

Domine
el Código Máquina
en su

AMSTRAD

CPC 6128-664-464

CLIVE GIFFORD-SCOTT VINCENT



Domine
el Código Máquina
en su

AMSTRAD

CPC 6128-664-464

EDICIÓN 2007

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 © 1986, Gifford/Vincent

Traducido del inglés por
M.C. Dopazo

Primera edición en español
Copyright © 1986

RA-MA

Carretera de Canillas, 144
28043 MADRID

Última edición en español por
DaDMaN / DaDoS
Diciembre - 2007

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 método, 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 probadas.

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

Capítulo 1	1
¿Qué es el código máquina?	1
Juego de instrucciones	2
A favor	2
Argumentos en contra	3
Capítulo 2	5
Sistemas numéricos y lenguaje ensamblador	5
Números de 8 y 16 bits	6
Lenguaje ensamblador	7
Comprando un ensamblador	8
Directrices	9
Capítulo 3	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
Capítulo 4	15
Su primer programa en código máquina	15
Un cargador hex	15
El programa	16
Variables	17
Acumulador	20

Capítulo 5	21
Pasando parámetros	21
Cambio del octeto inferior	22
Manejando cadenas	23
Una rutina de tratamiento de cadenas	23
Capítulo 6	25
Aritmética simple	25
Suma y resta	26
Punto y acarreo	28
No hay división	28
Capítulo 7	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
Capítulo 8	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, ADD, AND, BIT	47
CALL, CCF, CP, CPD, CPDR, CPI	48
CPIR, CPL, DAA, DEC, DEFB	49
DEFM, DEFS, DEFW, DI, DJNZ	50
EI, EQU, EX, EXX, HALT	51
IM, IN, INC, IND, INDR, INI, INIR, JP	52

JR, LD, LDD	53
LDDR, LDI, LDIR, NEG, NOP, OR, ORG	54
OUT, OUTD, OUTDR, - OUTI, OUTIR, POP, PUSH, RES	55
RET, RL, RLA, RLC, RLCA, RLD	56
RR, RRA, RRC, RRCA, RRD, RST, SBC, SCF	57
SET, SLA, SRA, SRL, SUB, XOR	58
Capítulo 9	59
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
Capítulo 10	67
Pantalla y rutinas de la ROM	67
Doscientos de alto	67
Paquete de rutinas en código máquina	71
Uno - Leer un carácter	73
BASIC	73
Ensamblador	74
Dos - Girar hacia la izquierda	75
BASIC	75
Ensamblador	76
Tres - Girar hacia la derecha	79
BASIC	79
Ensamblador	80
Cuatro - Letras gigantes	83
BASIC	83
Ensamblador	85

Cinco – Impresión masiva	89
BASIC	89
Ensamblador	90
Seis – Relleno de la pantalla	93
BASIC	93
Ensamblador	94
Siete – LOAD/SAVE sin cabecera	95
BASIC	95
Ensamblador	96
Ocho – Música por interrupciones	99
BASIC	100
Ensamblador	103
Nueve – Monitor de código máquina	115
BASIC	115
Ensamblador	119
Diez – Movimiento de bloques	125
Bloques hacia arriba	125
BASIC	126
Ensamblador	127
Bloques hacia abajo	130
BASIC	130
Ensamblador	131
Bloques hacia la izquierda	134
BASIC	134
Ensamblador	136
Bloques hacia la derecha	141
BASIC	142
Ensamblador	143
Once – Acordes RSX	151
BASIC	152
Envolventes	154
Ensamblador	156
Órgano de acordes	163
BASIC	164

Doce – Compresores de pantalla	165
Compresor 1	165
BASIC	165
Ensamblador	166
Compresor 2	168
BASIC	169
Ensamblador	170
Trece – DOKE y DEEK	173
BASIC	173
Ensamblador	174
Catorce – El paquete de escritura de juegos	177
BASIC	178
Bombardero	182

Apéndices

Apéndice A	189
Mapa de memoria	189
Apéndice B	191
Códigos de operación Z80	191
Sumario de instrucciones	
- y funciones del Z80	192
Listado por grupos	
- de las instrucciones del Z80	199
Instrucciones ordenadas	
- por código de operación	211
Apéndice C	225

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 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

CAPÍTULO 1

¿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 la 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 consisten 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 programar 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.

CAPÍTULO 2

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 10, los binarios en base 2, y los hexadecimales en base 16.

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 16. 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 8 y 16 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 2 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:

$$\begin{aligned} & \mathbf{256 \text{ veces el octeto alto} + \text{el octeto bajo}} \\ & \mathbf{256 * 03 + C9 = 256 * 3 + 201 = 969} \end{aligned}$$

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 = 1111000

Por lo tanto, 1111001 es el equivalente binario del número -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 2. 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 en 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 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 libras, 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 de código máquina, todo por 12.95 libras.

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 Freepost, Europa House, 68 Chester Road, Hazel Grove, Stockport SK7 5NY, Reino Unido. Cada número atrasado cuesta alrededor de 1.25 libras, y una cinta que contiene todos los programas de la revista cuesta unas 3.75 libras.

En nuestro país podemos obtener un buen ensamblador como es el **DEVPAC** de HISOFT, que distribuye Indescomp, por unas 6500Pt. 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. También lo puede encontrar en **MICROTOD0**, en la calle Orense 3 de Madrid. Este paquete incluye también un estupendo desensamblador que nos ayudará a analizar los programas de los que no dispongamos listado.

Directrices

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 o desactivar la impresión mientras se ejecuta el ensamblaje del código. La sintaxis exacta de este comando varía dependiendo del 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.

CAPÍTULO 3

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 una 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 las 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 Sólo 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 sólo 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 de Periféricos Programable y es la unión entre la CPU y el PSG, el puerto 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 tecllea PRINT FRE(0), verá que tiene a su disposición unos 42K de memoria. Sin embargo, sólo 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 ordenador. 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 4000, 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 "PROGRAMA", B, inicio, longitud, entrada

El parámetro inicio 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 "PROGRAMA" 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)).

CAPÍTULO 4

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 sávelo 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 ultima
    linea":GOTO 30
50 b$=LEFT$(a$,2)
60 POKE n,UAL("&" +b$)
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 de 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 Amstrad, 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 mas 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 sólo 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. Sólo pueden contener un número entre 0 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 0 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 registro contenido en el 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 6, cuando tratemos la aritmética simple.

Hay un número de LDs específicos al registro A. Es el único con 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 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 un CALL 40000 para que la rutina cargue en 41001 el calor 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: Señalizador de sustracción o de negación.

Bit 0: Señalizador de acarreo, o C. (No confundir con el reg. C).

El señalizador de **medio-acarreo** lo ponen las instrucciones aritméticas si hay un acarreo desde el bit 3 al bit 4 o, en el caso de pares de registros, desde el bit 11 al bit 12.

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 incomprendible de momento. Ya lo verá claro durante el curso (se lo prometemos), y entonces podrá volver a releer esta sección comprendiéndola completamente.

CAPÍTULO 5

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 o en complemento a 2 de 16 bits, de forma que 31092, 9, 220 y F% (donde F% es igual 1000) están permitidos.

Cuando 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 un CALL 40000,258, (IX+0) contendrá el octeto bajo 2 e (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 un 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 el 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 estará 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 el 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	

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, sólo 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 sólo 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 un 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\$.

CAPÍTULO 6

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 1 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 decreenta 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 cómo 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 les 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 registros no es modificado por esta instrucción. Las operaciones aritméticas con los registros índice funcionan así:

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

IX 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, sólo 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 (múltiplo de 2), 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 = x3
ADD A,B
```

```
LD B,A
ADD A,A
ADD A,A = x7
ADD A,A
SUB A,B
```

Observe las 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.

CAPÍTULO 7

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 en medio, 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 sólo 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, al segunda es POP. Si hace PUSH del número 5 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 fue el último del que hicimos PUSH); el segundo número que obtendríamos con POP sería 9ABC, y el tercero sería el 5.

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 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 0 ó 1, 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	

¿Por qué sólo 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 la 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 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 GOTO del BASIC. La única diferencia es que el código máquina no usa números de línea, de forma que ha de hacer GOTO 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 GOTO 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 GOTO, 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 y 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 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 sólo 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 delante un número de octetos dado. En muchas formas es mejor que JP, porque tiene sólo 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 -5, 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 podemos 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 + 0**. JR 1 significa que ejecute la **siguiente instrucción + 1**. 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 usar 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 GOTO del BASIC sería poco útil si no se usara en conjunto con la instrucción IF/THEN GOTO. Necesitamos cierto tipo de saltos **condicionales**, 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 fue cero THEN salto relativo de e octetos
28ee	JR	Z,e	IF el último resultado calculado fue 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:

C3qqpp	JP	pq	Salto a la dirección pq.
--------	----	----	--------------------------

C2qqpp	JP	NZ,pq	IF el último resultado calculado no fue cero THEN salto a la dirección pq.
CAqqpp	JP	Z,pq	IF el último resultado calculado fue cero THEN salto a la dirección pq.
D2qqpp	JP	NC,pq	IF ACARREO=0 THEN salto a la dirección pq.
DAqqpp	JP	C,pq	IF ACARREO?1 THEN salto a la dirección pq.
E2qqpp	JP	PO,pq	Ver más abajo.
EAqqpp	JP	PE,pq	Ver mas abajo.
F2qqpp	JP	P,pq	IF el último resultado calculado fue positivo (+) THEN salto a la dirección pq.
FAqqpp	JP	M,pq	IF el último resultado calculado fue 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 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?

Respuesta: 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) + 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 de convención positivo/negativo (también llamada convención de “complemento a 2”) 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, sólo hay

otro tipo de instrucción necesaria para hacer a JP y JR tan potentes como IF/THEN/GOTO. 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 sí 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 hincapié 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 GOTO...      CP B / JR C...
(Esta sólo funciona si asumimos que todos
 los números son positivos)
IF A<B THEN GOTO...      CP B / JP M...
```

CALL

También en código máquina podemos llamar a **subrutinas**. El equivalente en código máquina del GOSUB del BASIC es la instrucción CALL. CALL pq se utiliza para hacer un GOSUB 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:

CDqppp	CALL pq	C9	RET
C4qppp	CALL NZ, pq	C0	RET NZ
CCqppp	CALL Z, pq	C8	RET Z
D4qppp	CALL NC, pq	D0	RET NC
DCqppp	CALL C, pq	D8	RET C
E4qppp	CALL PO, pq	E0	RET PO
	"CALL NU, pq"		"RET NU"
ECqppp	CALL PE, pq	E8	RET PE
	"CALL U, pq"		"RET U"
F4qppp	CALL P, pq	F0	RET P
FCqppp	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 asegurarse 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.

CAPÍTULO 8

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 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 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) sólo 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 es 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 Alternativo de Registros, se compone de 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 sólo hemos visto unas cuantas instrucciones Z80, por lo que suponemos que estará interesado en 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 La instrucción 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 el 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 por qué (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 sólo 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 GOSUB. 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 sólo 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 JP 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 6 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 hez, pero no siempre es necesario, ej. DEFB 3A,45d,1101110b,"f" es válido. Los datos se insertan dentro del programa en código máquina, en el punto en que ocurren y en el orden en el 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. (Sólo 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 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 por 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 qué 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 un $LET\ r=IN(256*B+C)$. 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 $GOTO\ 10$, 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 $GOTO$ del BASIC. Tenemos también saltos condicionales, por ejemplo $JP\ NZ,7300$ significa $IF\ no\ cero\ THEN\ salta\ a\ la\ dirección\ 7300$ (es decir, 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 $GOTO\ N$ comprenderá $JP\ (HL)$. $JP\ (HL)$ significa $GOTO\ HL$. 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. Sólo 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 transfiere 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 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 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 2 octetos).

OUT La instrucción OUT tiene dos formas. La primera es OUT (n),A - equivalente a decir $OUT(256 \cdot A + n), A$. La segunda forma es OUT (C),r y equivale a $OUT(256 \cdot B + C), r$. 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 un DEC HL seguido por un 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 delecteo, 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 'pseudopar' 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 1. 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 la 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 1. 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 sólo 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 sólo 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 sólo 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) sólo 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 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).

CAPÍTULO 9

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      LD   A,102      (01100110 en BIN)
06 2B      LD   B,43       (00101011 en BIN)
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, sólo 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 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, la instrucción AND cancela el sexto bit haciéndolo igual a 0, 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 0. Poniendo a cero el sexto bit del registro A garantizamos que se resta 32 del valor del código, obteniendo así la letra correspondiente pero invertida 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 3) y el valor del registro B será ahora de 7 decimal, en binario 00000111.

El punto más importante a tener en cuenta, no sólo 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 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).

Girar 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.

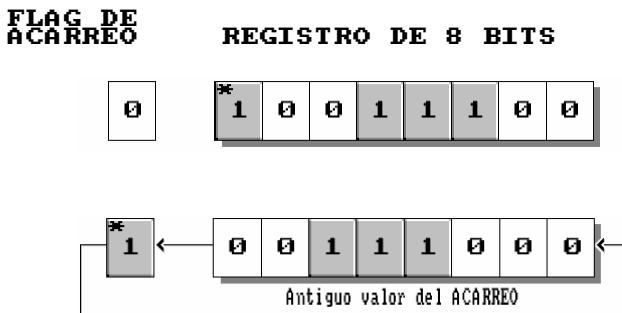
El giro hacia la derecha (RRC) es justo lo opuesto.

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 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/derecha 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)



*** Indica el bit 'a' que sale**

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

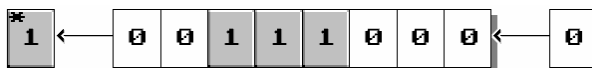
Los desplazamientos son similares a los giros, como se ve en este tercer diagrama.

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 128). 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 2 y SRL divide el número por 2. Por lo tanto, para multiplicar un número por 8, basta con hacer SLA tres veces.

CAPÍTULO 10

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 ó 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 0. 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 1 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 ($24 \cdot 80 + 1$) hasta el 2000 ($625 \cdot 80$). 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.

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=fuego 1 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 al puerto Centronics (normalmente para la impresora).
BB39	Pone la repetición de la tecla o la quita.
BB21	Obtiene los estados de Bloqueo Mayúsculas y Bloqueo 'Shift'.

Paquete de Rutinas en Código Máquina

- 1.** Leer un Carácter
- 2.** Girar hacia la Izquierda
- 3.** Girar hacia la Derecha
- 4.** Letras Gigantes
- 5.** Impresión Masiva
- 6.** Relleno de la Pantalla
- 7.** LOAD/SAVE Sin Cabecera
- 8.** Música por Interrupciones
- 9.** Monitor de Código Máquina
- 10.** Movimiento de Bloques de Pantalla
- 11.** Acordes RSX
- 12.** Compresores de Pantalla
- 13.** DEEK y DOKE
- 14.** Paquete de Escritura de Juegos

UNO – Leer un Carácter

Si se pregunta 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 ejecútelolo:

```
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 el 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 debe 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 #BB75
BB60 0030  RDCHAR  DEFL #BB60
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 D06601 0090  LD H,(IX+1)
9C52 77 0100  LD (HL),A
9C53 C9 0110  RET
                                0120  END
```

DOS – Girar hacia la Izquierda

Esta rutina 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,00,1A,13,D5,E5,16
110 DATA 80,1E,80,06,00,82,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.

A partir de ahora mostraremos los listados únicamente con la nomenclatura en ensamblador, sin referencias a las direcciones de memoria usadas, para facilitar la legibilidad del listado.

```
0001      ;      GIRAR A LA IZQUIERDA
0002      ;
0010      ORG      40000
0020      CURSOR DEFL #BB75
0030      MATRIX DEFL #BBA5
0040      TABLE DEFL #BBAE
0050      ROMENA DEFL #B906
0060      ROMDIS DEFL #B909
0070      TXTOUT DEFL #BB5A
0080      LD      L, (IX+0)
0090      LD      H, (IX+1)
0100      LD      A, (HL)
0110      OR      A
0120      RET     Z
0130      INC     HL
```

0140		LD	E, (HL)
0150		INC	HL
0160		LD	D, (HL)
0170		LD	L, (IX+2)
0180		LD	H, (IX+4)
0190		LD	B, A
0200	NXTCHR	PUSH	BC
0210		PUSH	HL
0220		CALL	CURSOR
0230		LD	A, (DE)
0240		CALL	ROTCHR
0250		INC	DE
0260		POP	HL
0270		DEC	L
0280		POP	BC
0290		DJNZ	NXTCHR
0300		RET	
0310	ROTCHR	PUSH	DE
0320		CALL	MATRIX
0330		EX	DE, HL
0340		CALL	TABLE
0350		LD	B, 7
0360	ADD7	INC	HL
0370		DJNZ	ADD7
0380		PUSH	AF
0390		CALL	ROMENA
0400		LD	C, 8
0410	NXTBYT	LD	A, (DE)
0420		INC	DE
0430		PUSH	DE
0440		PUSH	HL

0450	LD	D, #80
0460	LD	E, #80
0470	LD	B, 8
0480	NXTBIT ADD	D
0490	RL	(HL)
0500	DEC	HL
0510	OR	E
0520	SRL	D
0530	SRA	E
0540	SET	6, E
0550	DJNZ	NXTBIT
0560	POP	HL
0570	POP	DE
0580	DEC	C
0590	JR	NZ, NXTBYT
0600	CALL	ROMDIS
0610	POP	AF
0620	CALL	TXTOUT
0630	POP	DE
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 llamada es el mismo que en la anterior:

```
CALL 40000,X,Y,CA$
```

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	;		
0010		ORG	40000
0020	CURSOR	DEFL	#BB75
0030	MATRIX	DEFL	#BBA5
0040	TABLE	DEFL	#BBAAE
0050	ROMENA	DEFL	#B906
0060	ROMDIS	DEFL	#B909
0070	TXTOUT	DEFL	#BB5A
0080		LD	L, (IX+0)
0090		LD	H, (IX+1)
0100		LD	A, (HL)
0110		OR	A
0120		RET	Z
0130		INC	HL
0140		LD	E, (HL)
0150		INC	HL
0160		LD	D, (HL)
0170		LD	L, (IX+2)
0180		LD	H, (IX+4)
0190		LD	B, A
0200	NXTCHR	PUSH	BC
0210		PUSH	HL
0220		CALL	CURSOR
0230		LD	A, (DE)
0240		CALL	ROTCHR
0250		INC	DE
0260		POP	HL
0270		INC	L
0280		POP	BC
0290		DJNZ	NXTCHR
0300		RET	
0310	ROTCHR	PUSH	DE
0320		CALL	MATRIX

0330		EX	DE, HL
0340		CALL	TABLE
0350		PUSH	AF
0360		CALL	ROMENA
0370		LD	C, 8
0380	NXTBYT	LD	A, (DE)
0390		INC	DE
0400		PUSH	DE
0410		PUSH	HL
0420		LD	D, #80
0430		LD	E, #80
0440		LD	B, 8
0450	NXTBIT	ADD	D
0460		RR	(HL)
0470		INC	HL
0480		OR	E
0490		SRL	D
0500		SRA	E
0510		SET	6, E
0520		DJNZ	NXTBIT
0530		POP	HL
0540		POP	DE
0550		DEC	C
0560		JR	NZ, NXTBYT
0570		CALL	ROMDIS
0580		POP	AF
0590		CALL	TXTOUT
0600		POP	DE
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 tecldea 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 qué 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      ;
0010      ORG      40000
0020      ROMENA  DEFL  #B906
0030      ROMDIS  DEFL  #B909
0040      MATRIX  DEFL  #BBA5
0050      TABLE  DEFL  #BBAE
0060      CURSOR  DEFL  #BB75
0070      GETPEN  DEFL  #BB93
0080      SETPEN  DEFL  #BB90
0090      TXTOUT  DEFL  #BB5A
0100      CALL    GETPEN
0110      PUSH    AF
0120      LD      L, (IX+#04)
0130      LD      H, (IX+#05)
0140      LD      B, (HL)
0150      INC     HL
0160      LD      E, (HL)
0170      INC     HL
0180      LD      D, (HL)
0190      LD      L, (IX+#06)
0200      LD      H, (IX+#08)
0210      NXTCHR  PUSH  BC
0220      PUSH    DE
0230      PUSH    HL
0240      LD      A, (DE)
0250      LD      B, A
0260      CALL    ROMENA
0270      LD      A, B
0280      CALL    BIGCHR
0290      LD      B, A
0300      CALL    ROMDIS
0310      POP     HL

```

0320	LD	E, L
0330	LD	D, H
0340	CALL	CURSOR
0350	LD	A, (IX+#02)
0360	CALL	SETPEN
0370	LD	A, B
0380	CALL	TXTOUT
0390	INC	A
0400	CALL	TXTOUT
0410	LD	L, E
0420	LD	H, D
0430	INC	L
0440	CALL	CURSOR
0450	LD	A, (IX+#00)
0460	CALL	SETPEN
0470	LD	A, B
0480	INC	A
0490	INC	A
0500	CALL	TXTOUT
0510	INC	A
0520	CALL	TXTOUT
0530	LD	L, E
0540	LD	H, D
0550	INC	H
0560	INC	H
0570	POP	DE
0580	INC	DE
0590	POP	BC
0600	DJNZ	NXTCHR
0610	POP	AF
0620	CALL	SETPEN
0630	RET	
0640	BIGCHR CALL	MATRIX
0650	EX	DE, HL

0660		CALL	TABLE
0670		PUSH	AF
0680		LD	C, #02
0690	NXTSET	LD	B, #04
0700	NXTROW	PUSH	BC
0710		LD	A, (DE)
0720		RRCA	
0730		RRCA	
0740		RRCA	
0750		RRCA	
0760		LD	B, #04
0770	LFTBYT	RRA	
0780		RR	(HL)
0790		SRA	(HL)
0800		DJNZ	LFTBYT
0810		LD	A, (HL)
0820		INC	HL
0830		LD	(HL), A
0840		LD	B, #07
0850	NXT1	INC	HL
0860		DJNZ	NXT1
0870		LD	A, (DE)
0880		LD	B, #04
0890	RGTTYT	RRA	
0900		RR	(HL)
0910		SRA	(HL)
0920		DJNZ	RGTTYT
0930		LD	A, (HL)
0940		INC	HL
0950		LD	(HL), A
0960		LD	B, #07
0970	NXT2	DEC	HL
0980		DJNZ	NXT2
0990		INC	DE

1000		POP	BC
1010		DJNZ	NXTROW
1020		LD	B, #08
1030	NXT3	INC	HL
1040		DJNZ	NXT3
1050		DEC	C
1060		JR	NZ, NXTSET
1070		POP	AF
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 escriben 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,ES,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 del CALL es:

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

Los parámetros del 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 quieres usar más de un carácter, llame a la rutina una vez por cada uno que quiera 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      ;
0010      ORG      40000
0020      GETPEN  DEFL  #BB93
0030      SETPEN  DEFL  #BB90
0040      MATRIX  DEFL  #BBA5
0050      CURSOR  DEFL  #BB75
0060      TXTOUT  DEFL  #BB5A
0070      ROMENA  DEFL  #B906
0080      ROMDIS  DEFL  #B909
0090      CALL    GETPEN
0100      PUSH    AF
0110      CALL    ROMENA
0120      LD      A, (IX+#04)
0130      CALL    MATRIX
0140      EX      DE, HL
0150      LD      L, (IX+#06)
0160      LD      H, (IX+#08)
0170      LD      B, #08
0180      NXTROW  PUSH    BC
```

0190		LD	C, #80
0200		LD	B, #02
0210	AGAIN	PUSH	BC
0220		PUSH	HL
0230		CALL	CURSOR
0240		BIT	0, B
0250		JR	Z, COL2
0260		LD	A, (IX+#00)
0270		JR	SETCOL
0280	COL2	LD	A, (IX+#02)
0290	SETCOL	CALL	SETPEN
0300		POP	HL
0310		LD	B, #08
0320	NXTCOL	LD	A, (DE)
0330		AND	C
0340		JR	Z, SPACE
0350		LD	A, #8F
0360		JR	PRINT
0370	SPACE	LD	A, #20
0380	PRINT	CALL	TXTOUT
0390		CALL	TXTOUT
0400		SRL	C
0410		DJNZ	NXTCOL
0420		INC	L
0430		POP	BC
0440		DJNZ	AGAIN
0450		INC	DE
0460		POP	BC
0470		DJNZ	NXTROW
0480		CALL	ROMDIS
0490		POP	AF
0500		CALL	SETPEN
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 sólo 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, y le dice a la rutina la cantidad de espacios de la pantalla que se van a rellenar con el carácter. En modo 1, es de $40 \times 25 = 1000$. En modo 0 hay sólo 500 (20×25) 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      ;
0010      ORG      40000
0020      PRNT     DEFL      #BBSA
0030      LD       A, (IX+#00)
01E803 0040      LD       BC,1000
0050      LOOP    PUSH      AF
0060      PUSH     BC
0070      CALL    PRNT
0080      POP      BC
0090      DEC      BC
0100      LD       A, B
0110      OR       C
0120      JR       Z, FIN
0130      POP      AF
0140      JR       LOOP
0150      FIN     POP      AF
0160      RET
0170      END

```

Si tiene un ensamblador, sólo 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 hecho 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 (o el resto de modelos con unidad de cinta externa).

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 'LOAD/SAVE 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,BB,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 del 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 SYNC la usa el Amstrad para distinguir la cabecera (carácter 44) de los datos (representados por un valor SYNC 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      ;      LOAD/SAVE SIN CABECERA
0002      ;
0010      ORG      40000
0020      ;      RUTINA DE SAVE
0030      SAVE    DEFL    #BC9E
0040      LOAD    DEFL    #BCA1
0050      TXTOUT  DEFL    #BB5A
0060      KRESET  DEFL    #BB03
0070      CP      #03
0080      JR      Z, SUSYNC
0090      LD      HL, PARAM

```


0100		CP	#02
0110		JR	NZ, ERROR
0120		XOR	A
0130		LD	E, (IX+#00)
0140		LD	D, (IX+#01)
0150		LD	L, (IX+#02)
0160		LD	H, (IX+#03)
0170		JR	SVESET
0180	SVSYNC	LD	A, (IX+#00)
0190		LD	E, (IX+#02)
0200		LD	D, (IX+#03)
0210		LD	L, (IX+#04)
0220		LD	H, (IX+#05)
0230	SVESET	CALL	SAVE
0240		JR	NC, ESC
0250		RET	
0260	;		
0270	;	RUTINA DE LOAD	
0280		CP	#03
0290		JR	Z, LDSYNC
0300		LD	HL, PARAM
0310		CP	#02
0320		JR	NZ, ERROR
0330		XOR	A
0340		LD	E, (IX+#00)
0350		LD	D, (IX+#01)
0360		LD	L, (IX+#02)
0370		LD	H, (IX+#03)
0380		JR	LDSET
0390	LDSYNC	LD	A, (IX+#00)
0400		LD	E, (IX+#02)
0410		LD	D, (IX+#03)
0420		LD	L, (IX+#04)
0430		LD	H, (IX+#05)

0440	LDSET	CALL	LOAD
0450		RET	C
0460		OR	A
0470		JR	Z, ESC
0480		LD	HL, TAPERR
0490		JR	ERROR
0500	ESC	CALL	KRESET
0510		LD	HL, ESCAPE
0520	ERROR	LD	A, (HL)
0530		INC	HL
0540		BIT	7, A
0550		JR	NZ, ENDERR
0560		CALL	TXTOUT
0570		JR	ERROR
0580	ENDERR	RES	7, A
0590		CALL	TXTOUT
0600		LD	A, #07
0610		CALL	TXTOUT
0620		RET	
0630	PARAM	DEFM	"*FALTA"
0640		DEFM	" PARAMETRO"
0650		DEFB	"*"+#80
0660	TAPERR	DEFM	"*ERROR"
0670		DEFM	" LECTURA"
0680		DEFB	"*"+#80
0690	ESCAPE	DEFM	"*ESCAPE"
0700		DEFB	"*"+#80
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
```

El primer CALL debe ir seguido 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 se mire en el manual del Amstrad los 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). El CALL que tiene la subrutina sólo 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 sólo un ejemplo. El sonido usa solamente el canal 1 y el ENV de volumen, la envolvente 15. Esta envolvente puede ser redefinida en 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      ;
0010      ORG      40000
0020      AMPENU  DEFL  #BCBC
0030      ADDSND  DEFL  #BCAA
0040      OR      A
0050      JR      Z, NXTSND
0060      LD      HL, MUSIC
0070      LD      (DATADR+1), HL
0080      LD      A, #01
0090      LD      (TIME), A
0100      LD      A, #0F
0110      LD      HL, AMPL
0120      CALL   AMPENU
0130      RET
0140      NXTSND LD      HL, TIME
0150      DEC     (HL)
0160      RET     NZ
0170      DATADR LD      DE, MUSIC
0180      LD      A, (DE)
0190      OR      A
0200      RET     Z
0210      INC     DE
0220      LD      (HL), A
0230      INC     A
0240      JR      NZ, CONT
0250      LD      A, (DE)
0260      LD      (HL), A
0270      LD      HL, MUSIC
0280      LD      (DATADR+1), HL
0290      RET
0300      CONT   LD      A, (DE)
0310      LD      L, A
0320      INC     DE

```

0330		LD	A, (DE)
0340		LD	H, A
0350		INC	DE
0360		LD	(DATADR+1), DE
0370		LD	(TONE), HL
0380		LD	HL, SOUND
0390		CALL	ADDSND
0400		RET	
0410	TIME	DEFB	0
0420	AMPL	DEFB	2, 3, 5, 1
0430		DEFB	15, 255, 10
0440	SOUND	DEFB	129, 15, 0
0450	TONE	DEFB	0, 0, 0, 0, 0, 0
0460	MUSIC	DEFB	2
0470		DEFW	426
0480		DEFB	2
0490		DEFW	402
0500		DEFB	1
0510		DEFW	379
0520		DEFB	2
0530		DEFW	239
0540		DEFB	1
0550		DEFW	379
0560		DEFB	2
0570		DEFW	239
0580		DEFB	1
0590		DEFW	379
0600		DEFB	6
0610		DEFW	239
0620		DEFB	1
0630		DEFW	239
0640		DEFB	1
0650		DEFW	213
0660		DEFB	1

0670	DEFW	201
0680	DEFB	1
0690	DEFW	190
0700	DEFB	1
0710	DEFW	239
0720	DEFB	1
0730	DEFW	213
0740	DEFB	2
0750	DEFW	190
0760	DEFB	1
0770	DEFW	253
0780	DEFB	2
0790	DEFW	213
0800	DEFB	6
0810	DEFW	239
0820	DEFB	2
0830	DEFW	426
0840	DEFB	2
0850	DEFW	402
0860	DEFB	1
0870	DEFW	379
0880	DEFB	2
0890	DEFW	239
0900	DEFB	1
0910	DEFW	379
0920	DEFB	2
0930	DEFW	239
0940	DEFB	1
0950	DEFW	379
0960	DEFB	6
0970	DEFW	239
0980	DEFB	1
0990	DEFW	284
1000	DEFB	1

1010	DEFW	319
1020	DEFB	1
1030	DEFW	338
1040	DEFB	1
1050	DEFW	284
1060	DEFB	1
1070	DEFW	239
1080	DEFB	2
1090	DEFW	190
1100	DEFB	1
1110	DEFW	213
1120	DEFB	1
1130	DEFW	239
1140	DEFB	1
1150	DEFW	284
1160	DEFB	6
1170	DEFW	213
1180	DEFB	2
1190	DEFW	426
1200	DEFB	2
1210	DEFW	402
1220	DEFB	1
1230	DEFW	379
1240	DEFB	2
1250	DEFW	239
1260	DEFB	1
1270	DEFW	379
1280	DEFB	2
1290	DEFW	239
1300	DEFB	1
1310	DEFW	379
1320	DEFB	6
1330	DEFW	239
1340	DEFB	1

1350	DEFW	239
1360	DEFB	1
1370	DEFW	213
1380	DEFB	1
1390	DEFW	201
1400	DEFB	1
1410	DEFW	190
1420	DEFB	1
1430	DEFW	239
1440	DEFB	1
1450	DEFW	213
1460	DEFB	2
1470	DEFW	190
1480	DEFB	1
1490	DEFW	253
1500	DEFB	2
1510	DEFW	213
1520	DEFB	6
1530	DEFW	239
1540	DEFB	1
1550	DEFW	239
1560	DEFB	1
1570	DEFW	213
1580	DEFB	1
1590	DEFW	190
1600	DEFB	1
1610	DEFW	239
1620	DEFB	1
1630	DEFW	213
1640	DEFB	2
1650	DEFW	190
1660	DEFB	1
1670	DEFW	239
1680	DEFB	1

1690	DEFW	213
1700	DEFB	1
1710	DEFW	239
1720	DEFB	1
1730	DEFW	190
1740	DEFB	1
1750	DEFW	239
1760	DEFB	1
1770	DEFW	213
1780	DEFB	2
1790	DEFW	190
1800	DEFB	1
1810	DEFW	239
1820	DEFB	1
1830	DEFW	213
1840	DEFB	1
1850	DEFW	239
1860	DEFB	1
1870	DEFW	190
1880	DEFB	1
1890	DEFW	239
1900	DEFB	1
1910	DEFW	213
1920	DEFB	2
1930	DEFW	190
1940	DEFB	1
1950	DEFW	253
1960	DEFB	2
1970	DEFW	213
1980	DEFB	6
1990	DEFW	239
2000	DEFB	1
2010	DEFW	379
2020	DEFB	1

2030	DEFW	358
2040	DEFB	1
2050	DEFW	338
2060	DEFB	2
2070	DEFW	319
2080	DEFB	1
2090	DEFW	284
2100	DEFB	2
2110	DEFW	319
2120	DEFB	1
2130	DEFW	379
2140	DEFB	1
2150	DEFW	358
2160	DEFB	1
2170	DEFW	338
2180	DEFB	2
2190	DEFW	319
2200	DEFB	1
2210	DEFW	284
2220	DEFB	2
2230	DEFW	319
2240	DEFB	1
2250	DEFW	190
2260	DEFB	1
2270	DEFW	239
2280	DEFB	1
2290	DEFW	319
2300	DEFB	1
2310	DEFW	284
2320	DEFB	1
2330	DEFW	253
2340	DEFB	1
2350	DEFW	239
2360	DEFB	1

2370	DEFW	213
2380	DEFB	1
2390	DEFW	190
2400	DEFB	1
2410	DEFW	213
2420	DEFB	1
2430	DEFW	239
2440	DEFB	1
2450	DEFW	213
2460	DEFB	5
2470	DEFW	319
2480	DEFB	1
2490	DEFW	319
2500	DEFB	1
2510	DEFW	379
2520	DEFB	1
2530	DEFW	358
2540	DEFB	2
2550	DEFW	319
2560	DEFB	1
2570	DEFW	284
2580	DEFB	2
2590	DEFW	319
2600	DEFB	1
2610	DEFW	379
2620	DEFB	1
2630	DEFW	358
2640	DEFB	1
2650	DEFW	338
2660	DEFB	2
2670	DEFW	319
2680	DEFB	1
2690	DEFW	284
2700	DEFB	2

2710	DEFW	319
2720	DEFB	1
2730	DEFW	319
2740	DEFB	1
2750	DEFW	284
2760	DEFB	1
2770	DEFW	268
2780	DEFB	1
2790	DEFW	253
2800	DEFB	2
2810	DEFW	253
2820	DEFB	2
2830	DEFW	253
2840	DEFB	1
2850	DEFW	284
2860	DEFB	1
2870	DEFW	338
2880	DEFB	1
2890	DEFW	426
2900	DEFB	5
2910	DEFW	319
2920	DEFB	1
2930	DEFW	379
2940	DEFB	1
2950	DEFW	358
2960	DEFB	1
2970	DEFW	338
2980	DEFB	2
2990	DEFW	319
3000	DEFB	1
3010	DEFW	284
3020	DEFB	2
3030	DEFW	319
3040	DEFB	1

3050	DEFW	379
3060	DEFB	1
3070	DEFW	358
3080	DEFB	1
3090	DEFW	338
3100	DEFB	2
3110	DEFW	319
3120	DEFB	1
3130	DEFW	284
3140	DEFB	2
3150	DEFW	319
3160	DEFB	1
3170	DEFW	190
3180	DEFB	1
3190	DEFW	239
3200	DEFB	1
3210	DEFW	319
3220	DEFB	1
3230	DEFW	284
3240	DEFB	1
3250	DEFW	253
3260	DEFB	1
3270	DEFW	239
3280	DEFB	1
3290	DEFW	213
3300	DEFB	1
3310	DEFW	190
3320	DEFB	1
3330	DEFW	213
3340	DEFB	1
3350	DEFW	239
3360	DEFB	1
3370	DEFW	213
3380	DEFB	5

3390	DEFW	239
3400	DEFB	1
3410	DEFW	319
3420	DEFB	1
3430	DEFW	338
3440	DEFB	1
3450	DEFW	319
3460	DEFB	2
3470	DEFW	239
3480	DEFB	1
3490	DEFW	284
3500	DEFB	2
3510	DEFW	239
3520	DEFB	1
3530	DEFW	284
3540	DEFB	1
3550	DEFW	239
3560	DEFB	1
3570	DEFW	284
3580	DEFB	1
3590	DEFW	319
3600	DEFB	1
3610	DEFW	239
3620	DEFB	1
3630	DEFW	190
3640	DEFB	2
3650	DEFW	159
3660	DEFB	1
3670	DEFW	190
3680	DEFB	1
3690	DEFW	239
3700	DEFB	1
3710	DEFW	319
3720	DEFB	2

3730	DEFW	284
3740	DEFB	2
3750	DEFW	239
3760	DEFB	1
3770	DEFW	190
3780	DEFB	3
3790	DEFW	213
3800	DEFB	6
3810	DEFW	239
3820	DEFB	255,1
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 deCodigoMaquina"
```

```

50 IF PEEK(40000)=&DD AND PEEK(40001)=&6E THEN GOTO 90
60 FOR n=40000 TO 40276
70 READ a$:POKE n,VAL("&"+a$)
80 NEXT
90 LOCATE 11,6:INPUT "Direccion de comienzo";st
100 LOCATE 11,8:PRINT "Para examinar direcciones"
110 LOCATE 11,9:PRINT "&0000-&4000 y &C0000-&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 THEN GOTO 140
160 start%=-INT(UNT(st))-25
170 POKE scroll,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 THEN GOTO 230
240 POKE start%,VAL("&"+h$+l$)
250 POKE scroll,1:GOTO 180
260 a$=UPPER$(INKEY$):IF a$<"0" OR a$>"F" OR a$>"9" AND
    a$<"A" THEN GOTO 260
270 PRINT a$;
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,D5,47,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? &73F

Para examinar direcciones
&0000-&4000 y &C000-&FFFF
(A) ROM o (B) RAM?

073F	41	A
0740	6D	m
0741	73	s
0742	74	t
0743	72	r
0744	61	a
0745	64	d
0746	00	
0747	4F	O
0748	72	r
0749	69	i
074A	6F	o
074B	6E	n
074C	00	
074D	53	S
074E	63	c
074F	68	h
0750	6E	n
0751	65	e
0752	69	i
0753	64	d
0754	65	e
0755	72	r
0756	00	
0757	41	A

```

0001      ;      MONITOR C/M
0002      ;
0010      ORG      40000
0020      UROMON  DEFL  #B900
0030      UROMOF  DEFL  #B903
0040      LROMON  DEFL  #B906
0050      LROMOF  DEFL  #B909
0060      TSTKEY  DEFL  #BB1E
0070      KRESET  DEFL  #BB03
0080      SCROLL  DEFL  #BC4D
0090      TXTOUT  DEFL  #BB5A
0100      LD      L, (IX+#00)
0110      LD      H, (IX+#01)
0120      LD      E, (HL)
0130      INC     HL
0140      LD      D, (HL)
0150      LD      L, (IX+#02)
0160      LD      H, (IX+#03)
0170      LD      A, (HL)
0180      OR      A
0190      JR      Z, ROMOFF
0200      CALL   UROMON
0210      CALL   LROMON
0220      JR      ROMSET
0230      ROMOFF  CALL   UROMOF
0240      CALL   LROMOF
0250      ROMSET  LD      B, #19
0260      SETUP  PUSH   BC
0270      LD      A, #19
0280      CALL   NOWAIT
0290      POP    BC
0300      DJNZ   SETUP
0310      MAIN   LD      A, #39

```

0320		CALL	TSTKEY
0330		JR	Z,NOT3
0340		LD	A,#19
0350		CALL	NOWAIT
0360		JR	MAIN
0370	NOT3	LD	A,#3F
0380		CALL	TSTKEY
0390		JR	Z,NOTX
0400		LD	A,#01
0410		CALL	NOWAIT
0420		JR	MAIN
0430	NOTX	LD	A,#3A
0440		CALL	TSTKEY
0450		JR	Z,NOTE
0460		LD	A,#19
0470		CALL	PAUSA
0480		JR	MAIN
0490	NOTE	LD	A,#3D
0500		CALL	TSTKEY
0510		JR	Z,NOTD
0520		LD	A,#01
0530		CALL	PAUSA
0540		JR	MAIN
0550	NOTD	LD	A,#12
0560		CALL	TSTKEY
0570		JR	NZ,ENTER
0580		LD	A,#43
0590		CALL	TSTKEY
0600		JR	Z,MAIN
0610		LD	L,(IX+#02)
0620		LD	H,(IX+#03)
0630		LD	(HL),#63
0640	ENTER	LD	L,(IX+#00)
0650		LD	H,(IX+#01)

0660		LD	(HL), E
0670		INC	HL
0680		LD	(HL), D
0690		CALL	KRESET
0700		RET	
0710	PAUSA	PUSH	AF
0720		LD	BC, #5000
0730	DELAY	DEC	BC
0740		LD	A, B
0750		OR	C
0760		JR	NZ, DELAY
0770		POP	AF
0780	NOWAIT	PUSH	AF
0790		PUSH	DE
0800		LD	B, A
0810		SRL	B
0820		SRL	B
0830		SRL	B
0840		SRL	B
0850		XOR	A
0860		CALL	SCROLL
0870		LD	A, #1F
0880		CALL	TXTOUT
0890		LD	A, #05
0900		CALL	TXTOUT
0910		POP	DE
0920		POP	AF
0930		PUSH	AF
0940		CALL	TXTOUT
0950		CP	#01
0960		JR	Z, DOWN
0970		INC	DE
0980		LD	HL, #0018
0990		ADD	HL, DE

1000		JR	PRINTT
1010	DOWN	DEC	DE
1020		LD	H, D
1030		LD	L, E
1040	PRINTT	LD	B, H
1050		CALL	HEXOUT
1060		LD	B, L
1070		CALL	HEXOUT
1080		LD	A, #1F
1090		CALL	TXTOUT
1100		LD	A, #0D
1110		CALL	TXTOUT
1120		POP	AF
1130		PUSH	AF
1140		CALL	TXTOUT
1150		LD	B, (HL)
1160		CALL	HEXOUT
1170		POP	BC
1180		LD	A, (HL)
1190		CP	#21
1200		RET	M
1210		CP	#7F
1220		RET	P
1230		LD	A, #1F
1240		CALL	TXTOUT
1250		LD	A, #13
1260		CALL	TXTOUT
1270		LD	A, B
1280		CALL	TXTOUT
1290		LD	A, (HL)
1300		CALL	TXTOUT
1310		RET	
1320	HEXOUT	LD	A, B
1330		AND	#0F

1340		LD	C, A
1350		SRL	B
1360		SRL	B
1370		SRL	B
1380		SRL	B
1390		LD	A, B
1400		LD	B, #02
1410	DIGIT	CP	#0A
1420		JP	P, LETTER
1430		ADD	#30
1440		JR	PRNTCH
1450	LETTER	ADD	#37
1460	PRNTCH	CALL	TXTOUT
1470		LD	A, C
1480		DJNZ	DIGIT
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, con 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 sólo 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.

```

0001      ;      SCROLL BLOQUE ARRIBA
0002      ;
0010      ORG      40000
0020      CHRPOS  DEFL  #BC1A
0030      LD      L, (IX+#02)
0040      LD      H, (IX+#06)
0050      DEC     L
0060      DEC     H
0070      LD      A, (IX+#04)
0080      SUB     H
0090      LD      C, A
0100      LD      A, (IX+#00)
0110      SUB     L
0120      LD      D, A
0130      CALL   CHRPOS
0140      LD      A, #00
0150      CHRWID  ADD   C
0160      DJNZ   CHRWID
0170      LD      E, A
0180      JR     START
0190      NXTROW  PUSH  DE
0200      LD      B, E
0210      LD      E, L
0220      LD      A, H
0230      ADD    #38
0240      LD      D, A
0250      LD      A, L
0260      ADD    #50
0270      LD      L, A
0280      JR     NC, NEWLIN
0290      INC    H
0300      LD      A, H
0310      AND    #07
0320      JR     NZ, NEWLIN
0330      LD      A, H
0340      SUB    #08

```

0350		LD	H, A
0360	NEMLIN	PUSH	HL
0370	LASTLN	LD	A, (HL)
0380		LD	(DE), A
0390		INC	E
0400		JR	NZ, DEOK
0410		INC	D
0420		LD	A, D
0430		AND	#07
0440		JR	NZ, DEOK
0450		LD	A, D
0460		SUB	#08
0470		LD	D, A
0480	DEOK	INC	L
0490		JR	NZ, HLOK
0500		INC	H
0510		LD	A, H
0520		AND	#07
0530		JR	NZ, HLOK
0540		LD	A, H
0550		SUB	#08
0560		LD	H, A
0570	HLOK	DJNZ	LASTLN
0580		POP	HL
0590		POP	DE
0600	START	PUSH	DE
0610		PUSH	HL
0620		LD	C, E
0630	NXTCHR	PUSH	HL
0640		LD	B, #07
0650	NXTLIN	LD	E, L
0660		LD	D, H
0670		LD	A, H
0680		ADD	#08
0690		LD	H, A
0700		LD	A, (HL)

0710		LD	(DE), A
0720		DJNZ	NXTLIN
0730		POP	HL
0740		INC	L
0750		JR	NZ, OK
0760		INC	H
0770		LD	A, H
0780		AND	#07
0790		JR	NZ, OK
0800		LD	A, H
0810		SUB	#08
0820		LD	H, A
0830	OK	DEC	C
0840		JR	NZ, NXTCHR
0850		POP	HL
0860		POP	DE
0870		DEC	D
0880		JR	NZ, NXTROW
0890		LD	A, H
0900		ADD	#38
0910		LD	H, A
0920		LD	B, E
0930	BLANK	LD	(HL), #00
0940		INC	L
0950		JR	NZ, CONT
0960		INC	H
0970		LD	A, H
0980		AND	#07
0990		JR	NZ, CONT
1000		LD	A, H
1010		SUB	#08
1020		LD	H, A
1030	CONT	DJNZ	BLANK
1040		RET	
1050		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, un simple CALL dará como resultado un movimiento de un punto, por lo que será necesario un bucle si intenta mover varios puntos el bloque.

```

0001      ;      SCROLL BLOQUE ABAJO
0002      ;
0010      ORG      40000
0020      CHRPOS  DEFL  #BC1A
0030      LD       L, (IX+#00)
0040      LD       H, (IX+#06)
0050      DEC      L
0060      DEC      H
0070      LD       A, (IX+#04)
0080      SUB      H
0090      LD       C, A
0100      LD       A, L
0110      ADD      #02
0120      SUB      (IX+#02)
0130      LD       D, A
0140      CALL    CHRPOS
0150      LD       A, H
0160      ADD      #38
0170      LD       H, A
0180      LD       A, #00
0190      CHRWID  ADD      C
0200      DJNZ    CHRWID
0210      LD       E, A
0220      JR       START
0230      NXTROW  PUSH    DE
0240      LD       B, E
0250      LD       E, L
0260      LD       A, H
0270      SUB      #38
0280      LD       D, A
0290      LD       A, L
0300      SUB      #50
0310      LD       L, A
0320      JR       NC, L9C7A

```

0330		LD	A, H
0340		DEC	H
0350		AND	#07
0360		JR	NZ, L9C7A
0370		LD	A, H
0380		ADD	#08
0390		LD	H, A
0400	NEWLIN	PUSH	HL
0410	LATLN	LD	A, (HL)
0420		LD	(DE), A
0430		INC	E
0440		JR	NZ, DEOK
0450		INC	D
0460		LD	A, D
0470		AND	#07
0480		JR	NZ, DEOK
0490		LD	A, D
0500		SUB	#08
0510		LD	D, A
0520	DEOK	INC	L
0530		JR	NZ, HLOK
0540		INC	H
0550		LD	A, H
0560		AND	#07
0570		JR	NZ, HLOK
0580		LD	A, H
0590		SUB	#08
0600		LD	H, A
0610	HLOK	DJNZ	LATLN
0620		POP	HL
0630		POP	DE
0640	START	PUSH	DE
0650		PUSH	HL
0660		LD	C, E

0670	NXTCHR	PUSH	HL
0680		LD	B, #07
0690	NXTLIN	LD	E, L
0700		LD	D, H
0710		LD	A, H
0720		SUB	#08
0730		LD	H, A
0740		LD	A, (HL)
0750		LD	(DE), A
0760		DJNZ	NXTLIN
0770		POP	HL
0780		INC	L
0790		JR	NZ, OK
0800		INC	H
0810		LD	A, H
0820		AND	#07
0830		JR	NZ, OK
0840		LD	A, H
0850		SUB	#08
0860		LD	H, A
0870	OK	DEC	C
0880		JR	NZ, NXTCHR
0890		POP	HL
0900		POP	DE
0910		DEC	D
0920		JR	NZ, NXTROW
0930		LD	A, H
0940		SUB	#38
0950		LD	H, A
0960		LD	B, E
0970	BLANK	LD	(HL), #00
0980		INC	L
0990		JR	NZ, CONT
1000		INC	H

1010		LD	A, H
1020		AND	#07
1030		JR	NZ, CONT
1040		LD	A, H
1050		SUB	#08
1060		LD	H, A
1070	CONT	DJNZ	BLANK
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, un 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      ;
0010      ORG      40000
0020      CHRPOS  DEFL  #BC1A
0030      SCMODE  DEFL  #BC11
0040      LD      L, (IX+#02)
0050      LD      H, (IX+#06)
0060      DEC     L
0070      DEC     H
0080      LD      A, (IX+#04)
0090      SUB     H
0100      LD      C, A
0110      LD      A, (IX+#00)
0120      SUB     L
0130      LD      D, A
0140      CALL   CHRPOS
0150      LD      A, #00
0160      CHRWID  ADD     C
0170      DJNZ   CHRWID
0180      LD      B, A
0190      CALL   SCMODE
0200      JP     C, MODE0
0210      JR     Z, MODE1
0220      LD      L, (IX+#02)
0230      LD      H, (IX+#04)
0240      DEC     L
0250      DEC     H
0260      PUSH   BC
0270      CALL   CHRPOS
0280      POP    BC
0290      ROW2   LD      C, #08
0300      LINE2  PUSH   BC
0310      PUSH   HL
0320      OR     A

```


0330	BYTE2	RL	(HL)
0340		PUSH	AF
0350		LD	A, L
0360		DEC	L
0370		OR	A
0380		JR	NZ, OK2
0390		LD	A, H
0400		DEC	H
0410		AND	#07
0420		JR	NZ, OK2
0430		LD	A, H
0440		ADD	#08
0450		LD	H, A
0460	OK2	POP	AF
0470		DJNZ	BYTE2
0480		POP	HL
0490		LD	A, H
0500		ADD	#08
0510		LD	H, A
0520		POP	BC
0530		DEC	C
0540		JR	NZ, LINE2
0550		LD	A, H
0560		SUB	#40
0570		LD	H, A
0580		LD	A, L
0590		ADD	#50
0600		LD	L, A
0610		JR	NC, DONE2
0620		INC	H
0630		LD	A, H
0640		AND	#07
0650		JR	NZ, DONE2
0660		LD	A, H

0670		SUB	#08
0680		LD	H, A
0690	DONE2	DEC	D
0700		JR	NZ, ROW2
0710		RET	
0720	MODE1	DEC	B
0730	ROW1	PUSH	DE
0740		LD	C, #08
0750	LINE1	PUSH	BC
0760		PUSH	HL
0770		SLA	(HL)
0780	BYTE1	LD	E, L
0790		LD	D, H
0800		INC	L
0810		JR	NZ, OK1
0820		INC	H
0830		LD	A, H
0840		AND	#07
0850		JR	NZ, OK1
0860		LD	A, H
0870		SUB	#08
0880		LD	H, A
0890	OK1	LD	A, (DE)
0900		SLA	(HL)
0910		JR	C, SETBIT
0920		RES	4, A
0930		JR	BITOK
0940	SETBIT	SET	4, A
0950	BITOK	BIT	4, (HL)
0960		JR	Z, BITSET
0970		SET	0, A
0980	BITSET	LD	(DE), A
0990		DJNZ	BYTE1
1000		RES	4, (HL)

1010		POP	HL
1020		LD	A, H
1030		ADD	#08
1040		LD	H, A
1050		POP	BC
1060		DEC	C
1070		JR	NZ, LINE1
1080		LD	A, H
1090		SUB	#40
1100		LD	H, A
1110		LD	A, L
1120		ADD	#50
1130		LD	L, A
1140		JR	NC, DONE1
1150		INC	H
1160		LD	A, H
1170		AND	#07
1180		JR	NZ, DONE1
1190		LD	A, H
1200		SUB	#08
1210		LD	H, A
1220	DONE1	POP	DE
1230		DEC	D
1240		JR	NZ, ROW1
1250		RET	
1260	MODE0	DEC	B
1270	ROW0	PUSH	DE
1280		LD	C, #08
1290	LINE0	PUSH	BC
1300		PUSH	HL
1310	BYTE0	PUSH	BC
1320		LD	E, L
1330		LD	D, H
1340		INC	L

1350		JR	NZ, OK0
1360		INC	H
1370		LD	A, H
1380		AND	#07
1390		JR	NZ, OK0
1400		LD	A, H
1410		SUB	#08
1420		LD	H, A
1430	OK0	LD	A, (HL)
1440		RRA	
1450		LD	C, A
1460		LD	A, (DE)
1470		RLA	
1480		RLA	
1490		LD	B, #04
1500	NXTROT	RLA	
1510		SLA	C
1520		SLA	C
1530		RLA	
1540		DJNZ	NXTROT
1550		LD	(DE), A
1560		POP	BC
1570		DJNZ	BYTE0
1580		LD	A, (HL)
1590		RLA	
1600		LD	B, #04
1610	LSTBYT	RLCA	
1620		SLA	A
1630		DJNZ	LSTBYT
1640		LD	(HL), A
1650		POP	HL
1660		LD	A, H
1670		ADD	#08
1680		LD	H, A

1690		POP	BC
1700		DEC	C
1710		JR	NZ, LINE0
1720		LD	A, H
1730		SUB	#40
1740		LD	H, A
1750		LD	A, L
1760		ADD	#50
1770		LD	L, A
1780		JR	NC, DONE0
1790		INC	H
1800		LD	A, H
1810		AND	#07
1820		JR	NZ, DONE0
1830		LD	A, H
1840		SUB	#08
1850		LD	H, A
1860	DONE0	POP	DE
1870		DEC	D
1880		JR	NZ, ROW0
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 rutinas anteriores:

CALL 40000, L, R, U, D

Y como en las otras rutinas, un 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      ;
0010          ORG      40000
0020      CHRPOS  DEFL   #BC1A
0030      SCMODE  DEFL   #BC11
0040          LD      L, (IX+#02)
0050          LD      H, (IX+#04)
0060          DEC     L
0070          DEC     H
0080          LD      A, H
0090          ADD     #02
0100          SUB     (IX+#06)
0110          LD      C, A
0120          LD      A, (IX+#00)
0130          SUB     L
0140          LD      D, A
0150          CALL   CHRPOS
0160          DEC     HL
0170          LD      A, #00
0180      CHRWID  ADD     C
0190          INC     HL
0200          DJNZ   CHRWID
```

0210		LD	B, A
0220		CALL	SCMODE
0230		JP	C, MODE0
0240		JR	Z, MODE1
0250		LD	L, (IX+#02)
0260		LD	H, (IX+#06)
0270		DEC	L
0280		DEC	H
0290		PUSH	BC
0300		CALL	CHRPOS
0310		POP	BC
0320	ROW2	LD	C, #08
0330	LINE2	PUSH	BC
0340		PUSH	HL
0350		OR	A
0360	BYTE2	RR	(HL)
0370		PUSH	AF
0380		INC	L
0390		JR	NZ, OK2
0400		INC	H
0410		LD	A, H
0420		AND	#07
0430		JR	NZ, OK2
0440		LD	A, H
0450		SUB	#08
0460		LD	H, A
0470	OK2	POP	AF
0480		DJNZ	BYTE2
0490		POP	HL
0500		LD	A, H
0510		ADD	#08

0520		LD	H, A
0530		POP	BC
0540		DEC	C
0550		JR	NZ, LINE2
0560		LD	A, H
0570		SUB	#40
0580		LD	H, A
0590		LD	A, L
0600		ADD	#50
0610		LD	L, A
0620		JR	NC, DONE2
0630		INC	H
0640		LD	A, H
0650		AND	#07
0660		JR	NZ, DONE2
0670		LD	A, H
0680		SUB	#08
0690		LD	H, A
0700	DONE2	DEC	D
0710		JR	NZ, ROW2
0720		RET	
0730	MODE1	DEC	B
0740	ROW1	PUSH	DE
0750		LD	C, #08
0760	LINE1	PUSH	BC
0770		PUSH	HL
0780		SRL	(HL)
0790	BYTE1	LD	E, L
0800		LD	D, H
0810		LD	A, L
0820		DEC	L

0830		OR	A
0840		JR	NZ, OK1
0850		LD	A, H
0860		DEC	H
0870		AND	#07
0880		JR	NZ, OK1
0890		LD	A, H
0900		ADD	#08
0910		LD	H, A
0920	OK1	LD	A, (DE)
0930		SRL	(HL)
0940		JR	C, SETBIT
0950		RES	3, A
0960		JR	BITOK
0970	SETBIT	SET	3, A
0980	BITOK	BIT	3, (HL)
0990		JR	Z, BITSET
1000		SET	7, A
1010	BITSET	LD	(DE), A
1020		DJNZ	BYTE1
1030		RES	3, (HL)
1040		POP	HL
1050		LD	A, H
1060		ADD	#08
1070		LD	H, A
1080		POP	BC
1090		DEC	C
1100		JR	NZ, LINE1
1110		LD	A, H
1120		SUB	#40
1130		LD	H, A

1140		LD	A, L
1150		ADD	#50
1160		LD	L, A
1170		JR	NC, DONE1
1180		INC	H
1190		LD	A, H
1200		AND	#07
1210		JR	NZ, DONE1
1220		LD	A, H
1230		SUB	#08
1240		LD	H, A
1250	DONE1	POP	DE
1260		DEC	D
1270		JR	NZ, ROW1
1280		RET	
1290	MODE0	DEC	B
1300	ROW0	PUSH	DE
1310		LD	C, #08
1320	LINE0	PUSH	BC
1330		PUSH	HL
1340	BYTE0	PUSH	BC
1350		LD	E, L
1360		LD	D, H
1370		LD	A, L
1380		DEC	L
1390		OR	A
1400		JR	NZ, OK0
1410		LD	A, H
1420		DEC	H
1430		AND	#07
1440		JR	NZ, OK0

1450		LD	A, H
1460		ADD	#08
1470		LD	H, A
1480	OK0	LD	A, (HL)
1490		RLA	
1500		LD	C, A
1510		LD	A, (DE)
1520		RRA	
1530		RRA	
1540		LD	B, #04
1550	NXTROT	RRA	
1560		SRL	C
1570		SRL	C
1580		RRA	
1590		DJNZ	NXTROT
1600		LD	(DE), A
1610		POP	BC
1620		DJNZ	BYTE0
1630		LD	A, (HL)
1640		RRA	
1650		LD	B, #04
1660	LSTBYT	RRCA	
1670		SRL	A
1680		DJNZ	LSTBYT
1690		LD	(HL), A
1700		POP	HL
1710		LD	A, H
1720		ADD	#08
1730		LD	H, A
1740		POP	BC
1750		DEC	C

1760		JR	NZ, LINE0
1770		LD	A, H
1780		SUB	#40
1790		LD	H, A
1800		LD	A, L
1810		ADD	#50
1820		LD	L, A
1830		JR	NC, DONE0
1840		INC	H
1850		LD	A, H
1860		AND	#07
1870		JR	NZ, DONE0
1880		LD	A, H
1890		SUB	#08
1900		LD	H, A
1910	DONE0	POP	DE
1920		DEC	D
1930		JR	NZ, ROW0
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
ICMAYOR	C Mayor
ICMENOR	C Menor
ICSMAYOR	C# Mayor o Db. Mayor
ICSMENOR	C# Menor o Db. Menor
IDMAYOR	D Mayor
IDMENOR	D Menor
IDSMAYOR	D# Mayor o Eb. Mayor
IDSMENOR	D# Menor o Eb. Menor
IEMAYOR	E Mayor
IEMENOR	E Menor
IFMAYOR	F Mayor
IFMENOR	F Menor
IFSMAYOR	F# Mayor o Gb. Mayor
IFSMENOR	F# Menor o GB. Menor
IGMAYOR	G Mayor

I GMENOR	G Menor
I GSMAYOR	G# Mayor o Ab. Mayor
I GSMENOR	G# Menor o Ab. Menor
I AMAYOR	A Mayor
I AMENOR	A Menor
I ASMAYOR	A# Mayor o Bb. Mayor
I ASMENOR	A# Menor o Bb. Menor
I BMAYOR	B Mayor
I 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 AFTER 240
20 FOR n=40000 TO 40617
30 READ a$:POKE n,VAL("&"a$)
40 NEXT
50 DATA 01,4A,9C,21,94,9C,CD,D1,BC,C9
60 DATA 98,9C,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 18,18,4C,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 00,1E,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 envoltura 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 envoltura 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 envoltura al final del comando del acorde. Por ejemplo, si su envoltura era ENV 1,3,4,1,15,-1,10 con el primer número indicando el número de la envoltura, debe añadir un ‘1’ al final del comando de acorde. Ej. |CMAYOR,1.

El volumen inicial del acorde depende de la envoltura que se use. El programa lo fija de la siguiente forma. Si el tamaño de paso de la primera sección de la envoltura 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 envoltura 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  
ICMAYOR,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  
ICMAYOR,0,4  
IDMAYOR,0,4
```

En este ejemplo, solamente sonará el acorde D Mayor, ya que el primer acorde se cortará tan pronto como empiece. 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 cómo hacerlo:

```
ENV 5,8,0,100  
IEMAYOR,0,5  
FOR T=1 TO 300:NEXT  
ICMAYOR,0,5  
FOR T=1 TO 250:NEXT  
IDMAYOR,0,5
```

```

0001      ;      ACORDES RSX
0002      ;
0010      ORG      40000
0020      SNDRES  DEFL  #BCA7
0030      SUMSND  DEFL  #BCAA
0040      AMPDIR  DEFL  #BCC2
0050      LOGEXT  DEFL  #BCD1
0060      LD      BC, TABLA
0070      LD      HL, ESPAC
0080      CALL   LOGEXT
0090      RET
0100      TABLA  DEFW  NOMBTB
0110      JP      CMAY
0120      JP      CMEN
0130      JP      CSMAY
0140      JP      CSMEN
0150      JP      DMAY
0160      JP      DMEN
0170      JP      DSMAY
0180      JP      DSMEN
0190      JP      EMAY
0200      JP      EMEN
0210      JP      FMAY
0220      JP      FMEN
0230      JP      FSMAY
0240      JP      FSMEN
0250      JP      GMAY
0260      JP      GMEN
0270      JP      GSMAY
0280      JP      GSMEN

```

0290		JP	AMAY
0300		JP	AMEN
0310		JP	ASMAY
0320		JP	ASMEN
0330		JP	BMAY
0340		JP	BMEN
0350	ESPAC	DEFB	0,0,0,0
0360	NOMBTB	DEFM	"CMAYO"
0370		DEFB	"R"+#80
0380		DEFM	"CMENO"
0390		DEFB	"R"+#80
0400		DEFM	"CSMAYO"
0410		DEFB	"R"+#80
0420		DEFM	"CSMENO"
0430		DEFB	"R"+#80
0440		DEFM	"DMAYO"
0450		DEFB	"R"+#80
0460		DEFM	"DMENO"
0470		DEFB	"R"+#80
0480		DEFM	"DSMAYO"
0490		DEFB	"R"+#80
0500		DEFM	"DSMENO"
0510		DEFB	"R"+#80
0520		DEFM	"EMAYO"
0530		DEFB	"R"+#80
0540		DEFM	"EMENO"
0550		DEFB	"R"+#80
0560		DEFM	"FMAYO"
0570		DEFB	"R"+#80
0580		DEFM	"FMENO"
0590		DEFB	"R"+#80

0600		DEFM	"FSMAYO"
0610		DEFB	"R"+#80
0620		DEFM	"FSMENO"
0630		DEFB	"R"+#80
0640		DEFM	"GMAYO"
0650		DEFB	"R"+#80
0660		DEFM	"GMENO"
0670		DEFB	"R"+#80
0680		DEFM	"GSMAYO"
0690		DEFB	"R"+#80
0700		DEFM	"GSMENO"
0710		DEFB	"R"+#80
0720		DEFM	"AMAYO"
0730		DEFB	"R"+#80
0740		DEFM	"AMENO"
0750		DEFB	"R"+#80
0760		DEFM	"ASMAYO"
0770		DEFB	"R"+#80
0780		DEFM	"ASMENO"
0790		DEFB	"R"+#80
0800		DEFM	"BMAYO"
0810		DEFB	"R"+#80
0820		DEFM	"BMENO"
0830		DEFB	"R"+#80
0840		DEFB	0
0850	C MAY	LD	E, #00
0860		JR	ACORD
0870	C MEN	LD	E, #06
0880		JR	ACORD
0890	C SMAY	LD	E, #0C
0900		JR	ACORD

0910	CSMEN	LD	E, #12
0920		JR	ACORD
0930	DMAY	LD	E, #18
0940		JR	ACORD
0950	DMEN	LD	E, #1E
0960		JR	ACORD
0970	DSMAY	LD	E, #24
0980		JR	ACORD
0990	DSMEN	LD	E, #2A
1000		JR	ACORD
1010	EMAY	LD	E, #30
1020		JR	ACORD
1030	EMEN	LD	E, #36
1040		JR	ACORD
1050	FMAY	LD	E, #3C
1060		JR	ACORD
1070	FMEN	LD	E, #42
1080		JR	ACORD
1090	FSMAY	LD	E, #48
1100		JR	ACORD
1110	FSMEN	LD	E, #4E
1120		JR	ACORD
1130	GMAY	LD	E, #54
1140		JR	ACORD
1150	GMEN	LD	E, #5A
1160		JR	ACORD
1170	GSMAY	LD	E, #60
1180		JR	ACORD
1190	GSMEN	LD	E, #66
1200		JR	ACORD
1210	AMAY	LD	E, #6C

1220		JR	ACORD
1230	AMEN	LD	E, #72
1240		JR	ACORD
1250	ASMAY	LD	E, #78
1260		JR	ACORD
1270	ASMEN	LD	E, #7E
1280		JR	ACORD
1290	BMAY	LD	E, #84
1300		JR	ACORD
1310	BMEN	LD	E, #8A
1320		JR	ACORD
1330	ACORD	LD	HL, DATOS
1340		LD	D, #00
1350		ADD	HL, DE
1360		LD	E, (HL)
1370		INC	HL
1380		LD	D, (HL)
1390		INC	HL
1400		LD	(TON01), DE
1410		LD	E, (HL)
1420		INC	HL
1430		LD	D, (HL)
1440		INC	HL
1450		LD	(TON02), DE
1460		LD	E, (HL)
1470		INC	HL
1480		LD	D, (HL)
1490		LD	(TON03), DE
1500		PUSH	AF
1510		OR	A
1520		JR	Z, PONENU

1530		LD	A, (IX+#00)
1540	PONENU	LD	(ENU1), A
1550		LD	(ENU2), A
1560		LD	(ENU3), A
1570		OR	A
1580		JR	Z, NEGTUO
1590		CALL	AMPDIR
1600		RET	NC
1610		INC	HL
1620		INC	HL
1630		LD	A, (HL)
1640		OR	A
1650		JR	Z, NEGTUO
1660		BIT	7, A
1670		JR	NZ, NEGTUO
1680		LD	A, #00
1690		JR	PONAMP
1700	NEGTUO	LD	A, #0F
1710	PONAMP	LD	(AMPL1), A
1720		LD	(AMPL2), A
1730		LD	(AMPL3), A
1740		POP	AF
1750		CP	#02
1760		CALL	Z, SNDRES
1770		LD	HL, SND1
1780	ACOMP	CALL	SUMSND
1790		JR	NC, ACOMP
1800		LD	HL, SND2
1810	BCOMP	CALL	SUMSND
1820		JR	NC, BCOMP
1830		LD	HL, SND3

1840	CCOMP	CALL	SUMSND
1850		JR	NC, CCOMP
1860		RET	
1870	SND1	DEFB	49
1880	ENV1	DEFB	0, 0
1890	TON01	DEFB	0, 0, 0
1900	AMPL1	DEFB	0, 0, 0
1910	SND2	DEFB	42
1920	ENV2	DEFB	0, 0
1930	TON02	DEFB	0, 0, 0
1940	AMPL2	DEFB	0, 0, 0
1950	SND3	DEFB	28
1960	ENV3	DEFB	0, 0
1970	TON03	DEFB	0, 0, 0
1980	AMPL3	DEFB	0, 0, 0
1990	DAT05	DEFW	638, 478, 379
2000		DEFW	638, 478, 402
2010		DEFW	602, 451, 358
2020		DEFW	602, 451, 379
2030		DEFW	676, 568, 426
2040		DEFW	568, 426, 358
2050		DEFW	638, 536, 402
2060		DEFW	676, 536, 402
2070		DEFW	602, 506, 379
2080		DEFW	638, 506, 379
2090		DEFW	568, 478, 358
2100		DEFW	602, 478, 358
2110		DEFW	676, 536, 451
2120		DEFW	676, 568, 451
2130		DEFW	638, 506, 426
2140		DEFW	638, 536, 426

2150	DEFW	602,478,402
2160	DEFW	602,506,402
2170	DEFW	568,451,379
2180	DEFW	568,478,379
2190	DEFW	536,426,358
2200	DEFW	536,451,358
2210	DEFW	676,506,402
2220	DEFW	676,506,426
2230	END	

Órgano 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 en 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 THEN GOTO 40
70 IF a=0 THEN GOTO 120
80 n=n+1
90 IF n<13 THEN 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 ICMAYOR,0,1:RETURN
150 IDMAYOR,0,1:RETURN
160 IEMAYOR,0,1:RETURN
170 IFMAYOR,0,1:RETURN
180 IGMAYOR,0,1:RETURN
190 IAMAYOR,0,1:RETURN
200 IBMAYOR,0,1:RETURN
210 ICMAYOR,0,1:RETURN
220 ICSMAYOR,0,1:RETURN
230 IDSMAYOR,0,1:RETURN
240 IFSMAYOR,0,1:RETURN
250 IGSMAYOR,0,1:RETURN
260 IASMAYOR,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 de 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 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 43830,0A%
```

En la variable A% se almacena 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      ;
0010      ORG      43800
0020      ;      RUTINA 1
0030      LD       DE,#C000
0040      LD       HL,43799
0050      UNCOMPT LD      A,(HL)
0060      LD       (DE),A
0070      INC      DE

```

0080		DEC	HL
0090		BIT	7, D
0100		RET	Z
0110		OR	A
0120		JR	NZ, UNCOMPT
0130		LD	B, (HL)
0140		DEC	HL
0150		DEC	B
0160		JR	Z, UNCOMPT
0170	REPEAT	LD	(DE), A
0180		INC	DE
0190		DJNZ	REPEAT
0200		BIT	7, D
0210		JR	NZ, UNCOMPT
0220		RET	
0230	;	RUTINA	2
0240		CP	#01
0250		RET	NZ
0260		LD	DE, #C000
0270		LD	HL, 43799
0280	COMPCT	LD	A, (DE)
0290		LD	(HL), A
0300		INC	DE
0310		DEC	HL
0320		BIT	7, D
0330		JR	Z, TOTAL
0340		OR	A
0350		JR	NZ, COMPCT
0360		LD	B, #01
0370	COUNT	LD	A, (DE)
0380		OR	A
0390		JR	NZ, GOTLEN
0400		INC	DE
0410		BIT	7, D

0420		JR	Z, LAST
0430		INC	B
0440		JR	NZ, COUNT
0450	GOTLEN	LD	(HL), B
0460		DEC	HL
0470		JR	COMPCT
0480	LAST	INC	B
0490		LD	(HL), B
0500		DEC	HL
0510	TOTAL	EX	DE, HL
0520		LD	HL, 43799
0530		OR	A
0540		SBC	HL, DE
0550		EX	DE, HL
0560		LD	L, (IX+#00)
0570		LD	H, (IX+#01)
0580		LD	(HL), E
0590		INC	HL
0600		LD	(HL), D
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      ;
0010      ORG      43800
0020      ;      RUTINA 1
0030      LD      DE, #C000
0040      LD      HL, 43799
0050      UNCMPT LD      A, (HL)
0060      DEC      HL
0070      LD      B, (HL)
0080      DEC      HL
0090      REPEAT LD      (DE), A
0100      INC      DE
0110      DJNZ     REPEAT
0120      BIT     7, D
0130      JR      NZ, UNCMPT
0140      RET
0150      ;      RUTINA 2
0160      CP      #01
0170      RET     NZ
0180      LD      DE, #C000
0190      LD      HL, 43799
0200      COMPCT LD      A, (DE)
0210      LD      (HL), A
0220      INC      DE
0230      DEC      HL
0240      LD      B, #01
0250      BIT     7, D
0260      JR      Z, TOTAL
0270      LD      C, A
0280      COUNT  LD      A, (DE)
0290      CP      C
0300      JR      NZ, GOTLEN
0310      INC      DE

```

0320		BIT	7, D
0330		JR	Z, LAST
0340		INC	B
0350		JR	NZ, COUNT
0360	GOTLEN	LD	(HL), B
0370		DEC	HL
0380		JR	COMPCT
0390	LAST	INC	B
0400	TOTAL	LD	(HL), B
0410		DEC	HL
0420		EX	DE, HL
0430		LD	HL, 43799
0440		OR	A
0450		SBC	HL, DE
0460		EX	DE, HL
0470		LD	L, (IX+#00)
0480		LD	H, (IX+#01)
0490		LD	(HL), E
0500		INC	HL
0510		LD	(HL), D
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 esa 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 tienen 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 un 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 de 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 y DEEK
0002      ;
0010      ORG      40000
0020      LOGEXT  DEFL  #BCD1
0030      LD       BC, TABLA
0040      LD       HL, ESPACI
0050      CALL    LOGEXT
0060      RET
0070      TABLA   DEFW  NOMBTB
0080      JP      DOKE
0090      JP      DEEK
0100      ESPACI  DEFB  0, 0, 0, 0
0110      NOMBTB  DEFM  "DOK"
0120      DEFB   "E"+#80
0130      DEFW   "DEE"
0140      DEFB   "K"+#80
0150      DEFB   0
0160      DOKE   CP    #02
0170      RET    NZ

```

0180		LD	L, (IX+#02)
0190		LD	H, (IX+#03)
0200		LD	A, (IX+#00)
0210		LD	(HL), A
0220		INC	HL
0230		LD	A, (IX+#01)
0240		LD	(HL), A
0250		RET	
0260	DEEK	CP	#02
0270		RET	NZ
0280		LD	L, (IX+#02)
0290		LD	H, (IX+#03)
0300		LD	E, (IX+#00)
0310		LD	D, (IX+#01)
0320		LD	A, (HL)
0330		INC	HL
0340		LD	(DE), A
0350		INC	DE
0360		LD	A, (HL)
0370		LD	(DE), A
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:

**IEXPLODE, IREADCHAR, IBIGPRINT, IUSCROLL,
IDSCROLL, ILSCROLL, IRSCROLL, ICOMMANDS**

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 DE 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 THEN GOTO 110
90 IF sum<>x(c) THEN PRINT "*DATOS ERRONEOS*";CHR$(7):PRINT
    "Compruebe las lineas ";210+100*c;" a ";300+100*c:END
100 sum=0:c=c+1
110 NEXT
120 IF sum<>x(c) THEN PRINT "*DATOS ERRONEOS*";CHR$(7):PRINT
    "Compruebe la linea 1410"
130 '
140 '          DATOS DE VERIFICACION
150 '
160 DATA 2B79,2F09,2A7A,27A3,263B,2434,2C42
170 DATA 2782,2593,2A89,278C,29D3,04DD
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,4C,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,BD,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 4B,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,2B,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 tecleado y comprobado el código máquina, se debe salvar con una sentencia como esta:

```
SAVE "mcjuego.bin",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 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.bin",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 247,231,90,231,129,66,36,60,126
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 THEN GOTO 470
350 IF v=30 AND w=23 THEN GOTO 1120
360 FOR n=1 TO 3:NEXT
370 IF c=1 THEN |READCHAR,v+4,w+1,@ch%:IF ch%>248 THEN
    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 THEN 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 IREADCHAR,x,y,@ch%:IF ch%>32 THEN t=y:GOTO 710
490 IREADCHAR,x,y+1,@ch%:IREADCHAR,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 THEN GOTO 700
530 IF c=1 THEN IREADCHAR,v+4,w+1,@ch%:IF ch%>248 THEN
    GOTO 920
540 IRESCROLL,v,v+4,w,w+1
550 IDSCROLL,x,x,y,y+2:IDSCROLL,x,x,y,y+2
560 c=c+1:IF c=9 THEN c=1:v=v+1
570 IF v<37 THEN 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 IREADCHAR,x,y+2,@ch%:IF ch%>32 THEN GOTO 700
670 IF y<23 THEN 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*q,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 THEN
    GOTO 920
790 |RSCROLL,v,v+4,w,w+1
800 IF q=4 THEN |DSCROLL,x,z,t,b
810 c=c+1:IF c=9 THEN c=1:v=v+1
820 IF v<37 THEN 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 THEN GOTO 1050
950 t=w:b=t+2
960 |READCHAR,v,b,@ch%:IF ch%>32 THEN GOTO 980
970 b=b+1:IF b<26 THEN 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 THEN GOTO 1030
1020 b=b+1:IF b<26 THEN 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" THEN GOTO 210
1100 IF r$("<"N" THEN GOTO 1080
1110 END
1120 FOR n=1 TO 88
1130 SOUND 135,100-n,2,7
1140 |RSCROLL,30,40,19,24
1150 IF n MOD 3=0 THEN |USCROLL,30,40,19,24
1160 NEXT
1170 PEN 2
1180 m$=" 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.

APÉNDICES

A – Mapa de Memoria

B – Códigos de Operación del Z80

C – Conversión Hexadecimal a Decimal

APÉNDICE – 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.

00000	MEMORIA DE PANTALLA POR DEFECTO	010000	ROM SUPERIOR (banco solapable)
C000	PILA, FECHA Y BLOQUES DE SALTO	C000	
B100	DATOS PRIMARIOS		
AC00	DATOS SECUNDARIOS		
????	MEMORIA LIBRE (Programa del usuario y Variables)	HIMEM	
????	DATOS PRIMARIOS		
????	DATOS SECUNDARIOS	4000	ROM INFERIOR
0040	AREA DEL 'FIRMWARE'		
0000		0000	

APÉNDICE – 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 ó 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 que 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 ;que BC = 0 o coinciden 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 registros 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 el ;puerto indicado por el acumulador y el dato
IN	reg, (C)	;Introduce un dato en reg desde el puerto ;indicado 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 del puerto indicado por ;el registro C y decrementa HL y B
INDR		;Carga (HL) con el dato del puerto indicado por ;el registro C, decrementa HL y B, y repite ;hasta que B = 0
INI		;Carga (HL) con el dato del puerto indicado por ;el registro C, decrementa B e incrementa HL
INIR		;Carga (HL) con el dato del puerto indicado por ;el registro C, decrementa 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,operandl	;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 IY
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), decremента ;DE, BC y HL
LDDR		;Carga (HL) dentro de (DE), decremента ;DE, BC y HL, y repite hasta que BC = 0
LDI		;Carga (HL) dentro de (DE), incrementa ;DE y HL, y decremента BC

LDIR		;Carga (HL) dentro de (DE), incrementa ;DE y HL, decrementa BC y repite hasta ;que BC = 0
NEG		;Niega el acumulador
NOP		;No operación
OR	operand1	;OR lógico entre el operando y el acumulador
OTDR		;Carga (HL) dentro del puerto C, decrementa ;B y HL, y repite hasta que B = 0
OTIR		;Carga (HL) dentro del puerto C, incrementa ;HL, decrementa B y repite hasta que B = 0
OUT	(C), reg	;Carga el registro en el puerto C
OUT	(dato), A	;Carga el acumulador en el puerto dato
OUTD		;Carga (HL) en el puerto C, decrementa HL y B
OUTI		;Carga (HL) en el puerto 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 por Grupos de las Instrucciones del Z80

A continuación le mostraremos una serie de tablas que reúnen la totalidad del juego de instrucciones del procesador Z80 (incluyendo las 'no oficiales'). Cada tabla corresponde a 'un grupo' y contienen un conjunto de instrucciones, asociado a la función que desempeñan.

Para su correcta interpretación, le aconsejamos que no pierda de vista la siguiente leyenda:

Notas: **r,r'** indican cualquier registro del tipo **A, B, C, D, E, H, L**.
p,p' indican cualquier registro del tipo **A, B, C, D, E, IX_H, IX_L**.
q,q' indican cualquier registro del tipo **A, B, C, D, E, IY_H, IY_L**.
dd_L, dd_H indican el octeto de menor o mayor peso del registro respectivamente.
dd indica cualquiera de los pares de registros **BC, DE, HL, SP**.
pp indica cualquiera de los pares de registros **BC, DE, IX, SP**.
qq es cualquiera de los pares de registros **BC, DE, HL, AF**.
rr indica cualquiera de los pares de registros **BC, DE, IY, SP**.
ss indica cualquiera de los pares de registros **BC, DE, HL, SP**.
CY se refiere al flip-flop de Carry.
e es un número con signo en complemento a dos dentro del rango **<-126, 129>**.
d indica el desplazamiento.
n indica un número de un octeto (**8 bits**) de tamaño.
nn indica un número de dos octetos (**16 bits**) de tamaño.
***** indica una instrucción **no oficial**.

Señaladores: **•** = no afectado, **0** = reseteado, **1** = seteado,
b = depende del resultado de la operación,
X = valor desconocido,
IFF₂ = se copia el flip-flop 2 de la interrupción.

Grupo de Carga de 8 bits

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N	C			
LD r, r'	$r \leftarrow r'$		1	4
LD p, p'	$p \leftarrow p'$	DD	2	8
LD q, q'	$q \leftarrow q'$	FD	2	8
LD r, n	$r \leftarrow n$		2	7
LD p, n	$p \leftarrow n$	DD	3	11
LD q, n	$q \leftarrow n$	FD	3	11
LD r, (HL)	$r \leftarrow (HL)$		2	7
LD r, (IX + d)	$r \leftarrow (IX + d)$	DD	5	19
LD r, (IY + d)	$r \leftarrow (IY + d)$	FD	5	19
LD (HL), r	$(HL) \leftarrow r$		2	7
LD (IX + d), r	$(IX + d) \leftarrow r$	DD	5	19
LD (IY + d), r	$(IY + d) \leftarrow r$	FD	5	19
LD (HL), n	$(HL) \leftarrow n$	36	3	10
LD (IX + d), n	$(IX + d) \leftarrow n$	DD 36	5	19
LD (IY + d), n	$(IY + d) \leftarrow n$	FD 36	5	19
LD A, (BC)	$A \leftarrow (BC)$	0A	2	7
LD A, (DE)	$A \leftarrow (DE)$	1A	2	7
LD A, (nn)	$A \leftarrow (nn)$	3A	4	13
LD (BC), A	$(BC) \leftarrow A$	02	2	7
LD (DE), A	$(DE) \leftarrow A$	12	2	7
LD (nn), A	$(nn) \leftarrow A$	32	4	13
LD A, I	$A \leftarrow I$	b	b	b	0	b	IFF ₂	0	.	ED 57	2	9
LD A, R	$A \leftarrow R$	b	b	b	0	b	IFF ₂	0	.	ED 5F	2	9
LD I, A	$I \leftarrow A$	ED 47	2	9
LD R, A	$R \leftarrow A$	ED 4F	2	9

Grupo de Carga de 16 bits

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N	C			
LD dd, nn	$dd \leftarrow nn$		3	10
LD IX, nn	$IX \leftarrow nn$	DD 21	4	14
LD IY, nn	$IY \leftarrow nn$	FD 21	4	14
LD HL, (nn)	$L \leftarrow (nn)$ $H \leftarrow (nn+1)$	2A	5	16
LD dd, (nn)	$dd_L \leftarrow (nn)$ $dd_H \leftarrow (nn+1)$	ED	6	20
LD IX, (nn)	$IX_L \leftarrow (nn)$ $IX_H \leftarrow (nn+1)$	DD 2A	6	20
LD IY, (nn)	$IY_L \leftarrow (nn)$ $IY_H \leftarrow (nn+1)$	FD 2A	6	20
LD (nn), HL	$(nn) \leftarrow L$ $(nn+1) \leftarrow H$	22	5	16
LD (nn), dd	$(nn) \leftarrow dd_L$ $(nn+1) \leftarrow dd_H$	DD	6	20
LD (nn), IX	$(nn) \leftarrow IX_L$ $(nn+1) \leftarrow IX_H$	DD 22	6	20
LD (nn), IY	$(nn) \leftarrow IY_L$ $(nn+1) \leftarrow IY_H$	FD 22	6	20
LD SP, HL	$SP \leftarrow HL$	F9	1	6
LD SP, IX	$SP \leftarrow IX$	DD F9	2	10
LD SP, IY	$SP \leftarrow IY$	FD F9	2	10
PUSH qq	$SP \leftarrow SP - 1$ $(SP) \leftarrow qq_H$ $SP \leftarrow SP - 1$ $(SP) \leftarrow qq_L$		3	11
PUSH IX	$SP \leftarrow SP - 1$ $(SP) \leftarrow IX_H$ $SP \leftarrow SP - 1$ $(SP) \leftarrow IX_L$	DD E5	4	15

PUSH IY	$SP \leftarrow SP - 1$ $(SP) \leftarrow IY_H$ $SP \leftarrow SP - 1$ $(SP) \leftarrow IY_L$	FD E5	4	15
POP qq	$(SP) \leftarrow qq_L$ $SP \leftarrow SP + 1$ $(SP) \leftarrow qq_H$ $SP \leftarrow SP + 1$		3	10
POP IX	$(SP) \leftarrow IX_L$ $SP \leftarrow SP + 1$ $(SP) \leftarrow IX_H$ $SP \leftarrow SP + 1$	DD E1	4	14
POP IY	$(SP) \leftarrow IY_L$ $SP \leftarrow SP + 1$ $(SP) \leftarrow IY_H$ $SP \leftarrow SP + 1$	FD E1	4	14

*Grupo de Intercambio,
Transferencia de Bloques y Búsqueda*

Nemotécnico	Operación Simbólica	Señalizadores							Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N			
EX DE, HL	$DE \leftrightarrow HL$							EB	1	4
EX AF, AF'	$AF \leftrightarrow AF'$							08	1	4
EXX	$BC \leftrightarrow BC'$ $DE \leftrightarrow DE'$ $HL \leftrightarrow HL'$							D9	1	4
EX (SP), HL	$(SP+1) \leftrightarrow H$ $(SP) \leftrightarrow L$							E3	5	19
EX (SP), IX	$(SP+1) \leftrightarrow IX_H$ $(SP) \leftrightarrow IX_L$							DD E3	6	23
EX (SP), IY	$(SP+1) \leftrightarrow IY_H$ $(SP) \leftrightarrow IY_L$							FD E3	6	23
LDI	$(DE) \leftarrow (HL)$ $DE \leftarrow DE + 1$ $HL \leftarrow HL + 1$ $BC \leftarrow BC - 1$. . .	b	0	b	b	0	.	ED A0	4	16

LDIR	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC - 1 Repite hasta: BC = 0	• • b 0 b 0 0 •	ED B0	5 4	21 16
LDD	(DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1	• • b 0 b b 0 •	ED A8	4	16
LDDR	(DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1 Repite hasta: BC = 0	• • b 0 b 0 0 •	ED B8	5 4	21 16
CPI	A - (HL) HL ← HL + 1 BC ← BC - 1	b b b b b b 1 •	ED A1	4	16
CPIR	A - (HL) HL ← HL + 1 BC ← BC - 1 Repite hasta: A = (HL) ó BC = 0	b b b b b b 1 •	ED B1	5 4	21 16
CPD	A - (HL) HL ← HL - 1 BC ← BC - 1	b b b b b b 1 •	ED A9	4	16
CPDR	A - (HL) HL ← HL - 1 BC ← BC - 1 Repite hasta: A = (HL) ó BC = 0	b b b b b b 1 •	ED B9	5 4	21 16

Grupo Aritmético y Lógico de 8 bits

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N	C			
ADD A, r	$A \leftarrow A + r$	b	b	b	b	b	V	0	b		1	4
ADD A, p*	$A \leftarrow A + p$	b	b	b	b	b	V	0	b	DD	2	8
ADD A, q*	$A \leftarrow A + q$	b	b	b	b	b	V	0	b	FD	2	8
ADD A, n	$A \leftarrow A + n$	b	b	b	b	b	V	0	b		2	8
ADD A, (HL)	$A \leftarrow A + (HL)$	b	b	b	b	b	V	0	b		2	7
ADD A, (IX + d)	$A \leftarrow A + (IX + d)$	b	b	b	b	b	V	0	b	DD	5	19
ADD A, (IY + d)	$A \leftarrow A + (IY + d)$	b	b	b	b	b	V	0	b	FD	5	19
ADC A, s	$A \leftarrow A + s + CY$	b	b	b	b	b	V	0	b			
SUB A, s	$A \leftarrow A - s$	b	b	b	b	b	V	1	b			
SBC A, s	$A \leftarrow A - s - CY$	b	b	b	b	b	V	1	b			
AND s	$A \leftarrow A \text{ AND } s$	b	b	b	1	b	P	0	0			
OR s	$A \leftarrow A \text{ OR } s$	b	b	b	0	b	P	0	0			
XOR s	$A \leftarrow A \text{ XOR } s$	b	b	b	0	b	P	0	0			
CP s	$A - s$	b	b	b ¹	b	b ¹	V	1	b			
INC r	$r \leftarrow r + 1$	b	b	b	b	b	V	0	•		1	4
INC p*	$p \leftarrow p + 1$	b	b	b	b	b	V	0	•	DD	2	8
INC q*	$q \leftarrow q + 1$	b	b	b	b	b	V	0	•	FD	2	8
INC (HL)	$(HL) \leftarrow (HL) + 1$	b	b	b	b	b	V	0	•		3	11
INC (IX + d)	$(IX + d) \leftarrow (IX + d) + 1$	b	b	b	b	b	V	0	•	DD	6	23
INC (IY + d)	$(IY + d) \leftarrow (IY + d) + 1$	b	b	b	b	b	V	0	•	FD	6	23
DEC m	$m \leftarrow m - 1$	b	b	b	b	b	V	1	•			

Grupo Aritmético y Lógico de 16 bits

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.	
		S	Z	F5	H	F3	P/V	N	C				
ADD HL, ss	$HL \leftarrow HL + ss$	•	•	b	b	b	b	•	0	b		3	11
ADC HL, ss	$HL \leftarrow HL + ss + CY$	b	b	b	b	b	V	0	b		ED	4	15
SBC HL, ss	$HL \leftarrow HL - ss - CY$	b	b	b	b	b	V	1	b		ED	4	15
ADD IX, pp	$IX \leftarrow IX + pp$	•	•	b	b	b	•	0	b		DD	4	15
ADD IY, rr	$IY \leftarrow IY + rr$	•	•	b	b	b	•	0	b		FD	4	15
INC ss	$ss \leftarrow ss + 1$	•	•	•	•	•	•	•	•			1	6
INC IX	$IX \leftarrow IX + 1$	•	•	•	•	•	•	•	•		DD 23	2	10
INC IY	$IY \leftarrow IY + 1$	•	•	•	•	•	•	•	•		FD 23	2	10
DEC ss	$ss \leftarrow ss - 1$	•	•	•	•	•	•	•	•			1	6
DEC IX	$IX \leftarrow IX - 1$	•	•	•	•	•	•	•	•		DD 2B	2	10
DEC IY	$IY \leftarrow IY - 1$	•	•	•	•	•	•	•	•		FD 2B	2	10


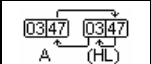
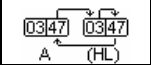
Grupo Aritmético de Propósito General y Control de la CPU

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.	
		S	Z	F5	H	F3	P/V	N	C				
DAA	Ajusta el acumulador a modo decimal	b	b	b	b	b	P	•	b		27	1	4
CPL	$A \leftarrow A$	•	•	b	1	b	•	1	•		2F	1	4
NEG	$A \leftarrow 0 - A$	b	b	b	b	b	V	1	b		ED 44	2	8
CCF	$CY \leftarrow CY$	•	•	b	b	b	•	0	b		3F	1	4
SCF	$CY \leftarrow 1$	•	•	b	0	b	•	0	1		37	1	4
NOP	No operación	•	•	•	•	•	•	•	•		00	1	4
HALT	CPU detenida	•	•	•	•	•	•	•	•		76	1	4

DI	IFF ₁ ← 0 IFF ₂ ← 0	F3	1	4
EI	IFF ₁ ← 1 IFF ₂ ← 1	FB	1	4
IM 0	Modo de interrupción 0	ED 46	2	8
IM 1	Modo de interrupción 1	ED 56	2	8
IM 2	Modo de interrupción 2	ED 5E	2	8

Grupo de Rotaciones y Desplazamientos

Nemotécnico	Operación Simbólica	Señalizadores							Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N			
RLCA	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow$. .	b	0	b	. 0	b	07	1	4	
RLA	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow$. .	b	0	b	. 0	b	17	1	4	
RRCA	$\overline{7} \rightarrow 0 \rightarrow \overline{CY}$. .	b	0	b	. 0	b	0F	1	4	
RRA	$\overline{7} \rightarrow 0 \rightarrow \overline{CY}$. .	b	0	b	. 0	b	1F	1	4	
RLC r	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow$	b b	b	0	b	P 0	b	CB	2	8	
RLC (HL)	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow$	b b	b	0	b	P 0	b	CB	4	15	
RLC (IX + d)	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow$	b b	b	0	b	P 0	b	DD CB	6	23	
RLC (IY + d)	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow$	b b	b	0	b	P 0	b	FD CB	6	23	
LD r, RLC (IX + d)*	r ← (IX + d) RLC r (IX + d) ← r	b b	b	0	b	P 0	b	DD CB	6	23	
LD r, RLC (IY + d)*	r ← (IY + d) RLC r (IY + d) ← r	b b	b	0	b	P 0	b	FD CB	6	23	
RL m	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow$	b b	b	0	b	P 0	b				
RRC m	$\overline{7} \rightarrow 0 \rightarrow \overline{CY}$	b b	b	0	b	P 0	b				
RR m	$\overline{7} \rightarrow 0 \rightarrow \overline{CY}$	b b	b	0	b	P 0	b				
SLA m	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow 0$	b b	b	0	b	P 0	b				
SLL m*	$\overline{CY} \leftarrow \overline{7} \leftarrow 0 \leftarrow 1$	b b	b	0	b	P 0	b				

SRA m		b b b 0 b P 0 b			
SRL m		b b b 0 b P 0 b			
RLD		b b b 0 b P 0 .	ED 6F	5	18
RRD		b b b 0 b P 0 .	ED 67	5	18

Grupo de Manipulación de bits

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N	C			
BIT b, r	$Z \leftarrow \overline{r_b}$	b	b	b	1	b	b	0	.	CB	2	8
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	b	b	b	1	b	b	0	.	CB	3	12
BIT b, (IX + d)	$Z \leftarrow \overline{(IX + d)_b}$	b	b	b	1	b	b	0	.	DD CB	5	20
BIT b, (IY + d)	$Z \leftarrow \overline{(IY + d)_b}$	b	b	b	1	b	b	0	.	FD CB	5	20
SET b, r	$r_b \leftarrow 1$	CB	2	8
SET b, (HL)	$(HL)_b \leftarrow 1$	CB	4	15
SET b, (IX + d)	$(IX + d)_b \leftarrow 1$	DD CB	6	23
SET b, (IY + d)	$(IY + d)_b \leftarrow 1$	FD CB	6	23
LD r, SET b, (IX + d)*	$r \leftarrow (IX + d)$ $r_b \leftarrow 1$ $(IX + d) \leftarrow r$	DD CB	6	23
LD r, SET b, (IY + d)*	$r \leftarrow (IY + d)$ $r_b \leftarrow 1$ $(IY + d) \leftarrow r$	FD CB	6	23
RES b, m	$m_b \leftarrow 0$ $m \equiv r, (HL), (IX+d), (IY+d)$			

Grupos de Entrada y Salida

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N	C			
IN A, (n)	$A \leftarrow (n)$	DB	3	11
IN r, (C)	$r \leftarrow (C)$	b	b	b	0	b	P	0	.	ED	3	12
IN (C)* ó IN F, (C)*	Sólo afecta a los flags.	b	b	b	0	b	P	0	.	ED 70	3	12
INI	$(HL) \leftarrow (C)$ $HL \leftarrow HL + 1$ $B \leftarrow B - 1$	b	b	b	b	b	X	b	b	ED A2	4	16
INIR	$(HL) \leftarrow (C)$ $HL \leftarrow HL + 1$ $B \leftarrow B - 1$ Repite hasta $B = 0$	0	1	0	b	0	X	b	b	ED B2	5 4	21 16
IND	$(HL) \leftarrow (C)$ $HL \leftarrow HL - 1$ $B \leftarrow B - 1$	b	b	b	b	b	X	b	b	ED AA	4	16
INDR	$(HL) \leftarrow (C)$ $HL \leftarrow HL - 1$ $B \leftarrow B - 1$ Repite hasta $B = 0$	0	1	0	b	0	X	b	b	ED BA	5 4	21 16
OUT (n), A	$(n) \leftarrow A$	D3	3	11
OUT (C), r	$(C) \leftarrow r$	ED	3	12
OUT (C), 0*	$(C) \leftarrow 0$	ED 71	3	12
OUTI	$(C) \leftarrow (HL)$ $HL \leftarrow HL + 1$ $B \leftarrow B - 1$	b	b	b	X	b	X	X	X	ED A3	4	16
OTIR	$(C) \leftarrow (HL)$ $HL \leftarrow HL + 1$ $B \leftarrow B - 1$ Repite hasta $B = 0$	0	1	0	X	0	X	X	X	ED B3	5 4	21 16
OUTD	$(C) \leftarrow (HL)$ $HL \leftarrow HL - 1$ $B \leftarrow B - 1$	b	b	b	X	b	X	X	X	ED AB	4	16
OTDR	$(C) \leftarrow (HL)$ $HL \leftarrow HL - 1$ $B \leftarrow B - 1$ Repite hasta $B = 0$	0	1	0	X	0	X	X	X	ED BB	5 4	21 16

Grupo de Call y Return (llamada y retorno)

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N	C			
CALL nn	$SP \leftarrow SP - 1$ $(SP) \leftarrow PC_H$ $SP \leftarrow SP - 1$ $(SP) \leftarrow PC_L$ $PC \leftarrow nn$	CD	5	17
CALL cc, nn	si cc es verdadero, $SP \leftarrow SP - 1$ $(SP) \leftarrow PC_H$ $SP \leftarrow SP - 1$ $(SP) \leftarrow PC_L$ $PC \leftarrow nn$		3 5	10 17
RET	$PC_L \leftarrow (SP)$ $SP \leftarrow SP + 1$ $PC_H \leftarrow (SP)$ $SP \leftarrow SP + 1$	C9	3	10
RET cc	si cc es verdadero, $PC_L \leftarrow (SP)$ $SP \leftarrow SP + 1$ $PC_H \leftarrow (SP)$ $SP \leftarrow SP + 1$		1 3	5 11
RETI	$PC_L \leftarrow (SP)$ $SP \leftarrow SP + 1$ $PC_H \leftarrow (SP)$ $SP \leftarrow SP + 1$	ED 4D	4	14
RETN	$PC_L \leftarrow (SP)$ $SP \leftarrow SP + 1$ $PC_H \leftarrow (SP)$ $SP \leftarrow SP + 1$ $IFF_1 \leftarrow IFF_2$	ED 45	4	14
RST p	$SP \leftarrow SP - 1$ $(SP) \leftarrow PC_H$ $SP \leftarrow SP - 1$ $(SP) \leftarrow PC_L$ $PC \leftarrow p$		3	11

Grupo de Salto

Nemotécnico	Operación Simbólica	Señalizadores								Código Objeto	Ciclos C.M.	Ts.
		S	Z	F5	H	F3	P/V	N	C			
JP nn	PC ← nn	•	•	•	•	•	•	•	•	C3	3	10
JP cc, nn	si cc es verdadero, PC ← nn	•	•	•	•	•	•	•	•		3	10
JR e	PC ← PC + e	•	•	•	•	•	•	•	•	18	3	12
JR ss, e	si ss es verdadero, PC ← PC + e	•	•	•	•	•	•	•	•		3 2	12 7
JP HL	PC ← HL	•	•	•	•	•	•	•	•	E9	1	4
JP IX	PC ← IX	•	•	•	•	•	•	•	•	DD E9	2	8
JP IY	PC ← IY	•	•	•	•	•	•	•	•	FD E9	2	8
DJNZ e	B ← B - 1 si B ≠ 0 PC ← PC + e	•	•	•	•	•	•	•	•	10	2 3	8 13

Instrucciones Ordenadas por Código de Operación

Si una instrucción EDxx no se encuentra en la lista, debería operar como dos NOPs.

Si alguna instrucción DDxx o FDxx no se encuentra en la lista, debería funcionar igual que sin los prefijos DD o FD.

Un asterisco (*) tras una instrucción indica que se trata de una instrucción no oficial.

00	NOP	1E n	LD E, n
01 n n	LD BC, nn	1F	RRA
02	LD (BC), A	20 n	JR NZ, PC + n
03	INC BC	21 n n	LD HL, nn
04	INC B	22 n n	LD (nn), HL
05	DEC B	23	INC HL
06 n	LD B, n	24	INC H
07	RLCA	25	DEC H
08	EX AF, AF'	26 n	LD H, n
09	ADD HL, BC	27	DAA
0A	LD A, (BC)	28 n	JR Z, PC + n
0B	DEC BC	29	ADD HL, HL
0C	INC C	2A n n	LD HL, (nn)
0D	DEC C	2B	DEC HL
0E n	LD C, n	2C	INC L
0F	RRCA	2D	DEC L
10 n	DJNZ PC + n	2E n	LD L, n
11 n n	LD DE, nn	2F	CPL
12	LD (DE), A	30 n	JR NC, PC + n
13	INC DE	31 n n	LD SP, nn
14	INC D	32 n n	LD (nn), A
15	DEC D	33	INC SP
16 n	LD D, n	34	INC (HL)
17	RLA	35	DEC (HL)
18 n	JR PC + n	36 n	LD (HL), n
19	ADD HL, DE	37	SCF
1A	LD A, (DE)	38 n	JR C, PC + n
1B	DEC DE	39	ADD HL, SP
1C	INC E	3A n n	LD A, (nn)
1D	DEC E	3B	DEC SP

3C	INC A	6F	LD L, A
3D	DEC A	70	LD (HL), B
3E n	LD A, n	71	LD (HL), C
3F	CCF	72	LD (HL), D
40	LD B, B	73	LD (HL), E
41	LD B, C	74	LD (HL), H
42	LD B, D	75	LD (HL), L
43	LD B, E	76	HALT
44	LD B, H	77	LD (HL), A
45	LD B, L	78	LD A, B
46	LD B, (HL)	79	LD A, C
47	LD B, A	7A	LD A, D
48	LD C, B	7B	LD A, E
49	LD C, C	7C	LD A, H
4A	LD C, D	7D	LD A, L
4B	LD C, E	7E	LD A, (HL)
4C	LD C, H	7F	LD A, A
4D	LD C, L	80	ADD A, B
4E	LD C, (HL)	81	ADD A, C
4F	LD C, A	82	ADD A, D
50	LD D, B	83	ADD A, E
51	LD D, C	84	ADD A, H
52	LD D, D	85	ADD A, L
53	LD D, E	86	ADD A, (HL)
54	LD D, H	87	ADD A, A
55	LD D, L	88	ADC A, B
56	LD D, (HL)	89	ADC A, C
57	LD D, A	8A	ADC A, D
58	LD E, B	8B	ADC A, E
59	LD E, C	8C	ADC A, H
5A	LD E, D	8D	ADC A, L
5B	LD E, E	8E	ADC A, (HL)
5C	LD E, H	8F	ADC A, A
5D	LD E, L	90	SUB B
5E	LD E, (HL)	91	SUB C
5F	LD E, A	92	SUB D
60	LD H, B	93	SUB E
61	LD H, C	94	SUB H
62	LD H, D	95	SUB L
63	LD H, E	96	SUB (HL)
64	LD H, H	97	SUB A
65	LD H, L	98	SBC A, B
66	LD H, (HL)	99	SBC A, C
67	LD H, A	9A	SBC A, D
68	LD L, B	9B	SBC A, E
69	LD L, C	9C	SBC A, H
6A	LD L, D	9D	SBC A, L
6B	LD L, E	9E	SBC A, (HL)
6C	LD L, H	9F	SBC A, A
6D	LD L, L	A0	AND B
6E	LD L, (HL)	A1	AND C

A2	AND D	CB0A	RRC D
A3	AND E	CB0B	RRC E
A4	AND H	CB0C	RRC H
A5	AND L	CB0D	RRC L
A6	AND (HL)	CB0E	RRC (HL)
A7	AND A	CB0F	RRC A
A8	XOR B	CB10	RL B
A9	XOR C	CB11	RL C
AA	XOR D	CB12	RL D
AB	XOR E	CB13	RL E
AC	XOR H	CB14	RL H
AD	XOR L	CB15	RL L
AE	XOR (HL)	CB16	RL (HL)
AF	XOR A	CB17	RL A
B0	OR B	CB18	RR B
B1	OR C	CB19	RR C
B2	OR D	CB1A	RR D
B3	OR E	CB1B	RR E
B4	OR H	CB1C	RR H
B5	OR L	CB1D	RR L
B6	OR (HL)	CB1E	RR (HL)
B7	OR A	CB1F	RR A
B8	CP B	CB20	SLA B
B9	CP C	CB21	SLA C
BA	CP D	CB22	SLA D
BB	CP E	CB23	SLA E
BC	CP H	CB24	SLA H
BD	CP L	CB25	SLA L
BE	CP (HL)	CB26	SLA (HL)
BF	CP A	CB27	SLA A
C0	RET NZ	CB28	SRA B
C1	POP BC	CB29	SRA C
C2 n n	JP NZ, nn	CB2A	SRA D
C3 n n	JP nn	CB2B	SRA E
C4 n n	CALL NZ, nn	CB2C	SRA H
C5	PUSH BC	CB2D	SRA L
C6 n	ADD A, n	CB2E	SRA (HL)
C7	RST 0h	CB2F	SRA A
C8	RET Z	CB30	SLL B*
C9	RET	CB31	SLL C*
CA n n	JP Z, nn	CB32	SLL D*
CB00	RLC B	CB33	SLL E*
CB01	RLC C	CB34	SLL H*
CB02	RLC D	CB35	SLL L*
CB03	RLC E	CB36	SLL (HL)*
CB04	RLC H	CB37	SLL A*
CB05	RLC L	CB38	SRL B
CB06	RLC (HL)	CB39	SRL C
CB07	RLC A	CB3A	SRL D
CB08	RRC B	CB3B	SRL E
CB09	RRC C	CB3C	SRL H

CB3D	SRL L	CB70	BIT 6, B
CB3E	SRL (HL)	CB71	BIT 6, C
CB3F	SRL A	CB72	BIT 6, D
CB40	BIT 0, B	CB73	BIT 6, E
CB41	BIT 0, C	CB74	BIT 6, H
CB42	BIT 0, D	CB75	BIT 6, L
CB43	BIT 0, E	CB76	BIT 6, (HL)
CB44	BIT 0, H	CB77	BIT 6, A
CB45	BIT 0, L	CB78	BIT 7, B
CB46	BIT 0, (HL)	CB79	BIT 7, C
CB47	BIT 0, A	CB7A	BIT 7, D
CB48	BIT 1, B	CB7B	BIT 7, E
CB49	BIT 1, C	CB7C	BIT 7, H
CB4A	BIT 1, D	CB7D	BIT 7, L
CB4B	BIT 1, E	CB7E	BIT 7, (HL)
CB4C	BIT 1, H	CB7F	BIT 7, A
CB4D	BIT 1, L	CB80	RES 0, B
CB4E	BIT 1, (HL)	CB81	RES 0, C
CB4F	BIT 1, A	CB82	RES 0, D
CB50	BIT 2, B	CB83	RES 0, E
CB51	BIT 2, C	CB84	RES 0, H
CB52	BIT 2, D	CB85	RES 0, L
CB53	BIT 2, E	CB86	RES 0, (HL)
CB54	BIT 2, H	CB87	RES 0, A
CB55	BIT 2, L	CB88	RES 1, B
CB56	BIT 2, (HL)	CB89	RES 1, C
CB57	BIT 2, A	CB8A	RES 1, D
CB58	BIT 3, B	CB8B	RES 1, E
CB59	BIT 3, C	CB8C	RES 1, H
CB5A	BIT 3, D	CB8D	RES 1, L
CB5B	BIT 3, E	CB8E	RES 1, (HL)
CB5C	BIT 3, H	CB8F	RES 1, A
CB5D	BIT 3, L	CB90	RES 2, B
CB5E	BIT 3, (HL)	CB91	RES 2, C
CB5F	BIT 3, A	CB92	RES 2, D
CB60	BIT 4, B	CB93	RES 2, E
CB61	BIT 4, C	CB94	RES 2, H
CB62	BIT 4, D	CB95	RES 2, L
CB63	BIT 4, E	CB96	RES 2, (HL)
CB64	BIT 4, H	CB97	RES 2, A
CB65	BIT 4, L	CB98	RES 3, B
CB66	BIT 4, (HL)	CB99	RES 3, C
CB67	BIT 4, A	CB9A	RES 3, D
CB68	BIT 5, B	CB9B	RES 3, E
CB69	BIT 5, C	CB9C	RES 3, H
CB6A	BIT 5, D	CB9D	RES 3, L
CB6B	BIT 5, E	CB9E	RES 3, (HL)
CB6C	BIT 5, H	CB9F	RES 3, A
CB6D	BIT 5, L	CBA0	RES 4, B
CB6E	BIT 5, (HL)	CBA1	RES 4, C
CB6F	BIT 5, A	CBA2	RES 4, D

CBA3 RES 4, E
 CBA4 RES 4, H
 CBA5 RES 4, L
 CBA6 RES 4, (HL)
 CBA7 RES 4, A
 CBA8 RES 5, B
 CBA9 RES 5, C
 CBAA RES 5, D
 CBAB RES 5, E
 CBAC RES 5, H
 CBAD RES 5, L
 CBAE RES 5, (HL)
 CBAF RES 5, A
 CBB0 RES 6, B
 CBB1 RES 6, C
 CBB2 RES 6, D
 CBB3 RES 6, E
 CBB4 RES 6, H
 CBB5 RES 6, L
 CBB6 RES 6, (HL)
 CBB7 RES 6, A
 CBB8 RES 7, B
 CBB9 RES 7, C
 CBBA RES 7, D
 CBBB RES 7, E
 CBBC RES 7, H
 CBBD RES 7, L
 CBBE RES 7, (HL)
 CBBF RES 7, A
 CBC0 SET 0, B
 CBC1 SET 0, C
 CBC2 SET 0, D
 CBC3 SET 0, E
 CBC4 SET 0, H
 CBC5 SET 0, L
 CBC6 SET 0, (HL)
 CBC7 SET 0, A
 CBC8 SET 1, B
 CBC9 SET 1, C
 CBCA SET 1, D
 CBCB SET 1, E
 CBCC SET 1, H
 CBCD SET 1, L
 CBCE SET 1, (HL)
 CBCF SET 1, A
 CBD0 SET 2, B
 CBD1 SET 2, C
 CBD2 SET 2, D
 CBD3 SET 2, E
 CBD4 SET 2, H
 CBD5 SET 2, L

CBD6 SET 2, (HL)
 CBD7 SET 2, A
 CBD8 SET 3, B
 CBD9 SET 3, C
 CBDA SET 3, D
 CBDB SET 3, E
 CBDC SET 3, H
 CBDD SET 3, L
 CBDE SET 3, (HL)
 CBDF SET 3, A
 CBE0 SET 4, B
 CBE1 SET 4, C
 CBE2 SET 4, D
 CBE3 SET 4, E
 CBE4 SET 4, H
 CBE5 SET 4, L
 CBE6 SET 4, (HL)
 CBE7 SET 4, A
 CBE8 SET 5, B
 CBE9 SET 5, C
 CBEA SET 5, D
 CBEB SET 5, E
 CBEC SET 5, H
 CBED SET 5, L
 CBEE SET 5, (HL)
 CBEF SET 5, A
 CBF0 SET 6, B
 CBF1 SET 6, C
 CBF2 SET 6, D
 CBF3 SET 6, E
 CBF4 SET 6, H
 CBF5 SET 6, L
 CBF6 SET 6, (HL)
 CBF7 SET 6, A
 CBF8 SET 7, B
 CBF9 SET 7, C
 CBFA SET 7, D
 CBFB SET 7, E
 CBFC SET 7, H
 CBFD SET 7, L
 CBFE SET 7, (HL)
 CBFF SET 7, A
 CC n n CALL Z, nn
 CD n n CALL nn
 CE n ADC A, n
 CF RST 8h
 D0 RET NC
 D1 POP DE
 D2 n n JP NC, nn
 D3 n OUT (n), A
 D4 n n CALL NC, nn

DD5	PUSH DE	DD6D	LD IX _L , IX _L *
DD6 n	SUB n	DD6E d	LD L, (IX + d)
DD7	RST 10h	DD6F	LD IX _L , A*
DD8	RET	DD70 d	LD (IX + d), B
DD9	EXX	DD71 d	LD (IX + d), C
DDA n n	JP C, nn	DD72 d	LD (IX + d), D
DDB n	IN A, (n)	DD73 d	LD (IX + d), E
DDC n n	CALL C, nn	DD74 d	LD (IX + d), H
DD09	ADD IX, BC	DD75 d	LD (IX + d), L
DD19	ADD IX, DE	DD77 d	LD (IX + d), A
DD21 n n	LD IX, nn	DD7C	LD A, IX _H *
DD22 n n	LD (nn), IX	DD7D	LD A, IX _L *
DD23	INC IX	DD7E d	LD A, (IX + d)
DD24	INC IX _H *	DD84	ADD A, IX _H *
DD25	DEC IX _H *	DD85	ADD A, IX _L *
DD26 n	LD IX _H , n*	DD86 d	ADD A, (IX + d)
DD29	ADD IX, IX	DD8C	ADC A, IX _H *
DD2A n n	LD IX, (nn)	DD8D	ADC A, IX _L *
DD2B	DEC IX	DD8E d	ADC A, (IX + d)
DD2C	INC IX _L *	DD94	SUB IX _H *
DD2D	DEC IX _L *	DD95	SUB IX _L *
DD2E n	LD IX _L , n*	DD96 d	SUB (IX + d)
DD34 d	INC (IX + d)	DD9C	SBC A, IX _H *
DD35 d	DEC (IX + d)	DD9D	SBC A, IX _L *
DD36 d n	LD (IX + d), n	DD9E d	SBC A, (IX + d)
DD39	ADD IX, SP	DDA4	AND IX _H *
DD44	LD B, IX _H *	DDA5	AND IX _L *
DD45	LD B, IX _L *	DDA6 d	AND (IX + d)
DD46 d	LD B, (IX + d)	DDAC	XOR IX _H *
DD4C	LD C, IX _H *	DDAD	XOR IX _L *
DD4D	LD C, IX _L *	DDAE d	XOR (IX + d)
DD4E d	LD C, (IX + d)	DDB4	OR IX _H *
DD54	LD D, IX _H *	DDB5	OR IX _L *
DD55	LD D, IX _L *	DDB6 d	OR (IX + d)
DD56 d	LD D, (IX + d)	DDBC	CP IX _H *
DD5C	LD E, IX _H *	DDBD	CP IX _L *
DD5D	LD E, IX _L *	DDBE d	CP (IX + d)
DD5E d	LD E, (IX + d)	DDCB d 00	LD B, RLC (IX + d)*
DD60	LD IX _H , B*	DDCB d 01	LD C, RLC (IX + d)*
DD61	LD IX _H , C*	DDCB d 02	LD D, RLC (IX + d)*
DD62	LD IX _H , D*	DDCB d 03	LD E, RLC (IX + d)*
DD63	LD IX _H , E*	DDCB d 04	LD H, RLC (IX + d)*
DD64	LD IX _H , IX _H *	DDCB d 05	LD L, RLC (IX + d)*
DD65	LD IX _H , IX _L *	DDCB d 06	RLC (IX + d)
DD66 d	LD H, (IX + d)	DDCB d 07	LD A, RLC (IX + d)*
DD67	LD IX _H , A*	DDCB d 08	LD B, RRC (IX + d)*
DD68	LD IX _L , B*	DDCB d 09	LD C, RRC (IX + d)*
DD69	LD IX _L , C*	DDCB d 0A	LD D, RRC (IX + d)*
DD6A	LD IX _L , D*	DDCB d 0B	LD E, RRC (IX + d)*
DD6B	LD IX _L , E*	DDCB d 0C	LD H, RRC (IX + d)*
DD6C	LD IX _L , IX _H *	DDCB d 0D	LD L, RRC (IX + d)*

DDCB d 0E RRC (IX + d)
 DDCB d 0F LD A, RRC (IX + d)*
 DDCB d 10 LD B, RL (IX + d)*
 DDCB d 11 LD C, RL (IX + d)*
 DDCB d 12 LD D, RL (IX + d)*
 DDCB d 13 LD E, RL (IX + d)*
 DDCB d 14 LD H, RL (IX + d)*
 DDCB d 15 LD L, RL (IX + d)*
 DDCB d 16 RL (IX + d)
 DDCB d 17 LD A, RL (IX + d)*
 DDCB d 18 LD B, RR (IX + d)*
 DDCB d 19 LD C, RR (IX + d)*
 DDCB d 1A LD D, RR (IX + d)*
 DDCB d 1B LD E, RR (IX + d)*
 DDCB d 1C LD H, RR (IX + d)*
 DDCB d 1D LD L, RR (IX + d)*
 DDCB d 1E RR (IX + d)
 DDCB d 1F LD A, RR (IX + d)*
 DDCB d 20 LD B, SLA (IX + d)*
 DDCB d 21 LD C, SLA (IX + d)*
 DDCB d 22 LD D, SLA (IX + d)*
 DDCB d 23 LD E, SLA (IX + d)*
 DDCB d 24 LD H, SLA (IX + d)*
 DDCB d 25 LD L, SLA (IX + d)*
 DDCB d 26 SLA (IX + d)
 DDCB d 27 LD A, SLA (IX + d)*
 DDCB d 28 LD B, SRA (IX + d)*
 DDCB d 29 LD C, SRA (IX + d)*
 DDCB d 2A LD D, SRA (IX + d)*
 DDCB d 2B LD E, SRA (IX + d)*
 DDCB d 2C LD H, SRA (IX + d)*
 DDCB d 2D LD L, SRA (IX + d)*
 DDCB d 2E SRA (IX + d)
 DDCB d 2F LD A, SRA (IX + d)*
 DDCB d 30 LD B, SLL (IX + d)*
 DDCB d 31 LD C, SLL (IX + d)*
 DDCB d 32 LD D, SLL (IX + d)*
 DDCB d 33 LD E, SLL (IX + d)*
 DDCB d 34 LD H, SLL (IX + d)*
 DDCB d 35 LD L, SLL (IX + d)*
 DDCB d 36 SLL (IX + d)*
 DDCB d 37 LD A, SLL (IX + d)*
 DDCB d 38 LD B, SRL (IX + d)*
 DDCB d 39 LD C, SRL (IX + d)*
 DDCB d 3A LD D, SRL (IX + d)*
 DDCB d 3B LD E, SRL (IX + d)*
 DDCB d 3C LD H, SRL (IX + d)*
 DDCB d 3D LD L, SRL (IX + d)*
 DDCB d 3E SRL (IX + d)
 DDCB d 3F LD A, SRL (IX + d)*
 DDCB d 40 BIT 0, (IX + d)*

DDCB d 41 BIT 0, (IX + d)*
 DDCB d 42 BIT 0, (IX + d)*
 DDCB d 43 BIT 0, (IX + d)*
 DDCB d 44 BIT 0, (IX + d)*
 DDCB d 45 BIT 0, (IX + d)*
 DDCB d 46 BIT 0, (IX + d)
 DDCB d 47 BIT 0, (IX + d)*
 DDCB d 48 BIT 1, (IX + d)*
 DDCB d 49 BIT 1, (IX + d)*
 DDCB d 4A BIT 1, (IX + d)*
 DDCB d 4B BIT 1, (IX + d)*
 DDCB d 4C BIT 1, (IX + d)*
 DDCB d 4D BIT 1, (IX + d)*
 DDCB d 4E BIT 1, (IX + d)
 DDCB d 4F BIT 1, (IX + d)*
 DDCB d 50 BIT 2, (IX + d)*
 DDCB d 51 BIT 2, (IX + d)*
 DDCB d 52 BIT 2, (IX + d)*
 DDCB d 53 BIT 2, (IX + d)*
 DDCB d 54 BIT 2, (IX + d)*
 DDCB d 55 BIT 2, (IX + d)*
 DDCB d 56 BIT 2, (IX + d)
 DDCB d 57 BIT 2, (IX + d)*
 DDCB d 58 BIT 3, (IX + d)*
 DDCB d 59 BIT 3, (IX + d)*
 DDCB d 5A BIT 3, (IX + d)*
 DDCB d 5B BIT 3, (IX + d)*
 DDCB d 5C BIT 3, (IX + d)*
 DDCB d 5D BIT 3, (IX + d)*
 DDCB d 5E BIT 3, (IX + d)
 DDCB d 5F BIT 3, (IX + d)*
 DDCB d 60 BIT 4, (IX + d)*
 DDCB d 61 BIT 4, (IX + d)*
 DDCB d 62 BIT 4, (IX + d)*
 DDCB d 63 BIT 4, (IX + d)*
 DDCB d 64 BIT 4, (IX + d)*
 DDCB d 65 BIT 4, (IX + d)*
 DDCB d 66 BIT 4, (IX + d)
 DDCB d 67 BIT 4, (IX + d)*
 DDCB d 68 BIT 5, (IX + d)*
 DDCB d 69 BIT 5, (IX + d)*
 DDCB d 6A BIT 5, (IX + d)*
 DDCB d 6B BIT 5, (IX + d)*
 DDCB d 6C BIT 5, (IX + d)*
 DDCB d 6D BIT 5, (IX + d)*
 DDCB d 6E BIT 5, (IX + d)
 DDCB d 6F BIT 5, (IX + d)*
 DDCB d 70 BIT 6, (IX + d)*
 DDCB d 71 BIT 6, (IX + d)*
 DDCB d 72 BIT 6, (IX + d)*
 DDCB d 73 BIT 6, (IX + d)*

DDCB d 74 BIT 6, (IX + d)*
 DDCB d 75 BIT 6, (IX + d)*
 DDCB d 76 BIT 6, (IX + d)*
 DDCB d 77 BIT 6, (IX + d)*
 DDCB d 78 BIT 7, (IX + d)*
 DDCB d 79 BIT 7, (IX + d)*
 DDCB d 7A BIT 7, (IX + d)*
 DDCB d 7B BIT 7, (IX + d)*
 DDCB d 7C BIT 7, (IX + d)*
 DDCB d 7D BIT 7, (IX + d)*
 DDCB d 7E BIT 7, (IX + d)*
 DDCB d 7F BIT 7, (IX + d)*
 DDCB d 80 LD B, RES 0, (IX + d)*
 DDCB d 81 LD C, RES 0, (IX + d)*
 DDCB d 82 LD D, RES 0, (IX + d)*
 DDCB d 83 LD E, RES 0, (IX + d)*
 DDCB d 84 LD H, RES 0, (IX + d)*
 DDCB d 85 LD L, RES 0, (IX + d)*
 DDCB d 86 RES 0, (IX + d)
 DDCB d 87 LD A, RES 0, (IX + d)*
 DDCB d 88 LD B, RES 1, (IX + d)*
 DDCB d 89 LD C, RES 1, (IX + d)*
 DDCB d 8A LD D, RES 1, (IX + d)*
 DDCB d 8B LD E, RES 1, (IX + d)*
 DDCB d 8C LD H, RES 1, (IX + d)*
 DDCB d 8D LD L, RES 1, (IX + d)*
 DDCB d 8E RES 1, (IX + d)
 DDCB d 8F LD A, RES 1, (IX + d)*
 DDCB d 90 LD B, RES 2, (IX + d)*
 DDCB d 91 LD C, RES 2, (IX + d)*
 DDCB d 92 LD D, RES 2, (IX + d)*
 DDCB d 93 LD E, RES 2, (IX + d)*
 DDCB d 94 LD H, RES 2, (IX + d)*
 DDCB d 95 LD L, RES 2, (IX + d)*
 DDCB d 96 RES 2, (IX + d)
 DDCB d 97 LD A, RES 2, (IX + d)*
 DDCB d 98 LD B, RES 3, (IX + d)*
 DDCB d 99 LD C, RES 3, (IX + d)*
 DDCB d 9A LD D, RES 3, (IX + d)*
 DDCB d 9B LD E, RES 3, (IX + d)*
 DDCB d 9C LD H, RES 3, (IX + d)*
 DDCB d 9D LD L, RES 3, (IX + d)*
 DDCB d 9E RES 3, (IX + d)
 DDCB d 9F LD A, RES 3, (IX + d)*
 DDCB d A0 LD B, RES 4, (IX + d)*
 DDCB d A1 LD C, RES 4, (IX + d)*
 DDCB d A2 LD D, RES 4, (IX + d)*
 DDCB d A3 LD E, RES 4, (IX + d)*
 DDCB d A4 LD H, RES 4, (IX + d)*
 DDCB d A5 LD L, RES 4, (IX + d)*
 DDCB d A6 RES 4, (IX + d)

DDCB d A7 LD A, RES 4, (IX + d)*
 DDCB d A8 LD B, RES 5, (IX + d)*
 DDCB d A9 LD C, RES 5, (IX + d)*
 DDCB d AA LD D, RES 5, (IX + d)*
 DDCB d AB LD E, RES 5, (IX + d)*
 DDCB d AC LD H, RES 5, (IX + d)*
 DDCB d AD LD L, RES 5, (IX + d)*
 DDCB d AE RES 5, (IX + d)
 DDCB d AF LD A, RES 5, (IX + d)*
 DDCB d B0 LD B, RES 6, (IX + d)*
 DDCB d B1 LD C, RES 6, (IX + d)*
 DDCB d B2 LD D, RES 6, (IX + d)*
 DDCB d B3 LD E, RES 6, (IX + d)*
 DDCB d B4 LD H, RES 6, (IX + d)*
 DDCB d B5 LD L, RES 6, (IX + d)*
 DDCB d B6 RES 6, (IX + d)
 DDCB d B7 LD A, RES 6, (IX + d)*
 DDCB d B8 LD B, RES 7, (IX + d)*
 DDCB d B9 LD C, RES 7, (IX + d)*
 DDCB d BA LD D, RES 7, (IX + d)*
 DDCB d BB LD E, RES 7, (IX + d)*
 DDCB d BC LD H, RES 7, (IX + d)*
 DDCB d BD LD L, RES 7, (IX + d)*
 DDCB d BE RES 7, (IX + d)
 DDCB d BF LD A, RES 7, (IX + d)*
 DDCB d C0 LD B, SET 0, (IX + d)*
 DDCB d C1 LD C, SET 0, (IX + d)*
 DDCB d C2 LD D, SET 0, (IX + d)*
 DDCB d C3 LD E, SET 0, (IX + d)*
 DDCB d C4 LD H, SET 0, (IX + d)*
 DDCB d C5 LD L, SET 0, (IX + d)*
 DDCB d C6 SET 0, (IX + d)
 DDCB d C7 LD A, SET 0, (IX + d)*
 DDCB d C8 LD B, SET 1, (IX + d)*
 DDCB d C9 LD C, SET 1, (IX + d)*
 DDCB d CA LD D, SET 1, (IX + d)*
 DDCB d CB LD E, SET 1, (IX + d)*
 DDCB d CC LD H, SET 1, (IX + d)*
 DDCB d CD LD L, SET 1, (IX + d)*
 DDCB d CE SET 1, (IX + d)
 DDCB d CF LD A, SET 1, (IX + d)*
 DDCB d D0 LD B, SET 2, (IX + d)*
 DDCB d D1 LD C, SET 2, (IX + d)*
 DDCB d D2 LD D, SET 2, (IX + d)*
 DDCB d D3 LD E, SET 2, (IX + d)*
 DDCB d D4 LD H, SET 2, (IX + d)*
 DDCB d D5 LD L, SET 2, (IX + d)*
 DDCB d D6 SET 2, (IX + d)
 DDCB d D7 LD A, SET 2, (IX + d)*
 DDCB d D8 LD B, SET 3, (IX + d)*
 DDCB d D9 LD C, SET 3, (IX + d)*

DDCB d DA LD D, SET 3, (IX + d)*
 DDCB d DB LD E, SET 3, (IX + d)*
 DDCB d DC LD H, SET 3, (IX + d)*
 DDCB d DD LD L, SET 3, (IX + d)*
 DDCB d DE SET 3, (IX + d)
 DDCB d DF LD A, SET 3, (IX + d)*
 DDCB d E0 LD B, SET 4, (IX + d)*
 DDCB d E1 LD C, SET 4, (IX + d)*
 DDCB d E2 LD D, SET 4, (IX + d)*
 DDCB d E3 LD E, SET 4, (IX + d)*
 DDCB d E4 LD H, SET 4, (IX + d)*
 DDCB d E5 LD L, SET 4, (IX + d)*
 DDCB d E6 SET 4, (IX + d)
 DDCB d E7 LD A, SET 4, (IX + d)*
 DDCB d E8 LD B, SET 5, (IX + d)*
 DDCB d E9 LD C, SET 5, (IX + d)*
 DDCB d EA LD D, SET 5, (IX + d)*
 DDCB d EB LD E, SET 5, (IX + d)*
 DDCB d EC LD H, SET 5, (IX + d)*
 DDCB d ED LD L, SET 5, (IX + d)*
 DDCB d EE SET 5, (IX + d)
 DDCB d EF LD A, SET 5, (IX + d)*
 DDCB d F0 LD B, SET 6, (IX + d)*
 DDCB d F1 LD C, SET 6, (IX + d)*
 DDCB d F2 LD D, SET 6, (IX + d)*
 DDCB d F3 LD E, SET 6, (IX + d)*
 DDCB d F4 LD H, SET 6, (IX + d)*
 DDCB d F5 LD L, SET 6, (IX + d)*
 DDCB d F6 SET 6, (IX + d)
 DDCB d F7 LD A, SET 6, (IX + d)*
 DDCB d F8 LD B, SET 7, (IX + d)*
 DDCB d F9 LD C, SET 7, (IX + d)*
 DDCB d FA LD D, SET 7, (IX + d)*
 DDCB d FB LD E, SET 7, (IX + d)*
 DDCB d FC LD H, SET 7, (IX + d)*
 DDCB d FD LD L, SET 7, (IX + d)*
 DDCB d FE SET 7, (IX + d)
 DDCB d FF LD A, SET 7, (IX + d)*
 DDE1 POP IX
 DDE3 EX (SP), IX
 DDE5 PUSH IX
 DDE9 JP (IX)
 DDF9 LD SP, IX
 DE n SBC A, n
 DF RST 18h
 E0 RET PO
 E1 POP HL
 E2 n n JP PO, nn
 E3 EX (SP), HL
 E4 n n CALL PO, nn
 E5 PUSH HL

E6 n AND n
 E7 RST 20h
 E8 RET PE
 E9 JP (HL)
 EA n n JP PE, (nn)
 EB EX DE, HL
 EC n n CALL PE, nn
 ED40 IN B, (C)
 ED41 OUT (C), B
 ED42 SBC HL, BC
 ED43 n n LD (nn), BC
 ED44 NEG
 ED45 RETN
 ED46 IM 0
 ED47 LD I, A
 ED48 IN C, (C)
 ED49 OUT (C), C
 ED4A ADC HL, BC
 ED4B n n LD BC, (nn)
 ED4C NEG*
 ED4D RETI
 ED4E IM 0/1*
 ED4F LD R, A
 ED50 IN D, (C)
 ED51 OUT (C), D
 ED52 SBC HL, DE
 ED53 n n LD (nn), DE
 ED54 NEG*
 ED55 RETN*
 ED56 IM 1
 ED57 LD A, I
 ED58 IN E, (C)
 ED59 OUT (C), E
 ED5A ADC HL, DE
 ED5B n n LD DE, (nn)
 ED5C NEG*
 ED5D RETN*
 ED5E IM 2
 ED5F LD A, R
 ED60 IN H, (C)
 ED61 OUT (C), H
 ED62 SBC HL, HL
 ED63 n n LD (nn), HL
 ED64 NEG*
 ED65 RETN*
 ED66 IM 0*
 ED67 RRD
 ED68 IN L, (C)
 ED69 OUT (C), L
 ED6A ADC HL, HL
 ED6B n n LD HL, (nn)

ED6C	NEG*	FD21 n n	LD IY, nn
ED6D	RETN*	FD22 n n	LD (nn), IY
ED6E	IM 0/1*	FD23	INC IY
ED6F	RLD	FD24	INC IY _H *
ED70	IN (C)* / IN F, (C)*	FD25	DEC IY _H *
ED71	OUT (C), 0*	FD26 n	LD IY _H , n*
ED72	SBC HL, SP	FD29	ADD IY, IY
ED73 n n	LD (nn), SP	FD2A n n	LD IY, (nn)
ED74	NEG*	FD2B	DEC IY
ED75	RETN*	FD2C	INC IY _L *
ED76	IM 1*	FD2D	DEC IY _L *
ED78	IN A, (C)	FD2E n	LD IY _L , n*
ED79	OUT (C), A	FD34 d	INC (IY + d)
ED7A	ADC HL, SP	FD35 d	DEC (IY + d)
ED7B n n	LD SP, (nn)	FD36 d n	LD (IY + d), n
ED7C	NEG*	FD39	ADD IY, SP
ED7D	RETN*	FD44	LD B, IY _H *
ED7E	IM 2*	FD45	LD B, IY _L *
EDA0	LDI	FD46 d	LD B, (IY + d)
EDA1	CPI	FD4C	LD C, IY _i *
EDA2	INI	FD4D	LD C, IY _L *
EDA3	OUTI	FD4E d	LD C, (IY + d)
EDA8	LDD	FD54	LD D, IY _H *
EDA9	CPD	FD55	LD D, IY _L *
EDAA	IND	FD56 d	LD D, (IY + d)
EDAB	OUTD	FD5C	LD E, IY _H *
EDB0	LDIR	FD5D	LD E, IY _L *
EDB1	CPIR	FD5E d	LD E, (IY + d)
EDB2	INIR	FD60	LD IY _H , B*
EDB3	OTIR	FD61	LD IY _H , C*
EDB8	LDDR	FD62	LD IY _H , D*
EDB9	CPDR	FD63	LD IY _H , E*
EDBA	INDR	FD64	LD IY _H , IY _H *
EDBB	OTDR	FD65	LD IY _H , IY _L *
EE n	XOR n	FD66 d	LD H, (IY + d)
EF	RST 28h	FD67	LD IY _H , A*
F0	RET P	FD68	LD IY _L , B*
F1	POP AF	FD69	LD IY _L , C*
F2 n n	JP P, nn	FD6A	LD IY _L , D*
F3	DI	FD6B	LD IY _L , E*
F4 n n	CALL P, nn	FD6C	LD IY _L , IY _H *
F5	PUSH AF	FD6D	LD IY _L , IY _L *
F6 n	OR n	FD6E d	LD L, (IY + d)
F7	RST 30h	FD6F	LD IY _L , A*
F8	RET M	FD70 d	LD (IY + d), B
F9	LD SP, HL	FD71 d	LD (IY + d), C
FA n n	JP M, nn	FD72 d	LD (IY + d), D
FB	EI	FD73 d	LD (IY + d), E
FC n n	CALL M, nn	FD74 d	LD (IY + d), H
FD09	ADD IY, BC	FD75 d	LD (IY + d), L
FD19	ADD IY, DE	FD77 d	LD (IY + d), A

FD7C	LD A, IY _H *	FDCB d 18	LD B, RR (IY + d)*
FD7D	LD A, IY _L *	FDCB d 19	LD C, RR (IY + d)*
FD7E d	LD A, (IY + d)	FDCB d 1A	LD D, RR (IY + d)*
FD84	ADD A, IY _H *	FDCB d 1B	LD E, RR (IY + d)*
FD85	ADD A, IY _L *	FDCB d 1C	LD H, RR (IY + d)*
FD86 d	ADD A, (IY + d)	FDCB d 1D	LD L, RR (IY + d)*
FD8C	ADC A, IY _H *	FDCB d 1E	RR (IY + d)
FD8D	ADC A, IY _L *	FDCB d 1F	LD A, RR (IY + d)*
FD8E d	ADC A, (IY + d)	FDCB d 20	LD B, SLA (IY + d)*
FD94	SUB IY _H *	FDCB d 21	LD C, SLA (IY + d)*
FD95	SUB IY _L *	FDCB d 22	LD D, SLA (IY + d)*
FD96 d	SUB (IY + d)	FDCB d 23	LD E, SLA (IY + d)*
FD9C	SBC A, IY _H *	FDCB d 24	LD H, SLA (IY + d)*
FD9D	SBC A, IY _L *	FDCB d 25	LD L, SLA (IY + d)*
FD9E d	SBC A, (IY + d)	FDCB d 26	SLA (IY + d)
FDA4	AND IY _H *	FDCB d 27	LD A, SLA (IY + d)*
FDA5	AND IY _L *	FDCB d 28	LD B, SRA (IY + d)*
FDA6 d	AND (IY + d)	FDCB d 29	LD C, SRA (IY + d)*
FDAC	XOR IY _H *	FDCB d 2A	LD D, SRA (IY + d)*
FDAD	XOR IY _L *	FDCB d 2B	LD E, SRA (IY + d)*
FDAE d	XOR (IY + d)	FDCB d 2C	LD H, SRA (IY + d)*
FDB4	OR IY _H *	FDCB d 2D	LD L, SRA (IY + d)*
FDB5	OR IY _L *	FDCB d 2E	SRA (IY + d)
FDB6 d	OR (IY + d)	FDCB d 2F	LD A, SRA (IY + d)*
FDBC	CP IY _H *	FDCB d 30	LD B, SLL (IY + d)*
FDDB	CP IY _L *	FDCB d 31	LD C, SLL (IY + d)*
FDBE d	CP (IY + d)	FDCB d 32	LD D, SLL (IY + d)*
FDCB d 00	LD B, RLC (IY + d)*	FDCB d 33	LD E, SLL (IY + d)*
FDCB d 01	LD C, RLC (IY + d)*	FDCB d 34	LD H, SLL (IY + d)*
FDCB d 02	LD D, RLC (IY + d)*	FDCB d 35	LD L, SLL (IY + d)*
FDCB d 03	LD E, RLC (IY + d)*	FDCB d 36	SLL (IY + d)*
FDCB d 04	LD H, RLC (IY + d)*	FDCB d 37	LD A, SLL (IY + d)*
FDCB d 05	LD L, RLC (IY + d)*	FDCB d 38	LD B, SRL (IY + d)*
FDCB d 06	RLC (IY + d)	FDCB d 39	LD C, SRL (IY + d)*
FDCB d 07	LD A, RLC (IY + d)*	FDCB d 3A	LD D, SRL (IY + d)*
FDCB d 08	LD B, RRC (IY + d)*	FDCB d 3B	LD E, SRL (IY + d)*
FDCB d 09	LD C, RRC (IY + d)*	FDCB d 3C	LD H, SRL (IY + d)*
FDCB d 0A	LD D, RRC (IY + d)*	FDCB d 3D	LD L, SRL (IY + d)*
FDCB d 0B	LD E, RRC (IY + d)*	FDCB d 3E	SRL (IY + d)
FDCB d 0C	LD H, RRC (IY + d)*	FDCB d 3F	LD A, SRL (IY + d)*
FDCB d 0D	LD L, RRC (IY + d)*	FDCB d 40	BIT 0, (IY + d)*
FDCB d 0E	RRC (IY + d)	FDCB d 41	BIT 0, (IY + d)*
FDCB d 0F	LD A, RRC (IY + d)*	FDCB d 42	BIT 0, (IY + d)*
FDCB d 10	LD B, RL (IY + d)*	FDCB d 43	BIT 0, (IY + d)*
FDCB d 11	LD C, RL (IY + d)*	FDCB d 44	BIT 0, (IY + d)*
FDCB d 12	LD D, RL (IY + d)*	FDCB d 45	BIT 0, (IY + d)*
FDCB d 13	LD E, RL (IY + d)*	FDCB d 46	BIT 0, (IY + d)
FDCB d 14	LD H, RL (IY + d)*	FDCB d 47	BIT 0, (IY + d)*
FDCB d 15	LD L, RL (IY + d)*	FDCB d 48	BIT 1, (IY + d)*
FDCB d 16	RL (IY + d)	FDCB d 49	BIT 1, (IY + d)*
FDCB d 17	LD A, RL (IY + d)*	FDCB d 4A	BIT 1, (IY + d)*

FDCB d 4B	BIT 1, (IY + d)*	FDCB d 7E	BIT 7, (IY + d)
FDCB d 4C	BIT 1, (IY + d)*	FDCB d 7F	BIT 7, (IY + d)*
FDCB d 4D	BIT 1, (IY + d)*	FDCB d 80	LD B, RES 0, (IY + d)*
FDCB d 4E	BIT 1, (IY + d)	FDCB d 81	LD C, RES 0, (IY + d)*
FDCB d 4F	BIT 1, (IY + d)*	FDCB d 82	LD D, RES 0, (IY + d)*
FDCB d 50	BIT 2, (IY + d)*	FDCB d 83	LD E, RES 0, (IY + d)*
FDCB d 51	BIT 2, (IY + d)*	FDCB d 84	LD H, RES 0, (IY + d)*
FDCB d 52	BIT 2, (IY + d)*	FDCB d 85	LD L, RES 0, (IY + d)*
FDCB d 53	BIT 2, (IY + d)*	FDCB d 86	RES 0, (IY + d)
FDCB d 54	BIT 2, (IY + d)*	FDCB d 87	LD A, RES 0, (IY + d)*
FDCB d 55	BIT 2, (IY + d)*	FDCB d 88	LD B, RES 1, (IY + d)*
FDCB d 56	BIT 2, (IY + d)	FDCB d 89	LD C, RES 1, (IY + d)*
FDCB d 57	BIT 2, (IY + d)*	FDCB d 8A	LD D, RES 1, (IY + d)*
FDCB d 58	BIT 3, (IY + d)*	FDCB d 8B	LD E, RES 1, (IY + d)*
FDCB d 59	BIT 3, (IY + d)*	FDCB d 8C	LD H, RES 1, (IY + d)*
FDCB d 5A	BIT 3, (IY + d)*	FDCB d 8D	LD L, RES 1, (IY + d)*
FDCB d 5B	BIT 3, (IY + d)*	FDCB d 8E	RES 1, (IY + d)
FDCB d 5C	BIT 3, (IY + d)*	FDCB d 8F	LD A, RES 1, (IY + d)*
FDCB d 5D	BIT 3, (IY + d)*	FDCB d 90	LD B, RES 2, (IY + d)*
FDCB d 5E	BIT 3, (IY + d)	FDCB d 91	LD C, RES 2, (IY + d)*
FDCB d 5F	BIT 3, (IY + d)*	FDCB d 92	LD D, RES 2, (IY + d)*
FDCB d 60	BIT 4, (IY + d)*	FDCB d 93	LD E, RES 2, (IY + d)*
FDCB d 61	BIT 4, (IY + d)*	FDCB d 94	LD H, RES 2, (IY + d)*
FDCB d 62	BIT 4, (IY + d)*	FDCB d 95	LD L, RES 2, (IY + d)*
FDCB d 63	BIT 4, (IY + d)*	FDCB d 96	RES 2, (IY + d)
FDCB d 64	BIT 4, (IY + d)*	FDCB d 97	LD A, RES 2, (IY + d)*
FDCB d 65	BIT 4, (IY + d)*	FDCB d 98	LD B, RES 3, (IY + d)*
FDCB d 66	BIT 4, (IY + d)	FDCB d 99	LD C, RES 3, (IY + d)*
FDCB d 67	BIT 4, (IY + d)*	FDCB d 9A	LD D, RES 3, (IY + d)*
FDCB d 68	BIT 5, (IY + d)*	FDCB d 9B	LD E, RES 3, (IY + d)*
FDCB d 69	BIT 5, (IY + d)*	FDCB d 9C	LD H, RES 3, (IY + d)*
FDCB d 6A	BIT 5, (IY + d)*	FDCB d 9D	LD L, RES 3, (IY + d)*
FDCB d 6B	BIT 5, (IY + d)*	FDCB d 9E	RES 3, (IY + d)
FDCB d 6C	BIT 5, (IY + d)*	FDCB d 9F	LD A, RES 3, (IY + d)*
FDCB d 6D	BIT 5, (IY + d)*	FDCB d A0	LD B, RES 4, (IY + d)*
FDCB d 6E	BIT 5, (IY + d)	FDCB d A1	LD C, RES 4, (IY + d)*
FDCB d 6F	BIT 5, (IY + d)*	FDCB d A2	LD D, RES 4, (IY + d)*
FDCB d 70	BIT 6, (IY + d)*	FDCB d A3	LD E, RES 4, (IY + d)*
FDCB d 71	BIT 6, (IY + d)*	FDCB d A4	LD H, RES 4, (IY + d)*
FDCB d 72	BIT 6, (IY + d)*	FDCB d A5	LD L, RES 4, (IY + d)*
FDCB d 73	BIT 6, (IY + d)*	FDCB d A6	RES 4, (IY + d)
FDCB d 74	BIT 6, (IY + d)*	FDCB d A7	LD A, RES 4, (IY + d)*
FDCB d 75	BIT 6, (IY + d)*	FDCB d A8	LD B, RES 5, (IY + d)*
FDCB d 76	BIT 6, (IY + d)	FDCB d A9	LD C, RES 5, (IY + d)*
FDCB d 77	BIT 6, (IY + d)*	FDCB d AA	LD D, RES 5, (IY + d)*
FDCB d 78	BIT 7, (IY + d)*	FDCB d AB	LD E, RES 5, (IY + d)*
FDCB d 79	BIT 7, (IY + d)*	FDCB d AC	LD H, RES 5, (IY + d)*
FDCB d 7A	BIT 7, (IY + d)*	FDCB d AD	LD L, RES 5, (IY + d)*
FDCB d 7B	BIT 7, (IY + d)*	FDCB d AE	RES 5, (IY + d)
FDCB d 7C	BIT 7, (IY + d)*	FDCB d AF	LD A, RES 5, (IY + d)*
FDCB d 7D	BIT 7, (IY + d)*	FDCB d B0	LD B, RES 6, (IY + d)*

FDCB d B1 LD C, RES 6, (IY + d)*
 FDCB d B2 LD D, RES 6, (IY + d)*
 FDCB d B3 LD E, RES 6, (IY + d)*
 FDCB d B4 LD H, RES 6, (IY + d)*
 FDCB d B5 LD L, RES 6, (IY + d)*
 FDCB d B6 RES 6, (IY + d)
 FDCB d B7 LD A, RES 6, (IY + d)*
 FDCB d B8 LD B, RES 7, (IY + d)*
 FDCB d B9 LD C, RES 7, (IY + d)*
 FDCB d BA LD D, RES 7, (IY + d)*
 FDCB d BB LD E, RES 7, (IY + d)*
 FDCB d BC LD H, RES 7, (IY + d)*
 FDCB d BD LD L, RES 7, (IY + d)*
 FDCB d BE RES 7, (IY + d)
 FDCB d BF LD A, RES 7, (IY + d)*
 FDCB d C0 LD B, SET 0, (IY + d)*
 FDCB d C1 LD C, SET 0, (IY + d)*
 FDCB d C2 LD D, SET 0, (IY + d)*
 FDCB d C3 LD E, SET 0, (IY + d)*
 FDCB d C4 LD H, SET 0, (IY + d)*
 FDCB d C5 LD L, SET 0, (IY + d)*
 FDCB d C6 SET 0, (IY + d)
 FDCB d C7 LD A, SET 0, (IY + d)*
 FDCB d C8 LD B, SET 1, (IY + d)*
 FDCB d C9 LD C, SET 1, (IY + d)*
 FDCB d CA LD D, SET 1, (IY + d)*
 FDCB d CB LD E, SET 1, (IY + d)*
 FDCB d CC LD H, SET 1, (IY + d)*
 FDCB d CD LD L, SET 1, (IY + d)*
 FDCB d CE SET 1, (IY + d)
 FDCB d CF LD A, SET 1, (IY + d)*
 FDCB d D0 LD B, SET 2, (IY + d)*
 FDCB d D1 LD C, SET 2, (IY + d)*
 FDCB d D2 LD D, SET 2, (IY + d)*
 FDCB d D3 LD E, SET 2, (IY + d)*
 FDCB d D4 LD H, SET 2, (IY + d)*
 FDCB d D5 LD L, SET 2, (IY + d)*
 FDCB d D6 SET 2, (IY + d)
 FDCB d D7 LD A, SET 2, (IY + d)*
 FDCB d D8 LD B, SET 3, (IY + d)*
 FDCB d D9 LD C, SET 3, (IY + d)*
 FDCB d DA LD D, SET 3, (IY + d)*
 FDCB d DB LD E, SET 3, (IY + d)*

FDCB d DC LD H, SET 3, (IY + d)*
 FDCB d DD LD L, SET 3, (IY + d)*
 FDCB d DE SET 3, (IY + d)
 FDCB d DF LD A, SET 3, (IY + d)*
 FDCB d E0 LD B, SET 4, (IY + d)*
 FDCB d E1 LD C, SET 4, (IY + d)*
 FDCB d E2 LD D, SET 4, (IY + d)*
 FDCB d E3 LD E, SET 4, (IY + d)*
 FDCB d E4 LD H, SET 4, (IY + d)*
 FDCB d E5 LD L, SET 4, (IY + d)*
 FDCB d E6 SET 4, (IY + d)
 FDCB d E7 LD A, SET 4, (IY + d)*
 FDCB d E8 LD B, SET 5, (IY + d)*
 FDCB d E9 LD C, SET 5, (IY + d)*
 FDCB d EA LD D, SET 5, (IY + d)*
 FDCB d EB LD E, SET 5, (IY + d)*
 FDCB d EC LD H, SET 5, (IY + d)*
 FDCB d ED LD L, SET 5, (IY + d)*
 FDCB d EE SET 5, (IY + d)
 FDCB d EF LD A, SET 5, (IY + d)*
 FDCB d F0 LD B, SET 6, (IY + d)*
 FDCB d F1 LD C, SET 6, (IY + d)*
 FDCB d F2 LD D, SET 6, (IY + d)*
 FDCB d F3 LD E, SET 6, (IY + d)*
 FDCB d F4 LD H, SET 6, (IY + d)*
 FDCB d F5 LD L, SET 6, (IY + d)*
 FDCB d F6 SET 6, (IY + d)
 FDCB d F7 LD A, SET 6, (IY + d)*
 FDCB d F8 LD B, SET 7, (IY + d)*
 FDCB d F9 LD C, SET 7, (IY + d)*
 FDCB d FA LD D, SET 7, (IY + d)*
 FDCB d FB LD E, SET 7, (IY + d)*
 FDCB d FC LD H, SET 7, (IY + d)*
 FDCB d FD LD L, SET 7, (IY + d)*
 FDCB d FE SET 7, (IY + d)
 FDCB d FF LD A, SET 7, (IY + d)*
 FDE1 POP IY
 FDE3 EX (SP), IY
 FDE5 PUSH IY
 FDE9 JP (IY)
 FDF9 LD SP, IY
 FE n CP n
 FF RST 38h

APÉNDICE – C

Tabla de Conversión Hexadecimal a Decimal

Utilice la siguiente tabla para convertir números entre ambos sistemas de forma rápida y sin necesidad de realizar ningún cálculo:

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

28044 MADRID

ISBN 84-86381-15-0