```
3610          EX   DE,HL                ;VUELVE A INTERCAMBIAR DE Y HL
3620          RET
3630
3640 ***** VARIABLES DEL SUBSISTEMA SPM *****
3650
3660 AREA1:   DEFS 108                 ;ESTABLECE 108 CEROS VAR.SUBS.(12*9)
```

EL PERFIL HARDWARE

La programación en su más bajo nivel, es decir, en código máquina, se encuentra íntimamente relacionada con el comportamiento de la máquina en sí; es por ello que un estudio de la arquitectura básica de nuestro ordenador, desde el punto de vista de su programación, nos será de gran utilidad siempre que intentemos profundizar en el manejo del mismo, desde el lenguaje máquina.

GATE ARRAY

Como podemos comprobar en el diagrama adjunto, el sistema se desarrolla en torno al microprocesador Z-80, teniendo éste como lugarteniente, destinado a las tareas más arduas, a la *Gate Array,* encargada del control lógico de todo el sistema: desde los modos de pantalla y su color hasta la ROM.

La *gate array* es un circuito *custom,* es decir, especialmente diseñado para Amstrad, que desde el punto de vista de la programación, puede considerarse como un port de ocho bits, controlable a partir de la dirección #7F??

Los dos bits de mayor peso (bit 7 y bit 6) controlan su aplicación:

00 → Carga del registro de colores.
01 → Carga de la memoria de colores.
10 → Control de vídeo e intercambio de ROMs.
11 → Reservados.

En su función de control de vídeo e intercambios de ROMs, los seis bits restantes (los de menor peso, de 5 a 0) toman el siguiente significado:

Bit 5 → 0
Bit 4 → 1 Restaura a cero el dispositivo de interrupción.
Bit 3 → 0/1 Habilita/Deshabilita la ROM superior.
Bit 2 → 0/1 Habilita/Deshabilita la ROM inferior.
Bits 1 y 0 → 0/1 Control de modos de pantalla según la configuración:

00 → Modo 0.
01 → Modo 1.
10 → Modo 2.
11 → Sin uso.

En su función de carga de registro de colores, los seis bits menos significativos adquieren el siguiente valor:

Bit 5 → 0
Bit 4 → 0/1 Carga el código de tinta/borde.
Bits 3 a 0 → ???? Establecen el código de tinta. 15 colores disponibles.

Por último, cuando se trata de cargar la memoria de colores, los valores de los bits 5 a 0 son:

Bit 5 → 0
Bits 4 a 0 → ????? Establecen la decodificación del valor del registro de colores. Hasta 31 colores disponibles, según el modo de pantalla.

LA MEMORIA

Las memorias RAM y ROM constituyen, lógicamente, otra parte esencial del sistema, y se encuentran mapeadas, a efectos de direccionamiento por el microprocesador, en los bloques que se muestran en la figura.

Las 32 K de ROM, en las cuales reside el firmware (sistema operativo e intérprete BASIC) se halla dividida en dos zonas de 16 K cada una: la baja *(lower ROM),* desde #0000 hasta #3FFF y el bloque alto *(upper ROM),* desde #C000 hasta #FFFF, posiciones éstas, como veremos, circunstancialmente empleadas para albergar la pantalla.

Este último detalle acerca de la *upper ROM,* nos hace entender el sentido de la importante misión de la *gate array.* Si efectuamos un POKE desde BASIC a las direcciones #C000 en adelante, probablemente obtendremos algún tipo de línea de color en la pantalla. Ello se debe a que en estas direcciones de memoria, se halla la zona de pantalla. El sistema de mapeado, controlado por nuestro Amstrad, es el que le permite discernir, al referirse a las posiciones compartidas, si deseamos acceder a la memoria de pantalla, o bien a la zona alta de ROM, aprovechando así el espacio de memoria disponible al tope de su capacidad.

Por otra parte, estas dos memorias pueden ser controladas separadamente por la *gate array,* tanto dentro como fuera del circuito normal. Así, existe una señal en el port de expansión que permite desconectar la ROM interna, de forma que el microprocesador pueda acceder a ROM ubicada en el exterior.

En cuanto a los 64 K de RAM (en los 464 y 664 y la RAM principal de los 6128), se halla obviamente entre #0000 y #FFFF, motivo por el cual comparte sus primeros y últimos 16 K con la ROM. Este sistema no nos generará mayor problema a la hora de escribir, puesto que evidentemente se entiende que dicha operación se efectuará sobre RAM. No obstante, en el momento de la lectura, es posible que en posiciones compartidas queramos acceder a ROM o RAM; en tal caso, estudiaremos las rutinas del firmware que nos permiten seleccionar la memoria apropiada.

Como ya sabemos, la capacidad de direccionamiento del Z-80 es de sólo 64 K, motivo por el cual, al emplearse esta cantidad de memoria para RAM, es preciso seguir un sistema de conmutación de bancos, que posibilita el manejo de las 32 K de ROM del sistema, y que indirectamente permite extender dicha memoria ROM, en expan-

siones externas, hasta un total de 4032 K (252 expansiones de 16 K cada una).

Cuando cualquiera de las dos ROM se encuentra habilitada, las lecturas correspondientes al área que ocupan (#0000 a #3FFF la baja y #C000 a #FFFF la alta) se suponen hacia la ROM, mientras que si se hallan desactivadas, dichas operaciones se dirigen, obviamente, a la RAM. Por contra, cualquiera que sea el estado de las ROM, las escrituras siempre se realizan en RAM.

No obstante, dejaremos aquí el complicado sistema de conmutación, dado que nos preocupa fundamentalmente el área de memoria RAM, puesto que es donde supuestamente se desenvolverá nuestro trabajo.

PPI

Como podemos constatar en el diagrama de bloques, otro chip de especial relevancia es el PPI 8255 *(Parallel Peripheral Interface,* Interface Periférico Paralelo), cuya función se halla en relación con el teclado y joysticks, PSG (generador de sonido) e indirectamente el altavoz, casete y port de impresora.

El PPI dispone de tres ports. El port A se emplea en las comunicaciones con el PSG, conmutándose para entrada o salida, según convenga. La utilidad de este modo se pondrá de manifiesto al hablar del PSG. El port B se emplea esencialmente en el control del casete y de la señal BUSY de la impresora Centronics. Por último, el C se utiliza como port de salida en el control del motor del casete, o escritura de datos en el mismo, así como en la selección de filas del teclado y en las señales de STROBE de datos para el PSG.

El 8255 es un chip diseñado inicialmente para los microprocesadores de la serie INTEL 8080 con los cuales el Z-80 es compatible, como versión ampliada y mejorada. En el PPI accedemos a efectos de programación a 24 bits (lectura/escritura), que desde el punto de vista lógico se distribuyen en dos grupos de 12 bits utilizados de dos modos fundamentales:

— Como tres ports de entrada/salida de 4 bits.
— Como un port de entrada/salida de 8 bits más cuatro bits para *handshaking* (control de transmisiones).

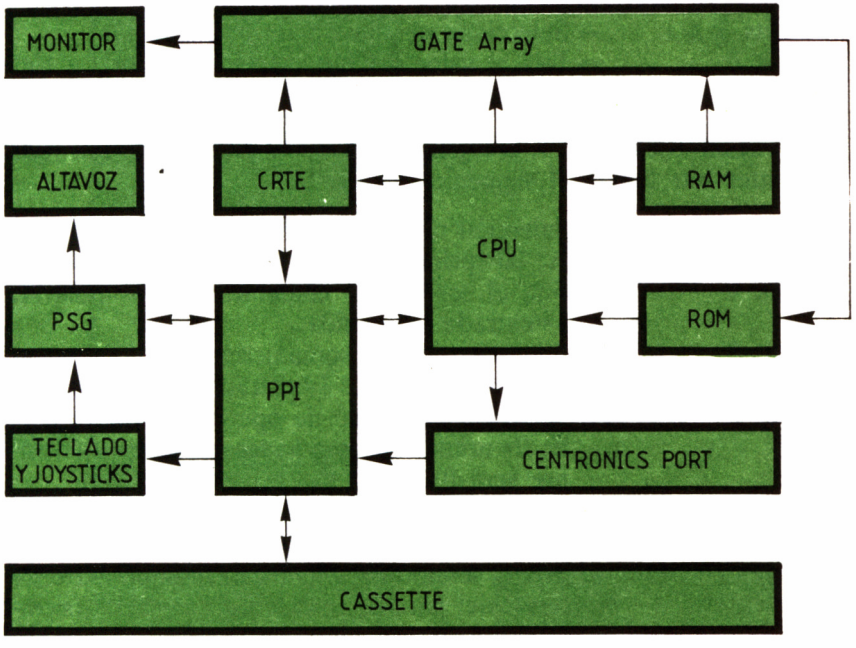
No obstante, como ya hemos dicho antes, se puede hacer una división del PPI en tres ports de 8 bits (A, B y C), en el que el port C se divide en dos bloques de 4 bits, que conforman los grupos de 12 bits con los ports A y B. Las misiones de los 24 bits son las siguientes:

Port A (entrada/salida)

Bits 7 a 0 → Se corresponden con los datos (D7 a D0) para el PSG.

Port B (sólo entrada)

Bit 7 → Lectura de datos del casete.
Bit 6 → Señal de BUSY de la impresora Centronics.
Bit 5 →
Bit 4 →
Bit 3 → No disponible.
Bit 2 →
Bit 1 →
Bit 0 → Interrupción de CRT *(Cathode Rays Tube,* Tubo de Rayos Catódicos).



Port C (sólo salida)

Bit 7 → BDIR en la salida del PSG (señal del AY-3-8912 que estudiaremos al final de este capítulo).
Bit 6 → BC1 en la salida del PSG (señal del AY-3-8912 que estudiaremos al final de este capítulo).
Bit 5 → Escritura de datos en el casete.
Bit 4 → Arranque/parada del casete.
Bits 3 a 0 → Selección de fila investigada en el teclado.

El PPI se programa a través de un único registro de control de sólo escritura, para lo cual se efectúa un OUT en el port #F7??, siguiendo la significación de bits que a continuación se relaciona:

Bit 7 → Selección para 0=asignación de bits del port C / 1=port de control.

Si el bit 7 se halla a 0, el registro se utiliza para asignar los valores de los bits del port C y los restantes bits toman el siguiente significado:

Bits 6 a 4 → Sin utilizar.
Bits 3 a 1 → Selecciona el número de bit a afectar.
Bit 0 → Fija el valor del bit seleccionado por los tres anteriores.

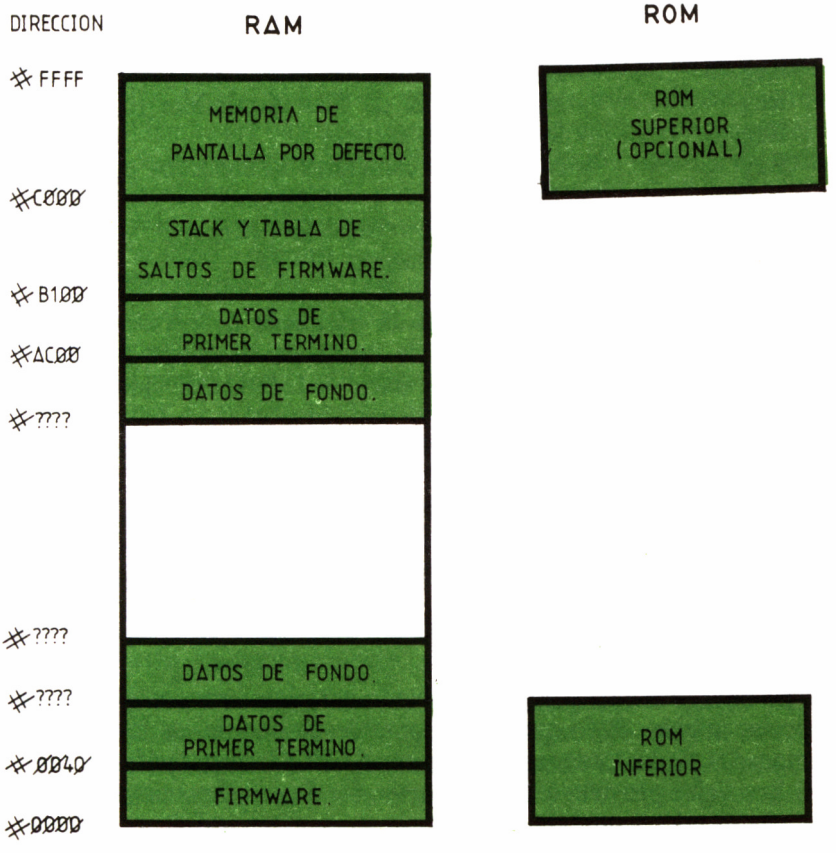
Si el bit 7 se halla a 1, el registro actúa como port de control, y los restantes bits toman el siguiente significado:

Bits 6 y 5 → Selección del modo de operación para el port A; 00=primer modo / 01=segundo modo.
Bit 4 → Fija el sentido de comunicación del port A; 1=entrada / 0=salida.
Bit 3 → Fija el sentido de comunicación de la parte superior del port C; 1=entrada / 0=salida.
Bit 2 → Selección del modo de operación para el port B; 0=primer modo / 1=segundo modo (siempre será 0).
Bit 1 → Fija el sentido de comunicación del port B; 1=entrada / 0=salida (siempre será 1).
Bit 0 → Fija el sentido de comunicación de la parte inferior del port C. 1=entrada / 0=salida (siempre será 0).

Finalmente, gracias al registro de control, es posible seleccionar el modo y sentidos de comunicación apropiados para, a continuación, proceder a la lectura o escritura, según convenga, en el port adecuado a cada caso.

CRTC

El CRTC 6845 *(Cathode Ray Tube Controller,* Controlador del Tubo de Rayos Catódicos), en conjunción con la *gate array,* supervisa todo el flujo de señales hacia la pantalla.



Se trata de un port de entradas/salidas de 8 bits que puede controlarse gracias a sus diecinueve registros internos, el último de los cuales hace las veces de buffer para la programación de los dieciocho restantes.

El port #BC?? se emplea para seleccionar las direcciones de los registros, y el #BD?? para la escritura de datos en el registro. Tengamos en cuenta que todos los registros son sólo de escritura, a excepción del 14 y 15, debido a que controlan la posición del cursor y, por tanto, pueden ser leídos para averiguar la ubicación del mismo.

Los cometidos y valores posibles para cada uno de los registros son los siguientes:

Reg. 0 → Número total de espacio para caracteres disponibles en horizontal (0 - 255).
Reg. 1 → Número de caracteres visualizados en horizontal (0 - 255).
Reg. 2 → Sincronismo horizontal (0 - 255).
Reg. 3 → Longitud de la sincronización (0 - 15).
Reg. 4 → Número total de filas disponibles (0 - 127).
Reg. 5 → Sincronismo vertical (0 - 31).
Reg. 6. → Número de caracteres visualizados en vertical (0 - 127).
Reg. 7 → Sincronismo vertical (0 - 127).
Reg. 8 → Modo entrelazado (0 - 3).
Reg. 9 → *Scanning* (0 - 31).
Reg. 10 → Línea inicial de búsqueda de cursor (0 - 31)
Reg. 11 → Línea final de búsqueda de cursor (0 - 31).
Regs. 12 y 13 → Dirección de inicio de la RAM de vídeo, expresada en peso alto-bajo.
Regs. 14 y 15 → Posición del cursor, expresada en peso alto-bajo.
Regs. 16 y 17 → Control para el lápiz óptico.

PSG

El PSG, del cual ya hemos hablado anteriormente, es un chip de especial interés, dado que es responsable de una de las facetas más destacadas de los ordenadores Amstrad CPC: la generación de sonido. Sus siglas provienen de *Programmable Sound Generator* (generador de sonido programable) y como su nombre indica, es de gran

utilidad para el programador, dado que es posible controlarlo desde lenguaje máquina, obteniéndose de él efectos sonoros de considerable calidad.

En concreto, se trata de un circuito integrado bien conocido, el AY-3-8912, pues también lo incorporan para la gestión de su sonido, equipos tan conocidos como los Spectravídeo. Su importancia reside en los tres canales de generación de sonido que puede controlar independientemente, con sus correspondientes envolventes, así como uno de ruido blanco (notas de frecuencia aleatoria y corta duración que emitidas reiteradamente simulan ruido). Asimismo, dispone de un port de entrada/salida, conmutado por el PPI, como hemos visto anteriormente, que en modo entrada permite controlar el teclado y los joysticks.

Su estructura general se puede dividir en los siguientes elementos:

— Generadores de sonido: Posee tres generadores de sonido independientes, denominados canales (A, B y C), cada uno de los cuales puede ser utilizado en solitario o en combinación con los demás, gracias al mezclador, que más adelante veremos.
— Generador de ruido blanco: El AY-3-8912 posee un generador de ruido blanco, consistente en la emisión sucesiva de notas de frecuencia aleatoria y muy corta duración, para simulación de ruido, lo cual permite obtener de este PSG efectos sonoros de notable realismo.
— Mezclador: Es la fase que permite la mezcla de los diversos canales y el generador de ruido blanco, para obtener efectos polifónicos.
— Control de amplitud fija: Este control de la amplitud se efectúa por el propio microprocesador.
— Control de amplitud variable: En este caso, la amplitud la controla el PSG, gracias a hasta ocho envolventes.
— Conversor digital/analógico: Este conversor permite emitir como señales analógicas los sonidos, que lógicamente en el interior del ordenador no son más que números (magnitudes digitales).
— Port de entrada/salida: De este elemento obtuvimos información ya anteriormente.

A efectos de programación, el PSG dispone de quince registros, de los cuales los catorce primeros se dedican a la emisión de sonido y el último (Reg. 14) al control del port de entrada/salida, aunque sólo se emplea en funciones de entrada: lectura del teclado y joysticks. En relación directa con este registro se halla el sexto bit del registro 7,

dado que, como estudiaremos a continuación, es el que controla el sentido del port, que como acabamos de decir, en nuestro caso siempre es de entrada.

Los primeros tres pares de registros (Reg. 0 a Reg. 5) controlan, respectivamente, las frecuencias de sonido para los canales A, B y C. Los primeros bytes de cada pareja tienen un valor que oscila entre 0 y 255 y se emplean para el ajuste fino de la frecuencia, mientras que de los segundos bytes (Reg. 1, Reg. 3 y Reg. 5) se utilizan tan sólo los cuatro bits menos significativos; es decir, toman valores entre 0 y 15, y su misión es el ajuste burdo de la frecuencia. El valor numérico de la frecuencia se obtiene dividiendo 125.000 entre la frecuencia que se desea obtener en Herzios.

El séptimo registro tiene una función en todo similar a los anteriores, y su valor de frecuencia se determina también mediante la misma fórmula, aunque se aplica al generador de ruido blanco. Sólo se emplean sus cinco bits menos significativos, motivo por el cual sus valores oscilan entre 0 y 31.

El octavo registro es de gran importancia, dado que controla la mezcla de los tres canales y el generador de ruido, así como el sentido del port de entrada/salida, gracias a su bit 6, como ya hemos visto líneas atrás. Dicho bit estará a 0 para la entrada (estado normal) y a 1 para la salida (no se emplea). Por otra parte, el bit más significativo (bit 7) no se utiliza. Así, el significado de los seis bits de menor peso, que afectan a la emisión combinada de sonidos, queda de la siguiente manera:

Bit 5	Ruido blanco en C.
Bit 4	Ruido blanco en B.
Bit 3	Ruido blanco en A.
Bit 2	Sonido en C.
Bit 1	Sonido en B.
Bit 0	Sonido en A.

En todo caso, el bit a 0 significa emitir el ruido o sonido, y a 1 desactivar la emisión por el canal correspondiente.

Los registros 8 a 10 controlan la amplitud de los canales A, B y C, respectivamente, empleándose a tal fin tan sólo los cuatro bits menos significativos de cada uno (valores de 0 a 15). El quinto bit (bit 4), fija el modo de control de amplitud: estática (0) o por envolvente (1).

El par de registros 11 y 12 controlan el período de la envolvente, siguiendo un método similar al empleado para fijar la frecuencia en

los tres primeros pares de registros. En esta ocasión, la fórmula es: V=125000*T/16; donde V es el valor y T el período.

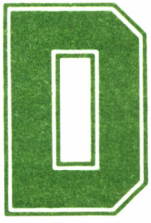
Cuando en el bit 4 de los registros 8 a 10 se indica el control mediante envolvente, los cuatro bits de menor peso del registro 13 asignan a la envolvente la forma a emplear entre las ocho disponibles:

Núm. Env.	Bits 3210	Valores decimales	Forma de la Envolvente
1 →	00?? →	0, 1, 2 ó 3 →	Un ciclo simple con comienzo en la amplitud máxima y caída hasta cero.
2 →	01?? →	4, 5, 6 ó 7 →	Un ciclo simple con comienzo en cero y ascensión hasta su máxima amplitud, con caída posterior hasta cero.
3 →	1000 →	8 →	La envolvente del tipo 1 repetida continuadamente.
4 →	1010 →	10 →	La envolvente del tipo 3, pero con un ascenso hasta el máximo (ataque) más escalonado.
5 →	1011 →	11 →	La envolvente del tipo 1, pero con regreso al valor máximo al final de la misma.
6 →	1100 →	12 →	La envolvente del tipo 2 repetida continuadamente.
7 →	1101 →	13 →	La envolvente del tipo 2, pero con regreso al valor máximo al final de la misma.
8 →	1110 →	14 →	La envolvente del tipo 6, pero con un ascenso hasta el máximo (ataque) más escalonado.

Como ya sabemos, el PSG es accesible a través de los ports A y C del PPI. Para su programación directa, los bits 7 y 6, respectivamente, del port C del PPI, controlan las señales BDIR y BC1 del PSG, cuyo significado detallamos como final de este capítulo:

BDIR=1 y BC1=1 → Selección del registro a afectar; su número se halla en el bus DO a D7.
BDIR=1 y BC1=0 → El valor a asignar al registro seleccionado se halla en el bus DO a D7.
BDIR=0 y BC1=1 → Ubicación en el bus D0 a D7 del valor leído en el registro seleccionado.
BDIR=0 y BC1=0 → No se utiliza.

PRIMER CONTACTO CON EL FIRMWARE



el mismo modo que hemos establecido en el capítulo anterior un primer contacto con el hardware, desde el punto de vista de la programación de los principales chips que lo integran, pasaremos a continuación rápida revista al firmware de nuestro ordenador, estableciendo así una cierta división, que nos permitirá abordar sus distintos bloques de una manera ordenada, para conseguir que este libro se convierta en un futuro, en un rápido, cómodo y completo manual de consulta.

DIVISIÓN DE LA MEMORIA

La memoria de interés a efectos del estudio del firmware puede ser dividida en tres grandes áreas:

El espacio de trabajo; destinado a albergar fundamentalmente las variables del sistema y los vectores para el acceso a las rutinas de la ROM baja.

La ROM alta; que contiene el intérprete BASIC.

La ROM baja; en cuyo contenido se cuenta el generador de caracteres, las rutinas matemáticas y las rutinas del sistema que a continuación detallaremos.

Dichas rutinas del sistema se pueden dividir a su vez en diferentes bloques, según su finalidad:

— Control del teclado.
— Presentación de texto.
— Presentación de gráficos.
— Gestión de la pantalla.
— Gestión del sonido.
— Gestión del casete.
— Kernel (rutinas básicas del sistema operativo).
— Gestión del hardware a bajo nivel (rutinas básicas).
— Tabla de saltos.

Conocimientos Previos

Antes de adentrarnos en la descripción exhaustiva de las diferentes secciones que integran el firmware, es necesario que sepamos la forma de manejar esa información, es decir, la forma exacta en que podemos hacer uso de las diferentes rutinas.

En primer lugar, hemos de tener en cuenta que no es tan importante dónde se hallen físicamente ubicadas las rutinas en la memoria, como cuál es el modo de acceder a ellas. Así, por ejemplo, si lo que nos interesa es escribir un carácter en la pantalla, será para nosotros de mayor utilidad acceder a la rutina del sistema apropiada, más que investigar cómo efectúa su trabajo, para poder hacer nosotros una réplica de la misma.

El porqué de esta afirmación se encuentra, por una parte, en el ahorro de memoria que supone el uso del firmware, y por otra, en que nos asegura cumplir todos y cada uno de los requisitos que conlleva cualquier acción. Así, por ejemplo, escribir un carácter, no se limita a presentar éste en la pantalla, sino que lleva asociadas ciertas tareas complementarias, como por ejemplo actualizar la posición del cursor, comprobar si es necesario efectuar un scrolling de la pantalla, etc...

La Tabla De Saltos

El sistema principal de control del firmware es el manejo de la tabla de saltos. Pese a que en un principio puede parecer más complejo que el simple acceso a la dirección absoluta de ubicación de las

rutinas, comprobaremos que se trata de un método extraordinariamente más eficaz, cómodo y útil.

Existen una serie de rutinas que conforman el firmware, cada una de las cuales tiene un cometido concreto. Tomemos por ejemplo la conocida como KM WAIT KEY, cuya función es esperar hasta que se produzca la pulsación de una tecla. Pues bien, para acceder a ella, necesitamos saber, fundamentalmente, el lugar al cual efectuar un **CALL** para su ejecución.

Si esta rutina se encontrara en una determinada posición, por ejemplo #1000 (se trata tan sólo de un ejemplo), sería suficiente con que ejecutáramos **CALL &1000** para obtener el efecto deseado. No obstante, para el manejo del firmware no operaremos de esta manera, sino que el lugar donde efectuaremos la llamada será otro punto de la memoria, que a su vez contendrá el salto a la dirección en donde se encuentra la rutina en cuestión.

Para explicar este aparente enrevesamiento gratuito, partiremos de el dato fundamental: la dirección de la tabla de saltos correspondiente a KM WAIT KEY es #BB18. Así, podremos comprobar su efecto con un simple **CALL &BB18.** En principio, a nosotros no nos ha reportado mayor problema emplear este sistema, dado que el mismo esfuerzo supone efectuar un **CALL** a #BB18 que a la supuesta dirección #1000 que habíamos otorgado como posición absoluta de la rutina. Ahora bien, ¿qué ventajas nos reporta?

En primer lugar, la comprensión del mecanismo no ofrece ningún problema, puesto que la tabla de saltos tiene una estructura extremadamente simple. A cada rutina, le corresponden tres bytes consecutivos de la tabla de saltos: el primero de ellos es #C3 (código de ensamblador JP de salto incondicional a una dirección absoluta); el segundo y el tercero serán, por tanto, en byte bajo-byte alto, la dirección absoluta de ubicación de la rutina. Es decir, la tabla de saltos no contiene más que sucesivas instrucciones de salto absoluto (de ahí su nombre) a las diferentes rutinas del sistema.

La primera utilidad se hace evidente a la hora de compatibilizar posibles cambios en las rutinas del software. Supongamos que Amstrad decidiese sustituir la rutina KM WAIT KEY por otra que, cumpliendo el mismo cometido, fuera más rápida o eficaz. De ser algo más grande, la nueva rutina no podría ubicarse en el lugar de la antigua, en cuyo caso, su posición absoluta debería cambiar. Así, un programa que hiciera uso de esta rutina funcionaría tan sólo en los modelos antiguos y no en los nuevos. Siguiendo el sistema de tabla de saltos, este problema no se produce, ya que simplemente hay que

alterar las posiciones #BB19 y #BB1A para que apunten a la nueva dirección de KM WAIT KEY. En todo caso, la llamada a ésta seguirá siendo **CALL &BB1A,** con lo que cualquier programa que haga uso de ella funcionará en los modelos nuevos.

Una última ventaja de este sistema reside en su gran flexibilidad de programación, dado que la tabla de saltos se encuentra en memoria RAM, lo cual quiere decir que puede ser modificada por el usuario a su antojo. Así, es posible interceptar cualquier rutina del sistema y desviarla a una propia. Por ejemplo, si creáramos una rutina especial de detección de teclas, que tomara en consideración la pulsación de SHIFT, no tendríamos más que modificar las posiciones #BB19 y #BB1A para que apuntaran a la posición de nuestra rutina, con lo cual el propio sistema haría uso de ella.

La tabla principal de saltos ocupa la zona #BB00 a #BD39; no obstante, existen otros bloques a continuación, de funcionamiento similar:

— Indirecciones del firmware: Se trata de saltos del mismo tipo que los estudiados anteriormente, es decir, triadas de bytes que producen las transferencias a posiciones absolutas. Se sitúan en el área comprendida entre #BDCD y #BDF3 y su cometido es la gestión de funciones muy básicas (de bajo nivel), como por ejemplo la presentación del cursor, la activación de un pixel, el borrado de la pantalla o la investigación de pulsación de la tecla BREAK.

Se trata de subrutinas de gran importancia, a las cuales recurren multitud de rutinas del sistema, debido a lo cual una acción sobre ellas puede tener gran trascendencia, no obstante, su utilidad no se halla fundamentalmente en el acceso a las mismas, dado que por lo general se dispone de rutinas de mayor nivel (tabla de saltos principal) más cómodas de usar, sino en su modificación para conseguir ciertas alteraciones del hardware.

— Tabla de saltos del Kernel alto: Permite al usuario habilitar, deshabilitar y acceder a las memorias ROM. Esta tabla de saltos se halla desde la dirección #B900 y no todos sus componentes son instrucciones de salto, sino que algunas son el comienzo de rutinas; no obstante, el usuario no debería alterar ninguna de las entradas de esta tabla de saltos.

— Tabla de saltos del Kernel bajo: Esta zona comprende las direcciones #0000 a #003F, algunas de cuyas rutinas pueden ser accedidas, por tanto, por instrucciones RST. Por lo general, las rutinas se hallan copiadas en RAM, motivo por el cual pueden ser accedidas aunque la ROM baja se encuentre habilitada.

Aunque en esta zona tampoco es recomendable efectuar alteraciones, se dispone de algunas direcciones de gran interés para una modificación por parte del usuario: la entrada a la interrupción (#0038) o la entrada a la reinicialización (desde #0000 a #0007).

EL CONTROLADOR DEL TECLADO

Cuando ponemos en funcionamiento el ordenador o lo reinicializamos, una zona del firmware queda dispuesta a recoger las pulsaciones que seguidamente efectuemos. Se trata del KEY MANAGER (controlador del teclado).

En el Amstrad, la exploración sistemática del teclado, los códigos generados por una determinada pulsación o el control de los joysticks está completamente gestionado vía software. En este sentido, cabe añadir que la construcción interna de los contactos que llegan a los mandos a distancia entran directamente dentro de la matriz del teclado y, por tanto, son interpretados, como si de la pulsación de una tecla más se tratara.

El controlador del teclado trabaja a tres niveles de operación diferentes. El más bajo se encarga de examinar regularmente el teclado; el medio, convierte las teclas presionadas en valores numéricos que son almacenados en un buffer intermedio destinado a tal efecto. Por último, el nivel alto se encarga de transformar o interpretar dichos valores convirtiéndolos en caracteres.

Empleando las rutinas descritas en este capítulo, se puede acceder en cualquier momento al nivel de trabajo que el usuario necesite, no siendo imprescindible mantenerse en uno determinado, ya que según el programa puede seleccionarse indistintamente el apropiado a su fin concreto.

También podemos observar las principales tareas gestionadas por este área del firmwäre. Entre ellas se cuentan, el control del tiempo de retardo hasta la autorrepetición de una tecla que continúa pulsada, esta velocidad de repetición, generación y definición de las teclas según se pulsen solas o conjuntamente con MAYS o CONTROL, etc.

El teclado es explorado cincuenta veces por segundo. Los contactos de la matriz que configura el teclado son leídos y un mapa de bits (bit map) anota las teclas pulsadas. Este mapa está preparado para determinar si algunas teclas específicas han sido presionadas. Esta es la misión de la rutina KM TEST KEY.

Si son detectadas nuevas pulsaciones, se anotan y almacenan en el buffer, mientras que si no se presiona una nueva tecla, entonces a la última se la autoriza para repetirse siempre y cuando continúe presionada. Para que esto sea posible, ha de mantenerse así durante al menos dos exploraciones consecutivas del teclado (2/50 segundos).

Existe un problema en el firmware del teclado: cuando tres teclas colocadas en las esquinas de un rectángulo en la matriz de teclado son pulsadas simultáneamente, la cuarta esquina sorprendentemente es la que aparece como resultado. Este efecto se puede comprobar, por ejemplo, con la sucesión Q, Z, P, la cual provoca la impresión en la pantalla del carácter (.).

No hay manera de evitar este problema, pues se trata de una cuestión del hardware de teclado. No obstante, las rutinas del firmware y del intérprete de BASIC están diseñadas de forma que esta circunstancia no presente problema alguno.

Las Tablas De Conversión

Cuando en un programa se efectúa una llamada a las rutinas KM WAIT KEY o KM READ KEY el sistema operativo busca en el buffer una tecla. En el caso de la primera rutina, aguarda hasta que encuentre una si es que el buffer estaba vacío. Tras esto, se traslada a la apropiada tabla de conversión encargada de interpretar su significado.

Existen tres tablas, y cuál es manejada, depende de si alguna de las teclas MAYS o CONTROL estaba activada conjuntamente con la última tecla pulsada. La primera se emplea si la tecla CONTROL estaba presionada. La segunda cuando MAYS pero no CONTROL, mientras que la tercera se utiliza si ninguna de las dos estaba activa. Los contenidos de estas tablas pueden ser modificados llamando a las

rutinas KM SET CONTROL, KM SET SHIFT y KM SET TRANSLATE, respectivamente.

El valor extraído de las tablas puede ser un código del sistema (token), un código de expansión (&80,&9F) o un carácter (&00,&7F o &A0,&FC). Existen tres códigos del sistema que son obedecidos inmediatamente cuando son encontrados. Se trata de &FF, el cual indica que la tecla presionada debe ser ignorada, &FE que provoca el intercambio de activado a desactivado o viceversa del modo shift lock (CONTROL+MAYS), y &FD, el cual bloquea/desbloquea el modo mayúsculas/minúsculas (CAPS LOCK).

Repetición Y Retardo

También existe una tabla, la cual acepta entradas por parte del usuario y que especifica qué teclas están autorizadas a repetirse. Puede ser alterada efectuando una llamada a la rutina KM SET REPEAT.

Tras la inicialización, los valores por defecto en esta tabla permiten la autorrepetición a todas las teclas excepto ESC, TAB, FIJA MAYS e INTRO. Tampoco tienen autorizada la repetición las teclas de función del teclado numérico.

La velocidad a la cual las teclas se repiten y el retardo hasta que se produce la primera repetición puede ser controlado mediante la rutina KM SET DELAY. Los valores por defecto producen 25 caracteres por segundo con un retardo de 6 décimas de segundo.

Para que la autorrepetición sea posible han de cumplirse las siguientes condiciones:

— El tiempo preciso ha pasado desde que la tecla fue por primera vez presionada o repetida.
— La tecla permanece pulsada.
— No se ha detectado la pulsación de ninguna nueva tecla.
— La tecla está autorizada a repetirse.
— No hay teclas almacenadas en el buffer.

El Control De Los Joysticks

Para la interpretación de las señales procedentes de los mandos a distancia se sigue el mismo sistema que si de teclas se tratara. Su

estado puede determinarse llamando a la rutina KM GET JOYSTICK.

En principio se averigua si la palanca fue desplazada o se pulsó el botón de disparo mediante la rutina KM TEST KEY. Seguidamente, estos valores son convertidos en caracteres utilizando las tablas de conversión adecuadas. En este punto surge un pequeño problema, pues la velocidad de repetición definida en el teclado puede ser muy inferior a la que se precisa en el joystick. No obstante, ésta puede ser elevada a riesgo de convertir el teclado en ingovernable.

RUPTURAS

Cuando en una exploración del teclado se detecta que la tecla ESC está pulsada, puede suceder la ruptura en la ejecución de un programa, siempre y cuando esté habilitada la rutina que la gestiona (KM ARM BREAK o KM DISARM BREAK).

Si la tecla ESC se encuentra presionada, se efectúa una llamada a KM TEST BREAK. Esta comprueba si CONTROL, MAYS y ESC han sido pulsadas simultáneamente. Si esto es así, el sistema se reinicializa ejecutando RST 0. En caso contrario, se invoca al mecanismo de ruptura.

TECLAS DE FUNCIÓN Y CÓDIGOS DE EXPANSIÓN

El controlador del teclado permite hasta 32 códigos de expansión (tokens), los cuales pueden ser definidos en las tablas de conversión. Cada uno está asociado por una cadena almacenada en un buffer, que puede ser definido empleando la rutina KM EXP BUFFER.

Este debe tener una capacidad de, por lo menos, 44 bytes, para permitir el almacenamiento de las cadenas de expansión. Estas últimas pueden ser definidas efectuando una llamada a KM SET EXPAND.

Habitualmente, todas estas operaciones es más fácil efectuarlas desde BASIC manejando los comandos KEY y KEY DEF, así como SPEED KEY, para asignar la velocidad de autorrepetición y el tiempo de retardo hasta que ésta se produce.

KM INITIALISE

Dirección: #BB00

Descripción: Realiza una reinicialización completa del controlador del teclado *(Key Manager),* perdiéndose su estado anterior. La indirección del controlador del teclado (KM TEST BREAK) toma el valor de su rutina por defecto, el buffer se reinicializa, así como el buffer de expansión, y las expansiones se fijan a las cadenas por defecto. Las tablas de correspondencia de tecla se inicializan igualmente, del mismo modo que los estados de repetición y sus tiempos. Finalmente, se suprimen los posibles bloqueos de mayúsculas y SHIFT. Tengamos en cuenta que esta rutina habilita las interrupciones.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL modificados.

KM RESET

Dirección: #BB03

Descripción: Reinicializa las indirecciones y buffers del controlador del teclado. La indirección del controlador del teclado (KM TEST BREAK) toma el valor de su rutina por defecto, el buffer se reinicializa, así como el buffer de expansión y las expansiones se fijan a las cadenas por defecto. Esta rutina también habilita las interrupciones.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL modificados.

KM WAIT OHAR

Dirección: #BB06

Descripción: Aguarda la llegada de un carácter desde el teclado o desde la cadena actual de expansión. Son varias las fuentes de las que se puede tomar el carácter y, por tanto, son investigadas siempre en un mismo orden:

— Caracteres retornados por la rutina KM CHAR RETURN.
— El próximo carácter de una cadena de expansión.
— El primer carácter de una cadena de expansión.
— Un carácter de una tabla de correspondencia de teclas.

Por otra parte, los *token* se expanden a sus cadenas correspon-

dientes cuando proceden de la tabla de teclas, pero se tratan como un carácter, con su código correspondiente, cuando se toman de cadenas de expansión.

Entrada: Sin parámetros.

Salida: Acarreo a 1 y el resto de las banderas modificadas. A contiene el carácter.

KM READ CHAR

Dirección: #BB09

Descripción: Toma un carácter desde el teclado o la cadena actual de expansión. A diferencia de KM WAIT CHAR, esta rutina no espera la llegada del carácter y retorna de inmediato si no existe ninguno disponible. Rigen las mismas fuentes de caracteres que para la rutina anterior.

Entrada: Sin parámetros.

Salida: Si existe algún carácter disponible, A contiene su código y la bandera de acarreo regresa a 1. De no ser así, A vuelve modificado y el acarreo a 0. En todo caso, se modifican el resto de las banderas.

KM CHAR RETURN

Dirección: #BB0C

Descripción: Envía un carácter al teclado, de forma que puede ser capturado posteriormente por cualquiera de las dos rutinas anteriores. Tengamos en cuenta que sólo es posible tener un carácter en este estado, de manera que si se envía otro desde esta rutina, sin haber leído previamente el anterior, éste se pierde; además, no es posible emplear el código 255, dado que se emplea como marca de «no existe carácter retornado».

Entrada: A contiene el carácter en cuestión.

Salida: Se conserva el valor de todos los registros y banderas.

KM SET EXPAND

Dirección: #BB0F

Descripción: Prepara la cadena de expansión asociada a un *token*. La cadena siempre debe residir en RAM y sus caracteres no son expandidos. Esta rutina habilita las interrupciones.

Entrada: B contiene el código de *token* a expandir, C la longitud de la cadena y HL la dirección de la misma.

Salida: Acarreo a 1 si la expansión se ha desarrollado correctamente; en caso contrario (cadena demasiado larga o *token* inválido), la bandera vuelve a 0. Se modifican el resto de las banderas y los registros A, BC, DE y HL.

KM GET EXPAND

Dirección: #BB12

Descripción: Toma un carácter de una cadena de expansión, los cuales comienzan a numerarse desde cero.

Entrada: A contiene el código de *token* y L el número de carácter.

Salida: Si existe el carácter, A contiene su código y la bandera de acarreo vuelve a 1. En caso contrario, A retorna modificado y la bandera de acarreo a 0. En todo caso, DE y el resto de las banderas se modifican.

KM EXP BUFFER

Dirección: #BB15

Descripción: Fija la dirección y longitud del buffer para la cadena de expansión. El buffer no debe encontrarse bajo una ROM y debe tener al menos una longitud de 44 caracteres. Si el nuevo buffer es demasiado corto, el antiguo no sufre alteración. La rutina habilita las interrupciones.

Entrada: DE contiene la dirección de inicio del buffer y HL su longitud.

Salida: Si el buffer está correctamente definido, la bandera de acarreo vuelve a 1, en caso contrario (es demasiado corto), a 0. En todo caso, A, BC, DE, HL y el resto de las banderas se modifican.

KM WAIT KEY

Dirección: #BB18

Descripción: Aguarda la pulsación de una tecla o la recoge del buffer del teclado.

Entrada: Sin parámetros.

Salida: A contiene el carácter o *token* y la bandera de acarreo vuelve a 1. El resto de las banderas se modifican.

KM READ KEY

Dirección: #BB1B

Descripción: Recoge una tecla del buffer del teclado, si es que existe alguna presente.

Entrada: Sin parámetros.

Salida: A contiene el carácter o *token* y la bandera de acarreo a 1, siempre y cuando existiera alguna tecla disponible en el momento de la ejecución; de no ser así, la bandera de acarreo retorna a 0 y A modificado. El resto de banderas se modifican.

KM TEST KEY

Dirección: #BB1E

Descripción: Comprueba si una tecla determinada o botón del joystick se halla pulsado. Los estados de MAYS (SHIFT) y CONTROL se denotan con los bits 5 y 7, respectivamente, a 1.

Entrada: A contiene el código de tecla.

Salida: Si la tecla está presionada se retorna la bandera de cero a 1; y en caso contrario a 0. Siempre vuelve la bandera de acarreo a 0, C contiene el estado de MAYS (SHIFT) y CONTROL y los registros A y HL, así como el resto de las banderas, se modifican.

KM GET STATE

Dirección: #BB21

Descripción: Comprueba el estado del teclado en lo referente al bloqueo de mayúsculas y SHIFT. #FF significa bloqueo habilitado y 0 desactivado (estado por defecto).

Entrada: Sin parámetros.

Salida: H retorna el estado de bloqueo de mayúsculas y L el de SHIFT. Sólo AF se modifica.

KM GET JOYSTICK

Dirección: #BB24

Descripción: Investiga la situación actual de los joysticks. Esta lectura no se efectúa directamente en el hardware, sino en el mapa del

estado del teclado, actualizado aproximadamente cada cincuentavo de segundo. El significado de los bits de la lectura es el siguiente: 0 arriba, 1 abajo, 2 izquierda, 3 derecha, 4 fuego 2, 5 fuego 1, 6 botón de reserva (normalmente desconectado) y el bit 7 siempre es cero. Los bits a 1 denotan la activación del interruptor correspondiente.

Entrada: Sin parámetros.

Salida: A y H contienen el estado del joystick 0 y L el del 1. Las banderas se modifican.

KM SET TRANSLATE

Dirección: #BB27

Descripción: Prefija el carácter o *token* correspondiente a una tecla, al ser esta pulsada en ausencia de CONTROL y MAYS (shift). Si el código no es válido (mayor que 79) no se lleva a cabo ninguna acción.

Entrada: A contiene un número de tecla y B su nuevo código asociado.

Salida: AF y HL se modifican.

KM GET TRANSLATE

Dirección: #BB2A

Descripción: Investiga qué carácter o *token* corresponde a una tecla, cuando esta es pulsada sin CONTROL ni MAYS (SHIFT). La validez del código no es comprobada, de forma que en caso de ser mayor que 79 la traducción efectuada carece de significado real.

Entrada: A contiene un número de tecla.

Salida: A contiene el código asociado y el par HL y las banderas se modifican.

KM SET SHIFT

Dirección: #BB2D

Descripción: Fija qué carácter o *token* corresponderá a una tecla cuando esta sea pulsada sin CONTROL, pero sí con MAYS (SHIFT) o con el teclado en estado de bloqueo de mayúsculas. Si el código no es válido (mayor que 79) no se realiza ninguna acción.

Entrada: A contiene un número de tecla y B el nuevo código asociado.

Salida: AF y HL se modifican.

KM GET SHIFT

Dirección: #BB30

Descripción: Investiga qué carácter o *token* corresponde a una tecla cuando se pulsa sin CONTROL, pero sí con MAYS (SHIFT) o con el teclado en estado de bloqueo de mayúsculas. La validez del código no es comprobada, de forma que en caso de ser mayor que 79 la traducción efectuada carece de significado real.

Entrada: A contiene un código de tecla.

Salida: A es el código asociado y HL y las banderas se modifican.

KM SET CONTROL

Dirección: #BB33

Descripción: Fija el carácter o *token* correspondiente a una tecla pulsada simultáneamente con CONTROL. Si el código no es válido (mayor que 79) no se realiza ninguna acción.

Entrada: A contiene el número de tecla y B el código asociado.

Salida: AF y HL se modifican.

KM GET CONTROL

Dirección: #BB36.

Descripción: Investiga qué carácter o *token* corresponde a una determinada tecla cuando es pulsada simultáneamente con CONTROL. La validez del código no es comprobada, de forma que en caso de ser mayor que 79 la traducción efectuada carece de significado real.

Entrada: A contiene el código de tecla.

Salida: A contiene el código asociado y HL y las banderas se modifican.

KM SET REPEAT

Dirección: #BB39

Descripción: Determina la posibilidad de repetición de una tecla. Si el código no es válido (mayor que 79) no se realiza ninguna acción.

Entrada: A contiene la tecla a afectar; B es #FF si se activa la repetición o 0 en caso contrario.

Salida: AF, BC y HL se modifican.

KM GET REPEAT

Dirección: #BB3C

Descripción: Investiga el estado de repetición de una determinada tecla. La validez del código no es comprobada, de forma que en caso de ser mayor que 79 la traducción efectuada carece de significado real.

Entrada: A contiene el número de tecla.

Salida: La bandera de cero retorna a 0 si la tecla tiene repetición, o a 1 en caso contrario. Por otra parte, la bandera de acarreo vuelve siempre a 0 y A y HL, así como el resto de las banderas, se modifican.

KM SET DELAY

Dirección: #BB3F

Descripción: Ajusta el tiempo de comienzo de repetición y el transcurrido entre interacciones. Este tiempo se mide en accesos a la lectura del teclado, es decir, cincuentavos de segundo, tomándose el valor 0 como 256.

Entrada: H contiene el tiempo inicial y L el intervalo entre repeticiones.

Salida: AF se modifica.

KM GET DELAY

Dirección: #BB42

Descripción: Investiga el estado de los parámetros fijados por la

rutina anterior. Este tiempo se mide en accesos a la lectura del teclado, es decir, cincuentavos de segundo, tomándose el valor 0 como 256.

Entrada: Sin parámetros.

Salida: H contiene el tiempo inicial y L el intervalo entre repeticiones.

KM ARM BREAKS

Dirección: #BB45

Descripción: Conecta el mecanismo de BREAK (interrupción por ESC), para que tenga efecto la rutina KM BREAK EVENT. Esta rutina habilita las interrupciones.

Entrada: DE contiene la dirección de la rutina de tratamiento y C la dirección de selección de ROM.

Salida: AF, BC, DE y HL se modifican.

KM DISARM BREAKS

Dirección: #BB48

Descripción: Deshabilita el mecanismo de BREAK. Esta rutina habilita las interrupciones.

Entrada: Sin parámetros.

Salida: AF y HL se modifican.

KM BREAK EVENT

Dirección: #BB4B

Descripción: Genera una interrupción por BREAK (ESC), siempre y cuando se halle habilitado este mecanismo. De producirse la interrupción, se incluye en el buffer el *token* #EF, procediéndose de inmediato a la deshabilitación del mecanismo, para evitar que se sucedan interrupciones de interrupciones.

Entrada: Sin parámetros.

Salida: AF y HL se modifican.

VISUALIZADOR DE TEXTO

as rutinas de control de la pantalla de texto (TXT VDU) constituyen un bloque del firmware del Amstrad, el cual abarca desde la dirección #BB4E a la #BBB9. Su labor principal consiste tanto en imprimir caracteres en la pantalla, como en leer lo que allí se encuentra escrito.

Estas rutinas están capacitadas para imprimir a través de 8 canales independientes, 256 caracteres diferentes sobre la pantalla, si bien los 32 primeros (0 a 31) son interpretados usualmente como códigos de control, con el siguiente significado:

Cód.	Núm. de parám.	Descripción
0	0	Sin efecto.
1	1	Escribe el carácter indicado como parámetro, aunque éste sea un código de control (0 a 31).
2	0	Deshabilita la presentación del cursor.
3	0	Habilita la presentación del cursor.
4	1	Pasa la pantalla al modo indicado por el parámetro.
5	1	Escribe el carácter indicado como parámetro utilizando el GRAPHIC VDU.

Cód.	Núm. de parám.	Descripción
6	0	Habilita el TEXT VDU.
7	0	Emite un pitido.
8	0	Una vez que hace válida la posición del cursor, retrocede un carácter.
9	0	Una vez que hace válida la posición del cursor, avanza un carácter.
10	0	Una vez que hace válida la posición del cursor, avanza una línea.
11	0	Una vez que hace válida la posición del cursor, retrocede una línea.
12	0	Borra la ventana actual y desplaza el cursor a su posición de origen (esquina superior izquierda de la ventana).
13	0	Una vez que hace válida la posición del cursor, lo desplaza a la primera columna de la misma línea de la ventana donde se encuentra.
14	1	Establece el color de tinta para fondo al valor proporcionado por el parámetro.
15	1	Establece el color de tinta para primer término al valor proporcionado por el parámetro.
16	0	Una vez que hace válida la posición del cursor, la borra con la tinta de fondo actual.
17	0	Una vez que hace válida la posición del cursor, borra con la tinta de fondo actual las comprendidas entre ésta y la primera columna de la misma línea de la ventana.
18	0	Una vez que hace válida la posición del cursor, borra con la tinta de fondo actual las comprendidas entre ésta y la última columna de la misma línea de la ventana.
19	0	Una vez que hace válida la posición del cursor, borra con la tinta de fondo actual las comprendidas entre ésta y la esquina superior izquierda de la ventana.
20	0	Una vez que hace válida la posición del cursor, borra con la tinta de fondo actual las comprendidas entre ésta y la esquina inferior derecha de la ventana.
21	0	Deshabilita el text VDU.
22	1	Establece el modo de escritura según el parámetro (1=Opaco; 2=Transparente).
23	1	Establece el modo de escritura de GRAPHIC VDU según el parámetro (1=Opaco; 2=XOR; 3=AND; 4=OR).

Cód.	Núm. de parám.	Descripción
24	0	Intercambia los colores actuales de tinta para fondo y primer término.
25	9	Establece la forma de carácter definido. El primer parámetro corresponde al código de carácter a asignar y los ocho restantes a los bytes que configuran su matriz, empezando por la primera línea.
26	4	Establece los límites de la ventana de texto. Los dos primeros parámetros corresponden a las columnas izquierda y derecha (se selecciona el valor menor para la izquierda) y los dos siguientes a las filas (se selecciona el valor menor para la superior).
27	0	Sin efecto especial.
28	3	Establece los colores a emplear en una tinta. El primer parámetro es el número de tinta y los dos siguientes los colores (si son iguales el color no es intermitente).
29	2	Establece los colores para el marco de la pantalla según sus dos parámetros (si son iguales no se produce intermitencia).
30	0	Desplaza la posición actual del cursor a origen de la ventana (esquina superior izquierda).
31	2	Desplaza el cursor a las coordenadas de la ventana proporcionadas por los parámetros, en la forma columna y fila, respectivamente, teniendo en cuenta que la posición de origen de la ventana se considera la 1,1.

CORRIENTES Y VENTANAS

Cada corriente lleva asociado un área de pantalla sobre el cual imprimir cuando ésta sea seleccionada. A esta zona se la conoce como ventana de texto. Al principio, durante la inicialización, el Sistema asigna a todas las corrientes las mismas condiciones de partida, es decir, selecciona su ventana asociada al tamaño de la pantalla completa, habilita la aparición del cursor de texto y lo sitúa en la esquina superior izquierda. Además, selecciona el papel número 0 (azul), la pluma número 1 (amarillo brillante) y el modo de escritura opaco, de forma que todo lo que a continuación se escriba aparezca bajo estas condiciones.

Finalmente, tan solo le resta seleccionar una de las corrientes, y

asume por defecto la número 0; a menos que variemos esta situación, toda la información emitida hacia la pantalla lo hará a través de esta corriente. Tras la inicialización, podemos modificar todas estas condiciones de partida, valiéndonos de las rutinas firmware habilitadas a tal fin; por ejemplo, cambiando el tamaño de todas las ventanas.

La más pequeña que es posible definir abarca un único carácter, es decir, mide una sola columna, y otro aspecto curioso de estas zonas de presentación, es que cuando varias ventanas se solapan, o lo que es lo mismo, coinciden unas sobre otras, al no existir preferencias de impresión, cualquier scroll (desplazamiento de la información escrita en ellas) que las afecte, arrastrará igualmente las zonas que estuvieran debajo.

El Cursor

El cursor es un bloque compacto que abarca completamente el espacio destinado a un carácter, y es representado en modo invertido respecto del estado asignado en ese momento a la ventana de texto (en cuanto a atributos de color de pluma y papel) donde ha de aparecer.

Es posible evitar que se imprima sobre la ventana en cuestión, utilizando la rutina TXT CUR OFF, la cual lo deshabilita. Para restablecer la situación inicial nos valdremos de TXT CUR ON.

Cada canal lleva asociado una posición actual para cursor. Esta es la que ocupará el siguiente carácter al ser impreso en esta zona de la pantalla. De cuatro formas podemos alterar la posición del cursor: a través de las rutinas TXT SET CURSOR, TXT SET ROW y TXT SET COLUMN, o enviando códigos de control de la pantalla de texto. También es factible cambiar su forma a través de las rutinas TXT DRAW CURSOR y TXT UNDRAW CURSOR.

Caracteres

Un carácter se representa en la pantalla como un grupo de 8 puntos (pixels) de ancho por 8 de alto. Por tanto, el número máximo de carácteres sobre la pantalla está en relación directa con el modo de ésta.

Modo	Puntos	Caracteres
0	160 × 200	20 × 25
1	320 × 200	40 × 25
2	640 × 200	80 × 25

Cada carácter lleva asociado una «matriz» o grupo de 8 bytes que determinan su forma. El primer byte corresponde a la fila superior, mientras el último, a la de más abajo. Dentro de esta fila particular, el punto situado más a la izquierda es el bit más significativo. Cuando un bit se encuentra a 1 se escribe en color de la pluma asignada al canal, mientras que si está a 0, es escrito del color del papel, siempre y cuando el modo de impresión sea el opaco.

En modo transparente, los pixels que ocupan la posición actual del cursor no son modificados, y el nuevo carácter a ser impreso, se «sobreimprime» en el anterior, facilitando la creación de caracteres compuestos. Las matrices para los carácteres están normalmente almacenadas en la ROM, pero podemos trasladarlas a la RAM donde modificarlas.

Así, por ejemplo, cuando efectuamos la lectura de un carácter impreso en la pantalla (COPYCHAR), el firmware lo convierte a la forma de matriz y lo compara con el juego que actualmente tenga almacenado para determinar de cuál se trata.

IMPRESIÓN DE CARACTERES

La rutina principal asociada a la impresión de caracteres es TXT OUTPUT, la cual obedece los códigos de control entre 0 y 31, e imprime los restantes carácteres entre 32 y 255. Para ello, llama a TXT OUT ACTION, perteneciente al grupo de rutinas clave (INDIRECTIONS) del TXT VDU, la cual efectúa el trabajo de impresión, determinando previamente, si el carácter es imprimible, si se trata de un código de control o si es uno de los parámetros de algún código de control.

Cualquier carácter escrito ha de pasar previamente a través de un buffer de control intermedio, con capacidad para 9 parámetros, que es compartido por todas las corrientes. Por ello, no debe cambiarse la corriente seleccionada en la actualidad a otra diferente, hasta que una secuencia enviada al buffer no haya sido completada. De no respetar este principio, los resultados son impredecibles.

Los códigos de control aceptados son los estándar ASCII, pero en nuestras manos obra la posibilidad de cambiar su forma de operar aprovechando la rutina TXT GET CONTROLS. En caso de duda

sobre el estado del buffer o los códigos de control, TXT RESET los inicializa a los valores por defecto. La impresión de un carácter atraviesa las siguientes etapas:

1. Se envía hacia el buffer un código de control y se averigua el número de parámetros que requiere.
2. Si son precisos más parámetros se sigue el punto 4.
3. Los siguientes carácteres son enviados hacia el buffer, hasta que sean impresos u obedecidos.
4. Se consigue la dirección a la cual llamar para procesar el código y es entonces ejecutada.
5. Se vacía el buffer y se repite el mismo proceso.

Como señalábamos anteriormente, el cometido de un código de control puede ser alterado mediante la llamada a la rutina TXT GET CONTROLS. Esta contiene una entrada de 3 bytes, donde el primero debe almacenar el número de parámetros y los dos siguientes la dirección de llamada para ejecutarlo. Para deshabilitar una corriente de texto se emplea la rutina TXT VDU ENABLE. Por último, señalar que una llamada a cualquiera de estas dos últimas subrutinas provoca la limpieza del buffer, circunstancia que puede ser aprovechada cuando su contenido es desconocido, por ejemplo, tras la impresión de un mensaje de error.

TXT INITIALISE

Dirección: #BB4E

Descripción: Realiza una reinicialización completa de la VDU de texto *(Video Display Unit,* Unidad Visualizadora). Sus indirecciones (TXT DRAW CURSOR, TXT UNDRAW CURSOR, TXT WRITE CHAR, TXT UNWRITE CHAR y TXT OUT ACTION) toman los valores de sus rutinas por defecto, la tabla de códigos de control toma asimismo sus valores por defecto, la tabla de caracteres definidos por el usuario se vacía y se selecciona la corriente *(stream)* 0. Todas las corrientes toman sus valores por defecto: fondo a tinta 0, primer plano a tinta 1, ventana de texto para toda la pantalla, cursor habilitado aunque apagado, modo de escritura opaco, VDU habilitado, modo gráfico desconectado y cursor a la esquina superior izquierda de la ventana.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL modificados.

TXT RESET

Dirección: #BB51
Descripción: Reinicializa las indirecciones de la VDU de texto y la tabla de caracteres de control.
Entrada: Sin parámetros.
Salida: AF, BC, DE y HL modificados.

TXT VDU ENABLE

Dirección: #BB54
Descripción: Habilita la escritura de caracteres en la pantalla mediante TXT OUTPUT y TXT WR CHAR, por la corriente seleccionada, así como el parpadeo del cursor, mediante la llamada a TXT CUR ENABLE. El buffer de caracteres de control utilizado por TXT OUTPUT se vacía.
Entrada: Sin parámetros.
Salida: AF modificado.

TXT VDU DISABLE

Dirección: #BB57
Descripción: Tiene el efecto contrario a la rutina anterior, aplicándose igualmente a la corriente seleccionada y el cursor.
Entrada: Sin parámetros.
Salida: AF modificado.

TXT OUTPUT

Dirección: #BB5A
Descripción: Emite un carácter o código de control a la corriente actual. La rutina utiliza la indirección TXT OUT ACTION. El buffer para códigos de control sólo admite 9 parámetros. Si se halla habilitado el modo gráfico se utiliza para la escritura la rutina GRA WR CHAR.
Entrada: A contiene el carácter a enviar.
Salida: No hay modificaciones.

TXT WR CHAR

Dirección: #BB5D
Descripción: Escribe un carácter en la posición de pantalla señalada por el cursor de la corriente actual. Los caracteres de control (de 0 a #1F) se imprimen y no obedecen. La rutina actualiza la posición del cursor y comprueba antes de escribir que este se halle en una posición válida (TXT VALIDATE).
Entrada: A contiene el carácter a escribir.
Salida: AF, BC, DE y HL son modificados.

TXT RD CHAR

Dirección: #BB60
Descripción: Lee un carácter de la pantalla, en la posición del cursor de la corriente actual. La lectura del carácter se efectúa por comparación con el área generadora de los mismos.
Entrada: Sin parámetros.
Salida: Si se ha reconocido el carácter, el código de éste retorna en A y la bandera de acarreo aparece a 1; en caso contrario, A contiene 0 y la bandera de acarreo se halla a 0. En todo caso, se modifican el resto de las banderas.

TXT SET GRAPHIC

Dirección: #BB63
Descripción: Habilita o deshabilita el modo gráfico en la corriente actual.
Entrada: A distinto de 0 para habilitar y A igual a 0 para deshabilitar.
Salida: AF modificado.

TXT WIN ENABLE

Dirección: #BB66
Descripción: Delimita la ventana en la corriente actual. Es preciso señalar la primera y última columna y fila de la ventana, tomadas en coordenadas absolutas desde la esquina superior izquierda de la pantalla (0,0). Se toma como primera columna el valor más pequeño de

H y D, y como primera fila, la menor de L y E. Aunque la ventana definida no se borra, el cursor sí se desplaza a su esquina superior izquierda.

Entrada: H y D contienen los valores para la primera y última columnas (no importa el orden) y L y E, los de las filas.

Salida: AF, BC, DE y HL se modifican.

TXT GET WINDOW

Dirección: #BB69

Descripción: Averigua el tamaño de la ventana actual en la corriente seleccionada, especificando si esta cubre la totalidad de la pantalla. El sistema de coordenadas empleado para informar de la posición es igual que el de la rutina anterior.

Entrada: Sin parámetros.

Salida: Si la ventana cubre toda la pantalla, la bandera de acarreo se hace 0; en caso contrario, 1. Siempre se retornan los valores de la primera columna en H, la primera fila en L, la última columna en D y la última fila en E, así como A modificado.

TXT CLEAR WINDOW

Dirección: #BB6C

Descripción: Borra la ventana seleccionada de la corriente actual, utilizando para ello el color de fondo correspondiente y fijando la posición del cursor a la esquina superior izquierda de la ventana.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL se modifican.

TXT SET COLUMN

Dirección: #BB6F

Descripción: Desplaza el cursor a cualquier columna de la ventana actual en la corriente seleccionada; en caso de encontrarse el cursor habilitado y encendido, se dibuja en su nueva posición, proporcionándose ésta en coordenadas lógicas con respecto a la ventana; es decir, la primera columna de la ventana corresponde al valor 1.

Entrada: A contiene la columna.

Salida: AF y HL se modifican.

TXT SET ROW

Dirección: #BB72
Descripción: Tiene función equivalente a la rutina anterior, pero aplicada a las filas.
Entrada: A contiene la fila.
Salida: AF y HL se modifican.

TXT SET CURSOR

Dirección: #BB75
Descripción: Combina la acción de las dos rutinas anteriores.
Entrada: H contiene la columna y L la fila.
Salida: AF y HL se modifican.

TXT GET CURSOR

Dirección: #BB78
Descripción: Investiga la posición actual del cursor en la ventana de la corriente seleccionada, así como el número de veces que ésta se ha desplazado. Para la correcta interpretación de este último dato, hay que tener en cuenta que el contador se decrementa cada vez que la ventana se desplaza hacia arriba y viceversa, motivo por el cual sólo toma auténtico sentido cuando se le compara con un valor obtenido previamente. Por otra parte, las coordenadas empleadas para obtener la posición del cursor son las habituales en las últimas rutinas.
Entrada: Sin parámetros.
Salida: H y L contienen la columna y fila, respectivamente y A el contador de scrolls. Las banderas se modifican.

TXT CUR ENABLE

Dirección: #BB7B
Descripción: Habilita la presentación del cursor en la posición correspondiente de pantalla por la corriente seleccionada. La acción tomará efecto inmediato siempre que el cursor se encuentre encendido.
Entrada: Sin parámetros.
Salida: AF se modifica.

TXT CUR DISABLE

Dirección: #BB7E
Descripción: Contrarresta la acción de la rutina anterior, haciendo desaparecer inmediatamente la presentación del cursor de la pantalla, si es que se hallaba presente.
Entrada: Sin parámetros.
Salida: AF se modifica.

TXT CUR ON

Dirección: #BB81
Descripción: Enciende el cursor, permitiendo su presentación en la posición correspondiente de pantalla por la corriente seleccionada. La acción tomará efecto inmediato siempre que el cursor se encuentre habilitado.
Entrada: Sin parámetros.
Salida: No se producen modificaciones.

TXT CUR OFF

Dirección: #BB84
Descripción: Contrarresta la acción de la rutina anterior, haciendo desaparecer inmediatamente la representación del cursor de la pantalla, si es que se hallaba presente.
Entrada: Sin parámetros.
Salida: No se producen modificaciones.

TXT VALIDATE

Dirección: #BB87
Descripción: Comprueba si una determinada posición de la pantalla tiene valor apropiado para la ventana actual. Las coordenadas se proporcionan por el sistema habitual (respecto a la ventana, siendo 1,1 la esquina superior izquierda). En caso de que la posición no se considere válida, se devuelve una nueva posición admisible, basándose para ello en las normas siguientes:

— Si la posición sobrepasa por la derecha el margen derecho, se retorna a la primera columna de la ventana en su siguiente fila.
— Si rebasa la última fila, se desplaza la pantalla hacia arriba, quedando fijada la posición a la misma columna de la última fila.
— Si permanece a la izquierda del margen izquierdo, se desplaza a la última columna de la fila anterior.
— Si queda por encima de la primera fila, se desplaza la pantalla hacia abajo, permaneciendo en la misma columna, pero de la primera fila de la ventana.

Entrada: H y L contienen la columna y fila, respectivamente, determinantes de la posición a comprobar.

Salida: Si no se produce desplazamiento de la pantalla, la bandera de acarreo vuelve a 1 y B se modifica. Si se produce desplazamiento, la bandera de acarreo es 0, pero B contiene #FF para scrolls hacia arriba y #00 para abajo. En todo caso, H y L contendrán, respectivamente, la columna y fila correspondiente a la posición válida, modificándose A y el resto de las banderas.

TXT PLACE CURSOR

Dirección: #BB8A

Descripción: Representa en la pantalla un marcador de cursor, en la posición correspondiente de la corriente actual. Dicho marcador se obtiene operando un or exclusivo del contenido de la pantalla en la posición del cursor con el or exclusivo de los colores de fondo y primer término. La rutina permite representar varios cursores en una ventana, teniendo el usuario que cuidar de su representación, dado que las rutinas de alto nivel TXT OUTPUT o TXT SET CURSOR se ocupan del manejo del cursor del sistema. Por otra parte, no se debe llamar dos veces seguidas a esta rutina, no sin antes haber pasado por la de borrado del cursor (TXT REMOVE CURSOR), dado que se desvirtuaría la información oculta bajo éste.

Entrada: Sin parámetros.

Salida: AF se modifican.

TXT REMOVE CURSOR

Dirección: #BB8D

Descripción: Se utiliza en combinación con la rutina anterior, para

hacer desaparecer el cursor, volviendo a su estado la información de la pantalla que ocultaba.

Entrada: Sin parámetros.

Salida: AF se modifica.

TXT SET PEN

Dirección: #BB90

Descripción: Selecciona la tinta para el primer plano de texto. Previamente, se asegura que el valor sea legal válido, es decir, que se halle dentro del margen permitido por el modo de pantalla.

Entrada: A contiene la tinta a emplear.

Salida: AF y HL se modifican.

TXT GET PEN

Dirección: #BB93

Descripción: Averigua la tinta seleccionada para el primer término en la presentación de caracteres en la ventana de la corriente actual.

Entrada: Sin parámetros.

Salida: A contiene la tinta y las banderas se modifican.

TXT SET PAPER

Dirección: #BB96.

Descripción: Su efecto es en todo similar a TXT SET PEN, pero referido al fondo.

Entrada: A contiene la tinta a emplear.

Salida: AF y HL se modifican.

TXT GET PAPER

Dirección: #BB99

Descripción: Actúa de forma análoga a TXT GET PEN, pero referida al fondo.

Entrada: Sin parámetros.

Salida: A contiene la tinta y las banderas se modifican.

TXT INVERSE

Dirección: #BB9C
Descripción: Intercambia las tintas de fondo y primer término seleccionadas para la corriente actual.
Entrada: Sin parámetros.
Salida: AF y HL se modifican.

TXT SET BACK

Dirección: #BB9F
Descripción: Selecciona el modo de escritura opaca (representación del color de fondo junto con el carácter) o transparente (representación del primer término, sin afectar el estado de la pantalla en los puntos correspondientes al fondo).
Entrada: A igual a 0 implica modo opaco y distinto de cero, transparente.
Salida: AF y HL se modifican.

TXT GET BACK

Dirección: #BBA2
Descripción: Averigua el modo de escritura para la corriente seleccionada (opaco o transparente).
Entrada: Sin parámetros.
Salida: A contiene 0 para el modo opaco y cualquier valor distinto de cero para el transparente. DE, HL y las banderas se modifican.

TXT GET MATRIX

Dirección: #BBA5
Descripción: Averigua la posición de comienzo de la matriz de un determinado carácter, así como si se trata de un carácter definido por el usuario. La tabla generadora de caracteres se puede hallar tanto en RAM como en ROM, suponiendo la rutina que se ha seleccionado el banco de memoria oportuno. La matriz de cada carácter se halla compuesta por ocho bytes consecutivos, cada uno de los cuales correspon-

de a las líneas primera a la octava, respectivamente, que lo configuran. Así pues, la forma de un carácter se delimita como un cuadrado de 8 puntos de lado (cada byte tiene 8 bits), de forma que los pixels encendidos (color de primer término) se denotan con bits a 1 y los apagados (fondo) a 0. El bit se considera el correspondiente al lado derecho del carácter.

Entrada: A contiene el carácter a localizar en la tabla generadora.

Salida: Si el carácter es definible por el usuario, la bandera de acarreo retorna a 1 y si el generador de caracteres se halla en la ROM baja a 0. En todo caso, HL contiene la dirección base (primer byte o línea del carácter) de la matriz a buscar y A y el resto de las banderas se modifican.

TXT SET MATRIX

Dirección: #BBA8

Descripción: Fija una nueva matriz para un carácter definible por el usuario.

Entrada: A contiene el carácter al cual corresponde la nueva matriz cuya base es HL.

Salida: Si el carácter no es definible por el usuario, la rutina no tendrá efecto y regresará con la bandera de acarreo a 1 (acarreo a 0 en caso contrario). En todo caso, A, BC, DE, HL y el resto de las banderas se modifican.

TXT SET M TABLE

Dirección: #BBAB

Descripción: Fija la base de la tabla generadora de caracteres de usuario y su longitud. Si el primer carácter especificado se halla en el margen 0 a 255, las matrices comprendidas entre ese valor y el final se copian a la nueva tabla definida por el usuario. De no ser así, se considera que la tabla no va a contener matrices. En cuanto a su longitud, será de 256 menos el primer carácter, todo ello multiplicado por 8.

Entrada: DE contiene el primer carácter de la tabla y HL la dirección de comienzo de la nueva tabla.

Salida: Si no existía ninguna tabla de caracteres definidos por el usuario anterior la bandera de acarreo vuelve a 0 y A y HL se modifi-

can; en caso contrario, la bandera de acarreo retorna a 1, A contiene el primer carácter de la tabla antigua y HL la dirección base antigua. En todo caso, BC, DE y el resto de las banderas se modifican.

TXT GET M TABLE

Dirección: #BBAE

Descripción: Obtiene la dirección base de la actual tabla de caracteres definidos por el usuario, así como el primer carácter de la misma.

Entrada: Sin parámetros.

Salida: Si no existe tabla de caracteres definidos por el usuario, la bandera de acarreo queda a 0 y A y HL se modifican; en caso contrario, la bandera de acarreo retorna a 1. A contiene el primer carácter de la tabla y HL su dirección base. En todo caso, el resto de las banderas se modifican.

TXT GET CONTROLS

Dirección: #BBB1

Descripción: Averigua la dirección de la tabla de caracteres de control. Dicha tabla tiene las entradas correspondientes a los códigos de control de #00 a #1F, con tres bytes para cada una de ellas, cuyo significado es el siguiente: el primer byte indica el número de parámetros necesario para el código y los otros dos, la dirección de la rutina que se debe llamar para ejecutar la función del mismo, una vez que se hallan recibido todos los parámetros, lo cuales, como ya hemos visto anteriormente, no pueden ser nunca más de nueve. La rutina de tratamiento debe hallarse en las 32 K centrales de RAM y debe cumplir las siguientes normas:

— A la entrada, A debe contener el último carácter añadido al buffer, B la longitud del buffer (incluido el código de control), C el mismo valor que A y HL la dirección del buffer de códigos de control.

— A la salida, sólo AF, BC, DE y HL se modifican.

Entrada: Sin parámetros.

Salida: HL contiene la dirección base de la tabla de caracteres de control.

TXT STR SELECT

Dirección: #BBB4

Descripción: Selecciona una corriente para la VDU de texto, asegurándose que quede en el rango admisible (máscara con #07). Muchos atributos de la VDU de texto pueden asignarse independientemente de la corriente seleccionada. No obstante, es importante asegurarse que la corriente seleccionada es la adecuada antes de alterar dichos atributos: tinta para primer término, tinta para fondo, posición del cursor, límite de ventana, habilitación/deshabilitación del cursor, cursor encendido/apagado, VDU habilitada/deshabilitada, modo de escritura de carácter y modo gráfico.

Entrada: A contiene la corriente.

Salida: A contiene la corriente previamente seleccionada. HL se modifica.

TXT SWAP STREAMS

Dirección: #BBB7

Descripción: Intercambia los estados de dos corrientes, aunque sin modificar el número de corriente actual. Los atributos intercambiados son los mismos que se exponen en la rutina anterior.

Entrada: B y C contienen los números de corriente a intercambiar.

Salida: AF, BC, DE y HL se modifican.

EL MODO GRÁFICO

Este bloque de rutinas del firmware, cuyos saltos se comprenden entre las direcciones #BBBA y #BBFC, tienen como finalidad el control de la pantalla en su modo gráfico, es decir, desde el punto de vista de cada uno de los pixels que la integran.

GRA INITIALISE

Dirección: #BBBA.

Descripción: Realiza una reinicialización completa del modo gráfico de la VDU *(Video Display Unit,* Unidad de Visualización), perdiéndose su estado anterior. Todas sus variables e indirecciones toman el valor de su rutina por defecto (GRA PLOT, GRA TEST y GRA LINE), fija a 0 el valor de tinta para fondo gráfico, a 1 el de primer término (gráfico), el origen de coordenadas a la esquina inferior izquierda de la pantalla, desplazando el cursor gráfico a dicha posición y fija la ventana gráfica a las dimensiones de la totalidad de la pantalla, aunque sin borrarla.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL modificados.

GRA RESET

Dirección: #BBBD
Descripción: Reinicializa las indirecciones de la VDU gráfica a sus rutinas por defecto (GRA PLOT, GRA TEST y GRA LINE).
Entrada: Sin parámetros.
Salida: AF, BC, DE y HL se modifican.

GRA MOVE ABSOLUTE

Dirección: #BBC0
Descripción: Desplaza el cursor a una posición absoluta, relativa al origen de coordenadas.
Entrada: DE contiene la coordenada X y HL la Y.
Salida: AF, BC, DE yHL se modifican.

GRA MOVE RELATIVE

Dirección: #BBC3
Descripción: Desplaza el cursor a una posición relativa con respecto a la posición actual del mismo.
Entrada: DE contiene el desplazamiento en el eje X y HL el desplazamiento en el eje Y.
Salida: AF, BC, DE y HL se modifican.

GRA ASK CURSOR

Dirección: #BBC6
Descripción: Averigua la posición actual del cursor gráfico, en coordenadas relativas al origen.
Entrada: Sin parámetros.
Salida: DE contiene la coordenada X y HL la Y y AF se modifica.

GRA SET ORIGIN

Dirección: #BBC9
Descripción: Establece el punto origen de coordenadas, relativo al

origen absoluto 0,0 (esquina inferior izquierda de la pantalla), desplazando al nuevo origen el cursor gráfico.

Entrada: DE contiene la coordenada X y HL la Y.

Salida: AF, BC, DE y HL se modifican.

GRA GET ORIGIN

Dirección: #BBCC

Descripción: Obtiene las coordenadas del origen de coordenadas, obviamente consideradas con respecto al origen absoluto de la pantalla 0,0 (esquina inferior izquierda).

Entrada: Sin parámetros.

Salida: DE contiene la coordenada X y HL la Y.

GRA WIN WIDTH

Dirección: #BBCF

Descripción: Establece la anchura de la ventana gráfica, fijando su primera y última columna, tomando como origen de coordenadas el absoluto de la pantalla 0,0 (esquina inferior izquierda). Hay que tener en cuenta que las coordenadas se hacen coincidir con un byte completo, por lo que dependiendo del modo de pantalla, sus valores serán:

Modo	Margen izquierdo	Margen Derecho
0	Múltiplo de 2	Múltiplo de 2 menos 1
1	Múltiplo de 4	Múltiplo de 4 menos 1
2	Múltiplo de 8	Múltiplo de 8 menos 1

En caso de que los valores proporcionados no se ajusten a esta norma, la rutina se encargará de validarlos, para lo cual aproximará el margen izquierdo al más cercano por su izquierda y el derecho al más cercano por la derecha.

En todo caso, la rutina considerará automáticamente el punto más hacia la izquierda como el margen izquierdo y el mayor como el derecho, de manera que es indiferente el orden en que se proporcionen los parámetros.

Entrada: DE contiene una coordenada horizontal y HL la otra.

Salida: AF, BC, DE y HL se modifican.

GRA WIN HEIGHT

Dirección: #BBD2
Descripción: Establece la altura de la ventana gráfica, fijando su primera y última fila, tomando como origen de coordenadas el absoluto de la pantalla 0,0 (esquina inferior izquierda).

En todo caso, la rutina considerará automáticamente el punto más hacia arriba como el margen superior, y el menor como el inferior, de manera que es indiferente el orden en que se proporcionen los parámetros.

Entrada: DE contiene una coordenada vertical y HL la otra.
Salida: AF, BC, DE y HL se modifican.

GRA GET W WIDTH

Dirección: #BBD5
Descripción: Averigua las coordenadas de anchura de la ventana gráfica con respecto al origen absoluto 0,0 de la pantalla (esquina inferior izquierda).

Entrada: Sin parámetros.
Salida: DE contiene el margen izquierdo, HL el derecho y AF se modifica.

GRA GET W HEIGHT

Dirección: #BBD8
Descripción: Averigua las coordenadas de altura de la ventana gráfica con respecto al origen absoluto 0,0 de la pantalla (esquina inferior izquierda).

Entrada: Sin parámetros.
Salida: DE contiene el margen superior, HL el inferior y AF se modifica.

GRA CLEAR WINDOW

Dirección: #BBDB
Descripción: Borra la ventana gráfica empleando el color de fon-

do, retornando el cursor gráfico al origen del usuario (no tiene porqué ser el absoluto 0,0 de la pantalla).

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL se modifican.

GRA SET PEN

Dirección: #BBDE

Descripción: Asigna el color para primer término en el modo gráfico, asegurándose la rutina que el valor corresponda con el modo de pantalla (mediante máscara).

Entrada: A contiene la tinta.

Salida: AF se modifica.

GRA GET PEN

Dirección: #BBE1

Descripción: Averigua la tinta de primer término empleada en el modo gráfico.

Entrada: Sin parámetros.

Salida: A contiene la tinta y las banderas se modifican.

GRA SET PAPER

Dirección: #BBE4

Descripción: Asigna el color para fondo en el modo gráfico, asegurándose la rutina que el valor corresponda con el modo de pantalla (mediante máscara).

Entrada: A contiene la tinta.

Salida: AF se modifica.

GRA GET PAPER

Dirección: #BBE7

Descripción: Averigua la tinta de fondo empleada en el modo gráfico.

Entrada: Sin parámetros.
Salida: A contiene la tinta y las banderas se modifican.

GRA PLOT ABSOLUTE

Dirección. #BBEA
Descripción: El cursor gráfico se desplaza al punto requerido y a éste se le asigna la tinta de primer término, siempre y cuando quede dentro de la pantalla; de no ser así, la rutina regresa. En todo caso, las coordenadas se deberán proporcionar como relativas al origen fijado por el usuario (no tiene porqué ser el absoluto 0,0 de la pantalla, aunque éste sea el valor por defecto).
Entrada: DE contiene la coordenada X y HL la Y.
Salida: AF, BC, DE y HL se modifican.

GRA PLOT RELATIVE

Dirección: #BBED
Descripción: Su efecto es en todo similar a la rutina anterior, salvo que las coordenadas se consideran como un desplazamiento relativo respecto al cursor gráfico.
Entrada: DE contiene el desplazamiento en el eje X y HL en el eje Y.
Salida: AF, BC, DE y HL se modifican.

GRA TEST ABSOLUTE

Dirección: #BBF0
Descripción: Averigua la tinta asignada al punto cuyas coordenadas se facilitan, al cual se desplaza el cursor gráfico, relativas con respecto al origen fijado por el usuario. Si el punto quedara fuera de la ventana, se devolvería el color de fondo del modo gráfico.
Entrada: DE contiene la coordenada X y HL la Y.
Salida: A contiene el valor de tinta. BC, DE, HL y las banderas se modifican.

GRA TEST RELATIVE

Dirección: #BBF3

Descripción: Su efecto es en todo similar a la rutina anterior, salvo que las coordenadas se consideran como un desplazamiento relativo respecto al cursor gráfico.

Entrada: DE contiene el desplazamiento en el eje X y HL en el eje Y.

Salida: A contiene la tinta. BC, DE, HL y las banderas se modifican.

GRA LINE ABSOLUTE

Dirección: #BBF6

Descripción: Traza una línea en la tinta de primer término desde la posición actual del cursor, hasta la coordenada absoluta proporcionada, relativa al origen fijado por el usuario, quedando ubicado el cursor gráfico al final de la misma. Todo punto del trazado que quede fuera de la ventana gráfica no será tenido en cuenta.

Entrada: DE contiene la coordenada X del punto final de la línea y HL la Y.

Salida: AF, BC, DE y HL se modifican.

GRA LINE RELATIVE

Dirección: #BBF9

Descripción: Su efecto es en todo similar a la rutina anterior, salvo que las coordenadas se consideran como un desplazamiento relativo respecto al cursor gráfico.

Entrada: DE contiene el desplazamiento en el eje X y HL en el eje Y.

Salida: AF, BC, DE y HL se modifican.

GRA WR CHAR

Dirección: #BBFC

Descripción: Escribe un carácter en la posición del cursor gráfico, ajustándose a éste la esquina superior izquierda de la matriz del ca-

rácter. Como si se tratase de puntos cualquiera, los pixels del carácter que excedan los límites de la ventana no serán tenidos en consideración y además se utilizará la tinta de primer término y fondo de la VDU gráfica para el carácter, aunque en la VDU de texto esté fijado el modo de escritura transparente. Asimismo, el cursor gráfico se desplazará hacia la izquierda la anchura del carácter, teniendo en cuenta para ello las dimensiones del mismo, según el modo de pantalla. También se representarán los caracteres de control.

Entrada: A contiene el carácter a escribir.

Salida: AF, BC, DE y HL se modifican.

EL CONTROLADOR DE LA PANTALLA

ste bloque de rutinas del firmware, cuyas direcciones de saltos abarcan las posiciones #BBFF a #BC62, tiene como misión el control del hardware de la pantalla (CRTC). Así pues, es a estas rutinas a las que acude el Visualizador de texto (TEXT VDU) y el gráfico (GRAPHIC VDU) para poder llevar a cabo la mayoría de sus funciones.

Encontraremos, por tanto, que algunas de las rutinas del controlador de pantalla (SCREEN PACK) pueden sustituir a las de los visualizadores de text y gráficos; no obstante, es más conveniente acudir a estas últimas, dado que se encargarán además de actualizar determinadas condiciones, como pueden ser Variables del Sistema que contiene información sobre la posición del cursor, la ubicación de la memoria de pantalla, etc...

SCR INITIALISE

Dirección: #BBFF

Descripción: Realiza una reinicialización completa del controlador de la pantalla. Todas sus variables e indirecciones toman el valor de

su rutina por defecto (SCR READ, SCR WRITE y SCR MODE CLEAR), las tintas toman su valor por defecto, así como los períodos de intermitencia de colores y el modo de pantalla (MODE 1). La dirección base de la memoria de pantalla se establece en #C000 (hasta #FFFF, bajo la ROM superior), el desplazamiento de la pantalla se establece en 0, la pantalla se borra con tinta 0 y se fija el modo opaco de escritura para la VDU gráfica.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL modificados.

SCR RESET

Dirección: #BC02

Descripción: Reinicializa las indirecciones (SCR READ, SCR WRITE y SCR MODE CLEAR) y las tintas toman su valor por defecto, así como los períodos de intermitencia de colores, fijándose a opaco el modo de escritura para la VDU gráfica.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL se modifican.

SCR SET OFFSET

Dirección: #BC05

Descripción: Fija el desplazamiento de la base de la pantalla, lo cual puede permitir un efecto de scroll. El desplazamiento se enmascara con #07FE, asegurándose que sea un valor par. El desplazamiento y la base se envían conjuntamente al hardware (CRTC). Si bien este desplazamiento puede fijarse directamente a través de las rutinas de bajo nivel, es más recomendable emplear esta rutina, dado que en caso contrario la VDU de gráficos y texto pueden actuar de manera errónea, al no haber sido advertidas del desplazamiento.

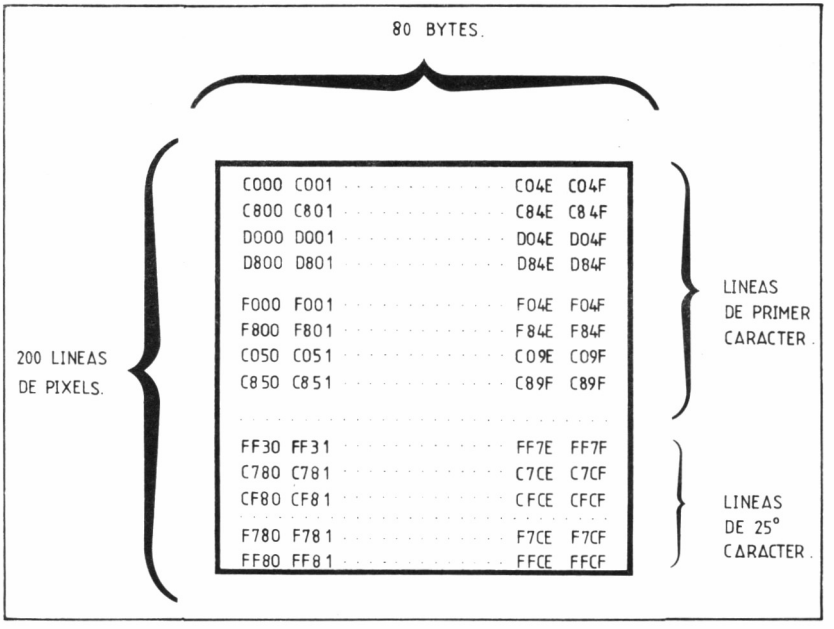
Entrada: HL contiene el desplazamiento.

Salida: AF y HL se modifican.

SCR SET BASE

Dirección: #BC08

Descripción: Establece la dirección base de la zona de pantalla, en bloques de 16 Kb.



Entrada: A contiene el byte más significativo de la dirección base a establecer.

Salida: AF y HL se modifican.

SCR GET LOCATION

Dirección: #BC0B

Descripción: Averigua las direcciones de base y desplazamiento de la pantalla.

Entrada: Sin parámetros.

Salida: A contiene el byte más significativo de la dirección base y HL el desplazamiento. Las banderas se modifican.

SCR SET MODE

Dirección: #BC0E

Descripción: Establece el modo de pantalla, utilizando #03 como

máscara. Las tintas se ajustan al modo escogido, la pantalla se borra, todas las ventanas toman sus valores por defecto (toda la pantalla) y el origen de coordenadas del usuario vuelve al de la pantalla 0,0 (esquina inferior izquierda).

Entrada: A contiene el modo.

Salida: AF, BC, DE y HL se modifican.

SCR GET MODE

Dirección: #BC11

Descripción: Averigua el modo de pantalla.

Entrada: Sin parámetros.

Salida: Modo 0: acarreo a 1, cero a 0 y A=0. Modo 1: acarreo a 0, cero a 1 y A=1. Modo 2: acarreo a 0, cero a 0 y A=2. En todo caso, se modifica el resto de banderas.

SCR CLEAR

Dirección: #BC14

Descripción: Borra la pantalla, utilizando la tinta 0, fijándose a 0 el desplazamiento de la pantalla.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL se modifican.

SCR CHAR LIMITS

Dirección: #BC17

Descripción: Averigua el último número de fila y columna de la pantalla en el modo vigente, tomando como origen la esquina superior izquierda con coordenadas 0,0.

Entrada: Sin parámetros.

Salida: B Contiene la última columna y C la última fila. AF se modifica.

SCR CHAR POSITION

Dirección: #BC1A

Descripción: Averigua la posición física de pantalla en la que co-

rresponde comenzar a escribir un carácter (pixel superior izquierdo de la matriz), así como la anchura en bytes que corresponde al carácter según el modo de pantalla. Las coordenadas del carácter se deben proporcionar siguiendo la misma norma que en la rutina anterior.

Entrada: H contiene la columna y L la fila.

Salida: HL contiene la dirección inicial de la pantalla que corresponde a las coordenadas de carácter proporcionadas y B la anchura en bytes que el carácter ocupará en la pantalla. AF se modifica.

SCR DOT POSITION

Dirección: #BC1D

Descripción: Obtiene la dirección de pantalla de un pixel cuyas coordenadas se proporcionan, teniendo como base el origen absoluto 0,0 (esquina inferior izquierda de la pantalla), así como la máscara para el pixel y el número de pixels correspondientes a un byte de pantalla según el modo vigente.

Entrada: DE contiene la coordenada X del pixel y HL la Y.

Salida: HL contiene la dirección física de pantalla correspondiente al pixel, C su máscara y B el número de pixels que integran un byte de pantalla menos 1. AF y DE se modifican.

SCR NEXT BYTE

Dirección: #BC20

Descripción: Dada una dirección de pantalla, calcula el desplazamiento de la pantalla a un byte más a la derecha.

Entrada: HL contiene la dirección inicial.

Salida: HL contiene ahora la dirección actualizada y AF se modifica.

SCR PREV BYTE

Dirección: #BC23

Descripción: Dada una dirección de pantalla, calcula el desplazamiento de la pantalla a un byte más a la izquierda.

Entrada: HL contiene la dirección inicial.

Salida: HL contiene ahora la dirección actualizada y AF se modifica.

SCR NEXT LINE

Dirección: #BC26
Descripción: Dada una dirección de pantalla, calcula el desplazamiento de la pantalla a un byte más abajo.
Entrada: HL contiene la dirección inicial.
Salida: HL contiene ahora la dirección actualizada y AF se modifica.

SCR NEXT LINE

Dirección: #BC29
Descripción: Dada una dirección de pantalla, calcula el desplazamiento de la pantalla a un byte más arriba.
Entrada: HL contiene la dirección inicial.
Salida: HL contiene ahora la dirección actualizada y AF se modifica.

SCR INK ENCODE

Dirección: #BC2C
Descripción: Codifica una tinta de forma que asigne el color a los pixels adecuados de un byte de la pantalla. La codificación es simple en el modo 2, dado que los pixels se corresponden con los bytes de la tinta codificada, estando a 1 los de color de primer término, y a 0 los de fondo. No obstante, en los restantes modos van intercalados. Así, en el modo 1, los cuatro pixels que integran un byte, se traduce de izquierda a derecha en los bits 3-7, 2-6, 1-5 y 0-4, respectivamente; en el modo 0, los dos pixels que integran cada byte corresponden de izquierda a derecha a los bits 1-5-3-7 y 0-4-2-6, respectivamente.
Entrada: A contiene la tinta.
Salida: A contiene la tinta codificada y las banderas se modifican.

SCR INK DECODE

Dirección: #BC2F
Descripción: Decodifica una tinta codificada.

Entrada: A contiene la tinta codificada.
Salida: A contiene la tinta decodificada y las banderas se modifican.

SCR SET INK

Dirección: #BC32
Descripción: Asigna los dos colores para una tinta. La situación de intermitencia se produce cuando el segundo color es distinto del primero.
Entrada: A contiene el código de tinta, B el primer color y C el segundo.
Salida: AF, BC, DE y HL se modifican.

SCR GET INK

Dirección: #BC35
Descripción: Averigua los colores correspondientes a una tinta.
Entrada: A contiene la tinta.
Salida: B contiene el primer color y C el segundo, mientras que AF, DE y HL se modifican.

SCR SET BORDER

Dirección: #BC38
Descripción: Asigna los colores del marco de la pantalla.
Entrada: B contiene el primer color y C el segundo (si son distintos se produce intermitencia).
Salida: AF, BC, DE y HL se modifican.

SCR GET BORDER

Dirección: #BC3B
Descripción: Averigua los colores asignados al marco de la pantalla.
Entrada: Sin parámetros.
Salida: B contiene el primer color y C el segundo. AF, DE y HL se modifican.

SCR SET FLASHING

Dirección: #BC3E

Descripción: Establece los períodos de permanencia de cada uno de los colores en intermitencia en tintas y marco. Dichos períodos se miden en cincuentavos de segundo y se pueden establecer entre 1 y 255 (tomándose 0 como 256). Por defecto se fijan ambos a 10.

Entrada: H es la duración del primer color y L la del segundo.

Salida: AF y HL se modifican.

SCR GET FLASHING

Dirección: #BC41

Descripción: Averigua los períodos actuales de permanencia en intermitencias de colores.

Entrada: Sin parámetros.

Salida: H contiene el período del primer color, L el del segundo y AF se modifica.

SCR FILL BOX

Dirección: #BC44

Descripción: Rellena un área de caracteres de la pantalla con una tinta determinada. La zona afectada se delimita en coordenadas absolutas con respecto al origen absoluto de texto 0,0 (esquina superior izquierda).

Entrada: A es la tinta codificada, H y D las columnas primera y última, respectivamente, que marcan la zona en horizontal y L y E las filas que la marcan en vertical (L superior y E inferior).

Salida: AF, BC, DE y HL se modifican.

SCR FLOOD BOX

Dirección: #BC47

Descripción: Rellena un área de pantalla expresada en bytes, indicando el byte superior izquierdo de la zona a cubrir y su anchura en bytes y altura en líneas, así como el color escogido (por valores 0 se toma 256).

Entrada: C es la tinta codificada, HL la dirección de la esquina superior izquierda, D los bytes de anchura y E las líneas de altura.

Salida: AF, BC, DE y HL se modifican.

SCR CHAR INVERT

Dirección: #BC4A

Descripción: Invierte las tintas de un carácter, siguiendo el mismo sistema explicado al estudiar la presentación del cursor de texto.

Entrada: B y C contienen dos tintas codificadas y H y L, respectivamente, la columna y fila del carácter, utilizando coordenadas absolutas con origen 0,0 en la esquina superior izquierda de la pantalla.

Salida: AF, BC, DE y HL se modifican.

SCR HW SCROLL

Dirección: #BC4D

Descripción: Efectúa un scroll hardware de la pantalla una fila hacia arriba o abajo (8 líneas de pixels).

Entrada: B indica el sentido del desplazamiento; B=0 abajo y B<>0 arriba. A contiene la tinta codificada que se debe emplear para borrar la línea que aparece arriba o abajo tras el desplazamiento.

Salida: AF, BC, DE y HL se modifican.

SCR SW ROLL

Dirección: #BC50

Descripción: Efectúa un scroll de una zona de pantalla una fila hacia arriba o abajo (8 líneas de pixels). Las coordenadas que delimitan la zona a tratar son relativas al origen absoluto de texto 0,0 situado en la esquina superior izquierda de la pantalla.

Entrada: A contiene la tinta codificada para borrar la nueva línea, H la columna izquierda y D la columna derecha que delimitan la zona horizontalmente y L y E las filas superior e inferior, respectivamente, que la delimitan verticalmente. Como en la rutina anterior, B indica el sentido de desplazamiento; B=0 abajo y B<>0 arriba.

Salida: AF, BC, DE y HL se modifican.

SCR UNPACK

Dirección: #BC53

Descripción: Convierte una matriz de carácter a la máscara de pixels que se empleará para su presentación en la pantalla, atendiendo al modo vigente. Así pues, el área para efectuar la conversión dependerá en su longitud del modo de la pantalla: 1 byte (MODE 2), 2 bytes (MODE 1) o 4 bytes (MODE 0). Si un bit de la matriz está a 1 se prepara la máscara apropiada en el área de conversión, de no ser así no se incluye (los bits se fijan a 0).

Entrada: HL contiene la dirección de la matriz y DE la del área de conversión.

Salida: AF, BC, DE y HL se modifican.

SCR REPACK

Dirección: #BC56

Descripción: Obtiene la matriz correspondiente a un carácter de la pantalla, por comparación con la tinta empleada. Las coordenadas del carácter se suministran, como es habitual, con referencia al origen absoluto de texto 0,0 (esquina superior izquierda de la pantalla).

Entrada: A contiene la tinta codificada a comparar, H y L las coordenadas de columna y fila física del carácter a leer, respectivamente, y DE el área de construcción de la matriz.

Salida: AF, BC, DE y HL se modifican.

SCR ACCESS

Dirección: #BC59

Descripción: Establece el modo de escritura para la VDU gráfica. Los modos son los siguientes: 0 (OPACO) en el que la tinta corresponde directamente con la tinta empleada; 1 (XOR) en que se opera un or exclusivo de la tinta a emplear con la actual del pixel a escribir; 2 (AND) en que la operación efectuada entre las tintas nueva y antigua es un AND y modo 3 (OR) para operaciones de or entre tintas.

Entrada: A contiene el modo.

Salida: AF, BC, DE y HL se modifican.

SCR PIXELS

Dirección: #BC5C

Descripción: Escribe un pixel o pixels en la pantalla, ignorando el modo de escritura de la VDU gráfica, teniendo en cuenta una tinta codificada y una máscara para el pixel o pixels afectados.

Entrada: B contiene la tinta codificada, C la máscara del pixel o pixels y HL la dirección en la que se aplica la máscara.

Salida: AF se modifica.

SCR HORIZONTAL

Dirección: #BC5F

Descripción: Traza una línea horizontal en la pantalla del color indicado y con precisión de pixel. Las coordenadas deberán suministrarse relativas al origen absoluto 0,0 (esquina inferior izquierda). Se tiene en cuenta el modo de escritura de la VDU gráfica.

Entrada: A contiene la tinta codificada, DE la coordenada X de comienzo de la línea, BC la de fin y HL la coordenada Y.

Salida: AF, BC, DE y HL modificados.

SCR VERTICAL

Dirección: #BC62

Descripción: Cumple una función en todo similar a la rutina anterior, aunque traza líneas verticales.

Entrada: A contiene la tinta codificada, HL la coordenada Y de comienzo de la línea, BC la de fin y DE la coordenada X.

Salida: AF, BC, DE y HL modificados.

EL CONTROLADOR DEL CASETE

Cuando apareció en el mercado el CPC 464, una de sus características más notables consistía en llevar incorporado en el mismo chasis, junto al teclado, una unidad de casete. Aquel añadido no suponía ninguna novedad espectacular. De hecho, algunas marcas ya habían adoptado dicho sistema; en especial, en sus ordenadores portátiles.

No obstante, un sistema compacto ordenador-periférico, reduce notablemente el número de conexiones y cables sobre la mesa, eliminando los consiguientes problemas que de esto se derivan.

Pero si bien al principio no se hacía palpable, con el paso del tiempo, los pioneros de aquel primer Amstrad coincidían en un punto muy particular: la excelente fiabilidad en todas las operaciones de lectura/escritura entre casete y ordenador.

Alguien puede objetar que también de cuando en cuando falla. De acuerdo, pero hemos de pensar que se trata de un sistema mecánico, y como tal, tiende a desajustarse con el paso del tiempo. Aún así, en la inmensa mayoría de las ocasiones, el error no es imputable al casete, sino al deterioro progresivo y a la pérdida de propiedades de la cinta magnética.

Son precisamente las rutinas desarrolladas en el firmware del Sistema las que proporcionan una fiabilidad tan aceptable, pues todos los impulsos generados o la interpretación de éstos, se efectúa al cien por cien vía software.

El Gestor De Casete

Si ojeamos el manual de BASIC de nuestro Amstrad y nos centramos en las líneas que encontramos acerca del comando SPEED WRITE, quizás para algunos sean suficientes, pero los que buscamos profundizar en las interioridades de nuestro micro quedamos un tanto perplejos ante tan sucinta explicación.

«Establece la velocidad de transmisión de datos al magnetófono», hasta aquí nos sentimos orgullosos de la posibilidad de doblar la velocidad inicial por defecto, situándola en 2000 baudios (bits por segundo), cifra en absoluto desdeñable.

Sin embargo, si continuamos, se nos indica que para leer la información contenida en una cinta, no es necesario indicarle al ordenador a qué velocidad fue grabada: ¡la determina automáticamente!

Lo cierto es que uno de los mayores problemas entre los pequeños casetes es que a pesar de tener normalizada la velocidad de avance de la cinta a 4,75 cm/s., no es exactamente la misma de un modelo a otro y, por tanto, un programa que no ocasione ningún problema en uno de estos equipos, puede ser misión más que imposible cargarlo en la memoria de nuestro ordenador desde una grabadora diferente (aunque esté en perfectas condiciones).

Todo ello fue cuidadosamente estudiado al diseñar el firmware del Amstrad, y en él encontramos la respuesta a las afirmaciones vertidas en el manual de BASIC.

Un bit se escribe en la cinta como un período de bajo nivel seguido de otro idéntico a nivel alto. Para establecer la diferencia entre 1s y 0s lógicos, la señal correspondiente al 1 se mantiene activa doble tiempo que la del 0. Es decir, los 1s tienen frecuencia mitad que los 0s.

La velocidad de escritura de la información sobre el casete, la fija la subrutina CAS SET SPEED, cuya misión consiste en definir el período para el 0 lógico, y a partir de éste, como decíamos en el párrafo anterior, el del 1. Además, se establece un parámetro más de extraordinaria importancia: la precompensación.

Este factor es precisamente el que evita problemas de compatibili-

dad entre cintas grabadas con distintos aparatos. En concreto, alarga el período de los 1s y acorta los 0s. De esta manera, se consigue disminuir al máximo los errores procedentes de las variaciones de velocidad entre grabación y reproducción.

Para establecer la velocidad seleccionada, en baudios, se sigue la fórmula:

Número de baudios = 1.000.000/3 × longitud de medio 0 = 333.333/longitud de medio 0.

La longitud de medio 0 se expresa en microsegundos, y debe estar comprendida entre 130 y 480 para garantizar una cierta fiabilidad. De lo contrario, los errores de lectura/escritura estarán al orden del día.

Por ejemplo, en el caso de la velocidad inicial por defecto (1.000 baudios), 333 microsegundos es la mitad del período del 0 asignada por el ordenador para conseguir este valor. No obstante, la tasa de baudios así obtenida es tan sólo aproximada, es decir, considerando que el bloque de datos está formado por idéntico número de 1s que de 0s (recordemos que el período del 1 es doble que el del 0).

La precompensación se expresa igualmente en microsegundos. Ha de variar entre 0 y 255, aunque como antes, los valores altos provocarán errores en la transmisión. A 1.000 baudios, el firmware asume un valor de 25, mientras que a 2.000 son 50 microsegundos los sumados o restados a los períodos del 1 ó 0, respectivamente, buscando que el aspecto de la onda final sea el ideal al ser leída del casete.

Trenes De Impulsos

En los diagramas hemos representado esquemáticamente el formato general de la información contenida en un fichero cualquiera almacenado en cinta magnética. Analicemos paso a paso cada uno de los campos.

Cada fichero queda dividido en bloques, cada uno de los cuales consta de tres campos característicos: el primero es una señal de puesta en funcionamiento para el motor del casete, además de separar unos bloques de otros. El segundo es el registro cabecera del fichero, el cual contiene información de vital importancia para el tratamiento posterior de datos. Finalmente, encontramos el registro de datos.

Además de los tres anteriores, cuando se trata del primer y último bloque de un fichero, aparecen otros dos campos destinados a marcar aún más la separación entre los ficheros.

El registro de datos se subdivide a su vez en una serie de segmen-

tos con capacidad cada uno para almacenar 256 bytes, comenzando por un tono guía *(leader)*, cuyo objetivo consiste por una parte en elevar al óptimo el nivel de grabación en los casetes que lo incorporen automático y, por otra, en detectar la velocidad de grabación de la información cuando ésta es de nuevo reproducida. Tras el último segmento encontramos un bloque final o cola *(trailer)*, formado por 32 bits 1.

El número de segmentos en cada bloque varía de 1 a 8 (normalmente 8), rellenándose los últimos de ceros durante la escritura si no existen suficientes datos para cubrirlos.

El *leader* presente al comienzo de todos los registros, se compone de 2048 bits 1, seguidos de un bit 0 que marca el final del tono guía, más un byte de sincronismo destinado a conocer la naturaleza de los datos. Es decir, señala la diferencia entre datos de la cabecera y datos del fichero propiamente (&2C y &16, respectivamente).

El Registro De Cabecera

Todos los bloques del fichero contienen un registro de cabecera que guarda la información referente a los datos allí almacenados. Está formado por 64 bytes que el sistema operativo utiliza para varios propósitos. Este registro consta de un único segmento.

Los primeros 16 bytes (0 a 15) conservan el nombre del fichero completado con carácteres nulos. El 16 contiene el número de bloque, y si en el 17 aparece un valor distinto de cero, significa que los datos a continuación, pertenecen al último bloque del fichero.

El 18, en función de sus bits individuales, proporciona la información sobre «tipo o naturaleza» de fichero. En concreto: si el bit 0 está alzado, significa fichero protegido. Si en los bits 1 a 3 aparece un 0, quiere decir fichero BASIC, un 1 significa binario, un 2 imagen de pantalla, mientras que un 3 se interpreta como ASCII. Además, en el caso de estos últimos, en los bits del 4 al 7 ha de aparecer un 1. Para los demás, estos cuatro bits permanecerán a 0.

Siguiendo hacia adelante por la cabecera, llegamos a los bytes 19 y 20, los cuales almacenan la longitud del bloque en octetos, y las posiciones 21 y 22 contienen la dirección de comienzo donde estos fueron grabados. Si el contenido del 23 es diferente de cero, significa que se trata del primer bloque de un fichero.

Aquí finalizan los denominados «Campos del Sistema» y a continuación se encuentran los «Campos del Usuario», pues son utilizables por éste según lo considere o no necesario.

En las posiciones 24 y 25 está almacenada la longitud total del fichero, y en la 25 y 26, la dirección de ejecución de los programas en código máquina. Finalmente, los bytes comprendidos del 28 al 63 no están utilizados y se pueden emplear para cualquier propósito que se crea conveniente.

CAS INITIALISE

Dirección: #BC65
Descripción: Realiza una reinicialización completa del controlador del casete. Todas las corrientes se cierran, la velocidad de escritura toma el valor por defecto y los mensajes de petición de acción se conectan.
Entrada: Sin parámetros.
Salida: AF, BC, DE y HL modificados.

CAS SET SPEED

Dirección: #BC68
Descripción: Fija la velocidad de escritura en el casete, estableciendo la longitud de los bits y la cantidad de precompensación de escritura. La velocidad se obtiene como la longitud en microsegundos de la mitad de un bit 0; siendo un bit 1 dos veces más largo que un 0. La velocidad de transferencia media, medida en baudios, se puede calcular, supuesto un número igual de bits 0 y 1, según la ecuación: 1000000/(3*longitud de medio 0). Los valores aplicables para la longitud de medio 0, antes de caer en errores de lectura o escritura, se hallan comprendidos en el margen 130-480 microsegundos.

Por precompensación se entiende la longitud (en microsegundos) que se debe añadir a la mitad de un bit a 1 ó restar a la mitad de un bit a 0, bajo ciertas condiciones. A mayor velocidad es precisa una mayor precompensación. Dicho valor puede oscilar entre 0 y 255, aunque las precompensaciones más altas abocan a errores de lectura o escritura.

Para 1.000 baudios se emplea una longitud de medio 0 de 333 microsegundos y una precompensación de 25 microsegundos, mientras que para 2.000 baudios, sus valores son 167 y 50 microsegundos, respectivamente.

Entrada: HL contiene la longitud en microsegundos de medio bit 0 y A la longitud en microsegundos de precompensación.

Salida: AF y HL se modifican.

CAS NOISY

Dirección: #BC6B

Descripción: Deshabilita o habilita la presentación en pantalla de los mensajes de petición de acción al usuario (Press PLAY then any key, etc...), aunque no los de error (Read error..., Write error a y Rewind tape).

Entrada: A contiene 0 para la habilitación de mensajes y es distinto de cero para la deshabilitación.

Salida: AF se modifica.

CAS START MOTOR

Dirección: #BC6E

Descripción: Activa el motor del casete, esperando un intervalo de unos dos segundos, para que ésta alcance la velocidad óptima en caso de que estuviera apagado.

Entrada: Sin parámetros.

Salida: A contiene el estado previo del motor, la bandera de acarreo vuelve a 1 si la operación fue llevada a cabo sin problemas, y a 0 si en el período de espera para la obtención de la velocidad óptima el usuario pulsó ESCape. El resto de banderas se modifican.

CAS STOP MOTOR

Dirección: #BC71

Descripción: Detiene el motor del casete, indicando su estado previo, retornando sin aguardar a que el motor quede totalmente parado.

Entrada: Sin parámetros.

Salida: A contiene el estado previo del motor y la bandera de acarreo vuelve a 1 si la acción llevó a cabo sin problemas o a 0 si el usuario pulsó ESCape. El resto de banderas se modifican.

CAS RESTORE MOTOR

Dirección: #BC74

Descripción: Restablece el estado previo del motor del casete, actuando para su puesta en marcha o detención tal como se indica en las dos rutinas precedentes.

Entrada: A contiene el estado previo del motor.

Salida: Si la acción correspondiente se llevó a cabo, ya sea parada o puesta en marcha del motor, la bandera de acarreo vuelve a 1; en caso contrario (se pulsó ESCape) retorna a 0. En todo caso, A y el resto de las banderas se modifican.

CAS IN OPEN

Dirección: #BC77

Descripción: Abre un fichero para lectura, preparando la corriente adecuada y leyendo su primer bloque. Se abre un buffer de 2 Kb para almacenar el contenido de los bloques, que permanecerá abierto hasta que se cierre el fichero. El nombre especificado del fichero es truncado a 16 caracteres o completado a esta cantidad con caracteres 0; por otra parte, sus caracteres se pasan a mayúsculas, entendiéndose que un fichero sin nombre o que comience por un carácter 0, implica la búsqueda del primer fichero encontrado en la cinta.

Entrada: B contiene la longitud del nombre de fichero, DE la dirección del buffer y HL la del nombre.

Salida: Se modifican IX y las banderas. Si el fichero fue abierto sin problemas, la bandera de acarreo vuelve a 1, la de cero a 0, A contiene el tipo de fichero (información extraída de la cabecera), BC la longitud del fichero (dato extraído también de la cabecera), DE la dirección de carga para los datos (igualmente de la cabecera) y HL la dirección del buffer que contiene la información de la cabecera. Si la corriente se está utilizando, la bandera de acarreo vuelve a 0, al igual que la de cero, y se modifican A, BC, DE y HL. Por último, si el usuario pulsa ESCape, la bandera de acarreo retorna a 0, la de cero a 1 y A, BC, DE y HL se modifican.

CAS IN CLOSE

Dirección: #BC7A

Descripción: Cierra el fichero de entrada, marcándolo como cerrado.

Entrada: Sin parámetros.

Salida: Si la corriente se cerró sin problemas la bandera de acarreo vuelve a 1; en caso contrario (si no estaba abierta) a 0. En todo caso, A, BC, DE y HL se modifican, así como el resto de las banderas.

CAS IN ABANDON

Dirección: #BC7D

Descripción: Abandona de inmediato la lectura de la corriente de entrada y la cierra.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL se modifican.

CAS IN CHAR

Dirección: #BC80

Descripción: Lee un carácter del fichero de entrada. Una vez que se ha utilizado esta rutina, no es posible continuar la lectura mediante CAS IN DIRECT.

Entrada: Sin parámetros.

Salida: Si el fichero se leyó correctamente, la bandera de acarreo vuelve a 1 y la de cero a 0, mientras que A contiene el carácter leído del fichero. Si se alcanzó el fin de fichero o éste no está abierto, la bandera de acarreo y la de cero quedan a cero y A se modifica. Por otra parte, si el usuario pulsó ESCape, la bandera de acarreo vuelve a 0 y la de cero a 1, con A modificado. En todo caso IX y el resto de las banderas se modifican.

CAS IN DIRECT

Dirección: #BC83

Descripción: Lee y almacena en RAM directamente un fichero de entrada, en vez de hacerlo carácter a carácter, motivo por el cual no es posible utilizar esta rutina si la corriente fue utilizada para el acceso por carácter (CAS IN CHAR). Tampoco es posible leer directamente del fichero dos veces, dado que esto simplemente estropea la copia del fichero leída.

Entrada: HL contiene la dirección en la que se ubicará el fichero.

Salida: Si el fichero se leyó correctamente, la bandera de acarreo vuelve a 1 y la de cero a 0, mientras que HL conserva la dirección de entrada, desde la cabecera. Si el fichero no se abrió, la bandera de acarreo y la de cero retornan a 0 y HL se modifica. Por otra parte, si el usuario pulsó ESCape, la bandera de acarreo vuelve a 0 y la de cero a 1, con HL modificado. En todo caso, A, BC, DE e IX, así como el resto de las banderas, se modifican.

CAS RETURN

Dirección: #BC86

Descripción: Retorna al buffer el último carácter por CAS IN CHAR, de manera que éste será el que se lea la próxima vez que se ejecute dicha rutina.

Entrada: Sin parámetros.

Salida: No se modifica ninguna bandera ni registro.

CAS TEST EOF

Dirección: #BC89

Descripción: Averigua si se ha alcanzado el final de fichero (EOF, *End Of File).* Tras acceder a esta rutina no es posible acceder a CAS RETURN, sin haber leído un carácter previamente, ni al uso de la lectura directa, dado que fija la corriente en el modo de lectura de carácter.

Entrada: Sin parámetros.

Salida: Si no hay EOF (fin de fichero) la bandera de acarreo vuelve a 1 y la de cero a 0. Si se detecta EOF, el acarreo y la bandera de cero quedan a 0. En caso de pulsar ESCape, el acarreo retorna a 0, pero el cero a 1. En todo caso, A e IX, junto con el resto de las banderas, se modifican.

CAS OUT OPEN

Dirección: #BC8C

Descripción: Abre un fichero para escritura, preparando la corriente adecuada. Se abre un buffer de 2 Kb para almacenar el contenido de los bloques, que permanecerá abierto hasta que se cierre el fi-

chero. El nombre especificado del fichero es truncado a 16 caracteres o completado a esta cantidad con caracteres 0; por otra parte, sus caracteres se pasan a mayúsculas.

Entrada: B contiene la longitud del nombre de fichero, DE la dirección del buffer y HL la del nombre.

Salida: Se modifican A, BC, DE e IX. Si el fichero fue abierto sin problemas la bandera de acarreo vuelve a 1, la de cero a 0 y HL contiene la dirección del buffer en que se encuentra la cabecera que será escrita al comienzo de cada bloque. Si la corriente ya se está utilizando, la bandera de acarreo vuelve a 0, al igual que la de cero, y se modifica HL. Por último, si el usuario pulsa ESCape, la bandera de acarreo retorna a 0 y la de cero a 1.

CAS OUT CLOSE

Dirección: #BC8F

Descripción: Cierra el fichero de salida, marcándolo como cerrado y vuelca en la cinta el último bloque de información que quedara por enviar.

Entrada: Sin parámetros.

Salida: Si la corriente se cerró sin problemas la bandera de acarreo vuelve a 1 y la de cero a 0; en caso contrario (si no estaba abierta) a 0 ambas. Si el usuario pulsa ESCape el fichero queda abierto, la bandera de acarreo vuelve a 0 y la de cero a 1. En todo caso, A, BC, DE, HL e IX se modifican, así como el resto de las banderas.

CAS OUT ABANDON

Dirección: #BC92

Descripción: Abandona de inmediato la escritura y marca la corriente como cerrada, sin enviar la información pendiente en el buffer.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL se modifican.

CAS OUT CHAR

Dirección: #BC95

Descripción: Añade un carácter al buffer para escritura, y si éste

está completo, lo vuelca en la cinta previamente. Una vez que se ha utilizado esta rutina, no es posible continuar la escritura en modo directo.

Entrada: A contiene el carácter a escribir.

Salida: Si el carácter se escribió correctamente, la bandera de acarreo vuelve a 1 y la de cero a 0. Si el fichero no está abierto, la bandera de acarreo y la de cero quedan a cero. Por otra parte, si el usuario pulsa ESCape, la bandera de acarreo vuelve a 0 y la de cero a 1. En todo caso A e IX y el resto de las banderas se modifican.

CAS OUT DIRECT

Dirección: #BC98

Descripción: Escribe directamente en el fichero de salida la información contenida en memoria, debiéndose emplear a continuación CAS OUT CLOSE para cerciorarse que el último bloque presente en el buffer sea vaciado. No es posible intercambiar este método de escritura con el de CAS OUT CHAR, una vez que cualquiera de las dos rutinas ha sido seleccionada.

Entrada: HL contiene la dirección en la que se ubican los datos a escribir, A contiene el tipo de fichero (para la cabecera), BC la dirección de entrada (para la cabecera) y DE la longitud de la información a escribir.

Salida: Si el fichero se escribió correctamente, la bandera de acarreo vuelve a 1 y la de cero a 0. Si el fichero no se abrió, la bandera de acarreo y la de cero retornan a 0. Por otra parte, si el usuario pulsó ESCape, la bandera de acarreo vuelve a 0 y la de cero a 1. En todo caso, A, BC, DE, HL e IX, así como el resto de las banderas, se modifican.

CAS CATALOG

Dirección: #BC9B

Descripción: Efectúa un catálogo de la cinta, imprimiendo la información en pantalla, al tiempo que se comprueba el estado de los ficheros. Esta rutina hace uso de la corriente de lectura, motivo por el cual debe encontrarse cerrada. La presentación de mensajes se activa (CAS NOISY). La lectura y presentación de la información de las cabeceras continúa hasta que se presiona la tecla ESCape. El distintivo

del tipo de fichero se obtiene sumando 36 (carácter $) al tipo de fichero y utilizando #0F como máscara.

Entrada: DE contiene la dirección del buffer de 2 Kb.

Salida: Si el catálogo se efectúa correctamente, la bandera de acarreo vuelve a 1 y la de cero a 0. Si la corriente ya estaba utilizándose, la bandera de acarreo y la de cero regresan a 0. Si sucede un error, la de acarreo retorna a 0 y la de cero a 1. En todo caso, A, BC, DE, HL, IX y el resto de las banderas se modifican.

CAS WRITE

Dirección: #BC9E

Descripción: Escribe un registro en el casete. Es la rutina empleada por las de alto nivel para escribir la cabecera y los datos de un fichero. Los códigos de error retornados son: 0 interrupción por ESCape y 1 error de escritura. Las interrupciones se deshabilitan.

Entrada: HL contiene la dirección de los datos a escribir, DE la longitud de los datos (por valor 0 se toma 65536) y A contiene el carácter identificativo para el final de un bloque de datos o una cabecera (#2C para cabecera y #16 para datos).

Salida: Si el registro se escribió correctamente la bandera de acarreo se pone a 1 y A se modifica. Si sucede un error o se pulsa ESCape, la bandera de acarreo queda a 0 y A regresa el código de error. En todo caso, BC, DE, HL e IX se modifican.

CAS READ

Dirección: #BCA1

Descripción: Lee una parte o un registro completo. Los errores retornados son: 0 (ESCape), 1 (se encontró un bit demasiado largo) y 2 (otros errores). Las interrupciones se deshabilitan.

Entrada: HL contiene la dirección para los datos leídos, DE la longitud del registro o parte del registro y A el byte identificador, tal como en la rutina anterior.

Salida: Si la información se ha leído correctamente, la bandera de acarreo se pone a 1 y A se modifica. De ocurrir un error, o pulsarse la tecla ESCape, el acarreo pasa a 0 y A retorna el código de error. En todo caso, BC, DE, HL, IX y el resto de las banderas se modifican.

CAS CHECK

Dirección: #BCA4

Descripción: Contrasta el contenido de una zona de memoria con datos de la cinta. Los posibles errores son los mismos tres de la rutina anterior, más un cuarto que denota la diferencia entre la memoria y el contenido de la cinta. Las interrupciones se deshabilitan.

Entrada: HL contiene la dirección de los datos a comprobar, DE la longitud de los mismos (0 se toma como 65536) y A el byte identificador de tipo, según la misma norma de las rutinas anteriores.

Salida: Si la comprobación finalizó con éxito, la bandera de acarreo vuelve a 1 y A se modifica. Si sucedió un error o se pulsó ESCape, el acarreo regresa a 0 y A contiene el código de error. En todo caso, BC, DE, HL, IX y el resto de banderas se modifican.

EL CONTROLADOR DE SONIDO

El controlador del sonido contiene las rutinas encargadas de tratar con el PSG AY-3-8912, de cuyas características hemos hablado anteriormente. No obstante, la mayoría del control sobre la emisión de sonido se ejerce más vía software, que aprovechando el propio hardware del PSG.

SOUND RESET

Dirección: #BCA7
Descripción: Reinicializa el controlador de sonido, cualquier emisión de sonido se detiene y apaga, y se borran las colas de espera. Esta rutina habilita las interrupciones.
Entrada: Sin parámetros.
Salida: AF, BC, DE y HL modificados.

SOUND QUEUE

Dirección: #BCAA
Descripción: Intenta añadir un sonido a la cola de espera de cual-

quier canal. En caso de que ésta se encontrara llena, no se produce ninguna acción.

Entrada: HL contiene la dirección de un programa de sonido.

Salida: Si el sonido se añadió a la cola o colas la bandera de acarreo vuelve a 1 y HL se modifica. Si al menos una cola estaba llena, el acarreo regresa a 0 y en todo caso, A, BC, DE, IX y el resto de banderas se modifican.

SOUND CHECK

Dirección: #BCAD

Descripción: Investiga la disponibilidad de espacio en un determinado canal y su estado. Las conexiones de los bits 0, 1 y 2 del Acumulador a la entrada de la rutina, comunican a ésta el canal a testear: A, B y C, respectivamente. En el supuesto de hallarse más de uno de esos bits a 1, se retornará el valor del primero de los estudiados, que siguen el orden ascendente del número de bit. La llamada a esta rutina desactiva el mecanismo de sucesos en cola de sonidos, cuando la cola tiene un espacio libre para el canal retornado. La rutina puede que habilite las interrupciones.

Entrada: A contiene el canal a investigar.

Salida: BC, DE, HL y las banderas se modifican. A contiene la información relativa al canal investigado, según la siguiente codificación de bits: 0, 1 y 2 el número de espacios libres en la cola de espera del canal; bit 3 a 1 cuando el canal espera para acorde con el canal A; 4 para espera de acorde con el canal B; 5 acorde con C; 6 canal retenido y bit 7 activado para canal en emisión de sonido.

SOUND ARM EVENT

Dirección: #BCB0

Descripción: Activa el mecanismo de sucesos para cuando queda espacio en una determinada cola de sonido de un canal. La codificación del canal al cual afectará el mecanismo se hace patente en el registro A, siguiendo la misma codificación que en la rutina anterior. Es preciso que el bloque de tratamiento por el mecanismo de suceso se halla habilitado previamente gracias a KL INIT EVENT. Por otra parte, el mecanismo se desactiva cada vez que se dispara, o bien con el

acceso a las rutinas SOUND QUEUE o SOUND CHECK. Esta rutina puede habilitar las interrupciones.

Entrada: A contiene el indicador de canal y HL la dirección del área de tratamiento del suceso.

Salida: AF, BC, DE y HL se modifican.

SOUND RELEASE

Dirección: #BCB3

Descripción: Libera los sonidos afectados por el bit de retención. La codificación del canal o canales a liberar se lleva a cabo en el acumulador siguiendo la misma codificación que en las rutinas anteriores, aunque en este caso, si se tomara en consideración la activación de más de uno de los tres bits o de los tres. La rutina puede habilitar las interrupciones.

Entrada: A contiene la codificación de bits de los canales a liberar.

Salida: AF, BC, DE, HL y las banderas se modifican.

SOUND HOLD

Dirección: #BCB6

Descripción: Retiene la emisión de todos los sonidos, la cual podrá continuarse, desde el punto más próximo posible a donde fue detenida, gracias a SOUND QUEUE, SOUND RELEASE o SOUND CONTINUE. Esta rutina habilita las interrupciones.

Entrada: Sin parámetros.

Salida: Si un sonido estaba emitiéndose, la bandera de acarreo vuelve a 1; de no ser así, a 0. En todo caso, A, BC, HL y el resto de las banderas se modifican.

SOUND CONTINUE

Dirección: #BCB9

Descripción: Continúa la emisión de los sonidos retenidos por SOUND HOLD. Esta rutina puede habilitar las interrupciones.

Entrada: Sin parámetros.

Salida: AF, BC, DE e IX se modifican.

SOUND AMPL ENVELOPE

Dirección: #BCBC

Descripción: Establece la forma de una determinada envolvente (entre 1 y 15). Para ello, toma la información sobre la forma de la envolvente de un bloque de datos, que puede hallarse tanto en ROM como en RAM, aunque no en una RAM bajo ROM, y cuyo significado en distribución de bytes es el siguiente: el byte 0 indica el número de secciones de esta envolvente y los restantes hasta el 15, constituyen triadas de bytes que dan forma a cada una de las cinco posibles secciones que pueden integrar la envolvente. Si el byte 0 del bloque viene a 0, es decir, indica que la envolvente carece de sección alguna, la rutina adopta la intención de un volumen constante de 2 segundos de duración.

En lo referente a las secciones, cada uno de sus tres bytes indican, respectivamente el número de pasos, el tamaño de los mismos y el tiempo de espera. Además, si el primer byte (número de pasos) es mayor que 127 (bit 7 a 1), se considera que la envolvente es hardware en vez de software.

En las envolventes software, los números de pasos en el margen 1 a 127, hacen que se añada el tamaño de los mismos al volumen el número de veces especificado por el número de pasos, con una pausa de una centésima de segundo multiplicado por el tiempo de espera, después de cada adición. En caso de que el número de pasos sea 0, se toma el tamaño de paso como un establecimiento de volumen absoluto, con una única espera de una centésima de segundo. Tras el cálculo del nuevo volumen, se le aplica la máscara #0F para asegurar su validez. Por otra parte, un tiempo de espera de 0 se toma por el valor 256 (256/100 segundos).

En lo referente a las envolventes hardware, sus bytes toman el significado siguiente: byte 0 forma de envolvente y bytes 1 y 2 período de la misma. La forma de la envolvente que se enmascara con #7F se envía al registro 13 del PSG (generador de sonido) indicando su forma y si esta se repite. En cuanto al período, se dirige a los registros 11 y 12, indicando la longitud de la envolvente hardware. En todo caso, cualquier sección de envolvente software, tras una sección hardware debería asegurarse que se toma el tiempo suficiente como para dejar actuar a la sección hardware, lo cual se puede conseguir con un tamaño de paso 0 y el número de pasos y tiempo de espera adecuados.

Entrada: A contiene un número de envolvente y HL la dirección del bloque de datos para su construcción.

Salida: Si la envolvente se ha generado correctamente, la bandera de acarreo vuelve a 1, HL contiene la dirección del bloque de datos más 16 y los registros A y BC se modifican; de no ser así, simplemente el acarreo regresa a 0. En todo caso, DE y el resto de las banderas se modifican.

SOUND TONE ENVELOPE

Dirección: #BCBF

Descripción: Establece la forma de una determinada envolvente (entre 1 y 15). Para ello, toma la información sobre la forma de la envolvente de un bloque de datos, que puede hallarse tanto en ROM como en RAM, aunque no es una RAM bajo ROM, y cuyo significado en distribución de bytes es el siguiente: el byte 0 indica el número de secciones de esta envolvente y los restantes hasta el 15, constituyen triadas de bytes que dan forma a cada una de las cinco posibles secciones que pueden integrar la envolvente. Si el byte 0 del bloque viene a 0, es decir, la envolvente carece de secciones, no se genera ninguna envolvente.

En lo referente a las secciones, cada uno de sus tres bytes indican respectivamente, el número de pasos, el tamaño de los mismos y el tiempo de espera. En esta ocasión, el bit 7 del número de pasos indica la repetición de la envolvente, lo cual implica que cuando se llega a la última sección, se recomienza la envolvente desde la primera.

Si el número de pasos se halla en el margen #00 a #EF, se considera una sección relativa. El tamaño de paso se considera extendido con signo, lo cual hace que se añada al período de tono actual el número de veces especificado por el número de pasos, con una pausa de una centésima de segundo multiplicado por el tiempo de espera, después de cada adición. El chip sólo utiliza los 12 bits menos significativos (módulo #1000). En caso de que el número de pasos sea 0, se toma como 1, mientras que un tiempo de espera de 0 se toma por el valor 256 (256/100 segundos).

Si el número de pasos se halla en el margen #F0 a #FF, se considera una sección absoluta. Los cuatro bits menos significativos del número de pasos se consideran como el byte más significativo del período de tono, y el tamaño de paso como el byte menos significativo. El período de tono se establece de inmediato, seguido de una pausa de el tiempo de espera en centésimas de segundo.

SOUND A ADDRESS

Dirección: #BCC2

Descripción: Investiga el área de ubicación de datos para una envolvente determinada.

Entrada: A contiene el número de envolvente.

Salida: Si se localiza la envolvente, la bandera de acarreo vuelve a 1, HL contiene la dirección del área de datos y BC la longitud de una envolvente (16 bytes); de no ser así, el acarreo vuelve a 0 y HL se modifica. En todo caso, A y el resto de las banderas se modifican.

SOUND T ADDRESS

Dirección: #BCC5

Descripción: Investiga el área de ubicación de una envolvente de tono.

Entrada: A contiene una envolvente de tono.

Salida: Idéntica a la rutina anterior.

EL KERNEL

Nos encontramos probablemente ante la zona del firmware de mayor complicación, lo cual implica que antes de enfrentarnos a ella debemos poseer ciertos conocimientos generales, por ejemplo sobre interrupciones, los cuales se escapan del objetivo de este volumen. No obstante, procedemos ahora a una descripción de las rutinas de Kernel, que puede servir de orientación para todos aquellos que deseen profundizar aún más en el sistema operativo, localizando en otras fuentes información adicional.

KL CHOKE OFF

Dirección: #BCC8

Descripción: Reinicializa el Kernel, borrando las colas de sucesos y los diversos contadores. No afecta a la generación de sonido ni a la investigación del estado del teclado.

Entrada: Sin parámetros.

Salida: B contiene la dirección de ROM seleccionada del progra-

ma ejecutado en ROM actualmente (si existe), C la del ejecutado en RAM y DE la indirección en la cual se entró al programa ROM actual. AF y HL se modifican.

KL ROM WALK

Dirección: #BCCB
Descripción: Encuentra e inicializa todas las ROM de fondo.
Entrada: DE contiene la primera dirección de memoria utilizable y HL la última.
Salida: DE contiene la nueva primera dirección de memoria utilizable y HL la nueva última dirección. AF y BC se modifican.

KL INIT BACK

Dirección: #BCCE
Descripción: Inicializa una ROM de fondo determinada.
Entrada: C indica la dirección de selección de ROM de la ROM a inicializar, DE contiene la primera dirección de memoria utilizable y HL la última.
Salida: DE contiene la nueva primera dirección de memoria utilizable y HL la nueva última dirección. AF y B se modifican.

KL LOG EXT

Dirección: #BCD1
Descripción: Introduce un RSX en el firmware.
Entrada: BC contiene la dirección de la tabla de comandos RSX y HL la del área de trabajo de 4 bytes en RAM a emplear por Kernel.
Salida: DE se modifica.

KL FIND COMAND

Dirección: #BCD4
Descripción: Localiza un comando en RSX, ROM de fondo o ROM de primer término. En este último caso, si la búsqueda tiene éxito se entra directamente en el comando, sin regresar de esta ruti-

na. El nombre del comando sólo tiene 16 caracteres significativos, aunque puede tener cualquier longitud, y su final debe marcarse con el bit siete del último carácter a 1, además de estar escrito todo en mayúsculas.

Entrada: HL es la dirección del nombre del comando a buscar.

Salida: Si se localiza en RSX o ROM de fondo, la bandera de acarreo vuelve a 1, C contiene la dirección de selección de ROM y HL la dirección de la rutina. Si el comando no se localizó, el acarreo regresa a 0 y C y HL se modifican. En todo caso, A, B y DE se modifican.

KL NEW FRAME FLY

Dirección: #BCD7

Descripción: El Kernel mantiene una lista de sucesos a atender en el momento oportuno. Esta es la rutina que inicializa un bloque de tratamiento de un suceso y lo añade a la lista (si no estaba ya en ella). Cada bloque está compuesto por 9 bytes, debiendo estar residente en las 32 K centrales (RAM). Esta rutina habilita las interrupciones.

Entrada: B contiene el tipo de suceso, C la dirección de selección de ROM donde se encuentra la rutina de tratamiento del suceso, DE la dirección de dicha rutina y HL la dirección del mecanismo de tratamiento de sucesos de Kernel.

Salida: AF, DE y HL se modifican.

KL ADD FRAME FLY

Dirección: #BCDA

Descripción: Añade un bloque de tratamiento de un suceso a la lista de Kernel.

Entrada: HL contiene la dirección del mecanismo de tratamiento de sucesos Kernel.

Salida: AF, DE y HL se modifican.

KL DEL FRAME BLOCK

Dirección: #BCDD

Descripción: Suprime un bloque de tratamiento de un suceso de la lista de Kernel.

Entrada: HL contiene la dirección del mecanismo de tratamiento de sucesos Kernel.

Salida: AF, DE y HL se modifican.

KL NEW FAST TICKER

Dirección: #BCE0

Descripción: El Kernel mantiene una lista de sucesos a atender cada vez que llega una interrupción del *timer* de 1/300 segundos. Esta es la rutina que inicializa un bloque de tratamiento de un suceso y lo añade a la lista (si no estaba ya en ella). Cada bloque está compuesto por 9 bytes, debiendo estar residente en las 32 K centrales (RAM). Esta rutina habilita las interrupciones.

Entrada: B contiene el tipo de suceso, C la dirección de selección de ROM donde se encuentra la rutina de tratamiento del suceso, DE la dirección de dicha rutina y HL la dirección del mecanismo de tratamiento de sucesos rápido de Kernel.

Salida: AF, DE y HL se modifican.

KL ADD FAST TICKER

Dirección: #BCE3

Descripción: Añade un bloque de tratamiento rápido de un suceso a la lista de Kernel.

Entrada: HL contiene la dirección del mecanismo de tratamiento rápido de sucesos Kernel.

Salida: AF, DE y HL se modifican.

KL DEL FAST TICKER

Dirección: #BCE6

Descripción: Suprime un bloque de tratamiento rápido de un suceso de la lista de Kernel.

Entrada: HL contiene la dirección del mecanismo de tratamiento rápido de sucesos Kernel.

Salida: AF, DE y HL se modifican.

KL ADD TICKER

Dirección: #BCE9

Descripción: Añade un bloque de tratamiento por decremento de suceso a la lista de Kernel. Dicha lista la integran bloques a los cuales se accede cada 1/50 segundos, para decrementar un contador; cuando éste llega a cero, se dispara la rutina de tratamiento de suceso y el contador se restablece a su valor original gracias a un byte del bloque, denominado valor de recarga, que contiene el valor original del contador que se decrementa.

Entrada: HL contiene la dirección del bloque de suceso por decremento, BC el valor para la entrada de recarga y DE el valor para la entrada del contador.

Salida: AF, BC, DE y HL se modifican.

KL DEL TICKER

Dirección: #BCEC

Descripción: Suprime un bloque de tratamiento por decremento de un suceso de la lista de Kernel.

Entrada: HL contiene la dirección del mecanismo de tratamiento por decremento de sucesos Kernel.

Salida: Si se encontró el suceso, la bandera de acarreo vuelve a 1 y DE contiene la cuenta restante hasta el próximo suceso; de no ser así, el acarreo regresa a 0 y DE modificado. En todo caso, A, HL y el resto de las banderas se modifican.

KL INIT EVENT

Dirección: #BCEF

Descripción: Inicializa todas las entradas en un bloque de sucesos. El bloque de sucesos tiene 7 bytes de largo y debe residir en las 32 K centrales de RAM. En cuanto al tipo de suceso, sus bits se distribuyen de la siguiente manera: 0 dirección próxima (a 0 implica ROM baja o 32 K centrales); bits 1 a 4 prioridad de suceso síncrono; el bit 5 debe ser cero; bit 6 suceso especial; y por último, el bit 7 denota un suceso asíncrono. El significado del bit de suceso especial depende directamente del estado de suceso síncrono o asíncrono.

Los sucesos síncronos especiales tienen prioridad superior a cual-

quier otro suceso síncrono normal, la cual se expresa en los bits 1 a 4, sin que ningún suceso pueda tener nunca prioridad 0. Una característica que distingue a los sucesos síncronos especiales es que estos no pueden ser deshabilitados mediante KL EVENT DISABLE, mientras que los síncronos normales sí.

Cuando de sucesos asíncronos se trata, los especiales se acceden directamente desde la interrupción, mientras que los normales se procesan justo antes de regresar de la interrupción.

Entrada: HL contiene la dirección del bloque de tratamiento de sucesos. B el tipo de suceso, C la dirección de selección de ROM para la rutina de tratamiento del suceso y DE la dirección de la rutina de tratamiento del suceso.

Salida: HL contiene la dirección del bloque de tratamiento se sucesos más 7.

KL EVENT

Dirección: #BCF2
Descripción: Dispara el tratamiento de un bloqueo de sucesos.
Entrada: HL contiene la dirección del bloque de sucesos.
Salida: AF, BC, DE y HL se modifican.

KL SYNC RESET

Dirección: #BCF5
Descripción: Borra la cola de espera de sucesos síncronos, reinicializando la prioridad actual de sucesos.
Entrada: Sin parámetros.
Salida: AF y HL se modifican.

KL DEL SYNCHRONOUS

Dirección: #BCF8
Descripción: Suprime un suceso síncrono de la cola de espera.
Entrada: HL contiene la dirección del bloque de tratamiento del suceso.
Salida: AF, BC, DE y HL se modifican.

KL NEXT SYNC

Dirección: #BCFB

Descripción: Procesa el próximo suceso síncrono de la cola de espera. Si existe algún suceso en la cola de sucesos síncronos cuya prioridad es mayor que la actual, lo suprime de la cola, establece el valor actual de la prioridad a la del suceso suprimido y retorna a la antigua.

Entrada: Sin parámetros.

Salida: Si existe suceso a procesar, la bandera de acarreo vuelve a 0, A contiene la prioridad anterior y HL la dirección del bloque de tratamiento del suceso; de no ser así, el acarreo retorna a 1 y tanto A como HL se modifican. En todo caso, DE se modifica.

KL DO SYNC

Dirección: #BCFE

Descripción: Ejecuta la rutina de tratamiento de un suceso determinado.

Entrada: HL contiene la dirección del bloque de tratamiento del suceso.

Salida: AF, BC, DE y HL se modifican.

KL DONE SYNC

Dirección: #BD01

Descripción: Una vez que se efectúa el proceso de un suceso síncrono mediante KL DO SYNC es preciso acceder a esta rutina para restablecer la prioridad actual y continuar con la cuenta para le ejecución de sucesos.

Entrada: A contiene la prioridad de suceso antigua y HL la dirección del bloque de tratamiento de suceso.

Salida: AF, BC, DE y HL se modifican.

KL EVENT DISABLE

Dirección: #BD04

Descripción: Evita el proceso de los sucesos síncronos normales (no los de tipo especial).

Entrada: Sin parámetros.
Salida: HL se modifica.

KL EVENT ENABLE

Dirección: #BD07
Descripción: Habilita el mecanismo de proceso de los sucesos síncronos normales.
Entrada: Sin parámetros.
Salida: HL se modifica.

KL DISARM EVENT

Dirección: #BD0A
Descripción: Deshabilita el suceso pasando su contador a valor negativo.
Entrada: HL contiene la dirección del bloque de tratamiento del suceso.
Salida: AF se modifica.

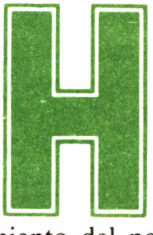
KL TIME PLEASE

Dirección: #BD0D
Descripción: Averigua el valor actual del contador (actualizado cada 1/300 segundos).
Entrada: Sin parámetros.
Salida: Los pares de registros DE y HL actúan conjuntamente para portar el contador de 4 bytes, en el cual 0 tiene el mayor peso y L el menor.

KL TIME SET

Dirección: #BD10
Descripción: Establece el valor del contador.
Entrada: Como en la rutina anterior, DEHL actúan conjuntamente para fijar el valor de 4 bytes del contador.
Salida: AF se modifica.

EL GESTOR DE HARDWARE A BAJO NIVEL



e aquí un breve grupo de rutinas, cuyo cometido fundamental es servir de interface a bajo nivel con el hardware integrante del Sistema, incluido el tratamiento del port Centronics y la carga y ejecución de programas.

Tengamos en cuenta, que el tratarse de rutinas de bajo nivel, implica que encontraremos en el firmware algunas otras con las que llevar a cabo determinada función, de una manera más cómoda que con estas de bajo nivel. En este sentido, se produce un fenómeno similar al que se da al tratar la pantalla desde el controlador de la misma, en vez de empleando las unidades de visualización de texto o gráficos (TEXT o GRAPHIC VDU).

No obstante, la presente descripción de las rutinas de tratamiento hardware, nos podrán proporcionar una idea de cómo opera el firmware a nivel de máquina, más que constituir herramientas que vayamos a manejar usualmente en nuestros programas.

MC BOOT PROGRAM

Dirección: #BD13

Descripción: Carga y ejecuta un programa. En caso de que falle el sistema de carga, se continúa la ejecución del programa actual. En todo caso, hemos de tener en cuenta que no se producirá el retorno desde esta rutina.

Entrada: HL contiene la dirección de la rutina de carga del programa.

Salida: No tiene salida.

MC START PROGRAM

Dirección: #BD16

Descripción: Inicializa completamente el Sistema e inicia la ejecución de un programa. Como en el caso anterior, se trata de una rutina sin retorno.

Entrada: C contiene la selección de ROM adecuada y HL la dirección de entrada.

Salida: No tiene salida.

MC WAIT FLYBACK

Dirección: #BD19

Descripción: Aguarda la llegada de una interrupción de barrido, es decir, la señal enviada por el CRTC cuando se inicia el período de retrazado vertical.

Entrada: Sin parámetros.

Salida: Sin modificación alguna.

MC SET MODE

Dirección: #BD1C

Descripción: Establece el modo de pantalla.

Entrada: A contiene el modo de pantalla.

Salida: AF se modifica.

MC SCREEN OFFSET

Dirección: #BD1F

Descripción: Establece la base y desplazamiento de pantalla para enviar al CRTC.

Entrada: A contiene la nueva base, en bloques de 16 Kb, y HL el desplazamiento de pantalla sobre la dirección base.

Salida: AF se modifica.

MC CLEAR INKS

Dirección: #BD22
Descripción: Habilita para todas las tintas al mismo color, de manera que se produce un efecto instantáneo de borrado de la pantalla. Establece también el color de marco.
Entrada: DE contiene la dirección de un vector de tinta, cuya forma es byte 0 para marco y byte 1 para tintas.
Salida: AF se modifica.

MC SET INKS

Dirección: #BD25
Descripción: Establece los colores para todas las tintas y el marco.
Entrada: DE contiene la dirección de un vector de tinta; en esta ocasión, los bytes 1 a 16 del vector contiene los valores para cada una de las tintas.
Salida: AF se modifica.

MC RESET PRINTER

Dirección: #BD28
Descripción: Reinicializa la indirección de impresora a su rutina por defecto (MC WAIT PRINTER).
Entrada: Sin parámetros.
Salida: AF, BC, DE y HL se modifican.

MC PRINT CHAR

Dirección: #BD2B
Descripción: Envía un carácter al port Centronics o espera un período de unos 0,4 segundos si la impresora está BUSY, de modo que se puede investigar la pulsación de BREAK.
Entrada: A contiene el carácter a enviar.
Salida: Si el carácter se pudo enviar, la bandera de acarreo vuelve a 1; de no ser así, a 0. En todo caso, A y el resto de las banderas se modifican.

MC BUSY PRINTER

Dirección: #BD2E
Descripción: Investiga si el port Centronics está ocupado (BUSY).
Entrada: Sin parámetros.
Salida: Si BUSY, el acarreo vuelve a 1, de no ser así, a 1. En todo caso, el resto de banderas se modifican.

MC SEND PRINTER

Dirección: #BD31
Descripción: Envía un carácter a la impresora, la cual no debe estar ocupada (BUSY).
Entrada: A contiene el carácter a enviar.
Salida: La bandera de acarreo vuelve a 1 y A y el resto de banderas se modifican.

MC SOUND REGISTER

Dirección: #BD34
Descripción: Envía un dato a un registro del PSG.
Entrada: A contiene el número de registro y C el dato.
Salida: AF y BC se modifican.

JUMP RESTORE

Dirección: #BD37
Descripción: Por proximidad y comodidad incluimos esta rutina en el presente bloque, aunque en realidad configura por sí misma una zona aparte; su misión es restaurar los valores de saltos por defecto del bloque principal de saltos del firmware que hemos venido estudiando.
Entrada: Sin parámetros.
Salida: AF, BC, DE y HL se modifican.

LAS INDIRECCIONES

asaremos ahora el estudio de aquellas zonas del firmware que hemos dado en llamar indirecciones, gracias a las cuales, se realizan funciones vitales del sistema, de manera que una alteración en las mismas nos permitirá variar ostensiblemente la forma de comportarse del firmware, sin necesidad de cambiar un gran número de rutinas.

IND: TXT DRAW CURSOR

Dirección: #BDCD
Descripción: Emplaza el cursor en la pantalla, siempre y cuando este se halle habilitado y encendido.
Entrada: Sin parámetros.
Salida: AF se modifica.

IND: TXT UNDRAW CURSOR

Dirección: #BDD0
Descripción: Cumple la función inversa a la rutina anterior.
Entrada: Sin parámetros.
Salida: AF se modifica.

IND: TXT WRITE CHAR

Dirección: #BDD3
Descripción: Escribe un carácter en la pantalla en una determinada posición de carácter, cuyas coordenadas se proporcionan como absolutas relativas a la esquina superior izquierda de la pantalla (0,0).
Entrada: A contiene el carácter a escribir, H la columna y L la fila.
Salida: AF, BC, DE y HL se modifican.

IND: TXT UNWRITE

Dirección: #BDD6
Descripción: Lee un carácter de la pantalla de la posición cuyas coordenadas se proporcionan, siguiendo la misma norma que en la rutina anterior.
Entrada: H y L contienen la columna y fila a leer, respectivamente.
Salida: Si el carácter se identificó, su código vuelve en A y la bandera de acarreo regresa a 1; de no ser así, la bandera de acarreo y A contienen 0. En todo caso, BC, DE, HL y el resto de banderas se modifican.

IND: TXT OUT ACTION

Dirección: #BDD9
Descripción: Imprime un carácter u obedece un código de control.
Entrada: A contiene el carácter o código.
Salida: AF, BC, DE y HL se modifican.

IND: GRA PLOT

Dirección: #BDDC
Descripción: Enciende un pixel de la pantalla, cuyas coordenadas se proporcionan en referencia al origen establecido por el usuario.
Entrada: DE contiene la coordenada X y HL la Y.
Salida: AF, BC, DE y HL se modifican.

IND: GRA TEST

Dirección: #BDDF
Descripción: Investiga si un pixel de la pantalla se halla encendido, retornando su estado.
Entrada: Como en la rutina anterior.
Salida: A contiene el valor decodificado de la tinta. BC, DE y HL se modifican.

IND: GRA LINE

Dirección: #BDE2
Descripción: Traza una línea en el modo actual y con la tinta gráfica actual, entre la posición del cursor gráfico y un punto final cuyas coordenadas se facilitan absolutas con respecto al origen fijado por el usuario.
Entrada: DE contiene la coordenada X del punto final y HL la Y.
Salida: AF, BC, DE y HL se modifican.

IND: SCR READ

Dirección: #BDE5
Descripción: Lee y decodifica la tinta de un pixel de la pantalla.
Entrada: HL contiene la dirección del byte de pantalla en el cual se encuentra el pixel y C la máscara para determinar su posición exacta dentro del byte.
Salida: A contiene la tinta decodificada y las banderas se modifican.

IND: SCR WRITE

Dirección: #BDE8
Descripción: Activa un pixel o pixels de la pantalla siguiendo el modo actual de escritura gráfica.
Entrada: HL contiene la dirección del byte de pantalla en el cual se encuentra el pixel, C la máscara para determinar su posición exacta dentro del byte y B la tinta codificada.
Salida: AF se modifica.

IND: SCR MODE CLEAR

Dirección: #BDEB
Descripción: Borra la pantalla llenando su memoria con la tinta cero, tal como ocurre al cambiar de modo de pantalla.
Entrada: Sin parámetros.
Salida: AF, BC, DE y HL se modifican.

IND: KM TEST BREAK

Dirección: #BDEE
Descripción: Comprueba la pulsación de ESCape y de la secuencia de reinicialización SHIFT (MAYS) + CONTROL + ESCape. A esta rutina debe accederse con las interrupciones deshabilitadas.
Entrada: C contiene el estado de las teclas SHIFT (MAYS) y CONTROL.
Salida: AF y HL se modifican.

IND: MC WAIT PRINTER

Dirección: #BDF1
Descripción: En caso de que la impresora conectada al port Centronics se encuentre en situación de disponible (no BUSY), esta rutina envía un carácter. De no ser así, espera un período de tiempo para comprobar si puede enviar el carácter.
Entrada: A contiene el carácter a enviar.
Salida: Si fue posible enviar el carácter, la bandera de acarreo vuelve a 1; si se agotó el tiempo de espera sin que la impresora dejara de estar ocupada (BUSY), regresa el acarreo a 0. En todo caso, A y BC se modifican.

TABLA DE SALTOS DEL KERNEL ALTO

sta zona de saltos se halla separada del bloque principal, y sus rutinas se encuentran fundamentalmente relacionadas con los estados de ROM y su selección.

HI: KL U ROM ENABLE

Dirección: #B900
Descripción: Habilita la ROM superior.
Entrada: Sin parámetros.
Salida: A contiene el estado previo de la ROM. Las banderas se modifican.

HI: KL U ROM DISABLE

Dirección: #B903
Descripción: Deshabilita la ROM superior.
Entrada: Sin parámetros.
Salida: A contiene el estado previo de la ROM. Las banderas se modifican.

HI: KL L ROM ENABLE

Dirección: #B906
Descripción: Habilita la ROM inferior.
Entrada: Sin parámetros.
Salida: A contiene el estado previo de la ROM. Las banderas se modifican.

HI: KL L ROM DISABLE

Dirección: #B909
Descripción: Deshabilita la ROM inferior.
Entrada: Sin parámetros.
Salida: A contiene el estado previo de la ROM. Las banderas se modifican.

HI: KL ROM RESTORE

Dirección: #B90C
Descripción: Restaura el estado previo de la ROM.
Entrada: A contiene el estado previo de la ROM.
Salida: AF se modifica.

HI: KL ROM SELECT

Dirección: #B90F
Descripción: Selecciona una ROM superior determinada y la habilita.
Entrada: C contiene la dirección de selección de la ROM.
Salida: C contiene la dirección de selección de la ROM anteriormente seleccionada y B su estado previo. AF se modifica.

HI: KL CURR SELECTION

Dirección: #B912
Descripción: Investiga qué ROM superior se halla actualmente seleccionada.

Entrada: Sin parámetros.
Salida: A contiene la dirección de selección de la ROM actualmente seleccionada.

HI: KL PROBE ROM

Dirección: #B915
Descripción: Investiga el tipo y versión de una ROM.
Entrada: C contiene la dirección de selección de la ROM a investigar.
Salida: A contiene el tipo de ROM, L su número de marca, H su número de versión y B y las banderas se modifican.

HI: KL ROM DESELECT

Dirección: #B918
Descripción: Restaura el estado y selección de ROM previo al último acceso a KL ROM SELECT.
Entrada: B contiene el estado previo y C la dirección de selección de la ROM previa.
Salida: C contiene la dirección de selección de la ROM actualmente seleccionada y B se modifica.

HI: KL LDIR

Dirección: #B91B
Descripción: Efectúa una instrucción Z80 LDIR de la RAM, con las dos ROM (superior e inferior) deshabilitadas.
Entrada: Los parámetros BC, DE y HL requeridos por LDIR.
Salida: BC, DE, HL y las banderas tal como las deja LDIR.

HI: KL LDDR

Dirección: #B91E
Descripción: Efectúa una instrucción Z80 LDDR de la RAM, con las dos ROM (superior e inferior) deshabilitadas.
Entrada: Los parámetros BC, DE y HL requeridos por LDDR.
Salida: BC, DE, HL y las banderas tal como las deja LDDR.

HI: KL POLL SYNCHRONOUS

Dirección: #B921

Descripción: Comprueba si un suceso con mayor prioridad que el actual se halla pendiente de atención en la cola de espera.

Entrada: Sin parámetros.

Salida: Si existe un suceso de mayor prioridad, la bandera de acarreo vuelve a 1; de no ser así, a 0. En todo caso, A y el resto de las banderas se modifican.

TABLA DE SALTOS DEL KERNEL BAJO

La zona baja de la memoria, que comprende desde las direcciones #0000 a #003F, contiene las RST y cierto número de rutinas de uso fundamental para el Kernel. Como quiera que esta información puede ser preciso utilizarla con la ROM inferior habilitada, dichas rutinas se copian a la RAM durante el proceso de inicialización.

En cuanto a las RST se refiere, es de interés destacar la presencia del RST 7, que controla las interrupciones y del RST 6, de gran importancia, puesto que se encuentra disponible para el usuario. Por último, mientras que la RST 0 se encarga de la reinicialización del sistema, el resto (de 1 a 5) se destinan a incrementar la flexibilidad del juego de instrucciones del Z80, en lo que se refiere a saltos y llamadas.

LOW: RESET ENTRY

Dirección: #0000

Descripción: Esta rutina puede accederse mediante RST 0. Efectúa la reinicialización completa del Sistema, tanto hardware como software, como en el proceso de encendido de la máquina. Una vez

finalizada su misión, se transfiere el control a la entrada por defecto de la ROM 0; así pues, esta rutina no tiene retorno.

Entrada: Sin parámetros.

Salida: Sin retorno.

LOW: LOW JUMP

Dirección: #0008

Descripción: Esta rutina puede accederse mediante RST 1. Salta a la ROM inferior o RAM, tomando como dirección de salto la contenida en los dos bytes siguientes, de los cuales los bits 14 y 15 indican los estados requeridos de la ROM inferior y superior, respectivamente (a 1 habilitada y a 0 deshabilitada).

Entrada: Sin parámetros.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada.

LOW: KL LOW PCHL

Dirección: #000B

Descripción: Salta a la ROM inferior o RAM, tomando como dirección de salto la contenida en los dos bytes de HL, de los cuales los bits 14 y 15 indican los estados requeridos de la ROM inferior y superior, respectivamente (a 1 habilitada y a 0 deshabilitada).

Entrada: HL contiene la dirección de salto (bits 0 a 13) y los estados requeridos de ROM (bits 14 y 15), aunque como en la rutina anterior, todos los registros y banderas se pasan intactos a la rutina que se va a acceder.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada.

LOW: PCBC INSTRUCTION

Dirección: #000E

Descripción: Implementa una instrucción JP (BC) de la cual carece el Z80. Su efecto es en todo similar a JP (HL), aunque actuando con BC para determinar la dirección del salto.

Entrada: BC contiene la dirección de salto aunque como en la rutina anterior, todos los registros y banderas se pasan intactos a la rutina que se va a acceder.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada.

LOW: SIDE CALL

Dirección: #0010

Descripción: Esta rutina puede accederse mediante RST 2. Efectúa una llamada de tipo CALL a una ROM paralela, cuya dirección se proporciona a continuación. Los dos bytes siguientes contienen la dirección de la rutina a acceder y el incremento de ROM sobre la actualmente seleccionada. Los bits 0 a 13 contienen una dirección que sumada a #C000 proporcionan la dirección definitiva a llamar y los bits 14 y 15 toman valores de 0 a 3, para indicar la cantidad a sumar a la actual dirección de selección de ROM para obtener la dirección de selección de ROM adecuada.

Entrada: Todos los registros y banderas se transfieren sin alteración a la rutina a llamar, a excepción de IY, que apunta a un área de datos en una ROM superior de fondo.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada, a excepción de IY que no se modifica.

LOW: KL SIDE PCHL

Dirección: #0013

Descripción: Efectúa un salto del tipo JP (HL) a una ROM superior paralela; la dirección a la cual acceder se proporciona a continuación, siguiendo la misma norma que en la rutina anterior.

Entrada: HL contiene la dirección de la ROM superior paralela a acceder (más #C000) y el incremento para obtener la ROM adecuada. Todos los registros y banderas se transfieren sin alteración a la rutina a llamar, a excepción de IY, que apunta a un área de datos en una ROM superior de fondo.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada, a excepción de IY que no se modifica.

LOW: PCDE INSTRUCTION

Dirección: #0016

Descripción: Implementa una instrucción JP (DE) de la cual carece el Z80. Su efecto es en todo similar a JP (HL), aunque actuando con DE para determinar la dirección del salto.

Entrada: DE contiene la dirección de salto aunque como en la rutina anterior, todos los registros y banderas se pasan intactos a la rutina que se va a acceder.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada.

LOW: FAR CALL

Dirección: #0018

Descripción: Esta rutina puede accederse mediante RST 3. Efectúa una llamada a una subrutina de tipo CALL en RAM o ROM. La dirección se indica en los dos bytes subsiguientes, y el estado de ROM en un tercer byte.

Entrada: Todos los registros y banderas se transfieren sin alteración a la rutina a llamar, a excepción de IY, que apunta a un área de datos en una ROM superior de fondo.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada, a excepción de IY que no se modifica.

LOW: HL FAR PCHL

Dirección: #001B

Descripción: Realiza una función análoga a la de la rutina anterior, aunque la dirección de llamada se proporciona en HL y la ROM en C.

Entrada: HL contiene la dirección de llamada y C la ROM adecuada, aunque todos los registros y banderas se transfieren sin alteración a la rutina a llamar, a excepción de IY, que apunta a un área de datos en una ROM superior de fondo.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada, a excepción de IY que no se modifica.

LOW: PCHL INSTRUCTION

Dirección: #001E
Descripción: Su efecto es en todo similar a JP (HL).
Entrada: HL contiene la dirección de salto aunque todos los registros y banderas se pasan intactos a la rutina que se va acceder.
Salida: Las condiciones que imponga la rutina accedida a través de esta llamada.

LOW: RAM LAM

Dirección: #0020
Descripción: Esta rutina puede accederse mediante RST 4. Realiza la misma función de la instrucción LD A,(HL), aunque tomando siempre el contenido de RAM, cualquiera que sea el estado de habilitación de ROMs.
Entrada: HL contiene la dirección de RAM a leer.
Salida: A contiene el valor almacenado en la dirección RAM que señala HL.

LOW: KL FAR ICALL

Dirección: #0023
Descripción: Efectúa una llamada a una subrutina de tipo CALL en RAM o ROM. La dirección se indica en los dos bytes, y el estado de ROM en un tercer byte; este área de datos viene indicada por HL.
Entrada: HL apunta a la zona de 3 bytes donde se especifica la dirección a llamar y el estado de ROM. Todos los registros y banderas se transfieren sin alteración a la rutina a acceder, a excepción de IY, que apunta a un área de datos en una ROM superior de fondo.
Salida: Las condiciones que imponga la rutina accedida a través de esta llamada, a excepción de IY que no se modifica.

LOW: FIRM JUMP

Dirección: #0028
Descripción: Esta rutina puede accederse mediante RST 5. Efectúa un salto a la dirección de la ROM inferior o las 32 Kb centrales de

RAM, cuya dirección se indica en los dos bytes que siguen. La ROM inferior se habilita antes del salto y se deshabilita (en vez de restaurarse) a su retorno.

Entrada: Todos los registros y banderas se transfieren sin alteración a la rutina a llamar.

Salida: Las condiciones que imponga la rutina accedida a través de esta llamada.

LOW: USER RESTART

Dirección: #0030

Descripción: A esta rutina se asignan las direcciones de #0030 a #0037, accesibles mediante RST 6, que se encuentran a disposición del usuario.

Entrada: Ninguna.

Salida: Ninguna.

LOW: INTERRUPT ENTRY

Dirección: #0038

Descripción: Esta rutina puede accederse mediante RST 7. Es el punto de entrada de la rutina de interrupción hardware.

Entrada: Sin parámetros.

Salida: Ningún registro ni bandera se modifica.

LOW: EXT INTERRUPT

Dirección: #003B

Descripción: Esta rutina controla las interrupciones generadas externamente. Cuando se detecta la señal de interrupción, la ROM inferior se deshabilita y se accede a esta rutina.

Entrada: Sin parámetros.

Salida: AF, BC, DE y HL se modifican.

Nos adentraremos ahora en el complejo mundo del código máquina, continuando la ardua tarea emprendida en el décimo volumen de esta GRAN BIBLIOTECA AMSTRAD. Desde la generación de las RSX (extensiones del sistema residentes), tan «temidas» como útiles, hasta una completa descripción de las rutinas del firmware.

450 ptas.
(incluido IVA)
Precio en Canarias, Ceuta y Melilla: 435 ptas.