

Domine  
el Código Máquina  
en su

**AMSTRAD**

CPC 6128-664-464

CLIVE GIFFORD - SCOTT VINCENT



**ra-ma**

Domine  
el Código Máquina  
en su

AMSTRAD

CPC 6128-664-464

CLIVE GIFFORD - SCOTT VINCENT



Título de la obra original  
MASTERING MACHINE CODE ON YOUR  
AMSTRAD 464/664/6128  
por Clive Gifford y Scott Vincent

Copyright (C) 1986, Gifford/Vincent

Traducido del inglés por  
M. C. Dopazo

Primera edición en español

Copyright (C) 1986

**RA-MA**

Carretera de Canillas, 144  
28043 MADRID

I.S.B.N.: 84-86381-15-0

Depósito Legal: M. 17825 - 1986

#### RESERVADOS TODOS LOS DERECHOS

Prohibida la reproducción parcial o total de este libro, así como el almacenamiento en un sistema informático, la transmisión en cualquier forma o por cualquier medio, electrónico, mecánico, fotocopia, registro o cualquier otro métodos, sin el permiso previo y por escrito de los propietarios del Copyright, que, así mismo, tienen los derechos exclusivos sobre las rutinas incluidas en este libro, y sus nombres.

Existe un disco, para el Amstrad CPC 6128, que contiene todas las rutinas que aparecen en este libro, grabadas y debidamente comprobadas.

Para obtener una copia, dirigirse directamente a editorial RA-MA, Carretera de Canillas, 144, 28043 Madrid.

Cualquier consulta referida a este libro o a su contenido, deberá dirigirse a Editorial RA-MA

# CONTENIDO

Contenido - Tim Hartnell

Introducción de los autores

<b>Capítulo uno</b> .....	1
¿Qué es el código máquina? .....	1
Juego de instrucciones .....	2
A favor .....	2
Argumentos en contra .....	3
<b>Capítulo dos</b> .....	5
Sistemas numéricos y lenguaje ensamblador ...	5
Números de ocho y diez y seis bits .....	6
Lenguaje ensamblador .....	7
Comprando un ensamblador .....	8
Directrices .....	9
<b>Capítulo tres</b> .....	11
Usando el código máquina en el Amstrad .....	11
El hardware .....	11
El sonido .....	12
Escribiendo programas en código máquina .....	12
SYMBOL AFTER .....	13
Salvando el programa .....	14
Carga del programa .....	14
<b>Capítulo cuatro</b> .....	15
Su primer programa en código máquina .....	15
Un cargador hex .....	15
El programa .....	16
Variables .....	17

<b>Capítulo cinco</b> .....	21
Pasando parámetros .....	21
Cambio del octeto inferior .....	22
Manejando cadenas .....	23
Una rutina de tratamiento de cadenas .....	23
<b>Capítulo seis</b> .....	25
Aritmética simple .....	25
Suma y resta .....	26
Punto y acarreo .....	28
No hay división .....	28
<b>Capítulo siete</b> .....	31
Pila y saltos .....	31
La pila .....	31
PUSH .....	33
POP .....	33
Alterando el SP .....	34
Tenga cuidado .....	36
No hace nada .....	37
JR y JP .....	38
CALL .....	41
<b>Capítulo ocho</b> .....	45
Diccionario de términos de código máquina ..	45
Registros .....	45
IX .....	45
IY .....	46
SP .....	46
A .....	46
B,C,D,E,H y L .....	46
El juego alternativo de registros .....	46
F .....	46
Todas las instrucciones .....	47
ADC .....	47
ADD .....	47
AND .....	47
BIT .....	47
CALL .....	48
CCF .....	48
CP .....	48
CPD .....	48
CPDR .....	48

CPI	48
CPIR	48
CPL	48
DAA	49
DEC	49
DEFB	49
DEFM	49
DEFS	50
DEFW	50
DI	50
DJNZ	50
EI	50
EQU	50
EX	51
EXX	51
HALT	51
IM	51
IN	51
INC	51
IND	52
INDR	52
INI	52
INIR	52
JP	52
JR	52
LD	52
LDD	53
LDDR	53
LDI	53
LDIR	53
NEG	53
NOP	53
OR	54
ORG	54
OUT	54
OUTD	54
OTDR	54
OUTI	54
OTIR	54
POP	54
PUSH	55
RES	55
RET	55
RL	55
RLA	55
RLC	55
RLCA	56

RLD .....	56
RR .....	56
RRA .....	56
RRC .....	56
RRCA .....	56
RRD .....	56
RST .....	56
SBC .....	56
SCF .....	56
SET .....	57
SLA .....	57
SRA .....	57
SRL .....	57
SUB .....	57
XOR .....	57
<b>Capítulo nueve .....</b>	<b>59</b>
Operadores lógicos y manipulación de bits ..	59
AND .....	59
OR .....	60
XOR .....	61
SET y RES .....	62
Girar y desplazar .....	63
RLC y RRC .....	64
RL .....	65
SLA .....	65
<b>Capítulo Diez .....</b>	<b>67</b>
Pantalla y rutinas de la ROM .....	67
Doscientos de alto .....	67
<b>Paquete de rutinas en código máquina .....</b>	<b>71</b>
<b>Uno - Leer un carácter .....</b>	<b>73</b>
BASIC .....	73
Ensamblador .....	74
<b>Dos - Girar hacia la izquierda .....</b>	<b>75</b>
BASIC .....	75
Ensamblador .....	76

<b>Tres - Girar hacia la derecha</b> .....	79
BASIC .....	79
Ensamblador .....	80
<b>Cuatro - Letras gigantes</b> .....	83
BASIC .....	83
Ensamblador .....	84
<b>Cinco - Impresión masiva</b> .....	89
BASIC .....	89
Ensamblador .....	90
<b>Seis - Relleno de la pantalla</b> .....	93
BASIC .....	93
Ensamblador .....	94
<b>Siete - LOAD/SAVE sin cabecera</b> .....	95
BASIC .....	95
Ensamblador .....	96
<b>Ocho - Música por interrupciones</b> .....	99
BASIC .....	100
Ensamblador .....	103
<b>Nueve - Monitor de código máquina</b> .....	115
BASIC .....	115
Ensamblador .....	119
<b>Diez - Movimiento de bloques</b> .....	125
Bloque hacia arriba .....	125
BASIC .....	125
Ensamblador .....	126
Bloque hacia abajo .....	130
BASIC .....	130
Ensamblador .....	131

Bloque hacia la izquierda .....	134
BASIC .....	134
Ensamblador .....	136
Bloque hacia la derecha .....	141
BASIC .....	142
Ensamblador .....	144
<b>Once - Acordes RSX .....</b>	<b>151</b>
BASIC .....	152
Envolventes.....	154
Ensamblador .....	156
Organo de acordes .....	163
BASIC .....	163
<b>Doce - Compresores de pantalla .....</b>	<b>165</b>
Compresor 1 .....	165
BASIC .....	165
Ensamblador .....	166
Compresor 2 .....	168
BASIC .....	169
Ensamblador .....	170
<b>Trece - DOKE y DEEK .....</b>	<b>173</b>
BASIC .....	173
Ensamblador .....	174
<b>Catorce - El paquete de escritura de juegos .....</b>	<b>177</b>
BASIC .....	177
Bombardero .....	182
<b>Apéndices.</b>	
Apéndice A .....	189
Mapa de memoria.....	189
Apéndice B.....	191
Códigos de operación Z80 .....	191
Apéndice C .....	227
Conversión hexadecimal a decimal .....	227

## **Introducción de los Autores**

Esperamos que este libro le ayude a dominar la programación en código máquina. Hemos intentado escribir una guía fácil de seguir, para dar una visión interna del trabajo del código máquina y de sus comandos más importantes, así como proporcionar un grupo de cuadros y tablas útiles. Estos ocupan casi la mitad del libro. El balance final es una generosa selección de rutinas en código máquina. Estas rutinas incluyen un cargador BASIC fácil de usar, que se puede teclear y poner en funcionamiento en pocos minutos. Y, aunque no pretenda aprender en este preciso momento las interioridades del código máquina, este libro le proporcionará una interesante librería de útiles y (en algunos casos) divertidas rutinas.

Escribir este libro ha sido un trabajo duro pero divertido. Esperamos que que usted obtenga tanto beneficio leyéndolo como nosotros escribiéndolo.

**Scott Vincent  
Clive Gifford**

**Londres, Marzo 1986**

## **Prefacio - Tim Hartnell**

Ahora tiene la oportunidad de aprender a programar en código máquina en su ordenador Amstrad. No le importe lo que diga la gente, no es tan fácil como para aprenderlo sin cierta dificultad.

Sin embargo, con una buena guía, aún el terreno más difícil se puede vadear. Creo que Clive y Scott - dos programadores muy competentes, con gran experiencia en libros y 'software' a sus espaldas - son los guías ideales para ayudarle a comprender las interioridades de la programación en código máquina del Amstrad.

Debe ir trabajando a lo largo del libro, saltándose las secciones que le presenten una especial dificultad la primera vez que las lea. Cuando haya terminado su primera lectura, tendrá los suficientes conocimientos como para poder comprender aquellas secciones que dejó sin completar la primera vez que pasó por ellas.

Si lo hace así, verá que es mucho más fácil la tarea, y sus progresos no se verán detenidos por una sección particular que parezca más difícil que las otras.

Y no olvide las rutinas de código máquina, listas para funcionar, de este libro (incluido un paquete completo para escribir programas de juegos) que puede usar junto con sus programas BASIC, aunque en este momento no comprenda cómo funcionan.

Ahora ya es el momento de comenzar a aprender a dominar el código máquina en su Amstrad.

**Tim Hartnell,  
Londres, 1986**

# CAPITULO UNO

## ¿Qué es el Código Máquina?

Lo primero que debemos saber es que el código máquina no es un código mágico que crea instantáneamente programas de la misma calidad que el 'software' comercial. Es lento y laborioso de escribir, muy difícil de depurar y frecuentemente produce resultados que son muy poco mejores que una rutina similar en BASIC. No es un lenguaje maravilloso y creemos que se le da demasiada importancia a los términos 'programa 100% en código máquina' y 'programador de código máquina'.

Para comprender como funciona el código máquina, necesitamos echar primero una breve ojeada al trabajo interno de su ordenador. El procesador de su ordenador consta de miles de "puertas", o interruptores que pueden estar abiertos(off) o cerrados(on). Se pueden usar números binarios para representar sus posiciones fácilmente, haciendo una correspondencia entre el uno=cerrado, y el cero=abierto. Sin embargo, introducir cientos de unos y ceros para conseguir un comando que funcione no es un uso eficiente de nuestro tiempo, por esto se crearon los lenguajes de "alto nivel". El BASIC es uno de estos lenguajes. Reemplaza la masa de ceros y unos por algo fácil de entender, comandos similares a palabras inglesas. No es muy difícil de comprender lo que hacen comandos como PRINT, CLEAR y END (suponiendo que tenemos un ligero conocimiento del idioma inglés, si no es así, no se preocupe, son pocas y de fácil comprensión cuando se han usado unas cuantas veces). Se necesita un intérprete para traducir los comandos BASIC a dígitos binarios que pueda entender el ordenador, y esta interpretación es la que retarda enormemente la velocidad de operación.

El código máquina es la serie de instrucciones a 'nivel de máquina', en las que se convierte el BASIC. Estas instrucciones de bajo nivel pueden ser ejecutadas directamente por el procesador.

El 'cerebro' de su ordenador, donde se desarrolla la mayor parte de la acción, es la Unidad Central de Proceso o CPU. Cada diseño de CPU tiene su propia colección de instrucciones de máquina de bajo nivel, conocidas como Juego de Instrucciones.

La CPU de su Amstrad es uno de los diseños más populares que se han fabricado en microprocesadores, el chip CPU Z80 de Zilog. Por esto, este libro trata de la programación en código máquina Z80 y de cómo usar el Juego de Instrucciones Z80. (Muchos otros ordenadores usan el chip Z80, el Spectrum y los de la serie MSX son algunos de ellos. Pero en cada máquina hay unas determinadas condiciones que, aunque usen el mismo procesador Z80, las hace ligeramente diferentes desde el punto de vista de su programación en código máquina. Un buen ejemplo es la forma poco usual con que el Amstrad maneja la pantalla).

## El Juego de Instrucciones

Cada instrucción de código máquina forma parte del lenguaje de bajo nivel. Pueden necesitarse media docena o más de instrucciones en código máquina para emular un comando simple de BASIC.

Hay muchas maneras de representar el código máquina. Una de estas formas es el Lenguaje Ensamblador que utiliza un nombre descriptivo pequeño, conocido como *nemotécnico*, para cada instrucción. Otro método consiste en usar un número decimal. También se pueden usar números binarios y hexadecimales (base 16, que son los que usaremos en los siguientes capítulos). Por ejemplo, la instrucción que hace que el procesador regrese desde el código máquina al BASIC tiene el valor decimal 201, su valor en binario sería 11001001, en hexadecimal tendría el valor C9 y en nemotécnico sería RET.

Ahora que tenemos una breve idea de lo que es el código máquina, echemos una ojeada a las razones a favor y en contra de su uso.

### A Favor

El código máquina, cuando está bien escrito, suele ser bastante más rápido que una rutina comparable en BASIC.

El código máquina le permite realizar acciones que son imposibles en BASIC, tales como síntesis de la palabra. Así mismo, el código máquina suele ocupar mucha menos memoria que un programa similar en BASIC.

También conlleva un gran prestigio el ser capaz de progra-

mar en código máquina. Programas que funcionan perfectamente en BASIC suelen ser rechazados por los críticos porque no están escritos en código máquina, aunque no produzca ninguna mejora el hacerlo.

### Argumentos en Contra

El código máquina suele necesitar muchas instrucciones para emular una acción simple.

El código máquina tiende a ser difícil de leer, comprender y depurar. Escribir programas en código máquina suele ser más difícil y ocupar más tiempo que escribirlos en un lenguaje de alto nivel como el BASIC.

En código máquina es muy difícil realizar cualquier cosa más allá de la simple aritmética, y el código puede ser difícil de transferir de una máquina a otra. Es bastante difícil transferir un programa en código máquina desde una máquina con procesador Z80 a otra también basada en el Z80, pero intentar transferir una rutina Z80 a, pongamos, un Commodore 64 (que usa la CPU 6502) es prácticamente imposible.

# CAPITULO DOS

## Sistemas Numéricos y Lenguaje Ensamblador

Los tres sistemas numéricos que suelen usarse más frecuentemente en código máquina son decimal, binario y hexadecimal.

Como ya sabe, los números decimales se construyen en base diez, los binarios en base dos, y los hexadecimales en base diez y seis.

Imagine que dentro de la CPU de su Amstrad hay una pila de pequeñas casillas, llamadas direcciones. Cada dirección puede contener ocho dígitos binarios (y un grupo de ocho dígitos binarios es conocido, por alguna oculta razón, como octeto o byte). Obviamente, si los ocho dígitos son ceros, el número contenido en esa dirección será el cero. Sin embargo, si los ocho dígitos son unos (ej. 11111111), la dirección contendrá el número 255. Por lo tanto, una dirección puede contener un número entre cero y 255.

El comando BIN\$ del Amstrad convierte números decimales en sus equivalentes binarios:

```
BIN$(V,N)
```

En esta sentencia, V es el valor en decimal que va a ser convertido y N es el número de dígitos mostrados. (El último parámetro no es necesario, en cuyo caso el Amstrad imprimirá el resultado sin ningún cero delante. PRINT BIN\$(4) nos dará como respuesta 100, mientras que si especifica que quiere ocho dígitos en el resultado, obtendremos 00000100).

Hexadecimal (o "hex") es el nombre correcto del sistema de numeración basado en diez y seis. El rango de dígitos es 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. La notación hexadecimal se usa en código máquina debido a su estrecha relación con, y su facilidad para convertir en binario, el

sistema de numeración natural de los ordenadores. Ahora, un dígito hexadecimal es igual a cuatro dígitos binarios(bits) o a medio octeto (un octeto contiene ocho bits). ¡Al medio octeto se le llama "nibble"! Por lo tanto, 0 en hex es igual a 0000 en binario, 2 en hex es 0010 en binario y F en hex es igual a 1111 en binario. PRINT HEX\$(201) producirá como respuesta C9. Este resultado se obtiene multiplicando el primer dígito hex por 16 (C=12) y sumándole el valor decimal del segundo dígito hex (9=9), o  $192+9$  que es igual a 201.

Si quiere incluir un número hex en su programa, debe ir precedido por "&" o "&H". Un número binario debe ir precedido por "&X".

### Números de Ocho y Diez y seis Bits

Hasta este momento hemos tratado con números en el rango decimal de 0 a 255. Estos son, como dijimos anteriormente, números de 8 bits. Sin embargo, para obtener números mayores que 255 necesitamos números de 8 octetos (16 bits). Estos nos dan un rango bastante más alto, entre 0 y 65535. Si tenemos un número de 16 bits que en hex se representa como 03C9, el 03 se conoce como el octeto **más significativo** o **alto**, mientras que el otro octeto, C9, es el octeto **bajo** o **menos significativo**. El valor total de un número de 16 bits se calcula así:

256 veces el octeto alto más el octeto bajo

$$256 * 03 + C9 = 256 * 3 + 201 = 969$$

Por lo tanto, hemos visto cómo se obtienen los números positivos entre 0 y 65535, pero ¿qué pasa con los números negativos?. Dentro de un octeto tenemos el bit más significativo y el menos significativo. El bit más significativo es el de más a la izquierda, y el de más a la derecha es el menos significativo. El bit más significativo (cuyo valor puede ser 0 ó 128) lo usamos como **signo** del número. Esto nos acorta el tamaño máximo de un número de 8 bits, a 127. Sin embargo mantiene el rango de 256, ya que ahora podemos representar cualquier valor entre -128 y +127. Si ponemos el bit de signo a cero, hacemos que el resto del octeto tome un valor **positivo**, mientras que si lo ponemos a uno, tendremos un valor **negativo**. Los números de 16 bits funcionan de forma similar, usando el bit de más a la izquierda del octeto más

significativo como **bit de signo**, que es como se le llama.

Sin embargo, crear números binarios negativos no es tan simple como parece. Se debe seguir la regla de que la suma del correspondiente número positivo al valor negativo debe dar cero. Para conseguirlo, debemos usar un concepto numérico llamado **Complemento a Dos**. Este proceso se hace en dos partes. Primero, se debe invertir el valor de cada bit dentro del octeto, de forma que si un bit es igual a 1, debe pasar a cero y viceversa. Este es el proceso real de complementar. Una vez que se ha hecho esto, se añade 1 al resultado, obteniendo el valor binario de signo opuesto al valor del número.

Podemos aclarar un poco más este concepto con el siguiente ejemplo:

Para cambiar +7 a -7:

+7 = 00000111

El complemento del número es = 11111000

Por lo tanto, 11111001 es el equivalente binario de -7

(Este sistema funciona igualmente para convertir el signo menos en más. Habrá notado que usando este sistema se obtiene el bit de signo correctamente. Los números de 16 bits se calculan de la misma forma - el BASIC del Amstrad guarda las variables enteras como números de 16 bits en complemento a dos. Como con los números de 8 bits, el rango se mantiene igual, aunque el máximo número se reduce a la mitad, 32767).

## Lenguaje Ensamblador

Esperamos que esté todavía con nosotros. Desgraciadamente no nos queda más remedio que seguir estos pasos antes de pasar a las cosas realmente interesantes. El **lenguaje Ensamblador** ofrece una alternativa al uso de números decimales y hexadecimales para representar las rutinas e instrucciones de código máquina. Mientras que una instrucción de código máquina se muestra como un número decimal o hex, en lenguaje ensamblador se muestra como un nemotécnico, un nombre abreviado. Una lista de números es bastante difícil de seguir, pero una lista de nemotécnicos como RET (abreviatura de RETURN, retorno), JP (abreviatura de JUMP, salto) y LD

(abreviatura de LOAD, cargar) es más fácil de comprender.

En las últimas páginas de este libro hay un apéndice que contiene todas las instrucciones de Z80. Se han puesto en los dos formatos, nemotécnico y hexadecimal. Para poder introducir más fácilmente los programas de este libro, se han incluido también en dos formatos, en BASIC y en lenguaje ensamblador. El listado del BASIC es el que le resultará más fácil de usar en este momento, ya que se puede teclear directamente en el Amstrad y ejecutarlo sin más. El listado en lenguaje ensamblador, necesita un programa especial (llamado **ensamblador** o **assembler**) para convertir los nemotécnicos en código que pueda entender el Amstrad.

### Comprando un Ensamblador

Existen multitud de ensambladores en el mercado. Aunque ofrecen gran variedad de facilidades, todos realizan la misma tarea esencial, convertir los listados en lenguaje ensamblador en el código apropiado. No se debe subestimar el valor e importancia de un ensamblador. Puede que encuentre extremadamente difícil escribir sus primeros programas en código máquina en decimal o en hex, o convertir a mano los nemotécnicos en hex sin un ensamblador. A medida que usted progresa en el mundo del código máquina, se convencerá de que es mucho más fácil escribir las rutinas largas en un ensamblador. Si escoge el correcto, también habrá escogido un grupo de facilidades sumamente útiles para ayudarle en la escritura, depuración y almacenamiento de sus rutinas en código máquina.

De momento olvidemos lo dicho hasta ahora, no necesita un ensamblador para usar el material incluido en este libro. Si va a introducirse seriamente en el mundo del código máquina, piénselo, pues debe tomar en consideración la necesidad de comprar uno muy pronto.

Si está decidido a aprender y usar el código máquina, deberá comprar un ensamblador con todas las facilidades disponibles, preferentemente uno que venga con un programa monitor (un programa que le permite moverse dentro de la memoria). Entre los mejores disponibles está **The Code Machine** por **Picturesque Software** que viene a costar en el Reino Unido unas 19.95£, que es una buena inversión. No vamos a entretenernos en discutir todas sus facilidades, baste saber que es el que se ha usado para escribir este libro. Picturesque está en 6, Corkscrew Hill, West Wickham, Kent BR4 9BB, Reino Unido.

Si prefiere algo más económico, puede echarle una ojeada al **Curso de Lenguaje Ensamblador del Amstrad** proporcionado por **Glentop Publishers** en **Barnet, Herts.** El ensamblador que incluye es bastante bueno, y además viene con un curso el código máquina, todo por 12.95£.

Si usted es un programador pobre, puede intentar conseguir el número 7 (Julio de 1985) de la revista '**Computing With The Amstrad**', que incluye un interesante listado de un ensamblador. Los ejemplares atrasados se pueden obtener de **Freeport, Europa House, 68 Chester Road, Hazel Grove, Stockport SK7 5NY, Reino Unido.** Cada número atrasado cuesta alrededor de 1.25£, y una cinta que contiene todos los programas de la revista cuesta unas 3.75£.

En nuestro país podemos obtener un buen ensamblador como es el **DEVPAC** de **HISOFT**, que distribuye **Indescomp**, por unas 6500R. Se puede encontrar también en disco para el **CPC6128** en los comercios especializados como **Sinclair Store, Peek and Poke** o en la sección de microinformática de los grandes almacenes. Este paquete incluye también un estupendo desensamblador que nos ayudará a analizar los programas de los que no dispongamos listado.

## Directrices

En nuestro país podemos obtener uno de los mejores ensambladores, **DEVPAC** de **HISOFT**, que distribuye **Indescomp** por unas 6500R. Lo puede encontrar en los comercios especializados como **MICROTODD**, en la calle **Orense 3** de **Madrid** o en cualquiera de las tiendas de **Sinclair Store** o de **Peek and Poke** o en la sección de microinformática de los grandes almacenes. Este paquete incluye también un estupendo desensamblador que nos ayudará a analizar los programas de los que no dispongamos listado.

- ORG** - define la dirección de la primera parte del código ensamblado.
- END** - marca el final del programa.
- EQU** - asigna un valor a un nombre de etiqueta.
- DEFL** - le permite dar un valor a una etiqueta, tantas veces como lo necesite.

- DEFB - asigna un valor a un octeto, en la dirección actual de ensamblaje.
- DEFW - asigna un valor a los dos octetos siguientes a la dirección actual de ensamblaje (es la versión de doble octeto de DEFB).
- DEFS - crea un número de octetos en blanco desde la dirección actual de ensamblaje.
- DEFM - permite introducir mensajes en forma de cadena de caracteres.
- PRNT - permite activar e desactivar la impresión mientras se ejecuta el ensamblaje del código. La sintaxis exacta de este comando varía dependiendo ensamblador.
- ENT - Define el punto de entrada del programa.

No se preocupe si no entiende totalmente estas directrices. Cuando comience a usar el ensamblador o a estudiar los listados en lenguaje ensamblador, podrá volver a leer estas páginas y le resultará más sencillo de aprender el significado de cada una.

## CAPITULO TRES

# Usando el Código Máquina en el Amstrad

En este capítulo veremos el 'hardware' del Amstrad, cómo preparar la máquina para un programa en código máquina y cómo salvar y cargar el programa.

### El Hardware

Echemos una breve ojeada a las diversas partes que componen el ordenador Amstrad. Estas partes se pueden dividir en entrada, salida y proceso. La entrada, obviamente, incluye el teclado, pero también se puede referir a un<sup>a</sup> palanca de juego o a un 'ratón'.

La salida incluye el monitor y cualquier interfaz que permita mandar datos a una impresora o a cualquier otro dispositivo externo.

Las partes del Amstrad que se encargan del proceso interno son la más importantes desde el punto de vista del código máquina. Como ya mencionamos en el primer capítulo, el Amstrad está basado en la CPU Z80. La CPU es la unidad 'pensante' del ordenador, que se encarga de coordinar el trabajo de las demás partes. La memoria es, por supuesto, la más importante de ellas.

La memoria es de dos tipos básicos, ROM y RAM. ROM es la abreviatura de 'Read Only Memory' (Memoria de Solo Lectura) y no puede ser alterada por medio del código máquina. Esta invulnerabilidad es necesaria para mantener información importante - el sistema operativo y el intérprete BASIC. Podemos investigar los valores almacenados en la ROM, que pueden ser muy útiles como veremos más adelante.

RAM es la abreviatura de 'Random Access Memory' (Memoria de Acceso Aleatorio) y es el tipo de memoria que se usa para almacenar los programas. No solo se puede leer, también se

puede alterar.

La memoria del Amstrad consta de 65536 posiciones de RAM y 32768 de ROM. Puede que le sean más familiares los términos 32K y 64K (donde 1K son 1024 octetos). Cada posición de memoria es capaz de almacenar un número de 8 bits que, como vimos en el capítulo anterior, puede representar un número decimal dentro del rango 0 a 255. También hemos visto que a esta posición de memoria de 8 bits se le llama octeto.

Una vez aprendido esto, vayamos a ver las otras partes del Amstrad.

## El Sonido

El chip PSG es el Generador de Sonido Programable del Amstrad. Es un AY-3-8912, el mismo que se puede encontrar en otros ordenadores. El PPI es el Interfaz Periférico Programable y es la unión entre la CPU y el PSG, la puerta de impresora, la pantalla y el cassette y/o la unidad de disco. El CRTC también tiene que ver con la pantalla (su nombre completo es Controlador de Tubo de Rayos Catódicos).

La última pieza que vamos a ver del hardware del Amstrad es el 'Gate Array' (se puede traducir por 'Matriz de puertas'). Es un dispositivo propio del Amstrad, que ayuda a solapar la ROM y la RAM así como a la generación de la imagen de la pantalla. El solapamiento de la ROM y la RAM se tratará, junto con el mapa de memoria, un poco más adelante.

## Escribiendo Programas en Código Máquina

Saliendo ya del hardware del Amstrad, vamos a ver los procedimientos involucrados en la escritura de código máquina.

Primeramente, debemos reservar un área de memoria para colocar el programa en código máquina. Aunque su Amstrad tiene 64K (o más, si tiene una máquina 6128), no toda la memoria está disponible para sus programas. Si teclea PRINT FRE(0), verá que tiene a su disposición unos 42K de memoria. Sin embargo, solo va a necesitar una fracción para la mayoría de los programas en código máquina.

Esta memoria disponible para el usuario tiene una frontera sobre la que está situado el código que controla el ordena-

dor. Este límite superior se llama HIMEM y es una variable que puede imprimirse y alterarse. Si enciende su Amstrad e introduce PRINT HIMEM, obtendrá un resultado de unos 43K, dependiendo del modelo de Amstrad que posea. El valor de HIMEM se puede alterar poniendo en su lugar un valor inferior al original, por medio del comando MEMORY. MEMORY 40000 baja HIMEM a la dirección 40000 dejando libre una parte de la RAM para que la use con su código máquina o rutina. En el resto del libro hacemos esto para dejar espacio a nuestros programas. El área que hay sobre HIMEM no se ve afectada por el comando NEW.

## SYMBOL AFTER

Hay otro punto a tener en cuenta en la reserva de memoria. El comando SYMBOL AFTER reserva memoria para los gráficos definidos por el usuario (UDG). Esta memoria se reserva inmediatamente debajo de HIMEM y permanece aunque se altere el HIMEM. Si, por ejemplo, HIMEM estaba al principio en 40000 y la alteramos a 36000, la memoria para los UDGs estarán por encima de HIMEM y no le será posible acceder a ellos al ordenador. Esto nos puede causar algunos problemas.

Si mira a una de las muchas rutinas de la segunda parte del libro, verá que la primera línea del programa suele ser:

```
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
```

Esta línea libera toda la memoria reservada para los UDGs, altera HIMEM bajándolo a 39999 (lo que significa que la primera dirección disponible para la rutina de código máquina es 40000, uno por encima) y después reserva memoria para 16 UDGs, empezando desde la nueva posición de HIMEM hacia abajo. (Para una explicación más detallada de SYMBOL AFTER y de los gráficos definidos por el usuario, diríjase al manual del Amstrad).

Una vez que ha introducido su programa en código máquina en la memoria, sávelo antes de ejecutarlo. No de RUN al programa sin haber salvado una copia del mismo a la cinta o al disco. A diferencia de los programas en BASIC, con sus mensajes de error, si el programa en código máquina no funciona adecuadamente, lo más probable es que le falle el ordenador obligándole a apagarlo y volverlo a encender de nuevo y perdiendo un trabajo que le puede haber llevado largo tiempo.

## Salvando el Programa

Para salvar un programa en código máquina, no tenemos más que añadir algunos parámetros extras al comando SAVE normal. El nuevo formato es:

```
SAVE "NOMBREPRG",B,dir. principio,longitud,punto entr.
```

La dirección de comienzo es la dirección de memoria desde la que se salvará el programa, mientras que el punto de entrada es la dirección desde la que se ejecuta el programa.

La longitud del programa es el número de octetos que ocupa la rutina. Se puede calcular con la siguiente fórmula: Dirección final de la Rutina - Dirección de comienzo de la Rutina +1.

## Carga del Programa

La carga de un programa en código máquina es muy simple. Solamente debe teclear LOAD "NOMBREPGM" o incluso LOAD "". Se puede reubicar el programa especificando una nueva dirección de comienzo, ej. LOAD "PANTALLA",30000. Sin embargo, debe tener en cuenta que los programas pueden contener instrucciones que sean dependientes de la dirección original de comienzo (esto se ve más claro en los siguientes capítulos, particularmente en el que se trata las instrucciones de salto (JP)).

# CAPITULO CUATRO

## Su Primer Programa en Código Máquina

Ahora ya tiene una idea acerca de la preparación para la rutina en código máquina. Así mismo, conoce lo que es el lenguaje ensamblador. Ya ha llegado el momento de usar estos conocimientos en su primer programa en código máquina.

Hace poco, leí en una revista dedicada al Amstrad una carta de un lector desesperado porque no podía hacer funcionar su código máquina. Tecleaba los números hexadecimales, como C9, y nemotécnicos, como RET y LD, directamente en la máquina. Por supuesto, todo lo que consiguió fueron montones de mensajes de 'syntax error'.

Nosotros ya sabemos que este método no es el correcto ya que (a) para usar nemotécnicos necesitamos un ensamblador y (b) el código debe ser introducido en la memoria del Amstrad mediante POKEs.

### Un Cargador Hex

Para realizar este trabajo necesita un pequeño programa conocido como cargador. Las rutinas se suelen presentar con un cargador incorporado, como todas las rutinas de la segunda mitad de este libro. Para las rutinas muy cortas, como las que vamos a tratar en algunos de los siguientes capítulos, es mejor usar un programa cargador especial. Este que presentamos aquí es un cargador muy simple (hemos visto algunos que añaden muchas divertidas facilidades que aumentan el número de páginas del programa, para realizar la misma función que un programa de diez líneas).

Teclee el siguiente programa y sálvelo a disco o a cinta. Lo va a necesitar unas cuantas veces antes de terminar con el libro.

```

1                CARGADOR HEXADECIMAL
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 n=40000
30 INPUT a$
40 IF LEN(a$)/2<>INT(LEN(a$)/2) THEN PRINT "*ERROR* vuelva
    a introducir la última línea":GOTO 30
50 b$=LEFT$(a$,2)
60 POKE n,VAL("&" +b$)
70 n=n+1
80 a$=RIGHT$(a$,LEN(a$)-2)
90 IF a$="" GOTO 30 ELSE 50

```

Veamos lo que hace realmente el cargador hexadecimal. La línea 10 prepara la reserva de memoria (como se explicó en un capítulo anterior). La línea 30 acepta una cadena de dígitos hex y comprueba que se le ha dado un número caracteres par, ej. si la cadena tiene nueve caracteres, entonces la cadena debe tener un error en la entrada de datos.

La línea 50 toma los dos primeros caracteres de la cadena y hace un POKE de ellos en la memoria del Amstard, comenzando en la dirección 40000. La línea 80 elimina esos dos caracteres de la cadena y, si quedan más dígitos vuelve a la línea 50 donde se repite el proceso completo. Si no hay más dígitos, el programa vuelve y pide otros datos. Cuando haya terminado la entrada de datos, pulse ESC dos veces para cortar la ejecución del programa.

## El Programa

Ahora veamos nuestro magnífico y potente programa en código máquina. Se lo ofrecemos en forma de nemotécnico y en hexadecimal ...¿está preparado?...

```
RET      C9
```

No hay mucho que ver ¿verdad?. Estas instrucciones ya las hemos visto anteriormente. Es el comando RETURN que devuelve control desde una subrutina y, si no hay subrutinas, vuelve al BASIC. Por lo tanto está admirablemente indicado para

nuestros propósitos. Si funciona conseguiremos volver al BASIC sin problemas, y si no funciona no habremos perdido más que un minuto tecleándolo. Asegúrese de salvar el Cargador Hexadecimal a cinta antes de ejecutarlo.

Ejecute el cargador hexadecimal, introduzca el par de dígitos hex y vuelva al BASIC. La rutina ya está introducida en la memoria de su Amstrad. Para arrancarla necesita usar el comando BASIC CALL. Este comando debe ir seguido por la dirección de memoria en la que comienza la rutina. En este caso, como la mayoría de las rutinas de este libro, comienzan en la dirección 40000. CALL 40000 ejecutará el programa y volverá al BASIC sin ningún problema. Si tiene problemas, apague su máquina, enciéndala de nuevo, cargue el cargador hexadecimal y compruebe el listado del programa con mucho cuidado. Después, ejecútelo y asegúrese de que introduce C9, el valor hex de RET.

CALL es un comando muy potente en el Amstrad. Lo comprobará cuando lea un poco más adelante el capítulo que trata de cómo pasar parámetros a, y desde los programas en código máquina.

## Variables

Los programas en BASIC no pueden sobrevivir sin el uso de variables. El código máquina tiene sus propias variables, conocidas como registros. Los registros están muy limitados si los comparamos con la libertad con que se usan las variables en BASIC.

Hay unos cuantos registros especializados, pero por el momento solo consideraremos los de propósito general. Hay seis de ellos: B, C, D, E, H y L. Cada registro es similar a una posición de memoria. Solo pueden contener un número entre cero y 255. Para aumentar esta capacidad, se emparejan los registros, obteniendo BC, DE y HL. Así son capaces de contener números de 16 bits en el rango entre cero y 65535.

Debemos usar estos registros en las diversas instrucciones de código máquina. La más común de ellas es LD, que es la abreviatura de LOAD. Es el equivalente al LET del BASIC. Por lo tanto, LET B=10 o B=10 tiene su equivalente en código máquina con LD B,10 (carga B con 10). Esta forma de cargar se conoce como **Direccionamiento Inmediato**.

Es bastante difícil escribir una rutina en código máquina que no use la instrucción LD. Aparece en varias formas. Como

ya hemos visto, se puede cargar un registro con un valor numérico constante. También se puede hacer LD a un registro con el valor de otro registro, como en LD B,C. Recuerde que siempre se carga el registro que aparece en primer lugar con el contenido del segundo.

#### Acumulador

Antes de seguir con las versiones disponibles de la instrucción LD, debemos conocer al más especializado de los registros. Al registro 'A' se le conoce como registro acumulador. Veremos sus facilidades en el capítulo seis, cuando tratemos la aritmética simple.

Hay un número de LDs específicos al registro A. Es el único registro de 8 bits capaz de cargarse directamente desde una posición de memoria. Se puede usar de la misma forma que los otros registros generales, en la forma LD A,C o LD A,190.

Veamos estos LDs especiales. LD A,(nn) carga A con el contenido de la dirección nn de memoria. Los paréntesis significan 'con el contenido de'. Es posible hacer el proceso a la inversa y cargar una posición específica de memoria con el valor del registro A. LD (nn),A lo hace, nn sigue siendo una posición de memoria.

Para demostrar todo lo expuesto, veamos una rutina de ejemplo. Siga para esta rutina el mismo procedimiento que para la anterior, usando el cargador hexadecimal.

Ensamblador	Hexadecimal
LD A,(41000)	3A 28 A0
LD (41001),A	32 29 A0
RET	C9

Haga una CALL 40000 para que la rutina cargue en 41001 el valor que hay en 41000, pero ejecutando una simple rutina en código máquina. Para comprobar que ha funcionado, teclee:

```
PRINT PEEK(41000);" ";PEEK(41001)
```

... y los números deben ser iguales.

Observe lo fácil que habría sido caer en una trampa en

este punto. Muchas rutinas de este libro comienzan en la dirección 40000. Habría sido muy fácil usar 40000 en esta, en vez de 41000. Sin embargo, si lo hubiéramos hecho, habríamos colocado un nuevo valor dentro de la rutina, con 40001 conteniendo la misma instrucción que la dirección 40000, LD A.

Esto debe ser evitado en cualquier rutina en código máquina. En esta no es muy importante el fallo, pero no es muy aconsejable permitir que una rutina se modifique a sí misma cuando es importante o compleja..

Veamos otro par de registros y su uso con la instrucción LD. IX e IY se llaman **registros índice**. Son de 16 bits y se usan especialmente para pasar parámetros desde el BASIC al código máquina y viceversa, como se verá en el próximo capítulo. LD IX, (dir) carga el registro IX con el contenido de la dirección.

F es el registro de señalizadores, llamado algunas veces **registro de estado**. De vez en cuando cohabita con el registro A de forma que nadie se da cuenta de ello.

Cada uno de los bits de F tiene su propósito específico.

Son:

- Bit 7: Señalizador de signo, o S.
- Bit 6: Señalizador de cero, o Z.
- Bit 5: No se usa.
- Bit 4: Señalizador de medio acarreo, o H.
- Bit 3: No se usa.
- Bit 2: Señalizador de paridad/rebasamiento, o PV, o P.
- Bit 1: El señalizador de sustracción o de negación.
- Bit 0: Señalizador de acarreo, o C. (No confundirlo con el registro C).

El señalizador de **medio-acarreo** lo ponen las instrucciones aritméticas si hay un acarreo desde el bit tres al bit cuatro o, en el caso de pares de registros, desde el bit once al bit doce.

El señalizador de **sustracción** es puesto por cualquier instrucción que implique una sustracción, y restaurado por cualquier instrucción que implique una suma.

No hay instrucciones que alteren directamente el valor de F. Sin embargo, a F se le puede introducir un valor xx, usando LD C,xx/PUSH BC/POP AF. De forma similar, se puede leer el estado de los señalizadores H y N usando PUSH AF/POP BC y examinando después los bits en el registro C.

No se preocupe si esto le parece incomprensible de momento. Ya lo verá claro durante el curso (se lo prometemos), y entonces podrá volver a releer esta sección comprendiéndola completamente.

# CAPITULO CINCO

## Pasando Parámetros

Ya sabemos que el comando CALL del BASIC se usa para acceder a una rutina en código máquina. CALL puede ir acompañado por algo más que la dirección de la rutina a la que se llama. Esta sección trata del uso del comando CALL para pasar parámetros tanto desde el BASIC al código máquina, como en sentido contrario.

Hay varios tipos de parámetros que se pueden pasar al, y desde, el código máquina. Veamos los tres tipos más corrientes.

El primero es un número entero que puede ser un número o una variable numérica entera. El valor debe estar en el rango de un número en complemento a dos de 16 bits, de forma que 31092, 9, 220 y F% (donde F% es igual a 1000) están permitidos.

Quando se llama a la rutina en código máquina, el registro A contiene el número de parámetros que siguen al comando CALL. Esto puede ser más o menos útil para usted. Cada parámetro es almacenado usando el registro índice IX (que hemos visto parcialmente en el capítulo anterior y que veremos más adelante en el diccionario de código máquina). El valor del parámetro de dos octetos se almacena con el octeto más bajo primero y el más alto a continuación.

El registro IX contiene la dirección del octeto más bajo del último parámetro. El octeto alto del último parámetro y los octetos de los demás, si los hay, se almacenan desde la dirección IX+1 en adelante. Por lo tanto, si se introduce una CALL 40000,258, (IX+0) contendrá el octeto bajo 2 y (IX+1) contendrá el octeto alto 1. (Si necesita refrescar la memoria: los números de 16 bits se calculan  $256 * \text{octeto alto} + \text{octeto bajo}$ , en este caso  $256 * 1 + 2 = 258$ ).

Supongamos que queremos hacer una CALL a 30000 para cargar el parámetro en un registro normal de dos octetos, como el HL, y que el parámetro es la variable B%. Su rutina en código máquina debe comenzar con LD L,(IX+0) seguida por LD H,(IX+1). Si fuera el tercero desde el último parámetro, debería ser LD L,(IX+4) seguido por LD H,(IX+5). En otras

palabras, el octeto bajo va siempre el primero y para todos los parámetros, hay dos octetos apuntados por el registro IX más un desplazamiento.

El segundo tipo de parámetro es también una variable entera pero, a diferencia del anterior, se puede devolver desde el código máquina al BASIC. Para conseguir esto en la CALL, se debe añadir un carácter '@' delante del nombre de la variable. (Naturalmente, debe ser una variable y no un número). CALL 40000,@B% hará que (IX+0) e (IX+1) contengan la dirección del valor de B%.

Por lo tanto, si quiere alterar el valor de B%, debe encontrar la dirección donde está almacenada examinando (IX+0) e (IX+1). Después debe modificar las dos posiciones de memoria que contienen el valor de B% antes de volver al BASIC.

Por ejemplo, si quiere alterar el valor de B% a 258, debe cargar las posiciones de memoria dadas por  $(IX+0)+256*(IX+1)$  con 2 (el octeto inferior) y las posiciones de memoria dadas por  $(IX+0)+256*(IX+1)+1$  con 1 (el octeto superior).

### Cambio del Octeto Inferior

Echemos una breve ojeada a otro pequeño programa en código máquina. Este acepta la CALL con un parámetro y cambia lo que hay en el octeto inferior del parámetro, por un 6. Si quiere puede usar el programa cargador hexadecimal para probar la rutina.

Ensamblador	Hexadecimal
LD L,(IX+0)	DD 6E 00
LD H,(IX+1)	DD 66 01
LD (HL),6	36 06
RET	C9

La primera línea toma el octeto inferior de la dirección del parámetro y lo pone en el registro L. La segunda línea toma el octeto superior de la dirección del parámetro y lo pone en el registro H. Con la dirección del parámetro cargada en el par de registros HL podemos alterar el valor de la variable, alterando la memoria apuntada por ese par de registros. En la tercera línea, lo cambiamos a 6. (En un

momento le mostraremos un par de cosas más que se pueden hacer con él). En la cuarta línea volvemos al BASIC.

Para usar la rutina teclee: `B%=0:CALL 40000,@B%`. Si imprimimos `B%` después de ejecutar la rutina, veremos que `B%` tiene ahora el valor 6. Puede jugar experimentando con el código máquina, altere el número hexadecimal de la tercera línea (06 en el listado) por otro valor y ejecute la rutina de nuevo. `B%` tendrá ahora el nuevo valor. Mirando los códigos de operación que hay al final del libro, altere la instrucción `LD (HL),nn` a `LD (HL),A`. Esto nos devolverá el número de parámetros disponibles.

### Manejando cadenas

El tercero y último de los tipos de parámetros que estamos viendo, es de cadena. Este funciona de una forma ligeramente diferente de la información numérica. Para empezar, solo puede usar variables de cadena; no puede poner `CALL 40000,"AMSTRAD"`. Aquí también debe preceder al nombre de la variable el carácter '@'

El registro IX contiene la dirección del parámetro, como en los casos numéricos, pero esta vez no es la dirección donde reside en memoria, sino la dirección de otra posición de memoria - el comienzo del **bloque de descripción de cadena**. Este bloque contiene la dirección donde comienza la cadena y su longitud.

El primer octeto del bloque de descripción de cadena es la longitud de la cadena, y tiene un valor entre 0 y 255. Los dos octetos siguientes forman la dirección de memoria donde comienza la cadena, y está en la forma habitual de el octeto inferior primero y el superior a continuación. Esta es la dirección del primer carácter de la cadena. La cadena debe estar definida de antemano, aunque solo sea como `A$=""`.

### Una Rutina de Tratamiento de cadenas

Esta pequeña rutina toma la cadena y nos dice su longitud. Para hacer esto necesitamos un parámetro adicional, una variable entera que contenga el valor devuelto.

Ensamblador	Hexadecimal
LD L, (IX+0)	DD 6E 00
LD H, (IX+1)	DD 66 01
LD A, (HL)	7E
LD C, (IX+2)	DD 4E 02
LD B, (IX+3)	DD 46 03
LD (BC), A	02
RET	C9

Ejecute ahora una CALL 40000,@V%,@A\$ con A\$ y V% definidas de antemano, al terminar tendrá en V% el valor de la longitud de la cadena A\$.

# CAPITULO SEIS

## Aritmética Simple

Ahora vamos a realizar algunas operaciones de aritmética simple en código máquina. Para hacerlo, necesitamos algunas instrucciones nuevas.

Las operaciones aritméticas más simples en código máquina son INC (incrementar) y DEC (decrementar). INC suma uno al registro especificado, que puede ser un registro de 8 bits, un par de registros de 16 bits o un registro de índice. Además, estas instrucciones pueden alterar el valor de una posición de memoria como en INC (HL) - esto se conoce como **direccionamiento indirecto**.

INC es un comando de un solo octeto. INC A sumará 1 al valor contenido en el registro A, mientras que INC IX hará lo mismo con el contenido del registro IX. Si está incrementando un registro de 8 bits y el valor del mismo pasa de 255, entonces el valor se hará cero. Si se incrementa un par de registros de 16 bits, cuando su valor pasa de 65535, también se vuelve cero.

DEC funciona de la misma forma que INC y todas las formaciones de instrucciones que son válidas para INC, también lo son para DEC. Estas son:

INC o DEC r            (r es un registro de 8 bits)  
INC o DEC rr           (rr es un par de registros de  
16 bits)  
  
INC o DEC IX  
INC o DEC IY  
INC o DEC (HL)  
INC o DEC (IX+d)  
INC o DEC (IY+d)

Si decrementa un registro de 8 bits cuando su valor es cero, obtendrá el valor 255 - el procedimiento opuesto al de

INC.

Mientras que las instrucciones INC y DEC de 8 bits afectan a la mayoría de los señalizadores, las de registros de 16 bits (cuando se usan con INC o DEC) no afectan a ninguno de ellos. Para las instrucciones de 8 bits, se pone el **señalizador de signo** (se hace igual a 1) si el bit 7 del resultado es 1. El **señalizador de cero** se pone si el resultado es cero.

El **medio acarreo** se pone si hay un acarreo desde el bit 4 del resultado. El **señalizador de rebasamiento** se pone si se altera el bit 7 del resultado. El **señalizador de sustracción** se pone si la última operación realizada ha sido una instrucción de resta, como DEC. Observe que el señalizador de acarreo permanece sin alterar.

## Suma y Resta

ADD y SUB son dos ejemplos de nemotécnicos fáciles de entender. Es bastante obvio que son abreviaturas de las palabras inglesas 'addition' (suma) y 'subtraction' (resta).

ADD y SUB funcionan solamente con un registro de 8 bits, el A, y con el par de registros HL y los registros índice, IX e IY, de 16 bits. Aparte de sus funciones, que difieren, la sintaxis es casi la misma.

ADD y SUB se pueden comparar con ciertos comandos del BASIC. Así nos damos una idea más clara de como se pueden usar en una rutina:

LET A=A+9	ADD A,9
LET A=A+C	ADD A,C
LET A=A-C	SUB A,C
LET A=A-178	SUB A,178

Solamente el registro A puede trabajar directamente con estos comandos. ADD B,9 o ADD C,D no están permitidos. Por lo tanto, si quiere realizar operaciones aritméticas con otros registros, debe usar un procedimiento similar a este, que resta 8 del registro E.

LD A,E (Carga A con el contenido de E)  
SUB 8 (resta 8 de A)  
LD E,A (carga E con A, el nuevo valor de E)

Siempre que tenga que hacer una suma o una resta usando un registro de 8 bits, debe asegurarse de que el resultado de la operación cabrá dentro de un número de 8 bits.

Para las operaciones que den un resultado mayor, se puede usar la suma y resta de 16 bits. La mayoría de las operaciones de aritmética de 16 bits se realizan con el par de registros HL. A estos se le pueden sumar y restar otros pares de registros, como vemos aquí:

ADD HL,BC  
ADD HL,DE  
ADD HL,HL  
ADD HL,SP

Tenga en cuenta que el segundo par de registro no es modificado por esta instrucción. Las operaciones aritméticas con los registro índice funcionan así:

ADD IX,BC  
ADD IX,DE  
ADD IX,IX  
ADD IX,SP

IY se maneja de la misma forma. No hay ninguna instrucción que pueda sumar directamente un número a HL. Una forma de hacer esto sería cargar el número deseado en DE y usar después ADD HL,DE, pero así perderíamos el contenido del par de registros DE. El señalizador de acarreo también resulta afectado por estas operaciones. Así, si el resultado obtenido es mayor de 65535, se pone el señalizador de acarreo a 1, en caso contrario se pone a 0.

## Punto y Acarreo

Hay otras dos instrucciones que tienen que ver con la suma y la resta. Funcionan de la misma manera que ADD y SUB, excepto en que hacen uso del **señalizador de acarreo**. Estas son ADC (suma con acarreo) y SBC (resta con acarreo). En el caso de la aritmética de 8 bits, solo funcionan con el registro A como las anteriores. En aritmética de 16 bits funcionan solamente con el par de registros HL, pero no con los registros índice IX e IY. ADC y SBC incluyen el señalizador de acarreo en el cálculo, por lo que su equivalente en BASIC sería:

```
LET A=A+5+acarreo          ADC  A,5
LET A=(A-12)-acarreo      SBC  A,12
```

Estas instrucciones son muy potentes ya que permiten realizar operaciones aritméticas con números de 32 bits. Algunas veces necesitará asegurarse de que el señalizador de acarreo está a cero antes de usarlo. Una forma de hacerlo es usando la instrucción AND A que no altera el valor contenido en A pero pone el señalizador de acarreo a cero, ya que no produce acarreo.

## No hay División

Ya habrá observado que no hay instrucciones de multiplicación ni de división. Si su programa requiere fórmulas complejas y cálculo, puede que sea mejor escribir esas partes en BASIC. Una desventaja del código máquina es su dificultad en manejar funciones matemáticas complejas.

Sin embargo, es posible emular la multiplicación fácilmente, usando la instrucción ADD. El registro A contendrá el valor original, y la respuesta una vez que se haya ejecutado la multiplicación.

La multiplicación por 2 es muy simple: ADD A,A

En efecto, cualquier número dentro del rango de 8 bits y que sea miembro de la familia, como  $2 \times 2 = 4 \times 2 = 8 \times 2 = 16 \times 2 = 32$ , puede usarse fácilmente como multiplicador.

```

ADD  A,A
ADD  A,A = multiplicación por 4 y

ADD  A,A
ADD  A,A
ADD  A,A = multiplicación por 8

```

Puede continuar así de forma que cuatro instrucciones ADD sirven para multiplicar por 16, cinco por 32 y así sucesivamente.

Otros multiplicadores no son tan fáciles de emular, pero se puede hacer usando la instrucción LD y en algunos casos SUB. Aún así, se pueden realizar con muy pocas instrucciones y pueden resultar interesantes para usarlas en sus rutinas en el futuro. He aquí unos ejemplos:

```

LD   B,A
ADD  A,A      =      x 3
ADD  A,B

```

```

LD   B,A
ADD  A,A
ADD  A,A      =      x 7
ADD  A,A
SUB  A,B

```

Observe la últimas instrucciones de la rutina de multiplicación por siete. Si mira la tabla de Códigos de Operación, verá que no están en ella. Lo que sucede es que SUB B es exactamente lo mismo que SUB A,B. Puede eliminar la A al escribirlo, ya que estas instrucciones solamente funcionan sobre el registro A. Esto se aplica a todas las operaciones SUB que parece que aparentemente usan un solo registro.

# CAPITULO SIETE

## Pila y Saltos

### La Pila

Hay un área de memoria RAM que está prevista para almacenar piezas de información que ayudan a la máquina a saber lo que está haciendo. Funciona de la siguiente forma:

La palabra "pila" (stack) es algo que se usa en informática para expresar exactamente lo que significa. Imagine una pila de cajas de cartón. Cada caja representa una posición de memoria, a la que se le asigna una dirección - pero si quiere saber lo que hay dentro de una caja determinada, la única que puede ver fácilmente es la de arriba. Si intenta coger una de enedio, se le caerán encima todas las cajas que hay sobre ella. Y, la única forma de añadir fácilmente una caja a la pila es poniéndola en la parte superior.

Las posiciones de memoria de la pila están situadas de la misma forma. Se pueden poner cosas encima, pero **solamente** encima, y solo se pueden tomar **de la parte superior**.

Hay dos palabras especiales para manejar la pila - una de ellas significa "apilar un nuevo número en lo alto" y la otra "tomar un número de lo alto"; la primera palabra es PUSH, la segunda es POP. Si hace PUSH del número cinco en la pila, y después hace PUSH del número 9ABC, y luego hace PUSH de, digamos, 8000h, el primer número que obtendremos al hacer POP sería 8000h, ya que este es el número que está ahora en lo alto de la pila (ya que fué el último del que hicimos PUSH); el segundo número que obtendríamos con POP sería 9ABC, y el tercero sería el cinco.

La pila se almacena en las direcciones más altas de la memoria del AMSTRAD, para evitar que el programa BASIC 'colisione' con la pila cuando uno de los dos crezca. La pila es realmente muy peculiar, ya que crece de **arriba hacia abajo**. Es más eficiente de esta forma. Por lo tanto recuerde que, la pila, o la **pila de la máquina** como se le suele llamar, es como una pila de cajas de cartón amontonadas en el suelo de una tienda, excepto que, desafiando las leyes de Newton, ha decidido apoyarse en el techo y crecer hacia abajo. La parte

alta - la única parte de la que puede tomar cosas - está hacia abajo.

La pila es sumamente importante en un ordenador, de forma que hay un registro especial que solamente se usa para almacenar la posición de lo **alto** de la pila (la parte con la dirección menor - la que podemos tomar). El registro se llama SP, que es la abreviatura de Stack Pointer (apuntador de pila). Es realmente un par de registros porque almacena dos octetos separados, pero a diferencia de los otros pares de registros (BC, DE y HL) **no podemos** separarlo en dos mitades - están pegados sólidamente.

Aquí es donde trabajan las instrucciones PUSH y POP. Lo vamos a explicar en hex porque así nos resultará más fácil. Supongamos que HL contiene el valor ABCD. Esto significa que H contiene el valor AB y L el valor CD. Ahora, la instrucción PUSH HL almacenará el número ABCD en lo alto de la pila. Esto lo hace almacenando primero la parte **alta** (AB), y después la parte **baja** (CD). Después alterará el valor de SP ya que se han añadido dos octetos más a la pila, y la posición de la parte alta habrá bajado dos posiciones.

Desafortunadamente no es posible almacenar registros simples en la pila. Solamente se puede hacer PUSH de **pares** de registros, así se puede hacer PUSH de BC, pero no de B solamente. Debemos tener en cuenta que PUSH BC no altera el valor de BC, simplemente copia su contenido sin cambiarlo. Esto, por supuesto, es aplicable a todas las instrucciones PUSH.

La instrucción PUSH se puede emular en BASIC con tres sentencias separadas:

```
PUSH HL           SP=SP-2
                  POKE SP+1,H
                  POKE SP,L
```

POP, por supuesto, funciona al revés. POP HL toma primero el registro L de la pila, y después el registro H. SP cambia también, ya que la parte alta de la pila se mueve.

POP HL

L=PEEK SP

H=PEEK (SP+1)

SP=SP+2

Verifique, usando las sentencias BASIC dadas, que PUSH HL seguida de POP DE es lo mismo que LD D,H seguido de LD E,L.

## PUSH

He aquí los códigos de la instrucción PUSH. Una de ellas requiere una pequeña explicación.

F5	PUSH AF
C5	PUSH BC
D5	PUSH DE
E5	PUSH HL

El par de registros AF, que normalmente no se pueden usar como tal, está construido por los dos registros A y F, de la misma forma que BC está formado por B y C. A es un registro que ya hemos usado, pero F es algo completamente diferente. La F significa Flags (señalizadores). Para comprender el trabajo de F debe mirarlo en forma **binaria**. F puede, por ejemplo, tener el valor 41h, que en binario es 01000001. Cada uno de los dígitos es cero o uno, y cada uno de los diferentes dígitos tiene un significado que corresponde a un señalizador. Uno de estos señalizadores, el de acarreo, está almacenado en el dígito binario más a la derecha de F.

## POP

Los códigos de las instrucciones POP son similares a los usados para PUSH. Estos son:

F1	POP	AF
C1	POP	BC
D1	POP	DE
E1	POP	HL

Una de las utilidades principales de PUSH AF y POP AF es hacer PUSH y POP del valor del registro A. El hecho de que F se introduzca también en la pila puede ser ignorado. PUSH AF almacenará el valor de A hasta que se necesite de nuevo, en cuyo momento se puede recuperar mediante el uso de POP AF. Esto puede ser útil si tiene que usar el registro A para realizar cálculos que no se pueden realizar con otro registro, pero donde el valor de A puede necesitarse en otro punto del programa.

Por ejemplo, para sumar 25 al valor de B sin alterar el valor de A (o cualquier otro registro):

F5	PUSH	AF	
78	LD	A,B	
C619	ADD	A,19h	(=25d)
47	LD	B,A	
F1	POP	AF	

¿Porqué solo se va a alterar el registro B y no cualquier otro? Trate de averiguar lo que hace la rutina anterior, antes de seguir leyendo.

### Alterando el SP

Se puede usar el registro SP casi en la misma forma que usamos BC y DE. Podemos sumarle o restarle, y también se puede cargar. Los códigos hex son:

F9	LD	SP,HL
31????	LD	SP,nn
ED7B????	LD	SP,(nn)

ED73????	LD	(nn),SP
39	ADD	HL,SP
ED7A	ADC	HL,SP
ED72	SBC	HL,SP
33	INC	SP
3B	DEC	SP

Estas instrucciones son muy potentes y útiles. Suponga que quiere intercambiar los valores de D y E sin alterar ninguna otra cosa. La siguiente rutina lo hará.

D5	PUSH	DE
D5	PUSH	DE
33	INC	SP
D1	POP	DE
33	INC	SP

La instrucción INC SP del final es necesaria para restaurar el apuntador de pila a su valor original. Si no lo hiciéramos podríamos conseguir un bonito fallo del ordenador.

SP no es el único registro especializado que se usa. Hay otro registro de dos octetos llamado PC, que significa Program Counter (Contador de Programa). Su trabajo consiste en recordar por donde va ejecutándose el programa. Cada vez que el Amstrad tiene que ejecutar una instrucción, echa una mirada a lo que le dice el PC. Si le dice A004, se irá a ejecutar la instrucción que hay en la posición A004 de memoria. Después se incrementa el valor de PC en el número de octetos que tiene la instrucción, de modo que la siguiente vez irá a ejecutar la siguiente instrucción en secuencia. Por ejemplo, si A004 contiene la instrucción LD A,B, el PC se incrementará para apuntar a A005. Si la instrucción en A005 fuera LD B,2 se incrementaría el PC en dos, una vez que la hubiera ejecutado, ya que LD B,2 es una instrucción de dos octetos. El PC estaría entonces apuntando a A007 donde empezaría la siguiente instrucción.

Si altera el valor de PC, el efecto es como el comando GO TO del BASIC. La única diferencia es que el código máquina no usa números de línea, de forma que ha de hacer GO TO a la dirección correcta en lugar de al número de línea. La

instrucción de código máquina que lo hace se llama JP, que es la abreviatura de Jump (salto). JP A000 significa GO TO dirección A000 y continúa ejecutando el código máquina desde ahí. Por supuesto, todo lo que hace esta instrucción es cargar en PC el valor A000 (pero sin incrementarlo al final de la instrucción) de modo que cree que la siguiente instrucción del programa está en A000. Es bastante más útil para nosotros pensar que funciona como un tipo de GO TO, ya que es para lo que la usamos.

### Tenga Cuidado

Tenga cuidado con el JP. Si crea un bucle infinito en código máquina puede ser muy **duro**. Se habrá hundido con él - no podrá cortarlo jamás, a menos que restaure la máquina o la apague. Un ejemplo de un bucle infinito puede ser:

```

77          A000      LD   (HL),A
23          A001      INC  HL
C300F0     A002      JP   A000

```

Hemos escrito la dirección en la columna de central. Normalmente no se hace así sino que se marcan las líneas importantes con **etiquetas**, o palabras que nos dicen qué líneas hacen cada cosa. Estas etiquetas no aparecen en hex, y solo tienen utilidad cuando se escriben para nuestro propio uso. Si por ejemplo, decidimos llamar START a la primera línea, nuestro bonito y estúpido programa se escribiría así:

```

77          START    LD   (HL),A
23                   INC  HL
C300F0              JP   START

```

Hay otra instrucción, similar a JP, llamada JR, o Jump Relative (Salto Relativo). Significa que salte hacia adelante un número de octetos dado. En muchas formas es mejor que JP, porque tiene solo dos octetos de longitud en lugar de tres, y porque se puede reubicar la rutina completa cambiando el destino de las instrucciones JP. JR 0 no tiene ningún efecto, y por lo tanto hace que se ejecute la siguiente

instrucción, JR 1 hace que salte la siguiente instrucción (suponiendo que ésta tenga un solo octeto de longitud). Para saltar una instrucción de dos octetos, o dos instrucciones de un octeto, necesitará usar JP 2.

También es posible saltar **hacia atrás** usando la instrucción JR, ya que hay una convención que hace que cualquier número hex entre 80h y FFh se tomará como un número negativo (que será realmente 256d menor que el número que representa). Observe que el número menos cinco, por ejemplo, se representa como FB, y por lo tanto es posible usar la instrucción JR -5, pero tenga en cuenta que debido a esta convención no podremos decir JR 129d, por ejemplo, porque 129d es 81h, que será tomado por -127d y hará un salto hacia atrás. Por lo tanto, el rango al que queda limitada esta instrucción es desde -128d a +127d.

### No Hace Nada

JR 0, como ya hemos dicho, no hace absolutamente nada. Continuará ejecutando la siguiente instrucción. Es importante recordar que los saltos relativos se cuentan desde la **siguiente instrucción**. JR 0 significa que ejecute la **siguiente instrucción más cero**. JR 1 significa que ejecute la **siguiente más uno**. Consecuentemente, si dijéramos JR -2 deberíamos contar hacia atrás dos octetos, empezando en cero con la siguiente instrucción. Observará que dos octetos antes está precisamente el comienzo de la instrucción que acabamos de ejecutar, JR -2. Por lo tanto, JR -2 es un bucle infinito sobre la misma instrucción, y no es una instrucción recomendable para usarla en nuestros programas.

El (realmente estúpido) programa del bucle infinito anterior se puede volver a escribir con un octeto menos usando JR en lugar de JP.

```
77          START    LD    (HL),A
23                               INC  HL
18FC        JR      START
```

Observe que he escrito "JR START" en vez de "JR -4". Esto hace al programa mucho más fácil de seguir, ya que todo lo que tenemos que hacer es buscar la etiqueta START para saber hacia donde nos lleva la instrucción JR.

JR y JP son más o menos inútiles por sí mismas sin condiciones añadidas, de la misma forma que el GO TO del BASIC sería poco útil si no se usara en conjunto con la instrucción IF/THEN GO TO. Necesitamos cierto tipo de saltos **condicionados**, de forma que podamos decir IF cierta condición es verdadera THEN salta a la nueva dirección. Sin esta facilidad, JP y JR solamente se podrían usar para crear bucles infinitos. Aunque el código máquina no tiene la misma flexibilidad que el BASIC, nos permite comprobar cuatro condiciones en JR, u ocho condiciones en JP. Estas son (para JR):

18ee	JR	e	Salto relativo de e octetos.
20ee	JR	NZ,ee	IF el último resultado calculado no fué cero THEN Salto relativo de e octetos.
28ee	JR	Z,e	IF el último resultado calculado fué cero THEN Salto relativo de e octetos.
30ee	JR	NC,e	IF ACARREO=0 THEN Salto relativo de e octetos.
38ee	JR	C,e	IF ACARREO=1 THEN Salto relativo de e octetos.

Y para JP:

C3qpp	JP	pq	Salto a la dirección pq.
-------	----	----	--------------------------

C2qapp	JP	NZ,pq	IF el último resultado calculado no fué cero THEN salto a la dirección pq.
CAqapp	JP	Z,pq	IF el último resultado calculado fué cero THEN salto a la dirección pq.
D2qapp	JP	NC,pq	IF ACARREO=0 THEN salto a la dirección pq.
DAqapp	JP	C,pq	IF ACARREO=1 THEN salto a la dirección pq.
E2qapp	JP	PO,pq	Ver más abajo.
EAqapp	JP	PE,pq	Ver más abajo.
F2qapp	JP	P,pq	IF el último resultado calculado fué positivo (+) THEN salto a la dirección pq.
FAqapp	JP	M,pq	IF el último resultado calculado fué negativo (-) THEN salto a la dirección pq.

Ahora, aunque está muy lejos de poder hacer lo que en BASIC sería IF A\$="HOLA" THEN PRINT "ADIOS", pronto verá que aún esta tarea tan tonta también se puede llevar a cabo con el código máquina. Primero tenemos que explicar dos de las instrucciones de la lista anterior - JP PO y JP PE. Como puede suponerse, JP PO significa IF PV=0 THEN salta a la dirección pq, y JP PE significa IF PV=1 THEN salta a la dirección pq. Pero ¿qué es PV exactamente?.

Repuesta: PV es otro señalizador - parecido al de ACARREO.

Solamente puede almacenar uno de dos valores - uno o cero. La P significa Parity (paridad) y la V significa overflow (rebasamiento), ya que el ordenador, como los buenos matemáticos, no puede deletrear correctamente. Su uso es bastante fácil de explicar:

JP PO se puede considerar como JP NV (salto por No-rebasamiento)

JP PE se puede considerar como JP V (salto por rebasamiento)

Técnicamente no puede escribir JP NV o JP V ya que no es una convención estándar, pero es una ayuda para la memoria. Pienso que no importa si usted es convencional o no mientras sepa lo que significa (NV=PO y V=PE).

Ahora, veamos el **rebasamiento**. Si consideramos los números entre 80 y FF como números negativos y entre 00 y 7F como números positivos, pueden suceder algunas cosas extrañas en la aritmética, si tratamos de cruzar la frontera. Por ejemplo, 41h (positivo) más 41h (positivo) será igual a 82h (¿negativo?). A este tipo de equivocación se le llama rebasamiento, por lo tanto JP V saltará si ocurre este tipo de rebasamiento y JP NV saltará si no ocurre. En términos simples, ocurre un rebasamiento si el resultado de cualquier operación aritmética, actuando en números en convención positivo/negativo (también llamada convención de "complemento a dos") da una respuesta con "signo equivocado".

Repitiendo una vez más: JP NV es una forma no estándar de escribir JP PO, y JP V es otra forma de escribir JP PE. Por lo tanto NV=PO y V=PE. Usted debe inventar alguna forma de recordarlo.

Todas estas instrucciones, si se combinan adecuadamente con otras, pueden comprobar cualquier situación concebible. En efecto, solo hay otro tipo de instrucción necesaria para hacer a JP y JR tan potentes como IF/THEN/GO TO - esa instrucción se llama CP, o ComParación.

CP compara el registro A con cualquier otro registro o con cualquier constante numérica. Por ejemplo, podemos tener CP B (Comparar A con B) o CP L (Comparar A con L) o CP 3E (Comparar A con el número 3E). Lo que hacen realmente estas instrucciones se puede comparar con la sentencia BASIC, DUMMY =

A-B.

En otras palabras, se realiza una resta, pero no se tiene en cuenta su resultado, y el valor de A permanece sin alterar. Sin embargo, los señalizadores si cambian y se ponen de acuerdo con el resultado. Si A contiene 05 y B contiene 06, después de la instrucción CP B, saltará con una instrucción JP NZ (ya que 05-06 no da cero), JP C saltará (ya que 05 es menor que 06), JP NV saltará (no hay rebasamiento ya que el resultado de 05-06=FF es negativo y se suponía que lo era) y JP M saltará (ya que FF es negativo). Aunque realmente se efectúa la resta, debemos hacer incapié en que el resultado es desechado, y el valor de A no cambia.

Puede hacer algunos trucos útiles con CP:

IF A=B THEN GOTO ... CP B / JR Z ...

IF A<B THEN GO TO ... CP B / JR C ...

(Esta solo funciona si asumimos que todos los números son positivos)

IF A<B THEN GO TO ... CP B / JP M ...

## CALL

También en código máquina podemos llamar a subrutinas. El equivalente en código máquina del GO SUB del BASIC es la instrucción CALL. CALL pq se utiliza para hacer un GO SUB a la subrutina cuyo punto de entrada está en la dirección pq. El equivalente a la instrucción RETURN es RET. Creo que le resultará familiar. RET tiene un doble propósito - al final de una subrutina significa "retorna desde esta subrutina"; si no hay subrutina de la que retornar, significa "retorna al BASIC". CALL y RET pueden tener condiciones impuestas, como vemos a continuación:

CDqapp	CALL pq	C9	RET
C4qapp	CALL NZ,pq	C0	RET NZ
CCqapp	CALL Z,pq	C8	RET Z
D4qapp	CALL NC,pq	D0	RET NC
DCqapp	CALL C,pq	D8	RET C
E4qapp	CALL PO,pq	E0	RET PO

	"CALL NV,pq"		"RET NV"
ECqapp	CALL PE,pq	E8	RET PE
	"CALL V,pq"		"RET V"
F4qapp	CALL P,pq	F0	RET P
FCqapp	CALL M,pq	F8	RET M

Como habrá imaginado, las instrucciones como RET Z se pueden usar también para volver condicionalmente al BASIC, ej. RET Z es igual a IF cero THEN RETURN al BASIC.

En BASIC no hay pila, por lo tanto no necesitamos preocuparnos de ella. En código máquina, sin embargo, si la hay y por lo tanto **necesitamos** preocuparnos de ella. Hay dos instrucciones además de PUSH y POP, que alteran la pila. Se llaman CALL y RETURN.

CALL pq es equivalente a PUSH "la dirección de la siguiente instrucción a ejecutar", seguido por JP pq.

RET es equivalente a POP DUMMY seguido por "salta a la dirección DUMMY".

Usted debe ser capaz de ver cómo este procedimiento hace que CALL y RET actúen como deben. Aunque es bastante eficiente, hay algunas cosas que debemos ver.

El valor de SP no debe ser alterado durante el curso de una subrutina, ya que CALL y RET tienen que ver con la pila. Usted puede hacer PUSH en la pila tantas veces como quiera durante la subrutina, siempre que haga el mismo número de POPs antes de intentar el retorno. Un truco muy interesante que debe conocer es cómo **alterar** la dirección de retorno de una subrutina. Digamos que queremos poner B000 como dirección de retorno. Veamos cómo se hace:

E1	POP HL
2100E0	LD HL,B000
E5	PUSH HL

La primera instrucción borra de la pila la dirección de retorno original. La segunda y tercera la reemplazan por una nueva dirección de retorno alternativa. Cuando llegue más adelante una instrucción RET, el control "retornará" a la dirección B000. Otro truco útil que debe conocer es cómo a-

segurarse de que sus subrutinas siempre saldrán con la pila "equilibrada". Una forma de hacerlo es almacenar el valor del apuntador de la pila en alguna parte para recogerlo al final.

# CAPITULO OCHO

## Diccionario de Términos de Código Máquina

Este capítulo es una pequeña guía de referencia de lo aprendido en los capítulos anteriores. Este diccionario contiene una breve descripción de todos los Códigos de Operación del Z80, los registros y un tipo de comandos llamado directrices que se encuentran en casi todos los listados de ensamblador. Estos resúmenes le dan la oportunidad de comprobar comandos particulares o facilidades que puede no haber comprendido completamente, así como una fuente de referencias para cuando, en el futuro, trate de usar los comandos o registros menos usados.

El diccionario incluye también detalles de los códigos de operación del Z80 que no han sido explicados. La lectura de la explicación que se proporciona aquí le dará un buen conocimiento de su uso y funcionamiento.

### Registros

Además de los registros A, B, C, D, E, H y L, y los señalizadores de signo, PV y acarreo, hay otro grupo de registros y señalizadores, aunque no todos ellos son útiles. Veamos primero los registros.

IX es un par de registros; sin embargo no se puede dividir en los octetos que lo componen, como el HL. Cualquier instrucción en código máquina que involucre al HL (siempre que no vaya entre paréntesis) puede ser escrito con el IX en vez de con el HL. (Hay tres excepciones a la regla: EX DE,HL ADC HL y SBC HL). El código hex de este tipo de instrucciones es DD seguido por el código hex correspondiente a la misma instrucción con el registro HL. Cualquier instrucción en código máquina que involucre al registro (HL) (entre paréntesis) se puede escribir con (IX+dd) en lugar de hacerlo

con el HL - la dd representa un octeto cualquiera. Por ejemplo, si tenemos una instrucción LD (HL),03 también existe una instrucción LD (IX+2A),03. El octeto de desplazamiento puede ser muy útil. (Hay una excepción a esta regla: JP (HL) solo se puede escribir como JP (IX) - no JP (IX+dd). El código de este tipo de instrucción es DD seguido del código hex de la instrucción correspondiente, pero con el octeto de desplazamiento insertado en el tercer octeto del código hex. Por ejemplo, el código hex para LD (HL),03 es 3603; por lo tanto, el código hex de LD (IX+2A),03 es DD362A03.

IY es otro par de registros. Se usa de la misma forma que el IX, excepto que el código hex de las instrucciones que involucran al registro IY usan el octeto FD en lugar del DD. Aunque la primera letra de los dos registros se la I, no tienen la parte alta común, y son totalmente independientes el uno del otro.

SP es el apuntador de la pila. es un par de registros como el BC o el DE, sin embargo, los dos octetos que lo componen no se pueden separar. SP siempre apunta al dato de más arriba de la pila de la máquina. Si cambia el valor de SP, se crea una nueva pila automáticamente en la dirección especificada. Las direcciones inmediatamente por debajo del valor actual de SP corren el riesgo de ser reescritas sin el consentimiento previo de la rutina de manejo de interrupciones del Z80 (ver DI).

A es un registro del que debe conocer todo. A veces se le llama el acumulador, ya que se puede usar de una o dos formas en las que no se puede usar ningún otro registro. (ej. ADD A,06).

B,C,D,E,H y L. Estos son los registros con los que tiene que estar familiarizado.

El Juego de Alternativo Registros, se compone de los registros A',B',C',D',E',F',H' y L' y son de poco uso, excepto para copiar el contenido de los registros normales por seguridad. Sin embargo, el Locomotive BASIC usa estos registros, por lo que no se recomienda usarlos en el Amstrad.

F es el registro que contiene los señalizadores que hemos visto anteriormente.

## Todas las instrucciones

Por el momento solo hemos visto unas cuantas instrucciones Z80, por lo que suponemos que estará interesado expandir su vocabulario. Aquí le damos una lista detallada de todas las instrucciones disponibles. Se van a tratar por orden alfabético, de forma que pueda usar este capítulo como un pequeño diccionario de instrucciones de código máquina. Por esta razón vamos a volver a ver los que ya hemos estudiado en capítulos anteriores. Puede ser interesante volver a leerlos para que nos sirva de refresco de la memoria.

**ADC** Empezamos con ADC. Aparece en dos formas: ADC A,r y ADC HL,s. La r se usa para especificar que puede ser cualquiera de los registros A, B, C, D, E, H, L, una constante numérica o el contenido de una dirección (HL), (IX+d) o (IY+d). ADC A,r es una instrucción de un solo octeto. Calcula la suma de A más r más el ACARREO y almacena resultado en A. ADC HL,s es una instrucción de dos octetos que suma HL más s más el ACARREO, y almacena el resultado en HL. La s significa cualquiera de los pares de registros BC, DE, HL, IX o IY. ¿Podría decirnos porqué (ignorando los señalizadores) ADC A,A hace lo mismo que RLA?.

**ADD** Muy similar a ADC, excepto que no se usa el señalizador de acarreo en la operación. Sin embargo sí le afecta el resultado final. Hay dos importantes diferencias entre ADC y ADD. Primero, el juego de instrucciones ADD HL,s (donde s significa lo mismo que en la instrucción ADC) son instrucciones de un octeto en lugar de dos. Segundo, está permitido usar otras dos instrucciones ADD IX,s y ADD IY,s.

**AND** Esta instrucción solo tiene una forma - AND r. El valor del registro A se altera de bit en bit. Si ese bit es cero, permanece sin modificar, en caso contrario toma el valor del bit de r correspondiente. Por lo tanto, AND 00 dará siempre cero como resultado y AND FF deja el registro A sin alterar. AND afecta a todos los señalizadores, el bit de acarreo se pone siempre a cero.

**BIT** El formato de esta instrucción es BIT n,r, donde n es un número entre cero y siete. La instrucción altera el señalizador de cero (solamente) de acuerdo con el valor actual del bit en cuestión. Si el bit es cero, el señalizador de cero se pone a uno, en caso contrario se pone a cero. Esta instrucción se puede usar en combinación con JR Z (que saltará si el bit era cero) o con RET NZ (que retornará si el

bit no es cero). BIT no altera el contenido de los registros, ni cambia el valor del señalizador de acarreo. Es una instrucción de dos octetos. Yo no suelo usarla casi nunca, pero cuando se necesita puede ser muy práctica.

**CALL** Ya hemos visto antes esta instrucción - es como el GO SUB. Su funcionamiento exacto es este: hace PUSH en la pila de la dirección de retorno y después salta a la dirección a la que llama. Debido a que la dirección de retorno (ahora en la pila) se usa en la instrucción RET, es de vital importancia que la subrutina no altere la pila. En la subrutina solo puede hacer PUSH de datos siempre que haga POP de ellos antes de retornar. CALL se puede usar también con condiciones - por ejemplo, CALL Z,pq (pq es una dirección absoluta) que significa IF está puesto el señalizador de cero, entonces haz CALL pq, en caso contrario continúa con la siguiente instrucción.

**CCF** Complementa el señalizador de acarreo. Si el señalizador de acarreo era cero, se pone a uno, y si era uno se pone a cero.

**CP** En la forma CP r, calcula el resultado de restar r de A, pero no almacena el resultado en ninguna parte. El valor anterior de A (y por supuesto el de r) permanecen sin alterar. Sin embargo alteran todos los señalizadores, de forma que se pueden usar instrucciones condicionales como JP Z o JP C. CP r seguido por JR Z saltará si A es igual a r (ya que A menos r es cero) y así sucesivamente.

**CPD** puede imaginarse esta instrucción como CP (HL) seguida de DEC HL y de DEC BC. El señalizador PV se pone a cero si BC se pone a cero al decrementarse, y a uno si no lo hace. El señalizador de cero se pone si la parte CP de la instrucción encuentra que A es igual a (HL), en caso contrario se pone a cero.

**CPDR** Básicamente es igual que CPD, excepto que la instrucción se ejecuta una y otra vez - es una forma de bucle automático. CPDR significa ComParar, Decrementar y Repetir. El bucle terminará en uno de estos dos casos: (I) cuando la comparación encuentra que A es igual que (HL) o (II) BC llega a cero, en cuyo caso se pone a cero el señalizador PV.

**CPI** Es como CPD, excepto que HL se incrementa en lugar de decrementarse.

**CPIR** Es como CPDR, excepto que HL se incrementa en lugar de decrementarse.

**CPL** Es una abreviatura de ComPLEMENTar. El registro A se

altera bit a bit. Si un bit particular está a uno, se pone a cero, y viceversa. En otras palabras, si A tenía 11010101 (en binario), después de una instrucción CPL se cambiaría a 00101010 (binario). Es el equivalente de restar A de FF. Los señalizadores no resultan afectados por esta instrucción.

**DAA** Suponga que quiere sumar 16 y 26 sin convertir los números a hex. Puede hacer lo siguiente: LD A,16 seguido por ADD A,26. Por desgracia, como la máquina trabaja en hex, el valor final de A será 3C en lugar de 42. La instrucción DAA (Ajuste Decimal del Acumulador) cambiará el valor de A de 3C a 42. Su funcionamiento es realmente complicado - toma nota de lo que ha ejecutado y si a sumado o restado; pero siempre funciona correctamente. Por ejemplo, la secuencia LD A,42 seguida de SUB 06 volverá a poner A con el valor 3C, pero esta vez el DAA lo cambiará a 36, ya que 42 menos seis es 36. La instrucción cambia cada uno de los señalizadores de acuerdo con el resultado.

**DEC** Esta es otra de las instrucciones que aparecen en dos formas. Puede ser DEC r (un registro simple) o DEC s (un par de registros). DEC r es fácil de comprender - el valor del registro r es decrementado (se le resta uno, o se cambia desde 00 a FF), el señalizador de acarreo permanece sin alterar y el de cero cambia como se espera que lo debería hacer. DEC s, sin embargo, es una instrucción traidora, el señalizador de cero no se altera!. En efecto, no se altera ninguno de los señalizadores. Por lo tanto, DEC BC/JR NZ,-3 es un bucle infinito o no tiene ningún efecto. Debe tener mucho cuidado y recordarlo siempre - muchos de nuestros primeros programas fallan debido a ello.

**DEFB** hablando técnicamente no es una instrucción de código máquina - es lo que se llama una directriz. La palabra DEFB debe ir seguida por uno o más octetos de datos, cada uno separado por una coma. Se suelen poner los datos en hex, pero no siempre es necesario, ej DEFB 3A,45d,11011110b,"f" es válido. Los datos se insertan dentro del programa en código máquina, en el punto en que ocurren y en el orden en que están listados. Los datos que forman parte de un programa en código máquina, no se deben ejecutar, ya que el Z80 no puede distinguir entre datos y programa.

**DEFM** es similar a DEFB, excepto que los datos que siguen a la palabra DEFM deben ser una cadena de caracteres flanqueados por comillas. Las comas dentro del texto se interpretan también como datos, no como separadores. Por ejemplo, DEFM "SOLSTICIO" hace que se inserten los octetos 53 4F 4C 53 54 49 43 49 4F dentro del programa. DEFM significa DEFINE Mensaje, al contrario que DEFB que significa DEFINE Bytes (Define Octetos).

**DEFS** Otra directriz. Esta significa **DEFine Space(s)** (Define espacios). La palabra **DEFS** debe ir seguida de una constante numérica. (Solo una, recuérdelo). El ensamblador insertará tantos ceros como indique el número. Por lo tanto, **DEFS 08** insertará ocho octetos en ese punto del programa. **DEFS** se usa principalmente para definir "variables" en RAM; ej. **PEDRO DEFS 02** (**PEDRO** es una etiqueta) y en algún otro lugar del programa **LD (PEDRO),HL**.

**DEFW** Una de las últimas directrices (por ahora). **DEFW** significa **DEFine Word** (Define palabra). Se usa de forma similar a **DEFB**, excepto que los datos tienen dos octetos de longitud, no uno, por lo que **DEFW 4000** es equivalente a **DEFB 00,40**. Observe cómo se han cambiado los octetos de orden. Con **DEFW** se podemos usar etiquetas y expresiones, por lo que **DEFW 7000,PEDRO,JUAN+3** es totalmente válido.

**DI** Significa **Disable Interrupts** (Inhabilitar interrupciones) y, aunque suena bastante confuso, su uso es inmensamente simple. Cincuenta veces cada segundo se manda un impulso a las patillas del chip **Z80**. Hay un señalizador llamado **IFF1**, que significa **Flip Flop 1** (un flip flop es un dispositivo que puede tomar uno de dos valores, como un interruptor de la luz), y el efecto del **DI** puede compararse a **RES IFF1**. Cuando el **Z80** recibe uno de estos impulsos, comprueba el valor del señalizador **IFF1**. Si está puesto, el ordenador actúa como si le hubiera llegado una instrucción **RST 38** (o **CALL 38**), con la dirección de retorno en la siguiente instrucción en secuencia. Si **IFF1** se restaura no se toma esa acción y cualquier programa en código máquina se seguirá ejecutando normalmente, el señalizador **IFF1** se debe poner a uno antes de intentar volver al **BASIC**.

**DJNZ** Otra abreviatura. Esta significa **Decrementar B** y saltar si no es cero. Por lo tanto, si **B** es 7, **DJNZ** lo reduce a 6 y salta al nuevo destino. Si **B** es uno, **DJNZ** lo pondrá a cero y no saltará. En su lugar, ejecutará la siguiente instrucción. La forma de esta instrucción es **DJNZ e**, donde **e** es un octeto simple. Si **B** se decrementa a cero, se ignora **e**. Si no, **e** especifica la longitud del salto. El desplazamiento se calcula como en una instrucción **JR**.

**EI** ¿Le suena? Es otra abreviatura. **EI** significa **Enable Interrupts** (Habilitar interrupciones), y es la opuesta a **DI**. Esta instrucción equivale a **SET IFF1**. Ver **DI** para una explicación más completa.

**EQU** Abreviatura de **EQUate** (igualar). No es una instrucción de código máquina, sino una directriz. Cada **EQU** debe tener una etiqueta, y la palabra **EQU** debe ir seguida de un número (en el rango **0000** a **FFFF**) o una expresión como **JUAN+2**.

Cuando el ensamblador encuentra la directriz, no toma ninguna acción. y no se insertan octetos en el programa - por lo tanto, no tiene importancia en que lugar del programa se sitúan los EQU, aunque se suelen colocar al principio del programa. Lo que hace es asignar un valor numérico (el valor dado) a su etiqueta. En otras palabras, si usted tiene ANA EQU 9000 y más adelante LD HL, (ANA), la instrucción se compila como LD HL, (9000).

**EX** significa EXchange (Intercambio). Esta instrucción intercambia los valores contenidos en determinados pares de registros. Hay cinco instrucciones EX - estas son EX AF, AF'; EX DE, HL; EX (SP), HL; EX (SP), IX; y EX (SP), IY. No alteran ningún señalizador, todo lo que hacen es intercambiar los valores - EX DE, HL reemplaza DE por el valor de HL y HL por el valor de DE. Las tres últimas son las más interesantes - el valor de HL (o IX o IY) se intercambian con el valor de la parte superior de la pila, de forma que LD BC, 0123/PUSH BC/LD HL, 4567/EX (SP), HL deja en BC el valor 4567 y en HL 0123. EX (SP), HL no mueve el apuntador de la pila, como tampoco lo hacen EX (SP), IX ni EX (SP), IY.

**EXX** Puede imaginarse esta instrucción como EX BC, B'C' seguida de EX DE, D'E' seguida por EX HL, H'L'. Básicamente, cada uno de los registros comunes (excepto A) se intercambia con su correspondiente registro alternativo.

**HALT** Cuando llega una instrucción HALT, el control esperará en ese punto del programa hasta que ocurra la siguiente interrupción. Cuando sucede esto, se ejecuta la instrucción RST 38 (CALL 0038) y a la vuelta, el control continúa desde la primera instrucción después de HALT. Observe que el señalizador IFF1 debe estar puesto a uno para que se ejecute el HALT, en caso contrario no ocurrirá nunca una interrupción. En ese caso, el HALT esperará literalmente para siempre. No hay forma de interrumpirlo, excepto apagando la máquina.

**IM !!!PELIGRO!!!** No use esta instrucción bajo ningún motivo.

**IN** Tiene dos formas. La primera es IN A, (n) donde n es una constante numérica. Es equivalente a LET A=IN(256\*A+n). La segunda forma es IN r, (C) donde r es un registro. Equivale a LET r=IN(256\*B+C). Los argumentos de In se refieren a un dispositivo hardware fuera del chip Z80 - un número diferente para cada dispositivo. En la forma IN A, (n) no se modifican los señalizadores; sin embargo, en IN r, (C) si resultan afectados.

**INC** ¡Que no cunda el pánico! volvemos a instrucciones sensibles que se pueden entender fácilmente. INC r incrementa el valor del registro r en uno, pero sin alterar el señalizador

de acarreo. INC s incrementa el valor de s en uno y tampoco altera los señalizadores.

**IND** IN con decremento. IND es equivalente a IN (HL), (C) seguido por DEC HL seguido por DEC B. No altera el señalizador de acarreo, pero el de cero refleja el nuevo valor de B.

**INDR** Es como IND pero se ejecuta la instrucción una y otra vez, parándose solamente cuando B a llegado a cero.

**INI** Es como IND, pero HL se incrementa en lugar de decrementarse.

**INIR** Es como INDR, pero HL se incrementa en lugar de decrementarse.

**JP** Si puede comprender lo que hace GO TO 10, entonces puede comprender JP 7300. El destino es una dirección, no un número de línea, pero el principio es exactamente el mismo. JP es el equivalente en código máquina al GO TO del BASIC. Tenemos también saltos condicionales, por ejemplo JP NZ,7300 significa IF no cero THEN salta a la dirección 7300 (En otras palabras, si el señalizador de cero no está puesto). Hay otra forma de JP que también tiene su analogía con el BASIC - con destino variable. Si comprende lo que hace GO TO N comprenderá JP (HL). JP (HL) significa GO TO HL. En esta forma no puede usar condiciones: por ejemplo, JP NC, (HL) no está permitido. Solamente se puede usar uno de los tres pares de registros como destino variable - Estos son HL, IX e IY. Son instrucciones bastante potentes, a pesar de todo - el contenido de HL puede ser el resultado de un cálculo, generado aleatoriamente.

**JR** Es la misma instrucción que JP pero ligeramente menos potente, aunque un octeto más corta. Solo se pueden usar cuatro de las ocho condiciones: Z, NZ, C y NC. Esto significa que es imposible usar (por ejemplo) JR PO. Tampoco está permitido decir JR (HL). JR no usa una dirección absoluta - la R significa Relativo. La instrucción se escribe como JR e (o JR Z,e) donde e es un solo octeto que especifica la longitud del salto. JR 0 no hace nada, ya que salta cero octetos hacia adelante. JR FE es un bucle infinito, ya que el control saltará hacia atrás a la misma instrucción JR FE. El octeto de desplazamiento comienza a contar desde la instrucción que sigue a la JR e. Si el octeto está entre 00 y 7F, el salto se realiza hacia adelante, si el octeto está entre 80 y FF, el salto se realiza hacia atrás.

**LD** Es la instrucción más usada en todo el código máquina. Todo lo que hace es transferir datos desde una posición a otra. Tiene muchas formas: la más simple puede ser LD r1,r2 -

que transfiera datos de un registro a otro. Otras formas son LD A, (BC), LD A, (DE) y LD A, (HL) - y sus inversas LD (BC), A, LD (DE), A y LD (HL), A. Recuerde que los paréntesis significan el contenido de una dirección. Los registros I y R se pueden cargar, en conjunción con A (pero solamente A) se pueden cargar los registros y pares de registros con constantes numéricas, los pares de registros con el contenido de cualquier dirección y a la inversa, cualquier dirección con el contenido de un par de registros (tenga en cuenta que los pares de registros almacenan dos octetos, no uno, y que se transfieren desde la dirección apuntada y la dirección apuntada más uno). También están permitidas LD A, (pq) y LD (pq), A (donde pq representa una dirección) y el SP se puede cargar desde HL, IX, IY o (pq). ((pq) se puede cargar desde SP pero HL, IX e IY no). En otras palabras - Hay muchas cosas que puede hacer y muchas que no. No puede decir LD HL, DE, por ejemplo (debe usar LD H, D y después LD L, E o viceversa). Afortunadamente, ya que LD se usa muy frecuentemente, es sumamente fácil familiarizarse con sus muchas formas.

**LDD** Load with Decrement (Carga con decremento). Efectivamente, equivale a LD (DE), (HL) seguido por DEC HL, DEC DE y DEC BC pero en una sola instrucción. Los señalizadores de acarreo y de cero permanecen sin alterar así como el de signo, sin embargo el PV se restaura a cero solamente si BC llega a cero. Por lo tanto, JP PO saltará solamente si BC es cero después de la instrucción.

**LDDR** Es como LDD pero la instrucción se ejecuta repetidamente hasta que BC llega a cero.

**LDI** Es como LDD pero DE y HL se incrementan en lugar de decrementarse. BC sigue decrementándose como en la anterior.

**LDIR** Es como LDI, pero la instrucción se ejecuta repetidamente hasta que BC llega a cero.

**NEG** NEGate (Niega) el acumulador (o registro A). Funciona ejecutando la resta de 00 menos A y cambiando todos los señalizadores de acuerdo con el resultado. Así, S refleja el signo del resultado, Z se pondrá solamente si A es cero. P se pondrá solamente si A es 80. C se pone siempre excepto cuando A es cero. NEG equivale a CPL seguido de INC A (ignorando los señalizadores).

**NOP** Esta extraña pérdida de tiempo (cuyo nombre accidentalmente es la abreviatura de NO oPeración) tiene un propósito muy simple - perder tiempo. Tiene dos usos principales: (I) como retraso, o (II) para eliminar código de máquina cuando se depura o edita. Supongo que el equivalente más cercano en

**BASIC** es una sentencia **REM** en blanco.

**OR** En la forma **OR r** es casi la opuesta de **AND r**. Se cambia el valor del registro **A** de bit en bit. Si alguno de los bits dados es uno, permanece sin alterar, en caso contrario toma el valor del bit correspondiente del registro **r**. Si **A** contiene **00**, entonces (ignorando los señalizadores) **OR r** es lo mismo que **LD A,r**. **OR FF** es efectivamente **LD A,FF**. Todos los señalizadores cambian como se espera, y el de acarreo se pone a cero.

**ORG** **ORG** es una directriz que no debe tener etiqueta asociada. La palabra **ORG** debe ir seguida por un número en el rango de **0000** a **FFFF**. Significa que todo el código máquina desde ese punto debe ser escrito en la dirección dada. Por lo tanto, **ORG 7000** seguido por **LD A,01** significa que la instrucción **LD A,01** reside en la dirección **7000**. A menos que lo que se encuentre a continuación sea otro **ORG**, la siguiente instrucción estará situada en **7002** (ya que **LD A,01** es una instrucción de dos octetos).

**OUT** La instrucción **OUT** tiene dos formas. La primera es **OUT (n),A** - es equivalente a decir **OUT (256\*A+n),A**. La segunda forma es **OUT (C),r** y equivale a **OUT (256\*B+C),r**. **OUT** manda datos fuera del chip **Z80** y hacia el 'hardware' que tiene alrededor. No tiene ningún efecto sobre los señalizadores.

**OUTD** **OUT** con Decremento. Equivale a **OUT (C),(HL)** seguido por **DEC HL** seguido por **DEC B**. El señalizador de acarreo no se altera, pero el de cero refleja el nuevo valor de **B**.

**OTDR** Tiene una ligera diferencia en el deletreo, que no altera el hecho de que sea una instrucción **OUT** con decremento y repetición. Equivale a **OUTD** repetida una y otra vez hasta que **B** llega a cero.

**OUTI** es como **OUTD**, excepto que **HL** se incrementa en lugar de decrementarse.

**OTIR** Es como **OTDR**, excepto que **HL** se incrementa en lugar de decrementarse.

**POP** Toma dos octetos de datos de lo alto de la pila y los carga en un par de registros. Se pueden usar los pares de registros **BC**, **DE**, **HL**, **IX** e **IY**. Además, se puede usar la instrucción **POP AF**, formando un 'seudo' par de registros con el acumulador y el registro de señalizadores. Específicamente, **POP** recoge el octeto más alto de la pila y lo pone en la parte baja del par de registros y el siguiente octeto en la parte alta. El apuntador de la pila **SP** se actualiza automáticamente.

**PUSH** es la instrucción opuesta a POP. Almacena el contenido de cualquier par de registros en lo alto de la pila. El valor de SP se actualiza para "recordar" que se ha añadido un nuevo dato a la pila. Después de una instrucción PUSH, el SP siempre apunta a la parte baja del dato de lo alto de la pila.

**RES** Con esta instrucción podemos alterar bits aislados de cualquier registro. RES es la abreviatura de REStaurar, que significa "cambiar a cero", por lo que RES es una instrucción que pone a cero cualquier bit requerido de un registro. Por ejemplo, para restaurar el bit 3 del registro D, lo único que hay que hacer es RES 3,D. RES no tiene efecto sobre ningún señalizador.

**RET** RET se usa para retornar desde una subrutina. Funciona haciendo POP de una dirección desde la pila y saltando a ella. Es posible alterar la dirección a la que retorna la subrutina alterando el valor de lo alto de la pila. Por ejemplo, POP HL/INC HL/PUSH HL incrementará la dirección de retorno en uno. Usted puede, por ejemplo, almacenar un octeto de datos inmediatamente después de la instrucción CALL, después hace POP HL/LD A, (HL)/INC HL/PUSH HL almacenará ese octeto en A mientras que se asegura que la subrutina retornará a la dirección siguiente a los datos. Otro truco es hacer PUSH de una dirección de retorno "artificial" dentro de la pila y después hacer JP (o JR) a la subrutina en lugar de usar la instrucción CALL, entonces volverá a donde le hayamos mandado. Si se necesita, se puede usar RET con condiciones. No altera a los señalizadores.

**RL** Gira hacia la izquierda. La forma de esta instrucción es RL r. Cada bit del registro especificado se mueve una posición hacia la izquierda. El bit de más a la izquierda se introduce en el señalizador de acarreo, y el de más a la derecha toma el valor previo del señalizador de acarreo. De ahí lo de girar a la izquierda. Por ejemplo, si B contenía 10010101 y el acarreo contenía cero, después de RL B, dejará B conteniendo 00101010 y el acarreo conteniendo uno. RL altera todos los señalizadores.

**RLA** Observe que no hay ningún espacio entre la L y la A. RLA es la forma más eficaz de hacer RL A. La instrucción es un octeto más corta y solo afecta al señalizador de acarreo.

**RLC** Girar a la izquierda sin acarreo. RLC r es casi lo mismo que RL r ya que cada bit del registro en cuestión se mueve una posición hacia la izquierda. Aquí, sin embargo, el bit de más a la izquierda pasa a ser el nuevo valor del señalizador de acarreo y al mismo tiempo el del bit de más a la derecha del registro. El valor que tenía el acarreo no

entra en el proceso. Se alteran todos los señalizadores.

**RLCA** En un octeto, en lugar de en dos, RLCA es lo mismo que RLC A, pero más rápido. En esta operación solo cambia el señalizador de acarreo.

**RLD** Ahora el más fantástico. RLD no se debe confundir con RL D, porque es una instrucción completamente diferente, que funciona así: el dígito superior de (HL) se desplaza hacia la izquierda pasando a ser el dígito inferior de A; éste, a su vez, pasa a ser el dígito inferior de (HL), y este último toma la posición del dígito superior de (HL). El dígito superior de A no varía. Esto es, si comenzamos con A conteniendo 25 y (HL) conteniendo A3, RLD cambiará las cosas de forma que A=2A, (HL)=35. RLD, por razones que solo conocen las mentes de los diseñadores, es la abreviatura de Rotate Left Decimal (giro decimal a la izquierda).

**RR** Es como RL, excepto que los bits se mueven hacia la derecha en lugar de hacia la izquierda.

**RRA** Es como RLA, excepto que los bits se mueven hacia la derecha en lugar de hacia la izquierda.

**RRC** Es como RLC, excepto que los bits se mueven hacia la derecha en lugar de hacia la izquierda.

**RRCA** Es como RLCA, excepto que los bits se mueven hacia la derecha en lugar de hacia la izquierda.

**RRD** Es como RLD, excepto que los bits se mueven hacia la derecha en lugar de hacia la izquierda.

**RST** Es igual que CALL excepto que la instrucción ocupa un solo octeto. Es menos potente por dos razones: (I) no puede usar condiciones (ej, RST 10 es legal, pero RST NZ,10 no lo es); y (II) solo puede especificarse una de ocho direcciones. Estas son 00, 08, 10, 18, 20, 28, 30 y 38. Ya que el Amstrad comienza a ejecutar la ROM desde la dirección 0000 en adelante, RST 00 es lo mismo que apagar y encender la máquina.

**SBC** SBC, igual que ADC, se puede usar en dos formas. La primera es SBC A,r, que lo primero que hace es restar r de A, y después resta el bit de acarreo. De forma similar SBC HL,s restará de HL, s más el bit de acarreo. SBC A,A es una instrucción muy útil - deja el bit de acarreo sin alterar pero pone 00 en A si no hay acarreo, y FF si lo hay.

**SCF** Set Carry Flag (Poner el señalizador de acarreo). Los demás señalizadores permanecen sin alterar.

**SET** Es la instrucción opuesta a RES. SET 4,H pondrá a uno el bit 4 del registro H. Se puede poner cualquier bit de cualquier registro.

**SLA** Shift Left Arithmetic (Desplazamiento aritmético hacia la izquierda). La forma de esta instrucción es SLA r. Es similar a la instrucción RL r, excepto que el bit de más a la derecha es reemplazado siempre por un cero. SLA r multiplica el registro r por dos.

**SRA** Shift Right Arithmetic (Desplazamiento Aritmético hacia la derecha). Usando SRA r podemos desplazar hacia la derecha cualquier registro. La instrucción es similar a RR r, excepto que el bit de más a la izquierda permanece sin cambiar. SRA r divide por dos el contenido del registro r, si el registro contiene un valor en formato de complemento a dos.

**SRL** Shift Right Logical (Desplazamiento lógico hacia la derecha), es similar a RR, excepto que aquí, el bit de más a la izquierda queda reemplazado por cero. SRL r divide el registro por dos, si el registro contiene un valor en formato de complemento a dos.

**SUB** Se escribe SUB r (algunas veces se escribe también como SUB A,r, solamente para confundir). Esta instrucción resta r del registro A. Tenga en cuenta que, a diferencia de ADD, no hay instrucción correspondiente SUB HL,s. Si quiere restar de HL, debe restaurar primero el bit de acarreo (por medio de la instrucción AND A) y después usar SBC HL,s.

**XOR** XOR cambia el valor del registro A bit a bit. Si un bit dado de A es idéntico que el correspondiente bit de r, ese bit de A es restaurado a cero, en caso contrario ese bit del registro A se pondrá a uno. XOR altera todos los señalizadores y, en particular, el de acarreo se restaura siempre. Observe que XOR A es lo mismo que LD A,00 (ignorando los señalizadores) y que XOR FF es lo mismo que CPL (ignorando también los señalizadores).

# CAPITULO NUEVE

## Operadores Lógicos y Manipulación de Bits

En este capítulo trataremos de un grupo de instrucciones que aparecen bajo el nombre genérico de operadores lógicos (AND, OR, XOR), y de otras que sirven para la manipulación de bits dentro de un octeto (SET, RES, BIT, Giros y Desplazamientos).

Antes de seguir adelante debemos dejar claro una cosa, de ahora en adelante se hablará de los bits dentro de un octeto por el número correspondiente al lugar que ocupan dentro de él. Este orden es como sigue:

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

Veamos primero los operadores lógicos. Se conocen también como **Operaciones Booleanas**. Cada operación se describe acompañada por su **Tabla de la Verdad**. Esta tabla nos muestra el resultado de las operaciones lógicas.

La tabla de la verdad de la instrucción **AND** es así:

A	B	A AND B
0	0	0
1	0	0
0	1	0
1	1	1

Después de ver esta tabla, hay unas cuantas cosas que deben ser explicadas. AND funciona con dos valores. El primero debe estar en el registro A, mientras que el otro debe ser un número de 8 bits, n o cualquier registro simple (incluido el A) o el valor que hay en la dirección apuntada por IX, IY

o HL. Nosotros hemos usado el registro B en el ejemplo anterior.

Debido a que A debe ser siempre el primero de los dos valores, el código de operación de AND A,C se escribe como AND C, asumiendo que A viene antes de la C.

La tabla de la verdad nos muestra el resultado de un AND de los bits de los dos registros. Esta tabla se aplica a los ocho bits de los registros. Si el bit 0 de A o el bit 0 de B son cero, el resultado será un bit 0 a cero. Si ambos bits son cero, el resultado será también cero, mientras que si ambos bits tienen el valor 1, el resultado del AND será 1.

El resultado de la operación se almacenará en el registro A. El valor del registro B (o del registro que se haya usado) permanece sin alterar. Es muy importante recordar esto. Aquí tenemos una pequeña rutina que nos muestra la instrucción AND en acción:

```
3E 66      LA  A,102      (102 en binario es 01100110)
06 2B      LD  B,43      (43 en binario es 00101011)
A0         AND B
32 28 A0   LD  (41000),A
```

La rutina que ejecuta AND A,B (43 AND 102) da el siguiente resultado:

```
01100110
00101011
00100010
```

Como resultado del AND, solo se ponen a uno los bits 1 y 5. El valor del registro A se introduce en la posición 41000 de memoria. Para comprobar que este valor es en efecto el 00100010 (de valor decimal 34), teclee PRINT PEEK(41000).

La tabla de la verdad de la instrucción OR es:

A	B	A OR B
0	0	0
1	0	1
0	1	1
1	1	1

OR funciona con los mismos valores que AND, estos son:

AND r	(donde r es un registro de 8 bits)
AND n	(donde n es un número de 8 bits)
AND (HL)	(el contenido de la memoria en HL)
AND (IX+d)	(el contenido de la posición de
AND (IY+d)	memoria IX o IY + desplazamiento)

La tabla de la verdad de la instrucción XOR es:

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

Este operador lógico funciona con los mismos registros y modos de direccionamiento que OR y AND. XOR es la abreviatura de Exclusive Oring (OR exclusivo) y es un procedimiento en el cual se asigna un 1 al bit del resultado si los bits correspondientes de los dos operandos son distintos. Si los dos bits son 0 o 1, el resultado en el registro A será 0.

Ya hemos visto los operadores lógicos. Ahora los veremos en acción. Son capaces de cambiar fácilmente un bit específico, y es lo que usa la próxima rutina. La podemos llamar "creador de mayúsculas". La rutina toma una cadena y convierte el primer carácter en mayúscula.

¿Cómo lo hace? Las letras mayúsculas en ASCII comienzan 32 códigos por debajo de las minúsculas. El programa toma el código del primer carácter y, usando el registro A para contener el valor 223, y la instrucción AND cancela el sexto bit haciéndolo igual a cero, es decir restando 32 del valor

del código. El código del carácter se devuelve al lugar en que estaba. Veámoslo en binario.

A=11011111

B=011????? (no sabemos el valor exacto de B, todo lo que sabemos es que va de 97 en adelante, pero el bit 6 está puesto siempre, que equivale a 32 decimal)

La operación AND toma los bits que son igual a 1 y da como resultado un bit a 1. Cualquiera de los bits que sean distintos darán como resultado un bit a cero. Poniendo a cero el sexto bit del registro A garantizamos que se resta 32 del valor del código, obteniendo así la letra correspondiente pero convertida en mayúscula.

Ensamblador	Decimal
LD L, (IX+0)	DD 6E 00
LD H, (IX+1)	FD 6E 01
INC HL	23
LD C, (HL)	4E
LD A, 223	3E DF
AND C	A1
LD (HL), A	77
RET	C9

## SET y RES

SET y RES nos permiten alterar el valor de bits determinados dentro de un registro de 8 bits. El formato de SET y RES es:

SET/RES n,r

Aquí, r es cualquier registro de 8 bits o una posición de memoria de 8 bits, apuntada por los registros HL, IX o IY. N es un número entre 0 y 7 e indica el bit sobre el que se va

a ejecutar la instrucción. (SET hace que el bit se ponga a 1 mientras que RESET hace que se ponga a cero).

```
LD  B,3   pone a 1 los bits primero y segundo, y si
          añadimos la instrucción...
SET 2,B   el tercer bit se pondrá a 1 (valor 4) y el
          valor del registro B será ahora de 7 decimal,
          en binario 00000111.
```

El punto más importante a tener en cuenta, no solo con las instrucciones SET y RES sino con todas las que acabamos de ver, es que los bits de un octeto se numeran del 0 al 7, de derecha a izquierda. Por lo tanto, si hablamos del tercer bit, este será el bit 2 (como en el ejemplo anterior). Recuerde esto, ya que muchos libros y revistas no lo aclaran, y causa mucha confusión.

Si tenemos estas dos instrucciones:

```
LD  C,7
RES 0,C
```

El valor del registro C pasará a ser 6, debido a que el primer bit (bit 0) se ha puesto a cero, restando uno al valor del registro C.

Se puede usar la instrucción BIT para saber el valor de un bit. Se debe especificar el número del bit y el registro, de la misma forma que con SET y RES. La respuesta no se pone en un registro, sino en uno de los señalizadores del registro F. El señalizador Z o de Cero se pone a 1 si el bit que se está comprobando es un cero. Si el bit es 1, el señalizador de Cero se pone a cero (lo contrario a lo que podría parecer en un principio).

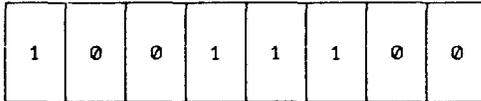
## Gírar y Desplazar

Vayamos a ver ahora las instrucciones de giro y desplazamiento. Los principios que rigen estas operaciones no son

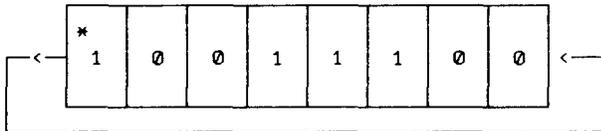
difíciles de entender, pero si no se explican bien pueden quedar algo confusas. Hemos requerido la ayuda de unos diagramas para explicarlo. Veamos el primer diagrama, que corresponde a un giro:

Giro hacia la izquierda (RLC)

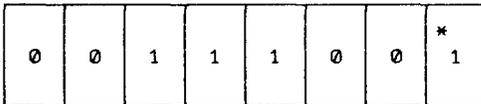
(1) Comenzamos con



(2) Operación de Giro



(3) Resultado final



\* Indica el bit que sale

Observe que esto es un RLC giro hacia la izquierda sin acarreo.

El giro hacia la derecha (RRC) es justo lo opuesto.

El giro es una operación que involucra un ciclo completo, y es en lo que se diferencia del desplazamiento (SHIFT), como veremos enseguida.

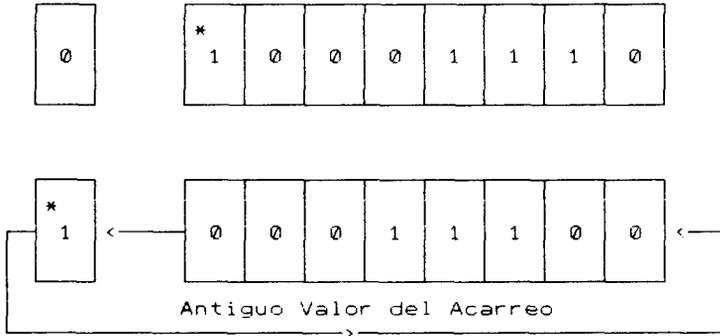
Los comandos que se han usado en este giro son RLC y RRC. Significan Rotate Left **without** Carry (Rotar hacia la izquierda **sin** acarreo) y Rotate Right **without** Carry (Rotar hacia la derecha **sin** acarreo). En el diagrama se puede ver lo que hacen.

Hay otros dos tipos de giro que se conocen como RL y RR (Rotate Left/Right **with** Carry (girar hacia la izquierda/de-

recha con acarreo)). Lo que hacen es tomar el bit del extremo y ponerlo dentro del bit de acarreo y tomar el valor que había en el bit de acarreo y ponerlo en el otro extremo del registro. El segundo diagrama es así:

Giro hacia la izquierda con Acarreo (RL)

Señalizador de acarreo



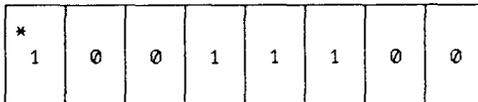
\* Indica el bit a que sale

Estas instrucciones de rotación trabajan con registros de 8 bits y valores dados por IX, IY y HL.

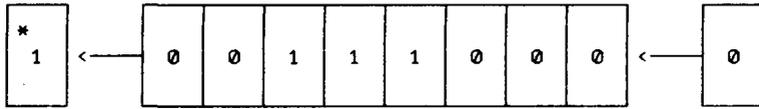
Los desplazamientos son similares a los giros, como se ve en este tercer diagrama.

Desplazamiento aritmético hacia la izquierda (SLA)

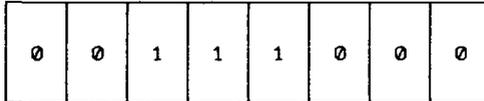
(1) Comenzamos con



(2) Operación de Desplazamiento



(3) Resultado final



\* Indica el bit que sale

Esto es una instrucción Shift Left Arithmetic (SLA) (Desplazamiento aritmético hacia la izquierda).

Hay tres tipos de instrucciones de desplazamiento: SLA, SRA y SRL. SLA significa Shift Left Arithmetic (Desplazamiento aritmético hacia la izquierda) y es la operación que se ha desarrollado en el diagrama. SRL significa Shift Right Logical (Desplazamiento lógico hacia la derecha) y funciona de forma similar a SLA, excepto que se desplaza hacia la derecha, el bit 7 se pone a cero y el bit 0 (el de más a la derecha) se introduce en el bit de acarreo, mientras que el resto de los bits se desplazan un bit hacia abajo. Por lo tanto una instrucción SRL con 11100100 dará como resultado 01110010.

La última instrucción de desplazamiento es SRA, que significa Shift Right Arithmetic (Desplazamiento aritmético hacia la derecha). Es similar a SRL, excepto que desplaza todos los bits excepto el 7, que se mantiene con su valor (cuando se usa con enteros con signo entre -127 y 127). Todas las instrucciones de desplazamiento funcionan con los mismos registros, como las instrucciones de rotación. Lo que puede que no sepa usted, aunque resulta lógico, es que SLA realmente multiplica el número por dos y SRL divide el número por 2. Por lo tanto, para multiplicar un número por 8, basta con hacer SLA tres veces.

## CAPITULO DIEZ

### Pantalla y Rutinas de la ROM

La imagen de la pantalla del Amstrad ocupa 16K (16384 octetos) en todos los modos. No se puede hacer nada por reducir este espacio. (Puede que usted quiera acceder a la pantalla directamente en alguno de sus programas en código máquina, en lugar de usar las rutinas disponibles en la ROM. Esto tiene la ventaja de hacer que los programas se ejecuten más rápidamente, en la mayoría de los casos. Por desgracia, el asociar una dirección de memoria con una posición de la pantalla del Amstrad puede ser una tarea imposible en un principio, debido a la forma en que está almacenada en memoria. También hay que tener en cuenta que un punto de la pantalla no se relaciona necesariamente con un bit concreto dentro de un octeto determinado, sino que depende del modo de pantalla que esté usando. En MODE 2 hay solamente dos colores disponibles, ya que cada bit puede estar en uno de dos estados, 1 o 0).

Vamos a ver cómo se almacena la pantalla, sabiendo que está almacenada desde la posición &C000 hasta &FFFF. Este será el caso normal a menos que un programa en código máquina la haya movido a algún otro lugar. Vamos a asumir que el desplazamiento es igual a cero. Este no es el caso si hacemos 'scroll' de la pantalla.

#### Doscientos de Alto

La pantalla tiene siempre 200 líneas de puntos de altura por 80 octetos de ancho. Cuando usted pone el modo de pantalla, le está diciendo al ordenador cómo usar estos 80 octetos; si quiere un número mayor de puntos a lo ancho de la pantalla o si quiere menos puntos, usando los bits extras para almacenar rangos más amplios de colores.

La pantalla se almacena como ocho bloques de 2K conteniendo cada bloque la información de una de las ocho líneas de altura de cada carácter. Por lo tanto, los primeros 80 octetos del bloque uno contienen la línea de puntos superior, que es la línea 1 de la primera fila de caracteres, el

segundo bloque de 80 octetos contiene la línea 1 de la segunda fila de caracteres, y así sucesivamente.

Esto significa que la línea 1 de la última fila de caracteres (fila 25) se almacena en el primer bloque desde el octeto 1921 ( $24 \times 80 + 1$ ) hasta el 2000 ( $625 \times 80$ ). Sin embargo, cada bloque es de 2K (2048 octetos) de longitud y por lo tanto no se usan los últimos 48 octetos de cada bloque. De la misma forma, el segundo bloque almacena información de la línea 2 de cada fila de caracteres y el bloque ocho almacena información de la línea 8 de cada fila de caracteres.

Si está un poco confundido, teclee el siguiente programa BASIC que hace POKE de cada octeto desde &C000 a &FFFF con 255. Esto significa que también altera los octetos no usados, pero al ordenador no le importa. Observe el orden en el que aparecen las líneas en la pantalla.

```
10 MODE 1
20 FOR n=&C000 TO &FFFF
30 POKE n,255
40 NEXT
```

Cambie el número del MODE de la línea 10 y ejecute el programa de nuevo. Observe que tarda el mismo tiempo en rellenar la pantalla en todos los modos y que aparecen distintos colores en cada modo.

Hay muchas rutinas de la ROM que se pueden usar para tareas como imprimir o inicializar. (Todas ellas están explicadas en el libro de la ROM suministrado con el Amstrad).

Aquí tenemos las direcciones y los nombres de algunas de las rutinas más útiles para un principiante en la programación en código máquina. Tenga en cuenta que todas las direcciones vienen dadas en hexadecimal.

Dirección de CALL	Descripción
BB00	Restaura la memoria intermedia del teclado.
BB18	Espera a que se pulse una tecla.

- BB24      Obtiene el valor de la palanca de juego  
          cuyos valores son:  
          0=arriba            1=abajo  
          2=izquierda        3=derecha  
          4=fuego1            5=fuego 2  
          6=sin uso            7=restaurar
- BB5A      Imprime un carácter en la pantalla.
- BBEA      dibuja un punto con DE=coordenada X y  
          HL=coordenada Y
- BBF6      Dibuja una línea desde la posición actual  
          del cursor hasta X, Y (almacenadas en los  
          mismos registros que la anterior).
- BC38      Pone el color del borde con los colores  
          almacenados en los registros B y C.
- BC3E      Pone los periodos de parpadeo de acuerdo con  
          los valores almacenados en el registro HL.
- BC68      Fija la velocidad de grabación del cassette.
- BCAA      Añade sonido a la cola de sonido.
- BCB6      Para el sonido.
- BCD1      Introduce un RSX en el sistema.
- BD31      Manda un carácter a la puerta Centronics  
          (normalmente para la impresora).
- BB39      Pone la repetición de tecla o la quita.
- BB21      Obtiene los estados de Bloqueo Mayúsculas y,  
          Bloqueo 'Shift'.

## Paquete de Rutinas en Código Máquina

Uno - Leer un Carácter  
Dos - Girar hacia la Izquierda  
Tres - Girar hacia la Derecha  
Cuatro - Letras Gigantes  
Cinco - Impresión Masiva  
Seis - Relleno de la Pantalla  
Siete - LOAD/SAVE Sin Cabecera  
Ocho - Música por Interrupciones  
Nueve - Monitor de Código Máquina  
Diez - Movimiento de Bloques de Pantalla  
Once - Acordes RSX  
Doce - Compresores de Pantalla  
Trece - DEEK y DOKE  
Catorce - Paquete de Escritura de Juegos

## UNO - Leer un Carácter

Si se preguntara a varios programadores BASIC cuál sería el comando que les gustaría tener en el Amstrad, no nos sorprendería que la mayoría de ellos pidieran un comando SCREEN\$. Para los no iniciados, SCREEN\$ y los comandos similares le dicen al programador qué carácter hay en una posición determinada de la pantalla. Por lo que, si el programador quiere escribir una rutina que tenga que comprobar si en el centro de la pantalla hay un asterisco (\*), puede escribir una línea como esta:

```
IF SCREEN$(20,12)=42 THEN ...
```

Los dos números que aparecen entre los paréntesis son las coordenadas de la posición del cursor y el 42 es el código ASCII del asterisco.

Desgraciadamente, el Amstrad no tiene este comando BASIC y esto hace la vida un poco más difícil a los programadores que desean escribir juegos con gráficos en movimiento. La siguiente rutina resuelve este problema y funciona en cualquiera de los tres modos de pantalla del Amstrad.

Teclee este programa y ejecútelo:

```
1          LEER UN CARACTER
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40019
30 READ a$:POKE n,VAL("&" + a$)
40 NEXT
50 DATA DD,6E,02,DD,66,04,CD,75,BB,CD
60 DATA 60,BB,DD,6E,00,DD,66,01,77,C9
```

Cuando quiera comprobar una posición específica de memoria, use el siguiente comando:

CHAR%=0: CALL 40000,X,Y,@CHAR%

Aquí, X e Y son las coordenadas de la posición que se quiere comprobar (en el mismo formato que el comando BASIC LOCATE). CHAR% es una variable especialmente creada que, una vez que se ha ejecutado la CALL, contendrá el código ASCII del carácter situado en la posición especificada. Si no se define CHAR% de antemano, ocurrirá un error **Improper Argument**.

La variable CHAR debe ser un entero y se puede definir seguida por el signo del porcentaje (%), como hemos hecho antes, o precediéndolo por un comando DEFINT C.

Esta es una de las rutinas más cortas de este libro, y lo es gracias a la ROM del Amstrad. La ROM contiene una rutina de "lectura de caracteres" a la que puede accederse desde el BASIC. Nuestro programa utiliza esta rutina y pasa los parámetros correctos a, y desde esta rutina de la ROM.

Este es el listado de la rutina en ensamblador:

```
0001 ;   LEE UN CARACTER
0002 ;
9C40      0010      ORG 40000
BB75      0020  CURSOR DEFL 0BB75H
BB60      0030  RDCHAR DEFL 0BB60H
9C40 DD6E02  0040      LD  L, (IX+2)
9C43 DD6604  0050      LD  H, (IX+4)
9C46 CD75BB  0060      CALL CURSOR
9C49 CD60BB  0070      CALL RDCHAR
9C4C DD6E00  0080      LD  L, (IX+0)
9C4F DD6601  0090      LD  H, (IX+1)
9C52 77      0100      LD  (HL), A
9C53 C9      0110      RET
          0120      END
```

# DOS - Girar hacia la Izquierda

Esta ruina gira una cadena de caracteres 90 grados hacia la izquierda, poniéndolos de lado hacia arriba de la pantalla.

```
1          GIRAR A LA IZQUIERDA
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40091
30 READ a$:POKE n,VAL("&"a$)
40 NEXT
50 DATA DD,6E,00,DD,66,01,7E,B7,C8,23
60 DATA 5E,23,56,DD,6E,02,DD,66,04,47
70 DATA C5,E5,CD,75,BB,1A,CD,64,9C,13
80 DATA E1,2D,C1,10,F1,C9,D5,CD,A5,BB
90 DATA EB,CD,AE,BB,06,07,23,10,FD,F5
100 DATA CD,06,B9,0E,08,1A,13,D5,E5,16
110 DATA 80,1E,80,06,08,82,CB,16,2B,B3
120 DATA CB,3A,CB,2B,CB,F3,10,F3,E1,D1
130 DATA 0D,20,E4,CD,09,B9,F1,CD,5A,BB
140 DATA D1,C9
```

El formato de este comando es:

```
CALL 40000,X,Y,@A$
```

X e Y son las coordenadas donde quiere que comience a escribirse y A\$ contiene la cadena que se va a escribir, y debe ir precedida por el carácter '@'. Esto nos permite pasar una cadena de caracteres desde el BASIC al código máquina. A\$ se usa aquí para definir cualquier variable de cadena pero no una cadena en sí misma (sin embargo X puede ser tanto

una variable numérica como un valor). X\$ y pp\$ funcionarán, pero "HOLA" no lo hará. En otras palabras, siempre se debe especificar una variable de cadena.

Las coordenadas X e Y son las mismas que se usan en el comando LOCATE del BASIC; Y puede ser cualquier valor de línea, entre 1 y 25, y X puede ser cualquier número de columna entre 1 y 20 en modo 0, 1 y 40 en modo 1, y 1 y 80 en modo 2.

La rutina toma el primer carácter de la cadena, la pone de lado y coloca el carácter dentro del primer UDG (Carácter Definido por el Usuario) disponible (en el cargador de BASIC lo definimos como el carácter 240). Después escribe el carácter en la pantalla. Esto hace que no podamos usar el primer UDG si tenemos esta rutina incorporada en el programa. Para salvar este inconveniente, si lo necesitamos, se puede definir un SYMBOL AFTER más bajo.

```
0001 ; GIRAR A LA IZQUIERDA
0002 ;
9C40 0010 ORG 40000
BB75 0020 CURSOR DEFL 0BB75H
9C40 DD6E00 0030 LD L, (IX+0)
9C43 DD6601 0040 LD H, (IX+1)
9C46 7E 0050 LD A, (HL)
9C47 B7 0060 OR A
9C48 C8 0070 RET Z
9C49 23 0080 INC HL
9C4A 5E 0090 LD E, (HL)
9C4B 23 0100 INC HL
9C4C 56 0110 LD D, (HL)
9C4D DD6E02 0120 LD L, (IX+2)
9C50 DD6604 0130 LD H, (IX+4)
9C53 47 0140 LD B, A
9C54 C5 0150 NXTCHR PUSH BC
9C55 E5 0160 PUSH HL
9C56 CD75BB 0170 CALL CURSOR
9C59 1A 0180 LD A, (DE)
```

9C5A	CD649C	0190	CALL	ROTCHR
9C5D	13	0200	INC	DE
9C5E	E1	0210	POP	HL
9C5F	2D	0220	DEC	L
9C60	C1	0230	POP	BC
9C61	10F1	0240	DJNZ	NXTCHR
9C63	C9	0250	RET	
9C64	D5	0260	ROTCHR	PUSH DE
BBA5		0270	MATRIX	DEFL 0BBA5H
BBAE		0280	TABLE	DEFL 0BBAEH
B906		0290	ROMENA	DEFL 0B906H
B909		0300	ROMDIS	DEFL 0B909H
BB5A		0310	TXTOUT	DEFL 0BB5AH
9C65	CDA5BB	0320	CALL	MATRIX
9C68	EB	0330	EX	DE, HL
9C69	CDAEBB	0340	CALL	TABLE
9C6C	0607	0350	LD	B, 7
9C6E	23	0360	ADD7	INC HL
9C6F	10FD	0370	DJNZ	ADD7
9C71	F5	0380	PUSH	AF
9C72	CD06B9	0390	CALL	ROMENA
9C75	0E08	0400	LD	C, 8
9C77	1A	0410	NXTBYT	LA A, (DE)
9C78	13	0420	INC	DE
9C79	D5	0430	PUSH	DE
9C7A	E5	0440	PUSH	HL
9C7B	1680	0450	LD	D, 128
9C7D	1E80	0460	LD	E, 128
9C7F	0608	0470	LD	B, 8
9C81	82	0480	NXTBIT	ADD A, D
9C82	CB16	0490	RL	(HL)
9C84	2B	0500	DEC	HL
9C85	B3	0510	OR	E
9C86	CB3A	0520	SRL	D

9C88 CB28	0530	SRA E
9C8A CBF3	0540	SET 6, E
9C8C 10F3	0550	DJNZ NXTBIT
9C8E E1	0560	POP HL
9C8F D1	0570	POP DE
9C90 0D	0580	DEC C
9C91 20E4	0590	JR NZ, NXTBYT
9C93 CD09B9	0600	CALL ROMDIS
9C96 F1	0610	POP AF
9C97 CD5ABB	0620	CALL TXTOUT
9C9A D1	0630	POP DE
9C9B C9	0640	RET
	0650	END

## TRES - Girar hacia la Derecha

Esta rutina viene a ser, más o menos, la opuesta a la anterior. Con ésta, la cadena se imprimirá en la pantalla hacia abajo, a 90 grados de la posición horizontal y a 180 grados de la posición de la rutina Girar hacia la Izquierda.

```
1      GIRAR A LA DERECHA
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40086
30 READ a$:POKE n,VAL("&" + a$)
40 NEXT
50 DATA DD,6E,00,DD,66,01,7E,B7,C8,23
60 DATA 5E,23,56,DD,6E,02,DD,66,04,47
70 DATA C5,E5,CD,75,BB,1A,CD,64,9C,13
80 DATA E1,2C,C1,10,F1,C9,D5,CD,A5,BB
90 DATA EB,CD,AE,BB,F5,CD,06,B9,0E,08
100 DATA 1A,13,D5,E5,16,80,1E,80,06,08
110 DATA 82,CB,1E,23,B3,CB,3A,CB,2B,CB
120 DATA F3,10,F3,E1,D1,0D,20,E4,CD,09
130 DATA B9,F1,CD,5A,BB,D1,C9
```

El formato de la llamada es el mismo que en la anterior:

```
CALL 40000,X,Y,@A$
```

Aquí X e Y son las coordenadas de comienzo de la cadena, y A\$ es la cadena que queremos escribir girada hacia abajo en la pantalla.

Las dos rutinas tienen mucho en común, como se puede ver en el listado del ensamblaje. Afecta a los UDGs de la misma forma que la rutina Girar hacia la Izquierda.

```

0001 ;      GIRAR A LA DERECHA
0002 ;

9C40      0010      ORG  40000
BB75      0020  CURSOR  DEFL 0BB75H
9C40 DD6E00  0030      LD   L, (IX+0)
9C43 DD6601  0040      LD   H, (IX+1)
9C46 7E      0050      LD   A, (HL)
9C47 B7      0060      OR   A
9C48 C8      0070      RET  Z
9C49 23      0080      INC  HL
9C4A 5E      0090      LD   E, (HL)
9C4B 23      0100      INC  HL
9C4C 56      0110      LD   D, (HL)
9C4D DD6E02  0120      LD   L, (IX+2)
9C50 DD6604  0130      LD   H, (IX+4)
9C53 47      0140      LD   B, A
9C54 C5      0150  NXTCHR  PUSH BC
9C55 E5      0160      PUSH HL
9C56 CD75BB  0170      CALL CURSOR
9C59 1A      0180      LD   A, (DE)
9C5A CD649C  0190      CALL ROTCHR
9C5D 13      0200      INC  DE
9C5E E1      0210      POP  HL
9C5F 2D      0220      DEC  L
9C60 C1      0230      POP  BC
9C61 10F1    0240      DJNZ NXTCHR
9C63 C9      0250      RET
9C64 D5      0260  ROTCHR  PUSH DE
BB45      0270  MATRIX  DEFL 0BBA5H
BBAE      0280  TABLE  DEFL 0BBAEH
B906      0290  ROMENA  DEFL 0B906H
B909      0300  ROMDIS  DEFL 0B909H
BB5A      0310  TXTOUT  DEFL 0BB5AH
9C65 CDA5BB  0320      CALL MATRIX

```

9C68	EB	0330	EX	DE, HL
9C69	CDAEBB	0340	CALL	TABLE
9C6C	F5	0350	PUSH	AF
9C6D	CD06B9	0360	CALL	ROMENA
9C70	0E08	0370	LD	C, 8
9C72	1A	0380	NXTBYT	LA A, (DE)
9C73	13	0390	INC	DE
9C74	D5	0400	PUSH	DE
9C75	E5	0410	PUSH	HL
9C76	1680	0420	LD	D, 128
9C78	1E80	0430	LD	E, 128
9C7A	0608	0440	LD	B, 8
9C7C	82	0450	NXTBIT	ADD A, D
9C7D	CB1E	0460	RR	(HL)
9C7F	23	0470	INC	HL
9C80	B3	0480	OR	E
9C81	CB3A	0490	SRL	D
9C83	CB2B	0500	SRA	E
9C85	CBF3	0510	SET	6, E
9C87	10F3	0520	DJNZ	NXTBIT
9C89	E1	0530	POP	HL
9C8A	D1	0540	POP	DE
9C8B	0D	0550	DEC	C
9C8C	20E4	0560	JR	NZ, NXTBYT
9C8E	CD09B9	0570	CALL	ROMDIS
9C91	F1	0580	POP	AF
9C92	CD5ABB	0590	CALL	TXTOUT
9C95	D1	0600	POP	DE
9C96	C9	0610	RET	
		0620	END	

## CUATRO - Letras Gigantes

Con esta rutina podemos disponer de caracteres de doble tamaño. Tiene muchos usos, como poner títulos a los listados o a las pantallas.

El cargador BASIC se tecllea como en las rutinas anteriores, se salva en cinta o disco y luego se ejecuta.

```
1 '          LETRAS GIGANTES
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40159
30 READ a$:POKE n,VAL("&" + a$)
40 NEXT
50 DATA CD,93,BB,F5,DD,6E,04,DD,66,05
60 DATA 46,23,5E,23,56,DD,6E,06,DD,66
70 DATA 08,C5,D5,E5,1A,47,CD,06,B9,78
80 DATA CD,9D,9C,47,CD,09,B9,E1,5D,54
90 DATA CD,75,BB,DD,7E,02,CD,90,BB,78
100 DATA CD,5A,BB,3C,CD,5A,BB,6B,62,2C
110 DATA CD,75,BB,DD,7E,00,CD,90,BB,78
120 DATA 3C,3C,CD,5A,BB,3C,CD,5A,BB,6B
130 DATA 62,24,24,D1,13,C1,10,BD,F1,CD
140 DATA 90,BB,C9,CD,A5,BB,EB,CD,AE,BB
150 DATA F5,0E,02,06,04,C5,1A,0F,0F,0F
160 DATA 0F,06,04,1F,CB,1E,CB,2E,10,F9
170 DATA 7E,23,77,06,07,23,10,FD,1A,06
180 DATA 04,1F,CB,1E,CB,2E,10,F9,7E,23
190 DATA 77,06,07,2B,10,FD,13,C1,10,D3
200 DATA 06,08,23,10,FD,0D,20,C9,F1,C9
```

Para usar esta rutina, debe teclear el siguiente comando:

CALL 40000,X,Y,@A\$,P1,P2

Por el momento debe ser capaz de usar los dos o tres primeros parámetros por usted mismo. En caso de que no sepa usarla, o esté algo confuso, le diré como hacerlo. X e Y son las coordenadas de la parte superior izquierda de la posición de la pantalla desde la que se van a comenzar a escribir los caracteres y A\$ es la variable de cadena que contiene el texto que se va a imprimir. Los nombres de variables que se usan en el ejemplo no tienen por que ser las que use usted en su programa; se puede usar cualquier variable numérica o de cadena y, para las coordenadas, se pueden utilizar números enteros directamente.

Este es también el caso de los dos parámetros finales, a los que hemos llamado P1 y P2. Pueden ser tanto variables numéricas como números, y lo que hacen es definir el color de los caracteres. P1 controla la mitad superior de cada carácter y P2 la inferior. Estas variables pueden contener cualquier número, ya que, si el parámetro se sale del rango de los valores usuales (que puede ver en la tabla de abajo), la rutina los enmascarará para que se acoplen a los valores normales del rango de colores.

Valores Normales	Modo	Rango
	0	0-15
	1	0-3
	2	0-1

La rutina vuelve a poner los colores que tenía antes de ser llamada. Esta rutina utiliza los cuatro primeros caracteres UDG para crear los caracteres de doble tamaño. Por lo tanto, asegúrese de que tiene disponibles por lo menos cuatro UDGs. (Si no está usando ningún UDG, deje en su programa el comando SYMBOL AFTER 240, que hay en el cargador BASIC).

```
0001 ; LETRAS GIGANTES
0002 ;
9C40 0010      ORG 40000
B906 0020 ROMENA DEFL 0B906H
```

B909	0030	ROMDIS	DEFL	0B909H
BBA5	0040	MATRIX	DEFL	0BBA5H
BBAE	0050	TABLE	DEFL	0BB75H
BB75	0060	CURSOR	DEFL	0BB75H
BB93	0070	GETPEN	DEFL	0BB93H
BB90	0080	SETPEN	DEFL	0BB90H
BB5A	0090	TXTOUT	DEFL	0BB5AH
9C40 CD93BB	0100	CALL	GETPEN	
9C43 F5	0110	PUSH	AF	
9C44 DD6E04	0120	LD	L, (IX+4)	
9C47 DD6605	0130	LD	H, (IX+5)	
9C4A 46	0140	LD	B, (HL)	
9C4B 23	0150	INC	HL	
9C4C 5E	0160	LD	E, (HL)	
9C4D 23	0170	INC	HL	
9C4E 56	0180	LD	D, (HL)	
9C4F DD6E06	0190	LD	L, (IX+6)	
9C52 DD6608	0200	LD	H, (IX+8)	
9C55 C5	0210	NXTCHR	PUSH	BC
9C56 D5	0220	PUSH	DE	
9C57 E5	0230	PUSH	HL	
9C58 1A	0240	LD	A, (DE)	
9C59 47	0250	LD	B, A	
9C5A CD06B9	0260	CALL	ROMENA	
9C5D 78	0270	LD	A, B	
9C5E CD9D9C	0280	CALL	BIGCHR	
9C61 47	0290	LD	B, A	
9C62 CD09B9	0300	CALL	ROMDIS	
9C65 E1	0310	POP	HL	
9C66 5D	0320	LD	E, L	
9C67 54	0330	LD	D, H	
9C68 CD75BB	0340	CALL	CURSOR	
9C6B DD7E02	0350	LD	A, (IX+2)	
9C6E CD90BB	0360	CALL	SETPEN	

9C71	78	0370	LD	A, B
9C72	CD5ABB	0380	CALL	TXTOUT
9C75	3C	0390	INC	A
9C76	CD5ABB	0400	CALL	TXTOUT
9C79	6B	0410	LD	L, E
9C7A	62	0420	LD	H, D
9C7B	2C	0430	INC	L
9C7C	CD75BB	0440	CALL	CURSOR
9C7F	DD7E00	0450	LD	A, (IX+0)
9C82	CD90BB	0460	CALL	SETPEN
9C85	78	0470	LD	A, B
9C86	3C	0480	INC	A
9C87	3C	0490	INC	A
9C88	CD5ABB	0500	CALL	TXTOUT
9C8B	3C	0510	INC	A
9C8C	CD5ABB	0520	CALL	TXTOUT
9C8F	6B	0530	LD	L, E
9C90	62	0540	LD	H, D
9C91	24	0550	INC	H
9C92	24	0560	INC	H
9C93	D1	0570	POP	DE
9C94	13	0580	INC	DE
9C95	C1	0590	POP	BC
9C96	10BD	0600	DJNZ	NXTCHR
9C98	F1	0610	POP	AF
9C99	CD90BB	0620	CALL	SETPEN
9C9C	C9	0630	RET	
9C9D	CDA5BB	0640	BIGCHR	CALL MATRIX
9CA0	EB	0650	EX	DE, HL
9CA1	CDAEBB	0660	CALL	TABLE
9CA4	F5	0670	PUSH	AF
9CA5	0E02	0680	LD	C, 2
9CA7	0604	0690	NXTSET	LD B, 4
9CA9	C5	0700	NXTROW	PUSH BC

9CAA 1A	0710		LD A, (DE)
9CAB 0F	0720		RRCA
9CAC 0F	0730		RRCA
9CAD 0F	0740		RRCA
9CAE 0F	0750		RRCA
9CAF 0604	0760		LD B, 4
9CB1 1F	0770	LFTBYT	RRA
9CB2 CB1E	0780		RR (HL)
9CB4 CB2E	0790		SRA (HL)
9CB6 10F9	0800		DJNZ LFTBYT
9CB8 7E	0810		LD A, (HL)
9CB9 23	0820		INC HL
9CBA 77	0830		LD (HL), A
9CBB 0607	0840		LD B, 7
9CBD 23	0850	NXT1	INC HL
9CBE 10FD	0860		DJNZ NXT1
9CC0 1A	0870		LD A, (DE)
9CC1 0604	0880		LD B, 4
9CC3 1F	0890	RGTBYT	RRA
9CC4 CB1E	0900		RR (HL)
9CC6 CB2E	0910		SRA (HL)
9CC8 10F9	0920		DJNZ RGTBYT
9CCA 7E	0930		LD A, (HL)
9CCB 23	0940		INC HL
9CCC 77	0950		LD (HL), A
9CCD 0607	0960		LD B, 7
9CCF 2B	0970	NXT2	DEC HL
9CD0 10FD	0980		DJNZ NXT2
9CD2 13	0990		INC DE
9CD3 C1	1000		POP BC
9CD4 10D3	1010		DJNZ NXTROW
9CD6 0608	1020		LD B, 8
9CD8 23	1030	NXT3	INC HL
9CD9 10FD	1040		DJNZ NXT3

9CDB 0D	1050	DEC C
9CDC 20C9	1060	JR NZ,NXTSET
9CDE F1	1070	POP AF
9CDF C9	1080	RET
	1090	END

## CINCO - Impresión Masiva

**Impresión Masiva** aumenta el tamaño de los caracteres, pero a diferente escala que **Caracteres Gigantes**. Cada carácter es 16 veces mayor que los normales, cuando se escribe en la pantalla.

Si echa una ojeada a las dos rutinas, **Caracteres Gigantes** e **Impresión Masiva**, verá muchas diferencias. En efecto, las dos rutinas son completamente diferentes. **Impresión Masiva** es mucho más sencilla de escribir que **Caracteres Gigantes**. La rutina **Impresión Masiva** asigna bloques de dos por dos del carácter CHR\$(143) (bloques sólidos) por cada punto del carácter normal, mientras que la rutina **Caracteres Gigantes** no se puede usar este método de ahorro de tiempo, resultando una rutina más larga y compleja.

```
1      '          IMPRESION MASIVA
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40085
30 READ a$:POKE n,VAL("&" + a$)
40 NEXT
50 DATA CD,93,BB,F5,CD,06,B9,DD,7E,04
60 DATA CD,A5,BB,EB,DD,6E,06,DD,66,08
70 DATA 06,08,C5,0E,80,06,02,C5,E5,CD
80 DATA 75,BB,CB,40,28,05,DD,7E,00,18
90 DATA 03,DD,7E,02,CD,90,BB,E1,06,08
100 DATA 1A,A1,28,04,3E,8F,18,02,3E,20
110 DATA CD,5A,BB,CD,5A,BB,CB,39,10,EC
120 DATA 2C,C1,10,D1,13,C1,10,C8,CD,09
130 DATA B9,F1,CD,90,BB,C9
```

El formato de la CALL es:

CALL 40000,X,Y,CHAR,P1,P2

Los parámetros de la CALL son similares a los de la rutina anterior, con una excepción. Mientras que en **Caracteres Gigantes** se imprimen cadenas de caracteres, en **Impresión Masiva** se le pasa el código ASCII de un carácter. Si quiere usar más de un carácter, llame a la rutina una vez por cada uno que quiere sacar a la pantalla.

La rutina funciona en cualquiera de los tres modos de pantalla y usa dos parámetros de color, de la misma forma que en **Caracteres Gigantes**. Si quiere caracteres de un solo color, ponga el mismo valor en P1 y P2.

```
0001 ;      IMPRESION MASIVA
0002 ;
9C40      0010      ORG  40000
BB93      0020 GETPEN DEFL 0BB93H
BB90      0030 SETPEN DEFL 0BB90H
BBA5      0040 MATRIX DEFL 0BBA5H
BB75      0050 CURSOR DEFL 0BB75H
BB5A      0060 TXTOUT DEFL 0BB5AH
B906      0070 ROMENA DEFL 0B906H
B909      0080 ROMDIS DEFL 0B909H
9C40 CD93BB  0090      CALL GETPEN
9C43 F5      0100      PUSH AF
9C44 CD06B9  0110      CALL ROMENA
9C47 DD7E04  0120      LD  A,(IX+4)
9C4A CDA5BB  0130      CALL MATRIX
9C4D EB      0140      EX  DE,HL
9C4E DD6E06  0150      LD  L,(IX+6)
9C51 DD6608  0160      LD  H,(IX+8)
9C54 0608    0170      LD  B,8
9C56 C5      0180 NXTROW PUSH BC
9C57 0E80    0190      LD  C,128
9C59 0602    0200      LD  B,2
```

9C5B C5	0210	AGAIN	PUSH BC
9C5C E5	0220		PUSH HL
9C5D CD75BB	0230		CALL CURSOR
9C60 CB40	0240		BIT 0, B
9C62 2805	0250		JR Z, COL2
9C64 DD7E00	0260		LD A, (IX+0)
9C67 1803	0270		JR SETCOL
9C69 DD7E02	0280	COL2	LD A, (IX+2)
9C6C CD90BB	0290	SETCOL	CALL SETPEN
9C6F E1	0300		POP HL
9C70 0608	0310		LD B, 8
9C72 1A	0320	NXTCOL	LD A, (DE)
9C73 A1	0330		AND C
9C74 2804	0340		JR Z, SPACE
9C76 3E8F	0350		LD A, 143
9C78 1802	0360		JR PRINT
9C7A 3E20	0370	SPACE	LD A, 32
9C7C CD5ABB	0380	PRINT	CALL TXTOUT
9C7F CD5ABB	0390		CALL TXTOUT
9C82 CB39	0400		SRL C
9C84 10EC	0410		DJNZ NXTCOL
9C86 2C	0420		INC L
9C87 C1	0430		POP BC
9C88 10D1	0440		DJNZ AGAIN
9C8A 13	0450		INC DE
9C8B C1	0460		POP BC
9C8C 10C8	0470		DJNZ NXTROW
9C8E CD09B9	0480		CALL ROMDIS
9C91 F1	0490		POP AF
9C92 CD90BB	0500		CALL SETPEN
9C95 C9	0510		RET
	0520		END

# SEIS - Relleno de la Pantalla

Esta simple rutina rellena toda la pantalla del Amstrad con el carácter ASCII que usted quiera.

```
1          RELLENO DE LA PANTALLA
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40021
30 READ a$:POKE n,VAL("&"a$)
40 NEXT
50 DATA DD,7E,00,01,E8,03,F5,C5,CD,5A
60 DATA BB,C1,0B,78,B1,28,03,F1,18,F2
70 DATA F1,C9
```

El formato de la llamada es bastante sencillo:

```
CALL 40000,A
```

A es el código ASCII del carácter que quiere imprimir en la pantalla y puede ser una variable o un número. Con CALL 40000,66, llenará la pantalla con Bs. La rutina solo funciona en MODE 1.

Aquí tiene una oportunidad de desarrollar un pequeño trabajo de alteración. Si mira el listado del ensamblaje de esta rutina, verá que la línea 40 tiene la instrucción LD BC,1000. Esta instrucción carga el registro BC con el valor 1000, y le dice a la rutina la cantidad de espacios de la pantalla que se van rellenar con el carácter. En modo 1, es de  $40 \times 25 = 1000$ . En modo 0 hay solo 500 ( $20 \times 25$ ) y en modo 2 hay el doble ( $80 \times 25 = 2000$ ). Por lo tanto, poniendo BC=500 podrá rellenar la pantalla en modo 0.

```

0001 ; RELLENO DE LA PANTALLA
0002 ;
9C40 0010 ORG 40000
BB5A 0020 PRNT DEFL 0BB5AH
9C40 DD7E00 0030 LD A, (IX+0)
9C43 01E803 0040 LD BC,1000
9C46 F5 0050 LOOP PUSH AF
9C47 C5 0060 PUSH BC
9C48 CD5ABB 0070 CALL PRNT
9C4B C1 0080 POP BC
9C4C 0B 0090 DEC BC
9C4D 78 0100 LD A,B
9C4E B1 0110 OR C
9C4F 2803 0120 JR Z,END
9C51 F1 0130 POP AF
9C52 18F2 0140 JR L00P
9C54 F1 0150 END POP AF
9C55 C9 0160 RET
0170 END

```

Si tiene un ensamblador, solo tiene que editar la línea 40 y modificarla para que se lea LD BC,500, pero si no lo tiene, le resultará un poco más complicado. Los tres pares de números a la izquierda del número de la línea 40 son los códigos de operación equivalentes al LD BC,1000. Debemos alterar los dos últimos pares para obtener el número que nosotros queremos. Usaremos el comando PRINT HEX\$(500) para obtener su valor hexadecimal, F4 01. Volviendo al listado BASIC, alteraremos los datos apropiados (los números quinto y sexto) con F4 01 y volveremos a ejecutar el programa, después de haberlo salvado a cinta o a disco. El programa debe funcionar correctamente en modo 0. Una vez echo esto, lo podrá alterar fácilmente para trabajar en modo 2 o para imprimir solamente unas cuantas líneas de un carácter determinado en un modo fijado.

# SIETE - LOAD/SAVE sin Cabecera

Esta rutina está destinada solamente a los ordenadores Amstrad que utilizan cinta, como el CPC464.

Si tiene un 464 y desea cargar un programa largo, habrá observado que en cada fichero carga un gran número de bloques. Cada bloque va precedido por una cabecera que contiene información sobre las direcciones de comienzo y la longitud de los datos.

Este programa elimina las cabeceras, haciendo que se cargue y se grabe el programa como un gran bloque, acortando en gran medida el tiempo que se necesita para cargarlo y grabarlo. Una vez que se llama a la rutina, no lanza ningún mensaje.

```
1          SIN CABECERA
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40167
30 READ a$:POKE n,VAL("&" + a$)
40 NEXT
50 DATA FE,03,28,16,21,C0,9C,FE,02,20
60 DATA 5F,AF,DD,5E,00,DD,56,01,DD,6E
70 DATA 02,DD,66,03,18,0F,DD,7E,00,DD
80 DATA 5E,02,DD,56,03,DD,6E,04,DD,66
90 DATA 05,CD,9E,BC,30,36,C9,FE,03,28
100 DATA 16,21,C0,9C,FE,02,20,30,AF,DD
110 DATA 5E,00,DD,56,01,DD,6E,02,DD,66
120 DATA 03,18,0F,DD,7E,00,DD,5E,02,DD
130 DATA 56,03,DD,6E,04,DD,66,05,CD,A1
140 DATA BC,D8,B7,28,05,21,D3,9C,18,06
150 DATA CD,03,B8,21,DF,9C,7E,23,CB,7F
160 DATA 20,05,CD,5A,BB,18,F5,CB,BF,CD
170 DATA 5A,BB,3E,07,CD,5A,BB,C9,2A,46
```

```

180 DATA 41,4C,54,41,20,50,41,52,41,4D
190 DATA 45,54,52,4F,AA,2A,45,52,52,4F
200 DATA 52,20,4C,45,43,54,55,52,41,AA
210 DATA 2A,45,53,43,41,50,45,AA

```

El formato de la CALL es:

Para SAVE:

```
CALL 40000,S,L,SYNC
```

Para LOAD:

```
CALL 40047,S,L,SYNC
```

S es la dirección de comienzo y L la longitud del programa, ya que no disponemos de cabeceras para almacenar esta información. La variable SYNCH la usa el Amstrad para distinguir la cabecera (carácter 44) de los datos (representados por un valor SYNCH de 22). Como no tenemos cabeceras, el valor incluido no es importante, excepto que para el LOAD se debe usar el mismo valor que se usó para el SAVE

```

0001 ;          SIN CABECERA
0002 ;
9C40      0010      ORG 40000
0020 ;          RUTINA DE SAVE
BC9E      0030 SAVE  DEFL 0BC9EH
BCA1      0040 LOAD  DEFL 0BCA1H
BB5A      0050 TXTOUT DEFL 0BB5AH
BB03      0060 KRESET DEFL 0BB03H
9C40 FE03   0070      CP    3
9C42 2816   0080      JR    Z,SVSYNC
9C44 21C09C 0090      LD    HL,PARAM
9C47 FE02   0100      CP    2
9C49 205F   0110      JR    NZ,ERROR

```

9C4B	AF	0120	XOR	A
9C4C	DD5E00	0130	LD	E, (IX+0)
9C4F	DD5601	0140	LD	D, (XI+1)
9C52	DD6E02	0150	LD	L, (IX+2)
9C55	DD6603	0160	LD	H, (XI+3)
9C58	180F	0170	JR	SVESET
9C5A	DD7E00	0180	SVSYNC LD	A, (IX+0)
9C5D	DD5E02	0190	LD	E, (IX+2)
9C60	DD5603	0200	LE	D, (IX+3)
9C63	DD6E04	0210	LD	L, (IX+4)
9C66	DD6605	0220	LD	H, (IX+5)
9C69	CD9EBC	0230	SVESET CALL	SAVE
9C6C	3036	0240	JR	NC, ESC
9C6E	C9	0250	RET	
		0260	;	
		0270	;	RUTINA DE LOAD
9C6F	FE03	0280	CP	3
9C71	2816	0290	JR	Z, LDSYNC
9C73	21C09C	0300	LD	HL, PARAM
9C76	FE02	0310	CP	2
9C78	2030	0320	JR	NZ, ERROR
9C7A	AF	0330	XOR	A
9C7B	DD5E00	0340	LD	E, (IX+0)
9C7E	DD5601	0350	LD	D, (IX+1)
9C81	DD6E02	0360	LD	L, (IX+2)
9C84	DD6603	0370	LD	H, (IX+3)
9C87	180F	0380	JR	LDSET
9C89	DD7E00	0390	LDSYNC LD	A, (IX+0)
9C8C	DD5E02	0400	LD	E, (IX+2)
9C8F	DD5603	0410	LD	D, (IX+3)
9C92	DD6E04	0420	LD	L, (IX+4)
9C95	DD6605	0430	LD	H, (IX+5)
9C98	CDA1BC	0440	LDSET CALL	LOAD
9C9B	D8	0450	RET	C

9C9C B7	0460	OR	A
9C9D 2805	0470	JR	Z, ESC
9C9F 21D39C	0480	LD	HL, TAPERR
9CA2 1806	0490	JR	ERROR
9CA4 CD03BB	0500	ESC CALL	KRESET
9CA7 21DF9C	0510	LD	HL, ESCAPE
9CAA 7E	0520	ERROR LD	A, (HL)
9CAB 23	0530	INC	HL
9CAC CB7F	0540	BIT	7, A
9CAE 2005	0550	JR	NZ, ENDERR
9CB0 CD5ABB	0560	CALL	TXTOUT
9CB3 18F5	0570	JR	ERROR
9CB5 CBBF	0580	ENDERR RES	7, A
9CB7 CD5ABB	0590	CALL	TXTOUT
9CBA 3E07	0600	LD	A, 7
9CBC CD5ABB	0610	CALL	TXTOUT
9CBF C9	0620	RET	
9CC0	0630	PARAM DEFM	"*FALTA"
9CC6	0640	DEFM	" PARAMETRO"
9CD0 AA	0650	DEFB	"*"+80H
9CD1	0660	TAPERR DEFM	"*ERROR"
9CD7	0670	DEFM	" LECTURA"
9CDF AA	0680	DEFB	"*"+80H
9CE0	0690	ESCAPE DEFM	"*ESCAPE"
9CE7 AA	0700	DEFB	"*"+80H
	0710	END	

## OCHO - Música por Interrupciones

Muchos de los juegos comerciales hacen sonar una música continuamente mientras se juega. Hemos querido producir un efecto similar para que lo pueda incluir usted en sus propios juegos. El resultado es este programa.

Este programa es uno de los más largos del libro. El almacenamiento de los datos de la música es una labor que consume mucha memoria. Aún así, la rutina completa (incluida la música) ocupa solamente 590 octetos, dejándole aún 43K para su juego. Es necesario usar la rutina en código máquina junto con unas pocas instrucciones BASIC:

```
10 CALL 40000,0
20 EVERY 6 GOSUB 13000

... Resto del Juego ...

13000 CALL 40000:RETURN
```

La primera CALL debe ir seguida por un número cualquiera (aquí hemos elegido 0; realmente no importa el valor que se use). Esto inicia la rutina y prepara los datos para ejecutar. La segunda línea utiliza el comando de BASIC EVERY para controlar las interrupciones del sistema, por lo tanto le sugiero que mire en el manual del Amstrad, capítulos 9.3 y 10. Cada 6/50 de segundo, el programa saltará a la subrutina por medio de la línea 13000 (puede colocar esta subrutina en el lugar que usted desee). La CALL que tiene la subrutina solo ejecuta la siguiente nota de la canción, y después vuelve a continuar con el programa normal.

Se pueden cambiar los 6/50 de segundo por cualquier tiempo que se desee, esto es solo un ejemplo. El sonido usa solamente el canal 1 y el ENV de volumen, la envolvente 15. Esta envolvente puede ser redefinida en el BASIC para crear efectos extraños. Se puede parar la música inhabilitando las interrupciones. Por suerte tenemos un comando BASIC que lo

puede hacer, se llama DI. El comando opuesto es EI (Habilita las interrupciones).

```
1      '          MUSICA POR INTERRUPCIONES
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40590
30 READ a$:POKE n,VAL("&"a$).
40 NEXT
50 DATA B7,28,14,21,95,9C,22,5D,9C,3E
60 DATA 01,32,84,9C,3E,0F,21,85,9C,CD
70 DATA BC,BC,C9,21,84,9C,35,C0,11,95
80 DATA 9C,1A,B7,C8,13,77,3C,20,09,1A
90 DATA 77,21,95,9C,22,5D,9C,C9,1A,6F
100 DATA 13,1A,67,13,ED,53,5D,9C,22,8F
110 DATA 9C,21,8C,9C,CD,AA,BC,C9,00,02
120 DATA 03,05,01,0F,FF,0A,81,0F,00,00
130 DATA 00,00,00,00,00
140 REM   DATOS DE LA MUSICA
150 DATA 02,AA,01,02,92
160 DATA 01,01,7B,01,02,EF,00,01,7B,01
170 DATA 02,EF,00,01,7B,01,06,EF,00,01
180 DATA EF,00,01,D5,00,01,C9,00,01,BE
190 DATA 00,01,EF,00,01,D5,00,02,BE,00
200 DATA 01,FD,00,02,D5,00,06,EF,00,02
210 DATA AA,01,02,92,01,01,7B,01,02,EF
220 DATA 00,01,7B,01,02,EF,00,01,7B,01
230 DATA 06,EF,00,01,1C,01,01,3F,01,01
240 DATA 52,01,01,1C,01,01,EF,00,02,BE
250 DATA 00,01,D5,00,01,EF,00,01,1C,01
260 DATA 06,D5,00,02,AA,01,02,92,01,01
270 DATA 7B,01,02,EF,00,01,7B,01,02,EF
280 DATA 00,01,7B,01,06,EF,00,01,EF,00
290 DATA 01,D5,00,01,C9,00,01,BE,00,01
300 DATA EF,00,01,D5,00,02,BE,00,01,FD
```

310 DATA 00,02,D5,00,06,EF,00,01,EF,00  
320 DATA 01,D5,00,01,BE,00,01,EF,00,01  
330 DATA D5,00,02,BE,00,01,EF,00,01,D5  
340 DATA 00,01,EF,00,01,BE,00,01,EF,00  
350 DATA 01,D5,00,02,BE,00,01,EF,00,01  
360 DATA D5,00,01,EF,00,01,BE,00,01,EF  
370 DATA 00,01,D5,00,02,BE,00,01,FD,00  
380 DATA 02,D5,00,06,EF,00,01,7B,01,01  
390 DATA 66,01,01,52,01,02,3F,01,01,1C  
400 DATA 01,02,3F,01,01,7B,01,01,66,01  
410 DATA 01,52,01,02,3F,01,01,1C,01,02  
420 DATA 3F,01,01,BE,00,01,EF,00,01,3F  
430 DATA 01,01,1C,01,01,FD,00,01,EF,00  
440 DATA 01,D5,00,01,BE,00,01,D5,00,01  
450 DATA EF,00,01,D5,00,05,3F,01,01,3F  
460 DATA 01,01,7B,01,01,66,01,02,3F,01  
470 DATA 01,1C,01,02,3F,01,01,7B,01,01  
480 DATA 66,01,01,52,01,02,3F,01,01,1C  
490 DATA 01,02,3F,01,01,3F,01,01,1C,01  
500 DATA 01,0C,01,01,FD,00,02,FD,00,02  
510 DATA FD,00,01,1C,01,01,52,01,01,AA  
520 DATA 01,05,3F,01,01,7B,01,01,66,01  
530 DATA 01,52,01,02,3F,01,01,1C,01,02  
540 DATA 3F,01,01,7B,01,01,66,01,01,52  
550 DATA 01,02,3F,01,01,1C,01,02,3F,01  
560 DATA 01,BE,00,01,EF,00,01,3F,01,01  
570 DATA 1C,01,01,FD,00,01,EF,00,01,D5  
580 DATA 00,01,BE,00,01,D5,00,01,EF,00  
590 DATA 01,D5,00,05,EF,00,01,3F,01,01  
600 DATA 52,01,01,3F,01,02,EF,00,01,1C  
610 DATA 01,02,EF,00,01,1C,01,01,EF,00  
620 DATA 01,1C,01,01,3F,01,01,EF,00,01  
630 DATA BE,00,02,9F,00,01,BE,00,01,EF  
640 DATA 00,01,3F,01,02,1C,01,02,EF,00

650 DATA 01,BE,00,03,D5,00,06,EF,00,FF

660 DATA 01

Cada nota de la canción consta de tres octetos de datos. El primero especifica la longitud de la nota:

- 1 = semicorchea
- 2 = corchea
- 3 = corchea con puntillo
- 4 = negra
- 6 = negra con puntillo
- 8 = blanca
- 12 = blanca con puntillo

El tono de la nota puede estar, teóricamente entre 1 y 4000, pero en la práctica usted usará seguramente un rango entre 50 y 1300. Esto significa que ha de usar dos octetos para contener el valor. El octeto dos contiene el octeto bajo de la nota y el octeto tres contiene el alto. Esto corresponde a la fórmula:

$$\text{Tono} = \text{Octeto Dos} + 256 * \text{Octeto Tres}$$

Los datos de la música comienzan en la línea 150 del cargador de BASIC. Es más sencillo usar el ensamblador para introducir los datos de la música, pero si no tiene uno, puede obtener los valores hexadecimales de cada octeto, e introducirlos en el cargador BASIC. Tenga siempre en cuenta que el octeto inferior precede siempre al superior.

Hay dos valores de longitud que tienen un efecto especial. Cero marca el final de los datos y dejará de ejecutar la música aunque siga llamándose a la subrutina. El segundo valor especial es 255,n que le dice al programa que repita la canción después de esperar n, donde n es comparable al valor de una de las notas (vea la tabla anterior).

```

0001 ;   MUSICA POR INTERRUPCIONES
0002 ;
9C40      0010      ORG  40000
BCBC      0020  AMPENV  DEFL  0BCBCH
BCAA      0030  ADDSND  DEFL  0BCAAH
9C40 B7      0040      OR   A
9C41 2814     0050      JR   Z, NXTSND
9C43 21959C   0060      LD   HL, MUSIC
9C46 225D9C   0070      LD   (DATADR+1), HL
9C49 3E01     0080      LD   A, 1
9C4B 32849C   0090      LD   (TIME), A
9C4E 3E0F     0100      LD   A, 15
9C50 21859C   0110      LD   HL, AMPL
9C53 CDBCBC   0120      CALL AMPENV
9C56 C9       0130      RET
9C57 21849C   0140  NXTSND  LD   HL, TIME
9C5A 35       0150      DEC  (HL)
9C5B C0       0160      RET  NZ
9C5C 11959C   0170  DATADR  LD   DE, MUSIC
9C5F 1A       0180      LD   A, (DE)
9C60 B7       0190      OR   A
9C61 C8       0200      RET  Z
9C62 13       0210      INC  DE
9C63 77       0220      LD   (HL), A
9C64 3C       0230      INC  A
9C65 2009     0240      JR   NZ, CONT
9C67 1A       0250      LD   A, (DE)
9C68 77       0260      LD   (HL), A
9C69 21959C   0270      LD   HL, MUSIC
9C6C 225D9C   0280      LD   (DATADR+1), HL
9C6F C9       0290      RET
9C70 1A       0300  CONT  LD   A, (DE)
9C71 6F       0310      LD   L, A
9C72 13       0320      INC  DE

```

9C73	1A	0330	LD	A, (DE)
9C74	67	0340	LD	H, A
9C75	13	0350	INC	DE
9C76	ED535D9C	0360	LD	(DATADR+1), DE
9C7A	228F9C	0370	LD	(TONE), HL
9C7D	218C9C	0380	LD	HL, SOUND
9C80	CDAABC	0390	CALL	ADDSDND
9C83	C9	0400	RET	
9C84	00	0410	TIME	DEFB 0
9C85	02	0420	AMPL	DEFB 2, 3, 5, 1
	03	05	01	
9C89	0F	0430		DEFB 15, 255, 10
	FF	0A		
9C8C	81	0440	SOUND	DEFB 129, 15, 0
	0F	00		
9C8F	00	0450	TONE	DEFB 0, 0, 0, 0, 0, 0
	00	00	00	00 00
9C95	02	0460	MUSIC	DEFB 2
9C96	AA01	0470		DEFW 426
9C98	02	0480		DEFB 2
9C9A	9201	0490		DEFW 402
9C9B	01	0500		DEFB 1
9C9D	7B01	0510		DEFW 379
9C9E	02	0520		DEFB 2
9C9F	EF00	0530		DEFW 239
9CA1	01	0540		DEFB 1
9CA3	7B01	0550		DEFW 379
9CA4	02	0560		DEFB 2
9CA5	EF00	0570		DEFW 239
9CA7	01	0580		DEFB 1
9CA8	7B01	0590		DEFW 379
9CAA	06	0600		DEFB 6
9CAB	EF00	0610		DEFW 239
9CAD	01	0620		DEFB 1

9CAE EF00	0630	DEFW 239
9CB0 01	0640	DEFB 1
9CB1 D500	0650	DEFW 213
9CB3 01	0660	DEFB 1
9CB4 C900	0670	DEFW 201
9CB6 01	0680	DEFB 1
9CB7 BE00	0690	DEFW 190
9CB9 01	0700	DEFB 1
9CBA EF00	0710	DEFW 239
9CBC 01	0720	DEFB 1
9CBD D500	0730	DEFW 213
9CBF 02	0740	DEFB 2
9CC0 BE00	0750	DEFW 190
9CC2 01	0760	DEFB 1
9CC3 FD00	0770	DEFW 253
9CC5 02	0780	DEFB 2
9CC6 D500	0790	DEFW 213
9CC8 06	0800	DEFB 6
9CC9 EF00	0810	DEFW 239
9CCB 02	0820	DEFB 2
9CCC AA01	0830	DEFW 426
9CCE 02	0840	DEFB 2
9CCF 9201	0850	DEFW 402
9CD1 01	0860	DEFB 1
9CD2 7B01	0870	DEFW 379
9CD4 02	0880	DEFB 2
9CD5 EF00	0890	DEFW 239
9CD7 01	0900	DEFB 1
9CD8 7B01	0910	DEFW 379
9CDA 02	0920	DEFB 2
9CDB EF00	0930	DEFW 239
9CDD 01	0940	DEFB 1
9CDE 7B01	0950	DEFW 379
9CE0 06	0960	DEFB 6

9CE1 EF00	0970	DEFW 239
9CE3 01	0980	DEFB 1
9CE4 1C01	0990	DEFW 284
9CE6 01	1000	DEFB 1
9CE7 3F01	1010	DEFW 319
9CE9 01	1020	DEFB 1
9CEA 5201	1030	DEFW 338
9CEC 01	1040	DEFB 1
9CED 1C01	1050	DEFW 284
9CEF 01	1060	DEFB 1
9CF0 EF00	1070	DEFW 239
9CF2 02	1080	DEFB 2
9CF3 BE00	1090	DEFW 190
9CF5 01	1100	DEFB 1
9CF6 D500	1110	DEFW 213
9CF8 01	1120	DEFB 1
9CF9 EF00	1130	DEFW 239
9CFB 01	1140	DEFB 1
9CFC 1C01	1150	DEFW 284
9CFE 06	1160	DEFB 6
9CFF D500	1170	DEFW 213
9D01 02	1180	DEFB 2
9D02 AA01	1190	DEFW 426
9D04 02	1200	DEFB 2
9D05 9201	1210	DEFW 402
9D07 01	1220	DEFB 1
9D08 7B01	1230	DEFW 379
9D0A 02	1240	DEFB 2
9D0B EF00	1250	DEFW 239
9D0D 01	1260	DEFB 1
9D0E 7B01	1270	DEFW 379
9D10 02	1280	DEFB 2
9D11 EF00	1290	DEFW 239
9D13 01	1300	DEFB 1

9D14 7B01	1310	DEFW 379
9D16 06	1320	DEFB 6
9D17 EF00	1330	DEFW 239
9D19 01	1340	DEFB 1
9D1A EF00	1350	DEFW 239
9D1C 01	1360	DEFB 1
9D1D D500	1370	DEFW 213
9D1F 01	1380	DEFB 1
9D20 C900	1390	DEFW 201
9D22 01	1400	DEFB 1
9D23 BE00	1410	DEFW 190
9D25 01	1420	DEFB 1
9D26 EF00	1430	DEFW 239
9D28 01	1440	DEFB 1
9D29 D500	1450	DEFW 213
9D2B 02	1460	DEFB 2
9D2C BE00	1470	DEFW 190
9D2E 01	1480	DEFB 1
9D2F FD00	1490	DEFW 253
9D31 02	1500	DEFB 2
9D32 D500	1510	DEFW 213
9D34 06	1520	DEFB 6
9D35 EF00	1530	DEFW 239
9D37 01	1540	DEFB 1
9D38 EF00	1550	DEFW 239
9D3A 01	1560	DEFB 1
9D3B D500	1570	DEFW 213
9D3D 01	1580	DEFB 1
9D3E BE00	1590	DEFW 190
9D40 01	1600	DEFB 1
9D41 EF00	1610	DEFW 239
9D43 01	1620	DEFB 1
9D44 D500	1630	DEFW 213
9D46 02	1640	DEFB 2

9D47 BE00	1650	DEFW 190
9D49 01	1660	DEFB 1
9D4A EF00	1670	DEFW 239
9D4C 01	1680	DEFB 1
9D4D D500	1690	DEFW 213
9D4F 01	1700	DEFB 1
9D50 EF00	1710	DEFW 239
9D52 01	1720	DEFB 1
9D53 BE00	1730	DEFW 190
9D55 01	1740	DEFB 1
9D56 EF00	1750	DEFW 239
9D58 01	1760	DEFB 1
9D59 D500	1770	DEFW 213
9D5B 02	1780	DEFB 2
9D5C BE00	1790	DEFW 190
9D5E 01	1800	DEFB 1
9D5F EF00	1810	DEFW 239
9D61 01	1820	DEFB 1
9D62 D500	1830	DEFW 213
9D64 01	1840	DEFB 1
9D65 EF00	1850	DEFW 239
9D67 01	1860	DEFB 1
9D68 BE00	1870	DEFW 190
9D6A 01	1880	DEFB 1
9D6B EF00	1890	DEFW 239
9D6D 01	1900	DEFB 1
9D6E D500	1910	DEFW 213
9D70 02	1920	DEFB 2
9D71 BE00	1930	DEFW 190
9D73 01	1940	DEFB 1
9D74 FD00	1950	DEFW 253
9D76 02	1960	DEFB 2
9D77 D500	1970	DEFW 213
9D79 06	1980	DEFB 6

9D7A EF00	1990	DEFW 239
9D7C 01	2000	DEFB 1
9D7D 7B01	2010	DEFW 379
9D7F 01	2020	DEFB 1
9D80 6601	2030	DEFW 358
9D82 01	2040	DEFB 1
9D83 5201	2050	DEFW 338
9D85 02	2060	DEFB 2
9D86 3F01	2070	DEFW 319
9D88 01	2080	DEFB 1
9D89 1C01	2090	DEFW 284
9D8B 02	2100	DEFB 2
9D8C 3F01	2110	DEFW 319
9D8E 01	2120	DEFB 1
9D8F 7B01	2130	DEFW 379
9D91 01	2140	DEFB 1
9D92 6601	2150	DEFW 358
9D94 01	2160	DEFB 1
9D95 5201	2170	DEFW 338
9D97 02	2180	DEFB 2
9D98 3F01	2190	DEFW 319
9D9A 01	2200	DEFB 1
9D9B 1C01	2210	DEFW 284
9D9D 02	2220	DEFB 2
9D9E 3F01	2230	DEFW 319
9DA0 01	2240	DEFB 1
9DA1 BE00	2250	DEFW 190
9DA3 01	2260	DEFB 1
9DA4 EF00	2270	DEFW 239
9DA6 01	2280	DEFB 1
9DA7 3F01	2290	DEFW 319
9DA9 01	2300	DEFB 1
9DAA 1C01	2310	DEFW 284
9DAC 01	2320	DEFB 1

9DAD FD00	2330	DEFW 253
9DAF 01	2340	DEFB 1
9DB0 EF00	2350	DEFW 239
9DB2 01	2360	DEFB 1
9DB3 D500	2370	DEFW 213
9DB5 01	2380	DEFB 1
9DB6 BE00	2390	DEFW 190
9DB8 01	2400	DEFB 1
9DB9 D500	2410	DEFW 213
9DBB 01	2420	DEFB 1
9DBC EF00	2430	DEFW 239
9DBE 01	2440	DEFB 1
9DBF D500	2450	DEFW 213
9DC1 05	2460	DEFB 5
9DC2 3F01	2470	DEFW 319
9DC4 01	2480	DEFB 1
9DC5 3F01	2490	DEFW 319
9DC7 01	2500	DEFB 1
9DC8 7B01	2510	DEFW 379
9DCA 01	2520	DEFB 1
9DCB 6601	2530	DEFW 358
9DCD 02	2540	DEFB 2
9DCE 3F01	2550	DEFW 319
9DD0 01	2560	DEFB 1
9DD1 1C01	2570	DEFW 284
9DD3 02	2580	DEFB 2
9DD4 3F01	2590	DEFW 319
9DD6 01	2600	DEFB 1
9DD7 7B01	2610	DEFW 379
9DD9 01	2620	DEFB 1
9DDA 6601	2630	DEFW 358
9DDC 01	2640	DEFB 1
9DDD 5201	2650	DEFW 338
9DDF 02	2660	DEFB 2

9DE0 3F01	2670	DEFW 319
9DE2 01	2680	DEFB 1
9DE3 1C01	2690	DEFW 284
9DE5 02	2700	DEFB 2
9DE6 3F01	2710	DEFW 319
9DE8 01	2720	DEFB 1
9DE9 3F01	2730	DEFW 319
9DEB 01	2740	DEFB 1
9DEC 1C01	2750	DEFW 284
9DEE 01	2760	DEFB 1
9DEF 0C01	2770	DEFW 268
9DF1 01	2780	DEFB 1
9DF2 FD00	2790	DEFW 253
9DF4 02	2800	DEFB 2
9DF5 FD00	2810	DEFW 253
9DF7 02	2820	DEFB 2
9DF8 FD00	2830	DEFW 253
9DFA 01	2840	DEFB 1
9DFB 1C01	2850	DEFW 284
9DFD 01	2860	DEFB 1
9DFE 5201	2870	DEFW 338
9E00 01	2880	DEFB 1
9E01 AA01	2890	DEFW 426
9E03 05	2900	DEFB 5
9E04 3F01	2910	DEFW 319
9E06 01	2920	DEFB 1
9E07 7B01	2930	DEFW 379
9E09 01	2940	DEFB 1
9E0A 6601	2950	DEFW 358
9E0C 01	2960	DEFB 1
9E0D 5201	2970	DEFW 338
9E0F 02	2980	DEFB 2
9E10 3F01	2990	DEFW 319
9E12 01	3000	DEFB 1

9E13 1C01	3010	DEFW 284
9E15 02	3020	DEFB 2
9E16 3F01	3030	DEFW 319
9E18 01	3040	DEFB 1
9E19 7B01	3050	DEFW 379
9E1B 01	3060	DEFB 1
9E1C 6601	3070	DEFW 358
9E1E 01	3080	DEFB 1
9E1F 5201	3090	DEFW 338
9E21 02	3100	DEFB 2
9E22 3F01	3110	DEFW 319
9E24 01	3120	DEFB 1
9E25 1C01	3130	DEFW 284
9E27 02	3140	DEFB 2
9E28 3F01	3150	DEFW 319
9E2A 01	3160	DEFB 1
9E2B BE00	3170	DEFW 190
9E2D 01	3180	DEFB 1
9E2E EF00	3190	DEFW 239
9E30 01	3200	DEFB 1
9E31 3F01	3210	DEFW 319
9E33 01	3220	DEFB 1
9E34 1C01	3230	DEFW 284
9E36 01	3240	DEFB 1
9E37 FD00	3250	DEFW 253
9E39 01	3260	DEFB 1
9E3A EF00	3270	DEFW 239
9E3C 01	3280	DEFB 1
9E3D D500	3290	DEFW 213
9E3F 01	3300	DEFB 1
9E40 BE00	3310	DEFW 190
9E42 01	3320	DEFB 1
9E43 D500	3330	DEFW 213
9E45 01	3340	DEFB 1

9E46 EF00	3350	DEFW 239
9E48 01	3360	DEFB 1
9E49 D500	3370	DEFW 213
9E4B 05	3380	DEFB 5
9E4C EF00	3390	DEFW 239
9E4E 01	3400	DEFB 1
9E4F 3F01	3410	DEFW 319
9E51 01	3420	DEFB 1
9E52 5201	3430	DEFW 338
9E54 01	3440	DEFB 1
9E55 3F01	3450	DEFW 319
9E57 02	3460	DEFB 2
9E58 EF00	3470	DEFW 239
9E5A 01	3480	DEFB 1
9E5B 1C01	3490	DEFW 284
9E5D 02	3500	DEFB 2
9E5E EF00	3510	DEFW 239
9E60 01	3520	DEFB 1
9E61 1C01	3530	DEFW 284
9E63 01	3540	DEFB 1
9E64 EF00	3550	DEFW 239
9E66 01	3560	DEFB 1
9E67 1C01	3570	DEFW 284
9E69 01	3580	DEFB 1
9E6A 3F01	3590	DEFW 319
9E6C 01	3600	DEFB 1
9E6D EF00	3610	DEFW 239
9E6F 01	3620	DEFB 1
9E70 BE00	3630	DEFW 190
9E72 02	3640	DEFB 2
9E73 9F00	3650	DEFW 159
9E75 01	3660	DEFB 1
9E76 BE00	3670	DEFW 190
9E78 01	3680	DEFB 1

9E79 EF00	3690	DEFW 239
9E7B 01	3700	DEFB 1
9E7C 3F01	3710	DEFW 319
9E7E 02	3720	DEFB 2
9E7F 1C01	3730	DEFW 284
9E81 02	3740	DEFB 2
9E82 EF00	3750	DEFW 239
9E84 01	3760	DEFB 1
9E85 BE00	3770	DEFW 190
9E87 03	3780	DEFB 3
9E88 D500	3790	DEFW 213
9E8A 06	3800	DEFB 6
9E8B EF00	3810	DEFW 239
9E8D FF	3820	DEFB 255,1
01		
	3830	END

# NUEVE - Monitor de Código Máquina

Este programa le permitirá examinar el contenido de la memoria de su Amstrad. Puede ver en la pantalla el contenido de su RAM y ROM. Además, puede alterar el contenido de las posiciones de memoria, haciendo que el programa sea algo más que un simple "analizador de memoria".

Los comandos disponibles son:

- '3' = Avance rápido por la memoria.
- 'E' = Avance lento por la memoria.
- 'D' = Retroceso lento por la memoria.
- 'X' = Retroceso rápido por la memoria.
- 'Q' = Salir. (No puede usar ESC para cortar el programa)
- 'ENTER' para introducir un número en una dirección.

Hay varias cosas que debe tener en cuenta cuando pulse la tecla ENTER. La dirección que se puede alterar cuando se pulsa la tecla ENTER es la que está en la línea que aparece en lo alto de la pantalla. Segundo, tenga mucho cuidado cuando intente alterar algún valor, particularmente entre 40000 y 40276, ya que es donde está almacenado el programa monitor.

```
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 CLS
30 monitor=40000:scroll=40034
40 LOCATE 11,2:PRINT "Monitor de Código Máquina"
50 IF PEEK(40000)=&DD AND PEEK(40001)=&E6 GOTO 90
60 FOR n=40000 TO 40276
70 READ a$:POKE n,VAL("&" + a$)
```

```

80 NEXT
90 LOCATE 11,6:INPUT "Dirección de comienzo";st
100 LOCATE 11,8:PRINT "Para examinar direcciones"
110 LOCATE 11,9:PRINT "&0000-&4000 y &C000-&FFFF"
120 LOCATE 11,10:PRINT "(A) ROM o (B) RAM?"
130 rom%=0
140 IF INKEY(69)=0 THEN rom%=1:GOTO 160
150 IF INKEY(54)<>0 GOTO 140
160 start%=INT(UNT(st))-25
170 POKE scroll1,25
180 CALL monitor,@rom%,@start%
190 IF rom%=99 THEN CLS:END
200 LOCATE 12,1:PRINT "> <":LOCATE 13,1
210 GOSUB 260:h$=a$
220 GOSUB 260:l$=a$
230 IF INKEY(57)*INKEY(58)*INKEY(61)=0 GOTO 230
240 POKE start%,VAL("&"+h$+l$)
250 POKE scroll1,1:GOTO 180
260 a$=UPPER$(INKEY$):IF a$<"0" OR a$>"F" OR a$>"9"
    AND a$<"A" GOTO 260
270 PRINT a$;
280 RETURN
290 DATA DD,6E,00,DD,66,01,5E,23,56,DD
300 DATA 6E,02,DD,66,03,7E,B7,28,08,CD
310 DATA 00,B9,CD,06,B9,18,06,CD,03,B9
320 DATA CD,09,B9,06,19,C5,3E,19,CD,D1
330 DATA 9C,C1,10,F7,3E,39,CD,1E,BB,28
340 DATA 07,3E,19,CD,D1,9C,18,F2,3E,3F
350 DATA CD,1E,BB,28,07,3E,01,CD,D1,9C
360 DATA 18,E4,3E,3A,CD,1E,BB,28,07,3E
370 DATA 19,CD,C7,9C,18,D6,3E,3D,CD,1E
380 DATA BB,28,07,3E,01,CD,C7,9C,18,C8
390 DATA 3E,12,CD,1E,BB,20,0F,3E,43,CD
400 DATA 1E,BB,28,BA,DD,6E,02,DD,66,03

```

410 DATA 36,63,DD,6E,00,DD,66,01,73,23  
420 DATA 72,CD,03,BB,C9,F5,01,00,50,0B  
430 DATA 78,B1,20,FB,F1,F5,DS,47,CB,38  
440 DATA CB,38,CB,38,CB,38,AF,CD,4D,BC  
450 DATA 3E,1F,CD,5A,BB,3E,05,CD,5A,BB  
460 DATA D1,F1,F5,CD,5A,BB,FE,01,28,07  
470 DATA 13,21,18,00,19,18,03,1B,62,6B  
480 DATA 44,CD,34,9D,45,CD,34,9D,3E,1F  
490 DATA CD,5A,BB,3E,0D,CD,5A,BB,F1,F5  
500 DATA CD,5A,BB,46,CD,34,9D,C1,7E,FE  
510 DATA 21,F8,FE,7F,F0,3E,1F,CD,5A,BB  
520 DATA 3E,13,CD,5A,BB,78,CD,5A,BB,7E  
530 DATA CD,5A,BB,C9,78,E6,0F,4F,CB,38  
540 DATA CB,38,CB,38,CB,38,78,06,02,FE  
550 DATA 0A,F2,4C,9D,C6,30,18,02,C6,37  
560 DATA CD,5A,BB,79,10,EF,C9

Observe que los 64 primeros octetos de la ROM son idénticos a los 64 primeros octetos de la RAM. Esto se debe a que estos primeros octetos constituyen el bloque de salto a rutinas y se copia desde la ROM a la RAM cuando se enciende la máquina.

Cuando ejecute el programa, éste le pedirá la dirección desde la que quiere comenzar. Introdúzcala en decimal o en hexadecimal. Si usa hex, la dirección debe ir precedida por el carácter '&'. Como puede ver en el listado que proporciona el programa en acción, todas las direcciones y valores se muestran en hexadecimal.

Monitor de Código Máquina

Dirección de comienzo? &742

Para examinar direcciones

&0000-&4000 y &C000-&FFFF

(A) ROM o (B) RAM?

0742	53	S
0743	63	c
0744	68	h
0745	6E	n
0746	65	e
0747	69	i
0748	64	d
0749	65	e
074A	72	r
074B	00	
074C	0A	
074D	20	
074E	41	A
074F	77	w
0750	61	a
0751	00	
0752	0A	
0753	20	
0754	53	S
0755	6F	o
0756	6C	l
0757	61	a
0758	76	v
0759	6F	o
075A	78	x

```

0001 ;      MONITOR C/M
0002 ;
9C40      0010      ORG  40000
B900      0020 UROMON DEFL 0B900H
B903      0030 UROMOF DEFL 0B903H
B906      0040 LROMON DEFL 0B906H
B909      0050 LROMOF DEFL 0B909H
BB1E      0060 TSTKEY DEFL 0BB1EH
BB03      0070 KRESET DEFL 0BB03H
BC4D      0080 SCROLL DEFL 0BC4DH
BB5A      0090 TXTOUT DEFL 0BB5AH
9C40 DD6E00 0100      LD   L, (IX+0)
9C43 DD6601 0110      LD   H, (IX+1)
9C46 5E     0120      LD   E, (HL)
9C47 23     0130      INC  HL
9C48 56     0140      LD   D, (HL)
9C49 DD6E02 0150      LD   L, (IX+2)
9C4C DD6603 0160      LD   H, (IX+3)
9C4F 7E     0170      LD   A, (HL)
9C50 B7     0180      OR   A
9C51 2808   0190      JR   Z, ROMOFF
9C53 CD00B9 0200      CALL UROMON
9C56 CD06B9 0210      CALL LROMON
9C59 1806   0220      JR   ROMSET
9C5B CD03B9 0230 ROMOFF CALL UROMOF
9C5E CD09B9 0240      CALL LROMOF
9C61 0619   0250 ROMSET LD   B, 25
9C63 C5     0260 SETUP PUSH BC
9C64 3E19   0270      LD   A, 25
9C66 CDD19C 0280      CALL NOWAIT
9C69 C1     0290      POP  BC
9C6A 10F7   0300      DJNZ SETUP
9C6C 3E39   0310 MAIN  LD   A, 57

```

9C6E	CD1EBB	0320		CALL	TSTKEY
9C71	2807	0330		JR	Z, NOT3
9C73	3E19	0340		LD	A, 25
9C75	CDD19C	0350		CALL	NOWAIT
9C78	18F2	0360		JR	MAIN
9C7A	3E3F	0370	NOT3	LD	A, 63
9C7C	CD1EBB	0380		CALL	TSTKEY
9C7F	2807	0390		JR	Z, NOTX
9C81	3E01	0400		LD	A, 1
9C83	CDD19C	0410		CALL	NOWAIT
9C86	18E4	0420		JR	MAIN
9C88	3E3A	0430	NOTX	LD	A, 58
9C8A	CD1EBB	0440		CALL	TSTKEY
9C8D	2807	0450		JR	Z, NOTE
9C8F	3E19	0460		LD	A, 25
9C91	CDC79C	0470		CALL	PAUSE
9C94	18D6	0480		JR	MAIN
9C96	3E3D	0490	NOTE	LD	A, 61
9C98	CD1EBB	0500		CALL	TSTKEY
9C9B	2807	0510		JR	Z, NOTD
9C9D	3E01	0520		LD	A, 1
9C9F	CDC79C	0530		CALL	PAUSE
9CA2	18C8	0540		JR	MAIN
9CA4	3E12	0550	NOTD	LD	A, 18
9CA6	CD1EBB	0560		CALL	TSTKEY
9CA9	200F	0570		JR	NZ, ENTER
9CAB	3E43	0580		LD	A, 67
9CAD	CD1EBB	0590		CALL	TSTKEY
9CB0	28BA	0600		JR	Z, MAIN
9CB2	DD6E02	0610		LD	L, (IX+2)
9CB5	DD6603	0620		LD	H, (IX+3)
9CB8	3663	0630		LD	(HL), 99
9CBA	DD6E00	0640	ENTER	LD	L, (IX+0)
9CBD	DD6601	0650		LD	H, (IX+1)

9CC0	73	0660	LD	(HL), E
9CC1	23	0670	INC	HL
9CC2	72	0680	LD	(HL), D
9CC3	CD03BB	0690	CALL	KRESET
9CC6	C9	0700	RET	
9CC7	F5	0710	PAUSE	PUSH AF
9CC8	010050	0720	LD	BC, 5000H
9CCB	0B	0730	DELAY	DEC BC
9CCC	78	0740	LD	A, B
9CCD	B1	0750	OR	C
9CCE	20FB	0760	JR	NZ, DELAY
9CD0	F1	0770	POP	AF
9CD1	F5	0780	NOWAIT	PUSH AF
9CD2	D5	0790		PUSH DE
9CD3	47	0800	LD	B, A
9CD4	CB38	0810	SRL	B
9CD6	CB38	0820	SRL	B
9CD8	CB38	0830	SRL	B
9CDA	CB38	0840	SRL	B
9CDC	AF	0850	XOR	A
9CDD	CD4DBC	0860	CALL	SCROLL
9CE0	3E1F	0870	LD	A, 31
9CE2	CD5ABB	0880	CALL	TXTOUT
9CE5	3E05	0890	LD	A, 5
9CE7	CD5ABB	0900	CALL	TXTOUT
9CEA	D1	0910	POP	DE
9CEB	F1	0920	POP	AF
9CEC	F5	0930	PUSH	AF
9CED	CD5ABB	0940	CALL	TXTOUT
9CF0	FE01	0950	CP	1
9CF2	2807	0960	JR	Z, DOWN
9CF4	13	0970	INC	DE
9CF5	211800	0980	LD	HL, 24
9CF8	19	0990	ADD	HL, DE

9CF9	1803	1000	JR	PRINT
9CFB	1B	1010	DOWN	DEC DE
9CFC	62	1020	LD	H, D
9CFD	6B	1030	LD	L, E
9CFE	44	1040	PRINT	LD B, H
9CFF	CD349D	1050	CALL	HEXOUT
9D02	45	1060	LD	B, L
9D03	CD349D	1070	CALL	HEXOUT
9D06	3E1F	1080	LD	A, 31
9D08	CD5ABB	1090	CALL	TXTOUT
9D0B	3E0D	1100	LA	A, 13
9D0D	CD5ABB	1110	CALL	TXTOUT
9D10	F1	1120	POP	AF
9D11	F5	1130	PUSH	AF
9D12	CD5ABB	1140	CALL	TXTOUT
9D15	46	1150	LD	B, (HL)
9D16	CD349D	1160	CALL	HEXOUT
9D19	C1	1170	POP	BC
9D1A	7E	1180	LD	A, (HL)
9D1B	FE21	1190	CP	33
9D1D	F8	1200	RET	M
9D1E	FE7F	1210	CP	127
9D20	F0	1220	RET	P
9D21	3E1F	1230	LD	A, 31
9D23	CD5ABB	1240	CALL	TXTOUT
9D26	3E13	1250	LD	A, 19
9D28	CD5ABB	1260	CALL	TXTOUT
9D2B	78	1270	LD	A, B
9D2C	CD5ABB	1280	CALL	TXTOUT
9D2F	7E	1290	LD	A, (HL)
9D30	CD5ABB	1300	CALL	TXTOUT
9D33	C9	1310	RET	
9D34	78	1320	HEXOUT	LD A, B
9D35	E60F	1330	AND	15

9D37	4F	1340	LD	C, A
9D38	CB38	1350	SRL	B
9D3A	CB38	1360	SRL	B
9D3C	CB38	1370	SRL	B
9D3E	CB38	1380	SRL	B
9D40	78	1390	LD	A, B
9D41	0602	1400	LD	B, 2
9D43	FE0A	1410	DIGIT CP	10
9D45	F24C9D	1420	JP	P, LETTER
9D48	C630	1430	ADD	A, 48
9D4A	1802	1440	JR	PRNTCH
9D4C	C637	1450	LETTER ADD	A, 55
9D4E	CD5ABB	1460	PRNTCH CALL	TXTOUT
9D51	79	1470	LD	A, C
9D52	10EF	1480	DJNZ	DIGIT
9D54	C9	1490	RET	
		1500	END	

# DIEZ - Movimiento de Bloques

Estamos muy orgullosos de estas rutinas. Las encontrará muy útiles para programas que requieren gráficos.

## Movimiento de un bloque de pantalla hacia arriba

Todos los libros de código máquina incluyen algunas rutinas para mover la pantalla ('scroll'). Suelen estar restringidas a una línea, una columna o toda la pantalla. Nuestra rutina para hacer **Movimiento de un Bloque hacia Arriba** (y las otras tres rutinas que cubren el resto de las direcciones, abajo, hacia la izquierda y hacia la derecha) le permitirán especificar un bloque, o ventana en la pantalla, y hacer que se mueva suavemente.

Un movimiento suave es un movimiento punto a punto. Aún en código máquina, esto resulta lento, pero será lo suficientemente rápido para la mayoría de las exigencias. Su gran valor reside en lo suave del movimiento, en comparación con el que se hace carácter a carácter.

La definición del bloque se hace de modo similar al comando WINDOW. Su formato es:

```
CALL 40000,L,R,U,D
```

En este comando, L es la coordenada de más a la izquierda del bloque, R es la coordenada de más a la derecha, U es la superior y D la coordenada inferior. Por lo tanto, el bloque solo puede ser cuadrado o rectangular. Obviamente, L debe ser igual o menor que R; y U debe ser igual o menor que D. Tenga en cuenta que L, R, U y D deben ser variables numéricas o números.

```
1 ' SCROLL DE UN BLOQUE HACIA ARRIBA
```

```
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
```

```

20 FOR n=40000 TO 40145
30 READ a$:POKE n,VAL("&" +a$)
40 NEXT
50 DATA DD,6E,02,DD,66,06,2D,25,DD,7E
60 DATA 04,94,4F,DD,7E,00,95,57,CD,1A
70 DATA BC,3E,00,81,10,FD,5F,18,38,D5
80 DATA 43,5D,7C,C6,38,57,7D,C6,50,6F
90 DATA 30,0A,24,7C,E6,07,20,04,7C,D6
100 DATA 08,67,E5,7E,12,1C,20,0A,14,7A
110 DATA E6,07,20,04,7A,D6,08,57,2C,20
120 DATA 0A,24,7C,E6,07,20,04,7C,D6,08
130 DATA 67,10,E2,E1,D1,D5,E5,4B,E5,06
140 DATA 07,5D,54,7C,C6,08,67,7E,12,10
150 DATA F6,E1,2C,20,0A,24,7C,E6,07,20
160 DATA 04,7C,D6,08,67,0D,20,E2,E1,D1
170 DATA 15,20,A2,7C,C6,38,67,43,36,00
180 DATA 2C,20,0A,24,7C,E6,07,20,04,7C
190 DATA D6,08,67,10,EF,C9

```

Cuando ejecute el comando CALL, el bloque especificado se moverá hacia arriba un punto, desaparecerá la línea superior y se rellenará con blancos la inferior. En la mayoría de los casos no será suficiente mover el bloque un solo punto. Para moverlo un número determinado de veces, use un bucle FOR/NEXT.

Hay unas pocas limitaciones en el uso de este programa. Funciona en cualquier modo de pantalla, y se pueden mover varios bloques a la vez. Por supuesto, cuanto más bloques tenga en la pantalla, así como cuanto mayor sea el tamaño de estos, más lento será el movimiento. Por lo tanto, es mejor restringir el tamaño y el número de bloques a los estrictamente necesarios. Para conseguir un efecto más interesante, intente mover dos bloques que se solapen uno a otro.

```

0010 ;SCROLL BLOQUE ARRIBA
0020 ;

```

9C40	0030	ORG	40000
BC1A	0040	CHRPOS	DEFL 0BC1AH
9C40 DD6E02	0050	LD	L, (IX+2)
9C43 DD6606	0060	LD	H, (IX+6)
9C46 2D	0070	DEC	L
9C47 25	0080	DEC	H
9C48 DD7E04	0090	LD	A, (IX+4)
9C4B 94	0100	SUB	H
9C4C 4F	0110	LD	C, A
9C4D DD7E00	0120	LD	A, (IX+0)
9C50 95	0130	SUB	L
9C51 57	0140	LD	D, A
9C52 CD1ABC	0150	CALL	CHRPOS
9C55 3E00	0160	LD	A, 0
9C57 81	0170	CHRWID	ADD A, C
9C58 10FD	0180	DJNZ	CHRWID
9C5A 5F	0190	LD	E, A
9C5B 1838	0200	JR	START
9C5D D5	0210	NXTROW	PUSH DE
9C5E 43	0220	LD	B, E
9C5F 5D	0230	LD	E, L
9C60 7C	0240	LD	A, H
9C61 C638	0250	ADD	A, 56
9C63 57	0260	LD	D, A
9C64 7D	0270	LD	A, L
9C65 C650	0280	ADD	A, 80
9C67 6F	0290	LD	L, A
9C68 300A	0300	JR	NC, NEWLIN
9C6A 24	0310	INC	H
9C6B 7C	0320	LD	A, H
9C6C E607	0330	AND	7
9C6E 2004	0340	JR	NZ, NEWLIN
9C70 7C	0350	LD	A, H
9C71 D608	0360	SUB	8

9C73	67	0370	LD	H, A
9C74	E5	0380	NEWLIN	PUSH HL
9C75	7E	0390	LASTLN	LD A, (HL)
9C76	12	0400	LD	(DE), A
9C77	1C	0410	INC	E
9C78	200A	0420	JR	NZ, DEOK
9C7A	14	0430	INC	D
9C7B	7A	0440	LD	A, D
9C7C	E607	0450	AND	7
9C7E	2004	0460	JR	NZ, DEOK
9C80	7A	0470	LD	A, D
9C81	D608	0480	SUB	8
9C83	57	0490	LD	D, A
9C84	2C	0500	DEOK	INC L
9C85	200A	0510	JR	NZ, HLOK
9C87	24	0520	INC	H
9C88	7C	0530	LD	A, H
9C89	E607	0540	AND	7
9C8B	2004	0550	JR	NZ, HLOK
9C8D	7C	0560	LD	A, H
9C8E	D608	0570	SUB	8
9C90	67	0580	LD	H, A
9C91	10E2	0590	HLOK	DJNZ LASTLN
9C93	E1	0600	POP	HL
9C94	D1	0610	POP	DE
9C95	D5	0620	START	PUSH DE
9C96	E5	0630	PUSH	HL
9C97	4B	0640	LD	C, E
9C98	E5	0650	NXTCHR	PUSH HL
9C99	0607	0660	LD	B, 7
9C9B	5D	0670	NXTLIN	LD E, L
9C9C	54	0680	LD	D, H
9C9D	7C	0690	LD	A, H
9C9E	C608	0700	ADD	A, 8

9CA0	67	0710	LD	H, A
9CA1	7E	0720	LD	A, (HL)
9CA2	12	0730	LD	(DE), A
9CA3	10F6	0740	DJNZ	NXTLIN
9CA5	E1	0750	POP	HL
9CA6	2C	0760	INC	L
9CA7	200A	0770	JR	NZ, OK
9CA9	24	0780	INC	H
9CAA	7C	0790	LD	A, H
9CAB	E607	0800	AND	7
9CAD	2004	0810	JR	NZ, OK
9CAF	7C	0820	LD	A, H
9CB0	D608	0830	SUB	8
9CB2	67	0840	LD	H, A
9CB3	0D	0850	DEC	C
9CB4	20E2	0860	JR	NZ, NXTCHR
9CB6	E1	0870	POP	HL
9CB7	D1	0880	POP	DE
9CB8	15	0890	DEC	D
9CB9	20A2	0900	JR	NZ, NXTROW
9CBB	7C	0910	LD	A, H
9CBC	C638	0920	ADD	A, 56
9CBE	67	0930	LD	H, A
9CBF	43	0940	LD	B, E
9CC0	3600	0950	LD	(HL), 0
9CC2	2C	0960	INC	L
9CC3	200A	0970	JR	NZ, CONT
9CC5	24	0980	INC	H
9CC6	7C	0990	LD	A, H
9CC7	E607	1000	AND	7
9CC9	2004	1010	JR	NZ, CONT
9CCB	7C	1020	LD	A, H
9CCC	D608	1030	SUB	8
9CCE	67	1040	LD	H, A

9CCF 10EF	1050 CONT	DJNZ BLANK
9CD1 C9	1060	RET
	1070	END

### Movimiento de un Bloque de Pantalla hacia abajo

Esta rutina tiene un diseño y uso similar a la anterior, usada para mover un bloque de pantalla hacia arriba. Funciona en los tres modos de pantalla y el formato es exactamente igual al anterior:

```
. CALL 40000,L,R,U,D
```

```
1 ' SCROLL DE UN BLOQUE HACIA ABAJO
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40151
30 READ a$:POKE n,VAL("&" + a$)
40 NEXT
50 DATA DD,6E,00,DD,66,06,2D,25,DD,7E
60 DATA 04,94,4F,7D,C6,02,DD,96,02,57
70 DATA CD,1A,BC,7C,C6,38,67,3E,00,81
80 DATA 10,FD,5F,18,38,D5,43,5D,7C,D6
90 DATA 38,57,7D,D6,50,6F,30,0A,7C,25
100 DATA E6,07,20,04,7C,C6,08,67,E5,7E
110 DATA 12,1C,20,0A,14,7A,E6,07,20,04
120 DATA 7A,D6,08,57,2C,20,0A,24,7C,E6
130 DATA 07,20,04,7C,D6,08,67,10,E2,E1
140 DATA D1,D5,E5,4B,E5,06,07,5D,54,7C
150 DATA D6,08,67,7E,12,10,F6,E1,2C,20
160 DATA 0A,24,7C,E6,07,20,04,7C,D6,08
170 DATA 67,0D,20,E2,E1,D1,15,20,A2,7C
180 DATA D6,38,67,43,36,00,2C,20,0A,24
190 DATA 7C,E6,07,20,04,7C,D6,08,67,10
```

200 DATA EF,C9

De nuevo, L es izquierda, R es derecha, U es arriba y D es abajo. Como en la rutina anterior, una simple CALL dará como resultado un movimiento de un punto, por lo que será necesario un bucle si intenta mover varios punto el bloque.

```
0001 ; SCROLL BLOQUE ABAJO
0002 ;
9C40      0010      ORG  40000
BC1A      0020  CHRPOS  DEFL 0BC1AH
9C40 DD6E00  0030      LD   L(IX+0)
9C43 DD6606  0040      LD   H,(IX+6)
9C46 2D      0050      DEC  L
9C47 25      0060      DEC  H
9C48 DD7E04  0070      LD   A,(IX+4)
9C4B 94      0080      SUB  H
9C4C 4F      0090      LD   C,A
9C4D 7D      0100      LD   A,L
9C4E C602    0110      ADD  A,2
9C50 DD9602  0120      SUB  (IX+2)
9C53 57      0130      LD   D,A
9C54 CD1ABC  0140      CALL CHRPOS
9C57 7C      0150      LD   A,H
9C58 C638    0160      ADD  A,56
9C5A 67      0170      LD   H,A
9C5B 3E00    0180      LD   A,0
9C5D 81      0190  CHRWID  ADD  A,C
9C5E 10FD    0200      DJNZ CHRWID
9C60 5F      0210      LD   E,A
9C61 1838    0220      JR   START
9C63 D5      0230  NXTROW  PUSH DE
9C64 43      0240      LD   B,E
```

9C65 5D	0250	LE	E, L
9C66 7C	0260	LE	A, H
9C67 D638	0270	SUB	56
9C69 57	0280	LD	D, A
9C6A 7D	0290	LD	A, L
9C6B D650	0300	SUB	80
9C6D 6F	0310	LD	L, A
9C6E 300A	0320	JR	NC, NEWLIN
9C70 7C	0330	LD	A, H
9C71 25	0340	DEC	H
9C72 E607	0350	AND	7
9C74 2004	0360	JR	NZ, NEWLIN
9C76 7C	0370	LD	A, H
9C77 C608	0380	ADD	A, 8
9C79 67	0390	LD	H, A
9C7A E5	0400	NEWLIN PUSH	HL
9C7B 7E	0410	LATLN LD	A, (HL)
9C7C 12	0420	LD	(DE), A
9C7D 1C	0430	INC	E
9C7E 200A	0440	JR	NZ, DEOK
9C80 14	0450	INC	D
9C81 7A	0460	LD	A, D
9C82 E607	0470	AND	7
9C84 2004	0480	JR	NZ, DEOK
9C86 7A	0490	LD	A, D
9C87 D608	0500	SUB	8
9C89 57	0510	LD	D, A
9C8A 2C	0520	DEOK INC	L
9C8B 200A	0530	JR	NZ, HLOK
9C8D 24	0540	INC	H
9C8E 7C	0550	LD	A, H
9C8F E607	0560	AND	7
9C91 2004	0570	JR	NZ, HLOK
9C93 7C	0580	LD	A, H

9C94	D608	0590	SUB	8
9C96	67	0600	LD	H, A
9C97	10E2	0610	HLOK	DJNZ LASTLN
9C99	E1	0620	POP	HL
9C9A	D1	0630	POP	DE
9C9B	D5	0640	START	PUSH DE
9C9C	E5	0650	PUSH	HL
9C9D	4B	0660	LD	C, E
9C9E	E5	0670	NXTCHR	PUSH HL
9C9F	0607	0680	LD	B, 7
9CA1	5D	0690	NXTLIN	LD E, L
9CA2	54	0700	LD	D, H
9CA3	7C	0710	LD	A, H
9CA4	D608	0720	SUB	8
9CA6	67	0730	LD	H, A
9CA7	7E	0740	LD	A, (HL)
9CA8	12	0750	LD	(DE), A
9CA9	10F6	0760	DJNZ	NXTLIN
9CAB	E1	0770	POP	HL
9CAC	2C	0780	INC	L
9CAD	200A	0790	JR	NZ, OK
9CAF	24	0800	INC	H
9CB0	7C	0810	LD	A, H
9CB1	E607	0820	AND	7
9CB3	2004	0830	JR	NZ, OK
9CB5	7C	0840	LD	A, H
9CB6	D608	0850	SUB	8
9CB8	67	0860	LD	H, A
9CB9	0D	0870	OK	DEC C
9CBA	20E2	0880	JR	NZ, NXTCHR
9CBC	E1	0890	POP	HL
9CBD	D1	0900	POP	DE
9CBE	15	0910	DEC	D
9CBF	20A2	0920	JR	NZ, NXTROW

9CC1	7C	0930	LD	A, H
9CC2	D638	0940	SUB	56
9CC4	67	0950	LD	H, A
9CC5	43	0960	LD	B, E
9CC6	3600	0970	BLANK	LD (HL), 0
9CC8	2C	0980	INC	L
9CC9	200A	0990	JR	NZ, CONT
9CCB	24	1000	INC	H
9CCC	7C	1010	LD	A, H
9CCD	E607	1020	AND	7
9CCF	2004	1030	JR	NZ, CONT
9CD1	7C	1040	LD	A, H
9CD2	D608	1050	SUB	8
9CD4	67	1060	LD	H, A
9CD5	10EF	1070	CONT	DJNZ BLANK
9CD7	C9	1080	RET	
		1090	END	

### Movimiento de un Bloque de Pantalla hacia la Izquierda

La rutina para mover un bloque hacia la izquierda (y hacia la derecha) funciona de la misma forma que hacia arriba y hacia abajo, la rutina crea el efecto con algunas diferencias. Esto se debe principalmente, a que el movimiento horizontal es opuesto al movimiento vertical de las dos rutinas anteriores.

El formato es exactamente el mismo:

```
CALL 40000, L, R, U, D
```

```
1 ' SCROLL DE UN BLOQUE HACIA LA IZQUIERDA
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40271
30 READ a$:POKE n, VAL("&" + a$)
```

40 NEXT  
50 DATA DD,6E,02,DD,66,06,2D,25,DD,7E  
60 DATA 04,94,4F,DD,7E,00,95,57,CD,1A  
70 DATA BC,3E,00,81,10,FD,47,CD,11,BC  
80 DATA DA,FA,9C,28,48,DD,6E,02,DD,66  
90 DATA 04,2D,25,C5,CD,1A,BC,C1,0E,08  
100 DATA C5,E5,B7,CB,16,F5,7D,2D,B7,20  
110 DATA 0A,7C,25,E6,07,20,04,7C,C6,08  
120 DATA 67,F1,10,EB,E1,7C,C6,08,67,C1  
130 DATA 0D,20,DF,7C,D6,40,67,7D,C6,50  
140 DATA 6F,30,0A,24,7C,E6,07,20,04,7C  
150 DATA D6,08,67,15,20,C6,C9,05,D5,0E  
160 DATA 08,C5,E5,CB,26,5D,54,2C,20,0A  
170 DATA 24,7C,E6,07,20,04,7C,D6,08,67  
180 DATA 1A,CB,26,38,04,CB,A7,18,02,CB  
190 DATA E7,CB,66,28,02,CB,C7,12,10,DD  
200 DATA CB,A6,E1,7C,C6,08,67,C1,0D,20  
210 DATA CE,7C,D6,40,67,7D,C6,50,6F,30  
220 DATA 0A,24,7C,E6,07,20,04,7C,D6,08  
230 DATA 67,D1,15,20,B3,C9,05,D5,0E,08  
240 DATA C5,E5,C5,5D,54,2C,20,0A,24,7C  
250 DATA E6,07,20,04,7C,D6,08,67,7E,1F  
260 DATA 4F,1A,17,17,06,04,17,CB,21,CB  
270 DATA 21,17,10,F8,12,C1,10,DC,7E,17  
280 DATA 06,04,07,CB,27,10,FB,77,E1,7C  
290 DATA C6,08,67,C1,0D,20,C7,7C,D6,40  
300 DATA 67,7D,C6,50,6F,30,0A,24,7C,E6  
310 DATA 07,20,04,7C,D6,08,67,D1,15,20  
320 DATA AC,C9

Como en las anteriores, L es izquierda, R derecha, U arriba y D abajo. Como en las otras rutinas, una simple CALL hace un movimiento de un solo punto, por lo que es necesario un bucle si quiere mover más puntos. Esta rutina también funciona en los tres modos de pantalla, como la de

Movimiento de un bloque de pantalla hacia la Izquierda.

```

0001 ; SCROLL BLOQUE IZQUIERDA
0002 ;
9C40      0010      ORG  40000
BC1A      0020  CHRPOS  DEFL 0BC1AH
BC11      0030  SCMODE  DEFL 0BC11H
9C40 DD6E02      0040      LD   L, (IX+2)
9C43 DD6E06      0050      LD   H, (IX+6)
9C46 20        0060      DEC  L
9C47 25        0070      DEC  H
9C48 DD7E04      0080      LD   A, (IX+4)
9C4E 34        0090      SUB  H
9C4C 4F        0100      LD   C, A
9C4D DD7E00      0110      LD   A, (IX+0)
9C50 95        0120      SUB  L
9C51 57        0130      LD   D, A
9C52 CD1AEC      0140      CALL CHRPOS
9C53 3E00        0150      LD   A, 0
9C57 81        0160  CHRWID  ADD  A, C
9C58 10FD        0170      DJNZ CHRWID
9C5A 47        0180      LD   B, A
9C5B CD11EC      0190      CALL SCMODE
9C5E DAFA9C      0200      JP   C, MODE0
9C61 2848        0210      JR   Z, MODE1
9C63 DD6E02      0220      LD   L, (IX+2)
9C66 DD6E04      0230      LD   H, (IX+4)
9C69 20        0240      DEC  L
9C6A 25        0250      DEC  H
9C6B C5        0260      PUSH BC
9C6C CD1ABC      0270      CALL CHRPOS
9C6F C1        0280      POP  BC
9C70 0E08        0290  ROW2   LD   C, 8

```

9C72	C5	0300	LINE2	PUSH	BC
9C73	E5	0310		PUSH	HL
9C74	B7	0320		OR	A
9C75	CB16	0330	BYTE2	PL	(HL)
9C77	F5	0340		PUSH	AF
9C78	7D	0350		LD	A, L
9C79	2D	0360		DEC	L
9C7A	B7	0370		OR	A
9C7B	200A	0380		JR	NZ, OK2
9C7D	7C	0390		LD	A, H
9C7E	25	0400		DEC	H
9C7F	E607	0410		AND	7
9C81	2004	0420		JR	NZ, OK2
9C83	7C	0430		LD	A, H
9C84	C608	0440		ADD	A, 8
9C86	67	0450		LD	H, A
9C87	F1	0460	OK2	POP	AF
9C88	10EB	0470		DJNZ	BYTE2
9C8A	E1	0480		POP	HL
9C8B	7C	0490		LD	A, H
9C8C	C608	0500		ADD	A, 8
9C8E	67	0510		LD	H, A
9C8F	C1	0520		POP	BC
9C90	0D	0530		DEC	C
9C91	20DF	0540		JR	NZ, LINE2
9C93	7C	0550		LD	A, H
9C94	D640	0560		SUB	64
9C96	67	0570		LD	H, A
9C97	7D	0580		LD	A, L
9C98	C650	0590		ADD	A, 80
9C9A	6F	0600		LD	L, A
9C9B	300A	0610		JR	NC, DONE2
9C9D	24	0620		INC	H
9C9E	7C	0630		LD	A, H

9C9F	E607	0640	AND	7
9CA1	2004	0650	JR	NZ, DONE2
9CA3	7C	0660	LD	A, H
9CA4	D608	0670	SUB	8
9CA6	67	0680	LD	H, A
9CA7	15	0690	DONE2	DEC D
9CA8	20C6	0700	JR	NZ, ROW2
9CAA	C9	0710	RET	
9CAB	05	0720	MODE1	DEC B
9CAC	D5	0730	ROW1	PUSH DE
9CAD	0E08	0740	LD	C, 8
9CAF	C5	0750	LINE1	PUSH BC
9CB0	E5	0760	PUSH	HL
9CB1	CB26	0770	SLA	(HL)
9CB3	5D	0780	BYTE1	LD E, L
9CB4	54	0790	LD	D, H
9CB5	2C	0800	INC	L
9CB6	200A	0810	JR	NZ, OK1
9CB8	24	0820	INC	H
9CB9	7C	0830	LD	A, H
9CBA	E607	0840	AND	7
9CBC	2004	0850	JR	NZ, OK1
9CBE	7C	0860	LD	A, H
9CBF	D608	0870	SUB	8
9CC1	67	0880	LD	H, A
9CC2	1A	0890	OK1	LD A, (DE)
9CC3	CB26	0900	SLA	(HL)
9CC5	3804	0910	JR	C, SETBIT
9CC7	CBA7	0920	RES	4, A
9CC9	1802	0930	JR	BITOK
9CCB	CBE7	0940	SETBIT	SET 4, A
9CCD	CB66	0950	BITOK	BIT 4, (HL)
9CCF	2802	0960	JR	Z, BITSET
9CD1	CBC7	0970	SET	0, A

9CD3	12	0980	BITSET	LD	(DE), A
9CD4	10DD	0990		DJNZ	BYTE1
9CD6	CBA6	1000		RES	4, (HL)
9CD8	E1	1010		POP	HL
9CD9	7C	1020		LD	A, H
9CDA	C608	1030		ADD	A, 8
9CDC	67	1040		LD	H, A
9CDD	C1	1050		POP	BC
9CDE	0D	1060		DEC	C
9CDF	20CE	1070		JR	NZ, LINE1
9CE1	7C	1080		LD	A, H
9CE2	D640	1090		SUB	64
9CE4	67	1100		LD	H, A
9CE5	7D	1110		LD	A, L
9CE6	C650	1120		ADD	A, 80
9CE8	6F	1130		LD	L, A
9CE9	300A	1140		JR	NC, DONE1
9CEB	24	1150		INC	H
9CEC	7C	1160		LD	A, H
9CED	E607	1170		AND	7
9CEF	2004	1180		JR	NZ, DONE1
9CF1	7C	1190		LD	A, H
9CF2	D608	1200		SUB	8
9CF4	67	1210		LD	H, A
9CF5	D1	1220	DONE1	POP	DE
9CF6	15	1230		DEC	D
9CF7	20B3	1240		JR	NZ, ROW1
9CF9	C9	1250		RET	
9CFA	05	1260	MODE0	DEC	B
9CFB	D5	1270	ROW0	PUSH	DE
9CFC	0E08	1280		LD	C, 8
9CFE	C5	1290	LINE0	PUSH	BC
9CFF	E5	1300		PUSH	HL
9D00	C5	1310	BYTE0	PUSH	BC

9D01	5D	1320	LD	E, L
9D02	54	1330	LD	D, H
9D03	2C	1340	INC	L
9D04	200A	1350	JR	NZ, OK0
9D06	24	1360	INC	H
9D07	7C	1370	LD	A, H
9D08	E607	1380	AND	7
9D0A	2004	1390	JR	NZ, OK0
9D0C	7C	1400	LD	A, H
9D0D	D608	1410	SUB	8
9D0E	67	1420	LD	H, A
9D10	7E	1430	LD	A, (HL) OK0
9D11	1F	1440	RRA	
9D12	4F	1450	LD	C, A
9D13	1A	1460	LD	A, (DE)
9D14	17	1470	RLA	
9D15	17	1480	RLA	
9D16	0604	1490	LD	B, 4
9D18	17	1500	NXTROT	RLA
9D19	CB21	1510	SLA	C
9D1B	CB21	1520	SLA	C
9D1D	17	1530	RLA	
9D1E	10F8	1540	DJNZ	NXTROT
9D20	12	1550	LD	(DE), A
9D21	C1	1560	POP	BC
9D22	10DC	1570	DJNZ	BYTE0
9D24	7E	1580	LD	A, (HL)
9d25	17	1590	RLA	
9D26	0604	1600	LD	B, 4
9D28	07	1610	LSTBYT	RLCA
9D29	CB27	1620	SLA	A
9D2B	10FB	1630	DJNZ	LSTBYT
9D2D	77	1640	LD	(HL), A
9D2E	E1	1650	POP	HL

902F 7C	1660	LD	A, H
9030 C608	1670	ADD	A, 8
9032 67	1680	LD	H, A
9033 C1	1690	POP	BC
9034 0D	1700	DEC	C
9035 20C7	1710	JR	NZ, LINE0
9037 7C	1720	LD	A, H
9038 D640	1730	SUB	64
903A 67	1740	LD	H, A
903B 7D	1750	LD	A, L
903C C650	1760	ADD	A, 80
903E 6F	1770	LD	L, A
903F 300A	1780	JR	NC, DONE0
9041 24	1790	INC	H
9042 7C	1800	LD	A, H
9043 E607	1810	AND	7
9045 2004	1820	JR	NZ, DONE0
9047 7C	1830	LD	A, H
9048 D608	1840	SUB	8
904A 67	1850	LD	H, A
904B D1	1860	POP	DE
904C 15	1870	DEC	D
904D 20AC	1880	JR	NZ, ROW0
904F C9	1890	RET	
	1900	END	

### Movimiento de un Bloque de Pantalla hacia la Derecha

Una vez vistas las otras rutinas de movimiento de bloques de pantalla hacia arriba, hacia abajo y hacia la izquierda, no es muy difícil saber cómo funciona esta rutina.

Una vez más. el formato que se usa es exactamente el mismo que en las rutina anteriores:

CALL 40000,L,R,U,D

Y como en las otras rutinas, una simple CALL hará un movimiento del bloque de pantalla un solo punto hacia la derecha, por lo que será necesario un bucle para obtener movimientos más amplios.

```
1 ' SCROLL DE UN BLOQUE HACIA LA DERECHA
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40277
30 READ a$:POKE n,VAL("&" + a$)
40 NEXT
50 DATA DD,6E,02,DD,66,04,2D,25,7C,C6
60 DATA 02,DD,96,06,4F,DD,7E,00,95,57
70 DATA CD,1A,BC,2B,3E,00,81,23,10,FC
80 DATA 47,CD,11,BC,DA,FE,9C,28,46,DD
90 DATA 6E,02,DD,66,06,2D,25,C5,CD,1A
100 DATA BC,C1,0E,08,C5,E5,B7,CB,1E,F5
110 DATA 2C,20,0A,24,7C,E6,07,20,04,7C
120 DATA D6,08,67,F1,10,ED,E1,7C,C6,08
130 DATA 67,C1,0D,20,E1,7C,D6,40,67,7D
140 DATA C6,50,6F,30,0A,24,7C,E6,07,20
150 DATA 04,7C,D6,08,67,15,20,C8,C9,05
160 DATA D5,0E,08,C5,E5,CB,3E,5D,54,7D
170 DATA 2D,B7,20,0A,7C,25,E6,07,20,04
180 DATA 7C,C6,08,67,1A,CB,3E,38,04,CB
190 DATA 9F,18,02,CB,DF,CB,5E,28,02,CB
200 DATA FF,12,10,DB,CB,9E,E1,7C,C6,08
210 DATA 67,C1,0D,20,CC,7C,D6,40,67,7D
220 DATA C6,50,6F,30,0A,24,7C,E6,07,20
230 DATA 04,7C,D6,08,67,D1,15,20,B1,C9
240 DATA 05,D5,0E,08,C5,E5,C5,5D,54,7D
250 DATA 2D,B7,20,0A,7C,25,E6,07,20,04
260 DATA 7C,C6,08,67,7E,17,4F,1A,1F,1F
```

270 DATA 06,04,1F,CB,39,CB,39,1F,10,F8  
 280 DATA 12,C1,10,DA,7E,1F,06,04,0F,CB  
 290 DATA 3F,10,FB,77,E1,7C,C6,08,67,C1  
 300 DATA 0D,20,C5,7C,D6,40,67,7D,C6,50  
 310 DATA 6F,30,0A,24,7C,E6,07,20,04,7C  
 320 DATA D6,08,67,D1,15,20,AA,C9

---

```

                                0001 ;   SCROLL BLOQUE DERECHA
                                0002 ;
9C40          0010          ORG  40000
BC1A          0020  CHRPOS  DEFL 0BC1AH
BC11          0030  SCMODE  DEFL 0BC11H
9C40 DD6E02   0040          LD   L, (IX+2)
9C43 DD6604   0050          LD   H, (IX+4)
9C46 2D       0060          DEC  L
9C47 25       0070          DEC  H
9C48 7C       0080          LD   A,H
9C49 C602     0090          ADD  A,Z
9C4B DD9606   0100          SUB  (IX+6)
9C4E 4F       0110          LD   C,A
9C4F DD7E00   0120          LD   A, (IX+0)
9C52 95       0130          SUB  L
9C53 57       0140          LD   D,A
9C54 CD1ABC   0150          CALL CHRPOS
9C57 2B       0160          DEC  HL
9C58 3E00     0170          LD   A,0
9C5A 81       0180  CHRWID  ADD  A,C
9C5B 23       0190          INC  HL
9C5C 10FC     0200          DJNZ CHRWID
  
```

9C5E	47	0210	LD	B, A
9C5F	CD11BC	0220	CALL	SCMODE
9C62	DAFE9C	0230	JP	C, MODE0
9C65	2846	0240	JR	Z, MODE1
9C67	DD6E02	0250	LD	L, (IX+2)
9C6A	DD6606	0260	LD	H, (IX+6)
9C6D	2D	0270	DEC	L
9C6E	25	0280	DEC	H
9C6F	C5	0290	PUSH	BC
9C70	CD1ABC	0300	CALL	CHRPOS
9C73	C1	0310	POP	BC
9C74	0E08	0320	ROW2	LD C, 8
9C76	C5	0330	LINE2	PUSH BC
9C77	E5	0340		PUSH HL
9C78	B7	0350	OR	A
9C79	CB1E	0360	BYTE2	RR (HL)
9C7B	F5	0370		PUSH AF
9C7C	2C	0380		INC L
9C7D	200A	0390		JR NZ, OK2
9C7F	24	0400		INC H
9C80	7C	0410		LD A, H
9C81	E607	0420		AND 7
9C83	2004	0430		JR NZ, OK2
9C85	7C	0440		LD A, H
9C86	D608	0450		SUB 8
9C88	67	0460		LD H, A
9C89	F1	0470	OK2	POP AF
9C8A	10ED	0480		DJNZ BYTE2
9C8C	E1	0490		POP HL
9C8D	7C	0500		LD A, H
9C8E	C608	0510		ADD A, 8
9C90	67	0520		LD H, A
9C91	C1	0530		POP BC
9C92	0D	0540		DEC C

9C93	20E1	0550	JR	NZ, LINE2
9C95	7C	0560	LD	A, H
9C96	D640	0570	SUB	64
9C98	67	0580	LD	H, A
9C99	7D	0590	LD	A, L
9C9A	C650	0600	ADD	A, 80
9C9C	6F	0610	LD	L, A
9C9D	300A	0620	JR	NC, DONE2
9C9F	24	0630	INC	H
9CA0	7C	0640	LD	A, H
9CA1	E607	0650	AND	7
9CA3	2004	0660	JR	NZ, DONE2
9CA5	7C	0670	LD	A, H
9CA6	D608	0680	SUB	8
9CA8	67	0690	LD	H, A
9CA9	15	0700	DONE2	DEC D
9CAA	20C8	0710	JR	NZ, ROW2
9CAC	C9	0720	RET	
9CAD	05	0730	MODE1	DEC B
9CAE	D5	0740	ROW1	PUSH DE
9CAF	0E08	0750		LD C, 8
9CB1	C5	0760	LINE1	PUSH BC
9CB2	E5	0770		PUSH HL
9CB3	CB3E	0780	SRL	(HL)
9CB5	5D	0790	BYTE1	LD E, L
9CB6	54	0800		LD D, H
9CB7	7D	0810		LD A, L
9CB8	2D	0820	DEC	L
9CB9	B7	0830	OR	A
9CBA	200A	0840	JR	NZ, OK1
9CBC	7C	0850	LD	A, H
9CBD	25	0860	DEC	H
9CBE	E607	0870	AND	7
9CC0	2004	0880	JR	NZ, OK1

9CC2	7C	0890	LD	A, H
9CC3	C608	0900	ADD	A, 8
9CC5	67	0910	LD	H, A
9CC6	1A	0920	LD	A, (DE)
9CC7	CB3E	0930	SRL	(HL)
9CC9	3804	0940	JR	C, SETBIT
9CCB	CB9F	0950	RES	3, A
9CCD	1802	0960	JR	BITOK
9CCF	CBDF	0970	SETBIT	SET 3, A
9CD1	CB5E	0980	BITOK	BIT 3, (HL)
9CD3	2802	0990	JR	Z, BITSET
9CD5	CBFF	1000	SET	7, A
9CD7	12	1010	BITSET	LD (DE), A
9CD8	100B	1020	DJNZ	BYTE1
9CDA	CB9E	1030	RES	3, (HL)
9CDC	E1	1040	POP	HL
9CDD	7C	1050	LD	A, H
9CDE	C608	1060	ADD	A, 8
9CE0	67	1070	LD	H, A
9CE1	C1	1080	POP	BC
9CE2	0D	1090	DEC	C
9CE3	20CC	1100	JR	NZ, LINE1
9CE5	7C	1110	LD	A, H
9CE6	D640	1120	SUB	64
9CE8	67	1130	LD	H, A
9CE9	7D	1140	LD	A, L
9CEA	C650	1150	ADD	A, 80
9CEC	6F	1160	LD	L, A
9CED	300A	1170	JR	NC, DONE1
9CEF	24	1180	INC	H
9CF0	7C	1190	LD	A, H
9CF1	E607	1200	AND	7
9CF3	2004	1210	JR	NZ, DONE1
9CF5	7C	1220	LD	A, H

9CF6	D608	1230	SUB	8
9CF8	67	1240	LD	H, A
9CF9	D1	1250	DONE1	POP DE
9CFA	15	1260	DEC	D
9CFB	20B1	1270	JR	NZ, ROW1
9CFD	C9	1280	RET	
9CFE	05	1290	MODE0	DEC B
9CFF	D5	1300	ROW0	PUSH DE
9D00	0E08	1310	LD	C, 8
9D02	C5	1320	LINE0	PUSH BC
9D03	E5	1330	PUSH	HL
9D04	C5	1340	BYTE0	PUSH BC
9D05	5D	1350	LD	E, L
9D06	54	1360	LD	D, H
9D07	7D	1370	LD	A, L
9D08	2D	1380	DEC	L
9D09	B7	1390	OR	A
9D0A	200A	1400	JR	NZ, OK0
9D0C	7C	1410	LD	A, H
9D0D	25	1420	DEC	H
9D0E	E607	1430	AND	7
9D10	2004	1440	JR	NZ, OK0
9D12	7C	1450	LD	A, H
9D13	C608	1460	ADD	A, 8
9D15	67	1470	LD	H, A
9D16	7E	1480	OK0	LD A, (HL)
9D17	17	1490	RLA	
9D18	4F	1500	LD	C, A
9D19	1A	1510	LD	A, (DE)
9D1A	1F	1520	RRA	
9D1B	1F	1530	RRA	
9D1C	0604	1540	LD	B, 4
9D1E	1F	1550	NXTROT	RRA
9D1F	CB39	1560	SRL	C

9D21	CB39	1570	SRL	C
9D23	1F	1580	RRA	
9D24	10F8	1590	DJNZ	NXTROT
9D26	12	1600	LD	(DE), A
9D27	C1	1610	POP	BC
9D28	10DA	1620	DJNZ	BYTE0
9D2A	7E	1630	LD	A, (HL)
9D2B	1F	1640	RRA	
9D2C	0604	1650	LD	B, 4
9D2E	0F	1660	LSTBYT	RRCA
9D2F	CB3F	1670	SRL	A
9D31	10FB	1680	DJNZ	LSTBYT
9D33	77	1690	LD	(HL), A
9D34	E1	1700	POP	HL
9D35	7C	1710	LD	A, H
9D36	C608	1720	ADD	A, 8
9D38	67	1730	LD	H, A
9D39	C1	1740	POP	BC
9D3A	0D	1750	DEC	C
9D3E	20C5	1760	JR	NZ, LINE0
9D3D	7C	1770	LD	A, H
9D3E	D640	1780	SUB	64
9D40	67	1790	LD	H, A
9D41	7D	1800	LD	A, L
9D42	C650	1810	ADD	A, 80
9D44	6F	1820	LD	L, A
9D45	300A	1830	JR	NC, DONE0
9D47	24	1840	INC	H
9D48	7C	1850	LD	A, H
9D49	E607	1860	AND	7
9D4B	2004	1870	JR	NZ, DONE0
9D4D	7C	1880	LD	A, H
9D4E	D608	1890	SUB	8
9D50	67	1900	LD	H, A

9D51 D1	1910	DONE0	POP	DE
9D52 15	1920		DEC	D
9D53 20AA	1930		JR	NZ, ROW0
9D55 C9	1940		RET	
	1950		END	

## ONCE - Acordes RSX

El Amstrad está equipado con una facilidad maravillosa que se puede usar para crear comandos nuevos que son aceptados como palabras clave por el BASIC. Esta facilidad se conoce como RSX y esta rutina la usa para crear 24 comandos nuevos. Cada uno de estos comandos ejecuta un acorde determinado a través del sintetizador de sonido del Amstrad.

Un acorde, como seguramente sabrá, es una combinación de varias notas musicales en armonía, que pueden ser tocadas simultáneamente. Tecleando CALL 40000, tendrá acceso a 24 de los acordes más populares, que aparecen listados más abajo. (Tenga en cuenta que no hay espacios entre las partes de los nuevos comandos y que '#' equivale a sostenido y b. a bemol.

Nuevo Comando	Acorde
:CMAYOR	C Mayor
:CMENOR	C Menor
:CSMAYOR	C# Mayor o Db.Mayor
:CSMENOR	C# Menor o Db.Menor
:DMAYOR	D Mayor
:DMENOR	D Menor
:DSMAYOR	D# Mayor o Eb.Mayor
:DSMENOR	D# Menor o Eb.Menor
:EMAYOR	E Mayor
:EMENOR	E Menor
:FMAYOR	F Mayor
:FMENOR	F Menor
:FSMAYOR	F# Mayor o Gb.Mayor
:FSMENOR	F# Menor o Gb.Menor

:GMAYOR	G Mayor
:GMENOR	G Menor
:GSMAYOR	G# Mayor o Ab.Mayor
:GSMENOR	G# Menor o Ab.Menor
:AMAYOR	A Mayor
:AMENOR	A Menor
:ASMAYOR	A# Mayor o Bb.Mayor
:ASMENOR	A# Menor o Bb.Menor
:BMAYOR	B Mayor
:BMENOR	B Menor

Todos los acordes se componen de tres notas, y usan los tres canales de sonido del Amstrad para producirlos.

Para obtener un acorde, solamente debe teclear uno de los comandos. Es una buena idea poner el control de volumen a unas tres cuartas partes de su recorrido, ya que la salida de sonido es demasiado potente para el altavoz integrado, como para reproducir el sonido sin algo de distorsión.

1

#### ACORDES RSX

10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTEP 240

20 FOR n=40000 TO 40617

30 READ a\$:POKE n,VAL("&"&a\$)

40 NEXT

50 DATA 01,4A,90,21,94,90,CD,D1,BC,C9

60 DATA 98,90,C3,33,9D,C3,37,9D,C3,3B

70 DATA 9D,C3,3F,9D,C3,43,9D,C3,47,9D

80 DATA C3,4B,9D,C3,4F,9D,C3,53,9D,C3

90 DATA 57,9D,C3,5B,9D,C3,5F,9D,C3,63

100 DATA 9D,C3,67,9D,C3,6B,9D,C3,6F,9D

110 DATA C3,73,9D,C3,77,9D,C3,7B,9D,C3

120 DATA 7F,9D,C3,83,9D,C3,87,9D,C3,8B

130 DATA 9D,C3,8F,9D,00,00,00,00,43,4D

140 DATA 41, 59, 4F, D2, 43, 4D, 45, 4E, 4F, D2  
150 DATA 43, 53, 4D, 41, 59, 4F, D2, 43, 53, 4D  
160 DATA 45, 4E, 4F, D2, 44, 4D, 41, 59, 4F, D2  
170 DATA 44, 4D, 45, 4E, 4F, D2, 44, 53, 4D, 41  
180 DATA 59, 4F, D2, 44, 53, 4D, 45, 4E, 4F, D2  
190 DATA 45, 4D, 41, 59, 4F, D2, 45, 4D, 45, 4E  
200 DATA 4F, D2, 46, 4D, 41, 59, 4F, D2, 46, 4D  
210 DATA 45, 4E, 4F, D2, 46, 53, 4D, 41, 59, 4F  
220 DATA D2, 46, 53, 4D, 45, 4E, 4F, D2, 47, 4D  
230 DATA 41, 59, 4F, D2, 47, 4D, 45, 4E, 4F, D2  
240 DATA 47, 53, 4D, 41, 59, 4F, D2, 47, 53, 4D  
250 DATA 45, 4E, 4F, D2, 41, 4D, 41, 59, 4F, D2  
260 DATA 41, 4D, 45, 4E, 4F, D2, 41, 53, 4D, 41  
270 DATA 59, 4F, D2, 41, 53, 4D, 45, 4E, 4F, D2  
280 DATA 42, 4D, 41, 59, 4F, D2, 42, 4D, 45, 4E  
290 DATA 4F, D2, 00, 1E, 00, 18, 5C, 1E, 06, 18  
300 DATA 58, 1E, 0C, 18, 54, 1E, 12, 18, 50, 1E  
310 DATA 18, 19, 4C, 1E, 1E, 18, 48, 1E, 24, 18  
320 DATA 44, 1E, 2A, 18, 40, 1E, 30, 18, 3C, 1E  
330 DATA 36, 18, 38, 1E, 3C, 18, 34, 1E, 42, 18  
340 DATA 30, 1E, 48, 18, 2C, 1E, 4E, 18, 28, 1E  
350 DATA 54, 18, 24, 1E, 5A, 18, 20, 1E, 60, 18  
360 DATA 1C, 1E, 66, 18, 18, 1E, 6C, 18, 14, 1E  
370 DATA 72, 18, 10, 1E, 78, 18, 0C, 1E, 7E, 18  
380 DATA 08, 1E, 84, 18, 04, 1E, 8A, 18, 00, 21  
390 DATA 1A, 9E, 16, 00, 19, 5E, 23, 56, 23, ED  
400 DATA 53, 02, 9E, 5E, 23, 56, 23, ED, 53, 0B  
410 DATA 9E, 5E, 23, 56, ED, 53, 14, 9E, F5, B7  
420 DATA 28, 03, DD, 7E, 00, 32, 00, 9E, 32, 09  
430 DATA 9E, 32, 12, 9E, B7, 28, 12, CD, C2, BC  
440 DATA D0, 23, 23, 7E, B7, 28, 08, CB, 7F, 20  
450 DATA 04, 3E, 00, 18, 02, 3E, 0F, 32, 05, 9E  
460 DATA 32, 0E, 9E, 32, 17, 9E, F1, FE, 02, CC  
470 DATA A7, BC, 21, FF, 9D, CD, AA, BC, 30, FB

480 DATA 21,08,9E,CD,AA,BC,30,FB,21,11  
 490 DATA 9E,CD,AA,BC,30,FB,C9,31,00,00  
 500 DATA 00,00,00,00,00,00,2A,00,00,00  
 510 DATA 00,00,00,00,00,1C,00,00,00,00  
 520 DATA 00,00,00,00,7E,02,DE,01,7B,01  
 530 DATA 7E,02,DE,01,92,01,5A,02,C3,01  
 540 DATA 66,01,5A,02,C3,01,7B,01,A4,02  
 550 DATA 38,02,AA,01,38,02,AA,01,66,01  
 560 DATA 7E,02,18,02,92,01,A4,02,18,02  
 570 DATA 92,01,5A,02,FA,01,7B,01,7E,02  
 580 DATA FA,01,7B,01,38,02,DE,01,66,01  
 590 DATA 5A,02,DE,01,66,01,A4,02,18,02  
 600 DATA C3,01,A4,02,38,02,C3,01,7E,02  
 610 DATA FA,01,AA,01,7E,02,18,02,AA,01  
 620 DATA 5A,02,DE,01,92,01,5A,02,FA,01  
 630 DATA 92,01,38,02,C3,01,7B,01,38,02  
 640 DATA DE,01,7B,01,18,02,AA,01,66,01  
 650 DATA 18,02,C3,01,66,01,A4,02,FA,01  
 660 DATA 92,01,A4,02,FA,01,AA,01

## Envolventes

Veamos las partes más complejas del uso de estos tres acordes, usándolos con envolventes de sonido. Si 'toca' el acorde básico sin parámetros adicionales, obtendrá un sonido básico similar al de un acorde de 'órgano'. La rutina pone automáticamente el comando con la envolvente 0, que hace que cada acorde dure unos dos segundos.

Es posible crear envolventes definidas por usted mismo, que se pueden incorporar dentro del sonido del acorde. Para hacerlo, prepare una envolvente de sonido normal (si no está familiarizado con las facilidades de sonido del Amstrad, mire en el manual del usuario). A continuación, asigne un número de envolvente al final del comando del acorde. Por ejemplo, si su envolvente era ENV 1,3,4,1,15,-1,10 con el primer número indicando el número de la envolvente, debe añadir un '1' al final del comando de acorde. ej :CMAYOR,1.

El volumen inicial del acorde depende de la envolvente que se use. El programa lo fija de la siguiente forma. Si el tamaño del paso de la primera sección de la envolvente que se

está usando es positivo (excluido el 0), el volumen inicial se pone a 0. Si es negativo, el volumen inicial se pone al valor máximo, 15. Esto da a la envolvente el mayor rango de volumen con que puede trabajar.

Si usa un paso de envolvente de 0, puede hacer que el acorde dure todo lo que usted quiera. Por ejemplo:

```
ENV 2,10,0,100  
:CMAYOR,2
```

Esto toca el acorde C Mayor durante diez segundos mientras que ENV 2,n,0,100 tocará el acorde durante n segundos.

Cuando se ejecutan los comandos, la rutina esperará hasta que las tres memorias intermedias de sonido estén disponibles, antes de añadir las notas que preparan el acorde en las colas de sonido. Si quiere limpiar las colas de sonido inmediatamente después de que el comando sea ejecutado, debe añadir un parámetro extra **antes** del número de envolvente.

```
ENV 4,15,-1,20  
:CMAYOR,0,4  
:DMAYOR,0,4
```

En este ejemplo, solamente sonará el acorde D Mayor, ya que el primer acorde se cortará tan pronto como empieza. Se puede usar **cualquier** valor para este parámetro extra. La rutina busca solamente la presencia de un parámetro extra, y no el valor de ese parámetro.

Obviamente, usted no estará muy dispuesto a perder su primer acorde. Puede evitar esto poniendo un bucle de espera entre cada acorde. A continuación tenemos un ejemplo de como hacerlo:

```
ENV 5,8,0,100  
:EMAYOR,0,5  
FOR T=1 TO 300:NEXT  
:CMAYOR,0,5  
FOR T=1 TO 250:NEXT  
:DMAYOR,0,5
```

	0001 ;	ACORDES	
	0002 ;		
9C40	0010	OPG	40000
BCA7	0020	SNDRES	DEFL 0BCA7H
BCAA	0030	SUMSND	DEFL 0BCAAH
BCC2	0040	AMPDIP	DEFL 0BCC2H
BCD1	0050	LOGEXT	DEFL 0BCD1H
9C40 014A9C	0060	LD	BC, TABLA
9C43 21949C	0070	LD	HL, ESPAC
9C46 CDD1BC	0080	CALL	LOGEXT
9C49 C9	0090	RET	
9C4A 989C	0100	TABLA	DEFW NOMBTB
9C4C C3339D	0110	JP	CMAY
9C4F C3379D	0120	JP	CMEN
9C52 C33B9D	0130	JP	CSMAY
9C55 C33F9D	0140	JP	CSMEN
9C58 C3439D	0150	JP	DMAY
9C5B C3479D	0160	JP	DMEN
9C5E C34B9D	0170	JP	DSMAY
9C61 C34F9D	0180	JP	DSMEN
9C64 C3539D	0190	JP	EMAY
9C67 C3579D	0200	JP	EMEN
9C6A C35B9D	0210	JP	FMAY
9C6D C35F9D	0220	JP	FMEN
9C70 C3639D	0230	JP	FSMAY
9C73 C3679D	0240	JP	FSMEN
9C76 C36B9D	0250	JP	GMAY
9C79 C36F9D	0260	JP	GMEN
9C7C C3739D	0270	JP	GSMAY
9C7F C3779D	0280	JP	GSMEN
9C82 C37B9D	0290	JP	AMAY
9C85 C37F9D	0300	JP	AMEN
9C88 C3839D	0310	JP	ASMAY
9C8B C3879D	0320	JP	ASHEN

9C8E	C38B9D	0330	JP	BMAY
9C91	C38F9D	0340	JP	BMEN
9C94	00	0350	ESPAC	DEFB 0, 0, 0, 0
	00 00 00			
9C98		0360	NOMBTB	DEFM "CMAYO"
9C9D	D2	0370		DEFB "R"+80H
9C9E		0380		DEFM "CMENO"
9CA3	D2	0390		DEFB "R"+80H
9CA4		0400		DEFM "CSMAYO"
9CAA	D2	0410		DEFB "R"+80H
9CAB		0420		DEFM "CSMENO"
9CB1	D2	0430		DEFB "R"+80
9CB2		0440		DEFM "DMAYO"
9CB7	D2	0450		DEFB "R"+80H
9CB8		0460		DEFM "DMENO"
9CBD	D2	0470		DEFB "R"+80H
9CBE		0480		DEFM "DSMAYO"
9CC4	D2	0490		DEFB "R"+80H
9CC5		0500		DEFM "DSMENO"
9CCB	D2	0510		DEFB "R"+80H
9CCC		0520		DEFM "EMAYO"
9CD1	D2	0530		DEFB "R"+80H
9CD2		0540		DEFM "EMENO"
9CD7	D2	0550		DEFB "R"+80H
9CD8		0560		DEFM "FMAYO"
9CDD	D2	0570		DEFB "R"+80H
9CDE		0580		DEFM "FMENO"
9CE3	D2	0590		DEFB "R"+80H
9CE4		0600		DEFM "FSMAYO"
9CEA	D2	0610		DEFB "R"+80H
9CEB		0620		DEFM "FSMENO"
9CF1	D2	0630		DEFB "R"+80H
9CF2		0640		DEFM "GMAYO"
9CF7	D2	0650		DEFB "R"+80H

9CF8	0660		DEFM "GMENO"
9CFD D2	0670		DEFB "R"+80H
9CFE	0680		DEFM "GSMAYO"
9D04 D2	0690		DEFB "R"+80H
9D05	0700		DEFM "GSMENO"
9D0B D2	0710		DEFB "R"+80H
9D0C	0720		DEFM "AMAYO"
9D11 D2	0730		DEFB "R"+80H
9D12	0740		DEFM "AMENO"
9D17 D2	0750		DEFB "R"+80H
9D18	0760		DEFM "ASMAYO"
9D1E D2	0770		DEFB "R"+80H
9D1F "	0780		DEFM "ASMENO"
9D25 D2	0790		DEFB "R"+80H
9D26	0800		DEFM "BMAYO"
9D2B D2	0810		DEFB "R"+80H
9D2C	0820		DEFM "BMENO"
9D31 D2	0830		DEFB "R"+80H
9D32 00	0840		DEFB 0
9D33 1E00	0850	CMAY	LD E, 0
9D35 185C	0860		JR ACORD
9D37 1E06	0870	CMEN	LD E, 6
9D39 1858	0880		JR ACORD
9D3B 1E0C	0890	CSMAY	LD E, 12
9D3D 1854	0900		JR ACORD
9D3F 1E12	0910	CSMEN	LD E, 18
9D41 1850	0920		JR ACORD
9D43 1E18	0930	DMAY	LD E, 24
9D45 184C	0940		JR ACORD
9D47 1E1E	0950	DMEN	LD E, 30
9D49 1848	0960		JR ACORD
9D4B 1E24	0970	DSMAY	LD E, 36
9D4D 1844	0980		JR ACORD
9D4F 1E2A	0990	DSMEN	LD E, 42

9D51	1840	1000	JR	ACORD
9D53	1E30	1010	EMAY	LD E, 48
9D55	183C	1020	JR	ACORD
9D57	1E36	1030	EMEN	LD E, 54
9D59	1838	1040	JR	ACORD
9D5B	1E3C	1050	FMAY	LD E, 60
9D5D	1834	1060	JR	ACORD
9D5F	1E42	1070	FMEN	LD E, 66
9D61	1830	1080	JR	ACORD
9D63	1E48	1090	FSMAY	LD E, 72
9D65	182C	1100	JR	ACORD
9D67	1E4E	1110	FSMEN	LD E, 78
9D69	1828	1120	JR	ACORD
9D6B	1E54	1130	GMAY	LD E, 84
9D6D	1824	1140	JR	ACORD
9D6F	1E5A	1150	GMEN	LD E, 90
9D71	1820	1160	JR	ACORD
9D73	1E60	1170	GSMAY	LD E, 96
9D75	181C	1180	JR	ACORD
9D77	1E66	1190	GSMEN	LD E, 102
9D79	1818	1200	JR	ACORD
9D7B	1E6C	1210	AMAY	LD E, 108
9D7D	1814	1220	JR	ACORD
9D7F	1E72	1230	AMEN	LD E, 114
9D81	1810	1240	JR	ACORD
9D83	1E78	1250	ASMAY	LD E, 120
9D85	180C	1260	JR	ACORD
9D87	1E7E	1270	ASMEN	LD E, 126
9D89	1808	1280	JR	ACORD
9D8B	1E84	1290	BMAY	LD E, 132
9D8D	1804	1300	JR	ACORD
9D8F	1E8A	1310	BMEN	LD E, 138
9D91	1800	1320	JR	ACORD
9D93	211A9E	1330	ACORD	LD HL, DATOS

9D96	1600	1340	LD	D, 0
9D98	19	1350	ADD	HL, DE
9D99	5E	1360	LD	E, (HL)
9D9A	23	1370	INC	HL
9D9B	56	1380	LD	D, (HL)
9D9C	23	1390	INC	HL
9D9D	ED53029E	1400	LD	(TON01), DE
9DA1	5E	1410	LD	E, (HL)
9DA2	23	1420	INC	HL
9DA3	56	1430	LD	D, (HL)
9DA4	23	1440	INC	HL
9DA5	ED530B9E	1450	LD	(TON02), DE
9DA9	5E	1460	LD	E, (HL)
9DAA	23	1470	INC	HL
9DAB	56	1480	LD	D, (HL)
9DAC	ED53149E	1490	LD	(TON03), DE
9DB0	F5	1500	PUSH	AF
9DB1	B7	1510	OR	A
9DB2	2803	1520	JR	Z, PONENV
9DB4	DD7E00	1530	LD	A, (IX+0)
9DB7	32009E	1540	PONENV LD	(ENV1), A
9DBA	32099E	1550	LD	(ENV2), A
9DBD	32129E	1560	LD	(ENV3), A
9DC0	B7	1570	OR	A
9DC1	2812	1580	JR	Z, NEGTV0
9DC3	CDC2BC	1590	BSCENV CALL	AMPDIP
9DC6	D0	1600	RET	NC
9DC7	23	1610	INC	HL
9DC8	23	1620	INC	HL
9DC9	7E	1630	LD	A, (HL)
9DCA	B7	1640	OR	A
9DCB	2808	1650	JR	Z, NEGTV0
9DCD	CB7F	1660	BIT	7, A
9DCF	2004	1670	JR	NZ, NEGTV0

9DD1	3E00	1680	LD	A, 0
9DD3	1802	1690	JR	PONAMP
9DD5	3E0F	1700	NEGTV0	LD A, 15
9DD7	32059E	1710	PONAMP	LD (AMPL1), A
9DDA	320E9E	1720	LD	(AMPL2), A
9DDD	32179E	1730	LD	(AMPL3), A
9DE0	F1	1740	POP	AF
9DE1	FE02	1750	CP	2
9DE3	0CA7BC	1760	CALL	Z, SNDPES
9DE6	21FF9D	1770	LD	HL, SND1
9DE9	0DAABC	1780	ACOMP	CALL SUMSND
9DEC	30FB	1790	JR	NC, ACOMP
9DEE	21089E	1800	LD	HL, SND2
9DF1	0DAABC	1810	BCOMP	CALL SUMSND
9DF4	30FB	1820	JR	NC, BCOMP
9DF6	21119E	1830	LD	HL, SND3
9DF9	0DAABC	1840	CCOMP	CALL SUMSND
9DFC	30FB	1850	JR	NC, CCOMP
9DFE	09	1860	RETF	
9DFF	31	1870	SND1	DEFB 49
9E00	00	1880	ENV1	DEFB 0, 0
	00			
9E02	00	1890	TON01	DEFB 0, 0, 0
	00 00			
9E05	00	1900	AMPL1	DEFB 0, 0, 0
	00 00			
9E08	2A	1910	SND2	DEFB 42
9E09	00	1920	ENV2	DEFB 0, 0
	00			
9E0B	00	1930	TON02	DEFB 0, 0, 0
	00 00			
9E0E	00	1940	AMPL2	DEFB 0, 0, 0
	00 00			
9E11	1C	1950	SND3	DEFB 28

9E12	00	1960	ENV3	DEFB	0,0
	00				
9E14	00	1970	TON03	DEFB	0,0,0
	00				00
9E17	00	1980	AMPL3	DEFB	0,0,0
	00				00
9E1A	7E02	1990	DAT05	DEFW	638,478,379
	DE01		7B01		
9E20	7E02	2000		DEFW	638,478,402
	DE01		9201		
9E26	5A02	2010		DEFW	602,451,358
	C301		6601		
9E2C	5A02	2020		DEFW	602,451,379
	C301		7B01		
9E32	A402	2030		DEFW	676,568,426
	3802		AA01		
9E38	3802	2040		DEFW	568,426,358
	AA01		6601		
9E3E	7E02	2050		DEFW	638,536,402
	1802		9201		
9E44	A402	2060		DEFW	676,536,402
	1802		9201		
9E4A	5A02	2070		DEFW	602,506,379
	FA01		7B01		
9E50	7E02	2080		DEFW	638,506,379
	FA01		7B01		
9E56	3802	2090		DEFW	568,478,358
	DE01		6601		
9E5C	5A02	2100		DEFW	602,478,358
	DE01		6601		
9E62	A402	2110		DEFW	676,536,451
	1802		C301		
9E68	A402	2120		DEFW	676,568,451
	3802		C301		

9E6E	7E02	2130	DEFW 638,506,426
	FA01	AA01	
9E74	7E02	2140	DEFW 638,536,426
	1802	AA01	
9E7A	5A02	2150	DEFW 602,478,402
	DE01	9201	
9E80	5A02	2160	DEFW 602,506,402
	FA01	9201	
9E86	3802	2170	DEFW 568,451,379
	C301	7B01	
9E8C	3802	2180	DEFW 568,478,379
	DE01	7B01	
9E92	1802	2190	DEFW 536,426,358
	AA01	6601	
9E98	1802	2200	DEFW 536,451,358
	C301	6601	
9E9E	A402	2210	DEFW 676,506,402
	FA01	9201	
9EA4	A402	2220	DEFW 676,506,426
	FA01	AA01	
		2230	END

### Organo de Acordes

No podemos dejar esta rutina sin proporcionar un programa de órgano de acordes. Sin embargo, no usaremos líneas como `IF INKEY$="A" THEN !AMAYOR`, como hemos hecho con otros programas para controlar interrupciones y envolventes. Nuestro programa interpreta todos los acordes principales, notas naturales, agudas y graves.

Aquí tenemos el 'teclado':

	2	3		5	6	7	
	Q	W	E	R	T	Y	U
							I

Y estas son las notas que representan:

C#	D#	F#	G#	A#			
C	D	E	F	G	A	B	C

```
10 ENV 1,10,0,100
20 DIM k(12):m=-1
30 FOR n=0 TO 12:READ k(n):NEXT
40 n=0
50 a=INKEY(k(n))
60 IF a=0 AND n=m GOTO 40
70 IF a=0 GOTO 120
80 n=n+1
90 IF n<13 GOTO 50
100 SOUND 135,0,0,0:m=-1
110 GOTO 40
120 ON n+1 GOSUB 140,150,160,170,180,190,200,210,
    220,230,240,250,260
130 m=n:GOTO 40
140 :CMAYOR,0,1:RETURN
150 :DMAYOR,0,1:RETURN
160 :EMAYOR,0,1:RETURN
170 :FMAYOR,0,1:RETURN
180 :GMAYOR,0,1:RETURN
190 :AMAYOR,0,1:RETURN
200 :BMAYOR,0,1:RETURN
210 :CMAYOR,0,1:RETURN
220 :CSMAYOR,0,1:RETURN
230 :DSMAYOR,0,1:RETURN
240 :FSMAYOR,0,1:RETURN
250 :GSMAYOR,0,1:RETURN
260 :ASMAYOR,0,1:RETURN
270 DATA 67,59,58,50,51,43,42,35,65,57,49,48,41
```

# DOCE - Compresores de Pantalla

Estas dos rutinas de compresión de pantallas solamente funcionan en los modelos Amstrad que usan cinta. Debido a la dirección donde almacenan las pantallas comprimidas, los comandos de manejo de discos resultan afectados por lo que no es posible salvar el resultado de la compresión mas que a cinta.

## Compresor 1

La pantalla del Amstrad se encuentra almacenada en 16K de memoria. Es mucha memoria RAM para almacenar lo que suele ser una simple imagen de pantalla. La mayoría de la pantalla se haya rellena de espacios en blanco y tiene el valor 0. Una sección de pantalla de 20 puntos se representa en la RAM como veinte ceros. Se desperdicia gran cantidad de memoria. Nuestra rutina compresora hace un mejor uso de la memoria de la pantalla.

Este programa mira primero una sección de la RAM de pantalla. Si tiene un valor distinto de cero, lo almacena tal como está, sin alteraciones. Si el valor es 0, cuenta la cantidad de ellos que hay consecutivos y almacena un solo cero con un segundo valor que indica el número de ellos que ha contado antes de encontrar otro carácter distinto.

```
1      COMPRESOR 1
10 SYMBOL AFTER 256:MEMORY 20000:SYMBOL AFTER 240
20 FOR n=43800 TO 43888
30 READ a$:POKE n,VAL("&" +a$)
40 NEXT
50 DATA 11,00,C0,21,17,AB,7E,12,13,2B
60 DATA CB,7A,C8,B7,20,F6,46,2B,05,28
70 DATA F1,12,13,10,FC,CB,7A,20,E9,C9
80 DATA FE,01,C0,11,00,C0,21,17,AB,1A
90 DATA 77,13,2B,CB,7A,28,18,B7,20,F5
```

```

100 DATA 06,01,1A,B7,20,08,13,CB,7A,28
110 DATA 07,04,20,F4,70,2B,18,E3,04,70
120 DATA 2B,EB,21,17,AB,B7,ED,52,EB,DD
130 DATA 6E,00,DD,66,01,73,23,72,C9

```

Para comprimir una pantalla, use los siguientes comandos:

```
A%=0:CALL 43830.@A%
```

En la variable A% devuelve el número de octetos que ocupa la pantalla, de forma que pueda calcular la cantidad de memoria ahorrada. (Esto también es aplicable al segundo programa compresor que sigue a este).

Para retornar la pantalla a su estado inicial, teclee:

```
CALL 43800
```

Para salvar una pantalla, comprímala primero de la forma explicada, y teclee después:

```
SAVE "NOMBRE",B,43800-A%,A%+30
```

Para cargarla y mostrarla, haga un LOAD "" y después CALL 43800.

Cuando salve la pantalla, debe salvar también la rutina de compresión.

```

                                0001 ;           COMPRESOR 1
                                0002 ;
AB18                            0010           ORG 43800
                                0020 ;           Rutina1
AB18 1100C0                     0030           LD DE,0C00H
AB1B 2117AB                     0040           LD HL,43799
AB1E 7E                          0050 UNCMPT LD A,(HL)

```

AB1F 12	0060	LD	(DE), A
AB20 13	0070	INC	DE
AB21 2B	0080	DEC	HL
AB22 CB7A	0090	BIT	7, D
AB24 C8	0100	RET	Z
AB25 B7	0110	OR	A
AB26 20F6	0120	JR	NZ, UNCMPT
AB28 46	0130	LD	B, (HL)
AB29 2B	0140	DEC	HL
AB2A 05	0150	DEC	B
AB2B 28F1	0160	JR	Z, UNCMPT
AB2D 12	0170	REPEAT LD	(DE), A
AB2E 13	0180	INC	DE
AB2F 10FC	0190	DJNZ	REPEAT
AB31 CB7A	0200	BIT	7, D
AB33 20E9	0210	JR	NZ, UNCMPT
AB35 C9	0220	PET	
	0230 ;		RUTINA 2
AB36 FE01	0240	CP	1
AB38 C0	0250	RET	NZ
AB39 1100C0	0260	LD	DE, 00C00H
AB3C 2117AB	0270	LD	HL, 43799
AB3F 1A	0280	COMPCT LD	A, (DE)
AB40 77	0290	LD	(HL), A
AB41 13	0300	INC	DE
AB42 2B	0310	DEC	HL
AB43 CB7A	0320	BIT	7, D
AB45 2818	0330	JR	Z, TOTAL
AB47 B7	0340	OR	A
AB48 20F5	0350	JR	NZ, COMPCT
AB4A 0601	0360	LD	B, 1
AB4C 1A	0370	COUNT LD	A, (DE)
AB4D B7	0380	OR	A
AB4E 2008	0390	JR	NZ, GOTLEN

AB50 13	0400	INC	DE
AB51 CB7A	0410	BIT	7,D
AB53 2907	0420	JR	Z, LAST
AB55 04	0430	INC	B
AB56 20F4	0440	JR	NZ, COUNT
AB58 70	0450	GOTLEN LD	(HL), B
AB59 2B	0460	DEC	HL
AB5A 18E3	0470	JR	COMPCT
AB5C 04	0480	LAST INC	B
AB5D 70	0490	LD	(HL), B
AB5E 2B	0500	DEC	HL
AB5F EB	0510	TOTAL EX	DE, HL
AB60 2117AB	0520	LD	HL, 43799
AB63 B7	0530	OR	A
AB64 ED52	0540	SBC	HL, DE
AB66 EB	0550	EX	DE, HL
AB67 DD6E00	0560	LD	L, (IX+0)
AB6A DD6601	0570	LD	H, (IX+1)
AB6D 73	0580	LD	(HL), E
AB6E 23	0590	INC	HL
AB6F 72	0600	LD	(HL), D
AB70 C9	0610	RET	
	0620	END	

## Compresor 2

A diferencia de la rutina anterior, esta comprime **todos** los caracteres que se encuentra repetidos, aunque no sean ceros. Si, por ejemplo, una porción de la pantalla es:

3,160,2,2,2,2,2,2,8,8,8,8

La rutina la almacena así:

3,1,160,1,2,6,8,4

El mayor problema de esta rutina es que si la pantalla es demasiado complicada, no comprime nada y puede que ocupe más de las 16K usadas para almacenar la pantalla en circunstancias normales. Sin embargo, como funciona con la mayoría de las pantallas, puede intentar usarla primero.

```
1          COMPRESOR 2
10 SYMBOL AFTER 256:MEMORY 20000:SYMBOL AFTER 240
20 FOR n=43800 TO 43875
30 READ a$:POKE n,VAL("&" +a$)
40 NEXT
50 DATA 11,00,C0,21,17,AB,7E,2B,46,2B
60 DATA 12,13,10,FC,CB,7A,20,F4,C9,FE
70 DATA 01,C0,11,00,C0,21,17,AB,1A,77
80 DATA 13,2B,06,01,CB,7A,28,12,4F,1A
90 DATA B9,20,08,13,CB,7A,28,07,04,20
100 DATA F4,70,2B,18,E5,04,70,2B,EB,21
110 DATA 17,AB,B7,ED,52,EB,DD,6E,00,DD
120 DATA 66,01,73,23,72,C9
```

Para comprimir la pantalla:

```
A%=0:CALL 43819,@A%
```

Para descomprimirla:

```
CALL 43800
```

Para salvar una pantalla y el descompresor (necesario para mostrar la pantalla de forma normal) teclee:

```
SAVE "NOMBRE",B,43800-A%,A%+19
```

En este caso B no es una variable, sino el signo que le

dice al ordenador que está salvando un fichero binario.

```

                                0001 ;           COMPRESOR 2
                                0002 ;
AB18                            0010          ORG  43800
                                0020 ;           RUTINA1
AB18 1100C0                      0030          LD  DE,0C000H
AB1B 2117AB                      0040          LD  HL,43799
AB1E 7E                          0050  UNCMPT LD  A,(HL)
AB1F 2B                          0060          DEC  HL
AB20 46                          0070          LD  B,(HL)
AB21 2B                          0080          DEC  HL
AB22 12                          0090  REPEAT LD  (DE),A
AB23 13                          0100          INC  DE
AB24 10FC                        0110          DJNZ REPEAT
AB26 CB7A                        0120          BIT  7,D
AB28 20F4                        0130          JR   NZ,UNCMPT
AB2A C9                          0140          RET
                                0150 ;           RUTINA 1
AB2B FE01                        0160          CP   1
AB2D C0                          0170          RET  NZ
AB2E 1100C0                      0180          LD  DE,0C000H
AB31 2117AB                      0190          LD  HL,43799
AB34 1A                          0200  COMPCT LD  A,(DE)
AB35 77                          0210          LD  (HL),A
AB36 13                          0220          INC  DE
AB37 2B                          0230          DEC  HL
AB38 0601                        0240          LD  B,1
AB3A CB7A                        0250          BIT  7,D
AB3C 2812                        0260          JR   Z,TOTAL
AB3E 4F                          0270          LD  C,A
AB3F 1A                          0280  COUNT LD  A,(DE)
AB40 B9                          0290          CP   C
```

AB41	2008	0300	JR	NZ, GOTLEN
AB43	13	0310	INC	DE
AB44	CB7A	0320	BIT	7, D
AB46	2807	0330	JR	Z, LAST
AB48	04	0340	INC	B
AB49	20F4	0350	JR	NZ, COUNT
AB4B	70	0360	GOTLEN LD	(HL), B
AB4C	2B	0370	DEC	HL
AB4D	18E5	0380	JR	COMPCT
AB4F	04	0390	LAST INC	B
AB50	70	0400	TOTAL LD	(HL), B
AB51	2B	0410	DEC	HL
AB52	EB	0420	EX	DE, HL
AB53	2117AB	0430	LD	HL, 43799
AB56	B7	0440	OR	A
AB57	ED52	0450	SBC	HL, DE
AB59	EB	0460	EX	DE, HL
AB5A	DD6E00	0470	LD	L, (IX+0)
AB5D	DD6601	0480	LD	H, (IX+1)
AB60	73	0490	LD	(HL), E
AB61	23	0500	INC	HL
AB62	72	0510	LD	(HL), D
AB63	09	0520	RET	
		0530	END	

## TRECE - DOKE y DEEK

Cuando se programa en código máquina se suele hacer PEEK y POKE de números de 16 bits. Mientras que los números de 8 bits tienen un rango de 256, los de 16 bits lo tienen de 65535. Los números de 16 bits usan dos octetos, y para hacer un POKE de un número de dos octetos, se debe usar esta pequeña fórmula:

$$\text{POKE octeto1} + 256 * \text{octeto2}$$

Esto mismo se aplica cuando se hace un PEEK de dos octetos de una posición de memoria. Algunos ordenadores tiene suficiente suerte como para tener comandos que pueden hacer Doble-PEEK y Doble-POKE, conocidos como DEEK y DOKE. Esta rutina proporciona dos comandos nuevos de BASIC, !DOKE y !DEEK.

```
1          DOKE/DEEK
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240
20 FOR n=40000 TO 40071
30 READ a$:POKE n,VAL("&"a$)
40 NEXT
50 DATA 01,4A,9C,21,52,9C,CD,D1,BC,C9
60 DATA 56,9C,C3,5F,9C,C3,72,9C,00,00
70 DATA 00,00,44,4F,4B,C5,44,45,45,CB
80 DATA 00,FE,02,C0,DD,6E,02,DD,66,03
90 DATA DD,7E,00,77,23,DD,7E,01,77,C9
100 DATA FE,02,C0,DD,6E,02,DD,66,03,DD
110 DATA 5E,00,DD,56,01,7E,23,12,13,7E
120 DATA 12,C9
```

Primero debe hacer una CALL 40000 para preparar los nuevos comandos, para usar !DOKE, debe ir seguido por una dirección

válida de memoria y por un número de 16 bits. Los dos valores deben ser decimales. !DOKE,30000,343 dividirá el valor 343 en dos usando la fórmula inversa a la anterior y colocando los dos valores en las posiciones de memoria 30000 y 30001.

!DEEK es un poco más complicada. Se usa con el formato:

N%=0: !DEEK, A, @N%

A es la primera dirección de la que se va a hacer PEEK (!DEEK mirará en A y A+1) mientras que N% contiene el valor retornado por el !DEEK. N% se debe definir antes del !DEEK. Como puede ver por el signo %, debe ser una variable entera. Recuérdelo si quiere imprimir el valor real de DEEK. Si el valor devuelto es negativo, necesitará imprimir 65536+N% para obtener el valor real.

```

0001 ; DOKE DEEK
0002 ;
9C40 0010 ORG 40000
BCD1 0020 LOGEXT DEFL 0BCD1H
9C40 014A9C 0030 LD BC, TABLA
9C43 21529C 0040 LD HL, ESPACI
9C46 CDD1BC 0050 CALL LOGEXT
9C49 C9 0060 RET
9C4A 569C 0070 TABLA DEFW NOMBTB
9C4C C35F9C 0080 JP DOKE
9C4F C3729C 0090 JP DEEK
9C52 00 0100 ESPACI DEFB 0, 0, 0, 0
00 00 00
9C56 0110 NOMBTB DEFM "DOK"
9C59 C5 0120 DEFB "E"+80H
9C5A 0130 DEFW "DEE"
9C5D CB 0140 DEFB "K"+80H
9C5E 00 0150 DEFB 0
9C5F FE02 0160 DOKE CP 2

```

9C61	C0	0170		RET	NZ
9C62	DD6E02	0180		LD	L, (IX+2)
9C65	DD6603	0190		LD	H, (IX+3)
9C68	DD7E00	0200		LD	A, (IX+0)
9C6B	77	0210		LD	(HL), A
9C6C	23	0220		INC	HL
9C6D	DD7E01	0230		LD	A, (IX+1)
9C70	77	0240		LD	(HL), A
9C71	C9	0250		RET	
9C72	FE02	0260	DEEK	CP	2
9C74	C0	0270		RET	NZ
9C75	DD6E02	0280		LD	L, (IX+2)
9C78	DD6603	0290		LD	H, (IX+3)
9C7B	DD5E00	0300		LD	E, (IX+0)
9C7E	DD5601	0310		LD	D, (IX+1)
9C81	7E	0320		LD	A, (HL)
9C82	23	0330		INC	HL
9C83	12	0340		LD	(DE), A
9C84	13	0350		INC	DE
9C85	7E	0360		LD	A, (HL)
9C86	12	0370		LD	(DE), A
9C87	C9	0380		RET	
		0390		END	

# CATORCE - El Paquete de Escritura de Juegos

Para terminar este libro tenemos un programa que nos proporciona una selección de las rutinas en código máquina en un solo paquete fácil de usar. La rutina crea nuevas palabras clave de BASIC, usando las llamadas RSX.

```
:EXPLODE, :READCHAR, :BIGPRINT, :USCROLL,  
:DSCROLL, :LSCROLL, :RSCROLL, :COMMANDS
```

Veámoslas por turno. EXPLODE no necesita ningún parámetro. Simplemente crea un sonido de explosión, muy usado en los programas de juegos. READCHAR y BIGPRINT son dos comandos que corresponden al título de las rutinas que hemos visto en este libro. BIGPRINT crea caracteres de doble tamaño mientras que READCHAR calcula el carácter ASCII en una posición de pantalla determinada.

USCROLL, DSCROLL, LSCROLL y RSCROLL son las cuatro rutinas de movimiento de pantalla que hemos visto anteriormente, con la primera letra indicando la dirección del movimiento. COMMANDS nos proporciona una lista de los comandos anteriores.

Para comprender cómo funciona cada uno y los parámetros que necesitan, debe dirigirse a cada rutina en particular.

Este paquete ofrece la mayor parte de las rutinas en código máquina necesarias para escribir un buen juego "híbrido", parte en BASIC y parte en código máquina.

```
1 PAQUETE ESCRITURA DE JUEGOS  
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 240  
20 DIM x(12)  
30 FOR c=0 TO 12:READ a$:x(c)=VAL("&" + a$):NEXT  
40 c=0:sum=0
```

```

50 FOR n=40000 TO 41207
60 READ a$:v=VAL("&"a$)
70 sum=sum+v:POKE n,v
80 IF (n+1-40000) MOD 100<>0 GOTO 110
90 IF sum<>x(c) THEN PRINT "*DATOS ERRONEOS*";CHR$(7):
PRINT "Compruebe las líneas ";210+100*c;" a ";
300+100*c:END
100 sum=0:c=c+1
110 NEXT
120 IF sum<>x(c) THEN PRINT "*DATOS EPRONEOS*";CHR$(7):
PRINT "Compruebe la línea 1410"
130 '
140 '          DATOS DE VERIFICACION
150 '
160 DATA 2B79,2F09,2A7A,27A3,263B,2434,2C42
170 DATA 2782,2593,2A89,278C,29D3,04D0
180 '
190 '          CODIGO MAQUINA
200 '
210 DATA 01,4A,9C,21,64,9C,CD,D1,BC,C9
220 DATA 68,9C,C3,A4,9C,C3,C3,9C,C3,D7
230 DATA 9C,C3,77,9D,C3,09,9E,C3,A1,9E
240 DATA C3,B1,9F,C3,C7,A0,00,00,00,00
250 DATA 45,58,50,4C,4F,44,C5,52,45,41
260 DATA 44,43,48,41,D2,42,49,47,50,52
270 DATA 49,4E,D4,55,53,43,52,4F,4C,CC
280 DATA 44,53,43,52,4F,4C,CC,40,53,43
290 DATA 52,4F,4C,CC,52,53,43,52,4F,4C
300 DATA CC,43,4F,4D,4D,41,4E,44,D3,00
310 DATA CD,A7,BC,3E,01,21,B6,9C,CD,BC
320 DATA BC,21,BA,9C,CD,AA,BC,C9,01,0F
330 DATA FF,19,01,01,00,00,00,0F,0F,00
340 DATA 00,DD,6E,02,DD,66,04,CD,75,BB
350 DATA CD,60,BB,DD,6E,00,DD,66,01,77

```

360 DATA C9, CD, 93, BB, F5, DD, 6E, 04, DD, 66  
370 DATA 05, 46, 23, 5E, 23, 56, DD, 6E, 06, DD  
380 DATA 66, 08, C5, D5, E5, 1A, 47, CD, 06, B9  
390 DATA 78, CD, 34, 9D, 47, CD, 09, B9, E1, 5D  
400 DATA 54, CD, 75, BB, DD, 7E, 02, CD, 90, BB  
410 DATA 78, CD, 5A, BB, 3C, CD, 5A, BB, 6B, 62  
420 DATA 2C, CD, 75, BB, DD, 7E, 00, CD, 90, BB  
430 DATA 78, 3C, 3C, CD, 5A, BB, 3C, CD, 5A, BB  
440 DATA 6B, 62, 24, 24, D1, 13, C1, 10, 8D, F1  
450 DATA CD, 90, BB, C9, CD, A5, BB, EB, CD, AE  
460 DATA BB, F5, 0E, 02, 06, 04, C5, 1A, 0F, 0F  
470 DATA 0F, 0F, 06, 04, 1F, CB, 1E, CB, 2E, 10  
480 DATA F9, 7E, 23, 77, 06, 07, 23, 10, FD, 1A  
490 DATA 06, 04, 1F, CB, 1E, CB, 2E, 10, F9, 7E  
500 DATA 23, 77, 06, 07, 2B, 10, FD, 13, C1, 10  
510 DATA D3, 06, 08, 23, 10, FD, 0D, 20, C9, F1  
520 DATA C9, DD, 6E, 02, DD, 66, 06, 2D, 25, DD  
530 DATA 7E, 04, 94, 4F, DD, 7E, 00, 95, 57, CD  
540 DATA 1A, BC, 3E, 00, 81, 10, FD, 5F, 18, 38  
550 DATA D5, 43, 5D, 7C, C6, 38, 57, 7D, C6, 50  
560 DATA 6F, 30, 0A, 24, 7C, E6, 07, 20, 04, 7C  
570 DATA D6, 08, 67, E5, 7E, 12, 1C, 20, 0A, 14  
580 DATA 7A, E6, 07, 20, 04, 7A, D6, 08, 57, 2C  
590 DATA 20, 0A, 24, 7C, E6, 07, 20, 04, 7C, D6  
600 DATA 08, 67, 10, E2, E1, D1, D5, E5, 4B, E5  
610 DATA 06, 07, 5D, 54, 7C, C6, 08, 67, 7E, 12  
620 DATA 10, F6, E1, 2C, 20, 0A, 24, 7C, E6, 07  
630 DATA 20, 04, 7C, D6, 08, 67, 0D, 20, E2, E1  
640 DATA D1, 15, 20, A2, 7C, C6, 38, 67, 43, 36  
650 DATA 00, 2C, 20, 0A, 24, 7C, E6, 07, 20, 04  
660 DATA 7C, D6, 08, 67, 10, EF, C9, DD, 6E, 00  
670 DATA DD, 66, 06, 2D, 25, DD, 7E, 04, 94, 4F  
680 DATA 7D, C6, 02, DD, 96, 02, 57, CD, 1A, BC  
690 DATA 7C, C6, 38, 67, 3E, 00, 81, 10, FD, 5F

700 DATA 18, 38, D5, 43, 5D, 7C, D6, 38, 57, 7D  
710 DATA D6, 50, 6F, 30, 0A, 7C, 25, E6, 07, 20  
720 DATA 04, 7C, C6, 08, 67, E5, 7E, 12, 1C, 20  
730 DATA 0A, 14, 7A, E6, 07, 20, 04, 7A, D6, 08  
740 DATA 57, 2C, 20, 0A, 24, 7C, E6, 07, 20, 04  
750 DATA 7C, D6, 08, 67, 10, E2, E1, D1, D5, E5,  
760 DATA 48, E5, 06, 07, 5D, 54, 7C, D6, 08, 67  
770 DATA 7E, 12, 10, F6, E1, 2C, 20, 0A, 24, 7C  
780 DATA E6, 07, 20, 04, 7C, D6, 08, 67, 0D, 20  
790 DATA E2, E1, D1, 15, 20, A2, 7C, D6, 38, 67  
800 DATA 43, 36, 00, 2C, 20, 0A, 24, 7C, E6, 07  
810 DATA 20, 04, 7C, D6, 08, 67, 10, EF, C9, DD  
820 DATA 6E, 02, DD, 66, 06, 2D, 25, DD, 7E, 04  
830 DATA 94, 4F, DD, 7E, 00, 95, 57, CD, 1A, BC  
840 DATA 3E, 00, 81, 10, FD, 47, CD, 11, BC, DA  
850 DATA 5B, 9F, 28, 48, DD, 6E, 02, DD, 66, 04  
860 DATA 2D, 25, C5, CD, 1A, BC, C1, 0E, 08, C5  
870 DATA E5, B7, CB, 16, F5, 7D, 2D, B7, 20, 0A  
880 DATA 7C, 25, E6, 07, 20, 04, 7C, C6, 08, 67  
890 DATA F1, 10, EB, E1, 7C, C6, 08, 67, C1, 0D  
900 DATA 20, DF, 7C, D6, 40, 67, 7D, C6, 50, 6F  
910 DATA 30, 0A, 24, 7C, E6, 07, 20, 04, 7C, D6  
920 DATA 08, 67, 15, 20, C6, C9, 05, D5, 0E, 08  
930 DATA C5, E5, CB, 26, 5D, 54, 2C, 20, 0A, 24  
940 DATA 7C, E6, 07, 20, 04, 7C, D6, 08, 67, 1A  
950 DATA CB, 26, 38, 04, CB, A7, 18, 02, CB, E7  
960 DATA CB, 66, 28, 02, CB, C7, 12, 10, DD, CB  
970 DATA A6, E1, 7C, C6, 08, 67, C1, 0D, 20, CE  
980 DATA 7C, D6, 40, 67, 7D, C6, 50, 6F, 30, 0A  
990 DATA 24, 7C, E6, 07, 20, 04, 7C, D6, 08, 67  
1000 DATA D1, 15, 20, B3, C9, 05, D5, 0E, 08, C5  
1010 DATA E5, C5, 5D, 54, 2C, 20, 0A, 24, 7C, E6  
1020 DATA 07, 20, 04, 7C, D6, 08, 67, 7E, 1F, 4F  
1030 DATA 1A, 17, 17, 06, 04, 17, CB, 21, CB, 21

1040 DATA 17, 10, F8, 12, C1, 10, DC, 7E, 17, 06  
1050 DATA 04, 07, CB, 27, 10, FB, 77, E1, 7C, C6  
1060 DATA 08, 67, C1, 0D, 20, C7, 7C, D6, 40, 67  
1070 DATA 7D, C6, 50, 6F, 30, 0A, 24, 7C, E6, 07  
1080 DATA 20, 04, 7C, D6, 08, 67, D1, 15, 20, AC  
1090 DATA C9, DD, 6E, 02, DD, 66, 04, 2D, 25, 7C  
1100 DATA C6, 02, DD, 96, 06, 4F, DD, 7E, 00, 95  
1110 DATA 57, CD, 1A, BC, 28, 3E, 00, 81, 23, 10  
1120 DATA FC, 47, CD, 11, BC, DA, 6F, A0, 28, 46  
1130 DATA DD, 6E, 02, DD, 66, 06, 2D, 25, C5, CD  
1140 DATA 1A, BC, C1, 0E, 08, C5, E5, B7, CB, 1E  
1150 DATA F5, 2C, 20, 0A, 24, 7C, E6, 07, 20, 04  
1160 DATA 7C, D6, 08, 67, F1, 10, ED, E1, 7C, C6  
1170 DATA 08, 67, C1, 0D, 20, E1, 7C, D6, 40, 67  
1180 DATA 7D, C6, 50, 6F, 30, 0A, 24, 7C, E6, 07  
1190 DATA 20, 04, 7C, D6, 08, 67, 15, 20, C8, C9  
1200 DATA 05, D5, 0E, 08, C5, E5, CB, 3E, 5D, 54  
1210 DATA 7D, 2D, B7, 20, 0A, 7C, 25, E6, 07, 20  
1220 DATA 04, 7C, C6, 08, 67, 1A, CB, 3E, 38, 04  
1230 DATA CB, 9F, 18, 02, CB, DF, CB, 5E, 28, 02  
1240 DATA CB, FF, 12, 10, DB, CB, 9E, E1, 7C, C6  
1250 DATA 08, 67, C1, 0D, 20, CC, 7C, D6, 40, 67  
1260 DATA 7D, C6, 50, 6F, 30, 0A, 24, 7C, E6, 07  
1270 DATA 20, 04, 7C, D6, 08, 67, D1, 15, 20, B1  
1280 DATA C9, 05, D5, 0E, 08, C5, E5, C5, 5D, 54  
1290 DATA 7D, 2D, B7, 20, 0A, 7C, 25, E6, 07, 20  
1300 DATA 04, 7C, C6, 08, 67, 7E, 17, 4F, 1A, 1F  
1310 DATA 1F, 06, 04, 1F, CB, 39, CB, 39, 1F, 10  
1320 DATA F8, 12, C1, 10, DA, 7E, 1F, 06, 04, 0F  
1330 DATA CB, 3F, 10, FB, 77, E1, 7C, C6, 08, 67  
1340 DATA C1, 0D, 20, C5, 7C, D6, 40, 67, 7D, C6  
1350 DATA 50, 6F, 30, 0A, 24, 7C, E6, 07, 20, 04  
1360 DATA 7C, D6, 08, 67, D1, 15, 20, AA, C9, 21  
1370 DATA 68, 9C, 3E, 0D, CD, 5A, BB, 3E, 0A, CD

```
1380 DATA 5A, BB, 3E, 7C, CD, 5A, BB, 7E, 23, CB
1390 DATA 7F, 20, 05, CD, 5A, BB, 18, F5, CB, BF
1400 DATA CD, 5A, BB, 3E, 0D, CD, 5A, BB, 3E, 0A
1410 DATA CD, 5A, BB, 7E, B7, 20, DD, C9
```

Una vez teclado y comprobado el código máquina, se debe salvar con una sentencia como esta:

```
SAVE "mcjuego", B, 40000, 1208
```

### Bombardero

Para demostrar el uso de este paquete, tenemos el juego del Bombardero. Es, en nuestra imparcial opinión, una de las mejores versiones del popular juego del "bombardero que destruye los edificios para hacer una pista de aterrizaje", que hemos visto en el Amstrad. El programa incluye un diseño de avión hecho con ocho caracteres UDG, una gran bomba gráfica y muy buenos efectos de sonido.

El programa nos muestra lo que se puede conseguir en relativamente poco tiempo usando el paquete de escritura de juegos.

```
10 SYMBOL AFTER 256:MEMORY 39999:SYMBOL AFTER 236
20 LOAD "mcjuego", 40000
30 CALL 40000
40 ENT 1, 230, 1, 3, 230, 1, 3
50 ENT -2, 6, 2, 2, 4, -2, 2
60 RANDOMIZE TIME
70 SYMBOL 240, 120, 124, 62, 63, 31, 31, 63, 62
80 SYMBOL 241, 0, 0, 0, 0, 255, 255, 255, 253
90 SYMBOL 242, 0, 0, 0, 0, 224, 24, 206, 255
100 SYMBOL 243, 31, 60, 112, 0, 0, 0, 0, 0
110 SYMBOL 244, 251, 127, 15, 31, 31, 62, 60, 0
```

```

120 SYMBOL 245,251,255,240,192,128,0,0,0
130 SYMBOL 246,254,0,0,0,0,0,0,0
140 SYMBOL 247,231,90,231,129,166,36,60,126
150 SYMBOL 248,126,255,255,255,255,126,126,60
160 SYMBOL 249,255,255,241,241,255,241,241,255
170 SYMBOL 250,255,255,143,143,255,143,143,255
180 s$="      ": ' 4 ESPACIOS
190 a$=CHR$(240)+CHR$(241)+CHR$(241)+CHR$(242)
200 b$=CHR$(243)+CHR$(244)+CHR$(245)+CHR$(246)
210 INK 0,14:INK 1,0:INK 2,6:INK 3,24
220 BORDER 14:MODE 1
230 m$="Bombardero ":!BIGPRINT,10,1,@m$,3,2
240 FOR n=5 TO 37 STEP 2
250 x=15-INT(RND*10)
260 PEN 2+n/2 MOD 2
270 FOR y=x TO 25
280 LOCATE n,y:PRINT CHR$(249);CHR$(250)
290 NEXT: NEXT
300 ch%=0:cr%=0:PEN 1:LOCATE 1,1
310 PRINT a$:PRINT b$
320 SOUND 159,500,2000,3,0,0,1
330 v=1:w=1:c=1
340 IF w<23 AND INKEY(47)=0 GOTO 470
350 IF v=30 AND w=23 GOTO 1120
360 FOR n=1 TO 3:NEXT
370 IF c=1 THEN !READCHAR,v+4,w+1,@ch%:IF ch%=248
    GOTO 920
380 !RSCROLL,v,v+4,w,w+1
390 c=c+1:IF c=9 THEN c=1:v=v+1
400 IF v<37 GOTO 340
410 LOCATE 37,w:PRINT s$
420 LOCATE 37,w+1:PRINT s$
430 LOCATE 1,w+1:PRINT a$
440 LOCATE 1,w+2:PRINT b$

```

```

450 SOUND 129,500,2000,3,0,0,1
460 v=1:w=w+1:GOTO 340
470 d=c:x=v+2:y=w+2
480 !READCHAR,x,y,@ch%:IF ch%>32 THEN t=y:GOTO 710
490 !READCHAR,x,y+1,@ch%:!READCHAR,x,y+2,@cr%
500 SOUND 130,30,500,5,0,1
510 LOCATE x,y:PRINT CHR$(247):LOCATE x,y+1:PRINT
    CHR$(248)
520 IF ch%+cr%>64 GOTO 700
530 IF c=1 THEN !READCHAR,v+4,w+1,@ch%:IF ch%>248
    GOTO 920
540 !RSCROLL,v,v+4,w,w+1
550 !DSCROLL,x,x,y,y+2:!DSCROLL,x,x,y,y+2
560 c=c+1:IF c=9 THEN c=1:v=v+1
570 IF v<37 GOTO 640
580 LOCATE 37,w:PRINT s$
590 LOCATE 37,w+1:PRINT s$
600 LOCATE 1,w+1:PRINT a$
610 LOCATE 1,w+2:PRINT b$
620 SOUND 129,500,2000,3,0,0,1
630 v=1:w=w+1
640 b=y
650 y=y-(d=c)-(d-1=(c+3) MOD 8)
660 IF b<y THEN !READCHAR,x,y+2,@ch%:IF ch%>32 GOTO 700
670 IF y<23 GOTO 530
680 LOCATE x,y:PRINT " ":LOCATE x,y+1:PRINT " "
690 SOUND 130,0,0,0:GOTO 340
700 t=y+2:LOCATE x,y:PRINT " ":LOCATE x+(x MOD 2=0),
    y+1:PRINT " ":' 2 espacios
710 IF t<11 OR t<23 AND RND<0.7 THEN b=t+INT((24-t)/3)
    ELSE b=25
720 IF RND>0.6 OR b=25 AND RND>0.3 THEN q=4 ELSE q=8
730 m=(1+b-t)*(q-2):z=x+1
740 IF x MOD 2=0 THEN z=x:x=x-1

```

```

750 SOUND 130,100*a,800,15,0,2
760 FOR n=1 TO m
770 :DSCROLL,x,z,t,b
780 IF c=1 THEN :READCHAR,v+4,w+1,@ch%: IF ch%>248
    GOTO 920
790 :RSCROLL,v,v+4,w,w+1
800 IF a=4 THEN :DSCROLL,x,z,t,b
810 c=c+1: IF c=9 THEN c=1:v=v+1
820 IF v<37 GOTO 890
830 LOCATE 37,w:PRINT s$
840 LOCATE 37,w+1:PRINT s$
850 LOCATE 1,w+1:PRINT a$
860 LOCATE 1,w+2:PRINT b$
870 SOUND 129,500,2000,3,0,0,1
880 v=1:w=w+1
890 NEXT
900 SOUND 130,0,0,0
910 GOTO 340
920 SOUND 135,0,0,0
930 :EXPLODE: INK 2,6,24: INK 3,24,6: SPEED INK 2,2
940 IF w>23 GOTO 1050
950 t=w: b=t+2
960 :READCHAR,v,b,@ch%: IF ch%>32 GOTO 980
970 b=b+1: IF b<26 GOTO 960
980 b=b-1:m=(b-t-1)*8
990 FOR n=1 TO m: :DSCROLL,v,v+1,t,b:NEXT
1000 t=w: b=t+2
1010 :READCHAR,v+2,b,@ch%: IF ch%>32 GOTO 1030
1020 b=b+1: IF b<26 GOTO 1010
1030 b=b-1:m=(b-t-1)*8
1040 FOR n=1 TO m: :DSCROLL,v+2,v+3,t,b:NEXT
1050 PEN 1: FOR m=1 TO 2000: NEXT
1060 LOCATE 12,2: PRINT "Otro Juego? (S/N)"
1070 BORDER 1

```

```

1080 r$=UPPER$(INKEY$)
1090 IF r$="S" GOTO 210
1100 IF r$<>"N" GOTO 1080
1110 END
1120 FOR n=1 TO 88
1130 SOUND 135,100-n,2,7
1140 :RSCROLL,30,40,19,24
1150 IF n MOD 3=0 THEN :USCROLL,30,40,19,24
1160 NEXT
1170 PEN 2
1180 m$=" ENHORABUENA " : :BIGPRINT,6,12,@m$,2,3:GOTO 1050

```

La barra de espaciado hace caer las bombas, y el objetivo consiste en limpiar la pantalla de edificios antes de aterrizar, y despegar de nuevo. Cuando juegue con él, observe la suavidad del movimiento del avión a través de la pantalla e intente adivinar dónde se han usado los nuevos comandos.

# APENDICES

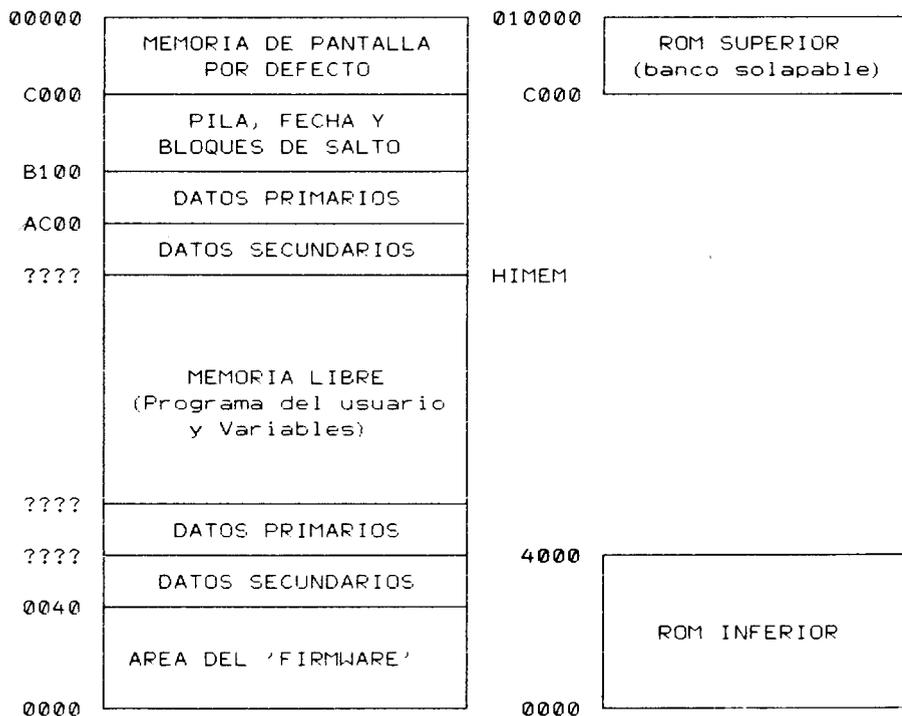
- A - Mapa de Memoria
- B - Códigos de Operación del Z80
- C - Conversión Hexadecimal a Decimal

# APENDICE A

## Mapa de Memoria

Debemos mencionar algunos puntos relacionados con el mapa de la memoria del Amstrad.

Primero, el mapa de memoria es bastante complicado, ya que deben operar 64K de RAM y 32K de ROM en los 64K de memoria disponible - de ahí el cambio de bancos de memoria. Segundo, todas las direcciones dadas en el mapa de la pantalla están en hexadecimal. Tercero, observe las direcciones de seis dígitos en lo alto del mapa de la pantalla. 010000 es la frontera superior y es, en efecto, uno más alto que la dirección real, que es FFFF.



## APENDICE B

En este apéndice le damos una lista completa de los códigos de operación e instrucciones del Z80.

El significado de los símbolos es:

Reg	A, B, C, D, E, H, L
Regpar	HL, BC, DE, SP
Modo	1, 2, 3
Operand1	A, B, C, D, E, H, L, (HL), (IX+des), (IY+des), datos
Operando	A, B, C, D, E, H, L, (HL), (IX+des), (IY+des)
Des	desplazamiento, un número entre 0 y 255
Datos	un número entre 0 y 255
Dir	una dirección entre 0 y 65535
Cc	condiciones Z, NZ, C, NC, P, M, PE, PO
C	condiciones Z, NZ, C, NC
Vector	dirección 0, 8, 16, 24, 32, 40, 48 o 56

## Sumario de Instrucciones y Funciones del Z80

ADC	A,operand1	;Suma un operando y el bit de acarreo ;al acumulador
ADC	HL,regpar	;Suma un par de registros y el bit de ;acarreo al acumulador
ADD	A,operand1	;Suma un operando al acumulador
ADD	HL,regpar	;Suma un par de registros a HL
ADD	IX,regpar	;Suma un par de registros a IX
ADD	IX,IX	;Suma IX a sí mismo
ADD	IY,regpar	;Suma un par de registros a IY
ADD	IY,IY	;Suma IY a sí mismo
AND	operand1	;AND lógico de un operando con el ;acumulador
BIT	b,operand1	;Comprueba el bit b del operando
CALL	dir	;Llama a una dirección
CALL	cc,dir	;Llama a una dirección si CC es válido
CCF		;Complementa el señalizador de acarreo
CP	operand1	;Compara el operando con el acumulador
CPD		;Compara (HL) con el acumulador, ;decrementa HL y BC
CPDR		;Compara (HL) con el acumulador, ;decrementa HL y BC, y repite hasta ;BC = 0 o coincidan los valores
CPI		;Compara (HL) con el acumulador, ;incrementa HL y decrementa BC

CPIR		;Compara (HL) con el acumulador, ;incrementa HL, decrementa BC, y repite ;hasta BC = 0 o coincidan los valores
CPL		;Complementa (invierte) el acumulador
DAA		;Ajuste decimal del acumulador
DEC	operando	;Decrementa el operando
DEC	IX	;Decrementa IX
DEC	IY	;Decrementa IY
DEC	SP	;Decrementa el apuntador de la pila
DI		;Inhabilita las interrupciones
DJNZ	des	;Decrementa B y salta al desplazamiento ;si B <> 0
EI		;Habilita las interrupciones
EX	(SP),HL	;Intercambia (SP) y HL
EX	(SP),IX	;Intercambia (SP) e IX
EX	(SP),IY	;Intercambia (SP) e IY
EX	AF,AF'	;Intercambia AF con los alternados AF
EX	DE,HL	;Intercambia DE con HL
EXX		;Intercambia los registro generales con ;los alternados
HALT		;Suspende las operaciones de la CPU
IM	modo	;Activa el modo de interrupción
IN	A, (dato)	;Introduce un dato en el acumulador ;desde la puerta indicada por el ;acumulador y el dato
IN	reg, (C)	;Introduce un dato en reg desde la ;puerta indicada por el registro C
INC	operando	;Incrementa el operando
INC	regpar	;Incrementa un par de registros

INC	IX	;Incrementa IX
INC	IY	;Incrementa IY
IND		;Carga (HL) con el dato de la puerta ;indicada por el registro C y decremента ;HL y B
INDR		;Carga (HL) con el dato de la puerta ;indicada por el registro C, decremента ;HL y B, y repite hasta que B = 0
INI		;Carga (HL) con el dato de la puerta ;indicada por el registro C, decremента ;B e incrementa HL
INIR		;Carga (HL) con el dato de la puerta ;indicada por el registro C, decremента ;B e incrementa HL, y repite hasta que ;B = 0
JP	(HL)	;Salta a la dirección de HL
JP	(IX)	;Salta a la dirección de IX
JP	(IY)	;Salta a la dirección de IY
JP	dir	;Salta a la dirección
JP	cc,dir	;Carga el contador de programa con la ;dirección si cc es válido
JR	des	;Salta el desplazamiento
JR	c,des	;Salta el desplazamiento si c es válido
LD	A,I	;Carga el vector de interrupción dentro ;del acumulador
LD	A,operand1	;Carga el operando dentro del acumulador
LD	A,R	;Carga el REFRESH dentro del acumulador
LD	(BC),A	;Carga el acumulador en (BC)
LD	(DE),A	;Carga el acumulador en (DE)
LD	(HL),dato	;Carga el dato en (HL)
LD	HL,(dir)	;Carga (dir) dentro de HL

LD (HL),reg ;Carga el registro en (HL)

LD I,A ;Carga el acumulador en el vector de  
;interrupción

LD IX,dir ;Carga la dirección en IX

LD IX,(dir) ;Carga (dir) en IX

LD (IX+des),dato ;Carga el dato dentro de  
;IX + desplazamiento

LD IY,dir ;Carga la dirección en IX

LD IY,(dir) ;Carga (dir) en IY

LD (IY+des),dato ;Carga el dato dentro de  
;IY + desplazamiento

LD (dir),A ;Carga el acumulador dentro de (dir)

LD (dir),regpar ;Carga el par de registros dentro  
;de (dir)

LD (dir),IX ;Carga IX dentro de (dir)

LD (dir),IY ;Carga IY dentro de (dir)

LD regpar,dir ;Carga la dirección dentro del par  
;de registros

LD R,A ;Carga el acumulador dentro del REFRESH

LD reg,operand1 ;Carga el operando dentro del registro

LD SP,HL ;Carga HL dentro del apuntador de pila

LD SP,IX ;Carga IX dentro del apuntador de pila

LD SP,IY ;Carga IY dentro del apuntador de pila

LDD ;Carga (HL) dentro de (DE), decrementa  
;DE, BC y HL

LDDR ;Carga (HL) dentro de (DE), decrementa  
;DE, BC y HL, y repite hasta que BC = 0

LDI ;Carga (HL) dentro de (DE), incrementa  
;DE y HL, y decrementa BC

LDIR		;Carga (HL) dentro de (DE), incrementa ;DE y HL, decrementa BC, y repite hasta ;BC = 0
NEG		;Niega el acumulador
NOP		;No operación
OR	operand1	;OR lógico entre el operando y el ;acumulador
OTDR		;Carga (HL) dentro de la puerta C, ;decrementa B y HL, y repite hasta ;que B = 0
OTIR		;Carga (HL) dentro de la puerta C, ;incrementa HL, decrementa B y repite ;hasta que B = 0
OUT	(C),reg	;Carga el registro en la puerta C
OUT	(dato),A	;Carga el acumulador en la puerta dato
OUTD		;Carga (HL) en la puerta C, decrementa ;HL y B
OUTI		;Carga (HL) en la puerta C, incrementa ;HL y decrementa B
POP	regpar	;Recupera un par de registros desde ;la pila
POP	IX	;Recupera IX desde la pila
POP	IY	;Recupera IY desde la pila
PUSH	regpar	;Introduce un par de registros dentro ;de la pila
PUSH	IX	;Introduce IX dentro de la pila
PUSH	IY	;Introduce IY dentro de la pila
RES	b,operando	;Restaura el bit b del operando
RET		;Retorno
RET	cc	;Retorna si la condición cc es válida
RETI		;Retorno desde una interrupción

RETN		;Retorno desde una interrupción ;no enmascarable
RL	operando	;Gira el operando hacia la izquierda ;a través del bit de acarreo
RLA		;Gira el acumulador hacia la izquierda ;a través del bit de acarreo
RLC	operando	;Giro circular del operando hacia la ;izquierda
RLCA		;Giro circular del acumulador hacia la ;izquierda
RLD		;Giro circular hacia la izquierda entre ;los dos dígitos del octeto apuntado por ;HL y el segundo dígito del acumulador
RR	operando	;Gira el operando hacia la derecha a ;través del bit de acarreo
RRA		;Gira el acumulador hacia la derecha a ;través del bit de acarreo
RRC	operando	;Giro circular del operando hacia ;la derecha
RRCA		;Giro circular del acumulador hacia ;la derecha
RRD		;Giro circular hacia la derecha entre ;los dos dígitos del octeto apuntado por ;HL y el segundo dígito del acumulador
RST	vector	;Rearrancar en la dirección del vector
SBC	A,operando	;Restar el operando del acumulador junto ;con el bit de acarreo
SBC	HL,regpar	;Restar el par de registros de HL junto ;con el bit de acarreo
SCF		;Pone el bit de acarreo a 1
SET	b,operando	;Pone el bit b del operando a 1
SLA	operando	;Desplazamiento aritmético del operando ;hacia la izquierda

SRA	operando	;Desplazamiento aritmético del operando ;hacia la derecha
SRL	operando	;Desplazamiento lógico del operando ;hacia la derecha
SUB	operando	;Resta el operando del acumulador
XOR	operando	;OR exclusivo del operando dentro del ;acumulador

# Listado Alfabético de las Instrucciones del Z80

NEMOTECNICO	CODIGO OBJETO	SEÑALIZADORES
ADC A, (HL)	8E	C Z P/V S
ADC A, (IX+0)	DD8E00	C Z P/V S
ADC A, (IY+0)	FD8E00	C Z P/V S
ADC A, 0	CE00	C Z P/V S
ADC A, A	8F	C Z P/V S
ADC A, B	88	C Z P/V S
ADC A, C	89	C Z P/V S
ADC A, D	8A	C Z P/V S
ADC A, E	8B	C Z P/V S
ADC A, H	8C	C Z P/V S
ADC A, L	8D	C Z P/V S
ADC HL, BC	ED4A	C Z P/V S
ADC HL, DE	ED5A	C Z P/V S
ADC HL, HL	ED6A	C Z P/V S
ADC HL, SP	ED7A	C Z P/V S
ADD A, (HL)	86	C Z P/V S
ADD A, (IX+0)	DD8600	C Z P/V S
ADD A, (IY+0)	FD8600	C Z P/V S
ADD A, 0	C600	C Z P/V S
ADD A, A	87	C Z P/V S

ADD A, B	80	C Z P/V S
ADD A, C	81	C Z P/V S
ADD A, D	82	C Z P/V S
ADD A, E	83	C Z P/V S
ADD A, H	84	C Z P/V S
ADD A, L	85	C Z P/V S
ADD HL, BC	09	C
ADD HL, DE	19	C
ADD HL, HL	29	C
ADD HL, SP	39	C
ADD IX, BC	DD09	C
ADD IX, DE	DD19	C
ADD IX, IX	DD29	C
ADD IX, SP	DD39	C
ADD IY, BC	FD09	C
ADD IY, DE	FD19	C
ADD IY, IY	FD29	C
ADD IY, SP	FD39	C
AND (HL)	A6	Z P/V S
AND (IX+0)	DDA600	Z P/V S
AND (IY+0)	FDA600	Z P/V S
AND 0	E600	Z P/V S
AND A	A7	Z P/V S
AND B	A0	Z P/V S
AND C	A1	Z P/V S
AND D	A2	Z P/V S

AND E	A3	Z P/V S
AND H	A4	Z P/V S
AND L	A5	Z P/V S
BIT 0, (HL)	CB46	Z
BIT 0, (IX+0)	DDCB0046	Z
BIT 0, (IY+0)	FDCB0046	Z
BIT 0, A	CB47	Z
BIT 0, B	CB40	Z
BIT 0, C	CB41	Z
BIT 0, D	CB42	Z
BIT 0, E	CB43	Z
BIT 0, H	CB44	Z
BIT 0, L	CB45	Z
BIT 1, (HL)	CB4E	Z
BIT 1, (IX+0)	DDCB004E	Z
BIT 1, (IY+0)	FDCB004E	Z
BIT 1, A	CB4F	Z
BIT 1, B	CB48	Z
BIT 1, C	CB49	Z
BIT 1, D	CB4A	Z
BIT 1, E	CB4B	Z
BIT 1, H	CB4C	Z
BIT 1, L	CB4D	Z
BIT 2, (HL)	CB56	Z
BIT 2, (IX+0)	DDCB0056	Z
BIT 2, (IY+0)	FDCB0056	Z

BIT 2, A	CB57	Z
BIT 2, B	CB50	Z
BIT 2, C	CB51	Z
BIT 2, D	CB52	Z
BIT 2, E	CB53	Z
BIT 2, H	CB54	Z
BIT 2, L	CB55	Z
BIT 3, (HL)	CB5E	Z
BIT 3, (IX+0)	DDCB005E	Z
BIT 3, (IY+0)	FDCB005E	Z
BIT 3, A	CB5F	Z
BIT 3, B	CB58	Z
BIT 3, C	CB59	Z
BIT 3, D	CB5A	Z
BIT 3, E	CB5B	Z
BIT 3, H	CB5C	Z
BIT 3, L	CB5D	Z
BIT 4, (HL)	CB66	Z
BIT 4, (IX+0)	DDCB0066	Z
BIT 4, (IY+0)	FDCB0066	Z
BIT 4, A	CB67	Z
BIT 4, B	CB60	Z
BIT 4, C	CB61	Z
BIT 4, D	CB62	Z
BIT 4, E	CB63	Z

BIT 4, H	CB64	Z
BIT 4, L	CB65	Z
BIT 5, (HL)	CB6E	Z
BIT 5, (IX+0)	DDCB006E	Z
BIT 5, (IY+0)	FDCB006E	Z
BIT 5, A	CB6F	Z
BIT 5, B	CB68	Z
BIT 5, C	CB69	Z
BIT 5, D	CB6A	Z
BIT 5, E	CB6B	Z
BIT 5, H	CB6C	Z
BIT 5, L	CB6D	Z
BIT 6, (HL)	CB76	Z
BIT 6, (IX+0)	DDCB0076	Z
BIT 6, (IY+0)	FDCB0076	Z
BIT 6, A	CB77	Z
BIT 6, B	CB70	Z
BIT 6, C	CB71	Z
BIT 6, D	CB72	Z
BIT 6, E	CB73	Z
BIT 6, H	CB74	Z
BIT 6, L	CB75	Z
BIT 7, (HL)	CB7E	Z
BIT 7, (IX+0)	DDCB007E	Z
BIT 7, (IY+0)	FDCB007E	Z
BIT 7, A	CB7F	Z

BIT 7, B	CB78	Z
BIT 7, C	CB79	Z
BIT 7, D	CB7A	Z
BIT 7, E	CB7B	Z
BIT 7, H	CB7C	Z
BIT 7, L	CB7D	Z
CALL 0	CD0000	
CALL C, 0	DC0000	
CALL M, 0	FC0000	
CALL NC, 0	D40000	
CALL NZ, 0	C40000	
CALL P, 0	F40000	
CALL PE, 0	EC0000	
CALL PD, 0	E40000	
CALL Z, 0	CC0000	
CCF	3F	C
CP 0	FE00	C Z P/V S
CP (HL)	BE	C Z P/V S
CP (IX+0)	DDBE00	C Z P/V S
CP (IY+0)	FDBE00	C Z P/V S
CP A	BF	C Z P/V S
CP B	B8	C Z P/V S
CP C	B9	C Z P/V S
CP D	BA	C Z P/V S
CP E	BB	C Z P/V S
CP H	BC	C Z P/V S

CP	L	BD	C Z P/V S
CPD		EDA9	Z P/V S
CPDR		EDB9	Z P/V S
CPI		EDA1	Z P/V S
CPIR		EDB1	Z P/V S
CPL		2F	
DAA	(HL)	27	C Z P/V S
DEC	(HL)	35	Z P/V S
DEC	(IX+0)	DD3500	Z P/V S
DEC	(IY+0)	FD3500	Z P/V S
DEC	A	3D	Z P/V S
DEC	B	05	Z P/V S
DEC	BC	0B	
DEC	C	0D	Z P/V S
DEC	D	15	Z P/V S
DEC	DE	1B	
DEC	E	1D	Z P/V S
DEC	H	25	Z P/V S
DEC	HL	2B	
DEC	IX	DD2B	
DEC	IY	FD2B	
DEC	L	2D	Z P/V S
DEC	SP	3B	
DI		F3	
DJNZ	0	1000	
EI		FB	

EX	(SP), HL	E3	
EX	(SP), IX	DDE3	
EX	(SP), IY	FDE3	
EX	AF, AF'	08	C Z P/V S
EX	DE, HL	EB	
EXX		D9	
HALT		76	
IM	0	ED46	
IM	1	ED56	
IM	2	ED5E	
IN	A, (0)	DB00	
IN	A, (C)	ED78	Z P/V S
IN	B, (C)	ED40	Z P/V S
IN	C, (C)	ED48	Z P/V S
IN	D, (C)	ED50	Z P/V S
IN	E, (C)	ED58	Z P/V S
IN	H, (C)	ED60	Z P/V S
IN	L, (C)	ED68	Z P/V S
INC	(HL)	34	Z P/V S
INC	(IX+0)	DD3400	Z P/V S
INC	(IY+0)	FD3400	Z P/V S
INC	A	3C	Z P/V S
INC	B	04	Z P/V S
INC	BC	03	
INC	C	0C	Z P/V S
INC	D	14	Z P/V S

INC	DE	13	
INC	E	1C	Z P/V S
INC	H	24	Z P/V S
INC	HL	23	
INC	IX	DD23	
INC	IY	FD23	
INC	L	2C	Z P/V S
INC	SP	33	
IND		EDAA	Z
INDR		EDBA	
INI		EDA2	Z
INIR		EDB2	
JP	0	C30000	
JP	(HL)	E9	
JP	(IX)	DDE9	
JP	(IY)	FDE9	
JP	C, 0	DA0000	
JP	M, 0	FA0000	
JP	NC, 0	D20000	
JP	NZ, 0	C20000	
JP	P, 0	F20000	
JP	PE, 0	EA0000	
JP	PO, 0	E20000	
JP	Z, 0	CA0000	
JR	0	1800	
JR	C, 0	3800	

JR	NC, 0	3000
JR	NZ, 0	2000
JR	Z, 0	2800
LD	(BC), A	02
LD	(DE), A	12
LD	(HL), 0	3600
LD	(HL), A	77
LD	(HL), B	70
LD	(HL), C	71
LD	(HL), D	72
LD	(HL), E	73
LD	(HL), H	74
LD	(HL), L	75
LD	(IX+0), 0	DD360000
LD	(IX+0), A	DD7700
LD	(IX+0), B	DD7000
LD	(IX+0), C	DD7100
LD	(IX+0), D	DD7200
LD	(IX+0), E	DD7300
LD	(IX+0), H	DD7400
LD	(IX+0), L	DD7500
LD	(IY+0), 0	FD360000
LD	(IY+0), A	FD7700
LD	(IY+0), B	FD7000
LD	(IY+0), C	FD7100
LD	(IY+0), D	FD7200

LD	(IY+0), E	FD7300	
LD	(IY+0), H	FD7400	
LD	(IY+0), L	FD7500	
LD	(0000), A	320000	
LD	(0000), BC	ED430000	
LD	(0000), DE	ED530000	
LD	(0000), HL	220000	
LD	(0000), IX	DD220000	
LD	(0000), IY	FD220000	
LD	(0000), IX	ED730000	
LD	A, (BC)	0A	
LD	A, (DE)	1A	
LD	A, (HL)	7E	
LD	A, (IX+0)	DD7E00	
LD	A, (IY+0)	FD7E00	
LD	A, (0000)	3A0000	
LD	A, 0	3E00	
LD	A, A	7F	
LD	A, B	78	
LD	A, C	79	
LD	A, D	7A	
LD	A, E	7B	
LD	A, H	7C	
LD	A, I	ED57	Z P/V S
LD	A, L	7D	
LD	A, R	ED5F	Z P/V S

LD	B, (HL)	46
LD	B, (IX+0)	DD4600
LD	B, (IY+0)	FD4600
LD	B, 0	0600
LD	B, A	47
LD	B, B	40
LD	B, C	41
LD	B, D	42
LD	B, E	43
LD	B, H	44
LD	B, L	45
LD	BC, (0000)	ED4B0000
LD	BC, 0000	010000
LD	C, (HL)	4E
LD	C, (IX+0)	DD4E00
LD	C, (IY+0)	FD4E00
LD	C, 0	0E00
LD	C, A	4F
LD	C, B	48
LD	C, C	49
LD	C, D	4A
LD	C, E	4B
LD	C, H	4C
LD	C, L	4D
LD	D, (HL)	56
LD	D, (IX+0)	DD5600

LD	D, (IY+0)	FD5600
LD	D, 0	1600
LD	D, A	57
LD	D, B	50
LD	D, C	51
LD	D, D	52
LD	D, E	53
LD	D, H	54
LD	D, L	55
LD	DE, (0000)	ED5B0000
LD	DE, 0000	110000
LD	E, (HL)	5E
LD	E, (IX+0)	DD5E00
LD	E, (IY+0)	FD5E00
LD	E, 0	1E00
LD	E, A	5F
LD	E, B	58
LD	E, C	59
LD	E, D	5A
LD	E, E	5B
LD	E, H	5C
LD	E, L	5D
LD	H, (HL)	66
LD	H, (IX+0)	DD6600
LD	H, (IY+0)	FD6600
LD	H, 0	2600

LD	H, A	67
LD	H, B	60
LD	H, C	61
LD	H, D	62
LD	H, E	63
LD	H, H	64
LD	H, L	65
LD	HL, (0000)	2A0000
LD	HL, (0000)	ED6B0000
LD	HL, 0000	210000
LD	I, A	ED47
LD	IX, (0000)	DD2A0000
LD	IX, 0000	DD210000
LD	IY, (0000)	FD2A0000
LD	IY, 0000	FD210000
LD	L, (HL)	6E
LD	L, (IX+0)	DD6E00
LD	L, (IY+0)	FD6E00
LD	L, 0	2E00
LD	L, A	6F
LD	L, B	68
LD	L, C	69
LD	L, D	6A
LD	L, E	6B
LD	L, H	6C
LD	L, L	6D

LD	R, A	ED4F	
LD	SP, (0000)	ED7B0000	
LD	SP, 0000	310000	
LD	SP, HL	F9	
LD	SP, IX	DDF9	
LD	SP, IY	FDF9	
LDD		EDA8	P/V
LDDR		EDB8	
LDI		EDA0	P/V
LDIR		EDB0	
NEG		ED44	C Z P/V S
NOP		00	
OR	(HL)	B6	Z P/V S
OR	(IX+0)	DDB600	Z P/V S
OR	(IY+0)	FDB600	Z P/V S
OR	0	F600	Z P/V S
OR	A	B7	Z P/V S
OR	B	B0	Z P/V S
OR	C	B1	Z P/V S
OR	D	B2	Z P/V S
OR	E	B3	Z P/V S
OR	H	B4	Z P/V S
OR	L	B5	Z P/V S
OTDR		ED8B	
OTIR		EDB3	
OUT	(C), A	ED79	

OUT (C), B	ED41	
OUT (C), C	ED49	
OUT (C), D	ED51	
OUT (C), E	ED59	
OUT (C), H	ED61	
OUT (C), L	ED69	
OUT (0), A	D300	
OUTD	EDAB	Z
OUTI	EDA3	Z
POP AF	F1	C Z P/V S
POP BC	C1	
POP DE	D1	
POP HL	E1	
POP IX	DDE1	
POP IY	FDE1	
PUSH AF	F5	
PUSH BC	C5	
PUSH DE	D5	
PUSH HL	E5	
PUSH IX	DDE5	
PUSH IY	FDE5	
RES 0, (HL)	CB86	
RES 0, (IX+0)	DDCB0086	
RES 0, (IY+0)	FDCB0066	
RES 0, A	CB87	
RES 0, B	CB80	

RES 0, C	CB81
RES 0, D	CB82
RES 0, E	CB83
RES 0, H	CB84
RES 0, L	CB85
RES 1, (HL)	CB8E
RES 1, (IX+0)	DDCB008E
RES 1, (IY+0)	FDCB008E
RES 1, A	CB8F
RES 1, B	CB88
RES 1, C	CB89
RES 1, D	CB8A
RES 1, E	CB8B
RES 1, H	CB8C
RES 1, L	CB8D
RES 2, (HL)	CB96
RES 2, (IX+0)	DDCB0096
RES 2, (IY+0)	FDCB0096
RES 2, A	CB97
RES 2, B	CB90
RES 2, C	CB91
RES 2, D	CB92
RES 2, E	CB93
RES 2, H	CB94
RES 2, L	CB95
RES 3, (HL)	CB9E

RES 3, (IX+0)	DDCB009E
RES 3, (IY+0)	FDCB009E
RES 3, A	CB9F
RES 3, B	CB98
RES 3, C	CB99
RES 3, D	CB9A
RES 3, E	CB9B
RES 3, H	CB9C
RES 3, L	CB9D
RES 4, (HL)	CBA6
RES 4, (IX+0)	DDCB00A6
RES 4, (IY+0)	FDCB00A6
RES 4, A	CBA7
RES 4, B	CBA0
RES 4, C	CBA1
RES 4, D	CBA2
RES 4, E	CBA3
RES 4, H	CBA4
RES 4, L	CBA5
RES 5, (HL)	CBAE
RES 5, (IX+0)	DDCB00AE
RES 5, (IY+0)	FDCB00AE
RES 5, A	CBAF
RES 5, B	CBA8
RES 5, C	CBA9
RES 5, D	CBAA

RES 5, E	CBAB
RES 5, H	CBAC
RES 5, L	CBAD
RES 6, (HL)	CBB6
RES 6, (IX+0)	DDCB00B6
RES 6, (IY+0)	FDCB00B6
RES 6, A	CBB7
RES 6, B	CBB0
RES 6, C	CBB1
RES 6, D	CBB2
RES 6, E	CBB3
RES 6, H	CBB4
RES 6, L	CBB5
RES 7, (HL)	CBBE
RES 7, (IX+0)	DDCB00BE
RES 7, (IY+0)	FDCB00BE
RES 7, A	CBBF
RES 7, B	CBB8
RES 7, C	CBB9
RES 7, D	CBBA
RES 7, E	CBBB
RES 7, H	CBBC
RES 7, L	CBBD
RET	C9
RET C	D8
RET M	F8

RET	NC	D0	
RET	NZ	C0	
RET	P	F0	
RET	PE	E8	
RET	PO	E0	
RET	Z	C8	
RETI		ED4D	
RETN		ED45	
RL	(HL)	CB16	C Z P/V S
RL	(IX+0)	DDCB0016	C Z P/V S
RL	(IX+0)	FDCB0016	C Z P/V S
RL	A	CB17	C Z P/V S
RL	B	CB10	C Z P/V S
RL	C	CB11	C Z P/V S
RL	D	CB12	C Z P/V S
RL	E	CB13	C Z P/V S
RL	H	CB14	C Z P/V S
RL	L	CB15	C Z P/V S
RLA		17	C
RLC	(HL)	CB06	C Z P/V S
RLC	(IX+0)	DDCB0006	C Z P/V S
RLC	(IX+0)	FDCB0006	C Z P/V S
RLC	A	CB07	C Z P/V S
RLC	B	CB00	C Z P/V S
RLC	C	CB01	C Z P/V S
RLC	D	CB02	C Z P/V S

RLC	E	CB03	C Z P/V S
RLC	H	CB04	C Z P/V S
RLC	L	CB05	C Z P/V S
RLCA		07	C
RLD		ED6F	Z P/V S
RR	(HL)	CB1E	C Z P/V S
RR	(IX+0)	DDCB001E	C Z P/V S
RR	(IX+0)	FDCB001E	C Z P/V S
RR	A	CB1F	C Z P/V S
RR	B	CB18	C Z P/V S
RR	C	CB19	C Z P/V S
RR	D	CB1A	C Z P/V S
RR	E	CB1B	C Z P/V S
RR	H	CB1C	C Z P/V S
RR	L	CB1D	C Z P/V S
RRA		1F	C
RRC	(HL)	CB0E	C Z P/V S
RRC	(IX+0)	DDCB000E	C Z P/V S
RRC	(IX+0)	FDCB000E	C Z P/V S
RRC	A	CB0F	C Z P/V S
RRC	B	CB08	C Z P/V S
RRC	C	CB09	C Z P/V S
RRC	D	CB0A	C Z P/V S
RRC	E	CB0B	C Z P/V S
RRC	H	CB0C	C Z P/V S
RRC	L	CB0D	C Z P/V S

RRCA	0F	C
RPD	ED67	Z P/V S
RST 00H	C7	
RST 08H	CF	
RST 10H	D7	
RST 18H	DF	
RST 20H	E7	
RST 28H	EF	
RST 30H	F7	
RST 38H	FF	
SBC A, (HL)	9E	C Z P/V S
SBC A, (IX+0)	DD9E00	C Z P/V S
SBC A, (IY+0)	FD9E00	C Z P/V S
SBC A, 0	DE00	C Z P/V S
SBC A, A	9F	C Z P/V S
SBC A, B	98	C Z P/V S
SBC A, C	99	C Z P/V S
SBC A, D	9A	C Z P/V S
SBC A, E	9B	C Z P/V S
SBC A, H	9C	C Z P/V S
SBC A, L	9D	C Z P/V S
SBC HL, BC	ED42	C Z P/V S
SBC HL, DE	ED52	C Z P/V S
SBC HL, HL	ED62	C Z P/V S
SBC HL, SP	ED72	C Z P/V S
SCF	37	C

SET 0, (HL)	CBC6
SET 0, (IX+0)	DDCB00C6
SET 0, (IY+0)	FDCB00C6
SET 0, A	CB87
SET 0, B	CBC0
SET 0, C	CBC1
SET 0, D	CBC2
SET 0, E	CBC3
SET 0, H	CBC4
SET 0, L	CBC5
SET 1, (HL)	CBCE
SET 1, (IX+0)	DDCB00CE
SET 1, (IY+0)	FDCB00CE
SET 1, A	CBCF
SET 1, B	CBC8
SET 1, C	CBC9
SET 1, D	CBCA
SET 1, E	CBCB
SET 1, H	CBCC
SET 1, L	CBCD
SET 2, (HL)	CBD6
SET 2, (IX+0)	DDCB00D6
SET 2, (IY+0)	FDCB00D6
SET 2, A	CBD7
SET 2, B	CBD0
SET 2, C	CBD1

SET 2, D	CBD2
SET 2, E	CBD3
SET 2, H	CBD4
SET 2, L	CBD5
SET 3, (HL)	CBDE
SET 3, (IX+0)	DDCB00DE
SET 3, (IY+0)	FDCB00DE
SET 3, A	CBDF
SET 3, B	CBD8
SET 3, C	CBD9
SET 3, D	CBDA
SET 3, E	CBDB
SET 3, H	CBDC
SET 3, L	CBDD
SET 4, (HL)	CBE6
SET 4, (IX+0)	DDCB00E6
SET 4, (IY+0)	FDCB00E6
SET 4, A	CBE7
SET 4, B	CBE0
SET 4, C	CBE1
SET 4, D	CBE2
SET 4, E	CBE3
SET 4, H	CBE4
SET 4, L	CBE5
SET 5, (HL)	CBEE
SET 5, (IX+0)	DDCB00EE

SET 5, (IY+0)	FDCB00EE
SET 5, A	CBEF
SET 5, B	CBEB
SET 5, C	CBE9
SET 5, D	CBEA
SET 5, E	CBEB
SET 5, H	CBEC
SET 5, L	CBED
SET 6, (HL)	CBF6
SET 6, (IX+0)	DDCB00F6
SET 6, (IY+0)	FDCB00F6
SET 6, A	CBF7
SET 6, B	CBF0
SET 6, C	CBF1
SET 6, D	CBF2
SET 6, E	CBF3
SET 6, H	CBF4
SET 6, L	CBF5
SET 7, (HL)	CBFE
SET 7, (IX+0)	DDCB00FE
SET 7, (IY+0)	FDCB00FE
SET 7, A	CBFF
SET 7, B	CBF8
SET 7, C	CBF9
SET 7, D	CBFA
SET 7, E	CBFB

SET	7, H	CBFC	
SET	7, L	CBFD	
SLA	(HL)	CB26	C Z P/V S
SLA	(IX+0)	DDCB0026	C Z P/V S
SLA	(IX+0)	FDCB0026	C Z P/V S
SLA	A	CB27	C Z P/V S
SLA	B	CB20	C Z P/V S
SLA	C	CB21	C Z P/V S
SLA	D	CB22	C Z P/V S
SLA	E	CB23	C Z P/V S
SLA	H	CB24	C Z P/V S
SLA	L	CB25	C Z P/V S
SRA	(HL)	CB2E	C Z P/V S
SRA	(IX+0)	DDCB002E	C Z P/V S
SRA	(IX+0)	FDCB002E	C Z P/V S
SRA	A	CB2F	C Z P/V S
SRA	B	CB28	C Z P/V S
SRA	C	CB29	C Z P/V S
SRA	D	CB2A	C Z P/V S
SRA	E	CB2B	C Z P/V S
SRA	H	CB2C	C Z P/V S
SRA	L	CB2D	C Z P/V S
SRL	(HL)	CB3E	C Z P/V S
SRL	(IX+0)	DDCB003E	C Z P/V S
SRL	(IX+0)	FDCB003E	C Z P/V S
SRL	A	CB3F	C Z P/V S

SRL	B	CB38	C Z P/V S
SRL	C	CB39	C Z P/V S
SRL	D	CB3A	C Z P/V S
SRL	E	CB3B	C Z P/V S
SRL	H	CB3C	C Z P/V S
SRL	L	CB3D	C Z P/V S
SUB	0	D600	C Z P/V S
SUB	(HL)	96	C Z P/V S
SUB	(IX+0)	DD9600	C Z P/V S
SUB	(IX+0)	FD9600	C Z P/V S
SUB	A	97	C Z P/V S
SUB	B	90	C Z P/V S
SUB	C	91	C Z P/V S
SUB	D	92	C Z P/V S
SUB	E	93	C Z P/V S
SUB	H	94	C Z P/V S
SUB	L	95	C Z P/V S
XOR	(HL)	AE	Z P/V S
XOR	(IX+0)	DDAE00	Z P/V S
XOR	(IX+0)	FDAE00	Z P/V S
XOR	0	EE00	Z P/V S
XOR	A	AF	Z P/V S
XOR	B	A8	Z P/V S
XOR	C	A9	Z P/V S
XOR	D	AA	Z P/V S
XOR	E	AB	Z P/V S

XOR H

AC

Z P/V S

XOR L

AD

Z P/V S

# APENDICE C

## Conversión Hexadecimal a Decimal

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

# AMSTRAD

## CPC 6128-664-464

### **Sobre nosotros**

Son bienvenidas las solicitudes de catálogos, así como otro tipo de propuestas.

### **Escriban a:**

RA-MA  
Editorial  
Crta. de Canillas, 144  
28043 MADRID

**ISBN 84-86381-15-0**