

## SISTEMAS DE NUMERACION

### Sistema decimal

Desde antiguo el Hombre ha ideado sistemas para numerar objetos, algunos sistemas primitivos han llegado hasta nuestros días, tal es el caso de los "números romanos", pero sin duda el más extendido en la actualidad es el sistema decimal de números arábigos, llamado así por ser los árabes sus creadores.

En el sistema decimal, los números se forman por combinación de 10 signos distintos: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. Cada uno de estos signos tiene un valor, y el valor del número que forman se haya multiplicando el valor de cada uno de ellos por 10 elevado a la potencia correspondiente a su situación en el número, siendo 0 el de más a la derecha, 1 el siguiente y así sucesivamente. De esta forma, el número 5348 sería igual a:

$$5348 = 5 \cdot 10^3 + 3 \cdot 10^2 + 4 \cdot 10^1 + 8 \cdot 10^0 \\ = 5 \cdot 1000 + 3 \cdot 100 + 4 \cdot 10 + 8 \cdot 1$$

La misma denominación del número nos lo recuerda, decimos: cinco mil, trescientos, cuarenta y ocho. El sistema decimal es de uso tan frecuente que no vale la pena insistir en él, pero es importante hacer notar que la base de los exponentes es siempre 10 y por tanto, este sistema se denomina también "de base 10". Es posible crear sistemas que utilicen una base distinta, y de hecho, estos sistemas son muy usados en informática.

### Sistema binario

Un ordenador es una máquina esencialmente binaria, su componente básico es el transistor que sólo admite dos estados posibles, o bien pasa corriente, o bien no pasa.

Los "puristas" podrían objetar que un transistor puede tener múltiples estados dependiendo de la cantidad de corriente que pase; es cierto, pero la medición de esta cantidad de corriente implica una imprecisión que podría crear ambigüedades, y un ordenador no admite ambigüedad. De forma que por debajo de un determinado valor, se considera que no pasa corriente, y por encima de otro, se considera que sí pasa.

Arbitrariamente, asociamos estos dos estados con dos dígitos, cuando no pasa corriente, decimos que tenemos un "0" y cuando pasa, decimos que tenemos un "1". De esta forma, podremos representar mediante un dígito el estado de un interruptor: "1" cuando esté encendido y "0" cuando esté apagado.

Si tenemos una serie de interruptores puestos en fila, podríamos representar el estado de todos ellos mediante un número binario. En la FIGURA 1 vemos una serie de interruptores cuyo estado podría ser definido mediante el número binario: "1001".

Como se ve, el sistema binario es perfectamente adecuado para su uso en circuitos.

tos electrónicos. A cada "1" o "0" de un número binario le llamaremos "dígito binario", que puede abreviarse como "bit" (contracción de "binary digit").

El valor de un número binario se haya de la misma forma que en el sistema decimal, excepto que esta vez, la base es "2". Así el número "10011" será:

$$10011 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Es decir:

$$10011 = 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 19 \text{ en decimal}$$

Ya hemos visto implícitamente, cómo transformar un número binario en decimal, el proceso inverso (transformar un número decimal en binario), lo veremos más adelante, cuando estudiemos el método general para transformar números en cualquier base.

### Operaciones aritméticas en binario

Los números binarios se pueden sumar, restar, multiplicar y dividir de igual forma que los decimales, sólo es necesario conocer las "tablas" correspondientes. Veamos primero la suma.

Para sumar en decimal los números 19 y 28, los colocamos de la siguiente forma:

$$\begin{array}{r} 19 \\ + 28 \\ \hline \end{array}$$

# CODIGO MAQUINA

Jesús Alonso Rodríguez

**C**omienza aquí un nuevo curso dedicado al estudio y utilización de uno de los lenguajes más profesionales entre los usuarios de ordenadores, el Código Máquina. Con él, al igual que hiciéramos con su antecesor, el Basic, pretendemos cubrir las necesidades de nuestros lectores y ofrecerles una idea clara, ayudándonos de ejemplos y todo tipo de gráficos, de este lenguaje, tanto para iniciados como para los que quieran llegar a serlo.

HOBBY PRESS, S.A.



# CODIGO MAQUINA

La diferencia esencial entre un lenguaje de alto nivel, como el BASIC, y el código máquina, es que, mientras el primero se escribe en un lenguaje coloquial empleando como base el idioma inglés, el segundo ahorra memoria y tiempo de proceso a cambio de escribirlo en unos códigos que representan los bits que entiende el microprocesador.

Cuando se manda ejecutar un comando BASIC al ordenador, es el propio programa monitor el que interpreta y ejecuta ese comando. En un programa escrito en BASIC se iría haciendo así por cada comando o instrucción. En cambio, en un programa escrito en código máquina cada instrucción es leída directamente por el microprocesador y ejecutada de inmediato.

Como desventaja, la realización de un programa en código máquina no exige un planteamiento más minucioso del problema.

Se puede pues deducir que programar en BASIC es más

fácil, se emplea un lenguaje casi humano, pero se desperdicia una cantidad tremenda de memoria y tiempo de microprocesador, mientras que con el código máquina se ahorra parte de esa memoria y mucho en tiempo de proceso, pero es necesario usar unos códigos nemotécnicos para facilitar lo que sería una secuencia aparentemente aleatoria.

Este código nemotécnico es lo que se denomina ASSEMBLER.

El curso comenzará por explicar lo que es un código de máquina, analizando las diferencias entre intérprete, ensamblador y compilador. También se verá el porqué de utilizar sistemas de numeración distintos al decimal. Posteriormente se estudiará la arquitectura del microprocesador Z-80 para entrar ya a estudiar todo el repertorio de instrucciones y formatos así como las técnicas de programación de más utilidad. Finalmente, estudiaremos el fun-

cionamiento de un programa ensamblador y los recursos que proporciona.

Durante todos los capítulos se irán viendo ejemplos clarificadores y ejercicios de dificultad ascendente para afianzar los conocimientos.

Para justificar el esfuerzo necesario en aprender a programar en ASSEMBLER o código máquina, hay que tener en cuenta lo siguiente:

a) En el mejor de los casos, en el Spectrum se dispone de 48 K de memoria.

b) Los programas de utilidad y los juegos más sofisticados están en este lenguaje.

c) El programa monitor o sistema operativo (almacenado en la ROM) también lo usa, lo que nos permitirá investigarlo.

Por último, añadir que no es necesario dominar el BASIC, es más, ni siquiera conocerlo, para aprender a programar en código máquina, si bien como la lógica es la misma, facilitara su comprensión.

re la actuación del programa MONITOR, por lo cual es muchísimo más rápido.

Tanto el lenguaje de máquina como el simbólico hasta aquí visto es imaginario, sólo nos ha valido para la mayor comprensión del tema. He-

mos ideado un microprocesador sumamente sencillo con el fin de que el lector comprendiera fácilmente lo que es un código máquina. A partir de ahora, nos ceñiremos al microprocesador Z-80 de ZILOG, su repertorio de

instrucciones abarca más de 500, el formato de instrucción no es tan sencillo como el visto aquí y trabaja sobre posiciones de memoria de 8 bits; no obstante, los principios básicos de funcionamiento son los mismos. ■





ETIQUETA (opcional)	NEMOTECNICO/REGISTRO (opcional)	POSICION DE MEMORIA (opcional)
------------------------	------------------------------------	-----------------------------------

## Codificación del supuesto en lenguaje simbólico:

Campos:

RESULTADO	(0)
OTROCASO	(1)
CONSTANTE	(31) (valor dec. de 11111)
NUEVE	(9)
NUMEROS = 22	(valor dec. de 10110)

Cuando ponemos el número entre paréntesis indicamos el contenido del campo, y cuando se pone el signo "=" nos referimos al valor que tiene el literal. Cualquier símbolo tiene que diferenciar entre dirección y contenido.

Instrucciones:

SUMAR	CAV1	NUEVE
DEC	SUM/1	NUMEROS
SNC/1	SUMAR	
SC/A	NOSUMA	
RES	CONSTANTE	
SC/A	NOSUMA	
SUM	CONSTANTE	
AL/A	RESULTADO	
FIN		
NOSUMA	CAV1	CONSTANTE
AL/1	OTROCASO	
FIN		

¿Qué se ha hecho?

1.º Definir los campos.

Damos a unos campos un nombre y un contenido inicial, siempre que queramos tomar su contenido nos acordaremos sólo del nombre del campo.

2.º Definir constantes.

DORES o COMPILADORES que cogen las instrucciones, las colocan una detrás de otra en lenguaje máquina, calculan las direcciones relativas de cada campo o etiqueta y dan como resultado un programa en código máquina que se llama *código objeto*. Este programa posteriormente se carga en una posición de memoria de la máquina y ese cargador le suma a las direcciones relativas el valor de la dirección de carga con lo cual tenemos un programa listo para ejecutarse, a este programa se le llama *absoluto*. Todos los ensambladores que existen para el Spectrum, dan como resultado un programa absoluto.

En el supuesto que hemos realizado en una máquina imaginaria, el programa absoluto es la primera secuencia de números que hicimos.

## Ejecución

El programa absoluto en código máquina lo ejecuta el microprocesador directamente según los siguientes pasos:

- lee instrucción
- incrementa puntero siguiente instrucción
- ejecuta instrucción.

Cuando hay una instrucción que modifica la secuencia del programa lo que hace es modificar el puntero de la siguiente instrucción (de forma equivalente a un GOTO en BASIC, pero en vez de mandar a un número de línea, manda a una posición de memoria apuntada por una etiqueta).

Como se ve, la ejecución de un programa absoluto no requiere la participación de ningún otro programa, como en el caso del BASIC que requiere

# CODIGO MAQUINA Y ASSEMBLER

## Lenguaje de máquina

Un lenguaje de máquina es aquel con el que trabaja el microprocesador; para reaccionar y efectuar la operación que se desea, necesita de una secuencia de señales eléctricas almacenadas como "unos" y "ceros" en las posiciones de la memoria. Una y solo una secuencia de señales concreta, realiza una determinada operación. Identificaremos a partir de ahora la existencia de señal con un "1" y la ausencia con un "0".

## Microprocesador imaginario

El Spectrum trabaja con el microprocesador Z-80 cuyo funcionamiento se explicará en el Capítulo 3 de este curso. El Z-80 es un microprocesador un tanto complejo, de forma que para introducirnos en el estudio del código máquina vamos a idear un microprocesador imaginario con un funcionamiento extremadamente simplificado.

Supongamos un microprocesador que tiene un registro de índice "I" y uno aritmético "A", a los que identifica como

"01" y "10" respectivamente. Un registro en un microprocesador es un campo interno modificable; denominamos campo a un lugar donde se almacenan datos; de esta forma, un registro es algo similar a una posición de memoria pero interno al microprocesador, su función es parecida a la de las variables en el BASIC.

También dispone del siguiente repertorio de instrucciones, cada una de las cuales tiene asignado un código de operación:

OPERACION	CODIGO
Cargar registro	001
Almacenar registro	010
Sumar en registro aritmético	011
Restar en registro aritmético	100
Saltar por contenido cero	101
Saltar por contenido no cero	110
Decrementar registro índice	111

Suponemos que el ordenador en el que está incorporado utiliza posiciones de memoria de 10 bits (el Spectrum las utiliza de 8). Nuestro microprocesador trabaja con un formato fijo para entender la secuencia de señales tal que:

- Los tres primeros bits son el identificativo o código

de la operación que se quiere realizar.

- Los dos siguientes son el identificativo del registro con que se opera.
- Los cinco siguientes y últimos indican la posición de memoria, si procede, que va desde 00000 a 11111.

El formato de instrucción quedaría como se muestra en la FIGURA 1 y las instrucciones serían las siguientes:

CARGAR REGISTRO:

Definición: Carga el registro indicado con el contenido de la posición de memoria.  
Formato:

0	0	1	x	x	x	x	x	x	x
---	---	---	---	---	---	---	---	---	---

ALMACENAR REGISTRO:

Definición: Almacena el contenido del registro indicado en la posición de memoria.  
Formato:

0	1	0	x	x	x	x	x	x	x
---	---	---	---	---	---	---	---	---	---

SUMAR EN REGISTRO ARITMETICO:

Definición: Suma en el registro aritmético el contenido



FIGURA 1



de la posición de memoria que resulta de sumar la posición de memoria indicada en la instrucción con el contenido del registro índice si está indicado.  
Formato:



RESTAR EN REGISTRO ARITMETICO:

Definición: Resta en el registro aritmético el contenido de la posición de memoria que resulta de sumar la posición de memoria indicada en la instrucción con el contenido del registro índice si está indicado.  
Formato:



SALTAR POR CONTENIDO CERO:

Definición: Salta a la posición de memoria indicada si el valor del registro señalado es cero.  
Formato:



SALTAR POR CONTENIDO NO CERO:

Definición: Salta a la posición de memoria indicada si el valor del registro señalado es distinto de cero.  
Formato:



DECREMENTAR EL REGISTRO INDICE:

Definición: Resta uno al valor del registro índice.  
Formato:



Definido ya este microprocesador con la única intención de hacer más comprensibles los conceptos que se pretenden adquirir vamos, siguiendo la misma línea, a dar solución a un supuesto problema.

### Supuesto

Se quiere sumar el contenido de las diez posiciones de memoria a partir de la posición 10110. Si todos los valores son cero o el resultado es 11111, almacenaremos 11111 en la posición 00000; si no, ponemos el resultado en la posición 00001.

Para irnos acostumbrando a trabajar con métodos de programación empezaremos por hacer el organigrama. Es interesante intentar hacerlo en un papel aparte y luego comprobar resultados. No tiene por qué ser exactamente igual; cualquier organigrama es válido siempre que funcione. Un posible organigrama está representado en la FIGURA 2.

### Codificación

Basándonos en los códigos definidos anteriormente, iremos definiendo las posiciones de memoria.

Campos:

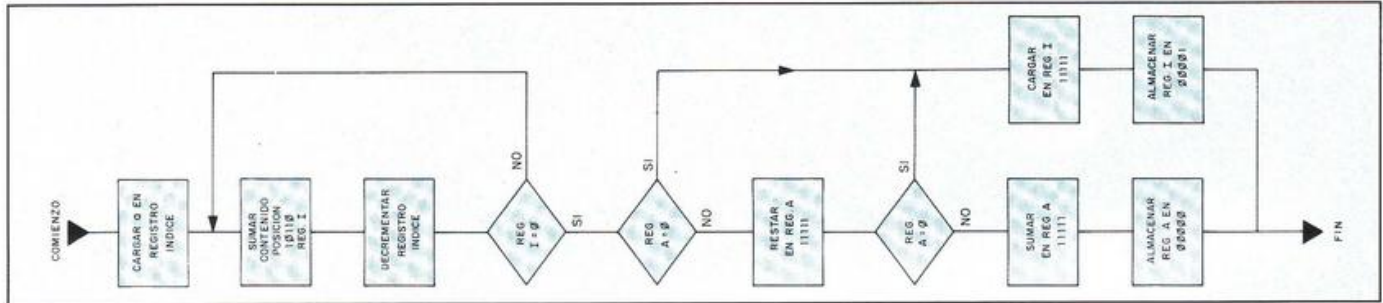


FIGURA 2.

DIRECCION	CONTENIDO
00011	0000011111
00100	0000001001 (9 en binario)

El programa lo cargaremos a partir de la posición de memoria 01000. Instrucciones:

DIRECCION	INSTRUCCION	COMENTARIOS
01000	001 01 00100	(Cargar el registro "I" con un 9)
01001	011 01 10110	(Suma contenido 10110 + reg. "I")
01010	111 00 00000	(Decrementa el registro "I")
01011	110 01 01001	(Seguir sumando si no van 9 pos.)
01100	101 10 10100	(Saltar si resultado=cero a 10100)
01101	100 00 00011	(Restar a la suma el valor 11111)
01110	101 10 10100	(Saltar si resultado=cero a 10100)
01111	011 00 00011	(Recuperar valor acumulado)
10000	010 10 00000	(Almacenar resultado en pos. 00000)
SALIDA		
10100	001 01 00011	(Cargar en reg. "I" el valor 11111)
10101	010 01 00001	(Almacenar resultado en pos. 00001)
SALIDA		

La memoria quedaría configurada de la siguiente manera:

...	00	...	01	...	10	...	11
000..	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
001..	0000000001	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
010..	0010100100	0110110110	1110000000	1100101001	0110000000	0000000000	0000000000
011..	1011010100	1000000011	1011010100	0110000000	0000000000	0000000000	0000000000
100..	0101000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
101..	0010100011	0100100001	0000000000	0000000000	0000000000	0000000000	0000000000
110..	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
111..	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000

Como se puede ver, esto supone un trabajo tremendo, y la facilidad de cometer errores es evidente. Siguiendo nuestro desarrollo de microprocesador simulado, tenemos que buscar una manera de facilitar el trabajo, para lo cual vamos a hacer corresponder a cada instrucción con una secuencia de letras que nos sirva para recordarla y nos de una idea de la operación que debe realizar. A esto se le denomina **representación simbólica** o código nemotécnico.

OPERACION	CODIGO MAQUINA	NEMOTENICO
Cargar registro	001	CA
Almacenar registro	010	AL
Suma en registro aritmético	011	SUM
Restar en registro aritmético	100	RES
Saltar si contenido cero	101	SC
Saltar si contenido no-cero	110	SN
Decrementar registro "I"	111	DEC

Se verá que es más fácil recordar el código nemotécnico o simbólico que los números de código máquina.

Sigamos simplificando, cuando nos refiramos a los registros en lugar de llamarlos 01 y 10 llamaremos al registro índice "I" y al registro aritmético "A".

Registro índice	I	código 01
Registro aritmético	A	código 10

Por último, cada vez que nos refiramos a una posición de memoria en lugar de recordar el valor numérico de su dirección, le daremos un nombre o literal, este literal tiene el valor de la dirección que representa y se le conoce con el nombre genérico de **etiqueta**.

Entonces diremos que la representación simbólica de una instrucción es la siguiente:



# EJERCICIOS

1. Transformar a decimal y hexadecimal sin mirar la FIGURA 2, los siguientes números:

- 111010010  
 - 01101101  
 - 00110100

2. Realizar las siguientes sumas en binario:

01110100  
 + 10010100  
 =

3. Realizar las siguientes restas en binario:

11001011  
 - 01001001  
 =

(En estos ejercicios, se pueden comprobar los resultados pasando los números a decimal con ayuda de la FIGURA 2 y operándolos en decimal).

4. Poner los siguientes números en "complemento a 2".

- 00001111  
 - 11110000  
 - 01010101

5. Realizar un AND, un OR y un EXOR entre los siguientes números:

- 01010101 y 10101010  
 - 00001110 y 00111110  
 - 11110101 y 11110101

6. Convertir a hexadecimal y a binario los siguientes números decimales (puede verificar los resultados con ayuda de la FIGURA 2 o del PROGRAMA 1):

- 255  
 - 65535  
 - 23296

Si ha resuelto correctamente estos ejercicios, ya sabe casi todo lo que hay que saber sobre los sistemas binario y hexadecimal. ¡Enhorabuena!

Y a continuación hacemos: "9 más 8 igual 17 y me llevo 1, 1 más 1 más 2 igual 4" el resultado es 47, es decir:

$$\begin{array}{r} 1 \\ 19 \\ + 28 \\ \hline = 47 \end{array}$$

Al sumar 9 y 8 nos da un resultado superior a 9, es decir, superior al dígito más alto de nuestro sistema, por lo que decimos que "nos llevamos una", esta "una" que "nos llevamos" se denomina en binario "acarreo" (o "carry" en inglés). Cuando se suma en binario, es necesario tener en cuenta el acarreo.

Vamos ahora a ver la "tabla de sumar" en binario:

$$\begin{array}{l} 0 + 0 = 0 \\ 0 + 1 = 1 \\ 1 + 1 = 10 \end{array}$$

Es decir, cuando sumamos 1 más 1 el resultado es 0 pero se produce un acarreo de 1. Veamos un ejemplo: vamos a sumar los números 19 y 28 en binario. 19 es 10011 y 28 es 11100, de esta forma:

$$\begin{array}{r} 1 \\ 10011 \\ + 11100 \\ \hline = 101111 \end{array}$$

El resultado es 101111, o bien 47 en decimal. Al sumar los dos unos de la izquierda, se ha producido un acarreo que hemos anotado encima de la siguiente columna. ¡Al igual que en decimal, los ceros a la izquierda son irrelevantes.

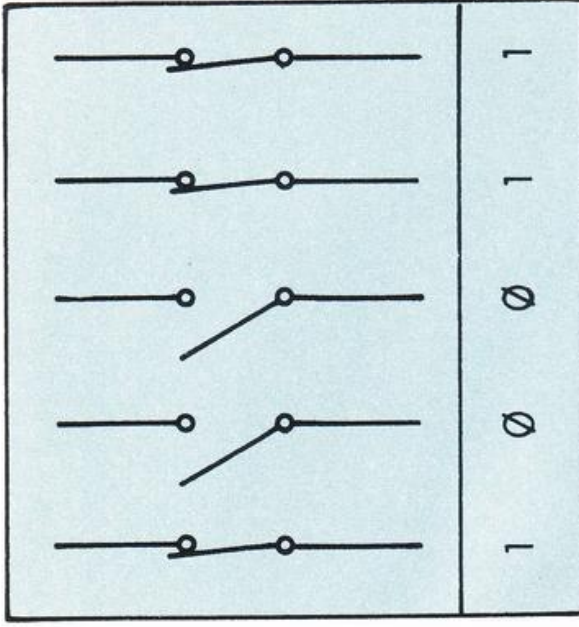


Figura 1.

Es interesante saber cómo realiza el ordenador esta operación. El programador de Basic, seguramente, esté familiarizado con los operadores lógicos AND, OR y NOT, pero quizá no lo esté tanto con el operador EXOR (OR exclusivo). A continuación se muestran las "tablas de verdad" correspondientes a estos operandos. Es conveniente manejar con facilidad el operador EXOR, ya que se utiliza con frecuencia al programar en código máquina.

$$\begin{array}{l} 0 \text{ AND } 0 = 0 \\ 0 \text{ AND } 1 = 0 \\ 1 \text{ AND } 0 = 0 \\ 1 \text{ AND } 1 = 1 \end{array}$$

$$\begin{array}{l} 0 \text{ OR } 0 = 0 \\ 0 \text{ OR } 1 = 1 \\ 1 \text{ OR } 0 = 1 \\ 1 \text{ OR } 1 = 1 \end{array}$$

$$\begin{array}{l} \text{NOT } 0 = 1 \\ \text{NOT } 1 = 0 \end{array}$$

$$\begin{array}{l} 0 \text{ EXOR } 0 = 0 \\ 0 \text{ EXOR } 1 = 1 \\ 1 \text{ EXOR } 0 = 1 \\ 1 \text{ EXOR } 1 = 0 \end{array}$$

El equivalente eléctrico del operador AND, serían dos interruptores en serie, el del operador OR, dos interruptores en paralelo. Siguiendo la misma analogía, el equivalente eléctrico del operador EXOR, serían dos interruptores conmutados, como los que se utilizan frecuentemente para encender y apagar la luz de una habitación desde dos puntos distintos.

Vistas estas tablas, no es difícil deducir la correspondencia con la tabla de sumar vista anteriormente. La suma se obtiene mediante una operación EXOR entre los dos bits, y el acarreo se obtiene mediante una operación AND. Esta correspondencia es la que permite a un ordenador realizar cálculos, ya que eléctrico



tricamente, sólo es posible realizar operaciones lógicas.

No obstante, no se preocupe el lector por tener que realizar operaciones lógicas bit a bit cada vez que quiera sumar dos números, el juego de instrucciones del microprocesador Z-80, afortunadamente, incluye las instrucciones necesarias para realizar sumas y restas de forma bastante sencilla.

### Números negativos

Hemos visto cómo suma una ordenador, pero ¿cómo resta? Para conseguir restar en binario, tendremos que establecer antes un convenio sobre qué consideramos números negativos.

Se denomina "complemento a 1" de un bit, a lo que le falta a ese bit para ser "1", es decir, el complemento de "1" es "0" y el de "0" es "1" (el complemento a 1 viene dado por el operador lógico NOT). Por tanto, el complemento a 1 de un número binario es el resultado de cambiar sus "unos" por "ceros" y sus "ceros" por "unos". Por ejemplo: el complemento a 1 de "10011101" es "01100010".

Por otro lado, se denomina "complemento a 2" al "complemento a 1" más 1, es decir, al resultado de cambiar los unos por ceros y los ceros por unos, y luego sumar uno. Veamos algunos ejemplos:

NUMERO ORIGINAL	COMPLEMENTO A 1	COMPLEMENTO A 2
11001011	00110100	00110101
00100100	11011011	11011100
01010101	10101010	10101011
10101010	01010101	01010110

Podría parecer algo arbitrario, sin embargo es sumamente útil, ya que como veremos a continuación, es posible usar el complemento a 2 de un número como su negativo. Para restar dos números, sumamos al "minuendo" el complemento a 2 del "sustraendo". Vamos a ver algunos ejemplos, trabajando con números de 8 bits, que son los que utiliza el Z-80 (de hecho, también utiliza números de 16 bits, pero sería demasiado largo para un simple ejemplo). Vamos a restar 28 menos 19, para lo cual sumamos a 28 el complemento a 2 de 19:

$$\begin{array}{r} 28 \quad 00011100 \\ - 19 \quad 11101101 \\ \hline = 9 \quad 100001001 \end{array}$$

Obtenemos el número "00010001" con un acarreo de "1". El acarreo nos indica que el resultado es positivo, es decir, el resultado es "+9" como cabía esperar.

Ahora vamos a realizar la operación inversa, es decir, vamos a restar 19 menos 28 por tanto, tendremos que sumar a 19 el complemento a 2 de 28:

$$\begin{array}{r} 19 \quad 00010011 \\ - 28 \quad 11100100 \\ \hline = -9 \quad 011110111 \end{array}$$

Esta vez hemos obtenido el número "01111011" con un

acarreo de "0". El hecho de que el acarreo sea "0" nos indica que el número es negativo, y "1111011" es, precisamente, complemento a 2 de "9", es decir, "-9" como también cabía esperar.

Un hábil lector habrá comprobado que, trabajando con números de 8 bits, podemos saber si un número es negativo con sólo mirar el primer bit (el de más a la izquierda): si este bit es "1", el número será negativo. Bien, esto no siempre es cierto. Trabajando con 8 bits, se pueden representar 256 números distintos (desde 0 hasta 255), el Z-80 los considera casi siempre, todos positivos, pero hay veces que considera positivos a los 128 primeros (desde 0 a 127) y negativos a los 128 restantes (desde 128 a 255). En este último caso, si funciona la regla explicada anteriormente, y de hecho, al bit de más a la izquierda se le denomina "bit de signo".

De esta forma, 255 sería equivalente a "-1", 254 a "-2", 253 a "-3", y así sucesivamente hasta 128 que sería en realidad, "-128". Podría parecer un poco "liso", pero con un poco de imaginación, se puede asimilar al funcionamiento de un cuenta-kilómetros de automóvil. Si partimos de un número cualquiera y vamos restando 1, llegará un momento que obtendremos "00000000", si a continuación volvemos a restar 1, obtendremos "11111111" (más un acarreo, lógicamente), es decir, "255" en decimal, si volvemos a restar 1, obtendremos "254": parece lógico asignar a "255" el valor "-1" y a "254" el "-2" y así sucesivamente. En la segunda columna de la FIGURA 2, podrá ilustrar esto con mayor claridad.

$$\begin{array}{r} 23730 \quad 2 \\ \hline 03 \quad 11865 \quad 2 \\ 17 \quad 18 \quad 5932 \quad 2 \\ 13 \quad 06 \quad 19 \quad 2966 \quad 2 \\ 10 \quad 05 \quad 13 \quad 09 \quad 1483 \quad 2 \\ 0 \quad 1 \quad 12 \quad 16 \quad 08 \quad 741 \\ \hline 0 \quad 06 \quad 03 \quad 1 \\ \hline 741 \quad 2 \\ \hline 14 \quad 370 \quad 2 \\ \hline 01 \quad 17 \quad 185 \quad 2 \\ 1 \quad 10 \quad 05 \quad 92 \quad 2 \\ 0 \quad 12 \quad 46 \quad 2 \\ 0 \quad 06 \quad 23 \quad 2 \\ 0 \quad 03 \quad 11 \quad 2 \\ 1 \quad 1 \quad 5 \quad 2 \\ 1 \quad 2 \quad 2 \\ 0 \quad 1 \quad 2 \\ 1 \quad 0 \end{array}$$

$$23730 = 101110010110010$$

Figura 5.

de ahora representaremos los números hexadecimales seguidos de una "h" y los binarios, seguidos de una "b".

Con el PROGRAMA 1, podrá introducir un número en decimal, binario o Hexa, y el programa devolverá como resultado, ese mismo número en decimal, Hexa y binario.

Cuando introduzca un número binario, indíquelo terminando el número con una "b".

Con el PROGRAMA 1, podrá introducir un número en decimal, binario o Hexa, y el programa devolverá como resultado, ese mismo número en decimal, Hexa y binario.



## PROGRAMA 1

```

10 REM PROGRAM 1 *****
*****
**
**      CONVERSION DE BASES
**
*****
100 INPUT "Numero? ";c$: RANDOM
110 IF c$="STOP" THEN USR 3582
120 IF c$="9" THEN GO TO 9999
130 LET a$="0123456789ABCDEF"
140 IF c$="0" THEN GO TO 100
150 LET n$=""
160 FOR n=1 TO LEN c$
170 LET a$=a$+a$(VAL(c$(n)))
180 FOR n=1 TO LEN c$ IF c$(n)
190 OR c$(n)>"9" THEN GO TO 3000
200 NEXT n:LET a$=VAL c$: GO TO
3000
3000 PRINT AT 21,0;"(ERROR) de s
3100 GOTO 100
3200 LET c$=c$ TO LEN c$-1: FO
3300 LET R=n-1: IF c$(n)<"0" OR
3400 c$(n)>"9" THEN GO TO 3000
3500 NEXT n
3600 LET a$=a$+a$(VAL c$)
3700 LET a$=a$+a$(VAL c$(a$))
3800 LET c$=c$ TO LEN c$-1: FO
3900 FOR n=1 TO LEN c$ IF c$(n)
4000 OR c$(n)>"9" THEN GO SUB 2500
4100 NEXT n
4200 LET a$=a$+a$(VAL c$)
4300 LET a$=a$+a$(VAL c$(a$))
4400 LET c$=c$ TO LEN c$-1: FO
4500 FOR n=1 TO LEN c$ IF c$(n)
4600 OR c$(n)>"9" THEN GO SUB 2500
4700 NEXT n
4800 LET a$=a$+a$(VAL c$)
4900 LET a$=a$+a$(VAL c$(a$))
5000 LET c$=c$ TO LEN c$-1: FO
5100 FOR n=1 TO LEN c$ IF c$(n)
5200 OR c$(n)>"9" THEN GO SUB 2500
5300 NEXT n
5400 LET a$=a$+a$(VAL c$)
5500 LET a$=a$+a$(VAL c$(a$))
5600 LET c$=c$ TO LEN c$-1: FO
5700 FOR n=1 TO LEN c$ IF c$(n)
5800 OR c$(n)>"9" THEN GO SUB 2500
5900 NEXT n
6000 LET a$=a$+a$(VAL c$)
6100 LET a$=a$+a$(VAL c$(a$))
6200 LET c$=c$ TO LEN c$-1: FO
6300 FOR n=1 TO LEN c$ IF c$(n)
6400 OR c$(n)>"9" THEN GO SUB 2500
6500 NEXT n
6600 LET a$=a$+a$(VAL c$)
6700 LET a$=a$+a$(VAL c$(a$))
6800 LET c$=c$ TO LEN c$-1: FO
6900 FOR n=1 TO LEN c$ IF c$(n)
7000 OR c$(n)>"9" THEN GO SUB 2500
7100 NEXT n
7200 LET a$=a$+a$(VAL c$)
7300 LET a$=a$+a$(VAL c$(a$))
7400 LET c$=c$ TO LEN c$-1: FO
7500 FOR n=1 TO LEN c$ IF c$(n)
7600 OR c$(n)>"9" THEN GO SUB 2500
7700 NEXT n
7800 LET a$=a$+a$(VAL c$)
7900 LET a$=a$+a$(VAL c$(a$))
8000 LET c$=c$ TO LEN c$-1: FO
8100 FOR n=1 TO LEN c$ IF c$(n)
8200 OR c$(n)>"9" THEN GO SUB 2500
8300 NEXT n
8400 LET a$=a$+a$(VAL c$)
8500 LET a$=a$+a$(VAL c$(a$))
8600 LET c$=c$ TO LEN c$-1: FO
8700 FOR n=1 TO LEN c$ IF c$(n)
8800 OR c$(n)>"9" THEN GO SUB 2500
8900 NEXT n
9000 LET a$=a$+a$(VAL c$)
9100 LET a$=a$+a$(VAL c$(a$))
9200 LET c$=c$ TO LEN c$-1: FO
9300 FOR n=1 TO LEN c$ IF c$(n)
9400 OR c$(n)>"9" THEN GO SUB 2500
9500 NEXT n
9600 LET a$=a$+a$(VAL c$)
9700 LET a$=a$+a$(VAL c$(a$))
9800 LET c$=c$ TO LEN c$-1: FO
9900 FOR n=1 TO LEN c$ IF c$(n)
10000 OR c$(n)>"9" THEN GO SUB 2500
10100 NEXT n
10200 LET a$=a$+a$(VAL c$)
10300 LET a$=a$+a$(VAL c$(a$))
10400 LET c$=c$ TO LEN c$-1: FO
10500 FOR n=1 TO LEN c$ IF c$(n)
10600 OR c$(n)>"9" THEN GO SUB 2500
10700 NEXT n
10800 LET a$=a$+a$(VAL c$)
10900 LET a$=a$+a$(VAL c$(a$))
11000 LET c$=c$ TO LEN c$-1: FO
11100 FOR n=1 TO LEN c$ IF c$(n)
11200 OR c$(n)>"9" THEN GO SUB 2500
11300 NEXT n
11400 LET a$=a$+a$(VAL c$)
11500 LET a$=a$+a$(VAL c$(a$))
11600 LET c$=c$ TO LEN c$-1: FO
11700 FOR n=1 TO LEN c$ IF c$(n)
11800 OR c$(n)>"9" THEN GO SUB 2500
11900 NEXT n
12000 LET a$=a$+a$(VAL c$)
12100 LET a$=a$+a$(VAL c$(a$))
12200 LET c$=c$ TO LEN c$-1: FO
12300 FOR n=1 TO LEN c$ IF c$(n)
12400 OR c$(n)>"9" THEN GO SUB 2500
12500 NEXT n
12600 LET a$=a$+a$(VAL c$)
12700 LET a$=a$+a$(VAL c$(a$))
12800 LET c$=c$ TO LEN c$-1: FO
12900 FOR n=1 TO LEN c$ IF c$(n)
13000 OR c$(n)>"9" THEN GO SUB 2500
13100 NEXT n
13200 LET a$=a$+a$(VAL c$)
13300 LET a$=a$+a$(VAL c$(a$))
13400 LET c$=c$ TO LEN c$-1: FO
13500 FOR n=1 TO LEN c$ IF c$(n)
13600 OR c$(n)>"9" THEN GO SUB 2500
13700 NEXT n
13800 LET a$=a$+a$(VAL c$)
13900 LET a$=a$+a$(VAL c$(a$))
14000 LET c$=c$ TO LEN c$-1: FO
14100 FOR n=1 TO LEN c$ IF c$(n)
14200 OR c$(n)>"9" THEN GO SUB 2500
14300 NEXT n
14400 LET a$=a$+a$(VAL c$)
14500 LET a$=a$+a$(VAL c$(a$))
14600 LET c$=c$ TO LEN c$-1: FO
14700 FOR n=1 TO LEN c$ IF c$(n)
14800 OR c$(n)>"9" THEN GO SUB 2500
14900 NEXT n
15000 LET a$=a$+a$(VAL c$)
15100 LET a$=a$+a$(VAL c$(a$))
15200 LET c$=c$ TO LEN c$-1: FO
15300 FOR n=1 TO LEN c$ IF c$(n)
15400 OR c$(n)>"9" THEN GO SUB 2500
15500 NEXT n
15600 LET a$=a$+a$(VAL c$)
15700 LET a$=a$+a$(VAL c$(a$))
15800 LET c$=c$ TO LEN c$-1: FO
15900 FOR n=1 TO LEN c$ IF c$(n)
16000 OR c$(n)>"9" THEN GO SUB 2500
16100 NEXT n
16200 LET a$=a$+a$(VAL c$)
16300 LET a$=a$+a$(VAL c$(a$))
16400 LET c$=c$ TO LEN c$-1: FO
16500 FOR n=1 TO LEN c$ IF c$(n)
16600 OR c$(n)>"9" THEN GO SUB 2500
16700 NEXT n
16800 LET a$=a$+a$(VAL c$)
16900 LET a$=a$+a$(VAL c$(a$))
17000 LET c$=c$ TO LEN c$-1: FO
17100 FOR n=1 TO LEN c$ IF c$(n)
17200 OR c$(n)>"9" THEN GO SUB 2500
17300 NEXT n
17400 LET a$=a$+a$(VAL c$)
17500 LET a$=a$+a$(VAL c$(a$))
17600 LET c$=c$ TO LEN c$-1: FO
17700 FOR n=1 TO LEN c$ IF c$(n)
17800 OR c$(n)>"9" THEN GO SUB 2500
17900 NEXT n
18000 LET a$=a$+a$(VAL c$)
18100 LET a$=a$+a$(VAL c$(a$))
18200 LET c$=c$ TO LEN c$-1: FO
18300 FOR n=1 TO LEN c$ IF c$(n)
18400 OR c$(n)>"9" THEN GO SUB 2500
18500 NEXT n
18600 LET a$=a$+a$(VAL c$)
18700 LET a$=a$+a$(VAL c$(a$))
18800 LET c$=c$ TO LEN c$-1: FO
18900 FOR n=1 TO LEN c$ IF c$(n)
19000 OR c$(n)>"9" THEN GO SUB 2500
19100 NEXT n
19200 LET a$=a$+a$(VAL c$)
19300 LET a$=a$+a$(VAL c$(a$))
19400 LET c$=c$ TO LEN c$-1: FO
19500 FOR n=1 TO LEN c$ IF c$(n)
19600 OR c$(n)>"9" THEN GO SUB 2500
19700 NEXT n
19800 LET a$=a$+a$(VAL c$)
19900 LET a$=a$+a$(VAL c$(a$))
20000 LET c$=c$ TO LEN c$-1: FO
20100 FOR n=1 TO LEN c$ IF c$(n)
20200 OR c$(n)>"9" THEN GO SUB 2500
20300 NEXT n
20400 LET a$=a$+a$(VAL c$)
20500 LET a$=a$+a$(VAL c$(a$))
20600 LET c$=c$ TO LEN c$-1: FO
20700 FOR n=1 TO LEN c$ IF c$(n)
20800 OR c$(n)>"9" THEN GO SUB 2500
20900 NEXT n
21000 LET a$=a$+a$(VAL c$)
21100 LET a$=a$+a$(VAL c$(a$))
21200 LET c$=c$ TO LEN c$-1: FO
21300 FOR n=1 TO LEN c$ IF c$(n)
21400 OR c$(n)>"9" THEN GO SUB 2500
21500 NEXT n
21600 LET a$=a$+a$(VAL c$)
21700 LET a$=a$+a$(VAL c$(a$))
21800 LET c$=c$ TO LEN c$-1: FO
21900 FOR n=1 TO LEN c$ IF c$(n)
22000 OR c$(n)>"9" THEN GO SUB 2500
22100 NEXT n
22200 LET a$=a$+a$(VAL c$)
22300 LET a$=a$+a$(VAL c$(a$))
22400 LET c$=c$ TO LEN c$-1: FO
22500 FOR n=1 TO LEN c$ IF c$(n)
22600 OR c$(n)>"9" THEN GO SUB 2500
22700 NEXT n
22800 LET a$=a$+a$(VAL c$)
22900 LET a$=a$+a$(VAL c$(a$))
23000 LET c$=c$ TO LEN c$-1: FO
23100 FOR n=1 TO LEN c$ IF c$(n)
23200 OR c$(n)>"9" THEN GO SUB 2500
23300 NEXT n
23400 LET a$=a$+a$(VAL c$)
23500 LET a$=a$+a$(VAL c$(a$))
23600 LET c$=c$ TO LEN c$-1: FO
23700 FOR n=1 TO LEN c$ IF c$(n)
23800 OR c$(n)>"9" THEN GO SUB 2500
23900 NEXT n
24000 LET a$=a$+a$(VAL c$)
24100 LET a$=a$+a$(VAL c$(a$))
24200 LET c$=c$ TO LEN c$-1: FO
24300 FOR n=1 TO LEN c$ IF c$(n)
24400 OR c$(n)>"9" THEN GO SUB 2500
24500 NEXT n
246
```

```

3350 IF e$="a" AND e$="f" THEN
3360 LET b=ODE e$-55
3370 LET a=a+b*16%: NEXT n: GO
3380 TO 3000
3390 IF c$(n) "a" OR c$(n) "f" T
3400 THEN
3410 RETURN
3420 GO TO 300
3430 a=0 OR a>a+55536 THEN GO
3440 TO 3500
3450 DEC: (a) A HEX. (b$)
3460 LET b$=""
3470 LET c=a
3480 LET c=COINT (c/16): LET res
3490 =c%16: IF res<10 THEN LET e$
3500 =STR$ INT res
3510 IF res>10 THEN LET e$=CHR$
3520 (res+55)
3530 b$=b$+e$
3540 LET c=coc: IF
3550 c=0 THEN GO TO 3030
3560 IF c<10 THEN LET e$=STR$ IN
3570 T c
3580 IF c>10 THEN LET e$=CHR$ (
3590 c-55)
3600 LET b$=b$+b$
3610 DEC: (a) A BIN. (a$)
3620 LET a$=""
3630 LET a=a+INT c/2: LET res=
3640 LET e$=STR$ INT res: LE
3650 T a$+e$+a$
3660 LET c=coc: IF c=2 T
3670 THEN GO TO 3320
3680 LET e$=STR$ INT c: LET a$=e
3690 +a$
3700 GO TO 4000
3710 PRINT AT 0,0 "ERROR" fuer
3720 BIN:PRINT AT 0,0 "Dec. Hexa
3730 PRINT AT 0,0 "Dec. Hexa
3740 BIN:PRINT AT 0,0 "TAB:
3750 BIN:GO TO 100

```

### Sistema hexadecimal

Necesitamos 16 dígitos para representar todas las posibles combinaciones, como solo conocemos 10 dígitos distintos (del 0 al 9), utilizaremos las 6 primeras letras mayúsculas del abecedario (de la "A" a la "F"). En la FIGURA 3 se pueden ver las 16 combinaciones posibles con 4 bits y su equivalente en hexadecimal.

Supongamos el número binario "01101100", siguiendo la tabla de la FIGURA 3, podemos escribirlo como "6Ch". Hemos escrito "6" en lugar de "0110" y "C" en lugar de "1100", la "h" se añade al final para indicar que se trata de un número hexadecimal y no confundirlo con uno decimal. A los números hexadecimales se les denomina con frecuencia, simplemente, "Hexa".

La forma de transformar un número Hexa en decimal, es

sumamente sencilla, basta con multiplicar el valor de cada dígito por 16 elevado a la correspondiente potencia (como hacíamos anteriormente para los binarios y decimales); habrá que tener en cuenta, que "A" vale 10, "B" vale 11, "C" vale 12, "D" vale 13, "E" vale 14 y "F" vale 15. Veamos algún ejemplo, vamos a convertir a decimal el número "5CB2h":

$$5CB2h = 2 \cdot 16^0 + 8 \cdot 16^1 + 0 \cdot 16^2 + 5 \cdot 16^3$$

es decir:

$$5C82h = 2 \cdot 1 + 11 \cdot 16 + 12 \cdot 256 + 5 \cdot 4096 = 23730$$

El resultado es "23730" en decimal, precisamente la dirección de la variable del Sistema "RAMTOP". Las direcciones de memoria en el

## 14 CODIGO MAQUINA

**Figura 4.**

Dec.	-dec.	Hexa	Bin.	Dec.	-dec.	Hexa	Bin.
192	-64	C0	11000000	224	-32	E0	11100000
193	-63	C1	11000001	225	-31	E1	11100001
194	-62	C2	11000010	226	-30	E2	11100010
195	-61	C3	11000011	227	-29	E3	11100011
196	-60	C4	11000100	228	-28	E4	11100100
197	-59	C5	11000101	229	-27	E5	11100101
198	-58	C6	11000110	230	-26	E6	11100110
199	-57	C7	11000111	231	-25	E7	11100111
200	-56	C8	11001000	232	-24	E8	11101000
201	-55	C9	11001001	233	-23	E9	11101001
202	-54	CA	11001010	234	-22	EA	11101010
203	-53	CB	11001011	235	-21	EB	11101011
204	-52	CC	11001100	236	-20	EC	11101100
205	-51	CD	11001101	237	-19	ED	11101101
206	-50	CE	11001110	238	-18	EE	11101110
207	-49	CF	11001111	239	-17	EF	11101111
208	-48	D0	11010000	240	-16	F0	11110000
209	-47	D1	11010001	241	-15	F1	11110001
210	-46	D2	11010010	242	-14	F2	11110010
211	-45	D3	11010011	243	-13	F3	11110011
212	-44	D4	11010100	244	-12	F4	11110100
213	-43	D5	11010101	245	-11	F5	11110101
214	-42	D6	11010110	246	-10	F6	11110110
215	-41	D7	11010111	247	-9	F7	11110111
216	-40	D8	11011000	248	-8	F8	11111000
217	-39	D9	11011001	249	-7	F9	11111001
218	-38	DA	11011010	250	-6	FA	11111010
219	-37	DB	11011011	251	-5	FB	11111011
220	-36	DC	11011100	252	-4	FC	11111100
221	-35	DD	11011101	253	-3	FD	11111101
222	-34	DE	11011110	254	-2	FE	11111110
223	-33	DF	11011111	255	-1	FF	11111111

**Figura 2G.**

$$\begin{array}{r} 23730 \overline{) 16} \\ \underline{077} \quad 1483 \overline{) 16} \\ 133 \quad 043 \overline{) 92} \overline{) 16} \\ \underline{050} \quad \underline{11} \quad \underline{12} \quad \underline{5} \quad \underline{16} \\ \underline{02} \quad \underline{5} \quad \underline{0} \end{array}$$

**Figura 2H.**

**Figura 3.**

meros de base decimal a  
cualquier base.

Este proceso es algo tedioso para realizarlo "a mano", así que normalmente usaremos un programa de ordenador, de hecho, la mayoría de los ensambladores tienen una opción que nos permite pasar números a hexadecimal, y si aún no tiene un ensamblador, puede utilizar el PROGRAMA 1 para pasar números a y desde cualquier base.

No obstante, es necesario saber cómo se realiza el proceso, entre otras cosas, para ser capaz de escribir un programa que lo haga. Como regla general, para pasar un número de base decimal a cualquier base, se divide el número por la base sin sacar decimales, el resto es el primer dígito de nuestro número (empezando por la derecha). A continuación se vuelve a dividir el cociente, y se toma el nuevo resto como el siguiente número, y así sucesivamente hasta que obtenemos un cociente de cero. Recuerde que no debe sacar decimales.

En la FIGURA 4 hemos realizado el proceso para pasar el número 23730 a Hexa, y en la FIGURA 5 hemos pasado el mismo número a binario.

Como regla general, a partir

## CODIGO MAQUINA 11



Dec.	-dec.	Hex.	Bin.
0	----	00	00000000
1	----	01	00000001
2	----	02	00000010
3	----	03	00000011
4	----	04	00000100
5	----	05	00000101
6	----	06	00000110
7	----	07	00000111
8	----	08	00001000
9	----	09	00001001
10	----	0A	00001010
11	----	0B	00001011
12	----	0C	00001100
13	----	0D	00001101
14	----	0E	00001110
15	----	0F	00001111
16	----	10	00010000
17	----	11	00010001
18	----	12	00010010
19	----	13	00010011
20	----	14	00010100
21	----	15	00010101
22	----	16	00010110
23	----	17	00010111
24	----	18	00011000
25	----	19	00011001
26	----	1A	00011010
27	----	1B	00011011
28	----	1C	00011100
29	----	1D	00011101
30	----	1E	00011110
31	----	1F	00011111

Figura 2A.

Spectrum son siempre números Hexa de 4 cifras, la razón es que existen 65536 direcciones posibles, que es el número de combinaciones que se pueden hacer con 16 bits, es decir, cuatro cifras hexadecimales.

## 12 CODIGO MAQUINA

Dec.	-dec.	Hex.	Bin.
32	----	20	00100000
33	----	21	00100001
34	----	22	00100010
35	----	23	00100011
36	----	24	00100100
37	----	25	00100101
38	----	26	00100110
39	----	27	00100111
40	----	28	00101000
41	----	29	00101001
42	----	2A	00101010
43	----	2B	00101011
44	----	2C	00101100
45	----	2D	00101101
46	----	2E	00101110
47	----	2F	00101111
48	----	30	00110000
49	----	31	00110001
50	----	32	00110010
51	----	33	00110011
52	----	34	00110100
53	----	35	00110101
54	----	36	00110110
55	----	37	00110111
56	----	38	00111000
57	----	39	00111001
58	----	3A	00111010
59	----	3B	00111011
60	----	3C	00111100
61	----	3D	00111101
62	----	3E	00111110
63	----	3F	00111111

Figura 2B.

Si contempla un mapa de memoria del Spectrum, las direcciones que definen el inicio de las distintas zonas pueden parecer algo arbitrarias; pero estos números vistos en Hexa, resultan ser "números redondos"; vamos a

Dec.	-dec.	Hex.	Bin.
64	----	40	01000000
65	----	41	01000001
66	----	42	01000010
67	----	43	01000011
68	----	44	01000100
69	----	45	01000101
70	----	46	01000110
71	----	47	01000111
72	----	48	01001000
73	----	49	01001001
74	----	4A	01001010
75	----	4B	01001011
76	----	4C	01001100
77	----	4D	01001101
78	----	4E	01001110
79	----	4F	01001111
80	----	50	01010000
81	----	51	01010001
82	----	52	01010010
83	----	53	01010011
84	----	54	01010100
85	----	55	01010101
86	----	56	01010110
87	----	57	01010111
88	----	58	01011000
89	----	59	01011001
90	----	5A	01011010
91	----	5B	01011011
92	----	5C	01011100
93	----	5D	01011101
94	----	5E	01011110
95	----	5F	01011111

Figura 2C.

comprobarlo: 16384 (el principio de la RAM) es 4000h en Hexa, 65535 (el final de la RAM) es FFFFh, 1024 (1 K) es 0400h, 16 K es 4000h, 32 K es 8000h, 48 K es C000h y finalmente, 64 K es 10000h. El archivo de pantalla ocupa des-

Dec.	-dec.	Hex.	Bin.
96	----	60	01100000
97	----	61	01100001
98	----	62	01100010
99	----	63	01100011
100	----	64	01100100
101	----	65	01100101
102	----	66	01100110
103	----	67	01100111
104	----	68	01101000
105	----	69	01101001
106	----	6A	01101010
107	----	6B	01101011
108	----	6C	01101100
109	----	6D	01101101
110	----	6E	01101110
111	----	6F	01101111
112	----	70	01110000
113	----	71	01110001
114	----	72	01110010
115	----	73	01110011
116	----	74	01110100
117	----	75	01110101
118	----	76	01110110
119	----	77	01110111
120	----	78	01111000
121	----	79	01111001
122	----	7A	01111010
123	----	7B	01111011
124	----	7C	01111100
125	----	7D	01111101
126	----	7E	01111110
127	----	7F	01111111

Figura 2D.

de 4000 hasta 5800h, el de atributos desde 5800h hasta 5B00h, el buffer de impresora va desde 5B00 hasta 5C00h, la pantalla ocupa 1800h bytes, los atributos 300 bytes y el buffer de impresora 100h

Dec.	-dec.	Hex.	Bin.
128	-128	80	10000000
129	-127	81	10000001
130	-126	82	10000010
131	-125	83	10000011
132	-124	84	10000100
133	-123	85	10000101
134	-122	86	10000110
135	-121	87	10000111
136	-120	88	10001000
137	-119	89	10001001
138	-118	8A	10001010
139	-117	8B	10001011
140	-116	8C	10001100
141	-115	8D	10001101
142	-114	8E	10001110
143	-113	8F	10001111
144	-112	90	10010000
145	-111	91	10010001
146	-110	92	10010010
147	-109	93	10010011
148	-108	94	10010100
149	-107	95	10010101
150	-106	96	10010110
151	-105	97	10010111
152	-104	98	10011000
153	-103	99	10011001
154	-102	9A	10011010
155	-101	9B	10011011
156	-100	9C	10011100
157	-99	9D	10011101
158	-98	9E	10011110
159	-97	9F	10011111

Figura 2E.

bytes.  
Esto quiere decir que si hemos de trabajar directamente sobre la memoria, es preferible que nos vayamos acostumbrando a contar en hexadecimal.

Dec.	-dec.	Hex.	Bin.
160	-96	A0	10100000
161	-95	A1	10100001
162	-94	A2	10100010
163	-93	A3	10100011
164	-92	A4	10100100
165	-91	A5	10100101
166	-90	A6	10100110
167	-89	A7	10100111
168	-88	A8	10101000
169	-87	A9	10101001
170	-86	AA	10101010
171	-85	AB	10101011
172	-84	AC	10101100
173	-83	AD	10101101
174	-82	AE	10101110
175	-81	AF	10101111
176	-80	B0	10110000
177	-79	B1	10110001
178	-78	B2	10110010
179	-77	B3	10110011
180	-76	B4	10110100
181	-75	B5	10110101
182	-74	B6	10110110
183	-73	B7	10110111
184	-72	B8	10111000
185	-71	B9	10111001
186	-70	BA	10111010
187	-69	BB	10111011
188	-68	BC	10111100
189	-67	BD	10111101
190	-66	BE	10111110
191	-65	BF	10111111

Figura 2F.

**Conversión entre bases**  
Hasta ahora hemos visto cómo pasar números desde cualquier base a decimal; ahora vamos a estudiar el proceso inverso, pasar número

## CODIGO MAQUINA 13



litada, y el microprocesador decide aceptarla, podrá responder de tres modos distintos. Estos tres modos de interrupción, también se seleccionan por software, median las instrucciones: "IM0", "IM1" e "IM2" (Interrupt Mode 0, 1 y 2). Estos modos de respuesta se denominan respectivamente: MODO 0, MODO 1 y MODO 2.

#### MODO 0

En este modo de interrupción, el microprocesador deja libre el bus de datos para permitir que el dispositivo que ha solicitado la interrupción, inserte el código de operación correspondiente a una instrucción que será ejecutada seguidamente por el microprocesador.

En el Spectrum, este modo de interrupción es redundante, ya que si lo seleccionamos, cuando se vaya a ejecutar, no habrá ningún dispositivo que inserte ningún código de operación, por lo que el bus de datos contendrá "FF", que es precisamente el código de operación de "RST 38", instrucción que obliga al microprocesador a saltar a la posición de memoria 0038h, que es, como veremos ahora, lo que hace en el MODO 1.

#### MODO 1

En este modo de interrupción, el microprocesador responde a la interrupción, simplemente, saltando a la posición de memoria 0038h. En este caso se dice que el vector de interrupción es fijo.

En el Spectrum, se trabaja normalmente en MODO 1, ya que a partir de la posición de memoria 0038h se encuentra la rutina que lee el teclado.

#### MODO 2

Es el modo de interrupción más complejo del Z-80, y el que deberemos utilizar para nuestros fines. En este caso, el microprocesador responde de una forma bastante compleja que conviene analizar detenidamente: primero coje el contenido del registro superior de una dirección, el octeto inferior deberá suministrarlo el dispositivo que ha solicitado la interrupción (si no lo suministra, se entiende que es FFh). Acto seguido, lee el número almacenado en esa dirección y la siguiente, lo carga en el "PC", y continúa la ejecución desde ese punto.

Este modo de interrupción permite un salto indirecto a cualquier posición de memoria, hay que tener en cuenta que en el Spectrum el octeto de orden bajo de la dirección será siempre FFh y por tanto, la lectura de la dirección a la que hay que saltar se producirá desde una posición de memoria cuya dirección sea xxFFh, siendo xx el contenido del registro "I", que por motivos evidentes, se denomina: **Vector de página de interrupción**.

Teniendo en cuenta que en el Spectrum se produce una petición de interrupción enmascarable cada 20 milisegundos, podemos desactivar la interrupción (con "DI") para que nuestros programas corran más deprisa, o bien, utilizar la instrucción HALT para sincronizarlos con el barrido de la pantalla. Otra posibilidad es cambiar a MODO 2 y utilizar un vector de interrupción que salte a una rutina nuestra, con lo que ésta se ejecutará 50 veces por segundo.

#### Palabra de datos del Z-80

Como hemos visto anteriormente, el Z-80 es un microprocesador de 8 bits, esto quiere decir que cada vez que accede a la memoria, lee un octeto completo, que puede ser un código de operación o un dato.

Un octeto puede almacenar 256 números distintos (2 elevado a 8) pero el Z-80 tiene más de 256 instrucciones diferentes, por lo que algunos códigos de operación ocupan más de un byte. Por otro lado, en un gran número de instrucciones, el operando se ensambla como uno o varios bytes que siguen al código de operación. En la FIGURA 5 se pueden ver los distintos formatos de instrucción del Z-80.

#### Ciclos o tiempos

Para realizar las operaciones secuencialmente, el Z-80 necesita sincronizar todas sus señales internas y externas y disponer, por tanto, de un patrón de tiempo. Es lo que se denomina: *Reloj del microprocesador*.

El reloj del microprocesador está constituido por un oscilador electrónico controlado por un cristal de cuarzo, que entrega tres millones y medio de impulsos por segundo (3.5 MHz). Estos impulsos se introducen en el Z-80 a través de la patilla 6 denominada "RELOJ", y el microprocesado utiliza un número determinado de estos impulsos para cada operación.

La primera versión del Z-80 no aceptaba señales de reloj superiores a 2.5 MHz.

## EL MICROPROCESADOR Z-80

En el primer capítulo de este curso, ideamos un microprocesador imaginario con el fin de introducir fácilmente al lector en el código máquina. Ahora, vamos a explicar a

fondo lo que es un microprocesador y más concretamente, el microprocesador Z-80 de ZILOG, que es el que equipara el Spectrum, y el que utilizaremos en este curso.

#### Qué es un microprocesador

Un ordenador es una máquina fundamentalmente secuencial. Esto quiere decir

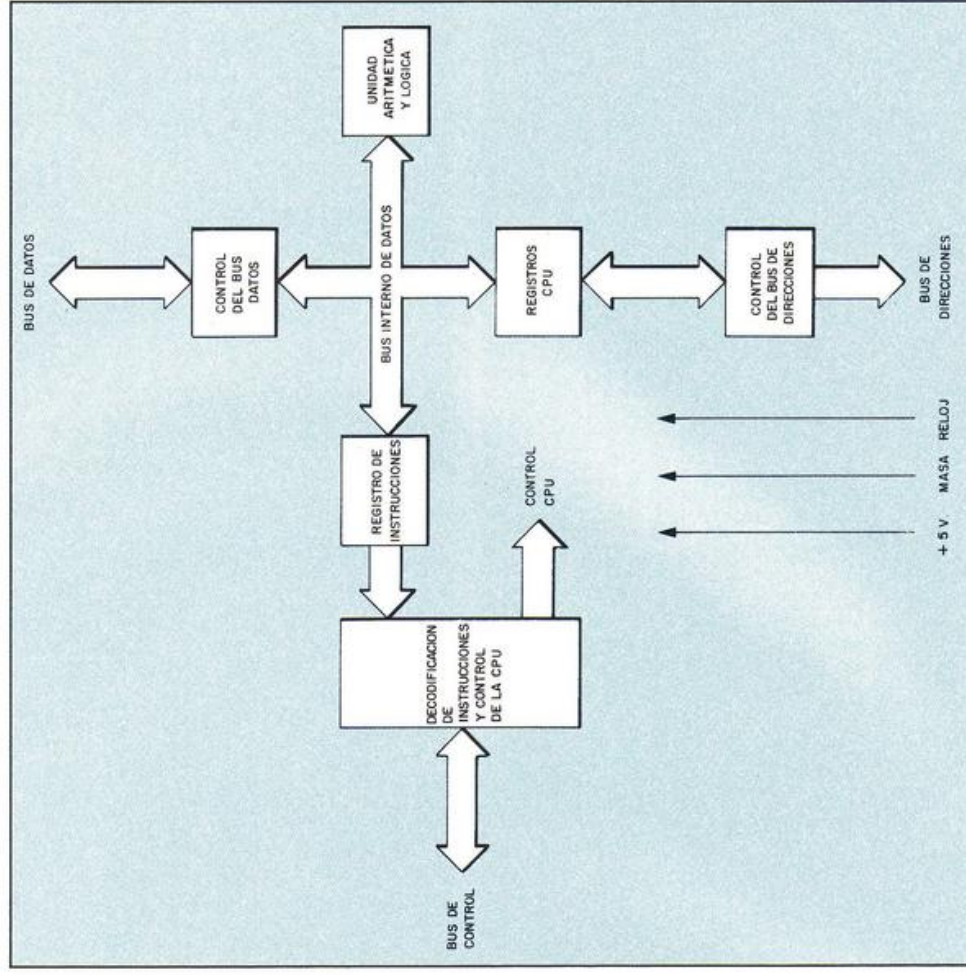


Figura 1. Diagrama de bloques del Z-80.



que realiza sus tareas una detrás de otra, siguiendo el orden en el que están las instrucciones para realizarlas.

Sus componentes básicos serán, por tanto, un lugar donde almacenar las instrucciones y datos (*Memoria*) y un elemento encargado de ejecutar los procesos indicados por esas instrucciones (*Unidad Central de Proceso o CPU*).

La CPU debe incluir todos los componentes necesarios para leer la memoria, decodificar las instrucciones y ejecutar cálculos aritméticos y lógicos. En los ordenadores de pequeño tamaño (minis y micros), la CPU está integrada dentro de un solo chip de silicio, a este chip se le conoce por el nombre de *microprocesador*.

Un microprocesador cons-

ta, normalmente, de una serie de registros, una *Unidad Aritmética-Lógica (ALU)* y los circuitos de control para la comunicación interna y externa. En la FIGURA 1 se puede ver el diagrama de bloques del microprocesador Z-80.

## Registros

Los registros constituyen una especie de pequeña memoria interna al microprocesador. El Z-80 tiene registros de 8 y de 16 bits, si bien los de 8 bits se pueden agrupar de 2 en 2 para formar uno de 16 bits. Todas las operaciones que realiza el Z-80 se hacen entre números contenidos en los registros, o bien, entre un registro y una posición de memoria; por eso se dice que el Z-80 es un microprocesador.

dor orientado hacia los registros. La posibilidad de agrupar dos registros de 8 bits para formar uno de 16, permite al Z-80 realizar operaciones de 16 bits a pesar de ser un microprocesador de 8 bits.

El Z-80 tiene, en total, 18 registros de 8 bits y 4 registros de 16 bits. Algunos son de uso general y otros tienen asignadas funciones específicas.

Como se ve, los registros cumplen en Código Máquina una función similar a la de las variables en Basic. La configuración de registros del Z-80 se muestra en la FIGURA 2.

## Registros especiales de 16 bits

Los cuatro registros especiales de 16 bits son: el Con-

barrer la pantalla del televisor. Esta interrupción se utiliza normalmente para leer el teclado, pero es posible utilizarla en nuestras propias rutinas para sincronizar el funcionamiento de nuestros programas con el barrido de la pantalla, lo que puede ser útil en caso de animación de figuras.

## Entrada NMI (Non Maskable Interrupt)

Petición de interrupción no enmascarable, esta interrupción se acepta siempre (salvo que haya presente una señal en BUSRQ) y obliga al microprocesador a saltar a la dirección 0066h independientemente del estado del flip/flop de aceptación y del modo de interrupción seleccionado.

## Entrada RESET

Esta entrada obliga al microprocesador a inicializarse, cargando todos los registros con "cero", incluido el "PC", por lo que la ejecución comienza desde la posición de memoria "0000".

En el Spectrum, esta señal se produce cada vez que se conecta el ordenador, o cada vez que se pulsa el botón de RESET en el Plus.

## Entrada BUSRQ (Bus Request)

Constituye una señal de petición de bus, al recibirla, el microprocesador responde activando la línea BUSAK y desconectándose de los buses de direcciones y datos para permitir el acceso directo a memoria de un dispositivo más rápido que él. Durante este tiempo, el microprocesador no regenera la memoria, por lo que el dispositivo que

ha hecho la petición debe encargarse de esta tarea.

## Salida BUSAK (Bus Acknowledge)

La utiliza el microprocesador para indicar el acuse de recibo a una petición de bus. Cuando se genera esta señal, el microprocesador se haya totalmente desconectado de los buses de direcciones y datos, con lo que no interfiere el acceso a memoria del dispositivo que ha pedido el bus.

## Las interrupciones en el Z-80

Cualquier microprocesador que valga el plástico que lo envuelve, tiene una posibilidad de interrumpir lo que está haciendo para atender inmediatamente a un dispositivo de alta prioridad que lo solicite, retornando a su tarea principal en el punto donde la dejó, cuando el servicio a este dispositivo haya finalizado.

El Z-80 como buen microprocesador que es, tiene varias posibilidades de interrupción que permiten el acceso con distinta prioridad.

## INTERRUPCION NO ENMASCARABLE (NMI)

Es la petición de interrupción de más alta prioridad, se acepta siempre, y se responde siempre de la misma forma: saltando a la posición de memoria 0066h.

En el Spectrum esta forma de interrupción no se utiliza, es más, se encuentra ingeniosamente anulada para facilitar la protección del software comercial; si activáramos a través del slot posterior, la línea NMI, nos encon-

tríamos con la desagradable sorpresa de que el ordenador ejecuta un salto a la posición de memoria "cero", re-inicializándose y borrando toda la memoria. Esto se debe a que la rutina de servicio a la interrupción que se encuentra a partir de 0066h salta a cero si el contenido de las posiciones de memoria 5CB0h y 5CB1h es "cero"; podemos evitar el salto a cero, almacenando un número distinto de cero en estas posiciones, pero en ese caso, se produciría un simple retorno y la interrupción sería ignorada.

## INTERRUPCION ENMASCARABLE (INT)

Se trata de la interrupción más usada en el Z-80 ya que permite definir el vector de interrupción, y lo que es más importante, decidir por software si se atiende o no la petición.

Se denomina *vector de interrupción* a la dirección de memoria a que se salta para ejecutar la rutina de servicio a la interrupción.

En el Z-80 existe un "mini-registro" de un solo bit que se denomina *flip/flop de aceptación de interrupción*. Si este registro está a "1", la petición de interrupción es aceptada, y si está a "0" es ignorada. Cuando el flip/flop de aceptación está a "0", se dice que la interrupción está *enmascarada*.

Existen dos instrucciones en el Z-80 que nos permiten enmascarar o habilitar la interrupción, estas instrucciones son: "DI" (Disable Interrupt) y "EI" (Enable Interrupt), se verán detalladamente cuando se estudien las instrucciones de control de la CPU.

Si la interrupción está habi-

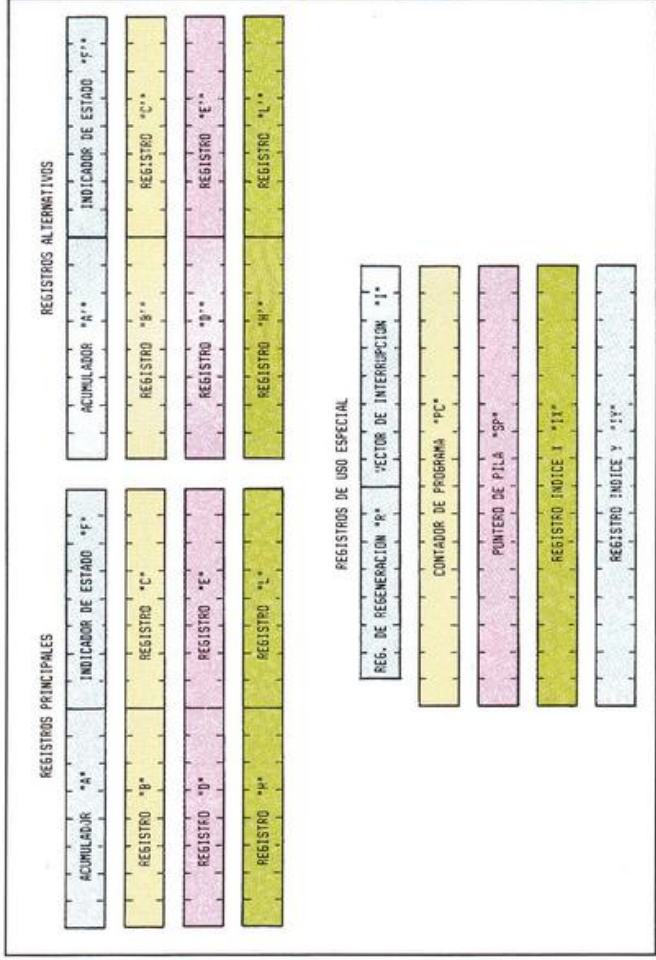


Figura 2. Registros del Z-80.



bits formado por 16 líneas eléctricas denominadas  $A_0 - A_{15}$ .

Bus de datos de 8 bits, formado por 8 líneas eléctricas denominadas  $D_0 - D_7$ .

Bus de control de 13 bits, formado por 13 líneas eléctricas denominadas:  $M1$ ,  $MREQ$ ,  $IORQ$ ,  $RD$ ,  $WR$ ,  $RFSH$ ,  $HALT$ ,  $WAIT$ ,  $INT$ ,  $NMI$ ,  $RESET$ ,  $BUSRQ$  y  $BUSAK$ .

Las tres patillas restantes hasta las 40 del chip son: la entrada de la señal de "reloj" (3.500.000 impulsos por segundo), la entrada de alimentación eléctrica (+5 voltios) y la conexión común a MASA.

Se dice que una entrada o salida está a nivel alto ("1") cuando su tensión con respecto a MASA es de +5V. Y se dice que está a nivel bajo ("0") cuando su tensión con respecto a MASA es de 0V.

Cuando el nombre de una línea tiene una raya encima, indica que es activa a nivel bajo, si no, se considera activa a nivel alto. Todas las salidas del Z-80 son *Triestado*, esto quiere decir que cuando no se están utilizando permanecen en un estado de alta impedancia que tiene el mismo efecto que si estuvieran desconectadas del circuito. A continuación veremos una a una todas las señales eléctricas del Z-80, representadas en la FIGURA 4.

$A_0 - A_{15}$ :

Constituyen un bus de direcciones que permite acceder a 65536 posiciones de memorias, o a 256 ports de entrada/salida. En las operaciones de entrada/salida, los ports se direccionan con los ocho bits inferiores del bus. Durante el tiempo de regeneración de memoria, los siete

bits inferiores contienen una dirección de regeneración.

$D_0 - D_7$ :

Constituyen un bus de datos bidireccional que permite al microprocesador tanto enviar datos como recibirlos. Se utiliza para el intercambio de datos con la memoria o con dispositivos de entrada/salida (ports).

Salida  $M1$  (Machine 1)

Se utiliza para indicar que el ciclo de máquina en curso es el de búsqueda de instrucción. También se activa junto con  $IORQ$  para indicar un acuse de recibo a una petición de interrupción.

Salida  $MREQ$  (Memory Request)

Se utiliza para indicar que el microprocesador desea acceder a la memoria.

Salida  $IORQ$  (Input/Output Request)

Se utiliza para indicar que el microprocesador desea acceder a un port de entrada/salida. También se utiliza junto con  $M1$  para indicar un acuse de recibo a una petición de interrupción.

Salida  $RD$  (Read)

Se utiliza para indicar que se desea leer una posición de memoria o un port de entrada/salida.

Salida  $WR$  (Write)

Se utiliza para indicar que se desea escribir en una posición de memoria o en un port de entrada/salida.

Salida  $RFSH$  (Refresh)

tador de programa (PC), el Puntero de pila (SP), el registro índice "X" (IX) y el registro índice "Y" (IY). A continuación vamos a verlos de uno en uno.

CONTADOR DE PROGRAMA (Program Counter "PC"):

Es el registro que contiene la dirección de memoria desde donde hay que leer la instrucción en curso, tras la ejecución de la instrucción el registro se incrementa para continuar con la siguiente, o se sustituye su valor por otro si se ha de ejecutar un salto o una llamada a subrutina. En el momento de conectar el ordenador, la señal de RESET pone este registro a "cero", por lo que la ejecución comienza desde la primera dirección de memoria.

PUNTERO DE PILA (Stack Pointer "SP"):

Una pila es una zona reservada de memoria cuyos datos están organizados como "último en entrar, primero en salir" (LIFO: Last In First Out), y sirve para almacenar determinados datos, como por ejemplo, la dirección de retorno tras una llamada a subrutina. De una pila sólo se puede recuperar cada vez el último dato que se ha introducido. El registro SP es el puntero de la pila de máquina. Apunta siempre al último dato introducido, los datos que se introducen en la pila de máquina tienen siempre dos bytes de longitud. Durante la rutina de inicialización, se carga este registro con un valor (inmediatamente debajo de RAMTOP) y cada vez que se mete un dato en la pila, el puntero (SP) se decrementa dos veces (la pila se expande hacia abajo).

La existencia de una pila permite la ejecución de llamadas a subrutinas, cada vez que se llama a una subrutina, se introduce en la pila el contenido actual del PC, se decrementa dos veces el SP y se carga el PC con la nueva dirección de la subrutina. Para retornar, se carga el PC con el contenido superior de la pila y se incrementa dos veces el SP. Este sistema permite la anidación de subrutinas hasta el límite de la memoria disponible para la pila.

Cuando se escribe un programa, hay que tener sumo cuidado para que la pila no crezca indefinidamente, ya que destruiría todos los datos almacenados en memoria, incluido el propio programa. Por otro lado, hay que tener cuidado para recuperar de la pila todos los datos almacenados durante una subrutina antes de intentar retornar, ya que de lo contrario habríamos "corrompido" la pila y el retorno no sería posible.

Nuestros programas en C/M se llaman desde Basic como una subrutina (con la función  $USR$ ) de forma que para retornar a Basic mediante una instrucción RET, debemos tener la pila en perfectas condiciones. No obstante, el Sistema Operativo del Spectrum permite un retorno a Basic, incluso con la pila "corrompida", mediante el uso de la instrucción RST 8 que se explicará más adelante.

REGISTROS INDICE

(Index X e Index Y "IX" e "IY"):

Estos registros sirven para manipular tablas, contienen las direcciones de base a las que se sumará un entero en

complemento a dos cuando se utilice direccionamiento indexado (esta forma de direccionamiento se verá más adelante).

El Sistema Operativo del Spectrum utiliza el registro IY como dirección de base para acceder a las variables del sistema, por lo que deberemos tener sumo cuidado si utilizamos este registro en nuestros programas.

## Registros especiales de 8 bits

VECTOR DE INTERRUPTACION (Interrupt "I"):

El Z-80 utiliza el dato contenido en este registro como octeto de orden alto de la dirección a la que deberá saltar cuando reciba una petición de interrupción enmascarable en "modo 2" (Las interrupciones del Z-80 se estudiarán más adelante en este mismo capítulo).

REGISTRO DE REGENERACION (Refresh "R"):

Como casi todos los lectores sabrán, el Spectrum utiliza memoria RAM dinámica, este tipo de memoria tiene que ser leída y vuelta a escribir continuamente. El Z-80 utiliza este registro de 7 bits como dirección para regenerar la memoria durante el tiempo de decodificación de cada instrucción.

## Registros alternativos

El Z-80 tiene dos grupos de 8 registros de 8 bits cada uno, que pueden ser usados de forma alternativa mediante



una instrucción de intercambio de contenidos. Cada uno de estos grupos lleva un acumulador, un indicador de estado y 6 registros de uso general.

#### ACUMULADOR (Acumulador "A"):

El Acumulador recibe los resultados de todas las operaciones aritméticas y lógicas que realiza el microprocesador que es, de hecho, el registro más usado del Z-80. Existen dos acumuladores, uno en cada grupo de registros alternativos (ver FIGURA 2) que se denominan respectivamente A y A'.

#### REGISTRO DE ESTADO (Flags "F"):

El registro de estado indica la ocurrencia de determinadas condiciones, tales como: paridad, cero, signo, acarreo, desbordamiento, que se producen tras una operación aritmética o lógica y que serán de gran utilidad en los saltos condicionales.

En la FIGURA 3 se puede ver la disposición de los distintos indicadores dentro del registro de estado.

Existen dos registros de estado, uno en cada grupo de registros alternativos, se denominan respectivamente F y F'.

#### REGISTROS DE USO GENERAL ("B", "C", "D", "E", "H" y "L"):

Cada grupo de registros alternativos tiene 6 registros de uso general que se denominan respectivamente B, C, D, E, H, L y B', C', D', E', H' y L'. Pueden agruparse de dos en dos para formar los registros: BC, DE, HL y BC', DE' y HL'.

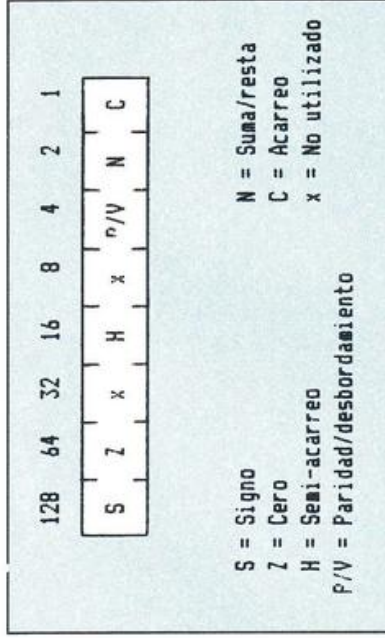


Figura 3. Indicadores de estado en el registro "F".

Una instrucción de intercambio de contenidos, permite seleccionar entre parejas de registros de uno u otro grupo.

Su aplicación es de uso general, si bien, algunos tienen funciones específicas asignadas en determinadas instrucciones, por ejemplo "HL" actúa como acumulador en las operaciones aritméticas de 16 bits, "B" actúa como contador en los bucles de iteración (instrucción DJNZ) y finalmente, en las transferencias de bloques, "HL" indica el origen, "DE" el destino y "BC" el número de bytes a transferir.

En el Sistema Operativo del Spectrum, el registro "BC" actúa como un puente de comunicación con el Basic, ya que cada vez que ejecutamos la función USR, lo que obtenemos como resultado es, precisamente, el contenido del registro "BC" en el momento de retornar, lo que nos permitirá pasar datos con facilidad desde Código Máquina a Basic.

#### Unidad Aritmética-Lógica

Otro componente fundamental del microprocesador

es la ALU o Unidad Aritmética-Lógica que es la encargada de realizar todas las operaciones en el interior del microprocesador. Las operaciones que puede realizar son:

- Desplazamiento
- Comparación
- Puesta a uno de bit
- Puesta a cero de bit
- Prueba de bit
- AND
- OR
- OR exclusiva (EXOR)
- Incremento
- Decremento
- Suma
- Resta
- Ajuste decimal

El desplazamiento consiste en una rotación, bit a bit, de un registro o una posición de memoria, puede incluir el indicador de acarreo del registro F. El efecto de rotar a la izquierda es el de multiplicar el número por 2, y el de rotarlo a la derecha es el de dividirlo por 2.

La comparación consiste en cotejar el acumulador con otro número y alterar los indicadores del registro F de acuerdo con el resultado de la comparación, permanece-

ciendo inalterado el contenido del acumulador.

Probar un bit consiste en ver si es "uno" o "cero" y anotar el resultado en el indicador de cero del registro F.

Incrementar es sumar "1", decrementar es restar "1".

La suma y la resta pueden ser con o sin acarreo.

El ajuste decimal consiste en transformar el número Hexa contenido en el acumulador y comprendido entre "00" y "FF", en un número decimal

codificado en binario (BCD) comprendido entre "00" y "99".

#### Buses

##### Registro de instrucciones

El registro de instrucciones no es accesible por el programador, se carga durante la lectura de una instrucción, con el contenido de la posición de memoria direccionada por el "PC", y retiene la instrucción hasta que es decodificada por el microprocesador.

Para comunicarse con la memoria y los periféricos, el Z-80 utiliza una serie de líneas eléctricas denominadas BUSES. Cada una de estas líneas se corresponde con una patilla del chip Z-80. Existen tres buses:

Bus de direcciones de 16

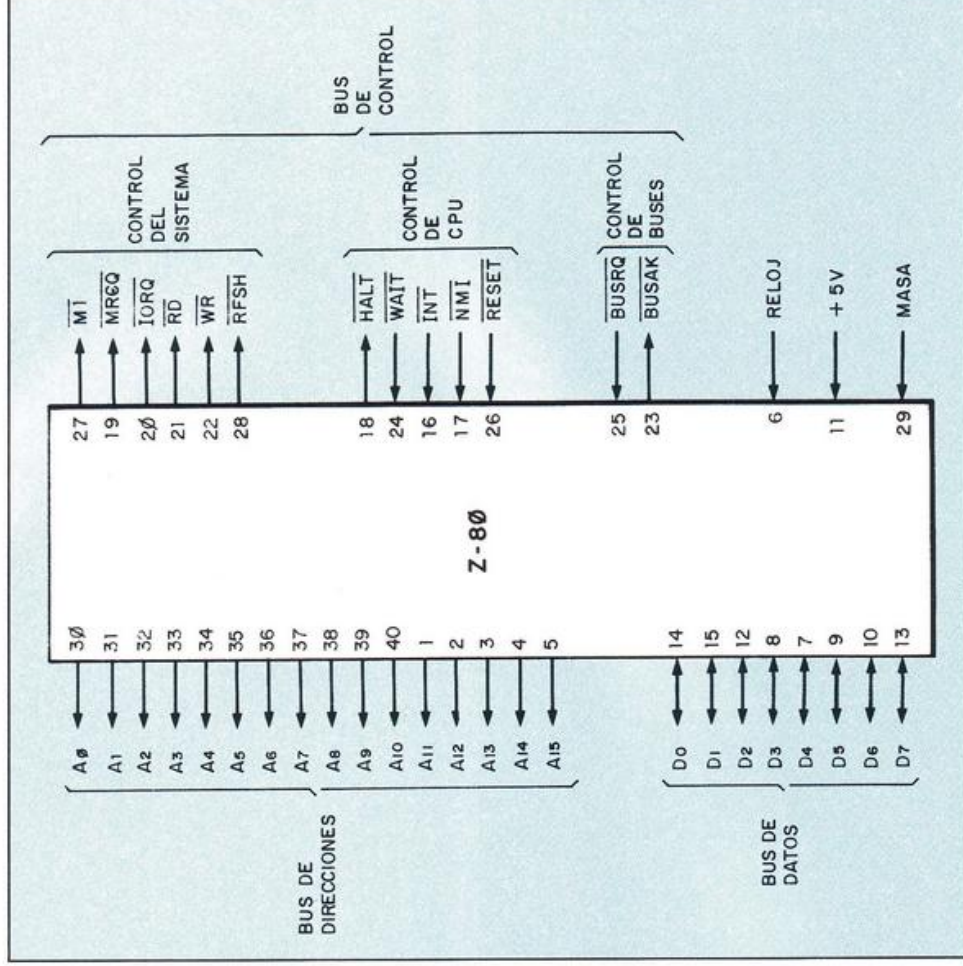


Figura 4. Configuración de patillas del Z-80.



CODIGO OPERACION
0 0 1 1 1 0 1 1
0 1 1 1 1 1 1 1

38h  
7fh

Existe una combinación diferente de bits para cada instrucción en el código de operación, a pesar de que la única diferencia sea el registro usado. Por lo tanto, habrá una parte fija y otra variable en el código de operación en función de dichos registros.

Otra cosa muy curiosa es ver cómo el microprocesador va leyendo una serie de octetos llenos de instrucciones cuando estos pueden tener cuatro longitudes distintas. Es muy sencillo. Se recordará que el microprocesador tiene un registro PC que indica la dirección de la memoria donde está la próxima instrucción a ejecutar. Cuando la lee, en función del código de operación, sabe cuantos octetos ocupa; en ese momento sólo tiene que incrementar el registro PC en esa cantidad, mientras va leyendo los restantes octetos de la instrucción, para que cuando vaya a leer la siguiente instrucción, este registro ya la tenga apuntada. Ver FIGURA 4-2.

### Necesidad de conocer el código máquina

Podríamos preguntarnos: voy a escribir mi programa en el lenguaje simbólico ASSEMBLER, tengo un ensamblador magnífico que me lo va a traducir, ¿qué necesidad tengo de conocer que bit tiene que estar activo y cual no?

Desde luego, el que tenga un buen ensamblador y escriba un programa de principio a

final, buena gana de estar descifrando códigos y pasándolos de binario a hexadecimal. Pero no siempre se dispone de un buen ensamblador o se quiere estar cargándolo para pocas instrucciones, con lo cual se meterían en la memoria los valores de las instrucciones y se saltaría a esa posición con lo que el microprocesador empezaría a trabajar. Además, existen algunas técnicas en programación que hacen necesario este conocimiento, por ejemplo:

1. Programas automodificables: La primera vez que pasan por una rutina pasan por una instrucción que ejecuta una operación; posteriormente, esa instrucción se modifica con lo que la próxima vez que pase ejecuta una operación distinta.

2. Parches: Durante la prueba del programa se encuentra un error, conociendo las posiciones de memoria donde está cargado el programa se puede modificar directamente y seguir la prueba, incluso si la modificación ocupa más espacio, se deja una zona del programa definida a ceros binarios, se hace sobre ella la modificación terminando con un retorno y desde la posición de error, se hace una llamada a esa rutina.

Por otro lado, no es necesario tener un ensamblador para poder utilizar el Spectrum en código máquina; es posible escribir el programa en Assembler y traducirlo a código máquina, manualmente; si el programa no es muy largo, la labor no resulta excesivamente tediosa. De hecho, es preferible aprender

primero a ensamblar «a mano», ya que esto proporciona un conocimiento más profundo del Assembler, y facilita la utilización posterior de un programa ensamblador.

En este mismo curso, describimos en detalle la utilización del que consideramos el mejor ensamblador que se ha escrito para el Spectrum, el «GENS 3», que desde ahora, recomendamos a nuestros lectores; mientras tanto, indicaremos con todo detalle la forma de ensamblar «a mano» cada instrucción y todos los ejemplos que demos, se podrán introducir en el ordenador sin necesidad de ensamblador, por lo que para seguir este curso, sólo es necesario disponer de un Spectrum.

### Formatos de instrucción en lenguaje simbólico

Como se recordará, el lenguaje simbólico es aquel en el que escribimos el programa fuente. Los códigos nemotécnicos que se han utilizado para el microprocesador Z-80, son abreviaturas de las palabras inglesas que definen la operación que realizan.

El formato de la instrucción y sus normas de sintaxis pueden variar algo en función del ensamblador elegido, pero siempre hay unas reglas mínimas que suelen cumplir y a esas nos referiremos hasta el capítulo que trate más profundamente el ensamblador. El formato normal es el siguiente:

ETIQUETA NEMOTECNICA  
OPERANDOS : COMENTARIOS

En el Spectrum se ha utilizado una versión más moderna denominada Z-80A, que admite señales de reloj de hasta 4 MHz, con lo que se consigue una mayor velocidad de ejecución.

En el Spectrum se ha utilizado una señal de reloj de 3.5 MHz en vez de los 4 tolerados, para evitar llevar al microprocesador al límite de su frecuencia de trabajo, lo que podría dar lugar a errores.

### CICLOS DE MAQUINA Y CICLOS DE INSTRUCCION

Se denomina *Ciclo de instrucción* al tiempo durante el cual el microprocesador ejecuta una instrucción completa.

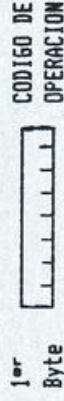
El ciclo de instrucción se subdivide a su vez, en ciclos de máquina. Un ciclo de máquina es el tiempo durante el cual el microprocesador realiza una operación elemental. Cada ciclo de máquina emplea varios ciclos (impulsos) de reloj.

Se denomina «M1» al ciclo de máquina correspondiente a la búsqueda del código de operación, durante el cual, la pata M1 del microprocesador se coloca a nivel bajo. El ciclo de máquina M1 ocupa 4 ciclos de reloj; un ciclo de reloj dura aproximadamente 0.29 microsegundos (millonésimas de segundo), por lo que el ciclo M1 dura 1.14 microsegundos.

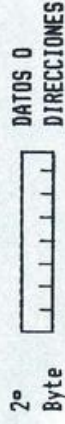
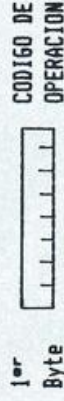
Un ciclo de memoria es una operación de lectura o escritura en memoria, emplea 3 ciclos de reloj, y dura 0.86 microsegundos.

En la FIGURA 6 se puede apreciar el cronograma (diagrama de tiempos) de una instrucción típica.

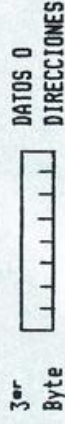
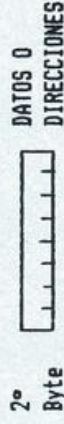
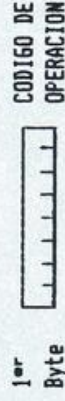
### INSTRUCCIONES DE UN SOLO OCTETO



### INSTRUCCIONES DE DOS OCTETOS



### INSTRUCCIONES DE TRES OCTETOS



### INSTRUCCIONES DE CUATRO OCTETOS

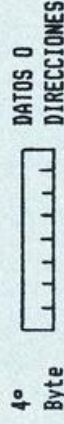
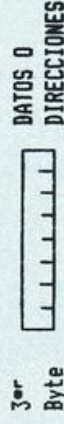
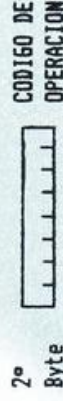
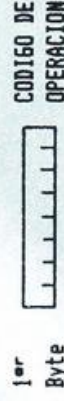


Figura 5. Formatos de instrucción del Z-80.



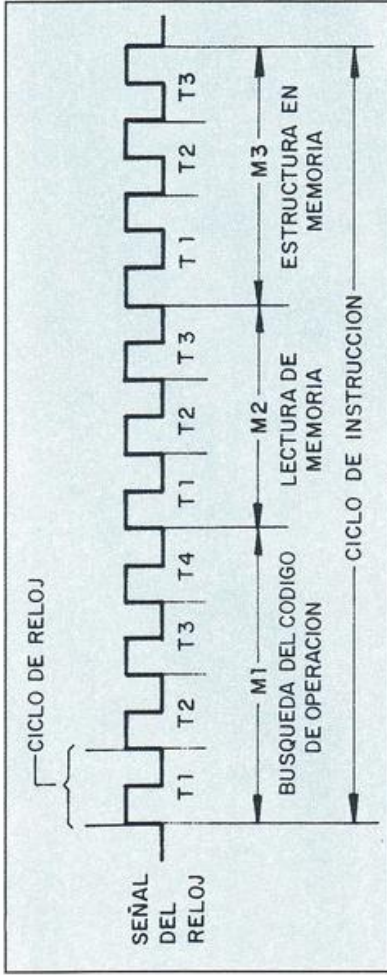


Figura 6. Cronograma de un ciclo típico de instrucción.

#### TIEMPOS DE EJECUCION

Como el lector habrá deducido ya, es posible calcular el tiempo de ejecución de una determinada rutina en C/M, a condición de conocer el número de ciclos de reloj que emplea cada una de sus instrucciones.

En lo sucesivo, cada vez que veamos una determinada instrucción, indicaremos el número de ciclos de reloj que emplea el microprocesador para ejecutarla, así como el número de veces que accede a memoria (ciclos de memoria).

Como ejemplo, veamos lo que se tarda en cargar el registro "A" con un número. Podemos utilizar la instrucción: LD A, #FF que carga el número 255 (FFh) en el acumulador; esta instrucción accede 2 veces a memoria (2 ciclos de memoria), una para buscar el código de operación (4 ciclos de reloj) y otra para buscar el número que ha de cargar en "A" (3 ciclos de reloj); lo que hace un total de 7 ciclos de reloj, es decir, unos  $7 \times 0.29 = 2$  microsegundos. Este ejemplo ilustra la enorme velocidad de ejecución del código máquina.

Para cargar el acumulador medio millón de veces en un segundo.

Supongamos que queremos sumar en el acumulador una lista de números, y usamos el registro HL para movernos a lo largo de esa lista, el bucle podría ser:

```
BUCLE ADC A, (HL)
INC HL
JR BUCLE
```

#### Modos de direccionamiento

En la mayor parte de las operaciones, el Z-80 utiliza datos almacenados en sus registros o en posiciones de memoria. Las formas posibles de indicarle la situación de estos datos, constituyen los diversos modos de direccionamiento.

#### DIRECCIONAMIENTO INMEDIATO:

En este modo de direccionamiento, el byte que sigue al código de operación en memoria, contiene el operando.

#### CODIGO DE OPERACION

uno o dos bytes

#### OPERANDO

un byte

definiciones se deben hacer con el máximo detalle, como si se pretendiese explicar a otra persona cuál es nuestro problema y cómo nos gustaría que nos lo solucionase.

#### b) Viabilidad de resolver el problema con un ordenador:

Lo que en informática profesional se llama *análisis técnico u orgánico*. En este paso se valorará la posibilidad de resolver el problema con el ordenador de que se dispone, así como de la periferia a nuestro alcance (impresora, microdrive, etc.). Se definirá toda la información del paso anterior como datos entendibles para el ordenador, indicando en qué código irán, cuánta memoria ocuparán, en qué lugar han de almacenarse y cómo se procesarán; lo mismo para los diseños de pantalla. En este paso es donde se debe decidir qué procesos se hacen por medio de subrutinas, y, como curiosidad, si el programa se hace en BASIC o en ASSEMBLER.

#### c) Realización del programa:

Es decir, la programación propiamente dicha. Este paso se divide en dos partes: la construcción del organigrama y la codificación. Estos dos pasos son complementarios, cuanto más se detalle el organigrama menos vueltas se da a la codificación, y viceversa.

1. El organigrama es una construcción gráfica del flujo o caminos posibles que tiene el programa.
2. La codificación es la escritura del lenguaje simbólico contemplando todas las alternativas posibles definidas en el organigrama.

Por supuesto, hay muchos programadores que nunca hacen un organigrama, pero

también hay muchos que se olvidan de codificar una rutina. En cualquier caso, para un programa sencillo o bien para un programador experto el organigrama puede no ser muy detallado o bien omitirse.

#### d) Prueba del programa:

En este paso se preparan unos datos de entrada al programa de los cuales ya se conocen los resultados que obtienen, contemplando, si es posible, todos los casos. Por ejemplo, si se hace un programa para resolver raíces cuadradas, se introducirán una serie de valores de los que ya se conoce el resultado.

e) Documentación: Una vez concluidos los pasos anteriores, se reúne todo el material e incluso se comentan los problemas o dificultades encontrados y cómo se han solucionado. Dependerá de lo ordenado que sea cada cual el obtener como resultado una buena documentación. Ver FIGURA 4-1.

Todo lo anteriormente descrito se puede hacer o no, pero son métodos ya muy estudiados que producen unos buenos resultados. La metodología anteriormente descrita es una orientación de la más elemental; se pueden usar técnicas más complicadas pero que se salen un poco de lo que es un microordenador. Se dice que en un programa vale todo, siempre y cuando funcione. Si existe un método que lleve a un buen resultado, no sólo que funcione, sino que sea lo más rápido posible, que no se repitan procesos, que ocupe la menor cantidad de memoria y que esté claro, ¿por qué no usarlo?

## Formatos de instrucción en código máquina

La memoria del SPECTRUM está dividida en octetos. Un octeto es una agrupación de 8 bits y un bit es la unidad más pequeña de información; sólo puede informar de dos estados: o está activo o no. Cuando de un SPECTRUM se dice que tiene 16K indica que tiene, aproximadamente, 16.000 octetos de memoria disponible, que son exactamente en decimal 16.384; para uno de 48K se multiplica esta cantidad por 3.

Los 48K de memoria disponible en RAM más los 16K que usa el programa monitor en ROM, suman 64K, que son 65.536 octetos. Esto es, desde la posición 0 de memoria a la 65.535, que es en hexadecimal FFFh y en binario 111111111111111b y a su vez la máxima cantidad que se puede escribir en dos octetos.

Las instrucciones en el SPECTRUM pueden ocupar 1, 2, 3 ó 4 octetos en función del código de operación y básicamente, tiene los formatos que se ven en la FIGURA 5 del CAPÍTULO 3.

Como se ve, lo que es común a todos los formatos es que los primeros tienen el código de operación y los últimos el operando. Sobre el operando hay que tener en cuenta que cuando son dos octetos el microprocesador espera encontrar el octeto de orden inferior en el primer octeto del operando, y el de orden superior en el segundo, de tal forma que al referirse al valor de operando 7F3Bh el ensamblador almacenará la instrucción en memoria de la siguiente manera:



# PROGRAMACION EN ASSEMBLER

En este capítulo se está ya en condiciones de saber qué es la programación en ASSEMBLER; esto es, escribir una serie de códigos entendibles por el usuario que posteriormente serán convertidos en código de máquina entendible por el microprocesador, en este caso el Z-80.

La programación en ASSEMBLER requiere cuidados especiales si se desea sacar el máximo rendimiento, por ejemplo, ante dos instrucciones que obtengan el mismo resultado se debe elegir aquella que tenga menos ciclos de máquina o de reloj, o aquella que ocupe menos posiciones de memoria; incluso en algunos casos habrá que elegir entre ocupar menos posiciones o ser más rápidos, en función de las necesidades que se tengan. Esto no quiere decir que sea necesario conocer de memoria los ciclos de cada instrucción; un manual de ASSEMBLER debe contener toda la información necesaria, con un método de acceso fácil, a pesar de que en algún caso resulte redundante. Se pretende que este curso, además de como tal, pueda servir en el futuro como un manual de consulta rápida, por lo que en algunos casos, es posible que el lector encuentre información reiterada.

Otra buena costumbre cuando se programa en ASSEMBLER es poner comentarios; siempre hay una manera de ponerlos en cada instrucción.

ción o intercalados entre ellas. Los comentarios sólo ocupan lugar en el código simbólico o programa fuente; cualquier ensamblador los ignora cuando convierte el programa en código de máquina, lo cual quiere decir que no ocupará más un programa de cada programa, documentación que nos será muy útil en el futuro.

Existen una serie de pasos que se deben seguir para dar por definitiva la realización de un programa. Cuanto más detallada sea la organización de estos pasos, más fácil será la realización del siguiente. El concepto programa abarca a todo lo que se puede realizar en una corriente de instrucciones dentro de un ordenador, por lo cual, tan programa es un juego, como llevar la contabilidad de una casa o la solución de problemas científicos; la metodología a seguir es la misma.

PASOS A SEGUIR:

- Planteamiento del programa:** Lo que en la informática profesional se llama *análisis funcional*. Es la definición de lo que se quiere realizar, la información de que se dispone, la información que se quiere obtener, la información que se quiere volver a utilizar, los formatos de pantalla deseados; todas estas

gunas orientaciones al final de este capítulo. Toda esta información, más los diseños que se hagan de pantalla, de campos o de tablas, se guardarán juntas en una carpeta o cosidas con grapa y se seguirá una buena documentación de cada programa, documentación que nos será muy útil en el futuro.

## Realización de un programa

Existen una serie de pasos que se deben seguir para dar por definitiva la realización de un programa. Cuanto más detallada sea la organización de estos pasos, más fácil será la realización del siguiente. El concepto programa abarca a todo lo que se puede realizar en una corriente de instrucciones dentro de un ordenador, por lo cual, tan programa es un juego, como llevar la contabilidad de una casa o la solución de problemas científicos; la metodología a seguir es la misma.

PASOS A SEGUIR:

- Planteamiento del programa:** Lo que en la informática profesional se llama *análisis funcional*. Es la definición de lo que se quiere realizar, la información de que se dispone, la información que se quiere obtener, la información que se quiere volver a utilizar, los formatos de pantalla deseados; todas estas

Un ejemplo podría ser cargar el acumulador con una constante, donde la constante es el byte que sigue al código de operación.

## DIRECCIONAMIENTO INMEDIATO EXTENDIDO:

Es igual que el anterior, salvo que el operando ocupa dos bytes, el primer byte es el octeto de orden bajo, y el segundo, el de orden alto.

### CODIGO DE OPERACION

uno o dos bytes

### OPERANDO (LSB)

Octeto de menos peso

### OPERANDO (MSB)

Octeto de más peso

Un ejemplo podría ser la carga de un registro doble con una constante que, lógicamente, ocuparía dos bytes de memoria.

## DIRECCIONAMIENTO RAPIDO DE PAGINA CERO

El Z-80 tiene unas instrucciones de salto rápido a una dirección de página cero. Hay ocho direcciones posibles donde se colocan las rutinas de uso más frecuente, de esta forma se puede llamar a estas rutinas empleando un solo byte.

En el Spectrum, estas direcciones se encuentran utilizadas por la ROM para las rutinas de más uso, y son las siguientes:

RST 00h : Inicialización  
RST 08h : Salida de error  
RST 10h : Imprimir un carácter  
RST 18h : Leer un carácter

RST 20h : Leer el siguiente carácter  
RST 28h : Entrada al calculador  
RST 30h : Hacer espacio en memoria  
RST 38h : Leer el teclado

## DIRECCIONAMIENTO RELATIVO

En este caso, el byte que sigue al código de operación se emplea como un entero en suma a la dirección actual almacenada en el «PC».

### CODIGO DE OPERACION

un byte (salto relativo)

### OPERANDO

entero en complemento a 2

Este modo de direccionamiento permite efectuar saltos relativos, con lo que las rutinas pueden ser *reubicables* es decir, correr de igual forma en cualquier dirección de memoria.

## DIRECCIONAMIENTO INDEXADO

En esta forma de direccionamiento, el byte de datos que sigue al código de operación contiene un entero de desplazamiento en complemento a dos, que se suma al contenido actual del registro índice correspondiente, para apuntar a una dirección de memoria. El código de operación tiene siempre dos bytes, el primer byte es «Ddh» siempre que se utilice el registro «IX» y «FDh» siempre que se utilice el «IY».

### DD o FD

índice usado

### CODIGO DE OPERACION

un byte

### DESPLAZAMIENTO

entero en compl. a dos

## DIRECCIONAMIENTO DE REGISTROS

En muchos de los códigos de operación, hay ciertos bits que especifican a qué registro se refiere la instrucción, permaneciendo inalterados el resto de los bits. Un ejemplo podría ser la instrucción:

### LD C,B

Que significa: «cargar en el registro «C» el contenido del registro «B»».

## DIRECCIONAMIENTO IMPLICITO

En este caso, la situación de los datos está implícita en el código de operación. Por ejemplo, en las operaciones aritméticas de 8 bits, el registro «A» (acumulador) es siempre el que recibe los resultados.

## DIRECCIONAMIENTO INDIRECTO

En esta forma de direccionamiento, el contenido de un registro doble se utiliza como dirección a partir de la cual hay que cargar el dato. Un ejemplo podría ser:

### LD A, [HL]

Que significa: «carga el registro A con el contenido de la dirección de memoria apuntada por el registro HL». En este caso el registro HL se utiliza como *puntero* para «apuntar» a una dirección de



memoria, siempre que un registro se utilice como puntero, su nombre aparecerá, en el código simbólico, encerrado entre paréntesis, significando: «donde apunta el contenido de».

También se puede utilizar como puntero, una constante de dos bytes, ensamblada a continuación del código objeto, por ejemplo:

**LD A, [5C37]**

Que significa: «carga el registro A con el contenido de la dirección de memoria 5C37h». Cuando hagamos estos en Assembler, normalmente utilizaremos una «etiqueta» de la siguiente forma:

**LD A, [ETIQUE]  
ETIQUE EQU #5C37**

De esta forma, solo tendremos que definir la etiqueta una vez, pero podremos usarla todas la veces que queramos sin tener que recordar de memoria los números. Los nombres de las variables del Sistema en el Spectrum son, precisamente, etiquetas del código fuente del Sistema Operativo. El uso de las etiquetas se verá en profundidad cuando estudiemos el manejo de ensambladores.

En algunos casos, el direccionamiento indirecto se utiliza para especificar operandos de 16 bits (dos bytes), en este caso, el puntero apunta al byte de menos peso, siendo el de más peso el siguiente. Por ejemplo:

**LD HL, [5C37]**

Que significa: «carga el registro L con el contenido de la

posición de memoria 5C37h, y el registro H con el contenido de la posición de memoria siguiente (5C38h)».

## DIRECCIONAMIENTO DE BITS

Un gran número de instrucciones del Z-80 trabajan directamente sobre bits individuales de registros o posiciones de memoria. En este caso, se utiliza uno de los métodos de direccionamiento anteriores para indicar el registro o posición de memoria en concreto, y tres bits del código de operación para indicar a qué bit de ese registro o posición de memoria nos referimos.

## MODOS DE DIRECCIONAMIENTO COMBINADOS

Muchas instrucciones incluyen más de un operando, en estos casos, se pueden combinar más de un modo de direccionamiento dentro de una misma instrucción, por ejemplo:

**LD [IX+7], A**

Que utiliza directamente el destino y el direccionamiento inmediato para la fuente. Significa: «carga en la posición de memoria apuntada por el contenido del registro IX más 7, el contenido del registro A (acumulador)».

## Instrucciones del Z-80

El Z-80 puede ejecutar un gran número de instrucciones, podemos ordenarlas en los siguientes grupos:

CARGA E INTERCAMBIO

Permiten desplazar datos entre registros, o entre estos y posiciones de memoria. También se puede intercambiar el contenido de dos registros, o el de dos grupos alternativos.

## ARITMETICAS Y LOGICAS

Permiten realizar operaciones aritméticas o lógicas entre el acumulador y un registro o posición de memoria. Los resultados se almacenan en el acumulador, y los indicadores del registro «F» se ponen a «1» o a «0» en función del resultado de la operación.

## BUSQUEDA Y TRANSFERENCIA DE BLOQUES

Se trata de las más poderosas instrucciones del Z-80, es posible transferir todo un bloque de memoria con una sola instrucción; también es posible examinar todo un bloque de memoria para buscar un determinado dato de un byte.

## ROTACION Y DESPLAZAMIENTO

Permiten la rotación bit a bit del dato almacenado en un registro o una posición de memoria, las rotaciones pueden incluir el indicador de acarreo del registro «F».

## MANIPULACION DE BITS

Permiten tratar de formar independiente cada bit de un registro o una posición de memoria, es posible poner un bit a «1», ponerlo a «0» o examinar si es «1» o «0».

## SALTO LLAMADA Y RETORNO

Permite alterar la secuen-

cia normal del programa para saltar a otro lugar de la memoria o ejecutar una subrutina. También es posible retornar desde una subrutina al punto donde se la llamó.

## ENTRA Y SALIDA

Permiten leer y escribir datos en los ports de entrada/salida, con lo cual se comunica el ordenador con el mundo exterior.

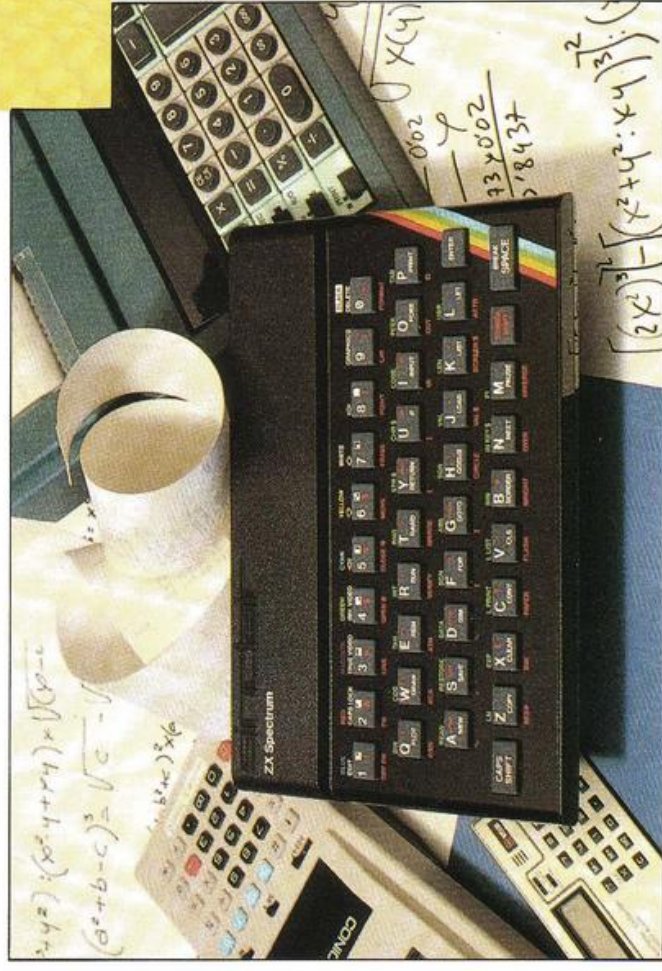
## CONTROL CPU

Se utilizan para controlar el propio funcionamiento del microprocesador, inhibir o habilitar interrupciones, cambiar el modo de interrupción, detener el funcionamiento del microprocesador, etc.

En los capítulos posteriores de este curso, se irán viendo detenidamente, una a una, todas las instrucciones

de cada uno de estos grupos y la forma de utilizarlas en nuestros programas.

Antes, en el capítulo siguiente, se verán los conceptos básicos de la programación en Assembler, y las formas de almacenar y ejecutar nuestros programas en Código Máquina.





de la primera dirección).  
Nuestro programa en Basic, empezaría por:

```
10 CLEAR 49999
```

A continuación, utilizaremos un bucle FOR...NEXT para introducir el código.

```
20 FOR N=50000 TO 50005
30 READ A:POKE N,A
40 DATA 12,65,97,80,66
50 DATA 51,15,71,30,25
```

Ahora, sólo nos queda ejecutar el programa; para ello utilizaremos la funciónUSR, que como todos saben, nos devuelve en el retorno, el contenido del par de registros BC (como regla nemotécnica, acuérdesse de "Basic Comunicador", -comunicador con el Basic-). USR, como toda función, debe ir precedida de un comando, el que utilizemos, dependerá de lo que queramos hacer con el resultado; si no nos importa el valor de BC en el retorno, podemos hacer RANDOMIZE USR... que sólo ocupa dos bytes. Si queremos imprimir el resultado, podemos hacer PRINT USR... y si queremos asignar el resultado a una variable, para luego trabajar con él, podemos hacer LET A=USR... En cualquier caso, detrás de USR deberá ir la dirección a partir de la cual se debe ejecutar nuestro programa. Sustituiremos que en nuestro ejemplo, no nos importa el resultado, así que haríamos:

```
70 RANDOMIZE USR 50005
```

Con lo que el Sistema Operativo para el control a nuestro programa en C/M, hasta que el microprocesador se encuentre una instrucción de retorno, ya que el S/O (Sistema Operativo) trata nuestro programa como si se tratase de una subrutina suya; esto se verá más claramente cuando estudiemos el capítulo dedicado a las subrutinas.

### Codificación hexadecimal

Con el procedimiento visto hasta ahora, utilizamos 10 números metidos en DATAs, para representar un programa de 10 bytes de longitud. Estas DATAs, nos ocuparán cerca de 70 bytes de memoria dentro del programa Basic; si tuviéramos que representar en DATAs un programa de 2K (2048 bytes), probablemente, no nos cabrían los DATAs en un 16K. Para evitar esta forma de malgastar la memoria, existe un procedimiento al que quizá esté acostumbrado el lector por los listados de nuestra revista, este procedimiento consiste en codificar el programa en hexadecimal, e introducirlo como una cadena de caracteres, que sólo ocupará en DATAs el doble de la longitud del programa. Veámoslo con un ejemplo:

Primero haríamos:

```
10 CLEAR 49999
```

De la misma forma que antes, pero esta vez, definiremos una función que nos ayude a decodificarlo.

```
20 DEF FN A$(N) = MID$(CODE$,N,1)
30 DEF FN B$(N) = MID$(CODE$,N+1,1)
40 DEF FN C$(N) = MID$(CODE$,N+2,1)
50 DEF FN D$(N) = MID$(CODE$,N+3,1)
60 DEF FN E$(N) = MID$(CODE$,N+4,1)
70 DEF FN F$(N) = MID$(CODE$,N+5,1)
80 DEF FN G$(N) = MID$(CODE$,N+6,1)
90 DEF FN H$(N) = MID$(CODE$,N+7,1)
100 DEF FN I$(N) = MID$(CODE$,N+8,1)
110 DEF FN J$(N) = MID$(CODE$,N+9,1)
120 DEF FN K$(N) = MID$(CODE$,N+10,1)
130 DEF FN L$(N) = MID$(CODE$,N+11,1)
140 DEF FN M$(N) = MID$(CODE$,N+12,1)
150 DEF FN N$(N) = MID$(CODE$,N+13,1)
160 DEF FN O$(N) = MID$(CODE$,N+14,1)
170 DEF FN P$(N) = MID$(CODE$,N+15,1)
180 DEF FN Q$(N) = MID$(CODE$,N+16,1)
190 DEF FN R$(N) = MID$(CODE$,N+17,1)
200 DEF FN S$(N) = MID$(CODE$,N+18,1)
210 DEF FN T$(N) = MID$(CODE$,N+19,1)
220 DEF FN U$(N) = MID$(CODE$,N+20,1)
230 DEF FN V$(N) = MID$(CODE$,N+21,1)
240 DEF FN W$(N) = MID$(CODE$,N+22,1)
250 DEF FN X$(N) = MID$(CODE$,N+23,1)
260 DEF FN Y$(N) = MID$(CODE$,N+24,1)
270 DEF FN Z$(N) = MID$(CODE$,N+25,1)
280 DEF FN AA$(N) = MID$(CODE$,N+26,1)
290 DEF FN AB$(N) = MID$(CODE$,N+27,1)
300 DEF FN AC$(N) = MID$(CODE$,N+28,1)
310 DEF FN AD$(N) = MID$(CODE$,N+29,1)
320 DEF FN AE$(N) = MID$(CODE$,N+30,1)
330 DEF FN AF$(N) = MID$(CODE$,N+31,1)
340 DEF FN AG$(N) = MID$(CODE$,N+32,1)
350 DEF FN AH$(N) = MID$(CODE$,N+33,1)
360 DEF FN AI$(N) = MID$(CODE$,N+34,1)
370 DEF FN AJ$(N) = MID$(CODE$,N+35,1)
380 DEF FN AK$(N) = MID$(CODE$,N+36,1)
390 DEF FN AL$(N) = MID$(CODE$,N+37,1)
400 DEF FN AM$(N) = MID$(CODE$,N+38,1)
410 DEF FN AN$(N) = MID$(CODE$,N+39,1)
420 DEF FN AO$(N) = MID$(CODE$,N+40,1)
430 DEF FN AP$(N) = MID$(CODE$,N+41,1)
440 DEF FN AQ$(N) = MID$(CODE$,N+42,1)
450 DEF FN AR$(N) = MID$(CODE$,N+43,1)
460 DEF FN AS$(N) = MID$(CODE$,N+44,1)
470 DEF FN AT$(N) = MID$(CODE$,N+45,1)
480 DEF FN AU$(N) = MID$(CODE$,N+46,1)
490 DEF FN AV$(N) = MID$(CODE$,N+47,1)
500 DEF FN AW$(N) = MID$(CODE$,N+48,1)
510 DEF FN AX$(N) = MID$(CODE$,N+49,1)
520 DEF FN AY$(N) = MID$(CODE$,N+50,1)
530 DEF FN AZ$(N) = MID$(CODE$,N+51,1)
540 DEF FN BA$(N) = MID$(CODE$,N+52,1)
550 DEF FN BB$(N) = MID$(CODE$,N+53,1)
560 DEF FN BC$(N) = MID$(CODE$,N+54,1)
570 DEF FN BD$(N) = MID$(CODE$,N+55,1)
580 DEF FN BE$(N) = MID$(CODE$,N+56,1)
590 DEF FN BF$(N) = MID$(CODE$,N+57,1)
600 DEF FN BG$(N) = MID$(CODE$,N+58,1)
610 DEF FN BH$(N) = MID$(CODE$,N+59,1)
620 DEF FN BI$(N) = MID$(CODE$,N+60,1)
630 DEF FN BJ$(N) = MID$(CODE$,N+61,1)
640 DEF FN BK$(N) = MID$(CODE$,N+62,1)
650 DEF FN BL$(N) = MID$(CODE$,N+63,1)
660 DEF FN BM$(N) = MID$(CODE$,N+64,1)
670 DEF FN BN$(N) = MID$(CODE$,N+65,1)
680 DEF FN BO$(N) = MID$(CODE$,N+66,1)
690 DEF FN BP$(N) = MID$(CODE$,N+67,1)
700 DEF FN BQ$(N) = MID$(CODE$,N+68,1)
710 DEF FN BR$(N) = MID$(CODE$,N+69,1)
720 DEF FN BS$(N) = MID$(CODE$,N+70,1)
730 DEF FN BT$(N) = MID$(CODE$,N+71,1)
740 DEF FN BU$(N) = MID$(CODE$,N+72,1)
750 DEF FN BV$(N) = MID$(CODE$,N+73,1)
760 DEF FN BW$(N) = MID$(CODE$,N+74,1)
770 DEF FN BX$(N) = MID$(CODE$,N+75,1)
780 DEF FN BY$(N) = MID$(CODE$,N+76,1)
790 DEF FN BZ$(N) = MID$(CODE$,N+77,1)
800 DEF FN CA$(N) = MID$(CODE$,N+78,1)
810 DEF FN CB$(N) = MID$(CODE$,N+79,1)
820 DEF FN CC$(N) = MID$(CODE$,N+80,1)
830 DEF FN CD$(N) = MID$(CODE$,N+81,1)
840 DEF FN CE$(N) = MID$(CODE$,N+82,1)
850 DEF FN CF$(N) = MID$(CODE$,N+83,1)
860 DEF FN CG$(N) = MID$(CODE$,N+84,1)
870 DEF FN CH$(N) = MID$(CODE$,N+85,1)
880 DEF FN CI$(N) = MID$(CODE$,N+86,1)
890 DEF FN CJ$(N) = MID$(CODE$,N+87,1)
900 DEF FN CK$(N) = MID$(CODE$,N+88,1)
910 DEF FN CL$(N) = MID$(CODE$,N+89,1)
920 DEF FN CM$(N) = MID$(CODE$,N+90,1)
930 DEF FN CN$(N) = MID$(CODE$,N+91,1)
940 DEF FN CO$(N) = MID$(CODE$,N+92,1)
950 DEF FN CP$(N) = MID$(CODE$,N+93,1)
960 DEF FN CQ$(N) = MID$(CODE$,N+94,1)
970 DEF FN CR$(N) = MID$(CODE$,N+95,1)
980 DEF FN CS$(N) = MID$(CODE$,N+96,1)
990 DEF FN CT$(N) = MID$(CODE$,N+97,1)
1000 DEF FN CU$(N) = MID$(CODE$,N+98,1)
1010 DEF FN CV$(N) = MID$(CODE$,N+99,1)
1020 DEF FN CW$(N) = MID$(CODE$,N+100,1)
1030 DEF FN CX$(N) = MID$(CODE$,N+101,1)
1040 DEF FN CY$(N) = MID$(CODE$,N+102,1)
1050 DEF FN CZ$(N) = MID$(CODE$,N+103,1)
1060 DEF FN DA$(N) = MID$(CODE$,N+104,1)
1070 DEF FN DB$(N) = MID$(CODE$,N+105,1)
1080 DEF FN DC$(N) = MID$(CODE$,N+106,1)
1090 DEF FN DD$(N) = MID$(CODE$,N+107,1)
1100 DEF FN DE$(N) = MID$(CODE$,N+108,1)
1110 DEF FN DF$(N) = MID$(CODE$,N+109,1)
1120 DEF FN DG$(N) = MID$(CODE$,N+110,1)
1130 DEF FN DH$(N) = MID$(CODE$,N+111,1)
1140 DEF FN DI$(N) = MID$(CODE$,N+112,1)
1150 DEF FN DJ$(N) = MID$(CODE$,N+113,1)
1160 DEF FN DK$(N) = MID$(CODE$,N+114,1)
1170 DEF FN DL$(N) = MID$(CODE$,N+115,1)
1180 DEF FN DM$(N) = MID$(CODE$,N+116,1)
1190 DEF FN DN$(N) = MID$(CODE$,N+117,1)
1200 DEF FN DO$(N) = MID$(CODE$,N+118,1)
1210 DEF FN DP$(N) = MID$(CODE$,N+119,1)
1220 DEF FN DQ$(N) = MID$(CODE$,N+120,1)
1230 DEF FN DR$(N) = MID$(CODE$,N+121,1)
1240 DEF FN DS$(N) = MID$(CODE$,N+122,1)
1250 DEF FN DT$(N) = MID$(CODE$,N+123,1)
1260 DEF FN DU$(N) = MID$(CODE$,N+124,1)
1270 DEF FN DV$(N) = MID$(CODE$,N+125,1)
1280 DEF FN DW$(N) = MID$(CODE$,N+126,1)
1290 DEF FN DX$(N) = MID$(CODE$,N+127,1)
1300 DEF FN DY$(N) = MID$(CODE$,N+128,1)
1310 DEF FN DZ$(N) = MID$(CODE$,N+129,1)
1320 DEF FN EA$(N) = MID$(CODE$,N+130,1)
1330 DEF FN EB$(N) = MID$(CODE$,N+131,1)
1340 DEF FN EC$(N) = MID$(CODE$,N+132,1)
1350 DEF FN ED$(N) = MID$(CODE$,N+133,1)
1360 DEF FN EE$(N) = MID$(CODE$,N+134,1)
1370 DEF FN EF$(N) = MID$(CODE$,N+135,1)
1380 DEF FN EG$(N) = MID$(CODE$,N+136,1)
1390 DEF FN EH$(N) = MID$(CODE$,N+137,1)
1400 DEF FN EI$(N) = MID$(CODE$,N+138,1)
1410 DEF FN EJ$(N) = MID$(CODE$,N+139,1)
1420 DEF FN EK$(N) = MID$(CODE$,N+140,1)
1430 DEF FN EL$(N) = MID$(CODE$,N+141,1)
1440 DEF FN EM$(N) = MID$(CODE$,N+142,1)
1450 DEF FN EN$(N) = MID$(CODE$,N+143,1)
1460 DEF FN EO$(N) = MID$(CODE$,N+144,1)
1470 DEF FN EP$(N) = MID$(CODE$,N+145,1)
1480 DEF FN EQ$(N) = MID$(CODE$,N+146,1)
1490 DEF FN ER$(N) = MID$(CODE$,N+147,1)
1500 DEF FN ES$(N) = MID$(CODE$,N+148,1)
1510 DEF FN ET$(N) = MID$(CODE$,N+149,1)
1520 DEF FN EU$(N) = MID$(CODE$,N+150,1)
1530 DEF FN EV$(N) = MID$(CODE$,N+151,1)
1540 DEF FN EW$(N) = MID$(CODE$,N+152,1)
1550 DEF FN EX$(N) = MID$(CODE$,N+153,1)
1560 DEF FN EY$(N) = MID$(CODE$,N+154,1)
1570 DEF FN EZ$(N) = MID$(CODE$,N+155,1)
1580 DEF FN FA$(N) = MID$(CODE$,N+156,1)
1590 DEF FN FB$(N) = MID$(CODE$,N+157,1)
1600 DEF FN FC$(N) = MID$(CODE$,N+158,1)
1610 DEF FN FD$(N) = MID$(CODE$,N+159,1)
1620 DEF FN FE$(N) = MID$(CODE$,N+160,1)
1630 DEF FN FF$(N) = MID$(CODE$,N+161,1)
1640 DEF FN FG$(N) = MID$(CODE$,N+162,1)
1650 DEF FN FH$(N) = MID$(CODE$,N+163,1)
1660 DEF FN FI$(N) = MID$(CODE$,N+164,1)
1670 DEF FN FJ$(N) = MID$(CODE$,N+165,1)
1680 DEF FN FK$(N) = MID$(CODE$,N+166,1)
1690 DEF FN FL$(N) = MID$(CODE$,N+167,1)
1700 DEF FN FM$(N) = MID$(CODE$,N+168,1)
1710 DEF FN FN$(N) = MID$(CODE$,N+169,1)
1720 DEF FN FO$(N) = MID$(CODE$,N+170,1)
1730 DEF FN FP$(N) = MID$(CODE$,N+171,1)
1740 DEF FN FQ$(N) = MID$(CODE$,N+172,1)
1750 DEF FN FR$(N) = MID$(CODE$,N+173,1)
1760 DEF FN FS$(N) = MID$(CODE$,N+174,1)
1770 DEF FN FT$(N) = MID$(CODE$,N+175,1)
1780 DEF FN FU$(N) = MID$(CODE$,N+176,1)
1790 DEF FN FV$(N) = MID$(CODE$,N+177,1)
1800 DEF FN FW$(N) = MID$(CODE$,N+178,1)
1810 DEF FN FX$(N) = MID$(CODE$,N+179,1)
1820 DEF FN FY$(N) = MID$(CODE$,N+180,1)
1830 DEF FN FZ$(N) = MID$(CODE$,N+181,1)
1840 DEF FN GA$(N) = MID$(CODE$,N+182,1)
1850 DEF FN GB$(N) = MID$(CODE$,N+183,1)
1860 DEF FN GC$(N) = MID$(CODE$,N+184,1)
1870 DEF FN GD$(N) = MID$(CODE$,N+185,1)
1880 DEF FN GE$(N) = MID$(CODE$,N+186,1)
1890 DEF FN GF$(N) = MID$(CODE$,N+187,1)
1900 DEF FN GG$(N) = MID$(CODE$,N+188,1)
1910 DEF FN GH$(N) = MID$(CODE$,N+189,1)
1920 DEF FN GI$(N) = MID$(CODE$,N+190,1)
1930 DEF FN GJ$(N) = MID$(CODE$,N+191,1)
1940 DEF FN GK$(N) = MID$(CODE$,N+192,1)
1950 DEF FN GL$(N) = MID$(CODE$,N+193,1)
1960 DEF FN GM$(N) = MID$(CODE$,N+194,1)
1970 DEF FN GN$(N) = MID$(CODE$,N+195,1)
1980 DEF FN GO$(N) = MID$(CODE$,N+196,1)
1990 DEF FN GP$(N) = MID$(CODE$,N+197,1)
2000 DEF FN GQ$(N) = MID$(CODE$,N+198,1)
2010 DEF FN GR$(N) = MID$(CODE$,N+199,1)
2020 DEF FN GS$(N) = MID$(CODE$,N+200,1)
2030 DEF FN GT$(N) = MID$(CODE$,N+201,1)
2040 DEF FN GU$(N) = MID$(CODE$,N+202,1)
2050 DEF FN GV$(N) = MID$(CODE$,N+203,1)
2060 DEF FN GW$(N) = MID$(CODE$,N+204,1)
2070 DEF FN GX$(N) = MID$(CODE$,N+205,1)
2080 DEF FN GY$(N) = MID$(CODE$,N+206,1)
2090 DEF FN GZ$(N) = MID$(CODE$,N+207,1)
2100 DEF FN HA$(N) = MID$(CODE$,N+208,1)
2110 DEF FN HB$(N) = MID$(CODE$,N+209,1)
2120 DEF FN HC$(N) = MID$(CODE$,N+210,1)
2130 DEF FN HD$(N) = MID$(CODE$,N+211,1)
2140 DEF FN HE$(N) = MID$(CODE$,N+212,1)
2150 DEF FN HF$(N) = MID$(CODE$,N+213,1)
2160 DEF FN HG$(N) = MID$(CODE$,N+214,1)
2170 DEF FN HH$(N) = MID$(CODE$,N+215,1)
2180 DEF FN HI$(N) = MID$(CODE$,N+216,1)
2190 DEF FN HJ$(N) = MID$(CODE$,N+217,1)
2200 DEF FN HK$(N) = MID$(CODE$,N+218,1)
2210 DEF FN HL$(N) = MID$(CODE$,N+219,1)
2220 DEF FN HM$(N) = MID$(CODE$,N+220,1)
2230 DEF FN HN$(N) = MID$(CODE$,N+221,1)
2240 DEF FN HO$(N) = MID$(CODE$,N+222,1)
2250 DEF FN HP$(N) = MID$(CODE$,N+223,1)
2260 DEF FN HQ$(N) = MID$(CODE$,N+224,1)
2270 DEF FN HR$(N) = MID$(CODE$,N+225,1)
2280 DEF FN HS$(N) = MID$(CODE$,N+226,1)
2290 DEF FN HT$(N) = MID$(CODE$,N+227,1)
2300 DEF FN HU$(N) = MID$(CODE$,N+228,1)
2310 DEF FN HV$(N) = MID$(CODE$,N+229,1)
2320 DEF FN HW$(N) = MID$(CODE$,N+230,1)
2330 DEF FN HX$(N) = MID$(CODE$,N+231,1)
2340 DEF FN HY$(N) = MID$(CODE$,N+232,1)
2350 DEF FN HZ$(N) = MID$(CODE$,N+233,1)
2360 DEF FN IA$(N) = MID$(CODE$,N+234,1)
2370 DEF FN IB$(N) = MID$(CODE$,N+235,1)
2380 DEF FN IC$(N) = MID$(CODE$,N+236,1)
2390 DEF FN ID$(N) = MID$(CODE$,N+237,1)
2400 DEF FN IE$(N) = MID$(CODE$,N+238,1)
2410 DEF FN IF$(N) = MID$(CODE$,N+239,1)
2420 DEF FN IG$(N) = MID$(CODE$,N+240,1)
2430 DEF FN IH$(N) = MID$(CODE$,N+241,1)
2440 DEF FN II$(N) = MID$(CODE$,N+242,1)
2450 DEF FN IJ$(N) = MID$(CODE$,N+243,1)
2460 DEF FN IK$(N) = MID$(CODE$,N+244,1)
2470 DEF FN IL$(N) = MID$(CODE$,N+245,1)
2480 DEF FN IM$(N) = MID$(CODE$,N+246,1)
2490 DEF FN IN$(N) = MID$(CODE$,N+247,1)
2500 DEF FN IO$(N) = MID$(CODE$,N+248,1)
2510 DEF FN IP$(N) = MID$(CODE$,N+249,1)
2520 DEF FN IQ$(N) = MID$(CODE$,N+250,1)
2530 DEF FN IR$(N) = MID$(CODE$,N+251,1)
2540 DEF FN IS$(N) = MID$(CODE$,N+252,1)
2550 DEF FN IT$(N) = MID$(CODE$,N+253,1)
2560 DEF FN IU$(N) = MID$(CODE$,N+254,1)
2570 DEF FN IV$(N) = MID$(CODE$,N+255,1)
2580 DEF FN IW$(N) = MID$(CODE$,N+256,1)
2590 DEF FN IX$(N) = MID$(CODE$,N+257,1)
2600 DEF FN IY$(N) = MID$(CODE$,N+258,1)
2610 DEF FN IZ$(N) = MID$(CODE$,N+259,1)
2620 DEF FN JA$(N) = MID$(CODE$,N+260,1)
2630 DEF FN JB$(N) = MID$(CODE$,N+261,1)
2640 DEF FN JC$(N) = MID$(CODE$,N+262,1)
2650 DEF FN JD$(N) = MID$(CODE$,N+263,1)
2660 DEF FN JE$(N) = MID$(CODE$,N+264,1)
2670 DEF FN JF$(N) = MID$(CODE$,N+265,1)
2680 DEF FN JG$(N) = MID$(CODE$,N+266,1)
2690 DEF FN JH$(N) = MID$(CODE$,N+267,1)
2700 DEF FN JI$(N) = MID$(CODE$,N+268,1)
2710 DEF FN JJ$(N) = MID$(CODE$,N+269,1)
2720 DEF FN JK$(N) = MID$(CODE$,N+270,1)
2730 DEF FN JL$(N) = MID$(CODE$,N+271,1)
2740 DEF FN JM$(N) = MID$(CODE$,N+272,1)
2750 DEF FN JN$(N) = MID$(CODE$,N+273,1)
2760 DEF FN JO$(N) = MID$(CODE$,N+274,1)
2770 DEF FN JP$(N) = MID$(CODE$,N+275,1)
2780 DEF FN JQ$(N) = MID$(CODE$,N+276,1)
2790 DEF FN JR$(N) = MID$(CODE$,N+277,1)
2800 DEF FN JS$(N) = MID$(CODE$,N+278,1)
2810 DEF FN JT$(N) = MID$(CODE$,N+279,1)
2820 DEF FN JU$(N) = MID$(CODE$,N+280,1)
2830 DEF FN JV$(N) = MID$(CODE$,N+281,1)
2840 DEF FN JW$(N) = MID$(CODE$,N+282,1)
2850 DEF FN JX$(N) = MID$(CODE$,N+283,1)
2860 DEF FN JY$(N) = MID$(CODE$,N+284,1)
2870 DEF FN JZ$(N) = MID$(CODE$,N+285,1)
2880 DEF FN KA$(N) = MID$(CODE$,N+286,1)
2890 DEF FN KB$(N) = MID$(CODE$,N+287,1)
2900 DEF FN KC$(N) = MID$(CODE$,N+288,1)
2910 DEF FN KD$(N) = MID$(CODE$,N+289,1)
2920 DEF FN KE$(N) = MID$(CODE$,N+290,1)
2930 DEF FN KF$(N) = MID$(CODE$,N+291,1)
2940 DEF FN KG$(N) = MID$(CODE$,N+292,1)
2950 DEF FN KH$(N) = MID$(CODE$,N+293,1)
2960 DEF FN KI$(N) = MID$(CODE$,N+294,1)
2970 DEF FN KJ$(N) = MID$(CODE$,N+295,1)
2980 DEF FN KK$(N) = MID$(CODE$,N+296,1)
2990 DEF FN KL$(N) = MID$(CODE$,N+297,1)
3000 DEF FN KM$(N) = MID$(CODE$,N+298,1)
3010 DEF FN KN$(N) = MID$(CODE$,N+299,1)
3020 DEF FN KO$(N) = MID$(CODE$,N+300,1)
3030 DEF FN KP$(N) = MID$(CODE$,N+301,1)
3040 DEF FN KQ$(N) = MID$(CODE$,N+302,1)
3050 DEF FN KR$(N) = MID$(CODE$,N+303,1)
3060 DEF FN KS$(N) = MID$(CODE$,N+304,1)
3070 DEF FN KT$(N) = MID$(CODE$,N+305,1)
3080 DEF FN KU$(N) = MID$(CODE$,N+306,1)
3090 DEF FN KV$(N) = MID$(CODE$,N+307,1)
3100 DEF FN KW$(N) = MID$(CODE$,N+308,1)
3110 DEF FN KX$(N) = MID$(CODE$,N+309,1)
3120 DEF FN KY$(N) = MID$(CODE$,N+310,1)
3130 DEF FN KZ$(N) = MID$(CODE$,N+311,1)
3140 DEF FN LA$(N) = MID$(CODE$,N+312,1)
3150 DEF FN LB$(N) = MID$(CODE$,N+313,1)
3160 DEF FN LC$(N) = MID$(CODE$,N+314,1)
3170 DEF FN LD$(N) = MID$(CODE$,N+315,1)
3180 DEF FN LE$(N) = MID$(CODE$,N+316,1)
3190 DEF FN LF$(N) = MID$(CODE$,N+317,1)
3200 DEF FN LG$(N) = MID$(CODE$,N+318,1)
3210 DEF FN LH$(N) = MID$(CODE$,N+319,1)
3220 DEF FN LI$(N) = MID$(CODE$,N+320,1)
3230 DEF FN LJ$(N) = MID$(CODE$,N+321,1)
3240 DEF FN LK$(N) = MID$(CODE$,N+322,1)
3250 DEF FN LL$(N) = MID$(CODE$,N+323,1)
3260 DEF FN LM$(N) = MID$(CODE$,N+324,1)
3270 DEF FN LN$(N) = MID$(CODE$,N+325,1)
3280 DEF FN LO$(N) = MID$(CODE$,N+326,1)
3290 DEF FN LP$(N) = MID$(CODE$,N+327,1)
3300 DEF FN LQ$(N) = MID$(CODE$,N+328,1)
3310 DEF FN LR$(N) = MID$(CODE$,N+329,1)
3320 DEF FN LS$(N) = MID$(CODE$,N+330,1)
3330 DEF FN LT$(N) = MID$(CODE$,N+331,1)
3340 DEF FN LU$(N) = MID$(CODE$,N+332,1)
3350 DEF FN LV$(N) = MID$(CODE$,N+333,1)
3360 DEF FN LW$(N) = MID$(CODE$,N+334,1)
3370 DEF FN LX$(N) = MID$(CODE$,N+335,1)
3380 DEF FN LY$(N) = MID$(CODE$,N+336,1)
3390 DEF FN LZ$(N) = MID$(CODE$,N+337,1)
3400 DEF FN MA$(N) = MID$(CODE$,N+338,1)
3410 DEF FN MB$(N) = MID$(CODE$,N+339,1)
3420 DEF FN MC$(N) = MID$(CODE$,N+340,1)
3430 DEF FN MD$(N) = MID$(CODE$,N+341,1)
3440 DEF FN ME$(N) = MID$(CODE$,N+342,1)
3450 DEF FN MF$(N) = MID$(CODE$,N+343,1)
3460 DEF FN MG$(N) = MID$(CODE$,N+344,1)
3470 DEF FN MH$(N) = MID$(CODE$,N+345,1)
3480 DEF FN MI$(N) = MID$(CODE$,N+346,1)
3490 DEF FN MJ$(N) = MID$(CODE$,N+347,1)
3500 DEF FN MK$(N) = MID$(CODE$,N+348,1)
3510 DEF FN ML$(N) = MID$(CODE$,N+349,1)
3520 DEF FN MM$(N) = MID$(CODE$,N+350,1)
3530 DEF FN MN$(N) = MID$(CODE$,N+351,1)
3540 DEF FN MO$(N) = MID$(CODE$,N+352,1)
3550 DEF FN MP$(N) = MID$(CODE$,N+353,1)
3560 DEF FN MQ$(N) = MID$(CODE$,N+354,1)
3570 DEF FN MR$(N) = MID$(CODE$,N+355,1)
3580 DEF FN MS$(N) = MID$(CODE$,N+356,1)
3590 DEF FN MT$(N) = MID$(CODE$,N+357,1)
3600 DEF FN MU$(N) = MID$(CODE$,N+358,1)
3610 DEF FN MV$(N) = MID$(CODE$,N+359,1)
3620 DEF FN MW$(N) = MID$(CODE$,N+360,1)
3630 DEF FN MX$(N) = MID$(CODE$,N+361,1)
3640 DEF FN MY$(N) = MID$(CODE$,N+362,1)
3650 DEF FN MZ$(N) = MID$(CODE$,N+363,1)
3660 DEF FN NA$(N) = MID$(CODE$,N+364,1)
3670 DEF FN NB$(N) = MID$(CODE$,N+365,1)
3680 DEF FN NC$(N) = MID$(CODE$,N+366,1)
3690 DEF FN ND$(N) = MID$(CODE$,N+367,1)
3700 DEF FN NE$(N) = MID$(CODE$,N+368,1)
3710 DEF FN NF$(N) = MID$(CODE$,N+369,1)
3720 DEF FN NG$(N) = MID$(CODE$,N+370,1)
3730 DEF FN NH$(N) = MID$(CODE$,N+371,1)
3740 DEF FN NI$(N) = MID$(CODE$,N+372,1)
3750 DEF FN NJ$(N) = MID$(CODE$,N+373,1)
3760 DEF FN NK$(N) = MID$(CODE$,N+374,1)
3770 DEF FN NL$(N) = MID$(CODE$,N+375,1)
3780 DEF FN NM$(N) = MID$(CODE$,N+376,1)
3790 DEF FN NN$(N) = MID$(CODE$,N+377,1)
3800 DEF FN NO$(N) = MID$(CODE$,N+378,1)
3810 DEF FN NP$(N) = MID$(CODE$,N+379,1)
3820 DEF FN NQ$(N) = MID$(CODE$,N+380,1)
3830 DEF FN NR$(N) = MID$(CODE$,N+381,1)
3840 DEF FN NS$(N) = MID$(CODE$,N+382,1)
3850 DEF FN NT$(N) = MID$(CODE$,N+383,1)
3860 DEF FN NU$(N) = MID$(CODE$,N+384,1)
3870 DEF FN NV$(N) = MID$(CODE$,N+385,1)
3880 DEF FN NW$(N) = MID$(CODE$,N+386,1)
3890 DEF FN NX$(N) = MID$(CODE$,N+387,1)
3900 DEF FN NY$(N) = MID$(CODE$,N+388,1)
3910 DEF FN NZ$(N) = MID$(CODE$,N+389,1)
3920 DEF FN OA$(N) = MID$(CODE$,N+390,1)
3930 DEF FN OB$(N) = MID$(CODE$,N+391,1)
3940 DEF FN OC$(N) = MID$(CODE$,N+392,1)
3950 DEF FN OD$(N) = MID$(CODE$,N+393,1)
3960 DEF FN OE$(N) = MID$(CODE$,N+394,1)
3970 DEF FN OF$(N) = MID$(CODE$,N+395,1)
3980 DEF FN OG$(N) = MID$(CODE$,N+396,1)
3990 DEF FN OH$(N) = MID$(CODE$,N+397,1)
4000 DEF FN OI$(N) = MID$(CODE$,N+398,1)
4010 DEF FN OJ$(N) = MID$(CODE$,N+399,1)
4020 DEF FN OK$(N) = MID$(CODE$,N+400,1)
4030 DEF FN OL$(N) = MID$(CODE$,N+401,1)
4040 DEF FN OM$(N) = MID$(CODE$,N+402,1)
4050 DEF FN ON$(N) = MID$(CODE$,N+403,1)
4060 DEF FN OO$(N) = MID$(CODE$,N+404,1)
4070 DEF FN OP$(N) = MID$(CODE$,N+405,1)
4080 DEF FN OQ$(N) = MID$(CODE$,N+406,1)
4090 DEF FN OR$(N) = MID$(CODE$,N+407,1)
4100 DEF FN OS$(N) = MID$(CODE$,N+408,1)
4110 DEF FN OT$(N) = MID$(CODE$,N+409,1)
4120 DEF FN OU$(N) = MID$(CODE$,N+410,1)
4130 DEF FN OV$(N) = MID$(CODE$,N+411,1)
4140 DEF FN OW$(N) = MID$(CODE$,N+412,1)
4150 DEF FN OX$(N) = MID$(CODE$,N+413,1)
4160 DEF FN OY$(N) = MID$(CODE$,N+414,1)
4170 DEF FN OZ$(N) = MID$(CODE$,N+415,1)
4180 DEF FN PA$(N) = MID$(CODE$,N+416,1)
4190 DEF FN PB$(N) = MID$(CODE$,N+417,1)
4200 DEF FN PC$(N) = MID$(CODE$,N+418,1)
4210 DEF FN PD$(N) = MID$(CODE$,N+419,1)
4220 DEF FN PE$(N) = MID$(CODE$,N+420,1)
4230 DEF FN PF$(N) = MID$(CODE$,N+421,1)
4240 DEF FN PG$(N) = MID$(CODE$,N+422,1)
4250 DEF FN PH$(N) = MID$(CODE$,N+423,1)
4260 DEF FN PI$(N) = MID$(CODE$,N+424,1)
4270 DEF FN PJ$(N) = MID$(CODE$,N+425,1)
4280 DEF FN PK$(N) = MID$(CODE$,N+426,1)
4290 DEF FN PL$(N) = MID$(CODE$,N+427,1)
4300 DEF FN PM$(N) = MID$(CODE$,N+428,1)
4310 DEF FN PN$(N) = MID$(CODE$,N+429,1)
4320 DEF FN PO$(N) = MID$(CODE$,N+430,1)
4330 DEF FN PP$(N) = MID$(CODE$,N+431,1)
4340 DEF FN PQ$(N) = MID$(CODE$,N+432,1)
4350 DEF FN PR$(N) = MID$(CODE$,N+433,1)
4360 DEF FN PS$(N) = MID$(CODE$,N+434,1)
4370 DEF FN PT$(N) = MID$(CODE$,N+435,1)
4380 DEF FN PU$(N) = MID$(CODE$,N+436,1)
4390 DEF FN PV$(N) = MID$(CODE$,N+437,1)
4400 DEF FN PW$(N) = MID$(CODE$,N+438,1)
4410 DEF FN PX$(N) = MID$(CODE$,N+439,1)
4420 DEF FN PY$(N) = MID$(CODE$,N+440,1)
4430 DEF FN PZ$(N) = MID$(CODE$,N+441,1)
4440 DEF FN QA$(N) = MID$(CODE$,N+442,1)
4450 DEF FN QB$(N) = MID$(CODE$,N+443,1)
4460 DEF FN QC$(N) = MID$(CODE$,N+444,1)
4470 DEF FN QD$(N) = MID$(CODE$,N+445,1)
4480 DEF FN QE$(N) = MID$(CODE$,N+446,1)
4490 DEF FN QF$(N) = MID$(CODE$,N+447,1)
4500 DEF FN QG$(N) = MID$(CODE$,N+448,1)
4510 DEF FN QH$(N) = MID$(CODE$,N+449,1)
4520 DEF FN QI$(N) = MID$(CODE$,N+450,1)
4530 DEF FN QJ$(N) = MID$(CODE$,N+451,1)
4540 DEF FN QK$(N) = MID$(CODE$,N+452,1)
4550 DEF FN QL$(N) = MID$(CODE$,N+453,1)
4560 DEF FN QM$(N) = MID$(CODE$,N+454,1)
4570 DEF FN QN$(N) = MID$(CODE$,N+455,1)
4580 DEF FN QO$(N) = MID$(CODE$,N+456,1)
4590 DEF FN QP$(N) = MID$(CODE$,N+457,1)
4600 DEF FN QQ$(N) = MID$(CODE$,N+458,1)
4610 DEF FN QR$(N) = MID$(CODE$,N+459,1)
4620 DEF FN QS$(N) = MID$(CODE$,N+460,1)
4630 DEF FN QT$(N) = MID$(CODE$,N+461,1)
4640 DEF FN QU$(N) = MID$(CODE$,N+462,1)
4650 DEF FN QV$(N) = MID$(CODE$,N+463,1)
4660 DEF FN QW$(N) = MID$(CODE$,N+464,1)
4670 DEF FN QX$(N) = MID$(CODE$,N+465,1)
4680 DEF FN QY$(N) = MID$(CODE$,N+466,1)
4690 DEF FN QZ$(N) = MID$(CODE$,N+467,1)
4700 DEF FN RA$(N) = MID$(CODE$,N+468,1)
4710 DEF FN RB$(N) = MID$(CODE$,N+469,1)
4720 DEF FN RC$(N) = MID$(CODE$,N+470,1)
4730 DEF FN RD$(N) = MID$(CODE$,N+471,1)
4740 DEF FN RE$(N) = MID$(CODE$,N+472,1)
4750 DEF FN RF$(N) = MID$(CODE$,N+473,1)
4760 DEF FN RG$(N) = MID$(CODE$,N+474,1)
4770 DEF FN RH$(N) = MID$(CODE$,N+475,1)
4780 DEF FN RI$(N) = MID$(CODE$,N+476,1)
4790 DEF FN RJ$(N) = MID$(CODE$,N+477,1)
4800
```



**ETIQUETA.** La etiqueta es opcional, sólo debe ponerse cuando sea necesario referirse a esta instrucción desde otra, bien para saltar a ella o para modificarla. Como etiqueta sirve cualquier sucesión de letras o números siempre que emplee por una letra; los espacios no son significativos, por lo cual se usa el símbolo "\_" para separar palabras. Sólo los seis primeros caracteres son tratados como etiqueta.

Ejemplos:

ETIQUETA	ETIQUETA
etiqueta	etiqueta
MUEVE_	MUEVE_
2	(ilegal)
DOS	DOS
UN_2	UN_2

**NEMOTECNICO.** Es el código de la instrucción y siempre estará presente pues es el que propiamente la define. Consta de 1 a 4 letras mayúsculas que recuerdan en parte la operación que realizan.

Ejemplo:

Instrucción de carga en inglés	LOAD
nemotécnico	LD

**OPERANDOS.** Este es el campo más variable de la instrucción. Muchas instrucciones no tienen necesidad de que se les definan operandos, ya que éstos están implícitos en su operación. Otras tienen necesidad de tener definidos dos operandos, en este caso irán separados por coma ",". Operando podrá ser un número, una etiqueta, un registro o un par de registros. Cuando el valor del operando se refiera al contenido de la

posición de memoria indicada, el operando se pondrá entre paréntesis.

Ejemplos:

HL	Valor del par de registros HL
(HL)	Contenido de la posición de memoria direccionada por HL
36FAh	Valor hexadecimal 36FAh
(36FA)	Contenido de la posición de memoria 36FAh

**COMENTARIOS.** En un número limitado de caracteres, es una explicación del porqué y para qué de esta instrucción. Va separado de los operandos.

Ejemplo:

PERIODO LD A,30 : DIAS DEL MES

## Contador de posición

El ensamblador, en tiempo de ensamblaje (mientras está ensamblado), mantiene un contador de posición (location counter). Este contador tiene el valor de la dirección de la instrucción que se está ensamblando. Es posible acceder a este valor usando el símbolo "\$" (dólar) que lo representa. Este símbolo se usa como una etiqueta en el campo de operando de la instrucción, de tal forma que si se quiere saltar a diez posiciones de memoria más adelante, se saltaría a "\$+10". Cuando se use esta facilidad hay que tener en cuenta el número de octetos de cada instrucción.

1) Código simbólico: En el operando se usarán las siguientes claves:

r,r' = Uno de los registros A, B, C, D, E, H o L.

n = Una expresión o número, cuyo valor no supere el tamaño de un octeto. Entre 0 y 255.

nn = Una expresión o número, cuyo valor no supere el tamaño de dos octetos. Entre 0 y 65535.

d = Una expresión o número con valores comprendidos desde -128 a +127.

b = Una expresión o número con valores comprendidos entre 0 y 7.

e = Una expresión o número con valores comprendidos desde -126 a +129.

cc = Estado de los indicadores de condición en las instrucciones que los usan.

qq = Cualquiera de los pares de registros BC, DE, HL o AF.

ss = Cualquiera de los pares de registros BC, DE, HL, o SP.

pp = Cualquiera de los pares de registros BC, DE, IX o SP.

rr = Cualquiera de los pares de registros BC, DE, IX o SP.

s = Cualquiera r, n, (HL), (IX+d) o (IX+d).

m = Cualquiera r, (HL), (IX+d) o (IX+d).

2) Objeto: Donde se describirá la operación que realizará.

3) Código de máquina: Donde se presentará el código binario de la instrucción y el hexadecimal, si es posible.

4) Indicadores de condición que afecta: Siempre que la instrucción afecte los indicadores de condición se indicará cuáles y cómo los afecta.

Estos son:	C = acarreo
	N = suma/resta
	P/V = paridad/deshor-
	damiento
	H = semi-acarreo
	Z = cero
	S = signo

5) Número de ciclos de máquina: Número de veces que el microprocesador accede a la memoria.

6) Número de ciclos de reloj: Número de ciclos de reloj que necesita la instrucción para ejecutarse.

7) Ejemplos: Con cada instrucción, se dará un ejemplo que muestre sobre el par la forma en que actúa y cómo modifica los registros y las posiciones de memoria a las que afecta.

Por otro lado, también veremos ejemplos que se podrán introducir en el ordenador, y cuya realización explicaremos de forma exhaustiva. De cada ejemplo, se dará el listado de Assembly, para que quien lo desee, pueda teclearlo por medio de un ensamblador. Para quienes no dispongan de ensamblador, se acompañará cada ejemplo de un programa en Basic (también explicado), que introduzca el código en memoria y lo ejecute.

## Ejecución de código máquina en el Spectrum

Quienes dispongan de ensamblador, deberán mirar las instrucciones del mismo, pa-

ra ver cómo deben introducir sus programas en memoria. En cualquier caso, en un capítulo posterior, estudiaremos en profundidad el manejo de ensambladores, y concretamente, del GENS 3, que a pesar de todo, tiene el pequeño inconveniente de traer las instrucciones en inglés.

Por ahora, aprenderemos a utilizar el código máquina desde el Basic, construyendo pequeños programas cargadores de C/M.

Para introducir en el Spectrum un programa en C/M, empezaremos por escribirlo en Assembly sobre un papel. Una vez decidido en qué lugar de la memoria lo vamos a cargar, lo ensamblaremos a mano siguiendo las normas que daremos en los siguientes capítulos. El resultado, será una serie de números, comprendidos entre 0 y 255, que constituyen el código máquina propiamente dicho.

Mediante un bucle FOR ... NEXT en Basic, vamos introduciendo estos números en sucesivas posiciones de memoria a partir de la RAMTOP (que previamente habremos bajado). Y finalmente, utilizaremos la funciónUSR para ejecutarlo.

Veamos un ejemplo: Supongamos que el programa que deseamos cargar, está representado por los números: 12, 65, 87, 80, 68, 91, 18, 71, 33 y 27 (en este ejemplo, los números son aleatorios, así que no se moleste nadie en desensamblarlo, por que no tiene sentido). Supongamos también, que lo queremos introducir a partir de la dirección 50,000 y que se ejecuta a partir de 50,005 (un programa en C/M no tiene por qué ejecutarse siempre des-



de operación), generan una o varias palabras de datos y quedan definidas dentro del programa absoluto. El formato es el siguiente:

**ETIQUETA SEUDO-NEMOTECNICO**  
EXPRESION

**ETIQUETA:** Sigue las mismas normas que para instrucciones, y su uso está justificado por la necesidad de acceder a los datos. 'ólo es obligatorio con el directivo EQU.

**SEUDO-NEMOTECNICO:** Son una serie de caracteres en mayúsculas, basados en el idioma inglés, que recuerdan el tipo de dato que definen. Como más usuales citaremos:

**EQU expresión**  
Tiene que estar precedido por una etiqueta. Pone la etiqueta igual al valor de la expresión. La expresión no puede contener una etiqueta que no haya sido previamente valorada.

**DEFB expresión, expresión...**  
Cada expresión tiene que tener un valor que entre en un octeto. Coloca el valor de cada expresión en octetos consecutivos a partir del contador de posición.

**DEFW expresión, expresión...**  
Cada expresión tiene que tener un valor que entre en dos octetos. Coloca el valor de cada expresión en pares de octetos consecutivos a partir del contador de posición.

**DEFS expresión**  
Reserva un bloque de memoria.

moria, igual al valor de la expresión.

DEFM 's'

Define el contenido del octeto con el valor en código ASCII de las letras colocadas entre comillas.

## Diagrama de flujo

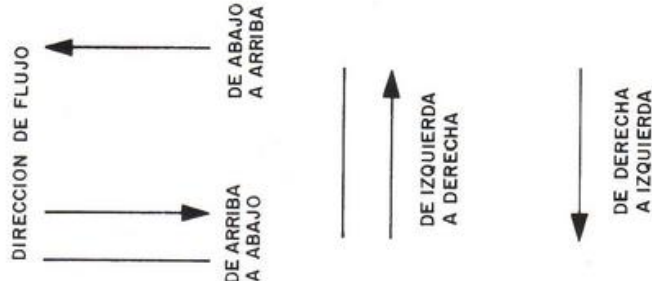
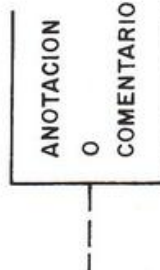
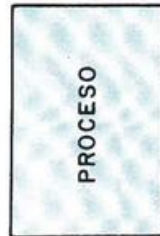
Conocidos también como organigramas u organogramas, son una construcción gráfica del programa. Un buen organigrama facilita la codificación posterior y proporciona una representación visual de todas las situaciones o ramas del programa.

Si se utilizan los símbolos estándar, cualquier otro usuario podrá entenderlo; por lo tanto, se definirán a continuación los más utilizados, que son suficientes para la realización de los organigramas del SPECTRUM.

Como normas generales se tendrá en cuenta:

- El tamaño de los símbolos es variable, sólo se deben mantener las proporciones.
- En el interior de los símbolos se debe escribir claro y conciso.
- Salvo que se indique lo contrario, la dirección del flujo en el organigrama es: de izquierda a derecha y de arriba a abajo.

Símbolos básicos



Representa la dirección del flujo del programa. Estas flechas o líneas unen los símbolos del organigrama. Las direcciones de arriba a abajo y de izquierda a derecha no es necesario señalarlas, las otras sí.

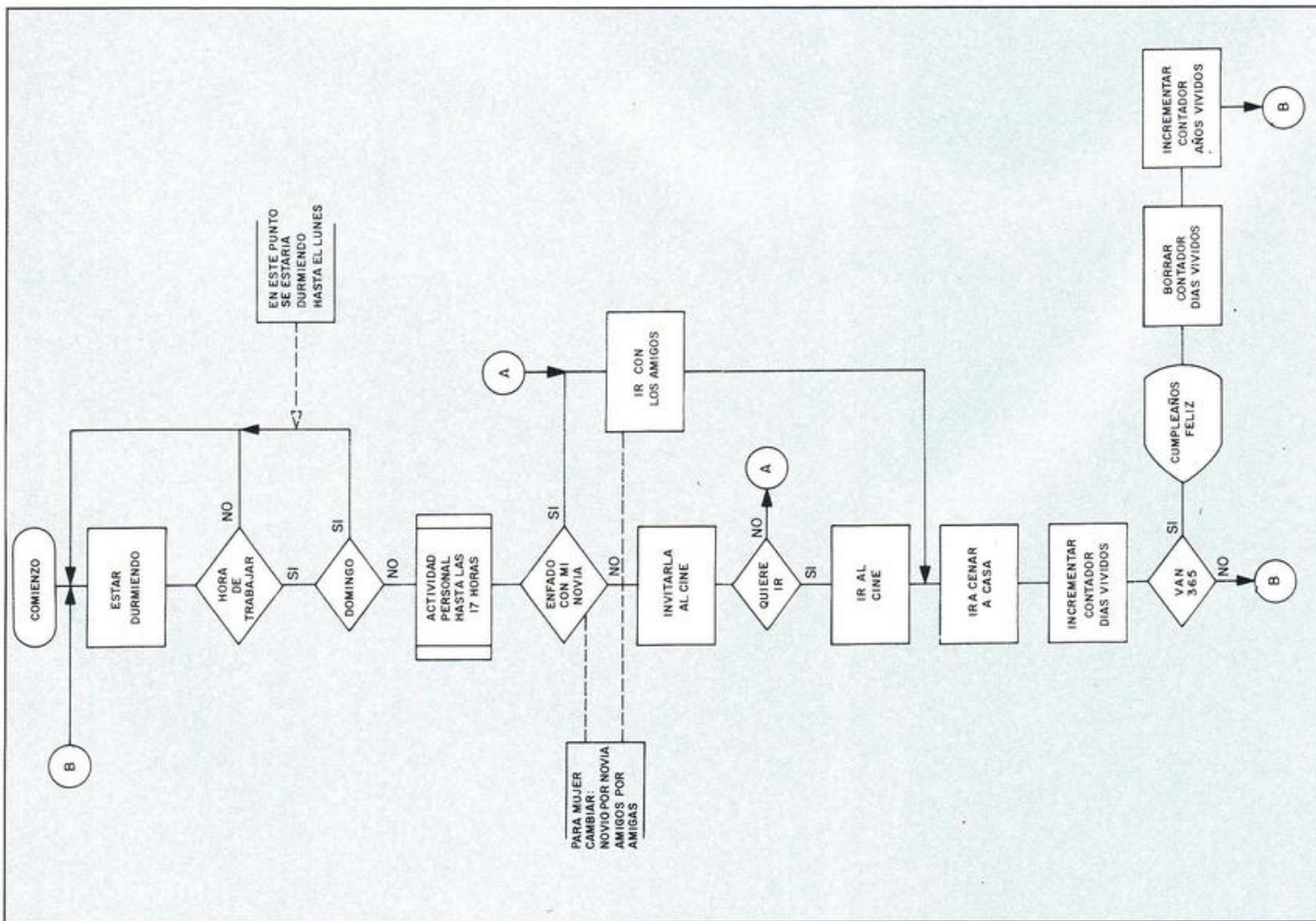


Figura 4.3.



Se usa para añadir comentarios o anotaciones marginales de tipo aclaratorio.

Simbolos especializados de entrada/salida



Representa una función de entrada/salida por medio de un documento. Por ejemplo, una impresión.



Representa una función de entrada/salida en la que la entrada es manual en tiempo de proceso. Ejemplos: teclado, interruptores, pulsado de botones, etc.



Representa una función de entrada/salida en la que la información es presentada para uso humano en tiempo de proceso. Ejemplo: indicadores, pantalla de video.



Representa una conexión dentro del organigrama, tanto de salida hacia, como de entrada por. Normalmente se pone una letra o un número para

Representa una función de entrada/salida sobre una cinta magnética. Por ejemplo, el cassette.

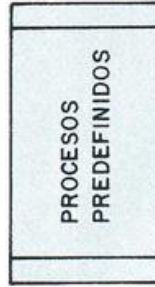


Representa una función de entrada/salida sobre un disco magnético.

Simbolos especializados



Representa una decisión dando paso a las alternativas que pueden ser seguidas.



Representa el nombre de un proceso que consiste en una o más operaciones. Por ejemplo, las subrutinas.

CONECTOR



Básicos



indicar hacia dónde se dirige o el nombre de la entrada. Una flecha marcará el sentido.

Ejemplos:

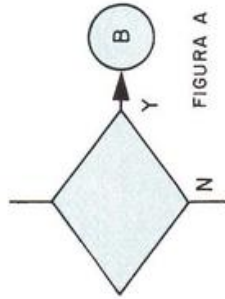


FIGURA A

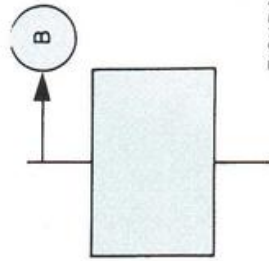


FIGURA B

La decisión "Y" se dirigirá a "B", que estará definido en otra parte del organigrama. El flujo de programa llega desde el punto "B", donde se le mando aquí.



Representa un punto terminal en el programa. Por ejemplo: el comienzo, el final, un punto de espera, un alto, una interrupción, etc.

Otros simbolos usados

Especializados de entrada/salida

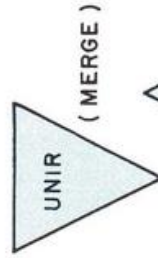


ENLACE DE COMUNICACIONES

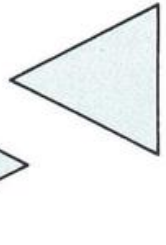


Este simbolo podria usarse para diferenciar entre el casette y el microdrive.

Especializados de proceso

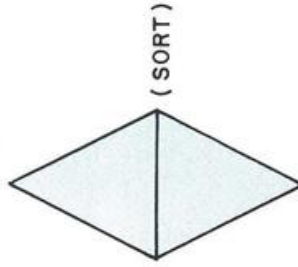


( MERGE )



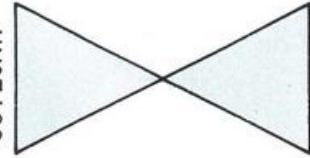
EXTRAER

ORDENAR



( SORT )

COTEJAR



Una tabla de saltos se puede representar de la siguiente manera:

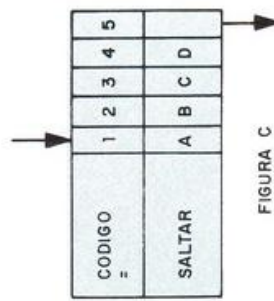


FIGURA C

En la FIGURA 4-3, se puede ver un ejemplo de lo que podría ser un organigrama que representara las actividades básicas de una persona. Creemos que el ejemplo es de por si bastante ilustrativo de cómo se hace un organigrama. Esperamos, no obstante, que ninguno de nuestros lectores rija su existencia por un bucle de tan escasas posibilidades.

## Presentación de las instrucciones

A partir del próximo capítulo iremos estudiando por grupos, todas las instrucciones que usa el Z-80. Veremos la forma de utilizarlas en Assembler, y la forma de ensamblarlas en código máquina para aquellos que no dispongan de ensamblador. También veremos una serie de ejemplos que irán creciendo en complejidad, y que el lector podrá teclear en su ordenador para irse habituando al uso de este lenguaje. Las instrucciones se presentarán de la siguiente manera:



Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

LD A,(DE)

Contenido del registro acumulador, no significativo.

Contenido del par de registros DE

(D): 0 1 0 0 1 1 1 1  
(E): 1 1 1 1 1 1 1 1

Contenido de la posición de memoria 4FFFh

(4FFFh): 1 1 1 0 1 1 1 0

Ejecutamos la instrucción:

LD A,(DE): 0 0 0 1 1 0 1 0

Contenido del acumulador después de la ejecución

(A): 1 1 1 0 1 1 1 0

LD A,(nn)

OBJETO:

Carga en el registro acumulador, el contenido de la posición de memoria direccionada por el operando "nn".

CODIGO MAQUINA:

0 0 1 1 1 0 1 0  
<-----n----->  
<-----n----->

3Ah  
LSB  
MSB

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

13

EJEMPLO:

CERO EQU #456A  
LD A,(CERO)

La primera línea de este ejemplo define una etiqueta, esta operación no tiene código máquina y sirve simplemente, para indicarle al ensamblador, que allí donde le digamos la palabra "CERO", debe entender que queremos decir el número 456Ah.

Esta instrucción también se podría haber escrito sin etiqueta de la siguiente forma:

LD A, (#456A)

La utilidad de las etiquetas es que si vamos a acceder a la posición 456Ah muchas veces, seguramente nos resulte más fácil recordar la palabra "CERO" que el número 456Ah.

Contenido de la posición de memoria 456Ah.

(456Ah): 0 0 0 0 0 0 0 0

Ejecutamos la instrucción:

LD A, (#456A):  
00111010 3Ah  
01101010 6Ah  
01000101 45h

Observe cómo se codifica el operando: el octeto de orden inferior (64h) se almacena

na, en la instrucción, delante del octeto de orden superior (45h).

Contenido del acumulador después de la ejecución

(A): 0 0 0 0 0 0 0 0

LD A,I

OBJETO:

Carga en el acumulador, el contenido del registro "I" (vector de página de interrupción), y carga en el indicador "P/V" del registro "F", el estado del flip/flop de aceptación de interrupción "IFF2", que será "1" si la interrupción está habilitada y "0" si está inhabilitada. De esta forma, es posible comprobar de una sola instrucción, el estado del microprocesador en cuanto a las interrupciones.

CODIGO MAQUINA:

1 1 1 0 1 1 0 1 Edh  
0 1 0 1 0 1 1 1 57h

INDICADORES DE CONDICION:

S (signo): Pone a "1" si "I" es negativo, es decir, si su bit de más peso es "1".  
Z (cero): Pone a "1" si "I" vale cero.

H (semiacarreo): Pone a "0"

P/V (Paridad/rebosamiento): Pone a "1" si las interrupciones están habilitadas y a "0" si están inhabilitadas.

N (suma/resta): Pone a "0".  
C (acarreo): Permanece con su estado anterior.

CICLOS DE MEMORIA:

2

Dónde ubicar un programa en C/M

en principio, un programa en código máquina se puede colocar en cualquier lugar de la memoria, de hecho, existen programas comerciales que la ocupan prácticamente por completo. No obstante, para nuestros fines existen zonas más adecuadas que otras.

Se supone que un programador aficionado, utilizará rutinas en C/M combinadas con un programa principal en Basic, por lo que habrá que respetar una zona de memoria para que el Basic pueda trabajar.

Básicamente, existen cuatro zonas donde situar nuestros programas:

1. Por encima de la RAMTOP.
2. En el buffer de impresora.
3. En el archivo de pantalla.
4. Dentro del programa Basic.

Veámoslas una por una:

1. **Por encima de la RAMTOP:** Es la zona más adecuada para colocar un programa en C/M, ya que queda protegido de borrados por el sistema Basic. En primer lugar, deberemos bajar la RAMTOP con el uso de CLEAR, como se vea en el ejemplo anterior. Una vez cargado nuestro programa, no podrá ser borrado ni siquiera con NEW; para volver a la situación inicial, de-

beremos teclear:

RANDOMIZE USR 0

Que si borrará el programa en C/M y todo lo que haya en la memoria del ordenador.

Otra forma de destruir nuestro programa sería, volver a subir la RAMTOP.

2. **En el buffer de impresora:** Existe en la RAM, una zona reservada de 256 bytes, que empieza en la dirección 23296 (B00h) y acaba en la 23551 (5BFFh); esta zona la utiliza el Spectrum cuando trabaja con una impresora tipo ZX-Printer (Alphacom-32 o Seikosha GP-50S); si no va a utilizar ninguna de estas impresoras, puede almacenar en esta zona una rutina corta (256 bytes máximo) que no le ocupará, por tanto, memoria en la zona de programa. Tenga en cuenta, no obstante, que su rutina será borrada si utiliza los comandos: NEW, LPRINT, LIST y COPY.

3. **En el archivo de pantalla:** En casos especiales, se utiliza el archivo de pantalla para almacenar programas en C/M, es una técnica usada en algunos copiadotes para no ocupar memoria útil. Si no desea "ensuciar" la pantalla, puede poner los atributos correspondientes al mismo color de tinta y papel, con lo que los bytes no se visualizarán en forma de pixels. Cuando utilice esta técnica, tenga en

cuenta que su programa puede ser corrompido por el uso de NEW, CLEAR y cualquier comando que afecte a la pantalla. El archivo de pantalla va desde 16384 (4000h) hasta 22527 (57FFh).

4. **Dentro del programa Basic:** Esta era la técnica usada en el ZX-81, consiste en hacer que la primera línea del programa sea una línea REM, con tantos espacios, como bytes tenga el programa C/M a almacenar. La dirección de inicio de esta zona es (PROG)+5. Este método tiene la ventaja de poder salvar juntos el Basic y el Código Máquina, si bien, su empleo no es recomendable si se tiene conectado el INTERFACE 1, ya que este dispositivo desplaza el programa Basic, y por tanto, nuestra rutina en C/M, a menos que ésta sea reubicable y entremos en ella, calculando cada vez la dirección de entrada a partir del contenido de la variable PROG. En este caso, nuestra rutina sólo se borra editando la línea, o borrando el programa Basic con NEW.

De todos éstos, el sistema usado con más frecuencia es el primero, y es el que usaremos en nuestros ejemplos. Si se tiene conectado un interfaz de impresora INDES-COMP, ha de tenerse en cuenta que su software ocupa los 1000 bytes más altos de la memoria.



# INSTRUCCIONES DE CARGA

Las instrucciones de carga transfieren contenidos de memoria a registros, de registros a memoria y entre registros.

Se trata del grupo principal de instrucciones del microprocesador, y su necesidad queda justificada, ya que todas las operaciones aritméticas y lógicas se hacen sobre registros del microprocesador, o entre estos y posiciones de memoria y casi siempre será necesario almacenar los resultados sobre la memoria.

Por otra parte, gran número de instrucciones utilizan registros para direccionar posiciones de memoria, bien sea mediante direccionamiento absoluto o indexado.

El formato básico de estas instrucciones es:

LD DESTINO, ORIGEN
--------------------

El código LD del inglés "LOAD" (carga), indica al microprocesador que debe cargar en el "DESTINO" el valor contenido en el "ORIGEN".

El "DESTINO" y el "ORIGEN", pueden ser tanto registros, como posiciones de memoria, utilizaremos "r" y "r'" para referirnos a los registros de 8 bits, afectados por la instrucción, y "dd" para referirnos a los de 16 bits (pares de registros).

Los valores de "r" y "r'" usados para el código de máquina en este grupo de ins-

trucciones, son los siguientes:

r y r'	registro
111	A
000	B
001	C
010	D
011	E
100	H
101	L

Los valores de "dd" usados para el código de máquina en este grupo de instrucciones, son los siguientes:

dd	par de registros
00	BC
01	DE
10	HL
11	SP

## Grupo de instrucciones de carga en registros

LD r,r'

OBJETO:

Carga el contenido del registro indicado por r', en el registro indicado por r.

CODIGO MAQUINA:

0 1 < r' > < r >
------------------

INDICADORES DE CONDICION A LOS QUE AFECTA

Ninguno

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

LD A,B
--------

El contenido de "A" no es significativo, ya que será destruido por la instrucción. Supongamos que el contenido de "B" es 43 en decimal, 2Bh en Hexa.

(B):	00101011	2Bh
------	----------	-----

Ejecutamos la instrucción: LD A,B que carga en el registro "A", el contenido del registro "B";

LD A,B:	01111000	7Bh
---------	----------	-----

Después de la ejecución, el registro "A" contendrá el valor que contenía el registro "B", mientras que el contenido de este último no se habrá modificado.

Contenido de "A" después de la ejecución:

(A):	00101011	2Bh
------	----------	-----

Contenido de "B" después de la ejecución:

(B):	00101011	2Bh
------	----------	-----

Como vimos en un capítulo anterior, los registros completos, en código máquina, una

LD (IX+3),7:	00h	11011101
	36h	00110110
	03h	00000011
	07h	00000111

Contenido de la posición 73Bfh después de la ejecución

(73Bfh):	0 0 0 0 0 1 1 1
----------	-----------------

LD (Y+d),n

OBJETO:

Carga el valor del número entero "n", en el octeto de la posición de memoria que resulta de sumar: el contenido del registro índice "Y" al entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127.

CODIGO MAQUINA:

FDh	1 1 1 1 1 1 0 1
36h	0 0 1 1 0 1 1 0
	<-----d----->
	<-----n----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

LD (Y+5),15
-------------

Le suponemos a "Y" un contenido de 5000h (20480), por lo que accederemos a la posición 5005h (20485). El

contenido de esta posición es irrelevante.

Contenido de "Y":

50h	0 1 0 1 0 0 0 0
00h	0 0 0 0 0 0 0 0

Ejecutamos la instrucción:

FDh	1 1 1 1 1 0 1
36h	0 0 1 1 0 1 1 0
05h	0 0 0 0 0 1 0 1
0Fh	0 0 0 0 1 1 1 1

Contenido de la posición 5005h después de la ejecución:

(5005h):	0 0 0 0 1 1 1 1
----------	-----------------

## Grupo de instrucciones de carga en registro acumulador

LD A,(BC)

OBJETO:

Carga en el registro acumulador, el contenido de la posición de memoria direccionada por el par de registros "BC".

CODIGO MAQUINA:

0Ah	0 0 0 0 1 0 1 0
-----	-----------------

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

LD A,(BC)
-----------

Supongamos que el par de registros "BC" contienen el número 76DFh (30431), esta es por tanto, la posición cuyo contenido cargaremos en el acumulador. Supongamos también, que el contenido de esta posición es AAh (170). El contenido del acumulador es irrelevante, ya que se pierde al ejecutar la instrucción.

Contenido del registro "BC".

(B):	0 1 1 1 0 1 1 0	76h
(C):	1 1 0 1 1 1 1 1	DFh

Contenido de la posición de memoria 76DFh.

(76DFh):	1 0 1 0 1 0 1 0	AAh
----------	-----------------	-----

Ejecutamos la instrucción:

LD A,(BC):	0 0 0 0 1 0 1 0	0Ah
------------	-----------------	-----

Contenido del acumulador después de la ejecución

(A):	1 0 1 0 1 0 1 0	AAh
------	-----------------	-----

LD A,(DE)

OBJETO:

Carga en el registro acumulador, el contenido de la posición de memoria direccionada por el par de registros "DE".

CODIGO MAQUINA:

1Ah	0 0 0 1 1 0 1 0
-----	-----------------

INDICADORES DE CONDICION:



INDICADORES DE CONDICIÓN:

Ninguno

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

LD (IV+10),B

Supongamos que el índice "IV" contiene 5F40h (24384), por lo que accederemos a la posición 5F4Ah (24394). Suponemos también, que el registro "B" contiene FFh (255). El contenido de la posición 5F4Ah no es significativo, ya que será destruido por la instrucción.

Contenido del índice "IV":

(IV):

Contenido del registro "B":

(B):

Ejecutamos la instrucción:

LD (IV+10),B

Contenido de la posición 5F4Ah después de la ejecución:

(5F4Ah):

LD (HL),n

OBJETO:

Carga el valor del número

entero "n", (entre 0 y 255) en la posición de memoria cuya dirección es el contenido del par de registros "HL".

CODIGO MAQUINA:

00110110

INDICADORES DE CONDICIÓN:

Ninguno

CICLOS DE MEMORIA:

3

CICLOS DE RELOJ:

10

EJEMPLO:

LD (HL),57

Este ejemplo se podría escribir también como: LD (HL), # 39 ya que 39h = 57.

Suponemos que el par de registros "HL" contiene 6ACBh (27339), por tanto, esa será la dirección de memoria a la que accederemos. El contenido de esta posición de memoria no es significativo, ya que será destruido por la instrucción.

Contenido de "HL":

(HL):

Ejecutamos la instrucción:

LD (HL),57

Contenido de la posición 6ACBh después de la ejecución:

(6ACBh):

LD (IX+d),n

OBJETO:

Carga el valor del número entero "n", en el octeto de la posición de memoria que resulta de sumar: el contenido del registro índice "IX" al entero de desplazamiento "d", el cual puede adquirir valores desde -128 a +127.

CODIGO MAQUINA:

11011101

INDICADORES DE CONDICIÓN:

Ninguno

CICLOS DE MAQUINA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

LD (IX+3),7

Suponemos que "IX" contiene 73BCh (29628), por lo que accederemos a la posición 73BF (29631), cuyo contenido es irrelevante.

Contenido de "IX"

(IX):

Ejecutamos la instrucción:

función similar a la de las variables en Basic, de forma que esta instrucción sería similar a la instrucción: LET A=B del Basic.

LD r,n

OBJETO:

Carga en el registro indicado por "r" el valor numérico "n" de 8 bits y en el rango de 0 a 255.

CODIGO MAQUINA:

00<---r--->110

INDICADORES DE CONDICIÓN A LOS QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

LD A,47

Esta instrucción carga el valor 47 decimal (2FH Hexa) en el registro "A", el contenido anterior de este registro se pierde al ejecutarse la instrucción.

La mayoría de los ensambladores, permiten introducir los números, tanto en decimal como en Hexa. Concretamente, en el caso del GENS 3, esta instrucción se podría escribir también como:

LD A, # 2F

El signo "#" delante del nú-

mero, indica a el ensamblador que se trata de un número hexadecimal.

Instrucción.

LD A,47:

00111110

Contenido de "A" después de la ejecución:

(A):

El equivalente en Basic de esta instrucción, sería: LET A=47

LD r,(HL)

OBJETO:

Carga en el registro indicado por "r", el contenido del octeto de memoria cuya dirección es el valor del par de registros HL.

CODIGO MAQUINA:

01<---r--->110

INDICADORES DE CONDICIÓN:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

LD B,(HL)

Esta instrucción carga en el registro "B", el contenido de la posición de memoria cuya dirección es el contenido del par de registros "HL". En este caso, estamos usando el mo-

do de direccionamiento indirecto para especificar el "ORIGEN".

Supongamos que el registro "HL" contiene el valor 5F47h (24391), el registro "H" contendrá 5Fh (95) y el registro "L" contendrá 47h (71); observe que 95x256+71=24391.

La posición de memoria cuyo contenido vamos a cargar, será por tanto, la 5F47h. Supongamos que a su vez, esta posición de memoria contiene el número 55h (85). Veamos cómo se desarrollan los acontecimientos.

Contenido del par de registros "HL":

(H):

Contenido de la posición de memoria 5F47h:

(5F47h):

Ejecutamos la instrucción:

LD B,(HL):

Tras la instrucción, sólo se habrá modificado el contenido del registro "B".

Contenido del registro "B" después de la instrucción:

(B):

LD r,(IX+d)

OBJETO:

Carga en el registro indicado por "r", el contenido de la posición de memoria, que resulta de sumar: el valor del registro índice "IX" con un entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127.



110111101
01 <---r---> 110
<-----d----->

00h

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

LD C, (IX+10)
---------------

En este caso, vamos a cargar el registro "C" con el contenido de la posición de memoria, cuya dirección es el resultado de sumar 10 al contenido del registro índice "IX".

Esta instrucción utiliza direccionamiento indexado para especificar el "ORIGEN"; obsérvese que el direccionamiento indexado es similar al indirecto, pero más sofisticado.

El contenido del registro "C" es irrelevante, ya que será destruido por la instrucción. Supongamos que el contenido de "IX" es 7743h (30531), por lo que accederemos a la posición de memoria 774Dh (30541). Supongamos también, que el contenido de esa posición de memoria es 41h (65).

Contenido de "IX":

011110111
010000011

(IX):

77h

43h

Contenido de la posición de

memoria 774Dh:

(774Dh): 010000001

EJEMPLO:

LD A, (Y-15)
--------------

De forma similar al ejemplo anterior, vamos a cargar el acumulador con el contenido de la posición de memoria direccionada por el índice "Y" menos 15.

Supongamos que el contenido de "Y" es 7743h (30531), direccionamos, por tanto la posición de memoria 7734h (30516), a la que a su vez, le suponemos un contenido de 42h (66).

011101111
010000011

(IV):

77h

43h

Contenido de la posición de memoria 7734h:

010000010
-----------

(7734h):

42h

Ejecutamos la Instrucción:

111111101
011111110
111100001

LD A, (Y-15);

FDh

7Eh

F1h

Contenido de "A" después de la ejecución:

010000010
-----------

(A):

42h

Obsérvese que hemos representado -15 como F1h, que es precisamente el complemento a 2 de 0Fh es decir, el negativo de 15.

En el Z-80, el primer byte del código de operación de todas las instrucciones que utilizan el registro "IX" es DDh, y el de todas las que utilizan el "IY" es FDh.

## Grupo de instrucciones de carga en memoria

LD (HL), r
------------

OBJETO:

Carga en contenido del registro indicado por r, en el octeto de memoria direccionado por el valor del par de registros HL.

CODIGO MAQUINA:

01110 <---r--->
-----------------

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

LD (HL), B
------------

Esta instrucción carga en la posición de memoria cuya dirección es el contenido de "HL", el contenido del registro "B". Los contenidos previos de "B" y "HL" no son alterados y si el contenido de la posición de memoria correspondiente.

En este caso, se utiliza direccionamiento indirecto para especificar el "DESTINO". Supongamos que "HL" contiene 4723h (18211), ésta será por tanto, la posición a la que accederemos. Supongamos asimismo, que el registro "B" tiene un contenido de 75h (117). El contenido de la posición de memoria 4723 es irre-

levante, ya que será destruido por la instrucción.

Contenido del par "HL":

(H): 010000111
(L): 001000011

47h

23h

Contenido de "B":

011101011
-----------

75h

Ejecutamos la instrucción:

LD (HL), B: 011100000
-----------------------

70h

Contenido de la posición 4723h después de la ejecución:

(4723h): 011101011
--------------------

75h

## LD (IX+d), r

OBJETO:

Carga el contenido del registro indicado por "r", en el octeto de la posición de memoria que resulta de sumar: el valor del registro índice "IX" con el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127.

CODIGO MAQUINA:

110111101
01110 <---r--->
<-----d----->

00h

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

LD (IX+7), C
--------------

Supongamos que "IX" contiene 75B3h (30131), por lo que accederemos a la posición 75BAh (30138), cuyo contenido es irrelevante. Suponemos también, que "C" contiene F0h (240).

Contenido del par "IX":

011101011
101100011

(IX):

75h

83h

Contenido de "C":

111100000
-----------

(C):

F0h

Ejecutamos la instrucción:

110111101
011100001
000001111

LD (IX+7), C:

00h

71h

07h

Contenido de la posición 75BAh después de la ejecución:

111100000
-----------

(75BAh):

F0h

LD (IY+d), r
--------------

OBJETO:

Carga el contenido del registro indicado por "r", en el octeto de la posición de memoria resultante de sumar: el valor del registro índice "IY" al entero de desplazamiento "d" el cual puede adquirir los valores desde -128 a +127.

CODIGO MAQUINA:

111111101
01110 <---r--->
<-----d----->

FDh



# EJEMPLO:

LD (# 4527), IX

Esta instrucción, carga el octeto de orden bajo del registro "IX", en la dirección 4527h, y el octeto de orden alto en la siguiente.

Suponemos, como ejemplo, que el registro "IX" contiene el número C3ECh. Contenido de "IX".

MSB:	1 1 0 0 0 0 1 1	C3h
(IX)		
LSB:	1 1 1 0 1 1 0 0	ECh

Ejecutamos la instrucción:

	11011101	D0h
	00100010	22h
	00100111	27h
	01000101	45h

LD (# 4527), IX:

Situación de la memoria después de la instrucción:

(4527h):	1 1 1 0 1 1 0 0	ECh
(4528h):	1 1 0 0 0 0 1 1	C3h

LD (nn), IX

## OBJETO:

Carga en la dirección "nn" de memoria, el octeto de orden inferior de registro índice "IX" y en la dirección "nn+1", el de orden superior.

CODIGO MAQUINA:

1 1 1 1 1 1 0 1	FDh
0 0 1 0 0 0 1 0	22h
<-----n----->	LSB
<-----n----->	MSB

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

20

EJEMPLO:

INDICE EQU # 774F  
LD (INDICE), IX

Este ejemplo es igual que el anterior, pero esta vez, hemos utilizado una etiqueta para referirnos al número 774Fh. Suponemos que el índice "IX", contiene DA5Dh. Contenido de "IX".

MSB:	1 1 0 1 1 0 1 0	DAh
(IX)		
LSB:	0 1 0 1 1 0 1 1	5Dh

Ejecutamos la instrucción:

	11111101	FDh
	00100010	22h
	01001111	4Fh
	01101111	77h

LD (# 774F), IX:

Contenido de la memoria después de la instrucción:

(774Fh):	0 1 0 1 1 1 0 1	5Dh
(7750h):	1 1 0 1 1 0 1 0	DAh

## Grupo de instrucciones de carga en registro SP

LD SP, HL

OBJETO:

Carga en el registro puntero de pila "SP", el contenido

del par de registros "HL".

CODIGO MAQUINA:

1 1 1 1 1 0 0 1	F9h
-----------------	-----

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

6

EJEMPLO:

LD SP, HL

Se trata de una instrucción rápida y que ocupa un solo byte de memoria. El microprocesador solo accede a memoria una vez, para leer el código de operación y se limita a realizar una transferencia interna entre registros.

Suponemos que el contenido de "HL" es F000h, éste será el número que se transferirá al puntero de pila y que será la nueva dirección de la pila de máquina. Contenido de "HL".

(HL):	1 1 1 1 0 0 0 0	F0h
(L):	0 0 0 0 0 0 0 0	00h

Ejecutamos la instrucción:

LD SP, HL:	1 1 1 1 0 0 0 1	F9h
------------	-----------------	-----

Contenido de "SP" después de la ejecución:

MSB:	1 1 1 1 0 0 0 0	F0h
(SP)		
LSB:	0 0 0 0 0 0 0 0	00h

Hay que tener sumo cuida-

CICLOS DE RELOJ:

9

EJEMPLO:

LD A, I

Suponemos que el registro "I" contiene el valor 9Fh y que las interrupciones están habilitadas. Contenido de "I".

(I):	1 0 0 1 1 1 1 1	9Fh
------	-----------------	-----

Ejecutamos la instrucción:

LD A, I:	1 1 1 0 1 1 0 1	EDh
	0 1 0 1 0 1 1 1	57h

Contenido de "A" después de la instrucción:

(A):	1 0 0 1 1 1 1 1	9Fh
------	-----------------	-----

Estado de "F" después de la instrucción:

S	Z	H	P/V	N	C
1	0	x	0	x	1 0 *

x: Estado indeterminado.

\*: El flag no cambia su estado anterior.

LD A, R

OBJETO:

Carga en el acumulador el contenido del registro "R" (registro de regeneración).

CODIGO MAQUINA:

1 1 1 0 1 1 0 1	EDh
0 1 0 1 0 1 1 1	57h

INDICADORES DE CONDICION:

S (signo): Pone a "1" si "R" es negativo.

Z (cero): Pone a "1" si "R" es cero.

H (semiacarreo): Pone a "0".

P/V (paridad/desbordamiento): Copia el estado de IFF2.

N (suma/resta): Pone a "0" C (acarreo): Preserva su contenido anterior.

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

9

EJEMPLO:

LD A, R

Suponemos que en el momento de la ejecución, el registro "R" contiene el número 3Ah. Y que las interrupciones están inhibidas. "R".

Contenido de "R".

(R):	0 1 1 1 0 1 0 0	3Ah
------	-----------------	-----

Ejecutamos la instrucción:

LD A, R:	1 1 1 0 1 1 0 1	EDh
	0 1 0 1 1 1 1 1	57h

Contenido de "A" después de la instrucción:

(A):	0 0 1 1 1 0 1 0	3Ah
------	-----------------	-----

Estado de "F" después de la instrucción:

S	Z	H	P/V	N	C
0	0	x	0	x	0 0 *

x: Estado indeterminado.  
\*: Preserva el estado anterior.

## Grupo de instrucciones para salvar el registro acumulador

LD (BC), A

OBJETO:

Carga, en el octeto de la posición de memoria direccionada por el valor del par de registros "BC", el contenido del registro acumulador.

CODIGO MAQUINA:

0 0 0 0 0 0 1 0	02h
-----------------	-----

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

LD (BC), A

Suponemos que el registro "BC" contiene el número C9C0h (51648) y que el acumulador contiene BCh (188). El contenido de la posición C9C0h es irrelevante, ya que será destruido por la instrucción.

Contenido del par de registros "BC".

(B):	1 1 0 0 1 0 0 1	C9h
(C):	1 1 0 0 0 0 0 0	C0h

Contenido del registro acumulador:

(A):	1 0 1 1 1 0 0 0	BCh
------	-----------------	-----



Ejecutamos la instrucción:

LD (BC),A: 0000000100 02h

Contenido de la posición C9C0h después de la ejecución:

(C9C0h): 10111100 8Ch

**LD (DE),A**

OBJETO:

Carga, en el octeto de memoria direccionado por el valor del par de registros "DE", el contenido del registro acumulador.

CODIGO MAQUINA:

00010010 12h

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

LD (DE),A

Suponemos que "DE" contiene 8000h (32768), y que "A" contiene FFh (255). El contenido de la posición 8000h es irrelevante.

Contenido del par de registros "DE":

(D): 10000000 80h  
(E): 00000000 00h

Contenido del registro acu-

mulador:

(A): 11111111 FFh

Ejecutamos la instrucción:

LD (DE),A: 00010010

Contenido de la posición 80000h después de la ejecución:

(8000h): 11111111 FFh

**LD (nn),A**

OBJETO:

Carga, en el octeto de memoria direccionado por el valor del operando "nn", el contenido del registro acumulador.

CODIGO MAQUINA:

00110010 32h  
LSB  
MSB

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

13

EJEMPLO:

LD (29016),A

Suponemos que el acumulador "A" contiene 33h (51). El contenido de la posición 7158h (29016) es irrelevante. La instrucción se podría haber escrito también como:

LD (#7158),A

Haciendo uso de la notación hexadecimal, o bien, con una etiqueta de la siguiente forma:

ETIQUETA EQU #7158  
LD (ETIQUETA),A

Contenido del registro acumulador:

(A): 00110011 33h

Ejecutamos la instrucción:

LD (#7158),A: 00110010 32h  
01011000 58h  
01110001 71h

Contenido de la 7158h después de la ejecución:

(7158h): 00110011 33h

**LD I,A**

OBJETO:

Carga en el registro "I" (vector de página de interrupción), el contenido del registro acumulador.

CODIGO MAQUINA:

11101101 Edh  
01000111 47h

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

9

CICLOS DE RELOJ:

16

EJEMPLO:

LD (#45FD),HL

Este grupo de instrucciones es el opuesto al visto anteriormente. De la misma forma que antes teníamos los números en memoria con el orden de los octetos invertidos, esta vez, el microprocesador se encarga de almacenarlos también, con el orden invertido. Esto hace que los dos grupos de instrucciones sean totalmente compatibles.

En otras instrucciones, también apreciaremos esta particularidad; como regla general, podemos decir que todos los números de dos bytes que se almacenan en la memoria, deberán guardarse con el orden de sus octetos invertidos (primero el menos significativo y luego el más significativo). He aquí la razón última de porqué las Variab-  
bles del Sistema tienen este formato.

En este ejemplo concreto, vamos a guardar en la dirección 45FDh y siguiente, el contenido del par "HL", que suponemos, es de AABHh.

Contenido del par de registros "HL":

(H): 10101010 AAh  
(L): 10111011 BBh

Ejecutamos la instrucción:

LD (#45FD),HL: 00100010 22h  
11111101 Fdh  
01000101 45h

Situación del número en

memoria, después de la instrucción:

(45FDh): 10111011 8Bh  
(45FEh): 10101010 AAh

**LD (nn),dd**

OBJETO:

Carga en la posición de memoria "nn", el octeto de orden inferior del par de registros indicados por "dd", y en la posición de memoria "nn+1" el octeto de orden superior.

CODIGO MAQUINA:

11101101 Edh  
01000111 LSB  
<-----0-----> MSB  
<-----0----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

20

EJEMPLO:

TODD EQU #45FA  
LD (TODD),BC

La palabra "TODO", es en este caso, una etiqueta, que sustituye al número 45AFh que es la dirección donde queremos almacenar el contenido del par "BC". Suponemos que este contenido es, por ejemplo, F00Fh. Contenido del par "BC":

(B): 11110000 F0h  
(C): 00001111 0Fh

Ejecutamos la instrucción:

LD (#45AF),BC: 11101101 Edh  
01000111 43h  
11111010 FAh  
01000101 45h

Contenido de las posiciones afectadas por la instrucción:

(TODD): 00001111 0Fh  
(TODD+1): 11110000 F0h

Recuerde que la palabra "TODO" es una etiqueta que equivale al número 45AFh.

**LD (nn),IX**

OBJETO:

Carga en la posición "nn" de memoria, el octeto de orden inferior del registro índice "IX" y en la posición "nn+1", el octeto de orden superior.

CODIGO MAQUINA:

11011101 00h  
00100010 22h  
<-----0-----> LSB  
<-----0-----> MSB

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

20



inferior del registro índice "IX", el octeto direccionado por "nn", y en la parte de orden superior, el octeto direccionado por "nn+1".

CODIGO MAQUINA:

00h	11011101
2Ah	00101010
LSB	<-----0----->
MSB	<-----0----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

8

CICLOS DE RELOJ:

20

EJEMPLO:

LD IX, (# 46F0)
-----------------

En este caso, vamos a hacer lo mismo que en ejemplos anteriores, pero cargando el índice "IX". Suponemos que vamos a cargar el número BBAAh (48042).

Situación del número en memoria:

46F0h:	10101010
46F1h:	10111011

Ejecutamos la instrucción:

00h	11011101
2Ah	00101010
40h	11110000
46h	01000110

Contenido del índice "IX" después de la ejecución:

MSB:	10111011
(IX)	
LSB:	10101010

LD IV, (nn)

OBJETO:

Carga, en la parte de orden inferior del registro índice "IV", el octeto de memoria direccionado por "nn", y en la parte superior, el octeto de memoria direccionado por "nn+1".

CODIGO MAQUINA:

FDh	11111101
2Ah	00101010
LSB	<-----0----->
MSB	<-----0----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

20

EJEMPLO:

GRUPO EQU #AAEE
LD IV, (GRUPO)

De nuevo, utilizamos una etiqueta. Llamamos "GRUPO" al número AAEEh. A partir de ese momento, cada vez que pongamos en un operando la palabra "GRUPO", el ensamblador considerará que nos referimos a este número. La instrucción podía haberse escrito también, como:

LD IV, (#AAEE)
----------------

El número que queremos cargar en "IV" es el 35BFh (13759), que está contenido en las posiciones de memoria AAEEh y AAEEh.

Situación del número en memoria:

AAEEh:	10111111
AAEFh:	00110101

Ejecutamos la instrucción:

FDh	11111101
2Ah	00101010
ECh	11101110
AAh	10101010

Contenido del índice "IV" después de la ejecución:

MSB:	00110101
(IV)	
LSB:	10111111

Grupo de instrucciones de carga en memoria, 16 bits

LD (nn),HL

OBJETO:

Carga en la dirección de memoria "nn" el contenido del registro "L" y en la dirección de memoria "nn+1", el contenido del registro "H".

CODIGO MAQUINA:

22h	00100010
LSB	<-----0----->
MSB	<-----0----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

5

EJEMPLO:

LD LA
-------

Suponemos que el acumulador contiene el número 02h. El contenido del registro "I" es irrelevante.

Contenido del registro acumulador:

(A):	00000010
------	----------

Ejecutamos la instrucción:

LD LA:	11101101
EDh	01000111
47h	01000111

Contenido del registro "I" después de la ejecución:

(I):	00000010
------	----------

LD R,A

OBJETO:

Carga en el registro "R" (registro de regeneración), el contenido del registro acumulador.

CODIGO MAQUINA:

EDh	11101101
4Fh	01001111

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

9

EJEMPLO:

LD R,A

Supongamos que el acumulador contiene el número 57h. El contenido del registro "R" es, de nuevo, irrelevante. Contenido del registro acumulador:

(A):	01010111
------	----------

Ejecutamos la instrucción:

LD R,A:	11101101
EDh	01001111
4Fh	01001111

Contenido del registro "R" después de la ejecución:

(R):	10101111
------	----------

Grupo de instrucciones de carga en registro, 16 bits

LD dd,nn

OBJETO:

Carga en el par de registros indicados por "dd" el número entero de dos octetos "nn".

CODIGO MAQUINA:

EDh	00000000
4Fh	00000000
LSB	<-----0----->
MSB	<-----0----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

3

CICLOS DE RELOJ:

10

EJEMPLO:

LD BC, #6A7F
--------------

Vamos a cargar el número 6A7Fh (27263) en el registro doble "BC", esto quiere decir, que cargaremos 6Ah (106) en el registro "B", y 7Fh (127) en el registro "C".

Esta instrucción podría haberse escrito también como:

LD BC, 27263
--------------

El contenido anterior del par de registros "BC" es irrelevante.

Ejecutamos la instrucción:

00h	00000001
7Fh	01111111
6Ah	01101010

Observe que el entero 6A7Fh se codifica con el orden de sus octetos invertido, es decir, primero el octeto menos significativo (LSB) y luego el más significativo (MSB).

Contenido de "BC" después de la ejecución:

(B):	01101010
(C):	01111111

Recuerde:

par	dd
BC	00
DE	01
HL	10
SP	11

LD IX,nn

OBJETO:

Carga en el registro índice



"IX" el número entero de dos octetos "nn".

#### CODIGO MAQUINA:

DDh	1 1 0 1 1 1 0 1
21h	0 0 1 0 0 0 0 1
LSB	<-----η----->
MSB	<-----η----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

14

EJEMPLO:

LD IX, 5

Esta instrucción se podría haber escrito también como:

LD IX, # 0005

De lo que se trata es de cargar el registro índice "IX" con el número 0005h (5).

Obsérvese como los enteros se codifican, de nuevo, con el orden invertido, es decir, primero va el octeto menos significativo y luego, el más significativo.

Ejecutamos la instrucción:

DDh	1 1 0 1 1 1 0 1
21h	0 0 1 0 0 0 0 1
05h	0 0 0 0 0 1 0 1
00h	0 0 0 0 0 0 0 0

Contenido del registro "IX" después de la ejecución

(IX): 0005000000000101

#### LD IY,nn

OBJETO:

Carga en el registro índice "Y" el número entero de dos octetos "nn".

CODIGO MAQUINA:

FDh	1 1 1 1 1 1 0 1
21h	0 0 1 0 0 0 0 1
LSB	<-----η----->
MSB	<-----η----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

14

EJEMPLO:

LD IY, # 33AA

De forma similar al ejemplo anterior, vamos a cargar el número 33AAh en el registro "IY". De nuevo, codificamos los octetos del entero, al revés.

Ejecutamos la instrucción:

FDh	1 1 1 1 1 1 0 1
21h	0 0 1 0 0 0 0 1
AAh	1 0 1 0 1 0 1 0
33h	0 0 1 1 0 0 1 1

Contenido del registro índice "IY" después de la ejecución:

(IY): 001100110101010

33AAh

#### LD HL, (nn)

OBJETO:

Carga en el registro "L" el octeto de memoria direccionado por "nn" y en el registro "H" el octeto de memoria direccionado por "nn+1".

CODIGO MAQUINA:

2Ah	0 0 1 0 1 0 1 0
LSB	<-----η----->
MSB	<-----η----->

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

16

EJEMPLO:

LD HL, # 6677

Vamos a cargar el par de registros "HL", con un número de dos bytes (16 bits) contenido en la memoria. Dado que el número tiene 16 bits, ocupará dos posiciones de memoria contiguas. Como operando de la instrucción, tenemos que dar la dirección de la primera de las dos posiciones.

Como de costumbre, el microprocesador considera que los octetos que componen el número, están en orden inverso, es decir, primero el de menos peso (LSB) y luego el de más peso (MSB).

Supongamos que el número

ro que vamos a cargar en "HL" es el 33FFh (13311), que se encuentra almacenado en las direcciones de memoria 6677h y 6678h (26231 y 26232). La posición 6677h contendrá FFh, y la 6678h contendrá 33h.

Situación de los octetos en la memoria:

6677h:	1 1 1 1 1 1 1 1
6678h:	0 0 1 1 0 0 1 1

Al codificar la instrucción en código máquina, también deberemos invertir el orden de los octetos en el operando:

Ejecutamos la instrucción:

LD HL, # 6677:	00101010
	01110111
	01100110

De forma que, una vez ejecutada la instrucción, el contenido de la dirección de memoria 6677h (LSB) se habrá cargado en el registro "L", y el contenido de 6678h (MSB), lo habrá hecho en el registro "H".

Contenido de "HL" después de la instrucción:

(H):	0 0 0 0 0 0 0 0
(L):	1 1 1 1 1 1 1 1

OBJETO:

Carga, en la parte de orden inferior del par de registros indicado por "dd", el octeto de memoria direccionado por "nn" y en la parte de orden superior, el octeto direccionado por "nn+1".

Esta instrucción es similar

a la que hemos descrito anteriormente, salvo que puede trabajar con todos los pares de registros, no sólo con el "HL". En compensación, ésta ocupa 4 bytes, en lugar de los tres que ocupaba la anterior.

CODIGO MAQUINA:

EDh	1 1 1 0 1 1 0 1
LSB	0 1 d d 1 0 1 1
MSB	<-----η----->

Par	"dd"
BC	00
DE	01
HL	10
SP	11

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

20

EJEMPLO:

NUMERO EQU # 6789
LD DE, (NUMERO)

En este ejemplo, vamos a cargar en "DE", el contenido de la posición de memoria 6789h y siguiente. El orden de los octetos está, de nuevo, invertido.

A partir de estas instrucciones y según nos vayamos adentrando en otras más complejas, es de suma importancia que el lector analice cuidadosamente los cuadros que acompañan a los ejemplos y que muestran el contenido de los registros y posiciones de memoria, según

van siendo afectados por la instrucción.

Supongamos que el número que vamos a cargar es el 72C8h (29384), de forma que la posición 6789h, contendría el número C8h y la posición 678Ah, el 72h.

En el código fuente, hemos utilizado una etiqueta, con el fin de que el lector se vaya habituando a su uso. No obstante, la instrucción podría haberse escrito como:

LD DE, # 6789

Situación del número en memoria:

6789h:	1 1 0 0 1 0 0 0
678Ah:	0 1 1 1 0 0 1 0

Ejecutamos la instrucción:

EDh	1 1 1 0 1 1 0 1
58h	0 1 0 1 1 0 1 1
89h	1 0 0 0 1 0 0 1
67h	0 1 1 0 0 1 1 1

Contenido de "DE" después de la ejecución:

(D):	0 1 1 1 0 0 1 0
(E):	1 1 0 0 1 0 0 0

Observe que con este código de máquina se puede codificar, también, la instrucción LD HL, (nn), sólo que ocuparía 4 octetos en lugar de 3. Cualquier ensamblador codificaría el código que nosotros octetos ocupase.

LD IX, (nn)

OBJETO:

Carga, en la parte de orden



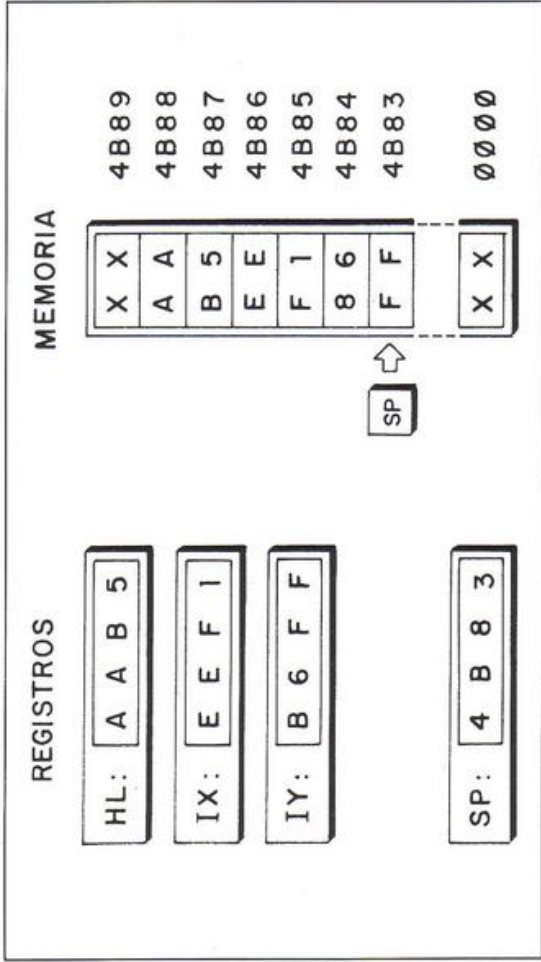


Fig. 5. 1D. Contenidos, después de ejecutar la instrucción: «PUSH IY».

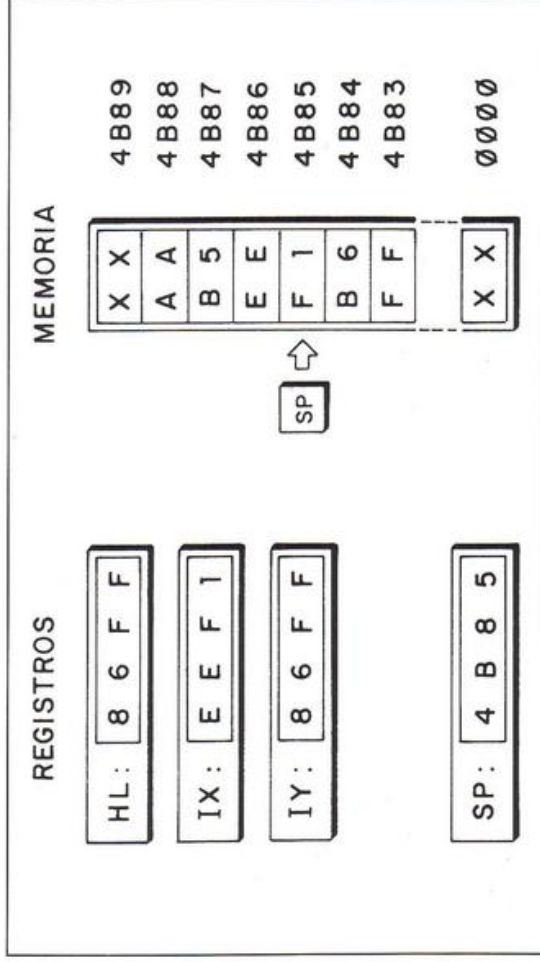


Fig. 5. 1E. Contenidos, después de ejecutar la instrucción: «POP HL».

hemos visto el proceso de expansión de la pila. Vamos a ver ahora, el proceso inverso: sacar datos de la pila.

Nuestra primera instrucción será:

POP HL

Que toma el último dato que hayamos introducido en la pila y lo coloca dentro del registro «HL».

En la FIGURA 5-1E, podemos ver cómo quedan pila y registros, después de esta instrucción. El último dato introducido en la pila (86FFh) ha pasado a ser el contenido del registro «HL» y el puntero

do cuando se ejecute esta instrucción, ya que el puntero de pila cambia de lugar y no será posible recuperar los datos que estuvieran guardados en la pila antigua.

LD SP,IX

OBJETO:  
Carga en el registro puntero de pila, «SP», el contenido del registro índice «IX».

CODIGO MAQUINA:  
00h 11011101  
F9h 11111001

INDICADORES DE CONDICION:  
Ninguno

CICLOS DE MEMORIA:  
2

CICLOS DE RELOJ:  
10

EJEMPLO:  
LD SP,IX

Otra vez, se trata de una simple transferencia de registros desde el índice «IX» al puntero de pila «SP». Supongamos que «IX» contiene C6E1h. Contenido de «IX»:

IX: 1100011011000001 C6E1h

Ejecutamos la instrucción:  
LD SP,IX: 11011101 11100001 00h 11011101 F9h 11111001

Contenido de «SP» después de la ejecución: 80D21h

(SP): 1100011011000001 C6E1h

LD SP,IY

OBJETO:  
Carga en el registro puntero de pila, «SP», el contenido del registro índice «IY».

CODIGO MAQUINA:  
FDh 11111101  
F9h 11111001

INDICADORES DE CONDICION:  
Ninguno

CICLOS DE MEMORIA:  
2

CICLOS DE RELOJ:  
10

EJEMPLO:  
LD SP,IY

Exactamente igual que el ejemplo anterior, pero con el índice «IY». Suponemos que su contenido es 8D21h.

Contenido del registro «IY»: IY: 1000110100100001 8D21h

Ejecutamos la instrucción:  
LD SP,IY: 11111101 11100001 FDh 11111001 F9h 11111001

Contenido del registro «SP» después de la ejecución: 80D21h

(SP): 1000110100100001 80D21h

## Grupo de instrucciones de manejo de pila

Una pila es una cola LIFO (last input first output), último en entrar primero en salir. El término pila es de uso habitual, se apilan cajas, revistas, etc. Pues bien, una pila en términos informáticos funciona igual, por ejemplo: una persona que compra todos los meses una revista, es fácil que las ordene en una pila; es decir, irá poniendo una encima de la anterior, de tal forma que la última colocada siempre estaría más al alcance.

De la misma manera, en un ordenador se pueden ir guardando en una tabla en memoria, mejor denominada cola, una serie de octetos, y en una palabra de control de dos octetos se guardaría la última dirección usada de la tabla, de forma que: para meter un nuevo octeto se sumaría uno a la palabra de control de tabla y se cargaría el octeto en esa dirección; para sacar un octeto se leería el octeto direccionado por la palabra de control y se le restaría uno a ésta.

Eso es lo que se pretende con las instrucciones que siguen, las cuales utilizan el registro puntero de pila «SP».

Para identificar los pares de registros usaremos el siguiente código:

qq	par
00	BC
01	DE
10	HL
11	AF

En el Spectrum, la pila se coloca en la parte alta de memoria, el sistema operativo la



situa inmediatamente debajo de RAMTOP, durante la rutina de inicialización. Esto lo hace, cargando el registro "SP" con la dirección inmediatamente inferior a la de RAMTOP.

Cada vez que utilizemos la instrucción PUSH, meteremos en la pila el contenido de un par de registros y cada vez que utilizemos la instrucción POP, sacaremos el dato más alto de la pila y lo asignaremos a un par de registros.

Nuestra pila se expande "hacia abajo", lo cual quiere decir que cuando hablemos de "la parte superior de la pila", en realidad, nos estaremos refiriendo a la dirección más baja de ésta.

Por otro lado, todos los datos que se almacenan en la pila, tienen dos bytes de longitud, por lo cual, el registro "SP" se incrementa o se decrementa de 2 en 2.

El proceso de introducir el contenido de un par de registros en la pila, conlleva las siguientes operaciones:

1. Se decrementa "SP".
2. Se transfiere el octeto de orden alto del par de registros correspondiente a la dirección apuntada por "SP".
3. Se vuelve a decrementar "SP".

4. Se transfiere el octeto de orden bajo del par correspondiente a la dirección apuntada por "SP".

El proceso de sacar un número de la pila, implica que el microprocesador realice las mismas operaciones a la inversa:

1. Se toma el contenido de la dirección apuntada por "SP" y se carga en el octeto de orden bajo del registro correspondiente.
2. Se incrementa "SP".
3. Se toma el contenido

de la dirección apuntado ahora por "SP" y se carga en el octeto de orden alto del registro correspondiente.

4. Se vuelve a incrementar "SP".

Algunos microprocesadores trabajan con dos pilas, una se denomina "pila de máquina" y otra "pila de usuario". La pila de máquina la utiliza el microprocesador para introducir sus datos y la pila de usuario, es la que el programador puede utilizar.

En el Z-80 no existe "pila de usuario", de forma que el programador debe usar la misma pila que la máquina. Esto lleva aparejados ciertos inconvenientes, así que vamos a ver para qué utiliza la máquina esta pila.

Cada vez que el microprocesador recibe una instrucción que le haga saltar a una subrutina, almacena en la pila la dirección a la que deberá retornar cuando termine esa rutina. Por tanto, siempre que dentro de una subrutina utilizemos la pila, deberemos asegurarnos de sacar todos los datos que hayamos introducido antes de intentar retornar, ya que de lo contrario, el microprocesador tomaría nuestro último dato como dirección de retorno; si esto ocurriera, se diría que nuestra subrutina "corrompe la pila". Es imposible retornar con éxito desde una subrutina que corrompa la pila, por lo que hay que procurar que esto nunca ocurra.

A continuación, vamos a ver las instrucciones que puede utilizar el programador para trabajar sobre la pila.

## PUSH qq

## OBJETO:

Introducir el contenido del par de registros indicado por "qq" en la pila apuntada por el registro "SP". Esta instrucción ejecuta los siguientes pasos: decrementa el valor del registro SP y carga el octeto de orden superior del par de registros indicado por "qq" en la dirección especificada por "SP"; a continuación vuelve a decrementar el registro "SP" y carga el octeto de orden inferior.

## CODIGO MAQUINA:

1 1 q q 0 1 0 1

## INDICADORES DE CONDICION:

Ninguno

## CICLOS DE MEMORIA:

3

## CICLOS DE RELOJ:

11

## EJEMPLO:

PUSH HL

Supongamos que el par "HL" contiene el número AAB5h y que el puntero de pila "SP", apunta a la dirección 4B89h, que será la del último dato introducido en la pila.

Contenido de "HL":

(H):	1 0 1 0 1 0 1 0	AAh
(L):	1 0 1 0 1 0 1 0	B5h

Contenido del puntero "SP":

(SP): 0100100110001001 4B89h

Ejecutamos la instrucción:

## REGISTROS

HL:	A A B 5
IX:	E E F 1
IY:	8 6 F F
SP:	4 B 8 5

## MEMORIA

X X	4 B 8 9
A A	4 B 8 8
B 5	4 B 8 7
E E	4 B 8 6
F 1	4 B 8 5
X X	4 B 8 4
X X	4 B 8 3
X X	0 0 0 0

SP

Fig. 5. 1C. Contenidos, después de ejecutar la instrucción: "PUSH IX".

registro "SP", de esta forma, nos indica cuál es el último dato introducido en la pila, es decir, el primero que podemos leer.

Partiendo de la situación que muestra esta figura, vamos a ejecutar la primera de nuestras instrucciones:

PUSH HL

Esta instrucción debe guardar en la pila, el contenido del par de registros "HL"; el registro "SP" se decrementará dos veces, y por tanto, el cuadrado que apunta a la memoria bajará dos casillas.

En la FIGURA 5-1B, podemos ver la situación después de que esta instrucción haya sido ejecutada. El registro "HL" contiene el mismo valor que antes, ya que éste ha sido copiado en la pila, pero no se ha destruido. Vemos que la dirección 4B88h contiene el número AAh y la dirección 4B87h, el número B5h, por

tanto, las dos juntas componen el número AAB5h que es, precisamente, el contenido de "HL" que queríamos preservar. Por otro lado, vemos que el cuadrado (a partir de ahora, lo llamaremos *puntero*) ha bajado dos casillas, precisamente, para apuntar al último dato introducido.

Si ahora utilizáramos la instrucción POP para recuperar un dato de la pila, sería precisamente este dato el que podríamos recuperar.

Vamos con la segunda de nuestras instrucciones:

PUSH IX

En este caso, vamos a guardar en la pila el contenido del registro "IX"; sin por ello, perder el dato que habíamos guardado anteriormente.

En la FIGURA 5-1C, se puede ver cómo quedan los contenidos después de esta última instrucción. La posición de memoria 4B86h contiene

el número EEh, y la 4B85h el número F1h; juntos forman EEFFh, que es, de nuevo, el contenido que queríamos preservar. El puntero (cuadrado) ha vuelto a bajar dos casillas, para apuntar, de nuevo, al último dato introducido.

Vamos ahora, a meter en la pila el último de nuestros datos: el contenido del registro "IY".

PUSH IY

Con esta instrucción, entra en la pila el número 86FFh. En la FIGURA 5-1D, podemos ver, de nuevo, cómo queda la pila después de esta instrucción. Ahora el puntero ha bajado a la casilla 4B83h, con lo que otra vez, apunta al último dato introducido.

Podríamos seguir metiendo datos en la pila indefinidamente, hasta que agotáramos la memoria disponible, pero con estos tres ejemplos, ya



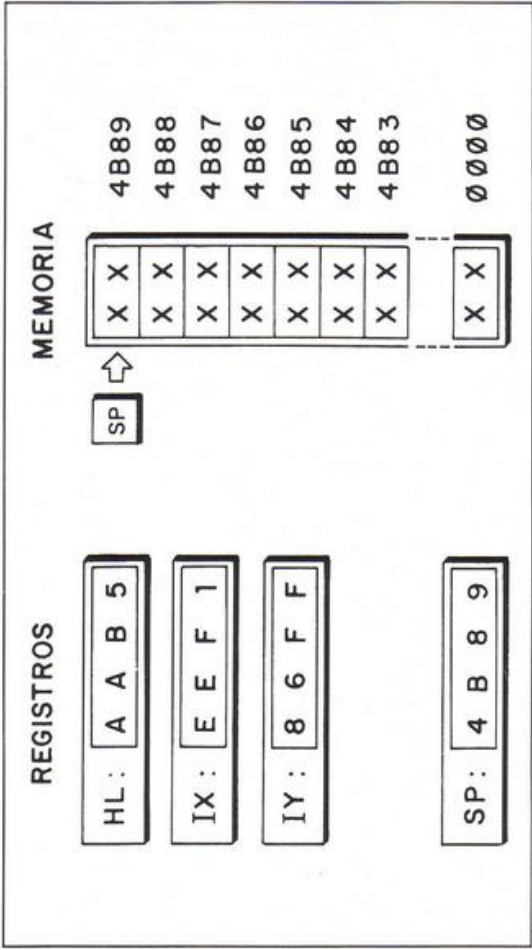


Fig. 5. 1A. Situación inicial de registros y posiciones de memoria.

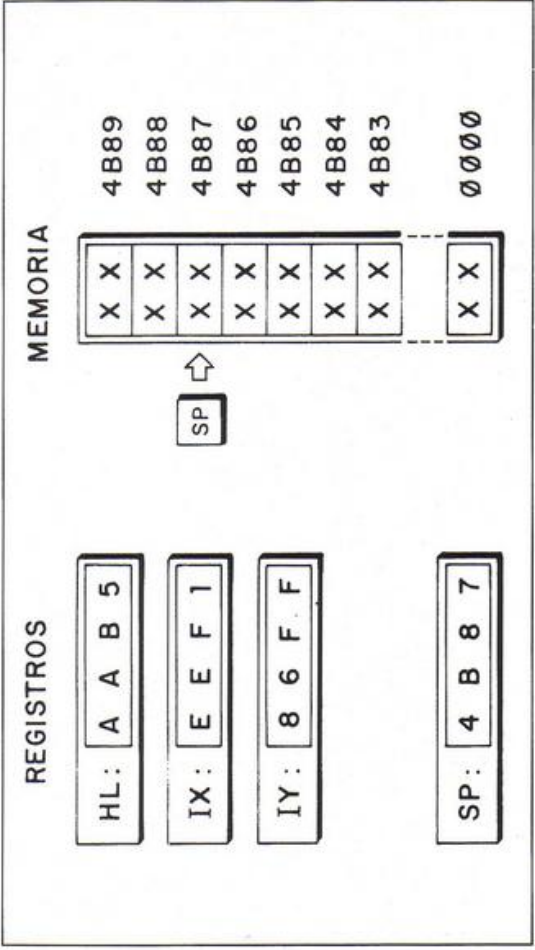


Fig. 5. 1B. Contenidos, después de ejecutar la instrucción: «PUSH HL».

El registro "SP", contiene moria donde está situada la verdadero contenido nos es 4B89h, que es la dirección de pila. A la derecha de cada ca- indiferente. memoria a partir de donde Vemos un cuadradito con crecerá la pila. las letras "SP" dentro de él; En la parte derecha de la fi- este cuadrado, apunta a la hexadecimal. En principio, to- casilla cuya dirección es pre- das las casillas contienen cisamente, el contenido del "xx", lo que significa que su

PUSH HL: 1 1 1 0 0 1 0 1 E5h

Contenido de "SP" después de la instrucción:

(SP): 0100101110000111 4B87h

Contenido de la pila

(4B87h): 1 0 1 1 0 1 0 1 B5h

(4B88h): 1 0 1 0 1 0 1 0 AAh

### PUSH IX

OBJETO: Introducir el contenido del registro índice "IX" en la pila apuntada por el registro "SP". Esta instrucción ejecuta los siguientes pasos: decrementa el valor del registro "SP" y carga el octeto de orden superior del registro "IX" en la dirección especificada por "SP", a continuación, vuelve a decrementar el registro "SP" y carga el octeto de orden inferior.

CODIGO MAQUINA: 1 1 0 1 1 0 1 0 01h 1 1 0 0 1 0 1 0 E5h

INDICADORES DE CONDICION: Ninguno

CICLOS DE MEMORIA: 4

CICLOS DE RELOJ: 15

EJEMPLO: PUSH IX

Supongamos que el índice "IX" contiene EE1Fh y "SP" está como lo dejamos después de la instrucción anterior, es decir, apuntando a 4B87h.

Contenido de "IX": 1 1 0 1 1 0 1 1 1 0 0 0 0 1 1 1 EE1Fh

Contenido de "SP": 0 1 0 0 1 0 1 1 1 0 0 0 0 0 1 1 1 1 4B87h

Ejecutamos la instrucción:

PUSH IX: 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 DDh 1 1 1 0 0 1 0 1 E5h

Contenido de "SP" tras la ejecución: 0 1 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 4B85h

Contenido de la pila

(4B85h): 1 1 1 1 0 0 0 1 F1h

(4B86h): 1 1 1 0 1 1 1 0 E5h

CODIGO MAQUINA: 1 1 1 1 1 1 0 1 FDh 1 1 1 0 0 1 0 1 E5h

INDICADORES DE CONDICION: Ninguno

CICLOS DE MEMORIA: 4

CICLOS DE RELOJ: 15

EJEMPLO: PUSH IY

Contenido de "IY": 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 86Ffh

Contenido de "SP": 0 1 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 4B85h

Ejecutamos la instrucción:

PUSH IY: 1 1 1 1 1 1 0 1 FDh 1 1 1 0 0 1 0 1 E5h



Contenido de "SP" después de la ejecución:

(SP): 0100101110000111 4887h

Contenido de "IX" después de la ejecución:

(IX): 1110111011110001 EEF1h

## POP IX

OBJETO:

Introducir en el registro índice "IX", los dos primeros octetos de la pila apuntada por el registro "SP". Esta instrucción ejecuta los siguientes pasos: carga en la parte inferior del registro índice "IX", el octeto de la dirección especificada por el registro "SP"; incrementa el registro "SP" y carga el siguiente octeto direccionado, en la parte superior del registro índice; por último, vuelve a incrementar el registro "SP".

CODIGO MAQUINA:

11111101	FDh
11100001	EH

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

14

EJEMPLO:

POP IX

trucción ejecuta los siguientes pasos: carga en la parte inferior del registro índice "IX", el octeto de la dirección especificada por el registro "SP"; incrementa el registro "SP" y carga el siguiente octeto direccionado, en la parte superior de registro índice; por último, vuelve a incrementar el registro "SP".

CODIGO MAQUINA:

11011101	DDh
11100001	EH

INDICADORES DE CONDICION:

Ninguno

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

14

EJEMPLO:

POP IX

Continuamos recuperando datos desde la pila, suponemos que la seguimos teniendo como estaba tras el ejemplo anterior.

Contenido de "SP":

(SP): 0100101110000111 4885h

Contenido de la pila como queda en el ejemplo anterior:

F1 EE B5 AA

Ejecutamos la instrucción:

POP IX: 11011101 DDh  
11100001 EH

CICLOS MEMORIA: 3

CICLOS DE RELOJ: 10

EJEMPLO:

POP HL

Suponemos que tenemos la pila y el registro puntero "SP", como quedó tras los ejemplos anteriores. Ahora, vamos a ir recuperando los datos desde la pila. Contenido de "SP":

(SP): 0100101110000111 4883h

Contenido de la pila, como quedó tras los ejemplos anteriores:

FF 86 F1 EE B5 AA

Ejecutamos la instrucción:

POP HL: 11100001 EH

Contenido de "SP" después de la ejecución:

(SP): 0100101110000101 4885h

Contenido de "HL" después de la ejecución:

10000110	86h
11111111	FFh

## POP IX

OBJETO:

Introducir en el registro índice "IX", los dos primeros octetos de la pila apuntada por el registro "SP". Esta ins-

Contenido de SP después de la ejecución:

(SP): 0100101110000111 4883h

Contenido de la pila:

11111111	FFh
10000110	86h

Contenido de la pila después de las ejecuciones anteriores

11111111	FFh
10000110	86h
11110001	FIh
11101110	EEh
10101010	85h
10101010	AAh

## POP qq

OBJETO:

Introducir en el par de registros indicado por "qq", los dos primeros octetos de la pila apuntada por el registro "SP". Esta instrucción ejecuta los siguientes pasos: carga en la parte inferior del par de registros indicado por "qq", el octeto de la dirección especificada por el registro "SP"; incrementa el registro "SP" y carga el siguiente octeto direccionado, en la parte superior del par de registros; por último, vuelve a incrementar el registro "SP".

CODIGO MAQUINA:

1 1 q q 0 0 0 1

INDICADORES DE CONDICION:

Ninguno

sin variar su contenido, comenzaría:

PUSH BC  
PUSH DE  
PUSH HL

Y terminaría:

POP HL  
POP DE  
POP BC

Observe cómo se recupera al revés de cómo se salvó, es decir, el primer registro que se recupera, es el último que se salvó. Recuerde que debe sacar de la pila todo lo que metió, antes de intentar retornar desde una subrutina.

## Una mirada gráfica a la pila

Para quien no esté familiarizado con los ordenadores, el funcionamiento de una pila, puede resultar algo difícil de comprender. Haciendo cierto el refrán "una imagen vale más que mil palabras", vamos a ver de un modo gráfico, lo que ocurre en la pila y en los registros correspondientes, durante la ejecución de las anteriores instrucciones.

Miremos la FIGURA 5-1A, que representa la situación inicial de la que partimos. A la izquierda de la figura, vemos cuatro "ventanas" etiquetadas: "HL", "IX", "IY" y "SP"; se trata de una representación gráfica de los registros del microprocesador.

Cada ventana muestra un número hexadecimal, que representa el contenido del registro correspondiente, por ejemplo, el registro "HL" contiene AAB5h, el "IX" contiene EEF1h, etc.



ravillas desde el primer momento. Lo más importante es ir aprendiendo todo claramente; las "virguerías" podrá hacerlas luego cada uno, no obstante, a lo largo del curso tenemos reservadas para nuestros lectores, maravillosas sorpresas.

Vamos con nuestro siguiente ejemplo, esta vez vamos a leer desde código máquina un número que habremos almacenado desde Basic en la variable del Sistema "SEED". Se trata de leer el contenido de SEED y sacarlo a pantalla a través del registro "BC". En esta ocasión, almacenaremos el programa a partir de la dirección de memoria 30000, para lo cual, bajaremos primero la RAMTOP a 29999. Estas direcciones son válidas tanto para los usuarios de 16 K como de 48 K.

Nuestro programa es el siguiente:

```
ORG 30000
LD HL, (SEED)
LD B,H
LD C,L
RET
SEED EQU #5C76
```

La primera línea: "ORG 30000" es un pseudo-memórico, no se puede ensamblar y su única finalidad es indicarle al ensamblador que deberá ensamblar el programa a partir de la dirección 30000.

La última línea "SEED EQU #5C76" tampoco se puede ensamblar, se trata de una definición de etiqueta, su finalidad es asignarle a la etiqueta "SEED" el valor 5C76h (23670). El programa simplificado, quedaría:

```
LD HL, (5C76)
LD B,H
LD C,L
RET
```

Para codificarlo, tomamos de nuevo las tablas y buscamos el código de:

```
LD HL, (nn)
```

que resulta ser 2Ah (42). A continuación, vendrá el operando invertido: 76h (118) y 5Ch (92). Ahora buscamos:

```
LD B,H y LD C,L
```

cuyos códigos resultan ser 4Dh (77). Finalmente, ponemos el código de RET, es decir, C9h (201). Nuestro programa queda de la siguiente forma:

```
2A,76,5C,4A,40,C9
```

O escrito en decimal:

```
42,118,92,68,77,201
```

Vamos a construir el programa Basic que lo introduce en memoria y lo ejecuta:

```
10000 CLEAR 30000 TO 30000
20000 BASIC 40500 TO 40500
30000 INPUT "SEED=";S
40000 PRINT S
50000 GOTO 10000
```

La línea 10 baja la RAMTOP para preservar nuestro programa contra borrados accidentales. Las líneas 20 y 30

cargan en memoria el programa que se encuentra en los datos de la línea 40. La línea 50 nos pide un valor para SEED, y la línea 60 lo introduce en la variable "SEED" siempre que este valor no sea cero. Finalmente, la línea 70 ejecuta nuestro programa en código máquina e imprime en pantalla el resultado.

En este ejemplo, vemos que es posible establecer una comunicación bidireccional entre Basic y Código Máquina para transferir datos; existen otras muchas formas de realizar esta comunicación que se irán viendo en ejemplos sucesivos.

En nuestro tercer ejemplo, vamos a leer la variable del Sistema RAMTOP desde código máquina, y utilizaremos la pila para sacarla a pantalla por el registro "BC". Asimismo, veremos cómo almacenar una rutina en código máquina dentro de una línea REM del programa Basic.

Primero leeremos el contenido de la variable RAMTOP, cargándolo sobre el registro "HL", luego transferiremos este contenido al "BC" a través de la pila; el programa podría ser el siguiente:

```
LD HL, (RAMTOP)
PUSH HL
POP BC
RET
RAMTOP EQU 23730
```

De nuevo, utilizamos una etiqueta que definimos en la última línea, antes de codificar el programa, eliminamos la etiqueta, quedando el programa simplificado:

## REGISTROS

```
HL: 8 6 F F
IX: E E F I
IY: 8 6 F F
SP: 4 B 8 7
```

## MEMORIA

```
X X 4B89
A A 4B88
B 5 4B87
E E 4B86
F I 4B85
8 6 4B84
F F 4B83
X X 0000
```

SP

Fig. 5-1F. Contenidos, después de ejecutar la instrucción: "POP IX".

ha subido dos casillas, para apuntar al dato anteriormente introducido.

Una observación interesante, es que el contenido de las casillas que componen la pila, no se ha modificado; el número 86FFh sigue estando ahí, aunque a nosotros nos da igual, recuérdese que sólo podemos acceder cada vez, al dato señalado por el puntero; Las casillas 4B84h y 4B83h que contienen el dato 86FFh, sólo se borrarán totalmente cuando la pila vuelva a expandirse.

Ahora, vamos a recuperar el siguiente dato de la pila y lo asignaremos al registro "IX":

POP IX

Esta instrucción toma el último dato de la pila y lo asigna al registro "IX"; recuérdese que el último dato, es siempre el que es apuntado por el puntero. Recuérdese también, que la dirección de la

casilla apuntada por el puntero es el contenido del registro "SP"; las letras "S" y "P", son las iniciales de "Stack Pointer", en Inglés, "Puntero de Pila".

La FIGURA 5-1F, muestra los contenidos después de ejecutar la instrucción "POP IX", vemos otra vez que los datos de la pila no se han perdido, pero para nosotros no existen, ya que el puntero ha vuelto a subir dos casillas y ahora se considera que el último dato de la pila es AAB5h.

Vamos a recuperar, por último, este dato:

POP IV

Esta instrucción, toma el último dato de la pila y lo asigna al registro "IV".

En la FIGURA 5-1G, vemos la situación final, el dato AAB5h ha sido asignado al registro "IV" y el puntero se ha vuelto a incrementar dos veces, para apuntar al mismo si-

tio que lo hacía al principio de estos ejemplos.

Ya hemos sacado de la pila todos los datos que habíamos introducido. El puntero ha quedado en la misma posición donde estaba al principio. Si nuestro ejemplo fuese una subrutina de un programa, en la dirección donde apunta ahora el puntero se encontraría almacenada la dirección de retorno y ahora sería posible retornar al punto desde donde se llamó a esta subrutina.

Con la pila se pueden hacer muchas cosas. Supongamos que en Basic queremos intercambiar el contenido de dos variables "a" y "b", en ese caso, necesitamos generar una tercera variable que nos sirva de "puente", de la forma:

```
10 LET c=a
20 LET a=b
30 LET b=c
```



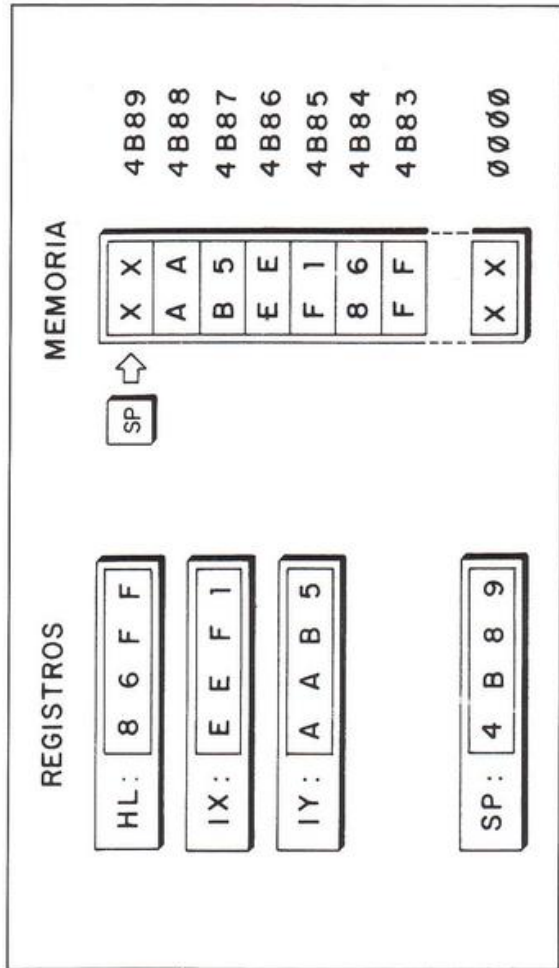


Fig. 5-1G. Contenidos, después de ejecutar la instrucción: «POP IY».

Trabajando en código máquina, podemos intercambiar el contenido de los registros, utilizando la pila en lugar de la variable "puente" del Basic, supongamos que queremos intercambiar los contenidos de los registros "BC" y "DE"; podríamos hacer:

```
PUSH BC
PUSH DE
POP BC
POP DE
```

Por supuesto, esto es sólo un ejemplo, no se quedan aquí las utilidades de la pila. Su función principal es la de salvar temporalmente, el contenido de algún registro, mientras se utiliza para algo y luego, restituirle, de nuevo su contenido. No obstante, con la pila se pueden hacer muchas más cosas, en capítulos posteriores veremos cómo utiliza la pila el intérprete de Basic, para poder retornar desde cualquier punto en caso de error.

curso. Hasta ese momento, suponemos que los programas se ejecutan en un orden puramente secuencial, desde la primera instrucción hasta la última.

### Ejemplos

A continuación, vamos a ver una serie de ejemplos prácticos que el lector podrá introducir en su ordenador, tanto si dispone de ensamblador, como si no.

A través de estos ejemplos, se pretende no sólo aprender a utilizar las instrucciones de carga, sino también, aprender a ensamblar un programa "a mano" y cargarlo desde Basic en cualquier lugar de la memoria.

Antes de eso, y como nota previa, vamos a ver la forma de retornar a Basic desde código máquina cuando finalice la ejecución de cada uno de nuestros programas. En general, llamaremos a nuestros programas con la función USR del Basic, esta función ejecutará nuestras rutinas como si se tratase de subrutinas del sistema operativo, por lo que el procedimiento de retornar a éste, será el mismo que para retornar desde cualquier subrutina, es decir, la instrucción "RET" que se ensambla como C9h (201) y es equivalente al RETURN del Basic.

Quizá esto se comprenda mejor cuando estudiemos las subrutinas en código máquina. Por ahora, nos basta con saber que al final de cada uno de nuestros programas, deberá ir la instrucción RET.

Empecemos por lo más sencillo, vamos simplemente a cargar un número en el re-

gistro "BC". Escogemos este registro, por que es su contenido el que nos devuelve la función USR cuando retornamos a Basic.

Nuestro primer programa en código máquina podría ser el siguiente:

```
LD BC, 27263
RET
```

Que también podría haberse escrito en hexadecimal de la siguiente forma:

```
LD BC, 6A7F
RET
```

Vamos a ensamblar a mano este sencillo ejemplo, y luego lo cargaremos en uno de los lugares que indicábamos en el capítulo 4, el buffer de impresora.

Cogemos las tablas de codificación, y vemos que la instrucción

```
LD BC, nn
```

tiene el código 01(1), de forma que éste será el primer byte de nuestro programa.

A continuación, deberemos poner el operando de dos bytes "nn", con el orden de los octetos invertido. Como en este caso, el operando es 6A7F, deberemos poner primero 7Fh (127) y luego, 6Ah (106). Finalmente, pondremos el código de RET, C9h (201).

Nuestro programa queda, por tanto, de la siguiente forma:

```
01,7F,6A,C9
```

O escrito en decimal:

```
1,127,106,201
```

Hasta ahora, hemos hecho todo esto sobre el papel; por fin llega el momento de poner en marcha nuestro querido Spectrum.

Para introducir los cuatro valores que componen nuestro código máquina, podemos POKEarlos en memoria ayudándonos de un bucle FOR...NEXT:

```
10 FOR N=23296 TO 23299
20 POKE N, USR N
30 NEXT N
40 PRINT
50 GOTO 10
```

Nuestro programa está en los datos de la línea 40, las líneas 10, 20 y 30 los van introduciendo secuencialmente en memoria, finalmente la línea 50 lo ejecuta imprimiendo el resultado de USR en el retorno.

Teclee el programa, revise que no haya habido errores, y pulse RUN...

Si todo ha ido correctamente, deberá ver el número 27263 en la esquina superior izquierda de la pantalla.

No parece un resultado muy espectacular comparado con los prodigios semimágicos que se suelen esperar del código máquina. Ciertamente, no se puede pretender más con cuatro bytes, un simple "PRINT a" de Basic implica la ejecución de cientos de instrucciones en código máquina.

No debe desanimarse el lector ni pretender hacer ma-



Código Fuente	Hexadecimal	Decimal
LD SP, HL	F9	249
LD SP, IX	DD, F9	221, 249
LD SP, IY	FD, F9	253, 249

Fig. 5-8. Grupo de carga de registro «SP».

Código Fuente	Hexadecimal	Decimal
PUSH BC	C5	197
PUSH DE	D5	213
PUSH HL	E5	229
PUSH AF	F5	245
PUSH IX	DD, E5	221, 229
PUSH IY	FD, E5	253, 229
POP BC	C1	193
POP DE	D1	209
POP HL	E1	225
POP AF	F1	241
POP IX	DD, E1	221, 225
POP IY	FD, E1	253, 225

Fig. 5-9. Grupo de manejo de pila.

### Carga del registro "PC"

Seguramente, el lector ya se habrá dado cuenta de que no hemos mencionado en ninguna instrucción al registro "PC"; este hecho se debe a que se trata de un registro especial, que tiene asignada una función muy específica. El registro "PC" o "Contador de Programa", contiene siempre la dirección en memoria de la siguiente instrucción a ejecutar, por lo que el hecho de cargarlo con un número,

implica que la siguiente instrucción será leída desde la posición de memoria apuntada por ese número, es decir, se producirá un salto o bifurcación en el flujo del programa. Vamos a verlo con un ejemplo.

Supongamos que acabamos de leer una instrucción de tres bytes de longitud, que ocupaba las posiciones 40000, 40001 y 40002. En este momento, el registro "PC" contiene el valor 40003 que es la dirección desde donde se leerá la siguiente instrucción.

Si la instrucción que estamos ejecutando, modifica el contenido del "PC", digamos que lo pone a 60000, la siguiente instrucción será leída desde esta dirección, con lo que se habrá producido un salto en el programa.

Los saltos y bifurcaciones tienen una importancia tan grande en cualquier lenguaje, que se ha reservado un grupo de instrucciones para este fin; se trata del grupo de instrucciones de "cambio de secuencia", que se estudiarán en el capítulo 10 de este

Fig. 5-2. Grupo de carga en registros.

Código Fuente	Hexadecimal	Decimal	Código Fuente	Hexadecimal	Decimal
LD A, A	7F	127	LD H, E	63	99
LD A, B	78	120	LD H, H	64	100
LD A, C	79	121	LD H, L	65	101
LD A, D	7A	122	LD L, A	6F	111
LD A, E	7B	123	LD L, B	68	104
LD A, H	7C	124	LD L, C	69	105
LD A, L	7D	125	LD L, D	6A	106
LD B, A	47	71	LD L, E	68	107
LD B, B	48	72	LD L, H	6C	108
LD B, C	41	65	LD L, L	6D	109
LD B, D	42	66	LD A, n	3E, n	62, n
LD B, E	43	67	LD B, n	06, n	6, n
LD B, H	44	68	LD C, n	0E, n	14, n
LD B, L	45	69	LD D, n	16, n	22, n
LD C, A	4F	79	LD E, n	1E, n	30, n
LD C, B	48	72	LD H, n	26, n	38, n
LD C, C	49	73	LD L, n	2E, n	46, n
LD C, D	4A	74	LD A, (HL)	7E	126
LD C, E	4B	75	LD B, (HL)	46	70
LD C, H	4C	76	LD C, (HL)	4E	78
LD C, L	4D	77	LD D, (HL)	56	86
LD D, A	57	87	LD E, (HL)	5E	94
LD D, B	58	88	LD H, (HL)	66	102
LD D, C	51	81	LD L, (HL)	6E	110
LD D, D	52	82	LD A, (IX+d)	DD, 7E, d	221, 126, d
LD D, E	53	83	LD B, (IX+d)	DD, 46, d	221, 70, d
LD D, H	54	84	LD C, (IX+d)	DD, 4E, d	221, 78, d
LD D, L	55	85	LD D, (IX+d)	DD, 56, d	221, 86, d
LD E, A	5F	95	LD E, (IX+d)	DD, 5E, d	221, 94, d
LD E, B	58	88	LD H, (IX+d)	DD, 66, d	221, 102, d
LD E, C	59	89	LD L, (IX+d)	DD, 6E, d	221, 110, d
LD E, D	5A	90	LD A, (IY+d)	FD, 7E, d	253, 126, d
LD E, E	5B	91	LD B, (IY+d)	FD, 46, d	253, 70, d
LD E, H	5C	92	LD C, (IY+d)	FD, 4E, d	253, 78, d
LD E, L	5D	93	LD D, (IY+d)	FD, 56, d	253, 86, d
LD H, A	67	103	LD E, (IY+d)	FD, 5E, d	253, 94, d
LD H, B	68	96	LD H, (IY+d)	FD, 66, d	253, 102, d
LD H, C	61	97	LD L, (IY+d)	FD, 6E, d	253, 110, d
LD H, D	62	98			



Código Fuente	Hexadecimal	Decimal
LD (HL), A	77	119
LD (HL), B	78	112
LD (HL), C	71	113
LD (HL), D	72	114
LD (HL), E	73	115
LD (HL), H	74	116
LD (HL), L	75	117
LD (IX+d), A	DD, 77, d	221, 119, d
LD (IX+d), B	DD, 78, d	221, 112, d
LD (IX+d), C	DD, 71, d	221, 113, d
LD (IX+d), D	DD, 72, d	221, 114, d
LD (IX+d), E	DD, 73, d	221, 115, d
LD (IX+d), H	DD, 74, d	221, 116, d
LD (IX+d), L	DD, 75, d	221, 117, d
LD (IY+d), A	FD, 77, d	253, 119, d
LD (IY+d), B	FD, 78, d	253, 112, d
LD (IY+d), C	FD, 71, d	253, 113, d
LD (IY+d), D	FD, 72, d	253, 114, d
LD (IY+d), E	FD, 73, d	253, 115, d
LD (IY+d), H	FD, 74, d	253, 116, d
LD (IY+d), L	FD, 75, d	253, 117, d
LD (HL), n	36, n	54, n
LD (IX+d), n	DD, 36, d, n	221, 54, d, n
LD (IY+d), n	FD, 36, d, n	253, 54, d, n

Fig. 5-3. Grupo de carga de memoria.

Código Fuente	Hexadecimal	Decimal
LD A, (BC)	0A	10
LD A, (DE)	1A	26
LD A, (nn)	3A, n, n	58, n, n
LD A, I	ED, 57	237, 87
LD A, R	ED, 5F	237, 95

Fig. 5-4. Grupo de carga en acumulador.

Código Fuente	Hexadecimal	Decimal
LD (BC), A	02	2
LD (DE), A	12	18
LD (nn), A	32, n, n	50, n, n
LD I, A	ED, 47	237, 71
LD R, A	ED, 4F	237, 79

Fig. 5-5. Grupo de salvar acumulador.

Código Fuente	Hexadecimal	Decimal
LD BC, nn	01, n, n	1, n, n
LD DE, nn	11, n, n	17, n, n
LD HL, nn	21, n, n	33, n, n
LD SP, nn	31, n, n	49, n, n
LD IX, nn	DD, 21, n, n	221, 33, n, n
LD IY, nn	FD, 21, n, n	253, 33, n, n
LD HL, (nn)	2A, n, n	42, n, n
LD BC, (nn)	ED, 48, n, n	237, 75, n, n
LD DE, (nn)	ED, 58, n, n	237, 91, n, n
LD SP, (nn)	ED, 78, n, n	237, 123, n, n
LD IX, (nn)	DD, 2A, n, n	221, 42, n, n
LD IY, (nn)	FD, 2A, n, n	253, 42, n, n

Fig. 5-6. Grupo de carga en registros de 16 bits.

Código Fuente	Hexadecimal	Decimal
LD (nn), HL	22, n, n	34, n, n
LD (nn), BC	ED, 43, n, n	237, 67, n, n
LD (nn), DE	ED, 53, n, n	237, 83, n, n
LD (nn), SP	ED, 73, n, n	237, 115, n, n
LD (nn), IX	DD, 22, n, n	221, 34, n, n
LD (nn), IY	FD, 22, n, n	253, 34, n, n

Fig. 5-7. Grupo de carga en memoria de 16 bits.



desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. Deja el resultado en el registro acumulador.

#### CODIGO DE MAQUINA:

FDh	1 1 1 1 1 1 0 1
8Sh	1 0 0 0 0 1 1 0
	< ———— 0 ———— >

#### INDICADORES DE CONDICION A LOS QUE AFECTA:

- S** ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso
- Z** ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso
- H** ; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso
- N** ; pone 0 - siempre
- C** ; pone 1 - si hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso
- P/V** ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

#### CICLOS DE MEMORIA:

5

#### CICLOS DE RELOJ:

19

#### EJEMPLO:

AUD A, (V-6)
--------------

En esta ocasión vamos a usar, de nuevo, direccionamiento indexado para acceder al operando. Los dos nú-

meros a sumar serán 7Fh y 03H, el resultado será 82h (F más 3 = 2 y nos llevamos una; 7 más 0 más 1 = 8). Como se ve, sumar en hexadecimal es lo mismo que hacerlo en decimal.

#### Valor del registro "Y"

(Y)	1 0 1 0 0 0 1 1
A3h	0 1 1 1 1 0 0 0
78h	

#### Valor de la posición de memoria A372h

(A372h)	0 1 1 1 1 1 1 1
Fh	

#### Valor del registro "A"

(A)	0 0 0 0 0 0 0 1
03h	

#### Instrucción

FDh	1 1 1 1 1 0 1
8Sh	1 0 0 0 0 1 1 0
FAh	1 1 1 1 1 0 1 0
AUD A, (V-6)	

#### Valor del registro "A" después de la ejecución

(A)	1 0 0 0 0 0 1 0
82h	

#### Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
1	0	1	1	0	0

Observe, que el indicador de signo (S) se activa por estar activo el bit 7, el tratar este número como negativo o no, dependerá del programador. El indicador P/V se activa por superar el máximo valor del octeto en complemento a dos (el bit de signo ha pasado de "0" a "1"). Finalmente, el indicador H está a "1" porque hubo acarreo desde el bit 3.

Las instrucciones de sumar, como su nombre indica, suman; pero con lo visto hasta el momento sólo suman un octeto. Por lo tanto se limita la suma al número 255, considerando todos positivos.

Este problema se soluciona con las instrucciones que se explican a continuación.

ADC (ADD with Carry), sumar con acarreo. Básicamente consiste en una suma binaria de dos octetos más el bit de acarreo. Esto quiere decir que si en una suma anterior el bit de acarreo está activo, "nos llevamos una", esa unidad hay que tenerla en cuenta en el octeto superior si existe.

Por ejemplo en una suma convencional en decimal.

7328+4256
-----------

al sumar las unidades 8 y 6 nos llevamos un 1 a las decenas; con las decenas y las centenas no hay acarreo, y de nuevo en las unidades de millar hay acarreo a las decenas de millar.

1 1	acarreo
7328	
+ 4256	
11584	

Visto esto, se entenderá fácilmente, que sumando octetos se acarrea 1 al octeto superior cuando se supera el valor decimal 255 (FFh). Ver FIGURA 6-1.

Esta es la manera de sumar, en binario, cantidades superiores a 255 decimal; usando las instrucciones que se describen a continuación.

LD HL, (#5C82)

PUSH HL  
POP BC  
RET

Ahora, codificamos el programa, buscamos en las tablas el código de:

LD HL, (nn)
-------------

que resulta ser 2Ah (42), a continuación van los operandos B2h (178) y 5C (92). El código de:

PUSH HL
---------

es E5h (229), y el de:

POP BC
--------

es C1h (193). Por último, colocamos el código de RET: C9h (201). El programa completo, queda de la siguiente forma:

2A,B2,5C,E5,C1,C9
-------------------

O, para quienes lo prefieran en decimal:

42,178,92,229,193,201
-----------------------

Ahora, sólo nos falta cargarlo en una línea REM de un programa en Basic. Nuestra rutina tiene 6 bytes, por lo que crearemos una línea REM con, por ejemplo, 6 asteriscos. Estos asteriscos serán sustituidos por los bytes del programa cuando éste se cargue.

El programa en Basic podría ser el siguiente:

```
1000 REM ***PEEK 23635+256*PEE
1001 FOR n=5 TO 10
1002   FOR a=0 TO 15
1003     NEXT a
1004   NEXT n
1005   PRINT USR (PROG+5)
```

Hay muchos puntos sutiles en este programa que conviene analizar detenidamente; como dijimos antes, la línea 10 contiene el espacio donde se cargará nuestra rutina en C/M.

En la línea 20, leemos la variable del Sistema PROG, para saber a partir de qué dirección de memoria está ubicada el programa Basic. Los dos primeros bytes de esta zona, constituyen el número de línea, los dos siguientes la longitud, y el quinto es el código de REM; a partir de ahí em- piezan los asteriscos, que es donde deberemos cargar el código máquina, es decir, desde "prog+5" hasta "prog+10" tal y como se ve en las líneas 30, 40 y 50. La línea 60 contiene el programa en DATAS. Finalmente, la línea 70 ejecuta el programa desde la dirección "prog+5". En este caso, es imprescindible que el argumento de USR vaya entre paréntesis; es muy fácil omitir los paréntesis, olvidándose de que la función USR tiene una prioridad más alta que la suma.

Una vez ejecutado el programa, la línea 10 quedaría:

10 REM \*SIN ^ RESTORE STR\$ <<>

Nuestro siguiente ejemplo es más vistoso, y algo más complejo. Vamos a dibujar una silueta en pantalla, y dado que la cosa va de pantalla,

almacenaremos esta rutina en el archivo de presentación visual, con lo que veremos físicamente los bytes que la componen, en forma de pixels en la primera línea.

El objetivo del programa es dibujar en la casilla central, la silueta de un muñeco, como si se tratara de un UDG. Dado que sólo podemos utilizar instrucciones de carga, el programa resulta considerablemente más largo de lo que es normal para trabajar con la pantalla.

En sucesivos ejemplos de capítulos más avanzados, iremos viendo otras formas más sencillas de imprimir en pantalla; y veremos también, cómo la peculiar manera en que está organizado el archivo de pantalla, que tan incómoda se hacía en Basic, resulta una gran ventaja cuando se trabaja en código máquina.

La forma más sencilla de imprimir un gráfico en pantalla, es almacenar en las ocho direcciones que componen una casilla, los ocho números que definen ese gráfico. En la FIGURA 5-10, vemos las direcciones de las posiciones de memoria correspondien-

tes a la casilla central de la pantalla, así como los datos que vamos a almacenar en esas posiciones, para visualizar nuestro muñeco.







Indicadores de condición después de la ejecución:

S	Z	H	P/V	N	C	
0	0	x	1	x	0	0

Observe, que hubo acarreo desde el bit 3.

**ADD A,n**

OBJETO:

Suma el registro acumulador "A" con el número entero de 8 bits "n", dejando el resultado en el registro acumulador.

CODIGO DE MAQUINA:

1	1	0	0	1	1	0	0	6h
<	—	n	—	>				

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso

H ; pone 1 - si hay acarreo desde el bit 3

pone 0 - en cualquier otro caso

N ; pone 0 - siempre  
C ; pone 1 - si hay acarreo desde el bit 7

pone 0 - en cualquier otro caso

P/V ; pone 1 - si hay desbordamiento (overflow)

pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

ADD A,24

Valor del registro "A":

(A): 0 0 1 0 0 1 1 0 26h

Instrucción

1	1	0	0	0	1	1	0	6h
0	0	0	1	1	0	0	0	18h

Valor del registro "A" después de la ejecución:

(A): 0 0 1 1 1 1 1 0 3eh

Indicadores de condición después de la ejecución:

S	Z	H	P/V	N	C
0	0	x	0	x	0

**ADD A,(HL)**

OBJETO:

Suma el registro acumulador "A" con el octeto de la posición de memoria direccionada por el contenido del par de registros "HL", y deja el resultado en el registro acumulador.

CODIGO DE MAQUINA:

1 0 0 0 0 1 1 0 86 h

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

Z ; pone 1 - si el resultado es cero

do es cero  
pone 0 - en cualquier otro caso

H ; pone 1 - si hay acarreo desde el bit 3

pone 0 - en cualquier otro caso

N ; pone 0 - siempre

C ; pone 1 - si hay acarreo desde el bit 7

otro caso

P/V ; pone 1 - si hay desbordamiento (overflow)

pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

ADD A,(HL)

Valor del par de registros "HL":

(H):	0	1	0	0	1	1	0	0	4Ch
(L):	1	1	1	1	0	0	1	1	F3h

Valor de la posición de memoria 4CF3h:

(4CF3h): 1 0 0 1 1 1 0 0 9Ch

Valor del registro "A":

(A): 1 0 1 1 0 0 0 0 80h

Instrucción:

ADD A,(HL) 1 0 0 0 0 1 1 0 86h

Valor del registro "A" después de la ejecución:

(A): 0 1 0 0 1 1 0 0 46h

Indicadores de condición después de la ejecución:

ne el código máquina en hexadecimal, y la 120 la suma de todos los bytes en decimal, que se utiliza como suma de comprobación.

Cuando se ejecute el programa, en la pantalla del ordenador tiene que aparecer algo similar a lo que se ve en la FIGURA 5-11.

En el siguiente capítulo, veremos las instrucciones que nos permiten realizar operaciones aritméticas y lógicas sobre los registros del microprocesador. Antes de ello, le recomendamos al lector que intente resolver los siguientes ejercicios, que le ayudarán a afianzar conocimientos.

## EJERCICIOS

1.- ¿Que valor retornará el siguiente programa en el registro "BC"? ¿Que valor tendremos en la variable del sistema "SEED" después de ejecutarlo?:

```
LD HL,#4BFF
LD BC,#1100
LD H,C
LD (SEED),BC
PUSH HL
POP BC
RET
SEED EQU #5C76
```

(Puede encontrar la solución por usted mismo, codificando el programa y ejecutándolo en el ordenador, para comprobar si la solución que ha dado es correcta).

2.- Codificar (ensamblar) el siguiente programa:

```
LD BC,#1234
LD A,(BC)
LD (1X+7),A
PUSH AF
POP BC
RET
```

3.- Escribir cuatro cargadores en Basic, cada uno de los cuales almacenen el programa anterior en uno de los siguientes lugares:

```
EN EL BUFFER DE IMPRESORA
ENCIMA DE RAMTOP
EN UNA LINEA REM
EN LA PANTALLA
```



## INSTRUCCIONES ARITMETICAS Y LOGICAS

El microprocesador Z-80 dispone de una unidad aritmética-lógica que le permite realizar una serie de operaciones, tanto aritméticas, como lógicas. Las aritméticas incluyen la suma y resta con o sin acarreo, incremento y decremento de un registro, comparaciones, ajuste decimal, complemento y negación. Las lógicas incluyen las operaciones que se realizan con los operadores "AND", "OR" y "XOR".

Antes de adentrarnos en el estudio de las instrucciones concretas, daremos una serie de definiciones útiles:

### SUMA SIN ACARREO:

Consiste en sumar al contenido del registro "A" un número y obtener el resultado en el registro "A". El indicador de acarreo no se tiene en cuenta para esta operación. Su esquema sería:

$$A \leftarrow A+n$$

### SUMA CON ACARREO:

Exactamente igual que la anterior, pero se suma también el indicador de acarreo del registro "F". De esta forma, se puede incluir en la suma el acarreo procedente de una suma anterior. Su esquema sería:

$$A \leftarrow A+n+CF$$

### RESTA SIN ACARREO:

Consiste en restar un número del contenido del registro "A", y obtener el resultado en este mismo registro. El indicador de acarreo no interviene en la operación. Se consideran números negativos los superiores a 127 (7Fh) de la forma que se explicó en el capítulo relativo a los sistemas de numeración; es decir, el número 255 (FFh) se considera "-1", el 254 (FEh) se considera "-2" y así sucesivamente, hasta 128 (80) que se considera "-128". El paso de 127 a 128 o viceversa se indica poniendo a "1" el flag de "overflow" (P/V) del registro "F". Su esquema sería:

$$A \leftarrow A-n$$

### RESTA CON ACARREO:

Igual que el anterior, salvo que también se resta el indicador de acarreo (CF) del registro "F". Su esquema sería:

$$A \leftarrow A-n-CF$$

### INCREMENTO:

Consiste en sumar uno al contenido de un registro que se especifica en la instrucción. Su esquema es:

$$R \leftarrow R+1$$

Donde "R" representa un registro cualquiera de 8 a 16 bits. Si se trata de un registro doble (de 16 bits) se incrementa el registro de orden ba-

jo (por ejemplo, en el "BC" se incrementa "C"), y si ello hace que éste pase a valer "0", se incrementa también el orden alto.

### DECREMENTO:

Es la inversa de la anterior, consiste en restar uno al contenido de un registro. Su esquema es:

$$R \leftarrow R-1$$

Si se trata de un registro doble, se decrementa el de orden bajo y, si esto hace que pase a valer 255 (FFh), se decrementa también el de orden alto.

Si el registro incrementado o decrementado es de 8 bits, resultan afectados los indicadores del registro "F".

### COMPARACIONES:

Estas instrucciones permiten comparar el contenido del acumulador con un número. Para ello, se resta el número del contenido del acumulador, pero el resultado no se almacena en ninguna parte, simplemente, se alteran determinados flags del registro "F", lo que nos indica si el número era menor, igual o mayor que el contenido del acumulador. Si era igual, se pone a "1" el flag "Z" (indicador de "cero"). Si el número era mayor, se pone a "1" el flag "S" (indicador de "signo").

### AJUSTE DECIMAL:

Esta instrucción realiza un ajuste del contenido del acumulador para que, en vez de estar comprendido entre "00h" y "FFh", lo esté entre "00h" y "99h". Si se produce acarreo, se indica mediante el flag correspondiente. Para realizar esta operación se toma en cuenta el estado de los indicadores de "acarreo" (C) y "semi-acarreo" (H). Su finalidad es la de permitir realizar operaciones en "BCD" (Decimal Codificado en Binario).

### COMPLEMENTO:

Consiste en realizar un "complemento a 1" del acumulador, es decir, cambiar los "unos" por "ceros" y los "ceros" por "unos".

### NEGACION:

Consiste en realizar un "complemento a 2" del acumulador, es decir, realizar un "complemento 1" y, luego, sumarle "1". Lo que se obtiene es el "negativo" del número que teníamos en el acumulador. El efecto es el mismo que si restáramos el acumulador de "cero", es decir:

$$A \leftarrow 0-A$$

### EL FLAG DE ACARREO:

Existen dos instrucciones que afectan al indicador de acarreo del registro "F", es posible ponerlo a "1" o "complementarlo" (ponerlo a "1" si era "0" y viceversa). No se ha previsto una instrucción para poner a "0" el flag de acarreo, dado que esto se puede conseguir haciendo un "AND" o un "OR" del acumulador consigo mismo. Veamos ya las instrucciones:

### Grupo de instrucciones aritméticas para 8 bits

En este grupo de instrucciones los registros usados se indican con "r" según el siguiente código:

"r"	código
A	111
B	000
C	001
D	010
E	011
H	100
L	101

pone 0 - en cualquier otro caso  
N ; pone 0 - siempre  
C ; pone 1 - si hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso  
P/V ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

NOTA: Se entiende que hay acarreo desde el bit 3 cuando éste pasa de ser "1" a ser "0". Se entiende que hay desbordamiento si el resultado pasa de ser "positivo" a ser "negativo" o viceversa. Estas observaciones son válidas para todas las operaciones aritméticas.

### CICLOS DE MEMORIA:

1

### CICLOS DE RELOJ:

4

### EJEMPLO:

$$ADD\ A,B$$

Valor del registro "A":

$$[A]: \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline \end{array} \quad 29h$$

Valor del registro "B":

$$[B]: \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array} \quad 4Ah$$

Instrucción:

$$ADD\ A,B: \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \quad 80h$$

Valor del registro "A" después de la ejecución:

$$[A]: \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ \hline \end{array} \quad 73h$$

El valor del registro "B" después de la ejecución no varía.



Valor del registro "A" después de la ejecución

(A): 0 1 0 1 1 0 1 1 58h

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 0 x 1 x 1 1 0

En este caso, hemos restado 91h menos 36h, obteniendo 5Bh. El número 91h puede ser negativo (-111) y 5Bh es siempre positivo, de forma que el microprocesador nos pone a "1" el indicador "P/V". Por si estábamos considerando los números como negativos.

**SUB (IY+d)**

OBJETO:

Resta al contenido del registro acumulador, el valor del octeto de memoria direccionado por el valor que resulta de: añadir al contenido del registro índice IY el entero de desplazamiento d, el cual puede adquirir los valores desde -128 a +127.

CODIGO DE MAQUINA:

FDh 96h  
1 1 1 1 1 1 0 1  
1 0 0 1 0 1 1 0

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z ; pone 1 - si el resultado es cero

pone 0 - en cualquier otro caso  
H ; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N ; pone 1 - siempre  
C ; pone 1 - si no hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso  
P/V; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

SUB (IY-34)

Valor del registro "IY"

IY: AAh 49h  
1 0 1 0 1 0 1 0  
0 1 0 0 1 0 0 1

Valor de la posición de memoria AA27h

(AA27h): 35h  
0 0 1 1 0 1 0 1

Valor del registro "A"

(A): 35h  
0 0 1 1 0 1 0 1

Instrucción

FDh 96h 0Ch  
1 1 1 1 1 1 0 1  
1 0 0 1 0 1 1 0  
1 1 0 1 1 1 1 0

Operación:

00110101  
+ 11001011  
-----  
00000000

Valor del registro "A" después de la ejecución

(A): 00000000 00h

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 1 x 0 x 0 1 0

En la suma que ha hecho el microprocesador ha habido acarreo y semiacarreo ("H"), pero como en la resta los acarreos se invierten, los indicadores "C" y "H" permanecen a "0".

-----

Con las instrucciones de restar vistas hasta el momento, la operación está limitada por el valor que puede contener un octeto, es decir, 255 considerando todos los valores como positivos. Para restar números más grandes, están las instrucciones cuyo nemotécnico es SBC.

SBC (Subtract with Carry), restar con acarreo. Se trata de una resta binaria de un octeto, con las mismas características que en las instrucciones SUB, sólo que al minuyendo se le resta primero el bit de acarreo del indicador de condición (C). Este bit se activa en una operación de resta cuando no hay acarreo desde el bit 7 y esto ocurre cuando el sustraendo es mayor que el minuendo (ver ejemplos de las instrucciones SUB).

**ADC A,r**

OBJETO:

Suma el registro acumulador "A", más el bit de acarreo, con el registro indicado por "r", dejando el resultado en el registro acumulador.

CODIGO DE MAQUINA:

1 0 0 1 <----->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H ; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso

N ; pone 0 - siempre  
C ; pone 1 - si hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso  
P/V ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

ADC A,D 6Ch  
0 1 1 0 1 1 0 0

Suponemos que tenemos

BINARIO  
11000110 10011100 01001001 10000000  
00011000 10110000 00111010 10000000  
0 + 11011111 01001100 10000100 00000000  
-----  
11011111 01001100 10000100 00000000

HEXADECIMAL

C6 9C 49 80 SUMANDO  
18 B0 3A 80 SUMANDO  
+ 1 0 1 ACARREO  
-----  
DF 4C 84 00 RESULTADO

DECIMAL

198 156 73 128 SUMANDO  
24 176 58 128 SUMANDO  
+ 1 0 1 ACARREO  
-----  
223 76 132 0 RESULTADO

Fig. 6-1. Suma de varios objetos con acarreo.

el flag de acarreo a "1", puede ser, por ejemplo, como resultado de una suma anterior. Por tanto, vamos a sumar 49h + 22h + 1. El resultado debe ser 6Ch.

Valor del registro "A"

(A): 49h  
0 1 0 0 1 0 0 1

Valor del registro "D"

(D): 22h  
0 0 1 0 0 1 0 0

Bit de acarreo = 1  
Instrucción

ADC A,D 8Ah  
1 0 0 0 1 0 1 0

Valor del registro "A" después de la ejecución

(A): 6Ch  
0 1 1 0 1 1 0 0

Indicadores de condición

BINARIO  
11000110 10011100 01001001 10000000  
00011000 10110000 00111010 10000000  
0 + 11011111 01001100 10000100 00000000  
-----  
11011111 01001100 10000100 00000000

HEXADECIMAL

C6 9C 49 80 SUMANDO  
18 B0 3A 80 SUMANDO  
+ 1 0 1 ACARREO  
-----  
DF 4C 84 00 RESULTADO

DECIMAL

198 156 73 128 SUMANDO  
24 176 58 128 SUMANDO  
+ 1 0 1 ACARREO  
-----  
223 76 132 0 RESULTADO

después de la ejecución

S Z H P/V N C  
0 0 x 0 x 0 0 0

Esta vez no ha habido acarreo, semi-acarreo ni cambio de signo.

**ADC A,n**

OBJETO:

Suma el registro acumulador "A", más el bit de acarreo, con el número entero de 8 bits "n", dejando el resultado en el registro acumulador.

CODIGO MAQUINA:

CCh  
1 1 0 0 1 1 1 0  
<----->



## INDICADORES DE CONDICION A LOS QUE AFECTA:

- S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso
- Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso
- H ; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso
- N ; pone 0 - siempre  
C ; pone 1 - si hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso

- P/V ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

2

## CICLOS DE RELOJ:

7

## EJEMPLO:

ADC A, 120
------------

Vamos a sumar A5h (165) más 78h (120) más 1, porque suponemos que el indicador de acarreo está a "1". El resultado deberá ser 11Eh (286); puesto que este resultado excede la capacidad del acumulador, el indicador de acarreo se pondrá a "1" y el acumulador contendrá 1Eh (30). Podemos comprobar que  $256 + 30 = 286$ , por tanto, el resultado es correcto.

Valor del registro "A"

(A):	1 0 1 0 0 1 0 1	A5h
------	-----------------	-----

Bit de acarreo = 1  
Instrucción

ADC A, 120	Cfh	/8h
11001110		
01111000		

Valor del registro "A" después de la ejecución

(A)	0 0 0 1 1 1 1 0	1Eh
-----	-----------------	-----

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	0	x	0
0	1	0	0	1	1

Como indicábamos antes, se nos ha "levantado" el flag de acarreo para indicar que el verdadero resultado es 256 más el contenido del acumulador. Si ahora sumáramos otros dos octetos de orden superior a estos, deberíamos tener en cuenta el acarreo.

## ADC A, (HL)

## OBJETO:

Suma el registro acumulador "A", más el bit de acarreo, con el octeto de la posición de memoria direccionada por el contenido del par de registros "HL", y deja el resultado en el registro acumulador.

## CODIGO DE MAQUINA:

1 0 0 0 1 1 1 0	8Eh
-----------------	-----

INDICADORES DE CONDICION A LOS QUE AFECTA:

- S ; pone 1 - si el resultado es negativo

- pone 0 - en cualquier otro caso
- Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso
- H ; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso
- N ; pone 0 - siempre  
C ; pone 1 - si hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso

- P/V ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

2

## CICLOS DE RELOJ:

7

## EJEMPLO:

ADC A, (HL)
-------------

Valor del par de registros "HL"

(H)	0 1 1 1 0 1 1 0	76h
(L)	1 0 0 0 1 1 1 0	8Eh

Valor de la posición de memoria 768Eh

(768Eh):	1 0 0 0 1 1 1 0	80h
----------	-----------------	-----

Valor del registro "A"

(A):	0 0 1 0 1 0 0 1	29h
------	-----------------	-----

## Instrucción

ADC A, (HL)	1 0 0 0 1 1 1 0	8Eh
-------------	-----------------	-----

Valor del registro "A" después de la ejecución

Sabemos que nos es posible trabajar con números negativos si consideramos negativo a todo número cuyo bit de más a la izquierda sea "1". En el caso de trabajar sólo con positivos, el rango permitido va de "0" a "FFh" (255) y si, tras una suma, el número resultante pasa de este rango, se nos pone a "1" el indicador de acarreo "C". Por otro lado, es posible que estemos trabajando con números positivos y negativos, en este caso, el rango permitido es de "80h" (-128) a "7Fh" (+127). El microprocesador nunca sabe cómo vamos a considerar el número contenido en el acumulador, así que, por si acaso, nos indica también si está fuera de este rango, poniendo a "1" el indicador de rebasamiento "P/V".

Vemos, por tanto, que "P/V" se pondrá a "1" siempre que un número pase de ser positivo a ser negativo, o viceversa, sin pasar por cero. Para verlo más claro, representemos todos los valores posibles en una recta que vaya desde "0" a "255", en mitad de la recta, tenemos el número "127". Cuando el contenido del acumulador pase de una mitad de la recta a otra a través del número "127", se pone a "1" el indicador "P/V" y cuando lo haga a través del número "255" y "0", se pone a "1" el indicador "C".

Veamos un ejemplo: Tenemos en el acumulador el número "7Fh" (127) y le sumamos "1", el resultado será "80h" que puede ser 128 o -128, según consideremos el número como positivo o como negativo; en este caso, se nos habrá puesto a "1" el indicador "P/V". Supongamos ahora que tenemos el número

"FFh" que puede ser 255 o -1; si le sumamos "1", obtenemos, como resultado "0"; en este caso, se habrá puesto a "1" el indicador de acarreo "C".

Otro tanto ocurriría si restáramos "1" a "80h", obtendríamos "7Fh" y "P/V" a "1". O si restáramos "1" a "0", resultaría "FFh" y el indicador "C" a "1". Cuando trabajemos sólo con números positivos, tendremos que tomar en cuenta el indicador "C", y cuando lo hagamos con números positivos y negativos, tomaremos en cuenta el indicador "P/V".

## SUB (IX+d)

## OBJETO:

Resta al contenido del registro acumulador, el valor del octeto de memoria direccionado por el valor que resulta de: añadir al contenido del registro índice IX el entero de desplazamiento d, el cual puede adquirir los valores desde -128 a +127.

## CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1	DDh
1 0 0 1 0 1 1 0	96h

INDICADORES DE CONDICION A LOS QUE AFECTA:

- S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso
- Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso
- H ; pone 1 - si no hay acarreo

- reore desde el bit 3  
pone 0 - en cualquier otro caso
- N ; pone 1 - siempre  
C ; pone 1 - si no hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso
- P/V ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

5

## CICLOS DE RELOJ:

19

## EJEMPLO:

SUB (IX+24)
-------------

Valor del registro "IX"

(IX):	1 0 0 0 0 1 0 1	85h
	0 1 1 1 0 0 1 1	73h

Valor de la posición de memoria 858Bh

(858Bh):	0 0 1 1 0 1 1 0	36h
----------	-----------------	-----

Valor del registro "A"

(A):	1 0 0 1 0 0 0 1	91h
------	-----------------	-----

## Instrucción

SUB (IX+24)	11011101	DDh
	10010110	96h
	00011000	18h

## Operación:

10010001	
+ 11001010	
-----	
01011011	



ser negativo. Veamos qué ocurre:

Valor del registro "A"

(A): 00000101 05h

Instrucción

SUB 12 11010110 06h  
00001100 0Ch

Operación:

```

00000101
+ 00001100
-----
11111001

```

Valor del registro "A" después de la ejecución

(A): 11111001 F9h

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 1 x 0 1 1

Observamos que cuando el minuendo es menor que el sustraendo no hay acarreo en la suma que realiza el microprocesador, por tanto, se activa el indicador "C" que, como sabemos, va invertido cuando se resta.

El resultado de la operación es F9h, es decir, -7 expresado en complemento a 2.

SUB (HL)

OBJETO:

Resta del registro acumulador el valor del octeto de memoria direccionado por el contenido del par de registros "HL". El resultado se deja en el registro acumulador.

CODIGO DE MAQUINA:

10010110 96h

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H ; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N ; pone 0 - siempre  
C ; pone 1 - si no hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso  
P/V; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

SUB (HL)

Valor del par de registros "HL"

(H) 01111011 7Bh  
(L) 01100010 62h

Valor de la posición de memoria 7B62h

(7B62h): 00111010 3Ah

Valor del registro "A"

(A): 01101111 6Fh

Instrucción

SUB (HL) 10010110 96h

Operación:

```

01101111
+ 11000110
-----
00110101

```

Valor del registro "A" después de la ejecución

(A): 00110101 35h

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 0 x 0 x 0 1 0

El funcionamiento del indicador "P/V" (Paridad/Rebosamiento) requiere una explicación por tratarse de uno de los puntos más oscuros de la programación en C/M. Este indicador tiene una doble función, en las operaciones lógicas actúa como indicador de paridad y en las aritméticas, como indicador de rebosamiento. Su función como indicador de paridad se verá cuando estudiemos las operaciones lógicas; ahora vamos a ver cómo actúa para indicarnos un rebosamiento.

(A): 10110111 87h

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 1 x 0 0 0

ADC A, (IX+d)

OBJETO:

Suma el registro acumulador "A", más el bit de acarreo, con el octeto de la posición de memoria direccionada por el valor que resulta de: añadir al contenido del registro índice "IX" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. Deja el resultado en el registro acumulador.

CODIGO MAQUINA:

11011101 0Dh  
10001110 8Eh

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H ; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N ; pone 0 - siempre  
C ; pone 1 - si hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso

otro caso  
P/V; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

ADC A, (IX-3)

Valor del registro "IX"

(IX): 10000111 87h  
10101010 AAh

Valor de la posición de memoria 87A7h

(87A7h): 00000001 01h

Valor del registro "A"

(A): 01111111 7Fh

Bit de acarreo = 0

Instrucción

11011101 0Dh  
10001110 8Eh  
11111101 FDh

Valor del registro "A" después de la ejecución

(A): 10000000 80h

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 1 x 1 0 0

ADC A, (Y+d)

OBJETO:

Suma el registro acumulador "A", más el bit de acarreo, con el octeto de la posición de memoria direccionada por el valor que resulta de: añadir al contenido del registro índice "Y" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. Deja el resultado en el registro acumulador.

CODIGO DE MAQUINA:

11111101 FDh  
10001110 8Eh

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H ; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N ; pone 0 - siempre  
C ; pone 1 - si hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso  
P/V; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19



# EJEMPLO:

ADC A, (Y+25)

Valor del registro "Y"

(Y):  
 F0h: 1 1 1 1 0 0 0 0  
 91h: 1 0 0 1 0 0 0 1

Valor de la posición de memoria F0AAh

(F0AAh): 0 0 0 0 1 1 1 1

Valor del registro "A"

(A): 1 1 1 1 0 0 0 0

Bit de acarreo = 0

Instrucción

ADC A, (Y+25)  
 F0h: 1 1 1 1 1 0 1  
 8Ch: 1 0 0 0 0 1 1 0  
 19h: 0 0 0 1 1 0 0 1

Valor del registro "A" después de la ejecución

(A): 1 1 1 1 1 1 1 1

Indicadores de condición después de la ejecución

S Z H P/V N C  
 1 0 x 0 x 0 0 0

La actividad de los indicadores, de condición, tanto en la suma (ADD) como en la su-  
 ma con acarreo (ADC), se ha-  
 ce según las siguientes con-  
 diciones:

"S": En este indicador se pone el mismo valor que ten-  
 ga el bit siete del acumulador después de la ejecución.  
 "Z": Este indicador se acti-  
 va, valor igual a 1, siempre

que todos los bits del registro acumulador sean cero des-  
 pués de la ejecución.  
 "H": Este indicador de aca-  
 rreo desde el bit 3, se activa  
 siempre que los cuatro bits  
 inferiores del registro acumu-  
 lador superen el valor Fh (15  
 decimal) después de la eje-  
 cución. Esto es independien-  
 te del valor que tengan los  
 cuatro bits superiores.

"N": Este indicador no tiene  
 significado para este grupo  
 de instrucciones y se pone  
 siempre a 0.  
 "C": Este indicador de aca-  
 rreo desde el bit 7, se activa  
 siempre que el registro acu-  
 mulador supere el valor FFh  
 (255 decimal) después de la  
 ejecución.  
 "P/V": Este indicador de  
 desbordamiento (overflow),  
 se activa si, después de la  
 ejecución, el registro acumu-  
 lador supera el valor +127 o  
 -128. Esto es, indica el cam-  
 bio de signo del número en  
 complemento a 2.

SUB (SUBtract), "restar" en  
 inglés. Básicamente esta fun-  
 ción consiste en restar del re-  
 gistro acumulador el valor in-  
 dicado por el operando. Esto  
 es una resta binaria en la que  
 el registro acumulador es el  
 minuendo y el operando indi-  
 ca el sustraendo.  
 La operación real que efec-  
 tua el microprocesador es:  
 complementar a dos el sus-  
 traendo y sumarlo con el mi-  
 nuendo. Conocer esta opera-  
 tiva es interesante para en-  
 tender cómo funciona el aca-  
 rreo, pero no es necesario  
 tenerla presente en el mo-  
 mento de construir el progra-  
 ma.  
 En una resta algebraica el  
 sustraendo es un número ne-  
 gativo y como se sabe para el  
 microprocesador Z80, los nú-

S ; pone 1 - si el resultado  
 es negativo  
 pone 0 - en cualquier  
 otro caso

Z ; pone 1 - si el resultado  
 es cero  
 pone 0 - en cualquier  
 otro caso

H ; pone 1 - si hay acarreo  
 desde el bit 3  
 pone 0 - en cualquier  
 otro caso

N ; pone 1 - siempre  
 C ; pone 1 - si hay acarreo  
 desde el bit 7  
 pone 0 - en cualquier  
 otro caso

P/V; pone 1 - si hay desbor-  
 damiento (overflow)  
 pone 0 - en cualquier  
 otro caso

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

SUB B

En este ejemplo, vamos a  
 restar el contenido del regis-  
 tro "B" del contenido del acu-  
 mulador "A". La operación se  
 podría representar esquemá-  
 ticamente como:

A ← A-B

Valor del registro "A"

(A): 0 0 1 1 0 1 0 1 35h

Valor del registro "B"

(B): 0 0 0 1 0 0 1 1 13h

Instrucción

SUB B 1 0 0 1 0 0 0 0 90h

Operación

00110101  
 + 11011011  
 -----  
 00100010

El valor del registro "B" des-  
 pués de la ejecución, no va-  
 ría.

Valor del registro "A" des-  
 pués de la ejecución

(A): 0 0 1 0 0 0 1 0 22h

Indicadores de condición  
 después de la ejecución

S Z H P/V N C  
 0 0 x 0 x 0 1 0

En este caso, el resultado  
 ha sido positivo y no ha habi-  
 do acarreo ni desbordamien-  
 to. Vemos que el indicador  
 "N" (SUMA/RESTA) se ha  
 puesto a "1"; al igual que to-  
 das las instrucciones de su-  
 ma ponían este indicador a  
 "cero", todas las de resta lo  
 ponen a "uno".

Vamos a ver detenidamente  
 lo que ha ocurrido: Le hemos  
 pedido al microprocesador  
 que reste 35h menos 13h.  
 Como el segundo número es  
 menor que el primero, el re-  
 sultado será positivo; concre-  
 tamente, el resultado deberá  
 ser 22h. El microprocesador  
 coje primero el número 13h y  
 le hace el complemento a 2 (lo  
 cambia de signo) resultando  
 EDh. A continuación, suma  
 35h + EDh, de lo que resulta  
 22h y un acarreo de "1". Co-  
 mo estamos restando, inverti-  
 mos el acarreo, con lo que re-  
 sulta el número 22h sin aca-  
 rreo.

Vamos a ver qué hubiera  
 ocurrido de hacerlo al revés:  
 restemos 13h menos 35h. El  
 resultado deberá ser -(22h),  
 es decir, el complemento a 2  
 de 22h o, lo que es lo mismo,  
 DEh. Primero cojemos el nú-  
 mero 35h y lo complementa-  
 mos a 2 (lo cambiamos de  
 signo) obteniendo CBh. Aho-  
 ra, sumamos 13h más CBh y  
 obtenemos DEh con un aca-  
 rreo de "cero". Como comple-  
 mentamos el acarreo, resulta  
 ser "uno", con lo que sabe-  
 mos que se trata de un resul-  
 tado negativo. Efectivamente,  
 si complementamos a 2 el nú-  
 mero DEh, obtenemos 22h  
 que es lo que teníamos que  
 obtener.

SUB n

OBJETO:

Resta del registro acumula-  
 dor "A" el entero de 8 bits "n",  
 dejando el resultado en el re-  
 gistro acumulador.

CODIGO DE MAQUINA:

1 1 0 1 0 1 1 0 D6h

INDICADORES DE  
 CONDICION A LOS  
 QUE AFECTA:

S ; pone 1 - si el resultado  
 es negativo  
 pone 0 - en cualquier  
 otro caso

Z ; pone 1 - si el resultado  
 es cero  
 pone 0 - en cualquier  
 otro caso

H ; pone 1 - si no hay aca-  
 rreo desde el bit 3  
 pone 0 - en cualquier  
 otro caso

N ; pone 0 - siempre  
 C ; pone 1 - si no hay aca-  
 rreo desde el bit 7  
 pone 0 - en cualquier  
 otro caso

P/V; pone 1 - si hay desbor-  
 damiento (overflow)  
 pone 0 - en cualquier  
 otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

SUB 12

En este caso, el sustraendo  
 es mayor que el minuendo,  
 por lo que el resultado deberá



```

10 CLEAR 30000
20 FOR C=30000 TO 31010
30   BEEP: A=POKE C,3: NEXT C
40 DATA 55,177,92,1,55,175,92
128 245,193,301
150 INPUT "ENTER OPERANDO ? ",A
30 IF A=255 OR A=0 THEN GO TO
50
70 POKE 23226,A
80 INPUT "SEGUNDO OPERANDO ? ",B
30 IF A=255 OR A=0 THEN GO TO
50
100 POKE 23229,B
110 LET A=A*B/31000
120 GO SUB 31000

```

Sería interesante que el lector intentara, ahora, ensamblar por su cuenta este programa, para ello, deberá proceder como hicimos nosotros en el caso anterior. Primero, copie el programa en un papel, ahora, vaya buscando cada instrucción en las tablas ("ANDA" se ensambla como A7h ó 167d). A continuación, escriba los operandos numéricos sin olvidar invertir el orden de los octetos y, finalmente, acuérdesse de ensamblar "RET" como C9h ó 201d.

**96 CODIGO MAQUINA**

Asymble	Unradical	Radical
AND A	A7	167
LD A, (R576)	3A,7A,5C	59,118,92
LD D,A	57	87
LD A, (R5C88)	2A,9A,5C	59,716,92
ADC A,A	5A	138
LD C,A	4F	79
LD D,A	3A,77A,5C	59,119,92
LD D,A	57	87
LD A, (R5C1)	2A,81,3C	59,777,92
ADC A,D	8A	138
LD B,A	47	71
PUSH AF	F5	245
POP DE	D1	289
LD (R5C88),DE	E9,57,8A,5C	27,283,176,92
RET	C9	281

No se preocupe si se ha equivocado en algo, sería mucho pedir que el primer programa que ensambra le saliera sin errores. Ahora, con los datos de la tercera columna (donde dice: "Decimal") podemos construir el programa en Basic que introduzca esta rutina en memoria, y la utilice para sumar dos números. Este programa es el listado que aparece con el nombre de PROGRAMA 2. No hace falta que lo copie entero, si lo desea, puede cargar el PROGRAMA 1 y reescribir las líneas 20, 40, 60, 70, 90, 100.

```

130 FOR N=LEN A$ TO 15
140 LET A$=A$+A$ NEXT N
1500 PRINT
1600 PRINT TAB 3;"ACUMULADOR";
SZKXUC"
200 PRINT "A$=";A$(1 TO 8);
210 GOTO 100
2200 PRINT DEC: (A) A BIN. (A$)
2300 LET C=A
2400 LET A$=INT C/256 LET C=INT C/256
2500 C=C/256
2600 A$=A$+A$
2700 GOTO 100 IF C=0
2800 RETURN
2900

```

pero observe que en las líneas 70 y 100 partimos los operandos en dos octetos operando de meterlos en sus direcciones de memoria correspondiente.

Puede, también, restar números si cambia "ADC A,D" por "SBC A,D"; para ello, puede cambiar los dos "138" de la línea 40 (del Basic) por "1154", o bien, detener el programa (con STOP) y teclear: POKE 31008,154: POKE 31017,154. Luego, no lo arranque con "RUN", sino, con "GO TO 50".

Como verá, el hecho de POKEar en la zona de programa hace que éste se comporte de forma distinta; he aquí la razón de los famosos "POKES" de "vidas infinitas" y similares que se utilizan en juegos comerciales (sección "Patatas Arriba" de la revista MICRO-MANIA).

5724-3615

Al resta 4-5 en realidad se hace 14-5 y nos llevamos una (acarreo), este acarreo lo sumamos a la decena 1 del sustraendo, que es lo mismo que restarlo en el minuendo a la decena 2 y resulta: 2-2 ó 1-1.

5724  
-3615  
-----  
2109

Pues esto mismo ocurre al sumar octetos, cuando el octeto sustraendo es mayor que el octeto minuendo y activarse por tanto el bit de acarreo, al tenerlo en cuenta con los octetos de orden superior; es como si en el minuendo se sumara 256 al octeto inferior y se restara uno al octeto superior.

Ver figura 6-2. Este es el uso más importante de la condición de acarreo para la resta.

**SBC A-1**

**OBJETO:**

Resta del registro acumulador "A" el contenido del registro especificado por "r", más el indicador de acarreo. Deja el resultado en el registro acumulador.

CODIGO DE MAQUINA:

otro caso  
P/V; pone 1 - si hay desbor-  
damiento (overflow)  
pone 0 - en cualquier  
otro caso

INDICADORES DE  
CONDICION A LOS  
QUE AFECTA:

CICLOS DE MEMORIA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

one case  
one case  
one case

**EJEMPLO:**

SUB A H

Valor del registro "A"

(A)  $\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$

Valor del registro "H"

CODIGO MAQUINA 89



(H): 01110111 //h

Indicador de acarreo (C)=0.

Instrucción

SUB A,H 10011000 9Ch

Operación:

(H) 01110111  
+C 0  
-----  
01110111  
comp1. a 2 10001001  
+(A) 11111111  
-----  
10001000

Valor del registro "A" después de la ejecución

(A): 10001000 80h

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 0 x 0 1 0

SBC A,n

OBJETO:

Resta del registro acumulador "A" el entero de 8 bits n, más el bit de acarreo. Deja el resultado en el registro acumulador.

CODIGO DE MAQUINA:

11011110 0Eh

INDICADORES DE CONDICION A LOS QUE AFECTA:

S : pone 1 - si el resultado

## GRUPO ARITMETICO DE 8 BITS (SUMA Y RESTA)

Código Fuente	Hexadecimal	Decimal
ADD A,A	87	135
ADD A,B	80	128
ADD A,C	81	129
ADD A,D	82	130
ADD A,E	83	131
ADD A,H	84	132
ADD A,L	85	133
ADD A,n	C6,n	198,n
ADD A,(HL)	86	134
ADD A,(IX+d)	DD,86,d	221,134,d
ADD A,(IY+d)	FD,86,d	253,134,d
ADC A,A	8F	143
ADC A,B	88	136
ADC A,C	89	137
ADC A,D	8A	138
ADC A,E	8B	139
ADC A,H	8C	140
ADC A,L	8D	141
ADC A,n	CE,n	206,n
ADC A,(HL)	8E	142
ADC A,(IX+d)	DD,8E,d	221,142,d
ADC A,(IY+d)	FD,8E,d	253,142,d
SUB A	97	151
SUB B	90	144
SUB C	91	145
SUB D	92	146
SUB E	93	147
SUB H	94	148
SUB L	95	149
SUB n	D6,n	214,n
SUB (HL)	96	150
SUB (IX+d)	DD,96,d	221,150,d
SUB (IY+d)	FD,96,d	253,150,d
SBC A,A	9F	159
SBC A,B	98	152
SBC A,C	99	153

samblado el programa antes de mirar la tabla anterior. Proyéctase a sí mismo que la próxima vez lo intentará.

Ya tenemos preparada la rutina en código máquina para sumar dos números. Seamos buenos con los que aún tienen solo 16K, y carguemos la rutina a partir de la dirección 31000.

Ha llegado el momento de pasar al olvidado Basic. El PROGRAMA 1 se encarga de todo. La línea 10 baja RAMTOP, las líneas 20 y 30 introducen en memoria nuestra rutina que se encuentra en los DATA de la línea 40. Las líneas 50 a 100 nos piden los dos operandos y los POKEan en memoria tras comprobar si están dentro de rango.

La línea 110 llama a nuestra rutina en C/M de forma que, al retornar, el contenido del registro "BC" se almacene en la variable "a". En 120 llamamos a la rutina 3100 que nos pasa el número a binario, esta subroutine es la misma que usamos en el programa para cambiar de base, del capítulo 3. Las líneas 130 y 140 completan el número con ceros a la izquierda para obtener, de nuevo, 16 bits. Finalmente, las líneas 200 a 220 imprimen en pantalla el valor que contenía el acumulador después de efectuar la suma y el estado de los indicadores en el registro "F". El significado de los indicadores es el siguiente:

S : Signo  
Z : Cero  
H : Semi-acarreo  
V : Desbordamiento  
N : Suma/Resta  
C : Acarreo

Los indicadores marcados "x" presentan un estado indeterminado y no habrá que tomarlos en cuenta.

Una vez que tenga el programa en memoria, pruebe a introducir distintos operandos comprendidos entre 0 y 255. Le sugerimos unos cuantos:

17 + 17 = 34  
15 + 240 = 255 (N)  
128 + 128 = 0 (Z,V,C)  
127 + 1 = 128 (S,H,V)  
3 + 127 = 130 (S,H,V)

Puede utilizar este mismo programa para la resta cambiando "ADD A,B" por "SUB B", es decir, el "128" de la línea 40 por un "144". Haga el cambio y ejecute el programa de nuevo, esta vez restará el segundo operando del primero. Si el segundo es mayor que el primero (resultado negativo), el indicador "C" se pondrá a "1" y el resultado aparecerá en complemento a 2.

Ahora vamos a complicar un poco más la cosa, se trata de hacer una rutina que permita sumar números superiores a 255. En este caso, usaremos la instrucción "ADC" (sumar con acarreo) para poder tener en cuenta, cuando sumemos un octeto, el acarreo procedente del anterior.

Introduciremos el primer operando en las direcciones 5CB0H (23728) y 5CB1h (23729) (primero el octeto menos significativo y luego el más significativo), y el segundo operando en 5C76h (23670) y 5C77h (23671).

El programa en Assembler puede ser algo como:

```
10 AND A
20 LD A, (#5C76)
30 LD D,A
40 LD A, (#5CB0)
50 ADC A,D
60 LD C,A
70 LD A, (#5C77)
80 LD D,A
90 LD A, (#5CB1)
100 ADC A,D
110 LD B,A
120 PUSH AF
130 POP DE
140 LD (#5CB0),DE
150 RET
```

La línea 10 pone a "cero" el indicador de acarreo; se trata de un pequeño "truco" que consiste en realizar un "AND" lógico del acumulador consigo mismo, con lo que su contenido no varía, pero no se pone a cero el indicador de acarreo. Más adelante, y dentro de este mismo capítulo, veremos las operaciones lógicas.

Las líneas 20, 30 y 40 cargan los octetos de orden bajo de los dos operandos. La línea 50 los opera (suma) y, si hay acarreo, lo guarda para la suma siguiente. La línea 60 guarda el resultado en "C" (octeto bajo de "BC").

La operación se vuelve a repetir para los octetos altos; las líneas 70, 80 y 90 cargan los operandos. La línea 100 los suma tomando en cuenta el acarreo procedente de la operación anterior. Finalmente, la línea 110 transfiere el resultado al registro "B" (octeto alto de "BC").



más adelante) y le indica si la última operación realizada ha sido una suma o una resta. Por tanto, todas las sumas lo ponen a "0" y todas las restas lo ponen a "1".

"C": Este indicador se activa (valor igual 1) cuando no hay acarreo desde el bit 7, después de la ejecución. Esto ocurre siempre que el valor absoluto del octeto del sustraendo es mayor que el valor absoluto del octeto del minuendo. Es el indicador que se emplea para las instrucciones de restar con acarreo. Interesa observar que este indicador funciona, en la resta, de forma contraria a como lo hace en la suma, es decir, se activa cuando NO hay acarreo en la suma que realiza el microprocesador tras complementar el sustraendo.

"P/V": Este indicador de desbordamiento (overflow) se activa (valor igual 1) siempre que el resultado de la resta haga que el acumulador pase de contener un número menor de 127 a contener uno mayor, o de contener uno mayor de -128 a contener uno menor. Indica, por tanto, un rebasamiento del margen comprendido entre -128 y +127. Se utiliza como indicador de rebasamiento cuando se trabaja con números en complemento a 2.

Hasta aquí hemos visto las instrucciones que nos han de servir para sumar y restar en código máquina. A continuación veremos las que se encargan de realizar operaciones lógicas tales como AND, OR y XOR. Pero antes, realizaremos unos cuantos ejemplos que podamos ejecutar

en el ordenador, y que sirvan para aclarar lo estudiado. También invitamos al lector a que intente resolver los ejercicios que se proponen, y que le darán una medida de cómo va asimilando los conocimientos.

## EJEMPLOS:

Al igual que en el capítulo anterior, vamos a hacer algunos programas en código máquina que nos demuestren el funcionamiento de las instrucciones de suma y resta. Al mismo tiempo, iremos corrigiendo práctica en la realización y ensablado de programas en Assembler.

Recomendamos al lector que no se limite a «leer por encima» este curso. Si desea, de verdad, aprender a programar en código máquina, debe seguir el curso encima de una mesa con lápiz y papel en la mano. Intente ensambalar cada programa por usted mismo, y no se limite a ver cómo lo hacemos nosotros; e incluya, a través de escribir sus propias rutinas. No se preocupe si el ordenador se le «cuelga» cincuenta veces, es totalmente normal, una rutina en código máquina rara vez funciona a la primera.

Vamos con el primero de nuestros programas. Se trata de sumar dos números sin acarreo. Utilizaremos un programa en Basic que se encargará de gestionar la entrada de datos, llamar a la rutina en C/M e imprimir los resultados, pero la suma la realizaremos en código máquina.

En principio, necesitamos POKEar los dos números que vamos a sumar en dos direcciones de memoria, desde donde serán leídos por la rutina

na C/M. Estas dos direcciones serán la 5CB0h (23728) para el primer operando, y la 5CB1h (23729) para el segundo; estas direcciones corresponden a una variable del sistema que no se usa.

Primero escribiremos el programa en C/M y luego el Basic. En Assembler, nuestra rutina podría ser algo así:

```
10 LD A, (5CB1)
20 LD B, A
30 LD A, (5CB0)
40 ADD A, B
50 PUSH AF
60 POP BC
70 RET
```

Las líneas 10, 20 y 30 leen los dos operandos desde las posiciones de memoria donde los almacenó el Basic. La línea 40 realizará la suma equivalente a:

$$[E] A = A + B$$

Las líneas 50 y 60 transfieren el resultado al registro "B" y los indicadores de estado del registro "F", al registro "C". Recuerde que el registro "BC" es lo que nos devuelve USR cuando retornamos a Basic. Mirando las tablas de codificación, podemos ensambalar el programa:

Assembler	Hexadecimal	Decimal
LD A, (5CB1)	: 3A, 81, 5C	: 58, 177, 92
LD B, A	: 47	: 71
LD A, (5CB0)	: 3A, 80, 5C	: 58, 176, 92
ADD A, B	: 8F	: 128
PUSH AF	: F5	: 245
POP BC	: C1	: 193
RET	: C9	: 201

Habría sido interesante que el propio lector hubiera en-

es negativo  
pone 0 - en cualquier otro caso  
Z : pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H : pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N : pone 0 - siempre  
C : pone 1 - si no hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso  
P/V: pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso  
CICLOS DE MEMORIA:  
2  
CICLOS DE RELOJ:  
7  
EJEMPLO:  
SBC A, 40

Valor del registro "A"  
00000000  
Indicador del acarreo (C) = 1  
Instrucción:  
SBC A, 40  
Operación:  
11011110 00h  
00101000 28h

Rest del registro acumulador "A", el valor del octeto de memoria direccionado por el contenido el par de registros HL, más el indicador de acarreo. El resultado se deja en el registro acumulador.

OBJETO:  
Rest del registro acumulador "A", el valor del octeto de memoria direccionado por el contenido el par de registros HL, más el indicador de acarreo. El resultado se deja en el registro acumulador.

Rest del registro acumulador "A", el valor del octeto de memoria direccionado por el contenido el par de registros HL, más el indicador de acarreo. El resultado se deja en el registro acumulador.

SBC A, D	9A	154
SBC A, E	9B	155
SBC A, H	9C	156
SBC A, L	9D	157
SBC A, n	DE, n	222, n
SBC A, (HL)	9E	158
SBC A, (IX+d)	DD, 9E, d	221, 158, d
SBC A, (IX+d)	FD, 9E, d	253, 158, d

Tabla de codificación para suma y resta.

Valor del registro "A" después de la ejecución  
11010111 D7h  
Indicadores de condición después de la ejecución.  
S Z H P/V N C  
S : pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z : pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H : pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N : pone 0 - siempre  
C : pone 1 - si no hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso  
P/V : pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

Objeto:  
Rest del registro acumulador "A", el valor del octeto de memoria direccionado por el contenido el par de registros HL, más el indicador de acarreo. El resultado se deja en el registro acumulador.

Objeto:  
Rest del registro acumulador "A", el valor del octeto de memoria direccionado por el contenido el par de registros HL, más el indicador de acarreo. El resultado se deja en el registro acumulador.

Objeto:  
Rest del registro acumulador "A", el valor del octeto de memoria direccionado por el contenido el par de registros HL, más el indicador de acarreo. El resultado se deja en el registro acumulador.

Objeto:  
Rest del registro acumulador "A", el valor del octeto de memoria direccionado por el contenido el par de registros HL, más el indicador de acarreo. El resultado se deja en el registro acumulador.



(H): 1 1 0 0 0 1 0 1  
(L): 0 0 0 1 1 0 0 0

Valor de la posición de memoria C518h

(C518h): 1 1 1 1 1 1 1 1

Valor del registro "A"

(A): 1 1 1 1 1 1 1 1

Indicador de acarreo = 0

Instrucción

SBC A,(HL): 10011110

Operación:

(C518h) 11111111  
+C  
-----  
11111111  
compl. a 2 00000001  
+(A) 11111111  
-----  
00000000

Valor del registro "A" después de la ejecución

(A): 0 0 0 0 0 0 0 0

Indicadores de condición después de la ejecución

S Z H P/V N C

0 1 x 0 x 1 1 0

Observe que hubo desbordamiento al pasar el registro "A" de un valor negativo a uno positivo.

SBC A,(IX+d)

OBJETO:

Resta al contenido del registro acumulador; el valor del octeto de memoria direccionado por el operando, más el indicador del acarreo. La dirección de memoria se calcula añadiendo al contenido del registro índice IX el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado se deja en el registro acumulador.

CODIGO MAQUINA:

00h 1 1 0 1 1 1 0 1  
9h 1 0 0 1 1 1 1 0  
<---d-->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H ; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N ; pone 0 - siempre  
C ; pone 1 - si no hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso

P/V ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

Indicadores de condición después de la ejecución

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

SBC A,(IX+30)

Valor del registro "IX"

(IX): 1 1 1 1 1 0 0 1  
F3h 0 0 0 1 1 0 0 1  
19h

Valor de la posición de memoria F337h

(F337h): 0 1 1 0 1 0 1 0

Valor del registro "A"

(A): 0 1 1 1 1 0 1 1

Indicador de acarreo (C)=1

Instrucción

00h 1 1 0 1 1 0 1  
9h 1 0 0 1 1 1 0  
10h 0 0 0 1 1 1 0

Operación:

(F337h) 0 1 1 0 1 0 1 0  
+C  
-----  
0 1 1 0 1 0 1 1  
compl. a 2 1 0 0 1 0 1 0 1  
+(A) 0 1 1 1 1 0 1 1  
-----  
0 0 0 1 0 0 0 0

Valor del registro "A" después de la ejecución

(A): 0 0 0 1 0 0 0 0

Indicadores de condición después de la ejecución

S Z H P/V N C

0 0 x 0 x 0 1 0

SBC A,(IX+d)

OBJETO:

Resta al contenido del registro acumulador "A", el valor del octeto de la memoria direccionado por el operando, más el indicador de acarreo. La dirección de memoria se calcula añadiendo al contenido del registro índice "IX" el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado se deja en el registro acumulador.

CODIGO MAQUINA:

FDh 1 1 1 1 1 1 0 1  
9h 1 0 0 1 1 1 1 0  
<---d-->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S ; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z ; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H ; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N ; pone 0 - siempre  
C ; pone 1 - si no hay acarreo desde el bit 7  
pone 0 - en cualquier otro caso

P/V ; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

Indicadores de condición después de la ejecución

(A): 1 0 0 0 1 1 1 0

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

SBC A,(Y-1)

Valor del registro "Y"

(Y): 0 1 1 0 0 1 1 0  
66h 1 1 1 0 0 0 0 1  
E1h

Valor de la posición de memoria 66E0h

(66E0h): 1 1 0 1 0 0 1 1

Valor del registro "A"

(A): 0 1 1 0 0 0 1 0

Indicador de acarreo (C)=1

Instrucción:

FDh 1 1 1 1 1 1 0 1  
9h 1 0 0 1 1 1 1 0  
FFh 1 1 1 1 1 1 1 1

Operación:

(66E0h) 1 1 0 1 0 0 1 1  
+C  
-----  
1  
compl. a 2 1 1 0 1 0 1 0 0  
+(A) 0 1 1 0 0 0 1 0  
-----  
1 0 0 0 1 1 1 0

Valor del registro "A" después de la ejecución

(A): 1 0 0 0 1 1 1 0

Indicadores de condición después de la ejecución

S Z H P/V N C

1 0 x 1 x 1 1 1

Observe que no hubo acarreo desde el bit 7, esto ocurre cuando el sustraendo es menor que el minuendo, como el indicador "C" está invertido en las instrucciones de resta, se pone a "1", esta situación da como resultado un número negativo. Si se hace el complemento a dos del resultado en el registro "A", nos da el valor 72, que con el signo "-" es el resultado entero de la operación (-72h).

La activación de los indicadores de condición, tanto en la resta (SUB) como en la resta con acarreo (SBC), se hace según las siguientes reglas:

"S": En este indicador se copia el bit 7 del acumulador para indicar si el número que contiene es positivo o negativo.

"Z": Este indicador se activa (valor igual 1) siempre que todos los bits del registro acumulador sean cero después de la ejecución.

"H": Este indicador se activa (valor igual 1) cuando no hay acarreo desde el bit 3 después de la ejecución. Esto ocurre siempre que el valor absoluto de los cuatro bits inferiores del sustraendo es mayor que el valor absoluto de los cuatro inferiores del minuendo.

"N": Este indicador lo toma en cuenta el microprocesador cuando hace un ajuste BCD del acumulador (se verá



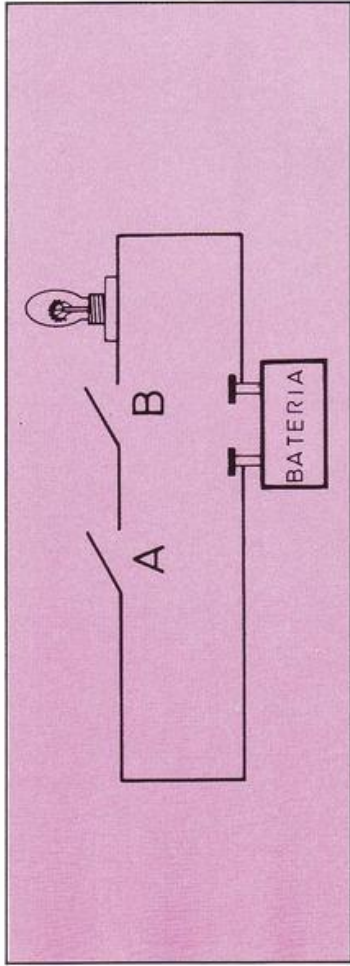


Fig. 6.5. Representación eléctrica de la operación AND.

bombilla se enciende es necesario que los dos estén conectados. Ver, en la Figura 6-5, que sólo cuando los interruptores A y B estén conectados se cerrará el circuito.

Resumiendo, el resultado es un solo cuando el bit de un operando y el del otro son 1. Básicamente el formato de esta instrucción es:

AND OPERANDO

El octeto indicado por el operando se enfrenta con el octeto del registro acumulador, el resultado se deja en este último y el operando no sufre variación; asimismo, se actualiza los indicadores en consonancia con el resultado. En las operaciones lógicas, el indicador "P/V" no indica rebotamiento, sino "paridad"; más adelante, veremos a fondo lo que se entiende por paridad.

AND r

OBJETO:

Realiza una operación lógica AND, bit a bit, entre el octeto del registro acumulador y el octeto del registro indicado

por "r". El resultado se deja en el registro acumulador.

CODIGO MAQUINA:

00100 <---r---

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad del resultado es par  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

AND C

En este ejemplo, realizaremos un AND lógico bit a bit entre el contenido del acumulador y el del registro "C". Cada bit del resultado será "1" si, y solo si, los dos bits correspondientes de cada operando son "1".

Valor del registro "A"

(A): 10110101 85h

Valor del registro "C"

(C): 10010110 98h

Instrucción

AND C 10100001 A1h

Valor del registro "A" después de la ejecución

(A): 10010100 94h

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 1 x 0 0 0

Observe que el indicador de condición P/V es 0 porque el número de "unos" en el resultado es 3 (impar).

El formato de esta instrucción, admite la operación

## Grupo de incremento y decremento para 8 bits

INC (INCrement), "incrementar" en inglés. Básicamente esta instrucción incrementa en uno el valor del octeto indicado en el operando.

Este tipo de instrucciones es muy útil en programación, es de uso común ir siguiendo secuencialmente una serie de octetos, lo cual resulta muy fácil, utilizando esta instrucción en combinación con otras que usen el mismo operando como índice. Lo mismo para utilizar matrices e ir variando el valor de la fila y columna.

Otro de los usos más comunes es para actualizar contadores. Los contadores son octetos que se utilizan para conocer el número de veces que ha ocurrido un determinado suceso.

Es fácil que el lector se pregunte el porqué de no utilizar las instrucciones de sumar (ADD); la ventaja principal es que las instrucciones de incremento no requieren el uso del registro acumulador, lo cual evita también el tiempo y el espacio de estar cargándolo y salvándolo para actualizar y conocer su contenido.

Para incrementar un contador situado en una posición de memoria con la instrucción ADD, es necesario hacer lo siguiente:

- Cargar el registro acumulador con el valor del octeto.
- Sumar uno al registro acumulador.
- Cargar la posición de memoria con el contenido del registro acumulador.

Mientras que con la instrucción INC se hace en un solo paso, el operando es in-

Código Fuente	Hexadecimal	Decimal
INC A	3C	60
INC B	04	42
INC C	0C	12
INC D	14	20
INC E	1C	28
INC H	24	36
INC L	2C	44
INC (HL)	34	52
INC (IX+d)	DB, 34, d	221, 52, d
INC (IY+d)	FD, 34, d	253, 52, d
DEC A	3D	61
DEC B	05	5
DEC C	0D	13
DEC D	15	21
DEC E	1D	29
DEC H	25	37
DEC L	2D	45
DEC (HL)	35	53
DEC (IX+d)	DB, 35, d	221, 53, d
DEC (IY+d)	FD, 35, d	253, 53, d

Fig. 6.3b. Grupo de incremento y decremento para 8 bits.

crementado y se actualizan los indicadores del registro "F" para poner de manifiesto la ocurrencia de determinadas condiciones (cero, signo, etc.) excepto el acarreo que no es afectado por estas instrucciones.

Veamos, ahora, los distintos formatos en que se nos puede presentar la instrucción INC según sus operandos.

INC r

OBJETO:

Incrementa en uno el valor del registro indicado por "r".

CODIGO DE MAQUINA:

00 <---r---> 100



## INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 – si el resultado es negativo

pone 0 – en cualquier otro caso

Z; pone 1 – si el resultado es cero

pone 0 – en cualquier otro caso

H; pone 1 – si hay acarreo desde el bit 3

pone 0 – en cualquier otro caso

N; pone 0 – siempre

P/V; pone 1 – si el valor de r era 7Fh antes de la operación

pone 0 – en cualquier otro caso

## CICLOS DE MEMORIA:

1

## CICLOS DE RELOJ:

4

## EJEMPLO:

INC B
-------

En este ejemplo, vamos a incrementar el contenido del registro "B", es decir, vamos a sumarle "1". Podíamos haber hecho:

LD	A,B
ADD	A,1
LD	B,A

El resultado hubiera sido el mismo, pero habríamos tenido que utilizar el acumulador, y nos ocuparía más memoria y más ciclos de reloj. En cambio, "INC B" lo hace directamente.

Valor del registro "B"

## 98 CODIGO MAQUINA

(B) 00000101 05h

## Instrucción

INC B 00000100 04h

Valor del registro "B" después de la ejecución

(B) 00000111 06h

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	0	0	x

En este caso, la instrucción no ha afectado a ningún indicador, dado que no se ha producido ninguna condición que así lo requiera. Obsérvese que el indicador "N" (suma/resta) permanece a "0", ya que lo que se ha producido es una suma (hemos sumado uno).

## INC (HL)

## OBJETO:

Incrementa en uno el valor del octeto direccionado por el par de registros "HL".

## CODIGO DE MAQUINA:

00110100 34h

## INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 – si el resultado es negativo

pone 0 – en cualquier otro caso

Z; pone 1 – si el resultado es cero

pone 0 – en cualquier otro caso

H; pone 1 – si hay acarreo desde el bit 3

pone 0 – en cualquier otro caso

N; pone 0 – siempre

P/V; pone 1 – si el valor del octeto era 7Fh antes de la operación

pone 0 – en cualquier otro caso

## CICLOS DE MEMORIA:

3

## CICLOS DE RELOJ:

11

## EJEMPLO:

INC (HL)
----------

Esta vez, vamos a incrementar el contenido de la posición de memoria cuya dirección es apuntada por el contenido de "HL".

Valor del par de registros "HL"

(H):	11001010	CAh
(L):	01101111	6Fh

Valor de la posición de memoria CA6Fh

(CA6Fh): 00111111 3Fh

## Instrucción

INC (HL): 00110100 34h

Valor de la posición de memoria CA6Fh después de la ejecución

(CA6Fh): 01000000 40h

Indicadores de condición después de la ejecución

## BIT

## OCTETO OPERANDO

## OPERACION

## OCTETO OPERANDO

## RESULTADO

7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	1
+	+	+	+	+	+	+	+
0	1	1	0	1	1	0	0
X	X	X	X	X	X	X	X

Fig. 6.4. Operación lógica.

Valor de la posición de memoria CB16h después de la ejecución

(CB16h): 00000000 00h

Indicadores de condición después de la ejecución.

S	Z	H	P/V	N	C
0	1	x	0	0	1

ga el valor 80h antes de la ejecución. Esto es, si hay desbordamiento de la cantidad negativa más pequeña que se puede expresar en un octeto al restar 1 a -128, con lo que se pasa a un número positivo +127. Ver ejemplo de DEC (HL).

"N": Este indicador carece de significado para estas instrucciones y, al igual que en la resta, se pone siempre a 1.

"C": Este indicador no resulta afectado por las instrucciones "DEC" conservando, por tanto, el estado que tuviera antes de la ejecución. Esta circunstancia es sumamente útil, ya que permite iterar un bucle sin perder el estado anterior del acarreo; cosa que, con las instrucciones de resta, no sería posible.

## Grupo de instrucciones lógicas

Las operaciones lógicas enfrentan bit a bit los octetos, de tal forma que el bit 0 del octeto resultado se define por el valor de los bit 0 de los operandos, el bit 1 con los bit 1 y así sucesivamente. Nunca depende el valor de un bit de los valores de sus bit superiores o inferiores. Si indicamos

1 AND 1 = 1  
1 AND 0 = 0  
0 AND 1 = 0  
0 AND 0 = 0

Es igual que dos interruptores conectados en serie (uno a continuación del otro) en un circuito eléctrico; para que la

la operación lógica con una doble flecha al operar dos octetos actuarían según la Figura 6-4.

En las operaciones lógicas, el valor uno se identifica con puesto (set) o encendido, y el valor 0 con quitado (clear) o apagado.

Existen tres operadores lógicos posibles: AND (conjunción), OR (disjunción) y XOR (disjunción excluyente). Vamos a verlas detenidamente una por una.

AND, conjunción copulativa inglesa; se traduce en castellano por "Y". Con esta palabra se define una operación lógica que consiste en que cuando los dos bits enfrentados son 1 el bit resultado es 1, en los demás casos el resultado es cero.

## CODIGO MAQUINA 103



1 1 0 1 1 1 0 1
0 0 1 1 0 1 0 1
<---d-->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N; pone 1 - siempre  
P/V; pone 1 - si el valor del octeto era 80h antes de la operación  
pone 0 - en cualquier otro caso  
H; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N; pone 1 - siempre  
P/V; pone 1 - si el valor del octeto era 80h antes de la operación  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

DEC (IX+127)
--------------

Valor del registro indice "IX"

0 1 1 1 0 1 0 0
0 1 0 0 1 0 0 1

Valor de la posición de memoria 74C8h.

74C8h: 0 0 0 0 0 0 0 0
------------------------

Instrucción

11011101
00110101
01111111

DEC (IX+127)

102 CODIGO MAQUINA

Operación:

00000000
+11111111
-----
11111111

Valor de la posición de memoria 74C8h después de la ejecución

74C8h: 1 1 1 1 1 1 1 1
------------------------

Indicadores de condición después de la ejecución

S Z H P/V N C
1 0 x 1 x 0 1 x

Observe, como al decrementar uno al valor 00h da como resultado FFh que es la representación de -1 en complemento a 2.

DEC (Y+d)

OBJETO:

Resta uno al valor del octeto direccionado por: añadir al contenido del registro indice "Y" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127.

CODIGO DE MAQUINA:

1 1 1 1 1 1 0 1
0 0 1 1 0 1 0 1
<---d-->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado

es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N; pone 1 - siempre  
P/V; pone 1 - si el valor del octeto era 80h antes de la operación  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

DEC (Y-128)
-------------

Valor del registro indice "Y"

1 1 0 1 1 1 0 1
1 0 0 1 0 1 1 0

Valor de la posición de memoria CB16h

CB16h: 0 0 0 0 0 0 0 1
------------------------

Instrucción

11111101
00110101
10000000

Operación:

00000001
+11111111
-----
00000000

S Z H P/V N C
0 0 x 1 x 0 0 x

Observe, que ha habido acarreo desde el bit 3.

INC (IX+d)

OBJETO:

Incrementa en uno el valor del octeto direccionado por: añadir al contenido del registro indice "IX" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127.

CODIGO MAQUINA:

1 1 0 1 1 1 0 1
0 0 1 1 0 1 0 0
<---d-->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N; pone 0 - siempre  
P/V; pone 1 - si el valor del octeto era 7Fh antes de la operación  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

INC (IX+7)
------------

Valor del registro indice "IX"

IX: 1 0 0 0 1 1 0 0
0 0 1 0 1 0 0 0

Valor de la posición de memoria 8C2Fh.

8C2Fh: 0 1 1 1 1 1 1 1
------------------------

Instrucción

INC (IX+7)
1 1 0 1 1 1 0 1
0 0 1 1 0 1 0 0
0 0 0 0 0 1 1 1

Valor de la posición de memoria 8C2Fh después de la ejecución.

8C2Fh: 1 0 0 0 0 0 0 0
------------------------

Indicadores de condición después de la ejecución

S Z H P/V N C
1 0 x 1 x 1 0 x

Observe, que el valor anterior del octeto era 7F, por lo tanto se activa el indicador "P/V".

INC (Y+d)

OBJETO:

Incrementa en uno el valor del octeto direccionado por: añadir al contenido del regis-

tro indice "Y" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127.

CODIGO MAQUINA:

1 1 1 1 1 1 0 1
0 0 1 1 0 1 0 0
<---d-->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - si hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso  
N; pone 0 - siempre  
P/V; pone 1 - si el valor del octeto era 7Fh antes de la operación  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

INC (Y-7)
-----------

Valor del registro indice "Y"

IV: 1 0 0 0 1 0 1 1
1 1 1 1 1 0 0 0

Valor de la posición de memoria 8BF1h

8BF1h: 1 1 1 1 1 1 1 1
------------------------

CODIGO MAQUINA 99



## Instrucción

INC (IX-7)	F0h
	34h
	F9h
	FDh

Valor de la posición de memoria 8BF1h después de la ejecución

(8BF1h):	00000000
----------	----------

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	1	x	1	x	0

Observe, que la única ocasión en que el octeto puede tener como resultado "0" es si anteriormente valía FFh.

La activación de los indicadores de condición, en las instrucciones INC, se hace según las siguientes reglas: "S": En este indicador se pone el mismo valor del bit 7 del octeto después de la ejecución.

"Z": Este indicador se activa, valor igual 1, si todos los bits del octeto son cero después de la ejecución.

"H": Este indicador se activa, valor igual 1, si hay acarreo en el octeto desde el bit 3, después de la ejecución; o lo que es lo mismo, los cuatro bits inferiores del octeto son 1 antes de la ejecución, independientemente del valor de los cuatro bits superiores.

"P/V": Este indicador se activa siempre que el octeto tenga el valor 7Fh antes de la ejecución. Esto es, hay desbordamiento de la máxima cantidad positiva que se puede expresar en un octeto en complemento a 2 (+127).

"N": Este indicador carece de significado para estas ins-

trucciones y se pone siempre a 0.

"C": Este indicador no resulta afectado por estas instrucciones y mantiene, por tanto su anterior contenido.

DEC (Decrement), "decrementar" en inglés. Básicamente esta operación resta "1" del octeto especificado mediante el operando.

Al igual que las instrucciones INC, estas instrucciones sirven para seguir una secuencia de octetos, con la diferencia de que se hace desde la dirección más alta a la inferior.

Otro de los usos más importantes es calcular el final de un proceso que se desea realizar "n" veces (bucle de iteración), para lo cual se carga "n" en el campo que actúa como contador, que puede ser un registro o una posición de memoria, en cada pasada del bucle, se resta "1" del contador, y cuando este llega a cero, se sale del bucle que se habrá iterado "n" veces; esto sería el equivalente a los bucles "FOR... NEXT" del Basic. Si empezáramos desde cero y fuéramos incrementando hasta alcanzar el valor "n", sería necesario comparar cada vez el valor del contador con "n" para ver si ya lo ha alcanzado; esta comparación es más compleja que comprobar si el contador es cero, ya que en ese caso, después de decrementarlo se habrá puesto a "1" el indicador "Z", por lo que nos bastará con comprobar este indicador. Cuando veamos las instrucciones de salto, estudiaremos en profundidad la forma de crear bucles en código máquina.

El uso de estas instrucciones en lugar de las de restar

4

## EJEMPLO:

DEC H
-------

En este ejemplo, restamos "1" al contenido del registro "H"; la operación sería equivalente a:

LD	AH
SUB	1
LD	HA

Excepto que "DEC H" no afecta al indicador de acarreo.

Valor del registro "H"

(H)	01101001
-----	----------

## Instrucción

DEC H:	00100101
--------	----------

## Operación:

01101001
+11111111
-----
01101000

Valor del registro "H" después de la ejecución

(H)	01101000
-----	----------

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	0	0	1

## DEC (HL)

## OBJETO:

Decrementa en uno el valor del octeto direccionado por el par de registros "HL".

## CODIGO DE MAQUINA:

00110101
----------

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero

pone 0 - en cualquier otro caso

H; pone 1 - si no hay acarreo desde el bit 3

pone 0 - en cualquier otro caso

N; pone 1 - siempre

P/V; pone 1 - si el valor del octeto era 80h antes de la operación

pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

3

## CICLOS DE RELOJ:

11

## EJEMPLO:

DEC (HL)
----------

Valor del par de registros "HL"

(H):	10000000
(L):	10111100

Valor de la posición de memoria 80BCh

(80BCh):	10000000
----------	----------

## Instrucción

DEC (HL):	00110101
-----------	----------

## Operación:

10000000
+11111111
-----
01111111

Valor de la posición de memoria +0BCh después de la ejecución

(0BCh):	01111111
---------	----------

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	1	x	1

Observe, que el indicador P/V se ha activado por pasar el valor del octeto de negativo a positivo al pretender restar un 1 a -128 que es el negativo más bajo que se puede expresar con un octeto en complemento a 2.

## DEC (IX+d)

## OBJETO:

Resta uno al valor del octeto direccionado por: añadir al contenido del registro índice "IX" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127.

## CODIGO DE MAQUINA:



# Ejemplo:

drive. El último octeto del bloque de escritura es un octeto de control; si al volver a leer el bloque escrito y analizar ese octeto de control no coincide con el calculado en bloque de entrada, da error de carga.

Este programa monitor, no emplea la paridad de carácter como control por el hecho de que los 8 bits de información del octeto contienen información válida. Este tipo de control sólo se puede hacer, cuando uno de los bits del octeto no tiene información, tal como ocurre en el caso del bit 7 del código ASCII. Recordemos que el código ASCII sólo utiliza los siete bits menos significativos del octeto.

Por último, volviendo al mensaje del ejemplo inicial, se describe a continuación cómo quedaría el mensaje después de aplicarle las dos técnicas de paridad de carácter y paridad longitudinal.

"H"	01001000
paridad par =>	01001000
"0"	01001111
paridad par =>	11001111
"L"	01001100
paridad par =>	11001100
"A"	01000001
paridad par =>	01000001
Oct. de control:	00001010

Esta rutina se puede hacer algo más sencilla, con instrucciones que se verán posteriormente, que modifican el valor de un bit, pero sólo se trata de entender el problema. Ver, a este respecto, el organigrama de la FIGURA 6-8.

En los dos octetos erróneos que se pusieron de ejemplo en el mensaje "MI NOMBRE ES PEPE", que resultó cambiar a "MI OOMBRE ES PEPA", ocurrió que, después del error, la paridad del octeto era impar, con lo cual no había confusión, pues se sabía, con certeza, que el mensaje era erróneo.

Se puede pensar que el error que produce el cambio de valor de un bit lo produzca en dos bits de forma que no altere la paridad del octeto. Esto es posible, desde luego, pero la probabilidad de que al deteriorarse un octeto mantenga la paridad esperada es mucho menor que el deterioro del octeto sin ninguna técnica para detectar el error. Aun así se puede bajar mucho más la probabilidad de errores sin detectar, usando una técnica denominada "paridad longitudinal".

La paridad longitudinal, consiste en contar el número de bits del mensaje total, tomados por columnas. Esto es: cuantos bits 0 activos hay en el mensaje, cuantos bits 1 activos hay, etc. En algunos mensajes estos números serían muy altos y para simplificar se suele hacer lo siguiente: si el número es par se coloca un 0 en un octeto de control en el bit correspondiente a la columna analizada, y en caso contrario se pone un 1, así en todos los bits de este octeto de control.

# PROGRAMA 2

```
10 CLEAR 30999
20 FOR n=31000 TO 31025
30 POKE n,0
40 POKE n,1
50 POKE n,2
60 POKE n,3
70 POKE n,4
80 POKE n,5
90 POKE n,6
100 POKE n,7
110 POKE n,8
120 POKE n,9
130 POKE n,10
140 POKE n,11
150 POKE n,12
160 POKE n,13
170 POKE n,14
180 POKE n,15
190 POKE n,16
200 POKE n,17
210 POKE n,18
220 POKE n,19
230 POKE n,20
240 POKE n,21
250 POKE n,22
260 POKE n,23
270 POKE n,24
280 POKE n,25
290 POKE n,26
300 POKE n,27
310 POKE n,28
320 POKE n,29
330 POKE n,30
340 POKE n,31
350 POKE n,32
360 POKE n,33
370 POKE n,34
380 POKE n,35
390 POKE n,36
400 POKE n,37
410 POKE n,38
420 POKE n,39
430 POKE n,40
440 POKE n,41
450 POKE n,42
460 POKE n,43
470 POKE n,44
480 POKE n,45
490 POKE n,46
500 POKE n,47
510 POKE n,48
520 POKE n,49
530 POKE n,50
540 POKE n,51
550 POKE n,52
560 POKE n,53
570 POKE n,54
580 POKE n,55
590 POKE n,56
600 POKE n,57
610 POKE n,58
620 POKE n,59
630 POKE n,60
640 POKE n,61
650 POKE n,62
660 POKE n,63
670 POKE n,64
680 POKE n,65
690 POKE n,66
700 POKE n,67
710 POKE n,68
720 POKE n,69
730 POKE n,70
740 POKE n,71
750 POKE n,72
760 POKE n,73
770 POKE n,74
780 POKE n,75
790 POKE n,76
800 POKE n,77
810 POKE n,78
820 POKE n,79
830 POKE n,80
840 POKE n,81
850 POKE n,82
860 POKE n,83
870 POKE n,84
880 POKE n,85
890 POKE n,86
900 POKE n,87
910 POKE n,88
920 POKE n,89
930 POKE n,90
940 POKE n,91
950 POKE n,92
960 POKE n,93
970 POKE n,94
980 POKE n,95
990 POKE n,96
1000 POKE n,97
1010 POKE n,98
1020 POKE n,99
1030 POKE n,100
1040 POKE n,101
1050 POKE n,102
1060 POKE n,103
1070 POKE n,104
1080 POKE n,105
1090 POKE n,106
1100 POKE n,107
1110 POKE n,108
1120 POKE n,109
1130 POKE n,110
1140 POKE n,111
1150 POKE n,112
1160 POKE n,113
1170 POKE n,114
1180 POKE n,115
1190 POKE n,116
1200 POKE n,117
1210 POKE n,118
1220 POKE n,119
1230 POKE n,120
1240 POKE n,121
1250 POKE n,122
1260 POKE n,123
1270 POKE n,124
1280 POKE n,125
1290 POKE n,126
1300 POKE n,127
1310 POKE n,128
1320 POKE n,129
1330 POKE n,130
1340 POKE n,131
1350 POKE n,132
1360 POKE n,133
1370 POKE n,134
1380 POKE n,135
1390 POKE n,136
1400 POKE n,137
1410 POKE n,138
1420 POKE n,139
1430 POKE n,140
1440 POKE n,141
1450 POKE n,142
1460 POKE n,143
1470 POKE n,144
1480 POKE n,145
1490 POKE n,146
1500 POKE n,147
1510 POKE n,148
1520 POKE n,149
1530 POKE n,150
1540 POKE n,151
1550 POKE n,152
1560 POKE n,153
1570 POKE n,154
1580 POKE n,155
1590 POKE n,156
1600 POKE n,157
1610 POKE n,158
1620 POKE n,159
1630 POKE n,160
1640 POKE n,161
1650 POKE n,162
1660 POKE n,163
1670 POKE n,164
1680 POKE n,165
1690 POKE n,166
1700 POKE n,167
1710 POKE n,168
1720 POKE n,169
1730 POKE n,170
1740 POKE n,171
1750 POKE n,172
1760 POKE n,173
1770 POKE n,174
1780 POKE n,175
1790 POKE n,176
1800 POKE n,177
1810 POKE n,178
1820 POKE n,179
1830 POKE n,180
1840 POKE n,181
1850 POKE n,182
1860 POKE n,183
1870 POKE n,184
1880 POKE n,185
1890 POKE n,186
1900 POKE n,187
1910 POKE n,188
1920 POKE n,189
1930 POKE n,190
1940 POKE n,191
1950 POKE n,192
1960 POKE n,193
1970 POKE n,194
1980 POKE n,195
1990 POKE n,196
2000 POKE n,197
2010 POKE n,198
2020 POKE n,199
2030 POKE n,200
2040 POKE n,201
2050 POKE n,202
2060 POKE n,203
2070 POKE n,204
2080 POKE n,205
2090 POKE n,206
2100 POKE n,207
2110 POKE n,208
2120 POKE n,209
2130 POKE n,210
2140 POKE n,211
2150 POKE n,212
2160 POKE n,213
2170 POKE n,214
2180 POKE n,215
2190 POKE n,216
2200 POKE n,217
2210 POKE n,218
2220 POKE n,219
2230 POKE n,220
2240 POKE n,221
2250 POKE n,222
2260 POKE n,223
2270 POKE n,224
2280 POKE n,225
2290 POKE n,226
2300 POKE n,227
2310 POKE n,228
2320 POKE n,229
2330 POKE n,230
2340 POKE n,231
2350 POKE n,232
2360 POKE n,233
2370 POKE n,234
2380 POKE n,235
2390 POKE n,236
2400 POKE n,237
2410 POKE n,238
2420 POKE n,239
2430 POKE n,240
2440 POKE n,241
2450 POKE n,242
2460 POKE n,243
2470 POKE n,244
2480 POKE n,245
2490 POKE n,246
2500 POKE n,247
2510 POKE n,248
2520 POKE n,249
2530 POKE n,250
2540 POKE n,251
2550 POKE n,252
2560 POKE n,253
2570 POKE n,254
2580 POKE n,255
2590 POKE n,256
2600 POKE n,257
2610 POKE n,258
2620 POKE n,259
2630 POKE n,260
2640 POKE n,261
2650 POKE n,262
2660 POKE n,263
2670 POKE n,264
2680 POKE n,265
2690 POKE n,266
2700 POKE n,267
2710 POKE n,268
2720 POKE n,269
2730 POKE n,270
2740 POKE n,271
2750 POKE n,272
2760 POKE n,273
2770 POKE n,274
2780 POKE n,275
2790 POKE n,276
2800 POKE n,277
2810 POKE n,278
2820 POKE n,279
2830 POKE n,280
2840 POKE n,281
2850 POKE n,282
2860 POKE n,283
2870 POKE n,284
2880 POKE n,285
2890 POKE n,286
2900 POKE n,287
2910 POKE n,288
2920 POKE n,289
2930 POKE n,290
2940 POKE n,291
2950 POKE n,292
2960 POKE n,293
2970 POKE n,294
2980 POKE n,295
2990 POKE n,296
3000 POKE n,297
3010 POKE n,298
3020 POKE n,299
3030 POKE n,300
3040 POKE n,301
3050 POKE n,302
3060 POKE n,303
3070 POKE n,304
3080 POKE n,305
3090 POKE n,306
3100 POKE n,307
3110 POKE n,308
3120 POKE n,309
3130 POKE n,310
3140 POKE n,311
3150 POKE n,312
3160 POKE n,313
3170 POKE n,314
3180 POKE n,315
3190 POKE n,316
3200 POKE n,317
3210 POKE n,318
3220 POKE n,319
3230 POKE n,320
3240 POKE n,321
3250 POKE n,322
3260 POKE n,323
3270 POKE n,324
3280 POKE n,325
3290 POKE n,326
3300 POKE n,327
3310 POKE n,328
3320 POKE n,329
3330 POKE n,330
3340 POKE n,331
3350 POKE n,332
3360 POKE n,333
3370 POKE n,334
3380 POKE n,335
3390 POKE n,336
3400 POKE n,337
3410 POKE n,338
3420 POKE n,339
3430 POKE n,340
3440 POKE n,341
3450 POKE n,342
3460 POKE n,343
3470 POKE n,344
3480 POKE n,345
3490 POKE n,346
3500 POKE n,347
3510 POKE n,348
3520 POKE n,349
3530 POKE n,350
3540 POKE n,351
3550 POKE n,352
3560 POKE n,353
3570 POKE n,354
3580 POKE n,355
3590 POKE n,356
3600 POKE n,357
3610 POKE n,358
3620 POKE n,359
3630 POKE n,360
3640 POKE n,361
3650 POKE n,362
3660 POKE n,363
3670 POKE n,364
3680 POKE n,365
3690 POKE n,366
3700 POKE n,367
3710 POKE n,368
3720 POKE n,369
3730 POKE n,370
3740 POKE n,371
3750 POKE n,372
3760 POKE n,373
3770 POKE n,374
3780 POKE n,375
3790 POKE n,376
3800 POKE n,377
3810 POKE n,378
3820 POKE n,379
3830 POKE n,380
3840 POKE n,381
3850 POKE n,382
3860 POKE n,383
3870 POKE n,384
3880 POKE n,385
3890 POKE n,386
3900 POKE n,387
3910 POKE n,388
3920 POKE n,389
3930 POKE n,390
3940 POKE n,391
3950 POKE n,392
3960 POKE n,393
3970 POKE n,394
3980 POKE n,395
3990 POKE n,396
4000 POKE n,397
4010 POKE n,398
4020 POKE n,399
4030 POKE n,400
4040 POKE n,401
4050 POKE n,402
4060 POKE n,403
4070 POKE n,404
4080 POKE n,405
4090 POKE n,406
4100 POKE n,407
4110 POKE n,408
4120 POKE n,409
4130 POKE n,410
4140 POKE n,411
4150 POKE n,412
4160 POKE n,413
4170 POKE n,414
4180 POKE n,415
4190 POKE n,416
4200 POKE n,417
4210 POKE n,418
4220 POKE n,419
4230 POKE n,420
4240 POKE n,421
4250 POKE n,422
4260 POKE n,423
4270 POKE n,424
4280 POKE n,425
4290 POKE n,426
4300 POKE n,427
4310 POKE n,428
4320 POKE n,429
4330 POKE n,430
4340 POKE n,431
4350 POKE n,432
4360 POKE n,433
4370 POKE n,434
4380 POKE n,435
4390 POKE n,436
4400 POKE n,437
4410 POKE n,438
4420 POKE n,439
4430 POKE n,440
4440 POKE n,441
4450 POKE n,442
4460 POKE n,443
4470 POKE n,444
4480 POKE n,445
4490 POKE n,446
4500 POKE n,447
4510 POKE n,448
4520 POKE n,449
4530 POKE n,450
4540 POKE n,451
4550 POKE n,452
4560 POKE n,453
4570 POKE n,454
4580 POKE n,455
4590 POKE n,456
4600 POKE n,457
4610 POKE n,458
4620 POKE n,459
4630 POKE n,460
4640 POKE n,461
4650 POKE n,462
4660 POKE n,463
4670 POKE n,464
4680 POKE n,465
4690 POKE n,466
4700 POKE n,467
4710 POKE n,468
4720 POKE n,469
4730 POKE n,470
4740 POKE n,471
4750 POKE n,472
4760 POKE n,473
4770 POKE n,474
4780 POKE n,475
4790 POKE n,476
4800 POKE n,477
4810 POKE n,478
4820 POKE n,479
4830 POKE n,480
4840 POKE n,481
4850 POKE n,482
4860 POKE n,483
4870 POKE n,484
4880 POKE n,485
4890 POKE n,486
4900 POKE n,487
4910 POKE n,488
4920 POKE n,489
4930 POKE n,490
4940 POKE n,491
4950 POKE n,492
4960 POKE n,493
4970 POKE n,494
4980 POKE n,495
4990 POKE n,496
5000 POKE n,497
5010 POKE n,498
5020 POKE n,499
5030 POKE n,500
5040 POKE n,501
5050 POKE n,502
5060 POKE n,503
5070 POKE n,504
5080 POKE n,505
5090 POKE n,506
5100 POKE n,507
5110 POKE n,508
5120 POKE n,509
5130 POKE n,510
5140 POKE n,511
5150 POKE n,512
5160 POKE n,513
5170 POKE n,514
5180 POKE n,515
5190 POKE n,516
5200 POKE n,517
5210 POKE n,518
5220 POKE n,519
5230 POKE n,520
5240 POKE n,521
5250 POKE n,522
5260 POKE n,523
5270 POKE n,524
5280 POKE n,525
5290 POKE n,526
5300 POKE n,527
5310 POKE n,528
5320 POKE n,529
5330 POKE n,530
5340 POKE n,531
5350 POKE n,532
5360 POKE n,533
5370 POKE n,534
5380 POKE n,535
5390 POKE n,536
5400 POKE n,537
5410 POKE n,538
5420 POKE n,539
5430 POKE n,540
5440 POKE n,541
5450 POKE n,542
5460 POKE n,543
5470 POKE n,544
5480 POKE n,545
5490 POKE n,546
5500 POKE n,547
5510 POKE n,548
5520 POKE n,549
5530 POKE n,550
5540 POKE n,551
5550 POKE n,552
5560 POKE n,553
5570 POKE n,554
5580 POKE n,555
5590 POKE n,556
5600 POKE n,557
5610 POKE n,558
5620 POKE n,559
5630 POKE n,560
5640 POKE n,561
5650 POKE n,562
5660 POKE n,563
5670 POKE n,564
5680 POKE n,565
5690 POKE n,566
5700 POKE n,567
5710 POKE n,568
5720 POKE n,569
5730 POKE n,570
5740 POKE n,571
5750 POKE n,572
5760 POKE n,573
5770 POKE n,574
5780 POKE n,575
5790 POKE n,576
5800 POKE n,577
5810 POKE n,578
5820 POKE n,579
5830 POKE n,580
5840 POKE n,581
5850 POKE n,582
5860 POKE n,583
5870 POKE n,584
5880 POKE n,585
5890 POKE n,586
5900 POKE n,587
5910 POKE n,588
5920 POKE n,589
5930 POKE n,590
5940 POKE n,591
5950 POKE n,592
5960 POKE n,593
5970 POKE n,594
5980 POKE n,595
5990 POKE n,596
6000 POKE n,597
6010 POKE n,598
6020 POKE n,599
6030 POKE n,600
6040 POKE n,601
6050 POKE n,602
6060 POKE n,603
6070 POKE n,604
6080 POKE n,605
6090 POKE n,606
6100 POKE n,607
6110 POKE n,608
6120 POKE n,609
6130 POKE n,610
6140 POKE n,611
6150 POKE n,612
6160 POKE n,613
6170 POKE n,614
6180 POKE n,615
6190 POKE n,616
6200 POKE n,617
6210 POKE n,618
6220 POKE n,619
6230 POKE n,620
6240 POKE n,621
6250 POKE n,622
6260 POKE n,623
6270 POKE n,624
6280 POKE n,625
6290 POKE n,626
6300 POKE n,627
6310 POKE n,628
6320 POKE n,629
6330 POKE n,630
6340 POKE n,631
6350 POKE n,632
6360 POKE n,633
6370 POKE n,634
6380 POKE n,635
6390 POKE n,636
6400 POKE n,637
6410 POKE n,638
6420 POKE n,639
6430 POKE n,640
6440 POKE n,641
6450 POKE n,642
6460 POKE n,643
6470 POKE n,644
6480 POKE n,645
6490 POKE n,646
6500 POKE n,647
6510 POKE n,648
6520 POKE n,649
6530 POKE n,650
6540 POKE n,651
6550 POKE n,652
6560 POKE n,653
6570 POKE n,654
6580 POKE n,655
6590 POKE n,656
6600 POKE n,657
6610 POKE n,658
6620 POKE n,659
6630 POKE n,660
6640 POKE n,661
6650 POKE n,662
6660 POKE n,663
6670 POKE n,664
6680 POKE n,665
6690 POKE n,666
6700 POKE n,667
6710 POKE n,668
6720 POKE n,669
6730 POKE n,670
6740 POKE n,671
6750 POKE n,672
6760 POKE n,673
6770 POKE n,674
6780 POKE n,675
6790 POKE n,676
6800 POKE n,677
6810 POKE n,678
6820 POKE n,679
6830 POKE n,680
6840 POKE n,681
6850 POKE n,682
6860 POKE n,683
6870 POKE n,684
6880 POKE n,685
6890 POKE n,686
6900 POKE n,687
6910 POKE n,688
6920 POKE n,689
6930 POKE n,690
6940 POKE n,691
6950 POKE n,692
6960 POKE n,693
6970 POKE n,694
6980 POKE n,695
6990 POKE n,696
7000 POKE n,697
7010 POKE n,698
7020 POKE n,699
7030 POKE n,700
7040 POKE n,701
7050 POKE n,702
7060 POKE n,703
7070 POKE n,704
7080 POKE n,705
7090 POKE n,706
7100 POKE n,707
7110 POKE n,708
7120 POKE n,709
7130 POKE n,710
7140 POKE n,711
7150 POKE n,712
7160 POKE n,713
7170 POKE n,714
7180 POKE n,715
7190 POKE n,716
7200 POKE n,717
7210 POKE n,718
7220 POKE n,719
7230 POKE n,720
7240 POKE n,721
7250 POKE n,722
7260 POKE n,723
7270 POKE n,724
7280 POKE n,725
7290 POKE n,726
7300 POKE n,727
7310 POKE n,728
7320 POKE n,729
7330 POKE n,730
7340 POKE n,731
7350 POKE n,732
7360 POKE n,733
7370 POKE n,734
7380 POKE n,735
7390 POKE n,736
7400 POKE n,737
7410 POKE n,738
7420 POKE n,739
7430 POKE n,740
7440 POKE n,741
7450 POKE n,742
7460 POKE n,743
7470 POKE n,744
7480 POKE n,745
7490 POKE n,746
7500 POKE n,747
7510 POKE n,748
7520 POKE n,749
7530 POKE n,750
7540 POKE n,751
7550 POKE n,752
7560 POKE n,753
7570 POKE n,754
7580 POKE n,755
7590 POKE n,756
7600 POKE n,757
7610 POKE n,758
7620 POKE n,759
7630 POKE n,760
7640 POKE n,761
7650 POKE n,762
7660 POKE n,763
7670 POKE n,764
7680 POKE n,765
7690 POKE n,766
7700 POKE n,767
7710 POKE n,768
7720 POKE n,769
7730 POKE n,770
7740 POKE n,771
7750 POKE n,772
7760 POKE n,773
7770 POKE n,774
7780 POKE n,775
7790 POKE n,776
7800 POKE n,777
7810 POKE n,778
7820 POKE n,779
7830 POKE n,780
7840 POKE n,781
7850 POKE n,782
7860 POKE n,783
7870 POKE n,784
7880 POKE n,785
7890 POKE n,786
7900 POKE n,787
7910 POKE n,788
7920 POKE n,789
7930 POKE n,790
7940 POKE n,791
7950 POKE n,792
7960 POKE n,793
7970 POKE n,794
7980 POKE n,795
7990 POKE n,796
8000 POKE n,797
8010 POKE n,798
8020 POKE n,799
8030 POKE n,800
8040 POKE n,801
8050 POKE n,802
8060 POKE n,803
8070 POKE n,804
8080 POKE n,805
8090 POKE n,806
8100 POKE n,807
8110 POKE n,808
8120 POKE n,809
8130 POKE n,810
8140 POKE n,811
8150 POKE n,812
8160 POKE n,813
8170 POKE n,814
8180 POKE n,815
8190 POKE n,816
8200 POKE n,817
8210 POKE n,818
8220 POKE n,819
8230 POKE n,820
8240 POKE n,821
8250 POKE n,822
8260 POKE n,823
8270 POKE n,824
8280 POKE n,825
8290 POKE n,826
8300 POKE n,827
8310 POKE n,828
8320 POKE n,829
8330 POKE n,830
8340 POKE n,831
8350 POKE n,832
8360 POKE n,833
8370 POKE n,834
8380 POKE n,835
8390 POKE n,836
8400 POKE n,837
8410 POKE n,838
8420 POKE n,839
8430 POKE n,840
8440 POKE n,841
8450 POKE n,842
8460 POKE n,843
8470 POKE n,844
8480 POKE n,845
8490 POKE n,846
8500 POKE n,847
8510 POKE n,848
8520 POKE n,849
8530 POKE n,850
8540 POKE n,851
8550 POKE n,852
8560 POKE n,853
8570 POKE n,854
8580 POKE n,855
8590 POKE n,856
8600 POKE n,857
8610 POKE n,858
8620 POKE n,859
8630 POKE n,860
8640 POKE n,861
8650 POKE n,862
8660 POKE n,863
8670 POKE n,864
8680 POKE n,865
8690 POKE n,866
8700 POKE n,867
8710 POKE n,868
8720 POKE n,869
8730 POKE n,870
8740 POKE n,871
8750 POKE n,872
8760 POKE n,873
8770 POKE n,874
8780 POKE n,875
8790 POKE n,876
8800 POKE n,877
8810 POKE n,878
8820 POKE n,879
8830 POKE n,880
8840 POKE n,881
8850 POKE n,882
8860 POKE n,883
8870 POKE n,884
8880 POKE n,885
8890 POKE n,886
8900 POKE n,887
8910 POKE n,888
8920 POKE n,889
8930 POKE n,890
8940 POKE n,891
8950 POKE n,892
8960 POKE n,893
8970 POKE n,894
8980 POKE n,895
8990 POKE n,896
9000 POKE n,897
9010 POKE n,898
9020 POKE n,899
9030 POKE n,900
9040 POKE n,901
9050 POKE n,902
9060 POKE n,903
9070 POKE n,904
9080 POKE n,905
9090 POKE n,906
9100 POKE n,907
9110 POKE n,908
9120 POKE n,909
9130 POKE n,910
9140 POKE n,911
9150 POKE n,912
9160 POKE n,913
9170 POKE n,914
9180 POKE n,915
9190 POKE n,916
9200 POKE n,917
9210 POKE n,918
9220 POKE n,919
9230 POKE n,920
9240 POKE n,921
9250 POKE n,922
9260 POKE n,923
9270 POKE n,924
9280 POKE n,925
9290 POKE n,926
9300 POKE n,927
9310 POKE n,928
9320 POKE n,929
9330 POKE n,930
9340 POKE n,931
9350 POKE n,932
9360 POKE n,933
9370 POKE n,934
9380 POKE n,935
9390 POKE n,936
9400 POKE n,937
9410 POKE n,938
9420 POKE n,939
9430 POKE n,940
9440 POKE n,941
9450 POKE n,942
9460 POKE n,943
9470 POKE n,944
9480 POKE n,945
9490 POKE n,946
9500 POKE n,947
9510 POKE n,948
9520 POKE n,949
9530 PO
```



## CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.

## CICLOS DE MEMORIA:

2

## CICLOS DE RELOJ:

7

## EJEMPLO:

AND (HL)

Valor del par de registros "HL"

(H): 11111111  
(L): 00000000

Valor de la posición de memoria FF00h

(FF00h): 10101010

Valor del registro "A"

(A): 01010101

## Instrucción

AND (HL); 10100110

Valor del registro "A" después de la ejecución

(A): 00000000

Indicadores de condición después de la ejecución

S Z H P/V N C

0 1 x 1 x 1 0 0

Se ha activado el "P/V" porque el cero se considera como "par", es decir, si hay "cero" unos en el resultado, se considera que el número de unos es par.

AND (IX+d)

## OBJETO:

Realiza una operación lógica AND, bit a bit, entre el octeto del registro acumulador y el octeto indicado por el operando. La dirección del octeto del operando es la que resulta de añadir al contenido del registro índice "IX" el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación se deja en el registro acumulador.

## CODIGO DE MAQUINA:

11011101  
10100110

## INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

5

## CICLOS DE RELOJ:

19

## EJEMPLO:

AND (IX+48)

Valor del registro índice "IX"

IX: 01011101  
50h: 00100101  
25h: 00100101

Valor de la posición de memoria 5D55h

(5D55h): 10011011

Valor del registro "A"

FFh: 11111111

## Instrucción

AND (IX+48)  
11011101  
10100110  
00110010

## Operación:

11111111  
AND 10011011  
= 10011011

Valor del registro "A" después de la ejecución

(A): 10011011

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 1 x 0 0 0

bado que es correcto se pone a cero el bit 7 para volver el mensaje al código ASCII. Si la paridad no fuera par, se habría detectado un error, y en función del trabajo que sea, se saca una información por la pantalla o se le pide al otro ordenador que repita el mensaje.

Si siguiendo con el ejemplo,

01001101  
par - correcto  
=> 01001101 "H"  
11001001  
par - correcto  
=> 01001001 "I"  
10100000  
par - correcto  
=> 00100000 " "  
01001110  
par - correcto  
=> 01001110 "N"

y así con todos los octetos del mensaje. Empleando instrucciones vistas hasta ahora se haría,

LD HL, MENSAJE ;Dirección mensaje.  
OTRO LD A, (HL) ;Carga en A los 8 bits del octeto.  
AND A ;Comprueba la paridad.  
< Si la paridad es impar >  
< se informa del error. >  
AND #7F ;Máscara para rehacer el código ASCII.  
LD (HL), A ;Guarda el carácter.  
INC HL ;Posición siguiente carácter.  
< Si no es el final del >  
< mensaje salta a OTRO. >  
< Si ha tratado todos los >  
< caracteres continua el >  
< proceso normal. >

Fig. 6-9. Construcción de controles de paridad con octeto de control.



```

=> pasa a 11001001

" " = 00100000
>>> 1 bit a 1 <<<
=> pasa a 10100000

"N" = 01001110
>>> 4 bits a 1 <<<
=> no se modifica

```

y así con todos los octetos del mensaje. Con las instrucciones vistas hasta el momento, se puede hacer:

```

LD HL, MENSAJE ; direcciona mensaje.
OTRA LD A, (HL) ; carga en A los 7 bits del
                ; código ASCII del carácter.
AND A          ; comprueba la paridad.
< Si la paridad es par >
< no hace nada y salta >
< a "CONT". >

AND A, 00000001 ; poner bit de paridad.
LD (HL), A       ; salvar nuevo octeto.
CONT INC HL      ; posicionar siguiente oct.

< Si no es final de mensaje >
< sigue saliendo a "OTRA". >

< Si ha tratado todos los >
< caracteres continúa el >
< proceso normal. >

```

En el programa ejemplo se destacan con los signos < > las que serían instrucciones de salto condicional que se verán más adelante, pero que ahora no son fundamentales para entender este procedimiento.

Ver, a este respecto, el organigrama de la FIGURA 6-7.

3) Para entrada, una vez recibido todo el mensaje, se irá tomando octeto a octeto viendo que la paridad sea siempre par; una vez compro-

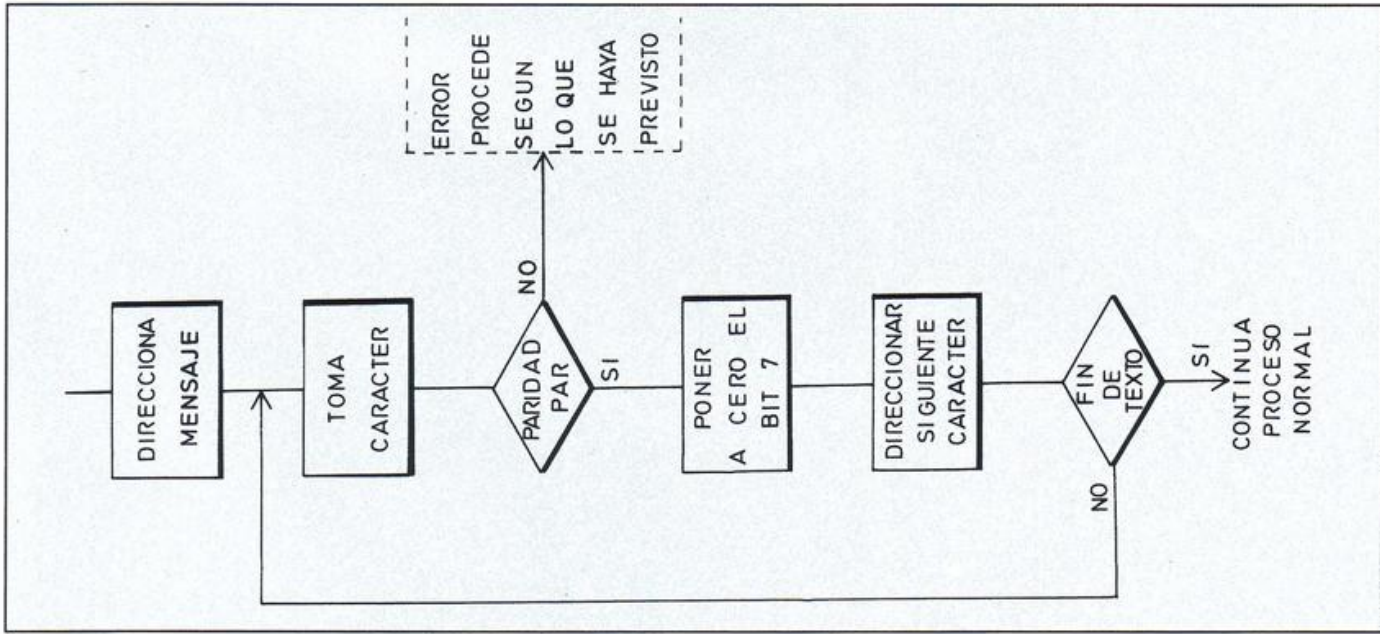


Fig. 6-8. Control de paridad en entrada.

## AND (IY+d)

### OBJETO:

Realiza una operación lógica AND, bit a bit, entre el octeto del registro acumulador y el octeto indicado por el operando. La dirección del octeto del operando es la que resulta de añadir al contenido del registro índice "IY" el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación se deja en el registro acumulador.

### CODIGO DE MAQUINA:

11111101	FDh
10100110	ADh
<-----d----->	

### EJEMPLO:

AND (IX+0)

Valor del registro índice "IY":

01111011	7Bh
10111000	8Bh

Valor de la posición de memoria 78B8h:

1110011000	0C8h
------------	------

Valor del registro "A":

00001000	08h
----------	-----

AND (IY+0)

11111101	FDh
10100110	ADh
00000000	00h

Operación:

### Control de paridad

Es fácil que algún lector se pregunte por la utilidad que tiene saber si un octeto tiene paridad par o impar.

Recordemos que paridad par en un octeto es cuando el número de bits activos (unos) que tiene es par y paridad impar es cuando el número de bits activos es impar.

### Ejemplos:

01101010	4 bits activos paridad par
01001010	3 bits activos paridad impar

00001000  
AND 11001100  
00001000

Valor del registro "A" después de la ejecución

(A): 00001000 08h

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	1	x	0

La activación de los indicadores de condición en las instrucciones AND, se hacen según las siguientes reglas:

"S": En este indicador se pone el mismo valor que el bit 7 del octeto después de la

### CICLOS DE MEMORIA:

5

### CICLOS DE RELOJ:

19



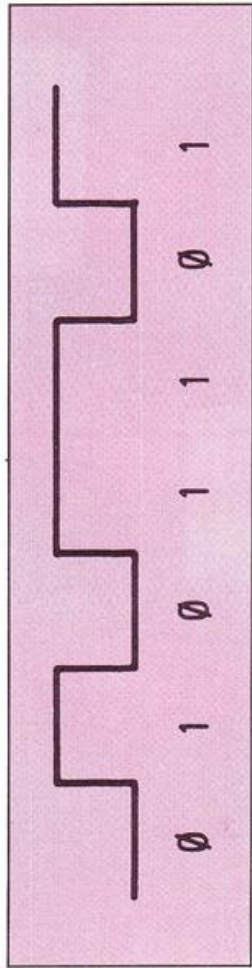


Fig. 6-6. Octeto en onda cuadrada.

El uso más frecuente de la paridad del octeto, también llamado paridad de carácter, es garantizar una correcta información. Cuando se quiere guardar un texto en algún periférico o bien se quiere enviar un texto a otro ordenador (no olvidar la posibilidad que existe de conectar dos SPEC-TRUM por red de área local) se usa la paridad de carácter para detectar cualquier error de un bit en el trasiego de la información que deteriore el octeto o carácter a que corresponde, por lo tanto deteriora la información enviada. Pongamos por ejemplo el envío del siguiente mensaje:

MI NOMBRE ES PEPE

Este mensaje codificado en el código ASCII, expresado en hexadecimal, quedaría como sigue:

"M" = 4Dh  
 "I" = 49h  
 " " = 20h  
 "N" = 4Eh  
 "O" = 4Fh  
 "M" = 4Dh  
 "B" = 42h  
 "R" = 52h  
 "E" = 45h  
 " " = 20h  
 "E" = 45h

"S" = 53h  
 " " = 20h  
 "P" = 50h  
 "E" = 45h  
 "P" = 50h  
 "E" = 45h

Al enviar este mensaje a un periférico (cassette, microdrive, etc.) o por una línea telefónica saldría como una sucesión de ceros y unos, esto es, una onda que tiende a ser cuadrada y que cuando tiene un nivel vale 0 y cuando tiene otro vale 1. Ver FIGURA 6-5.

De todas formas es más fácil expresarse en binario que en ondas cuadradas, lo único necesario es saber cómo se envían datos para entender el problema que puede acontecer.

El mensaje que sirve de ejemplo codificado en binario sería el siguiente:

01001101 = "M"  
 01001001 = "I"  
 00100000 = " "  
 01001110 = "N"  
 01001111 = "O"  
 01001101 = "M"  
 01000010 = "B"  
 01010010 = "R"  
 01000101 = "E"  
 00100000 = " "  
 01000101 = "E"

01010011 = "S"  
 00100000 = " "  
 01010000 = "P"  
 01000101 = "E"  
 01010000 = "P"  
 01000101 = "E"

En este caso el mensaje saldría por una línea hacia el dispositivo adecuado como una onda cuadrada. Supongamos por un momento que cualquier interferencia o ruido cambia el bit 0 del cuarto carácter por un "1" y el del bit 2 del último carácter por un "0". Esos dos caracteres quedarían de la siguiente manera:

Carácter enviado:  
 01001110 = 4Eh = "N"  
 (cambia bit 0)  
 Carácter modificado:  
 01001111 = 4Fh = "O"  
 Carácter enviado:  
 01000101 = 45h = "E"  
 (cambia bit 2)

Carácter modificado:  
 01000001 = 41h = "A"

En este caso el mensaje almacenado en el periférico o recibido en otro ordenador sería:

MI NOMBRE ES PEPA

Terrible confusión...

Este tipo de problemas se solucionan utilizando la paridad. Vamos a verlo haciendo uso, en lo posible, de las instrucciones Assembler vistas hasta el momento.

1) Hay que llegar a un acuerdo con nosotros mismos o con el usuario del otro ordenador. Por ejemplo en este caso llegamos al acuerdo de enviar mensajes con octetos que siempre tengan la paridad par.

2) Para la salida, cuando ya se ha formado el mensaje, se irá tomando octeto a octeto y viendo la paridad que tiene; si es par se deja como es, si es impar se pone a 1 el bit 7 del octeto. El código ASCII sólo ocupa siete bits, por lo tanto el octavo, que es el 7, puede servir para controlar la paridad.

Siguiendo con el ejemplo:

"M" = 01001101  
 >>> 4 bits a 1 <<<  
 => no se modifica  
 "I" = 01001001  
 >>> 3 bits a 1 <<<

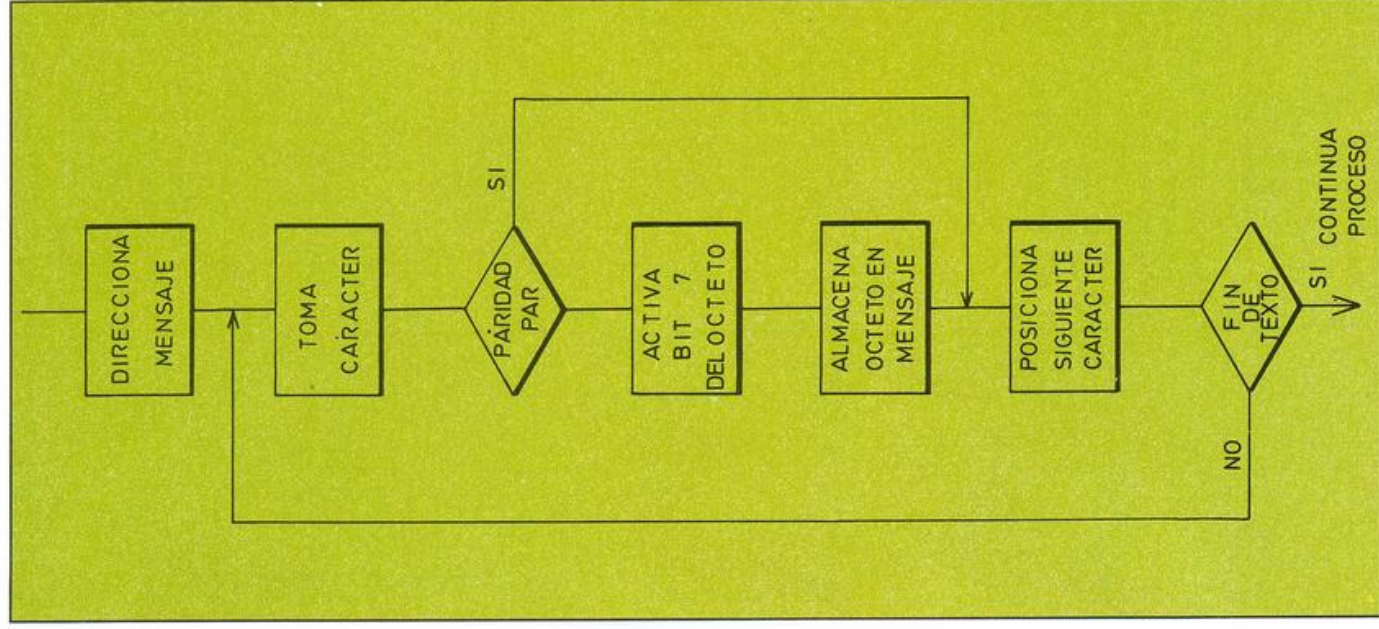


Fig. 6-7. Control de paridad en salida.



operar un número con su complementario, el resultado es "FFh". Si no entiendes por qué, repase atentamente lo visto hasta aquí del operador "XOR".

Efectivamente, habíamos dicho que al sumar un número con su complementario, el resultado era siempre "FFh"; pero lo que hace el operador "XOR" es, precisamente, sumar sin acarreo los bits uno a uno (el acarreo lo da el operador "AND"); entre números complementarios no hay acarreo, por tanto, da igual sumarlos que "XORearlos".

## XOR (IX+d)

### OBJETO:

Realiza una operación lógica XOR, bit a bit, entre el octeto del registro acumulador y el octeto indicado por el operando. La dirección del octeto del operando es la que resulta de añadir al contenido del registro índice "IX" el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación se deja en el registro acumulador.

### CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1	DDh
1 0 1 0 1 1 1 0	AEh
<-----d----->	

### INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

### CICLOS DE MEMORIA:

5

### CICLOS DE RELOJ:

19

### EJEMPLO:

XOR (IX+15)

Valor del registro índice "IX"

1 0 0 0 1 1 1 0	8Eh
0 0 1 1 1 0 1 0	3Ah

Valor de la posición de memoria 8E49h

8E49h: 1 0 1 0 1 0 1 0

Valor del registro "A"

Ah: 1 0 1 0 1 0 1 0

### Instrucción

1 0 1 1 1 0 1	DDh
1 0 1 0 1 1 1 0	AEh
0 0 0 0 1 1 1 1	0Fh

Valor del registro "A" después de la ejecución

Ah: 0 0 0 0 0 0 0 0

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	1	x	0	x	1 0 0 0

## XOR (IY+d)

### OBJETO:

Realiza una operación lógica XOR, bit a bit, entre el octeto del registro acumulador y el octeto indicado por el operando. La dirección del octeto del operando es la que resulta de añadir al contenido del registro índice "IY" el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación se deja en el registro acumulador.

### CODIGO DE MAQUINA:

1 1 1 1 1 1 0 1	FDh
1 0 1 0 1 1 1 0	AEh
<-----d----->	

### INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

### CICLOS DE MEMORIA:

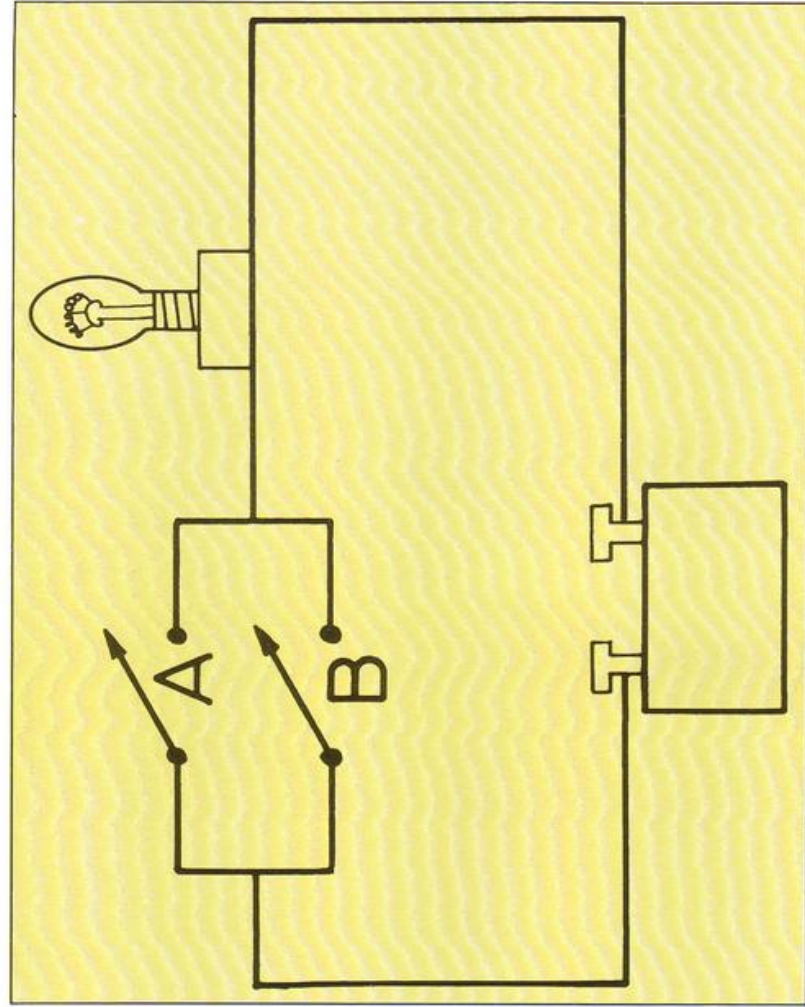


Fig. 6-12. Representación eléctrica de "OR".

### "tabla de verdad":

1	OR	1	=	1
1	OR	0	=	1
0	OR	1	=	1
0	OR	0	=	0

Su analogía eléctrica viene dada por dos interruptores conectados en paralelo (Ver FIGURA 6-12) para que la bombilla se encienda es suficiente con que uno de ellos ("A" o "B") esté cerrado.

Básicamente el formato de esta instrucción es:

### OR OPERANDO

El octeto indicado por el operando se enfrenta con el octeto del registro acumulador, ambos octetos se operan

bit a bit, el resultado se deja en el acumulador y el operando no sufre variación. En los indicadores de estado (registro "F") se anota la ocurrencia de "cero", "signo" o "paridad".

## OR r

### OBJETO:

Realiza una operación lógica OR, bit a bit, entre el octeto del registro acumulador y el octeto del registro indicado por "r". El resultado se deja en el registro acumulador.

Al igual que en el resto de instrucciones, los bits que identifican a "r" tienen el si-



guiente formato:

Registro	<--r-->
A	111
B	000
C	001
D	010
E	011
H	100
L	101

CODIGO DE MAQUINA:

1 0 1 1 0 <--r-->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

OR B

Vamos a realizar un "OR" del acumulador con el registro "B", lo que equivale a enfrentar cada bit del registro "B" con su correspondiente bit del acumulador; el resultado de cada bit será "1" si, y

solo si, alguno de los dos bits enfrentados es "1". Recuerdese que, en la operación "AND", era necesario que ambos bits fueran "1" para que el resultado fuera "1"; aqui es necesario que ambos bits sean "0" para que el resultado sea "0". Por esta razón, a veces se dice que la operación "OR" es la opuesta de la "AND".

Valor del registro "A"

(A): 0 1 0 0 0 1 1 0 46h

Valor del registro "B"

(B): 0 1 1 0 1 1 0 0 6Ch

Instrucción

OR B: 1 0 1 1 0 0 0 0 B0h

Valor del registro "A" después de la ejecución

(A): 0 1 1 0 1 1 1 0 6Eh

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 0 x 0 x 0 0 0 0

Al igual que ocurría con "AND", el formato de esta instrucción permite realizar un "OR" del acumulador consigo mismo. En este caso la instrucción "OR A" tiene, exactamente, el mismo efecto que "AND A", salvo que el indicador "H" del registro "F" se pone a "0" en lugar de a "1" como lo hacía con "AND A".

En lo que a nosotros nos interesa, podemos usar indistintamente "OR A" o "AND A" para poner a cero el bit de acarreo o comprobar la paridad del dato contenido en el

acumulador sin alterarlo.

OR n

OBJETO:

Realiza una operación lógica OR, bit a bit, entre el octeto del registro acumulador y el valor binario de "n". El resultado se deja en el registro acumulador.

CODIGO DE MAQUINA:

1 1 1 1 0 1 1 0 F6h

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

OR 7

Esta instrucción podría haberse escrito en hexadecimal como:

OR #07

Realiza una operación lógica XOR, bit a bit, entre el octeto del registro acumulador y el valor binario de "n". El resultado se deja en el registro acumulador.

CODIGO DE MAQUINA:

1 1 1 0 1 1 1 0 EEh

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

XOR 30

Como ya sabrá el lector, esta instrucción podría haberse escrito:

XOR #1E

En hexadecimal; o bien, en binario:

XOR %00011110

Valor del registro "A"

(A): 1 1 0 1 1 0 0 0 D8h

Instrucción

XOR 30: 1 1 1 0 1 1 1 0 EEh  
0 0 0 1 1 1 1 0 1Eh

Valor del registro "A" después de la ejecución

(A): 1 1 0 0 0 1 1 0 C6h

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 0 x 1 0 0 0

XOR (HL)

OBJETO:

Realiza una operación lógica XOR, bit a bit, entre el octeto del registro acumulador y el octeto de la posición de memoria direccionado por el contenido del par de registros "HL". El resultado se deja en el registro acumulador.

CODIGO DE MAQUINA:

1 0 1 0 1 1 1 0 AEh

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier

otro caso

H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone 0 - siempre  
P/V; pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

XOR (HL)

Valor del par de registros "HL"

(H): 1 0 1 0 0 0 1 1 A3h  
(L): 1 0 0 0 0 0 1 0 82h

Valor de la posición de memoria A382h

(A382h): 1 0 1 0 1 0 1 0 A6h

Valor del registro "A"

(A): 0 1 0 1 0 1 0 1 55h

Instrucción

XOR (HL): 1 0 1 0 1 1 1 0 AEh

Valor del registro "A" después de la ejecución

(A): 1 1 1 1 1 1 1 1 FFh

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 0 x 1 0 0 0

Como era de esperar, al



mantiene a cero.  
N: Este indicador carece de significado para estas instrucciones y se pone siempre a 0.

C: Este indicador carece de significado para estas instrucciones y se pone siempre a 0.

XOR: "exclusive OR" en inglés; se traduciría al castellano por: "O EXCLUSIVO". Con esta palabra se define una operación lógica que consiste en enfrentar los operandos bit a bit, de modo que cuando alguno de los dos bit enfrentados, y solo uno, tiene el valor 1 el bit resultado es 1, cuando ambos son 1 ó 0 el resultado es 0. Su tabla de verdad sería:

1	XOR	1	=	0
1	XOR	0	=	1
0	XOR	1	=	1
0	XOR	0	=	0

El circuito eléctrico que más se asemeja a esta instrucción sería el formado por dos interruptores conmutados de los que se utilizan frecuentemente para encender o apagar la luz de una habitación desde dos puntos distintos (ver FIGURA 6-13), en este circuito, variando solo uno de los interruptores, se enciende la bombilla y cuando se varían los dos no.

Básicamente el formato de esta instrucción es:

XOR OPERANDO

El octeto indicado por el operando se enfrenta, bit a bit, con el octeto del registro acumulador, el resultado se deja en este último y el operando no sufre variación.

## XOR r

### OBJETO:

Realiza una operación lógica XOR, bit a bit, entre el octeto del registro acumulador y el octeto del registro indicado por "r". El resultado se deja en el registro acumulador.

### CODIGO DE MAQUINA:

1 0 1 0 1 <--- r --->

### INDICADORES DE CONDICION A LOS QUE AFECTA:

S: pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z: pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H: pone 0 - siempre  
N: pone 0 - siempre  
C: pone 0 - siempre  
P/V: pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

### CICLOS DE MEMORIA:

1

### CICLOS DE RELOJ:

4

### EJEMPLO:

XOR C

Vamos a realizar un "XOR" lógico entre los contenidos de los registros "A" y "C". El primer operando ("A") se omite ya que no puede ser otro. Al igual que en todas las opera-

ciones lógicas, es irrelevante el orden de los operandos (cumplen la propiedad conmutativa). Cada bit del resultado será "1" si los dos bits correspondientes de cada operando son distintos, y será "0" si ambos son iguales. El resultado quedará, como de costumbre, en el registro "A".  
Valor del registro "A"

(A): 1 1 0 0 1 1 0 1

Valor del registro "C"

(C): 1 0 0 1 1 0 1 1

Instrucción

XOR C: 1 0 1 0 1 0 0 1

Valor del registro "A" después de la ejecución

(A): 0 1 0 1 0 1 1 0

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 0 x 0 x 1 0 0

La sintaxis de esta instrucción permite realizar un "XOR" del acumulador consigo mismo que dará, lógicamente, un resultado de "00h". La instrucción "XOR A" es una forma sencilla de cargar un "0" en el registro acumulador, ya que al enfrentar dos octetos iguales con una operación lógica "XOR" el resultado es "0" en todos los bits.

## XOR n

### OBJETO:

En algunos ensambladores es, también, posible escribir los operandos en binario; más adelante veremos que esto puede resultarnos, a veces, muy útil; en el caso concreto del GENS 3, podríamos haber escrito:

OR %00000111

Dado que este ensamblador utiliza el signo "%" para indicar que el número que sigue está en binario.

Valor del registro "A"

(A): 1 1 1 1 0 0 0 0

Instrucción

OR 7: 1 1 1 1 0 1 1 0  
0 0 0 0 0 1 1 1

Valor del registro "A" después de la ejecución

(A): 1 1 1 1 0 1 1 1

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 0 x 0 x 0 0 0

## OR (HL)

### OBJETO:

Realiza una operación lógica OR, bit a bit, entre el octeto del registro acumulador y el octeto de la posición de memoria direccionado por el contenido del par de registros "HL". El resultado se deja en el registro acumulador.

### CODIGO DE MAQUINA:

1 0 1 1 0 1 1 0

### INDICADORES DE CONDICION A LOS QUE AFECTA:

S: pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z: pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H: pone 0 - siempre  
N: pone 0 - siempre  
C: pone 0 - siempre  
P/V: pone 1 - si la paridad es par.  
pone 0 - en cualquier otro caso

### CICLOS DE MEMORIA:

2

### CICLOS DE RELOJ:

7

### EJEMPLO:

OR (HL)

Valor del par de registros "HL"

(H): 0 1 1 1 0 1 0 0  
(L): 1 0 0 0 1 0 1 0

Valor de la posición de memoria 748Ah

(748Ah): 0 1 1 0 1 0 0 1

Valor del registro "A"

(A): 1 0 0 0 1 1 1 0

Instrucción

OR (HL): 1 0 1 1 0 1 1 0

Valor del registro "A" después de la ejecución

pués de la ejecución

(A): 1 1 1 0 1 1 1 1

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 0 x 0 0 0

## OR (IX+d)

### OBJETO:

Realiza una operación lógica OR, bit a bit, entre el octeto del registro acumulador y el octeto indicado por el operando. La dirección del octeto operando es la que resulta de añadir al contenido del registro índice "IX" el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación se deja en el registro acumulador.

### CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1  
1 0 1 1 0 1 1 0  
<--- d --->

### INDICADORES DE CONDICION A LOS QUE AFECTA:

S: pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z: pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H: pone 0 - siempre  
N: pone 0 - siempre  
C: pone 0 - siempre  
P/V: pone 1 - si la paridad es par.



pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

OR (IX-15)

Valor del registro índice "IX"

(IX):  
1 0 0 1 0 0 1 0  
1 0 1 1 0 1 0 0

Valor de la posición de memoria 92A5h

(92A5h):  
1 1 1 1 1 1 1 1

Valor del registro "A"

0 1 0 1 0 1 0 1

Instrucción

OR (IX-15)  
00h 11011101  
86h 10110110  
F1h 11110001

Valor del registro "A" después de la ejecución

(A):  
1 1 1 1 1 1 1 1

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 0 x 1 0 0

En este ejemplo podemos ver que, si uno de los dos operandos es FFh, el resultado es siempre FFh. Por otro lado, si uno de los dos operandos es

00h, el resultado será igual al otro operando. En la operación AND ocurría exactamente al contrario. Esto se revelará muy útil cuando estudiemos la forma de poner "máscaras" a un octeto para aislar algunos de sus bits.

OR (Y+d)

OBJETO:

Realiza una operación lógica OR, bit a bit, entre el octeto del registro acumulador y el octeto indicado por el operando. La dirección del octeto operando es la que resulta de añadir al contenido del registro índice "IX" el valor del entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación se deja en el registro acumulador.

CODIGO DE MAQUINA:

FDh 1 1 1 1 1 1 0 1  
86h 1 0 1 1 0 1 1 0

INDICADORES DE CONDICION A LOS QUE AFECTA:

S: pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

Z: pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso

H: pone 0 - siempre  
N: pone 0 - siempre  
C: pone 0 - siempre  
P/V: pone 1 - si la paridad es par.

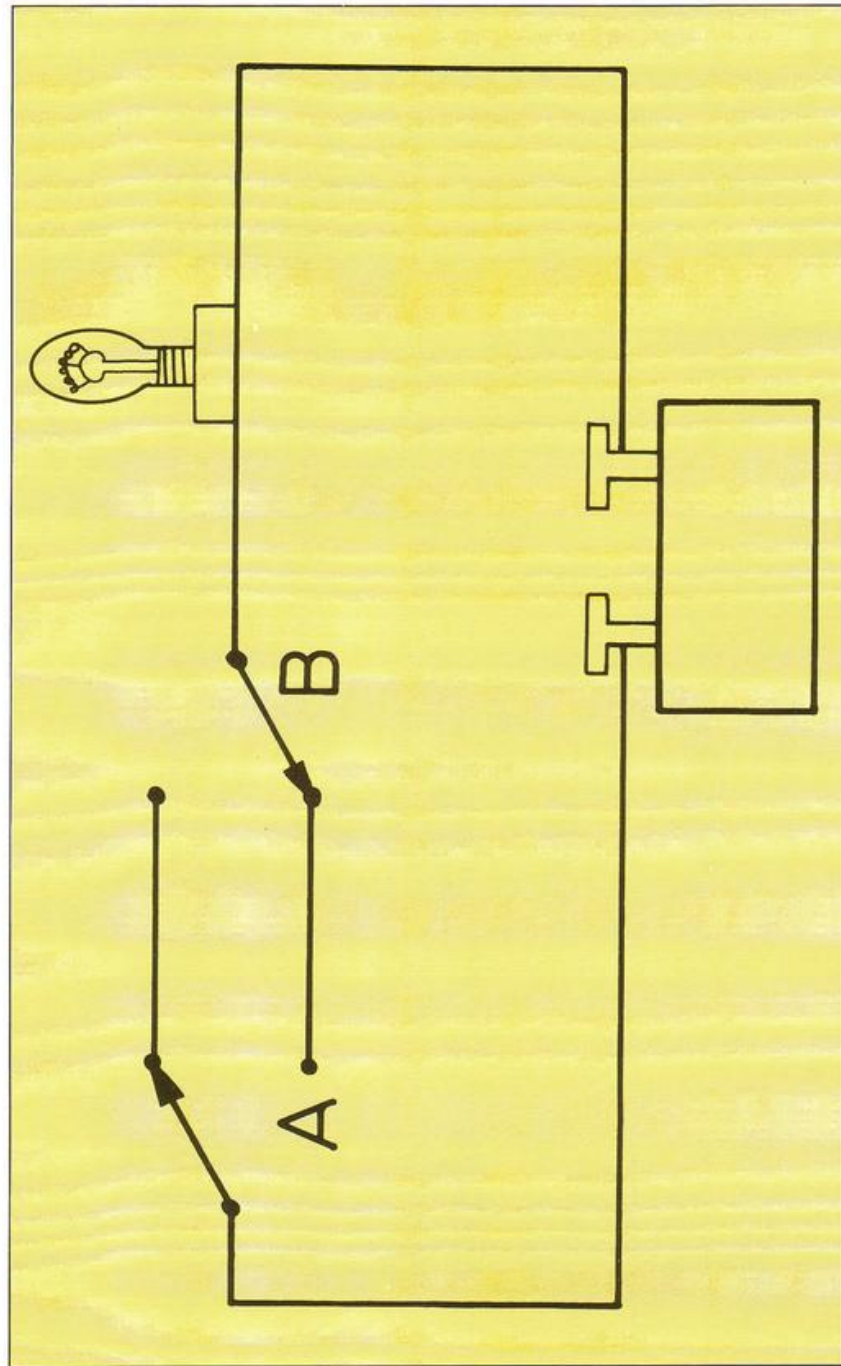


Fig. 6-13. Representación eléctrica de "XOR".

pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

EJEMPLO:

OR (Y+24)

Valor del registro índice "Y"

(Y):  
0 1 1 1 1 1 0 0  
1 0 0 0 0 1 1 0

Valor de la posición de memoria 7C9Eh

(7C9Eh):  
0 0 0 0 0 0 0 0

Valor del registro "A"

0 1 0 1 0 1 0 1

Instrucción

OR (Y+24)  
FDh 1 1 1 1 1 1 0 1  
86h 1 0 1 1 0 1 1 0  
18h 0 0 0 1 1 0 0 0

Valor del registro "A" después de la ejecución

(A):  
0 1 0 1 0 1 0 1

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 0 x 0 x 1 0 0

Vemos que el valor del registro "A" no ha variado, en efecto, es igual hacer "OR" que hacer "OR A", y también es igual hacer "AND" que hacer "AND A".

La activación de los indicadores de condición en las instrucciones OR, se hace según las siguientes reglas:

S: En este indicador se pone el mismo valor que el bit 7 del registro A, después de la ejecución.

Z: Este indicador se activa (valor igual 1) si todos los bits del registro A son cero después de la ejecución.

H: Este indicador no tiene significado para estas instrucciones y se pone siempre a 0.

P/V: Este indicador actúa en función de la paridad. Si el número de bits activos, en el registro acumulador, después de la ejecución es par, el indicador se activa; valor igual 1. En caso contrario se



"HL". El indicador de desbordamiento se ha activado al superarse la cantidad decimal 32767 que es el número positivo máximo que se puede representar en complemento a 2 en dos octetos.

## SBC HL, ss

### OBJETO:

Resta al contenido del par de registros "HL", el contenido del par de registros representados por "ss", más el indicador de acarreo (C) del registro "F". El resultado se deja en el par de registros "HL". La codificación de "ss" es la siguiente:

ss	reg.
00	BC
01	DE
10	HL
11	SP

### CODIGO DE MAQUINA:

1 1 1 0 1 1 0 1	Edh
0 1 ss 0 1 0	

### INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 1 - si no hay acarreo desde el bit 11  
pone 0 - en cualquier otro caso  
N; pone 1 - siempre  
C; pone 1 - si no hay acarreo

reos desde el bit 15 pone 0 - en cualquier otro caso  
P/V; pone 1 - si hay desbordamiento (overflow)  
pone 0 - en cualquier otro caso

### CICLOS DE MEMORIA:

4

### CICLOS DE RELOJ:

15

### EJEMPLO:

SBC HL, DE	
------------	--

Contenido del par de registros "DE":

(D):	1 0 1 1 0 0 1 1	89h
(E):	1 1 0 1 1 0 1 0	DAH

Contenido del par de registros "HL":

(H):	0 1 1 0 0 0 0 0	60h
(L):	1 0 0 0 1 1 1 0	86h

Indicador C=1	
---------------	--

### Instrucción

SBC HL, DE	Edh
	53h

Contenido del par de registros "HL" después de la ejecución

(H):	0 1 1 0 0 0 1 1	63h
(L):	0 1 0 1 0 0 1 1	53h

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	0	x	0

Para entender mejor el funcionamiento de las instrucciones de 16 bits ADC y SBC, en caso de tener alguna dificultad, se puede consultar el funcionamiento de las mismas para 8 bits. Las reglas son las mismas y en lo que se diferencian es en el tamaño de los operandos, además de que las de 16 bits usan como si fuera el acumulador, el par de registros "HL".

Por otro lado, y como se habrá observado ya, no existe la resta sin acarreo en 16 bits, razón por la cual no es necesario especificar el operando de destino en la instrucción "SUB", ya que siempre es "A".

## ADD IX, pp

### OBJETO:

Sumar al contenido del registro índice "IX", el contenido del par de registros representados por "pp". El resultado se deja en el registro índice "IX". La codificación de "pp" es la siguiente:

pp	reg.
00	BC
01	DE
10	IX
11	SP

### CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1	D0h
0 0 pp 1 0 0 1	

### INDICADORES DE CONDICION A LOS QUE AFECTA:

H; pone 1 - si hay acarreo desde el bit 11  
pone 0 - en cualquier

5

### CICLOS DE RELOJ:

19

### EJEMPLO:

XOR (Y-3)	
-----------	--

Valor del registro índice "Y"

(Y):	0 1 1 1 0 0 0 1	71h
	1 0 0 1 0 1 0 1	95h

Valor de la posición de memoria 7192h

(7192h):	1 0 0 1 1 0 1 0	9Ah
----------	-----------------	-----

Valor del registro "A"

	0 1 1 1 0 1 0 1	75h
--	-----------------	-----

### Instrucción

OR (Y-3);	FDh
	AEh
	FDh

Valor del registro "A" después de la ejecución

(A):	1 1 1 0 1 1 1 1	EFh
------	-----------------	-----

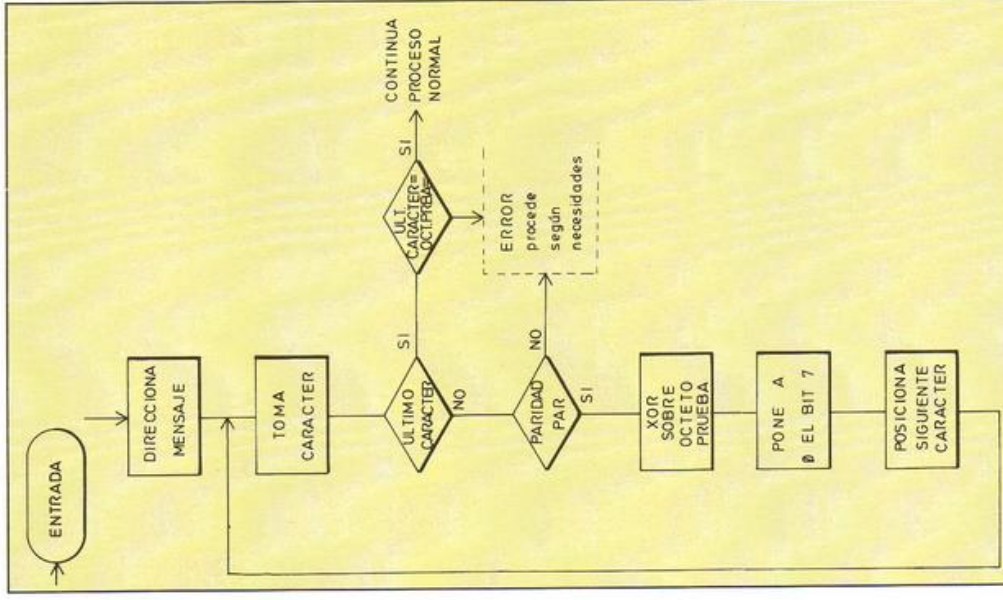
Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
1	0	x	0	x	0

La activación de los indicadores de condición en las instrucciones XOR, se hace según las siguientes reglas:

S: En este indicador se pone el mismo valor que tenga el bit 7 del registro A, después de la ejecución.

Z: Este indicador se activa



### Análisis de controles de paridad con octeto de control.

(valor igual a 1) si todos los bits del registro A son cero después de la ejecución.

H: Este indicador no tiene significado para estas instrucciones y se pone siempre a 0.

P/V: Este indicador actúa en función de la paridad. Si el número de bits activos en el registro acumulador, después de la ejecución es par, el

indicador se activa (valor igual a 1). En caso contrario se mantiene a cero.

N: Este indicador carece de significado para estas instrucciones y se pone siempre a 0.

C: Este indicador carece de significado para estas instrucciones y se pone siempre a 0.



## Máscaras

La utilidad de las instrucciones que permiten sumar y restar es evidente, en definitiva, un ordenador tiene que realizar cálculos y estamos acostumbrados a hacerlo en Basic; pero tal vez más de un lector se pregunte qué finalidad puede tener realizar operaciones lógicas entre octetos. Pues bien, vamos a explicar uno de los usos más frecuentes de los operadores lógicos en Assembler: las máscaras.

Supongamos que tenemos un octeto del que sólo nos interesan los cuatro bits inferiores, pero tras los cálculos que hemos realizado, es posible que los bits superiores contengan "unos" o "ceros" que nos interesa eliminar. En principio, parece que no hay manera de partir el octeto por la mitad, pero tal vez podamos aplicarle una operación lógica que nos elimine los cuatro bits superiores y mantenga inalterados los inferiores; veamos: si hacemos un "AND" de ese octeto con el número binario 00001111 ocurrirá lo siguiente:

```
Octeto:  xxxx1001  (x9h)
Máscara: AND 00001111  (0Fh)
Resultado: 00001001  (09h)
```

Donde hemos puesto (x), significa que puede haber tanto un "1" como un "0". Tenemos un octeto que contenía "X9h" (aquí la "X" significa cualquier número entre 0 y F), le hemos hecho un "AND" con el número "0Fh" y hemos ob-

tenido "09h", es decir, hemos eliminado los bits cuyo contenido no nos interesaba (los marcados con "x") y los hemos puesto todos a "cero". Pero podía habernos interesado ponerlos todos a "uno" y obtener "F9"; ¿por qué no?, vamos a verlo:

```
Octeto:  xxxx1001  (x9h)
Máscara: OR 11110000  (F0h)
Resultado: 11111001  (F9h)
```

Esta vez hemos hecho un "OR" con el número "F0h", con lo cual, los cuatro bits superiores toman valor "1", independientemente del valor que tuvieran antes, y los cuatro bits inferiores permanecen inalterados. Está claro que, conociendo el funcionamiento de los operadores "OR" y "AND", podemos aislar cualquier grupo de bits que nos interesen, y dejar los restantes a "cero" o a "uno".

Veamos otro ejemplo: si tenemos un octeto cuyo contenido es el código ASCII de una letra minúscula, y hacemos "AND #DF", obtenemos el código de esa misma letra en Mayúscula, con la ventaja adicional de que si la letra ya era Mayúscula, su código no habrá variado; veámoslo gráficamente:

```
Letra "a": 01101101  (6Dh)
Máscara: AND 11011111  (DFh)
Letra "A": 01001101  (4Dh)
```

En el caso contrario; podemos tener el código de una Mayúscula y convertirla en minúscula haciendo "OR #20"; vamos a verlo:

```
Letra "A": 01001101  (57h)
Máscara: OR 00100000  (20h)
Letra "a": 01101101  (77h)
```

¿Fácil verdad?, pues todo hay que agradecerlo a lo bien hecho que está el código ASCII, ya que una letra en Mayúscula y esa misma letra en minúsculas sólo se diferencian en que la primera tiene el bit 5 a "cero" y la segunda lo tiene a "uno".

Podemos hacer más cosas, por ejemplo, es posible saber si una letra es minúscula o Mayúscula con sólo hacer "AND #20" y mirar el indicador de cero (Z) del registro "F"; si la letra era Mayúscula, el resultado de la operación será "00h" y, por tanto, el indicador "Z" se habrá puesto a "1"; pero si era minúscula, el indicador permanecerá a "0" ya que el resultado habrá sido "20h".

A estas alturas parece evidente la razón de que llamemos máscara al número con el que operamos nuestro octeto, ya que la operación hace que unos bits "pasen" y otros "se queden" (podíamos haberlo llamado "filtro", pero los ingleses dicen "mask" y, en informática la influencia sajona es inevitable).

Evidentemente, la utilidad de las máscaras no se queda en lo visto hasta aquí, existen un sinfín de aplicaciones

```
00ss1001
```

## INDICADORES DE CONDICION A LOS QUE AFECTA:

H; pone 1 – si no hay acarreo desde el bit 11.  
pone 0 – en cualquier otro caso.  
N; pone 0 – siempre  
C; pone 1 – si hay acarreo desde el bit 15  
pone 0 – en cualquier otro caso

## CICLOS DE MEMORIA:

3

## CICLOS DE RELOJ:

11

## EJEMPLO:

```
ADD HL,BC
Contenido del par de registros "HL"
(H): 10010100 94h
(L): 10001010 8Ah
```

Contenido del par de registros "BC"

```
(B): 10010001 91h
(C): 01001100 4Ch
```

## Instrucción

ADD HL,BC: 00001001 09h

Contenido del par de registros "HL" después de la ejecución

```
(B): 10010001 91h
(C): 01001100 4Ch
(H): 00100101 25h
(L): 11010110 D5h
```

Indicadores de condición después de la ejecución

```
S Z H P/V N C
x x x 0 x x 0 1
```

## ADC HL, ss

## OBJETO:

Sumar al contenido del par de registros "HL", el contenido del par de registros representados por "ss", más el indicador de acarreo (C) del registro "F". El resultado se deja en el par de registros "HL".

La codificación de "ss" es la siguiente:

```
ss reg.
00 BC
01 DE
10 HL
11 SP
```

## CODIGO DE MAQUINA:

```
11101101 EDh
01ss1010
```

## INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 – si el resultado es negativo  
pone 0 – en cualquier otro caso  
Z; pone 1 – si el resultado es cero  
pone 0 – en cualquier otro caso  
H; pone 1 – si hay acarreo desde el bit 11  
pone 0 – en cualquier otro caso

N; pone 0 – siempre  
C; pone 1 – si hay acarreo desde el bit 15  
pone 0 – en cualquier otro caso.  
P/V; pone 1 – si hay desbordamiento (overflow)  
pone 0 – en cualquier otro caso

## CICLOS DE MEMORIA:

4

## CICLOS DE RELOJ:

15

## EJEMPLO:

```
ADC HL,HL
```

Contenido del par de registros "HL"

```
(H): 01101001 69h
(L): 00100110 26h
```

Indicador C=0

## Instrucción

```
ADC HL,HL: 11101101 EDh
01101010 6Ah
```

Contenido del par de registros "HL" después de la ejecución

```
(H): 11010010 D2h
(L): 01001100 4Ch
```

Indicadores de condición después de la ejecución

```
S Z H P/V N C
1 0 x 1 x 1 0 0
```

Observe que esta instrucción equivale a multiplicar por 2 el valor del par de registros



## OBJETO:

Compara (resta) el valor del octeto del registro acumulador con el valor del octeto direccionado por el operando. La dirección del operando se calcula añadiendo al registro índice "IY" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación activará los indicadores de condición según corresponde a la resta.

## CODIGO DE MAQUINA:

1 1 1 1 1 1 0 1	FDh
1 0 1 1 1 1 1 0	BEh
<-----d----->	

## INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero

pone 0 - en cualquier otro caso

H; pone 1 - si no hay acarreo desde el bit 3

pone 0 - en cualquier otro caso

N; pone 1 - siempre

C; pone 0 - si no hay acarreo desde el bit 7.

pone 1 - en cualquier otro caso.

P/V; pone 1 - si hay desbordamiento (overflow)

pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

5

## CICLOS DE RELOJ:

19

## EJEMPLO:

CP (IY+5)

Valor del registro índice "IY"

8Ch	1 0 0 0 1 1 0 0
8Fh	1 0 0 0 1 1 1 1

Valor de la posición de memoria 8C94h

(8C94h): 0 0 0 0 0 0 0 0

Valor del registro "A"

0 0 0 0 0 0 0 1

## Instrucción

FDh	1 1 1 1 1 1 0 1
BEh	1 0 1 1 1 1 1 0
05h	0 0 0 0 0 1 0 1

CP (IY+5):

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	0	x	0

La activación de los indicadores de condición en las instrucciones CP, se hace con las mismas reglas que las SUB y SBC.

La única diferencia está en la interpretación que se puede hacer con ellos.

Z: Si está activo indica que los dos octetos son iguales.

S, P/V y C: Pueden indicar la relación que existe entre los octetos, cual es el mayor o el menor; para ello es necesario conocer el tipo de datos que se maneja, valores absolutos o complemento a dos. El problema es complejo para sacar una norma sencilla y fiable.

donde será necesario su uso, por ejemplo, cuando veamos los programas encargados de detectar y generar paridades, utilizábamos máscaras; cuando veamos la forma de hayar las direcciones de pantalla partiendo de las coordenadas de un carácter, tendremos que realizar operaciones en las que determinados grupos de bits serán tratados de forma independiente tras haber sido aislados con una máscara; y por último, cuando imprimimos en modo "OVER 1", en realidad lo que hacemos es un "XOR" del nuevo dato con el anterior, por eso, si ponemos un pixel donde ya había otro, obtenemos un punto en blanco.

Más adelante, en los ejemplos, utilizaremos las máscaras de un modo práctico; de momento vamos a ver unas cuantas instrucciones más, pertenecientes al grupo aritmético-lógico.

## Grupo de instrucciones de comparación

CP, "Compare" en inglés, se traduce al castellano por comparar. Con este código se define una instrucción que compara el octeto representado por el operando con el registro acumulador.

Esto es cierto en parte, pues lo que realmente hace esta instrucción es restarle al valor del registro acumulador el valor del octeto representado; todo ello sin modificar ninguno de los dos. Lo que modifica esta instrucción son los indicadores de condición en función de dicha resta y con ellos se interpreta el resultado de dicha comparación.

Por ejemplo: si después de comparar (restar) el registro

Código Fuente	Hexadecimal	Decimal
AND A	A7	167
AND B	A0	160
AND C	A1	161
AND D	A2	162
AND E	A3	163
AND H	A4	164
AND L	A5	165
AND n	E6, n	230, n
AND (HL)	A6	166
AND (IX+d)	DD, A6, d	221, 166, d
AND (IY+d)	FD, A6, d	253, 166, d

acumulador con un octeto, se activa el indicador de condición Z, esto es, el resultado de la resta es cero, quiere decir que los dos octetos son iguales.

Por lo tanto una vez sabido que más que una comparación, se trata de una resta en la que no se modifica ninguno de los operandos se pueden dar por válidas las siguientes reglas después de una comparación:

Z = 1 ; A = octeto operando
Z = 0 ; A < > octeto operando
C = 1 ; A < octeto operando
C = 0 ; A >= octeto operando

Básicamente el formato de esta instrucción es:

CP OPERANDO

El octeto indicado por el operando se compara con el octeto del registro acumulador, el resultado activa los indicadores de condición como si se efectuase una resta del registro acumulador menos

el operando, pero sin alterar el dato contenido en el registro acumulador.

CP r

## OBJETO:

Compara (resta) el valor del octeto del registro acumulador con el valor del octeto del registro representado por "r". El resultado de esta operación activa los indicadores de condición como corresponde a una resta.

## CODIGO DE MAQUINA:

1 0 1 1 1 <-----r----->

## INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado



es cero  
pone 0 – en cualquier otro caso

H; pone 1 – si no hay acarreo desde el bit 3  
pone 0 – en cualquier otro caso

N; pone 1 – siempre  
C; pone 1 – si no hay acarreo desde el bit 7.  
pone 0 – en cualquier otro caso.

P/V; pone 1 – si hay desbordamiento (overflow)  
pone 0 – en cualquier otro caso

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

CP H

Valor del registro "A"

(A): 0 1 1 0 1 1 0 1 60h

Valor del registro "H"

(H): 0 1 1 0 1 1 0 1 60h

Instrucción

CP H: 1 0 1 1 1 1 0 0 8Ch

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 1 x 0 x 0 1 0

Observe que al ser iguales los dos octetos el resultado de la resta es cero, por lo tanto se activa el indicador Z.

CP n

OBJETO:

Compara (resta) el valor del octeto del registro acumulador con el octeto de valor "n". El resultado de esta operación activa los indicadores de condición según corresponden a la resta.

CODIGO DE MAQUINA:

FEh  
1 1 1 1 1 1 1 0  
<-----n----->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 – si el resultado es negativo  
pone 0 – en cualquier otro caso

Z; pone 1 – si el resultado es cero  
pone 0 – en cualquier otro caso

H; pone 1 – si no hay acarreo desde el bit 3  
pone 0 – en cualquier otro caso

N; pone 1 – siempre  
C; pone 1 – si no hay acarreo desde el bit 7.  
pone 0 – en cualquier otro caso.

P/V; pone 1 – si hay desbordamiento (overflow)  
pone 0 – en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

CP 25

Valor del registro "A"

(A): 0 0 0 0 1 0 1 0 0Ah

Instrucción

CP 25: 1 1 1 1 1 1 1 0 FEh  
0 0 0 1 1 0 0 1 19h

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 0 x 0 1 1

Observe que del análisis de los indicadores de condición se saca la conclusión de que "A" es menor que "n", dado que el "acarreo" se ha puesto a "1".

CP (HL)

OBJETO:

Compara (resta) el valor del octeto del registro acumulador con el valor del octeto de memoria direccionado por el contenido del par de registros "HL". El resultado de la operación activará los indicadores de condición como corresponde a la resta.

CODIGO DE MAQUINA:

1 0 1 1 1 1 1 0 8Ch

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 – si el resultado es negativo  
pone 0 – en cualquier otro caso

Z; pone 1 – si el resultado es cero  
pone 0 – en cualquier otro caso

H; pone 1 – si no hay acarreo desde el bit 3  
pone 0 – en cualquier otro caso

rreo desde el bit 3  
pone 0 – en cualquier otro caso

N; pone 1 – siempre  
C; pone 1 – si no hay acarreo desde el bit 7.  
pone 0 – en cualquier otro caso.

P/V; pone 1 – si hay desbordamiento (overflow)  
pone 0 – en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

7

EJEMPLO:

CP (HL)

Valor del par de registros "HL"

(H): 0 1 1 1 1 1 0 0 7Ch  
(L): 1 0 1 1 1 0 0 0 88h

Valor de la posición de memoria 7CB8h

(7CB8h): 0 0 1 0 0 1 0 0 24h

Valor del registro "A"

(A): 0 1 1 0 1 1 0 1 60h

Instrucción

CP (HL): 1 0 1 1 1 1 1 0 BEh

Indicadores de condición después de la ejecución

S Z H P/V N C  
0 0 x 0 x 0 1 0

Del análisis de los indicadores se puede sacar la con-

clusión de que "A" es mayor que el octeto operando, ya que no se han activado ni el "Z" (cero) ni el "C" (acarreo).

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

19

OBJETO:

Compara (resta) el valor del octeto del registro acumulador con el valor del octeto direccionado por el operando. La dirección del operando se calcula añadiendo al registro índice "IX" el entero de desplazamiento "d", el cual puede adquirir los valores desde -128 a +127. El resultado de la operación activará los indicadores de condición según corresponde a la resta.

CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1 0Dh  
1 0 1 1 1 1 1 0 BEh  
<-----d----->

INDICADORES DE CONDICION A LOS QUE AFECTA:

S; pone 1 – si el resultado es negativo  
pone 0 – en cualquier otro caso

Z; pone 1 – si el resultado es cero  
pone 0 – en cualquier otro caso

H; pone 1 – si no hay acarreo desde el bit 3  
pone 0 – en cualquier otro caso

N; pone 1 – siempre  
C; pone 0 – si no hay acarreo desde el bit 7.  
pone 1 – en cualquier otro caso.

Indicadores de condición después de la ejecución

S Z H P/V N C  
1 0 x 1 x 0 1 1

De nuevo, es posible ver que el operando era mayor que el resultado.

CP (IX+d)



Un número de cuatro dígitos necesitaría dos octetos, etc. Ejemplo, el número 3427 se definiría:

00110100	34h
00100111	27h

Un número con la cantidad de dígitos impar, justificaría el octeto más significativo a la derecha, por ejemplo 753 se-  
ría:

00000111	07h
01010011	53h

Como se puede ver los números decimales coinciden con los dígitos hexadecimales. Un octeto con valores en BCD nunca puede tener números hexadecimales entre A y F.

Para operar con octetos en este código se utilizan las instrucciones normales de suma y resta para registros de 8 bits y después de cada suma o resta es necesario ajustar el resultado a BCD. Por ejemplo al sumar 27 más 48 ocurriría lo siguiente:

$$\begin{array}{r} 27 \rightarrow \text{BCD } 00100111 \\ 48 \rightarrow \text{BCD } 01001000 \\ \hline \end{array}$$

la suma con ADD de estos dos octetos sería:

$$\begin{array}{r} 00100111 \\ +01001000 \\ \hline 01101111 \end{array}$$

resultado:

01101111	6fh
----------	-----

si a este resultado se le suma 06h, quedaría:

$$\begin{array}{r} 01101111 \\ +00000110 \\ \hline 01110101 \end{array}$$

resultado:

01110101	75h
----------	-----

Que vuelve a estar en código BCD; efectivamente 27+48=75. Al sumar 06h al resultado, lo hemos transformado, de nuevo, en BCD.

Con una resta ocurriría lo mismo: por ejemplo 31 menos 15:

$$\begin{array}{r} 31 \rightarrow \text{BCD } 00110001 \\ 15 \rightarrow \text{BCD } 00010101 \end{array}$$

al restar con la instrucción SUB (recuerde que es una suma con el sustraendo complementado a 2) sería:

$$\begin{array}{r} 00110001 \\ +11101011 \\ \hline 00011000 \end{array}$$

resultado:

00011000	1Ch
----------	-----

si a este resultado se le suma FAh, sería:

$$\begin{array}{r} 00011000 \\ +11110110 \\ \hline 00010110 \end{array}$$

resultado:

01101111	6fh
----------	-----

resultando:

00010110	16h
----------	-----

como se ve, vuelve a estar en BCD pues efectivamente 31-15=16. Esta vez hemos tenido que sumar FAh para pasar a BCD.

Todo este procedimiento parece complicado, pero existe una instrucción, DAA (Decimal Adjust in "A"), que añade la cantidad adecuada en cada octeto. Esta instrucción, que se explica a continuación, es la encargada de ajustar a BCD el octeto después de cada operación; con lo cual la única responsabilidad del programador es tener los operandos en BCD y aplicar la instrucción DAA después de cada operación, con esto obtendrá los resultados en BCD.

De todas formas, es muy improbable que tenga que utilizar esta instrucción en alguno de sus programas ya que para operar en decimal es más fácil utilizar el calculador de la ROM (lo veremos más adelante). Esta instrucción está pensada para pequeños sistemas construidos alrededor del Z-80, por ejemplo, para aplicaciones de control en tiempo real, donde no se dispone de un sofisticado Sistema Operativo que gestione las entradas y salidas en decimal.

La instrucción DAA utiliza como información inicial los indicadores en condición C, H y N.

El indicador C informa que existe acarreo en el octeto. Siempre que se presente, además de condicionar la cantidad que tiene que añadir, lo mantiene activo para poder añadirlo al octeto siguiente.

OBJETO:

Sumar al contenido del registro índice "IY", el contenido del par de registros representados por "rr". El resultado se deja en el registro índice "IY". Observe que decimos "rr" en lugar de "pp" porque "pp" incluye al registro "IX" y "rr" incluye al "IY", por tanto, es posible multiplicar por dos el contenido de cualquiera de los dos registros índices (ADD IX, IX o ADD IY, IY) pero no es posible sumarlos los dos (ADD IX, IY o ADD IY, IX). La codificación de "rr" es la siguiente:

rr	reg.
00	BC
01	DE
10	IY
11	SP

CODIGO DE MAQUINA:

1111101	FDh
00rr1001	

INDICADORES DE CONDICION A LOS QUE AFECTA:

H: pone 1 - si hay acarreo desde el bit 11  
pone 0 - en cualquier otro caso

N: pone 0 - siempre  
C: pone 1 - si hay acarreo desde el bit 15  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

15

EJEMPLO:

ADD IY, rr

otro caso

N: pone 0 - siempre  
C: pone 1 - si hay acarreo desde el bit 15  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

15

EJEMPLO:

ADD IX, SP
------------

Contenido del registro "SP"

00010000	10h
01101001	69h

Contenido del registro índice "IX"

01100110	66h
01101011	75h

Instrucción

1101101	DDh
00111001	39h

Contenido del registro índice "IX" después de la ejecución

01110110	76h
11011110	DEh

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
x	x	x	0	x	0

ADD IY, rr

ADD IY, IY

Contenido del registro índice "IY"

00000000	00h
00000010	02h

Instrucción

1111101	FDh
00101001	29h

Contenido del registro índice "IY" después de la ejecución

00000000	00h
00000100	04h

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
x	x	x	0	x	0

El resultado de este ejemplo ha sido el de multiplicar por dos el contenido del registro índice "IY" (lo hemos sumado consigo mismo).

La activación de los indicadores de condición en las instrucciones de suma y resta de 16 bits, se hace según las siguientes reglas:

S: Este indicador sólo se contempla en las instrucciones de sumar y restar con acarreo y se pone el mismo valor que tenga el bit 15 del par de registros "HL" después de la ejecución.

Z: Este indicador sólo se contempla en las instrucciones de sumar y restar con acarreo. Se activa, valor igual 1, si todos los bits del par de registros "HL" son cero después de la ejecución.



H: Este indicador actúa igual que para las instrucciones de sumar y restar de 8 bits, con la diferencia de que el acarreo o no acarreo se define en el bit 11 del registro doble donde se deja el resultado.

P/V: Este indicador actúa igual que en las instrucciones de suma y resta de 8 bits, con la diferencia de que el rango -128, 0, +127 es -32768, 0, +32767. Estos límites son los valores mínimos o máximos que se pueden representar en complemento a 2 en dos octetos.

N: Este indicador no tiene significado para este grupo de instrucciones y se pone a 0 o 1, según sea suma o resta.

C: Este indicador actúa igual que en las instrucciones de sumar y restar de 8 bits, con la diferencia de que el acarreo o no acarreo se define en el bit 15 del registro doble donde se deja el resultado.

### Grupo de Incremento y Decremento para 16 bits

Las instrucciones INC y DEC de 16 bits son básicamente iguales a las INC y DEC de 8 bits, las únicas diferencias son: que trabajan sobre registros dobles de 16 bits y que no afectan a los indicadores de condición.

**INC ss**

OBJETO:

Añade uno al contenido del registro doble representado por "ss". La codificación de "ss" es la siguiente:

Código Fuente	Hexadecimal	Decimal
OR A	B7	183
OR B	B0	176
OR C	B1	177
OR D	B2	178
OR E	B3	179
OR H	B4	180
OR L	B5	181
OR n	F6, n	246, n
OR (HL)	B6	182
OR (IX+d)	DD, B6, d	221, 182, d
OR (IY+d)	FD, B6, d	253, 182, d

Fig. 6-14. Tabla de codificación para el operador «OR».

Código Fuente	Hexadecimal	Decimal
XOR A	AF	175
XOR B	A8	168
XOR C	A9	169
XOR D	AA	170
XOR E	AB	171
XOR H	AC	172
XOR L	AD	173
XOR n	EE, n	238, n
XOR (HL)	AE	174
XOR (IX+d)	DD, AE, d	221, 174, d
XOR (IY+d)	FD, AE, d	253, 174, d

Fig. 6-15. Tabla de codificación para el operador «XOR».

ss	reg
00	BC
01	DE
10	HL
11	SP

CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

1

CODIGO DE MAQUINA:

00 ss 00 11

INDICADORES DE

CICLOS DE RELOJ:

6

EJEMPLO:

después de la ejecución

S	Z	H	P/V	N	C
1	0	x	1	x	0

CCF: "Complement Carry Flag"; en inglés "Complementar el indicador de acarreo"

**CCF**

OBJETO:

Invierte el valor del indicador de condición (C) en el registro "F"; es decir, pasa a valer "0" si antes valía "1" y viceversa.

CODIGO DE MAQUINA:

00 11 11 11

INDICADORES DE

CONDICION QUE AFECTA:

H: mantiene su valor anterior

N: pone 0 - siempre

C: pone 1 - si C era cero antes de la ejecución  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

CCF

Valor del registro "F"

S Z H P/V N C

0 0 x 1 x 0 0 1

Instrucción

CCF: 00 11 11 11

Valor del registro "F" después de la ejecución

S Z H P/V N C

0 0 x 1 x 0 0 0

No resulta alterado el contenido del acumulador ni de ningún otro registro.

SCF: "Set Carry Flag"; en inglés: "Poner a "1" el indicador de acarreo".

**SCF**

OBJETO:

Pone a 1 el valor del indicador de condición (C) del registro "F" independientemente del valor que tuviera antes.

CODIGO DE MAQUINA:

00 11 01 11

INDICADORES DE

CONDICION QUE AFECTA:

H: pone 0 - siempre

N: pone 0 - siempre

C: pone 1 - siempre

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

SCF

Valor del registro "F"

S Z H P/V N C

0 0 x 1 x 0 0 0

Instrucción

Instrucción

SCF: 00 11 01 11

Valor del registro "F" después de la ejecución

S Z H P/V N C

0 0 x 1 x 0 0 1

Si el indicador de acarreo fuera "1" antes de la ejecución, la instrucción, lógicamente, no lo modificaría.

Tal vez, el lector haya echado en falta una instrucción que ponga a "0" el indicador de acarreo. El caso es que esa instrucción no existe, simplemente, porque no es necesaria. La secuencia "SCF CCF" nos asegura que el indicador (C) acabe valiendo "0", pero ocupa dos bytes; hay una forma más fácil de poner el acarreo a "0", simplemente haga: "AND A". El contenido del acumulador no variará, pero el indicador de acarreo se pondrá, con toda certeza, a cero.

El micro procesador Z80, debido posiblemente a sus características de origen, tiene prevista una forma de operar con las instrucciones aritméticas en decimal.

Para ello emplea el código BCD (Binary Coded Decimal), código decimal expresado en binario. Los operandos, en este caso, tienen que estar definidos en dicho código.

El código BCD consiste en expresar cada dígito decimal en cuatro bits, de tal forma que en cada octeto entren dos dígitos.

Por ejemplo el número 19 se expresaría:

00 01 10 01



00101111 2fh

INDICADORES DE  
CONDICION QUE AFECTA:

H; pone 1 - siempre  
N; pone 1 - siempre

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

CPL 00101111

Contenido del registro "A"

(A): 10011011 98h

Instrucción

CPL: 00101111 2fh

Contenido del registro "A"  
después de la ejecución

01100100 64h

Indicadores de condición  
después de la ejecución

S Z H P/V N C  
x x x 1 x x 1 x

NEG, NEGate, negar en in-  
gls. Por negar un número se  
entiende complementarlo a 2;  
el complemento a 2 es el  
complemento a 1 + 1 y se utili-  
za como su negativo, por tan-  
to, esta instrucción lo que ha-  
ce realmente es cambiar de  
signo el contenido del acu-  
mulador (ver capítulo de sis-  
temas de numeración).

NEG

OBJETO:

Complementa a dos el con-  
tenido del registro acumula-  
dor, dejando en el mismo el  
resultado.

CODIGO DE MAQUINA:

11101101 4fh  
01000100 44h

INDICADORES DE  
CONDICION QUE AFECTA:

S; pone 1 - si el resultado  
es negativo  
pone 0 - en cualquier  
otro caso

Z; pone 1 - si el resultado  
es cero  
pone 0 - en cualquier  
otro caso

H; pone 1 - si no hay aca-  
reos desde el bit 3  
pone 0 - en cualquier  
otro caso

N; pone 1 - siempre  
C; pone 1 - si el acumula-  
dor era 00h antes de la  
operación  
pone 0 - en cualquier  
otro caso

P/V; pone 1 - si el acumula-  
dor era 80 antes de la  
operación  
pone 0 - en cualquier  
otro caso

Observe que la forma de  
activarse los indicadores de  
condición recuerda a las ins-  
trucciones de restar; esto es  
porque lo que realmente hace  
esta instrucción es restar a 0  
el contenido del acumulador  
(0-A). Recuerde que las ins-  
trucciones de restar lo prime-  
ro que hacen es complement-  
tar a dos el sustraendo y des-  
pués sumarle el minuendo;  
como el minuendo es siempre  
cero, el resultado es siempre  
el sustraendo complementa-  
do a 2. Como se puede dedu-

cir, a pesar de lo dicho, el indi-  
cador H siempre será 1. Este  
funcionamiento nos indica  
que el microprocesador utili-  
za para "negar" la misma cir-  
cuitría que para "restar"; es-  
to ocurre con muchas otras  
instrucciones y es muy lógico  
dado que, en un microproce-  
sador, lo que se pretende es  
meter el mayor número de  
funciones en el menor espa-  
cio posible.

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

8

EJEMPLO:

NEG

Contenido del registro "A"

(A): 01100101 65h

Instrucción

NEG: 11101101 4fh  
01000100 44h

Operación:

complemento a 1 = 10011010  
+ 00000001  
complemento a 2 = 10011011  
+ 00000000  
resultado = 10011011

Contenido del registro "A"  
después de la ejecución

10011011 98h

Indicadores de condición

INC HL

Contenido del par de regis-  
tros "HL"

(H): 01100100 64h  
(L): 11111111 Ffh

Instrucción

INC HL: 00100011 23h

Contenido del par de regis-  
tros "HL" después de la eje-  
cución

(H): 01100101 65h  
(L): 00000000 00h

El octeto alto de "HL" se ha  
incrementado en 1, dado que  
al incrementar el octeto bajo,  
éste ha pasado a valer "0".  
Vemos, por tanto, que el aca-  
reos se transmite de forma au-  
tomática desde el octeto bajo  
al alto.

INC IX

OBJETO:

Añade uno al contenido de  
registro índice "IX".

CODIGO DE MAQUINA:

11011101 DDh  
00100011 23h

INDICADORES DE  
CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

10

EJEMPLO:

INC IX

Contenido del registro indi-  
ce "IX"

(IX): 11111111 Ffh  
11111111 Ffh

Instrucción

INC IX: 11011101 DDh  
00100011 23h

Contenido del registro indi-  
ce "IX" después de la ejecu-  
ción

(IX): 00000000 00h  
00000000 00h

En este caso, el registro  
"IX" valía "FFFFh" antes de la  
instrucción, lo que hace que  
al sumarle "1", pase a valer  
"0"; no hay indicador de aca-  
reos que nos indique esta cir-  
cunstancia, pero ya veremos  
que no es necesario, ya que  
nunca tendremos que iterar  
un bucle o mover un puntero  
más de 65536 veces.

INC IX

OBJETO:

Añade uno al contenido del  
registro índice "IX".

CODIGO DE MAQUINA:

11111101 Fdh  
00100011 23h

INDICADORES DE  
CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

10

EJEMPLO:

INC IX

Contenido del registro indi-  
ce "IX"

(IX): 00000000 00h  
00000000 00h

Instrucción

INC IX: 11111101 Fdh  
00100011 23h

Contenido del registro indi-  
ce "IX" después de la ejecu-  
ción

(IX): 00000000 00h  
00000000 01h

Las instrucciones de de-  
crementar funcionan igual  
que las anteriores, salvo que  
restan "1" en vez de sumarlo.  
Tampoco afectan a los indi-  
cadores.

DEC ss

OBJETO:

Decrementa uno al conte-  
nido del registro doble repre-  
sentado por "ss". La codifica-  
ción de "ss" es la siguiente:



SS	regj
00	BC
01	DE
10	HL
11	SP

CODIGO DE MAQUINA:

00 0 1 0 1 1

INDICADORES DE  
CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

6

EJEMPLO:

DEC DE

Contenido del par de registros "DE"

(D):	0 1 1 0 0 1 1 0	66h
(E):	0 0 0 0 0 0 0 0	00h

Instrucción

DEC DE: 0 0 0 1 1 0 1 1 18h

Contenido del par de registros "DE" después de la ejecución

(D):	0 1 1 0 0 1 1 0	65h
(E):	1 1 1 1 1 1 1 1	Ffh

Vemos, de nuevo, que el acarreo se ha vuelto a transmitir al octeto alto desde el bajo. Este valia cero, y el decrementarlo ha hecho que pasase a valer "FFh", por lo que el alto ha sido decrementado también.

**DEC IX**

OBJETO:

Decrementa uno al contenido del registro indice "IX".

CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1	00h
0 0 1 0 1 0 1 1	2Bh

INDICADORES DE  
CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

10

EJEMPLO:

DEC IX

Contenido del registro indice "IX"

(IX):	0 0 0 0 0 0 0 0	00h
	0 0 0 0 0 0 0 0	00h

Instrucción

DEC IX:	1 1 0 1 1 1 0 1	00h
	0 0 1 0 1 0 1 1	2Bh

Contenido del registro indice "IX" después de la ejecución

(IX):	1 1 1 1 1 1 1 1	Ffh
	1 1 1 1 1 1 1 1	Ffh

Vemos que el registro "IX" ha pasado de valer "0" a valer "FFFFh"; de nuevo, no hay indicador que ponga de mani-

fiesto esta circunstancia pero, como dijimos antes, no es necesario.

**DEC IY**

OBJETO:

Decrementa uno al contenido del registro indice "IY".

CODIGO DE MAQUINA:

1 1 1 1 1 1 0 1	FDh
0 0 1 0 1 0 1 1	2Bh

INDICADORES DE  
CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

10

EJEMPLO:

DEC IY

Contenido del registro indice "IY"

(IY):	0 0 0 0 0 0 0 0	00h
	0 0 0 0 0 0 0 1	01h

Instrucción

DEC IY:	1 1 1 1 1 1 0 1	FDh
	0 0 1 0 1 0 1 1	2Bh

Contenido del registro indice "IY" después de la ejecución

(IY):	0 0 0 0 0 0 0 0	00h
	0 0 0 0 0 0 0 0	00h

Esta vez, el registro "IY" vale cero después de la ejecución y no se activa el indicador de "cero" (Z). Si estuviéramos haciendo un bucle de más de 256 iteraciones, y utilizáramos un determinado registro como contador, sería útil poder comprobar cuándo este registro se hace "cero" para salir del bucle; la instrucción DEC en registros de 16 bits no afecta al indicador de "cero", pero es posible hacer un pequeño truco: suponemos que estamos usando el registro "BC" como contador de nuestro bucle, podemos saber si hemos llegado a "cero" mediante la siguiente operación:

LD A, B  
OR C

Es decir, cargamos en "A" el contenido de "B" y le hacemos un "OR" con "C". Si el contenido de "BC" era "cero", el resultado de esta operación será también "cero", y se pondrá a "1" el indicador (Z) del registro "F".

Los usos más importantes de las instrucciones INC y DEC para registros de 16 bits son recorrer una tabla o zona de memoria desde un comienzo a un final o viceversa y establecer bucles con más de 256 iteraciones (los bucles se verán detenidamente en el siguiente capítulo).

Normalmente el rastreo de zonas de memoria se hace con registros indice o con el par de registros "HL". Un ejemplo muy sencillo sería mover una tabla de x octetos

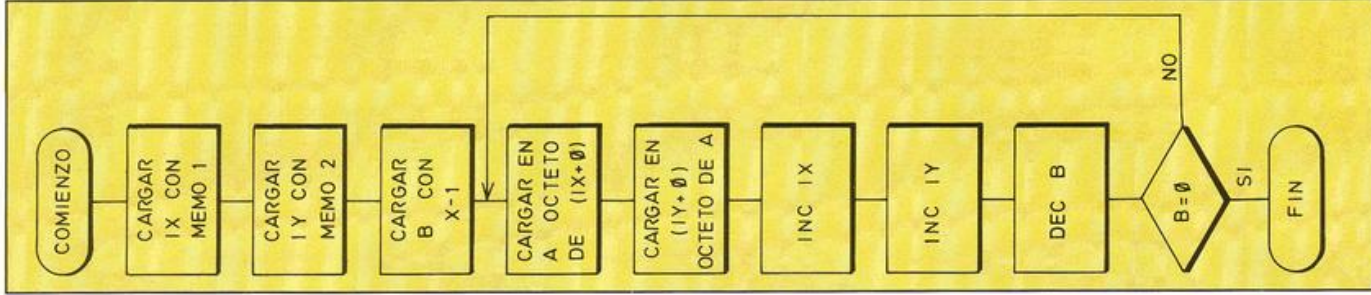


Fig. 6-16.

## Grupo de instrucciones aritméticas de uso general

En este grupo se encuadran unas instrucciones especiales que complementan a las anteriores, y que iremos viendo una a una, ya que guardan poca relación entre sí.

CPL, Complement, complemento en inglés. Por complementarlo a 1 se entiende invertir el valor de un número binario, más sencillo, cambiar todos los unos por ceros y los ceros por unos.

**CPL**

OBJETO:

Complementa a uno el contenido del registro acumulador, esto es, invierte el valor de todos sus bits.

CODIGO DE MAQUINA:



Longitud sin atributos=6144,  
Longitud con atributos=6912. Si desea trabajar sobre el fichero de atributos, las direcciones son: Inicio=22528, Longitud=768.

Si desea enmascarar textos generados con el "EDITOR", las direcciones de inicio de cada página puede encontrarlas en la página 8 del número 13 de MICROHOBBY; la longitud de cada página es de 1408 bytes. Si desea hacerlo con textos generados en un procesador TASSWORD o similar (NEWTEXT, CONTEXT, etc.) la dirección de inicio del texto está almacenada en las direcciones 62216 y 62217; la longitud del mismo es devuelta en la variable "a" cuando se retorna a Basic desde el editor en C/M.

A estas alturas, podemos dar por concluido este capítulo; en el próximo, trataremos sobre los saltos y bucles; a partir de ese momento, el lector deberá ser capaz de empezar a escribir sus propios programas.

Antes de eso, le recomendamos que resuelva los siguientes ejercicios para comprobar si tiene suficientemente afianzados los conocimientos.

NEMONICO	INDICADORES					No. DE BYTES	CICLOS MEM. REL.
	S	Z	H	X	P/V	C	
ADD HL,ss	.	.	x	x	.	1	3 11
ADD IX,pp	.	.	x	x	.	2	4 15
ADD IY,rr	.	.	x	x	.	2	4 15
ADC HL,ss	+	+	x	x	V	2	4 15
SBC HL,ss	+	+	x	x	V	2	4 15
INC ss	.	.	x	.	.	1	1 6
INC IX	.	.	x	.	.	2	2 10
INC IY	.	.	x	.	.	2	2 10
DEC ss	.	.	x	.	.	1	1 6
DEC IX	.	.	x	.	.	2	2 10
DEC IY	.	.	x	.	.	2	2 10

#### NOTAS:

1.- Las letras "ss" indican cualesquiera de los registros "BC", "DE", "HL" ó "SP"; las letras "pp", cualesquiera de "BC", "DE", "IX" ó "SP", y las "rr", cualesquiera de "BC", "DE", "IY" ó "SP".

2.- Los signos tienen el siguiente significado:  
"+": El indicador cambia de valor de acuerdo con el resultado de la instrucción.

"x": El bit adquiere un estado indeterminado.

".": El indicador conserva su anterior contenido.

"V": P/V actúa como indicador de rebasamiento.

"0": El indicador se pone siempre a cero.

"1": El indicador se pone siempre a uno.

Fig. 6-23. Tabla resumida de indicadores y ciclos para las instrucciones aritméticas y lógicas de 16 bits.

El indicador H informa que existe acarreo en el dígito de orden inferior, por lo tanto condiciona la cantidad que tiene que añadir.

El indicador N informa, según sea 1 ó 0, que la instrucción anterior fue una suma o una resta.

### DAA

#### OBJETO:

Ajusta el registro acumulador a BCD después de las instrucciones de suma o resta con operandos en ese código. Las operaciones previas posibles con ADD, ADC e INC como sumas y SUB, SBC, DEC y NEG como restas. La FIGURA 6-17 indica la operación que se realiza.

La interpretación de esta tabla se realiza de la siguiente manera:

Primero analiza el indicador de condición N, el cual señalará si se ha realizado anteriormente una suma o una resta; en segundo lugar analizará el indicador C y el valor de los cuatro bits superiores según unos rangos y por último el indicador H y el valor de los cuatro bits inferiores. Todo ello determinará el valor hexadecimal que se sumará al octeto y si se activa o no el indicador de acarreo C.

#### CODIGO DE MAQUINA:

00100111 27h

#### INDICADORES DE CONDICION QUE AFECTA:

S: pone 1 - si el bit 7 del registro A es 1 después de la ejecución

OPERACION	C ANTES DAA	VALOR HEXA-DECIMAL DEL DIGITO SUPERIOR bits 7-4	H ANTES DAA	VALOR HEXA-DECIMAL DEL DIGITO INFERIOR bits 3-0	NUMERO HEXA-DECIMAL QUE ARADE AL OCTETO	C DESPUES DAA
N=0	0	0-9	0	0-9	00	0
(ADD, ADC ó INC)	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
N=1	0	0-9	0	0-9	00	0
(SUB, SBC, DEC ó NEG)	0	0-8	1	6-F	FA	0
	1	7-F	0	0-9	A0	1
	1	6-F	1	6-F	9A	1

Fig. 6-17. Tabla de condiciones para la instrucción "DAA".

pone 0 - en cualquier otro caso

Z: pone 1 - si el registro A es 0 después de la ejecución

pone 0 - en cualquier otro caso

C: pone 0 ó 1 - según se indica en la FIGURA 6-17

P/V: pone 1 - si el registro A tiene paridad par después de la ejecución  
pone 0 - en cualquier otro caso

#### CICLOS DE MEMORIA:

1

01001000  
+00111000  
10000000

Operación anterior 48+38 en BCD

DAA

CICLOS DE RELOJ:

4

EJEMPLO:



Contenido del registro "A"

(A): 1 0 0 0 0 0 0 0 80h

Indicadores de condición de la operación anterior

S Z H P/V N C  
1 0 x 1 x 1 0 0

Instrucción

DAA: 0 0 1 0 0 1 1 1 27h

Condición según tercera línea de la figura 6-17 añade 06h al registro "A".

Contenido del registro "A" después de la ejecución

(A): 1 0 0 0 0 1 1 0 86h

Indicadores de condición después de la ejecución.

S Z H P/V N C  
1 0 x 0 x 0 0 0

No se preocupe si no ha entendido perfectamente el funcionamiento de esta instrucción, es muy posible que no la utilice ni una sola vez en su vida.

Con lo visto hasta aquí, terminamos la parte teórica de este capítulo dedicado a las instrucciones aritméticas y lógicas. En el capítulo siguiente veremos las instrucciones que nos permiten romper la secuencia del programa y saltar a cualquier posición, así como la forma de crear bucles en código máquina (equivalentes a las instrucciones "GO TO" y "FOR...NEXT" del Basic).

Antes de eso, vamos a ver las tablas de codificación y algunos ejemplos que aclaran lo estudiado hasta ahora.

```
5 REM PROGRAMA 3
*** OPERA Y COMPARA 2 NUMEROS ***
*** ** ** ** **
10 FOR n=23300 TO 23329
20 READ a: POKE n,a: NEXT n
30 DATA 237,91,118,92,123,162,
0,79,123,178,21,237,67,0,91,123,17,
7,1,237,123,166,245,209,123,230,65,
100 INPUT "Primer numero? ";a
110 INPUT "Segundo numero? ";b
120 POKE 23670,a: POKE 23671,b
130 RANDOMIZE USR 23300
200 CLS:PRINT "a: ",a AND "b: "
=210 PEEK 23296: OR "b: " = "PEE
K 23297
220 PRINT "a: " XOR "b: " = "PE
EK 23298
230 GO TO 250+PEEK 23299
250 PRINT "a: ",a > "b: GO TO 100
251 PRINT "a: " < "b: GO TO 100
314 PRINT "a: " = "b: GO TO 100
```

#### PROGRAMA 4

```
5 REM PROGRAMA 4
*** XOR de un bloque de memoria ***
*** ** ** ** **
10 FOR n=23301 TO 23321
20 READ a: POKE n,a: NEXT n
30 DATA 42,0,91,237,75,2,91,23,
177,32,247,201
70 INPUT "INICIO? ";INIC
100 INPUT "LONGITUD? ";LONG
120 INPUT "CLAVE? ";CLAVE
130 POKE 23300,CLAVE
140 POKE 23299,INT (LONG/255)
150 POKE 23298,INT (LONG/255)
299 POKE 23297,INT (INIC/255)
170 POKE 23296,INT (INIC/255)+PEEK 23
297
180 RANDOMIZE USR 23301
```

En la FIGURA 6-18 se encuentra la tabla de codificación para las instrucciones de comparación (CP). En la FIGURA 6-19 para las aritméticas de 16 bits (ADD, ADC y SBC). En la FIGURA 6-20 para las instrucciones de incremento y decremento en registros de 16 bits (INC y DEC). Finalmente, la FIGURA 6-21 contiene la tabla correspondiente a las instrucciones aritméticas de uso general (CPL, NEG, CCF, SCF y DAA).

te, luego comprobaremos resultados...

A nosotros nos ha quedado así:

```
50 42,0,91
60 237,75,2,91
70 237,91,4,91
80 126
90 171
100 119
110 35
120 11
130 120
140 177
>>> 150 32,247 <<<
160 201
```

Hemos señalado la línea que contiene la instrucción no vista hasta ahora; esperamos que, a pesar de todo, no haya tenido problemas.

Vamos a ver el programa Basic que sirve para hacer funcionar esta rutina. Su listado es el del PROGRAMA 4. Las tres primeras líneas (10, 20 y 30) sirven, como siempre, para introducir el código en memoria. Las líneas 100, 110, 120 nos piden los datos que las 130 a 170 se encargan de introducir en las variables correspondientes. Finalmente, la línea 180 llama a la rutina.

El "Inicio" y la "Longitud" pueden estar comprendidos entre "0" y "65535", mientras que la "Clave" debe ser un número comprendido entre "0" y "255".

Si desea comprobar, de una forma rápida, el funcionamiento de la rutina, puede hacerla trabajar sobre la pantalla. En este caso, las direcciones serán: Inicio=16384,

NEMONICO	INDICADORES										No. DE BYTES	CICLOS MEM. REL.
	S	Z	X	H	P/V	N	C					
ADD A,r	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	1	4
ADD A,n	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	2	7
ADD A,(HL)	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	1	7
ADD A,(IX+d)	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	3	19
ADD A,(IY+d)	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	3	19
ADC A,s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		
SUB s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		
SBC A,s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		
AND s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		
OR s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		
XOR s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		
CP s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		
INC r	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	1	4
INC (HL)	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	1	11
INC (IX+d)	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	3	23
INC (IY+d)	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	3	23
DEC s	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		

#### NOTAS:

- 1.- La letra "s" indica cualesquiera de "r", "n", "(HL)", "(IX+d)" o "(IY+d)", salvo en "DEC" que no vale "n". Los bytes y ciclos, en estos casos, son los mismos que para el grupo inmediato anterior; por ejemplo: "AND (HL)" tiene 1 byte, 2 ciclos de memoria y 7 de reloj, exactamente igual que "ADD A,(HL)".
- 2.- Los signos tienen el siguiente significado:
  - "r": El indicador conserva su anterior contenido.
  - "n": El indicador conserva su anterior contenido.
  - "v": P/V actúa como indicador de rebasamiento.
  - "p": P/V actúa como indicador de paridad.
  - "0": El indicador se pone siempre a cero.
  - "1": El indicador se pone siempre a uno.

Fig. 6-22. Tabla resumida de indicadores y ciclos para las instrucciones aritméticas y lógicas de 8 bits.



Se trata de una rutina que "barre" todo un bloque de memoria haciendo, en todos los bytes, un XOR con un número previamente introducido por el usuario. La enorme utilidad de esta rutina viene dada porque constituye una forma eficaz de enmascarar la información almacenada en memoria y protegerla de miradas indiscretas.

Efectivamente, si hacemos XOR con un determinado número a todo un bloque de memoria, la información almacenada quedará irreconocible; no obstante, bastará volver a hacer XOR con ese mismo número para recuperar la información inicial. Dado que sólo nosotros sabemos el número (o clave) utilizado la primera vez, sólo nosotros podremos desenmascarar la información y hacer uso de ella. Un "pirata" que intentara leerla, tendría que intentar desenmascararla con 256 números diferentes. Puede parecer poco, pero también es posible enmascarar la información 2 veces, primero con el número "a" y después con el "b"; para desenmascararla, sería necesario hacerlo primero con el "b" y luego con el "a". En este caso, el "pirata" se encuentra ante una clave compuesta por 65536 combinaciones posibles; lo más probable es que desista ¿no cree?

El método es tan eficaz que se usa con frecuencia en la protección de programas comerciales. Usted puede usarlo para enmascarar sus programas en código máquina, textos generados en un "procesador de textos" o, simplemente, pantallas; ya que la rutina se puede aplicar en cualquier lugar de la memoria. No le aconsejamos, no obstante,

que intente enmascarar con ella un programa en Basic, ya que el "cuelgue" del ordenador sería prácticamente seguro.

Luego indicaremos las direcciones a utilizar para enmascarar la pantalla o los bloques de texto, de momento, veamos cómo funciona la rutina. Su listado Assembler es el siguiente:

```
10 INIC EQU 23296
20 LONG EQU 23298
30 CLAVE EQU 23300
40 ORG 23301
50 LD HL, (INIC)
60 LD BC, (LONG)
70 LD DE, (CLAVE)
80 BUCLE LD A, (HL)
90 XOR E
100 LD (HL), A
110 INC BC
120 DEC BC
130 LD A, B
140 OR C
150 JR NZ, BUCLE
160 RET
```

Vamos a explicarlo por partes: Las líneas 10, 20 y 30 constituyen la definición de variables. La 40 indica la dirección donde se debe enmascarar (en el buffer de impresora, para que pueda actuar sobre toda la memoria). La línea 50 inicializa el valor del puntero "HL" que se carga con la dirección de inicio e irá apuntando, uno por uno, a todos los bytes de la zona a enmascarar. La línea 60 carga el contador de bytes "BC" con la longitud de la zona de memoria a enmascarar; este registro actuará como control del

bucle, que se iterará "BC" veces (decrementando "BC" en cada pasada, hasta que llegue a valer cero). En la línea 70, cargamos la clave en el registro "E"; la instrucción "LD E, (nn)" no existe, de modo que hemos tenido que utilizar la que carga el par "DE"; no obstante, el contenido de la dirección 23300 se cargará en "E", y en "D" se cargará el contenido de la siguiente (la primera del programa), pero este registro no le tendremos que utilizar para nada.

En la línea 80, entramos de lleno en el bucle; para indicarlo, está la etiqueta; en esta línea, se carga en "A" el contenido de la dirección apuntada por el puntero "HL". Este contenido es "XORreado" en la línea 90 y se restablece a su posición de memoria correspondiente en la línea 100. Las líneas 110 y 120 se encargan de INCREMENTAR el puntero y DECREMENTAR el contador, y las líneas 130 y 140 comprueban si este ha llegado a cero (recuerde que "DEC BC" no afecta a los indicadores, por lo que es necesario comprobarlo así). La línea 150 constituye un salto relativo condicional; se estu- diará en el capítulo próximo; de momento, bastenos saber que equivale a:

```
IF A<>0 THEN GO TO BUCLE
```

La línea 160 (la que nunca hay que olvidar) nos permite devolver el control al Basic. Una vez visto cómo funciona la rutina, vamos a ensamblarla; la instrucción "JR NZ, BUCLE" se ensambla como "32,247"; el resto tiene que saber hacerlo usted. Adelante

Código Fuente	Hexadecimal	Decimal
CP A	BF	191
CP B	B8	184
CP C	B9	185
CP D	BA	186
CP E	BB	187
CP H	BC	188
CP L	BD	189
CP n	FE, n	254, n
CP (HL)	BE	190
CP (IX+d)	DD, BE, d	221, 190, d
CP (IY+d)	FD, BE, d	253, 190, d

Fig. 6-18. Tabla de codificación para instrucciones de comparación.

Código Fuente	Hexadecimal	Decimal
INC BC	03	3
INC DE	13	19
INC HL	23	35
INC SP	33	51
INC IX	DD, 23	221, 35
INC IY	FD, 23	253, 35
DEC BC	0B	11
DEC DE	1B	27
DEC HL	2B	43
DEC SP	3B	59
DEC IX	DD, 2B	221, 43
DEC IY	FD, 2B	253, 43

Fig. 6-20. Tabla de codificación para instrucciones de incremento y decremento en 16 bits.

Código Fuente	Hexadecimal	Decimal
ADD HL, BC	09	9
ADD HL, DE	19	25
ADD HL, HL	29	41
ADD HL, SP	39	57
ADD IX, BC	DD, 09	221, 9
ADD IX, DE	DD, 19	221, 25
ADD IX, IX	DD, 29	221, 41
ADD IX, SP	DD, 39	221, 57
ADD IY, BC	FD, 09	253, 9
ADD IY, DE	FD, 19	253, 25
ADD IY, IX	FD, 29	253, 41
ADD IY, SP	FD, 39	253, 57
ADC HL, BC	ED, 0A	237, 10
ADC HL, DE	ED, 1A	237, 26
ADC HL, HL	ED, 2A	237, 42
ADC HL, SP	ED, 3A	237, 58
SBC HL, BC	ED, 42	237, 66
SBC HL, DE	ED, 52	237, 82
SBC HL, HL	ED, 62	237, 98
SBC HL, SP	ED, 72	237, 114

Fig. 6-19. Tabla de codificación para instrucciones aritméticas de 16 bits.

Código Fuente	Hexadecimal	Decimal
CPL	2F	47
NEG	ED, 44	237, 68
CCF	3F	63
SCF	37	55
DAA	27	39

Fig. 6-21. Tabla de codificación para instrucciones aritméticas de uso general.



## Ejemplos

Ha llegado ya el momento de volver a conectar nuestro querido Spectrum y empezar a aplicar en la práctica los conocimientos adquiridos. Insistimos en la importancia de que el lector no se limite a copiar los ejemplos; es imprescindible que tome una actitud participativa. Intente ensambalar los programas por sí mismo, y luego compare su resultado con el nuestro; intente, también, comprender el funcionamiento de cada programa y átrévase a realizar en ellos, sus propias modificaciones. ¿Qué pasaría si...?; no se lo pregunte, ¡hágalo!, y si se le "cuelga" el ordenador, paciencia y vuelta a intentar.

En esta ocasión, hemos preparado dos rutinas de ejemplo, la primera es muy ilustrativa pero de escasa aplicación práctica, simplemente, trate de comprender lo mejor posible su funcionamiento; la segunda, por el contrario, es de gran utilidad y seguro que la usará en sus programas.

Vamos, pues, con el primero de los ejemplos. Se trata de una rutina que recibe, como entrada, dos números de un byte (comprendidos entre 0 y 255), los opera con AND, OR y XOR dándonos el resultado de las tres operaciones, finalmente, los compara con "CP" para indicarnos cuál de los dos es mayor o si son iguales.

La rutina es muy corta y, por ello, la hemos colocado en el buffer de impresora, de esta forma sirve tanto para usuarios de 16K, como de 48K. Los dos números de entrada se deberán encontrar en las direcciones 23670 y

23671 respectivamente (que corresponden a la variable del sistema SEED), el programa en Basic que manejará esta rutina, se encargará de introducirlos en estas direcciones. La rutina nos devolverá los resultados de AND, OR y XOR en las direcciones 23296, 23297 y 23298 respectivamente y en la 23299 nos devolverá un número que dependerá del resultado de la comparación. Vamos a ver la rutina:

10	ENTR	EGU	23670
20	SAL1	EGU	23296
30	SAL2	EGU	23298
40		ORG	23300
50		LD	DE, (ENTR)
60		LD	A,E
70		AND	D
80		LD	C,A
90		LD	A,E
100		OR	D
110		LD	B,A
120		LD	(SAL1),BC
130		LD	A,E
140		XOR	D
150		LD	C,A
160		LD	A,E
170		CP	D
180		PUSH	AF
190		POP	DE
200		LD	A,E
210		AND	#41
220		LD	B,A
230		LD	(SAL2),BC
240		RET	

Las líneas 10, 20 y 30 constituyen la definición de variables, el pseudo-nemónico EQLU sirve para asignar a la etiqueta el valor numérico correspondiente. La línea 40 indica al ensamblador que deberá colocar el código objeto

a partir de la dirección 23300, el pseudo-nemónico ORG es la abreviatura de "ORiGen". La línea 50 carga en "DE" los dos números que vamos a operar, el primero de ellos en "E" y el segundo en "D". Las líneas 60, 70 y 80 los operan con AND y guardan el resultado en el registro "C". Las líneas 90, 100 y 110 los operan con OR y guardan el resultado en el registro "B". La línea 120 almacena ambos resultados en la variable "SAL1" (2 bytes). Las líneas 130, 140 y 150 operan los dos números con XOR y guardan el resultado en "C" desde donde será transferido a la variable "SAL2" en la línea 230. Las líneas 160 y 170 comparan los dos números. En las líneas 180, 190 y 200 se hace un pequeño truco para transferir el registro "F" al "A". En la línea 210 se pone una máscara para aislar los dos bits que nos interesan ("Z" y "C"); veamos esto gráficamente:

	S	I	x	H	x	P/V	N	C
(H):	x	x	x	x	x	x	x	x
AND:	0	1	0	0	0	0	0	1
=:	0	x	0	0	0	0	0	x

Como se ve, hemos aislado los indicadores de "cero" y "acarreo", dejando todos los bits restantes a "0". La línea 220 transfiere este resultado a "B", la 230 lo guarda en "SAL2" junto con el resultado de XOR y, finalmente, la línea 240 (la más importante) se encarga de devolver el control al Basic.

En la comparación pueden ocurrir tres cosas: que el primer número sea mayor que el segundo, que sea menor, o que ambos sean iguales; vamos a ver cómo quedan los

indicadores en cada caso y  
qué resultado nos devuelve la  
rutina en la dirección 23299:

CONDICION	"Z" "C"	Hex. Dec.	RESULTADO
Pria. > Seg.	0 0	00 0	
Pria. < Seg.	0 1	01 1	
Pria. = Seg.	1 0	40 64	

Si el primero es mayor que el segundo, obtendremos "0", si es menor obtendremos "1", y si ambos son iguales el resultado será "64" en decimal, es decir, "40h" en hexadecimal.

Vamos a ensamblar la rutina, no mire el listado que haya a continuación, e intente hacerlo por usted mismo...

Ahora, compruebe el resultado:

Evidentemente, las líneas 10, 20, 30 y 40 no hay que ensamblarlas va que no gene-

CONDICION	"Z" "C"	Hex. Dec.	RESULTADO
Pria. > Seg.	0 0	00 0	
Pria. < Seg.	0 1	01 1	
Pria. = Seg.	1 0	40 64	

ran código objeto; por lo demás, hemos mantenido la misma numeración que en el código fuente.

La única dificultad puede estar en las líneas 50, 120 y 230; donde dice "LD DE, (ENTR)" es como si dijera "LD DE, (23670)" (para eso está la línea 10) o, si lo prefiere en Hexa: "LD DE, (#5C76)". Lo mismo se puede decir respecto a "LD (SAL1),BC" y "LD (SAL2),BC" que equivalen, respectivamente, a: "LD (#5B00),BC" y "LD (#5B02),BC".

Ahora que tenemos ensablada nuestra rutina, no queda más que diseñar el programa en Basic que la haga funcionar. El PROGRAMA 3 cumple este cometido a la perfección, aunque es posible que cada lector prefiera escribir el suyo propio más adecuado a sus fines; vamos a comentarlo.

Las líneas 10, 20 y 30 se encargan, como de costumbre, de introducir en memoria el código máquina que se encuentra en los DATAS de la línea 30. Las líneas 100 y 110 nos piden los dos números a operar, que la línea 120 se encarga de introducir en la variable del sistema SEED; esta línea podría haber sido:

128 RANDOMIZE a+256#b

Pero no funcionaría si los dos números a operar fueran "cero" ya que, en ese caso, copiaría en SEED el contenido de los dos octetos menos significativos del contador de cuadros (variable del sistema FRAMES). La línea 130 llama a la rutina y las restantes se encargan de presentarnos en pantalla los resultados. En la línea 230 se ha usado un "artificio" que consiste en ar-

del contenido de la dirección 23299 que, como dijimos antes, puede valer "0", "1" o "64" ( $250 + 64 = 314$ ); por este sistema, se ahorran tres sentencias del tipo "IF...THEN". Finalmente, el programa retorna a la línea 100 para una nueva entrada, aunque puede ser desde cualquier dirección.

que además de "hacer algo" utilizan, exclusivamente, instrucciones vistas hasta el momento no ha sido tarea fácil; el principal problema se ha debido a no poder crear bucles. Para crear un bucle es necesario romper la secuencia del programa, pero aún no hemos visto las instrucciones que alteran esta secuencia; por otro lado, cualquier aplicación en código máquina se hace mediante bucles. Afortunadamente, en el siguiente capítulo, veremos las instrucciones de cambio de secuencia (saltos), así como las condicionales que se comportan de forma distinta en función del estado de los indicadores.

Con el fin de satisfacer la lógica impaciencia del lector, nos hemos anticipado un poco, y hemos preparado un ejemplo que si hace uso de una de estas instrucciones. En compensación, esta vez el ejemplo si vale para algo útil.



JR e

OBJETO:

Salta a la posición de memoria que resulta de añadir a la propia posición de la instrucción el entero de desplazamiento, el cual puede adquirir los valores desde -126 a +129.

CODIGO DE MAQUINA:

00011000	18h
<---e-2-->	

INDICADORES DE

CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

3

CICLOS DE RELOJ:

12

EJEMPLO:

En la dirección absoluta 45F6h

JR +129
---------

Queremos saltar a 4677h (18039d) y estamos en 45F6h (17910) por tanto, 18039-17910=129.

Contenido del registro PC al leer la instrucción la CPU

01000101	45h
11111000	F8h

El registro se ha incrementado dos veces, ahora contiene 45F8h (17912d) así que el valor que hay que sumar es 7Fh (127d) es decir, "e-2".

Instrucción

JR +129:	00011000	18h
	01111111	7Fh

Observe, de nuevo, que hemos ensamblado 7Fh (127d) es decir, "e-2".

Contenido del registro PC después de la ejecución

(PC):	01000110	46h
	01111011	77h

La siguiente instrucción a ejecutar será la de la posición de memoria 4677h; observe que esta dirección es igual a 45F8h + 7Fh o, si lo prefiere, 18039 = 17912 + 127 (127 = 129-2, es decir, e-2). Observe también que hemos tenido que ensamblar el valor 127 en lugar de 129, ya que el código objeto debe llevar el valor "e-2"; esto se debe a que, como decíamos antes, el desplazamiento se suma cuando el "PC" se ha incrementado dos veces para apuntar a la siguiente instrucción.

Si estuviéramos trabajando con un Ensamblador, hubiéramos puesto una etiqueta delante de la instrucción a donde queríamos saltar, por ejemplo, "LB1" y luego, habríamos hecho, simplemente:

JR LB1
--------

El Ensamblador se hubiera encargado de calcular el desplazamiento que habría que ensamblar en el código máquina. Todas estas consideraciones en cuanto al valor del desplazamiento son igualmente válidas en todos los ejemplos que siguen sobre saltos relativos.

JR C,e

OBJETO:

Si el indicador de acarreo "C" está activo; salta a la posición de memoria que resulta de añadir a la propia posición de la instrucción el entero de desplazamiento "e", el cual puede adquirir los valores desde -126 a +129.

Si el indicador de acarreo "C" no está activo; ejecuta la siguiente instrucción.

Las instrucciones de saltos condicionales son equivalentes a la sentencia Basic:

IF ... THEN GO TO ...

CODIGO DE MAQUINA:

00111000	38h
<---e-2-->	

INDICADORES DE  
CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

Si cumple la condición, 3  
Si no cumple la condición, 2

CICLOS DE RELOJ:

Si cumple la condición 12  
Si no cumple la condición 7

EJEMPLO:

En la dirección absoluta 8323h

JR C,+10
----------

GRUPO ARITMETICO-LOGICO DE USO ESPECIAL (Indicadores y ciclos)

NEMONICO	INDICADORES										No. DE		CICLOS	
	S	Z	X	H	X	P/V	N	C			BYTES		MEM.	REL.
DAA	0	0	x	0	x	0	0	0	0	0	1	1	1	4
CPL	.	.	x	1	x	.	1	.	.	.	1	1	1	4
NEG	0	0	x	0	x	0	0	0	0	0	2	2	2	8
CCF	.	.	x	x	x	.	0	0	0	0	1	1	1	4
SCF	.	.	x	0	x	.	0	0	0	0	1	1	1	4

NOTAS:

1.- Los signos tienen el siguiente significado:

"0": El indicador cambia de valor de acuerdo con el resultado de la instrucción.

"x": El bit adquiere un estado indeterminado.

".": El indicador conserva su anterior contenido.

"v": P/V actúa como indicador de reboseamiento.

"p": P/V actúa como indicador de paridad.

"0": El indicador se pone siempre a cero.

"1": El indicador se pone siempre a uno.

Fig. 6-24. Tabla resumida de indicadores y ciclos para las instrucciones aritméticas especiales.

EJERCICIOS

1.- ¿Que máscara deberíamos poner para identificar si el número, que contiene un octeto determinado, es par o impar?

2.- Complete la siguiente rutina, de forma que barra una zona de memoria cambiando en Mayúsculas todas las letras minúsculas que se encuentre:



```

10 LD HL, (INIC)
20 LD BC, (LONG)
30 BUCLE
40 .....
50 .....
60 INC HL
70 DEC BC
80 .....
90 .....
100 JR NZ, BUCLE
110 RET

```

3.- Igual que el ejercicio anterior, pero cambiando en minúsculas todas la Mayúsculas.

4.- ¿Que resultado obtendremos en el acumulador al final de esta rutina?:

```

10 LD A, #77
20 AND #59
30 OR #33
40 XOR #15
50 RET

```

5.- Queremos entrar en una tabla "indexada" donde utilizaremos un código, contenido en el acumulador, como "offset"; cada elemento de la tabla tiene 2 bytes de longitud; la dirección base de la tabla es 5F35h y tenemos que obtener el dato de salida en el par de registros "BC". Para ello, deberemos multiplicar por dos el contenido del acumulador (sumarlo consigo mismo) y añadirlo a la dirección base, luego, cargaremos el dato de la tabla en "BC". Complete la rutina que se encarga de hacerlo:

el indicador "P/V" está a "1" ("PE" = Parity Even, Paridad par).

Suponemos que el indicador de condición "P/V" está a "1".

Contenido del registro PC al leer la instrucción la CPU

(PC):	1 0 0 0 0 1 0 1	85h
	0 0 1 0 0 0 1 1	23h

Instrucción

JP PE, #E042:	EAh
	11101010
	01000010
	42h
	11100000
	E0h

Contenido del registro PC después de la ejecución

(PC):	1 1 1 0 0 0 0 0	E0h
	0 1 0 0 0 0 1 0	42h

La siguiente instrucción a ejecutar será la de la posición de memoria E042h. Se ha verificado el salto ya que la condición se cumplía.

EJEMPLO:

En la dirección absoluta A0A0h

JP P, #AA00
-------------

Saltar si el indicador de signo señala "positivo" ("S"=0). (Jump on Positive, Saltar si positivo).

Suponemos que el indicador de condición "S" es igual a "1".

Contenido del registro PC al leer la instrucción la CPU

(PC):	1 0 1 0 0 0 0 0	A0h
	1 0 1 0 0 0 1 1	A3h

Instrucción

JP P. #AA00:	11110010	F2h
	00000000	00h

Contenido del registro PC después de la ejecución

(PC):	1 0 1 0 0 0 0 0	A0h
	1 0 1 0 0 0 1 1	A3h

La siguiente instrucción a ejecutar será la de la posición de memoria A0A3h. No se ha verificado el salto por no cumplirse la condición.

## Instrucciones de salto relativo

JR: "Jump Relative", salto relativo en inglés. Provocan un salto a una posición de memoria próxima a aquella desde donde se salta; la dirección del operando no viene dada en forma absoluta, sino mediante un valor de "desplazamiento" que se suma al contenido actual del "PC"; este valor puede ser positivo o negativo, lo que permite saltos hacia delante o hacia atrás y, lo que es más importante, al no figurar en la instrucción un valor absoluto, esta funciona de igual forma independientemente de la posición de memoria donde esté colocada; dicho en otras palabras, las rutinas que utilicen saltos relativos son REUBICABLES, es decir, pueden correr en cualquier zona de la memoria.

Otra importante ventaja de los saltos relativos es que ocupan sólo 2 bytes frente a los 3 que ocupa un salto absoluto.

soluta. El inconveniente fundamental es que los saltos han de ser "cortos", ya que el punto de destino debe estar muy próximo a aquel desde donde se salta. Su utilidad fundamental reside en la creación de bucles, como veremos más adelante.

Lo más engorroso de utilizar saltos relativos es calcular el valor de desplazamiento que hay que sumar al "PC" para que el salto se dirija al lugar correcto de la memoria; tengase en cuenta que el desplazamiento se suma al "PC" cuando éste ya se ha incrementado dos veces para apuntar a la siguiente instrucción. Cuando se trabaja con un Ensamblador, se suele poner una etiqueta en el punto de destino del salto, luego se hace el salto relativo a la etiqueta y el Ensamblador se encarga de la desagradable tarea de calcular el desplazamiento. No obstante, desde un principio prometimos que el curso se podría seguir sin Ensamblador, así que estudiaremos con detalle la forma de calcular saltos relativos.

Cuando se salta hacia delante no hay problema, la dificultad consiste en calcular un salto hacia atrás, ya que en este caso, el desplazamiento es negativo, es decir, complemento a 2; precisamente para esto, nos va a servir muy bien la figura 2 del capítulo relativo a los sistemas de numeración (páginas 12, 13 y 14 del curso); la segunda columna de esta figura indica el valor absoluto que se corresponde con un determinado valor negativo en complemento a 2. Veamos primero las instrucciones de salto relativo y luego volveremos sobre esto más detenidamente.



o bien, con el uso de una etiqueta:

```
ETIQUETA EQU #9063
JP ETIQUETA
```

Contenido del registro "PC" inmediatamente después de que la CPU haya leído la instrucción:

(PC):	01110011	73h
	11110101	F5h

(El "PC" se ha incrementado tres veces tras leer la instrucción).

Instrucción

	11000011	C3h
	01100011	63h
	10010000	90h

JP #9063.

Contenido del registro PC después de la ejecución

(PC):	10010000	90h
	01100011	63h

La siguiente instrucción a ejecutar será leída desde la posición de memoria 9063h. Se habrá producido, por tanto, un salto en el programa.

JP cc,nn

OBJETO:

Salta a la posición de memoria indicada por "nn" si la condición "cc" es verdad; en caso contrario continúa con la siguiente instrucción.

Las condiciones "cc" son las que se indican en la tabla de la figura 7-1.

CODIGO DE MAQUINA:

11 <--cc--> 010	LSB
<--0-->	MSB

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

3

CICLOS DE RELOJ:

10

EJEMPLO:

En la dirección absoluta 8436h.

JP NZ, LABEL

La dirección de LABEL es 4723h.

El indicador de condición "Z" es igual a 0.

Contenido del registro PC al leer la instrucción la CPU:

(PC):	10000100	84h
	00111001	39h

Instrucción

	11000010	C2h
	00100011	23h
	01000111	47h

JP NZ, LABEL.

Contenido del registro PC después de la ejecución

(PC):	01000111	47h
	00100011	23h

La siguiente instrucción a ejecutar será la de la posición de memoria 4723h, es decir, la apuntada por "LABEL".

EJEMPLO:

En la dirección absoluta 4728h.

JP C, #4576

Saltar a 4576h si el indicador de acarreo está a "1", en caso contrario, continuar con la secuencia normal.

Suponemos que el indicador de condición "C" está a "0".

Contenido del registro PC al leer la instrucción la CPU

(PC):	01000111	47h
	00100101	28h

Instrucción

	11011010	D4h
	01110110	76h
	01000101	45h

JP C, #4576.

Contenido del registro PC después de la ejecución

(PC):	01000111	47h
	00100011	28h

La siguiente instrucción a ejecutar será la de la posición de memoria 4728h, es decir, no se ha producido el salto debido a que no se cumplía la condición.

EJEMPLO:

En la dirección absoluta 8520h.

JP PE, #E042

Saltar a la posición E042 si

## SOLUCIONES A LOS EJERCICIOS

1.- "AND #01" o bien: "AND 1". Si el número era par, el indicador "Z" se pondrá a "1".

2.- La rutina queda:

```
10 LD HL, (INIC)
20 LD BC, (LONG)
30 BUCLE LD A, (HL)
40 AND #DF
50 LD (HL), A
60 INC HL
70 DEC BC
80 LD A, B
90 OR C
100 JR NZ, BUCLE
110 RET
```

3.- En este caso, habría que cambiar la línea 40 por:

```
40 OR #20
```

Permaneciendo, lo restante, exactamente igual.



#### 4.- La operación sería la siguiente:

```

Valor inicial 01110111 77h
AND 01011001 59h
= 01010001 51h
OR 00110011 33h
= 01110011 73h
XOR 00010101 15h
= 01100110 66h
    
```

El resultado es, por tanto, 66h.

#### 5.- La rutina quedaría:

```

10 LD BC, #5F35
20 LD H, 0
30 LD L, A
40 ADD HL, HL
50 ADD HL, BC
60 LD C, (HL)
70 INC HL
80 LD B, (HL)
90 RET
    
```

Es de suma importancia que intente comprender el funcionamiento de esta rutina, ya que se trata de un procedimiento, para moverse por tablas, que se utiliza con mucha frecuencia.

----- 0 -----

## INSTRUCCIONES DE CAMBIO DE SECUENCIA

Estas instrucciones son también conocidas como: "de salto".

Recordemos que la secuencia del programa se va marcando por el contenido del registro de 16 bits "PC" (Program Counter) contador de programa. La CPU lee siempre la instrucción que está en la dirección de memoria apuntada por el registro PC y añade a éste la cantidad de octetos que tiene dicha instrucción; por lo tanto el registro PC, una vez leída la instrucción por la CPU, queda apuntando a la siguiente. Si en una instrucción se cargara el registro PC con un valor distinto, la siguiente instrucción a ejecutar no sería la que le sigue secuencialmente. Pero las instrucciones que cargan el registro "PC" no existen dentro del grupo de instrucciones de carga, son de tanta importancia que se les ha reservado un grupo propio, denominado "Grupo de instrucciones de cambio de secuencia". Veámoslas una por una:

Jump, "saltar" en inglés, es el nombre genérico de estas instrucciones. A pesar de que siempre que nos referimos a ellas diremos que saltan a tal posición, lo que realmente hacen es cargar el registro "PC" con la dirección de memoria de esa posición. El utilizar el verbo "saltar" se debe a que es más descriptivo de la operación que se realiza, que

cambiar de secuencia o cargar el registro "PC".

En el micro-procesado Z80 hay tres tipos de instrucciones de salto: ABSOLUTO, RELATIVO e INDIRECTO. Las de salto absoluto van a la posición de memoria marcada por el operando de la propia instrucción, bien sea mediante una etiqueta o un valor numérico. Las de salto relativo calculan la posición de memoria sumando a su propia dirección un entero de desplazamiento que puede adquirir valores comprendidos entre -126 y +129. Por último, las de salto indirecto toman la posición de memoria, a donde han de saltar, de un registro de 16 bits.

Las instrucciones de salto absoluto y relativo se pueden dividir en dos grupos: INCONDICIONALES y CONDICIONALES. Las incondicionales saltan siempre a la posición indicada; mientras que la condicionadas saltan si es cierta una condición que se les pone. Precisamente para estas últimas instrucciones se han activado los indicadores de condición en el registro "F", ahí es donde se verifica si la condición señalada es cierta o no.

#### Instrucciones de salto absoluto

JP: (Jump), en inglés, "saltar". Simplemente, provocan

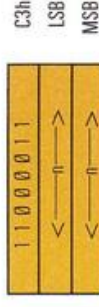
un salto absoluto a la dirección de memoria indicada por el operando.



#### OBJETO:

Salta a la posición de memoria indicada por "nn".

#### CODIGO DE MAQUINA:



#### INDICADORES DE CONDICION QUE AFECTA:

Ninguno

#### CICLOS DE MEMORIA:

3

#### CICLOS DE RELOJ:

10

#### EJEMPLO:

En la dirección absoluta 73F2h.

JP #9063

Observe que esta instrucción podría haber sido:

JP 36963



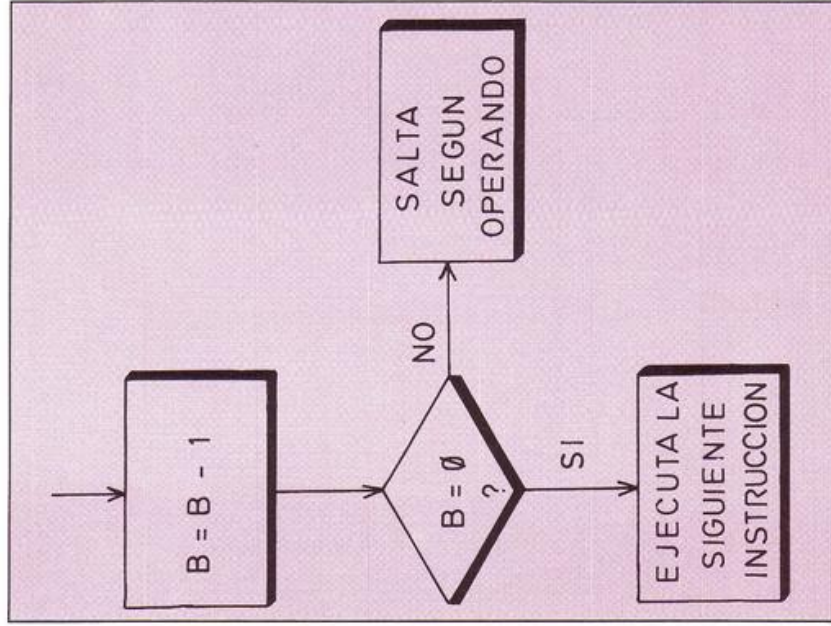


Fig. 7-2. Organigrama de la instrucción "DJNZe"

¿Qué ocurre cuando el multiplicador es igual a "0"?; dado el funcionamiento de "DJNZ", multiplicaríamos por 256. Esto provocaría un resultado erróneo, de forma que tendríamos que detectar cuando el multiplicador sea "0" para actuar en consecuencia.

Por otro lado, es necesario tomar los valores iniciales de algún sitio y almacenarlos en nuestra rutina en el buffer de impresora (aunque es totalmente reubicable) y utilizaremos las primeras direcciones como variables donde almacenar los datos de entrada y

el resultado; "VARI" en una variable de 2 octetos en las direcciones 23296 y 23297, en la entrada contiene el multiplicando y en la salida, los dos octetos menos significativos del resultado; "VAR2" es una variable de 1 octeto en la dirección 23298, en la entrada contiene el multiplicador y en la salida, el octeto más significativo del resultado. Dejamos otro byte libre y colocamos la rutina a partir de 23300 para que empiece en un número redondo que es más fácil de recordar. Hemos llamado a la rutina "MULTI" y su listado completo es:

130	MULTI	LD	HL, VAR2
140		LD	B, (HL)
150		LD	DE, (VAR1)
160		DEC	B
170		INC	B
180		JR	Z, CERO
190		XOR	A
200		LD	HL, 0
210	BUCLE	ADD	HL, DE
220		JR	C, CARRY
230	LOOP	DJNZ	BUCLE
240		LD	(VAR1), HL
250		LD	(VAR2), A
260		RET	
270	CARRY	INC	A
280		JR	LOOP
290	CERO	LD	HL, 0
300		LD	(VAR1), HL
310		RET	

Las líneas 130 y 140 cargan en "B" el contenido de "VAR2"; en 150 se carga en "DE" el contenido de "VAR1"; las líneas 160, 170 y 180 comprueban si "B" es "0", en cuyo caso, se salta a la línea 290. En 190 y 200 se carga "0" en "A" y "HL"; 210, 220 y 230 constituyen el bucle principal; si se produce acarreo, se incrementa "A" en 270 y 280. Finalmente, las líneas 240 y 250 cargan en "VAR1" y "VAR2" el resultado antes de retornar.

En la FIGURA 7-5 se puede ver el organigrama de esta rutina: un estudio detenido de este organigrama contribuirá a una mejor comprensión de su funcionamiento.

Vamos a ensamblarla. Las líneas 130 y 170 no deben dar problemas, simplemente, recuerda sustituir "VAR1" y "VAR2" por sus respectivos

Contenido del registro PC al leer la instrucción la CPU

(PC):	1 0 0 0 0 0 0 0
	0 0 1 0 0 1 0 1

Indicador C=0  
Instrucción

JR C,+10:	0 0 1 1 1 0 0 0
	0 0 0 0 1 0 0 0

(Ensamblamos 08h que es 10-2, es decir, e-2).

Contenido del registro PC después de la ejecución

(PC):	1 0 0 0 0 0 0 0
	0 0 1 0 0 1 0 1

No se cumplía la condición porque "C" estaba a "0", de forma que no se verifica el salto y pasa a ejecutarse la instrucción que sigue en el orden normal del programa.

JR NC,e

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

Si cumple la condición, 3  
Si no cumple la condición, 2

CICLOS DE RELOJ:

Si cumple la condición, 12  
Si no cumple la condición, 7

EJEMPLO:

En la dirección absoluta 7743h

JR NC,-7
----------

Contenido del registro PC al leer la instrucción la CPU

(PC):	0 1 1 1 0 1 1 1
	0 1 0 0 0 1 0 1

Indicador C=0  
Instrucción

JR NC,-7:	0 0 1 1 0 0 0 0
	1 1 1 1 1 0 1 1

OBJETO:

Si el indicador de acarreo "C" no está activo; salta a la posición de memoria que resulta de añadir a la propia posición de la instrucción el entero de desplazamiento "e", el cual puede adquirir los valores desde -126 a +129.

Si el indicador de acarreo "C" está activo, ejecuta la siguiente instrucción.

CODIGO DE MAQUINA:

	0 0 1 1 0 0 0 0
	<---e-2-->

OBJETO:

Si el indicador de cero "Z" está activo; salta a la posición de memoria que resulta de añadir a la propia posición de la instrucción el entero de desplazamiento "e", el cual puede adquirir los valores desde -126 a +129.

Si el indicador de cero Z no está activo; ejecuta la siguiente instrucción.

CODIGO DE MAQUINA:

	0 0 1 0 1 0 0 0
	<---e-2-->

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

Si cumple la condición, 3  
Si no cumple la condición, 2

CICLOS DE RELOJ:

Si cumple la condición, 12  
Si no cumple la condición, 7

EJEMPLO:

En la dirección absoluta 4590h

JR Z,+6
---------

Contenido del registro PC al leer la instrucción la CPU

(PC):	0 1 0 0 0 1 0 1
	1 0 0 1 0 0 1 0

Indicador Z=1  
Instrucción



JR Z+6:  
28h  
0 0 1 0 1 0 0 0  
0 0 0 0 0 1 0 0  
04h

Contenido del registro PC después de la ejecución

(PC):  
45h  
0 1 0 0 0 1 0 1  
1 0 0 1 0 1 1 0  
96h

La siguiente instrucción a ejecutar será la de la posición de memoria 4596h; se efectúa el salto.

JR NZ,e

OBJETO:

Si el indicador de acarreo "Z" no está activo; salta a la posición de memoria que resulta de añadir a la propia posición de la instrucción el entero de desplazamiento "e", el cual puede adquirir los valores desde -126 a +129.

Si el indicador de acarreo "Z" está activo; ejecuta la siguiente instrucción.

CODIGO DE MAQUINA:

0 0 1 0 0 0 0 0  
<---e-2-->  
20h

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

Si cumple la condición,  
3

Si no cumple la condición,  
2

CICLOS DE RELOJ:

Si cumple la condición,  
12  
Si no cumple la condición,  
7

EJEMPLO:

En la dirección absoluta 5000h

JR NZ,-126

Contenido del registro PC al leer la instrucción la CPU

(PC):  
50h  
0 1 0 1 0 0 0 0  
0 0 0 0 0 0 0 0  
00h

Indicador Z=1

Instrucción

JR NZ,-126;  
20h  
0 0 1 0 0 0 0 0  
1 0 0 0 0 0 0 0  
80h

Contenido del registro PC después de la ejecución

(PC):  
50h  
0 1 0 1 0 0 0 0  
0 0 0 0 0 0 1 0  
02h

Ejecuta la siguiente instrucción y no se verifica el salto.

-----

Existe, dentro del repertorio del Z-80, una instrucción compleja que resulta especialmente adecuada para establecer bucles de iteración; esta instrucción es "DJNZ", abreviatura de: "Decrement and Jump if Not Zero", en castellano: "Decrementa y Salta si No es Cero". Esta instrucción puede considerarse equivalente a la sentencia "FOR...NEXT" del Basic, ya que se usa para lo mismo, si bien, de distinta forma.

Si cumple la condición,  
3

Si no cumple la condición,  
2

CICLOS DE RELOJ:

DJNZ,e

OBJETO:

Decrementa el registro "B" (le resta "1").

Si el valor del registro "B" es distinto de cero; salta a la posición de memoria que resulta de añadir a la propia posición de la instrucción el entero de desplazamiento "e", el cual puede adquirir los valores desde -126 a +129.

Si el valor del registro "B" es cero; ejecuta la siguiente instrucción (ver FIGURA 7-2).

CODIGO DE MAQUINA:

0 0 0 1 0 0 0 0  
<---e-2-->  
10h

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

Si registro "B" diferente de cero,  
3

Si registro "B" igual a cero,  
2

CICLOS DE RELOJ:

Si registro "B" diferente de cero,  
13

Si registro "B" igual a cero,  
8

EJEMPLO:

En la dirección absoluta 4723h

DJNZ,-20

Contenido del registro "B"

## Ejemplos

Los ejemplos de este capítulo van a ser bastante más complicados que los de capítulos anteriores; el poder utilizar ya las instrucciones de salto, nos va a permitir crear bucles y hacer cosas más útiles. Desarrollaremos una rutina para multiplicar, otra para dividir y, finalmente, una rutina que borra la pantalla por trozos, dependiendo el trozo borrado del valor que contenga el acumulador.

Vamos a empezar por la rutina de multiplicar. Algunas CPUs de ordenadores mayores que el Spectrum, incluyen la instrucción de multiplicar en su Assembler; como no es el caso del Z-80, tal vez resulte útil disponer de una rutina de multiplicación que se podrá incluir dentro de un programa en cualquier momento que se necesite.

En el producto AxB, vamos a llamar "multiplicando" a "A" y "multiplicador" a "B". Básicamente, multiplicar "AxB" consiste en sumar "A" sobre si mismo tantas veces como indique "B"; parece un trabajo bastante adecuado para un bucle. Supongamos que tenemos el multiplicando en el registro "DE" y el multiplicador en el registro "B"; el bucle podría ser:

LD HL,0  
BUCLE ADD HL,DE  
DJNZ BUCLE

El contenido del registro "DE" se iría sumando sobre "HL" tantas veces como indicara el contenido "B". Eviden-

"cc" Código Significado Indicador

NZ 000 no cero Z=0

Z 001 cero Z=1

NC 010 no acarreo C=0

C 011 acarreo C=1

PO 100 paridad impar o P/V=0

PE 101 paridad par o P/V=1

P 110 desbordamiento S=0

M 111 signo positivo S=1

signo negativo

Fig. 7-1. Tabla de condiciones "cc" para la instrucción de salto "JP".

temente, el multiplicando no puede ser superior a 65535 (FFFFh) ni el multiplicador puede ser superior a 255 (FFh). Pero si multiplicamos 65535 por 255 obtenemos 16711425 (FEFF01h) y este número no se puede representar con dos octetos, de forma que, a lo largo de la ejecución del bucle, obtendremos varias veces un acarreo en el registro "HL"; si no tenemos que el resultado no tenga nada que ver con la realidad, será mejor que llevemos la cuenta de las veces que se produce acarreo. Algo más correcto sería:

XOR A  
LD HL,0  
BUCLE ADD HL,DE  
JR C,CARRY

Los puntos suspensivos indican que ahí tendrán que ir algunas instrucciones que hagan algo con el resultado antes de retornar. La diferencia entre esta rutina y la otra, es que detectamos cada vez que hay un acarreo y lo acumulamos en "A"; de esta forma, el resultado final de nuestra multiplicación vendrá dado por los contenidos de "A" y "HL" puestos en el orden: "AHL", es decir, "A" será el octeto de mayor peso y "L" será el de menor.

Persiste aún un problema:



#### OBJETO:

Salta a la posición de memoria direccionada por el contenido del registro índice IX.

#### CODIGO DE MAQUINA:

11011101	00h
11101001	E9h

#### INDICADORES DE CONDICION QUE AFECTA:

Ninguno

#### CICLOS DE MEMORIA:

2

#### CICLOS DE RELOJ:

8

#### EJEMPLO:

JP IX	
-------	--

#### Contenido del registro índice IX:

10110000	80h
11110000	F0h

#### Instrucción

11011101	00h
11101001	E9h

#### Contenido del registro "PC" después de la ejecución:

10110000	80h
11110000	F0h

La siguiente instrucción a ejecutar será la de la posición de memoria B0F0h.

#### 158 CODIGO MAQUINA

ejecutar será la de la posición de memoria 7684h.

----

Las instrucciones "JP (IX)" y "JP (IY)" son de escasa utilidad, ya que los registros índices rara vez se utilizan para direccionar saltos. Estas instrucciones son, más bien, una consecuencia del procedimiento que utiliza el microprocesador para decodificar las instrucciones que llevan direccionamiento indexado.

Si mira atentamente las tablas de codificación, verá que cualquier instrucción que utilice direccionamiento indirecto a través del registro "HL", puede funcionar con direccionamiento indexado si se antepone al código de operación "DD" para el índice "IX" o "FD" para el índice "IY"; esto es una regla general.

Hubiera sido más útil disponer de las instrucciones "JP (BC)" y "JP (DE)"; por desgracia, el repertorio del Z-80 no incluye estas instrucciones; no obstante, es posible simularlas mediante un artificio que utiliza en varias ocasiones el Sistema Operativo. El procedimiento se comprenderá mejor cuando estudiemos las subrutinas, pero podemos anticipar que consiste en "engañar" al microprocesador metiendo en la pila el contenido del registro y haciendo "RET" (retorno desde subrutina), con lo que tomará el valor que hemos metido en la pila como dirección de retorno. Por ejemplo, la instrucción "JP (BC)" se puede simular con: "PUSH BC" + "RET". Esta es una de las ventajas de compartir la pila de usuario con la máquina.

----

La siguiente instrucción a

(B): 000010105h

Contenido del registro PC al leer la instrucción la CPU

01000111	47h
00100101	25h

#### Instrucción

00010000	10h
11011100	0Eh

Contenido del registro "B" después de la ejecución

(B): 000010104h

Contenido del registro PC después de la ejecución

01000111	47h
00000011	03h

La siguiente instrucción a ejecutar será la de la posición de memoria 4703h. Se ha decrementado el registro "B", pero como todavía no ha llegado a "0", se efectúa el salto.

Si el registro "B" hubiera contenido "1" antes de la ejecución, al decrementarlo hubiera pasado a valer "0" con lo que no se habría producido el salto, y se hubiera ejecutado la siguiente instrucción.

A continuación, vamos a ver cómo se utilizan esta y otras instrucciones para crear, en código máquina, bucles de diferentes tipos.

#### Bucles

Por fin, ha llegado el momento de estudiar la que es, sin duda, la principal técnica de programación. En código

ción que pongamos entre "IF" y "THEN", en caso contrario, se ejecuta la siguiente instrucción y se sale del bucle. En Assembler podemos conseguir un efecto similar:

10 BUCLE	....	¡Comienza el bucle
20	....	
30	....	
40	....	
50	JP	cc, BUCLE

¿Qué hemos hecho?: tenemos una serie de instrucciones que deben repetirse mientras la condición "cc" se cumpla; ponemos esas instrucciones a partir de una etiqueta (en nuestro caso: "BUCLE") y al final colocamos una instrucción que obligue al microprocesador a saltar a la instrucción donde está la etiqueta solamente si se cumple la condición. Al igual que antes si ésta no se cumple, se saldrá del bucle y se continuará con el proceso normal.

Los saltos absolutos tienen el grave inconveniente de no permitir la reubicación del código en un lugar distinto de donde fueron ensamblados; por otro lado, ocupan 3 bytes en lugar de los 2 que ocupa un salto relativo; así que no parece mala idea de utilizar saltos relativos para los bucles; siempre, claro está, que las instrucciones que componen el bucle no ocupen más de 127 bytes.

En el ejemplo anterior, podemos cambiar la línea 50 por:

50	JR	cc, BUCLE
----	----	-----------

Esta vez el bucle sólo se cierra si se cumple la condi-

máquina (o en Assembler), necesitaremos uno o más bucles para casi cualquier cosa que queramos hacer, así que recomendamos al lector que ponga mucha atención en esta parte, y la lea las veces que sean necesarias hasta que consiga comprenderlo perfectamente.

Al igual que en Basic, existen varias formas de hacer un bucle: la más sencilla consiste en escribir una serie de instrucciones y terminar con un "GO TO" que mande, de nuevo, a la primera de ellas. En Basic sería algo así:

10 REM Comienza el bucle
20 .....
30 .....
40 .....
50 GO TO 10

Este bucle tiene un fallo gravísimo: el ordenador se queda atrapado en el eterno "B"; pero en Assembler no vamos a disponer de un "BREAK" tan fácilmente, así que será mejor que tengamos mucho cuidado de no crear bucles donde el ordenador se quede atrapado. Algo más correcto sería:

10 REM Comienza el bucle
20 .....
30 .....
40 .....
50 IF ... THEN GO TO 10

#### CODIGO MAQUINA 155



La condición "cc" puede ser cualquiera de: "Z", "NZ", "C" y "NC"; al ensamblar esta línea, ocupará solamente 2 bytes y, además, correrá exactamente igual en cualquier posición de memoria.

Ahora, vamos a prestar un poco de atención a la condición "cc" que nos permite cerrar el bucle; examine este programa en Basic:

```
10 LET b=12
20 REM comienza el bucle
30 ....
40 ....
50 LET b=b-1
60 IF b<>0 THEN GO TO 20
```

Es fácil comprobar que el bucle se repetirá 12 veces; en cada pasada se resta "1" a "b"; en la duodécima pasada, "b" llegará a valer "0", con lo que no se cumple la condición "b<>0" y se sale del bucle. En Assembler, esto sería algo así:

```
10 LD B,12
20 BUCLE .....
30 .....
40 .....
50 DEC B
60 JR NZ,BUCLE
```

La lógica es la misma: cargamos "12" en el registro "B" y lo decrementamos en cada pasada, cerrando el bucle mientras "B" sea distinto de "0". ¿Por qué hemos elegido el registro "B"? las dos instrucciones "DEC B" y "JR NZ, BUCLE" ocupan, en total, 3 bytes; pero éste es el sitio adecuado para colocar la instrucción "DJNZ" de la siguiente forma:

```
50 DJNZ BUCLE
```

Esto es lo que se denomina un "BUCLE DE ITERACION" y el registro "B" resulta especialmente idóneo para usarlo en este tipo de bucles; aun- que nada nos impide utilizar otros registros; pero tenga en cuenta que "DJNZ" sólo actúa sobre el registro "B".

Observe el siguiente programa en Basic:

```
10 LET c=15
20 REM Bucle exterior
30 LET b=12
40 REM Bucle interior
50 .....
60 .....
70 LET b=b-1
80 IF b<>0 THEN GO TO 40
90 LET c=c-1
100 IF c<>0 THEN GO TO 20
```

Es equivalente a:

```
10 FOR c=0 TO 15
20 FOR b=0 TO 12
30 .....
40 .....
50 NEXT b
60 NEXT c
```

Efectivamente, se trata de dos bucles "anidados", es decir, uno dentro del otro. En Assembler también podemos hacerlo:

```
10 LD C,15
20 BUC_1 LD B,12
30 BUC_2 .....
```

```
40 .....
50 .....
60 DJNZ BUC_2
70 DEC C
80 JR NZ,BUC_1
```

Parece que si empezamos a anidar muchos bucles uno dentro de otro, se nos acabarán pronto los registros. Bien, no es cierto, podemos utilizar la "pila".

```
10 LD B,18
20 BUC_1 PUSH BC
30 LD B,15
40 BUC_2 PUSH BC
50 LD B,12
60 BUC_3 .....
70 .....
80 .....
90 DJNZ BUC_3
100 POP BC
110 DJNZ BUC_2
120 POP BC
130 DJNZ BUC_1
```

Si se toma la molestia de seguir el curso al programa, verá que tenemos anidados tres bucles, cada uno dentro del otro, todos están contralados por el mismo registro y no hay posibilidad de confusión; vamos guardando los registros en la pila y recuperándolos sólo cuando es necesario.

Hasta ahora, hemos supuesto que el número de iteraciones (veces que tiene que repetirse el bucle) era menor de 256; pero ¿y si fuera mayor?; en ese caso, tendríamos que recurrir a un registro de 16 bits como contador del bucle, por ejemplo, el "BC". Existe un pequeño inconveniente, al decrementar el "BC" no resultan afectados

los indicadores, pero podemos evitarlo con un pequeño truco:

```
10 LD BC,500
20 BUCLE .....
30 .....
40 .....
50 DEC BC
60 LD A,B
70 OR C
80 JR NZ,BUCLE
```

¿Ingenioso verdad?; por desgracia no se nos ha ocurrido a nosotros, lo hacen todos los programadores de Assembler; se trata, simplemente, de cargar en "A" el contenido de "B" y hacerle un "OR" con "C"; el resultado sólo será "0" si ambos, "B" y "C", valen "0". Tiene el ligero inconveniente de modificar el contenido de "A", pero para eso tenemos la pila:

```
10 LD BC,500
20 PUSH AF
30 BUCLE POP AF
40 .....
50 .....
60 DEC BC
70 PUSH AF
80 LD A,B
90 OR C
100 JR NZ,BUCLE
110 POP AF
```

La cosa se complica un poco con los PUSH y POP, pero funciona de maravilla y el contenido de "A" no se altera en absoluto. No olvide que la línea donde se define el valor inicial del contador tiene que estar antes de la etiqueta

donde se inicia el bucle; un error muy típico de principiante consiste en colocar la etiqueta en la línea donde se carga el contador con el valor inicial, algo como:

```
10 BUCLE LD B,2
20 .....
30 .....
40 DJNZ BUCLE
```

Terrible error; el registro "B" siempre vale "2" y el ordenador se queda "enganchado" indefinidamente en el bucle.

Por último y para terminar con el tema de los bucles, vamos a contarle un truco: si desea iterar un bucle 256 veces, no podrá cargar el número "256" en el registro "B", pero no necesita recurrir a un registro de 16 bits; bastará con que cargue el registro "B" con "0" y el bucle se iterará 256 veces. Si desea más de 65536 iteraciones (cosa bastante improbable), lo más sencillo es anidar dos bucles; el número total de iteraciones vendrá dado por el producto de las iteraciones de cada uno de los dos bucles; con 65536 iteraciones en cada uno, hasta un microprocesador relativamente rápido como el Z-80A tardará unos cuantos segundos.

## Instrucciones de salto indirecto

Salta a la posición de memoria direccionada por el contenido del par de registros "HL".

CODIGO DE MAQUINA:

```
1 1 1 0 1 0 0 1 E9h
```

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

```
JP (HL)
```

Contenido del par de registros "HL"

```
(H): 1 0 0 0 1 0 1 0 8Ah
(L): 0 0 0 0 1 0 1 1 0Bh
```

Instrucción

```
JP (HL): 1 1 1 0 1 0 0 1 E9h
```

Contenido del registro "PC" después de la ejecución.

```
(PC): 1 0 0 0 1 0 1 0 8Ah
      0 0 0 0 1 0 1 1 0Bh
```

La siguiente instrucción a ejecutar será la de la posición de memoria 8A08h. El salto es incondicional y, por tanto, se verifica siempre.

JP (HL)

JP (IX)



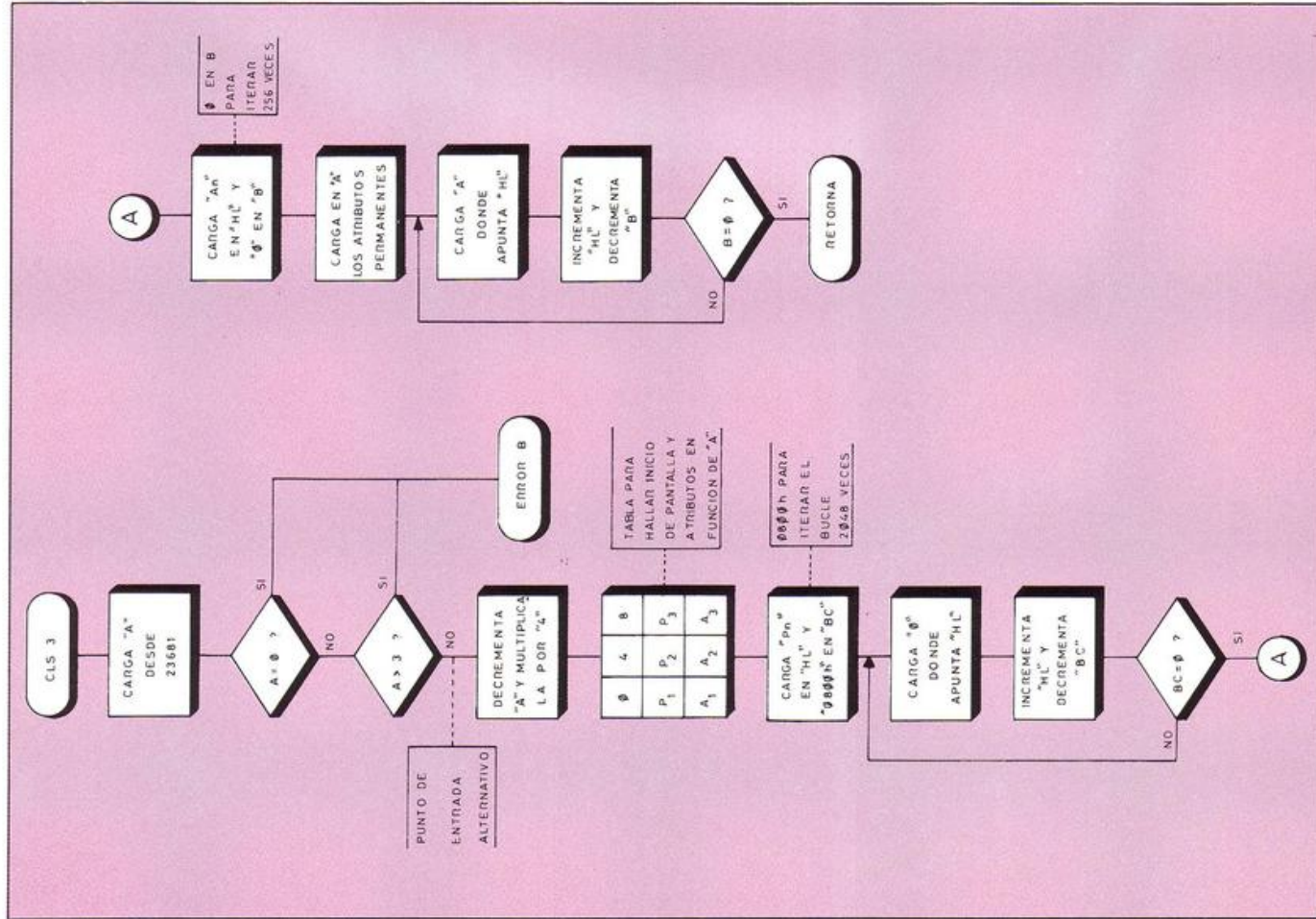


Fig. 7-8. Organigrama de la rutina para borrar la pantalla por tercios.

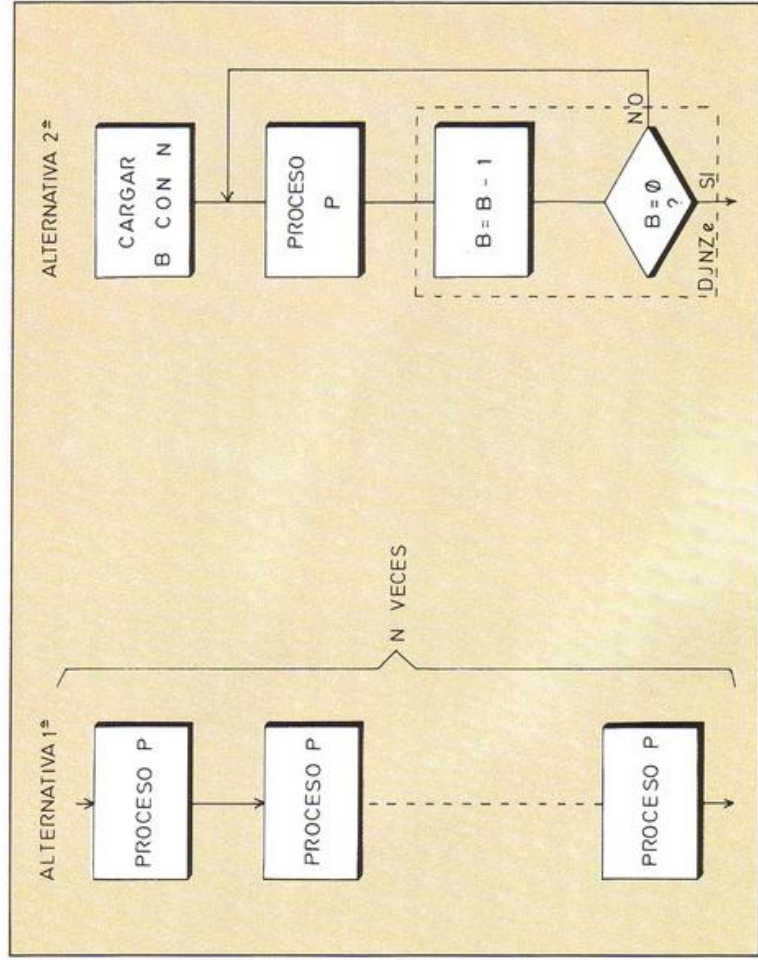


Fig. 7-3. Técnica de bucles.

valores: 23296 (5B00h) y 23298 (5B02h). El primer problema surge en el salto relativo de la línea 180; esta instrucción va a ocupar las direcciones 23310 y 23311 de forma que, cuando el micro acabe de leerla, el "PC" apuntará a 23312, y queremos que salte a 23331 que es donde está la etiqueta "CERO", así que el desplazamiento será:  $23331 - 23312 = 19$ ; ensamblaremos "40" que es el código de operación y "19" que es el desplazamiento. El siguiente salto relativo está en la línea 220 "JR CARRY", se ensamblará en las posiciones 23317 y 23318 y tiene que saltar a 23328, así que:  $23328 - 23319 = 9$ ; ensamblare-

mos "56" y "9". El salto de la línea 230 es hacia atrás, pero tampoco debe haber problema: "DJNZ BUCLE" está en 23319 y "BUCLE" está en 23320 y "BUCLE" está en 23316; así que  $23316 - 23321 = -5$ ; miramos en la segunda columna de la tabla de la página 14 y vemos que "-5" equivale a "251", de forma que ensamblamos "16" y "251". Es conveniente ensamblar primero todo el programa dejando en blanco el espacio equivalente a los saltos relativos, y luego calcular estos cuando ya se sabe que direcciones ocupa cada instrucción. Intente ensamblar por sí mismo toda la rutina y luego compruebe si le ha

quedado así:

130	23300	33,2,91
140	23303	70
150	23304	237,91,0,91
160	23308	5
170	23309	4
180	23310	40,19
190	23312	175
200	23313	33,0,0
210	23316	25
220	23317	56,9
230	23319	16,251
240	23321	34,0,91
250	23324	50,2,91
260	23327	201



270 23328 60  
280 23329 24, 244  
290 23331 33, 0, 0  
300 23334 34, 0, 91  
310 23337 201

Hemos representado, por este orden: número de línea, dirección de memoria y código máquina. Habrá comprobado que resulta sumamente tedioso ensambalar a mano rutinas con saltos relativos; no se preocupe por ello, precisamente por eso se inventaron los ensambladores; si se dedica a hacer programas en Assembler, es seguro que utilizará un ensamblador. En un capítulo posterior, enseñaremos a utilizarlos; pero, de momento, el ensambalar a mano le va a servir para comprender mejor el código máquina.

En el orden que seguimos habitualmente, ahora tocaría desarrollar un pequeño programa en Basic que nos permita utilizar esta rutina; pero hemos creído más lógico hacer, primero, una rutina para dividir y, luego, el programa en Basic que maneje las dos rutinas. Así que vamos con la división.

Dado que trabajaremos con números enteros, nuestra rutina de dividir no sacará decimales; se limitará a dividir un número por otro y darnos un cociente y un resto. Esto suele ser más útil, en Assembler, que sacar decimales; no obstante, la rutina es fácil de modificar; si quiere sacar 2 decimales, multiplique el resultado por 100 y siga dividiendo; si quiere 3 decimales, multiplíquelo por 1.000 y así sucesivamente.

## Código Fuente Hexadecimal Decimal

JP nn	C3,n,n	195,n,n
JP NZ,nn	C2,n,n	194,n,n
JP Z,nn	CA,n,n	202,n,n
JP NC,nn	D2,n,n	210,n,n
JP C,nn	DA,n,n	218,n,n
JP PO,nn	E2,n,n	226,n,n
JP PE,nn	EA,n,n	234,n,n
JP P,nn	F2,n,n	242,n,n
JP M,nn	FA,n,n	250,n,n
JP (HL)	E9	233
JP (IX)	DD,E9	221,233
JP (IY)	FD,E9	253,233
JR e	18,e	24,e
JR C,e	38,e	56,e
JR NC,e	30,e	48,e
JR Z,e	28,e	40,e
JR NZ,e	20,e	32,e
DJNZ e	10,e	16,e

Fig. 7-4. Tabla de codificación para las instrucciones de salto.

vamente, de forma similar a como se hace al dividir "a mano" sobre un papel.

En la operación "A/B" llamaremos "dividendo" a "A" y "divisor" a "B"; utilizaremos las mismas variables que para la multiplicación. "VAR1" almacenará, en la entrada, el dividendo y, en la salida, el cociente; "VAR2" almacenará, en la entrada, el divisor y,

en la salida, el resto. En estas condiciones, el dividendo no podrá ser mayor de 65535 ni el divisor mayor de 255; por tanto, el cociente estará siempre comprendido entre "0" y "65535", y el resto entre "0" y "254" (ambos inclusive).

Dividir "A" entre "B" consiste en ir restando "B" de "A" hasta que lo que quede sea menor que "B", en ese mo-

permanentes en curso, tal como están definidos en la variable del Sistema "ATTR-T" (dirección 23693).

Cada una de estas operaciones es llevada a cabo por un bucle. Hemos denominado "BUC\_1" al bucle que borra el archivo de presentación visual; al entrar en él, "HL" contiene la dirección de inicio del sector a borrar y "BC" contiene la longitud del mismo (0800h 2048 bytes). Su listado es:

```
420 BUC_1 XOR A
430 LD (HL), A
440 INC HL
450 DEC BC
460 LD A, B
470 OR C
480 JR NZ, BUC_1
```

La línea 420 carga "00" en "A", la 430 carga "00" en la dirección apuntada por "HL"; 440 y 450 incrementan el puntero "HL" y decrementan el contador "BC"; 460 y 470 comprueban si "BC" ha llegado a "0" y, en caso contrario, la línea 480 cierra el bucle.

El segundo bucle "BUC\_2" borra el archivo de atributos, se entra en él con "HL" conteniendo la dirección de comienzo y "B" la longitud de la zona a borrar; como es 256 bytes, "B" deberá contener "0" y "A" contendrá los atributos permanentes. Su listado es:

```
530 BUC_2 LD (HL), A
540 INC HL
550 DJNZ BUC_2
```

El funcionamiento es tan sencillo que no hace falta ninguna explicación. El contenido de "HL" al entrar en estos bucles dependerá del sector de pantalla que deseemos borrar, y estará en función del dato contenido en "A" al entrar en la rutina; pero, ¿cómo hallamos las direcciones de inicio de cada bloque en función de "A"? La solución más fácil es utilizar una tabla.

En determinado lugar de la memoria, almacenamos las direcciones de inicio de cada uno de los bloques en el siguiente orden:

```
4000h (pantalla 1)
5800h (atributos 1)
4800h (pantalla 2)
5900h (atributos 2)
5000h (pantalla 3)
5A00h (atributos 3)
```

Los números deberán estar almacenados en un formato susceptible de ser leído por el Z-80, es decir, primero irá el octeto menos significativo y luego el más significativo. El inicio de la tabla lo marcamos con la etiqueta "TABLA"; su pongamos que está a partir de 60032:

```
60032 0
60033 64
60034 0
60035 88
60036 0
60037 72
60038 0
60039 89
60040 0
60041 80
60042 0
60043 90
```

La etiqueta "TABLA" valdrá 60032 y a ese valor le llamaremos "dirección base de la tabla". Utilizaremos los cuatro primeros datos de la tabla cuando "A" valga "1", los cuatro segundos cuando valga "2" y los cuatro terceros cuando valga "3"; de esta forma, si restamos "1" a "A", multiplicamos por "4" y sumamos el resultado a la dirección base, estaremos apuntando al grupo de datos que nos interesan. Este es el funcionamiento básico de una tabla de "offset" (la más sencilla). En general, el inicio de la tabla se denomina "dirección base", cada elemento de la tabla puede tener "n" bytes de longitud; el número del elemento al que deseamos acceder (subíndice) debe estar comprendido entre "0" y "m-1" siendo "m" el número de elementos de la pantalla. En estas condiciones, multiplicamos el subíndice por "n" y le sumamos la dirección base con lo que el resultado queda apuntando al elemento de la tabla que nos interesa. En nuestro caso, la tabla tiene 3 elementos de 4 bytes cada uno.

Para crear tablas, resulta muy útil el pseudo-nemónico "DEFW" (DEFINE Word) que nos almacena un número entre "0" y "65535" en dos bytes y en el formato adecuado al Z-80; de esta forma, nuestra tabla queda:

```
330 TABLA DEFW #4000
340 DEFW #5800
350 DEFW #4800
360 DEFW #5900
370 DEFW #5000
380 DEFW #5A00
```



[illegible][illegible]

las direcciones en decimal en vez de en Hexa (+D+). En la línea 320 se ha puesto un "punto y coma" para separar una rutina de la otra. La rutina de dividir queda ensamblada a partir de 23338, es decir, a continuación de la anterior.

Ahora si ha llegado el momento de desarrollar el programa en Basic correspondiente. Este es el PROGRAMA 7-1; las líneas 20 a 40 cargan en memoria en código máquina de las dos rutinas; tenga sumo cuidado de no equivo-

El segundo de nuestros ejemplos es una rutina que permite borrar la pantalla "por trozos" en función del contenido del acumulador en el momento de entrar a ella.

El archivo de presentación visual tiene una disposición un tanto "curiosa" que explicaremos en profundidad más adelante. De momento, lo primero que se aprecia es que está dividido en tres bloques (si carga una pantalla desde la cinta lo verá claramente). Cada bloque tiene 2048 (0800h) bytes de longitud y sus direcciones respectivas de inicio son:

forma reubicable con la finalidad de que, en un futuro, pueda incluirse en sus programas en C/M cada vez que necesite multiplicar o dividir. Con los conocimientos adquiridos hasta ahora, es muy probable que algunos lectores consigan mejorar estas rutinas; en ese caso, agradeceremos sinceramente que nos remitan las rutinas mejoradas con el fin de establecer una mayor comunicación con los lectores que estén siguiendo el curso.

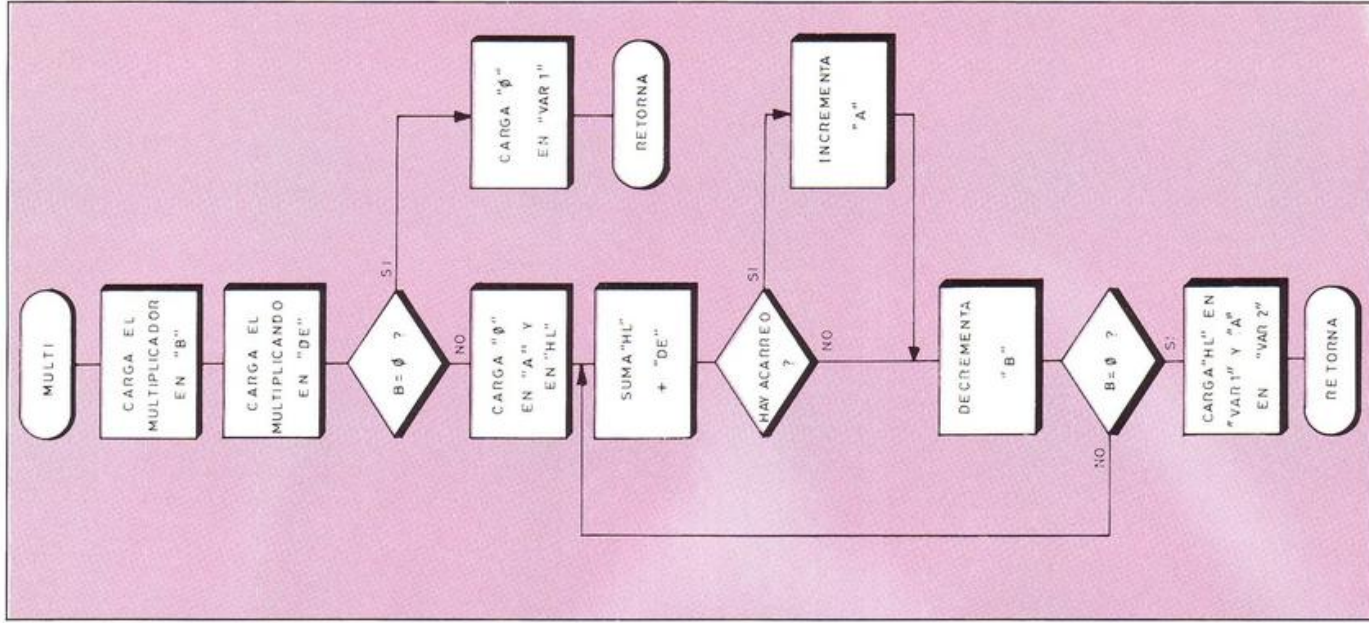
Nuestra rutina borrará el primer bloque si "A" contiene un 1, el segundo si contiene un 2 y el tercero si contiene un 3.

El borrado de una zona de pantalla conlleva dos operaciones: primero, se cargan con "00" todas las posiciones de memoria del archivo de presentación visual correspondiente a esa zona; después, se copian, en la parte del archivo de atributos correspondiente, los atributos

1°: 16384 (4000h)  
2°: 18432 (4800h)  
3°: 20480 (5000h)

El archivo de atributos está colocado de forma más lógica, pero podemos dividirlo en 3, pero podemos dividirlo en 3 bloques de 256 (100h) bytes de longitud; cada uno de los cuales se corresponde con un bloque del archivo de presentación visual. Sus direcciones de comienzo son:

1°: 22528 (5800h)  
2°: 22784 (5900h)  
3°: 23040 (5A00h)



**Fig. 7-5. Organigrama de la rutina para multiplicar.**

mento, lo que queda de "A" es el resto y el número de veces que hayamos podido restar es el cociente. De nuevo parece que lo ideal es un bucle.

¿Cómo detectamos que lo que queda de "A" es menor que "B"?; muy sencillo, vamos restando hasta que tengamos acarreo, en ese momento volvemos un paso atrás, es decir, sumamos una vez, y ya tenemos el resto; si hemos llevado la cuenta de estas restas, ese será el cociente más "1" (hemos restado una vez más de las que debíamos). Por otro lado, deberíamos comprobar que el divisor no sea "0" ya que, en ese caso, no podríamos efectuar la división. Vamos a ver la rutina:

330	DIV1	LD	HL, VAR2
340		LD	C, (HL)
350		DEC	C
360		INC	C
370		JR	Z, ERROR
380		LD	B, 0
390		LD	DE, 0
400		LD	HL, (VAR1)
410	LAZ0	AND	A
420		SBC	HL, BC
430		JR	C, FINAL
440		INC	DE
450		JR	LAZ0
460	FINAL	ADD	HL, BC
470		LD	A, L
480		LD	(VAR2), A
490		LD	(VAR1), DE
500		RET	
510	ERROR	RST	8
520		DEFB	5

Las líneas 330 y 340 cargan el divisor en "C" ("B" se pone a "0" en 380 para que "BC" contenga el divisor). En



350, 360 y 370 comprobamos si el divisor es "0" y, en ese caso, saltamos a 510 (luego explicaremos las instrucciones de 510 y 520 que parecen tan raras). "DE" será el contador de restas, así que lo ponemos inicialmente a "0" en 390; en 400 cargamos el dividendo en "HL" y ya estamos preparados para empezar a dividir. Dado que no disponemos de la instrucción "SUB HL, BC" y tenemos que usar "SBC", ponemos el indicador de acarreo a "0" para que no nos incorpore (AND A en la línea 410). En el bucle de las líneas 410, 420, 430, 440 y 450 vamos restando e incrementando "DE" hasta que se produzca un acarreo en una de las restas, momento en el que saltamos a la línea 460 donde sumamos "HL" con "BC". El cociente está en "DE" y el resto en "HL" pero, como no puede ser mayor de 254, nos basará con considerar que está en "L". En las líneas 470, 480 y 490 guardamos los resultados en "VAR1" y "VAR2" antes de retornar en 500.

Cuando estudiamos la pila, dijimos que el Sistema Operativo del Spectrum permite retornar a Basic en cualquier caso, incluso, con la pila desordenada; pues bien, ahora vamos a usar esta posibilidad. La instrucción "RST 8" (ReStart 8) obliga al microprocesador a saltar a la dirección de memoria "0008h", donde se encuentra la rutina de la ROM que maneja los errores. En cualquier momento de un programa, podemos hacer "RST 8" y lo que ocurrirá será que se detendrá la ejecución de cualquier programa, apareciendo, en la parte inferior de la pantalla,

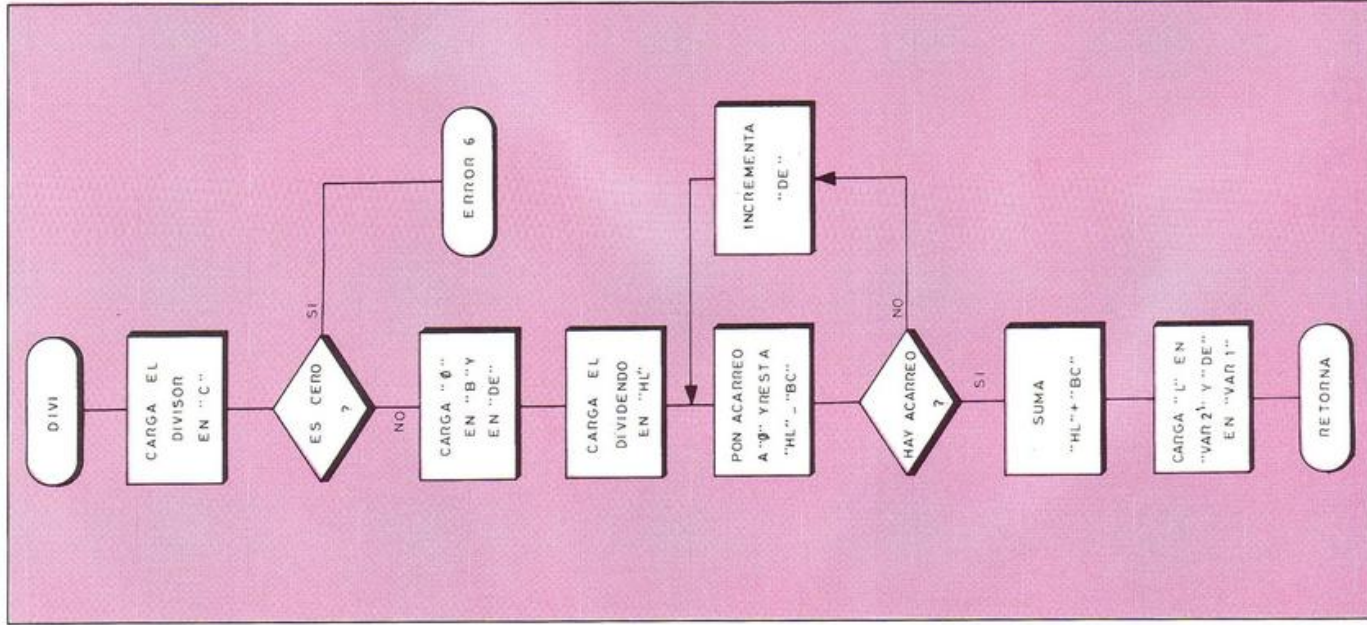


Fig. 7-6. Organigrama de la rutina para dividir.

Pass 1 errors: 00

23276	100	VAR1	LD	23276	330	DIV1	LD	HL, VAR2
23298	110	VAR2	LD	23298	340		LD	C, HL
23300	120		ORG	23300	350		DEC	C
23300	130	MULTI	LD	23300	360		INC	C
23303	140		LD	23303	370		JR	Z, ERROR
23304	150		LD	23304	380		LD	B, 0
23308	160		DEC	23308	390		LD	DE, 0
23309	170		INC	23309	400		LD	HL, VAR1
23310	180		JR	23310	410	LAZO	AND	A
23312	190		XOR	23312	420		SBC	HL, BC
23313	200		LD	23313	430		JR	C, FINAL
23316	210	BUCLE	ADD	23316	440		INC	DE
23317	220		JR	23317	450	FINAL	JR	LAZO
23319	230	LOOP	DJNZ	23319	460		ADD	HL, BC
23321	240		LD	23321	470		LD	A, L
23324	250		LD	23324	480		LD	(VAR2), A
23327	260		RET	23327	490		LD	(VAR1), DE
23328	270	CARRY	INC	23328	500		RET	B
23329	280		JR	23329	510	ERROR	RST	8
23331	290	CERO	LD	23331	520		DEFB	5
23334	300		LD	23334				
23337	310		RET	23337				
23337	320			23337				

Pass 2 errors: 00  
Table used: 130 from 170

Fig. 7-7. Listado completo de las rutinas para multiplicar y dividir

un mensaje de error (al igual que cuando ocurre un error en Basic); el mensaje que aparece depende del contenido de la posición de memoria siguiente a donde está la instrucción "RST 8"; esta posición de memoria debe contener un número que será igual al código del error menos "1".

La pseudo-instrucción "DEFB" no corresponde al juego de instrucciones del Z-80 (igual que "ORG" o "EQU"), se utiliza para indicarle al ensamblador que almacene un número determinado en la posición de memoria correspondiente. En nuestro caso, deseamos que, si el divisor es "0", el programa se detenga con el error: "6 Number too big", así que almacenamos un "5" (6-1) en la po-

sición de memoria siguiente a "RST 8".

En la FIGURA 7-6 tiene el organigrama de la rutina para dividir. Intente ensamblar la rutina por su cuenta y, luego, compruebe resultados. La instrucción "RST 8" se ensambla como "207". Este es nuestro listado:

430	23357	56,3
440	23359	19
450	23360	24,248
460	23362	9
470	23363	125
480	23364	50,2,91
490	23367	237,83,0,91
500	23371	201
510	23372	207
520	23373	5

330	23338	33,2,91
340	23341	78
350	23342	13
360	23343	12
370	23344	40,26
380	23346	6,0
390	23348	17,0,0
400	23351	42,0,91
410	23354	167
420	23355	237,66

En la FIGURA 7-7 tiene el listado completo de las dos rutinas tal y como lo produce un ensamblador "GENS3" cuando ensambla. Las líneas 10 y 20 son comandos del ensamblador que se han utilizado para no imprimir el código objeto (\*C-) y para imprimir



Ninguno

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

EX (SP),IX

Contenido del par de registros "SP"

(SP):  
C6h: 1 1 0 0 0 1 1 0  
78h: 0 1 1 1 1 0 0 0

Contenido del octeto de memoria C678h

C678h: 1 0 1 0 1 0 1 0

Contenido del octeto de memoria C679h

C679h: 0 1 0 1 0 1 0 1

Contenido del par de registros "IX"

(IX):  
63h: 0 1 1 0 0 0 1 1  
92h: 1 0 0 1 0 0 1 0

Instrucción

EX (SP),IX  
E3h: 1 1 0 1 1 1 0 1  
F3h: 1 1 1 0 0 0 1 1

Contenido del octeto de memoria C678h después de la ejecución

C678h: 1 0 0 1 0 0 1 0

Contenido del octeto de memoria C679h después de la ejecución

C679h: 0 1 1 0 0 0 1 1

Contenido del par de registros "IX" después de la ejecución

(IX):  
55h: 0 1 0 1 0 1 0 1  
A4h: 1 0 1 0 1 0 1 0

EX (SP),IX

OBJETO:

Intercambia el último dato de la pila con el contenido del registro índice "IX". (Ver FIGURA 8-1.)

CODIGO DE MAQUINA:

FDh: 1 1 1 1 1 1 0 1  
E3h: 1 1 1 0 0 0 1 1

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

EX (SP),IX

Contenido del par de registros "SP"

(SP):  
57h: 0 1 0 1 0 1 1 1  
88h: 1 0 0 0 1 0 0 0

Contenido del octeto de memoria 5788h

5788h: 1 0 0 1 1 1 1 0

Contenido del octeto de memoria 5789h

5789h: 0 0 1 1 0 1 0 0

Contenido del par de registros "IX"

(IX):  
09h: 0 0 0 0 1 0 0 1  
44h: 0 1 0 0 0 1 0 0

Instrucción

EX (SP),IX  
FDh: 1 1 1 1 1 1 0 1  
E3h: 1 1 1 0 0 0 1 1

Contenido del octeto de memoria 5788h después de la ejecución

5788h: 0 1 0 0 0 1 0 0

Contenido del octeto de memoria 5789h después de la ejecución

5789h: 0 0 0 0 1 0 0 1

Contenido del par de registros "IX" después de la ejecución

(IX):  
34h: 0 0 1 1 0 1 0 0  
9Eh: 1 0 0 1 1 1 1 0

Es conveniente hacer notar que, aunque estas instrucciones cumplen la propiedad conmutativa (da igual intercambiar (SP) con IX que intercambiar IX con (SP)), es necesario escribir los operandos en el orden que se indica ya que, de lo contrario, un ensamblador no las reconocería.

En la rutina para leer la tabla, entraremos con "A" conteniendo "1", "2" ó "3" y saldremos con "BC" conteniendo la dirección inicial de la zona correspondiente en el archivo de pantalla y "DE" la del archivo de atributos; el listado podría ser el siguiente:

```
180 CLS3 DEC A
190 ADD A,A
200 ADD A,A
210 LD HL,TABLA
220 LD B,0
230 LD C,A
240 ADC HL,BC
250 LD C,(HL)
260 INC HL
270 LD B,(HL)
280 INC HL
290 LD E,(HL)
300 INC HL
310 LD D,(HL)
```

La línea 180 resta "1" a "A", las 190 y 200 multiplican por "4", en 210, 220, 230 y 240 sumamos "A" a la dirección base de la tabla y obtenemos el resultado en "HL"; las siguientes líneas se limitan a ir cargando en "BC" y "DE" los datos correspondientes de la tabla. Dadas las direcciones usadas, los contenidos de "C" y "E" serán siempre "0", por lo que la rutina podría haberse simplificado bastante; pero hemos preferido hacerlo así para que el lector pueda usar esta rutina, u otra similar, siempre que tenga acceso a una tabla. Ya tenemos casito do el trabajo hecho, pero aún faltan algunos retoques: antes del "BUC\_1" deberemos transferir el dato de "BC" a "HL" y cargar "BC" con

"0800h"; antes del "BUC\_2" tenemos que pasar el dato de "DE" a "HL" y cargar "B" con "0" (esto se podría haber omitido, ya que "B" vale "0" como condición de salida del bucle anterior, no obstante, se ha incluido para mayor claridad); finalmente, deberemos añadir algunas instrucciones que lean el valor inicial de "A" (desde una posición de memoria donde habrá sido colocado en Basic antes de saltar a la rutina), y comprueben si está dentro de rango dando, en caso contrario, un informe de error; por ejemplo: "B Integer out of range". El listado completo de la rutina sería el siguiente:

```
100 OR5 60000
110 LD A,(23681)
120 AND A
130 JR Z,ERROR
140 CP 4
150 JR C,CLS3
160 ERROR RST 8
170 DEFB 00A
180 CLS3 DEC A
.....
320 JR CLS3_1
330 TABLA DEFW 40000
.....
390 CLS3_1 LD H,B
400 LD L,C
410 LD BC,40800
420 BUC_1 XOR A
.....
490 LD H,D
500 LD L,E
510 LD A,(23693)
520 LD B,0
530 BUC_2 LD (HL),A
.....
560 RET
```

La línea 110 carga "A" desde "23681"; en 120 y 130 comprobamos si es "0", y si es así, saltamos a "ERROR"; en 140 y 150 comprobamos si es menor de "4", y, en caso afirmativo, saltamos a "CLS3" que es la rutina propiamente dicha.

Para el tratamiento del error, hemos utilizado, de nuevo, la instrucción "RST 8" seguida del literal "00Ah" correspondiente al mensaje "B Integer out of range". A partir de "CLS3" están las instrucciones que manejan la tabla, en la línea 320 saltamos la tabla siguiendo en "CLS3\_1", donde nos preparamos para entrar en el primer bucle "BUC\_1"; a partir de la línea 490, nos preparamos para entrar en "BUC\_2" y en la 560, retornamos a Basic habiendo borrado el sector correspondiente de pantalla.

¿Le parece complicado?; no lo crea, programar en Assembler acaba siendo una tarea bastante rutinaria y, con el tiempo, comprobará que las rutinas de este tipo le salen "como churros".

En la FIGURA 7-8 tiene el organigrama para que lo vea más claro y, en la FIGURA 7-9, tiene el listado completo tal y como lo produce el "GENS-3" cuando ensambla.

Suponemos que, a estas alturas, ya habrá corrido a entender el ordenador para meter la rutina; vamos a ensamblarla. Recordemos que "RST 8" se ensambla como "207"; la tabla ya se la hemos dado ensamblada antes, así que no debe haber problema con ella; la etiqueta "TABLA" equivale a 60032 si ensambla la rutina a partir de 60000 (luego explicaremos como reubicarla); lo demás es fácil.



6.0000	58	129	92	167	40
6.0005	4	254	4	56	2
6.0010	297	18	61	135	135
6.0015	33	128	234	6	
6.0020	79	237	74	78	35
6.0025	78	35	94	35	86
6.0030	24	12	0	64	0
6.0035	88	0	72	0	89
6.0040	0	80	0	96	95
6.0045	185	1	0	8	175
6.0050	119	35	11	126	177
6.0055	32	248	98	187	58
6.0060	141	92	6	0	119
6.0065	35	16	252	28	

(OR6+16) = TABLA-256\*INT (TABLA/256)  
(OR6+17) = INT (TABLA/256)

170 CODIGO MAQUINA

10	*C-	20	*D+	30	40	50	60	70	80	90	100
600000		600000		600000		600000		600000		600000	
600000		600000		600000		600000		600000		600000	
600003		600003		600003		600003		600003		600003	
600004		600004		600004		600004		600004		600004	
600006		600006		600006		600006		600006		600006	
600008		600008		600008		600008		600008		600008	
600110	160	600110	160	600110	160	600110	160	600110	160	600110	160
600111	170	600111	170	600111	170	600111	170	600111	170	600111	170
600112	180	600112	180	600112	180	600112	180	600112	180	600112	180
600113	190	600113	190	600113	190	600113	190	600113	190	600113	190
600114	200	600114	200	600114	200	600114	200	600114	200	600114	200
600115	210	600115	210	600115	210	600115	210	600115	210	600115	210
600118	220	600118	220	600118	220	600118	220	600118	220	600118	220
600220	230	600220	230	600220	230	600220	230	600220	230	600220	230
600221	240	600221	240	600221	240	600221	240	600221	240	600221	240
600223	250	600223	250	600223	250	600223	250	600223	250	600223	250
600224	260	600224	260	600224	260	600224	260	600224	260	600224	260
600225	270	600225	270	600225	270	600225	270	600225	270	600225	270
600226	280	600226	280	600226	280	600226	280	600226	280	600226	280
600227	290	600227	290	600227	290	600227	290	600227	290	600227	290
600228	300	600228	300	600228	300	600228	300	600228	300	600228	300
600229	310	600229	310	600229	310	600229	310	600229	310	600229	310
600330	320	600330	320	600330	320	600330	320	600330	320	600330	320
600332	330	600332	330	600332	330	600332	330	600332	330	600332	330
600334	340	600334	340	600334	340	600334	340	600334	340	600334	340
600336	350	600336	350	600336	350	600336	350	600336	350	600336	350
600338	360	600338	360	600338	360	600338	360	600338	360	600338	360
600440	370	600440	370	600440	370	600440	370	600440	370	600440	370
600442	380	600442	380	600442	380	600442	380	600442	380	600442	380
600443	390	600443	390	600443	390	600443	390	600443	390	600443	390
600445	400	600445	400	600445	400	600445	400	600445	400	600445	400

170 CODIGO MAQUINA



**Fig. 8-1: Diagrama de funcionamiento del intercambio entre registros y memoria.**

Contenido del octeto de memoria 86F4h después de la ejecución

(SP) <sup>2</sup>	86h	F3h
1	0	0
0	0	0
0	1	1
1	1	1
1	1	0
1	0	1
1	1	1

Contenido del octeto de memoria 86F3h

86F3h: 00 1 1 00 0 1 31h

11100011	E3h
----------	-----

Contenido del par de registros "H1"

Intercambia el último dato de la pila con el contenido del registro índice "IX". (Ver FIGURA 8-1.)

22

### Instrucción

EX (SP),HL	11100011	E3h	CODIGO DE MAQUINA:
------------	----------	-----	--------------------

Contenido del octeto de memoria 86F3h después de la ejecución

Contenido del octeto de memoria 86F3h después de la ejecución

INDICADORES DE  
CONDICION QUE AFECTA:

CODIGO MAQUINA 175



Contenido del par de registros "BC", "DE" y "HL"

00h	00h
00 00 00 00 00	00 00 00 00 00

### Instrucción

11011001

Contenido del par de registros "BC", "DE" y "HL" después de la ejecución

Contenido del par de registros "AF"

11011001 460

Contenido del par de registros "BC" después de la ejecución

(B'):	0 1 1 0 1 1 1 1	6Fh
(C'):	1 0 0 0 0 1 0 1	85h

Contenido del par de registros "DE" después de la ejecución

(D'):	00010001	11h
(E'):	01101110	6Eh

Contenido del par de registros "HL" después de la ejecución

(H'):	01111000	78h
(L'):	00110100	34h

EX (SP), HL

EA (3F), NL

**OBJETO:**  
Cambia: el contenido del registro "H" por contenido del octeto de memoria direccionado por el registro "SP" +1; el contenido del registro "L" por el contenido del octeto de memoria direccionado por el registro "SP" +2.

**CODIGO MAQUINA 171**

**MEMBERSHIP**

El programa en Basic que sirve de ejemplo a esta rutina es el PROGRAMA 7-2. Las líneas 20 y 40 cargan la rutina en memoria (como verá, los DATA se van haciendo cada vez más largos). Las 100 a 120 llenan la pantalla de asteriscos. En 200 se nos pide un número entre "1" y "3" que in-

Animamos al lector a que utilice estas rutinas en sus propios programas, e incluso

a que las modifique o cons-  
truya otras similares; la mejor  
forma de aprender es practi-  
cando.

Gom lo visto hasta aquí, lle-  
gamos al final de este capítu-  
lo, sólo nos queda recomen-  
darle que resuelva los si-  
guientes ejercicios.

1.1.- Ensamble la instrucción "JP LABEL" sabiendo que la etiqueta "LABEL" está colocada delante de una instrucción que se ensamblará en la dirección 57538.

2.- Ensamble la instrucción "JR LABEL", situada en la dirección 62537, sabiendo que la etiqueta "LABEL" se encuentra situada delante de una instrucción que está ensamblada en la dirección 62493.

3.- Compruebe para qué valores de "A" se efectúa el salto en la siguiente rutina:

4.- Queremos saltar a una u otra dirección en función del contenido de "A" que puede ser desde "0" hasta "5". Escriba la rutina, sabiendo que las direcciones han de ser las siguientes:

A	Dirección
1	753Ah
2	8000h
3	BF5Ah
4	BF7Ch
5	C005h



# SOLUCIONES A LOS EJERCICIOS

1.- La solución es: C3h, C2h, E0h; ó bien: 195, 194, 224; (224+256+194=57538).

2.- La solución es: 18h, D2h; ó bien: 24, 210; (62493-62539=-46 y -46 equivale a 210 en complemento a 2).

3.- El salto se efectuará siempre que, en la comparación, no haya habido acarreo, es decir, cuando "A" contenga un número mayor o igual que 37. Solución: "A" >= 37\*.

4.- La rutina podría ser:

```

100 EJER_4 ADD A,A
110 LD B,0
120 LD C,A
130 LD HL,TABLA
140 ADD HL,BC
150 LD C,(HL)
160 INC HL
170 LD B,(HL)
180 LD H,B
190 LD L,C
200 JP (HL)
210 TABLA DEFH #753A
220 DEFH #8000
230 DEFH #BF5A
240 DEFH #BF7C
250 DEFH #C005
    
```

## INSTRUCCIONES DE INTERCAMBIO, TRANSFERENCIA Y BUSQUEDA

### Grupo de instrucciones de intercambio

EX: "EXchange", en inglés: intercambio.

Estas instrucciones consisten básicamente en el intercambio de los valores de los campos dados, de tal forma que, si A vale X y B vale Y, después de ejecutarse la instrucción A valga Y y B valga X.

Uno de los usos más importantes de estas instrucciones es el almacenamiento temporal del valor de un registro para poder utilizarlo, pero permitiendo su posterior recuperación.

Por ejemplo, supongamos que interesa el valor del registro A, pero se necesita ejecutar una instrucción aritmética que lo usa; lo más rápido es intercambiar el valor del registro A con otro que sea posible, ejecutar la instrucción aritmética y volver a intercambiar el registro A.

El formato básico es:

EX OPERANDO, OPERANDO

donde los operandos indican los registros que intercambian sus valores.

EX DE,HL

OBJETO:

Cambia el contenido del par de registros "DE", por el contenido del par de registros "HL" y el contenido del par de registros "HL", por el contenido del par de registros "DE".

CODIGO MAQUINA:

11101011 Ebh

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

EX DE,HL

Contenido del par de registros "DE"

(D): 10011111 9fh  
(E): 10000011 83h

Contenido del par de registros "HL"

(H): 00011001 19h  
(L): 10100010 A2h

Instrucción

EX DE,HL 11101011 Ebh

Contenido del par de registros "DE" después de la ejecución

(D): 00011001 19h  
(E): 10100010 A2h

Contenido del par de registros "HL" después de la ejecución

(H): 10011111 9fh  
(L): 10000011 83h

EX AF,AF

OBJETO:

Cambia el contenido del par de registros "AF", por el contenido de sus alternativos "AF" y el contenido del par de registros alternativos "AF", por el contenido del par de registros "AF".

CODIGO DE MAQUINA:

00001000 00h

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

Observe que, dado que los indicadores de condición están en el registro "F", después de ejecutarse esta instrucción quedan activos los indicadores



(B):	0	1	1	0	1	0	0	6Ch
(C):	0	0	0	1	0	0	0	10h

Indicadores de condición después de la ejecución:

S	Z	H	P/V	N	C
0	1	x	0	x	1
1	1	x	1	1	x

El octeto direccionado por "HL" contenía el mismo valor que el acumulador, por lo que el indicador "Z" se ha puesto a "1". Por otro lado, el registro "BC" permanece distinto de cero tras decrementarlo, por lo cual, el indicador "P/V" se pone también a "1".

## CPIR

### OBJETO:

Compara el contenido del registro acumulador con el octeto de memoria direccionado por el contenido del par de registros "HL". Si la comparación es verdadera se activará el indicador de condición "Z". A continuación incrementa en 1 el par de registros "HL" y decrementa en 1 el par de registros "BC". Si el resultado de la comparación es falso y el contenido del par de registros "BC" no es cero se repite la instrucción. La instrucción termina cuando el par de registros "BC" alcanza el valor cero o el resultado de la comparación es verdadero. Ver FIGURA 8-4.

Tenga en cuenta que si el par de registros "BC" es cero antes de la ejecución de la instrucción esta se repetirá para 64 K octetos salvo que encuentre un resultado ver-

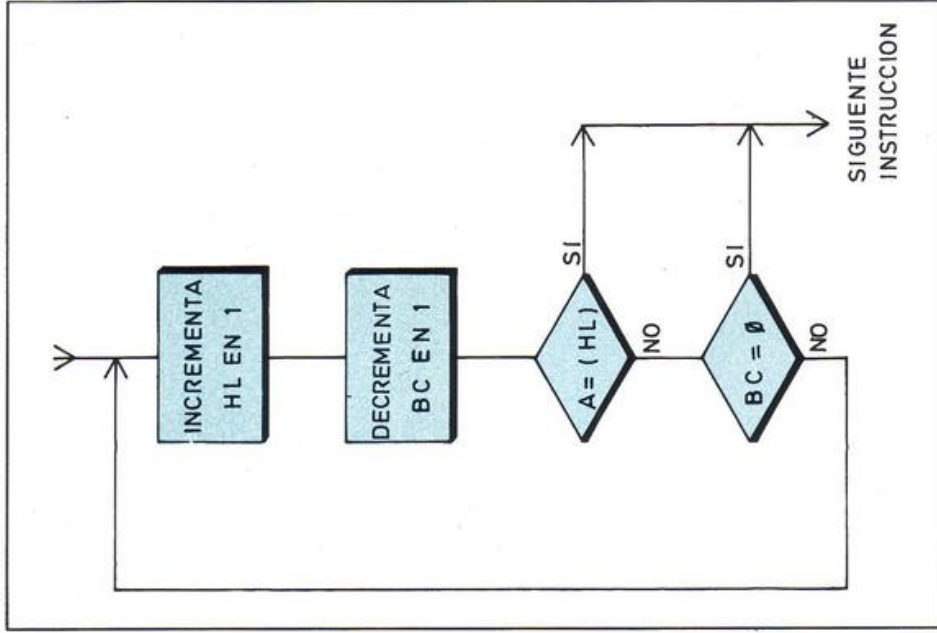


Fig. 8-4: Organigrama de la instrucción CPIR.

dadero. Esto es debido a que primero decrementa y luego compara, al decrementar 1 a 0000h, en el par de registros "BC" quedaría el valor FFFFh. Las interrupciones no paran la ejecución de esta instrucción por lo que se atenderían cuando terminase.

### CODIGO DE MAQUINA:

11101101	EDh
10110001	81h

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

Z; pone 1 - si el registro "A" es igual al octeto  
pone 0 - en cualquier otro caso

H; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso

## Grupo de instrucciones de transferencia

Supongamos que queremos transferir todo un bloque de memoria desde una zona hasta otra; por ejemplo, podría interesarnos transferir toda la pantalla a una dirección de memoria más alta (digamos, a partir de 40000) para guardar allí una copia segura.

El inicio de la pantalla es 16384 (4000h) y su longitud, con atributos incluidos, es 6912 bytes (1B00h). En Basic podríamos hacer:

```

10 LET inic=16384
20 LET dest=40000
30 LET cont=6912
40 POKE dest,PEEK inic
50 LET inic=inic+1
60 LET dest=dest+1
70 LET cont=cont-1
80 IF cont<>0 THEN GO TO 40
  
```

Este bucle tardaría una eternidad en ejecutarse y ocupa una cantidad ingente de memoria. Esta es una de las ocasiones en las que, o recurrimos al código máquina, o estamos perdidos. Vamos a ver cómo sería esta rutina en Assembler:

```

100 LD HL,16384
110 LD DE,40000
120 LD BC,6912
130 BUCLD LD A,(HL)
140 LD (DE),A
150 INC HL
160 INC DE
170 DEC BC
180 LD A,B
190 OR C
200 JR NZ,80UCLE
  
```

Esto ya es algo más racional, se ejecuta en un "abrir y cerrar de ojos" y ocupa bastante poca memoria. Hemos usado "HL" como puntero para movernos por el bloque "origen", "DE" para movernos por el "destino" y "BC" como contador de bytes a transferir. En cada pasada del bucle, hacemos la transferencia a través del registro "A", incrementamos los punteros y decrementamos el contador; si esto no es "cero", repetimos el bucle.

Es tan frecuente realizar transferencias de bloques en código máquina, que el microprocesador Z-80 posee una serie de instrucciones que nos van a ahorrar parte del trabajo en bucles de este tipo. Estas instrucciones se van a encargar de hacer todo lo que nosotros hacemos en las líneas 130 a 200; pero sin tocar el acumulador, es decir, transferir, ajustar punteros, decrementar el contador e, incluso, iterar el bucle. Son lo que se denomina: Instrucciones de transferencia.

En todas estas instrucciones el código mnemotécnico comienza por LD del inglés "load", cargar. La finalidad de estas instrucciones es transferir datos en memoria usando los pares de registros "HL" y "DE" como punteros; "HL" apuntará siempre a la primera o última dirección del bloque origen, "DE" lo mismo para el bloque destino y "BC" será el contador de bytes; como registra nemotécnica para no olvidar esto, puede asociar "DE" con la palabra "Destino" y "BC" con "Bytes Counter" (en inglés: Contador de Bytes).

El uso más frecuente es mover campos de memoria, bien sean éstos numéricos o

literales, evitando el paso por registros de CPU. El límite de longitud de estos campos lo marca la memoria disponible, teóricamente es de 64 K octetos.

El formato básico es el código mnemotécnico, que en todas ellas es distinto. Estas instrucciones no tienen operandos, ya que el direccionamiento de los mismos está implícito en el código de operación.

## LDI

### OBJETO:

Transfiere un octeto de memoria desde la posición direccionada por el par de registros "HL" a la posición direccionada por el par de registros "DE". A continuación incrementa 1 en ambos pares de registros y decrementa en 1 el par de registros "BC".

Equivaldría al siguiente programa:

```

100 LD (DE), (HL)
110 INC HL
120 INC DE
130 DEC BC
  
```

(Evidentemente, la instrucción de la línea 100 no existe, pero es una forma de ver que "LDI" transfiera el dato sin pasar por el acumulador).

### CODIGO DE MAQUINA:

11101101	EDh
10100000	A0h

H; pone 0 - siempre  
N; pone 0 - siempre



P/V; pone 1 - si BC-1 es diferente de cero  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

16

EJEMPLO:

LDI

Contenido del par de registros "HL"

(H): 01010110  
(L): 00001010

Contenido del par de registro "DE"

(D): 01111101  
(E): 10010010

Contenido del octeto de memoria 560Ah

560Ah: 10101010 A4h

El contenido del octeto de memoria 7D92h no es significativo.

Contenido del par de registros "BC"

(B): 00000000  
(C): 00000101

Instrucción

LDI: 11101101  
10100000 EDh ADh

Contenido del octeto de memoria 7D92h después de la ejecución

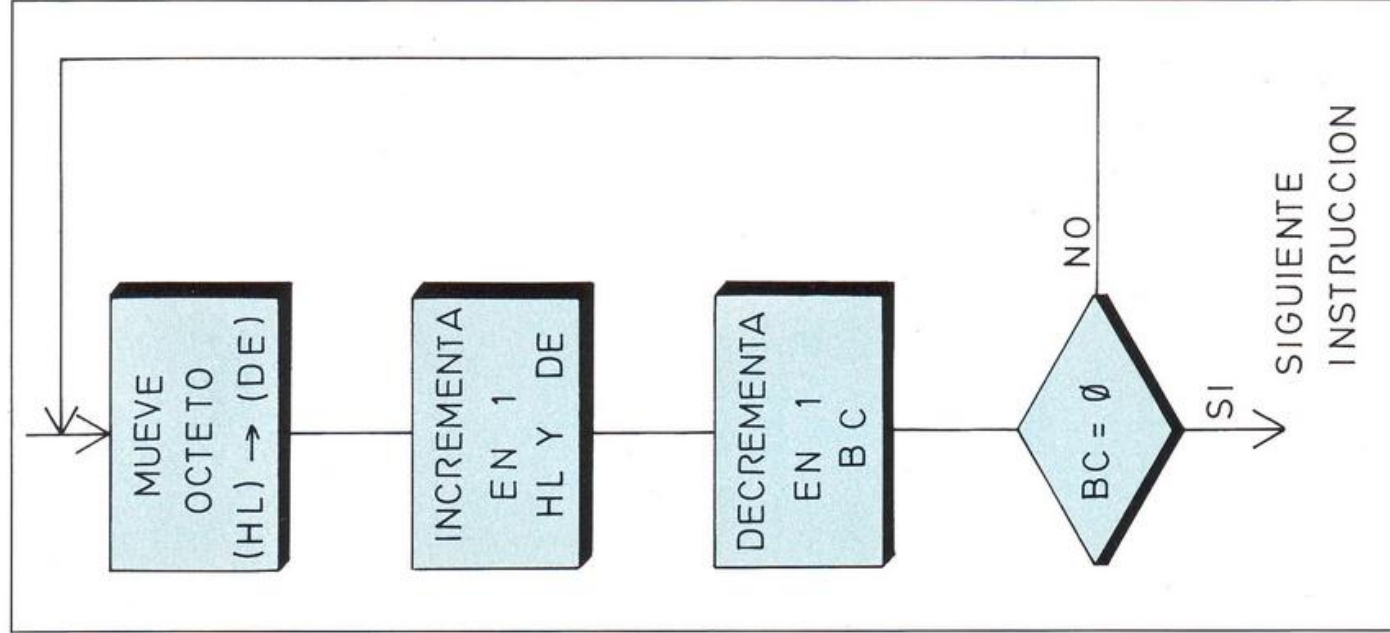


Fig. 8-2: Organigrama de la instrucción LDIR.

Lo que realmente hacen estas instrucciones, al igual que las ya vistas de comparar (CP), es una resta entre un octeto de memoria (direccional) por el contenido del par de registros "HL" y el registro acumulador, sin alterar ninguno de los dos. Por lo tanto los indicadores de condición se activarán según las reglas de las instrucciones de restar (SUB). Emplearemos a pesar de todo el término comparar, pues lo que realmente se pretende es encontrar un valor igual a uno dado y si se encuentra se activará el indicador de condición "Z" (cero).

Para más aclaraciones, si son necesarias, ver las explicaciones de las instrucciones CP y SUB.

Con la única diferencia de que el indicador "P/V" se pondrá a "0" si "BC" alcanza un valor cero al decrementarlo y se pondrá a "1" si "BC" se mantiene distinto de cero.

CODIGO DE MAQUINA:

EDh 11101101  
A1h 10100001

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso

Z; pone 1 - si el registro "A" es igual al octeto  
pone 0 - en cualquier otro caso

H; pone 1 - si no hay acarreo desde el bit 3  
pone 0 - en cualquier otro caso

N; pone 1 - siempre  
P/V; pone 1 - si BC-1 es diferente de cero  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

16

EJEMPLO:

CPI

Contenido del registro acumulador:

(A): 01110111 77h

Contenido del par de registros "HL":

(H): 10010110 96h  
(L): 00110111 37h

Contenido del octeto de memoria 9637h:

9637h: 01110111 77h

Contenido del par de registros "BC":

(B): 01101100 60h  
(C): 00010001 11h

Instrucción:

CPI: 11101101 EDh  
10100001 A1h

El contenido del registro "A" no ha variado con la ejecución.

Contenido del par de registros "HL" después de la ejecución:

(H): 10010110 96h  
(L): 00110000 30h

El contenido de la posición de memoria 9636h no ha variado con la ejecución.

Contenido del par de registros "BC" después de la ejecución:



Transfiere un octeto de memoria desde la posición direccionada por el par de registros "HL" a la posición direccionada por el par de registros "DE". Después decrementa 1 los pares de registros "HL", "DE" y "BC". A continuación, compara el par de registros "BC" con cero; y si no lo es repite la instrucción. Cuando el par de registros "BC" alcanza el valor cero se pasa a ejecutar la siguiente instrucción. Ver FIGURA 8-3.

Tenga en cuenta que si el par de registros "BC" es cero antes de la ejecución de la instrucción ésta se repetirá para 64K octetos. Esto es debido a que primero decrementa y luego compara, al decrementar 1 a 0000h, en el par de registros "BC" quedará el valor FFFFh.

Las interrupciones no paran la ejecución de esta instrucción por lo que se atenderán cuando terminase.

#### CODIGO DE MAQUINA:

1 1 1 0 1 1 0 1	Edh
1 0 1 1 1 0 0 0	B8h

INDICADORES DE CONDICION QUE AFECTA:

H; pone 0 - siempre  
N; pone 0 - siempre  
P/V; pone 0 - siempre

#### CICLOS DE MEMORIA:

Si "BC" diferente de 0  
5

Si "BC" igual a cero  
4

#### CICLOS DE RELOJ:

Si "BC" diferente de cero  
21

Si "BC" igual a cero  
16

#### EJEMPLO:

LDIR
------

Contenido del par de registros "HL"

(H):	0 1 1 1 0 1 0 0	74h
(L):	1 0 1 1 1 0 0 1	B9h

Contenido del par de registros "DE"

(D):	1 0 0 0 0 0 1 1	83h
(E):	0 0 0 1 0 1 1 1	17h

Contenido del par de registros "BC"

(B):	0 0 0 0 0 0 0 0	00h
(C):	0 0 0 0 0 1 0 1	05h

Contenido de las 5 posiciones de memoria anteriores a 74B9h

0 1 1 1 0 1 1 1	77h
0 1 1 0 0 1 1 0	66h
0 1 0 1 0 1 0 1	55h
0 1 0 0 0 1 0 0	44h
0 0 1 1 0 0 1 1	33h

El contenido de las 5 posiciones de memoria anteriores a 8317h no es significativo.

#### Instrucción

LDIR	1 1 1 0 1 1 0 1	Edh
	1 0 1 1 1 0 0 0	B8h

Contenido del par de registros "HL" después de la ejecución

(H):	0 1 1 1 0 1 0 0	74h
(L):	1 0 1 1 0 1 0 0	B4h

Contenido del par de registros "DE" después de la ejecución:

(D):	1 0 0 0 0 0 1 1	83h
(E):	0 0 0 1 0 0 1 0	12h

Contenido del par de registros "BC" después de la ejecución:

(B):	0 0 0 0 0 0 0 0	00h
(C):	0 0 0 0 0 0 0 0	00h

Contenido de las 5 posiciones de memoria anteriores a 8317h después de la ejecución:

0 1 1 1 0 1 1 1	77h
0 1 1 0 0 1 1 0	66h
0 1 0 1 0 1 0 1	55h
1 1 0 0 0 1 0 0	44h
0 1 1 1 0 0 1 1	33h

El contenido de las 5 posiciones de memoria anteriores a 47BCh no ha variado después de la ejecución.

Indicadores de condición después de la ejecución:

S	Z	H	P/V	N	C	
x	x	x	0	x	0	x

Es importante tener en cuenta que, de estas instrucciones, se sale siempre con el registro "BC" a cero y los registros "HL" y "DE" apuntando a la dirección siguiente o anterior a cada uno de los bloques afectados.

#### Grupo de instrucciones de búsqueda

Las instrucciones de búsqueda tienen por objeto buscar en una tabla o posición de memoria un valor igual a uno dado.

7092h: 1 0 1 0 1 0 1 0 AAh

El contenido del octeto de memoria 560Ah no ha variado con la ejecución.

Contenido del par de registros "HL" después de la ejecución

(H):	0 1 0 1 0 1 1 0	56h
(L):	1 0 0 1 0 0 1 1	93h

Contenido del par de registros "DE" después de la ejecución

(D):	0 1 1 1 1 1 0 1	70h
(E):	1 0 0 1 0 0 1 1	93h

Contenido del par de registros "BC" después de la ejecución

(B):	0 0 0 0 0 0 0 0	00h
(C):	0 0 0 0 0 1 0 0	04h

Indicados de condición después de la ejecución

S	Z	H	P/V	N	C		
x	x	x	0	x	1	0	x

#### LDIR

#### OBJETO:

Transfiere un octeto de memoria desde la posición direccionada por el par de registros "HL" a la posición direccionada por el par de registros "DE". Después incrementa 1 en ambos pares de registros y decrementa en 1 el par de registros "BC". A continuación, comprueba si el par de registros "BC" vale cero; y si no, repite la instrucción.

igual a cero, que gasta menos. Esto es debido a que la repetición se produce decrementando en 2 el registro contador de programa "PC".

#### EJEMPLO:

LDIR
------

Contenido del par de registros "HL"

(H):	0 1 0 0 0 1 1 1	47h
(L):	1 0 1 1 1 1 0 0	80h

Contenido del par de registros "DE"

(D):	1 0 0 0 1 0 0 0	88h
(E):	0 1 1 0 0 0 1 1	63h

Contenido del par de registros "BC"

(B):	0 0 0 0 0 0 0 0	00h
(C):	0 0 0 0 0 0 1 1	07h

Contenido de las 7 posiciones de memoria a partir de 47BCh

47BCh:	0 0 0 0 0 0 0 1	01h
	0 0 0 0 0 0 1 0	02h
	0 0 0 0 0 0 1 1	03h
	0 0 0 0 0 1 0 0	04h
	0 0 0 0 0 1 0 1	05h
	0 0 0 0 0 1 1 0	06h
	0 0 0 0 0 1 1 1	07h

El contenido de las 7 posiciones de memoria a partir de 8863h no es significativo.

#### Instrucción

LDIR:	1 1 1 0 1 1 0 1	Edh
	1 0 1 1 0 0 0 0	B8h



Contenido del par de registros "HL" después de la ejecución

(H):	01000111	47h
(L):	11000111	C3h

Contenido del par de registros "DE" después de la ejecución

(D):	10001000	88h
(E):	01101010	6Ah

Contenido del par de registros "BC" después de la ejecución

(B):	00000000	00h
(C):	00000000	00h

Contenido de las 7 posiciones de memoria a partir de 8863h después de la ejecución

8863h:	00000001	01h
	00000010	02h
	00000011	03h
	00000100	04h
	00000101	05h
	00000110	06h
	00000111	07h

El contenido de las 7 posiciones de memoria a partir de 47BCh no ha variado después de la ejecución.  
Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
x	x	x	0	0	x

El mnemónico "LDI" es abreviatura del inglés: "Load and Increment" (Carga e incremento). "LDIR" es abreviatura

tura de: "Load, Increment and Repeat" (Carga, incremento y repite).

Estas instrucciones permiten realizar transferencias de bloques, moviendo los punteros desde el inicio del bloque al final, es decir, desde la dirección más baja de cada bloque, hacia la más alta. Esto funciona perfectamente; incluso, si los dos bloques se solapan (tienen algunas direcciones en común), siempre y cuando, la dirección inicial del destino sea más alta que la del origen.

Pero ¿qué pasa si los bloques se solapan y el destino está más bajo que el origen? En este caso, el proceso de transferencia corrompería el bloque transferido (si lo representa gráficamente, lo verá con claridad). En este caso, sería útil disponer de instrucciones que hicieran lo mismo que "LDI" y "LDIR", pero moviendo los punteros desde el final de cada bloque hacia el principio, es decir, decrementándolos.

Como en el Z-80 todo está previsto, disponemos de estas instrucciones; se denominan: "LDD" (Load and Decrement) y "LDDR" (Load, Decrement and Repeat).

**LDD**

OBJETO:

Transfiere un octeto de memoria desde la posición direccionada por el par de registros "HL" a la posición direccionada por el par de registros "DE". A continuación

decrementa 1 los pares de registros "HL", "DE" y "BC".

CODIGO DE MAQUINA:

11101101	EDh
10101000	A8h

INDICADORES DE CONDICION QUE AFECTA:

H: pone 0 - siempre  
N: pone 0 - siempre  
P/V: pone 1 - si BC-1 es diferente de cero  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

16

EJEMPLO:

LDD	
-----	--

Contenido del par de registros "HL"

(H):	10000001	81h
(L):	10100110	A6h

Contenido del par de registros "DE"

(D):	10010010	92h
(E):	00000010	02h

Contenido del par de registros "BC"

(B):	00110110	36h
(C):	11001001	C9h

Contenido del octeto de memoria 81A6h

81A6h:	11111111	Ffh
--------	----------	-----

El contenido del octeto de memoria 9202h no es significativo.

Instrucción

LDD:	11101101	EDh
	10101000	A8h

Contenido del octeto de memoria 9202h después de la ejecución

9202h:	11111111	Ffh
--------	----------	-----

El contenido del octeto de memoria 81A6h no ha variado con la ejecución.

Contenido del par de registros "HL" después de la ejecución

(H):	10000000	81h
(L):	10100101	A5h

Contenido del par de registros "DE" después de la ejecución

(D):	10010010	92h
(E):	00000001	01h

Contenido del par de registros "BC" después de la ejecución

(B):	00110110	36h
(C):	11001000	C8h

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
x	x	x	0	1	0

**LDDR**

OBJETO:

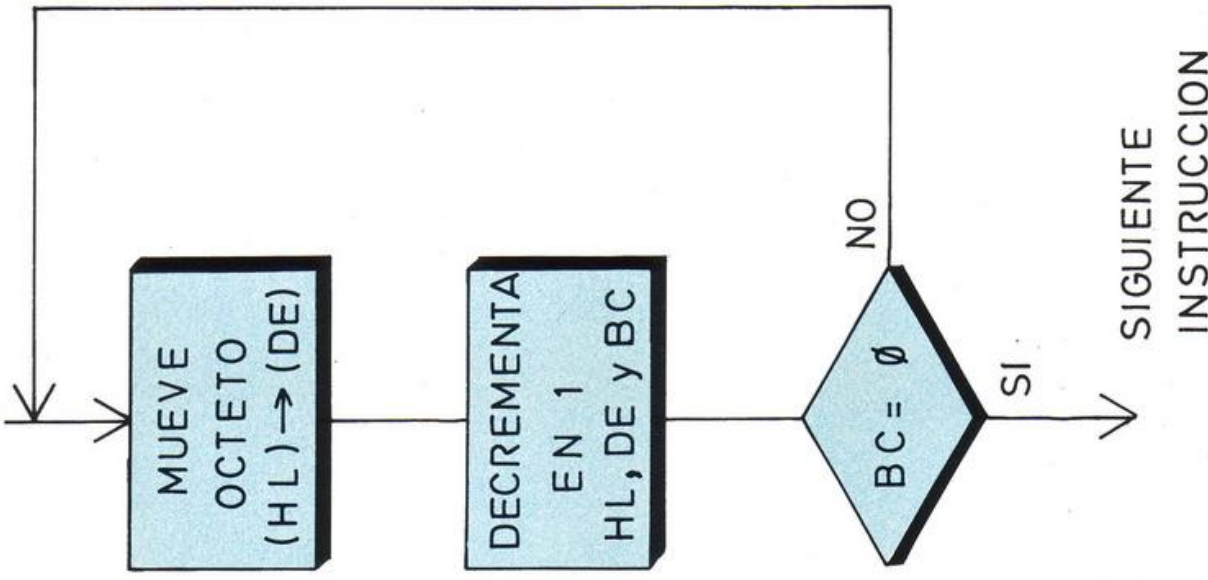


Fig. 8-3: Organigrama de la instrucción LDDR.



mido el «XOR A» que había en la línea 420 y, en vez de hacer «LD (HL),A» hacemos «LD (HL),L». Hemos simplificado el listado (este tiene dos líneas menos) y, además, funciona más deprisa. Aún podría simplificarse más; pues todo que todos los octetos menos significativos son «ceros», podríamos almacenar en la tabla sólo los más significativos y cargar «L» y «E» con «0». En su día no se hizo así porque se pretendía que el ejemplo ilustrara, también, la forma de utilizar una tabla de «offset».

La rutina «CLS3» del capítulo 7 quedaría:

```
100 ORG 60000
110 LD A, (23681)
120 AND A
130 JR Z, ERROR
140 CP 4
150 JR C, CLS3
160 ERROR RST 8
170 DEFB #0A
180 CLS3 DEC A
190 ADD A, A
210 LD HL, TABLA
220 LD B, 0
230 LD C, A
240 ADC HL, BC
250 LD B, (HL)
260 INC HL
270 LD D, (HL)
280 LD E, 0
320 JR CLS3_1
330 TABLA DEFB #40, #58
340 DEFB #48, #59
350 DEFB #50, #5A
390 CLS3_1 LD H, B
400 LD L, 0
410 .....
```

A partir de la línea 410 continúa como está arriba. Hemos intentado mantener la numeración en lo que fuera posible, pero al desaparecer líneas, quedan huecos vacíos entre los números. Esta rutina ocupa menos bytes que la original y, además, es más rápida. ¿Qué más se puede pedir?

Ahora sí, empezamos con los ejemplos específicos de este capítulo. El primero de ellos sirve para intercambiar entre sí dos de los tres bloques de pantalla (los que borra la rutina anterior). Existen tres posibilidades: se puede intercambiar el primer bloque con el segundo, el segundo con el tercero o el primero con el tercero (nótese que intercambiar, por ejemplo, el segundo bloque con el primero es lo mismo que intercambiar el primero con el segundo, por tanto, en los tres casos expuestos están todas las posibilidades). Los bloques que se intercambiarán dependerán del contenido del acumulador que podrá ser «4», «5» ó «6». Se han elegido estos valores para que esta rutina sea compatible con la anterior; veamos un cuadro resumido:

```
A=4 => 1<->2
A=5 => 2<->3
A=6 => 1<->3
```

La idea es que, mezclando estas rutinas y otras que veremos, se pueda llegar a hacer un auténtico «procesador de pantalla» que haga un gran número de operaciones dependiendo del valor de «A». Llamamos archivo de pantalla a los bytes cuyas direc-

ciones están comprendidas entre 4000h (16384) y 57FFh (22527) ambos inclusive y que contienen los 49152 pixels que componen una imagen. Asimismo, llamamos archivo de atributos a los bytes con direcciones comprendidas entre 5800h (22528) y 5AFFh (23295) que contienen los atributos de color, brillo y parpadeo para cada grupo de 64 pixels.

Podemos dividir cada uno de estos archivos en tres zonas iguales; cada una de ellas contendrá 2048 bytes en el caso del fichero de pantalla y 256 bytes en el de atributos.

Intercambiar entre sí dos de estas zonas implica que los datos contenidos en cada una de sus direcciones de memoria se intercambian entre sí, y esto ocurre tanto para el archivo de pantalla como para el de atributos. Necesitaremos, por tanto, dos bucles; uno que intercambie los bytes del fichero de pantalla y otro que haga lo mismo con los bytes correspondientes del fichero de atributos. Para no andar manejando números engorrosos, asignaremos una etiqueta para la dirección de comienzo de cada una de estas zonas; las etiquetas serán «DIS\_1», «DIS\_2» y «DIS\_3» para el archivo de pantalla y «ATT\_1», «ATT\_2» y «ATT\_3» para el de atributos; sus direcciones (según se definen en el programa) serán:

```
660 DIS_1 EQU #4000
670 DIS_2 EQU #4800
680 DIS_3 EQU #5000
690 ATT_1 EQU #5800
700 ATT_2 EQU #5900
710 ATT_3 EQU #5A00
```

PV; pone 1 - si BC-1 es diferente de cero  
PV; pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

Si «BC» diferente de 0 y «A» diferente del octeto

5  
Si «BC» igual a 0 o «A» igual al octeto  
4

CICLOS DE RELOJ:

Si «BC» diferente de 0 y «A» diferente del octeto

21  
Si «BC» igual a 0 o «A» igual al octeto

EJEMPLO:

CPDR

Contenido del acumulador (octeto buscado):

0A: 1 0 0 0 0 0 1 0 82h

Contenido del par de registros «HL»:

0H: 1 1 1 0 0 0 0 0 E0  
0L: 1 0 0 0 0 0 0 0 80h

Contenido del par de registros «BC»:

0B: 0 0 0 0 0 0 0 0 00h  
0C: 0 0 0 0 0 1 1 1 07h

Contenido de las 7 posiciones de memoria a partir de E080h:

E080h 01111111 7Eh  
01111111 7Fh  
10000000 80h  
10000001 81h  
10000010 82h  
10000011 83h  
10000100 84h

Instrucción

CPDR: 1 1 1 0 1 1 0 1 E0h  
1 0 1 1 0 0 0 1 81h

El contenido del registro «A» no ha variado con la ejecución.

Contenido del par de registros «HL» después de la ejecución:

0H: 1 1 1 0 0 0 0 0 E0h  
0L: 1 0 0 0 1 0 0 0 84h

Contenido del par de registros «BC» después de la ejecución:

0B: 0 0 0 0 0 0 0 0 00h  
0C: 0 0 0 0 0 1 1 1 03h

El contenido de las 7 posiciones de memoria a partir de E080h no ha variado después de la ejecución.

Indicadores de condición después de la ejecución:

S Z H P V N C

0 1 x 0 x 1 1 x

Se ha encontrado un octeto igual al contenido del acumulador en la posición de memoria E084h, por lo que la ejecución se ha detenido en este punto; «Z» está a «1» para indicar que se ha encontrado el octeto; «PV» está a «1» porque «BC» no ha llegado a valer cero; «HL» contiene la dirección del octeto cuyo contenido es igual al del acumulador.

CPD

OBJETO:

Compara el contenido del

registro acumulador con el octeto de memoria direccionado por el contenido par de registros «HL». Si la comparación, es verdadera se activará el indicador de condición «Z». A continuación decrementa en 1 los pares de registros «HL» y «BC».

Funciona igual que CPI pero realizando la búsqueda desde el final del bloque hacia el principio.

CODIGO DE MAQUINA:

1 1 1 0 1 1 0 1 E0h  
1 0 1 1 0 0 0 1 A0h

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo

S; pone 0 - en cualquier otro caso

Z; pone 1 - si el registro «A» es igual al octeto

Z; 0 - en cualquier otro caso

H; pone 1 - si no hay acarreo desde el bit 3

H; pone 0 - en cualquier otro caso

N; pone 1 - siempre

PV; pone 1 - si BC-1 es diferente de cero

PV; pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

16

EJEMPLO:

CPD

Contenido del registro acumulador:



(A): 00100000 20h

Contenido del par de registros «HL»:

(H): 10111001 B9h  
(L): 10001000 88h

Contenido del octeto de memoria B988h:

B988h: 00000010 02h

Contenido del par de registros «BC»:

(B): 00000000 00h  
(C): 00000000 00h

Instrucción:

CPD: 11101101 Edh  
10101001 A8h

El contenido del registro «A» no ha variado con la ejecución.

Contenido del par de registros «HL» después de la ejecución:

(H): 10111001 B9h  
(L): 10001111 87h

El contenido de la posición de memoria B988h no ha variado con la ejecución.

Contenido del par de registros «BC» después de la ejecución:

(B): 11111111 Ffh  
(C): 11111111 Ffh

Indicadores de condición después de la ejecución:

S Z H P/V N C

00x0x01x

el resultado de la comparación es verdadero. Ver Figura 8-5.

Tenga en cuenta que si el par de registros «BC» es cero antes de la ejecución de la instrucción esta se repetirá para 64K octetos salvo que encuentre un resultado verdadero. Esto es debido a que primero decrementa y luego compara, al decrementar 1 a 0000h, en el par de registros «BC» quedaría el valor FFFh.

Las interrupciones no paran la ejecución de esta instrucción por lo que se atenderían cuando terminase.

CODIGO DE MAQUINA:

11101101 Edh  
10101001 B9h

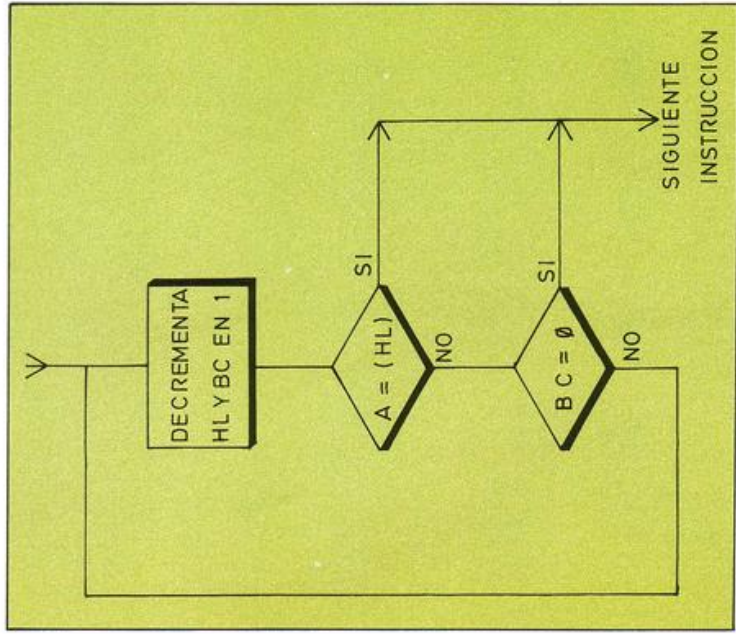


Fig. 8-5. Organigrama de la instrucción CPDR.

ciones fundamentales, como las de rotación o subrutinas; no obstante, estamos ya en disposición de escribir rutinas que puedan pasar a formar parte de la «biblioteca» particular de cada lector.

Programar en Assembler no es tarea fácil o, cuando menos, resulta bastante trabajoso, por lo que, los programadores, han ideado métodos para no tener que repetir el mismo trabajo varias veces. Uno de esos métodos es lo que se denomina: «biblioteca de rutinas». Programando en Assembler, hay una serie de tareas que se repiten en casi todos los programas; por ejemplo, borrar la pantalla, leer una tecla, hacer una pausa de «x» segundos, imprimir un dato en pantalla, buscar un dato en memoria, etc. La mayoría de los programadores escriben estas rutinas una sola vez y las guardan para utilizarlas en sucesivos programas; de esta forma, acaban teniendo una enorme colección de rutinas ya escritas y, cuando tienen que escribir un programa, combinan unas cuantas de ellas y ya tienen escrito más de la mitad del programa. A esta colección de rutinas se le denomina «biblioteca»; es conveniente que la mayoría de ellas estén escritas de forma reubicable, ya que así, resultará más fácil combinarlas.

Al hacer el curso, se nos ocurrió que los ejemplos pudieran servir para que cada lector iniciara, con ellos, su biblioteca particular. En este capítulo hemos preparado tres ejemplos que seguramente utilizará muchas veces. El primero de ellos sirve para intercambiar entre sí bloques de pantalla, y pretende

«LDIR»; el primer byte será copiado en todos los restantes.

Este método puede ser utilizado cada vez que se quiera llenar una zona de memoria con un determinado octeto. Un ejercicio interesante para el lector sería modificar la rutina que borra la pantalla por trozos, de forma que lo haga utilizando «LDIR»; recuerde que cuando borre el fichero de atributos, no será un «0» lo que tendrá que cargar, sino el valor de los atributos permanentes en curso, que está almacenado en la variable del sistema «ATTR-P» (dirección 23693). Si modificamos las rutinas «BUC-1» y «BUC-2» de este ejemplo, quedarán así:

```
410 LD BC, #07FF
420 BUC_1 PUSH DE
430 LD (HL), L
440 LD D, H
450 LD E, 1
460 LDIR
470 BUC_2 POP HL
480 LD D, H
490 LD E, 1
500 LD A, (23693)
510 LD (HL), A
520 LD C, #FF
530 LDIR
540 RET
```

En la línea 420 guardamos el contenido de «DE» que es la dirección de inicio de la zona de atributos; luego, en la línea 470, la recuperamos cargándola en «HL». Dado que el octeto menos significativo de todas las direcciones que usamos es «00», hemos supri-



empezaríamos a buscar el 00h, es de suponer que muchos números comiencen por 00h, por lo tanto, es más fácil organizar una búsqueda hacia atrás buscando el número más significativo, en este caso el 72h; una vez encontrado se analiza si el que le sigue hacia atrás es F3h, 74h, 00h y 00h.

Es norma habitual y aconsejable que los campos numéricos se justifiquen a la derecha y los literales a la izquierda, quiere esto decir que si en 5 octetos se quiere almacenar el literal «PEPE», se haga.

P E P E  
00 00 74 F3 72

y si se quiere almacenar el número 74F372h se haga,

Una vez esto claro se podría aplicar como norma general a practicar que la búsqueda de una cadena literal se realice hacia delante y la búsqueda de una cadena numérica se realice hacia atrás. Ver organigrama de Figura 8-10.

El tipo de búsqueda visto hasta aquí es secuencial, es decir, se va mirando secuencialmente, bien sea hacia delante o hacia atrás unos octetos consecutivos de memoria.

Podría darse el caso de tener los datos organizados de forma que estén mezclados numéricos y literales además de diferentes tipos de información, eso sí, en una estructura previamente fijada. Por ejemplo, supongamos que se tiene información sobre libros y se coloca de la siguiente manera:

20 octetos-AUTOR  
20 octetos-TÍTULO  
15 octetos-ISBN  
4 octetos-PRECIO  
6 octetos-FECHA DE COMPRA  
1 octeto-CODIGO TEMÁTICO  
TOTAL 66 octetos

A esta estructura se la llama registro de datos, cada una de sus subdivisiones se llaman campos y una serie de registros forman un fichero, todo ello independientemente del soporte en el que estén (memoria, cinta o disco). Más adelante en el curso se hablará de forma de organizar datos, bases de datos, etc. pero de momento es bueno diferenciar campo, registro y fichero.

Continuemos con las instrucciones de búsqueda que es lo que nos ocupa. Supongamos que, en un fichero así organizado, queremos saber si existe y dónde está un libro llamado «MOMO».

Un método sería buscar en todo el fichero la cadena de caracteres «MOMO» pero este tiene dos problemas fundamentales:

1. Hay que rastrear todo el fichero conscientes de que en más de las dos terceras partes de él no existe la información referente al título.

2. Una vez encontrada la cadena de caracteres «MOMO» no estaríamos seguros de haber encontrado un autor o un título.

El método más válido sería el secuencial con índice, es decir, se buscaría sólo en los octetos del campo TÍTULO. Para lo cual una vez determinado el primer registro se buscaría en los 20 octetos restantes la cadena de caracteres «MOMO» y si no se encuentra se pasaría al siguiente registro.

terres «MOMO» y si no se encuentra se sumaría a la dirección actual el tamaño del registro de datos menos 20, en este caso 46 con lo cual se estaría direccionando el campo TÍTULO del siguiente registro.

Observe que la palabra registro tiene dos usos muy utilizados en informática y muy diferentes en su función. Uno es el ya más familiar registro de CPU y otro es el registro de datos. No siempre que se emplea esta palabra se especifica si es CPU o datos, pero el contexto y el tema lo aclararán suficientemente.

Hasta aquí, hemos hecho una primera aproximación al manejo de tablas de datos; existen otros métodos más complejos que permiten, incluso, manejar tablas en las que los datos no tengan todos la misma longitud; pero no es el momento de abrumar al lector entrando excesivamente en profundidades; en un capítulo posterior, cuando veamos las técnicas de programación, haremos un estudio exhaustivo con ejemplos de lo que son y cómo se crean las bases de datos.

### Ejemplos

El primer ejemplo que vemos en el curso, se limitaba a cargar el registro «BC» con un número y retornar, decíamos entonces que no parecía algo muy vistoso para lo que suelen evocar las palabras «código máquina»; a medida que hemos ido aprendiendo más instrucciones, nos ha sido posible aumentar el nivel de complejidad de nuestros ejemplos e ir haciendo cosas cada vez más útiles. Aún nos faltan por ver algunas instrucciones.

### INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo  
S; pone 0 - en cualquier otro caso  
Z; pone 1 - si el registro «A» es igual al octeto  
Z; pone 0 - en cualquier otro caso  
H; pone 1 - si no hay acarreo desde el bit 3  
H; pone 0 - en cualquier otro caso  
N; pone 1 - siempre  
PV; pone 1 - si BC-1 es diferente de cero  
PV; pone 0 - en cualquier otro caso

### CICLOS DE MEMORIA:

Si «BC» diferente de 0 y «A» diferente del octeto: 5  
Si «BC» igual a 0 o «A» igual al octeto: 4

### CICLOS DE RELOJ:

Si «BC» diferente de 0 y «A» diferente del octeto: 21  
Si «BC» igual a 0 ó «A» igual al octeto: 16

### EJEMPLO:

CPDR: 00000000 00h  
IBI: 00000000 00h  
ICI: 00000111 05h  
FFh  
11111111  
FFh  
10000100 84h  
00100011 23h

### INSTRUCCIONES DE INTERCAMBIO

Código Fuente	Hexadecimal	Decimal
EX DE,HL	EB	235
EX AF,AF'	08	8
EX (SP),HL	E3	227
EX (SP),IX	DD,E3	221,227
EX (SP),IY	FD,E3	253,227
EXX	D9	217

Fig. 8-6. Tabla de codificación para las instrucciones de intercambio.

### INSTRUCCIONES DE TRANSFERENCIA

Código Fuente	Hexadecimal	Decimal
LDD	ED,AB	237,168
LDDR	ED,B8	237,184
LDI	ED,A0	237,160
LDIR	ED,B0	237,176

Fig. 8-7. Tabla de codificación para las instrucciones de transferencia.

Contenido del par de registros «BC»:  
IBI: 00000000 00h  
ICI: 00000111 05h  
Instrucción: 11101101 EDh  
10111001 B8h  
Contenido de las 5 posiciones de memoria anteriores a 8423h:  
00110011 33h  
00110100 34h  
00110101 35h  
00110110 36h  
00110111 37h  
8423h: 00110111  
El contenido del registro «A» no ha variado con la ejecución.  
Contenido del par de registros «HL» después de la ejecución:  
HL: 11100000 B8h  
IL: 00011110 1Eh



Contenido del par de registros «BC» después de la ejecución:

IB:	00000000	00h
IC:	00000000	00h

El contenido de las 5 posiciones de memoria anteriores a E080h no han variado después de la ejecución.

Indicadores de condición después de la ejecución:

S Z H P V N C

0	0	x	0	x	0	1	x
---	---	---	---	---	---	---	---

En este caso, no se ha encontrado ningún octeto cuyo contenido sea igual al del acumulador, por tanto, la instrucción ha terminado cuando «BC» ha llegado a valer cero. El indicador «Z» está a «0» porque no se ha encontrado el octeto y el «PV» está, también, a «0» porque el registro «BC» ha llegado a valer cero.

#### Tablas de codificación

A continuación, vamos a ver las tablas que nos indican el código máquina que corresponde a cada instrucción; recuerde que éstas son las tablas que utilizaremos para ensamblar las rutinas «a mano».

En la Figura 8-6 tenemos la tabla correspondiente a las instrucciones de intercambio; en la Figura 8-7, la correspondiente a las de transferencia y, finalmente, en la Figura 8-8 tenemos la tabla correspondiente a las instrucciones de búsqueda.

Hemos añadido otra tabla en la Figura 8-9 que constituye un resumen de la forma en que estas instrucciones afectan a los indicadores, así como el número de bytes que

#### INSTRUCCIONES DE BÚSQUEDA

Código Fuente	Hexadecimal	Decimal
CPD	ED, A9	237, 169
CPDR	ED, B9	237, 185
CPI	ED, A1	237, 161
CPIR	ED, B1	237, 177

Fig. 8-8. Tabla de codificación para las instrucciones de búsqueda.

#### GRUPO DE INTERCAMBIO, TRANSFERENCIA Y BÚSQUEDA

NEMONICO	INDICADORES							No. DE BYTES	CICLOS MEM. REL.
	S	Z	x	H	P/V	N	C		
EX DE, HL	.	.	.	.	.	.	.	1	1 4
EX AF, AF'	.	.	.	.	.	.	.	1	1 4
EX (SP), HL	.	.	.	.	.	.	.	1	5 19
EX (SP), IX	.	.	.	.	.	.	.	2	6 23
EX (SP), IY	.	.	.	.	.	.	.	2	6 23
EXX	.	.	.	.	.	.	.	1	1 4
LDI	.	.	x	0	x	0	0	2	4 16
LDIR	.	.	x	0	x	0	0	2	5(4) 21(16)
LDD	.	.	x	0	x	0	0	2	4 16
LDDR	.	.	x	0	x	0	0	2	5(4) 21(16)
CFI	↑	↑	x	↑	x	↑	1	2	4 16
CFIR	↑	↑	x	↑	x	↑	1	2	5(4) 21(16)
CPD	↑	↑	x	↑	x	↑	1	2	4 16
CPDR	↑	↑	x	↑	x	↑	1	2	5(4) 21(16)

#### NOTAS:

- Los signos tienen el siguiente significado:
  - ↑: El indicador cambia de valor de acuerdo con el resultado de la instrucción.
  - x: El bit adquiere un estado indeterminado.
  - 0: El indicador no es afectado por la instrucción, por lo que conserva su anterior contenido.
  - 0': El indicador se pone siempre a cero.
  - 1': El indicador se pone siempre a uno.

Fig. 8-9. Tabla de indicadores y ciclos para las instrucciones de intercambio, transferencia y búsqueda.

ocupan y el número de ciclos de memoria y reloj que emplean.

#### Métodos de búsqueda

Las instrucciones de búsqueda se usan normalmente para localizar en parte o en la totalidad de la memoria un valor que se supone que existe. También se suelen usar, aunque menos, para asegurarse de que un determinado valor no existe.

Cuando el valor que se quiere buscar ocupa un octeto, la aplicación de las instrucciones es inmediata, y una vez ejecutada sabremos si dicho valor existe o no, en caso de que exista también sabremos dónde se encuentra.

El problema se complica cuando se quiere buscar un valor que ocupa más de un octeto. Por ejemplo en una tabla donde existen nombres propios, queremos buscar «PEPE». La forma de actuar en un caso como éste sería:

- Buscar primero una «P».
- Si no se encuentra no hay ningún «PEPE».
- Si se encuentra habrá que comparar si los siguientes caracteres son «E», «P» y «E», lo cual se puede hacer con la instrucción CP o bien con las de búsqueda que incrementan el registro que se usa como índice.

Si lo que se busca es un valor numérico que ocupa más de un octeto es de mayor utilidad emplear otro método, a saber: supongamos que se quiere buscar el número 000074F372h en una tabla que contiene números de 5 octetos como máximo. Si se emplea el método anterior

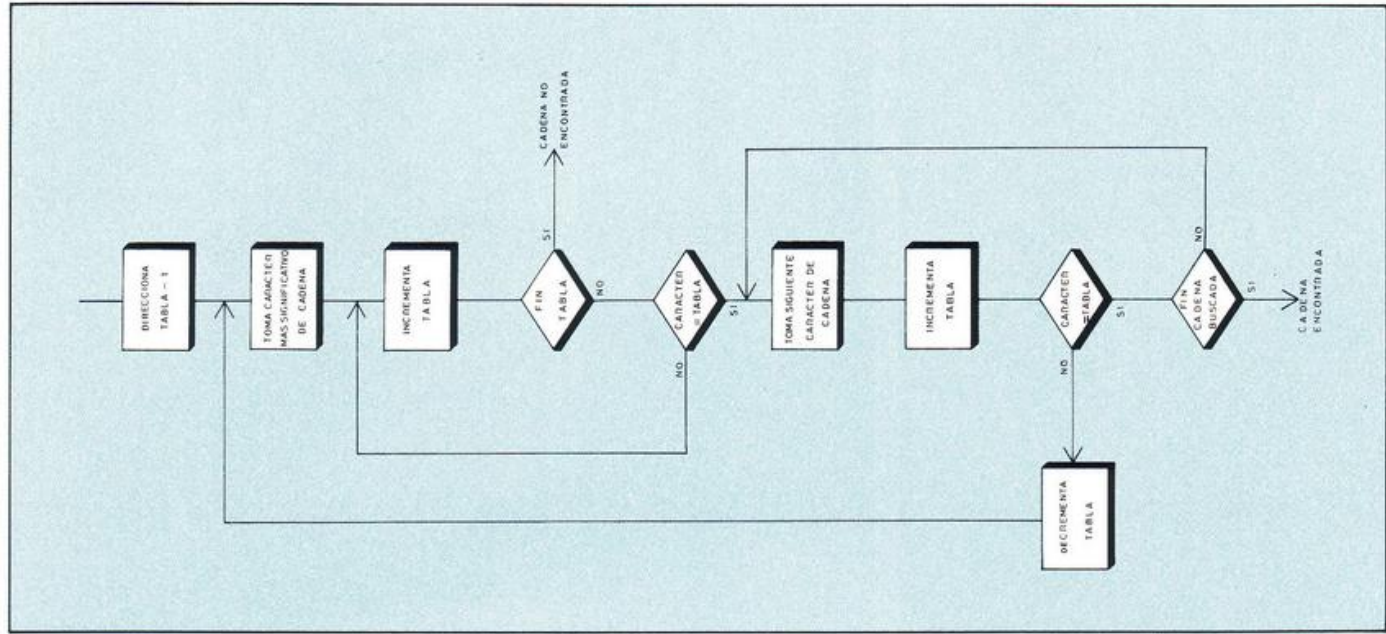


Fig. 8-10. Organigrama de una búsqueda en cadena.



la cual la cadena no puede incluir ningún 255.

Primero, rastreamos la zona indicada en busca del primer carácter de la cadena; si no lo encontramos, es que la cadena no existe, así que almacenamos un «0» en «BC» y retornamos. Si se encuentra el primer carácter, se van comparando los siguientes con cada uno de los que componen la cadena. Si se llega al final de ésta, se carga en «BC» la dirección donde se encontró el primer carácter y se retorna. Si, antes de llegar al final de la cadena, falla alguna de las comparaciones, se abandona este bucle y se sigue la búsqueda del primer carácter.

La rutina está escrita de forma que no requiere que toda la cadena se halle dentro de la zona rastreada, basta con que lo esté el primer carácter.

El trabajo más arduo es rastrear la zona buscando el primer carácter, este trabajo lo hemos encomendado a la instrucción «CPIR», por tanto, rastreamos la zona de abajo a arriba. Vayamos viendo el listado:

```

100 ORG 23350
110 LD HL, (23296)
120 LD BC, (23298)
130 CONT LD A, (23300)
140 CP 255
150 JR Z, FIN_1
160 CPIR

```

En 110 cargamos en «HL» el inicio de la zona y, en 120, cargamos en «BC» la longitud. La línea 130 lleva la etiqueta «CONT» porque es el si-

tio al que tendremos que volver cuando queramos continuar una búsqueda interrumpida; en esta línea, cargamos en «A» el primer carácter de la cadena. En 140 comprobamos si este carácter es 255 en cuyo caso, estaríamos buscando una cadena vacía; si es así, la línea 150 saltará a la etiqueta «FIN—Z» que retornaría con «BC» a «0» como si no se hubiera encontrado la cadena, ya que una cadena vacía no puede buscarse.

En este punto, tenemos en «HL» el inicio de la zona a rastrear, en «BC» su longitud y en «A» el carácter que estamos buscando; así que nada más apropiado que entrar en una instrucción «CPIR».

A la salida de esta instrucción, pueden ocurrir dos cosas: que se haya encontrado el carácter o que no se haya encontrado; en el primer caso, el indicador «Z» estará a «1» y, en el segundo caso, es-

```

170 JR NZ, FIN_2
180 PUSH HL
190 PUSH BC
200 LD BC, 23301
210 BUCLE LD A, (BC)
220 CP 255
230 JR Z, FIN
240 CP (HL)
250 JR Z, OK
260 POP BC
270 POP HL
280 JR CONT
290 OK INC HL
300 INC BC
310 JR BUCLE

```

En la línea 170 saltamos a «FIN—Z» si no se ha encontrado el carácter que buscábamos. Si se ha encontrado, preservamos los contenidos de «HL» y «BC» por si hay que seguir la búsqueda. En este momento, «HL» apuntará a la dirección siguiente a donde se ha encontrado el primer carácter de la cadena. Cargamos en «BC» 23301 que es la dirección del segundo carácter y entramos en un bucle donde vamos comparando cada carácter de la cadena con 255 para ver si ésta se ha acabado; si es así, saltamos a «FIN», si no, comparamos el carácter de la cadena con el correspondiente de la zona rastreada; si no son iguales, recuperamos los registros «HL» y «BC» y volvemos a «CONT» para continuar la búsqueda; si son iguales, incrementamos «HL» y «BC» y cerramos el bucle (subrutina «OK»).



Fig. 8-14. Listado completo de la rutina para transferir pantallas.

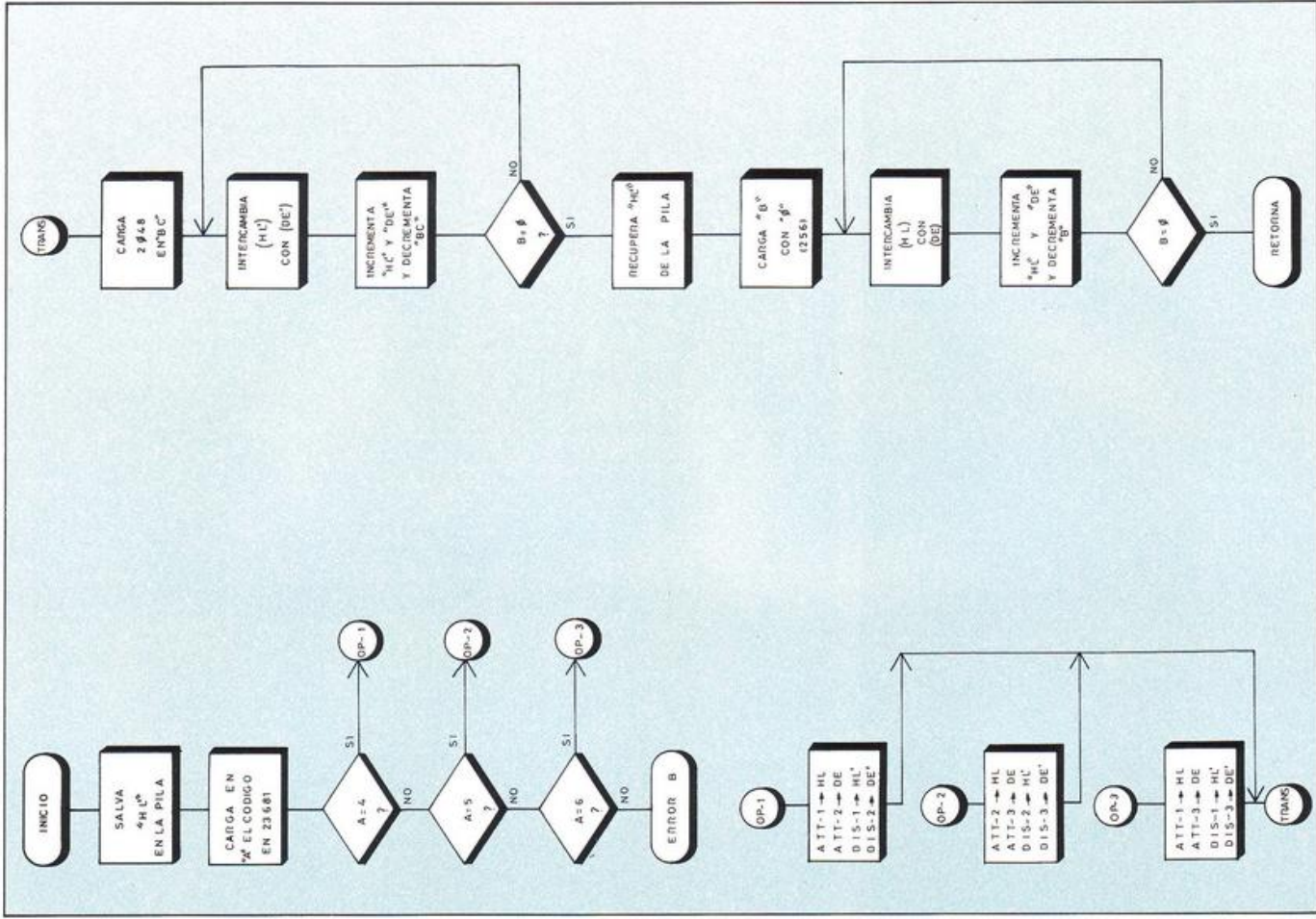


Fig. 8-11. Organigrama de la rutina para intercambiar zona de pantalla.



Esta definición de etiquetas formará parte de nuestro programa, de ahí los números de línea que están colocados antes de cada etiqueta.

Antes de seguir, haremos una pequeña observación de gran importancia cuando utilicemos las instrucciones «EX» y «EXX». Cuando programamos en código máquina el Spectrum, podemos utilizar todos los registros del microprocesador, pero algunos de ellos contienen datos importantes para el Sistema y, si alteramos su contenido, pueden ocurrir desastres cuando retornemos al Basic; estos registros son: «HL», «IX» e «IY». El primero es el registro «HL» del «SET» alternativo, este registro contiene la dirección del siguiente literal a ejecutar por el calculador (no se preocupe si no entiende, ya estudiaremos el calculador); no conviene alterar el contenido de este registro, por ello, es mejor preservarlo al principio de la rutina y recuperarlo al final, antes de retornar. El segundo, es el índice «IX», este registro contiene la dirección base para acceder a las variables del sistema mediante direccionamiento indexado; el contenido de este registro es reinicializado cuando se retorna con «RET», pero NO si se retorna con «RST 8»; por tanto, es preferible no tocarlo. El registro «IY» contiene el vector de página de interrupción (también las interrupciones serán estudiadas con más detalle); no es utilizado habitualmente por el sistema, pero a nosotros no nos sirve de mucho, así que es mejor no tocarlo a menos que, deliberadamente, queramos cambiar el vector de interrupción.

Todo esto es importante

porque, en esta rutina, vamos a utilizar los registros alternativos, así que guardaremos «HL» al principio de la rutina y lo recuperaremos al final, antes de retornar. En la Figura 8-11 tiene el organigrama de la rutina completa; primero se guarda «HL» haciendo lo siguiente:

```
110 EXX
120 PUSH HL
130 EXX
```

Después cargamos en «A» el contenido de la dirección 23681 y saltamos a las etiquetas «OP\_1», «OP\_2» u «OP\_3» según que el contenido de «A» sea «4», «5» o «6»; si fuera distinto de cualquiera de estos valores, retornamos mediante «RST 8» con el informe de error «B»:

```
140 LD A, (23681)
150 CP 4
160 JR Z, OP_1
170 CP 5
180 JR Z, OP_2
190 CP 6
200 JR Z, OP_3
210 RST 8
220 DEFB #0A
```

En «OP\_1», «OP\_2» u «OP\_3», cargamos en «HL», «DE», «HL» y «DE» las direcciones de inicio de los bloques a intercambiar que, serán distintas en cada caso; luego, la rutina continúa en «TRANS»:

```
230 OP_1 LD HL, ATT_1
240 LD DE, ATT_2
```

```
250 EXX
260 LD HL, DIS_1
270 LD DE, DIS_2
280 JR TRANS
290 OP_2 LD HL, ATT_2
300 LD DE, ATT_3
310 EXX
320 LD HL, DIS_2
330 LD DE, DIS_3
340 JR TRANS
350 OP_3 LD HL, ATT_1
360 LD DE, ATT_3
370 EXX
380 LD HL, DIS_1
390 LD DE, DIS_3
```

De esta forma, entramos en la rutina de transferencia «TRANS» con «HL» conteniendo el inicio de un bloque de pantalla, «DE» el inicio del otro. «HL» el inicio de un bloque de atributos y «DE» el inicio del otro. La rutina «TRANS» consta de dos bucles; el primero es «BUC\_1»:

```
400 TRANS LD BC, 2048
410 BUC_1 LD A, (HL)
420 EX AF, AF'
430 LD A, (DE)
440 LD (HL), A
450 EX AF, AF'
460 LD (DE), A
470 INC HL
480 INC DE
490 DEC BC
500 LD A, B
510 OR C
520 JR NZ, BUC_1
```

En 400 cargamos «BC» con 2048 que será el número de iteraciones del bucle; en 410

es que no podemos, la (el mensaje «Bytes...» lo im-pide) y, además, el mensaje: «Start tape...» nos «machaca» las dos líneas inferiores. En este caso, resulta muy útil transferir la pantalla a una zona superior y salvarla desde allí. Esto es lo que pretendemos con el Programa 8-2. Primero pregunta dónde se quiere almacenar la pantalla, una buena dirección es de 50000. Luego, nos pide que la carguemos desde el cassette y la transfiera a la dirección que le hayamos dado. Después, la salvará y verificará desde esta dirección.

Si quiere usar esta pantalla como cabecera de un programa suyo, deberá cargarla con LOAD «SCREEN\$», aunque queda mejor cargarla en una zona más alta de memoria y utilizar esta rutina para recuperarla, con lo que la pantalla aparecerá «de golpe» como en muchos juegos comerciales (han sido muchos los lectores que han escrito a la sección «Consultorio» de MICROHOBBY preguntando cómo se hacía esto; bien, ahora ya lo saben).

Por supuesto, el Programa 8-2 no pretende ser más que un ejemplo. Esta rutina puede ser usada para muchas otras cosas que dependerán de la imaginación de cada cual. Nosotros le recomendamos que cargue dos pantallas en distintos lugares de la memoria y las vaya alternando en el televisor. La velocidad de transferencia es tan rápida que le parecerá estar viendo las dos a la vez.

El tercero y último, de los ejemplos que hemos preparado para este capítulo es, sin duda, el más útil. Se trata de

## PROGRAMA 2

```
10 REM PROGRAMA 8-2
20 FOR n=0 TO 21
30 READ a: POKE 23296+n, a: NEXT n
40 DATA 42, 118, 92, 17, 0, 64, 229,
58, 129, 92, 230, 1, 32, 1, 235, 1, 0, 27,
237, 176, 193, 201
100 INPUT "Dirección de almacenamiento? "; a: RANDOMIZE a: CLEAR a-1
120 PRINT "Ponga en marcha el cassette para cargar la pantalla."
130 LOAD "SCREEN$
140 POKE 23681, 0: LET a=USR 23296
150 CLS: PRINT "La pantalla está almacenada a partir de: "; a; "Meta la cinta donde quiere guardarla."
160 INPUT "Nombre que le va a dar: "; a$
170 POKE 23681, 1: RANDOMIZE USR 23296: SAVE a$CODE a, 6912
180 CLS: PRINT "Rebobine el cassette y pongalo en "PLAY" para verificar."
190 VERIFY a$CODE a
200 RANDOMIZE USR 23296: PRINT AT 10, 11: PAPER 7: INK 0: "CORRECTO": PAUSE 0
```

una potente rutina que permite buscar un determinado grupo de octetos dentro de una zona de memoria. Pueden ser desde 1 hasta 50 octetos, y cada uno puede valer desde 0 hasta 254 (el valor 255 está prohibido, ya veremos por qué). Si se encuentra la cadena buscada, se retorna en donde está y, si no se encuentra, se retorna un «0».

Este tipo de rutinas se utilizan con frecuencia para buscar cadenas alfanuméricas, líneas de Basic, bloques de código máquina, etc. Resulta muy útil, por ejemplo, saber a partir de qué dirección está almacenada una determinada línea de Basic, o una determinada variable alfanumérica. Para quienes tengan el moni-

tor «MONS-3», esta rutina hace lo mismo que el comando «G».

La rutina es reubicable, pero la hemos colocado en el buffer de impresora, a partir de la dirección 23350, dejando desde la 23296 hasta la 23349 para almacenar ciertas variables que usará la propia rutina.

En 23296 y 23297 almacenaremos la dirección inicial del bloque donde vamos a realizar la búsqueda. En 23298 y 23299 almacenaremos la longitud de este bloque y a partir de 23300 irán los códigos que componen la cadena a buscar. Después del último de estos códigos deberá ir el número 255 que servirá para indicar a la rutina el fin de la cadena; ésta es la razón por



contenga un número impar (1, 3, 5, etc.) y estará a «0» siempre que contenga un número par (0, 2, 4, etc.). El uso de flags es tan frecuente que el ZX-80 dispone de un grupo de instrucciones que permiten manejar los bits de forma independiente. Estas instrucciones se verán más adelante, de momento, podemos comprobar nuestro flag haciendo «AND 1» y comprobando el indicador de cero «Z» del registro «F»; este indicador será «1» si el flag era «0» y viceversa.

Por otro lado, hemos añadido dos instrucciones a la rutina para que nos devuelva, en el retorno (a través de «BC»), la dirección donde ha quedado almacenada la pantalla.

En la Figura 8-13, tiene el organigrama de esta rutina, verá que es muy sencillo. Empezamos por cargar en «HL» el contenido de la variable del sistema «SEED», en «DE» cargamos 4000h (16384) que es la dirección de inicio de la pantalla y hacemos «PUSH HL» para preservar el contenido de «HL»; de esta forma, la rutina queda preparada para transferir desde la dirección apuntada por «HL» hacia la apuntada por «DE», es decir, para recuperar una pantalla ya almacenada; veamos esta parte del listado:

```
100 OR6 23296
110 LD HL, (SEED)
120 LD DE, #4000h
130 PUSH HL
```

Ahora, vamos a comprobar el flag, para saber qué operación debemos realizar. Si el

flag está a «1», la operación será recuperar, por tanto, no es necesaria ninguna modificación. Pero, si el flag está a cero, tenemos que transferir la pantalla, así que deberemos intercambiar los contenidos de «HL» y «DE» (recuerde que «HL» es el origen y «DE» es el destino). Seguimos con el listado:

```
140 LD A, (23681)
150 AND 1
160 JR NZ, RECU
170 EX DE, HL
```

Si el flag era «1», nos saltaremos la instrucción de la línea 170 (la etiqueta «RECU» está en la línea 180 como ahora veremos). A continuación, viene la parte de la rutina que realiza la transferencia propiamente dicha. Cargamos «BC» con 6912 que es el número de bytes a transferir y utilizamos la potente instrucción «LDIR»; luego, recuperamos en «BC» lo que habíamos guardado en la pila y retornamos:

```
180 RECU LD BC, 6912
190 LDIR
200 POP BC
210 RET
220 SEED EQU 23670h
```

La línea 220 sirve para definir el valor de la etiqueta «SEED» que hemos usado en la línea 110.

Hemos colocado la rutina en el buffer de impresora porque es muy corta, y así, no nos ocupa memoria en ningún otro sitio; pero, es perfec-

tamente reubicable y puede correr igual en cualquier parte de la memoria. En la Figura 8-14, puede ver el listado de la rutina tal como lo produce un «GENS-3» cuando ensambla.

Ahora, vamos a ensamblar la rutina «a mano»; para los que no tengan facilidad en convertir números a hexadecimal, puede serles útil la siguiente tabla:

```
16384 = 4000h
23681 = 5C81h
23670 = 5C76h
6912 = 1B00h
```

La etiqueta «RECU» está en 23311 y el salto relativo «JR NZ, RECU» se ensambla en 23308 y 23309, de modo que el desplazamiento será «1». Ensamble la rutina y, después, compruebe su resultado: Estará correcto si le ha quedado así:

```
110 23296 42,118,92
120 23299 17,0,64
130 23302 229
140 23303 58,129,92
150 23306 230,1
160 23308 32,1
170 23310 235
180 23311 1,0,27
190 23314 237,176
200 23316 193
210 23317 201
```

Vemos que la rutina ocupa 22 bytes. Vamos a intentar hacer algo útil con ella.

Uno de los problemas de salvar en cinta una pantalla

cargamos en «A» el octeto de la primera zona y lo pasamos a «A» en 420; en 430 y 440 pasamos el octeto de la segunda zona a la primera; en 450 recuperamos el contenido de «A», y en 460 lo colocamos en la dirección correspondiente de la segunda zona; el resto del bucle está formado por las habituales instrucciones de incrementar punteros, decrementar contador y cerrar el bucle si «BC» difiere de cero. A continuación, recuperamos el dato de «HL», que habíamos metido en la pila y lo metemos, de nuevo, en «HL» al tiempo que sacamos a «HL» y «DE» las direcciones de inicio de los bloques de atributos a intercambiar:

```
530 POP HL
540 EXX
```

Ahora viene el bucle «BUC\_2» para intercambiar las zonas de atributos:

```
550 LD B,0
560 BUC_2 LD A, (HL)
570 EX AF, AF'
580 LD A, (DE)
590 LD (HL), A
600 EX AF, AF'
610 LD (DE), A
620 INC HL
630 INC DE
640 DJNZ BUC_2
650 RET
```

Este bucle es exactamente igual que el anterior, salvo que sólo tiene 256 iteraciones, por lo que cargamos «B»

HI-SOFT GENSYM ASSEMBLER

Copyright HI-SOFT 1983  
CURSO C/M MICROHBBY

Pass 1 errors: 00

```
10 4C-
20 4D+
60000 100 OR6
60000 110 EXX
60001 120 PUSH
60002 130 EXX
60003 140 LD
60004 150 CP
60005 160 JR
60006 170 CP
60007 180 JR
60008 190 CP
60009 200 JR
60010 210 DEF8
60011 220 DEF8
60012 230 DP_1
60013 240 LD
60014 250 EXX
60015 260 LD
60016 270 LD
60017 280 JR
60018 290 DP_2
60019 300 LD
60020 310 EXX
60021 320 LD
60022 330 LD
60023 340 LD
60024 350 LD
60025 360 LD
60026 370 LD
60027 380 LD
60028 390 LD
60029 400 TRANS
60030 410 BUC_1
60031 420 LD
60032 430 LD
60033 440 LD
60034 450 LD
60035 460 LD
60036 470 LD
60037 480 LD
60038 490 LD
60039 500 LD
60040 510 LD
60041 520 LD
60042 530 LD
60043 540 LD
60044 550 LD
60045 560 LD
60046 570 LD
60047 580 LD
60048 590 LD
60049 600 LD
60050 610 LD
60051 620 LD
60052 630 LD
60053 640 LD
60054 650 LD
60055 660 LD
60056 670 LD
60057 680 LD
60058 690 LD
60059 700 LD
60060 710 LD
60061 720 LD
60062 730 LD
60063 740 LD
60064 750 LD
60065 760 LD
60066 770 LD
60067 780 LD
60068 790 LD
60069 800 LD
60070 810 LD
60071 820 LD
60072 830 LD
60073 840 LD
60074 850 LD
60075 860 LD
60076 870 LD
60077 880 LD
60078 890 LD
60079 900 LD
60080 910 LD
60081 920 LD
60082 930 LD
60083 940 LD
60084 950 LD
60085 960 LD
60086 970 LD
60087 980 LD
60088 990 LD
60089 1000 LD
60090 1010 LD
60091 1020 LD
60092 1030 LD
60093 1040 LD
60094 1050 LD
60095 1060 LD
60096 1070 LD
60097 1080 LD
60098 1090 LD
60099 1100 LD
60100 1110 LD
60101 1120 LD
60102 1130 LD
60103 1140 LD
60104 1150 LD
60105 1160 LD
60106 1170 LD
60107 1180 LD
60108 1190 LD
60109 1200 LD
60110 1210 LD
60111 1220 LD
60112 1230 LD
60113 1240 LD
60114 1250 LD
60115 1260 LD
60116 1270 LD
60117 1280 LD
60118 1290 LD
60119 1300 LD
60120 1310 LD
60121 1320 LD
60122 1330 LD
60123 1340 LD
60124 1350 LD
60125 1360 LD
60126 1370 LD
60127 1380 LD
60128 1390 LD
60129 1400 LD
60130 1410 LD
60131 1420 LD
60132 1430 LD
60133 1440 LD
60134 1450 LD
60135 1460 LD
60136 1470 LD
60137 1480 LD
60138 1490 LD
60139 1500 LD
60140 1510 LD
60141 1520 LD
60142 1530 LD
60143 1540 LD
60144 1550 LD
60145 1560 LD
60146 1570 LD
60147 1580 LD
60148 1590 LD
60149 1600 LD
60150 1610 LD
60151 1620 LD
60152 1630 LD
60153 1640 LD
60154 1650 LD
60155 1660 LD
60156 1670 LD
60157 1680 LD
60158 1690 LD
60159 1700 LD
60160 1710 LD
60161 1720 LD
60162 1730 LD
60163 1740 LD
60164 1750 LD
60165 1760 LD
60166 1770 LD
60167 1780 LD
60168 1790 LD
60169 1800 LD
60170 1810 LD
60171 1820 LD
60172 1830 LD
60173 1840 LD
60174 1850 LD
60175 1860 LD
60176 1870 LD
60177 1880 LD
60178 1890 LD
60179 1900 LD
60180 1910 LD
60181 1920 LD
60182 1930 LD
60183 1940 LD
60184 1950 LD
60185 1960 LD
60186 1970 LD
60187 1980 LD
60188 1990 LD
60189 2000 LD
60190 2010 LD
60191 2020 LD
60192 2030 LD
60193 2040 LD
60194 2050 LD
60195 2060 LD
60196 2070 LD
60197 2080 LD
60198 2090 LD
60199 2100 LD
60200 2110 LD
60201 2120 LD
60202 2130 LD
60203 2140 LD
60204 2150 LD
60205 2160 LD
60206 2170 LD
60207 2180 LD
60208 2190 LD
60209 2200 LD
60210 2210 LD
60211 2220 LD
60212 2230 LD
60213 2240 LD
60214 2250 LD
60215 2260 LD
60216 2270 LD
60217 2280 LD
60218 2290 LD
60219 2300 LD
60220 2310 LD
60221 2320 LD
60222 2330 LD
60223 2340 LD
60224 2350 LD
60225 2360 LD
60226 2370 LD
60227 2380 LD
60228 2390 LD
60229 2400 LD
60230 2410 LD
60231 2420 LD
60232 2430 LD
60233 2440 LD
60234 2450 LD
60235 2460 LD
60236 2470 LD
60237 2480 LD
60238 2490 LD
60239 2500 LD
60240 2510 LD
60241 2520 LD
60242 2530 LD
60243 2540 LD
60244 2550 LD
60245 2560 LD
60246 2570 LD
60247 2580 LD
60248 2590 LD
60249 2600 LD
60250 2610 LD
60251 2620 LD
60252 2630 LD
60253 2640 LD
60254 2650 LD
60255 2660 LD
60256 2670 LD
60257 2680 LD
60258 2690 LD
60259 2700 LD
60260 2710 LD
60261 2720 LD
60262 2730 LD
60263 2740 LD
60264 2750 LD
60265 2760 LD
60266 2770 LD
60267 2780 LD
60268 2790 LD
60269 2800 LD
60270 2810 LD
60271 2820 LD
60272 2830 LD
60273 2840 LD
60274 2850 LD
60275 2860 LD
60276 2870 LD
60277 2880 LD
60278 2890 LD
60279 2900 LD
60280 2910 LD
60281 2920 LD
60282 2930 LD
60283 2940 LD
60284 2950 LD
60285 2960 LD
60286 2970 LD
60287 2980 LD
60288 2990 LD
60289 3000 LD
60290 3010 LD
60291 3020 LD
60292 3030 LD
60293 3040 LD
60294 3050 LD
60295 3060 LD
60296 3070 LD
60297 3080 LD
60298 3090 LD
60299 3100 LD
60300 3110 LD
60301 3120 LD
60302 3130 LD
60303 3140 LD
60304 3150 LD
60305 3160 LD
60306 3170 LD
60307 3180 LD
60308 3190 LD
60309 3200 LD
60310 3210 LD
60311 3220 LD
60312 3230 LD
60313 3240 LD
60314 3250 LD
60315 3260 LD
60316 3270 LD
60317 3280 LD
60318 3290 LD
60319 3300 LD
60320 3310 LD
60321 3320 LD
60322 3330 LD
60323 3340 LD
60324 3350 LD
60325 3360 LD
60326 3370 LD
60327 3380 LD
60328 3390 LD
60329 3400 LD
60330 3410 LD
60331 3420 LD
60332 3430 LD
60333 3440 LD
60334 3450 LD
60335 3460 LD
60336 3470 LD
60337 3480 LD
60338 3490 LD
60339 3500 LD
60340 3510 LD
60341 3520 LD
60342 3530 LD
60343 3540 LD
60344 3550 LD
60345 3560 LD
60346 3570 LD
60347 3580 LD
60348 3590 LD
60349 3600 LD
60350 3610 LD
60351 3620 LD
60352 3630 LD
60353 3640 LD
60354 3650 LD
60355 3660 LD
60356 3670 LD
60357 3680 LD
60358 3690 LD
60359 3700 LD
60360 3710 LD
60361 3720 LD
60362 3730 LD
60363 3740 LD
60364 3750 LD
60365 3760 LD
60366 3770 LD
60367 3780 LD
60368 3790 LD
60369 3800 LD
60370 3810 LD
60371 3820 LD
60372 3830 LD
60373 3840 LD
60374 3850 LD
60375 3860 LD
60376 3870 LD
60377 3880 LD
60378 3890 LD
60379 3900 LD
60380 3910 LD
60381 3920 LD
60382 3930 LD
60383 3940 LD
60384 3950 LD
60385 3960 LD
60386 3970 LD
60387 3980 LD
60388 3990 LD
60389 4000 LD
60390 4010 LD
60391 4020 LD
60392 4030 LD
60393 4040 LD
60394 4050 LD
60395 4060 LD
60396 4070 LD
60397 4080 LD
60398 4090 LD
60399 4100 LD
60400 4110 LD
60401 4120 LD
60402 4130 LD
60403 4140 LD
60404 4150 LD
60405 4160 LD
60406 4170 LD
60407 4180 LD
60408 4190 LD
60409 4200 LD
60410 4210 LD
60411 4220 LD
60412 4230 LD
60413 4240 LD
60414 4250 LD
60415 4260 LD
60416 4270 LD
60417 4280 LD
60418 4290 LD
60419 4300 LD
60420 4310 LD
60421 4320 LD
60422 4330 LD
60423 4340 LD
60424 4350 LD
60425 4360 LD
60426 4370 LD
60427 4380 LD
60428 4390 LD
60429 4400 LD
60430 4410 LD
60431 4420 LD
60432 4430 LD
60433 4440 LD
60434 4450 LD
60435 4460 LD
60436 4470 LD
60437 4480 LD
60438 4490 LD
60439 4500 LD
60440 4510 LD
60441 4520 LD
60442 4530 LD
60443 4540 LD
60444 4550 LD
60445 4560 LD
60446 4570 LD
60447 4580 LD
60448 4590 LD
60449 4600 LD
60450 4610 LD
60451 4620 LD
60452 4630 LD
60453 4640 LD
60454 4650 LD
60455 4660 LD
60456 4670 LD
60457 4680 LD
60458 4690 LD
60459 4700 LD
60460 4710 LD
60461 4720 LD
60462 4730 LD
60463 4740 LD
60464 4750 LD
60465 4760 LD
60466 4770 LD
60467 4780 LD
60468 4790 LD
60469 4800 LD
60470 4810 LD
60471 4820 LD
60472 4830 LD
60473 4840 LD
60474 4850 LD
60475 4860 LD
60476 4870 LD
60477 4880 LD
60478 4890 LD
60479 4900 LD
60480 4910 LD
60481 4920 LD
60482 4930 LD
60483 4940 LD
60484 4950 LD
60485 4960 LD
60486 4970 LD
60487 4980 LD
60488 4990 LD
60489 5000 LD
60490 5010 LD
60491 5020 LD
60492 5030 LD
60493 5040 LD
60494 5050 LD
60495 5060 LD
60496 5070 LD
60497 5080 LD
60498 5090 LD
60499 5100 LD
60500 5110 LD
60501 5120 LD
60502 5130 LD
60503 5140 LD
60504 5150 LD
60505 5160 LD
60506 5170 LD
60507 5180 LD
60508 5190 LD
60509 5200 LD
60510 5210 LD
60511 5220 LD
60512 5230 LD
60513 5240 LD
60514 5250 LD
60515 5260 LD
60516 5270 LD
60517 5280 LD
60518 5290 LD
60519 5300 LD
60520 5310 LD
60521 5320 LD
60522 5330 LD
60523 5340 LD
60524 5350 LD
60525 5360 LD
60526 5370 LD
60527 5380 LD
60528 5390 LD
60529 5400 LD
60530 5410 LD
60531 5420 LD
60532 5430 LD
60533 5440 LD
60534 5450 LD
60535 5460 LD
60536 5470 LD
60537 5480 LD
60538 5490 LD
60539 5500 LD
60540 5510 LD
60541 5520 LD
60542 5530 LD
60543 5540 LD
60544 5550 LD
60545 5560 LD
60546 5570 LD
60547 5580 LD
60548 5590 LD
60549 5600 LD
60550 5610 LD
60551 5620 LD
60552 5630 LD
60553 5640 LD
60554 5650 LD
60555 5660 LD
60556 5670 LD
60557 5680 LD
60558 5690 LD
60559 5700 LD
60560 5710 LD
60561 5720 LD
60562 5730 LD
60563 5740 LD
60564 5750 LD
60565 5760 LD
60566 5770 LD
60567 5780 LD
60568 5790 LD
60569 5800 LD
60570 5810 LD
60571 5820 LD
60572 5830 LD
60573 5840 LD
60574 5850 LD
60575 5860 LD
60576 5870 LD
60577 5880 LD
60578 5890 LD
60579 5900 LD
60580 5910 LD
60581 5920 LD
60582 5930 LD
60583 5940 LD
60584 5950 LD
60585 5960 LD
60586 5970 LD
60587 5980 LD
60588 5990 LD
60589 6000 LD
60590 6010 LD
60591 6020 LD
60592 6030 LD
60593 6040 LD
60594 6050 LD
60595 6060 LD
60596 6070 LD
60597 6080 LD
60598 6090 LD
60599 6100 LD
60600 6110 LD
60601 6120 LD
60602 6130 LD
60603 6140 LD
60604 6150 LD
60605 6160 LD
60606 6170 LD
60607 6180 LD
60608 6190 LD
60609 6200 LD
60610 6210 LD
60611 6220 LD
60612 6230 LD
60613 6240 LD
60614 6250 LD
60615 6260 LD
60616 6270 LD
60617 6280 LD
60618 6290 LD
60619 6300 LD
60620 6310 LD
60621 6320 LD
60622 6330 LD
60623 6340 LD
60624 6350 LD
60625 6360 LD
60626 6370 LD
60627 6380 LD
60628 6390 LD
60629 6400 LD
60630 6410 LD
60631 6420 LD
60632 6430 LD
60633 6440 LD
60634 6450 LD
60635 6460 LD
60636 6470 LD
60637 6480 LD
60638 6490 LD
60639 6500 LD
60640 6510 LD
60641 6520 LD
60642 6530 LD
60643 6540 LD
60644 6550 LD
60645 6560 LD
60646 6570 LD
60647 6580 LD
60648 6590 LD
60649 6600 LD
60650 6610 LD
60651 6620 LD
60652 6630 LD
60653 6640 LD
60654 6650 LD
60655 6660 LD
60656 6670 LD
60657 6680 LD
60658 6690 LD
60659 6700 LD
60660 6710 LD
60661 6720 LD
60662 6730 LD
60663 6740 LD
60664 6750 LD
60665 6760 LD
60666 6770 LD
60667 6780 LD
60668 6790 LD
60669 6800 LD
60670 6810 LD
60671 6820 LD
60672 6830 LD
60673 6840 LD
60674 6850 LD
60675 6860 LD
60676 6870 LD
60677 6880 LD
60678 6890 LD
60679 6900 LD
60680 6910 LD
60681 6920 LD
60682 6930 LD
60683 6940 LD
60684 6950 LD
60685 6960 LD
60686 6970 LD
60687 6980 LD
60688 6990 LD
60689 7000 LD
60690 7010 LD
60691 7020 LD
60692 7030 LD
60693 7040 LD
60694 7050 LD
60695 7060 LD
60696 7070 LD
60697 7080 LD
60698 7090 LD
60699 7100 LD
60700 7110 LD
60701 7120 LD
60702 7130 LD
60703 7140 LD
60704 7150 LD
60705 7160 LD
60706 7170 LD
60707 7180 LD
60708 7190 LD
60709 7200 LD
60710 7210 LD
60711 7220 LD
60712 7230 LD
60713 7240 LD
60714 7250 LD
60715 7260 LD
60716 7270 LD
60717 7280 LD
60718 7290 LD
60719 7300 LD
60720 7310 LD
60721 7320 LD
60722 7330 LD
60723 7340 LD
60724 7350 LD
60725 7360 LD
60726 7370 LD
60727 7380 LD
60728 7390 LD
60729 7400 LD
60730 7410 LD
60731 7420 LD
60732 7430 LD
60733 7440 LD
60734 7450 LD
60735 7460 LD
60736 7470 LD
60737 7480 LD
60738 7490 LD
60739 7500 LD
60740 7510 LD
60741 7520 LD
60742 7530 LD
60743 7540 LD
60744 7550 LD
60745 7560 LD
60746 7570 LD
60747 7580 LD
60748 7590 LD
60749 7600 LD
60750 7610 LD
60751 7620 LD
60752 7630 LD
60753 7640 LD
60754 7650 LD
60755 7660 LD
60756 7670 LD
60757 7680 LD
60758 7690 LD
60759 7700 LD
60760 7710 LD
60761 7720 LD
60762 7730 LD
60763 7740 LD
60764 7750 LD
60765 7760 LD
60766 7770 LD
60767 7780 LD
60768 7790 LD
60769 7800 LD
60770 7810 LD
60771 7820 LD
60772 7830 LD
60773 7840 LD
60774 7850 LD
60775 7860 LD
60776 7870 LD
60777 7880 LD
60778 7890 LD
60779 7900 LD
60780 7910 LD
60781 7920 LD
60782 7930 LD
60783 7940 LD
60784 7950 LD
60785 7960 LD
60786 7970 LD
60787 7980 LD
60788 7990 LD
60789 8000 LD
60790 8010 LD
60791 8020 LD
60792 8030 LD
60793 8040 LD
60794 8050 LD
60795 8060 LD
60796 8070 LD
60797 8080 LD
60798 8090 LD
60799 8100 LD
60800 8110 LD
60801 8120 LD
60802 8130 LD
60803 8140 LD
60804 8150 LD
60805 8160 LD
60806 8170 LD
60807 8180 LD
60808 8190 LD
60809 8200 LD
60810 8210 LD
60811 8220 LD
60812 8230 LD
60813 8240 LD
60814 8250 LD
60815 8260 LD
60816 8270 LD
60817 8280 LD
60818 8290 LD
60819 8300 LD
60820 8310 LD
60821 8320 LD
60822 8330 LD
60823 8340 LD
60824 8350 LD
60825 8360 LD
60826 8370 LD
60827 8380 LD
60828 8390 LD
60829 8400 LD
60830 8410 LD
60831 8420 LD
60832 8430 LD
60833
```



## PROGRAMA 1

```

10 REM PROGRAMA 8-1
20 CLEAR 59999
30 FOR n=0 TO 93: READ a: POKE
6000+n,a: NEXT n
40 DATA 217,229,217,58,129,92,
254,4,40,10,254,5,40,21,254,6,40,
32,207,10,33,0,88,17,0,89,217,3
3,0,64,17,0,72,24,28,33,0,89,17,
0,90,217,33,0,24,13,33,0,88,17,0,
90,217,33,0,64,17,0,80,1,0,8,126,
8,26,119,0,18,35,19,11,120,177,
32,243,225,217,6,0,126,8,26,119,
8,18,35,19,16,246,201
100 FOR n=1 TO 192: PRINT PAPER
1: INK 9;"1";: NEXT n
105 PRINT
110 FOR n=1 TO 192: PRINT PAPER
3: INK 9;"2";: NEXT n
115 PRINT
120 FOR n=1 TO 192: PRINT PAPER
5: INK 9;"3";: NEXT n
130 INPUT "Valor para "A"" (4,
5,0,6,1,2,3);a: POKE 23681,a: RANDO
MIZE USR 60000
140 GO TO 130

```

con «0» y utilizamos «D» para cerrar el bucle; finalmente, en la línea 650 retornamos con «RET».

Ya está acabada la rutina, parece complicada pero, cuando se comprende, resulta sumamente sencilla. En la Figura 8-12 tiene el listado completo tal como lo produce el «GENS-3» cuando ensamblos. Por nuestra parte, vamos a ensamblos la rutina «a mano» para que no se nos enfaden los que no tienen ensamblos.

La rutina es muy sencilla y no debe haber problemas para ensamblosla, en todo caso, resulta trabajoso porque son muchas instrucciones, pero ármese de paciencia que el trabajo vale la pena. Recuerde que, donde pone «ATT\_1», deberá poner « 5800 » y así con todas las etiquetas, excepto las correspondientes a los saltos relativos en los que deberá calcular el desplazamiento como aprendimos en el capítulo anterior.

Intente ensamblos por sí mismo, si no toda, al menos parte de la rutina y, luego, compruebe el resultado, corrija los errores y mire por qué los ha cometido para no volverlos a cometer...

Nos ha quedado así:

```

110 60000 217
120 60001 229
130 60002 217
140 60003 58,129,92
150 60006 254,4
160 60008 40,10
170 60010 254,5
180 60012 40,21
190 60014 254,6
200 60016 40,32
210 60018 207

```

Si ha sido usted capaz de ensamblos toda la rutina (aun con errores), ¡enhorabuena!., tiene usted una voluntad de hierro, será un gran programador, si, por el contrario, se ha cansado a la mitad, no se preocupe, no tiene más que comprarse un ensamblosador.

Ya tenemos el código objeto, ahora sólo falta meterlo en memoria y probarlo. El Programa 8-1 se encarga de ello; procure no equivocarse en las líneas DATA (40 y 50) o los resultados podrían ser imprevisibles. El funcionamiento del programa es muy sencillo y no creemos que necesite explicación.

El segundo de nuestros ejemplos es una rutina muy sencilla pero muy ilustrativa. Su utilidad es transferir una pantalla completa a una dirección más alta de memoria; aunque no es difícil adaptarla para que transfiera sólo parte de la pantalla o cualquier otro bloque de memoria.

Por otro lado, la rutina también permite recuperar una pantalla desde donde se la haya almacenado. Podrían haberse utilizado dos rutinas, una para transferir la pantalla y otra para recuperarla, pero ambas rutinas hubieran tenido un gran número de instrucciones comunes, así que hemos preferido utilizar una sola rutina que realice las dos tareas.

Para indicarle a esta rutina si ha de hacer una cosa u otra, utilizaremos un «FLAG» (en inglés: «Bandera»). Una bandera puede estar «levantada» o «bajada», por tanto, nos va a indicar una de entre dos condiciones posibles. En informática, utilizamos como bandera un bit; si está a «1», decimos que la «bandera» es-

está «levantada» (Flag a «1»); y si está a «0», decimos que está «bajada» (Flag a «0»).

En nuestra rutina, vamos a utilizar como flag el bit menos significativo de la posición de memoria 23681; si este bit está a «0», la rutina transferirá la pantalla a la dirección

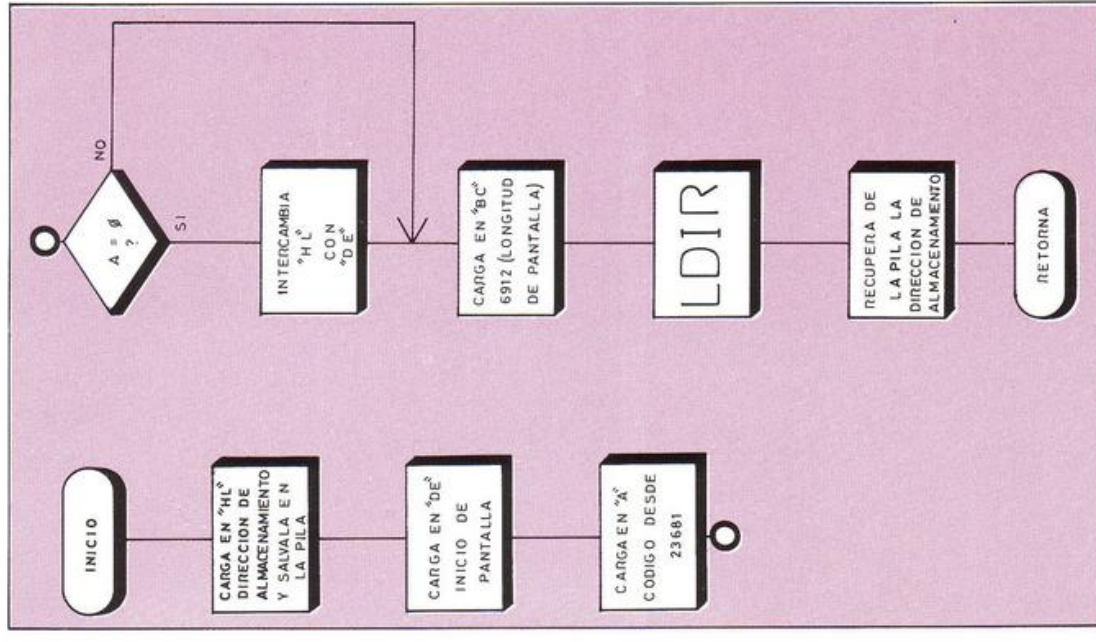


Fig. 8-13. Organigrama de la rutina para transferir pantallas.



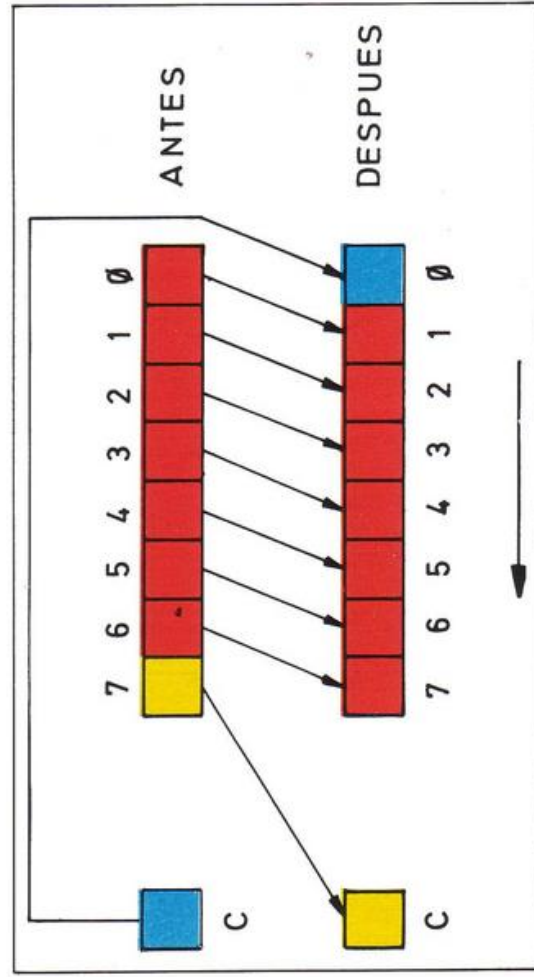


Fig. 9-2. Instrucciones RLA y RL.

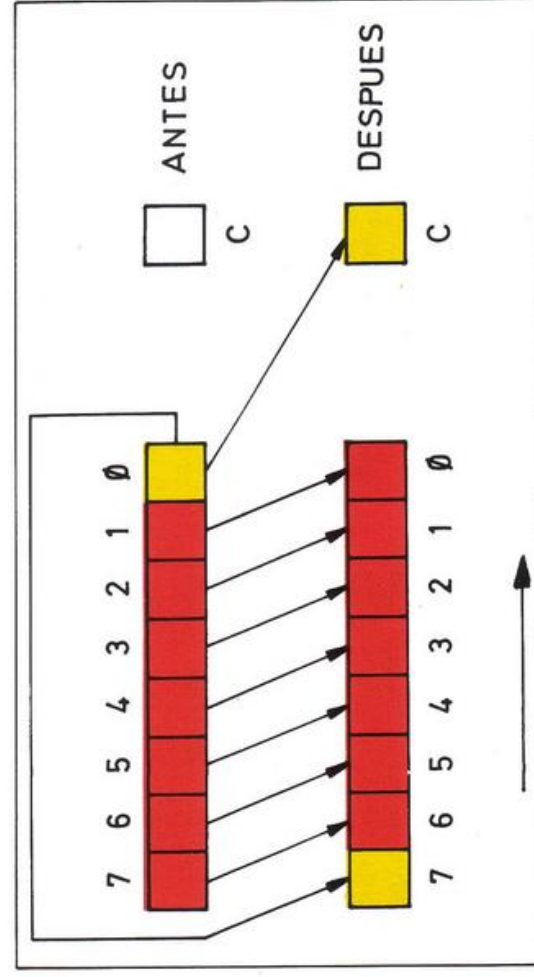


Fig. 9-3. Instrucciones RRCA y RRC.

CICLOS DE RELOJ: 4

EJEMPLO:

RRCA	73h	85h
Contenido del registro acumulador	0 1 1 1 0 0 1 1	1 0 1 1 1 0 0 1
Instrucción	0 0 0 0 1 1 1 1	
Indicadores de condición después de la ejecución		S Z H P V N C
Contenido del registro «A» después de la ejecución		x x x 0 x 0 1

La rutina se completa con las dos subrutinas de salida «FIN» y «FIN-Z»:

320	FIN	POP	HL
330		POP	BC
340		DEC	BC
350		RET	
360	FIN_2	LD	BC, 0
370		RET	

En «FIN», nos interesa pasar a «BC» la dirección donde se ha encontrado el primer carácter, así que recuperamos los registros en orden inverso a como los guardamos, para que lo que había en «HL» pase a «BC»; como esta dirección era la siguiente a donde se había encontrado el carácter, tenemos que decrementar «BC» antes de retornar.

En «FIN-Z», cargamos «cero» en «BC» y retornamos. El programa desde donde llamamos a esta rutina deberá comprobar, en el retorno, si «BC» es «0», en cuyo caso, actuará en la forma prevista para cuando la cadena no se encuentre.

Aunque la rutina es sencilla, su funcionamiento requiere un poco de atención para comprenderlo. Quizá le resulte útil una mirada al organigrama representado en la Figura 8-15.

En la Figura 8-16 tenemos el listado completo de la rutina tal como lo produciría un «GENS-3» al ensamblar.

Como de costumbre, ahora toca ensambalar la rutina «a mano». A continuación está la tabla de equivalencias «decimal-hexa» para las direcciones de esta rutina:

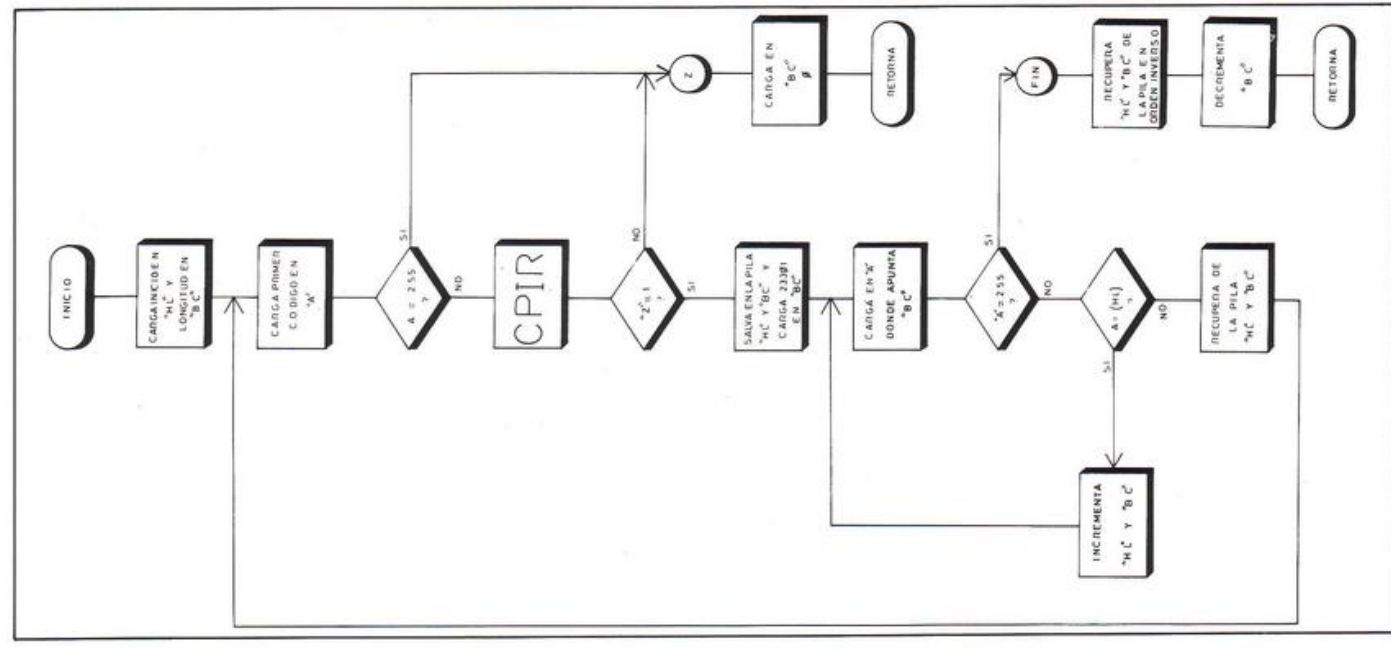


Fig. 8-15. Organigrama de la rutina para buscar cadenas.



23296 = 5800h  
 23298 = 5802h  
 23300 = 5804h  
 23301 = 5805h

Intente ensamblarla por sí mismo; recuerde, primero ensamble todas las instrucciones dejando en blanco los desplazamientos de los saltos relativos, luego, calcule éstos cuando sepa en qué dirección va cada instrucción. Mire a ver si le ha quedado así:

```

110 23350 42,0,91
120 23353 237,75,2,91
130 23357 58,4,91
140 23360 254,255
150 23362 40,29
160 23364 237,177
170 23366 32,25
180 23368 229
190 23369 197
200 23370 1,5,91
210 23373 10
220 23374 254,255
230 23376 40,11
240 23378 190
250 23379 40,4
260 23381 193
270 23382 225
280 23383 24,228
290 23385 35
300 23386 3
310 23387 24,240
320 23389 225
330 23390 193
340 23391 11
350 23392 201
360 23393 1,0,0
370 23396 201
  
```

★HISOFT GENSYM ASSEMBLER★  
 ZX SPECTRUM

Copyright HISOFT 1983  
 CURSO C/M MICROHOBBY

Pass 1 errors: 00

```

80 #C-
90 #D+
23350 100 ORG
23350 110 LD
23353 120 LD
23357 130 CONT
23360 140 CP
23362 150 JR
23364 160 CPIR
23366 170 JR
23368 180 PUSH
23369 190 PUSH
23370 200 LD
23373 210 BUCLE
23374 220 CP
23376 230 JR
23378 240 CP
23379 250 JR
23381 260 POP
23382 270 POP
23383 280 JR
23385 290 OK
23386 300 INC
23387 310 INC
23389 320 FIN
23390 330 POP
23391 340 POP
23392 350 DEC
23393 360 FIN_Z
23396 370 RET
  
```

Pass 2 errors: 00

```

BUCLE 584D CONT 583D
FIN 585D FIN_Z 5861
OK 5859
  
```

Table used: 67 from 144

Fig. 8-16. Listado completo de la rutina para buscar cadenas.

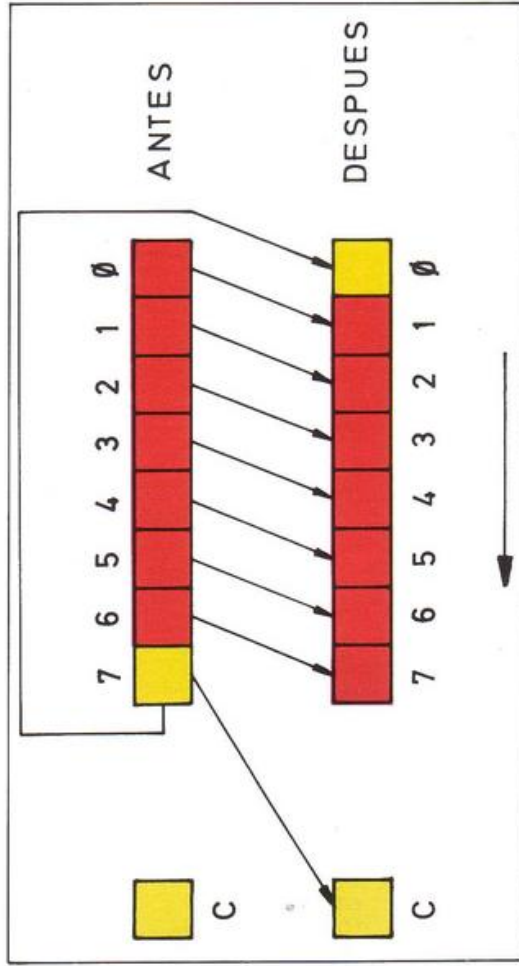


Fig. 9-1. Instrucciones RLCa y RLC.

**RLA**

OBJETO:

Rota a la izquierda el contenido del registro acumulador un bit. El contenido del bit 7 saliente se copia en el indicador de acarreo y el indicador de acarreo anterior se copia en el bit 0 entrante (Rotate Left Accumulator). Ver Figura 9-2.

CODIGO DE MAQUINA:

00010111 17h

INDICADORES DE

CONDICION QUE AFECTA:

H; pone 0 - siempre  
 N; pone 0 - siempre  
 C; pone el valor que tenía el bit 7 del registro A antes de la ejecución

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

RLA

Contenido del registro acumulador

(A): 00110100 34h

Indicador de acarreo C=1

Instrucción

RLA: 00010111 17h

Contenido del registro «A» después de la ejecución

(A): 01101001 68h

Indicadores de condición después de la ejecución

S Z H PV N C

x x x 0 x x 0 0

**RRCA**

OBJETO:

Rota a la derecha el contenido del registro acumulador un bit. El contenido del bit 0 saliente se copia en el bit 7 entrante y en el indicador de acarreo «C» (Rotate Right with Carry Accumulator). Ver Figura 9-3.

CODIGO DE MAQUINA:

00000111 0fh

INDICADORES DE

CONDICION QUE AFECTA:

H; pone 0 - siempre  
 N; pone 0 - siempre  
 C; pone el valor que tenía el bit 0 del registro A antes de la ejecución

CICLOS DE MEMORIA:

1



## GRUPO DE INSTRUCCIONES DE ROTACION Y DESPLAZAMIENTO

Este grupo de instrucciones tiene como misión mover los bits de un octeto de uno en uno cada vez, a derecha o izquierda. En algunos casos el bit que sale fuera del octeto da la vuelta y entra por el lugar que queda vacío, éstas son las instrucciones de rotación. En los casos en que el bit que sale del octeto desaparece, estamos ante las instrucciones de desplazamiento.

Como código nemotécnico se emplean las letras dadas a continuación, las cuales van asociadas con la palabra inglesa de origen y su significado castellano:

R=«Rotate»;  
S=«Shift»;  
C=«Carry»;  
A=«Accumulator»;  
L=«Left»;  
R=«Right»;

Rotación  
Desplazamiento  
Indicador de acarreo  
Registro acumulador  
Izquierda  
Derecha (cuando aparece en segundo lugar)

Por ejemplo el código «RRA» nos indicaría que es una instrucción de rotación, a la derecha y del registro acumulador (Rotate Right Accumulator).

RLCA

OBJETO:

Rota a la izquierda el contenido del registro acumulador un bit. El contenido del bit 7 saliente se copia en el bit 0 entrante y en el indicador de acarreo «C» (Rotate Left with Carry Accumulator). Ver Figura 9-1.

CODIGO DE MAQUINA:

0 0 0 0 0 1 1 1 0 7h

INDICADORES DE CONDICION QUE AFECTA:

H; pone 0 - siempre  
N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del registro A antes de la ejecución

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

RLCA

Contenido del registro acumulador

(A): 1 0 0 0 1 0 0 1 88h

Instrucción

RLCA: 0 0 0 0 0 1 1 1 0 7h

Contenido del registro «A» después de la ejecución

(A): 0 0 0 1 0 0 1 1 13h

Indicadores de condición después de la ejecución

S Z H P V N C  
x x x 0 x 0 1

En este caso, y rompiendo la costumbre, no hemos escrito ningún programa en Basic para demostrar el funcionamiento de esta rutina. La razón es que no tendría sentido, ya que la utilidad de esta rutina es la de servir como subrutina a otras.

No obstante, puede utilizarse para buscar la dirección de comienzo de una determinada línea de un programa Basic que tenga en memoria, o el sitio donde está almacenada determinada variable alfanumérica. Para ello, no tiene más que «POKEar» los valores correspondientes en las primeras direcciones del búfer.

fer de impresora y llamar a la rutina con «PRINT USR 23350»; le imprimirá la dirección donde ha encontrado la coincidencia o «0» si no la ha encontrado.

Otro uso interesante de esta rutina es el de explorar la RROM en busca de determinados datos; por ejemplo, intentar descubrir en qué dirección de la ROM está el mensaje «(c) 1982 Sinclair Research Ltd». Si lo encuentra, tal vez le dé por buscar el resto de los mensajes. En ese caso, tenga en cuenta que la letra final de cada uno de los mensajes está en memoria con el bit 7 puesto a «1», es

decir, el número que encontrará en la memoria es 128 más el código de la letra.

En el siguiente capítulo, veremos un grupo de instrucciones muy interesantes que nos van a permitir realizar rotaciones dentro de los registros o entre éstos y posiciones de memoria. Pero antes de pasar a ello, sería interesante que resolviésemos los siguientes ejercicios para comprobar si tienen bien afianzados los conocimientos adquiridos hasta aquí. Si no es capaz de resolver alguno, no se limite a mirar la solución, intente descubrir dónde y por qué falló.

### EJERCICIOS

1.- ¿Que valor contendrá «DE» al final de la siguiente rutina?:

100 LD HL,23000  
110 LD DE,50000  
120 LD BC,2000  
130 LD1R

2.- ¿Que valor tendrá «HL» al final de la siguiente rutina, si el contenido de la posición de memoria 50017 es «234»?:

100 LD HL,50000  
110 LD BC,50  
120 LD A,234  
130 CPIR

3.- Escriba una rutina que busque el código «FFh» en los 100 bytes anteriores a la dirección 50000, realizando la búsqueda «hacia atrás».



4.- Escriba una rutina que realice un "Scroll" ascendente del archivo de atributos, perdiendo la primera línea.

5.- Igual que la anterior, pero haciendo un "Scroll" descendente. (En este caso, se perderá la última línea).

----- 0 -----

5.- La rutina sería:

```
100 LD HL,23263
110 LD DE,23295
120 LD BC,736
130 LDOR
```

Esta vez, 23295 es la dirección final del archivo de atributos y 23263 es la última dirección de la penúltima línea. Hemos usado "LDOR" porque los bloques se solapan, por tanto, la transferencia hay que hacerla "hacia atrás".

----- 0 -----

## SOLUCIONES A LOS EJERCICIOS

1.- "DE" contendrá 52000, ya que se habrá incrementado 2000 veces.

2.- "HL" contendrá 50018, ya que el código 234 se ha encontrado en 50017 y "HL" sale, siempre, apuntando una dirección por encima de donde se ha encontrado el código.

3.- La rutina sería:

```
100 LD HL,49999
110 LD BC,100
120 LD A,0FF
130 CPDR
```

Observe que hemos empleado "CPDR" porque había que realizar la búsqueda "hacia atrás".

4.- La rutina sería:

```
100 LD HL,22560
110 LD DE,22528
120 LD BC,736
130 LDIR
```

La dirección 22528 es la primera del archivo de atributos y 22560 es la primera de la segunda línea. 736 es 23x32.



# CODIGO DE MAQUINA:

11001011	CBh
00001110	0h

## INDICADORES DE

### CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso

H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del octeto antes de la ejecución  
P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

4

## CICLOS DE RELOJ:

15

## EJEMPLO:

11011101	CBh
11001011	0h

Contenido del par de registros «HL»

01110000	70h
00100111	27h

Contenido del octeto de memoria 7027h

00000001	01h
----------	-----

## Instrucción

11001010	CBh
00001110	0h

Contenido del octeto de memoria 7027h después de la ejecución

10000000	80h
----------	-----

El contenido del par de registros «HL» no ha variado  
Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
1	0	x	0	0	1

## RRC (IX + d)

## OBJETO:

Rota a la derecha, un bit, el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante. Ver Figura 9-3.

## CODIGO DE MAQUINA:

11011101	D0h
11001011	CBh
00001110	0h

## INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía

el bit 0 del octeto antes de la ejecución  
P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

6

## CICLOS DE RELOJ:

23

## EJEMPLO:

10000101	85h
00100111	27h

Contenido del registro índice «IX»

10000101	85h
00100111	27h

Contenido del octeto de memoria 856Ah (IX + 43)

00101010	2Ah
----------	-----

## Instrucción

11011101	D0h
11001010	CBh
00101011	28h
00001110	0h

Contenido del octeto de memoria 856Ah después de la ejecución

00010101	15h
----------	-----

## Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	0	0	0

## RRC (IY + d)

## OBJETO:

Rota a la derecha, un bit, el octeto de memoria direc-

# RRA

## OBJETO:

Rota a la derecha el contenido del registro acumulador un bit. El contenido del bit 0 saliente se copia en el indicador de acarreo y el indicador de acarreo anterior se copia en el bit 7 entrante (Rotate Right Accumulator). Ver Figura 9-4.

## CODIGO DE MAQUINA:

00011111	1Fh
----------	-----

## INDICADORES DE CONDICION QUE AFECTA:

H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del registro A antes de la ejecución

# CICLOS DE MEMORIA:

1

# CICLOS DE RELOJ:

4

## EJEMPLO:

11110000	F0h
----------	-----

Contenido del registro acumulador

11110000	F0h
----------	-----

## Indicador de acarreo C=0

## Instrucción

00011111	1Fh
----------	-----

Contenido del registro «A» después de la ejecución

01111000	78h
----------	-----

Indicadores de condición después de la ejecución

# S Z H P V N C

x	x	x	x	x	0	0	0	0
---	---	---	---	---	---	---	---	---

# RLC r

## OBJETO:

Rota a la izquierda el contenido del registro representado por «r» un bit. El contenido del bit 7 saliente se copia en el bit 0 entrante y en el indicador de acarreo «C» (Rotate Left with Carry «r»). El código de representación de «r» es el señalado más abajo. Ver Figura 9-1.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

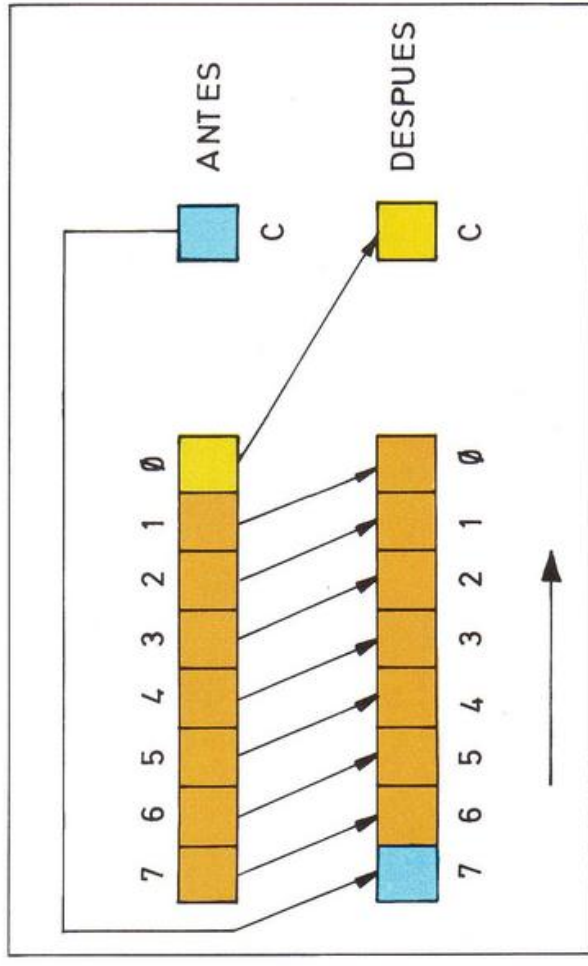


Fig. 9-4. Instrucciones RPA y RR.



## CODIGO DE MAQUINA:

CBh	1 1 0 0 1 0 1 1
	0 0 0 0 0 0 0 0

## INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del registro r antes de la ejecución

PV; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

2

## CICLOS DE RELOJ:

8

## EJEMPLO:

RLC D	1 1 0 0 1 0 1 1
	0 0 0 0 0 1 1 0

## Contenido del registro «D»

ID:	1 0 1 0 1 0 1 0
-----	-----------------

## Instrucción

RLC D:	1 1 0 0 1 0 1 0
	0 0 0 0 0 1 1 0

## Contenido del registro «D» después de la ejecución

ID:	0 1 0 1 0 1 0 1
-----	-----------------

## Indicadores de condición después de la ejecución

S	Z	H	PV	N	C
0	0	x	0	x	1

## RLC (HL)

## OBJETO:

Rota a la izquierda, un bit, el octeto de memoria direccionado por el contenido del par de registros «HL». El contenido del bit 7 saliente se copia en el bit 0 entrante y en el indicador de acarreo «C» (Rotate Left with Carry (HL)). Ver Figura 9-1.

## CODIGO DE MAQUINA:

CBh	1 1 0 0 1 0 1 1
0Bh	0 0 0 0 0 1 1 0

## INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del octeto antes de la ejecución

PV; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

4

## CICLOS DE RELOJ:

15

## EJEMPLO:

RLC (HL)	1 1 0 0 1 0 1 1
	0 0 0 0 0 1 1 0

## Contenido del par de registros «HL»

HL:	0 1 1 1 0 0 1 1
HL:	0 1 0 0 0 1 1 0

## Contenido del octeto de memoria 7342h

7342h:	0 1 0 1 0 1 0 1
--------	-----------------

## Instrucción

RLC (HL):	1 1 0 0 1 0 1 0
	0 0 0 0 0 1 1 0

## Contenido del octeto de memoria 7342h después de la ejecución

7342h:	1 0 1 0 1 0 1 0
--------	-----------------

El contenido del par de registros «HL» no ha variado. Indicadores de condición después de la ejecución

S	Z	H	PV	N	C
1	0	x	0	x	1

## RLC (IX + d)

## OBJETO:

Rota a la izquierda, un bit, el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 7 saliente se copia en el bit 0 entrante y en el indicador de acarreo «C». Ver Figura 9-1.

## CODIGO DE MAQUINA:

DDh	1 1 0 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
0Bh	0 0 0 0 0 1 1 0

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del octeto antes de la ejecución

PV; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

6

## CICLOS DE RELOJ:

23

## EJEMPLO:

RL (IY + 0)	1 1 1 1 0 1 0 1
	1 1 0 0 0 1 1 1

## Contenido del registro índice «IY»

IY:	1 1 1 1 0 1 0 1
	1 1 0 0 0 1 1 1

## Contenido del octeto de memoria F5C3h

F5C3h:	1 1 0 0 0 1 0 1
--------	-----------------

## Indicador de acarreo C = 1

## Instrucción

RL (IY + 0):	1 1 0 1 1 1 0 1
	1 1 0 0 0 1 1 0
	0 0 0 0 0 0 0 0
	0 0 0 1 0 1 1 0

## Contenido del octeto de memoria F5C3h después de la ejecución

F5C3h:	1 0 0 0 1 0 1 1
--------	-----------------

## Indicadores de condición después de la ejecución

S	Z	H	PV	N	C
1	0	x	0	x	1

## RRC r

## OBJETO:

Rota a la derecha, un bit, el contenido del registro representado por «r». El contenido del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante. Ver Figura 9-3. El código de representación de «r» es el señalado más abajo.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

## CODIGO DE MAQUINA:

CBh	1 1 0 0 1 0 1 1
	0 0 0 0 1 1 0 0

## INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 0 del registro r antes de la ejecución

PV; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

## CICLOS DE MEMORIA:

2

## CICLOS DE RELOJ:

8

## EJEMPLO:

RRC L	1 0 0 1 0 1 1 0
-------	-----------------

## Contenido del registro «L»

IL:	1 0 0 1 0 1 1 0
-----	-----------------

## Instrucción

RRC L:	1 1 0 0 1 0 1 0
	0 0 0 0 1 1 0 1

## Contenido del registro «L» después de la ejecución

IL:	0 1 0 0 1 0 1 1
-----	-----------------

## Indicadores de condición después de la ejecución

S	Z	H	PV	N	C
0	0	x	0	x	1

## RRC (HL)

## OBJETO:

Rota a la derecha, un bit, el octeto de memoria direccionado por el contenido del par de registros «HL». El contenido del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante. Ver Figura 9-3.



DH:	0 1 1 0 0 0 1 1	63h
LI:	0 0 1 1 0 1 1 0	36h

Contenido del octeto de memoria 636h

636h:	0 0 0 0 0 0 0 0	00h
-------	-----------------	-----

Indicador de acarreo C = 1

Instrucción

RL (HL)	1 1 0 0 1 0 1 0	CBh
	0 0 0 1 0 1 1 0	16h

Contenido del octeto de memoria 636h después de la ejecución

636h:	0 0 0 0 0 0 0 1	01h
-------	-----------------	-----

El contenido del par de registros «HL» no ha variado  
Indicadores de condición después de la ejecución

S	Z	H	PV	N	C
0	0	x	0	x	0

**RL (IX + d)**

OBJETO:

Rota a la izquierda, un bit, el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 7 saliente se copia en el indicador de acarreo «C» y el indicador de acarreo anterior se copia en el bit 0 entrante. Ver Figura 9-2.

**214 CODIGO MAQUINA**

CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1	DDh
1 1 0 0 1 0 1 1	CBh
0 0 0 1 0 1 1 0	16h

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 7 del octeto antes de la ejecución  
PV; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

Contenido del octeto de memoria 636h después de la ejecución

636h:	0 0 0 0 0 0 0 1	01h
-------	-----------------	-----

El contenido del par de registros «HL» no ha variado  
Indicadores de condición después de la ejecución

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

RL (IX + 40)	
--------------	--

Contenido del registro índice «IX»

1 0 0 0 0 0 1 0	82h
0 0 0 0 0 0 0 0	00h

Contenido del octeto de memoria 8228h

8228h:	1 1 1 1 1 1 1 1	Ffh
--------	-----------------	-----

Indicador de acarreo C = 0

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 7 del octeto antes de la ejecución  
PV; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

Contenido del octeto de memoria 8228h después de la ejecución

8228h:	1 1 1 1 1 1 1 0	Fh
--------	-----------------	----

Indicadores de condición después de la ejecución  
S; Z H PV N C  
1 0 x 0 x 0 0 1

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

RL (IX + 2)	
-------------	--

Contenido del registro índice «IX»

1 1 1 1 0 0 0 0	F0h
0 1 0 1 1 0 1 0	5Ah

Contenido del octeto de memoria F05Ch

F05Ch:	1 1 1 1 1 1 1 0	Fh
--------	-----------------	----

Instrucción

1 1 0 1 1 1 0 1	DD
1 1 0 0 1 0 1 1	CBh
0 0 0 0 0 0 1 0	02h
0 0 0 0 0 1 1 0	06h

Contenido del octeto de memoria F05Ch después de la ejecución

F05Ch:	1 1 1 1 1 1 0 1	FDh
--------	-----------------	-----

Indicadores de condición después de la ejecución  
S Z H PV N C  
1 0 x 0 x 0 0 1

EJEMPLO:

**RLC (IY + d)**

OBJETO:

Rota a la izquierda, un bit, el octeto de memoria direccionado por el contenido del registro índice «IY» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 7 saliente se copia en el bit 0 entrante y en el indicador de acarreo «C». Ver Figura 9-1.

CODIGO DE MAQUINA:

1 1 1 1 1 1 0 1	FDh
1 1 0 0 1 0 1 1	CBh
0 0 0 0 0 1 1 0	06h

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 7 del octeto antes de la ejecución  
PV; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

OBJETO:

Rota a la izquierda, un bit, el contenido del registro representado por «r». El contenido del bit 7 saliente se copia en el indicador de acarreo «C», y el valor anterior del indicador de acarreo se copia

**CODIGO MAQUINA 211**



en el bit 0 entrante. Ver Figura 9-2. El código de representación de «r» es el señalado más abajo.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

#### CODIGO DE MAQUINA:

11001011	CBh
00010	14h

#### INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 7 del registro r antes de la ejecución  
P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

#### CICLOS DE MEMORIA:

2

#### CICLOS DE RELOJ:

8

#### EJEMPLO:

RL H	00000010	02h
------	----------	-----

Contenido del registro «H»

02h

Indicador de acarreo C=0

#### Instrucción

RL H:	11001010	CBh
	00010100	14h

Contenido del registro «H» después de la ejecución

04h	00000100
-----	----------

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	0	x	0	0

#### RL (HL)

#### OBJETO:

Rota a la izquierda, un bit, el octeto de memoria direccionado por el contenido del par de registros «HL». El contenido del bit 7 saliente se copia en el indicador de acarreo «C» y el indicador de acarreo anterior se copia en el bit 0 entrante. Ver Figura 9-2.

#### CODIGO DE MAQUINA:

11001011	CBh
00010110	16h

#### INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía

#### INSTRUCCIONES DE ROTACION

Código Fuente	Hexadecimal	Decimal
RL A	17	23
RL B	CB,10	203,16
RL C	CB,11	203,17
RL D	CB,12	203,18
RL E	CB,13	203,19
RL H	CB,14	203,20
RL L	CB,15	203,21
RL (HL)	CB,16	203,22
RL (IX+d)	DD,CB,d,16	221,203,d,22
RL (IY+d)	FD,CB,d,16	253,203,d,22
RLC A	CB,07	203,7
RLC B	CB,08	203,8
RLC C	CB,09	203,9
RLC D	CB,0A	203,10
RLC E	CB,0B	203,11
RLC H	CB,0C	203,12
RLC L	CB,0D	203,13
RLC (HL)	CB,0E	203,14
RLC (IX+d)	DD,CB,d,0E	221,203,d,14
RLC (IY+d)	FD,CB,d,0E	253,203,d,14
RR A	1F	31
RR B	CB,18	203,24
RR C	CB,19	203,25
RR D	CB,1A	203,26
RR E	CB,1B	203,27
RR H	CB,1C	203,28
RR L	CB,1D	203,29
RR (HL)	CB,1E	203,30
RR (IX+d)	DD,CB,d,1E	221,203,d,30
RR (IY+d)	FD,CB,d,1E	253,203,d,30
RRC A	CB,0F	203,15
RRC B	CB,08	203,8
RRC C	CB,09	203,9
RRC D	CB,0A	203,10
RRC E	CB,0B	203,11
RRC H	CB,0C	203,12
RRC L	CB,0D	203,13
RRC (HL)	CB,0E	203,14
RRC (IX+d)	DD,CB,d,0E	221,203,d,14
RRC (IY+d)	FD,CB,d,0E	253,203,d,14

Fig. 9-5. Tabla de codificación para instrucciones de rotación.

#### INSTRUCCIONES DE ROTACION

NEMONICO	INDICADORES	No. DE BYTES	CICLOS MEM. REL.
RLC A	S Z H x P/V N C	1	1
RLC r	S Z H x P/V N C	2	2
RLC (HL)	S Z H x P/V N C	2	4
RLC (IX+d)	S Z H x P/V N C	4	6
RLC (IY+d)	S Z H x P/V N C	4	6
RL A	S Z H x P/V N C	1	1
RL s	S Z H x P/V N C	2	2
RRC A	S Z H x P/V N C	1	1
RRC s	S Z H x P/V N C	2	2
RR A	S Z H x P/V N C	1	1
RR s	S Z H x P/V N C	2	2

#### NOTAS:

- Los signos tienen el siguiente significado:  
"↑": El indicador cambia de valor de acuerdo con el resultado de la instrucción.  
"x": El bit adquiere un estado indeterminado.  
"-": El indicador no es afectado por la instrucción y conserva su anterior contenido.  
"0": El indicador se pone siempre a "cero".  
"P": El indicador "P/V" actúa como indicador de paridad.
- La letra "r" indica cualquiera de los registros: "B", "C", "D", "E", "H" o "L".
- La letra "s" indica cualquiera de los operandos: "r", "(HL)", "(IX+d)" o "(IY+d)".

Fig. 9-6. Tabla resumida de indicadores y ciclos para las instrucciones de rotación.

el bit 7 del octeto antes de la ejecución

P/V; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

#### EJEMPLO:

RL (HL)	15
---------	----

#### CICLOS DE MEMORIA:

4

Contenido del par de registros «HL»



H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del octeto antes de la ejecución  
P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

SRA (IX+47)

Contenido del registro índice «IX»

73h	0 1 1 1 0 0 1 1
40h	0 1 0 0 0 0 0 0

Contenido del octeto de memoria 736Fh

736Fh: 0 1 1 1 0 0 0 0

Instrucción

00	1 1 0 1 1 1 0 1
CBh	1 1 0 0 1 0 1 0
2Fh	0 0 1 0 1 1 1 1
2Eh	0 0 1 0 1 1 1 0

Contenido del octeto de memoria 736Fh después de la ejecución

736Fh: 0 0 1 1 1 0 1 0

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C	
0	0	x	0	1	0	0

SRA (IY + d)

OBJETO:

Desplaza a la derecha, un bit, el octeto de memoria direccionado por el contenido del registro índice «IY» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante se pone el mismo valor que tenía el bit 7 anteriormente. Ver Figura 9-8.

CODIGO DE MAQUINA:

FDh	1 1 1 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
2Eh	0 0 1 0 1 1 1 0

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del octeto antes de la ejecución

P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

SRA (IY-2)

Contenido del registro índice «IY»

A7h	1 0 1 0 0 1 1 1
24h	0 0 1 0 0 1 0 0

Contenido del octeto de memoria A722h

A722h: 1 0 0 0 0 0 0 0

Instrucción

00	1 1 0 1 1 1 0 1
CBh	1 1 0 0 1 0 1 0
FEh	1 1 1 1 1 1 1 0
2Eh	0 0 1 0 1 1 1 0

Contenido del octeto de memoria A722h después de la ejecución

A722h: 1 1 0 0 0 0 0 0

Indicadores de condición después de la ejecución

S	Z	H	P	V	N	C	
1	0	x	0	x	1	0	0

SRL r

OBJETO:

Desplaza a la derecha, un bit, el contenido del registro representado por «r». El bit 0 saliente se copia en el indicador de acarreo «C» y el bit 7 entrante se pone a 0. El código de representación de «r» es el señalado más abajo. Ver Figura 9-9.

cionado por el contenido del registro índice «IY» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante. Ver Figura 9-3.

CODIGO DE MAQUINA:

FDh	1 1 1 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
0Eh	0 0 0 0 1 1 1 0

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del octeto antes de la ejecución

P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

RRC (IY-10)

Contenido del registro índice «IY»

70h	0 1 1 1 0 0 0 0
08h	0 0 0 0 1 0 1 1

Contenido del octeto de memoria 7001h (IY-10)

7001h: 0 0 0 0 0 0 0 0

Instrucción

00	1 1 0 1 1 1 0 1
CBh	1 1 0 0 1 0 1 0
0Eh	1 1 1 1 0 1 1 0
0Eh	0 0 0 0 1 1 1 0

Contenido del octeto de memoria 7001h después de la ejecución

7001h: 0 0 0 0 0 0 0 0

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C	
0	1	x	0	1	0	0

RR r

OBJETO:

Rota a la derecha, un bit, el contenido del registro representado por «r». El contenido del bit 0 saliente se copia en el indicador de acarreo «C» y el valor del indicador de acarreo anterior se copia en el bit 7 entrante. El código de representación de «r» es el señalado más abajo. Ver Figura 9-4.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

CODIGO DE MAQUINA:

CBh	1 1 0 0 1 0 1 1
	0 0 0 1 1 1 → 1 →

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del registro «r» antes de la ejecución

P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

8

EJEMPLO:

RR B

Contenido del registro «B»

(B): 1 0 1 0 0 1 0 1

Indicador de acarreo C = 1

Instrucción

CBh	1 1 0 0 1 0 1 0
18h	0 0 0 1 1 0 0 0

Contenido del registro «B» después de la ejecución

(B): 1 1 0 1 0 0 1 0



S Z H P V N C  
1 0 x 0 x 1 0 1

RR (HL)

(H)	1 1 1 1 0 0 1 1	F3h
(L)	0 1 0 0 1 1 1 0	4Eh

E34Fh-00000001

RR (HL)	CBh	1H
11001010		
00011110		

11001011	CBh
00011110	TEh

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;



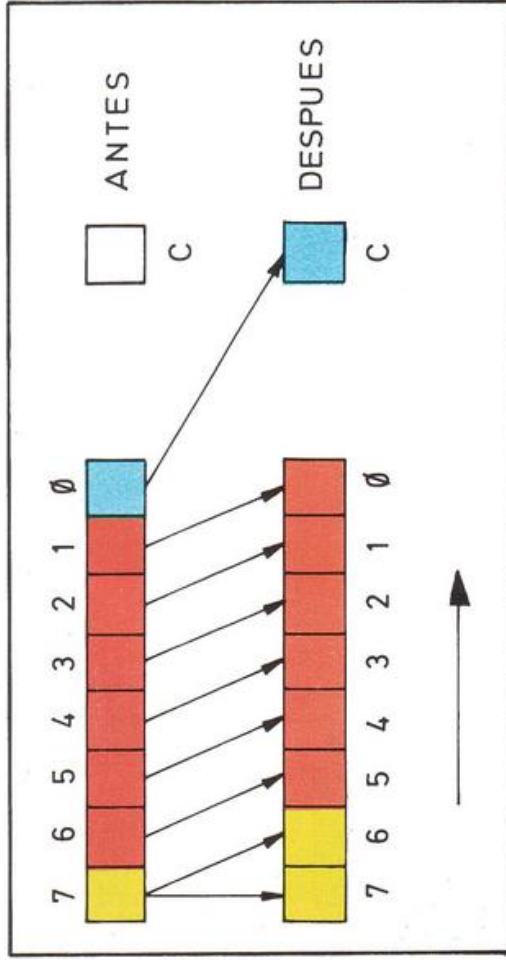


Fig. 9-8. Ilustración SRA.

P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

Contenido del octeto de memoria F3A2h después de la ejecución

F3A2h: 0000001000 04h

CICLOS DE RELOJ: 23

EJEMPLO:

SLA (IY=0)

Contenido del registro índice «IY»

(IY): 11110011 02h  
10100010 A2h

Contenido del octeto de memoria F3A2h

F3A2h: 00000010 02h

Instrucción

SLA (IY+0): 11011101 0Dh  
11001010 CBh  
00000000 00h  
00100110 26h

SRA r

OBJETO:

Desplaza a la derecha, un bit, el contenido del registro representado por «r». El bit 7 saliente se copia en el indicador de acarreo «C» y el bit 7 entrante se pone como el valor anterior del bit 7. El código de representación de «r» es el señalado más abajo. Ver Figura 9-8.

Instrucción

RR (IX+127h): 11011101 0Dh  
11001010 CBh  
01111111 7Fh  
00011110 1Eh

Contenido del octeto de memoria 7613h después de la ejecución

7613h: 10100010 A2h

Indicadores de condición después de la ejecución

S Z H P V N C  
1 0 x 0 x 0 0 0

RR (IY+d)

OBJETO:

Rota a la derecha, un bit, el octeto de memoria direccionado por el contenido del registro índice «IY» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 0 saliente se copia en el indicador de acarreo «C» y el valor del indicador de acarreo anterior se copia en el bit 7 entrante. Ver Figura 9-4.

CODIGO DE MAQUINA:

11111101 0Dh  
11001011 CBh  
00011110 1Eh

INDICADORES DE

CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 0 del octeto antes de la ejecución

P/V; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

CICLOS DE MEMORIA: 6

CICLOS DE RELOJ: 23

EJEMPLO:

RR (IY-128)

Contenido del registro índice «IY»

01110101 75h  
10010100 94h

Contenido del octeto de memoria 7514h (IY-128)

7514h: 10101010 A4h

Indicador de acarreo C=0

CODIGO DE MAQUINA:

11111101 0Dh  
11001011 CBh  
00011110 1Eh

INDICADORES DE

CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso

Indicadores de condición después de la ejecución

S Z H P V N C  
0 0 x 0 x 1 0 0

Tablas de codificación e indicadores

Antes de continuar con las instrucciones de desplazamiento, veamos, en la Figura 9-5, la tabla de codificación para las instrucciones de rotación vistas hasta ahora.

Asimismo, en la Figura 9-6, tenemos una tabla resumida de cómo afectan estas instrucciones a los indicadores, así como, el número de ciclos de memoria y reloj que emplea cada una.

SLA r

OBJETO:

Desplaza a la izquierda, un bit, el contenido del registro representado por «r». El bit 7 saliente se copia en el indicador de acarreo «C» y el bit 0 entrante se pone a cero. El código de representación de «r» es el señalado más abajo. Ver Figura 9-7.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

CODIGO DE MAQUINA:

11001011 0Bh  
00100110 0Ah



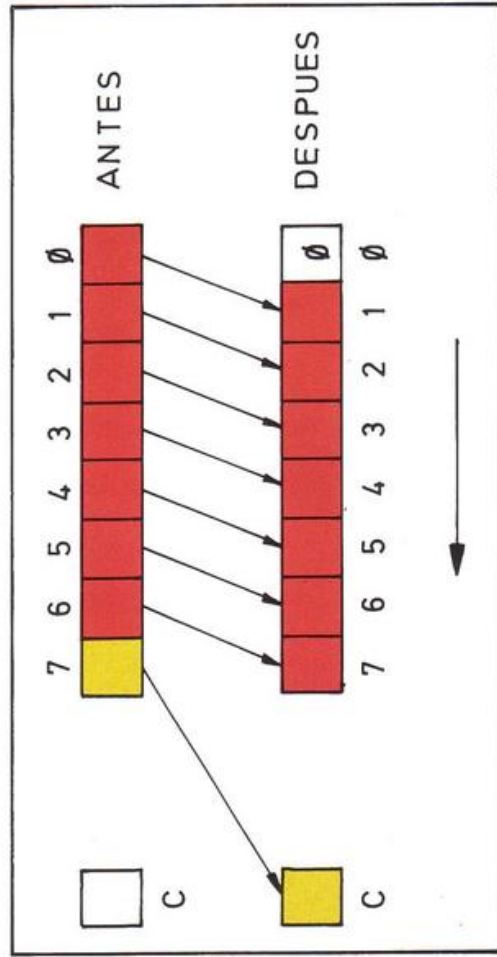


Fig. 9-7. Ilustración SLA.

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del registro «r» antes de la ejecución

PV; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

8

EJEMPLO:

SLA H

Contenido del registro «H»

(H): 11101011

SLA (HL)

OBJETO:

Desplaza a la izquierda, un bit, el octeto de memoria direccionado por el contenido del par de registros «HL». El contenido del bit 7 saliente se copia en el indicador de acarreo «C» y en el bit 0 entrante se pone cero. Ver Figura 9-7.

CODIGO DE MAQUINA:

11001010  
00100100

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del octeto antes de la ejecución

PV; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

15

EJEMPLO:

SLA (HL)

Contenido del par de registros «HL»

(H): 01100000  
(L): 00101110

Contenido del octeto de memoria 602Eh

602Eh: 10000000

Instrucción

SLA (HL): 11001010  
00100100

Contenido del octeto de memoria 602Eh después de la ejecución

602Eh: 00000000

El contenido del par de registros «HL» no ha variado

Indicadores de condición después de la ejecución

S Z H PV N C

SLA (IX + d)

OBJETO:

Desplaza a la izquierda, un bit, el octeto de memoria direccionado por el contenido del registro índice «IX» más el bit 7 del octeto antes de la ejecución

CODIGO DE MAQUINA:

11011101  
11001011  
00100110

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del octeto antes de la ejecución

PV; pone 1 - si la paridad es par;

pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

SLA (IX + 30)

Contenido del registro índice «IX»

(IX): 01110011  
10001000

Contenido del octeto de memoria 73A2h

73A2h: 01000101

Instrucción

SLA (IX + 30): 11011101  
11001010  
00011110  
00100110

Contenido del octeto de memoria 73A2h después de la ejecución

73A2h: 10001010

Indicadores de condición después de la ejecución

S Z H PV N C

SLA (IY + d)

OBJETO:

Desplaza a la izquierda, un bit, el octeto de memoria direccionado por el contenido del registro índice «IY» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127. El valor del bit 7 saliente se copia en el indicador de acarreo «C» y en el bit 0 entrante se pone cero. Ver Figura 9-7.

CODIGO DE MAQUINA:

11111101  
11001011  
00100110

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;

pone 0 - en cualquier otro caso

Z; pone 1 - si el resultado es cero;

pone 0 - en cualquier otro caso

H; pone 0 - siempre

N; pone 0 - siempre

C; pone el valor que tenía el bit 7 del octeto antes de la ejecución



mo llevan indicado el signo. Normalmente tienen un valor aritmético con el cual se puede operar. Por ejemplo el campo precio contendrá el valor de un producto; el campo longitud contendrá una medida en unas unidades previamente fijadas; etc.

Los campos alfanuméricos, contienen indistintamente números, letras y cualquier otro símbolo. Normalmente no sirven para operar aritméticamente ni tienen un significado por sí mismos. Por ejemplo el campo número de cuenta corriente es un código de números o de letras y números, pero no significa nada por sí mismo ni es un valor para operar con él.

Más adelante nos extendemos en estos temas al tratar sobre bases de datos. De momento nos limitaremos a la colocación de contenidos dentro de campos.

En los campos alfabéticos los contenidos se justifican a la izquierda. Esto quiere decir que la información empieza a colocarse en la primera posición del campo de izquierda a derecha y si quedan algunas posiciones libres, éstas estarán a la derecha. Ejemplos:

P E D R O	correcto
B L A N C O	incorrecto
J O S E	incorrecto
F R A N C I S C O	correcto

Seguramente, el lector ha

tenido que llenar algún im-

preso preparado para procesar por ordenador en el que hacían esta advertencia. En los campos numéricos los contenidos se justifican a la derecha. Esto es, las unidades se colocarán en la posición más a la derecha y se irá llenando el campo hacia la izquierda, de tal forma que si quedan algunas posiciones libres, éstas estarán a la izquierda del campo. Ejemplos:

3 4 3 7	correcto
7 4 8 2	incorrecto
2 9 6	incorrecto
1 2 3 4 5 6 7 8 9	correcto

Los campos alfanuméricos no tienen una colocación necesariamente fija, aunque lo normal es justificarlos a la derecha.

Una vez clara la colocación de los contenidos de los campos, nos centraremos en los de tipo numérico que es con los que, de alguna manera, podemos operar aritméticamente.

Como ya se comentó al comienzo de este grupo de instrucciones, éstas pueden servir para multiplicar y dividir. Desde luego no es el objetivo para el cual están hechas, pero dado que el microprocesador Z-80 carece de dichas operaciones, puede ser de gran utilidad conocer esta propiedad.

Comencemos por recordar el valor de cada unidad en el sistema de numeración deci-

mal. La unidad «n» vale «n» por 10 elevado a cero; la decena «n» vale «n» por 10 elevado a uno; la centena «n» vale «n» por 10 elevado a dos; etc. Ejemplo:

5 * 10 <sup>0</sup> = 5	1 = 5
7 * 10 <sup>1</sup> = 70	10 = 70
4 * 10 <sup>2</sup> = 400	100 = 400
3 * 10 <sup>3</sup> = 3000	1000 = 3000
total 3475	

De la misma manera, en un sistema de base 2 (binario) el valor de cada bit está en función del lugar que ocupa. El bit más a la derecha valdrá «n» por 2 elevado a cero, el siguiente «n» por 2 elevado a uno, el siguiente «n» por 2 elevado a dos, etc. Pero dado que el valor de un bit sólo puede ser cero o uno y cualquier número multiplicado por cero es cero se puede tener en cuenta sólo los bits que tengan valor uno y de la forma de 2 elevado a «e» representa el lugar que ocupa. Ejemplo:

1 1 0 1 0 1 1	
2 <sup>0</sup> = 1	
2 <sup>1</sup> = 2	
2 <sup>4</sup> = 16	
2 <sup>6</sup> = 64	
2 <sup>7</sup> = 128	
total	211

Efectivamente, el número binario 1101011 (D3h) tiene el valor decimal de 211.

Cuando a un número decimal le añadimos un cero a la derecha, éste queda multiplicado por diez. Por ejemplo si a 30 le añadimos un cero a la derecha queda 300 = 30 x 10. Supongamos

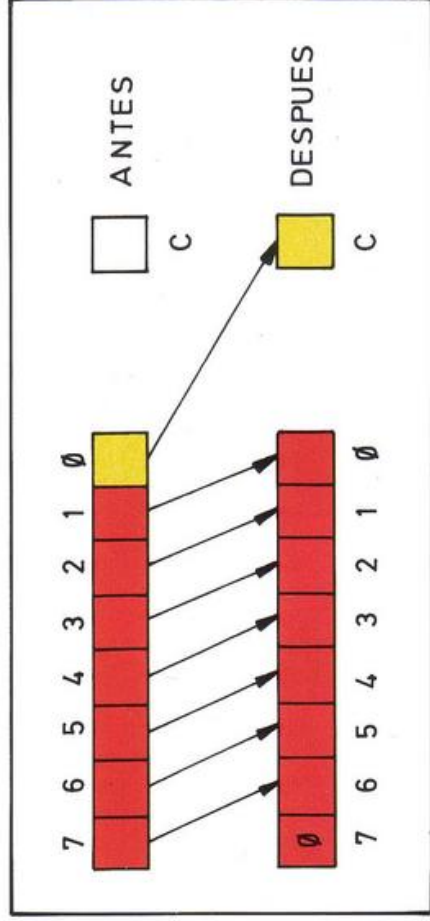


Fig. 9-9. Instrucción SRL.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

2

CICLOS DE RELOJ:

8

EJEMPLO:

SRL E
-------

CODIGO DE MAQUINA:

1 1 0 0 1 0 1 1	Cbh
0 0 1 1 1 1	

Contenido del registro «E»

1 1 1 1 1 1 1 1	Fh
-----------------	----

CODIGO DE MAQUINA:

1 1 0 0 1 0 1 1	Cbh
0 0 1 1 1 1 1 0	3Hh

OBJETO:

Desplaza a la derecha, un bit, el octeto de memoria direccionado por el contenido del par de registros «HL». El contenido del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante se pone un 1. Ver Figura 9-9.

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del registro r antes de la ejecución  
P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del octeto antes de la ejecución  
P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

Contenido del registro «E» después de la ejecución

0 1 1 1 1 1 1 1	7Hh
-----------------	-----

Indicadores de condición después de la ejecución

S	Z	H	P/V	N	C
0	0	x	0	0	1

SRL (HL)

CICLOS DE MEMORIA:



# CICLOS DE MEMORIA:

4

# CICLOS DE RELOJ:

15

# EJEMPLO:

SRL (HL)	1 0 1 0 0 1 0 1	A5h
----------	-----------------	-----

Contenido del par de registros «HL»

(H)	1 0 1 0 0 1 0 1	A5h
(L)	1 0 0 1 0 0 1 1	93h

Contenido del octeto de memoria A593h

A593h:	1 0 0 1 0 0 1 0	92h
--------	-----------------	-----

# Instrucción

SRL (HL)	1 1 0 0 1 0 1 0	CBh
	0 0 1 1 1 1 1 0	3Eh

Contenido del octeto de memoria A593h después de la ejecución

A593h:	0 1 0 0 1 0 0 1	49h
--------	-----------------	-----

El contenido del par de registros «HL» no ha variado. Indicadores de condición después de la ejecución

S Z H P V N C	0 0 x 0 x 0 0 0
---------------	-----------------

# SRL (IX + d)

# OBJETO:

Desplaza a la derecha, un bit, el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los

valores desde —128 a + 127. El valor del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante se pone el valor 1. Ver Figura 9-9.

# CODIGO DE MAQUINA:

1 1 0 1 1 1 0 1	DDh
1 1 0 0 1 0 1 1	CBh
0 0 1 1 1 1 1 0	3Eh

# INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;  
 pone 0 - en cualquier otro caso  
 Z; pone 1 - si el resultado es cero;  
 pone 0 - en cualquier otro caso

# Instrucción

SRL (HL)	1 1 0 0 1 0 1 0	CBh
	0 0 1 1 1 1 1 0	3Eh

Contenido del octeto de memoria A593h después de la ejecución

A593h:	0 1 0 0 1 0 0 1	49h
--------	-----------------	-----

El contenido del par de registros «HL» no ha variado. Indicadores de condición después de la ejecución

S Z H P V N C	0 0 x 0 x 0 0 0
---------------	-----------------

# SRL (IX + d)

# OBJETO:

Desplaza a la derecha, un bit, el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los

F39Ah:	0 1 0 1 0 1 0 1	55h
--------	-----------------	-----

# Instrucción

SRL (IX + 10):	1 1 0 1 1 1 0 1	DD
	1 1 0 0 1 0 1 0	CBh
	0 0 0 0 1 0 1 0	0Ah
	0 0 1 1 1 1 1 0	3Eh

Contenido del octeto de memoria F39Ah después de la ejecución

F39Ah:	0 0 1 0 1 0 1 0	2Ah
--------	-----------------	-----

Indicadores de condición después de la ejecución

S Z H P V N C	0 0 x 0 x 0 0 1
---------------	-----------------

# SRL (IX + d)

# OBJETO:

Desplaza a la derecha, un bit, el octeto de memoria direccionado por el contenido del registro índice «Y» más el entero de desplazamiento «d», el cual puede adquirir los valores desde —128 a + 127. El valor del bit 0 saliente se copia en el indicador de acarreo «C» y en el bit 7 entrante se pone el valor 1. Ver Figura 9-9.

# CODIGO DE MAQUINA:

1 1 1 1 1 1 0 1	DDh
1 1 0 0 1 0 1 1	CBh
0 0 1 1 1 1 1 0	3Eh

# INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el resultado es negativo;

# INSTRUCCIONES DE DESPLAZAMIENTO

NEMONICO	INDICADORES								No. DE BYTES	CICLOS	
	S	Z	X	H	P	V	N	C		MEM.	REL.
SRL r	0	0	x	0	x	P	0	0	2	2	8
SRL (HL)	0	0	x	0	x	P	0	0	2	4	15
SRL (IX+d)	0	0	x	0	x	P	0	0	4	6	23
SRL (IY+d)	0	0	x	0	x	P	0	0	4	6	23
SRA s	0	0	x	0	x	P	0	0			
SRL s	0	0	x	0	x	P	0	0			
RLD	0	0	x	0	x	P	0	0	2	5	18
RRD	0	0	x	0	x	P	0	0	2	5	18

# NOTAS:

- Los signos tienen el siguiente significado:  
 "0": El indicador cambia de valor de acuerdo con el resultado de la instrucción.  
 "x": El bit adquiere un estado indeterminado.  
 ".": El indicador no es afectado por la instrucción y conserva su anterior contenido.  
 "0": El indicador se pone siempre a "cero".  
 "P": El indicador "P/V" actúa como indicador de paridad.
- La letra "r" indica cualquiera de los registros: "A", "B", "C", "D", "E", "H" ó "L".
- La letra "s" indica cualquiera de los operandos: "r", "(HL)", "(IX+d)" ó "(IY+d)".

Fig. 9-13. Tabla resumida de indicadores y ciclos para las instrucciones de desplazamiento.

dad, el sueldo que gana o el número de su cuenta corriente.

Por su contenido los campos se suelen dividir en alfabéticos, numéricos y alfanuméricos. Los campos alfabé-

ticos, como su mismo nombre indica, contienen letras y a lo sumo signos de puntuación; normalmente contienen palabras en el idioma en uso y tienen significado por sí mismos. Por ejemplo, el cam-

po de nombre en un registro de datos contendrá nombre y apellidos de personas; el campo de color contendrá nombres de colores; etc. Los campos numéricos, contienen números y a lo su-



bla. Hacemos las rotaciones con RRD y vamos incrementando «HL» hasta llegar a la dirección más alta.

Esto no sólo sirve para meter o sacar datos, también puede ser útil si queremos operar, de alguna forma, los datos de la tabla y volverlos a almacenar en las mismas posiciones que ocupaban. En cualquier caso, siempre que utilizemos números en formato BCD, tendremos que recurrir al uso de las instrucciones RLD y RRD.

### Tablas de codificación

Ya hemos visto todas las instrucciones de desplazamiento que posee el Z-80. Ahora, y antes de pasar a los ejemplos, veamos la tabla que nos va a permitir codificarlas en decimal o Hexa, y una tabla resumida de cómo afectan a los indicadores y los ciclos de memoria y reloj que emplean. La primera está en la Figura 9-12 y la segunda, en la Figura 9-13.

### Multiplicación y división con instrucciones de rotación y desplazamiento

Antes, prometimos que íbamos a hablar de la forma en que se puede multiplicar y dividir utilizando rotaciones. Bien, ahora cumplimos la promesa:

La primera consideración a tener en cuenta es cómo se colocan los contenidos de los campos dentro de éstos.

Se recordará que un campo es una serie de octetos consecutivos con una unidad de información. Por ejemplo el nombre de una persona, su número de carnet de identi-

### INSTRUCCIONES DE DESPLAZAMIENTO

Código	Fuente	Hexadecimal	Decimal
SLA A		CB, 27	203, 39
SLA B		CB, 20	203, 32
SLA C		CB, 21	203, 33
SLA D		CB, 22	203, 34
SLA E		CB, 23	203, 35
SLA H		CB, 24	203, 36
SLA L		CB, 25	203, 37
SLA (HL)		CB, 26	203, 38
SLA (IX+d)		DD, CB, d, 26	221, 203, d, 38
SLA (IY+d)		FD, CB, d, 26	253, 203, d, 38
SRA A		CB, 2F	203, 47
SRA B		CB, 28	203, 40
SRA C		CB, 29	203, 41
SRA D		CB, 2A	203, 42
SRA E		CB, 2B	203, 43
SRA H		CB, 2C	203, 44
SRA L		CB, 2D	203, 45
SRA (HL)		CB, 2E	203, 46
SRA (IX+d)		DD, CB, d, 2E	221, 203, d, 46
SRA (IY+d)		FD, CB, d, 2E	253, 203, d, 46
SRL A		CB, 3F	203, 63
SRL B		CB, 38	203, 56
SRL C		CB, 39	203, 57
SRL D		CB, 3A	203, 58
SRL E		CB, 3B	203, 59
SRL H		CB, 3C	203, 60
SRL L		CB, 3D	203, 61
SRL (HL)		CB, 3E	203, 62
SRL (IX+d)		DD, CB, d, 3E	221, 203, d, 62
SRL (IY+d)		FD, CB, d, 3E	253, 203, d, 62
RLD		ED, 6F	237, 111
RRD		ED, 67	237, 103

Fig. 9-12. Tabla de codificación para las instrucciones de desplazamiento.

pone 0 - en cualquier otro caso  
Z; pone 1 - si el resultado es cero;  
pone 0 - en cualquier otro caso

H; pone 0 - siempre  
N; pone 0 - siempre  
C; pone el valor que tenía el bit 0 del octeto antes de la ejecución  
P/V; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

6

CICLOS DE RELOJ:

23

EJEMPLO:

SRA (IY+5)

Contenido del registro índice «IY»

IY: 01110101 75h  
10010101 95h

Contenido del octeto de memoria 759Ah (IY+5):

759Ah: 00000001 01h

Instrucción

SRL (IY+5):  
01110101 00h  
11001010 CBh  
00000101 05h  
00111110 3Ch

Contenido del octeto de memoria 759Ah después de la ejecución

759Ah: 00000000 00h

Indicadores de condición después de la ejecución

S Z H PV N C  
01x0x101

Las dos instrucciones de desplazamiento que nos quedan por ver son bastante atípicas. Su principal cometido es trabajar con números en BCD (decimal codificado en binario). Tienen la particularidad de que desplazan bits de 4 en 4 y no de 1 en 1 como las vistas hasta ahora. Además, el desplazamiento se produce entre el acumulador y la posición de memoria direccionada por el registro «HL», lo que nos va a permitir ir leyendo, uno a uno, una serie de dígitos en BCD: recuerda que un dígito BCD ocupa cuatro bits, es decir, en un octeto caben dos dígitos.

Cuando hablamos de la instrucción DAA, decíamos que servía para ajustar a BCD el resultado de una operación aritmética, pero era imprescindible que los datos, antes de la operación, estuvieran ya en BCD. Por tanto, es evidente que necesitábamos un sistema que nos permitiera introducir en memoria o sacar de la misma, dígitos BCD. Supongamos que tenemos, en memoria, una serie de octetos que contienen datos en BCD y supongamos, también, que tenemos que sacarlos a un canal de comunicación a través del registro «A» (lo más frecuente es que sea así); en ese caso, podemos direccionar mediante «HL» cualquiera de los dos extremos de la lista de datos, e ir usando repetidamente estas instrucciones para ir pasando los datos uno a uno (medio octeto cada vez) al registro «A».

Como en casi todas las

instrucciones del Z-80, estas también son simétricas, es decir, una nos permite leer los datos de «arriba» a «abajo», y la otra, de «abajo» a «arriba». De cualquier forma, no es frecuente que trabajemos en BCD con el Spectrum, ya que tenemos maravillosas rutinas del sistema operativo para gestionar números en decimal, pero no obstante, es conveniente conocer el manejo de estas instrucciones ya que, en algunos casos, es posible que les encontremos aplicación práctica.

RLD

OBJETO:

Copia el valor de los cuatro bits de orden inferior de la posición de memoria, direccionada por el contenido del par de registros «HL», en los cuatro bits de orden superior del mismo octeto; el valor anterior de los cuatro bits de orden superior los copia en los cuatro bits de orden inferior del registro acumulador; finalmente, el valor anterior de los cuatro bits de orden inferior del registro acumulador los copia en los cuatro bits de orden inferior del octeto. Los cuatro bits de orden superior del registro acumulador permanecen inalterados. Ver Figura 9-10.

CODIGO DE MAQUINA:

EDh 11101101  
6Fh 01101111

INDICADORES DE CONDICION QUE AFECTA:

CODIGO MAQUINA 227



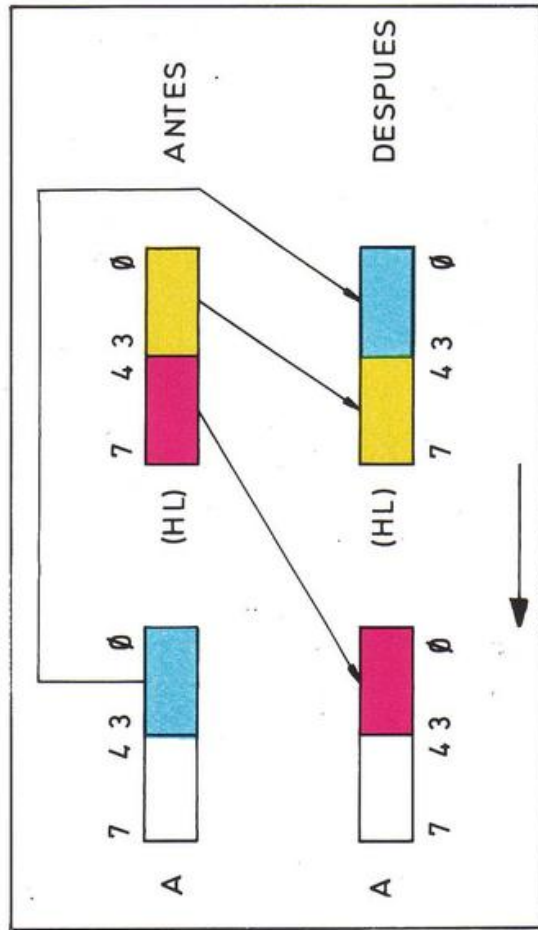


Fig. 9-10. Instrucción RRD.

S; pone 1 - si el registro «A» es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el registro «A» es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
P/V; pone 1 - si la paridad del registro «A» es par;  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

18

EJEMPLO:



Contenido del registro «A»



Indicadores de condición después de la ejecución



Contenido del par de registros «HL»



Contenido de la posición de memoria 9243h



Instrucción



Contenido del registro «A» después de la ejecución



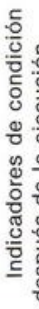
Contenido del registro «A» después de la ejecución



Contenido del octeto de memoria 9243h después de la ejecución



Contenido del registro «A»



Indicadores de condición después de la ejecución



Contenido del par de registros «HL»



OBJETO:

Copia el valor de los cuatro bits de orden superior de la posición de memoria, direccionada por el contenido del par de registros «HL», en los cuatro bits de orden inferior del mismo octeto; el valor anterior de los cuatro bits de orden inferior los copia en los cuatro bits de orden inferior del registro acumulador; finalmente, el valor anterior de los cuatro bits de orden inferior del registro acumulador los copia en los cuatro bits de orden inferior del octeto de memoria 9243h después de la ejecución. Los cuatro bits de orden superior del registro acumulador permanecen inalterados. Ver Figura 9-11.

CODIGO DE MAQUINA:



Indicadores de condición después de la ejecución



Contenido del par de registros «HL»

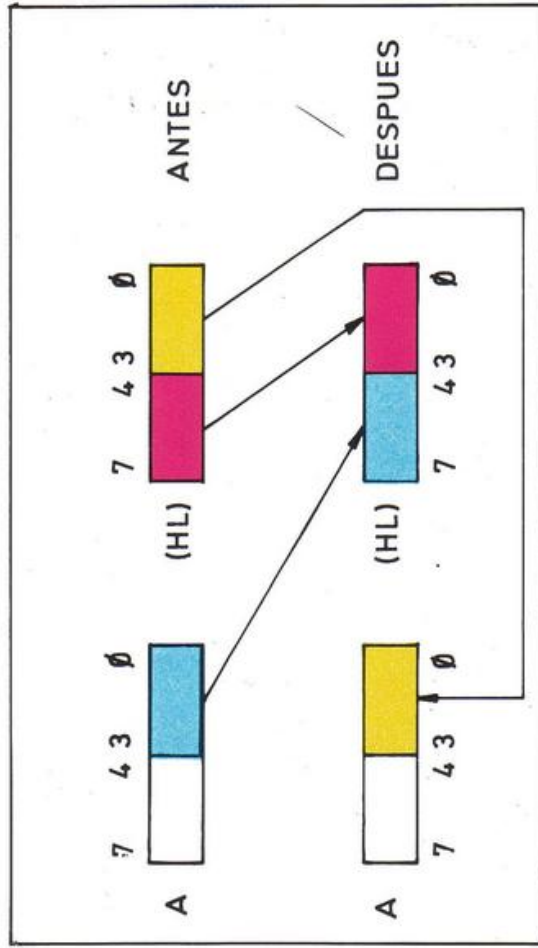


Fig. 9-10. Instrucción RRD.

INDICADORES DE CONDICION QUE AFECTA:

S; pone 1 - si el registro «A» es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el registro «A» es cero;  
pone 0 - en cualquier otro caso  
H; pone 0 - siempre  
N; pone 0 - siempre  
P/V; pone 1 - si la paridad del registro «A» es par;  
pone 0 - en cualquier otro caso

CICLOS DE MEMORIA:

5

CICLOS DE RELOJ:

18

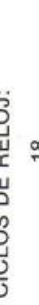
EJEMPLO:



Contenido del registro «A»



Indicadores de condición después de la ejecución



Contenido del par de registros «HL»

Contenido del par de registros «HL»



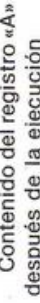
Contenido de la posición de memoria 7743h



Instrucción



Contenido del registro «A» después de la ejecución



Contenido del octeto de memoria 7743h después de la ejecución



Indicadores de condición después de la ejecución



Contenido del par de registros «HL»

S Z H P V N C  
1 0 x 0 x 0 0 x

Como se ve fácilmente en las Figuras 9-10 y 9-11, la instrucción RRD nos permite leer los datos BCD de «arriba» a «abajo», es decir, de direcciones más altas a direcciones más bajas. Vamos a ver el procedimiento: cargamos en «A» un cero, y en «HL» la dirección más alta de la tabla de datos. Hacemos una rotación RLD; ahora tenemos el primer dato en «A»; podemos llamar a una rutina que haga algo con él. Hacemos otra rotación, ahora tenemos el segundo dato; llamamos, otra vez a la rutina. Hacemos una tercera rotación y ya tenemos los dos datos como al principio. Ahora, decrementamos «HL» y volvemos a iniciar el proceso; así, hasta la dirección más baja de la tabla.

Si queremos leer la tabla de «abajo» a «arriba», empecemos por cargar «HL» con la dirección más baja de la ta-







Cada uno de sus bits pasarían de valer 2 elevado a «0», a valer 2 elevado a («e» + 1):

0 0 0 1 0	-2
1 1 1 1 1	
0 0 1 0 0	-4
1 1 1 1 1	

Desplazarlo otra posición equivaldría a multiplicarlo otra vez por dos, esto es multiplicar el número original por  $4 = 2 \times 2$ ; un nuevo desplazamiento sería  $8 = 2 \times 2 \times 2$  y así sucesivamente. Lo que se observa es que un nuevo desplazamiento implica multiplicar por la siguiente potencia de 2, esto es 2, 4, 8, 16, 32, etc. Por lo tanto, podríamos enunciar la siguiente regla: para multiplicar un número binario por una potencia de 2 se desplaza el número a la izquierda tantas veces como valor tenga el exponente.

$$\text{NUMERO-BINARIO} \times 2^n = \text{NUMERO-BINARIO} \text{ desplazado } n \text{ veces}$$

#### EJEMPLOS:

$$\begin{array}{r} 0001100 \times 2^3 = \\ -1100000 : 12 \times 8 = 96 \\ 0011010 \times 2^2 = \\ -1101000 : 26 \times 4 = 104 \end{array}$$

Pero no siempre se quiere multiplicar un número por una potencia de dos, es más, la mayoría de las veces no es así.

En un micro-procesador que carece de instrucción de multiplicar, una forma inmediata de solucionar el problema es sumar el multiplicando tantas veces como valor tenga el multiplicador; por ejemplo  $30 \times 3$  sería

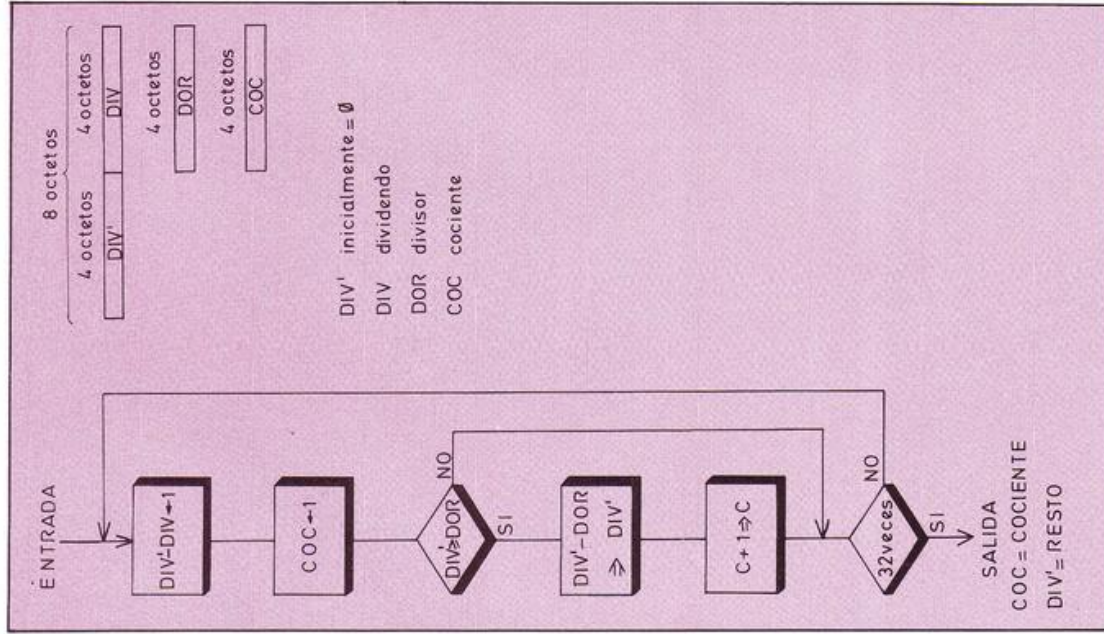


Fig. 9-15. División por desplazamientos.

$30 + 30 + 30$ . En el caso de multiplicadores de poco valor, ésta sería la solución más sencilla, se ejecutarían poco más de tres instrucciones. Pero vamos a ver otra técnica que resulta mejor en el caso de multiplicadores grandes.

de igual forma que la primera. Finalmente, se acaba con la tercera zona que está organizada de la misma forma que las dos anteriores. Esto es lo que nos permitía intercambiarlas (en la rutina para intercambiar zonas de pantalla) sin ningún problema; sin embargo, no es tan fácil intercambiar líneas, ya que sus scans no son consecutivos.

Si ha entendido a la perfección las explicaciones dadas hasta aquí sobre el archivo de pantalla, debe tener usted un coeficiente intelectual de super-dotado. Lo mejor, en cualquier caso, es que lo vuelva a leer detenidamente y, si es necesario, que se ayude de lápiz y papel para entenderlo mejor. La organización del archivo de pantalla no es nada fácil de comprender, pero es imprescindible manejarlo a la perfección para hacer rutinas de código máquina que trabajen sobre ella.

Pudiera parecer que con esta organización tan caótica, manejar la pantalla en código máquina es cosa de locos. Sin embargo, ahora veremos que resulta extremadamente fácil; es más, resulta más sencillo así que si los scans fueran consecutivos.

Ahora que, tras arduo esfuerzo, hemos sido capaces de comprender cómo está organizada la pantalla, vamos a olvidarnos de ello durante un momento. Lo que a nosotros nos interesa es ver de qué forma podemos hallar las direcciones de los 8 bytes que componen un determinado carácter, partiendo de sus coordenadas. Como ya sabe el lector, el archivo de pantalla ocupa los 6 primeros Kibytes de la RAM, es decir,



Fig. 9-16. Formato de una dirección en el archivo de pantalla.

está colocado a partir de la dirección 16384 (4000h).

Por una curiosa «coincidencia» totalmente intencionada, podemos componer la dirección de cualquier octeto de la pantalla con sólo colocar de determinada forma los bits individuales de cada una de sus coordenadas. Vamos a verlo con sumo detalle porque es muy importante:

Dado que los números de línea y columna no pueden valer más de 23 y 31 respectivamente, cada uno de ellos sólo nos ocupa 5 bits. Supongamos que tenemos en «DE» las coordenadas de un determinado carácter, el registro «D» contendrá el número de línea que podrá estar comprendido entre 0 y 23, por tanto, sus tres bits de más a la izquierda serán «ceros». Llamaremos desde «l0» hasta «l4»

a los cinco bits que nos indican la línea. De la misma forma, el registro «E» tendrá sus tres bits de la izquierda a «c0» y llamaremos desde «c0» hasta «c4» a los cinco bits de la derecha que nos van a determinar la columna.

Queda un último dato: cada carácter tiene 8 octetos que pertenecen a 8 scans diferentes, y cada octeto tiene su dirección. Normalmente, nos interesará hallar la dirección del primero de estos octetos, pero no siempre así que será mejor añadir un dato más. Dado que se trata de 8 posibilidades, podemos representarlo con 3 bits, «s00» será el octeto perteneciente al primer scan de la línea y «111» el que pertenece al último scan de la misma. Vamos a llamarlos desde «s0» hasta «s2». Tenemos:

Número de línea:	0	0	0	14	13	12	11	10
Número de columna:	0	0	0	c4	c3	c2	c1	c0
Número de scan:						52	51	50



Los atributos estamos buscando, «D» deberá contener la línea y «E» la columna. A la salida de la rutina, tendremos, en «HL», la dirección del octeto del archivo de atributos, correspondiente a este carácter. El listado Assembler de «ATR» es el siguiente:

```

100 ATR LD A,D
110 SRA A
120 SRA A
130 SRA A
140 ADD A,#58
150 LD H,A
160 LD A,D
170 AND #7
180 RRC A
190 RRC A
200 RRC A
210 ADD A,E
220 LD L,A
230 RET

```

Vamos a ver cómo funciona con un ejemplo: Supongamos que queremos hallar la dirección de los atributos correspondientes al carácter cuyas coordenadas son (12,15), es decir, línea 12 (0Ch), columna 15 (0Fh). Veamos los pasos:

```

LD A,D ; 0C -> A
SRA A ; A/2
SRA A ; A/2
SRA A ; A/2
ADD A,#58 ; A + #58
LD H,A ; A -> H
LD A,D ; D -> A
AND #7 ; A and #7
RRC A ; A/2
RRC A ; A/2
RRC A ; A/2
ADD A,E ; A + E
LD L,A ; A -> L
RET

```

En el retorno, «HL» contiene el número # 598F, es decir, 22927 en decimal. Vemos que, efectivamente:

$$22528 + 12 \times 32 + 15 = 22927$$

Por lo que 22927 es la dirección de los atributos correspondientes al carácter de coordenadas (12,15). Esta rutina trabaja perfectamente siempre que la línea esté comprendida entre 0 y 23; y la columna lo esté entre 0 y 31.

Antes de proseguir, conviene hacer referencia a la forma en que se almacenan los atributos, de un determinado carácter, dentro de cada byte del fichero de atributos. Empezando por la derecha, los tres primeros bytes almacenan el color de la tinta; los tres segundos, el color del papel; el séptimo, el flag de brillo y el octavo, el flag de padeo. Cuando la ULA va leyendo el fichero de pantalla para mandarlo al televisor, lee también los datos del fichero de atributos para mandar correctamente los colores de cada pixel.

Una vez vista la organización del fichero de atributos,

```

A= 00001100 = #0C
A= 00000110 = #06
A= 00000011 = #03
A= 00000001 = #01
A= 01011001 = #59
H= 01011001 = #59
A= 00001100 = #0C
A= 00000100 = #04
A= 00000010 = #02
A= 00000001 = #01
A= 10000000 = #80
A= 10001111 = #8F
L= 10001111 = #8F

```

vamos a ver la del fichero de pantalla. En este caso, las cosas no son tan sencillas como en el anterior. Sabemos que cada scan ocupa 32 bytes de memoria; parecería lógico que las direcciones de memoria de scans consecutivos fueran, también, consecutivas. Por desgracia, esto no es así.

Podemos considerar al archivo de pantalla como dividido en tres zonas de 2048 bytes cada una. La primera, correspondería a las 8 primeras líneas (0 a 7); la segunda, a las 8 segundas (8 a 15) y la tercera, a las 8 terceras (16 a 23). Para nuestros efectos, la pantalla consta de 24 líneas, así que podemos olvidarnos de que las dos líneas inferiores no son accesibles. Desde código máquina podremos acceder por igual a cualquier línea de la pantalla.

Ahora, vamos a estudiar la disposición de cada una de estas tres zonas, empezando por la primera. Cada línea se compone de 8 scans; pero no son consecutivos. Los primeros 32 bytes del archivo, contienen el primer scan de la primera línea (línea 0); los 32 siguientes, contienen el primer scan de la segunda línea; y así sucesivamente hasta el octavo grupo de 32 bytes que contiene el primer scan de la línea 7.

El noveno grupo de 32 bytes contiene el segundo scan de la primera línea, el décimo grupo, contiene el segundo scan de la segunda y así sucesivamente, hasta llegar al último grupo que contendrá el último scan de la octava línea (línea 7).

A partir de aquí, se comienza con la segunda zona de la pantalla que está organizada

mos MUL y multiplicador que llamaremos MOR, podemos descomponer el multiplicador en sus sumandos; y los sumandos pueden ser el valor relativo de cada bit; el resultado lo dejaremos en un campo llamado RES.

La técnica consiste en ir multiplicando MUL por cada sumando de MOR, que siempre será una potencia de dos, lo cual podemos hacer simplemente desplazando a la izquierda. Veamos un ejemplo:

```

30 x 26
30 = 11110 binario
26 = 11010 binario
11010 * 2^1 = 2^3 + 2^4 = 2 + 8 = 10
11110 * 2^1 = 111100
11110 * 2^2 = 1111000
11110 * 2^4 = 11110000
suma total en RES 1100001100 = 780
Efectivamente 30 x 26 = 780.

```

El lector es posible que ya se diera cuenta que es la misma técnica que se sigue cuando se hace una operación de números de varias cifras en un papel, sólo que los desplazamientos se realizan inconscientemente.

#### EJEMPLO:

```

3456 x 752
+1
3456
x 752
-----
8912
17280
24192
2598912
desplazamiento
desplazamiento

```

#### RECOMENDACIONES:

a) Si se opera con un solo

octeto, ni los multiplicandos ni el resultado podrán superar 256 decimal.

b) Si se opera con números que ocupan más de un octeto, hay que tener en cuenta el bit que sale por la izquierda en los desplazamientos y el acarreo en las sumas; lo cual es posible ya que estas instrucciones (sumas y desplazamientos) dejan ambos en el indicador de acarreo «C».

c) Es muy conveniente poner límites fijos a los multiplicandos, en función de las necesidades. Lo más útil es que los dos tengan el mismo tamaño y el resultado la suma de los dos tamaños. De tal forma que si se decide que ambos ocupen un máximo de cuatro octetos el resultado estará dimensionado con ocho.

Ver Figura 9-14 que es el organigrama de una rutina de multiplicación de dos números MUL y MOR de 4 octetos de tamaño dejando el resultado en RES que tiene 8 octetos de tamaño. Los desplazamientos se señalan con una flecha indicando el sentido y un número indicando la cantidad. El campo RES se crea inicialmente a ceros. Tenga en cuenta que, tanto en las sumas sobre RES como en los desplazamientos de MUL y MOR, hay que manejar el indicador de acarreo «C» ya que los números están en más de un octeto.

En la división, podemos obtener soluciones parecidas así que vamos a analizar, paso a paso, el mecanismo, igual que con la multiplicación.

Recordemos que para dividir en decimal entre la unidad seguida de ceros o sea, po-

tencias de diez, se corre una coma tantos lugares a la izquierda como ceros tenga la unidad, lo que es lo mismo, como valor tenga el exponente de la base, en este caso el 10. Esto es lo mismo que desplazar el número a la derecha, y el resto serán los números salientes justificados a la derecha. Ejemplo:

```

400:10
4 0 0
1 1 1 1 1 1 1
4 0
1 1 1 1 1 1 1
374:100
3 7 4
1 1 1 1 1 1 1
3
1 1 1 1 1 1 1
resto
7 4
1 1 1 1 1 1 1

```

Por el mismo motivo, si desplazamos un número binario una posición a la derecha, este número queda dividido por la base. Cada uno de sus bits pasarían de valer 2 elevado a «e», a valer 2 elevado a («e»-1).

```

0 1 0 1 0 = 10
1 1 1 1 1 1 1
0 0 1 0 1 = 5
1 1 1 1 1 1 1

```

Desplazarlo otra posición equivaldría a dividirlo otra vez por dos, esto es, dividir el número original por  $4 = 2 \times 2$ , y así sucesivamente. Lo que se observa es que un nuevo desplazamiento implica dividir por la siguiente potencia de 2,



esto es 2, 4, 8, 16, 32, etc. Por lo tanto podríamos enunciar la siguiente regla: para dividir un número binario por una potencia de 2 se desplaza el número a la derecha tantas veces como valor tenga el exponente, y el resto serán los bits salientes por la derecha justificados a la derecha.

NUMERO_BINARIO: 2 <sup>5</sup> =
NUMERO_BINARIO → «5» veces
Ejemplos:
0 1 1 1 1 0 0 : 2 <sup>5</sup> =
0 0 0 0 1 1 1 resto 1 0 0
688 - 7 resto 4
0 0 1 0 1 0 0 : 2 <sup>5</sup> =
0 0 0 0 1 1 0 resto 0
20 - 4 = 5 resto 0

Al igual que en la multiplicación, no siempre se quiere multiplicar un número que sea una potencia de dos, es más, la mayoría de las veces no es así.

En un micro-procesador que carece de instrucción de dividir, una forma inmediata de solucionar el problema es restar el divisor al dividendo hasta llegar a cero o a un número menor que el divisor, entonces el cociente sería el número de restas. Ejemplo 40:10 habría que restar cuatro veces 10 hasta alcanzar el valor cero. Resultan pocas ejecuciones cuando la relación entre el dividendo y el divisor es pequeña y muchas cuando es grande.

Es necesario encontrar una técnica menos costosa, para lo cual empezaremos por observar cómo se realiza una operación de dividir números decimales de varias cifras en un papel:

c) Definiremos un campo DIV' de igual tamaño que DIV y pegado a él por la izquierda de forma que entre los dos DIV' DIV formen un campo doble.

Y ahora, siguiendo los pasos parecidos a la división decimal desplazaremos a la izquierda DIV' DIV hasta tener en DIV' un valor igual o mayor que DOR (divisor).

En este momento el número por el que hay que multiplicar el divisor siempre es 1 el cual se pone en el campo COC (cociente).

A partir de este punto se irán desplazando COC y DIV' DIV hasta alcanzar el valor del divisor DOR, momento en que se restará de DIV' el divisor y se sumará uno al cociente.

Parece un lío pero pruebe con un ejemplo, nunca falla; primero uno sencillo, 1001010:110, y después uno más complicado como 101101101010101:1011001.

El resto será el valor que quede en DIV' cuando no haya más bits en el dividendo.

Ver organigrama de la Figura 9-11 hecho para campos fijos de 4 octetos con los números justificados a la derecha.

## Los archivos de pantalla y atributos

Antes de estudiar cómo el Spectrum almacena, en memoria, la imagen que vemos en la pantalla, tal vez sea conveniente echar un vistazo a la forma en que se genera una imagen de televisión.

Como casi todos los lectores sabrán, la imagen de un televisor está compuesta por líneas horizontales. En la norma de televisión europea, se

utilizan 625 líneas para formar un cuadro, aunque no todas entran en la composición de la imagen. El haz de electrones barre la imagen de izquierda a derecha y de arriba a abajo, por tanto, cuando termina de formar un cuadro, se encuentra en el extremo inferior derecho. Antes de formar el siguiente cuadro, el haz debe retornar al extremo superior izquierdo en la operación que se denomina «retorno de cuadro». Durante esta operación, se pierden algunas líneas, ya que el barrido horizontal continúa funcionando.

En el caso del Spectrum, aún se pierden unas cuantas líneas más en las zonas superior e inferior del «BORDER», así como parte de las restantes en los extremos derecho e izquierdo de la pantalla (el resto del «BORDER»). Al final, quedan 384 líneas en la parte de la pantalla que denominamos «PAPER». De estas líneas, el Spectrum sólo utiliza la mitad, es decir, una si y una no. Por tanto, la imagen enviada por el Spectrum al televisor consta de 192 líneas.

A efectos de terminología, llamaremos «SCAN» a cada una de estas 192 líneas. Tendremos, por tanto, 192 «scans» en pantalla (la palabra «scan» significa, en inglés: «barrido»; utilizamos la terminología inglesa por ser la de uso más común en todo el mundo).

Cada scan de pantalla consta de 256 pixels. Un «pixel» es un elemento de imagen, un punto, que puede tener el color del papel o el de la tinta. Cuando hacemos un «PLOT» en Basic, ponemos un pixel del color de la tinta.

Cada pixel está controlado por un bit del archivo de pantalla de forma que, si el bit co-

respondiente es un «1», el pixel tendrá color de tinta y si es un «0», lo tendrá de papel.

Los colores de la tinta y el papel, así como los atributos de brillo y parpadeo, se encuentran en otra zona de memoria denominada «archivo de atributos». Llamamos «carácter» a un conjunto de 64 pixels, colocados en una matriz de 8 x 8. Por tanto, cada carácter ocupa 8 pixels de 8 scans consecutivos. Los 8 pixels de cada scan dentro de un carácter, están agrupados en una determinada posición de memoria, es decir, componen un octeto. Por tanto, cada carácter consta de 8 octetos; pero estos no son consecutivos. Por otro lado, cada elemento del archivo de atributos afecta a un carácter, por ello, hay 8 veces menos octetos en el archivo de atributos que en el de pantalla.

Si llamamos «línea» a un conjunto de 32 caracteres colocados uno al lado del otro, y «columna» a un conjunto de 24 caracteres colocados uno encima del otro, podemos decir que la pantalla del Spectrum tiene 24 líneas y 32 columnas, es decir, 24 x 32 = 768 caracteres. Por tanto, el archivo de atributos tendrá 768 bytes (u octetos) y el de pantalla, 768 x 8 = 6144 bytes. Si al llegar a este punto, no ha comprendido algo de lo anterior, vuelva a leerlo más despacio antes de proseguir.

Vamos a empezar por describir cómo está colocado el archivo de atributos ya que es el más fácil, luego, pasaremos al de pantalla que es bastante más complejo.

Los 768 bytes colocados a partir de la dirección de memoria 22525 (5800h) inclusive, constituyen el archivo de atri-

butos. El primer byte corresponde al carácter del ángulo superior izquierdo (coordenada 0,0). El segundo, al siguiente de esa línea (coordenada 0,1) y así sucesivamente hasta el último que corresponde al carácter del ángulo inferior derecho (coordenada 23,31; aunque esta coordenada no existe en Basic, ya que corresponde a las dos líneas inferiores que, como se sabe, son usadas por el canal «K»).

La finalidad de todo esto es ser capaces de averiguar la dirección del atributo de un determinado carácter, conociendo sus coordenadas. En este caso, el problema tiene fácil solución: bastará con multiplicar el número de línea por 32, sumarle el número de columna y, al resultado, sumarle 22528 que es la dirección base de este archivo. En realidad, no difiere mucho de una tabla indexada con dos subíndices o, si lo prefiere, una matriz de dos dimensiones.

Para hacer esto, en código máquina, tendremos en cuenta que multiplicar un número por 32 equivale a desplazarlo 5 lugares a la izquierda. También podemos usar la rutina de multiplico publicada en este mismo curso, aunque es más lenta. Si, después de todo, decide hacerlo por rotación, se le escapará por el lado izquierdo del registro. Afortunadamente, existe una rutina que lo hace de forma sencilla. Hemos denominado «ATR» a la rutina, ya que sirve para buscar un atributo determinado. En ella se entra con «DE» conteniendo las coordenadas del carácter cu-



Por último, hemos utilizado un nuevo pseudo-nemónico: «DEFM» que significa: «DEFINE Message» es decir, definir un mensaje. Lo que hace el ensamblador cuando se encuentra con este pseudo-nemónico es colocar en los bytes siguientes los códigos de los caracteres que componen el mensaje encerrado entre comillas.

Ahora ya, vamos a ver cómo funciona la rutina. En la línea 1010 cargamos en «HL» la dirección de inicio del mensaje. Luego, cargamos en «B» la longitud del mismo (en este caso, 20 bytes). A continuación, entramos en un bucle donde iremos cargando, uno a uno los bytes que componen el mensaje, en el registro «A», preservando «HL» y «BC» en la pila, llamando a la rutina «IMP-A», recuperando los registros, incrementando el puntero y cerrando el bucle hasta que se hayan impreso los 20 caracteres que componen el mensaje.

Para utilizarlo desde Basic, podemos hacer:

```
PRINT AT 11,CON: RANDOMIZE USR 60500
```

Donde «li» y «co» son las coordenadas del punto a partir del cual deseamos imprimir el mensaje. Esta rutina se puede emplear para imprimir cualquier mensaje siempre que su longitud no exceda de 255 caracteres.

También es importante señalar que la rutina «IMP-A» lee los caracteres del font direccionado por la variable del Sistema «CHARS» de forma que, si hemos cargado otro juego de caracteres y lo tenemos direccionado mediante

60104 700	LD	D, A
60105 710	LD	(DF_CC), DE
60106 720	POP	DE
60107 730	LD	HL, #1021
60110 740	SBC	HL, DE
60113 750	LD	(S_P03N), HL
60116 770	RET	
23606 780	CHARS	23606
23607 790	DF_CC	23607
23608 800	BAND	23608
23609 810	S_P03N	23609
3502 820	SCROLL	00DPE
990 1		
60500 1000	TEST	00500
60501 1010	LD	HL, MENS
60502 1020	LD	B, 20
60503 1030	LD	BC, A, (HL)
60504 1040	PUSH	BC
60507 1050	PUSH	HL
60508 1060	CALL	IMP_A
60511 1070	POP	HL
60512 1080	INC	HL
60513 1090	POP	BC
60514 1100	DJNZ	LOOP
60516 1110	RET	
60517 1120	MENS	DEFM "curso C/M "
60527 1130	DEFM	"MICROHBBV"

Pass 2 errors: 00

Table used: 192 from 246

Fig. 9-18. Listado completo de «IMP-A» y «TEST».

«CHARS», la rutina «IMP-A» usará, precisamente, esos caracteres. En la Figura 9-18 tenemos el listado completo de las rutinas vistas hasta ahora.

Para indicarle a la rutina si ha de imprimir en normal, cursiva, negrita o ambas, lo haremos «POKEando» en la dirección 23681. Si escribimos un «0», la rutina imprimirá normalmente; con un «1» imprimirá letra cursiva; con un «2» la letra será negrita y, finalmente, con un «3» será cursiva y negrita a la vez.

Ya hemos llegado al punto donde toca ensamblar el programa. Esta vez, y para variar, lo ensamblaremos en hexadecimal en lugar de hacerlo en decimal como hasta ahora.

## Ejemplos:

Para imprimir un determinado carácter en la pantalla, partiremos de su código, es decir, entraremos en la rutina con el acumulador (registro «A») conteniendo el código del carácter. Pero el hecho de imprimir un determinado carácter, implica activar una serie de pixels que, todos juntos, formarán la letra o signo en cuestión. Cada carácter se compone de 64 pixels, es decir, de 8 octetos; para imprimirlo, tenemos que transferir esos 8 octetos desde una tabla donde estén definidos todos los caracteres, hasta la pantalla. Las tablas que contienen la definición de caracteres, se suelen denominar «Fonts». La ROM del Spectrum incluye un «Font» que contiene la definición de los 96 caracteres con códigos comprendidos entre 32 y 127 (ambos inclusive). Nosotros podríamos diseñar una tabla que contuviera nuestros propios caracteres y con el diseño que más nos gustara; pero, de momento, vamos a utilizar el Font de la ROM para no complicarnos demasiado la vida.

100	IMP_A	LD	DE, (CHARS)
110		LD	H, 0
120		LD	L, A
130		ADD	HL, HL
140		ADD	HL, HL
150		ADD	HL, HL
160		ADD	HL, DE
170		EX	DE, HL

La dirección base del Font está en la variable del Sistema «CHARS» y desde ahí la leemos en la línea 100. En 110 y 120, cargamos en «HL» el código del carácter. En 130, 140 y 150 lo multiplicamos por 8. En 160 lo sumamos a la dirección base y en 170 pasamos la dirección del carácter en el Font al registro «DE».

A la salida de esta rutina, ya tenemos el registro «DE» apuntando al primero de los ocho octetos que definen el carácter.

A continuación, necesitamos hallar la dirección de pantalla correspondiente al primer octeto del lugar donde vayamos a colocar el carácter según indiquen las coordenadas. Pretendemos que esta rutina sea compatible con el Sistema operativo del Spectrum, es decir, que podamos utilizarla alternándola con sentencias «PRINT». En este caso, lo mejor es que utilizemos las coordenadas en curso del Basic.

Cada carácter viene definido por 8 bytes consecutivos, por lo que el Font es, en realidad, una tabla de 96 elementos donde cada elemento tiene 8 bytes de longitud. El primer carácter será el de código 32 (espacio) que tiene los ocho octetos a «0». Para movernos por la tabla y apuntar al primer octeto de un determinado carácter, deberemos hacer lo siguiente: primero, multiplicamos por 8 el código del carácter; a continuación, sumamos al resultado un número

que nos permita imprimir un carácter en una determinada posición de la pantalla.

Mediante los ejemplos de este capítulo, iremos desarrollando esa rutina y, al final, podremos incorporar a nuestro «procesador de pantallas».



Ya tenemos en «DE» la dirección de nuestro carácter en el Font, y en «HL» la dirección del archivo de pantalla a partir de donde habrá que imprimirlo. Lo normal ahora, sería entrar en un bucle que fuera leyendo cada octeto apuntado por «DE» y almacenándolo donde apunta «HL» incrementando «DE» y «H» en cada pasada (recuerde que incrementamos el octeto alto de la dirección de pantalla para pasar al siguiente scan dentro del mismo carácter, por eso incrementamos «H» y no «HL»).

El bucle podría ser algo así como:

```
190 IMPR LD B,8
200 BUCLE LD A,(DE)
210 LD (HL),A
220 INC DE
230 INC H
240 DJNZ BUCLE
```

A la salida del bucle, tendríamos en pantalla los ocho bytes que definen el carácter y sólo nos faltaría actualizar las coordenadas antes de retornar. Hemos llamado «IMPR» a la rutina porque sirve para imprimir; vamos a ver detenidamente cómo funciona.

En la línea 190 cargamos un «8» en «B» porque será el número de iteraciones de nuestro bucle. En 200 cargamos en «A» el primer byte del carácter y los transferimos a la pantalla en 210. En 220 y 230 incrementamos los punteros y en 240 cerramos el bucle.

Esta rutina hace lo mismo que «PRINT» en el Basic, sal-

vo que no maneja colores ni códigos de control; en compensación, es considerablemente más rápida.

Podríamos hacer una rutina de impresión que maneja los colores (archivo de atributos) pero no tendría demasiado sentido hacer algo que ya tenemos en la ROM, así que vamos a intentar algo más original. Podemos modificar esta rutina de impresión para que nos imprima letras cursivas o negritas. La letra cursiva la obtenemos desplazando los tres primeros octetos del carácter a la derecha y los tres últimos a la izquierda, dejando los dos centrales inalterados; este tipo de letra también se denomina «italica». La negrita podemos obtenerla haciendo un «OR» de cada byte con el mismo desplazado a la derecha; este tipo de letra también se denomina «bold». Estas dos posibilidades no son excluyentes, no hay ningún problema en que la letra sea cursiva y bold al mismo tiempo, de hecho, es la letra más bonita.

De alguna forma tiene que saber la rutina qué tipo de letra queremos; para ello utilizaremos dos «flags» que señalarán dos bits de la dirección de memoria 23681 (5C81h) que etiquetamos como «BAND» (por «banderas»). El primer bit por la derecha (el de menos peso) indicará letra cursiva cuando esté a «1». El segundo, indicará letra bold (también cuando esté a «1»). En el caso de que ambos estén a «0» se imprimirá letra normal y si ambos están a «1» la letra será cursiva y bold.

Los dos bits serán comprobados en cada pasada del bucle para actuar en consecuencia. También será necesario

listado la pseudo-instrucción «ORG 60000».

Ahora, vamos a hacer una pequeña rutina que ensamblaremos a partir de 60500:

```
1000 TEST ORG 60500
1010 LD HL,MENS
1020 LD B,20
1030 LOOP LD A,(HL)
1040 PUSH BC
1050 PUSH HL
1060 CALL IMP_A
1070 POP HL
1080 INC HL
1090 POP BC
1100 DJNZ LOOP
1110 RET
1120 MENS DEFN "curso C/M"
1130 DEFN "MICROHOBBY"
```

Hay muchas cosas nuevas en esta rutina así que vamos a verla con detenimiento. En primer lugar, hemos utilizado por segunda vez el pseudo-número «ORG». Aunque ensambláramos las dos rutinas juntas, no hay problema por saber la rutina qué tipo de letra queremos; para ello utilizaremos dos «flags» que señalarán dos bits de la dirección de memoria 23681 (5C81h) que etiquetamos como «BAND» (por «banderas»). El primer bit por la derecha (el de menos peso) indicará letra cursiva cuando esté a «1». El segundo, indicará letra bold (también cuando esté a «1»). En el caso de que ambos estén a «0» se imprimirá letra normal y si ambos están a «1» la letra será cursiva y bold.

En segundo lugar, hemos vuelto a utilizar la instrucción «CALL» así que no nos quedará más remedio que explicar cómo se ensambla.

```
10 2C-
20 2D+
30 IMPRIME_CODIGO_EN_A"
40
60000 90 ORG 60000
60000 100 IMP_A LD DE,(CHARS)
60004 110 LD H,0
60006 120 LD L,A
60007 130 ADD HL,HL
60008 140 ADD HL,HL
60009 150 ADD HL,HL
60010 160 ADD HL,DE
60011 170 EX DE,HL
60012 180 LD HL,(DF_CC)
60015 190 IMPR_1 LD B,8
60017 200 BUC_1 LD A,(DE)
60018 210 LD (HL),A
60019 220 LD A,(BAND)
60022 230 AND 1
60024 240 JR Z,NOCURS
60026 250 SRL (HL)
60028 260 LD A,B
60029 270 CP 5
60031 280 JR NC,NOCURS
60033 290 SLA (HL)
60035 300 CP 3
60037 310 JR NC,NOCURS
60039 320 SLA (HL)
60041 330 NOCURS LD A,(BAND)
60044 340 AND 2
60046 350 JR Z,NOBOLD
60048 360 LD A,(HL)
60049 370 SRL A
60051 380 OR (HL)
60052 390 LD DE,(S_POSN)
60053 400 NOBOLD INC HL,01821
60054 410 INC DE,HL
60055 420 DJNZ BUC_1 DE,HL
60057 440 LD DE,(S_POSN)
60061 450 LD HL,01821
60064 460 BBC DE,HL
60066 470 EX DE,HL
60067 480 INC E
60068 490 LD A,E
60069 500 CP 32
60071 510 JR C,BIGUE
60073 520 LD DE,0
60075 530 INC D
60076 540 LD A,D
60077 550 CP 21
60079 560 JR C,BIGUE
60081 570 CALL SCROLL
60084 580 LD DE,01400
60087 590 BIGUE PUSH A,D
60088 600 DIR LD A,D
60089 610 AND 007
60091 620 RRC A
60093 630 RRC A
60095 640 RRC A
60097 650 OR E
60098 660 LD E,A
60099 670 LD A,D
60100 680 AND 018
60102 690 OR 040
```

```
180 LD HL,(DF_CC)
```



deberá encargarse de que nunca se vuelquen más de 22 líneas de una sola vez.

En realidad, la pantalla tiene 24 líneas y podríamos haberlas utilizado todas, pero hemos querido que la rutina sea compatible con el Sistema Operativo Basic, así que hemos respetado las dos líneas inferiores para que las pueda seguir usando el canal «K». No obstante, quien desee puede modificar la rutina para que trabaje sobre toda la pantalla. Para ello, lo único que hay que hacer es poner «23» en lugar de «21» en la línea 550 y « 1600» en lugar de « 1400» en la línea 580.

Nos hemos quedado «colgados» con unos puntos suspensivos después de la etiqueta «SIGUE» donde teníamos en «DE» las nuevas coordenadas. Ahora tenemos que calcular la nueva dirección del archivo de pantalla correspondiente a estas coordenadas. Vamos a ver cómo seguiría la rutina:

```
590 SIGUE PUSH DE
600 DIR LD A,D
610 AND #07
620 RRC A
630 RRC A
640 RRC A
650 OR E
660 LD E,A
670 LD A,D
680 AND #18
690 OR #40
700 LD D,A
```

Primero, salvamos «DE» en la pila ya que luego lo necesitaremos. Después, entramos en la rutina «DIR» que se

explicó anteriormente. En esta rutina, lo que hacemos es calcular la dirección de pantalla que corresponde a las nuevas coordenadas. A la salida, tendremos esta dirección en «DE». Vamos a ver qué hacemos con ella:

```
720 LD (DF_CC),DE
730 POP DE
740 LD HL,#1821
750 SBC HL,DE
760 LD (S_POSN),HL
770 RET
```

Primero guardamos en «DF-CC» la nueva dirección de pantalla, luego recuperamos las coordenadas de la pila, las invertimos restándolas de 1821h y almacenamos el resultado en «S-POSN»; finalmente, retornamos en la línea 770.

Para que la rutina «IMP-A» esté completa, sólo nos falta definir el valor de algunas etiquetas, así que vamos a ello:

```
780 CHARS EQU 2360h
790 DF_CC EQU 2368h
800 BAND EQU 2368h
810 S_POSN EQU 2368h
820 SROLL EQU #0FE
```

Con esto termina la rutina «IMP-A». Cada vez que la llamemos con el código de un carácter en «A», nos imprimirá ese carácter y actualizará las variables del sistema necesarias para que el Basic siga funcionando.

Hasta ahora, todas las rutinas que hemos hecho po-

drían ser llamadas directamente desde el Basic. Sin embargo, no es este el caso de «IMP-A» ya que, si la llamamos con **USR**, no sabremos el contenido de «A» y nos imprimirá un carácter aleatorio. En realidad, esta rutina no ha sido concebida para llamarla con **USR**, aunque luego veremos una forma de usarla directamente desde Basic.

Por si algún lector aún no se había dado cuenta, lo que estamos intentando es construir todo un programa para gestionar la pantalla. Hemos hecho una serie de rutinas que trabajaban de forma distinta según el valor que contuviera el registro «A» (Borrado por trozos, intercambio de bloques, etc.). Para todas ellas, el valor de «A» tenía que ser menor de «32». Pues bien, esta rutina será, en su día, una subrutina del programa para gestionar la pantalla, concretamente, se encargará de imprimir un carácter cuando el contenido de «A» sea mayor de 32.

De momento, no nos parece bien hacer esperar al lector, así que vamos a ver de qué forma podemos utilizar esta rutina. En principio, lo más sencillo es coger una zona de memoria y almacenar en ella los códigos de los caracteres que componen un determinado mensaje para, luego, hacer un bucle que fuera cargando los códigos en «A» uno por uno y llamando a esta rutina para que los imprimiera.

Vamos a imprimir el mensaje: «curso C/M MICROHOBAY», utilizando esta rutina. «IMP-A» es reubicable, pero vamos a ensamblarla a partir de la dirección **60000**. Para ello, añadimos al principio del

```
190 IMPR_1 LD B,B
200 BUC_1 LD A,(DE)
210 LD (HL),A
220 LD A,(BAND)
230 AND 1
240 JR Z,NOCURS
250 SRL (HL)
260 LD A,B
270 CP 5
280 JR NC,NOCURS
```

el contenido de «B», así que lo comparamos con «3» para ver si estamos en los dos scans de la mitad del carácter. Si es así, saltamos a «NOCURS». Si no, ejecutamos un desplazamiento más a la izquierda en la línea 320.

De esta forma, los tres primeros scans quedan desplazados a la derecha, los dos de en medio quedan tal como están y los tres últimos quedan desplazados a la izquierda lográndose el efecto de letra cursiva. Ver Figura 9-17.

En las líneas 330, 340 y 350 comprobamos el flag de «bold» (letra negrita). Supongamos que está a «1» en cuyo caso no se produce el salto a «NOBOLD» y se continúa en la línea 360. En esta línea, cargamos en «A» el octeto que acabamos de transferir. En 370 lo desplazamos a la derecha. En 380 le hacemos un «OR» con el mismo y, finalmente (en la línea 390), colocamos el resultado en el lugar correspondiente del archivo de pantalla.

A partir de «NOBOLD», la rutina continúa de forma normal: se incrementan los punteros «DE» y «H» y se cierra el bucle para el siguiente scan o se sale del mismo si ya se han completado los ocho scans correspondientes a un carácter.

Ya tenemos el carácter impreso en la pantalla, ahora nos queda actualizar las variables del Sistema que contienen las coordenadas y la dirección en el archivo de pre-sentación visual.

El procedimiento a seguir es el siguiente:

1. Se leen las coordenadas antiguas.
2. Se incrementa el número de columna.

```
290 SLA (HL)
300 CP 3
310 JR NC,NOCURS
320 SLA (HL)
330 NOCURS LD A,(BAND)
340 AND 2
350 JR Z,NOBOLD
360 LD A,(HL)
370 SRL A
380 OR (HL)
390 LD (HL),A
400 NOBOLD INC DE
410 INC H
420 DJNZ BUC_1
```

Si ambos flags están a «0» saltaremos primero a «NOCURS» y luego a «NOBOLD» con lo que la rutina será igual que «IMPR».

Supongamos que el primer bit de «BAND», es decir, el flag de cursiva, está a «1». En ese caso, saldremos de la línea 230 con el indicador de «cero» a «0» y no se producirá el salto a «NOCURS» sino que el programa seguirá por la línea 250.

En esta línea, empezamos por desplazar a la derecha el octeto que acabábamos de transferir al archivo de pantalla. A continuación, en las líneas 260, 270 y 280, comprobamos si nos hallamos en uno de los primeros tres scans del carácter, en cuyo caso, saltamos a «NOCURS». Esta comprobación se lleva a cabo mirando si el contenido de «B» (contador del bucle) es mayor de «5», en cuyo caso, al comparar con «5» no se producirá acarreo.

Si no es así, continuamos en la línea 290 donde desplazamos el octeto a la izquierda. Seguimos teniendo en «A»



3. Si es mayor de 31, se pone a cero y se incrementa el número de línea.
4. Si ésta es mayor de 21, se hace «scroll» hacia arriba de una línea y se pone 21 como número de línea.
5. Se calcula la nueva dirección de pantalla.
6. Se almacena la nueva dirección de pantalla.
7. Se almacenan las nuevas coordenadas.

Como indicamos antes, las coordenadas se encuentran en una variable del Sistema denominada «S-POSN» cuya dirección es 23688 y están almacenadas de forma invertida, es decir, no tenemos el número de líneas o columnas que van escritas, sino las que nos faltan para llegar al final más dos. Por tanto, para recuperar las coordenadas correc-

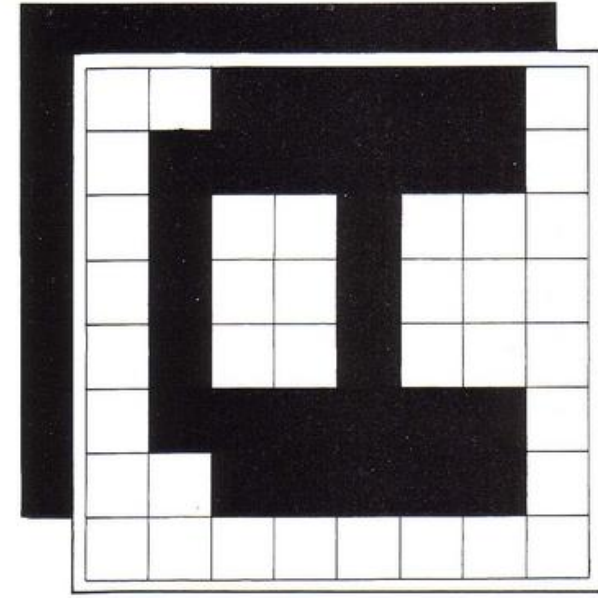


Fig. 9-17a. Formato de la letra «A».

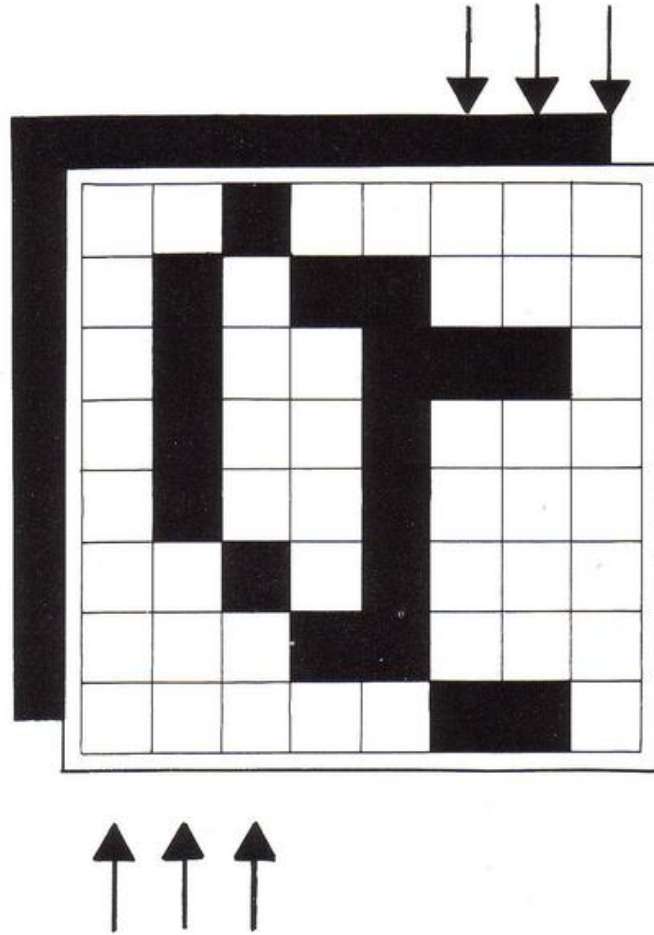


Fig. 9-17b. Formato de la «A» en cursiva.

tas tal y como nos interesan deberemos restar de 1821h el contenido de la variable «S-POSN». Vayamos viendo el listado:

```
440 LD DE, (S_POSN)
450 LD HL, #1821
460 SEC HL, DE
470 EX DE, HL
```

Empezamos por leer en «DE» el contenido de «S-POSN», luego lo restamos de 1821h y transferimos el resultado, de nuevo, a «DE». Por tanto, a la salida de esta rutina tenemos en «D» la línea donde hemos impreso nuestro carácter y en «E» la columna. Ahora, empezaremos por incrementar el número de columna. Sigamos con el listado:

```
480 INC E
490 LD A, E
500 CP 32
510 JR C, SIGUE
520 LD E, 0
530 INC D
540 LD A, D
550 CP 21
560 JR C, SIGUE
570 CALL SCROLL
580 LD DE, #1400
590 SIGUE .....
```

De momento, ignore la línea 570. Luego veremos para qué vale. Empezamos por incrementar «E». En las líneas 490, 500 y 510 comprobamos si es menor de 32 en cuyo caso, ya estarían actualizadas las coordenadas y saltaríamos a la etiqueta «SIGUE». Si el valor de «E» después de incrementar es 32 o mayor (nunca puede ser mayor de

ción «CALL» de la línea 570 es una llamada a subrutina. Aún no hemos estudiado estas instrucciones por lo que, de momento, nos conformaremos con saber que «CALL SCROLL» nos sirve para subir una línea toda la pantalla. Por supuesto, en capítulos posteriores estudiaremos no sólo las instrucciones de llamada a subrutinas, sino también algunas subrutinas de la ROM que, como «SCROLL», pueden resultarnos muy útiles al programar.

Como es evidente, cuando utilicemos esta rutina para imprimir, nunca nos saldrá el famoso mensaje «Scroll?» («Sigo?» en los españoles). De hecho, el Spectrum es el único ordenador donde ocurren estas cosas; lo normal es que la pantalla suba sin esperar a que lo digamos. El programa que lo utilice esta rutina para el volcado de datos por pantalla,

Fig. 9-17c. Formato de la «A» en bold o negrita.

32, pero el mismo trabajo nos cuesta comprobar si es igual o mayor), continuaríamos en la línea 520 donde cargamos un «0» en «E» e incrementamos «D» para colocarnos al principio de la siguiente línea.

Ahora debemos comprobar si hemos alcanzado la línea 21. Si no es así, saltamos a «SIGUE». Si, por el contrario, hemos llegado al final de la pantalla, será necesario realizar un «scroll» hacia arriba, es decir, subir toda la pantalla una línea y colocar la nueva posición de impresión al principio de la línea 20, lo que conseguimos cargando 1400h en «DE».

Para realizar el «scroll» hacia arriba, podíamos haber escrito una rutina en C/M que lo hiciera, pero como no nos gusta trabajar de balde y ya tenemos esa rutina en la ROM, hemos decidido que lo mejor es utilizarla. La instruc-



	INC	HL	Pass 2 errors:	00
601781190	LD	(HL),D	Table used:	251 from 277
601791200	RET			
601801210				

Fig. 9-20. Listado completo de la rutina de salida «Canal».

## EJERCICIOS

- 1.- Escriba una rutina que sustituya a "IMPR\_1" (puede llamarla "IMPR\_2") y que imprima los caracteres en "imagen de espejo".
- 2.- Modifique la rutina "ACT" para que "IMP\_A" trabaje por el canal "S".
- 3.- ¿Como haría para posicionar la impresión en un lugar determinado de la pantalla ayudándose de la rutina "IMP\_A"?.
- 4.- Si ejecutamos el comando OPEN #4,"S" ¿que modificación se produciría en la tabla de corrientes?, ¿y en la de canales?.
- 5.- Escriba una rutina que multiplique por 32 el dato contenido en el acumulador, devolviendo el resultado en el registro "HL". Efectue la multiplicación mediante desplazamientos.

daremos una pista: «CALL SCROLL» se ensambla como «CD,FE,0D» y «CALL IMP\_A» como «CD,60,EA». Para las direcciones se puede usar la siguiente tabla de equivalencia:

23606 = 5C36h
23604 = 5C84h
23601 = 5C81h
23600 = 5C80h

Vamos a ir viendo el ensamblado por trozos tal y como vimos las rutinas:

100 60000	ED,5B,36,5C
110 60004	26,00
120 60006	6F
130 60007	29
140 60008	29
150 60009	29
160 60010	19
170 60011	EB
180 60012	2A,84,5C

Esta es la parte donde calculábamos las direcciones en el font y en la pantalla. A continuación viene la rutina «IMPR-1»:

190 60015	06,08
200 60017	1A
210 60018	77
220 60019	3A,81,5C
230 60022	E6,01
240 60024	28,0F
250 60026	CB,3E
260 60028	78
270 60029	FE,05
280 60031	30,08
290 60033	CB,26

610 60089	E6,07
620 60091	CB,0F
630 60093	CB,0F
640 60095	CB,0F
650 60097	B3
660 60098	5F
670 60099	7A
680 60100	E6,18
690 60102	F6,40
700 60104	57
720 60105	ED,53,84,5C
730 60109	D1
740 60110	21,21,18
750 60113	ED,52
760 60115	22,88,5C
770 60118	C9

Con esto, queda completa la rutina «IMP\_A». Ahora vamos a ensamblar la rutina «TEST»:

1010 60500	21,65,EC
1020 60503	06,14
1030 60505	7E
1040 60506	C5
1050 60507	E5
1060 60508	CD,60,EA
1070 60511	E1
1080 60512	23
1090 60513	C1
1100 60514	10,F5
1110 60516	C9
1120 60517	63,75,72,73
	6F,20,43,2F
	40,20
1130 60527	4D,49,43,52
	4F,48,4F,42
	42,59

Finalmente, la parte encargada de calcular la nueva dirección de pantalla y almacenar ésta y las coordenadas en las variables del Sistema correspondiente:

590 60087	D5
600 60088	7A

A continuación viene la parte encargada de actualizar las coordenadas:

440 60057	ED,5B,88,5C
450 60061	21,21,18
460 60064	ED,52
470 60065	EB
480 60067	1C
490 60068	7B
500 60069	FE,20
510 60071	38,0E
520 60073	1E,00
530 60075	14
540 60076	7A
550 60077	FE,15
560 60079	38,06
570 60081	CD,FE,0D
580 60084	11,00,14



```

10 REM PROGRAMA 9-1
15 REM CARGA CODIGO MAQUINA
20 DEF FN a(a$) = 16*(CODE a$)
30 DEF FN b(b$) = 32*(CODE b$)
40 DEF FN c(c$) = 64*(CODE c$)
50 DEF FN d(d$) = 128*(CODE d$)
60 DEF FN e(e$) = 256*(CODE e$)
70 DEF FN f(f$) = 512*(CODE f$)
80 DEF FN g(g$) = 1024*(CODE g$)
90 DEF FN h(h$) = 2048*(CODE h$)
100 DEF FN i(i$) = 4096*(CODE i$)
110 DEF FN j(j$) = 8192*(CODE j$)
120 DEF FN k(k$) = 16384*(CODE k$)
130 DEF FN l(l$) = 32768*(CODE l$)
140 DEF FN m(m$) = 65536*(CODE m$)
150 DEF FN n(n$) = 131072*(CODE n$)
160 DEF FN o(o$) = 262144*(CODE o$)
170 DEF FN p(p$) = 524288*(CODE p$)
180 DEF FN q(q$) = 1048576*(CODE q$)
190 DEF FN r(r$) = 2097152*(CODE r$)
200 DEF FN s(s$) = 4194304*(CODE s$)
210 DEF FN t(t$) = 8388608*(CODE t$)
220 DEF FN u(u$) = 16777216*(CODE u$)
230 DEF FN v(v$) = 33554432*(CODE v$)
240 DEF FN w(w$) = 67108864*(CODE w$)
250 DEF FN x(x$) = 134217728*(CODE x$)
260 DEF FN y(y$) = 268435456*(CODE y$)
270 DEF FN z(z$) = 536870912*(CODE z$)
280 DEF FN aa(aa$) = 1073741824*(CODE aa$)
290 DEF FN ab(ab$) = 2147483648*(CODE ab$)
300 DEF FN ac(ac$) = 4294967296*(CODE ac$)
310 DEF FN ad(ad$) = 8589934592*(CODE ad$)
320 DEF FN ae(ae$) = 17179869184*(CODE ae$)
330 DEF FN af(af$) = 34359738368*(CODE af$)
340 DEF FN ag(ag$) = 68719476736*(CODE ag$)
350 DEF FN ah(ah$) = 137438953472*(CODE ah$)
360 DEF FN ai(ai$) = 274877906944*(CODE ai$)
370 DEF FN aj(aj$) = 549755813888*(CODE aj$)
380 DEF FN ak(ak$) = 1099511627776*(CODE ak$)
390 DEF FN al(al$) = 2199023255552*(CODE al$)
400 DEF FN am(am$) = 4398046511104*(CODE am$)
410 DEF FN an(an$) = 8796093022208*(CODE an$)
420 DEF FN ao(ao$) = 17592186044416*(CODE ao$)
430 DEF FN ap(ap$) = 35184372088832*(CODE ap$)
440 DEF FN aq(aq$) = 70368744177664*(CODE aq$)
450 DEF FN ar(ar$) = 140737488355328*(CODE ar$)
460 DEF FN as(as$) = 281474976710656*(CODE as$)
470 DEF FN at(at$) = 562949953421312*(CODE at$)
480 DEF FN au(au$) = 1125899906842624*(CODE au$)
490 DEF FN av(av$) = 2251799813685248*(CODE av$)
500 DEF FN aw(aw$) = 4503599627370496*(CODE aw$)
510 DEF FN ax(ax$) = 9007199254740992*(CODE ax$)
520 DEF FN ay(ay$) = 18014398509481984*(CODE ay$)
530 DEF FN az(az$) = 36028797018963968*(CODE az$)
540 DEF FN ba(ba$) = 72057594037927936*(CODE ba$)
550 DEF FN bb(bb$) = 144115188075855872*(CODE bb$)
560 DEF FN bc(bc$) = 288230376151711744*(CODE bc$)
570 DEF FN bd(bd$) = 576460752303423488*(CODE bd$)
580 DEF FN be(be$) = 1152921504606846976*(CODE be$)
590 DEF FN bf(bf$) = 2305843009213693952*(CODE bf$)
600 DEF FN bg(bg$) = 4611686018427387904*(CODE bg$)
610 DEF FN bh(bh$) = 9223372036854775808*(CODE bh$)
620 DEF FN bi(bi$) = 18446744073709551616*(CODE bi$)
630 DEF FN bj(bj$) = 36893488147419103232*(CODE bj$)
640 DEF FN bk(bk$) = 73786976294838206464*(CODE bk$)
650 DEF FN bl(bl$) = 147573952589676412928*(CODE bl$)
660 DEF FN bm(bm$) = 295147905179352825856*(CODE bm$)
670 DEF FN bn(bn$) = 590295810358705651712*(CODE bn$)
680 DEF FN bo(bo$) = 1180591620717411303424*(CODE bo$)
690 DEF FN bp(bp$) = 2361183241434822606848*(CODE bp$)
700 DEF FN bq(bq$) = 4722366482869645213696*(CODE bq$)
710 DEF FN br(br$) = 9444732965739290427392*(CODE br$)
720 DEF FN bs(bs$) = 18889465931478580854784*(CODE bs$)
730 DEF FN bt(bt$) = 37778931862957161709568*(CODE bt$)
740 DEF FN bu(bu$) = 75557863725914323419136*(CODE bu$)
750 DEF FN bv(bv$) = 151115727451828646838272*(CODE bv$)
760 DEF FN bw(bw$) = 302231454903657293676544*(CODE bw$)
770 DEF FN bx(bx$) = 604462909807314587353088*(CODE bx$)
780 DEF FN by(by$) = 1208925819614629174706176*(CODE by$)
790 DEF FN bz(bz$) = 2417851639229258349412352*(CODE bz$)
800 DEF FN ca(ca$) = 4835703278458516698824704*(CODE ca$)
810 DEF FN cb(cb$) = 9671406556917033397649408*(CODE cb$)
820 DEF FN cc(cc$) = 19342813113834066795298816*(CODE cc$)
830 DEF FN cd(cd$) = 38685626227668133598597632*(CODE cd$)
840 DEF FN ce(ce$) = 77371252455336267197195264*(CODE ce$)
850 DEF FN cf(cf$) = 154742504910672534394390528*(CODE cf$)
860 DEF FN cg(cg$) = 309485009821345068788781056*(CODE cg$)
870 DEF FN ch(ch$) = 618970019642690137577562112*(CODE ch$)
880 DEF FN ci(ci$) = 1237940039285380275155124224*(CODE ci$)
890 DEF FN cj(cj$) = 2475880078570760550310248448*(CODE cj$)
900 DEF FN ck(ck$) = 4951760157141521101020496896*(CODE ck$)
910 DEF FN cl(cl$) = 9903520314283042242040993792*(CODE cl$)
920 DEF FN cm(cm$) = 19807040628566084484081987584*(CODE cm$)
930 DEF FN cn(cn$) = 39614081257132168968163975168*(CODE cn$)
940 DEF FN co(co$) = 79228162514264337936327950336*(CODE co$)
950 DEF FN cp(cp$) = 158456325028528675872655900672*(CODE cp$)
960 DEF FN cq(cq$) = 316912650057057351745311801344*(CODE cq$)
970 DEF FN cr(cr$) = 633825300114114703490623682688*(CODE cr$)
980 DEF FN cs(cs$) = 1267650600228229406981247365376*(CODE cs$)
990 DEF FN ct(ct$) = 2535301200456458813962494730752*(CODE ct$)
1000 DEF FN cu(cu$) = 5070602400912917627924989461504*(CODE cu$)
1010 DEF FN cv(cv$) = 10141204801825835255849978923008*(CODE cv$)
1020 DEF FN cw(cw$) = 20282409603651670511699957846016*(CODE cw$)
1030 DEF FN cx(cx$) = 40564819207303341023399915692032*(CODE cx$)
1040 DEF FN cy(cy$) = 81129638414606682046799831384064*(CODE cy$)
1050 DEF FN cz(cz$) = 162259276829213364183599662768128*(CODE cz$)
1060 DEF FN da(da$) = 324518553658426728367199325536256*(CODE da$)
1070 DEF FN db(db$) = 649037107316853456734398651070512*(CODE db$)
1080 DEF FN dc(dc$) = 1298074214633716914688797302141024*(CODE dc$)
1090 DEF FN dd(dd$) = 259614842926743382937759460424448*(CODE dd$)
1100 DEF FN de(de$) = 519229685853486765875518920848896*(CODE de$)
1110 DEF FN df(df$) = 1038459371707533531731837841677792*(CODE df$)
1120 DEF FN dg(dg$) = 2076918743415067063463675683355584*(CODE dg$)
1130 DEF FN dh(dh$) = 4153837486830134126927351366711168*(CODE dh$)
1140 DEF FN di(di$) = 8307674973660268253854702732222336*(CODE di$)
1150 DEF FN dj(dj$) = 166153499473205365077094544444448*(CODE dj$)
1160 DEF FN dk(dk$) = 332306998946410730154189088888896*(CODE dk$)
1170 DEF FN dl(dl$) = 664613997892821460308378177777792*(CODE dl$)
1180 DEF FN dm(dm$) = 1329227995785642920616763555555584*(CODE dm$)
1190 DEF FN dn(dn$) = 2658455991571285841233527111111168*(CODE dn$)
1200 DEF FN do(do$) = 5316911983142571672467054222222336*(CODE do$)
1210 DEF FN dp(dp$) = 10633823966285143444934108544444704*(CODE dp$)
1220 DEF FN dq(dq$) = 212676479325702868888890888888896*(CODE dq$)
1230 DEF FN dr(dr$) = 425352958651405737777781777777792*(CODE dr$)
1240 DEF FN ds(ds$) = 850705917302811475555563555555584*(CODE ds$)
1250 DEF FN dt(dt$) = 1701411834605622951111127111111168*(CODE dt$)
1260 DEF FN du(du$) = 3402823669211245242222242222222336*(CODE du$)
1270 DEF FN dv(dv$) = 6805647338422490484444484444444704*(CODE dv$)
1280 DEF FN dw(dw$) = 1361129467684498096888888888888896*(CODE dw$)
1290 DEF FN dx(dx$) = 2722258935368996193777777777777792*(CODE dx$)
1300 DEF FN dy(dy$) = 5444517870737992387555555555555584*(CODE dy$)
1310 DEF FN dz(dz$) = 10889035741475984775111111111111168*(CODE dz$)
1320 DEF FN ea(ea$) = 2177807148315196954222222222222336*(CODE ea$)
1330 DEF FN eb(eb$) = 4355614296630393908444444444444704*(CODE eb$)
1340 DEF FN ec(ec$) = 871122859326078781688888888888896*(CODE ec$)
1350 DEF FN ed(ed$) = 1742245718552157563377777777777792*(CODE ed$)
1360 DEF FN ee(ee$) = 3484491437110315126755555555555584*(CODE ee$)
1370 DEF FN ef(ef$) = 6968982874220620253511111111111168*(CODE ef$)
1380 DEF FN eg(eg$) = 1393796574844124050702222222222336*(CODE eg$)
1390 DEF FN eh(eh$) = 2787593149688248101404444444444704*(CODE eh$)
1400 DEF FN ei(ei$) = 557518629937649620280888888888896*(CODE ei$)
1410 DEF FN ej(ej$) = 1115037259875299249561777777777792*(CODE ej$)
1420 DEF FN ek(ek$) = 2230074519750598499135555555555584*(CODE ek$)
1430 DEF FN el(el$) = 4460149039501196992711111111111168*(CODE el$)
1440 DEF FN em(em$) = 892029807900239398542222222222336*(CODE em$)
1450 DEF FN en(en$) = 1784059615800478797084444444444704*(CODE en$)
1460 DEF FN eo(eo$) = 356811923160095759416888888888896*(CODE eo$)
1470 DEF FN ep(ep$) = 713623846320191518833777777777792*(CODE ep$)
1480 DEF FN eq(eq$) = 1427247692640383037667555555555584*(CODE eq$)
1490 DEF FN er(er$) = 2854495385280766075335111111111168*(CODE er$)
1500 DEF FN es(es$) = 5708990770561532150670222222222336*(CODE es$)
1510 DEF FN et(et$) = 1141798154312266421133404444444704*(CODE et$)
1520 DEF FN eu(eu$) = 228359630862453284266888888888896*(CODE eu$)
1530 DEF FN ev(ev$) = 456719261724906568533777777777792*(CODE ev$)
1540 DEF FN ew(ew$) = 9134385234498131370675555555555584*(CODE ew$)
1550 DEF FN ex(ex$) = 1826877046899626274135111111111168*(CODE ex$)
1560 DEF FN ey(ey$) = 3653754093799252548270222222222336*(CODE ey$)
1570 DEF FN ez(ez$) = 7307508187598505096540444444444704*(CODE ez$)
1580 DEF FN fa(fa$) = 146150163751970101930888888888896*(CODE fa$)
1590 DEF FN fb(fb$) = 292300327503940203861777777777792*(CODE fb$)
1600 DEF FN fc(fc$) = 5846006550078804077235555555555584*(CODE fc$)
1610 DEF FN fd(fd$) = 1169201310015660815447022222222336*(CODE fd$)
1620 DEF FN fe(fe$) = 2338402620031321630884444444444704*(CODE fe$)
1630 DEF FN ff(ff$) = 467680524006264326176888888888896*(CODE ff$)
1640 DEF FN fg(fg$) = 935361048012528652353777777777792*(CODE fg$)
1650 DEF FN fh(fh$) = 1870722096025057304707555555555584*(CODE fh$)
1660 DEF FN fi(fi$) = 3741444192050114609415111111111168*(CODE fi$)
1670 DEF FN fj(fj$) = 748288838410022921883022222222336*(CODE fj$)
1680 DEF FN fk(fk$) = 1496577676820045837766444444444704*(CODE fk$)
1690 DEF FN fl(fl$) = 299315535364009167553288888888896*(CODE fl$)
1700 DEF FN fm(fm$) = 598631070728018335106577777777792*(CODE fm$)
1710 DEF FN fn(fn$) = 1197262141566366670213155555555584*(CODE fn$)
1720 DEF FN fo(fo$) = 2394524283132733340426311111111168*(CODE fo$)
1730 DEF FN fp(fp$) = 478904856626546668085262222222336*(CODE fp$)
1740 DEF FN fq(fq$) = 9578097132530933361705444444444704*(CODE fq$)
1750 DEF FN fr(fr$) = 191561942650618663241108888888896*(CODE fr$)
1760 DEF FN fs(fs$) = 383123885301237327282217777777792*(CODE fs$)
1770 DEF FN ft(ft$) = 7662477706024746545644355555555584*(CODE ft$)
1780 DEF FN fu(fu$) = 153249540120494930912888888888896*(CODE fu$)
1790 DEF FN fv(fv$) = 306499080240989861825777777777792*(CODE fv$)
1800 DEF FN fw(fw$) = 6129981604819797237515555555555584*(CODE fw$)
1810 DEF FN fx(fx$) = 1225996320963959447503111111111168*(CODE fx$)
1820 DEF FN fy(fy$) = 245199264192791889500622222222336*(CODE fy$)
1830 DEF FN fz(fz$) = 4903985283855837790012444444444704*(CODE fz$)
1840 DEF FN ga(ga$) = 980797056771167558002488888888896*(CODE ga$)
1850 DEF FN gb(gb$) = 196159411354235511600497777777792*(CODE gb$)
1860 DEF FN gc(gc$) = 3923188227084710200099555555555584*(CODE gc$)
1870 DEF FN gd(gd$) = 7846376454169420400199111111111168*(CODE gd$)
1880 DEF FN ge(ge$) = 156927529083388408003822222222336*(CODE ge$)
1890 DEF FN gf(gf$) = 3138550581667768160076444444444704*(CODE gf$)
1900 DEF FN gg(gg$) = 627710116333533632015288888888896*(CODE gg$)
1910 DEF FN gh(gh$) = 125542023266706726402557777777792*(CODE gh$)
1920 DEF FN gi(gi$) = 2510840465334134528051155555555584*(CODE gi$)
1930 DEF FN gj(gj$) = 50216809306682690561022222222336*(CODE gj$)
1940 DEF FN gk(gk$) = 1004336186133813811224444444444704*(CODE gk$)
1950 DEF FN gl(gl$) = 200867237226762762244888888888896*(CODE gl$)
1960 DEF FN gm(gm$) = 401734474453525524489777777777792*(CODE gm$)
1970 DEF FN gn(gn$) = 8034689489070510489795555555555584*(CODE gn$)
1980 DEF FN go(go$) = 1606937897814102097951111111111168*(CODE go$)
1990 DEF FN gp(gp$) = 321387579562820419558222222222336*(CODE gp$)
2000 DEF FN gq(gq$) = 6427751591256408391164444444444704*(CODE gq$)
2010 DEF FN gr(gr$) = 128555031851320167823288888888896*(CODE gr$)
2020 DEF FN gs(gs$) = 257110063702640335646577777777792*(CODE gs$)
2030 DEF FN gt(gt$) = 5142201274052806712931555555555584*(CODE gt$)
2040 DEF FN gu(gu$) = 1028440254810561342586311111111168*(CODE gu$)
2050 DEF FN gv(gv$) = 205688050962112268517262222222336*(CODE gv$)
2060 DEF FN gw(gw$) = 4113761019242245370345444444444704*(CODE gw$)
2070 DEF FN gx(gx$) = 822752203848449074069088888888896*(CODE gx$)
2080 DEF FN gy(gy$) = 164550440769689814801377777777792*(CODE gy$)
2090 DEF FN gz(gz$) = 3291008815393796296027555555555584*(CODE gz$)
2100 DEF FN ha(ha$) = 6582017630787592592055111111111168*(CODE ha$)
2110 DEF FN hb(hb$) = 131640352615751851841102222222336*(CODE hb$)
2120 DEF FN hc(hc$) = 2632807052315037036822044444444704*(CODE hc$)
2130 DEF FN hd(hd$) = 526561410463007407364408888888896*(CODE hd$)
2140 DEF FN he(he$) = 105312282092601484728897777777792*(CODE he$)
2150 DEF FN hf(hf$) = 2106245641852029694577955555555584*(CODE hf$)
2160 DEF FN hg(hg$) = 4212491283704059389155911111111168*(CODE hg$)
2170 DEF FN hh(hh$) = 84249825674081187783118222222336*(CODE hh$)
2180 DEF FN hi(hi$) = 1684996513481623755662364444444704*(CODE hi$)
2190 DEF FN hj(hj$) = 336999302696324751132472888888896*(CODE hj$)
2200 DEF FN hk(hk$) = 673998605392649502264945777777792*(CODE hk$)
2210 DEF FN hl(hl$) = 1347997210785299004529891555555584*(CODE hl$)
2220 DEF FN hm(hm$) = 269599442157059800905978222222336*(CODE hm$)
2230 DEF FN hn(hn$) = 5391988843141196018119564444444704*(CODE hn$)
2240 DEF FN ho(ho$) = 1078397768228239203623911111111168*(CODE ho$)
2250 DEF FN hp(hp$) = 21567955364564784072478222222336*(CODE hp$)
2260 DEF FN hq(hq$) = 4313591072912956814495644444444704*(CODE hq$)
2270 DEF FN hr(hr$) = 862718214782591362899128888888896*(CODE hr$)
2280 DEF FN hs(hs$) = 172543642956518273779857777777792*(CODE hs$)
2290 DEF FN ht(ht$) = 3450872859130365475577155555555584*(CODE ht$)
2300 DEF FN hu(hu$) = 6901745718260730951154311111111168*(CODE hu$)
2310 DEF FN hv(hv$) = 13803491436421461902286222222336*(CODE hv$)
2320 DEF FN hw(hw$) = 2760698287284292380457244444444704*(CODE hw$)
2330 DEF FN hx(hx$) = 552139657456858476091448888888896*(CODE hx$)
2340 DEF FN hy(hy$) = 110427931491371694182289777777792*(CODE hy$)
2350 DEF FN hz(hz$) = 2208558629827433883645795555555584*(CODE hz$)
2360 DEF FN ia(ia$) = 4417117259654867767291591111111168*(CODE ia$)
2370 DEF FN ib(ib$) = 88342345193097355345831822222336*(CODE ib$)
2380 DEF FN ic(ic$) = 1766846937861947106916364444444704*(CODE ic$)
2390 DEF FN id(id$) = 353369387572389421383272888888896*(CODE id$)
2400 DEF FN ie(ie$) = 706738775144778842766545777777792*(CODE ie$)
2410 DEF FN if(if$) = 1413477550289557645533091555555584*(CODE if$)
2420 DEF FN ig(ig$) = 2826955100579115291066183111111168*(CODE ig$)
2430 DEF FN ih(ih$) = 56539102011582305821323622222336*(CODE ih$)
2440 DEF FN ii(ii$) = 1130782040231646116442644444444704*(CODE ii$)
2450 DEF FN ij(ij$) = 2261564080463292232884888888896*(CODE ij$)
2460 DEF FN ik(ik$) = 452312816092658446576977777777792*(CODE ik$)
2470 DEF FN il(il$) = 9046256321853168931539555555555584*(CODE il$)
2480 DEF FN im(im$) = 1809251264370633786307911111111168*(CODE im$)
2490 DEF FN in(in$) = 36185025287412675726158222222336*(CODE in$)
2500 DEF FN io(io$) = 7237005057482535145231644444444704*(CODE io$)
2510 DEF FN ip(ip$) = 144740101149650702904628888888896*(CODE ip$)
2520 DEF FN iq(iq$) = 289480202299301405809257777777792*(CODE iq$)
2530 DEF FN ir(ir$) = 5789604045986028116185155555555584*(CODE ir$)
2540 DEF FN is(is$) = 1157920809197205623637031111111168*(CODE is$)
2550 DEF FN it(it$) = 23158416183944112472740622222336*(CODE it$)
2560 DEF FN iu(iu$) = 4631683236788822494548124444444704*(CODE iu$)
2570 DEF FN iv(iv$) = 926336647357764498909624888888896*(CODE iv$)
2580 DEF FN iw(iw$) = 185267329471552897819124977777792*(CODE iw$)
2590 DEF FN ix(ix$) = 3705346589431057956382499555555584*(CODE ix$)
2600 DEF FN iy(iy$) = 7410693178862115912764991111111168*(CODE iy$)
2610 DEF FN iz(iz$) = 14821386377724231825489822222336*(CODE iz$)
2620 DEF FN ja(ja$) = 2964277275544846365097964444444704*(CODE ja$)
2630 DEF FN jb(jb$) = 592855455108969273019592888888896*(CODE jb$)
2640 DEF FN jc(jc$) = 118571091021793854603919577777792*(CODE jc$)
2650 DEF FN jd(jd$) = 2371421820435877092078391555555584*(CODE jd$)
2660 DEF FN je(je$) = 4742843640871754184156783111111168*(CODE je$)
2670 DEF FN jf(jf$) = 94856872817435083683135662222336*(CODE jf$)
2680 DEF FN jg(jg$) = 1897137456348701673662713244444704*(CODE jg$)
2690 DEF FN jh(jh$) = 379427491269740334732542648888896*(CODE jh$)
2700 DEF FN ji(ji$) = 758854982539480669465085297777792*(CODE ji$)
2710 DEF FN jj(jj$) = 1517709965078961338930170555555584*(CODE jj$)
2720 DEF FN jk(jk$) = 30354199301579226778603
```



Más adelante, podremos completar la rutina para que sea capaz de aceptar y manejar controles de posicionamiento como «AT» o «TAB». De momento, no deberemos mandar estos códigos, ya que producirían la impresión de cosas sin sentido.

Sin embargo, si será necesario que la rutina sea capaz de aceptar el código «13» ya que el Basic siempre lo manda, a menos que la sentencia acabe en «?».

Lo mejor es construir una pequeña rutina «filtro» que sólo deje pasar códigos imprimibles, es decir, aquellos comprendidos entre 32 y 127 ambos inclusive. No obstante, este «filtro» debe ser capaz de identificar el código «13» para saltar a una rutina que lo gestione. Vamos a ir construyendo este filtro:

100	CANAL	CP	13
110	JR	Z, ENTER	
120	CP	32	
130	JR	NC, LB1	
140	RET		
150	LB1	CP	128
160	JR	C, IMP_A	
170	RET		
180	ENTER	LD	DE, (S_POSN)
190	LD	HL, #1821	
200	SBC	HL, DE	
210	EX	DE, HL	
220	JR	INC_L1	

Recuerde que entramos en esta rutina con el código en el acumulador. Así que lo primero que hacemos es compararlo con «13», si resulta que el código es 13, saltaremos a «ENTER» desde donde se llamará a cabo el incremento de línea. Exactamente, lo que hacemos es cargar las coordenadas en curso en «DE» y sal-

tar a la etiqueta «INC\_L1» (Incrementar Línea). Esta etiqueta deberemos colocarla en la línea 520 de la rutina «IMP\_A» ya que a partir de este punto, lo que se hace es, precisamente, incrementar el número de línea.

Si el código no fuera «13», entraríamos en la línea 120 donde lo comparamos con «31» y retornamos si es igual o menor. Si no, saltamos a la línea 150 donde lo volvemos a comparar con «128» y retornamos si es igual o mayor. Finalmente, si el código está dentro del rango, saltamos a «IMP\_A» para que se imprima.

Esta rutina de «filtro», deberá ir colocada inmediatamente antes de «IMP\_A» de forma que podamos añadirla y ensambiarlas juntas. Como «IMP\_A» es reubicable, no hay problema en cambiarla de dirección. En la Figura 9-20 podemos ver el listado completo de «CANAL» + «IMP\_A» tal como quedaría al ponerlas juntas. Dejamos al lector la tarea de ensambiar a mano (si no tiene ensamblador, esta corta rutina.

Vamos a ver cómo «activamos» este canal. Ya que necesitamos los canales «S» y «K» para manejarlos por el Sistema, lo que haremos será colocar esta rutina en el canal «P», es decir, el correspondiente a la impresora. Para ello, no hay más que almacenar en «(CHANS) + 15» la dirección a partir de la cual haremos ensambiar la rutina.

Mientras tengamos activado este canal, no podremos utilizar la impresora, así que será mejor que preveamos una forma de «desactivarla» y dejar el canal «P» con su anterior contenido. Almacenaremos éste de forma temporal

alturas, quien haya seguido atentamente el curso tiene que ser capaz de realizar las modificaciones necesarias para que puedan correr en cualquier otra dirección.

Ya hemos dicho que esta rutina podría ser una subrutina de un programa para gestionar la pantalla; de hecho, hay otra forma más fácil de utilizarla desde el Basic. Cualquier rutina en la que se entre con el código de un carácter en «A» y haga algo con ese carácter, puede ser utilizada por el Sistema Operativo como un canal de salida. Exactamente igual que el canal «S» (parte superior de la pantalla), el «K» (parte inferior) o el «P» (impresora). Para ello, vamos a ver qué son y cómo se utilizan los «Canales de comunicación».

### Los canales de comunicación

La configuración básica de un ordenador está compuesta por el microprocesador y la memoria. No obstante, este conjunto no sirve para nada si no se puede comunicar con el exterior. Por tanto, lo normal es que se acompañe de un teclado, una pantalla y una impresora. Estos elementos se denominan periféricos.

A nivel máquina, sabemos que el microprocesador utiliza «ports» para comunicarse con todos los periféricos excepto la pantalla que está configurada como una «tabla» en memoria que la ULA se encarga de enviar al televisor.

Para el microprocesador, esta tabla es como un periférico más y como tal es gestionada por el Sistema Operativo.

Cuando se trabaja en lenguaje de alto nivel como el Basic, no resulta cómodo an-

dar escribiendo y leyendo en «ports» o en posiciones de memoria. Es más fácil tener una o varias instrucciones de entrada y otras de salida y utilizarlas dirigidas a determinados canales que comunican con cada periférico. Pero ¿qué es un canal?

Existen canales de entrada y canales de salida. De momento, vamos a estudiar sólo los de salida porque son los que nos interesan para hacer funcionar nuestra rutina.

Un canal es, básicamente, una rutina escrita en código máquina que es llamada por el sistema operativo con el código de un carácter en «A». La rutina deberá enviar el carácter o el código al periférico correspondiente y, luego, devolverá el control al Sistema Operativo.

Un canal tiene que ser también capaz de manejar ciertos códigos de control tales como el retorno de carro (13), salto de línea (10), borrado (12) y, en general, aquellos con códigos comprendidos entre «0» y «31» que incluyen controles de color, posicionamiento de la impresión, etc. Un mismo código puede tener significados distintos según el canal que se esté utilizando; por ejemplo, la impresora no aceptará controles de color.

El Spectrum, en su versión básica, utiliza cuatro canales. Se denominan «R», «S», «K» y «P». Algunos son sólo de salida y otros admiten también entradas. Suponemos que todos nuestros lectores están acostumbrados a utilizar los canales de comunicación en Basic, así que no profundizaremos más en ello y nos dedicaremos a estudiarlos desde el punto de vista del código máquina. Para mayor infor-

mación sobre el cometido de cada canal, se puede consultar el manual de Basic que viene con el ordenador o el capítulo 29 del «CURSO DE BASIC» publicado por MICROHOBBY.

Sabemos que el Basic utiliza «corrientes» que enlazan con determinados canales. Algunas están ya asignadas como por ejemplo, las #1, #2 y #3 que enlazan, respectivamente, con los canales «K», «S» y «P». Toda la gestión de canales y corrientes es llevada a cabo por dos tablas de «offset» que se encuentran en determinado lugar de la memoria. La primera es la tabla de corrientes situada a partir de la dirección 23568 y compuesta por 38 bytes. Cada elemento de esta tabla ocupa dos octetos que contienen (en el orden inverso habitual) un número que indica el «offset» de la tabla de canales, es decir, el desplazamiento desde «CHANS» para acceder al canal unido a esa corriente.

Si la corriente en cuestión está cerrada, el elemento correspondiente de la tabla de corrientes contendrá «0000», justo como era de esperar. Todo esto se comprenderá mejor con una mirada a la Figura 9-19 donde hemos representado esquemáticamente las dos tablas de corrientes y canales, así como la conexión entre ellas.

En el momento de conectar el ordenador la situación es la que se muestra en la figura. Las corrientes #3, #2 y #1 no son accesibles desde el Basic y sólo las utiliza el Sistema. De hecho, el Basic no nos admite la instrucción: OPEN #4, «R» ya que el canal «R» que conecta con el área de traba-



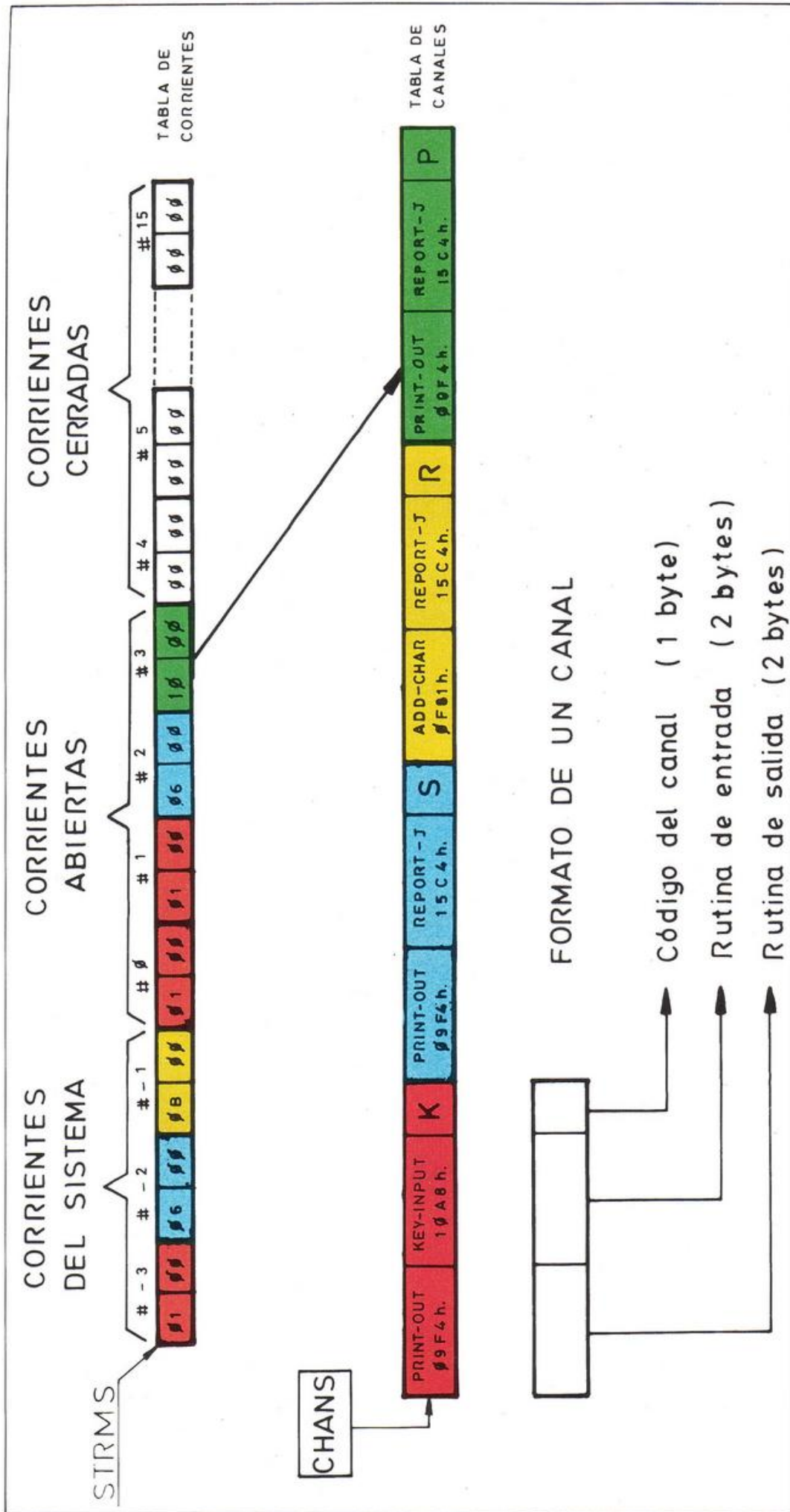


Fig. 9-19. Tablas de corrientes y canales.

jo sólo puede ser utilizado por el Sistema. Las corrientes #0, #1, #2 y #3 están permanentemente abiertas y conectadas con sus respectivos canales. No hay inconveniente en utilizar la instrucción «OPEN #» para conectarlas con otros, pero si

intentamos cerrarlas mediante «CLOSE #», obtendremos, por defecto, los canales inicialmente conectados. Habría, no obstante, una forma de cerrarlas que sería «PO-KEar» directamente «0» en «STRMS» + 10 con lo que quedaría cerrada la corriente

#2. No obstante, no le aconsejamos que lo haga ya que el Sistema se «colgaría» con toda seguridad.

Conociendo la disposición de estas tablas, podemos hacer algo mucho más interesante. Podemos cambiar la dirección de salida de uno de

los canales para apuntar a una rutina nuestra que gestione esta salida. Y aquí es donde entra «IMP-A»; esta rutina puede ser usada, con ciertas precauciones, como un canal de salida.

Vamos a ver qué nos manda el Basic a un canal. Supon-

gamos que vamos a ejecutar la instrucción:

La salida se hará por la corriente #2 que es la que corresponde, por defecto, al comando «PRINT». Los códigos enviados por el Basic serán:

PRINT AT 10.12; "HOLA"

22= Código de control «AT».  
10= Primer argumento de «AT».  
12= Segundo argumento de «AT».  
72= Código de «h».  
70= Código de «0».  
76= Código de «L».  
65= Código de «A».  
13= Retorno de carro.



Código Fuente	Hexadecimal	Decimal
BIT 0,A	C8,47	203,71
BIT 0,B	C8,40	203,64
BIT 0,C	C8,41	203,65
BIT 0,D	C8,42	203,66
BIT 0,E	C8,43	203,67
BIT 0,H	C8,44	203,68
BIT 0,L	C8,45	203,69
BIT 0,(HL)	C8,46	203,70
BIT 0,(IX+d)	D0,C8,d,46	221,203,d,70
BIT 0,(IY+d)	F0,C8,d,46	253,203,d,70
BIT 1,A	C8,4F	203,79
BIT 1,B	C8,48	203,72
BIT 1,C	C8,49	203,73
BIT 1,D	C8,4A	203,74
BIT 1,E	C8,4B	203,75
BIT 1,H	C8,4C	203,76
BIT 1,L	C8,4D	203,77
BIT 1,(HL)	C8,4E	203,78
BIT 1,(IX+d)	D0,C8,d,4E	221,203,d,78
BIT 1,(IY+d)	F0,C8,d,4E	253,203,d,78
BIT 2,A	C8,57	203,87
BIT 2,B	C8,50	203,80
BIT 2,C	C8,51	203,81
BIT 2,D	C8,52	203,82
BIT 2,E	C8,53	203,83
BIT 2,H	C8,54	203,84
BIT 2,L	C8,55	203,85
BIT 2,(HL)	C8,56	203,86
BIT 2,(IX+d)	D0,C8,d,56	221,203,d,86
BIT 2,(IY+d)	F0,C8,d,56	253,203,d,86
BIT 3,A	C8,5F	203,95
BIT 3,B	C8,58	203,88
BIT 3,C	C8,59	203,89
BIT 3,D	C8,5A	203,90
BIT 3,E	C8,5B	203,91
BIT 3,H	C8,5C	203,92
BIT 3,L	C8,5D	203,93
BIT 3,(HL)	C8,5E	203,94
BIT 3,(IX+d)	D0,C8,d,5E	221,203,d,94
BIT 3,(IY+d)	F0,C8,d,5E	253,203,d,94

1.- La rutina podría ser:

```

190 IMPR_2 LD B,0
200 BUCLE LD A,(DE)
210 LD C,0
220 BUC_2 RR A
230 RL (HL)
240 DEC C
250 JR NZ,BUC_2
260 INC DE
270 INC H
280 DJNZ BUCLE

```

Entre las líneas 210 y 250 hemos introducido otro bucle que vá sacando los bits uno a uno por la derecha de "A" y metiéndolos uno a uno, también por la derecha, en el octeto correspondiente del archivo de pantalla.

2.- La única modificación necesaria es que la nueva dirección no habrá de almacenarse en "CHANS"+15, sino en "CHANS"+5. Por tanto, solo habrá que cambiar la línea 1010 para que sea:

```
1010 LD DE,5
```

De paso, podemos modificar también la rutina "DESACT" cambiando, de la misma forma, la línea 1150.

3.- El procedimiento no puede ser más sencillo, basta con cargar en "DE" las nuevas coordenadas y hacer un salto a la etiqueta "SIGUE".

4.- Se almacenará, en el elemento #4 de la tabla, el dato "06" que corresponde al offset del canal "S". La dirección correspondiente al elemento #4 de la tabla de corrientes es "STRMS"+14, es decir, 23568+14 = 23582. En la tabla de canales no se producirá ninguna modificación.

5.- La rutina podría ser:

```

100 MULT_2 LD HL,0
110 LD B,5
120 LOOP SL A
130 JR NC,SIG
140 INC H
150 SIG DJNZ LOOP
160 LD L,A
170 RET

```

Como se ve, realizamos cinco desplazamientos a la izquierda en el acumulador, incrementando "H" cada vez que sale un bit por el indicador de acarreo. Finalmente, cargamos en "L" el dato que haya quedado en "A".

Fig. 10-1. Tabla de codificaciones para las instrucciones de prueba de bits.



## GRUPO DE INSTRUCCIONES DE MANIPULACION DE BITS

Estas instrucciones actúan sobre la más elemental unidad de información, el bit. No es frecuente encontrar en los procesadores instrucciones dedicadas a manejar individualmente los bits, aunque supone una gran comodidad y operatividad el tenerlas.

Es de todos conocido la cantidad de alternativas binarias que existen a nivel informativo (si-no, blanco-negro, alto-bajo, hombre-mujer, derecha-izquierda, etc.). Pues toda esa información que sólo tiene dos posibilidades, o sea, que es binaria la podemos almacenar en un bit.

En cualquier procedimiento mecanizado se emplea mucho esta posibilidad, supone un gran ahorro de memoria en el almacenamiento de datos. Para analizar el estado de un bit utilizado como soporte de información se emplean muchos tipos de instrucciones, por ejemplo, los operadores lógicos con máscaras, o bien las instrucciones de desplazamiento llevando el bit a la posición de signo o de acarreo. Esas instrucciones y otras como de suma y resta se emplean para activar o desactivar un bit. Pues bien, todo esto se puede realizar directamente con las instrucciones que vamos a ver.

En este grupo de instrucciones existen tres subgrupos, a saber:

- Prueba del bit (BIT): nos da la posibilidad de saber si un bit está activo o no.
- Activar bit o puesta a uno (SET): pone a 1 (activo) un bit.
- Desactivar bit o puesta a cero (RES): pone a 0 (desactivado o limpio) un bit.

El formato básico de estas instrucciones es el siguiente:

### CODIGO b, OPERANDO

Donde «b» indica el bit sobre el que se va a operar. Los bits se numeran de derecha a izquierda, de 0 a 7.

7 6 5 4 3 2 1 0

El bit sobre el que se va a operar (valor de «b») viene indicado, a su vez, por tres bits del código de operación según la siguiente tabla:

Bit	valor binario de «b»
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

### Prueba de bits

#### BIT b,r

#### OBJETO:

Pone en el indicador de condición «Z» el complemento del valor del bit indicado por «b» en el registro indicado por «r». El código de representación de «r» es el indicado más abajo.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

#### CODIGO DE MAQUINA:

1 1 0 0 1 0 1 1
0 1 + b → + r →

#### INDICADORES DE CONDICION QUE AFECTA:

Z; pone 1 - si el bit especificado es 0;

pone 0 - en cualquier otro caso

H; pone 1 - siempre

N; pone 0 - siempre

Valor del octeto de memoria 8328h

8328: 0 1 1 0 1 0 0 1 69h

#### Instrucción

DDh	1 1 0 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
04h	0 0 0 0 0 1 0 0
AEh	1 0 1 0 1 1 1 0

RES 5, (IX + 4)

Valor del octeto de memoria 8328h después de la ejecución

8328: 0 1 0 0 1 0 0 1 49h

#### RES b, (IX + d)

#### OBJETO:

Pone a 0 el valor del bit indicado por «b» en el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127.

#### CODIGO DE MAQUINA:

FDh	1 1 1 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
	← - d →
	1 0 ← b → 1 1 0

#### INDICADORES DE CONDICION QUE AFECTA:

Ninguno

#### CICLOS DE MEMORIA:

6

#### CICLOS DE RELOJ:

23

#### EJEMPLO:

RES 7, (IX - 10)

Contenido del registro índice «IX»

92h	1 0 0 1 0 0 1 0
4Ah	0 1 0 0 1 0 1 0

(IX):

Valor del octeto de memoria 9240h

9240h: 1 0 0 0 0 0 0 0

#### Instrucción

FDh	1 1 1 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
FBh	1 1 1 1 0 1 1 0
BEh	1 0 1 1 1 1 1 0

RES 7, (IX - 10)

Valor del octeto de memoria 9240h después de la ejecución

9240h: 0 0 0 0 0 0 0 0

#### Tablas de codificación

Dado el gran número de instrucciones que se utilizan para el manejo de bits, hemos realizado tres tablas de codificación. En la Figura 10-1 se encuentra la tabla con las instrucciones de prueba de bits, en la 10-2 la tabla con las de puesta a «1» y en la 10-3 las de puesta a «0». En la Figura 10-4 se encuentra la tabla resumida de indicadores y ciclos para todas estas instrucciones.

#### Los «Flags»

El término «Flag» se utiliza mucho en programación desde los primeros tiempos. Tiene su significado, al igual que

otros términos, en el inglés; en el cual quiere decir bandera o banderín en los deportes, como verbo se traduce por hacer señales con banderas. En informática se empezó a utilizar para indicar que determinada condición, estaba puesta o no, también se utilizaba la palabra «SWITCH», que significa interruptor.

La elección entre una palabra u otra (flag o switch) es más una cuestión de costumbre, aunque por regla general «flag» es un indicador de una condición y «switch» un cambio o bit que puede estar ON/OFF (conectado/desconectado). En ambos casos es una información binaria.

Los flags (banderas) son muy utilizados como condiciones por todos entendidos en la vida diaria. Por ejemplo una bandera roja en una playa indica que es peligroso bañarse; una serie de banderitas indican el camino a seguir en una pista de ski, etc.

También se da el uso de señales de tipo binario en otras aplicaciones, desde la protección de una cinta cassette a los semáforos. Todos ellos englobarían lo que en términos informáticos se entiende por flag.

Por tanto, flag sería una información binaria, que no tiene por qué ocupar más de un bit y que de cara a conseguir más efectividad debe ser fácil de poner, quitar y analizar.

Sería interminable enumerar aquí todas las aplicaciones que tienen los flags; de lo que se trata es de entender las posibilidades de su uso.

Uno de los usos más inmediatos es marcar las condiciones iniciales de un programa para posteriormente ir condicionando su ejecución.



7482h: 0 1 1 0 0 1 1 1

RES 3,H: CBh 9Ch  
1 1 0 0 1 0 1 1  
1 0 0 1 1 1 0 0

## Puesta a «0» de bits

**RES b, r**

### OBJETO:

Pone a 0 el bit indicado por «b» en el registro indicado por «r». El código de representación de «r» es el indicado más abajo.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

### CODIGO DE MAQUINA:

CBh  
1 1 0 0 1 0 1 1  
1 0 ← b → ← r →

### INDICADORES DE CONDICION QUE AFECTA:

Ninguno

### CICLOS DE MEMORIA:

2

### CICLOS DE RELOJ:

8

### EJEMPLO:

RES 3,H: CBh  
1 0 0 0 0 1 0  
1 0 0 1 0 0 1

### Valor del registro «H»

(H): 1 1 1 1 1 1 1 1 11h

### Instrucción

### Instrucción

RES 1, (HL): CBh 8Ch  
1 1 0 0 1 0 1 1  
1 0 0 0 1 1 1 0

Valor del octeto 8291h después de la ejecución

8291h: 0 0 0 0 0 0 0 0 00h

**RES b, (IX + d)**

### OBJETO:

Pone a 0 el valor del bit indicado por «b» en el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los valores desde -128 a +127.

### CODIGO DE MAQUINA:

00h CBh  
1 1 0 1 1 1 0 1  
1 1 0 0 1 0 1 1  
1 0 ← b → 1 1 0

### INDICADORES DE CONDICION QUE AFECTA:

Ninguno

### CICLOS DE MEMORIA:

6

### CICLOS DE RELOJ:

23

### EJEMPLO:

RES 5, (IX + 4): CBh  
1 1 0 0 1 0 1 1  
0 1 ← b → 1 1 0

Contenido del registro índice «IX»

(IX): 1 0 0 0 0 0 1 1 83h  
0 0 1 0 0 1 0 0 24h

### CICLOS DE MEMORIA:

2

### CICLOS DE RELOJ:

8

### EJEMPLO:

BIT 3,C: 75h  
0 1 1 1 0 1 0 1

### Valor del registro «C»

(C): 0 1 1 1 0 1 0 1 75h

### Instrucción

BIT 3,C: CBh 59h  
1 1 0 0 1 0 1 1  
0 1 0 1 1 0 0 1

El valor del registro «C» no varía con la ejecución Indicadores de condición después de la ejecución

S Z H PV N C

x 1 x 1 x x 0 x

**BIT b, (HL)**

### OBJETO:

Pone en el indicador de condición «Z» el complemento del valor del bit indicado por «b» en el octeto de memoria direccionado por el contenido del par de registros «HL».

### CODIGO DE MAQUINA:

CBh  
1 1 0 0 1 0 1 1  
0 1 ← b → 1 1 0

### INDICADORES DE CONDICION QUE AFECTA:

Z: pone 1 - si el bit especificado es 0;  
pone 0 - en cualquier otro caso

### CODIGO DE MAQUINA:

00h CBh  
1 1 0 1 1 1 0 1  
1 1 0 0 1 0 1 1  
0 1 ← b → 1 1 0

### INDICADORES DE CONDICION QUE AFECTA:

Z: pone 1 - si el bit especificado es 0;  
pone 0 - en cualquier otro caso

H: pone 1 - siempre  
N: pone 0 - siempre

### CICLOS DE MEMORIA:

5

### CICLOS DE RELOJ:

20

### EJEMPLO:

BIT 7, (IX + 20): 7fh  
0 1 1 1 1 1 1 1

Contenido del registro índice «IX»

A3h 82h  
1 0 1 0 0 0 1 1  
1 0 1 1 0 0 1 0

Valor del octeto de memoria A3C6h

A3C6h: 7fh  
0 1 1 1 1 1 1 1

### Instrucción

BIT 7, (IX + 20): DDh CBh 14h 79h  
1 1 0 1 1 1 0 1  
1 1 0 0 1 0 1 1  
0 0 0 1 0 1 0 0  
0 1 1 1 1 1 1 0

El valor del octeto de memoria A3C6h no varía con la ejecución Indicadores de condición después de la ejecución



S Z H PW N C  
x 1 x 1 x x 0 x

### BIT b, (IY + d)

#### OBJETO:

Pone en el indicador de condición «Z» el complemento del valor del bit indicado por «b» en el octeto de memoria direccionado por el contenido del registro índice «IY» más el entero de desplazamiento «d», el cual puede adquirir los valores desde  $-128$  a  $+127$ .

#### CODIGO DE MAQUINA:

FDh	1 1 1 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
	$\leftarrow d \rightarrow$
	0 1 $\leftarrow b \rightarrow$ 1 1 0

#### INDICADORES DE CONDICION QUE AFECTA:

Z: pone 1 - si el bit especificado es 0;  
 pone 0 - en cualquier otro caso  
 H: pone 1 - siempre  
 N: pone 0 - siempre

#### CICLOS DE MEMORIA:

5

#### CICLOS DE RELOJ:

20

#### EJEMPLO:

BIT 4, (IY-7)

Contenido del registro índice «IY»

IY:	0 1 1 1 0 0 1
	0 0 0 1 0 1 0

Valor del octeto de memoria 7913h

7913h:	0 1 0 1 1 0 1 0
--------	-----------------

#### Instrucción

FDh	1 1 1 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
FBh	1 1 1 1 1 0 0 1
61h	0 1 1 0 0 1 1 0

BIT 4, (IY-7)

El valor del octeto de memoria 7913h no varía con la ejecución

Indicadores de condición después de la ejecución

S	Z	H	PW	N	C
x	0	x	1	x	x

#### Puesta a «1» de bits

### SET b,r

#### OBJETO:

Pone a 1 el bit indicado por «b» en el registro indicado por «r». El código de representación de «r» es el indicado más abajo.

Registro	Código
B	000
C	001
D	010
E	011
H	100
L	101
A	111

#### CODIGO DE MAQUINA:

CBh	1 1 0 0 1 0 1 1
	1 1 $\leftarrow b \rightarrow$ r $\rightarrow$

#### INDICADORES DE CONDICION QUE AFECTA:

79H	0 1 1 1 0 0 1
1Ah	0 0 0 1 0 1 0

#### CICLOS DE RELOJ:

15

#### EJEMPLO:

SET 5, (HL)

Contenido del par de registros «HL»

HL:	1 0 0 1 0 0 1 1
HL:	0 1 1 1 0 1 0 0

Valor del octeto de memoria 9374h

9374h:	1 0 0 0 0 1 1 0
--------	-----------------

#### Instrucción

SET 5, (HL):	1 1 0 0 1 0 1 1
	1 1 1 0 1 1 1 0

Valor del octeto 9374h después de la ejecución

9374h:	1 0 1 0 0 1 1 0
--------	-----------------

### SET b, (IX + d)

#### OBJETO:

Pone a 1 el valor del bit indicado por «b» en el octeto de memoria direccionado por el contenido del registro índice «IX» más el entero de desplazamiento «d», el cual puede adquirir los valores desde  $-128$  a  $+127$ .

#### CODIGO DE MAQUINA:

DDh	1 1 0 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
	$\leftarrow d \rightarrow$
	1 1 $\leftarrow b \rightarrow$ 1 1 0

#### INDICADORES DE CONDICION QUE AFECTA:

Ninguno

#### CICLOS DE MEMORIA:

6

#### CICLOS DE RELOJ:

23

#### EJEMPLO:

SET 6, (IX+0)

Contenido del registro índice «IX»

IX:	1 0 1 1 0 0 1 1
	0 1 1 0 0 1 0 0

Valor del octeto de memoria B364h

B364h:	1 1 1 1 1 1 1 1
--------	-----------------

#### Instrucción

SET 6, (IX+0):	1 1 0 1 1 1 0 1
	1 1 0 0 1 0 1 1
	0 0 0 0 0 0 0 0
	1 1 1 1 0 1 1 0

Valor del octeto de memoria B364h después de la ejecución

B364h:	1 1 1 1 1 1 1 1
--------	-----------------

### SET b, (IY + d)

#### OBJETO:

Pone a 1 el valor del bit indicado por «b» en el octeto de memoria direccionado por el contenido del registro índice «IY» más el entero de desplazamiento «d», el cual puede adquirir los valores desde  $-128$  a  $+127$ .

#### CODIGO DE MAQUINA:

FDh	1 1 1 1 1 1 0 1
CBh	1 1 0 0 1 0 1 1
	$\leftarrow d \rightarrow$
	1 1 $\leftarrow b \rightarrow$ 1 1 0

#### INDICADORES DE CONDICION QUE AFECTA:

Ninguno

#### CICLOS DE MEMORIA:

6

#### CICLOS DE RELOJ:

23

#### EJEMPLO:

SET 2, (IY-1)

Contenido del registro índice «IY»

IY:	0 1 1 1 0 1 0 0
	1 0 1 1 0 0 1 1

Valor del octeto de memoria 74B2h

74B2h:	0 1 1 0 0 0 1 1
--------	-----------------

#### Instrucción

SET 2, (IY-1):	1 1 1 1 1 1 0 1
	1 1 0 0 1 0 1 1
	1 1 1 1 1 1 1 1
	1 1 0 1 0 1 1 0

Valor del octeto de memoria 74B2h después de la ejecución



mente, no existe un algoritmo de cálculo que sea válido para toda la pantalla (al menos, nosotros no lo hemos encontrado, pero si algún lector lo encuentra, no deje de comunicárnoslo).

La solución que nosotros proponemos para este problema, no tiene por qué ser, necesariamente, la única ni siquiera la mejor (en informática, como en casi todos los aspectos de la vida, no existen verdades absolutas), pero al menos, funciona de forma bastante aceptable y permite adaptarla para otros usos.

De momento, imaginemos que numeramos los scans de «0» a «191»; el primer scan de la pantalla, empezando por arriba, sería el scan «0» y el último sería el scan «191». Ahora, el problema se reduce a escribir dos rutinas: una que nos sintetice el número de scan a partir de una dirección dada y la otra, que nos sintetice una dirección a partir del número de scan. Por supuesto, esta segunda rutina no deberá afectar a los bits de la dirección que definen el número de columna, para poder cumplir el requisito de que funcione con un octeto perteneciente a cualquier columna de la pantalla.

Una vez que consigamos tener el número de scan, podemos incrementarlo para pasar al scan siguiente o decrementarlo para pasar al anterior. Aún podemos sacar más partido de este procedimiento: si, al decrementar el número de scan, éste pasase de valer «0» a valer «255» o si, al incrementarlo, pasase de valer «191» a valer «192», la nueva dirección obtenida caería fuera del archivo de pantalla, por lo que sería errónea y nos in-

dicaría que la dirección de partida pertenecía al primer o último scan respectivamente. Recordemos, del capítulo anterior, el formato de una dirección de pantalla:

BYTE MÁS SIGNIFICATIVO							
0	1	0	L4	L3	S2	S1	S0
F130			LINEA		SCAN		
BYTE MENOS SIGNIFICATIVO							
L2	L1	L0	C4	C3	C2	C1	C0
LINEA			COLUMNA				

Para obtener el número de scan a partir de aquí, no tendremos más que colocar los bits que definen la línea y el scan, en el orden adecuado:

NUMERO DE SCAN									
L4	L3	L2	L1	L0	S2	S1	S0	SCAN	
LINEA									

Supongamos que tenemos la dirección de pantalla en el registro «HL» y queremos formar el número de scan en el registro «A». Transportaremos los bits como se muestra en la Figura 10-7. La rutina para hacerlo, podría ser la siguiente:

140	S15_1	LD	A,H
150	AND	#07	
160	LD	B,A	
170	LD	A,H	
180	AND	#18	
190	SLA	A	
200	SLA	A	
210	SLA	A	
220	OR	B	
230	LD	B,A	
240	LD	A,L	

INSTRUCCIONES DE PUESTA A «1» (II)

Código Fuente	Hexadecimal	Decimal
SET 0,A	CB,C7	203,199
SET 0,B	CB,C0	203,192
SET 0,C	CB,C1	203,193
SET 0,D	CB,C2	203,194
SET 0,E	CB,C3	203,195
SET 0,H	CB,C4	203,196
SET 0,L	CB,C5	203,197
SET 0,(HL)	CB,C6	203,198
SET 0,(IX+d)	DD,CB,d,C6	221,203,d,198
SET 0,(IY+d)	FD,CB,d,C6	253,203,d,198
SET 1,A	CB,CF	203,207
SET 1,B	CB,C8	203,200
SET 1,C	CB,C9	203,201
SET 1,D	CB,CA	203,202
SET 1,E	CB,CB	203,203
SET 1,H	CB,CC	203,204
SET 1,L	CB,CD	203,205
SET 1,(HL)	CB,CE	203,206
SET 1,(IX+d)	DD,CB,d,CE	221,203,d,206
SET 1,(IY+d)	FD,CB,d,CE	253,203,d,206
SET 2,A	CB,D7	203,215
SET 2,B	CB,D0	203,208
SET 2,C	CB,D1	203,209
SET 2,D	CB,D2	203,210
SET 2,E	CB,D3	203,211
SET 2,H	CB,D4	203,212
SET 2,L	CB,D5	203,213
SET 2,(HL)	CB,D6	203,214
SET 2,(IX+d)	DD,CB,d,D6	221,203,d,214
SET 2,(IY+d)	FD,CB,d,D6	253,203,d,214
SET 3,A	CB,DF	203,223
SET 3,B	CB,D8	203,216
SET 3,C	CB,D9	203,217
SET 3,D	CB,DA	203,218
SET 3,E	CB,DB	203,219
SET 3,H	CB,DC	203,220
SET 3,L	CB,DD	203,221
SET 3,(HL)	CB,DE	203,222
SET 3,(IX+d)	DD,CB,d,DE	221,203,d,222
SET 3,(IY+d)	FD,CB,d,DE	253,203,d,222

INSTRUCCIONES DE PUESTA A «1» (III)

Código Fuente	Hexadecimal	Decimal
SET 4,A	CB,E7	203,231
SET 4,B	CB,E0	203,224
SET 4,C	CB,E1	203,225
SET 4,D	CB,E2	203,226
SET 4,E	CB,E3	203,227
SET 4,H	CB,E4	203,228
SET 4,L	CB,E5	203,229
SET 4,(HL)	CB,E6	203,230
SET 4,(IX+d)	DD,CB,d,E6	221,203,d,230
SET 4,(IY+d)	FD,CB,d,E6	253,203,d,230
SET 5,A	CB,EF	203,239
SET 5,B	CB,E8	203,232
SET 5,C	CB,E9	203,233
SET 5,D	CB,EA	203,234
SET 5,E	CB,EB	203,235
SET 5,H	CB,EC	203,236
SET 5,L	CB,ED	203,237
SET 5,(HL)	CB,EE	203,238
SET 5,(IX+d)	DD,CB,d,EE	221,203,d,238
SET 5,(IY+d)	FD,CB,d,EE	253,203,d,238
SET 6,A	CB,F7	203,247
SET 6,B	CB,F0	203,240
SET 6,C	CB,F1	203,241
SET 6,D	CB,F2	203,242
SET 6,E	CB,F3	203,243
SET 6,H	CB,F4	203,244
SET 6,L	CB,F5	203,245
SET 6,(HL)	CB,F6	203,246
SET 6,(IX+d)	DD,CB,d,F6	221,203,d,246
SET 6,(IY+d)	FD,CB,d,F6	253,203,d,246
SET 7,A	CB,FF	203,255
SET 7,B	CB,F8	203,248
SET 7,C	CB,F9	203,249
SET 7,D	CB,FA	203,250
SET 7,E	CB,FB	203,251
SET 7,H	CB,FC	203,252
SET 7,L	CB,FD	203,253
SET 7,(HL)	CB,FE	203,254
SET 7,(IX+d)	DD,CB,d,FE	221,203,d,254
SET 7,(IY+d)	FD,CB,d,FE	253,203,d,254

Fig. 10-2. Tabla de codificación para las instrucciones de puesta a «1» de bits.



fuera del bit 0 del registro «E»; podríamos añadir, entre la línea 300 y la 310, las siguientes instrucciones:

```
304 BIT 0,E
306 JR Z,NOCA_2
```

Antes de entrar en la rutina, deberíamos fijar el flag a «0» o «1» en función de qué tipo de scroll necesitemos; «SET 1,E» si queremos «scroll esférico» o «RES 1,E» si queremos «scroll lineal».

Una vez visto el scroll a derecha, para hacer la rutina no le realice a izquierda no tenemos más que copiar ésta cambiando algunas instrucciones:

```
70 LD HL,2527
80 LD C,192
90 BUC_2 LD B,32
100 AND A
110 BUC_1 RL (HL)
120 DEC HL
130 DJNZ BUC_1
140 JR NC,NOCA_1
150 LD (VAR),HL
160 LD IX,(VAR)
170 SET 0,(IX+32)
180 NOCA_1 DEC C
190 JR NZ,BUC_2
200 RET
```

En la línea 70 inicializamos el puntero a la última dirección de pantalla, ya que este scroll lo vamos a hacer de abajo a arriba. En la 110 rotamos a la izquierda en lugar de hacerlo a la derecha. En 120 decrementamos en vez de in-

crementar igual que en 180. Finalmente, en la línea 170, el bit que ponemos a «1» es el último del scan en lugar de el primero. Por lo demás, la rutina es igual que la anterior y sólo se diferencia en las etiquetas.

De nuevo, si no queremos «scroll esférico», podemos eliminar las líneas 140, 150, 160 y 170 o insertar las siguientes:

```
134 BIT 0,E
136 JR Z,NOCA_1
```

Para controlarlo mediante el mismo flag que en la rutina anterior. Aunque nada impide utilizar flags distintos para cada rutina con lo que podríamos hacer girar la pantalla de forma «esférica» en un sentido y de forma «lineal» en otro.

Como se ve, las posibilidades de utilización de estas rutinas sólo están limitadas por la imaginación del usuario. En cualquier caso, es importante escribir las rutinas que han de formar parte de una biblioteca, de forma que sean fácilmente modificables para adaptarla a cada uso concreto que se les quiera dar.

Tal vez a algún lector le interese saber cómo puede adaptar estas rutinas para que realicen el «scroll» de sólo un tercio de la pantalla. La adaptación no es difícil, y hemos preferido incluirla como uno de los ejercicios de este capítulo. Le recomendamos, por tanto, que intente resolverlo por sí mismo antes de mirar la solución.

El ensamblado de estas rutinas no presenta ningún problema. Le recomendamos que lo haga en hexadecimal y lo coloque en el formato del «CARGADOR UNIVERSAL» (10 octetos por línea con suma de comprobación al final). Si lo hace así, puede comprobar el resultado en las líneas 310 a 360 del programa Basic que las maneja y que publicamos en este capítulo con el título de «SCROLL TOTAL».

Vamos ahora con el scroll vertical. El planteamiento es el mismo de antes: tenemos que subir la pantalla un scan hacia arriba de forma que el primer scan pase a ser el último, el segundo pase al primero, el tercero al segundo y así sucesivamente. El sistema consiste en coger el primer scan, almacenarlo en otro lugar de la memoria (por ejemplo, el buffer de impresora), entrar en un bucle que, a partir del segundo scan, transfiera cada uno al anterior y, finalmente, deberemos recuperar el scan que guardamos en otra zona de memoria y transferirlo al lugar del último (esto, sólo si queremos que el scroll se realice de forma «esférica»). La solución sería muy sencilla si las direcciones de scans consecutivos fueran también consecutivas; por desgracia, esto no es así.

Lo primero que necesitamos es una rutina que nos dé la dirección del scan siguiente a uno dado. Para poder utilizar esta rutina en otros programas, podemos hacerla de forma que no nos de siempre la primera dirección del scan, sino la del octeto que se encuentra inmediatamente debajo de aquél de cuya dirección partimos. Lamentablemente,

Fig. 10-3. Tabla de codificación para las instrucciones de puesta a «0» de bits.

INSTRUCCIONES DE PUESTA A «0» (I)			INSTRUCCIONES DE PUESTA A «0» (II)		
Código Fuente	Hexadecimal	Decimal	Código Fuente	Hexadecimal	Decimal
RES 0,A	C8,87	203,135	RES 4,A	C8,A7	203,167
RES 0,B	C8,88	203,128	RES 4,B	C8,A8	203,168
RES 0,C	C8,81	203,129	RES 4,C	C8,A1	203,161
RES 0,D	C8,82	203,130	RES 4,D	C8,A2	203,162
RES 0,E	C8,83	203,131	RES 4,E	C8,A3	203,163
RES 0,H	C8,84	203,132	RES 4,H	C8,A4	203,164
RES 0,L	C8,85	203,133	RES 4,L	C8,A5	203,165
RES 0,(HL)	C8,86	203,134	RES 4,(HL)	C8,A6	203,166
RES 0,(IX+d)	D0,C8,d,86	221,203,d,134	RES 4,(IX+d)	D0,C8,d,A6	221,203,d,166
RES 0,(IY+d)	F0,C8,d,86	253,203,d,134	RES 4,(IY+d)	F0,C8,d,A6	253,203,d,166
RES 1,A	C8,8F	203,143	RES 5,A	C8,AF	203,175
RES 1,B	C8,88	203,136	RES 5,B	C8,A8	203,168
RES 1,C	C8,89	203,137	RES 5,C	C8,A9	203,169
RES 1,D	C8,8A	203,138	RES 5,D	C8,AA	203,170
RES 1,E	C8,8B	203,139	RES 5,E	C8,AB	203,171
RES 1,H	C8,8C	203,140	RES 5,H	C8,AC	203,172
RES 1,L	C8,8D	203,141	RES 5,L	C8,AD	203,173
RES 1,(HL)	C8,8E	203,142	RES 5,(HL)	C8,AE	203,174
RES 1,(IX+d)	D0,C8,d,8E	221,203,d,142	RES 5,(IX+d)	D0,C8,d,AE	221,203,d,174
RES 1,(IY+d)	F0,C8,d,8E	253,203,d,142	RES 5,(IY+d)	F0,C8,d,AE	253,203,d,174
RES 2,A	C8,97	203,151	RES 6,A	C8,87	203,183
RES 2,B	C8,98	203,144	RES 6,B	C8,88	203,176
RES 2,C	C8,91	203,145	RES 6,C	C8,81	203,177
RES 2,D	C8,92	203,146	RES 6,D	C8,82	203,178
RES 2,E	C8,93	203,147	RES 6,E	C8,83	203,179
RES 2,H	C8,94	203,148	RES 6,H	C8,84	203,180
RES 2,L	C8,95	203,149	RES 6,L	C8,85	203,181
RES 2,(HL)	C8,96	203,150	RES 6,(HL)	C8,86	203,182
RES 2,(IX+d)	D0,C8,d,96	221,203,d,150	RES 6,(IX+d)	D0,C8,d,86	221,203,d,182
RES 2,(IY+d)	F0,C8,d,96	253,203,d,150	RES 6,(IY+d)	F0,C8,d,86	253,203,d,182
RES 3,A	C8,9F	203,159	RES 7,A	C8,8F	203,191
RES 3,B	C8,98	203,152	RES 7,B	C8,88	203,184
RES 3,C	C8,99	203,153	RES 7,C	C8,89	203,185
RES 3,D	C8,9A	203,154	RES 7,D	C8,8A	203,186
RES 3,E	C8,9B	203,155	RES 7,E	C8,8B	203,187
RES 3,H	C8,9C	203,156	RES 7,H	C8,8C	203,188
RES 3,L	C8,9D	203,157	RES 7,L	C8,8D	203,189
RES 3,(HL)	C8,9E	203,158	RES 7,(HL)	C8,8E	203,190
RES 3,(IX+d)	D0,C8,d,9E	221,203,d,158	RES 7,(IX+d)	D0,C8,d,8E	221,203,d,190
RES 3,(IY+d)	F0,C8,d,9E	253,203,d,158	RES 7,(IY+d)	F0,C8,d,8E	253,203,d,190



ejemplo en un programa que establezca un diálogo con el usuario, podría definirse un flag que indique el sexo de éste, en un momento que el programa pregunte si es hombre o mujer se activaría o no el flag en función de la respuesta; a partir de este momento siempre que el programa se dirija personalmente al usuario consultará el flag y empleará el género masculino o femenino según corresponda.

En esta misma línea el programa monitor del SPEC-TRUM, activa un flag cuando se presiona por primera vez la tecla CAPS LOCK, por lo tanto al ir a poner una letra consulta el flag y como está activo la pone mayúscula; al presionar una segunda vez la tecla CAPS LOCK se borrará el flag. Es decir, lo que en última instancia decide si las letras son mayúsculas o minúsculas es un flag.

Un uso más amplio sería el de indicar los atributos de cualquier dato que se tenga almacenado. Por ejemplo, en un registro de un fichero de libros podríamos indicar con flags: si es bueno o malo, si lo hemos leído o no, su género literario, etc.

Otro uso, al cual podemos denominar dinámico, sería un flag que cambia en base a una condición variable. Por ejemplo, en un programa de juego, después que el jugador ha fallado varias veces seguidas se puede activar un flag, al tenerlo activo y por las consultas correspondientes el juego baja automáticamente el nivel; una vez realizados varios aciertos seguidos desactiva el flag y por tanto subirá el nivel.

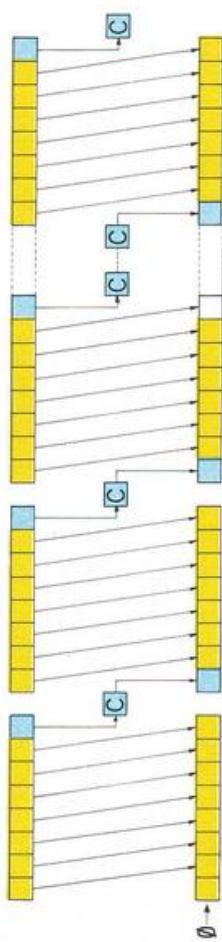


Fig. 10-6. Rotación encadenada de octetos.

bucle que rota a la derecha un scan completo de la forma que se ve en la Figura 10-6. En la línea 310 comprobamos si el bit que ha pasado al indicador de acarreo es cero, si es así, saltamos a «NOCA-2», si no, transferimos a «IX» por lo que la instrucción «SET 7, (HL-32)» no existe; por otro lado, la transferencia de «HL» a «IX» no podemos hacerla de forma directa, ya que no disponemos de una instrucción como «LD IX,HL» por lo que tenemos que recurrir al empleo de una variable intermedia que hemos denominado «VAR» y está situada en 23728 que es una dirección de las variables del Sistema que éste no utiliza. El último bloque cierra el bucle «BUC-4» y retorna cuando éste ya se ha ejecutado 192 veces.

En algunas aplicaciones, puede interesar que la parte de pantalla que va desapareciendo por la derecha, no aparezca por la izquierda, sino que se pierda definitivamente, en ese caso, bastará con suprimir las líneas 310, 320, 330 y 340. Aunque sería más elegante utilizar un «flag» de forma que, cuando el flag estuviera a «1», la pantalla rotase en forma esférica y cuando estuviese a «0», se perdiese lo que se escapara por un lado. Supongamos que el flag

Hemos separado el listado en 5 bloques para estudiarlo mejor. El primer bloque está compuesto por las líneas 240 y 250; la primera inicializa el valor del puntero «HL» al primer octeto de la rutina para la segunda fija el número de iteraciones que tendrá el bucle «BUC-4».

En el segundo bloque, se fija el número de iteraciones del bucle «BUC-3» y se pone a «0» el indicador de acarreo. El tercer bloque contiene el

```
=====
240 LD HL,16384
=====
```

270 CODIGO MAQUINA

NEMONICO	INDICADORES										No.DE BYTES	CICLOS MEM. REL.
	S	Z	X	H	P	V	N	C				
BIT b,r	x	+	x	1	x	x	+	+			2	2
BIT b, (HL)	x	+	x	1	x	x	+	+			2	3
BIT b, (IX+d)	x	+	x	1	x	x	+	+			4	5
BIT b, (IY+d)	x	+	x	1	x	x	+	+			4	5
SET b,r	.	.	x	.	x	.	.	.	.	.	2	2
SET b, (HL)	.	.	x	.	x	.	.	.	.	.	2	4
SET b, (IX+d)	.	.	x	.	x	.	.	.	.	.	4	6
SET b, (IY+d)	.	.	x	.	x	.	.	.	.	.	4	6
RES b,r	.	.	x	.	x	.	.	.	.	.	2	2
RES b, (HL)	.	.	x	.	x	.	.	.	.	.	2	4
RES b, (IX+d)	.	.	x	.	x	.	.	.	.	.	4	6
RES b, (IY+d)	.	.	x	.	x	.	.	.	.	.	4	6

#### NOTAS:

- Los signos tienen el siguiente significado:  
 "V": El indicador cambia de valor de acuerdo con el resultado de la instrucción.  
 "X": El bit adquiere un estado indeterminado.  
 ".": El indicador no es afectado por la instrucción y conserva su anterior contenido.  
 "0": El indicador se pone siempre a "cero".  
 "1": El indicador se pone siempre a "uno".

- La letra "r" indica cualquiera de los registros: "A", "B", "C", "D", "E", "H" o "L".

Fig. 10-4. Tabla resumida de indicadores y ciclos para las instrucciones de manipulación de bits.

#### UTILIZACION DE LOS FLAGS

jable usar el bit como flag.  
Una vez decidido que el bit «b» del octeto «o» indica la condición «c», nos limitaremos a utilizar la instrucción SET cuando lo queremos activar, la instrucción RES cuando lo queremos desactivar y la instrucción BIT asociada con un salto condicional del indicador «Z» cuando queramos analizar su estado. Esta

Un flag podría ser utilizado en cualquier campo, un octeto, varios octetos o un bit. Dada la condición binaria del flag y la facilidad que tiene el micro-procesador Z-80 de manejar bits de forma independiente, parece lo más aconsejable



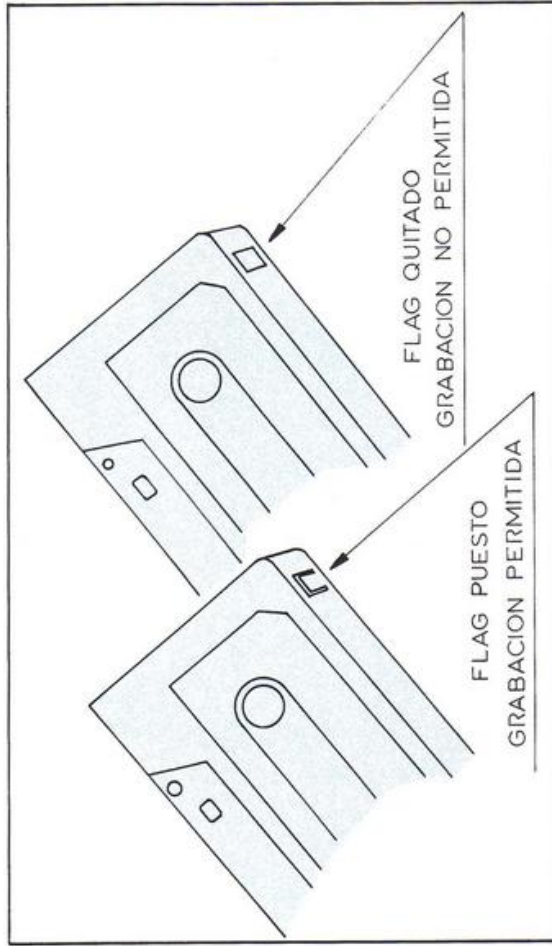


Fig. 10-5. Un ejemplo de «Flag» en la vida real.

sería la forma más inmediata y sencilla de utilizar un flag, tanto de manera estática como dinámica.

El uso de flags como atributos puede realizarse de formas más diversas. Para verlo lo ilustraremos siguiendo un ejemplo.

Supongamos que tenemos un fichero de libros en el que cada registro o ficha tiene el siguiente formato y contenido.

TÍTULO:	30 octetos
AUTOR:	30 octetos
TEMA:	1 octeto
bit 7 = 1	novela
bit 6 = 1	ensayo
bit 5 = 1	ciencia
bit 4 = 1	historia
bit 3 = 1	cuento
bit 2 = 1	poesía
bit 1 = 1	teatro
bit 0 = 1	otros

TOS, algunos de los atributos se llenarán después de haber leído el libro. Toda esta información, que ocupa tan poco espacio, se actualizaría con las instrucciones SET dado que el estado inicial es cero.

Supongamos que queremos ahora localizar una novela que no hayamos leído. El problema se limitaría a ir buscando con la instrucción BIT en el octeto TEMA de cada libro, si el bit 7 está activo. Cada vez que localicemos uno, miraremos con la misma instrucción el bit 4 del octeto ATRIBUTOS de ese libro; una vez localizado uno que no esté activo sacaremos a pantalla el TÍTULO y AUTOR. Para localizar este libro sólo hemos utilizado la instrucción BIT de manipulación de bits.

Una nueva utilización sería el caso de que un amigo nos pide un libro prestado y nos pone las siguientes condiciones: que trate sobre ciencia, de la cual sabe poco y le gus-

ATRIBUTOS: 1 octeto  
bit 7 = 1 bueno  
bit 6 = 1 interesante  
bit 5 = 1 malo  
bit 4 = 1 leído  
bit 3 = 1 para entendidos  
bit 2 = 1 para mayores  
bit 1 = 1 en inglés  
bit 0 = 1 en francés

EDITORIAL: 15 octetos  
FECHA: 1 octeto  
MES: 1 octeto  
AÑO: 1 octeto

Podría ponerse mucha más información, pero para lo que nos ocupa nos dedicaremos principalmente a los octetos denominados TEMA y ATRIBUTOS.

En el momento que estamos creando el fichero con los libros que se tienen o cada vez que se incluye un nuevo libro, se irán llenando los campos y activando los bits del octeto TEMA y ATRIBU-

taria que estuviera escrito en inglés, por otra parte nosotros preferimos dejarle uno que ya hayamos leído. Para localizar los libros de ciencia emplearíamos el mismo método que en el ejemplo anterior, pero éste resultaría engorroso para analizar el octeto de ATRIBUTOS. Un método bueno sería preparar un octeto con los bits que queremos analizar activos y hacer una operación lógica AND con el octeto ATRIBUTOS. Este octeto con los bits activos se llama «máscara». Para nuestro ejemplo la máscara sería la siguiente:

1 0 0 1 1 0 1 0

con la que analizaremos los atributos BUENO, LEIDO, PARA ENTENDIDOS y EN INGLÉS; pero como nuestro amigo no tiene mucha idea del tema tendremos que descartar todos los que tengan el atributo PARA ENTENDIDOS, esto se puede solucionar comparando el resultado de la instrucción AND con el valor:

1 0 0 1 0 0 1 0

con lo que sólo nos quedaríamos con los libros de CIENCIA, que son BUENOS, BASICOS, en INGLÉS y LEIDOS. Resumiendo la secuencia de instrucciones sería:

Primero: BIT para localizar los libros de tema CIENCIA.  
Segundo: una vez localizado uno, aplicar al octeto ATRIBUTOS un operador AND con una máscara que tenga unos en los flags que queremos analizar.

Tercero: aplicar al resultado una resta (CP) con los va-

lores de los flags que queremos que estén activos.

Cuarto: cuando esta instrucción nos dé como resultado cero, sacar a la pantalla el TÍTULO Y AUTOR pues ya hemos localizado un libro.

Los flags en informática están por todas partes. Dentro de las variables del Sistema en el Spectrum existen tres octetos con diferentes flags para control del programa monitor éstos son: «FLAGS», «TV-FLAG» y «FLAGS2». También los indicadores de condición son flags con un uso dinámico.

Como habrá comprendido el lector, la utilización de flags es ilimitada. Es muy recomendable utilizarlos al máximo pues es una información que ocupa muy poca memoria y su acceso resulta sencillo. Ya señalamos al principio de este curso que la primera aplicación del microprocesador Z-80 fue industrial, seguramente éste es el motivo por el cual trae incorporadas estas instrucciones de manipulación de bits y no trae otras más usuales como son la multiplicación, división o desplazamientos de más de un bit. En cualquier caso, la obligación de toda persona que pretende hacer un programa es usar al máximo la potencia del procesador y ahorrar la memoria del ordenador que éste está empleando.

## Ejemplos

En el Prólogo de este curso (número 52 de MICRO-HOBBY), publicamos una rutina de ejemplo que servía para mover la pantalla a izquierda y derecha, pixel a pixel. La única finalidad de esa rutina era mostrar al lector lo que se

podía conseguir empleando el Código Máquina. En ese momento, no se explicó su funcionamiento dado que, se supone, que el lector carecía del conocimiento necesario de Código Máquina para comprenderlo. Ahora, por fin, ha llegado el momento de explicar esta rutina y desarrollar otra que permita hacer lo mismo, pero en el sentido vertical, como ya prometíamos en el Prólogo.

Empezaremos por explicar la rutina de «SCROLL» lateral. Suponemos que casi todos los lectores teclearían el programa que manejaba esta rutina y se quedarían impacientes esperando la prometida rutina de «SCROLL» vertical. Posiblemente, algunos ya hayan intentado profundizar en su funcionamiento. Todas las instrucciones que se utilizan han sido vistas ya; sólo faltaban por ver las de las líneas 170 y 340 (SET 0, (IX + 32) y SET 7, (IX - 32) respectivamente), precisamente, las que hemos visto en este capítulo.

Suponemos que las rutinas no existen y empezaremos por el principio, es decir, por plantearnos el problema a resolver: Se trata de desplazar toda la pantalla un pixel a la derecha, teniendo en cuenta que los bits que se escapan por la derecha de cada scan deberán entrar por la izquierda del mismo, para conseguir el efecto de una pantalla circular. La solución más sencilla consiste en ir desplazando, uno a uno, los 192 scans que componen la pantalla empujando por el primero y acabando por el último.

Para cada scan, utilizaremos un bucle de 32 iteraciones en el que entraremos con «HL» apuntando al primer oc-



# SOLUCION A LOS EJERCICIOS

## SOLUCIONES A LOS EJERCICIOS

1.- Habrá que cambiar las siguientes líneas:

```
70 LD HL,20479 ; Final del segundo tercio.
80 LD C,64 ; 64 scans en un tercio.
240 LD HL,18432 ; Inicio del segundo tercio.
250 LD C,64 ; 64 scans en un tercio.
```

2.- Habrá que modificar la rutina, a partir de "FIN\_1" de la siguiente forma:

```
810 FIN_1 PUSH DE ; Preserva "DE".
820 LD A,(23681) ; Carga variable.
830 BIT 2,A ; Comprueba el flag.
840 JR NZ,FIN_2 ; Si es "1", salta a "FIN_2".
850 LD HL,23296 ; Borra el scan que se había
860 LD DE,23297 ; almacenado en el buffer de
870 LD BC,31 ; impresora.
880 XOR A
890 LD (HL),A
900 LDIR
910 FIN_2 POP DE ; Recupera "DE"
920 LD HL,23296 ; Sigue como antes de la
930 LD BC,32 ; modificación.
940 LDIR
950 RET
```

3.- El punto de entrada por "SCRA\_1" corresponde al scroll ascendente, y por "SCRA\_2" al scroll descendente.

```
100 SCRA_1 LD HL,22528 ; Transfiere los primeros
110 LD DE,23296 ; 32 bytes del archivo de atrib-
120 LD BC,32 ; utos a los 32 primeros del
130 LDIR ; buffer de impresora.
140 LD DE,22528 ; Transfiere, 32 bytes hacia
150 LD BC,768 ; atras, el archivo de atributos
160 LDIR ; más los 32 primeros bytes del
170 RET ; buffer de impresora y retorna.
180 SCRA_2 LD HL,23295 ; Transfiere, 32 bytes hacia
190 LD DE,23327 ; delante, el archivo de atrib-
200 LD BC,768 ; utos entrando en los 32 primeros
210 LDOR ; bytes del buffer de impresora.
220 LD HL,23327 ; Transfiere los 32 primeros
230 LD BC,32 ; bytes del buffer de impresora
240 LDOR ; a los 32 primeros de atributos.
250 RET ; Retorna.
```

Aprovechamos la circunstancia de que el buffer de impresora se encuentra inmediatamente a continuación del archivo de atributos.

dos huecos que nos quedan libres.

280: Finalmente, los mezclamos con lo que teníamos en «B».

A la salida de la rutina, tendremos en «A» el número de scan, comprendido entre «0» y «191». El proceso se entien- de perfectamente si se va mi- rando la Figura 10-7 al mismo tiempo.

Vamos ahora con la segun- da rutina, en este caso, ten- dremos que sintetizar, en «HL», la dirección correspon- diente a un scan dado. Escri- biremos la rutina de forma que no se alteren los cinco bits inferiores de «L» ya que son los que definen el núme- ro de columna. Suponemos que el número de scan se en- cuentra en el registro «A»; aunque de nuevo, utilizare- mos el «B» como almacena- miento temporal:

```
380 SBC_2 LD B,A
390 LD H,#40
400 AND #07
410 OR H
420 LD H,A
430 LD A,B
440 AND #00
450 SRL A
```

430: Ahora, volvemos a recu- perar el número de scan desde «B».

440: Esta vez, aislamos los dos bits de más peso con la máscara 00h (ob- sérvase el frecuente uso que hacemos de las máscaras en estas ruti- nas).

450: En esta línea y las dos siguientes, rotamos es- tos dos bits, tres luga- res a la derecha.

480: Los mezclamos con el contenido de «H».

490: Y almacenamos el re- sultado en «H». Ya tene- mos completo el byte alto de nuestra direc- ción.

500: Cargamos en «A» el byte bajo.

510: Borrados los tres bits superiores, dejando los cinco inferiores inaltera- dos.

520: Reponemos el conteni- do de «L». Ahora, «L» só- lo contiene los cinco bits que definen la co- lumna.

530: Recuperamos el núme- ro de scan desde «B».

540: Ahora, aislamos los bits 3, 4 y 5.

550: En esta línea y la si-

De la misma forma que an- tes, vamos a explicar, línea a línea, el funcionamiento de esta rutina:

380: Guardamos en «B» el contenido de «A» para que no sea destruido al operar.

390: Cargamos 40h en «H» para poner el bit 6 a «1» y los restantes a «0» (recuerde que 40h = 01000000b).

400: Aislamos los tres bits de menos peso de «A».

410: Los mezclamos con el contenido de «H».

420: Y almacenamos el re- sultado en «H».

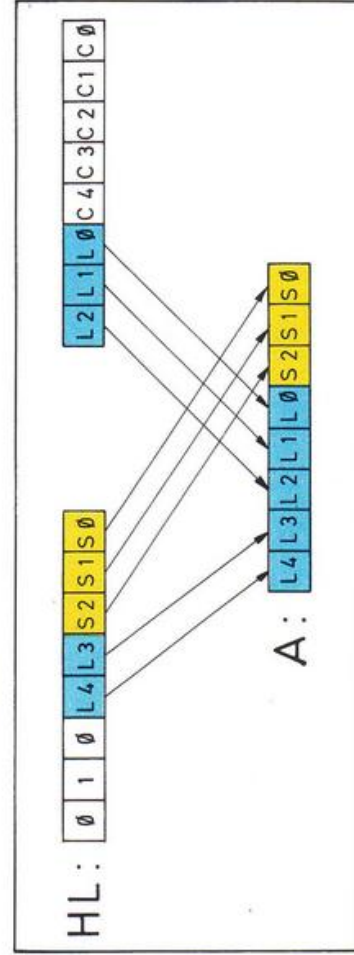


Fig. 10-7. Síntesis del número de scan a partir de la dirección de la pantalla.



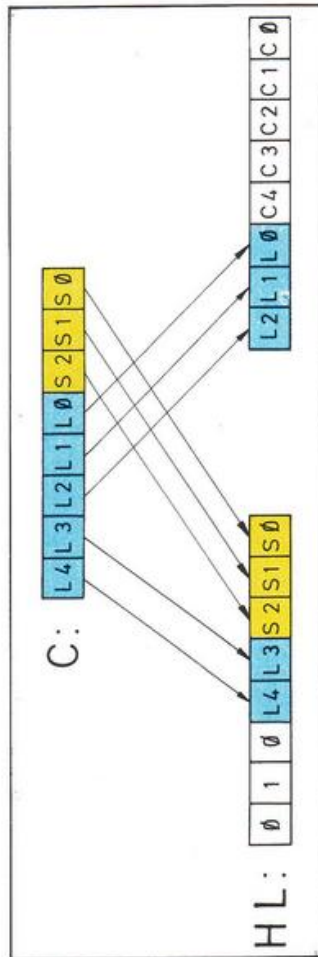


Fig. 10.8. Síntesis de la dirección de pantalla a partir del número de scan.

guiente, los rotamos dos lugares a la izquierda. 570: Los mezclamos con el contenido de «L». 580: Y finalmente, almacenamos el resultado en «L» con lo que queda completo el byte bajo de la dirección.

De nuevo, el procedimiento se entiende mejor consultando la Figura 10-8. A la salida de la rutina, tendremos en «HL» una dirección de pantalla correspondiente al scan cuyo número estaba en «A» y a la misma columna que definían los cinco bits inferiores de «L». Como detalle adicional, todavía tendremos en «B» el número de scan por si pudiera ser útil.

Ya tenemos las dos rutinas que necesitábamos, la primera que llama «SIG-1» y la segunda «SIG-2». Volvamos al problema inicial: se trataba de hallar la dirección del scan siguiente o del anterior a una dirección dada. Para hallar el scan siguiente, primero utilizamos «SIG-1», luego incrementamos el número de scan y finalmente, utilizamos «SIG-2». En el caso de tener que hallar la dirección del scan anterior, utilizamos el mismo procedimiento salvo

que decrementamos el número de scan en lugar de incrementarlo. En ambos casos, deberemos escribir las rutinas de forma que salgan con el indicador de acarreo a «1» si la dirección es correcta, y a «0» si cae fuera de la pantalla.

En los dos casos, tendremos que utilizar las rutinas «SIG-1» y «SIG-2» por lo que no parece muy adecuado escribir dos rutinas, una para hallar el scan siguiente y otra para el anterior. Lo mejor es escribir una sola rutina que nos dé la dirección del scan siguiente o del anterior en función del contenido de un flag. Utilizaremos el bit 0 del registro «C» como flag, la rutina tendrá tres puntos de entrada: Si entramos por «SIG», nos dará la dirección del octeto del scan siguiente a aquél cuya dirección contenga «HL». Si entramos por «ANT», nos dará la dirección del octeto del scan anterior. Finalmente, podremos entrar por «SIG-1» habiendo fijado, previamente, el bit 0 de «C» a «1» si queremos la dirección siguiente, o a «0» si queremos la anterior. Estas serían las líneas que habría que añadir a «SIG-1» y «SIG-2» para tener la rutina completa:

```

110 SIG SET 0,C
120 JR SIG_1
130 ANT RES 0,C
=====
140 SIG_1 .....
...
=====
290 BIT 0,C
300 JR Z,ANT_1
310 INC A
320 CP #0
330 PUSH AF
340 JR SIG_2
350 ANT_1 SUB 1
360 CCF
370 PUSH AF
=====
380 SIG_2 .....
...
=====
590 POP AF
600 RET

```

Vamos a explicar la rutina línea a línea:  
110: Ponemos a «1» el flag para que la dirección que hallamos sea la del scan siguiente.  
120: Saltamos a «SIG-1» (línea 140).  
130: Ponemos a «0» el flag para que la dirección

55086	250	SIG_1	LD	A,H	55133	560	SRL	A
55087	260	AND	#07		55135	570	SRL	A
55089	270	LD	B,A		55137	580	SRL	A
55090	280	LD	A,H		55139	590	OR	H
55091	290	AND	#18		55140	600	LD	H,A
55093	300	SLA	A		55141	610	LD	A,L
55095	310	SLA	A		55142	620	AND	#1F
55097	320	SLA	A		55144	630	LD	L,A
55099	330	OR	B		55145	640	LD	A,B
55100	340	LD	B,A		55146	650	AND	#38
55101	350	LD	A,L		55148	660	SLA	A
55102	360	AND	#E0		55150	670	SLA	A
55104	370	SRL	A		55152	680	OR	L
55106	380	OR	B		55153	690	LD	L,A
55108	390	OR	A		55154	700	POP	AF
55109	400	BIT	0,C		55155	710	POP	BC
55111	410	JR	Z,ANT_1		55156	720	POP	DE
55113	420	INC	A		55157	730	JR	NC,FIN_1
55114	430	CP	#C0		55159	740	PUSH	HL
55116	440	PUSH	AF		55160	750	PUSH	BC
55117	450	JR	SIG_2		55161	760	LD	BC,32
55119	460	SUB	1		55164	770	LDIR	BC
55121	470	CCF	AF		55166	780	POP	BC
55122	480	PUSH	AF		55167	790	POP	HL
55123	490	LD	B,A		55168	800	JR	BUC_3
55124	500	LD	H,#40		55170	810	LD	HL,23296
55126	510	AND	#07		55173	820	LD	BC,32
55128	520	OR	H		55176	830	LDIR	BC
55129	530	LD	H,A		55178	840	RET	
55130	540	LD	A,B					
55131	550	AND	#C0					

Pass 2 errors: 000

Pass 2 errors: 000

Fig. 10-10. Listado completo de la rutina scroll vertical.

## EJERCICIOS

- 1.- Modifique las rutinas de scroll a derecha e izquierda para que trabajen, exclusivamente, sobre el segundo tercio de pantalla, es decir, el central.
- 2.- Modifique la rutina de scroll vertical de forma que lo que se pierda por un extremo de la pantalla (arriba o abajo), no aparezca por el contrario (abajo o arriba) si el bit 2 de la dirección 23681 es «0» (scroll lineal) y funcione normalmente si este bit es «1» (scroll esférico).
- 3.- Escriba dos rutinas que realicen el scroll vertical en el archivo de atributos. Una de las rutinas deberá hacer el scroll hacia arriba y la otra, hacia abajo. El scroll ha de ser «esférico».



das del bucle vamos tratando los scans uno a uno hasta que terminemos con toda la pantalla, momento en el que salimos por «FIN\_1».

En la línea 810 («FIN\_1»), cargamos en «HL» la dirección inicial del buffer de impresora que será el origen de la última transferencia. El destino será el último scan que hayamos procesado, cuando dirección se encuentra ya en «DE» por lo que no es necesario cargarla. En 820 cargamos, de nuevo, la longitud en «BC»; no preservamos los registros porque es la última transferencia y ya no nos importa que se pierdan sus contenidos. Finalmente, realizamos la transferencia en la línea 830 y retornamos en 840.

Esta vez, se pone más complicado el modificar la rutina para que el scroll no sea «esférico»; el procedimiento consiste en borrar el buffer de impresora antes de efectuar la última transferencia. La modificación no es demasiado difícil de realizar. No hemos querido privar al lector del placer de realizarla por sí mismo,

MES» en la variable «SEED», lo cual no siempre resulta conveniente; en ese caso, tal vez sea mejor que las llame con «LEZ z=USR ...».

También es importante señalar que estas rutinas hacen el scroll solamente del fichero de pantalla, no del de atributos. Este último es bastante más fácil de «scrollar» ya que las direcciones correspondientes a líneas consecutivas, son también consecutivas. De nuevo, es un placer que tampoco queramos arrastar al lector, así que lo dejamos para los ejercicios.

Hasta aquí, hemos visto las instrucciones de manejo de bits. En el próximo capítulo, veremos las que se utilizan para llamar a subrutinas, aprenderemos algo más de la pila y, en los ejemplos, terminaremos el «procesador de pantallas» que, estamos seguros, resultará muy útil para que el lector lo incluya en sus propios programas. Ahora, le recomendamos que intente resolver los ejercicios antes de pasar al capítulo siguiente.

\*HISOFT GENSYM ASSEMBLER\*  
ZX SPECTRUM

Copyright HISOFT 1983  
CURSO C/M MICROHOB

Pass 1 errors: 0/0

50 \*C-  
60 \*D+  
70 ;  
80 ; SCROLL\_VERTICAL  
90 ;  
55060 100 ORG 55060

55060 110 SCR\_3 SET 0,C  
55062 120 LD HL,16384  
55065 130 JR SCR  
55067 140 SCR\_4 RES 0,C  
55069 150 LD HL,22496  
55072 160 SCR PUSH HL  
55073 170 LD DE,23296  
55076 180 PUSH BC  
55077 190 LD BC,32  
55080 200 LDIR  
55082 210 POP BC  
55083 220 POP HL  
55084 230 BUC\_3 PUSH HL  
55085 240 PUSH BC

sea, esta vez, la correspondiente al scan anterior.

- 140 a 280: RUTINA «SIG-1».
- 290: Comprobamos el flag.
- 300: Si es «0», saltamos a «ANT-1» (línea 350).
- 310: Incrementamos el número de scan.
- 320: Lo comparamos con 192 para poner a «1» el acarreo sólo si es menor.
- 330: Guardamos el estado actual de los indicadores (el único que nos interesa es el de acarreo).
- 340: Saltamos a «SIG-2» (línea 380).
- 350: Decrementamos el número de scan. Utilizamos «SUB 1» en lugar de «DEC a» ya que esta última no afecta al indicador de acarreo.
- 360: Complementamos el indicador de acarreo para que sólo valga «1» si la dirección es correcta.
- 370: Guardamos el estado de los indicadores.
- 380 a 580: RUTINA «SIG-2».
- 590: Recuperamos los indicadores para recuperar el de acarreo.
- 600: Retornamos.

A la salida de esta rutina, tendremos en «HL» la nueva dirección que podremos considerar correcta si el indicador de acarreo se encuentra a «1»; en caso contrario, sabemos que nos hemos salido de la pantalla. También, tendremos en «A» el número de scan que, de momento, no vamos a utilizar pero que alguna vez puede resultarnos útil.

De nuevo, le recomendamos que ensamble la rutina en hexadecimal y utilice el formato del «CARGADOR

UNIVERSAL». Deberá quedarle algo así:

- 1 08C118070B817C560747 1186
- 2 70E618C8270B270B2780 1280
- 3 477DE5E0B3FCB3FB0CB 1561
- 4 4128863CFEC0F5180AD6 1104
- 5 013FF5472640E687B467 1902
- 6 78E6C0B3FCB3FCB3FE4 1520
- 7 677DE61F6F78E638CB27 1248
- 8 C827B565FF1C90000000 976

En la Figura 10-9 puede ver el listado completo de esta rutina. La hemos ensamblado a partir de 60500 pero es perfectamente reubicable.

Ahora que podemos calcular la dirección de cada scan, estamos en disposición de preparar las rutinas de scroll vertical. A diferencia con el horizontal, esta vez utilizaremos una sola rutina que será capaz de realizar tanto el scroll ascendente como el descendente, en función del estado de un flag. De nuevo, utilizaremos como flag el bit 0 del registro «C», que deberá estar a «1» para un scroll ascendente y a «0» para uno descendente.

La rutina tendrá dos puntos de entrada: por «SCR-3» realizará un scroll ascendente de un pixel y por «SCR-4» lo hará descendente. Un punto de entrada alternativo será por «SCR» donde habrá que entrar con el flag a «1» y «HL» conteniendo el número 16384 para un scroll ascendente, o bien, con el flag a «0» y «HL» conteniendo 22496 para un scroll descendente.

El procedimiento a seguir consta de tres fases: Primero, transferimos el primer scan o el último (según se trate de

100	ORG	55060
110	SCR_3	SET 0,C
120	LD	HL,16384
130	JR	SCR
140	SCR_4	RES 0,C
150	LD	HL,22496
160	SCR	PUSH HL
170	LD	DE,23296
180	PUSH	BC
190	LD	BC,32
200	LDIR	
210	POP	BC
220	POP	HL
230	BUC_3	PUSH HL
240	PUSH	BC



Copyright HISOFT 1983  
CURSO C/M MICROHOBBY

Pass 1 errors: 00

```

10 *C-
20 *D+
30 ;
40 ; SIG/ANT
50 ;
60500 100 SIG ORG 60500
60500 110 SIG SET 0,C
60502 120 JR SIG_1
60504 130 ANT RES 0,C
60506 140 SIG_1 LD 0,H
60507 150 AND #07
60509 160 LD B,A
60510 170 LD A,H
60511 180 AND #18
60513 190 SLA A
60515 200 SLA A
60517 210 SLA A
60519 220 OR B
60520 230 LD B,A
60521 240 LD A,L
60522 250 AND #E0
60524 260 SRL A
60526 270 SRL A
60528 280 OR B
60529 290 BIT 0,C
60531 300 JR Z,ANT_1
60533 310 INC #C0
60534 320 CP AF
60536 330 PUSH AF
60537 340 JR SIG_2

```

Pass 2 errors: 00

Table used: 69 from 170

Fig. 10.9. Listado completo de la rutina «SIG/ANT».

## PROGRAMA 1

```

10 REM
20 REM ** SCROLL TOTAL **
30 REM Curso C/M MICROHOBBY
40 REM
50 CLEAR 54999: LET d=55000
60 DEF FN a(a$,n)=16*(CODE a$(n+
n)-48-7*(a$(n)>"9"))+(CODE a$(n+
n)-48-7*(a$(n+1)>"9"))
70 PRINT #1:"ESPERE 10 SEGUNDO
S"
80 FOR f=1 TO 18
90 READ a$,s: LET c=0
100 FOR n=1 TO LEN a$-1 STEP 2
110 LET a$=FN a(a$,n):POKE d,a
120 NEXT n
130 IF c<s THEN PRINT "<ERROR>
en la linea: ";d+10*f: STOP
140 NEXT f
200 REM DEMOSTRACION DEMOSTRACI
ON DE SCROLL TOTAL "PULSE"
210 CLS
220 REM Para SCROLL Arriba"
230 REM Para SCROLL Abajo"
240 REM Para SCR

```

```

926 DATA "B05CDDCB0FE0C20E8C9"
1002 DATA "013FF572640E607B467"
1002 DATA "78E6C0C83FCB3FCBFB4"
1002 DATA "677DE61F6F78E5638CB27"
1002 DATA "CB27B58FF1C1D1300BES"
1002 DATA "C5012000ED80C1E118AA"
1002 DATA "21005B012000ED80C900"
1002 DATA "771

```

```

=====
250 SIG_1 .....
...
=====
710 POP BC
720 POP DE
730 JR NC,FIN_1
740 PUSH HL
750 PUSH BC
760 LD BC,32
770 LDIR
780 POP BC
790 POP HL
800 JR BUC_3
810 FIN_1 LD HL,23296
820 LD BC,32
830 LDIR
840 RET
=====

```

Hemos puesto 55060 como dirección de origen para que esta rutina quede situada justo a continuación de las de scroll lateral; pero tanto una como otras son reubicables, así que puede ensambirlas en cualquier dirección.

Todas las transferencias se realizan con LDIR para que la rutina sea rápida. Como todos los lectores saben, la instrucción «LDIR» exige que «HL» contenga la primera edición del bloque origen, «DE» la del bloque destino y «BC» el número de bytes a transferir; pero, en nuestro caso, estos registros son punteros que con-

tienen las direcciones de inicio de los scans, y el registro «C» contiene el flag que indica si el scroll es ascendente o descendente, así que tendremos que salvar algunos registros en la pila, antes de cada transferencia. Podríamos haber utilizado el set de registros alternativos; pero, en muchos casos, nos valdrá para la transferencia, el valor de algún registro que ya tengamos, valor que perderíamos al ejecutar un «EXX».

Ahora ya, veamos el funcionamiento de la rutina: En 110 y 120 fijamos las condiciones iniciales para un scroll ascendente y saltamos a «SCR» en 130. En 140 y 150 fijamos las condiciones iniciales para un scroll descendente y continuamos en «SCR». Primero, guardamos «HL», cargamos la dirección del buffer de impresora en «DE» (línea 170), guardamos «BC» y cargamos en él la longitud de un scan (línea 190). En 200 transferimos un scan al buffer de impresora y, en 210 y 220, recuperamos los anteriores contenidos de «BC» y «HL». El scan transferido habrá sido el primero si «HL» contenía 16384, o el último si contenía 22496.

A partir de la línea 230, entramos en un bucle donde iremos transfiriendo cada scan al anterior o posterior (en función del estado del flag) y del que saldremos cuando la di-

rección que nos dé «SIG\_1» caiga fuera de la pantalla. Primero, preservamos los contenidos de «HL» y «BC», ya que la rutina «SIG\_1» los destruye. A continuación, entramos en «SIG\_1» que va desde la línea 250 a la 700. En 710 y 720 recuperamos el contenido de «BC» y pasamos a «DE» el anterior contenido de «HL»; el nuevo contenido de «HL» será el scan anterior o el siguiente, es decir, «HL» contendrá el origen y «DE» el destino para efectuar la transferencia de los 32 octetos de un scan. Antes de ello, tenemos que comprobar si estamos, aún, dentro de la pantalla, lo que hacemos en la línea 730 en función del estado del indicador de acarreo (recuerde que, a la salida de «SIG\_1», el acarreo está a «1» si la dirección es correcta y a «0» si cae fuera de la pantalla); si el acarreo está a «0», saltamos a «FIN\_1», si no, continuamos. En 740 y 750 volvemos a preservar «HL» y «BC» antes de realizar la transferencia. En 760 cargamos la longitud del scan en «BC» y, en 770, transferimos el scan. A continuación, recuperamos «BC» y «HL» y volvemos al inicio del bucle. En este punto, «HL» contiene la dirección del siguiente scan a procesar, es decir, el scan de destino de la siguiente transferencia; de esta forma, en sucesivas pasa-



«CC»	binario	significado
NZ	000	no cero
Z	001	cero
NC	010	no acarreo
C	011	acarreo
PO	100	paridad impar o no desbordamiento
PE	101	paridad par o desbordamiento
P	110	positivo
M	111	negativo

Contenido del registro «SP»

(SP):	1 1 1 1 0 1 1 1	F7h
	0 0 0 0 0 1 0 0	04h

Instrucción

CALL C,802Eh	1 1 1 0 1 1 1 0	DCh
	0 0 1 0 1 1 1 0	2Eh
	1 0 0 0 0 0 0 0	80h

Contenido del registro «SP» después de la ejecución

(SP):	1 1 1 1 0 1 1 1	F7h
	0 0 0 0 0 1 0 0	02h

Contenido de los octetos F702h y F703 después de la ejecución

F702h:	0 1 0 0 0 1 1 0	46h
F703h:	0 1 1 1 0 1 0 1	75h

Contenido del registro «PC» después de la ejecución

(SP):	1 0 0 0 0 0 0 0	80h
	0 0 1 0 1 1 1 0	2Eh

Recuerde:

Cuando los operandos de una instrucción indican una dirección de memoria, en el primer octeto del operando se pone la mitad menos significativa de la dirección y en el siguiente la más significativa; en una palabra, se invierten.

EJEMPLO:

CALL C,802Eh
--------------

Indicador de condición

C=1

Dirección del primer octeto de la instrucción

0 1 1 1 0 1 0 1	75h
0 1 0 0 0 1 1 1	43h

288 CODIGO MAQUINA

en el octeto inferior y la dirección señalada por (SP) + 1 en el octeto superior. A continuación incrementa en 2 el registro «SP». Con esto efectúa la vuelta desde una subrutina a la instrucción siguiente a una «CALL» que realizó la entrada.

CODIGO DE MAQUINA:

1 1 0 0 1 0 0 1	C9h
-----------------	-----

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

3

CICLOS DE RELOJ:

10

EJEMPLO:

RET
-----

Contenido del registro «SP»

(SP):	1 1 1 1 0 1 1 1	F7h
	0 0 0 0 0 1 0 0	02h

Contenido de los octetos F702h y F703h

F702h:	0 1 0 0 0 1 1 0	46h
F703h:	0 1 1 1 0 1 0 1	75h

Instrucción

RET	1 1 0 0 1 0 0 1	C9h
-----	-----------------	-----

Contenido del registro «SP» después de la ejecución

(SP):	1 1 1 1 0 1 1 1	F7h
	0 0 0 0 0 1 0 0	04h

Contenido del registro «PC» después de la ejecución

(PC):	0 1 1 1 0 1 0 1	75h
	0 1 0 0 0 1 1 0	46h

OBJETO:

Carga en el registro contador de programa «PC» el último dato de la pila de máquina, poniendo el contenido de la dirección señalada por (SP)

289 CODIGO MAQUINA

## GRUPO DE INSTRUCCIONES DE LLAMADA Y RETORNO

ta operación se ejecutan.

La forma en que se ejecutan estas instrucciones es la siguiente. En el momento en que se necesitan ejecutar se salta al comienzo de la subrutina, cuando se termina de ejecutar la última instrucción, se salta a la instrucción siguiente a la que salto a éstas. Este grupo de instrucciones es lo que se llama subrutina y la forma de saltar y retornar con el microprocesador Z-80 es con las instrucciones CALL y RET.

Las instrucciones CALL (llamada) y RET (retorno) utilizan la pila de máquina controlada por el registro «SP». Para ser más exactos son una mezcla de instrucciones PUSH y POP con las de cambio de secuencia JP.

La ejecución de una instrucción «CALL nn» es equivalente a la secuencia: «PUSH PC» y «LD PC,nn» suponiendo, claro está, que existieran estas instrucciones. Por otro lado, la instrucción «RET» sería equivalente a «POP PC».

De esta forma, vemos que las instrucciones de «llamada y retorno» constituyen una forma abreviada de escribir lo que, en realidad, serían instrucciones de carga trabajando sobre el registro «PC» (contador de programa); de igual manera que podíamos interpretar una instrucción «JP nn»

como si fuera «LD PC,nn».

Resumiendo: cuando ejecutamos una instrucción de llamada a subrutina, el microprocesador mete en la pila el contenido actual del «PC» (que estará apuntando a la siguiente instrucción) y carga en él, la dirección de la subrutina. Cuando retornamos desde subrutina, lo que hace el microprocesador es tomar el último dato de la pila y cargarlo en el «PC».

Dado que utilizamos la pila para almacenar las direcciones de retorno, es posible entrar en una subrutina desde otra, o lo que es lo mismo desde una subrutina llamar a otra, esto se conoce como anidamiento. La cantidad de subrutinas que pueden anidarse está sólo limitado por el tamaño de la pila de máquina, el tamaño de la pila de máquina está limitado por la colocación inicial de ella y la memoria disponible. Hay que tener un cuidado especial con las instrucciones RET, para llegar al punto de origen tiene que haber tantos RET como CALL se hayan realizado, ya que se va retornando cada vez a la última dirección de la pila.

En la Figura 11-1 hay una representación gráfica de llamadas a subrutina. Desde una secuencia de programa —llamada principal— se en-

CODIGO MAQUINA 281



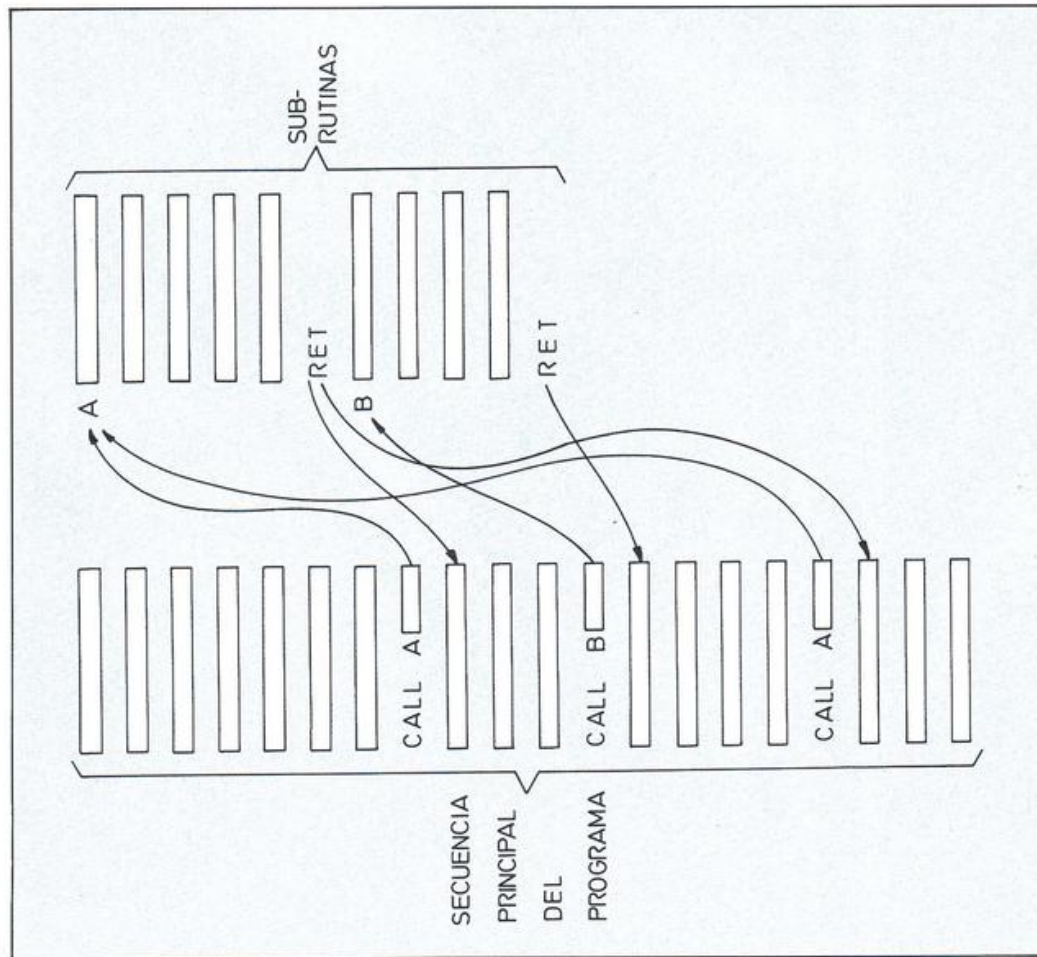


Fig. 11-1. Gráfico de sub-rutinas.

tra en varias subrutinas A y B, desde diferentes puntos. La Figura 11-2 muestra el anidamiento de subrutinas hasta un tercer nivel. Como se observará, una subrutina siempre tiene que terminar por «RET».

Por último en la Figura 11-3 se muestra el funcionamiento de la pila de máquina en

la instrucción RET se retorna a la última dirección guardada en la pila, de tal forma que siempre se garantiza el retorno a la instrucción, siguiente al «CALL» correspondiente. El registro «SP» va apuntando cada vez al nivel en uso, pero de la memoria no se borra el contenido de la pila, esta es la razón por la que a pesar de

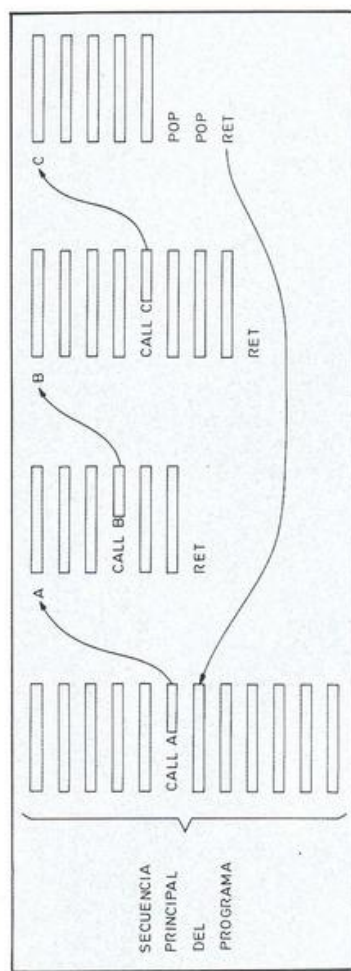


Fig. 11-4. Utilización de «pop» para provocar un salto en la secuencia de retorno.

control de programa «PC» en el número de octetos que tiene la instrucción, en este caso tres. Después guarda el valor del registro «PC» en la pila de máquina poniendo el contenido del octeto superior en la dirección señalada por (SP)-1 y el octeto inferior en la dirección señalada por (SP)-2. A continuación decrementa en 2 el registro «SP». Finalmente y para realizar el salto a la subrutina direccionada por el operando «nn», carga éste en el registro «PC».

#### EJEMPLO:

CALL 4856h

Dirección del primer octeto de la instrucción

70h  
00h

Contenido del registro «SP»

F7h  
06h

#### Instrucción

CALL 4856h: CDh 56h 48h

Contenido del registro «SP» después de la ejecución

F7h  
04h

Contenido de los octetos F704h y F705 después de la ejecución

F704h: 00000011  
F705h: 01100000

Contenido del registro «PC» después de la ejecución

#### OBJETO:

Lo primero que hace el microprocesador, al igual que en todas las instrucciones, es incrementar el registro de control de programa «PC» en el número de octetos que tiene la instrucción, en este caso tres. Si la condición «cc» es verdadera, guarda el valor del registro «PC» en la pila de máquina poniendo el contenido del octeto superior en la dirección señalada por (SP)-1 y el octeto inferior en la dirección señalada por (SP)-2; decrementa en 2 el registro «SP» y realiza el salto a la subrutina direccionada por el operando «nn», cargando éste en el registro «PC». Si la condición «cc» es falsa, ejecuta la siguiente instrucción. Los códigos nemotécnicos de condición «cc» y su valor binario para el microprocesador son indicados a continuación.



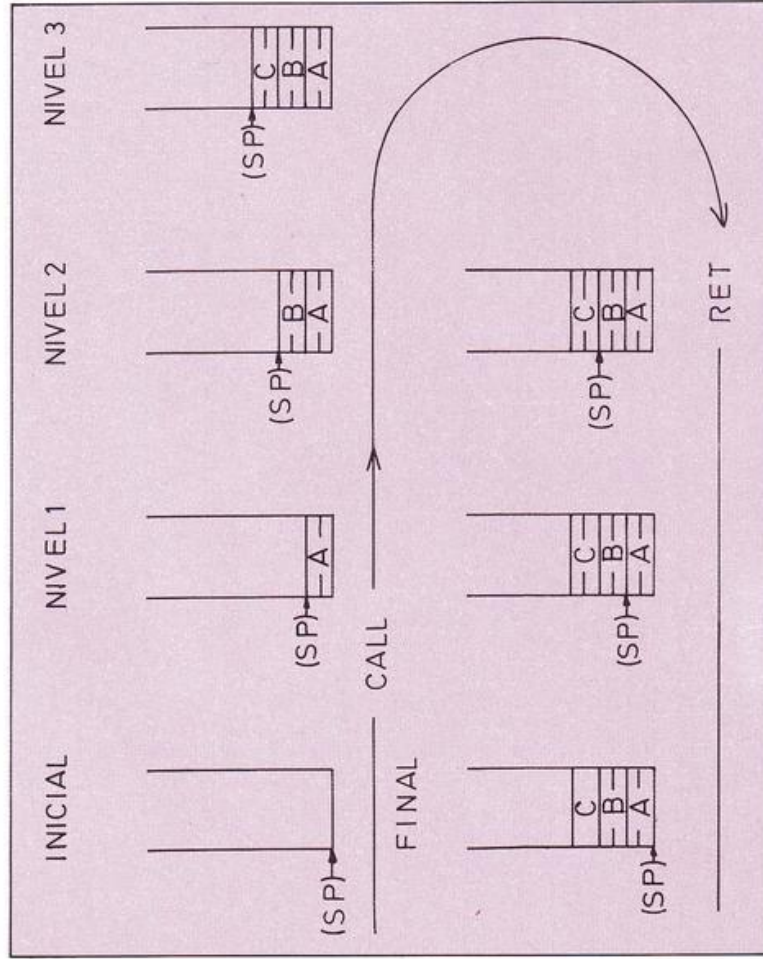


Fig. 11-3. Funcionamiento de la pila de máquina en subrutinas anidadas.

para que trabajen sobre las direcciones correctas. Este es el método empleado en el ensamblador «GENS-3» para que pueda ser cargado en cualquier dirección de memoria; por eso, la primera vez hay que entrar en él con un «USR» a la dirección de carga (para que actúe el reubicador) y la segunda, hay que entrar a la dirección de carga más 61.

Existen, también, instrucciones de llamada a subrutinas condicionales, es decir, que sólo se ejecutan si cierto indicador tiene determinado estado. De la misma forma, existen instrucciones de retorno, es decir, que devuelven el control a la CPU, estudiaremos a fondo todo lo relativo a las interrupciones que, curiosamente, parece ser uno de los temas más difíciles de entender de la programación. De momento, vamos a ver una a una, las instrucciones de llamada y retorno.

**CALL nn**

OBJETO:

Lo primero que hace el microprocesador, al igual que en todas las instrucciones, es incrementar el registro de

que la pila está vacía porque el registro «SP» apunta al comienzo, el contenido último de la pila permanece.

En un gran número de CPUs, existen dos pilas: la de máquina y la de usuario. La primera es la que utiliza el procesador cuando se ejecuta una llamada a subrutina y, la segunda, es la que utiliza el usuario para almacenar datos mediante «PUSH» y «POP». En el Z-80, el usuario comparte la pila con la máquina, de forma que ambas, pila de máquina y pila de usuario, son la misma. Esto tiene ventajas e inconvenientes. El principal inconveniente es que, cuando utilizemos la pila dentro de una subrutina, tendremos que preocuparnos por recuperar de la pila todos los datos que se hubieran metido antes de intentar retornar, ya que de lo contrario, el microprocesador confundiría uno de nuestros datos con la dirección de retorno. A este efecto se le denomina «corromper la pila».

Evidentemente, la ventaja es que podremos corromper la pila de forma intencionada para engañar al microprocesador y hacerle retornar a una dirección distinta de aquella desde donde se le llamó. Por ejemplo, supongamos que queremos que una subrutina retorne a su sitio correcto cuando, tras una operación determinada, el resultado no sea «0»; pero si es «0», queremos que retorne a la dirección E700h. La rutina podría terminar de la siguiente forma:

```

...      ...
...      JP  NZ,SIGUE
...      LD  BC,#E700
...      PUSH BC
...      RET
...      SIGUE

```

Otro truco interesante, que ya se mencionó en las instrucciones de salto, consiste en simular la instrucción «JP (DE)», que no existe, mediante la secuencia «PUSH DE» y «RET». Pero hay algo aún más útil: Podemos entrar en una subrutina habiendo fijado previamente cual será la dirección a donde queremos que retorne, por ejemplo, tenemos en «BC» la dirección de la subrutina y en «HL» la dirección a donde queremos que se produzca el retorno. Podemos hacer lo siguiente:

```

PUSH HL
PUSH BC
RET

```

No hemos inventado nada nuevo, este «truco» lo utiliza el Sistema Operativo para saltar a nuestras subrutinas cada vez que utilizamos la función «USR» con argumento numérico. En este caso, la dirección de retorno es 2D28h (11563). Todas nuestras rutinas son tratadas, por tanto, como subrutinas del Sistema Operativo.

En algunos casos, puede resultarnos útil saber que, cuando el S.O. entre en una de nuestras subrutinas (con USR), el registro «BC» contendrá, precisamente, la dirección de esta subrutina, por lo que una rutina nuestra puede saber en qué dirección está corriendo. Ya veremos cómo utilizar esto para escribir «reubicadores» de rutinas no reubicables.

Otro truco, delicado pero posible, es variar la secuencia de anidamiento. Supongamos una subrutina en la cual co-

nocemos su nivel de anidamiento y por el motivo que sea nos queremos saltar los retornos; pues una solución sería hacer tantas instrucciones «POP» como niveles se deseen saltar, menos una, y por último, ejecutar una instrucción RET. Ver Figura 11-4.

Para que el lector se dé cuenta de la importancia que tiene el conocimiento de las subrutinas, conviene saber que todo el Sistema Operativo del SPECTRUM está programado en base a ellas; lo que nos va a permitir, en muchos casos, utilizarlas dentro de nuestros programas para realizar tareas que ya están escritas en la ROM y ahorrar-nos el trabajo de volver a escribirlas.

Por otro lado, es muy recomendable que escribamos nuestros programas a base de subrutinas, cada una de las cuales probaremos por separado. De esta forma, si hay un error, será más fácil de detectar.

Un pequeño problema de las instrucciones «CALL» es que —al igual que «JP»— contienen como operando una dirección absoluta, por lo que las rutinas que contienen subrutinas no serán reubicables. A diferencia con los saltos, no existen «llamadas relativas», por lo que los «CALL» nunca serán reubicables.

No obstante, existe una forma de que un programa con subrutinas pueda correr en cualquier dirección de memoria; el sistema consiste en escribir un «reubicador» que localicemos delante del programa y que, mediante una tabla de direcciones relativas, modifique los operandos de todas las instrucciones «CALL»



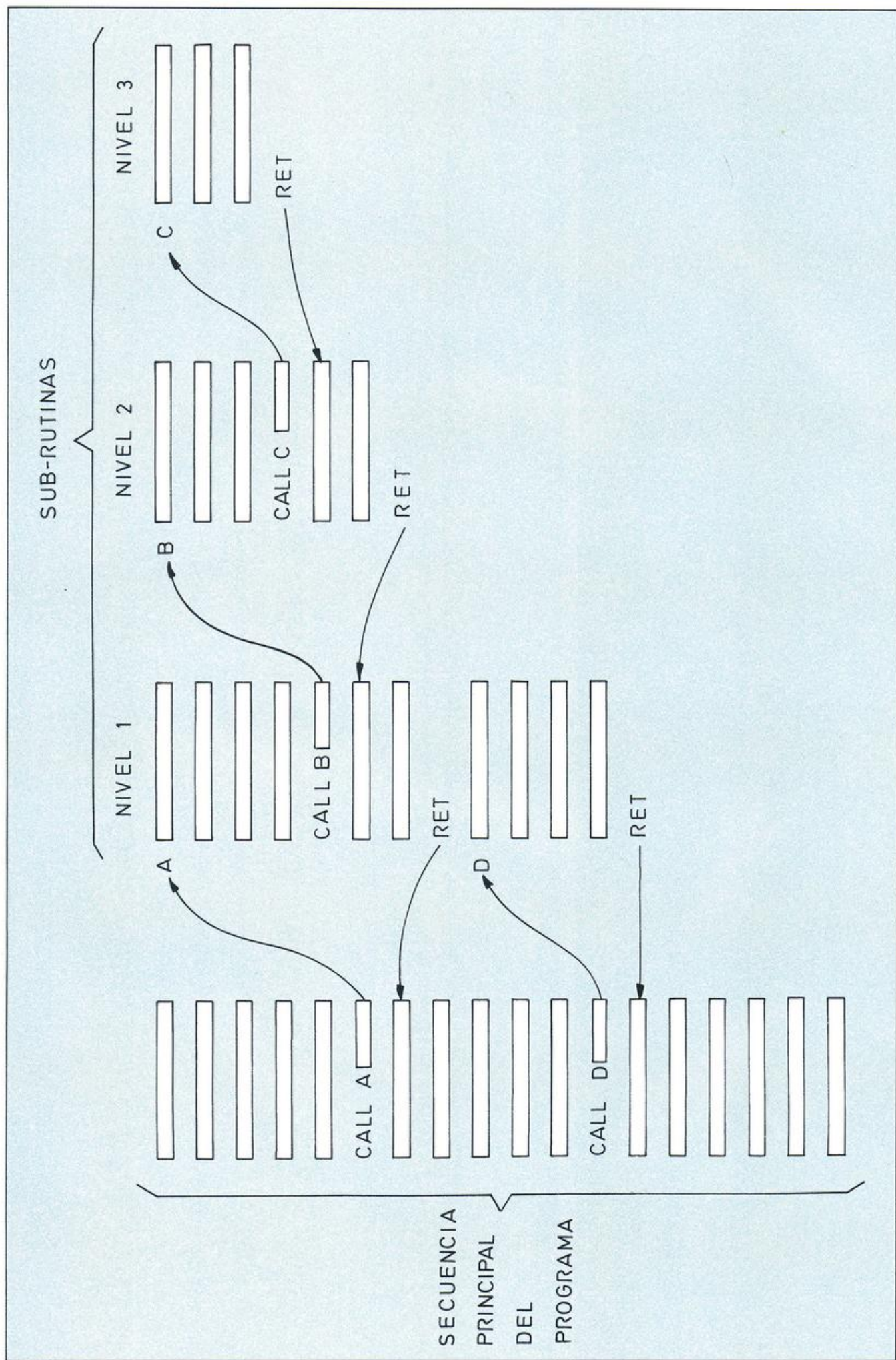


Fig. 11-2. Gráfico de sub-rutinas anidadas.



**CHRS\$ 18:** Scroll izquierda de atributos. Se produce un scroll del archivo de atributos, una columna a la izquierda. El scroll puede ser lineal o esférico dependiendo de como se hubiera fijado previamente. Inicialmente es lineal. En el caso de scroll lineal, lo que va entrando por el lado contrario son los atributos permanentes en curso.

**CHRS\$ 19:** Scroll derecha de atributos. Se produce un scroll del archivo de atributos, una columna a la derecha. El scroll se produce de la misma forma que en el caso anterior.

**CHRS\$ 20:** Scroll abajo de atributos. Se produce un scroll del archivo de atributos, una línea hacia abajo. El scroll se produce de la misma forma que en los dos casos anteriores.

**CHRS\$ 21:** Scroll arriba de atributos. Se produce un scroll del archivo de atributos, una línea hacia arriba. El scroll se produce de la misma forma que en los tres casos anteriores.

**CHRS\$ 22:** Control «AT». Tiene dos operandos que son los dos caracteres que le siguen. El primero indica la línea y el segundo la columna, donde deberá quedar posicionado el cursor. El siguiente carácter se imprimirá en esa posición. Trabaja de la misma forma que el «AT» del Basic, salvo que el primer operando no podrá ser mayor de 21, aun cuando se hubieran eliminado las dos líneas inferiores de la pantalla.

**CHRS\$ 23:** Fija letra cursiva. Pone a «1» el flag de cursiva en la variable «FLAGS» del programa (no en la del Sistema).

**CHRS\$ 31:** Código nulo. No produce ningún efecto.

**CHRS\$ 32 al CHRS\$ 255:** Im-

ma Operativo con el mismo nombre).

**CHRS\$ 24:** Fija letra negrita (bold). Pone a «1» el flag de bold en la variable «FLAGS» del programa.

**CHRS\$ 25:** Fija letra espejular (imagen de espejo). Pone a «1» el flag de espejo en la variable «FLAGS» del programa.

**CHRS\$ 26:** Fija letra normal. Pone a «0» los flags de cursiva, bold y espejo en la variable «FLAGS» del programa.

**CHRS\$ 27:** Fija scroll esférico. Pone a «1» el flag de scroll esférico en la variable «FLAGS» del programa.

**CHRS\$ 28:** Fija scroll lineal. Pone a «0» el flag de scroll esférico en la variable «FLAGS» del programa.

**CHRS\$ 29:** Transfiere la pantalla. Copia los archivos de pantalla y atributos en un bloque de 6912 bytes consecutivos, cuya primera dirección sea el contenido de la variable del Sistema «SEED». Al encender el ordenador, esta variable contiene «0», y cambia su valor cada vez que se hace uso de la función «RND» del Basic. Para almacenar el valor «n» en esta variable, basta con hacer: RANDOMIZE n donde «n» es un número entre «1» y «65535».

**CHRS\$ 30:** Recupera la pantalla. Copia, en los archivos de pantalla y atributos, el contenido de un bloque de 6912 bytes consecutivos, cuya primera dirección sea el contenido de la variable del Sistema «SEED».

**CHRS\$ 31:** Código nulo. No produce ningún efecto.

**CHRS\$ 32 al CHRS\$ 255:** Im-

prime el carácter que se corresponda con el código. La dirección del font de caracteres se toma de la variable del Sistema «CHARS». Es posible disponer un font de hasta 224 caracteres (el primero de los cuales deberá ser un espacio) en cualquier lugar de la memoria. Para direccionarlo, la variable del Sistema «CHARS» deberá contener un número que sea la dirección inicial del font, menos 256. La impresión se produce en letra normal, cursiva, negrita o espejo (imagen de espejo), dependiendo de cómo estén fijados los flags correspondientes de la variable «FLAGS» del programa (no de la del Sistema que tiene el mismo nombre). Es posible imprimir, simultáneamente, en cursiva y negrita; pero si está a «1» el flag de «espejo», no se tendrán en cuenta los contenidos de los flags de cursiva y bold. Tras la impresión, se actualiza, adecuadamente, la posición de impresión. Asimismo, se actualizan los contenidos de las variables del Sistema «S-POSN» y «DF-CC» para que la rutina sea compatible con el Basic; en caso necesario, se produce un scroll automático de una línea. La impresión se realiza de modo «transparente», es decir, no se modifica el byte correspondiente del archivo de atributos. No se reconocen, como tales, los códigos de gráficos y tokens.

El programa «PROPAN» utiliza dos variables internas. La primera de ellas se denomina «FLAGS», tiene un octeto y contiene los 8 flags que necesita el programa. La segunda se llama «VAR-1», tiene 2 octetos y se utiliza para almacenar

La siguiente instrucción a ejecutar será la que se encuentre en la dirección 7546h

## RET cc

OBJETO:

Si la condición «cc» es verdadera, carga en el registro contador de programa «PC» el último dato de la pila de máquina, poniendo el contenido de la dirección señalada por (SP) en el octeto inferior y la dirección señalada por (SP) + 1 en el octeto superior, después incrementa en 2 el registro «SP». Con esto efectúa la vuelta desde una subrutina a la instrucción siguiente a una «CALL» que realizó la entrada. Si la condición «cc» es falsa, ejecuta la siguiente instrucción. Los códigos nemotécnicos de «cc» y sus valores binarios se indican a continuación.

cc	binario	significado
NZ	000	no cero
Z	001	cero
NC	010	no acarreo
C	011	acarreo
PO	100	paridad impar o no desbordamiento
PE	101	paridad par o desbordamiento
P	110	positivo
M	111	negativo

CODIGO DE MAQUINA:

11111111	00000000
----------	----------

INDICADORES DE

CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h

00000000	00000000
01111111	00000000

Instrucción

RET Z: 11001000

Contenido del registro «SP» después de la ejecución

11111111	00000000
----------	----------

Contenido del registro «PC» después de la ejecución

Ninguno

CICLOS DE MEMORIA:

Si la condición «cc» es verdadera:

3

Si la condición «cc» es falsa:

1

CICLOS DE RELOJ:

Si la condición «cc» es verdadera:

11

Si la condición «cc» es falsa:

5

EJEMPLO:

RET Z

Indicador de condición Z = 1

Contenido del registro «SP»

11111111	00000000
----------	----------

Contenido de los octetos F704h y F705h



#### CICLOS DE MEMORIA:

4

#### CICLOS DE RELOJ:

14

#### EJEMPLO:

RET
-----

Contenido del registro «SP»

(SP):	11101101 01000110
-------	----------------------

Contenido de los octetos F746h y F747h

F746h:	11010100
F747h:	01101001

#### Instrucción

RET:	11101101 01001101
------	----------------------

Contenido del registro «SP» después de la ejecución

(SP):	11110111 01001000
-------	----------------------

Contenido del registro «PC» después de la ejecución

(PC):	01101001 11010100
-------	----------------------

La siguiente instrucción a ejecutar será la que se encuentra en la dirección 69D4h, es decir, la siguiente a aquella que se estaba ejecutando cuando se recibió la petición de interrupción.

RET

#### OBJETO:

Retorna desde una interrupción no enmascarable al punto por donde iba el programa cuando se produjo la petición de interrupción. Para ello, carga en el registro controlador de programa «PC» el último dato de la pila de máquinas, poniendo el contenido de la dirección señalada por (SP) en el octeto inferior y la dirección señalada por (SP)+1 en el octeto superior, después incrementa en 2 el registro «SP».

Además copia el estado del flip-flop IFF2 en el IFF1, con lo que éste recupera el valor que tenía antes de producirse la interrupción no enmascarable.

#### CODIGO DE MAQUINA:

EDh	11101101
45h	01000101

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

#### CICLOS DE MEMORIA:

4

#### CICLOS DE RELOJ:

14

#### EJEMPLO:

RET
-----

Contenido del registro «SP»

(SP):	11110111 01001000
-------	----------------------

Contenido de los octetos F748h y F749h

F748h:	00101101
F749h:	01011010

IFF1=0, IFF2=1

#### Instrucción

RET:	11101101 01000101
------	----------------------

Contenido del registro «SP» después de la ejecución

(SP):	11110111 01001010
-------	----------------------

IFF1=1 y IFF2=1, después de la ejecución

Contenido del registro «PC» después de la ejecución

(PC):	01010110 00101101
-------	----------------------

La siguiente instrucción a ejecutar será aquella que se encuentre en la dirección 562Dh, es decir, la siguiente a la que se estaba ejecutando cuando se recibió la petición de interrupción.

#### Observaciones:

Para entender y manejar mejor estas instrucciones es interesante que tenga en cuenta lo siguiente:

Las interrupciones son formas de trabajo que se utilizan principalmente para atender requerimientos que no admiten demora, resulta fundamental poder volver a la secuencia del programa que se estaba ejecutando una vez atendido ese requerimiento. Lo que garantizan las instrucciones RETI y RETN es la vuelta a la instrucción donde se interrumpió el programa. Las instrucciones RETI o RETN serán siempre la última

Por otro lado, los códigos de control van a permitirnos borrar la pantalla por trozos, intercambiar zonas, repetir «n» veces un carácter, mover el cursor, realizar «scroll» de pantalla o atributos tanto en modo esférico como en modo lineal e, incluso, transferir la pantalla completa a otra dirección de memoria o recuperación desde allí. Asimismo, se reconocen los códigos de «AT» y «ENTER» que funcionan de la misma manera que en Basic.

A continuación, se da una lista detallada de todos los códigos que utiliza «PRO-PAN»:

**CHRS 0:** Código nulo, no produce ningún efecto.

**CHRS 1:** Borra el primer tercio de pantalla. El borrado se produce poniendo a «0» los bits del archivo de pantalla y copiando los atributos permanentes en los bytes del archivo de atributos. No se restaura la posición de impresión.

**CHRS 2:** Borra el segundo tercio de pantalla. El borrado se produce de la misma forma que en el caso anterior.

**CHRS 3:** Borra el tercer tercio de pantalla. El borrado se produce de la misma forma que en los dos casos anteriores.

**CHRS 4:** Intercambia el primer tercio de la pantalla con el segundo. Se intercambian tanto los bytes del archivo de pantalla como los del de atributos. La posición de impresión no resulta afectada.

**CHRS 5:** Intercambia el segundo tercio de pantalla con el tercero. El intercambio se produce de la misma forma que en el caso anterior.

**CHRS 6:** Intercambia el primer tercio de pantalla con el tercero. El intercambio se produce de la misma forma que en los dos casos anteriores.

**CHRS 7:** Repetición de carácter. Utiliza dos operandos que serán los dos caracteres que le siguen; el primero de ellos indica las veces que se ha de repetir el carácter y el segundo es el carácter a repetir. Por ejemplo:

CHRS 7:CHRS 15:«a»;

provocaría la impresión de 15 asteriscos a partir de la posición actual de impresión. El segundo operando nunca puede ser «7», si lo fuera, la orden sería ignorada. Por ejemplo:

CHRS 7:CHRS 15:CHRS 7;

no provocaría ningún efecto. La posición de impresión se actualiza según el número de caracteres que se impriman. La rutina trabaja en modo «recurso».

**CHRS 8:** Cursor izquierda. Al igual que en Basic, retrocede una columna la posición de impresión. A diferencia del Basic, no va más allá de la columna «0».

**CHRS 9:** Cursor derecha. Avanza una columna la posición de impresión. No va más allá de la columna «31».

**CHRS 10:** Cursor abajo. Baja una línea la posición de impresión. No va más allá de la línea «21».

**CHRS 11:** Cursor arriba. Sube una línea de la posición de impresión. No va más allá de la línea «0».

**CHRS 12:** DELETE. Produce el borrado del carácter anterior a la actual posición de im-

presión. El borrado se produce retrocediendo el cursor, imprimiendo un espacio y volviendo a retroceder el cursor. Si la posición de impresión se encontrara en «0,0», se borraría el carácter situado en «0,0» quedando la posición de impresión, de nuevo, en «0,0». En cualquier otra posición de impresión, se borra el carácter anterior y la posición de impresión no varía.

**CHRS 13:** ENTER. Se avanza a la posición de impresión al inicio de la siguiente línea. Si se estuviera en la última línea, se produce un «scroll» automático de una línea. Hay que tener en cuenta que el Basic manda, automáticamente, este código al final de cualquier comando «PRINT» o «LPRINT» que no termine en «;».

**CHRS 14:** Scroll izquierda de pantalla. Se produce un «scroll» de la pantalla, un pixel a la izquierda. El scroll puede ser «lineal» o «esférico» dependiendo de como se hubiera fijado previamente. Al cargar el programa, queda preparado para «scroll lineal».

**CHRS 15:** Scroll derecha de la pantalla. Se produce un scroll de la pantalla, un pixel a la derecha. El scroll se produce de la misma forma que en el caso anterior.

**CHRS 16:** Scroll abajo de pantalla. Se produce un scroll de la pantalla, un scan hacia abajo. El scroll se produce de la misma forma que en los dos casos anteriores.

**CHRS 17:** Scroll arriba de pantalla. Se produce un scroll de la pantalla, un scan hacia arriba. El scroll se produce de la misma forma que en los tres casos anteriores.



ma en código máquina requiere un planteamiento bastante más cuidadoso que si se hiciera en Basic. Es importante definir, con toda claridad, qué es lo que se quiere hacer y cómo va a hacerse. Por pequeño que sea el programa, es necesario escribir un gran número de líneas y las posibilidades de error aumentan enormemente. Por todo esto, es recomendable que todo programa en código máquina esté compuesto por una colección de rutinas que serán llamadas desde el bucle principal para realizar cada una de las funciones elementales. Estas rutinas pueden escribirse y probarse, previamente, por separado. De esta forma, será más sencillo detectar y corregir los posibles errores.

A lo largo del curso, hemos ido viendo un gran número de rutinas sueltas. Algunas de ellas, trabajaban sobre la pantalla y, entre éstas, algunas realizaban una función u otra dependiendo del número que contuviera el acumulador al entrar en ellas (borrado parcial, intercambio de zonas de pantalla, etc.). También vimos lo que era un canal de salida; una rutina que hacía algo con el número que contuviera el acumulador al entrar en ella (por ejemplo, imprimir el carácter cuyo código fuera ese número).

Ahora, vamos a agrupar todas estas rutinas de gestión de pantalla, y unas cuantas más, para crear un auténtico «Procesador de Pantalla» que trabaje como un canal de salida. Es decir, en nuestro programa se entrará con un número en el acumulador y, dependiendo de este número, el programa realizará una u otra

REINICIOS DE PAGINA CERO		
Código Fuente	Hexadecimal	Decimal
RST #00	C7	199
RST #08	CF	207
RST #10	D7	215
RST #18	DF	223
RST #20	E7	231
RST #28	EF	239
RST #30	F7	247
RST #38	FF	255

Fig. 11-6. Tabla de codificación para los reinicios de página cero.

función sobre la pantalla. Si almacenamos la dirección de inicio de nuestro programa en la tabla de canales, podremos utilizarlo desde el Basic, como si se tratara de uno de los canales del Sistema Operativo. La utilidad de este programa es que va a permitirnos un gran número de efectos sobre la pantalla, tanto si programamos en Basic como si lo hacemos en código máquina.

El programa —al que, por razones evidentes, hemos denominado «PROPAN»— utilizará 30 códigos de control (del «1» al «30») que realizan distintas funciones. Los códigos «0» y «31» son ignorados y no realizan ninguna función, pero cabe la posibilidad de que el lector amplíe el programa utilizando estos códigos para realizar otras funciones que él incorpore. Finalmente, los códigos del «32» en adelante (hasta el «255») producen la impresión de caracteres. Si se

instrucción de las rutinas que atiendan las interrupciones o de la rutina que trata la interrupción no enmascararse respectivamente, si no fuera así, no se podría trabajar con esta técnica.

Los flip-flops IFF1 e IFF2 son, desde el punto de vista software, dos flags que controlan cuando se permiten o no las interrupciones. IFF1 cuando vale 1 las permite y cuando vale 0, IFF2 es el almacenamiento temporal del valor de IFF1.

Las interrupciones no enmascarables no están afectadas por el valor de IFF1 o IFF2, siempre se permiten.

En el programa monitor del SPECTRUM está anulada la interrupción no enmascarable. Si se recibiera una petición de este tipo, la rutina de servicio se limita a retornar a la secuencia principal. El objeto de esta medida es facilitar la protección de software comercial, ya que un programador experto, podría entrar en un programa protegido, provocando, desde fuera, una interrupción no enmascarable y saltando a una rutina que le devolviera el control.

Por otro lado, la ULA genera, cada 20 milisegundos, una petición de interrupción enmascarable que será atendida o no en función del estado del flip-flop IFF1. En el capítulo que trata de las instrucciones de control de la CPU, veremos detalladamente cómo funcionan las interrupciones y cómo utilizarlas en nuestros programas.

#### Reinicios de página cero

En todo programa, y especialmente en un Sistema Ope-

rativo, existen una serie de rutinas a las que hay que llamar frecuentemente desde distintos puntos del programa —imprimir un carácter, realizar un cálculo, leer un carácter de memoria, etc.—. Esto obligaría a utilizar un gran número de instrucciones «CALL» dirigidas al mismo punto desde varios lugares del programa. Cada instrucción «CALL» ocupa tres bytes, por tanto, el gasto de memoria resulta considerable.

Para evitar este desperdicio de memoria, el Z80 incorpora una instrucción que permite llamar a subrutinas localizadas en lugares fijos de la memoria. Cada una de estas llamadas sólo ocupa un byte, de forma que se reduce considerablemente el gasto de memoria. Esta instrucción tiene el nemónico «RST», abreviatura del Inglés «Restart» (Reinicio).

Las rutinas llamadas por esta instrucción han de estar situadas en las primeras direcciones de memoria, por ello, esta instrucción se denomina «Reinicio de página cero».

Todos los «Restart» del Spectrum se dirigen a rutinas útiles de la ROM que podremos utilizar en nuestros programas. De hecho, ya hemos utilizado una de ellas. ¿Recuerda el lector la misteriosa instrucción «RST #08» que nos detenía el programa con un informe de error? Pues bien, existen un total de 8 reinicios de página cero y cada uno de ellos tiene su utilidad. En principio, vamos a ver la instrucción y, luego, estudiaremos cada una de las rutinas a las que podemos llamar.

## RST P

### OBJETO:

Lo primero que hace el micro-procesador, al igual que en todas las instrucciones, es incrementar el registro contador de programa «PC» en el número de octetos que tiene la instrucción, en este caso uno. Después guarda el valor del registro «PC» en la pila de máquina poniendo el contenido del octeto superior en la dirección señalada por (SP)-2. A continuación decrementa en 2 el registro «SP». Finalmente carga en el registro «PC» la dirección de página cero indicada por el operando «P». La correspondencia para codificar la instrucción es como sigue:

«P»	«P»
00h	000
08h	001
10h	010
18h	011
20h	100
28h	101
30h	110
38h	111

con lo que comenzará a ejecutar las instrucciones que se encuentren en dichas direcciones.

### CODIGO DE MAQUINA:

11 ← 1 → 111

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

3



## CICLOS DE RELOJ:

11

### EJEMPLO:

RST #18

Dirección del primer octeto de la instrucción

0 1 0 1 0 0 1 0  
1 0 0 0 0 1 1 1

Contenido del registro «SP»

(SP): 1 1 1 1 0 0 0 0  
0 0 1 1 1 1 0 0

### Instrucción

RST #18: 1 1 0 1 1 1 1 1

Contenido del registro «SP» después de la ejecución

(SP): 1 1 1 1 0 0 0 0  
0 0 1 1 1 0 1 0

Contenido de los octetos F03Ah y F03B después de la ejecución

F03Ah: 1 0 0 0 0 1 0 0  
F03B: 0 1 0 1 0 0 1 0

Contenido del registro «PC» después de la ejecución

(PC): 0 0 0 0 0 0 0 0  
0 0 0 1 1 0 0 0

La siguiente instrucción a ejecutar está en la dirección 0018h

El programa monitor del SPECTRUM tiene las siguientes rutinas de página cero programadas:

Dirección 0000h

- 19 K Invalid colour
- 20 L BREAK into program
- 21 M RAMTOP no good
- 22 N Statement lost
- 23 O Invalid stream
- 24 P FN without DEF
- 25 Q Parameter error
- 26 R Tape loading error
- 27 "
- 28 Icl 1982 Sinclair Research Ltd

Por ejemplo, la secuencia para detener un programa de forma que aparezca, en pantalla, el mensaje «R Tape loading error», sería:

RST #8  
DEFB 26

Cuando expliquemos el uso del Microdrive en código máquina, veremos que es posible utilizar esta instrucción, con otros literales, para acceder a las funciones del INTERFACE 1. De momento, en la versión básica del Spectrum, si utilizamos otros literales, obtendremos informes sin sentido.

Dirección 0010h

PRINT-A-1  
Imprime el carácter o código de control cuyo valor en ASCII se pondrá previamente en el registro acumulador «A». Antes de ello, es necesario abrir un canal de comunicación, lo cual se consigue llamando a la rutina «CHAN-OPEN» de la ROM (dirección: 1601h) con el número de control en el acumulador. En resumen, se puede decir que ésta es la rutina general de acceso a todos los canales de salida.

Dirección 0018h  
GET-CHAR  
Comprueba si el carácter

de la posición de memoria direccionada por la variable CH-ADD es imprimible, en caso afirmativo retorna; en otro caso incrementa la variable CH-ADD hasta que encuentre un carácter imprimible o el código «ENTER».

Dirección 0020h

NEXT-CHAR  
Realiza la misma operación que la rutina anterior (GET-CHAR) comenzando por el carácter siguiente al direccionado por la variable CH-ADD.

Dirección 0028h

FP-CALC  
Esta es la entrada de la rutina del calculador. Dada la complejidad de esta rutina, le dedicaremos un capítulo aparte. No obstante, anticiparemos que las operaciones a realizar se indican, también, mediante literales. Cuando aprendamos a manejar esta rutina, tendremos a nuestra disposición toda la potencia de cálculo del Basic para usarla desde código máquina.

Dirección 0030h

BC-SPACES  
Deja un número de posiciones libres en el área de trabajo desplazando el resto de la memoria hacia arriba. El número de posiciones viene dado por el contenido del par de registros «BC» al entrar en la rutina.

Dirección 0038h

MASK-INT  
A esta posición salta el micro procesador cuando se produce una interrupción enmascarable (en MODO 1), lo cual ocurre en el SPECTRUM cada 20 milisegundos.

Esta rutina se utiliza para actualizar el reloj de tiempo real y para leer el teclado. El

## INSTRUCCIONES DE LLAMADA Y RETORNO

Código Fuente	Hexadecimal	Decimal
CALL nn	CD,n,n	205,n,n
CALL C,nn	DC,n,n	220,n,n
CALL NC,nn	D4,n,n	212,n,n
CALL Z,nn	CC,n,n	204,n,n
CALL NZ,nn	C4,n,n	196,n,n
CALL PE,nn	EC,n,n	236,n,n
CALL PO,nn	E4,n,n	228,n,n
CALL M,nn	FC,n,n	252,n,n
CALL P,nn	F4,n,n	244,n,n
RET	C9	201
RET C	D8	216
RET NC	D0	208
RET Z	C8	200
RET NZ	C0	192
RET PE	E8	232
RET PO	E0	224
RET M	F8	248
RET P	F0	240
RETI	ED,4D	237,77
RETN	ED,45	237,69

Fig. 11-5. Tabla de codificación para las instrucciones de llamada y retorno.

código ASCII del carácter, digito o signo pulsado lo devuelve esta rutina en la variable «LAST-K». También es posible llamarla desde nuestros programas para leer el teclado.

### Ejemplos

Terminado este capítulo, sólo nos restan por estudiar las instrucciones de entrada/salida y las de control de CPU. No obstante, estamos ya en disposición de escribir un auténtico programa en código máquina.

La realización de un programa

### Tablas de codificación

En la Figura 11-5 está la tabla de codificación para las instrucciones de llamada y re-



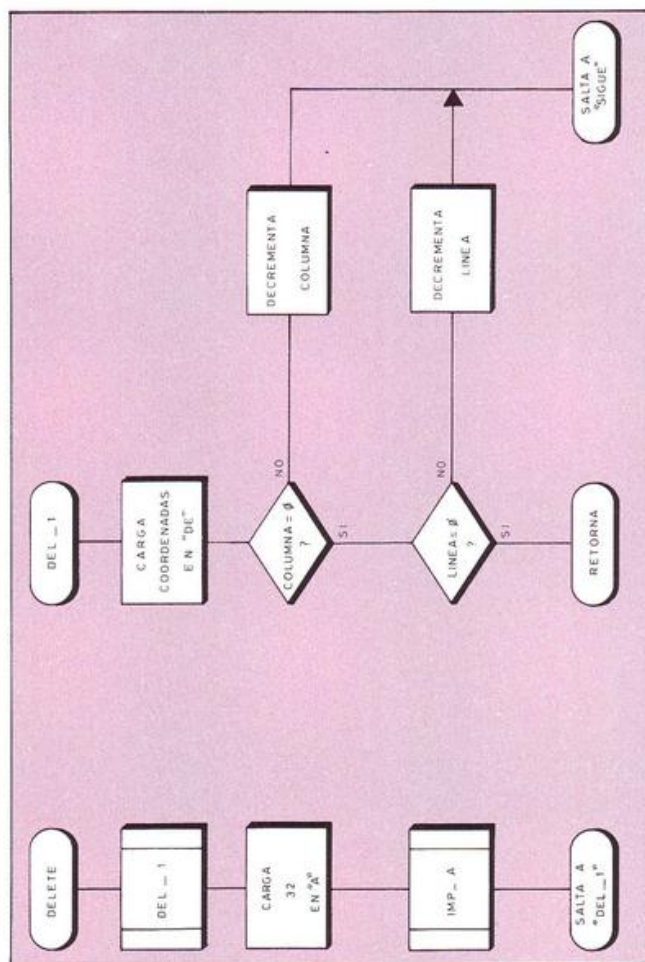


Fig. 11-12. Ordinograma de la rutina «DELETE».

carga, en «A», el número de columna, se retorna si es «0»; si no, se decrementa y se salta a «SIGUE» donde serán actualizadas las variables «SPOSN» y «DF-CC» del Sistema.

Líneas 1210 a la 1250: En este caso, se trata de incrementar el número de columna. En la línea 1230 se retorna si este ya era «31». También se sale saltando a «SIGUE».

Líneas 1260 a la 1300: Incrementa el número de línea. Se retorna si este ya era «21» antes de incrementarlo. Si se quiere que el código «AT» pueda posicionar el cursor en las dos líneas inferiores de la pantalla, basta con cambiar el «21» de la línea 1270 por un «23».

Líneas 1310 a la 1350: Al contrario de la anterior, esta

vez se trata de subir una línea. Se produce el retorno si el número de línea ya era «0».

DELETE:

Es la rutina de respuesta al código «12». Su misión es borrar el carácter anterior a la posición actual de impresión y dejar ésta en el lugar del carácter borrado. El listado es el siguiente:

1350	DELETE	CALL	DEL_1
1370		LD	A,32
1380		CALL	IMP_A
1390	DEL_1	LD	DE, (S POSNA
1400		SBC	HL, DE 821
1420		EX	DE, HL
1430		LD	A, E
1440		AND	A, 2, DEL_2
1450		JR	DEL_2
1460		DEC	E
1470		JC	SIGUE
1480	DEL_2	LD	A, D
1490		AND	A
1500		RET	Z
1510		DEC	D
1520		LD	E, 31
1530		JF	SIGUE

La subrutina «DEL-1» se encarga de retroceder la posición de impresión. El funcionamiento de «DELETE» consiste en: Primero: se llama a la rutina «DEL-1». Segundo: se imprime un espacio llamando a «IMP-A» con el código «32» en el acumulador. Tercero: se continúa en «DEL-1». De esta forma, la subrutina «DEL-1» se ejecuta dos veces, una antes del borrado y otra después. Vamos a estudiarla.

Linéas 1390 a la 1420: De igual forma que cuando movíamos los cursores, lo primero que hay que hacer es tener en «DE» las coordenadas en curso.

Líneas 1430 a la 1450: Si el número de columna es «0», habrá que colocarse en el último carácter de la línea anterior, para lo que se salta a «DEL-2».

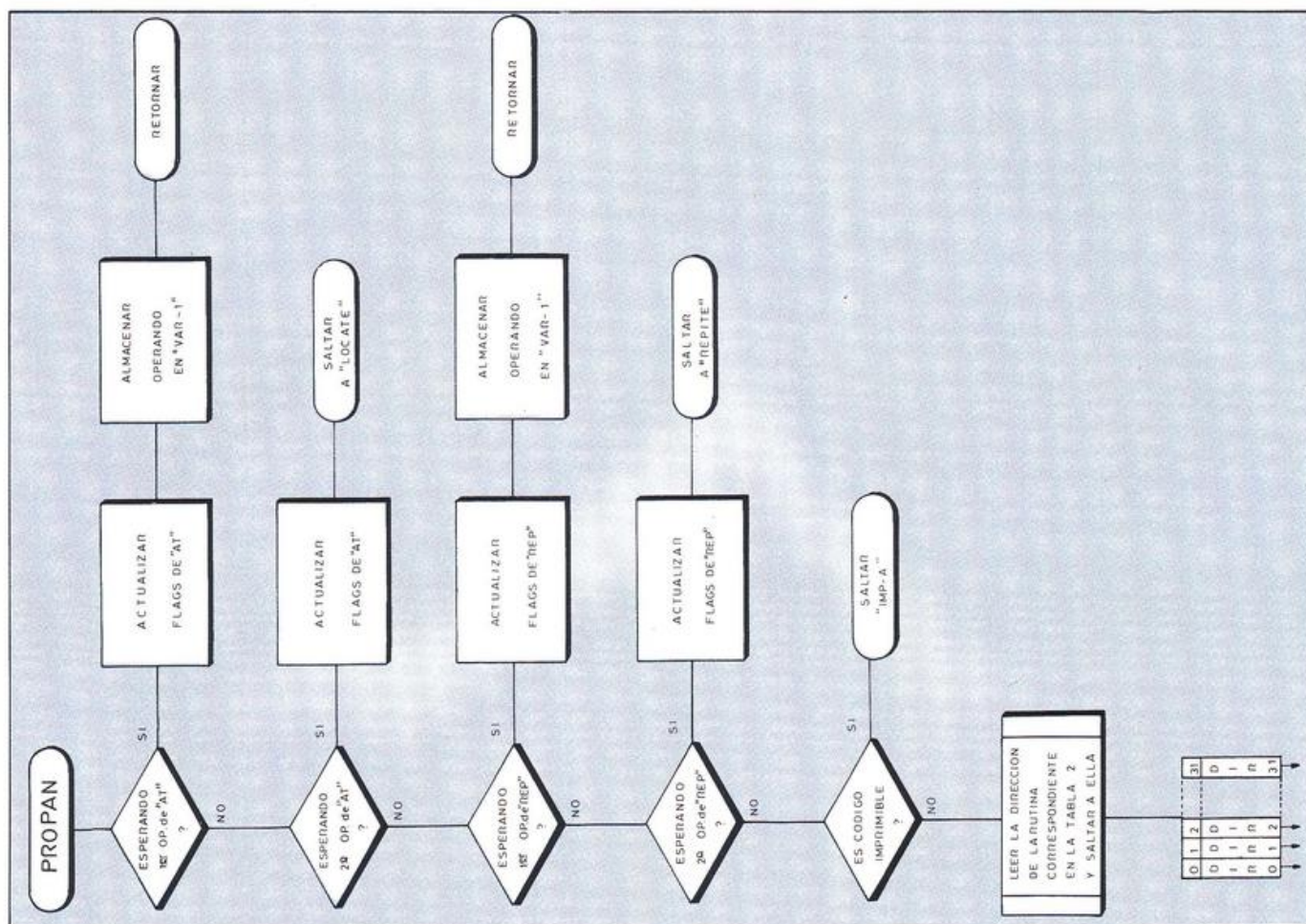


Fig. 11-7. Diagrama de flujo de la rutina de entrada al procesador de pantalla.



namiento temporal de algunos datos.  
La disposición y significación de los «FLAGS» son los siguientes:

**BIT 0:** Si está a «1», indica que la impresión se debe hacer con letra cursiva.

**BIT 1:** Si está a «1», indica que la impresión se debe hacer con letra negrita o bold.

**BIT 2:** Si está a «1», indica que la impresión se debe hacer con letra especular (imagen de espejo).

**BIT 3:** Si está a «1», indica que cualquier scroll deberá hacerse en modo esférico, es decir, lo que «sale» por un lado de la pantalla, «entra» por el contrario. Si está a «0», cualquier scroll será lineal, es decir, lo que sale por un lado de la pantalla, se pierde definitivamente y por el lado contrario entran ceros, en el caso de scroll de pantalla, o los atributos permanentes en cursor, en el caso de scroll de atributos.

**BIT 4:** Procesar primer operando de «AT». Si está a «1», indica que el carácter que se está procesando es el primer operando de un código «AT» (CHR\$ 22) previamente recibido.

**BIT 5:** Procesar segundo operando de «AT». Si está a «1», indica que el carácter que se está procesando es el segundo operando de un código «AT» (CHR\$ 22) previamente recibido.

**BIT 6:** Procesar primer operando de «REP». Si está a «1», indica que el carácter que se está procesando es el primer operando de un código de repetición (CHR\$ 7) previamente recibido. Este código se to-

maría como el número de repeticiones.

**BIT 7:** Procesar segundo operando de «REP». Si está a «1», indica que el carácter que se está procesando es el segundo operando de un código de repetición (CHR\$ 7) previamente recibido. Este código se tomaría como el del carácter a repetir.

Ahora ya tenemos claramente planteado lo que queremos hacer. Vamos a ver cómo hacerlo. Cada una de las distintas funciones que realizará «PROPAN», será llevada a cabo por una rutina; si bien, en ocasiones una misma rutina podrá realizar varias funciones similares. Por ejemplo, tendremos una rutina que nos borrará un tercio de la pantalla, pero será la misma rutina la que borre el primer tercio, el segundo o el tercero.

Por tanto, el programa deberá empezar con una rutina de entrada que se encargue de llamar a una u otra de las restantes rutinas, dependiendo de qué código contenga el acumulador. Esta rutina de entrada deberá tener en cuenta, también, si el carácter que entra es un operando de «AT» o de «CHR\$ 7» para tratarlo como tal; con este fin, se consultan los flags correspondientes.

El diagrama de flujo de esta rutina de entrada, está representado en la Figura 11-7. Como se ve, primero comprobamos los flags, uno a uno, para ver si se trata de un operando. Luego, vemos si el carácter es imprimible, en cuyo caso saltamos a la rutina «IMP-A». Si el código no corresponde a un carácter imprimible, miramos en una tabla, qué rutina tenemos que lla-

mar para que ejecute la función correspondiente.

El listado en Assembler de esta rutina de entrada es el siguiente:

```

100 PROPAN LD HL, FLAGS
110 BIT 4, (HL)
120 JR NZ, AT_1
130 BIT 5, (HL)
140 JR NZ, AT_2
150 BIT 6, (HL)
160 JR NZ, REP_1
170 BIT 7, (HL)
180 JR NZ, REP_2
190 CP 32
200 JP NC, IMP_A
210 LD C, A
220 ADD A, A
230 LD E, A
240 LD D, 0
250 LD HL, TABLA2
260 ADD HL, DE
270 LD E, (HL)
280 INC HL
290 LD D, (HL)
300 EX DE, HL
310 LD A, C
320 JP (HL)
330 AT_1 LD (VAR_1), A
340 RES 4, (HL)
350 SET 5, (HL)
360 RET
370 AT_2 RES 5, (HL)
380 JP LOCATE
390 REP_1 LD (VAR_1), A
400 RES 6, (HL)
410 SET 7, (HL)
420 RET
430 REP_2 RES 7, (HL)
440 JP REPITE
450 FLAGS DEFB 0
460 VAR_1 DEFB 0
470 TABLA2 DEFW NULO
480 DEFW NULO
490 DEFW CLS3
500 DEFW CLS3
510 DEFW CLS3
520 DEFW INTER
530 DEFW INTER
540 DEFW INTER
550 DEFW REP_0
560 DEFW CURSOR
570 DEFW CURSOR
580 DEFW CURSOR
590 DEFW DELETE
600 DEFW ENTER
610 DEFW ENTER
620 DEFW SCRF1
630 DEFW SCRFD
640 DEFW SCRFB
650 DEFW SCRFB
660 DEFW SCRA1
670 DEFW SCRAD
680 DEFW SCRA1
690 DEFW SCRA1
700 DEFW AT_0
710 DEFW SETFLA
720 DEFW SETFLA
730 DEFW SETFLA
740 DEFW SETFLA
750 DEFW SETFLA
760 DEFW SETFLA
770 DEFW TRAFAN
780 DEFW TRAFAN
790 DEFW NULO
800 NULO RET

```

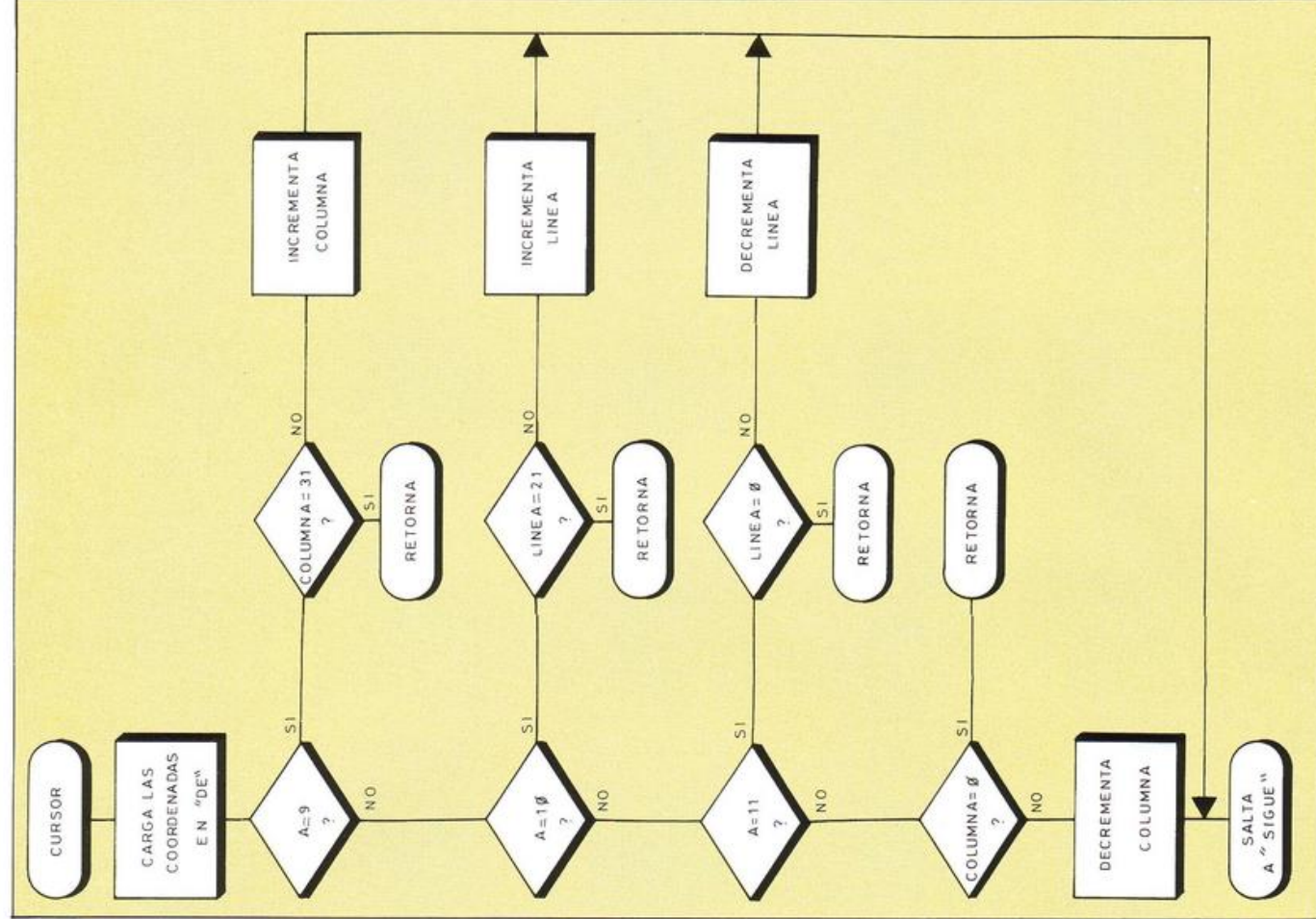


Fig. 11-11. Ordinograma de la rutina «CURSOR».



ta «SIGUE» de la rutina «IMP-A». En este punto, se entra con las coordenadas de una posición de impresión en el registro «DE» y la rutina «SIGUE» se encargará de actualizar las variables del Sistema «S-POSN» y «DF-CC» para que ésta sea la nueva posición de impresión.

#### REPITE:

En esta rutina se entra con el acumulador conteniendo el código del carácter o función a repetir y la variable «VAR-1» conteniendo el número de repeticiones. El listado Assembler es el siguiente:

```

890 REPITE CP 7
900 RET Z
910 LD HL,VAR_1
920 PUSH BC
930 REP1
940 PUSH AF
950 CALL PROFAN
960 POP AF
970 POP BC
980 DNZ REP1
990 RET

```

Hemos utilizado un procedimiento de programación denominado «recursivo» que consiste en hacer que una rutina se llame a sí misma. En este caso, la rutina llama a «PROPAN» tantas veces como tenga que repetir el carácter o función. Un procedimiento similar usa el Sistema Operativo del Spectrum para imprimir los «Tokens» expandidos.

Cuando se utilizan recursivos, es importante evitar que la rutina se encierre en un bucle sin fin, donde estaría llamándose a sí misma continuamente y expandiendo, indefinidamente, la pila. En este caso, lo hemos evitado haciendo que la rutina retorne si el código a repetir es «7». Vamos a ver lo que hace cada línea:

Líneas 890 y 900: Se retorna si el código a repetir es «7».

Líneas 910 y 920: Se transfiere al registro «B» el número de veces que habrá de repetirse el código que está en «A».

Líneas 930 a 980: Constituyen un bucle donde se va llamando a «PROPAN», con el código a repetir en «A», tantas veces como indique el contenido de «B». Antes de cada llamada, se preservan los contenidos de «AF» y «BC» que se recuperan después.

Línea 990: Retorna al Sistema.

#### REP-0:

Es la rutina de respuesta al código «7». Lo único que hace es poner a «1» el bit 6 de «FLAGS» para indicar al programa que el siguiente código será el primer operando. Su listado es el siguiente:

```

1000 REP_0 LD HL,FLAGS
1010 SET 6,(HL)
1020 RET

```

La rutina es tan sencilla que no creemos que requiera explicación.

#### AT-0:

Similar a la anterior, se trata de la rutina de respuesta al código «AT». En este caso, se pone a «1» el bit 4 de «FLAGS». El listado, tan sencillo como el anterior, es este:

```

1070 AT_0 LD HL,FLAGS
1080 SET 4,(HL)
1090 RET

```

#### CURSOR:

Se trata de la rutina que se encarga de mover la posición

de impresión. Se entra en ella con el acumulador conteniendo cualquiera de los códigos «8», «9», «10» u «11». La rutina ejecutará el movimiento en uno u otro sentido en función de este código. Veamos el listado:

```

1050 CURSOR LD DE,(S-POSN)
1060 LD SBC HL,41821
1070 EX DE,HL
1080 CP 9
1090 JR Z,CUR_2
1100 CP 10
1110 JR Z,CUR_3
1120 CP 11
1130 JR Z,CUR_3
1140 CP 11
1150 JR Z,CUR_4
1160 LD A,E
1170 AND A
1180 RET Z
1190 DEC E
1200 JP SIGUE
1210 CUR_2 LD A,E
1220 CP 21
1230 RET Z
1240 INC E
1250 JP SIGUE
1260 CUR_3 LD A,D
1270 CP 21
1280 RET Z
1290 INC D
1300 JP SIGUE
1310 CUR_4 LD A,D
1320 AND A
1330 RET Z
1340 DEC D
1350 JP SIGUE

```

Líneas 1060 a la 1090: Empezamos por cargar en «DE» las coordenadas en curso desde la variable del Sistema «S-POSN». Como están invertidas, tendremos que restarlas en 1821h para obtener el n.º de línea en «D» y el de columna en «E».

Líneas 1100 y 1110: Saltamos a «CUR-2» si el código en «A» es «9».

Líneas 1120 y 1130: Saltamos a «CUR-3» si el código es «10».

Líneas 1140 y 1150: Saltamos a «CUR-4» si el código en «A» es «11».

Líneas 1160 a la 1200: A estas líneas se llega si el código es «8» y se encargan de retroceder una columna. Se

Tenga en cuenta que, en esta rutina, entramos con el registro «A» (acumulador) conteniendo un código que puede ser imprimible, de control o un operando de «AT» o «CHR\$ 7». Vamos a ir comentando las líneas una a una:

Línea 100: Cargamos, en «HL», la dirección de «FLAGS» con el fin de facilitar su lectura.

Líneas 110 y 120: Saltamos a «AT-1» (línea 330) si el carácter en proceso es el primer operando de un código «AT» previamente recibido.

Líneas 130 y 140: Saltamos a «AT-2» (línea 370) si el carácter recibido es el segundo operando de un código «AT».

Líneas 150 y 160: Saltamos a «REP-1» (línea 390) si el carácter es el primer operando de un «CHR\$ 7» previamente recibido.

Líneas 170 y 180: Saltamos a «REP-2» (línea 430) si el carácter es el segundo operando de un «CHR\$ 7».

Líneas 190 y 200: Llegado este punto, ya sabemos que no se trata de ningún operando. En estas líneas, saltamos a «IMP-A» si el código es mayor de 31, es decir, si representa un carácter imprimible. Utilizamos un salto absoluto ya que «IMP-A» se encuentra fuera del rango permitido por los saltos relativos.

A partir de la línea 210, ya sabemos que el código no es imprimible, es decir, que se trata de un código de control. Lo que haremos será utilizarlo como un «offset» para entrar en la «TABLA2» y salir de ella con «HL» conteniendo la dirección a donde tenemos que saltar. Vamos a verlo detenidamente:

Línea 210: Preservamos en «C» el contenido de «A».

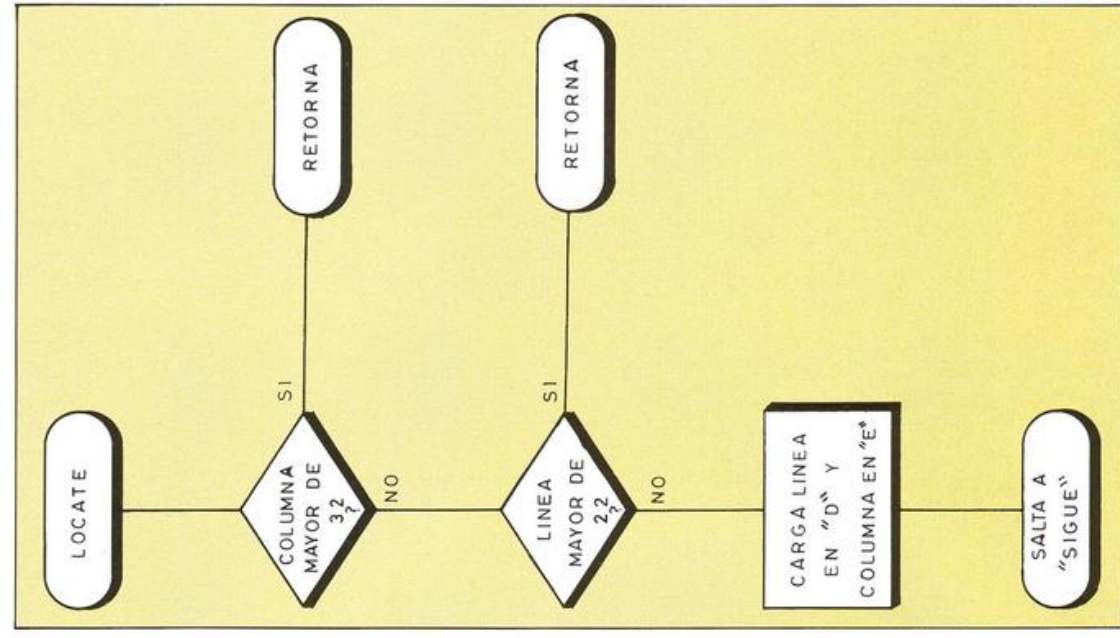


Fig. 11-8. Ordinograma de la rutina «LOCATE».

Línea 220: Multiplicamos el contenido de «A» por 2, ya que cada elemento de la tabla ocupa 2 bytes.

Líneas 230 y 240: Traspasamos, a «DE», el contenido de «A».

Línea 250: Cargamos en «HL» la dirección base de la

«TABLA2».

Línea 260: Sumamos, sobre «HL», el contenido de «DE». Con esto, el registro «HL» queda apuntando a un elemento de la tabla que contendrá el código que hubiera en «A». Este elemento será la dirección inicial de la



rutina que se encarga de reutilizar la función correspondiente a este código.

Líneas 270 a la 290: Cargamos en «DE» el elemento correspondiente de la tabla.

Línea 300: Pasamos este dato a «HL».

Línea 310: Recuperamos el primitivo valor de «A» que habíamos conservado en «C».

Línea 320: Saltamos a la rutina correspondiente que se va a encargar de realizar la función indicada por «A». En cualquiera de estas rutinas, entraremos, de nuevo, con el registro «A» conteniendo el código de la función a realizar. Para algunas rutinas, este código será útil; otras, simplemente lo ignorarán.

La «TABLA2» está colocada a partir de la línea 480. Observe que el primer y último elementos (correspondientes a los códigos «0» y «31») apuntan a la etiqueta «NULO» de la línea 800. Cuando se reciben estos códigos, se saltará a esta línea y se ejecutará un simple retorno. Si el lector quiere añadir funciones al procesador de pantalla, puede utilizar estos códigos cambiando las etiquetas «NULO» de la «TABLA2» por otras que indiquen el inicio de las rutinas correspondientes. Recuerde que, al ensambalar el programa, cada etiqueta de la tabla será sustituida por su significado, en este caso, por la dirección inicial de la rutina correspondiente.

Nos hemos dejado por ver las cuatro pequeñas rutinas colocadas a partir de la línea 330. Estas son las rutinas que se encargan de procesar los operandos de los códigos «AT» y «CHRS 7». Vamos a verlas:

Línea 330 a la 360: Se almacena, en «VAR-1», el contenido de «A» que es el primer operando de un «AT». Se pone a cero el flag de «primer operando de AT» y a uno el de «segundo operando de AT». De esta forma, el programa sabe que el siguiente código a recibir será el segundo operando de «AT». Finalmente, se retorna.

Líneas 370 y 380: se pone a cero el flag de «segundo operando de AT». En este punto, tendremos en «VAR-1» el primer operando (n.º de línea) y en «A» el segundo (n.º de columna); por tanto, no tenemos más que saltar a la rutina «LOCATE» que será la que se encargue de posicionar el cursor.

Líneas 390 a la 440: Exactamente igual que en «AT-1», almacenamos el primer operando en «VAR-1» y actualizamos los flags para que el programa interprete el siguiente código a recibir como segundo operando de «CHRS 7». Finalmente, retornamos.

Líneas 430 y 440: De la misma forma que en «AT-2», ponemos a cero el flag correspondiente y saltamos a la rutina «REPITE» que se encargará de repetir el carácter o código que esté en «A», tantas veces como indique el número contenido en «VAR-1». Si el contenido de «A» representa un código imprimible, se imprimirá las veces necesarias, si se trata de un código de control, se ejecutará la función correspondiente tantas veces como indique «VAR-1». Por último, si «A» contiene un «7», la orden será ignorada ya que, de ejecutarse, podría hacernos entrar en un bucle sin fin. De todo esto se encargará la rutina

«REPITE» cuyo listado veremos más adelante.

Líneas 450 a la 470: Hemos reservado tres bytes antes de la «TABLA2» para que sean las variables de nuestro programa. Observe que, a diferencia del Basic, en un programa escrito en Assembler, las variables pueden estar en cualquier lugar de la memoria, pero deberán tener un sitio fijo y será necesario declararlas. Precisamente, es lo que hacemos en estas líneas, inicializando sus contenidos a «0».

Ya hemos visto la rutina inicial de nuestro programa. A partir de ahora, iremos viendo, una a una, las rutinas que esta llama para realizar cada una de las funciones.

## LOCATE:

Es la rutina de respuesta al código «AT». Coloca el cursor en la línea indicada por «VAR-1» y la columna indicada por el contenido de «A». Su listado Assembler es el siguiente:

```

810 LOCATE CP 32
820 RET NC
830 LD E,A
840 LD A,(VAR_1)
850 CP 22
860 RET NC
870 LD D,A
880 JF SIGUE

```

Líneas 810 y 820: Retorna si el contenido de «A» (n.º de columna) es mayor de 31.

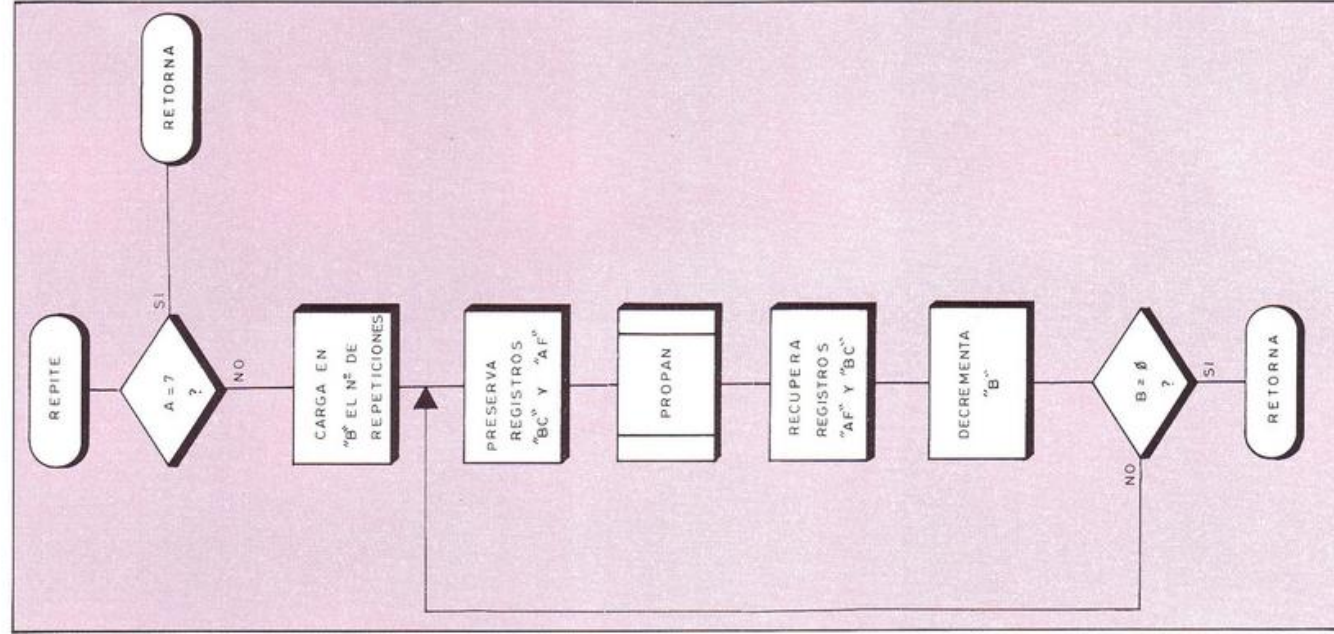
Línea 830: Carga el número de columna en «E».

Línea 840: Carga en «A» el número de línea.

Líneas 850 y 860: Retorna si el contenido de «A» (n.º de línea) es mayor de 21.

Línea 870: Carga el número de línea en «D».

Línea 880: Salta a la etiqueta



Ordinograma de la rutina «REPITE».

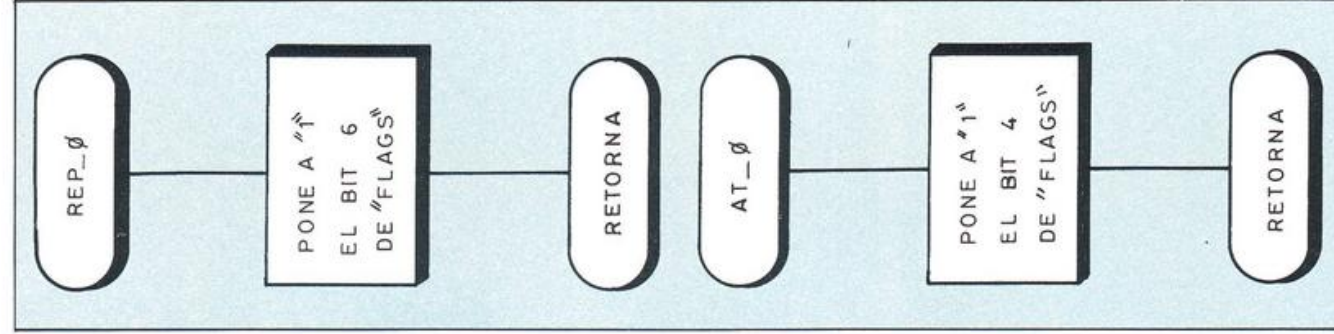


Fig. 11-10. Ordinograma de las rutinas.



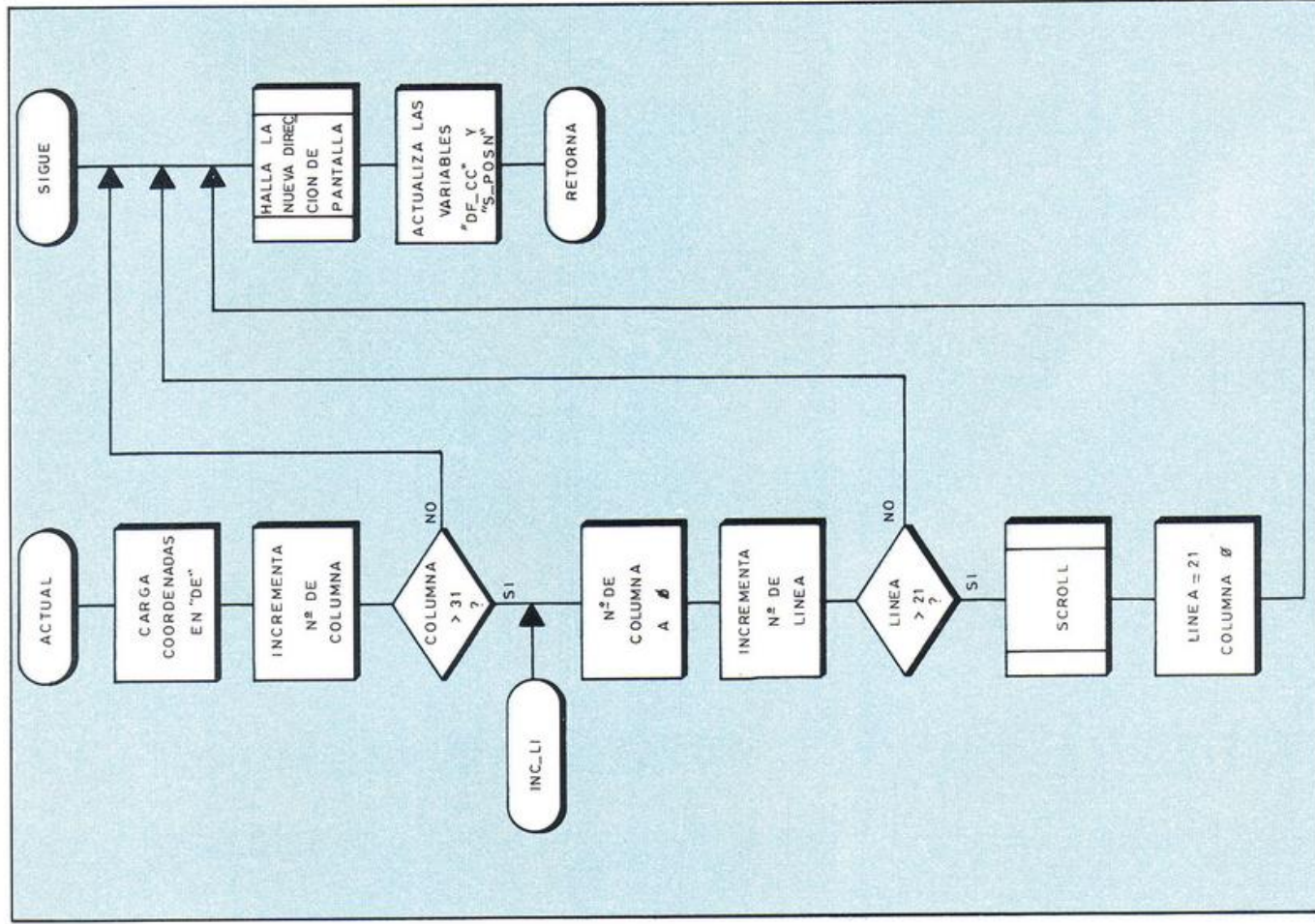


Fig. 11-18b. Ordinograma de las rutinas «ACTUAL», «INC-LI» y «SIGUE».

Líneas 1460 a la 1470: Simplemente, se decrementa el número de línea y se salta a «SIGUE» para actualizar las variables «S-POSN» y «DF-CC».

Líneas 1480 a la 1500: Se carga en «A» el número de línea y se retorna si este es «0».

Líneas 1510 a la 1530: Se decrementa el número de línea y se pone «31» en el de columna, con lo que estamos al final de la línea anterior. Como de costumbre, terminamos saltando a «SIGUE» para actualizar las variables.

#### CLS3:

Esta es la primera de las rutinas que ya habíamos visto. Su misión es borrar un tercio de la pantalla. En ella se entra con el acumulador conteniendo «1», «2» ó «3» para borrar el primero, segundo o tercer tercio respectivamente. Hemos simplificado bastante el listado con el uso de instrucciones «LDIR» y la eliminación de «ceros» innecesarios en la tabla:

1540	CLS3	DEC A
1550	ADD A,A	
1560	LD HL,TABLA	
1570	LD B,0	
1580	LD C,A	
1590	ADC HL,BC	
1600	LD B,(HL)	
1610	HL	
1620	LD D,(HL)	
1630	LD E,0	
1640	JR CLS3_1	
1650	TABLA	
1660	DEFB #40, #59	
1670	DEFB #48, #59	
1680	DEFB #50, #5A	
1690	CLS3_1	
1700	LD H,B	
1710	LD L,0	
1720	PUSH DE	
1730	LD (HL),L	
1740	LD D,H	
1750	LD E,1	
1760	LDIR	
1770	POP HL	
1780	LD D,H	
1790	LD E,1	
1800	LD A,(255*3)	
1810	LD (HL),A	
1820	LD C,#FF	
1830	LDIR	
1840	RET	

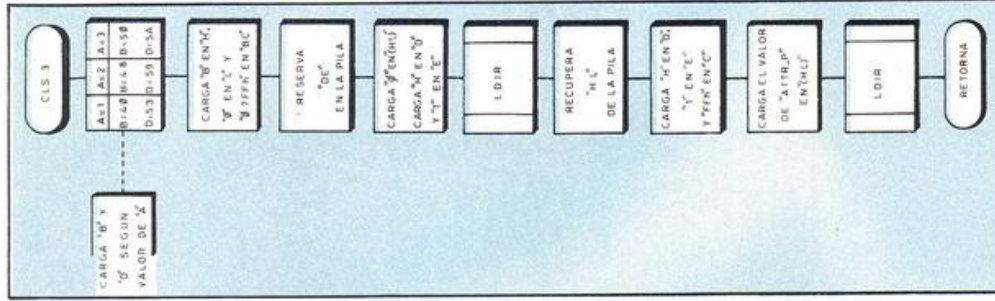


Fig. 11-13. Ordinograma de la rutina «CLS3».

Líneas 1540 a la 1630: Cada elemento de la tabla consta de dos bytes, el primero es el octeto alto de la dirección de inicio de un bloque en el archivo de pantalla; el segundo es lo mismo, pero para el archivo de atributos. En ambos casos, los octetos bajos por eso, los omitimos de la tabla.

bla. En estas líneas, cargamos el inicio del bloque de pantalla en «BC» y el del bloque de atributos en «DE».

Línea 1640: Continuamos en «CLS3-1» para saltarnos los bytes ocupados por la tabla.

Líneas 1650 a la 1670: La tabla de direcciones para cada uno de los bloques.

Líneas 1680 y 1690: Transferimos a «HL» la dirección de inicio del bloque en el archivo de pantalla.

Líneas 1700 y 1710: Cargamos en «BC» la longitud del bloque menos uno y preservamos «DE» que contiene la dirección de inicio del bloque en atributos.

Línea 1720: Cargamos un «0» en la primera dirección del bloque. Utilizamos «LD (HL), L» porque «L» vale «0».

Líneas 1730 y 1740: Copiamos en «DE» la dirección de inicio del bloque más uno.

Línea 1750: Copiamos el «0» del primer byte en todos los restantes del bloque.

Líneas 1760 a la 1780: Recuperamos en «HL» la dirección de inicio del bloque en el archivo de atributos y pasamos a «DE» esta dirección más uno.

Líneas 1790 y 1800: Cargamos en «A» los atributos permanentes en curso y los transferimos a la primera dirección del bloque.

Línea 1810: Como «B» ya contiene «0» (así ha salido del «LDIR» anterior), el hecho de cargar FFh en «C» hace que «BC» contenga «255», es decir, la longitud del bloque menos uno.

Línea 1820: Copiamos el primer byte en todos los restantes.

Línea 1830: Retorna al Sistema.

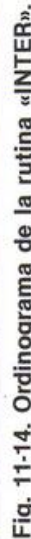


Esta rutina también la habíamos visto, aunque la habíamos hecho un poco más corta convirtiendo en subrutina un grupo de instrucciones que se repetían dos veces. Su listado es:

1840	INTER	EXX
1850	PUSH	HL
1860	EXX	S
1870	CF	Z,DF_2
1880	JR	6
1890	CF	Z,OF_3
1900	JR	HL,#5800
1910	LD	DE,#5900
1920	LD	HL,#4000
1930	EXX	DE,#4800
1940	LD	TRANS
1950	JR	TRANS
1960	LD	HL,#5000
1970	DF_2	DE,#5A00
1980	LD	DE,#5A00
1990	EXX	HL,#4000
2000	LD	DE,#5000
2010	JR	TRANS
2020	DF_3	HL,#5B00
2030	LD	DE,#5A00
2040	EXX	HL,#4000
2050	LD	DE,#5000
2060	LD	BC,204B
2070	TRANS	CALL INTE_1
2080	BUC_1	DEC BC
2090	LD	A,B
2100	LD	C
2110	JR	NZ,BUC_1
2120	POP	HL
2130	EXX	B,0
2140	LD	INTE_1
2150	BUC_2	DINZ BUC_2
2160	LD	A,(HL)
2170	INTF_1	AF,AF'
2180	RET	A,(DE)
2190	LD	(HL),A
2200	EX	AF,AF'
2210	LD	(DE),A
2220	INC	HL
2230	INC	DE
2240	RET	
2250		
2260		
2270		
2280		

Líneas 1870 a la 1900: En función de que el contenido de «A» sea «4», «5» ó «6», se sigue o se salta a «OP-2» u «OP-3».

Líneas 1910 a la 1960: Car-  
gamos en «HL'» y «DE'» las di-  
recciones del primer y segun-



correspondientes a los mismos bloques, pero, en el archivo de atributos. Saltamos

Líneas 2140 y 2150: Se recupera el valor de «HL» que se había preservado anteriormente y se intercambian los bloques.





Líneas 3070 a la 3100: En las 8 iteraciones del bucle, se van sacando, uno a uno, los bits del registro «A» por la derecha y se van introduciendo, también por la derecha, en la posición apuntada por «HL», con lo que el byte queda invertido.

Líneas 3110 a la 3130: Se incrementan los punteros y se cierra el bucle.

Líneas 3140 a la 3170: Se cargan en «DE» las coordenadas actuales.

Línea 3180: Se incrementa el número de columna.

Líneas 3190 a la 3210: Si no se ha llegado a la columna 32, se salta a «SIGUE».

Líneas 3220 y 3230: Se carga «0» como número de columna y se incrementa el número de línea.

Líneas 3240 a la 3260: Si no se ha llegado al final de la pantalla, se salta a «SIGUE».

Líneas 3270 y 3280: Se llama a la rutina «SCROLL» de la ROM para subir toda la pantalla una línea hacia arriba y se fija como coordenada la primera columna de la última línea.

Línea 3290: Se preservan las coordenadas.

Líneas 3300 a la 3400: Se calcula la dirección del archivo de pantalla correspondiente a estas coordenadas.

Línea 3410: Se almacena la nueva dirección de pantalla en la variable del Sistema «DFCC».

Línea 3420: Se recuperan las coordenadas.

Líneas 3430 a la 3450: Se almacenan las nuevas coordenadas en la variable del Sistema «S-POSN». Recuerde que las coordenadas deben ir invertidas, para lo cual, se las resta de 1821h.

Línea 3460: Se retorna al Sistema.

Líneas 3470 a la 3500: Se define el valor de las variables y etiquetas utilizadas en la rutina y que no estén definidas ya en ella.

SCRPI:  
(SCROLL de Pantalla a la izquierda). Esta es la primera de una colección de rutinas que se encargan de realizar los scroll de pantalla y atributos. En este caso se trata de desplazar toda la pantalla un pixel a la izquierda. Los pixels que se «escapan» por la izquierda, entrarán por la derecha si está a «1» el flag de «scroll esférico». De lo contrario, se perderán definitivamente y, por la derecha, entrarán «ceros». El listado de la rutina es el siguiente:

```

3510 SCRP1 LD HL, 23527
3520 LD C, 192
3530 LD B, 2
3540 AND A
3550 RL
3560 DJNZ B_1
3570 JR NC, NOCA_1
3580 LD A, (FLAGS)
3590 BIT 3, A
3600 JR 2, NOCA_1
3610 LD (VAR_1), HL
3620 LD IX, (VAR_1)
3630 SET 0, (IX+32)
3640 NOCA_1 DEC C
3650 JR NZ, B_2
3660 RET
3670

```

La rutina es bastante similar a las vistas en el Prólogo de este CURSO. Veamos la misión de cada línea:

Línea 3510: Se inicializa el puntero «HL» para contener la última dirección en el archivo de pantalla.

Línea 3520: Se inicializa «C» para contener el número de scans en pantalla. Este registro actuará como contador en un bucle que se repetirá para cada scan.

Línea 3530: Dentro de este bucle, hay otro que se encarga de desplazar cada scan. El

registro «B» actuará como contador y se inicializa a 32 en esta línea (para los 32 bytes de un scan).

Líneas 3540 a la 3570: Se pone a «0» el indicador de acarreo (para que no entre «basura» por la derecha) y se rota, a la izquierda, todo el contenido de un byte, entra por la izquierda del byte de su izquierda, la transferencia se realiza a través del indicador de acarreo.

Línea 3580: Al final del bucle, habrá acarreo si el pixel de más a la izquierda estaba a «1»; si no, se salta a «NOCA\_1» (línea 3650).

Líneas 3590 a la 3610: También se salta a «NOCA\_1» si el flag de «scroll esférico» está a «0», es decir, si el scroll ha de ser lineal.

Líneas 3620 a la 3640: Se pone a «1» el pixel de más a la derecha del scan.

Líneas 3650 y 3660: Decrementa el contador y cierra el bucle para pasar al siguiente scan hasta que se traten todos.

Línea 3670: Retorna al sistema.

SCRPD:  
(SCROLL de Pantalla a la Derecha). De funcionamiento similar a la anterior, rota el archivo de pantalla un pixel a la derecha. Veamos su listado:

```

3680 SCRPD LD HL, 16384
3690 LD C, 192
3700 LD B, 32
3710 AND A
3720 RR
3730 INC HL
3740 DJNZ B_3
3750 JR NC, NOCA_2
3760 LD A, (FLAGS)
3770 BIT 3, A
3780 JR 2, NOCA_2
3790 LD (VAR_1), HL
3800 LD IX, (VAR_1)
3810 SET 7, (IX-32)
3820 NOCA_2 DEC C
3830 JR NZ, B_4
3840 RET

```

registros para tener en «HL» y «DE» las direcciones correspondientes al archivo de atributos.

Línea 2160: Se carga un «0» en «B» para fijar un bucle con 256 iteraciones.

Líneas 2170 y 2180: Constituyen un bucle que intercambia los contenidos de cada byte de los dos bloques. De nuevo, los intercambios se realizan llamando a la rutina «INTE-1».

Línea 2190: Retorna al Sistema.

Líneas 2200 a la 2250: Intercambian los contenidos de la dirección apuntada por «HL» y la apuntada por «DE». El intercambio se produce sobre los registros «A» y «A'». Líneas 2260 a la 2280: incrementan los punteros y retornan.

#### SETFLA:

Es la rutina que se encarga de fijar los flags de letra cursiva, bold, especular y de scroll esférico, en respuesta a los códigos «23» al «28». Su listado es:

```

2290 SETFLA LD HL, FLAGS
2300 CP 23
2310 JR NZ, SET_1
2320 SET 0, (HL)
2330 RET
2340 SET_1 CP 24
2350 JR NZ, SET_2
2360 SET 1, (HL)
2370 RET
2380 SET_2 CP 25
2390 JR NZ, SET_3
2400 SET 2, (HL)
2410 RET
2420 SET_3 CP 27
2430 JR NZ, SET_4
2440 SET 3, (HL)
2450 RET
2460 SET_4 CP 28
2470 JR NZ, SET_5
2480 RES 3, (HL)
2490 RET
2500 SET_5 LD A, #F3
2510 AND (HL)
2520 LD (HL), A
2530 RET

```

Línea 2290: Carga en «HL» la dirección de «FLAGS».

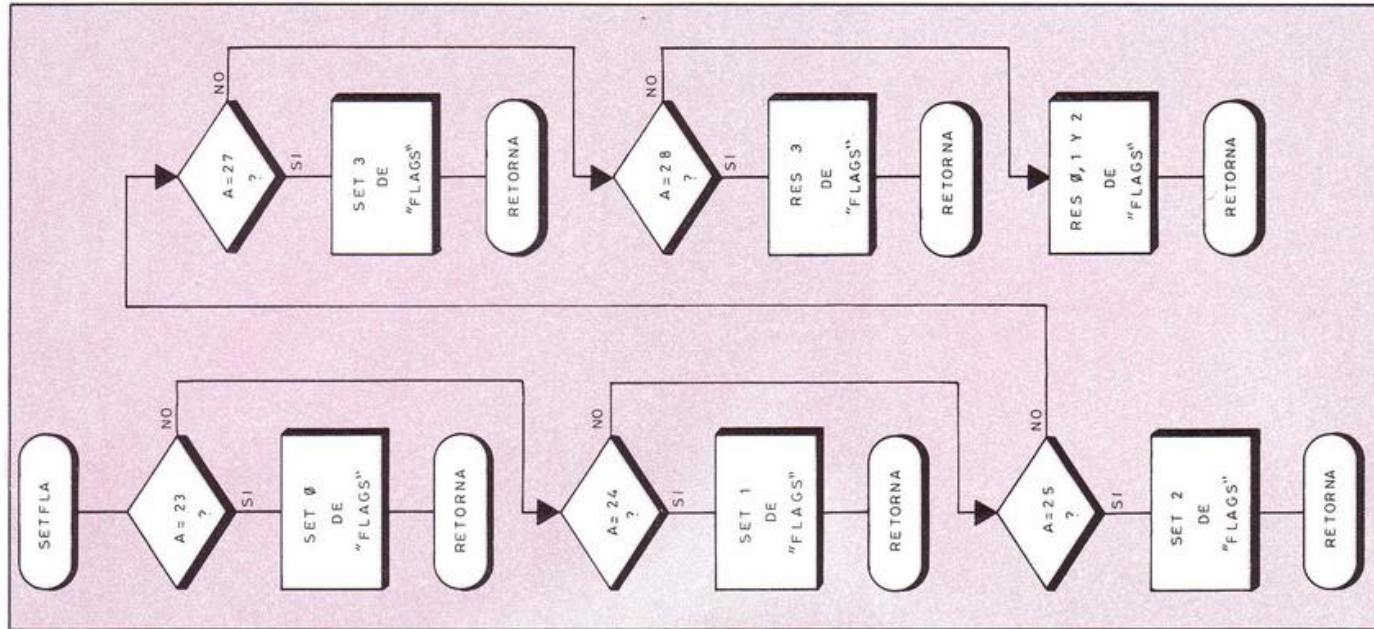


Fig. 11-15. Ordinograma de la rutina «SETFLA».



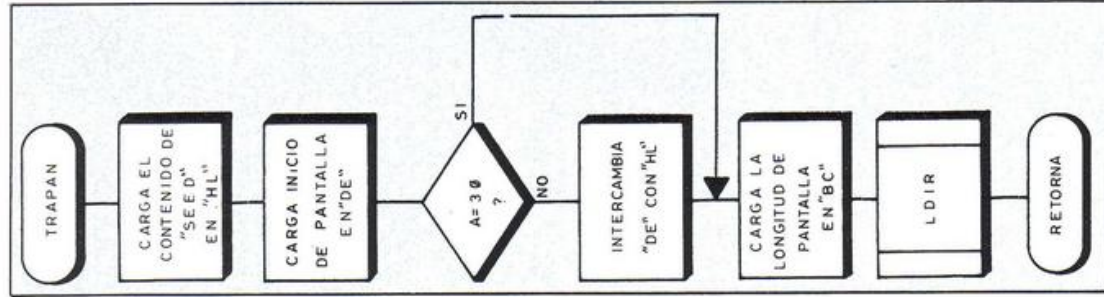


Fig. 11-16. Ordinograma de la rutina «TRAPAN».

Líneas 2300 y 2310: Si el código no es «23», salta a «SET-1».

Líneas 2320 y 2330: Pone a «1» el flag de «cursiva» y retorna al Sistema.

Líneas 2340 y 2350: Si el

código no es «24», salta a «SET-2».

Líneas 2360 y 2370: Pone a «1» el flag de «bold» (negrita) y retorna al Sistema.

Líneas 2400 y 2410: Pone a «1» el flag de «espejo» y retorna al Sistema.

Líneas 2420 y 2430: Si el código no es «27», salta a «SET-4».

Líneas 2440 y 2450: Pone a «1» el flag de «scroll esférico» y retorna al Sistema.

Líneas 2460 y 2470: Si el código no es «28», salta a «SET-5».

Líneas 2480 y 2490: Pone a «0» el flag de «scroll esférico» y retorna al Sistema.

Líneas 2500 a la 2530: Pone a «0» los flags de «cursiva», «bold» y «espejo», y retorna al Sistema. Como la puenta a «0» se hace mediante un «AND» con F8h, los restantes flags no resultan afectados.

#### TRAPAN:

Es la rutina que se encarga de transferir y recuperar la pantalla, a y desde una zona de memoria apuntada por el contenido de la variable del Sistema «SEED» en respuesta a los códigos «29» y «30».

La misma rutina se encarga de transferir y recuperar. Según el contenido de «A», se intercambian o no las direcciones de origen y destino. El listado es como sigue:

2540	TRAPAN	LD	HL, (SEED)
2550		LD	DE, #4000
2560		CP	30
2570		JR	Z, RECU
2580		EX	DE, HL
2590	RECU	LD	BC, 5912
2600		LDIR	
2610		RET	
2620	SEED	EDU	23670

Líneas 2540 y 2550: Cargan en «HL» el contenido de la variable «SEED» y en «DE» la di-

rección de inicio del archivo de pantalla. Con estos valores, la rutina queda preparada para recuperar una pantalla.

Líneas 2560 y 2570: Si el código es «30», salta a «RECU».

Línea 2580: Intercambia origen y destino de forma que la rutina quede preparada para transferir una pantalla.

Líneas 2590 y 2600: Carga en «BC» la longitud de pantalla más atributos y realiza la transferencia en uno u otro sentido.

Línea 2610: Retorna al Sistema.

Línea 2620: La variable «SEED» se encuentra en las direcciones 23670 y 23671. Hay que decirse al Ensamblador, porque él no lo sabe.

#### ENTER:

Es la rutina de respuesta al código «13». Se encarga de saltar al inicio de la línea siguiente. Su listado es:

2630	ENTER	LD	DE, (S_FOSN)
2640		LD	HL, #1821
2650		SBC	HL, DE
2660		EX	DE, HL
2670		JR	INC_L1

Se cargan en «DE» las coordenadas en curso y se salta a «INC-L1» dentro de la rutina «IMP-A». El salto puede ser relativo porque la rutina «IMP-A» viene a continuación.

#### IMP-A:

Esta es la rutina general para imprimir un carácter cuyo código se encuentre en «A». Asimismo, partes de esta rutina se utilizan desde otras, por ejemplo, la rutina «ENTER» utiliza a partir de «INC-L1» y todas las rutinas

que deben actualizar la posición de impresión tienen su salida a través de «SIGUE». El listado completo de «IMP-A» es el siguiente:

2680	IMP_A	LD	DE, (CHARS)
2690		LD	H, 0
2700		LD	L, A
2710		ADD	HL, HL
2720		ADD	HL, HL
2730		ADD	HL, HL
2740		ADD	HL, DE
2750		EX	DE, HL
2760		LD	HL, (DF_CC)
2770		LD	B, B
2780		LD	A, (FLAGS)
2790		BIT	2, A
2800		JR	NZ, IMPR_2
2810	BUCLE1	LD	A, (DE)
2820		LD	(HL), A
2830		LD	A, (FLAGS)
2840		AND	1
2850		JR	Z, NOCURS
2860		SRL	(HL)
2870		LD	A, B
2880		CF	5
2890		JR	NC, NOCURS
2900		SLA	(HL)
2910		CF	3
2920		JR	NC, NOCURS
2930		SLA	(HL)
2940		LD	A, (FLAGS)
2950		AND	2
2960		JR	Z, NOBOLD
2970		LD	A, (HL)
2980		SRL	A
2990		OR	(HL)
3000		LD	(HL), A
3010	NOBOLD	INC	DE
3020		INC	H
3030		DJNZ	BUCLE1
3040		JR	ACTUAL
3050	IMPR_2	LD	A, (DE)
3060		LD	C, B
3070	BUCLE2	RR	A
3080		RL	(HL)
3090		DEC	C
3100		JR	NZ, BUCLE2
3110		INC	DE
3120		INC	H
3130		DJNZ	IMPR_2
3140	ACTUAL	LD	DE, (S_FOSN)
3150		LD	HL, #1821
3160		SBC	HL, DE
3170		EX	DE, HL
3180		INC	E
3190		LD	A, E
3200		CP	32
3210		JR	C, SIGUE
3220		LD	E, 0
3230		INC	D
3240		LD	A, D
3250		CP	21
3260		JR	C, SIGUE
3270		CALL	SCROLL
3280		LD	DE, #1400
3290	SIGUE	PUSH	DE
3300		LD	A, D
3310		AND	#07
3320		RRC	A
3330		RRC	A
3340		RRC	A
3350		OR	E
3360		LD	E, A
3370		LD	A, D

Líneas 2940 a la 2960: Se salta a «NOBOLD» si no se ha seleccionado impresión en «negrita».

Líneas 2970 a la 3000: Se imprime un «OR» del byte con el mismo desplazado a la derecha para conseguir el efecto de aumentar el grosor en las líneas verticales.

Líneas 3010 a la 3030: Se incrementan los punteros y se cierra el bucle.

Línea 3040: La rutina continúa a partir de «ACTUAL» para actualizar la posición de impresión.

Líneas 3050 y 3060: Se carga en «A» un scan del carácter y se prepara el registro «C» para que sea el contador de un bucle con 8 iteraciones.

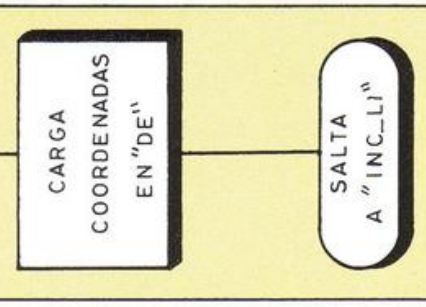


Fig. 11-17. Ordinograma de la rutina «ENTER».

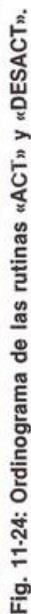
to de aumentar el grosor en las líneas verticales.

Líneas 3010 a la 3030: Se incrementan los punteros y se cierra el bucle.

Línea 3040: La rutina continúa a partir de «ACTUAL» para actualizar la posición de impresión.

Líneas 3050 y 3060: Se carga en «A» un scan del carácter y se prepara el registro «C» para que sea el contador de un bucle con 8 iteraciones.





ACT	LD	HL, (CHANS)
5370	LD	DE, 15
5380	LD	DE, 15
5390	ADD	HL, DE
5400	LD	E, (HL)
5410	INC	HL
5420	LD	D, (HL)
5430	LD	(NMT), DE
5440	LD	DE, DE, PROFAN
5450	LD	(HL), D
5460	DEC	HL
5470	LD	(HL), E

Copyright HISOFT 1983  
CURSO CM MICROHDBY





El funcionamiento es idéntico al caso anterior, por lo que sólo comentaremos aquellas líneas que difieren de uno a otro listado.

Línea 3580: Esta vez, inicializamos el puntero con la primera dirección de pantalla.

Líneas 3720 y 3730: En este caso, la rotación del scan se realiza hacia la derecha y se incrementa el puntero en vez de decrementarlo.

Línea 3810: El pixel que habrá que poner a «1» será, esta vez, el de la izquierda del scan.

Las demás líneas son iguales y sólo cambian las etiquetas.

#### SCRPR y SCRBP:

(Scroll de Pantalla aRriba y Scroll de Pantalla aBajo).

Una misma rutina realizará las dos funciones. La rutina tendrá dos puntos de entrada que fijarán el bit 0 de «C» que se usa como flag para indicar si el scroll es ascendente o descendente; también se fija el valor inicial del puntero que será distinto en ambos casos. Después, se continúa en «SCR» que realiza el scroll propiamente dicho. Este es el listado:

```

3850 SCRPR SET 0,C
3860 LD HL,16384
3870 JR SCR
3880 SCRFB RES 0,C
3890 LD HL,22496
3900 SCR PUSH HL
3910 LD DE,23296
3920 PUSH BC
3930 LD BC,32
3940 LDIR
3950 POP BC
3960 POP HL
3970 BU_3 PUSH BC
3980 PUSH BC
3990 LD A,H
4000 AND #07
4010 LD B,A
4020 LD A,H
4030 AND #18
4040 LD A,A
4050 SLA A
4060 SLA A

```

Comentemos cada línea.  
Líneas 3850 a la 3870: Punto de entrada para scroll ascendente. Se pone a «1» el flag y se inicializa el puntero a la primera dirección de pantalla. Se continúa saltando a «SCR».

Líneas 3880 y 3890: Punto de entrada para scroll descendente. Se pone a «0» el flag y se inicializa el puntero a la primera dirección del último scan de pantalla. Se continúa en «SCR».

Líneas 3900 a la 3960: Se empieza por transferir el primer o último scan al buffer de impresora por si hay que recuperarlo luego (sólo habrá que hacerlo en el caso de scroll esférico).

El bloque comprendido entre las líneas 3970 y 4540 constituye un bucle que copiará cada scan en el anterior o posterior hasta que se alcance el final de la pantalla.

Líneas 3970 y 3980: Se preservan los contenidos de «HL» y «BC» ya que «HL» es el puntero y «C» contiene el flag de ascendente/descendente.

Líneas 3990 a la 4130: Se construye en «A» el número de scan, partiendo del valor del puntero (dirección de pantalla).

Líneas 4140 y 4150: Si el flag indica scroll ascendente, se salta a «ANT—1» (en la línea 4200).

Líneas 4160 a 4190: Se incrementa el n.º de scan y se compara con «C0» (192) para poner a «1» el indicador de acarreo sólo si el nuevo scan cae dentro de la pantalla, es decir, si es menor de 192. Seguidamente, se preserva «AF» para preservar el estado actual del indicador de acarreo y el número de scan. Se sale saltando a «SIG—2» (línea 4230).

Líneas 4200 a la 4220: Al contrario que en el caso anterior, esta vez se decrementa el número de scan. Utilizamos «SUB 1» porque «DEC A» no afecta al indicador de acarreo.

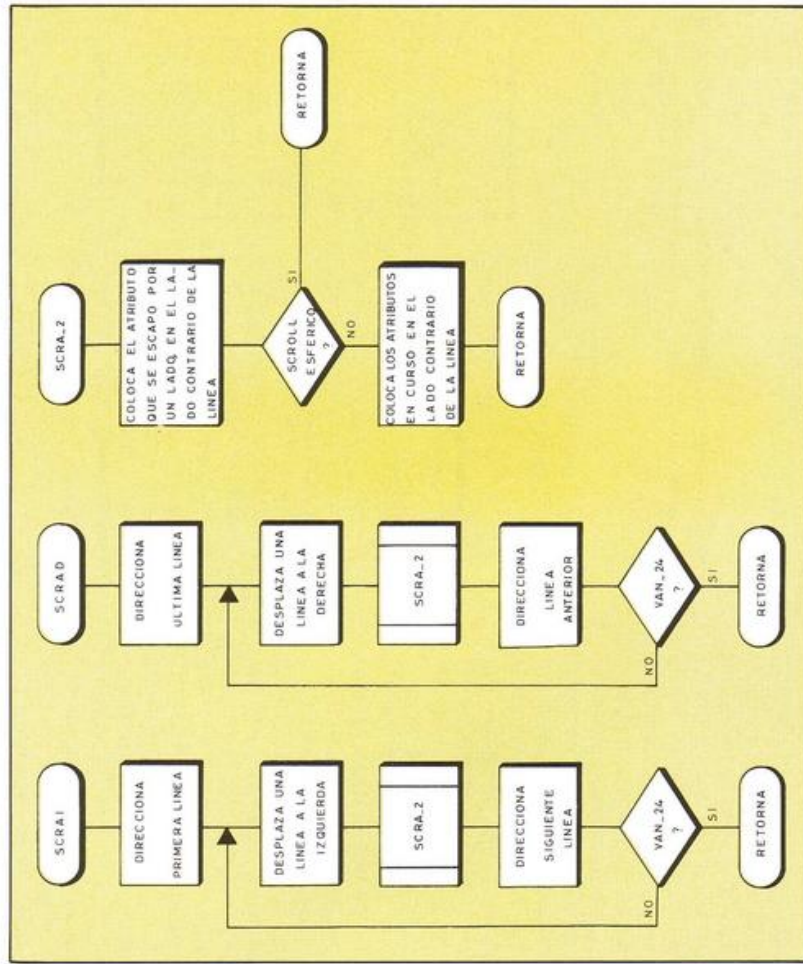


Fig. 11-23. Ordinograma de las rutinas «SCRAI» y «SCRAD».

Sólo nos faltan por ver las dos rutinas que sirven para activar y desactivar nuestro programa. Normalmente, la zona correspondiente al canal «P» en la tabla de canales, contiene la dirección de salida de la rutina que maneja la impresora. Si hacemos funcionar nuestra rutina por este canal, no podremos usar, simultáneamente, la impresora.

Cuando activamos el programa, guardamos la dirección de la rutina que maneja la impresora en las direcciones 23728 y 23729 que corresponden a una variable que el Sistema no utiliza. Y metemos, en su lugar, la dirección

de nuestro programa «PRO-PAN».

Al desactivar, cogemos la dirección que habíamos guardado en 23728 y 23729, y la volvemos a colocar en su lugar correspondiente dentro de la tabla de canales.

«ACT» es la rutina de activar y «DESACT» la de desactivar. Si se ensambla el programa en la dirección 60000, «ACT» quedará situado en 60955 y «DESACT» quedará en 60976. La dirección de activación será siempre 955 bytes más que la dirección donde se haya ensamblado. La de desactivación será 976 bytes más.



```

5230 LD BC, 31
5240 LDDR
5250 CALL SCRA_2
5260 DEC DE
5270 POP BC
5280 DJNZ BU_2
5290 RET
5300 SCRA_2 LD (DE), A
5310 LD A, (FLAGS)
5320 BIT S4
5330 RET NZ
5340 LD A, (22693)
5350 LD (DE), A
5360 RET

```

La subrutina «SCRA-2» se encarga de comprobar si el scroll ha de ser esférico o lineal y actuar en consecuencia. Veamos las rutinas.

La rutina «SCRA» realiza el scroll de atributos a la izquierda.

Líneas 5020 y 5030: Se inicializa el puntero «DE» para contener la dirección del primer byte de la primera línea del archivo de atributos. Se carga «24» en «B» para que actúe como contador en un bucle de 24 iteraciones, tantas como líneas de atributos tiene la pantalla.

Línea 5040: Se preserva el contenido de «BC» para que el valor de «B» no sea destruido durante la ejecución del bucle.

Línea 5050: Se carga en «A» el contenido del primer byte de la línea.

Líneas 5060 a la 5100: Se desplaza toda la línea un lugar a la izquierda. El registro «DE» sale apuntando al último byte de la línea.

Línea 5110: Se llama a «SCRA-2». Ver la explicación dada para las líneas 5300 a la 5360.

Línea 5120: Se incrementa «DE» para que apunte a la primera posición de la siguiente línea.

Líneas 5130 y 5140: Se recupera «BC» y se cierra el bucle.

Línea 5150: Se retorna al Sistema.

La rutina «SCRAD» es muy similar a ésta en cuanto a su funcionamiento.

Líneas 5160 y 5170: Se inicializa el puntero para apuntar a la última dirección de la última línea. Se prepara «B» para un bucle de 24 iteraciones. Esta vez, el scroll se hará de abajo a arriba.

Línea 5180: Exactamente igual que la 5040.

Línea 5190: Idem que la 5050.

Líneas 5200 a la 5240: Esta vez, se desplaza toda la línea un lugar a la derecha. El registro «DE» sale apuntando al primer byte de la línea.

Línea 5250: Se llama a «SCRA-2». Ver la explicación dada para las líneas 5300 a la 5360.

Línea 5260: Se decrementa «DE» para que apunte al último byte de la línea anterior.

Línea 5270 y 5280: Se recupera «BC» y se cierra el bucle.

Línea 5290: Se retorna al Sistema.

La subrutina «SCRA-2» deberá comprobar el flag de «es-

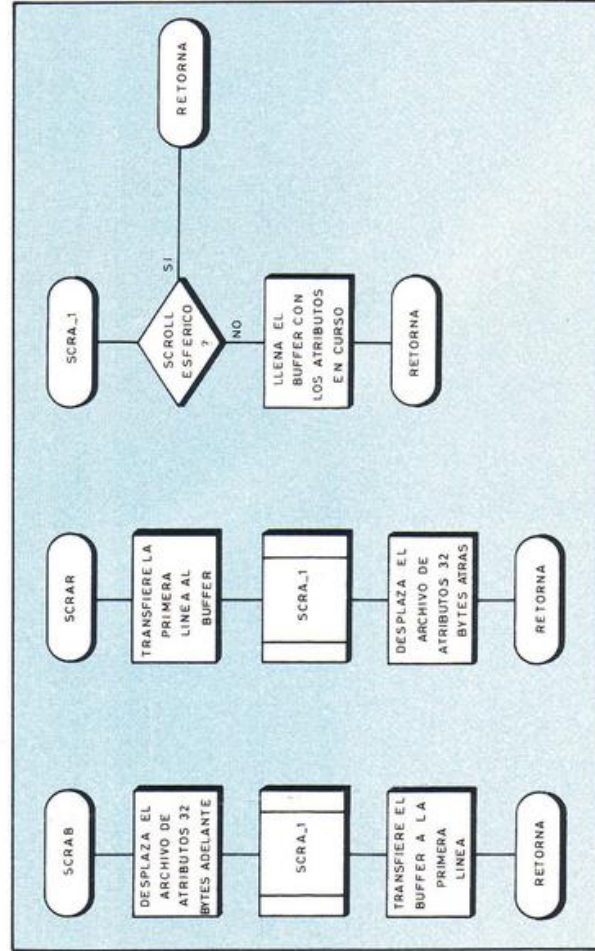


Fig. 11-22. Ordinograma de las rutinas «SCRA» y «SCRA-2».

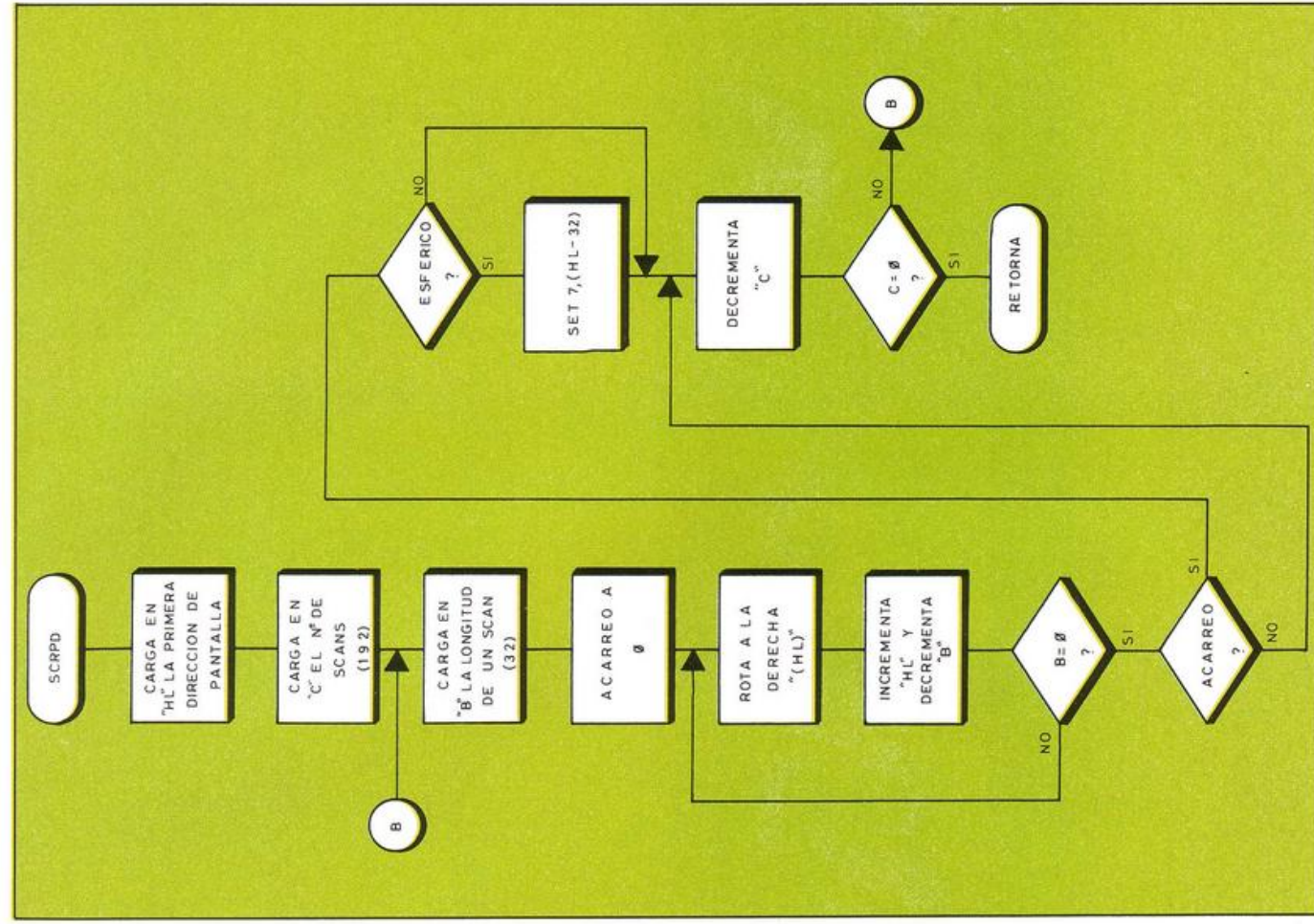


Fig. 11-20. Ordinograma de la rutina «SCRPD».



reco. De esta operación, el indicador sale a «0» si el n.º de scan es correcto, por lo que habrá que complementarlo en la línea 4210. Seguidamente, se preserva «AF» por la misma razón de antes y se sigue en «SIG—2».

Líneas 4230 a la 4430: Se construye una nueva dirección de pantalla en «HL» partiendo del número de scan. El proceso se realiza sin afectar a los cinco bits inferiores de «L» para no alterar el número de columna; aunque, en nuestro caso, éste será siempre «0».

Líneas 4440 a la 4470: Se recupera el n.º de scan en «A», el indicador de acarreo en «F», el flag de ascendente/descendente en «C» y la dirección del anterior scan en «DE». Finalmente, se salta a «FIN—1» (línea 4550) si el acarreo está a cero, indicando que ya se ha terminado con la pantalla, de lo contrario, se sigue en la línea 4480.

Líneas 4480 a la 4540: Se transfieren los 32 bytes desde el nuevo scan al anterior y se cierra el bucle saltando a «BU—3» (línea 3970).

Se llega a «FIN—1» (línea 4550) cuando ya se ha terminado con toda la pantalla y sólo resta ver qué se hace con el último scan dependiendo de que se haya seleccionado scroll esférico o scroll lineal.

Líneas 4550 a la 4580: Se preserva la dirección del último scan procesado y se salta a «FIN—2» (línea 4650) si se ha seleccionado scroll esférico. De lo contrario, se sigue en la línea 4590.

Líneas 4590 a la 4640: Si no se ha seleccionado scroll esférico, hay que borrar el scan que se había almacenado

do en el buffer de impresora. Esto es, precisamente, lo que se hace en estas líneas. Se continúa en «FIN—2» (línea 4650).

Líneas 4650 a la 4690: Se transfieren los 32 primeros bytes del buffer de impresora al último scan procesado cuya dirección está contenida en «DE». Este scan será el primero o el último de la pantalla, según que el scroll haya sido descendente o ascendente. Por último, se retorna al Sistema desde la línea 4690.

#### SCRAB y SCRAR:

(SCRAB de Atributos aBajo y SCRAR de Atributos aRriba). Se trata, realmente, de dos rutinas que realizan el scroll vertical en el fichero de atributos; pero ambas utilizan una subrutina común («SCRA—1») por lo que hemos creído preferible comentarlas juntas. Su listado es el siguiente:

```

4760 SCRAB LD HL,23295
4770 LD DE,23327
4780 LD BC,768
4790 LDDR
4795 PUSH DE
4798 CALL SCRA_1
4800 POP DE
4805 LD HL,23327
4810 LD BC,32
4815 LDDR
4820 RET
4825 SCRAR LD HL,23258
4830 LD DE,23296
4835 LD BC,72
4840 LDIR
4845 PUSH HL
4850 CALL SCRA_1
4855 POP HL
4860 LD DE,23258
4865 LD BC,768
4870 LDIR
4875 RET
4880 SCRAB_1 LD A,(FLAGS)
4885 BIT 3,A
4890 RET NZ
4895 LD HL,23296
4900 LD A,(23293)
4905 LD HL,A
4910 LD DE,23297
4915 LD BC,31
4920 LDIR
4925 RET

```

vo de atributos, más los 32 primeros bytes del buffer de impresora, 32 posiciones hacia atrás, con lo que el archivo se desplaza hacia arriba y los 32 primeros bytes del buffer de impresora entran en la última línea. Se termina retornando al Sistema desde la línea 4910.

En la subrutina «SCRA—1» se comprueba el flag de «esférico»; si está a «1», se retorna sin más; pero si está «0», se copian los atributos permanentes en curso, en los 32 primeros bytes del buffer de impresora.

Líneas 4920 a la 4940: Se comprueba el flag de esférico y se retorna si es «1».

Líneas 4950 a la 5010: Se copian los atributos permanentes en curso (dirección 23693) sobre los 32 primeros bytes del buffer de impresora. Finalmente, se retorna al punto desde donde se llamó a la subrutina.

#### SCRAL y SCRAD:

(SCRAL de Atributos a la Izquierda y SCRAD de Atributos a Derecha). Al igual que en el caso anterior, se trata de dos rutinas independientes que utilizan una subrutina común. Su listado es:

```

5020 SCRAL LD DE,23258
5030 LD B,24
5040 PUSH BC
5050 LD A,(DE)
5060 LD H,D
5070 LD L,E
5080 INC HL
5090 LD BC,31
5100 LDIR
5110 CALL SCRA_2
5120 INC DE
5130 POP BC
5140 DJNZ BU_1
5150 SCRAD LD DE,23295
5160 LD B,24
5170 PUSH BC
5180 LD A,(DE)
5190 LD H,D
5200 LD L,E
5210 DEC HL
5220 RET

```

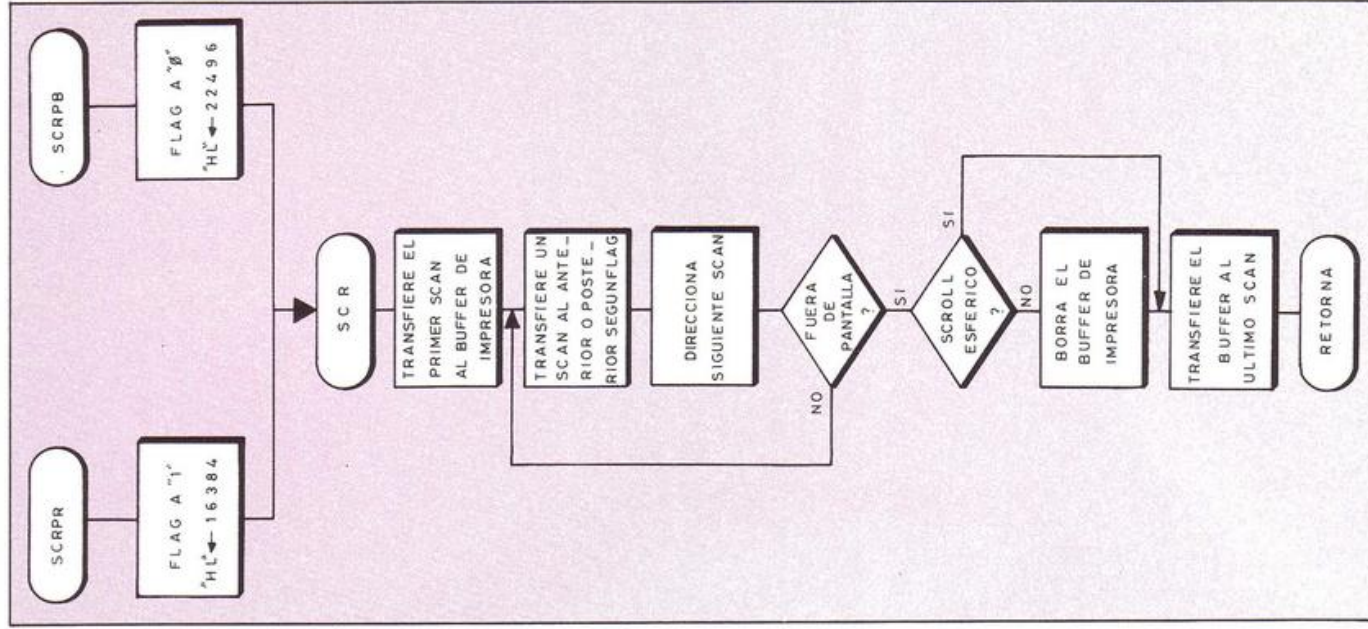


Fig. 11-21. Ordinograma de las rutinas «SCRAB» y «SCRPR».



33	06281E12100581110059D9	520	67	30123AA1EACB5F280B22	902
34	21004101100481C12100	521	68	A2EAD02A2EAD0C8E0FE	1957
35	5911005AD0921100481100	522	69	02E0E1C9CB0C121004018	988
36	5018002110058111005AD9	523	70	05CB8121E057E511005B	1018
37	210040110058111005AD9	524	71	C5012000EDB0C1E11005C	1487
38	3E80B78B120F8E1D906	525	72	7CE607477CE618CB27CB	1255
39	000CE3B10F8C97E081A	526	73	27CB827B0477DE6E0CB3F	1373
40	7708123313C921A1EAFE	527	74	CB3F804D6A128063CFE0C	1262
41	1720036B06C99F1E2003	528	75	F518040B0113FF5472510	969
42	0BCE00FE192003CB06C9	529	76	5F0C83F84677DE6E1F6F78	1531
43	FE1B2003CB06C99F1E2003	530	77	3FC083F84677DE6E1F6F78	1229
44	03CB9EC93EFA677C92A	531	78	D1300B827CB27B56FFC1	1496
45	765C110040FE1E2801EB	532	79	D1300B827CB27B56FFC1	1140
46	01001B0D0C9ED5B885C	533	80	C1E118AD53AA1EACB5F	1576
47	212118ED5EB1860ED5B	534	81	200D210058110158011F	310
48	365C26006F29292919EB	535	82	00AF77EDB0D121005801	1041
49	2A845C00083AA1EACB5F	536	83	2000EDB0C921FF5A11F	1072
50	20241A7738A1EAE60128	537	84	5B0110003EDB8D5DCBEC	1374
51	FE0C3E720E053008CB26	538	85	21011F5B0112000EDB8C9	1019
52	FE0C3E720E053008CB26	539	86	21011F5B0112000EDB8C9	499
53	0228057E0B3F66771324	540	87	80ESCDB0C921FF5A11F	1381
54	1008180E1A0E08CB1FCB	541	88	0003EDB0C921FF5A11F	744
55	160D20F9132410F2ED5B	542	89	5B011F00EDB0C9110058	842
56	885C212118ED5EB1C7B	543	90	0618C51A626B23011F00	525
57	FE20380E1E00147AFE15	544	91	EDB0C921FF5A11F	1539
58	3806C0FE00D110014D57A	545	92	11FF5A0618C51A626B23	853
59	FE0C3E720E053008CB26	546	93	011F00EDB0C921FF5A11F	1131
60	79E518F51A0572D53845C	547	94	011F00EDB0C921FF5A11F	1417
61	D1212118ED5EB1C7B	548	95	3A8D5C12C92A4F5C110F	755
62	2810FB30123AA1EACB5F	549	96	00195E2356ED5B805C11	845
63	280B2A2EAD02A2EAD0	550	97	60EA722B73C92A4F5C11	1033
64	CB20C60020E1C9211004B	551	98	0F0019ED5B805C732372	900
65	0EC00620A7CB1E2310F	552	99	C9000000000000000000	201
66		553	100		

Fig. 11-26. Listado de «Propan» en formato de cargador universal.

## EJERCICIOS

- 1.- En una subrutina, nos interesa retornar si el contenido de "A" es distinto de "0" y, en caso contrario, detener el programa con el informe "b Number too big" del Sistema. Escriba la parte correspondiente de la rutina.
- 2.- Escriba una rutina que imprima, desde código máquina, los caracteres cuyos códigos se encuentran en un bloque de 704 posiciones de memoria cuya primera dirección es 32000. La impresión deberá dirigirse a la pantalla por la corriente #2.
- 3.- Queremos llamar a una subrutina cuya dirección de comienzo está en el registro "DE". La subrutina acabará con un "RET", pero queremos que retorne a la dirección contenida en los dos bytes apuntados por "IX+12". ¿Como la llamaríamos?

----- 0 -----

Esta cabecera es lanzada por el programa, cada vez que empieza a ensamblar, tanto si lo hace por pantalla, como por impresora. La primera línea indica que se trata de la versión «3M» (adaptada para Microdrive). La segunda, que está escrito para correr en un Spectrum (también existe el «GENA» para Amstrad). Después de una línea en blanco, viene el mensaje de «Copy right». La segunda línea de este mensaje, era: «All rights reserved»; nosotros la hemos cambiado para personalizar nuestros listados.

A continuación viene una línea que dice:

Pass 1 errors: 00

El «GENS-3» es un ensamblador en dos pasadas, es decir: cuando ensambla, primero hace una pasada por todo el código fuente para construir una tabla de etiquetas y detectar posibles errores de sintaxis. Si no se detecta ningún error durante esta pasada, se imprime este mensaje y se procede a hacer la segunda, donde se genera el código objeto mientras se lista el programa.

Las líneas 10 y 20 del listado, contienen comandos del Ensamblador. Estos comandos deben iniciarse, siempre, con un asterisco y sólo tienen significado en tiempo de ensamblado. El comando «C» indica que, en el listado, se suprima la impresión del código objeto. De no hacerse así, éste se imprimiría en hexadecimal a continuación de las direcciones y antes de los números de línea.

El comando «D+» indica, al Ensamblador, que imprima las direcciones en decimal. De no existir este comando, las direcciones, al inicio de cada línea, se imprimirían en hexadecimal.

Las líneas 30, 40 y 50 son comentarios que deben empezar, siempre, por «punto y coma». Entre la línea 100 y la 550 está el programa propiamente dicho. Cada línea tiene cinco campos. El primer campo consta de 5 caracteres y es la dirección de memoria donde se ensambla el primer byte de la línea, el segundo campo tiene 4 caracteres y contiene el número de línea, el tercer campo consta de 6 caracteres y contiene las etiquetas, el cuarto consta de 4 caracteres y contiene la instrucción, por último, el quinto campo, de longitud variable, contiene los operandos de la instrucción. Si no hubiéramos puesto el comando «C», habría un campo más entre el primero y el segundo con 8 caracteres de longitud que contendría el código objeto de la línea.

Es posible añadir un campo más a cualquier línea, si se pone un «punto y coma» (;) y, a continuación, un comentario de cualquier longitud. Ningún campo puede contener un «espacio», ya que éste se interpretaría como código de fin de campo. Los espacios suelen sustituirse por el carácter «underscore» (—) que no hay que confundir con el «guión» o signo «menos» (-); aunque es frecuente que, en fotocomposición, nos lo confundan; ¡qué le vamos a hacer!

Tras el listado del programa, viene la línea:

Pass 2 errors: 00

Esta línea indica que tampoco ha habido errores en la segunda pasada. Finalmente, viene el mensaje:

Table used: 994 from 1100

Quiere decir que el Ensamblador ha utilizado 994 bytes de los 1100 que se reservaron, inicialmente, para la tabla de etiquetas.

No le recomendamos a nadie que ensamble este programa «a mano»; la tarea le llevaría horas. Si no tiene Ensamblador, puede utilizar el Cargador Universal de Código Máquina y copiar el listado de la FIGURA 11-26. En este caso, es imprescindible que haga el «DUMP» en la dirección 60000, ya que la rutina no es, en absoluto, reubicable. Para reubicarla en otra dirección, deberá modificar todos los «CALL», «JP» y las direcciones de tablas y variables.

A la vista del programa completo, es posible que algún lector se haya dado cuenta de que hay posibilidades de mejorarlo. Por ejemplo, la rutina «SETFLA» podría haberse sustituido por una colección de rutinas «SET—0» a «SET—5» que se accedieran directamente, desde la tabla inicial, con lo que se evitaría tener que comprobar dos veces el código. Algo similar podría haberse hecho con otras rutinas. También es posible que se pudiera haber ahorrado algo de memoria, utilizando más subrutinas, aunque, a costa de complicar el programa y hacerlo menos legible.



En el desarrollo de este programa ejemplo se ha pretendido no sólo hacer algo que funcionara, sino también ilustrar diversos métodos de programación que el lector pueda emplear en sus propios programas. Por otro lado, cada una de las rutinas funciona por sí sola (excepto, claro está, las que tienen su salida por «SIGUE»); de esta forma, no es necesario que utilice el programa completo cuando quiera conseguir un determinado efecto. Por ejemplo, si necesita escribir un programa en el que tendrá que hacer un «scroll» ascendente de pantalla, le bastará con utilizar el trozo de «PROPAN» comprendido entre las líneas 3850 y 4690 y llamarlo con «USR» en lugar de hacerlo con «LPRINT».

Veámos, ahora, la forma práctica de utilizar «PROPAN». Suponemos que ha cargado el programa, bien con un Ensamblador, bien con el Cargador Universal, y lo tiene colocado a partir de la dirección 60000. Puede salvarlo con:

```
SAVE "PROPAN" CODE 60000, 991
```

Cuando vaya a utilizarlo desde un programa en Basic, deberá tenerlo grabado a continuación del bloque de Basic, que deberá empezar por:

```
10 CLEAR 50000
20 LOAD "PROPAN" CODE
30 RANDOMIZE USR 60000
```

nea de pantalla con el papel y la tinta del mismo color y, luego, realizar 16 desplazamientos de pantalla hacia arriba en modo de «scroll» lineal». Veámos un posible programa:

```
10 LPRINT CHR$(28);
20 PAPER 1: INK 6
30 BORDER 1: CLS
40 READ A$
50 IF A$="" THEN STOP
60 LPRINT AT 21,0; INK 11:A$:
70 LPRINT CHR$(7);CHR$(16);CHR$(17);
80 GO TO 40
100 REM Rótulos
110 DATA "....."
120 DATA "....."
130 DATA "....."
140 DATA "....."
150 DATA "....."
```

```
LPRINT CHR$(24);AT 11,14;"HOLA"
```

El «CHR\$(24)» > sirve para fijar la impresión en negrita. Ahora, supongamos que no quiere borrar la pantalla, sino que la palabra «HOLA» se desplace hacia arriba hasta desaparecer:

```
FOR A=1 TO 96
LPRINT CHR$(17);
NEXT A
```

Esta sería una forma de hacerlo, pero ¿para qué quiere el código 7? Pruebe esto:

```
LPRINT CHR$(7);CHR$(96);CHR$(17);
```

Obtendrá el mismo efecto, pero con más rapidez y gastando menos memoria. Esta línea significa «Repetir 96 veces un desplazamiento de pantalla hacia arriba».

Una de las posibles utilidades de «PROPAN» es crear rótulos como los del cine, para utilizar en las películas de video domésticas. Nos referimos a los «créditos» que aparecen al final de la película, saliendo por debajo de la pantalla y desplazándose hacia arriba hasta desaparecer. Una forma de hacer esto, es imprimir cada rótulo en la última li-

Este programa genera rótulos amarillos sobre fondo azul. El efecto se puede mejorar con unos cuantos «PLOT» aleatorios para que los rótulos se muevan sobre un fondo de «estrellas».

Como ya indicamos, los códigos «0» y «31», no producen ningún efecto, y la rutina se limita a ignorarlos. Estos códigos pueden ser usados por el lector para algún fin; por ejemplo: el código «31» puede utilizarse para desactivar la rutina si en el último lugar de la «TABLA2». Se cambia «NULO» por «DESACT». Otra posibilidad es utilizar el código «0» para retroceder una posición el cursor, con la posibilidad de que pase del principio de una línea al final de la anterior; para ello, basta con cambiar el primer elemento de la «TABLA2» de «NULO» a «DEL-1».

Existen un gran número de modificaciones más que es posible hacer para adaptar el programa a sus propias necesidades. Confiamos en que la

608414770	LD	HL, 23327	609275200	LD	H, D
608444780	LD	BC, 32	609285210	LD	L, E
608474790	LD		609295220	DEC	HL
608494800	RET		609305230	LD	BC, 31
608504810	SCRAR		609315240	LD	SCRAR_2
608534820	LD	HL, 22528	609325250	CALL	DE
608564830	LD	DE, 23296	609335260	DEC	DE
608594840	LD	BC, 32	609345270	POP	BC
608614850	PUSH	HL	609405280	DJNZ	BU_2
608624860	CALL	SCRAR_1	609425290	RET	
608654870	POP	HL	609435300	LD	(DE), A
608664880	LD	DE, 22528	609445310	LD	A, (FLAGS)
608694890	LD	BC, 768	609475320	BIT	3, A
608724900	LD		609495330	RET	NZ
608744910	RET		609505340	LD	A, (23693)
608774920	SCRAR_1		609535350	LD	(DE), A
608784930	BIT	A, (FLAGS)	609545360	LD	
608804940	RET	3, A	609555370	RET	HL, (CHANS)
608814950	LD	NZ	609585380	LD	DE, 15
608844960	LD	HL, 23296	609615390	ADD	HL, DE
608874970	LD	A, (23693)	609625400	LD	E, (HL)
608884980	LD	(HL), A	609635410	INC	HL
608914990	LD	DE, 23297	609645420	LD	D, (HL)
608945000	LD	BC, 31	609655430	LD	(NMI), DE
608965010	LD		609675440	LD	DE, PROPAN
608975020	SCRAR_1		609725450	LD	(HL), D
609005030	LD	DE, 22528	609735460	DEC	HL
609025040	BU_1	B, 24	609745470		
609035050	PUSH	BC	609755480	RET	
609045060	LD	A, (DE)	609765490	DESACT	HL, (CHANS)
609055070	LD	H, D	609775500	LD	DE, 15
609065080	INC	L, E	609825510	ADD	HL, DE
609075090	LD	HL	609835520	LD	DE, (NMI)
609105100	LD	BC, 31	609875530	LD	(HL), E
609125110	CALL	SCRAR_2	609885540	INC	HL
609155120	INC	DE	609895550	LD	(HL), D
609165130	POP	BC	609905560	RET	23631
609175140	DJNZ	BU_1	236315570	CHANS	23728
609195150	RET		237285580	NMI	
609205160	SCRAR				
609235170	LD	DE, 23295			
609255180	LD	B, 24			
609265190	PUSH	BC			
	LD	A, (DE)			

Pass 2 errors: 00

Table used: 994 from 1100

Fig. 11-25. Listado Assembler completo del procesador de pantalla.

1	21815EACB565202020C8565520	1142	17	C110F7C921A1EACB565C9	1735
2	24C57620250B7E20205FE	1082	18	21A1EACB565C9E0D5B885C	1618
3	20D234EC4F875F160021	894	19	212118ED52E8FE09280F	962
4	4E4E195E235BEB79E932	1277	20	FE0A2813FE0B28177BA7	941
5	A2EACBACBEC9CBAC3	1979	21	C81DC3A2EC78FE1FC81C	1458
6	E5EAC32A2EACB6C6CFEC9	1952	22	C3A2EC7AFE15C814C3A2	1567
7	C8BEC3F3EA00C053E4EA	1802	23	EC7A7C815C3A2ECCD4C	1620
8	67EB67EB67EB98EB98EB	1788	24	EB3E20CD34ECED5B885C	1378
9	98EB44EB28ECCE7E7EC	1379	25	212118ED52E8FE09280F	978
10	98EB44EB28ECCE7E7EC	1727	26	1DC3A2EC7A7C815E1F	1193
11	13ED00CEDE1EDF8E99ED	1842	27	4FED4A4623561E00E000	1163
12	B2ED00CEDE1EDF8E99ED	2073	28	40584859505A602E0001	626
13	E4EBC9E9E9E9E9E9E9E9	1931	29	FF07D575541E01EDB0E1	1345
14	E4EBC9E9E9E9E9E9E9E9	1706	30	541E013A8D5C770EEDF	1031
15	FE15D057C3A2EC7E7E7E	1625	31	B0C9D9E5D9FE052813FE	1612
16	21A2E4E46C5F5C6B0EAF1	1717	32		



nado el capítulo. En el próximo estudiaremos las instrucciones que permiten al Z-80 comunicarse con el mundo exterior. Antes, recomendamos al lector que, como siempre, intente resolver los siguientes ejercicios.

demostración del lector sepa encontrarlas todas, así como que la rutina sea útil para realizar efectos de pantalla en sus programas.

Como ejemplo de lo que se puede tardar en escribir un programa en Assembler, ponemos el código de la rutina de

606103510	SCRPI	LD	HL, 22527	BIT	0, C
6061133520	B_2	LD	C, 192	JR	Z, ANT_1
6061153530	B_2	LD	B, 32	INC	A
6061173540	AND	CP	A	CP	#C0
6061183550	B_1	RL	(HL)	PUSH	AF
6062023560	DEC	HL	HL	JR	SIG_2
6062113570	DJNZ	B_1	ANT_1	SUB	1
6062233580	JR	NC, NOCA_1	SIG_2	CCF	AF
6062353590	LD	A, (FLAGS)	SIG_2	PUSH	AF
6062836000	BIT	3, A	SIG_2	LD	B, A
6063036100	JR	Z, NOCA_1	SIG_2	LD	H, #40
6063236200	LD	(VAR_1), HL	SIG_2	AND	#07
6063356300	LD	(VAR_1), HL	SIG_2	OR	H
6063936400	SET	IX, (VAR_1)	SIG_2	LD	H, A
6064336500	NOCA_1	DEC	C	LD	A, B
6064436600	JR	NZ, B_2	SIG_2	AND	#C0
6064636700	RET	HL, 16384	SIG_2	SRL	A
6064736800	SCRPI	LD	C, 192	SRL	A
6065036900	LD	B, 32	SIG_2	SRL	A
6065237000	B_4	LD	A	OR	H
6065437100	RR	(HL)	SIG_2	LD	A, A
6065537200	B_3	LD	A, L	LD	#1F
6065737300	INC	HL	A	LD	L, A
6065837400	DJNZ	B_3	A	AND	#3B
6066037500	JR	NC, NOCA_2	SIG_2	SLA	A
6066237600	LD	A, (FLAGS)	SIG_2	OR	L
6066537700	BIT	3, A	SIG_2	LD	L, A
6066737800	JR	Z, NOCA_2	SIG_2	POP	AF
6066937900	LD	(VAR_1), HL	SIG_2	POP	BC
6067238000	LD	IX, (VAR_1)	SIG_2	POP	DE
6067638100	SET	7, (IX-32)	SIG_2	JR	NC, FIN_1
6068038200	NOCA_2	DEC	C	HL	HL
6068138300	JR	NZ, B_4	SIG_2	PUSH	BC
6068338400	RET	HL, 16384	SIG_2	LD	BC, 32
6068438500	SCRPI	LD	0, C	LDIR	LD
6068638600	JR	SCR	HL, 22496	POP	BC
6068938700	RES	0, C	HL, 22496	POP	HL
6069138800	SCRPI	LD	HL, 22496	POP	BU_3
6069338900	LD	HL, 22496	HL, 22496	JR	DE
6069639000	SCR	LD	DE, 23296	PUSH	A, (FLAGS)
6069739100	LD	BC, 32	FIN_1	LD	3, A
6070039200	PUSH	BC	FIN_1	JR	NZ, FIN_2
6070139300	LD	BC, 32	FIN_1	LD	HL, 23296
6070439400	LDIR	LD	BC, 31	LD	DE, 23297
6070639500	POP	BC	(HL), A	LD	BC, 31
6070739600	POP	HL	(HL), A	XOR	A
6070839700	PUSH	HL	FIN_2	LDIR	LD
6070939800	PUSH	BC	FIN_2	POP	DE
6071039900	LD	A, H	FIN_2	LD	HL, 23296
6071140000	AND	#07	FIN_2	LD	BC, 32
6071340100	LD	B, A	FIN_2	LDIR	LD
6071440200	LD	A, H	FIN_2	POP	HL, 23296
6071540300	AND	#18	FIN_2	LD	BC, 32
6071740400	SLA	A	FIN_2	LD	BC, 31
6071940500	SLA	A	FIN_2	XOR	A
6072140600	SLA	A	FIN_2	LDIR	LD
6072240700	OR	B	FIN_2	POP	DE
6072440800	LD	B, A	FIN_2	LD	HL, 23296
6072540900	LD	A, L	FIN_2	LD	BC, 32
6072641000	AND	#E0	FIN_2	LDIR	LD
6072841100	SRL	A	FIN_2	RET	HL, 23296
6073041200	SRL	A	FIN_2	LD	DE, 23327
6073241300	OR	B	FIN_2	LD	BC, 768



60163 990	RET	HL, FLAGS	602751620	LD	D, (HL)	603922250	LD	(DE), A	605042880	CP	5
601641000	LD	6, (HL)	602761630	LD	CLS3_1	603932260	INC	HL	605062890	JR	NC, NOCURS
601671010	SET		602781640	JR	CLS3_1	603942270	INC	DE	605082900	SLA	(HL)
601691020	RET		602801650	DEFB	#40, #5B	603952280	RET		605102910	CP	3
601701030	LD	HL, FLAGS	602821660	DEFB	#48, #59	603962290	SETFLA	HL, FLAGS	605122920	JR	NC, NOCURS
601731040	SET	4, (HL)	602841670	DEFB	#50, #5A	603972300	CP	23	605142930	SLA	(HL)
601751050	RET		602861680	CLS3_1	H, B	6040012310	JR	NZ, SET_1	605162940	NOCURS	LD
601761060	LD	DE, (S_POSN)	602881690	LD	L, 0	604032320	SET	0, (HL)	605192950	AND	2
601801070	LD	HL, #1821	602891700	LD	BC, #07FF	604052330	RET		605212960	JR	Z, NOBOLD
601831080	SBC	HL, DE	602921710	PUSH	DE	604062340	SET_1	24	605222970	LD	A, (HL)
601851090	EX	DE, HL	602931720	LD	(HL), L	604082350	CP	1, (HL)	605242980	SRL	A
601861100	CP	9	602951730	LD	D, H	604102360	SET	NZ, SET_2	605262990	OR	(HL)
601881110	JR	Z, CUR_2	602971740	LD	E, 1	604122370	RET		605273000	LD	(HL), A
601901120	CP	10	602991750	LDIR	HL	604132380	SET_2	25	605283010	NOBOLD	INC
601921130	JR	Z, CUR_3	603011760	POP	HL	604152390	JR	NZ, SET_3	605293020	INC	H
601941140	CP	11	6030311770	LD	D, H	604172400	SET	2, (HL)	605303030	DJNZ	BUCLE1
601961150	JR	Z, CUR_4	6030511780	LD	E, 1	604192410	RET		605323040	JR	ACTUAL
601981160	LD	A, E	6030711790	LD	A, (23693)	604202420	CP	27	605343050	IMPR_2	LD
601991170	AND	A	6030911800	LD	(HL), A	604222430	JR	NZ, SET_4	605353060	LD	A, (DE)
602001180	RET	Z	603071810	LD	C, #FF	604242440	SET	3, (HL)	605373070	BUCLE2	RR
602011190	DEC	E	603091820	LDIR		604262450	RET		605393080	RL	A
602021200	JP	SIGUE	603111830	RET		604272460	SET_4	28	605413090	DEC	C
602051210	LD	A, E	603121840	EXX	HL	604292470	CP	NZ, SET_5	605423100	JR	NZ, BUCLE2
602061220	CP	31	603131850	PUSH		604312480	RES	3, (HL)	605443110	INC	DE
602081230	RET	Z	603141860	EXX		604332490	RET		605453120	INC	H
602091240	INC	E	603151870	CP	5	604342500	LD	A, #FB	605463130	DJNZ	IMPR_2
602101250	JP	SIGUE	603171880	JR	Z, OP_2	604362510	AND	(HL)	605483140	ACTUAL	LD
602131260	LD	A, D	603191890	CP	6	604372520	LD	(HL), A	605523150	LD	HL, #1821
602141270	CP	21	603211900	JR	Z, OP_3	604382530	RET		605533160	SBC	HL, DE
602161280	RET	Z	603231910	LD	HL, #4800	604392540	TRAPAN	HL, (SEED)	605573170	EX	DE, HL
602171290	INC	D	603261920	LD	DE, #4800	604422550	LD	DE, #4000	605583180	INC	E
602181300	JP	SIGUE	603291930	EXX	DE, #5900	604452560	CP	30	605593190	LD	A, E
602211310	LD	A, D	603301940	LD	HL, #4000	604472570	EX	Z, RECU	605603200	CP	32
602221320	AND	A	603331950	LD	DE, #4800	604492580	EX	DE, HL	605623210	JR	C, SIGUE
602231330	RET	Z	603361960	JR	TRANS	604522590	RECU	BC, 6912	605643220	INC_LI	LD
602241340	DEC	D	603381970	LD	HL, #5900	604532600	LDIR		605663230	INC	D
602251350	JP	SIGUE	603411980	LD	DE, #5A00	604552610	RET		605673240	LD	A, D
602281360	DELETE	DEL_1	603441990	EXX		236702620	SEED	23670	605683250	CP	21
602311370	LD	A, 32	603452000	LD	HL, #4800	604562630	ENTER	DE, (S_POSN)	605703260	JR	C, SIGUE
602331380	CALL	IMP_A	603482010	LD	DE, #5000	604602640	LD	HL, #1821	605723270	CALL	SCROLL
602401400	LD	DE, (S_POSN)	603512020	JR	TRANS	604632650	SBC	HL, DE	605753280	LD	DE, #1400
602431410	SBC	HL, #1821	603532030	LD	HL, #5800	604652660	EX	DE, HL	605783290	SIGUE	PUSH
602451420	EX	DE, HL	603562040	LD	DE, #5A00	604662670	JR	INC_LI	605793300	LD	A, D
602461430	LD	A, E	603592050	EXX		604682680	LD	DE, (CHARS)	605803310	AND	#07
602471440	AND	A	603622060	LD	HL, #4000	604722690	LD	H, 0	605823320	RRC	A
602481450	JR	Z, DEL_2	603632070	LD	DE, #5000	604742700	LD	L, A	605843330	RRC	A
602501460	DEC	E	603662080	LD	BC, 204B	604752710	ADD	HL, HL	605863340	RRC	A
602511470	JP	SIGUE	603692090	CALL	INTE_1	604762720	ADD	HL, HL	605883350	OR	E
602541480	LD	A, D	603722100	DEC	BC	604772730	ADD	HL, HL	605893360	LD	E, A
602551490	AND	A	603732110	LD	A, B	604782740	EX	DE, HL	605903370	LD	A, D
602561500	RET	Z	603742120	OR	C	604792750	LD	HL, (DF_CC)	605913380	AND	#18
602571510	DEC	D	603752130	JR	NZ, BUC_1	604802760	LD	B, 8	605933390	OR	H, A
602581520	LD	E, 31	603772140	POP	HL	604832770	LD	A, (FLAGS)	605953400	LD	D, A
602591530	JP	SIGUE	603782150	EXX		604852780	LD	A, (FLAGS)	605963410	LD	(DF_CC), DE
602601540	DEC	A	603792160	CALL	INTE_1	604882790	BIT	2, A	606003420	POP	DE
602631550	ADD	A, A	603812170	LD	BUC_2	604902800	JR	NZ, IMPR_2	606013430	LD	HL, #1821
602641560	LD	HL, TABLA	603842180	RET	BUC_2	604922810	LD	A, (DE)	606043440	SBC	HL, DE
602651570	LD	B, 0	603862190	LD	A, (HL)	604932820	LD	(HL), A	606063450	LD	(S_POSN), HL
602681580	LD	C, A	603872200	EX	AF, AF'	604942830	AND	1	606093460	RET	23606
602701590	ADC	HL, BC	603882210	LD	A, (DE)	604972840	JR	Z, NOCURS	236843470	CHARS	EQU
602711590	LD	B, (HL)	603892220	LD	(HL), A	604992850	SRL	(HL)	236843480	DF_CC	EQU
602731600	INC	HL	603902230	EX	AF, AF'	605012860	LD	A, B	236883490	S_POSN	EQU
602741610			603912240			605032870			358233500	SCROLL	EQU



## INDICADORES DE CONDICION QUE AFECTA:

Z; pone 1 - si B-1 es igual a cero;  
pone 0 - en cualquier otro caso

N; pone 1 - siempre

## CICLOS DE MEMORIA:

4

## CICLOS DE RELOJ:

16

## EJEMPLO:

IND 0 1 1 1 1 0 0 1

Contenido del registro «C».

(C): 0 1 1 1 1 0 0 1

Contenido del registro «B».

(B): 0 0 1 0 0 1 0 1

Contenido del par de registros «HL»

(H): 0 1 1 1 1 0 0 1  
(L): 1 0 1 0 0 1 1 1

El contenido de la posición de memoria 79A3h no es significativo

## Instrucción

IND: 1 1 1 0 1 1 0 1  
EDh: 1 0 1 0 1 0 1 0

Bus de direcciones resultante.

A8-A15: 0 0 1 0 0 1 0 1  
A0-A7: 0 1 1 1 1 0 0 1

Valor aparecido en bus de datos.

D0-D7: 0 1 0 0 1 0 1 0

Contenido del octeto 79A3h después de la ejecución

79A3h: 0 1 0 0 1 0 1 0

Contenido del registro «B» después de la ejecución

(B): 0 0 1 0 0 1 0 0

Contenido del par de registros «HL»

(H): 0 1 1 1 1 0 0 1  
(L): 1 0 1 0 0 1 0 1

Indicadores de condición después de la ejecución

S Z H P V N C

x 0 x x x x 1 x

Resultado de la operación.  
Desde el dispositivo conectado en port 79h ha entrado el carácter ASCII «J» (4Ah), el cual ha quedado almacenado en la posición de memoria 79A3h.

INDR

## OBJETO:

Coloca el contenido del registro «C» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro «B» se coloca en la mitad superior del bus de direcciones y puede utilizarse como contador de octetos. El octeto procedente del port seleccionado aparece en el bus de datos y se escribe en la posición de memoria direccionada por el contenido del par de registros «HL». Entonces

se decrementa el valor del par de registros «HL» y el valor del registro «B». Si el registro «B» alcanza el valor cero se termina la instrucción; en caso contrario el registro «PC» se decrementa en 2 con lo que se repite la instrucción. Las interrupciones no paran la ejecución de esta instrucción por lo que se atenderán cuando termine.

## CODIGO DE MAQUINA:

1 1 1 0 1 1 0 1  
1 0 1 1 1 0 1 0

INDICADORES DE CONDICION QUE AFECTA:

Z; pone 1 - siempre  
N; pone 1 - siempre

## CICLOS DE MEMORIA:

Si «B» diferente de cero

4

Si «B» igual cero

4

## CICLOS DE RELOJ:

Si «B» diferente de cero

21

Si «B» igual cero

16

## EJEMPLO:

INDR 0 0 1 0 0 1 0 1

Contenido del registro «C».

(C): 0 0 1 0 0 1 0 1

Contenido del registro «B».

(B): 0 0 0 0 0 1 0 1

Contenido del par de registros «HL»

1.- La subrutina podría terminar así:

```
.....
AND A           ;Levanta indicadores
RET NZ          ;Retorna si Z=0
RST #8          ;Llamada a "ERROR"
DEFB 5          ;Código del informe menos 1
```

2.- Haremos uso de «RST #10» haciendo, previamente, que la corriente en curso sea la #2.

```
LD A,2          ;La corriente en curso
CALL #1601      ;será la #2.
LD HL,32000     ;Inicializa puntero.
LD BC,704       ;Inicializa contador.
BUCLE LD A,(HL) ;Carga código.
PUSH HL         ;Preserva puntero.
PUSH BC         ;Preserva contador.
RST #10         ;Imprime carácter.
POP BC          ;Recupera contador.
POP HL          ;Recupera puntero.
INC HL          ;Incrementa puntero.
DEC BC          ;Decrementa contador
LD A,B          ;Comprueba si contador
OR C            ;ha llegado a cero.
JR NZ,BUCLE     ;Cierra el bucle.
RET            ;Retorna.
```

3.- La forma de llamar a la subrutina sería:

```
LD HL,(IX+2)    ;Dir. de retorno a "HL".
PUSH HL         ;Dir. de retorno a pila.
EX DE,HL        ;Dir. de inicio a "HL".
JP (HL)         ;Salta a dir. de inicio.
```



## GRUPO DE INSTRUCCIONES DE ENTRADA Y SALIDA

Una de las principales características que tiene que tener un ordenador es la posibilidad de comunicarse con el exterior. Este cometido es para el que se diseñaron las ins-

trucciones de entrada salida. Los ordenadores pequeños solucionan este problema utilizando los mismos buses que para acceder a la memoria; pero señalando, con el bus de

control, que el acceso pedido no es a memoria sino a un dispositivo de entrada/salida. Como vimos al principio del curso, el microprocesador Z-80 tiene tres buses: «bus de

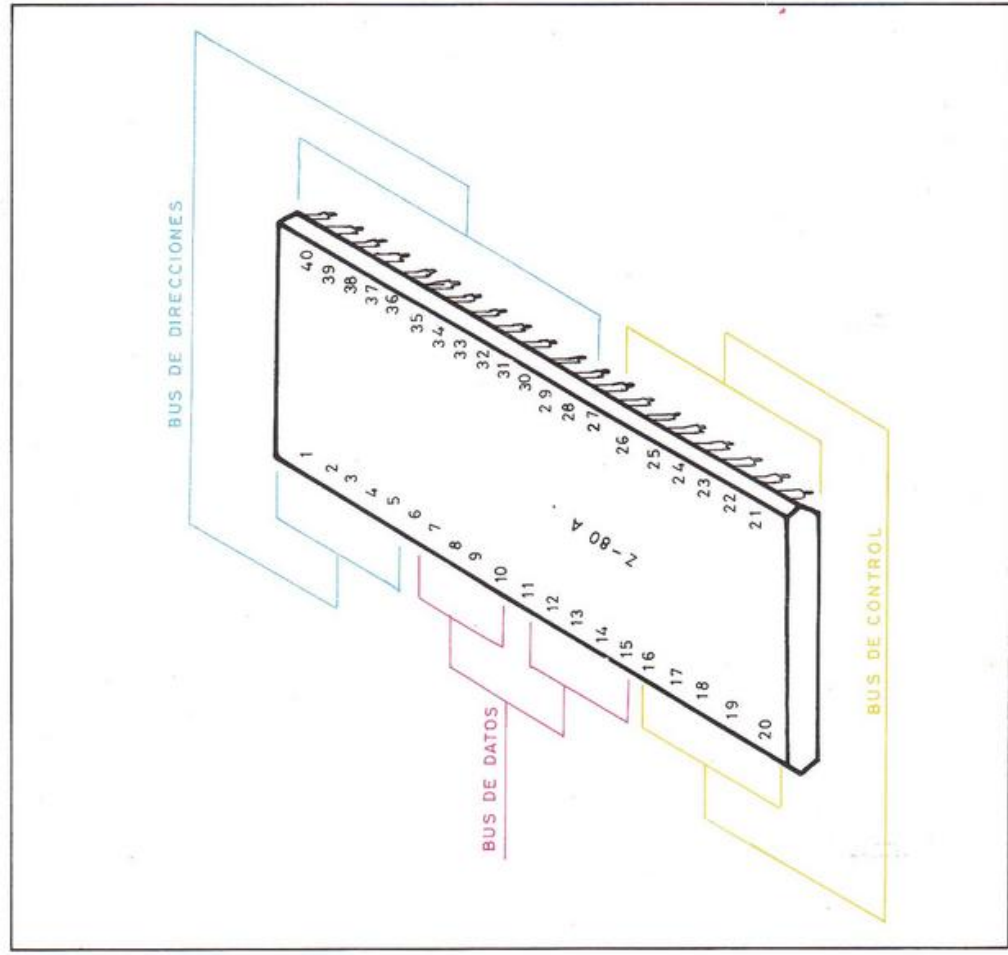


Fig. 12-1. Disposición de los buses en la pastilla del Z-80.

N; pone 1 - siempre

### CICLOS DE MEMORIA:

Si «B» diferente de cero

5

Si «B» igual cero

4

### CICLOS DE RELOJ:

Si «B» diferente de cero

21

Si «B» igual cero

16

### EJEMPLO:

INIR

Contenido del registro «C».

IC: 1 1 1 1 1 1 0 FEh

Contenido del registro «B».

IB: 0 0 0 0 0 1 1 0 0 6h

Contenido del par de registros «HL»

IH: 0 1 1 1 0 1 0 0 74h  
IL: 0 0 1 0 0 0 1 0 27h

El contenido de las posiciones de memoria desde 7422h a 7427h no es significativo.

### Instrucción

INIR: 1 1 1 0 1 1 0 1 EDh  
1 0 1 1 0 0 1 0 BZh

Primer bus de direcciones resultante.

AB A15: 0 0 0 0 0 1 1 0 0 6h  
AB A7: 1 1 1 1 1 1 1 0 FEh

Ultimo bus de direcciones resultante.

AB A15: 0 0 0 0 0 0 0 1 0 1h  
AB A7: 1 1 1 1 1 1 1 0 FEh

Valores aparecidos en bus de datos hasta que «B» es cero.

D0 D7: 0 1 0 0 0 1 0 0 44h

D0 D7: 0 1 0 0 0 1 0 1 45h

D0 D7: 0 1 0 0 0 1 1 0 46h

D0 D7: 0 1 0 0 0 1 1 1 47h

D0 D7: 0 1 0 0 1 0 0 0 48h

D0 D7: 0 1 0 0 1 0 0 1 49h

Contenido de los octetos 7422h a 7427h después de la ejecución

7422h: 0 1 0 0 0 1 0 0 44h

7423h: 0 1 0 0 0 1 0 1 45h

7424h: 0 1 0 0 0 1 1 0 46h

7425h: 0 1 0 0 0 1 1 1 47h

7426h: 0 1 0 0 1 0 0 0 48h

7427h: 0 1 0 0 1 0 0 1 49h

Contenido del registro «B» después de la ejecución

IB: 0 0 0 0 0 0 0 0 00h

Contenido del par de registros «HL»

IH: 0 1 1 1 0 1 0 0 74h  
IL: 0 0 1 0 1 0 0 1 28h

Indicadores de condición después de la ejecución

S Z H P V N C

x 1 x x x x 1 x

Resultado de la operación. Desde el dispositivo conectado en port FEh han entrado los caracteres ASCII «D», «E», «F», «G», «H» e «I» (44h, 45h, 46h, 47h, 48h y 49h), los cuales han quedado almacenados en las posiciones de memoria 7422h a 7427h.

IND

### OBJETO:

Coloca el contenido del registro «C» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro «B» se coloca en la mitad superior del bus de direcciones y puede utilizarse como contador de octetos. El octeto procedente del port seleccionado aparece en el bus de datos y se escribe en la posición de memoria direccionada por el contenido del par de registros «HL». Finalmente se decrementa el valor del par de registros «HL» y el del registro «B».

### CODIGO DE MAQUINA:

EDh 1 1 1 0 1 1 0 1  
A4h 1 0 1 0 1 0 1 0



**INI**

OBJETO:

Coloca el contenido del registro «C» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro «B» se coloca en la mitad superior del bus de direcciones y puede utilizarse como contador de octetos. El octeto procedente del port seleccionado aparece en el bus de datos y se escribe en la posición de memoria direccionada por el contenido del par de registros «HL». Finalmente se incrementa el valor del par de registros «HL» y se decrementa el valor del registro «B».

CODIGO DE MAQUINA:

11101101	Edh
10100010	A2h

INDICADORES DE CONDICION QUE AFECTA:

Z; pone 1 - si B-1 es igual a cero;  
pone 0 - en cualquier otro caso  
N; pone 1 - siempre

CICLOS DE MEMORIA:

4

CICLOS DE RELOJ:

16

EJEMPLO:

INI	
-----	--

Contenido del registro «C».

01110001	7h
----------	----

Contenido del registro «B».

00000111	07h
----------	-----

Contenido del par de registros «HL»

10010011	93h
00101010	2Ah

El contenido de la posición de memoria 932Ah no es significativo

Instrucción

11101101	Edh
10100010	A2h

Bus de direcciones resultante.

00000111	07h
01110001	7h

Valor aparecido en bus de datos.

01000011	43h
----------	-----

Contenido del octeto 932Ah después de la ejecución

01000011	43h
----------	-----

Contenido del registro «B» después de la ejecución

00000110	06h
----------	-----

Contenido del par de registros «HL»

10010011	93h
00101011	2Bh

Indicadores de condición después de la ejecución

S Z H PW N C

x 0 x x x 1 x	
---------------	--

Resultado de la operación. Desde el dispositivo conectado en el port 71h ha entrado el carácter ASCII «C» (43h), el cual ha quedado almacenado en la posición de memoria 932Ah.

**INIR**

OBJETO:

Coloca el contenido del registro «C» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro «B» se coloca en la mitad superior del bus de direcciones y puede utilizarse como contador de octetos. El octeto procedente del port seleccionado aparece en el bus de datos y se escribe en la posición de memoria direccionada por el contenido del par de registros «HL». Entonces se incrementa el valor del par de registros «HL» y se decrementa el valor del registro «B». Si el registro «B» alcanza el valor cero se termina la instrucción; en caso contrario el registro «PC» se decrementa en 2 con lo que se repite la instrucción.

Las interrupciones no paran la ejecución de esta instrucción por lo que se atenderán cuando termine.

CODIGO MAQUINA:

11101101	Edh
10100010	B2h

CURSO12A

INDICADORES DE CONDICION QUE AFECTA:

Z; pone 1 - siempre

direcciones», «bus de datos» y «bus de control» (ver Figura 12-1).

El bus de direcciones está formado por 15 patas (A0-A15) del micro-procesador, desde el punto de vista software, dos octetos. Con el valor de estos dos octetos se selecciona el dispositivo periférico al cual nos queremos dirigir, la capacidad teórica de periféricos es, por tanto, de 65536. El valor de estas 16 patas o bus de direcciones es controlado por medio de las instrucciones de entrada/salida.

El bus de datos está formado por 8 patas (D0-D7) del micro-procesador, desde el punto de vista software, un octeto. El valor de este octeto indica el dato que está entrando o saliendo según sea el caso.

El bus de control se utiliza para indicar el tipo de operación que se está realizando. Si se accede a memoria, se pondrá a cero la pata «MREQ». Y, si se accede a un dispositivo de entrada/salida,

se pondrá a cero la pata «IORQ». En ambos casos, para accesos de lectura se pone a cero la pata «RD» y para los de escritura, la «WR».

Resumiendo, se puede decir que una instrucción de entrada/salida actúa como una instrucción de lectura/escritura de memoria. Por medio del bus de direcciones posicionamos un periférico como si fuera una dirección de memoria y captura o almacena un valor en el bus de datos como si fuera el octeto de memoria.

La comunicación física con el exterior la realizan los ordenadores por medio de regladas que están conectadas directamente al micro-procesador. En el SPECTRUM es el famoso «slot de expansión».

Por tanto, siempre presente que el bus de direcciones se refiere a las patas A0-A15, parte menos significativa A0-A7 y parte más significativa A8-A15, y que sirve para seleccionar el periférico. El bus de datos se refiere a las patas D0-D7 e indica el valor a transferir.

En la Figura 12-2 se puede ver la semejanza que existe entre las instrucciones de acceso a memoria y las de entrada/salida. En cada caso el micro-procesador indica lo que quiere hacer mediante las señales de control. Realmente los buses de datos y direcciones los usa la CPU para todo acceso a su entorno bien sea a memoria RAM, ROM, periféricos, etc. Es igualmente válido para leer las instrucciones de memoria como para ejecutarlas. La razón por la cual hasta este momento no se han tratado es porque desde el punto de vista software, o sea del programador que pretende realizar un proceso, no existe necesidad de saber cómo se las apaña el micro-procesador para realizar lo que se le pida. Es al tratar las instrucciones de este grupo cuando más directamente entra el programador en contacto con estos buses.

Existe, no obstante, una pequeña diferencia en cuanto al acceso a los periféricos a través del bus de direcciones.

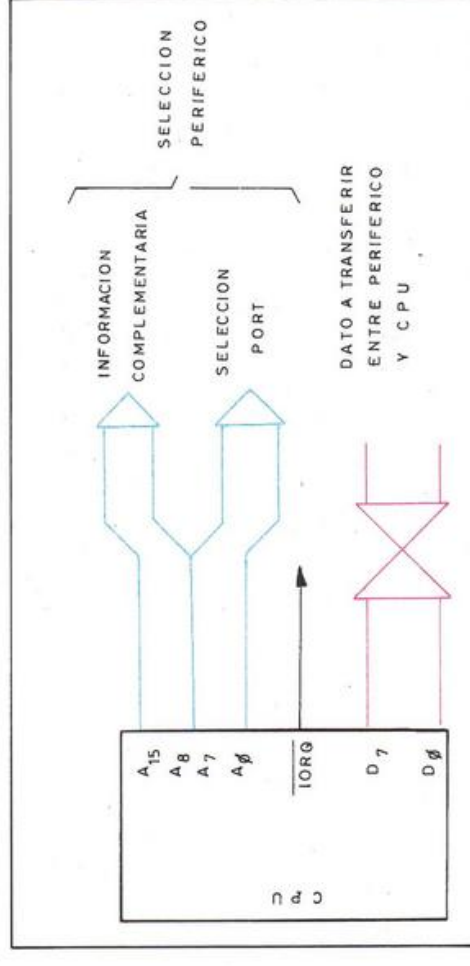


Fig. 12-2A. Direccionamiento en operaciones de entrada/salida.



Cuando dábamos una instrucción para acceder a una posición de memoria, utilizábamos dos octetos para indicar la dirección de esa posición; por ejemplo: Para cargar el Acumulador con el contenido de la posición «4532h», hacíamos: LD A, (#4532) que codificábamos con tres octetos. El primero era el código de operación y los dos siguientes contenían la dirección donde estaba el operando.

En el acceso a un dispositivo de entrada/salida, sólo dispondremos de un octeto para codificar el operando. Este octeto contendrá el número que aparecerá en la parte baja del bus de direcciones. Lo que aparecerá en la parte alta del mismo, será el contenido de un cierto registro; que podrá ser el «A» o el «B» dependiendo de la instrucción concreta que estemos procesando.

Se suele decir que el Z-80 sólo puede direccionar 256 ports de entrada salida. Esto es falso. En realidad, se pueden direccionar 65536 ports;

**OBJETO:**  
Coloca el valor del operando «n» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro acumulador se coloca en la mitad superior del bus de direcciones. El octeto procedente del port seleccionado aparece en el bus de datos y se escribe en el registro acumulador «A». En esta instrucción, se direcciona en modo «directo» la mitad inferior del bus, y en modo indirecto la mitad superior.

**CODIGO DE MAQUINA:**



**INDICADORES DE CONDICION QUE AFECTA:**

Ninguno

**CICLOS DE MEMORIA:**

3

**CICLOS DE RELOJ:**

11

**Instrucciones de entrada**

**IN A, (n)**

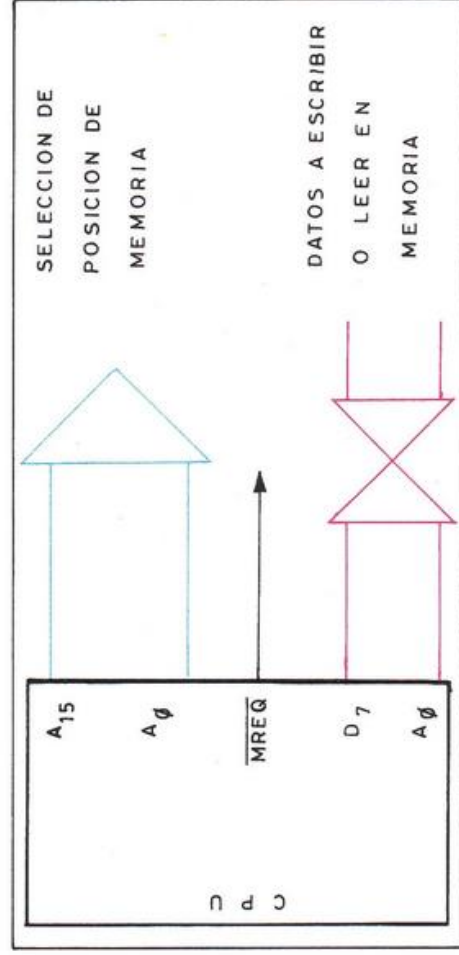


Fig. 12-2B. Direccionamiento en operaciones de acceso a memoria.

**EJEMPLO:**

**IN A, (27)**

Contenido del registro «A».

**(A):** 00111010 3Ah

**Instrucción**

**IN A, (27):** 11011011 00000001 D8h 18h

**Bus de direcciones resultante.**

**A8-A15:** 00111010 3Ah

**A0-A7:** 00000001 18h

**Valor aparecido en bus de datos (ejemplo arbitrario).**

**(D):** 01000001 41h

**Contenido del registro «A» después de la ejecución**

**(A):** 01000001 41h

**Resultado de la operación.**  
Desde el dispositivo conectado en el port 27 (18h) ha entrado el carácter ASCII «A» (41h), el cual ha quedado almacenado en el registro acumulador.

**IN r, (C)**

**OBJETO:**

Coloca el contenido del registro «C» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro «B» se coloca en la mitad superior del bus de direcciones. El octeto procedente del port seleccionado aparece en el bus de datos y se escribe en el registro acumulador «A». En esta instrucción, se direcciona en modo «directo» la mitad inferior del bus, y en modo indirecto la mitad superior.

dente del port seleccionado aparece en el bus de datos y se escribe en el registro representado por «r». El código de representación de «r» es el indicado a continuación.

Registro	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

En realidad, esta instrucción debería ser «IN r, (BC)» ya que es el contenido de «BC» lo que se coloca en el bus de direcciones. Se utiliza, por tanto, direccionamiento indirecto para todo el bus. No obstante, la forma correcta de escribirla es «IN r, (C)» y, si se escribe de otro modo, no será reconocida por ningún ensamblador.

**CODIGO DE MAQUINA:**



**INDICADORES DE CONDICION QUE AFECTA:**

S; pone 1 - si el octeto entrante es negativo;  
pone 0 - en cualquier otro caso  
Z; pone 1 - si el octeto entrante es cero;  
pone 0 - en cualquier otro caso

H; pone 0 - siempre  
N; pone 0 - siempre  
P; pone 1 - si la paridad es par;  
pone 0 - en cualquier otro caso  
**CICLOS DE MEMORIA:**

3

**CICLOS DE RELOJ:**

12

**EJEMPLO:**

**IN H, (C)**

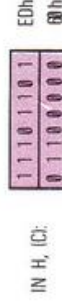
Contenido del registro «C».

**(C):** 00101010 2Ah

Contenido del registro «B».

**(B):** 01110100 74h

**Instrucción**



**Bus de direcciones resultante.**

**A8-A15:** 01110100 74h

**A0-A7:** 00101010 2Ah

**Valor aparecido en bus de datos (ejemplo arbitrario).**

**(D):** 01000010 42h

**Contenido del registro «H» después de la ejecución**

**(H):** 01000010 42h

**Indicadores de condición después de la ejecución**

S Z H PV N C

**Resultado de la operación.**

Desde el dispositivo conectado en port el 2Ah ha entrado el carácter ASCII «B» (42h), el cual ha quedado almacenado en el registro «H».



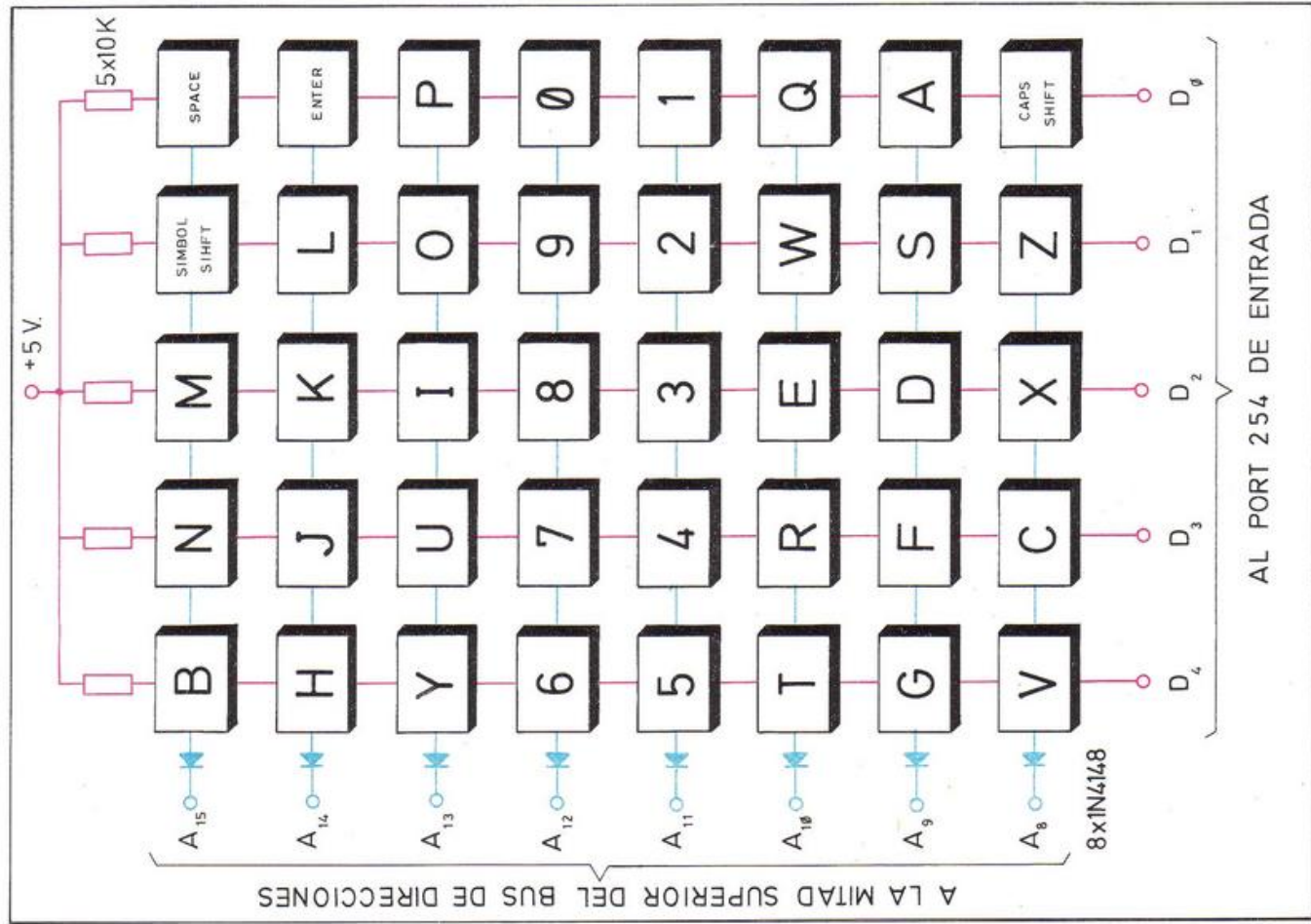


Fig. 12-4. Disposición electrónica del teclado en el Spectrum.

teto procedente del registro «A» se coloca en el bus de datos y se escribe en el dispositivo periférico seleccionado. De nuevo, se utiliza direccionamiento directo para la mitad inferior del bus y direccionamiento indirecto para la mitad superior.

CODIGO DE MAQUINA:

11010011	D3h
← n →	

CURSO12B

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

3

CICLOS DE RELOJ:

11

EJEMPLO:

OUT (46h)A
------------

Contenido del registro «A».

00110000	30h
----------	-----

Instrucción

11010011	D3h
01000110	46h

Bus de direcciones resultante.

00110000	30h
01000110	46h

Valor puesto en bus de datos.

CODIGO MAQUINA 337

01001101	4Dh
----------	-----

01001100	4Ch
----------	-----

01001011	4Bh
----------	-----

Contenido del registro «B» después de la ejecución

00000000	00h
----------	-----

Contenido del par de registros «HL»

10000000	80h
00110010	32h

Indicadores de condición después de la ejecución

S Z H PIV NC

x1xxxx1x
----------

Resultado de la operación. Desde el dispositivo conectado en port 22h han entrado los caracteres ASCII «K», «L», «M», «N» y «O» (4Bh, 4Ch, 4Dh, 4Eh y 4Fh), los cuales han quedado almacenados en las posiciones de memoria 8033h a 8037h en orden inverso.

Instrucciones de salida

OUT (n),A
-----------

OBJETO:

Coloca el valor del operando «n» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro acumulador se coloca en la mitad superior del bus de direcciones. El oc-

10000000	80h
00110111	37h

El contenido de las posiciones de memoria desde 8033h a 8037h no es significativo.

Instrucción

11101101	EDh
10111010	BAh

Primer bus de direcciones resultante.

00000101	05h
00100010	22h

Ultimo bus de direcciones resultante.

00000001	01h
00100010	22h

Valores aparecidos en bus de datos hasta que «B» es cero.

01001011	4Bh
----------	-----

01001100	4Ch
----------	-----

01001101	4Dh
----------	-----

01001110	4Eh
----------	-----

01001111	4Fh
----------	-----

Contenido de los octetos 8033h a 8037h después de la ejecución

01001111	4Fh
----------	-----

01001110	4Eh
----------	-----



00 D7: 0 0 1 1 0 0 0 0 30h

El contenido del registro «A» después de la ejecución no varía.

Resultado de la operación.  
En el dispositivo conectado en el port 46h se ha escrito el carácter ASCII «0» (30h), el cual ya estaba almacenado en el registro acumulador.

### OUT (C),r

OBJETO:

Coloca el contenido del registro «C» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro «B» se coloca en la mitad superior del bus de direcciones. El octeto contenido en el registro representado por «r» se coloca en el bus de datos y se escribe en el dispositivo periférico seleccionado. El código de representación de «r» es el indicado a continuación.

Registro	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

En este caso, el direccionamiento es indirecto para todo el bus. Por su funcionamiento, la instrucción podría haberse llamado, perfectamente: «OUT (BC),r»

CODIGO DE MAQUINA:

338 CODIGO MAQUINA

1 1 1 0 1 1 0 1	EDh
0 1 ← r → 0 0 1	

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

3

CICLOS DE RELOJ:

12

EJEMPLO:

OUT ICLE	2Fh
----------	-----

Contenido del registro «C»:

0 0 1 0 1 1 1 1	2Fh
-----------------	-----

Contenido del registro «B»:

0 0 0 0 1 1 1 1	0Fh
-----------------	-----

Contenido del registro «E»:

0 0 1 1 0 0 0 1	31h
-----------------	-----

Instrucción

1 1 1 0 1 1 0 1	EDh
0 1 0 1 1 0 0 1	58h

Bus de direcciones resultante.

0 0 0 0 1 1 1 1	0Fh
0 0 1 0 1 1 1 1	2Fh

Valor puesto en bus de datos.

0 0 1 1 0 0 0 1	31h
-----------------	-----

Contenido del registro «E» no ha variado después de la ejecución.

Resultado de la operación.  
En el dispositivo conectado

res del bus de direcciones para direccionar todos los periféricos, quedando los tres restantes, libres para el usuario. Los 8 bits superiores se utilizan para suministrar información adicional cuando se lee el teclado. Para cada periférico, se pone a «0» uno de los cinco bits, permaneciendo los restantes a «1». Por ejemplo: si se quiere leer el teclado, se coloca el número 254 (1111110b) en la parte inferior del bus de direcciones. Lo que se coloca en la parte superior, depende de la semi-fila que se desee leer. Sólo uno de estos 8 bits puede ser bajo al mismo tiempo ya que, de lo contrario, se accedería a más de un periférico simultáneamente, lo que podría crear confusión.

El joystick tipo «Kempston», utiliza uno de los tres bits libres para el usuario. Concretamente, el bit A5, por lo que se direcciona en el port 223 (1101111b).  
Vamos a centrarnos, primero, en el estudio del teclado. El teclado del Spectrum está dispuesto, electrónicamente, como una matriz de 8 filas y 5 columnas; en cada «cruce» hay una tecla. En la figura 12.4 se puede ver un esquema eléctrico del mismo. Cada una de las filas de esta matriz, se corresponde con una semi-fila del teclado original tal como aparece su disposición física en la carcasa del ordenador; por tanto, a partir de ahora las llamaremos «semi-filas». En los modelos «Plus» y «128K», existen más de 40 teclas. Esto se debe a que algunas de ellas actúan de forma simultánea sobre dos «cruces» de la matriz.

Las ocho líneas horizontales (color azul) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores

## INSTRUCCIONES DE ENTRADA/SALIDA

MEMONICO	INDICADORES	No. DE BYTES	CICLOS MEM. REL.
IN A, (n)	S . Z . X . H . X . P / V . N . C .	2	3 11
IN r, (C)	↑ . ↑ . X . ↑ . X . P .	2	3 12
INI	X . ↑ . X . X . X . X . 1 .	2	4 16
IND	X . ↑ . X . X . X . X . 1 .	2	4 16
INIR	X . 1 . X . X . X . X . 1 .	2	5(4) 21(16)
INDR	X . 1 . X . X . X . X . 1 .	2	5(4) 21(16)
OUT (n), A	. . X . X . X . X .	2	3 11
OUT (C), r	. . X . X . X . X .	2	3 12
OUTI	X . ↑ . X . X . X . X . 1 .	2	4 16
OUTD	X . ↑ . X . X . X . X . 1 .	2	4 16
OTIR	X . 1 . X . X . X . X . 1 .	2	5(4) 21(16)
OTDR	X . 1 . X . X . X . X . 1 .	2	5(4) 21(16)

### NOTAS:

- Los signos tienen el siguiente significado:  
"V": El indicador cambia de valor de acuerdo con el resultado de la instrucción.  
"X": El bit adquiere un estado indeterminado.  
".": El indicador no es afectado por la instrucción y conserva su anterior contenido.  
"0": El indicador se pone siempre a "cero".  
"1": El indicador se pone siempre a "uno".  
"P": El indicador "P/V" actúa como indicador de paridad.
- La letra "r" indica cualquiera de los registros: "A", "B", "C", "D", "E", "H" ó "L".

Fig. 12-3b. Tabla resumida de indicadores y ciclos de instrucciones de entrada-salida.

das a los ocho bits superiores del bus de direcciones. Los diodos sirven para evitar corto-circuitos entre líneas que podrían dañar al microprocesador. Las 5 líneas verticales (color rojo) están conectadas a los 5 bits inferiores



OUTI: 00110000 30h

El contenido de los octetos A324h a A328h no ha variado con la ejecución  
 Contenido del registro «B» después de la ejecución

IBI: 00000000 00h

Contenido del par de registros «HL»

HL: 10100011 A33h  
 L: 00100011 23h

Indicadores de condición después de la ejecución  
 S Z H PV N C

x 1 x x x x 1 x

Resultado de la operación.  
 En el dispositivo conectado en el port EEh se ha escrito los caracteres ASCII «4», «3», «2», «1» y «0» (34h, 33h, 32h, 31h, y 30h) 49h, los cuales estaban almacenados en las posiciones de memoria A324h a A328h en orden inverso.

### Tablas de codificación

En la figura 12-3 pueden verse las tablas de codificación para las instrucciones de entrada/salida.

### El teclado del Spectrum

Los «ports» (en español «puertos») de entrada/salida son similares a posiciones de memoria, pero que se comunican con periféricos. En la versión básica del Spectrum, sólo existen tres periféricos: El televisor (o monitor), el teclado y el cassette. Para el televisor no se utiliza ningún port, ya que la «ULA» se encarga de leer, directamente,

### INSTRUCCIONES DE ENTRADA/SALIDA

Código Fuente	Hexadecimal	Decimal
IN A, (n)	DB, n	219, n
IN A, (C)	ED, 78	237, 120
IN B, (C)	ED, 40	237, 64
IN C, (C)	ED, 48	237, 72
IN D, (C)	ED, 50	237, 80
IN E, (C)	ED, 58	237, 88
IN H, (C)	ED, 60	237, 96
IN L, (C)	ED, 68	237, 104
INI	ED, A2	237, 162
IND	ED, AA	237, 170
INIR	ED, B2	237, 178
INDR	ED, BA	237, 186
OUT (n), A	D3, n	211, n
OUT (C), A	ED, 79	237, 121
OUT (C), B	ED, 41	237, 65
OUT (C), C	ED, 49	237, 73
OUT (C), D	ED, 51	237, 81
OUT (C), E	ED, 59	237, 89
OUT (C), H	ED, 61	237, 97
OUT (C), L	ED, 69	237, 105
OUTI	ED, A3	237, 163
OUTD	ED, AB	237, 171
OTIR	ED, B3	237, 179
OTDR	ED, BB	237, 187

Fig. 12-3a. Tabla de codificación para instrucciones de entrada-salida.

las posiciones de memoria que contienen la imagen, y enviarla al modulador de video. El acceso al teclado y al cassette se realiza mediante el port 254. El Sistema Spectrum completo, utiliza los 5 bits inferiores de la memoria A324h a A328h.

Indicadores de condición después de la ejecución  
 S Z H PV N C

x 0 x x x x 1 x

Contenido del registro «C».

CI: 00110011 33h

Contenido del registro «B».

BI: 00001011 08h

Contenido del par de registros «HL»

HL: 10100011 A33h  
 L: 00100101 25h

Contenido de la posición de memoria A325h

A325h: 00110010 32h

### Instrucción

OUTI: 11101101 EDh  
 10100011 A33h

Bus de direcciones resultante.

A8-A15: 00001011 08h  
 A0-A7: 00110011 33h

Valor puesto en bus de datos.

D0-D7: 00110010 32h

El contenido del octeto A325h no ha variado con la ejecución  
 Contenido del registro «B» después de la ejecución

BI: 00001010 0Ah

Contenido del par de registros «HL»

HL: 10100011 A33h  
 L: 00100110 25h

INDICADORES DE CONDICION QUE AFECTA:  
 Z; pone 1 - siempre  
 N; pone 1 - siempre

CICLOS DE MEMORIA:

Si «B» diferente de cero  
 5

CICLOS DE RELOJ:

Si «B» diferente de cero  
 21

Si «B» igual a cero  
 16

EJEMPLO:

OTIR

Contenido del registro «C».

CI: 00100000 20h

Contenido del registro «B».

BI: 00000110 08h

Contenido del par de registros «HL»

HL: 10110110 86h  
 L: 01001000 48h

Contenido de las posiciones de memoria desde B648h a B64Dh

B648h: 00110011 33h

B649h: 00110100 34h

B64Ah: 00110101 35h

B64Bh: 00110110 36h

CODIGO MAQUINA:

11101101 EDh  
 10110011 B3h



B64Ch: 37h

tros «HL» después de la ejecución

B64Dh: 33h

Indicadores de condición después de la ejecución

Instrucción

OTIR: 01h 00h

Primer bus de direcciones resultante.

A8A15: 00h 00h

Ultimo bus de direcciones resultante.

A8A15: 00h 00h

Valores puestos en bus de datos hasta que «B» es cero.

OTDR: 33h

OTDR: 34h

OTDR: 35h

OTDR: 36h

OTDR: 37h

OTDR: 38h

Contenido de los octetos B64h a B64Dh no ha variado con la ejecución

Contenido del registro «B» después de la ejecución

OTDR: 00h

Contenido del par de registros «HL»

INDICADORES DE CONDICION QUE AFECTA:

A; pone 1 - si B-1 es igual a cero  
pone 0 - en cualquier otro caso  
N; pone 1 - siempre

CICLOS DE MEMORIA: 4

CICLOS DE RELOJ: 16

EJEMPLO:

OUTD: 00h 00h

Contenido del registro «C».

OTDR: 00h

Contenido del registro «B».

OTDR: 00h

Contenido del par de registros «HL»

OTDR: 00h 00h

Contenido de la posición de memoria F042h

OTDR: 00h 00h

Instrucción

OTDR: 00h 00h

Bus de direcciones resultante.

A8A15: 00h 00h

Valor puesto en bus de datos.

CODIGO DE MAQUINA:

00h 00h

OTDR: 39h

El contenido del octeto 79A3h no ha variado con la ejecución  
Contenido del registro «B» después de la ejecución

OTDR: 00h

Contenido del par de registros «HL»

OTDR: 00h 00h

Indicadores de condición después de la ejecución

OTDR: 00h 00h

Resultado de la operación.  
En el dispositivo conectado en port 79h se ha escrito el carácter ASCII «9» (39h), el cual estaba almacenado en la posición de memoria F042h.

OTDR

OBJETO:

Coloca el contenido del registro «C» en la mitad inferior del bus de direcciones para seleccionar un dispositivo de entrada/salida entre los 256 ports posibles. El contenido del registro «B» se coloca en la mitad superior del bus de direcciones y puede utilizarse como contador de octetos. El octeto de la posición de memoria direccionada por el par de registros «HL» se coloca en el bus de datos y se escribe en el dispositivo de entrada/salida seleccionado. En-

OTDR: 00h 00h

Contenido de las posiciones de memoria desde A324h a A328h

OTDR: 00h 00h

OTDR: 00h 00h

OTDR: 00h 00h

OTDR: 00h 00h

OTDR: 00h 00h

Instrucción

OTDR: 00h 00h

Primer bus de direcciones resultante.

OTDR: 00h 00h

Ultimo bus de direcciones resultante.

OTDR: 00h 00h

Valores aparecidos en bus de datos hasta que «B» es cero.

OTDR: 00h 00h

OTDR: 00h 00h

OTDR: 00h 00h

OTDR: 00h 00h



samblado, una a continuación de la otra, en el buffer de impresora. Veamos el funcionamiento de «TEST»:

Líneas 100 y 110: Cargamos, en «DE», la dirección inicial de la tabla y en «C», el número 40 para que actúe como contador en el bucle donde sacaremos los 40 bits.

Líneas 120 a la 160: Constituyen un bucle que saca un bit del bloque de cinco posiciones de memoria por el procedimiento de rotarlas todas a la izquierda. El bit sale en el indicador de acarreo.

Líneas 170 a la 190: Cargamos un «7» en «A» por si el bit es «1» y seguimos en «PULS» si es así. Si no, cargamos un «56» en «A» y continuamos. Al llegar a la línea 200 («PULS»), «A» contendrá un «7» si el bit era «1» (tecla pulsada) y contendrá un «56» si el bit era «0» (tecla no pulsada).

Línea 200: Pasamos el contenido del puntero «DE» a «HL».

Líneas 210 a la 240: Cargamos, en «DE», la dirección del atributo correspondiente al bit que se está procesando. Al mismo tiempo, incrementamos, 2 veces, el puntero para que quede apuntando a la dirección del siguiente atributo.

Línea 250: Pasamos la di-

rección del atributo a «HL».

Línea 260: Cargamos el valor de «A» en la dirección apuntada por «HL», es decir, en el atributo correspondiente al bit que se estuviera procesando.

Líneas 270 y 280: Decrementamos el contador y salimos a «BUC-4» para procesar el siguiente bit, a menos que el contador pase a «0» indicando que todos los bits se han procesado ya, en cuyo caso, se retorna en la línea 290.

Líneas 300 a la 690: Constituyen la tabla donde están almacenadas las direcciones de los 40 atributos.

Por supuesto, se podría haber eliminado el «RET» de la línea 280 de «TECL-2» y haber sustituido el «RET» de la línea 290 de «TEST» por un «JR TECL-2» con lo que el bucle se cerraría en código máquina. No obstante, hemos preferido hacerlo así y llamar a las rutinas desde Basic, ya que, de lo contrario, una vez entrado en el bucle, sería imposible salir de él.

Quien haya escrito unos cuantos programas en Basic, tal vez esté acostumbrado a crear, de vez en cuando, bucles de los que el programa no salga por sí solo. En Basic,

no representa ningún problema, se pulsa «BREAK» y se detiene el programa. Por desgracia, en Assembler esto no es tan fácil, si el microprocesador se «engancha» en un bucle, no hay quien lo haga salir de él. Por ello, es imprescindible crear condiciones de salida dentro de todo bucle. Una forma que podríamos haber usado es escribir una subrutina que compruebe si las teclas «Caps Shift» y «Space» están pulsadas al mismo tiempo y, en ese caso, detenga la ejecución del bucle y devuelva el control. En cada pasada del bucle, se llamaría a esta rutina y, por tanto, se podría salir del bucle. El tema nos ha parecido tan interesante, que hemos decidido dejarlo para los ejercicios.

Con esto termina el capítulo dedicado a las instrucciones de entrada/salida. Ya sabemos que queda por ver un grupo de instrucciones: las de control de la «CPU». Las veremos en el capítulo próximo, donde también estudiaremos —por fin— qué son y cómo se utilizan las interrupciones.

Antes de ello, le recomendamos, como de costumbre, que intente resolver los siguientes ejercicios.

\*HISOFT GENS3M ASSEMBLER\*  
ZX SPECTRUM

Copyright HISOFT 1983  
CURSO C/M MICROHOBBY

Pass 1 errors: 00

30 \*C-

da (recuerde que asociábamos el estado lógico «1» con una tensión positiva de 5 voltios y el estado «0» con una tensión de 0 voltios).

Cada semi-fila es leída de una vez, por lo que son necesarias 8 lecturas para leer el teclado completo. Supongamos que queremos leer la semi-fila que contiene las teclas «G», «F», «D», «S» y «A». En principio, tenemos que direccionar el port 254 como entrada y, simultáneamente, colocar el número 253 (11111101b) en la parte alta del bus de direcciones. Esto se puede conseguir con: «LD A,253» «IN A,(254)». Con ello, ponemos un «0» (0 voltios) en la línea A9 y un «1» (+5 voltios) en las demás. Si no hubiera ninguna tecla, de esta semi-fila, pulsada, las 5 resistencias harían que las 5 líneas D4 a D0 estuvieran a +5 voltios, es decir, a «1».

Supongamos que está pulsada la tecla «F». En ese caso, la corriente procedente de la segunda resistencia, se fuga a través del séptimo diodo hacia la línea A9 y aparece un «0» en D3. Si estuvieran pulsadas las teclas «F» y «S» simultáneamente, aparecerían dos ceros; uno en D3 y otro en D1. El valor de los bits D5, D6 y D7 depende del tipo de ULA que lleve el ordenador y del estado de la entrada «EAR», por lo que es preferible no tomarlos en cuenta. Lo mejor es poner una máscara cada vez que se lea una semi-fila, por ejemplo: «AND #1F».

Si están pulsadas las teclas «T» y «F», obtendremos un «0» en D4 al leer la semi-fila de «T» a «Q» y otro «0» en D3 al leer la semi-fila de «G» a «A». El teclado del Spec-

trum permite la pulsación de 2 teclas simultáneamente. En el caso de pulsar más de 2, puede ocurrir un problema común a todos los teclados matriciales o de exploración (es el nombre más comúnmente dado a este tipo de teclados).

Supongamos que pulsamos, simultáneamente, las teclas «G», «F» y «V». Al leer la semi-fila correspondiente a A9, la corriente de las dos primeras resistencias, se fugará por el diodo de A9, con lo que aparecerán ceros en D4 y D5 indicándonos que «G» y «F» están pulsadas. Cuando leamos la semi-fila de A8, la corriente de la primera resistencia se fugará por «V» hacia el diodo de A8; pero, también se fugará la corriente de la segunda resistencia a través de los cruces «F», «G» y «V», con lo que aparecerá un «0» en D3 aunque la tecla «C» no estuviera pulsada. Al ocurrir esto, el ordenador interpretará que se ha pulsado la tecla «C», aunque no sea así. Esto ocurre para cualquier combinación de cuatro teclas que ocupen los vértices de un rectángulo. Por ejemplo, si se pulsa «B», «M» y «T», el ordenador interpretará que se ha pulsado, también, «E». En general: cuando se pulsen, simultáneamente, tres teclas que ocupen tres vértices de un rectángulo, el ordenador entenderá que se ha pulsado, también, la que ocupa el cuarto vértice. Esto no ocurrirá, por ejemplo, para las teclas «B», «N» y «K», ya que no están en los cuatro vértices de un rectángulo.

Es muy importante tener en cuenta este efecto cuando se diseñen programas que requieran la pulsación de varias teclas a la vez. El programa

ejemplo de este capítulo nos va a permitir experimentar lo que ocurre con distintas configuraciones de teclas, al pulsarlas simultáneamente.

Ya hemos visto la forma teórica de leer el teclado de código máquina. La forma práctica la veremos en los ejemplos. No obstante, tenemos una rutina en la ROM que funciona en respuesta a la interrupción enmascarable y se encarga de leer el teclado cada 20 milisegundos y anotar en la variable «LAST-K» el código del carácter correspondiente a la última tecla pulsada. A continuación, veremos las restantes señales que afectan a otros periféricos en la versión básica del Spectrum.

La conexión «EAR» también se lee por el port 254 y su valor afecta al bit D6. En los Spectrum que llevan la ULA antigua (5C102 ó 5C112) el valor de este bit, en ausencia de señal, es «1». En los que llevan la ULA 6C0001, este valor suele ser «0», aunque, en algunos de ellos, el ruido producido en el circuito de audio hace que la entrada cambie de estado aleatoriamente (siempre, claro está, en ausencia de señal).

La salida «MIC» se excita con el bit D3 del port 254 configurado como salida. El bit D4 de este mismo port, es el que excita al altavoz. Finalmente, los bits D2, D1 y D0 establecen el color del borde, aunque, éste cambiará al pulsar una tecla desde el Basic.

Como ya dijimos antes, el joystick tipo Kempston se lee por el port 223 (11011111b); el bit D4 es «1» si se ha activado el «disparo», los bits D3 a D0 se ponen a uno para cada una de las direcciones



«arriba», «abajo», «izquierda» y «derecha» respectivamente; las direcciones diagonales harán que se pongan a «1» de estos bits. Visto todo esto, podemos pasar a la práctica con los ejemplos del capítulo.

### Ejemplos

En el capítulo anterior, vimos un programa de ejemplo bastante largo. Como suponemos que el lector se cansaría de teclearlo, esta vez los ejemplos serán más cortos. Aunque, no por ello, menos interesantes (al menos, así lo esperamos).

La primera de las rutinas que hemos preparado, ilustra la forma de leer una semi-fila cualquiera del teclado. Entramos en ella con un determinado valor en el registro «A». que nos va a determinar la semi-fila que leemos. A la salida, «A» contendrá un valor que estará en función de la tecla que hubiera pulsada. Veamos el listado:

```
100 TECL_1 CPL
110 IN A, 254
120 CPL
130 AND #1F
140 RET
```

Los valores de entrada serán los siguientes según la semi-fila a leer:

Valor de «A»	Semi-fila leída
1	«V» + «7/5»
2	«5» + «7»
4	«7» + «7»
8	«5» + «7»
16	«5» + «7»
32	«7» + «7»
64	«7» + «Enter»
128	«7» + «Space»

obstante, he aquí el código objeto:

```
1 230FE7E0FCF00000 1829
```

Por supuesto, es perfectamente reubicable. En la figura 12-5 se puede ver el listado completo de esta rutina. Nosotros la hemos ensamblando en el buffer de impresora.

El segundo de los ejemplos del capítulo es una rutina que nos va a permitir leer el teclado de una sola vez y saber qué tecla o teclas hay pulsadas, aunque haya más de una. Exploraremos todas las semi-filas del teclado y colocaremos un «1» por cada tecla que haya pulsada en las posiciones de memoria 23296 a la 23300. Cada bit de cada una de estas posiciones se corresponde con una tecla, de forma que bastará leer el bit en cuestión para saber si la tecla está pulsada. La correspondencia entre bits y teclas viene dada por la siguiente tabla:

DIRECCION	BIT: 7 6 5 4 3 2 1 0
23296	V C 1 2 7 5 6 F 0
23297	S A T R E M 0 5
23298	4 3 2 1 6 7 8 9
23299	# Y U I O P H 2
23300	K L ENT B N S S SP

El procedimiento a seguir será ir leyendo, una a una, todas las semi-filas y metiendo los 5 bits de cada una en el grupo de posiciones de memoria. Para ello, rotaremos el registro «A» a la izquierda y, a continuación, las cinco posiciones de memoria, con lo que la transferencia se realizará a través del indicador de acarreo. El registro «A» será

En la línea 100, comentamos el contenido de «A» para que sea «0» el bit correspondiente a la semi-fila a leer y «1» los bits restantes.

En la línea 110 se coloca el contenido de «A» en la parte alta del bus de direcciones y «254» en la parte baja, se hace una entrada desde este port y el dato entrante se escribe en «A».

En la línea 120 se comenta el contenido de «A» para que sean «1» los bits correspondientes a la(s) tecla(s) que hubiera pulsada(s) y «0» los bits restantes.

En la línea 130, se pone una máscara para dejar a «0» los tres bits superiores cuyo estado depende de factores bastante aleatorios, y que podrían crear confusión.

Finalmente, la línea 140 se encarga de retornar a la rutina desde donde se llamará a ésta.

En el retorno, el valor de «A» dependerá de la tecla que hubiera pulsada, según la siguiente tabla:

Valor de «A»	Tecla pulsada
1	«Space» + «7/5»
2	«5» + «7»
4	«7» + «7»
8	«5» + «7»
16	«5» + «7»

Se pueden obtener sumas de valores si hay dos o más teclas pulsadas. Por ejemplo: entramos en la rutina con «A» conteniendo un «4» y salimos con «A» conteniendo «17»; esto quiere decir que estaban pulsadas las teclas «T» y «Q».

La rutina ocupa, sólo 7 bytes y su ensamblo no debe presentar problemas, no

```
*HISOFT GEN3M ASSEMBLER*
ZX SPECTRUM
```

```
Copyright HISOFT 1983
CURSO C/M MICROHOBBY
```

```
Pass 1 errors: 00
```

```
30 *C-
40 *D+
50 ;
60 ; RUTINA: "TECLADO_2"
70 ;
80 ;
23301 90 ORG 23301
23301 100 TECL_2 LD BC, #FEFE
23304 110 BUC_1 IN A, (C)
23306 120 CPL
23307 130 SLA
23309 140 SLA
23311 150 SLA
23313 160 LD E, 5
23315 170 BUC_2 LD HL, 23296
23318 180 SLA
23320 190 LD D, 5
23322 200 BUC_3 RL (HL)
23324 210 INC HL
23325 220 DEC D
23326 230 JR NZ, BUC_3
23328 240 DEC D
23329 250 JR NZ, BUC_2
23331 260 RLC B
23333 270 JR C, BUC_1
23335 280 RET
```

```
Pass 2 errors: 00
```

```
Table used: 62 from 138
```

Fig. 12-8. Listado completo del «TECL-2».



[illegible]

No se queda aquí la utilidad de este programa. Quienes aspiren a mecanógrafos, encontrarán en él una magnífica ayuda para aprender a teclear sin mirar al teclado.

350 CODIGO MAQUINA

[illegible]

Para saber qué dirección del archivo de atributos corresponde a cada bit, hemos optado por un método bastante fácil, pero muy efectivo. Creamos una tabla de 80 bytes que contiene las direcciones de los atributos de cada carácter (2 bytes por cada dirección), colocadas en el mismo orden en el que van saliendo los bits del bloque de 5 posiciones de memoria. De esta forma, un puntero puede ir avanzando por la tabla y apuntando, siempre, al lugar donde está almacenada la dirección del atributo correspondiente al bit que se está procesando.

En la figura 12-9 se puede ver el listado de la rutina «TEST» y en la 12-8 el de la rutina «TECL-2». Las hemos en-

250 JR NZ,BUC\_2  
260 PLC B  
270 JR C,BUC\_1  
280 RET

Lo que vamos a hacer es leer, secuencialmente, cada semi-fila del teclado e ir guardando, de cada vez, el contenido de «A» en las cinco posiciones de memoria 23296 a la 23300. La forma de hacer

donde obteníamos el dato de cada semi-fila y habrá que rotarlo tres lugares a la izquierda, para que los bits que nos interesan se coloquen en los 5 bits superiores antes de iniciar la cadena de rotaciones.

Como complementamos el registro «A» después de leer cada semi-fila, cada bit de las 5 posiciones 23296 a 23300 será «1» si la tecla estaba pulsada y «0» si no lo estaba. Veamos el listado de la rutina:

```

100 TSC1_2 LD B0, #FFFF
110 BUC_1 IN A, (C)
120 CPL
130 SLA A
140 SLA A
150 SLA A
160 LD E, 5
170 BUC_2 LD HL, 23296
180 SLA A
190 LD D, 5
200 RL (HL)
210 INC HL
220 DEC D
230 JR NZ, BUC_3
240 DEC E
250 JR NZ, BUC_2
260 RLC B
270 JR C, BUC_1
280 RET

```

Fig. 12-5. Listado de la rutina «TECL-1».

esto último es rotar a la izquierda el contenido de «A» para que los bits vayan saliendo al indicador de acarreo y rotar, también a la izquierda, las cinco posiciones de memoria para que el indicador de acarreo vaya entrando en ellas.

Hay, por tanto, tres bucles anidados uno dentro de otro. El más exterior es «BUC-1», tiene 8 iteraciones y se encarga de ir leyendo cada semilla, complementar el valor de «A» y rotarlo tres veces a la izquierda para «quitar» los tres bits que no nos interesan; este bucle no utiliza variable de estado y pertenece a las li-



neas: 170, 180, 240 y 250; utiliza como contador el registro «E» cuyo valor se inicializa en la línea 160 y contiene, en su interior, al bucle «BUC-3».

El bucle más interior es «BUC-3» que se encarga de meter el indicador de acarreo en el bloque formado por las cinco posiciones de memoria, rotando éstas a la izquierda para que la entrada sea secuencial. El primer bit que entra por la derecha del bloque acabará, tras 40 rotaciones, en la izquierda. Este bucle tiene 5 iteraciones y está controlado por el registro «D» cuyo valor se inicializa en la línea 190. Pertenecen a «BUC-3» las líneas 200, 210, 220 y 230.

Al leer cada semi-fila, serán cinco bits los que entren en el bloque de 5 octetos. Tras la lectura de las ocho semi-filas, habrán entrado  $5 \times 8 = 40$  bits, es decir, uno por cada tecla y el bloque estará completo. Como la primera semi-fila que se lee es la correspondiente a las teclas «V» a «C/S», el primer bit que entrará será el correspondiente a la tecla «V» que acabará, por tanto, a la izquierda del bloque, es decir, en el bit 7 de 23300. La última semi-fila que se lee es la correspondiente a las teclas «B» a «SP»; por tanto, el último bit que entra es el correspondiente a la tecla «SP» (barra espaciadora) y quedará situado a la derecha del bloque, es decir, en el bit 0 de 23296.

Podría parecer que el procedimiento resulta extremadamente lento, ya que el bucle «BUC-3» se ejecuta  $5 \times 5 \times 8 = 200$  veces, el «BUC-2» 40 veces y el «BUC-1» 8 veces. Lo cierto es que todo se produce a tal velocidad que se puede consi-

derar instantáneo. El lector tendrá ocasión de comprobarlo cuando apliquemos esta rutina en el programa-ejemplo.

Ya tenemos una rutina que nos lee todo el teclado a la vez, localidad del rayo y nos permite pulsar todas las teclas que queremos simultáneamente. Ahora, vamos a ver qué podemos hacer con ella. La utilidad más inmediata de esta rutina es usarla en un juego que requiera la pulsación de varias teclas. Cada opción del juego no tendrá más que comprobar un bit determinado de una posición de memoria para ver si se ha pulsado determinada tecla, y actuar en consecuencia.

No obstante, a nosotros se nos ha ocurrido una idea más vistosa. Se trata de hacer una representación, en pantalla, del teclado del Spectrum y visualizar la pulsación de cada tecla, incluso si se pulsaran varias a la vez.

Representamos cada tecla con su letra y ocupando la posición que ocupa en el teclado. Para las teclas «Caps Shift», «Symbol Shift», «Enter» y «Space», creamos cuatro caracteres gráficos (UDGs). Mientras una tecla permanece pulsada, el carácter que la representa pasa a «video inverso», volviendo a su posición normal cuando se suelta la tecla. Si se pulsaran varias teclas a la vez, serán varios los caracteres que se muestran en «video inverso».

El programa 12-1, que hemos denominado «Prueba de teclado», se encarga de todo esto. Si lo desea, puede teclearlo ahora (antes de seguir leyendo), ya que funciona por sí solo.

Suponemos que ya ha te-

clado el programa. Póngalo en marcha con «RUN». Lo primero que aparecerá es el mensaje: «ESPERE 10 SEGUNDOS. Por Favor». Estos 10 segundos corresponden al tiempo que tarda el programa en leer el código máquina y los UDGs desde las líneas «DATA».

A continuación, le habrá salido una pantalla que contiene los 40 caracteres que representan el teclado del ordenador. Pulse cualquier tecla y manténgala oprimida. El carácter correspondiente pasará a «video inverso». Ahora, suéltela. El carácter volverá a video normal.

Puede experimentar pulsando varias teclas diferentes, incluso, varias a la vez. En algunos casos, observará que pasan a «video inverso» algunos caracteres que corresponden a teclas no pulsadas. No es que el programa funcione mal. Se trata del «efecto de matriz» que comentábamos al explicar la disposición del teclado. Vamos a estudiarlo.

Pulse las teclas «Q», «E» y «A» simultáneamente. Observará que también se ha activado el carácter correspondiente a la tecla «D». Ahora, pulse «Y», «B» y «O». Se activará el carácter correspondiente a «Symbol Shift». Mire si es capaz de encontrar más combinaciones de teclas donde ocurra esto.

Cada vez que pulse tres teclas que se encuentren en los vértices de un rectángulo, la tecla del cuarto vértice también se activará. Es importante que tenga en cuenta este efecto cuando diseñe programas que requieran la pulsación de varias teclas simultáneamente. Este

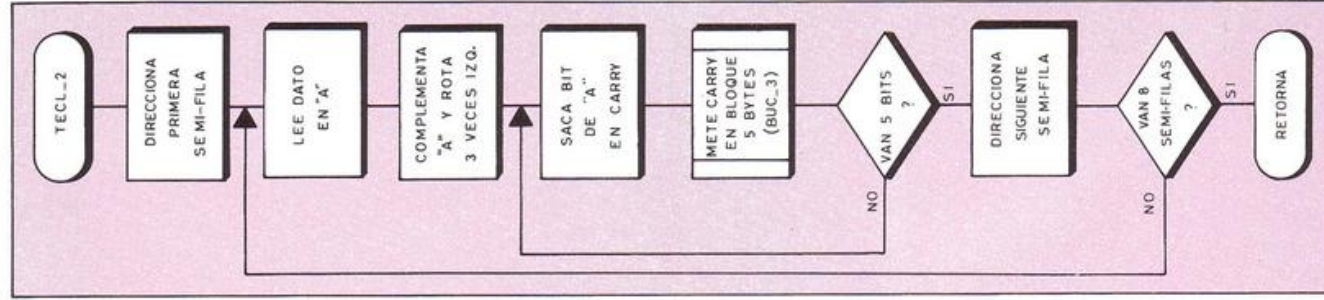


Fig. 12-6. Organigrama de la rutina «TELL-2».

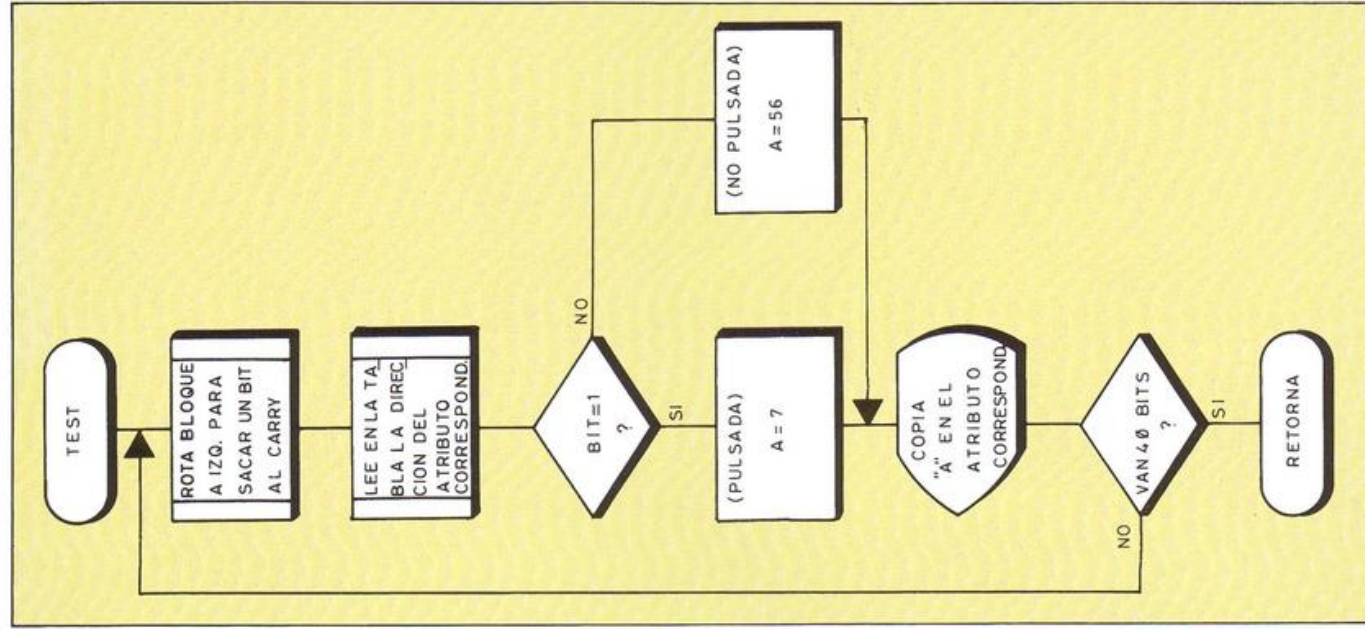


Fig. 12-7. Organigrama de la rutina «TEST».







23398	450	DEFW	22574
23400	460	DEFW	22571
23402	470	DEFW	22568
23404	480	DEFW	22565
23406	490	DEFW	22562
23408	500	DEFW	22577
23410	510	DEFW	22580
23412	520	DEFW	22583
23414	530	DEFW	22586
23416	540	DEFW	22589
23418	550	DEFW	22673
23420	560	DEFW	22676
23422	570	DEFW	22679
23424	580	DEFW	22682
23426	590	DEFW	22685
23428	600	DEFW	22769
23430	610	DEFW	22772
23432	620	DEFW	22775
23434	630	DEFW	22778
23436	640	DEFW	22781
23438	650	DEFW	22865
23440	660	DEFW	22868
23442	670	DEFW	22871
23444	680	DEFW	22874
23446	690	DEFW	22877

Pass 2 errors: 00

Table used: 71 from 210

Fig. 12-9. Listado completo de «TEST».

programa, que decida no co-  
ger el teléfono aunque suene.  
En ese caso, el dispositivo no  
será atendido aunque solici-  
te interrupción.

El Z-80 permite habilitar y  
dehabilitar interrupciones  
desde el programa. Cuando  
están deshabilitadas las inte-  
rpciones, una petición de  
interrupción enmascarable  
será ignorada, exactamente  
como si no se hubiera produ-  
cido.

La interrupción enmascara-  
ble está pensada para aten-  
der periféricos principalmen-  
te. Esta interrupción tiene tres  
modos de respuesta que tie-  
nen que ser habilitados por el  
programa en curso o por el  
sistema monitor. Las interrup-  
ciones enmascarables pue-  
den ser deshabilitadas por el  
programador mediante la ins-  
trucción «DI» y habilitadas  
mediante la instrucción «EI».  
Cuando el Z-80 recibe la se-  
ñal de «RESET», arranca con  
las interrupciones habilita-  
das.

Los tres modos de interrup-  
ción son: «modo 0», «modo 1»  
y «modo 2». Se pueden selec-  
cionar por programa con las  
instrucciones «IM 0», «IM 1»  
e «IM 2». Cuando se recibe  
una señal de «RESET», el Z-80  
arranca en «modo 0».

#### MODO 0

Cuando se produce una in-  
terrupción es este modo, el  
periférico deberá colocar, en  
el BUS de datos, un octeto  
que será una instrucción de  
RESTART de página cero. Por  
tanto se pasará el control a  
una de las ocho primeras po-  
siciones de memoria vistas  
en la instrucción «RST».

#### MODO 1

Cuando se produce una in-  
terrupción en este modo, se  
salta a la dirección de memo-  
ria 0038h, donde existirá una  
rutina para tratar esta inte-  
rrupción.

#### MODO 2

Este es el modo más poten-  
te. Cuando se produce una in-  
terrupción en este modo se  
construirá una dirección con  
el valor del registro «I» y el va-  
lor que deje el periférico en el  
BUS de datos. El registro «I»  
será la parte más significati-  
va y el BUS de datos la me-  
mos. Este valor se tomará co-  
mo una dirección de memoria  
y desde ella y la siguiente, le-  
erá la verdadera dirección a  
donde hay que saltar. El con-  
tenido del registro «I» se car-  
ga por programa como el de  
cualquier otro (instrucción  
«LD I,A»). Vamos a verlo más  
claro con un ejemplo:

Supongamos que seleccio-  
namos el «modo 2» de inte-  
rrupción y cargamos «7Eh» en  
el registro «I». Cuando se pro-  
duzca una petición de inte-  
rrupción, el Z-80 formará una  
dirección con el registro «I» y  
el bus de datos. Suponga-  
mos, también, que nuestro  
dispositivo no inserta nada en  
el bus de datos. En ese caso,  
la dirección formada sería  
«7EFFh» (el bus de datos es-  
tá a «FFh»). Previamente, ha-  
bremos metido en las direc-  
ciones «7EFFh» y «7F00h»  
unos determinados datos.  
Esos datos pueden ser «6Ah»  
y «C3h» respectivamente. En  
ese caso, el microprocesador  
saltaría a la dirección  
«C36Ah».

Vemos que, por este méto-  
do, podemos colocar una ru-  
tina que se ejecute por inte-  
rrupción, en cualquier lugar

de la memoria y direccionar-  
la de forma indirecta en «mo-  
do 2».

Para poder controlar de for-  
ma adecuada la inhibición o  
desbloqueo de las interrup-  
ciones la CPU tiene dos flags  
IFF1 e IFF2. El valor de IFF1,  
que sólo puede ser «0» ó «1»,  
controla el permiso para que  
se produzca una interrupción  
enmascarable. «0» no las per-  
mite y «1» las permite. IFF2 es  
un reflejo de IFF1, siempre  
que una instrucción modifica  
uno, el otro también es modi-  
ficado, excepto cuando se  
produce una interrupción no  
enmascarable. Cuando esto  
ocurre lo primero que se ha-  
ce es poner a «0» IFF1 para  
evitar que se dé una interrup-  
ción enmascarable, cuando  
se vuelve de la interrupción  
no enmascarable con la ins-  
trucción RETN, se copia el va-  
lor de IFF2 sobre IFF1 para re-  
cuperar el valor anterior.

#### Las interrupciones en el Spectrum

El microordenador Spec-  
trum no utiliza toda la poten-  
cia del Z-80 en materia de in-  
terrupciones. La primera ca-  
racterística es que la rutina  
de la posición de memoria  
0066h está preparada para re-  
tornar directamente, o saltar  
a la dirección 0000h, en fun-  
ción de que el contenido de  
las direcciones 23728 y 23729  
sea, o no, «cero», con lo que  
produce un RESET, esto es,  
pone a «0» toda la memoria e  
inicializa las variables del sis-  
tema. Resumiendo la interrup-  
ción no enmascarable está  
bloqueada por software.

Echémole un vistazo a la  
rutina, del sistema operativo  
que responde a la interrup-  
ción no enmascarable:



dispositivo de alta prioridad. La filosofía de las interrupciones está pensada, principalmente, para atender de forma adecuada las prioridades dentro de un ordenador. Cuando un programa tiene el control de la CPU no existe, en principio, nada que se lo pueda quitar... salvo una interrupción.

Supongamos que existe otra actividad que requiera ser atendida de forma inmediata, como es, por ejemplo, una entrada desde un periférico. La forma más utilizada es la interrupción. Por medio de una señal de interrupción la CPU sabe que existe un requerimiento de mayor prioridad que tiene que atender, en ese caso guarda todos los controles necesarios para devolver el control, en su momento, a la actividad en curso y pasa a atender la actividad que genere la interrupción. Una vez atendida dicha interrupción se devolverá, directa o indirectamente, el control a la actividad interrumpida.

El sistema que utiliza el Z-80 para quitar y devolver el control es la pila de máquina. Cuando ocurre una interrupción en el Z-80, la CPU guarda, en la pila de máquina, el contenido del registro «PC» y coloca, en este registro, la dirección que corresponda al tratamiento de la interrupción, la salida de esta rutina tendrá que ser con una de las instrucciones de retorno (RET) para tomar de la pila de máquina la dirección donde se interrumpió el programa.

En el micro-procesador Z-80 existen dos tipos de interrupciones: interrupción no enmascarable (NMI) e interrupción enmascarable (INT).

No se ha modificado ninguna posición de memoria ni registro.

Observe que el resultado de la ejecución de esta instrucción es el mismo que el de la instrucción «NOP», la diferencia está en la forma de salir de ellas. Para salir de un ciclo de instrucciones «NOP» tendrá que tener en cuenta el programador el tiempo que desea estar parado y ejecutar tantas instrucciones como desee, mientras que para salir de una instrucción «HALT» es necesaria una interrupción. Con un «RESET» también se sale pero reinicializando el ordenador, con lo que se pierde el propio programa que ejecutó la instrucción.

### Las interrupciones

Supongamos que está usted tecleando un programa en el Spectrum. En ese momento, suena el teléfono. Lo más probable es que interrumpa lo que está haciendo y se acerque a atender la llamada. El teléfono es un «dispositivo de alta prioridad». Hay que atenderle en el momento. No puede esperar.

Una vez que haya terminado de atender la llamada, recordará lo que estaba haciendo, en el punto donde lo dejó. La llamada de teléfono ha provocado una «interrupción» en su actividad. El dispositivo ha sido atendido inmediatamente y, luego, se ha retornado a la actividad principal.

De forma similar, el micro-procesador Z-80 puede interrumpir su actividad principal para atender la petición de interrupción desde un

### Interrupción no enmascarable

Esta interrupción está pensada para necesidades urgentes y no puede ser deshabilitada por el programa. Trabaja, por tanto, con prioridad absoluta.

Cuando se produce una interrupción no enmascarable, la CPU carga, en el registro «PC», la dirección de memoria 0066h. En esta dirección tendrá que existir una subrutina para tratar esta interrupción y el retorno de este tratamiento será por medio de la instrucción RETN (retorno de interrupción no enmascarable). Esta rutina sólo puede ser interrumpida por otra interrupción no enmascarable que recomenzaría la ejecución de la misma. Es decir, si se está en una rutina de servicio a una interrupción no enmascarable, las peticiones de interrupción enmascarable no serán atendidas.

Un simil que puede darnos idea de lo que es la interrupción no enmascarable, es el siguiente: supongamos que, mientras está tecleando el programa, se da cuenta de que se ha declarado un incendio en su casa. Con toda seguridad, dejará lo que está haciendo y acudirá a apagar el fuego. Es más, si, mientras está apagando el fuego, suena el teléfono, lo más probable es que no atienda la llamada. Una llamada de teléfono no es una interrupción de más baja prioridad que un incendio.

### Interrupción enmascarable

Puede ocurrir que tenga tanta prisa por terminar su

## EJERCICIOS

10) Escriba una subrutina que compruebe si se han pulsado, simultáneamente, las teclas «Caps Shift» y «Space». La rutina deberá retornar con el acarreo a «0» si ambas teclas están pulsadas y, con el acarreo a «1», en cualquier otro caso.

20) ¿Qué semi-fila del teclado leemos con la siguiente subrutina?

```
LD A,251
IN A,(254)
AND 31
RET
```

30) Tenemos una lista de 256 códigos en el buffer de impresora, y queremos enviarlos por el port 223. Escriba una subrutina que realice esta tarea. (Nuestro periférico no decodificará los 8 bits superiores del bus de direcciones, por lo que su contenido nos es indiferente).

----- 0 -----

## SOLUCION A LOS EJERCICIOS

30) La rutina podría ser:

```
SEND LD BC,223
LD HL,23296
OTIR
RET
```

Tan sencilla como parece, se trata de un claro ejemplo de la instrucción «OTIR». Primero cargamos «223» en «BC» para que «C» contenga «223» (la dirección del port) y «B» contenga «9». Recuerde que «9» es el contador de octetos, por lo que contiene «0» para 256 iteraciones. A continuación direccionamos el bloque de datos con «HL» y, todo lo demás, lo hace la instrucción «OTIR».

----- 0 -----



# SOLUCIONES A LOS EJERCICIOS

10) La rutina podría ser algo así:

```
TS_BRK LD A,#7F ;Semi-fila "g" a "SPACE".
IN A,(#FE) ;Lee el dato en "A".
RRA ;Rota el bit "g" al acarreo.
RET C ;Si es "g", retorna.
LD A,#FE ;Semi-fila "y" a "CAPS/SHIFT".
IN A,(FE) ;Lee el dato en "A".
RRA ;Rota el bit "g" al acarreo.
RET ;Retorna.
```

Lo cierto es que no hemos inventado nada. Esta rutina es, exactamente, la que utiliza el intérprete de Basic para leer la tecla "BREAK" después de ejecutar cada comando. La subrutina se denomina "BREAK\_KEY" y se encuentra en la dirección 1F54h (8020). Por tanto, cada vez que se hace un "CALL #1F54" la rutina devuelve el indicador de acarreo a "0" si están pulsadas las teclas de "BREAK".

20) La configuración que formamos en la parte alta del bus de direcciones es 251 (1111011b), por lo que, el bit que se pondrá a "0" es el A10 correspondiente a la semi-fila "Y" a "Q". Será, por tanto, esta semi-fila la que leamos.

## GRUPO DE INSTRUCCIONES DE CONTROL DE CPU

En este grupo de instrucciones se engloban todas aquellas que actúan directamente sobre la unidad de control de proceso (CPU) del micro-procesador. La mayor parte de estas instrucciones se refieren a las interrupciones, lo que tiene que tener en cuenta el lector es que no todas las facilidades del micro-procesador Z-80 están implementadas en el ordenador SPECTRUM.

En una primera pasada por todas las instrucciones veremos lo que hacen independientemente de las limitaciones puestas por Sinclair, las cuales se señalarán posteriormente.

Con este capítulo terminamos la parte del curso dedicada al funcionamiento de las instrucciones, esto es el repertorio de instrucciones del «Z-80 queda concluido.

### NOP

OBJETO:

El procesador central no realiza ninguna operación durante el tiempo de ejecución de esta instrucción.

CODIGO DE MAQUINA:

00000000 00h

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

NOP

Ningún valor de registros o memoria es significativo

Instrucción

NOP: 00000000 00h

No se ha modificado ninguna posición de memoria ni registro.

Observe que lo único que ha ocurrido con la ejecución de esta instrucción es que se ha cumplido un ciclo de memoria y han transcurrido 4 ciclos de reloj. Este es el uso que tiene esta instrucción es una instrucción para «perder el tiempo», pues es la única manera de tener parado el ordenador sin afectar a sus funciones internas.

HALT

Ningún valor de registros o memoria es significativo

Instrucción

CODIGO DE MEMORIA:

01101100 70h

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

1

CICLOS DE RELOJ:

4

EJEMPLO:

HALT



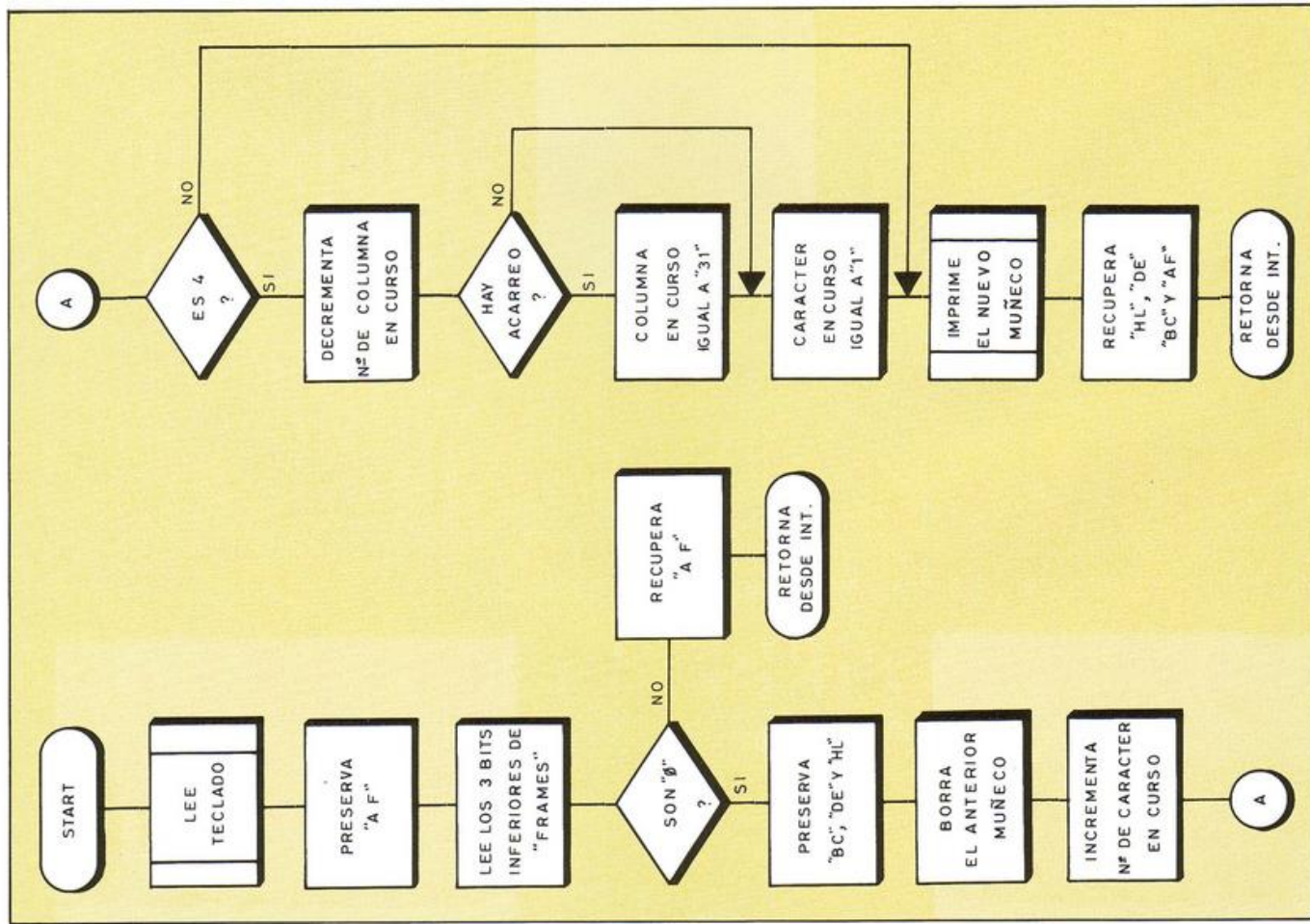


Fig. 13-4. Organigrama de la rutina que mueve el muñeco.

## IM 0

OBJETO:

Activa el modo de interrupción 0.

Cuando se produce una interrupción enmascarable con este modo activo, se ejecuta la instrucción que el periférico coloca en el BUS de datos.

CODIGO DE MAQUINA:

11101101	EDh
01000110	46h

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

8

EJEMPLO:

IM 0
------

Instrucción

IM 0:	11101101	EDh
	01000110	46h

Contenido del BUS de datos cuando se produce una interrupción.

11101101	D7h
----------	-----

La CPU pasa control a la rutina codificada a partir de la posición de memoria 0010h, debido a que el valor del BUS de datos corresponde a la instrucción: «RST #10».

## IM 1

OBJETO:

Activa el modo de interrupción 1.

Cuando se produce una interrupción enmascarable con este modo activo, la CPU pasa control a la rutina codificada a partir de la posición de memoria 0038h.

CODIGO DE MAQUINA:

11101101	EDh
01010110	58h

INDICADORES DE CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

8

EJEMPLO:

IM 1
------

Instrucción

IM 1:	11101101	EDh
	01010110	58h

Cuando se produzca una interrupción enmascarable la CPU pasará el control a la rutina codificada a partir de la posición de memoria 0038h.

11111111	FFh
----------	-----

Contenido de las posiciones de memoria 8AFFh y 8B00h:  
8AFFh: 10100000 Ah  
8B00h: 10111100 BCh

Cuando se produce una interrupción enmascarable con este modo activo, se ejecuta la rutina codificada a partir de la dirección almacenada en las dos posiciones de memoria cuya dirección más baja resulta de tomar el registro «I» como la parte más significativa y el BUS de datos como la parte menos significativa.

CODIGO DE MAQUINA:

11101101	EDh
01011110	5Ch

INDICADORES DE

CONDICION QUE AFECTA:

Ninguno

CICLOS DE MEMORIA:

2

CICLOS DE RELOJ:

8

EJEMPLO:

IM 2
------

Contenido del registro «I».

10001010	8Ah
----------	-----

Instrucción

IM 2:	11101101	EDh
	01011110	5Ch

Contenido del BUS de datos cuando se produce una interrupción

11111111	FFh
----------	-----

Contenido de las posiciones de memoria 8AFFh y 8B00h:

8AFFh:	10100000	Ah
8B00h:	10111100	BCh



La CPU pasa el control a la rutina codificada a partir de la posición de memoria BCA00h.

### Tablas de codificación

Dado que las rutinas de control de la CPU no afectan a los indicadores del registro «F», no hemos representado la habitual tabla resumida de indicadores y ciclos. Las instrucciones «NOP», «HALT», «DI» y «EI» ocupan un byte y emplean 1 ciclo de memoria, es decir, 4 de reloj. Las «IM 0», «IM 1» e «IM 2» ocupan dos bytes y emplean 2 ciclos de memoria y 8 de reloj.

En la Figura 13-1 se encuentra la tabla de codificación de estas instrucciones.

### Ejemplos

Tal vez lo más complicado de este capítulo sea el manejo de las interrupciones, concretamente, del «Modo 2» que es el que podemos utilizar para nuestros fines. Por ello, la rutina que hemos preparado funcionará como una rutina de respuesta a la interrupción enmascarable. Con esto, conseguiremos que se ejecute de forma, aparentemente, simultánea a cualquier otra tarea que esté realizando el ordenador. Siempre, claro está, que no se deshabiliten las interrupciones.

Antes de ver nuestra rutina, sería conveniente echar una mirada a la rutina de la ROM que se encarga, habitualmente, de responder a la interrupción enmascarable.

Desde que se conecta el ordenador, la ULA se encarga de poner a «0» la pata 16 del microprocesador, una vez cada 20 milisegundos, es decir,

## INSTRUCCIONES DE CONTROL DE CPU

Código	Fuente	Hexadecimal	Decimal
NOP		00	0
HALT		76	118
DI		F3	243
EI		FB	251
IM 0		ED, 46	237, 70
IM 1		ED, 56	237, 86
IM 2		ED, 5E	237, 94

Fig. 13-1. Tabla de codificación para las instrucciones de control de CPU.

50 veces por segundo. Esto ocurre, exactamente, en el momento en que se va a empezar a barrer una pantalla en el televisor. Dicho de otra forma, la señal de sincronismo de cuadro y la petición de interrupción se producen al mismo tiempo.

Durante la rutina de inicialización (después de un RESET o un comando «NEW»), durante las rutinas de cassette (comandos «SAVE», «LOAD», «VERIFY» y «MERGE») y durante la ejecución de un comando «BEEP», las interrupciones se encuentran deshabilitadas. En la inicialización se hace así, porque las variables del Sistema no están preparadas para procesar una interrupción y, en el resto de las ocasiones, porque el tiempo empleado en responder a la interrupción distorsionaría la temporización de señales que requieren estas rutinas.

Pero, ¿cómo conseguiremos que la rutina se ejecute una sola vez cada 8 interrupciones? Escribiremos la rutina de forma que sea «transparente» al Sistema, es decir, seguiremos leyendo el teclado y actualizando el contador «FRAMES» (con una llamada a «RST #38»). Como «FRAMES» se actualizará en cada interrupción, sus tres bits inferiores sólo serán «0» simultáneamente, una vez cada 8 interrupciones. Nuestra rutina leerá estos tres bits y sólo se ejecutará cuando los tres sean, simultáneamente, cero. Vayamos viendo como empuja la rutina:

```

290 START RST #38
300 PUSH AF
310 LD A, (FRAMES)
320 AND #07
330 JR Z, SIGUE
340 POP AF
350 RETI
360 SIGUE ....

```

Lo primero que hacemos es una llamada a «RST #38» para leer el teclado y actualizar «FRAMES». A continuación, preservamos el registro «A» para poder utilizarlo en esta parte de la rutina. Cargamos en «A» el contenido del octeto inferior de «FRAMES» y le hacemos un «AND» con el número 7 (00000111b) para aislar los tres bits inferiores. Si el resultado del «AND» es «0», saltamos a «SIGUE» para ejecutar la rutina. En caso contrario, recuperamos el contenido de «A» y retornamos desde interrupción.

Dado que empujamos por el «RST #38», el resto de la rutina se ejecuta con las inte-

rupciones habilitadas. No hay problema por ello, ya que el tiempo de ejecución es muy inferior al tiempo entre interrupciones. Podríamos haber puesto el «RST #38» al final de la rutina, pero habríamos tenido que ponerlo dos veces, puesto que la rutina tiene dos finales. Uno, si no se ejecuta, en la línea 350, el otro, al final de la rutina que empieza en «SIGUE».

Un problema que debemos plantearnos es la forma de imprimir y borrar nuestro muñeco. No podemos utilizar la rutina de la ROM (RST #10), ya que alteráramos la posición de impresión y el canal en curso (si éste fuera distinto del #2). Por ello, nos veremos obligados a desarrollar una pequeña rutina de impresión y otra de borrado. Como siempre trabajaremos sobre la línea 21 de la pantalla, la dirección del archivo de presentación visual será siempre: 01010000 101xxxx donde «xxxx» representa la columna donde se encuentre el muñeco que puede ir desde «1111» (31) hasta «0000» (0).

El procedimiento seguido para mover el muñeco será:

- 1.º) Borrar la posición correspondiente a la columna en curso.
- 2.º) Incrementar el n.º de carácter.
- 3.º) Si éste pasara a valer «4», ponerlo a «0» y decrementar el n.º de columna.
- 4.º) Si ésta pasara a valer «255» (—1), ponerla a «31».
- 5.º) Imprimir el nuevo muñeco en la nueva columna.
- 6.º) Retornar.

Antes de todo esto, habrá que preservar los registros que vayamos a utilizar y recuperarlos antes de retornar.

Veámoslo en Assembly:

```

360 SIGUE PUSH BC
370 PUSH DE
380 PUSH HL
390 LD A, (60157)
400 LD C, A
410 LD H, #50
420 LD A, #A0
430 OR C
440 LD L, A
450 XOR A
460 LD B, 8
470 BUC_1 LD (HL), A
480 INC H
490 DJNZ BUC_1

```

Entre 360 y 380 preservamos los registros. En 390 y 400 pasamos a «C» la columna en curso. Entre 410 y 440 componemos, en «HL» la dirección de pantalla correspondiente a esta columna. En 450 ponemos «A» a «0» y entre 460 y 490 imprimimos «ce» en los ocho bytes correspondientes de la pantalla con lo que el muñeco queda borrado.

Sigamos adelante:

```

500 LD A, (60158)
510 INC A
520 CP 4
530 JR C, CONT_2
540 LD A, C
550 SUB 1
560 JR NC, CONT_1
570 LD A, 31
580 CONT_1 LD C, A
590 LD (60157), A
600 LD A, 1
610 CONT_2 LD (60158), A

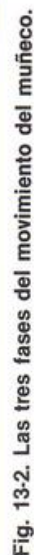
```



60189	470	BUC_1	LD	(HL),A
60190	480	INC	H	
60191	490	DJNZ	BUC_1	
60193	500	LD	A, (60158)	
60196	510	INC	A	
60197	520	CP	4	
60199	530	JR	C,CONT_2	
60201	540	LD	A,C	
60202	550	SUB	1	
60204	560	JR	NC,CONT_1	
60206	570	LD	A,31	
60208	580	CONT_1	C,A	
60209	590	LD	(60157),A	
60212	600	LD	A,1	
60214	610	CONT_2	(60158),A	
60217	620	LD	DE,60125	
60220	630	LD	H,0	
60222	640	LD	L,A	
60223	650	ADD	HL,HL	
60224	660	ADD	HL,HL	
60225	670	ADD	HL,HL	
60226	680	ADD	HL,DE	
60227	690	EX	DE,HL	
60228	700	LD	H,#50	
60230	710	LD	A,#A0	
60232	720	OR	C	
60233	730	LD	L,A	
60234	740	LD	B,8	
60236	750	BUC_2	A,(DE)	
60237	760	LD	(HL),A	
60238	770	INC	H	
60239	780	INC	DE	
60240	790	DJNZ	BUC_2	
60242	800	POP	HL	
60243	810	POP	DE	
60244	820	POP	BC	
60245	830	POP	AF	
60246	840	RETI		
23672	850	FRAMES EQU	23672	
Pass 2 errors: 00				
ACT	EAD7	BUC_1	EB1D	
BUC_2	EB4C	CONT_1	EB30	
CONT_2	EB36	DAT	EAE5	
DES	EAD6	FRAMES	5C78	
SIGUE_EB0D		START	EB01	
Table used:			130	from 224

Un detalle curioso de esta rutina es que no retorna con «RET!», como sería de esperar, sino con «E!» y «RET!». Lo cierto es que da lo mismo. Durante una rutina de respuesta a una interrupción enmascarable, las interrupciones quedan deshabilitadas. Se hace así para que, si el tiempo entre interrupciones es más corto que lo que tarda en ejecutarse la rutina, evite que el ordenador se quede atrapado en un bucle sin

No queremos que nuestro hombrecillo gane los 100 metros lisos y, además, esta velocidad impediría apreciar bien el efecto. Sería más adecuado dividir la velocidad por ocho, es decir, ejecutar la rutina solamente una vez por cada 8 interrupciones. De esta forma, tardará unos 16 segundos en cruzar la pantalla (exactamente, 15,36 segundos) que parece una velocidad más adecuada.



**Fig. 13-3. Listado completo de la rutina.**



fin donde iría expandiendo la pila indefinidamente. Sabemos que «RET» es equivalente a «POP PC» y «POP PC», por tanto, da lo mismo utilizar una forma que otra. Además, ambas ocupan 2 bytes.

Tal vez el lector se pregunte por qué se incluyó la instrucción «RET» en el juego de instrucciones del Z-80. Lo cierto es que, si afinamos mucho, ambas formas de retornar NO son totalmente equivalentes. La forma correcta es hacerlo con «RET» (que para eso está). Si utilizamos «EI» + «RET», corremos el riesgo de que llegue una petición de interrupción justo después de «EI» y antes de «RET», con lo que la rutina volvería a empezar antes de haber retornado, es decir, se anidaría sobre sí misma. Si esto ocurre una sola vez, no pasa nada grave, pero si el efecto se produce un gran número de veces, la pila de máquina empieza a expandirse indefinidamente y se produce el inevitable «crash» («cuelgue»).

No obstante y en este caso, la pequeña «chapuza» de los programadores de Sinclair no tiene importancia ya que las interrupciones se producen en tiempos claramente definidos y, por supuesto, mucho más largos que lo que tarda en ejecutarse la rutina. La razón de haberlo hecho así no está muy clara, pero tal vez se deba a que querían que la rutina fuera utilizable, desde un programa de usuario, para leer el teclado con «RST #38». En las rutinas que usted escriba y que trabajen por interrupción, es mejor que utilice

«RET» que es más «ortodoxo».

Ahora que hemos visto cómo funciona la rutina de interrupción de la ROM, vamos a diseñar una rutina nuestra que se ejecute por interrupción.

Hemos elegido un ejemplo, tal vez, poco práctico pero muy ilustrativo. Se trata de hacer que un pequeño muñeco se mueva, continuamente, por la pantalla, mientras el ordenador está haciendo cualquier otra cosa. El ejemplo servirá, también, para ilustrar una sencilla forma de hacer animaciones.

La rutina funcionará en respuesta a las peticiones de interrupción de la ULA, por lo que tendremos que escribir otra pequeña rutina que la active, cambiando a «Modo 2» de interrupción y cargando el vector de página de interrupción (registro «I») con un valor adecuado.

Lo primero que tenemos que hacer es decidir dónde colocamos nuestra rutina. Recordemos como trabaja la interrupción en «Modo 2»: Se forma una dirección con el vector de página (registro «I») y el contenido del bus de datos (en el Spectrum, siempre será FFh); al contenido de esta dirección y la siguiente lo llamaremos «vector de interrupción» (no confundir con el vector de página); este contenido será, precisamente, la dirección donde comenzará nuestra rutina.

Dado que el bus de datos siempre es «FFh», la dirección donde esté el vector de interrupción deberá terminar en «FFh». Hemos elegido la dirección «EAFH» (60159h), de forma que el registro «I» deberá contener «EAH». En esta di-

rección y la siguiente, estará almacenada la dirección de inicio de la rutina, por ejemplo: 60161 para que quede a continuación. Empecemos por ver la rutina de activación que carga «EAH» en el registro «I» y selecciona el «Modo 2» de interrupción:

```

110 ACT LD A, #EA
120 LD I, A
130 IM 2
140 RET

```

Lógicamente, habrá otra rutina para desactivar que cargue, de nuevo, «3Fh» en «I» y vuelva a seleccionar el «Modo 1»:

```

150 DES LD A, #3F
160 LD I, A
170 IM 1
180 RET

```

Ambas rutinas son muy similares y tan claras que no requieren explicación. Antes de proseguir con el programa, veamos cómo vamos a generar un muñeco que se mueva.

Para conseguir un efecto, medianamente convincente, de movimiento, utilizaremos tres caracteres distintos que representen tres fases del movimiento del muñeco. Estos tres caracteres están representados en la Figura 13-2 los hemos llamado «carácter 1, 2 y 3». El muñeco correrá hacia la izquierda; para ello, empezaremos por imprimir el

carácter 1 en la columna 31 de la fila 21 (va de «unos»). A continuación, imprimiremos el carácter 2 en esa misma columna. Luego, el carácter 3. Cuando hayamos terminado con los tres caracteres, empezaremos, de nuevo, por el 1, pero en la columna 30 y el muñeco se habrá desplazado una columna a la izquierda. Seguiremos así hasta la columna «0» y, después, volveremos a empezar por la 31. El efecto conseguido será el de un hombreillo corriendo de derecha a izquierda y reapareciendo, por la derecha, después de haber desaparecido por la izquierda.

La diferencia con los ejemplos vistos hasta ahora es que el muñeco seguirá corriendo mientras usted hace cualquier otra cosa con el ordenador, por ejemplo, mientras escribe un programa, lo lista o lo ejecuta. Únicamente se parará cuando maneje el cassette o ejecute un comando «BEEP» ya que, en estos casos, se deshabilitan las interrupciones.

A la derecha de cada carácter (en la Figura 13-2), se ven los bytes correspondientes en hexadecimal. Estos serán los datos que utilice nuestra rutina para dibujar el muñeco:

```

190 DATOS DEFB #0C, #0C, #06
200 DEFB #0D, #14, #0C
210 DEFB #13, #20, #30
220 DEFB #30, #18, #78
230 DEFB #10, #28, #2C
240 DEFB #20, #C0, #C0
250 DEFB #60, #00, #40
260 DEFB #C0, #60, #00
270 DEFB #1F, #01
280 DEFB #01, #EB

```

\*HISOF GENS3M ASSEMBLER\*  
ZX SPECTRUM

Copyright HISOFT 1983  
CURSO C/M MICROHOBBY

Pass 1 errors: 00

```

80 *C-
90 *D+
60119 100 ORG 60119
60119 110 ACT LD A, #EA
60121 120 LD I, A
60123 130 IM 2
60125 140 RET
60126 150 DES LD A, #3F
60128 160 LD I, A
60130 170 IM 1
60132 180 RET
60133 190 DAT DEFB #0C, #0C, #06
60136 200 DEFB #0D, #14, #0C
60139 210 DEFB #13, #20, #30
60142 220 DEFB #30, #18, #78
60145 230 DEFB #10, #28, #2C
60148 240 DEFB #20, #C0, #C0
60151 250 DEFB #60, #00, #40
60154 260 DEFB #C0, #60, #00
60157 270 DEFB #1F, #01
60159 280 DEFB #01, #EB
60161 290 START RST #38
60162 300 PUSH AF
60163 310 LD A, (FRAMES)
60166 320 AND #07
60168 330 JR Z, SIGUE
60170 340 POP AF
60171 350 RETI
60173 360 SIGUE PUSH BC
60174 370 PUSH DE
60175 380 PUSH HL
60176 390 LD A, (60157)
60179 400 LD C, A
60180 410 LD H, #50
60182 420 LD A, #A0
60184 430 OR C
60185 440 LD L, A
60186 450 XOR A
60187 460 LD B, B

```



MEMOTECHICO	OBJETO	Ref. pag.	Cod. pag.
LD IV, (nn)	Carga el registro IV con el octeto (nn).	54	69
LD (IV+d), n	Carga el octeto (IV+d) con el valor "n".	47	68
LD (IV+d), r	Carga el octeto (IV+d) con el valor del registro "r".	45	68
LD (nn), A	Carga el octeto (nn) con el valor del acumulador.	50	69
LD (nn), dd	Carga los octetos (nn) y (nn+1) con el valor del par de registros "dd".	55	69
LD (nn), HL	Carga los octetos (nn) y (nn+1) con el valor del par de registros "HL".	54	69
LD (nn), IX	Carga los octetos (nn) y (nn+1) con el valor del registro "IX".	55	69
LD (nn), IY	Carga los octetos (nn) y (nn+1) con el valor del registro "IY".	56	69
LD R, A	Carga el registro "R" con el valor del acumulador.	51	69
LD r, (HL)	Carga el registro "r" con el valor del octeto (HL).	43	67
LD r, (IX+d)	Carga el registro "r" con el valor del octeto (IX+d).	43	67
LD r, (IY+d)	Carga el registro "r" con el valor del octeto (IY+d).	44	67
LD r, n	Carga el registro "r" con el valor "n".	43	67
LD r, r'	Carga el registro "r" con el valor del registro "r'".	42	67
LD SP, HL	Carga el registro "SP" con el valor del registro "HL".	56	70
LD SP, IX	Carga el registro "SP" con el valor del registro "IX".	57	70
LD SP, IY	Carga el registro "SP" con el valor del registro "IY".	57	70
LDD	Carga el octeto (DE) con el valor del octeto (HL), decrementa "DE", "HL" y "BC".	180	187

Empezamos por cargar en «A» el n.º de carácter en curso. En 510 lo incrementamos y, en 520, lo comparamos con «4». Si es menor, saltamos a «CONT-2»; de lo contrario, la ejecución continúa en la línea 540, donde cargamos en «A» la columna en curso (antes ya la habíamos cargado en «C»).

En 550 decrementamos el n.º de columna. Utilizamos «SUB 1» porque «DEC A» no afectaría al indicador de acarreo. Si no se ha producido acarreo, quiere decir que no estábamos en la columna «0», así que saltamos a «CONT-1»; de lo contrario, cargamos «31» en «A» para que sea éste el nuevo n.º de columna.

En 580 («CONT-1»), cargamos en «C» el nuevo n.º de columna y lo guardamos en su variable correspondiente, en la línea 590. En 600, cargamos un «1» en «A» para que sea éste el nuevo n.º de carácter y guardamos este número, en su variable, en la línea 610.

Ahora sólo nos falta imprimir el carácter. Nuestro pequeño «font» provisional separará los datos colocados a partir de la línea 190. Esta línea quedará ensamblada a partir de 60133, por tanto, la dirección base de la tabla será 60125 (60133-8). Multiplicaremos el n.º de carácter por 8 y lo sumaremos a esta dirección base, para apuntar a los ocho datos que definen el carácter en cuestión:

620	LD	DE, 60125
630	LD	H, 0
640	LD	L, A
650	ADD	HL, HL

660	ADD	HL, HL
670	ADD	HL, HL
680	ADD	HL, DE
690	EX	DE, HL

El sistema es claro: cargamos la dirección base en «DE», el n.º de carácter en «L» y un «cero» en «H». Después, sumamos «HL» tres veces sobrepasando el acarreo, le sumamos el resultado a «DE».

Ahora, calcularemos la dirección de pantalla donde imprimirlo:

700	LD	H, 50
710	LD	A, 50
720	OR	C
730	LD	L, A

De la misma forma que antes, cargamos «50h» en «H» que será el octeto alto de la dirección. En «A» cargamos «A0h» y le hacemos un «OR» con el n.º de columna. Finalmente, transferimos el resultado a «L».

Ahora, tenemos la dirección de pantalla en «HL» y la dirección de los datos que forman el carácter en «DE». Sólo nos queda, por tanto, entrar en un bucle que realice la impresión:

740	LD	B, 8
750	BUC_2	LD A, (DE)
760	LD	(HL), A
770	INC	H
780	INC	DE
790	DJNZ	BUC_2

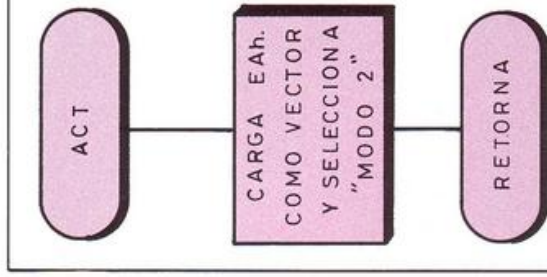


Fig. 13-5. Organigrama de las rutinas de activación y desactivación.



El bucle tendrá 8 iteraciones y «B» será el contador, en la línea 740 se inicializa su valor. En 750 y 760 transferimos el dato desde la dirección apuntada por «DE» a la apuntada por «HL». Incrementamos «H» para apuntar al siguiente scan de la misma columna y «DE» para apuntar al siguiente dato. Finalmente, cerramos el bucle en la línea 790.

Sólo nos queda recuperar los registros que habíamos preservado y retornar:

```
800 POP HL
810 POP DE
820 POP BC
830 POP AF
840 RETI
```

Lógicamente, los registros se recuperan en orden inverso a como se habían guardado. Podríamos retornar con «RET» ya que tenemos las interrupciones habilitadas (se habilitaron al salir del «RST #38»), pero lo hemos hecho

así para que el lector no se olvide de que una rutina de interrupción debe terminar, siempre, en «RETI».

En la figura 13-3 está el listado completo de la rutina. Tal vez le resulte ilustrativa una mirada a la figura 13-4 donde encontrará representación de su funcionamiento en forma de organigrama. Las rutinas de activación y desactivación se encuentran en la figura 13-5. Por último, quienes no dispongan, aún, de ensamblando, encontrarán en la figura 13-6 un listado de esta rutina

en el formato del Cargador Universal de Código Máquina.

Con esto, terminamos todas las instrucciones del Z-80. Los siguientes capítulos se dedicarán a comentar las particularidades del Spectrum que todo programador debe saber.

Como de costumbre, le recomendamos encarecidamente que resuelva los siguientes ejercicios, para comprobar si tiene afianzados los conocimientos adquiridos en este capítulo.

## EJERCICIOS

- 10) ¿Que ocurrirá si, mientras se está en una rutina de respuesta a una interrupción enmascarable, llega una petición de interrupción no enmascarable?
- 20) ¿Como responde el ordenador en el "Modo 0" de interrupción?
- 30) Escriba una rutina que, trabajando en respuesta a la interrupción enmascarable, lea la tecla "BREAK" ("CAPS SHIFT" + "SPACE") y detenga el programa, con el informe correspondiente, si estuviera pulsada.

Fig. 13-6. Listado de la rutina con formato de Cargador Universal. El «DUMP» deberá hacerse en la dirección 60119.

```
1 3EE4ED47ED5EC93E3FED 1498
2 47ED56C90C0C060D140C 670
3 13203030187810282C20 423
4 C0C060D040C06000E01 1055
5 01EBFF53A785CE60728 1283
6 03F1ED4DC5D5E53AFDEA 1742
7 4F26503EAB0B15FAF0608 896
8 772410FC3AFEEA3CCE04 1287
9 380D79D60130023E1F4F 627
10 32FDEA3E0132FEA11DD 1376
11 EA26006F29292919EB26 804
12 503EA0B16F060681A7724 785
13 1310FAE1D1C1F1ED4D00 1467
```

## NEMOTECNICO

NEMOTECNICO	OBJETO	Ref. pag.	Cod. pag.
JR NC,e	Salto relativo si el acarreo=0.	153	162
JR NZ,e	Salto relativo si no cero.	154	162
JR Z,e	Salto relativo si cero.	153	162
LD A,(BC)	Carga el acumulador con el octeto (BC).	47	68
LD A,(DE)	Carga el acumulador con el octeto (DE).	47	68
LD A,I	Carga el acumulador con el registro "I".	48	68
LD A,(nn)	Carga el acumulador con el octeto "nn".	48	68
LD A,R	Carga el acumulador con el registro "R".	49	68
LD (BC),A	Carga el octeto (BC) con el valor del acumulador.	49	69
LD (DE),A	Carga el octeto (DE) con el valor del acumulador.	50	69
LD (HL),n	Carga el octeto (HL) con el valor "n".	46	68
LD dd,nn	Carga el par de registros "dd" con el valor "nn".	51	69
LD HL,(nn)	Carga el par de registros "HL" con el octeto (nn).	52	69
LD (HL),r	Carga el octeto (HL) con el registro "r".	45	68
LD I,A	Carga el registro "I" con el acumulador.	50	69
LD IX,nn	Carga el registro "IX" con el valor "nn".	51	69
LD IX,(nn)	Carga el registro "IX" con el octeto (nn).	53	69
LD (IX+d),n	Carga el octeto (IX+d) con el valor "n".	46	68
LD (IX+d),r	Carga el octeto (IX+d) con el valor del registro "r".	45	68
LD IY,nn	Carga el registro "IY" con el valor "nn".	52	69



NEMOTECNICO	OBJETO	Ref. pag.	Cod. pag.
IN A, (n)	Carga el registro acumulador con el octeto que entra desde el periférico "n".	332	342
IN r, (C)	Carga el registro "r" con el octeto que entra desde el periférico "C".	333	342
INC (HL)	Incrementa el octeto (HL).	98	97
INC IX	Incrementa el registro "IX".	131	139
INC (IX+d)	Incrementa el octeto (IX+d).	99	97
INC IY	Incrementa el registro "IY".	131	139
INC (IY+d)	Incrementa el octeto (IY+d).	99	97
INC r	Incrementa el registro "r".	97	97
INC ss	Incrementa el par de registros "ss".	130	139
IND	Carga el octeto (HL) con el octeto que entra desde el periférico "C", decrementa "HL" y "B".	335	342
INDR	Carga el octeto (HL) con el octeto que entra desde el periférico "C", decrementa "HL" y "B", y repite hasta que "B"=0.	336	342
INI	Carga el octeto (HL) con el octeto que entra desde el periférico "C", incrementa "HL" y decrementa "B".	334	342
INIR	Carga el octeto (HL) con el octeto que entra desde el periférico "C", incrementa "HL", decrementa "B" y repite hasta que "B"=0.	334	342
JP (HL)	Salta a la dirección (HL).	157	162
JP (IX)	Salta a la dirección (IX).	157	162
JP (IY)	Salta a la dirección (IY).	158	162
JP cc,nn	Salta a la dirección "nn" si la condición "cc" es verdadera.	150	162
JP nn	Salta a la dirección "nn".	149	162
JR C,e	Salto relativo si el acarreo=1.	152	162
JR e	Salto relativo.	152	162

10) La petición de interrupción no enmascarable será atendida saltando a la dirección "0066h". Cuando se termine de atender, se continuará atendiendo la interrupción enmascarable. Esta anidación de interrupciones es posible gracias al funcionamiento de los "flip/flops" FF1 y FF2.

20) El microprocesador responde al "Modo 0" de interrupción, ejecutando el código de operación que se encuentre en el bus de datos (lo deberá insertar el periférico que solicite la interrupción). En el Spectrum, este modo de interrupción es redundante. La ULA no inserta ningún código de operación, por lo que el bus de datos se pone a "FFh"; pero éste es, precisamente, el código de operación de "RST #38", por lo que será esta la instrucción que ejecute el microprocesador, es decir, la misma que en "Modo 1".

30) Suponemos que direcciona adecuadamente la rutina en "Modo 2". El listado podría ser:

```

100 START PUSH AF
110 CALL #1F54
120 JR C,SIGUE
130 EI
140 RST #08
150 DEFB #14
160 SIGUE POP AF
170 JP #0038

```

En caso de no estar pulsada la tecla "BREAK", el retorno se produce desde la rutina que lee el teclado y actualiza "FRAMES". Esta rutina puede serle muy útil para poder detener un programa en código máquina que se haya "enganchado" en un bucle sin fin; siempre, claro está, que no tenga deshabilitadas las interrupciones.



NEMOTECNICO	OBJETO	Ref. pag.	Cod. pag.
ADC HL,ss	Suma con acarreo a "HL" el par de registros "ss".	126	139
ADC A,s	Suma con acarreo a "A" el operando "s".	82	90
ADD A,n	Suma en "A" el valor "n".	78	90
ADD A,(HL)	Suma en "A" el octeto (HL).	78	90
ADD A,(IX+d)	Suma en "A" el octeto (IX+d).	79	90
ADD A,(IY+d)	Suma en "A" el octeto (IY+d).	79	90
ADD HL,ss	Suma a "HL" el par de registros "ss".	126	139
ADD IX,pp	Suma a "IX" el par de registros "pp".	128	139
ADD IY,rr	Suma a "IY" el par de registros "rr".	129	139
AND s	Operación lógica AND entre operando "s" y registro "A".	104	123
BIT b,(HL)	Comprobación del bit "b" del octeto (HL).	259	264
BIT b,(IX+d)	Comprobación del bit "b" del octeto (IX+d).	259	264
BIT b,(IY+d)	Comprobación del bit "b" del octeto (IY+d).	260	264
BIT b,r	Comprobación del bit "b" del registro "r".	258	264
CALL cc,nn	Llamada a la sub-rutina en la dirección "nn" si la condición "cc" es verdadera.	287	293
CALL nn	Llamada a la sub-rutina en la dirección "nn".	286	293
CCF	Complementa el indicador de acarreo.	135	139
CP s	Compara el operando "s" con el registro acumulador.	123	139
CPD	Compara el octeto (HL) con el registro acumulador, decrementa "HL" y "BC".	185	188
CPDR	Compara el octeto (HL) con el registro acumulador, decrementa "HL" y "BC" y repite hasta que "BC"=0.	186	188

NEMOTECNICO	OBJETO	Ref. pag.	Cod. pag.
CPI	Compara el octeto (HL) con el registro acumulador, incrementa "HL" y decrementa "BC".	183	188
CPIR	Compara el octeto (HL) con el registro acumulador, incrementa "HL", decrementa "BC" y repite hasta "BC"=0.	184	188
CPL	Complementa a 1 el registro acumulador.	133	139
DAA	Ajusta a decimal el registro acumulador.	137	139
DEC m	Decrementa en 1 el operando "m".	100	97
DEC IX	Decrementa en 1 el registro "IX".	132	139
DEC IY	Decrementa en 1 el registro "IY".	132	139
DEC ss	Decrementa en 1 el par de registros "ss".	131	139
DI	Inhibe interrupciones.	360	362
DJNZ e	Decrementa "B" y salta relativo si "B" diferente de cero.	154	162
EI	Habilita interrupciones.	360	362
EX (SP),HL	Intercambia el octeto (SP) con el par de registros "HL".	174	187
EX (SP),IX	Intercambia el octeto (SP) con el registro "IX".	175	187
EX (SP),IY	Intercambia el octeto (SP) con el registro "IY".	176	187
EX AF,AF'	Intercambia los registros "AF" con "AF'".	173	187
EX DE,HL	Intercambia los registros "DE" con "HL".	173	187
EXX	Intercambia los registros "BC", "DE" y "HL" con "BC'", "DE'" y "HL'".	174	187
HALT	Espera interrupción o "RESET".	357	362
IM 0	Habilita el "Modo 0" de interrupción.	361	362
IM 1	Habilita el "Modo 1" de interrupción.	361	362
IM 2	Habilita el "Modo 2" de interrupción.	361	362



del buffer de edición tendrá efecto.

Los subcomandos del modo edición son los siguientes:

**Barra de espacio** - Pasa al siguiente carácter sin borrar. No se puede pasar del final de la línea.

**DELETE** - Pasa al carácter anterior sin borrar. No se puede ir más atrás del comienzo de la línea.

**Cursor derecha** - Pasa a la siguiente posición de tabulación.

**ENTER** - Introduce los cambios del buffer de edición sobre el programa fuente y sale del modo edición.

**Q** - Sale del modo edición e ignora los cambios realizados en el buffer.

**R** - Repone la línea del programa fuente en el buffer de edición.

**L** - Lista el resto de la línea desde el cursor hasta el final, permaneciendo en el modo de edición.

**K** - Borra el carácter señalado por el cursor.

**Z** - Borra desde el cursor hasta el final de la línea.

**F** - Busca una nueva cadena de caracteres f previamente definida. Los cambios efectuados con anterioridad quedan respaldados.

**S** - Sustituye la cadena de caracteres encontrada f por la s previamente definida.

**I** - Inserta caracteres en la posición del cursor hasta que se pulse ENTER.

**X** - Coloca el cursor después del último carácter de la línea i comienza a actuar como el comando I.

**C** - Cambia los caracteres po-

sicionados por el cursor hasta que se pulse ENTER.

### Ensamblaje y puesta en marcha

Para esta misión existen dos comandos.

#### A

Ensambla desde la primera línea de texto hasta la última.

#### R

Ejecuta el programa resultante si no hay errores de ensamblaje. La ejecución se hace en la dirección especificada en el directivo ENT.

### Comandos de cinta

#### P n,m,s

Salva en cinta el texto comprendido entre las líneas n y m, dándole el nombre s. No pide parámetros por tanto el cassette debe estar en posición de grabar.

#### G „s

Lee desde cinta el fichero con el nombre s y lo coloca a continuación del texto ya introducido. Este fichero fue previamente salvado a cinta por el comando P.

#### T n,m,s

Salva en cinta el texto comprendido entre la línea n y m para su posterior utilización por el directivo de ensamblador \*F. Por tanto, cuando un texto se pretenda añadir a otro ya introducido se utiliza-

rá el comando P. Si lo que se quiere es un texto para introducirlo dentro de un programa se utilizará el comando T. Esto resulta muy útil para codificar solamente una vez rutinas usadas en varios programas.

### Comandos de Microdrive

La versión «GENS-3M» está preparada para trabajar con Microdrive. Los comandos son los mismos que para cinta, con las siguientes modificaciones:

#### P n,m,h,s

Salva el código fuente desde la línea n hasta la m en el microdrive h con el nombre de fichero s.

#### G„h:s

Carga el código fuente cuyo nombre es «s» desde el microdrive número h.

### Otros comandos

#### B

Salta del ensamblador y retorna al Basic.

#### S

Cambia el carácter limitador de argumentos. Al iniciarse el ensamblador este carácter es la coma «,». El carácter separador no puede ser un espacio.

#### W n,m

Saca por impresora las líneas comprendidas entre n y m. Si se omite «n,m» se imprimirá el texto completo.

### NEMOTECNICO

### OBJETO

### Ref. pag.

### Cod. pag.

LDDR

Carga el octeto (DE) con el valor del octeto (HL), decremента "DE", "HL" y "BC", repite hasta que "BC"=0.

181

187

LDI

Carga el octeto (DE) con el valor del octeto (HL), incrementa "DE" y "HL", decremента "BC".

177

187

LDIR

Carga el octeto (DE) con el valor del octeto (HL), incrementa "DE" y "HL", decremента "BC", repite hasta que "BC"=0.

179

187

NEG

Complementa a 2 el acumulador.

134

139

NOP

No opera.

357

362

OR s

Operación lógica OR entre el operando "s" y el acumulador.

113

130

OTDR

Salida por el periférico "C" del octeto (HL), decremента "HL" y "B", repite hasta que "B"=0.

341

342

OTIR

Salida por el periférico "C" del octeto (HL), incrementa "HL", decremента "B", repite hasta que "B"=0.

339

342

OUT (C),r

Salida por el periférico "C" del valor del registro "r".

338

342

OUT (n),A

Salida por el periférico n del valor del acumulador.

337

342

OUTD

Salida por el periférico "C" del valor del octeto (HL), decremента "HL" y "B".

340

342

OUTI

Salida por el periférico "C" del valor del octeto (HL), incrementa "HL", decremента "B".

338

342

POP IX

Carga "IX" con los dos primeros octetos de la pila de máquina.

60

70

POP IY

Carga "IY" con los dos primeros octetos de la pila de máquina.

61

70

POP qq

Carga el par de registros "qq" con los dos primeros octetos de la pila de máquina.

60

70

PUSH IX

Carga los dos primeros octetos de la pila de máquina con "IX".

59

70



NEMOTECNICO	OBJETO	Ref. pag.	Cod. pag.
PUSH IY	Carga los dos primeros octetos de la pila de máquina con "IY".	59	70
PUSH qq	Carga los dos primeros octetos de la pila de máquina con el valor del par de registros "qq".	58	70
RES b,m	Pone a cero el bit "b" del operando "m".	262	266
RET	Retorno desde sub-rutina.	288	293
RET cc	Retorno desde sub-rutina si la condición "cc" es verdadera.	289	293
RETI	Retorno desde interrupción.	289	293
RETN	Retorno desde interrupción no enmascarable.	289	293
RL m	Rotación a la izquierda del octeto del operando "m" mas el acarreo.	213	212
RLA	Rotación a la izquierda del acumulador mas el acarreo.	207	212
RLC (HL)	Rotación a la izquierda del octeto (HL).	210	212
RLC (IX+d)	Rotación a la izquierda del octeto (IX+d).	210	212
RLC (IY+d)	Rotación a la izquierda del octeto (IY+d).	211	212
RLC r	Rotación a la izquierda del registro "r".	209	212
RLCA	Rotación a la izquierda del acumulador.	206	212
RLD	Rotación de 4 bits entre octeto (HL) y acumulador, octeto hacia la izquierda.	227	230
RR m	Rotación, a la derecha, del operando "m" mas el acarreo.	217	212
RRA	Rotación, a la derecha, del acumulador mas el acarreo.	209	212
RRC m	Rotación del operando "m" a la derecha.	215	212
RRCa	Rotación del acumulador a la derecha.	207	212

guientes teclas con un significado diferente del indicado en las mismas:

**EDIT** - Anula la última entrada

**DELETE** - Borra un espacio hacia atrás

**Cursor derecha** - Salta al siguiente tabulador

**Cursor izquierda** - Borra toda la línea

A continuación analizaremos los comandos más importantes sin pretender con ello excluir las especificaciones que en cada momento proporcione el realizador de un ensamblador. Como en el caso anterior nos centraremos en GENS 3.

### Comandos del editor

### INSERCIÓN DE TEXTO:

Se entiende como texto tanto las instrucciones como los comentarios, directivos del ensamblador y definición de campos.

### I n,m

Este comando pone al editor en modo de inserción automático, se irán introduciendo líneas de programa y éstas irán numeradas desde n saltando de m en m. Cada vez que el editor dé el cursor lo dará acompañado del número de línea correspondiente.

Esta es la forma en la que se va construyendo el programa fuente.

Al introducir el texto bien sea o no en modo inserción hay que tener en cuenta la siguiente actuación del editor. Si se está en modo automático el editor presenta el número

ro de la línea que se pretende incluir, si no es el primer campo a escribir; posteriormente se deja un espacio y se escribe la etiqueta, si no tiene etiqueta la sentencia se dejarán dos espacios con lo que el formato quedará perfectamente tabulado; a continuación irán el código nemotécnico y el operando; por último si se desea se pondrán los comentarios. Este sistema de ir dejando un solo espacio hace que el editor tabule correctamente el texto y permite que el ensamblador cuando se procese tome correctamente cada campo del formato de la instrucción.

Para salir del modo «Inserción», pulse **® EDIT** o **® CAPS SHIFT** + «1».

### LISTADO DE TEXTO:

Es necesario en múltiples ocasiones revisar lo introducido hasta el momento, si está en «Inserción», deberá salir al modo editor de la forma que se dijo antes.

### L n,m

Este comando lista el texto desde la línea n a la m, ambas inclusivas.

### K n

Este comando indica el número de líneas que se quiere visualizar de cada vez. Se utiliza en combinación con 'L', cada vez que se introduzca 'L' se visualizarán n líneas.

### MANIPULACION DEL TEXTO:

Este grupo de comandos está orientado a corregir el texto previamente introduciendo.

### D n,m

Borra todas las líneas desde la n a la m inclusive.

### M n,m

Mueve la línea n a la línea m borrando la que hubiera allí. La línea n permanece inalterada.

### N n,m

Renumerar el texto poniendo como primer número n en intervalos de m. Esto puede ser necesario si se han intercalado muchas líneas.

### F n,m,f,s

Busca en el texto comprendido entre las líneas n y m la cadena de caracteres f, si encuentra una cadena igual pone la línea en el buffer de edición y entra en dicho modo, el cual estudiaremos más detenidamente a continuación. El campo s es una cadena de caracteres que podrá sustituir a la cadena f en función de los comandos del modo edición. Los valores de los campos de F permanecen mientras no se definen otros.

### E n

Pasa la línea n al buffer de edición y entra en dicho modo. En este momento se podrá modificar la línea en curso sin afectar hasta que se decida, al texto original. Exactamente lo que se hace es copiar la línea desde el texto del programa fuente sobre un buffer que se displaya sobre la pantalla, sobre este texto se actúa modificándolo y cuando se considere oportuno, se salvará sobre el programa fuente, hasta que esto ocurra ninguna modificación



## Resultaría

CAMP02:	00h	00000000
	0Fh	00001111
	01h	00000001
	03h	00000111

## DEFS expresión

Incrementa el contador de posiciones en el valor de la expresión. Esto equivale a reservar una zona de memoria con una longitud igual al valor de la expresión.

## DEFM «S»

Define tantos octetos como caracteres tenga la cadena «S» con los valores ASCII de los mismos. El símbolo " limita la cadena.

Ejemplo:

LETRAS DEFM «ABCD»

## Resultaría

LETRAS:	41h	01000001
	42h	01000010
	43h	01000011
	44h	01000100

## Condicionales

## IF expresión

Si el valor de la expresión es cero el ensamblador no ensambla las siguientes instrucciones hasta encontrar un directivo ELSE o END; en caso contrario continúa normalmente.

## ELSE

Cambia el estado del ensamblador. Si estaba ensamblado deja de hacerlo; si no lo hacía pasa a hacerlo.

## END

Sea cual sea el estado anterior se pone a ensamblar.

## COMANDOS DEL ENSAMBLADOR:

Son también pseudonemotécnicos que a diferencia de los directivos no modifican el programa absoluto actuando únicamente sobre el listado resultante.

## \*E

Deja tres líneas en blanco.

## \*Hs

Imprime la cadena de caracteres después de cada comando \*E. \*H realiza un comando \*E de forma automática.

## \*S

Detiene el listado en esta línea. Sólo actúa sobre la pantalla, en la impresora no tiene efecto.

## \*L—

Detiene el listado a partir de esta línea tanto en pantalla como en impresora.

## \*L+

Continúa el listado a partir de esta línea tanto en pantalla como en impresora.

## \*D—

Pone las direcciones de los octetos en hexadecimal.

## \*D+

Pone las direcciones de los octetos en decimal.

## \*C—

Acorta la línea de listado

suprimiendo el código de máquina.

## \*C+

Vuelve a listar la línea completa.

## \*F (nombre de fichero)

Lee desde cinta el código agrupado bajo el nombre de fichero de 10 caracteres, el cual fue previamente salvado con el comando de editor T.

Caso de no poner nombre de fichero se tomará el primero de la cinta.

Este código se ensamblará con el resto del programa y el comando F será reconocido en la primera pasada del ensamblador.

## Editor

Un editor es un programa o una serie de rutinas que acompañan al ensamblador, su misión consiste en gestionar un fichero donde se introduce el programa fuente.

Un buen editor gestiona este fichero usando el mínimo de memoria por línea de codificación, lo cual consigue compactando los espacios. Tenga en cuenta que los miniordenadores del tipo del Spectrum no disponen de mucha memoria y dentro de esta memoria tiene que entrar el ensamblador con su editor, el programa fuente y el absoluto ejecutable.

Normalmente en modo editor se entra cuando se llama al ensamblador, tal es el caso del GENS 3. En este momento se está en condiciones de introducir los comandos para construir, modificar o editar el texto del programa simbólico.

Además se utilizan las si-

NEMOTECNICO	OBJETO	Ref. pag.	Cod. pag.
-------------	--------	-----------	-----------

RRD	Rotación de 4 bits entre octeto (HL) y acumulador, octeto a la derecha.	228	230
RST p	Llamada a dirección "p" de página 0.	291	293
SBC A,s	Restar del acumulador el operando "s" con acarreo.	89	90
SBC HL,ss	Restar del par de registros "HL" el operando "ss" con acarreo.	128	139
SCF	Poner a 1 el indicador de acarreo.	135	139
SET b, (HL)	Poner a 1 el bit "b" del octeto (HL).	260	265
SET b, (IX+d)	Poner a 1 el bit "b" del octeto (IX+d).	261	265
SET b, (IY+d)	Poner a 1 el bit "b" del octeto (IY+d).	261	265
SET b,r	Poner a 1 el bit "b" del registro "r".	260	265
SLA m	Desplazamiento aritmético a la izquierda del operando "m".	219	230
SRA m	Desplazamiento aritmético a la derecha del operando "m".	222	230
SRL m	Desplazamiento lógico a la derecha del operando "m".	224	230
SUB s	Restar del acumulador el operando "s".	84	90
XOR s	Operación, lógica OR exclusivo entre el operando "s" y el acumulador.	118	130



## MANEJO DE ENSAMBLADORES

Un ensamblador es un programa que, partiendo de una secuencia de instrucciones en código nemotécnico (Programa fuente), construye una secuencia de instrucciones en código máquina (Programa absoluto).

Una vez visto todo el repertorio de instrucciones del micro-procesador Z-80, tanto en su forma nemotécnica como en código máquina, no se le escapará al lector que lo más sencillo es escribir un programa con las instrucciones nemotécnicas. Pero este programa no puede ser ejecutado por el ordenador, no ocurre lo mismo que con el lenguaje Basic que cada instrucción es interpretada en el momento de la ejecución. Este programa fuente en ensamblar tiene que pasarse a un código entendible por el ordenador; ésta es la misión de un ensamblador.

Los ensambladores, para cumplir su misión, tienen que ir haciendo pasadas por las instrucciones para construir octetos en código máquina. Una sola pasada por las instrucciones no suele ser suficiente para convertir las totalidades, dependerá de la potencia del ensamblador el que se tengan que hacer dos o tres pasadas.

En una primera pasada se construirán todos los códigos de operación y se podrán poner los operandos si la variable ya ha sido definida que no siempre ocurrirá. Piense en

un salto a una instrucción con una etiqueta que esté situada más adelante.

En una segunda pasada se satisfarán los operandos que dependen de etiquetas situadas posteriormente. El caso del salto apuntado con anterioridad.

Una tercera pasada puede ser necesaria para las expresiones, si el ensamblador no tiene la limitación de que todos los valores que entren en una expresión estén definidos antes de la codificación de ésta, serán necesarios al menos tres pasadas por la secuencia de instrucciones, en caso contrario con dos pasadas es suficiente.

Cuando se ensambla un programa se le suele dar una dirección de comienzo para que todos los operandos estén calculados en base a esa dirección. Si no se hace así, será necesario un programa cargador que, en el momento de la carga, base la dirección del programa y por tanto sus operandos. Un programa en código de máquina sin base se llama relocable.

Otra de las funciones que suelen incorporar los ensambladores es la detección de errores tanto de sintaxis como de lógica, tales como código nemotécnico desconocido, etiqueta duplicada, mal dimensionado de campos, etc.

Después de analizar múltiples ensambladores actualmente en el mercado para el Zx Spectrum nos ha parecido

más interesante limitarnos al conocido «GENS-3». Las razones son las siguientes: es el más potente, por lo tanto el que más comandos tiene con lo que lo dicho sobre él podrá no estar en otros pero si lo será de forma muy parecida o igual; los códigos nemotécnicos y los formatos de las instrucciones son exactamente los mismos que hemos visto a lo largo de este curso y por último está comercializado con garantía de adquisición para los interesados.

Otra de las herramientas que incorpora un buen ensamblador es un editor. El editor consiste en una serie de comandos para construir el lenguaje fuente, los cuales permiten: borrar, intercalar, avanzar, retroceder, tabular, etc., todo aquello que simplifica la construcción de un programa fuente.

A pesar de las explicaciones de comandos que se den en este apartado, normalmente válidas para cualquier ensamblador, es inevitable leerse las instrucciones que acompañan a un programa ensamblador para aprovechar al máximo sus opciones.

Formato de la instrucción

ETIQUETA NEMOTÉCNICO OPERANDOS  
COMENTARIOS

### ETIQUETA:

Una etiqueta es una secuencia de caracteres, dígitos

y signos que representan un valor. Si la etiqueta precede a una instrucción su valor es la dirección de esta instrucción; si precede a una definición de campos es el valor de la dirección de comienzo de dichos campos y por último a una etiqueta se le puede dar un valor deseado por medio del directivo EQU.

En el caso de GENS 3 la etiqueta tiene una longitud determinada pero sólo son significativos los seis primeros caracteres, esto da la posibilidad de usar la etiqueta como comentario para entender el programa.

### CONTADOR DE POSICIONES:

El programa ensamblador conoce en cada momento la dirección de la instrucción que está analizando, esta dirección se representa por el signo '\$' y se conoce como contador de posiciones. El signo '\$' puede ser utilizado como parte del operando o en una expresión; una instrucción como JP \$ + 10 indicaría un salto de diez octetos desde la dirección del primer octeto de la instrucción. El contador de posiciones puede inicializarse con el directivo ORG.

### EXPRESIONES:

Una expresión es una sucesión de uno o varios términos separados por operadores.

Los términos pueden ser: constantes decimales constantes hexadecimales caracteres etiquetas contador de posiciones '\$' Las constantes decimales se definen con los propios di-

gitos, las hexadecimales anteponiendo el signo #, las binarias anteponiendo el signo % y los caracteres entre comillas. Ejemplos:

```
3648
#4E2
%100 101
(a)
```

Los operadores pueden ser:

+ suma  
- resta  
& AND  
(a) OR (a de arroba con S/S + «2»)  
! XOR  
\* multiplicación  
/ división  
? función MOD (módulo)

Los operadores se valoran de izquierda a derecha según aparecen en la expresión sin que exista ninguna prioridad ni siquiera por medio de paréntesis. Por ejemplo en una expresión del tipo:

term1 + term2 \* term3 - term4

primero se sumaría term1 y term2, se multiplicaría el resultado con term3 y se restaría al resultado term4.

Ejemplo de expresiones:

```
%0110 & %100 10
472 * #E
etiqueta5 + %110
```

Observe que los operandos y los términos pueden ir separados por espacios.

### DIRECTIVOS DEL ENSAMBLADOR:

Los directivos son códigos pseudo-nemotécnicos que se escriben con las instrucciones pero que no tienen una correspondencia con códigos

de máquina; sólo son entendidos por el ensamblador en tiempo de ensamblaje.

A continuación describiremos los más comunes. Valorar etiquetas:

### ETIQUETA EQU expresión

Iguala la etiqueta al valor de la expresión, de tal forma que cada vez que se utilice la etiqueta en un operando o en un operador, se estará usando su valor. Este es el único directivo en el que la etiqueta es obligatoria. Ejemplo:

NUMERO-INICIAL EQU 28

Resultaría

NUMERO: 00011100 1CH

Definición de campos:

### DEFB expresión, expresión,...

Define tantos octetos como expresiones tenga el operando con el valor de las mismas, por tanto el valor de la expresión no debe superar el tamaño del octeto. Ejemplo:

CAMPO DEFB 5\*7, NUMERO-INICIAL -1

Resultaría

CAMPO: 00001011 0Bh  
00011011 1Bh

### DEFW expresión, expresión,...

Define tantos pares de octetos como expresiones tenga el operando con el valor de las mismas, por tanto el valor de la expresión no debe superar el tamaño de dos octetos (16 bits). Ejemplo:



4.º)

Entraremos por «SA-FLAG» para poder controlar la longitud del tono guía.

Si hubiéramos querido salvarlo con su tono guía normal de 2 segundos y su flag normal de «FF», podríamos haber entrado por «SA-BYTES» de la siguiente forma:

```
LD IX,16384
LD DE,6912
LD A, #FF
CALL #04C2
```

SA-CONTRL: 0970h (2416)

- > DESCRIPCION: Realiza la función equivalente al comando «SAVE» del Spectrum. Salva un bloque de bytes con cabecera, flags y tonos guía correctos. Salva por pantalla el mensaje «Start tape, then press any key» y espera la pulsación de una tecla para empezar el proceso. Es necesario que se haya construido, previamente, una cabecera en algún lugar de la memoria.
- > ENTRADA: «HL» = Dirección de inicio del bloque.
- «IX» = Dirección de inicio de la cabecera.

- > SALIDA: «IX» = Final del bloque más 2.
- > REGISTROS ALTERADOS: «AF», «BC», «HL», «DE», «IX» y «AF».
- > FUNCIONAMIENTO: Se empieza por imprimir el mensaje «Start tape, then press any key» y esperar a que se pulse una tecla. A continuación, se salva la

cabecera con un flag «00» y un tono guía de 5 segundos. Luego, se hace una pausa de 50 interrupciones (1 segundo). Finalmente, se salva el bloque con flag «FF» y tono guía de 2 segundos.

Existe un punto de entrada alternativo donde evitamos que se imprima el mensaje y se espere la pulsación de tecla. Este punto de entrada es por la dirección 0984h (2436) y los requisitos son los mismos, excepto que el valor de «HL» tiene que estar, también, en la parte alta de la pila de máquina.

> EJEMPLO: Vamos a guardar la pantalla (como en el ejemplo anterior) pero esta vez lo haremos con cabecera, de forma que pueda ser cargada con el comando «LOAD» del Basic. La rutina puede ser:

```
LD HL,16384
LD IX,CABEC.
CALL #0970
.....
CABEC.
DEFB #03
DEFB "PANTALLA"
DEFB 0,0
DEFW 6912
DEFW 16384
DEFW 0
```

Hemos construido la cabecera a partir de la dirección dada por la etiqueta «CABEC.» el primer byte es «3» para indicar un bloque de bytes. Los 10 bytes siguientes contienen el nombre del fichero; ocho bytes para la palabra «PANTALLA» y los 2 restantes a «0». Los dos bytes siguientes contienen la longitud, los siguientes la dirección de inicio y los dos últimos están a «0» ya que, en este caso, no tienen significado.

## TABLA DE MANEJO DE GENS 3

Opciones de llamada del ensamblador

Opciones	Significado
1	Produce un listado de la tabla de símbolos
2	No genera código objeto
4	No produce ningún listado de ensamblaje
8	Imprime el listado en la impresora
16	Sitúa el código objeto después de la tabla de símbolos. En combinación con el directivo ORG puede colocar el código objeto en una parte de la memoria y estar diseñado para ejecutarse en otra
32	No comprueba dónde va el código objeto

Operadores de las expresiones

Operador	Significado
+	Suma
-	Resta
&	AND
!	OR
!	XOR
*	Multiplicación
/	División
?	Módulo

Directivos de ensamblador

Formato	ETIQUETA DIRECTIVO OPERANDO, OPERANDO,...
Directivo	Significado
ORG	Inicia el contador de direcciones
EQU	Da un valor a una etiqueta
DEFB	Define octetos
DEFW	Define palabras (dos octetos)
DEFS	Reserva espacio de memoria
DEFM	Define literales
ENT	Define dirección de ejecución del código objeto
IF	Condiciona el ensamblaje del código fuente
ELSE	Cambia el estado de la condición
END	Termina el ensamblado condicional

Comandos del ensamblador

Comando	Significado
*E	Envía tres líneas en blanco
*Hs	Pone s como cabecera después de *E
*S	Detiene el listado, cualquier tecla lo activa
*L-	Detiene el listado a partir de esta línea
*L+	Activa el listado a partir de esta línea
*D+	Contador de direcciones en decimal
*D-	Contador de direcciones en hexadecimal
*C-	No imprime código objeto
*C+	Vuelve a imprimir el código objeto
*F (f)	Intercala el código salvado con el nombre de fichero f



## Comandos del editor

Comando	Significado
I n,m	Pone editor en inserción automática desde línea n en intervalos m
L n,m	Lista texto desde n a m inclusive
K n	Establece número de líneas n para listar en partes
D n,m	Borra las líneas desde la n a la m
M n,m	Pasa el texto de la línea n a la m
N n,m	Renumerar el texto desde n con m de intervalo
F n,m,f,s	Busca la cadena f entre las líneas n y m
E n	Edita línea n y se pone en edición de línea.
P n,m,s	Ver subcomandos
G ,s	Salva en cinta el texto definido entre n y m bajo el nombre s
T n,m,s	Lee de cinta el fichero con el nombre s
A	Salva en cinta el texto definido entre n y m bajo el nombre s para uso posterior del comando de ensamblador *F
R	Ensambla el programa
B	Ejecuta el programa en la dirección definida con el directivo ENT si no hubo errores
C	Pasa control al programa monitor
S,d	Convierte ficheros de GENS 1 a GENS 3
V	Cambia limitador de los argumentos
W n,m	Muestra los valores de N1, N2, S1 y S2
X	Saca por impresora el texto comprendido entre n y m
	Saca a pantalla las direcciones de comienzo y final del fichero en decimal

## Subcomandos del editor de línea

Subcomando	Significado
Espacio	Pasa a siguiente carácter
DELETE	Pasa a carácter anterior
- >	Pasa a siguiente posición de tab
Q	Ignora cambios realizados
R	Recupera la línea desde el texto
L	Lista desde cursor a final de línea
K	Borra carácter
Z	Borra desde cursor a fin de línea
F	Busca siguiente cadena f
S	Cambia s por f
I	Inserta caracteres hasta pulsar ENTER
X	Pone el cursor en el último carácter de la línea y actúa como I
C	Cambia caracteres hasta pulsar ENTER

«00» = Programa Basic (Program.);  
 «01» = Matriz numérica (Number array);  
 «02» = Matriz de caracteres (Character array);  
 «03» = Bloque de bytes (Bytes);

2.º al 11.º): 10 caracteres conteniendo el nombre del bloque.  
 12.º y 13.º): Longitud total del bloque. En «Program», longitud de programa más variables.  
 14.º y 15.º): En «Program» línea de arranque para auto-ejecución. En «Bytes» dirección inicial de carga.  
 16.º y 17.º): En «Program» longitud del programa (sin variables).

La cabecera tiene un tono guía que dura 5 segundos. A continuación hay una pusa de aproximadamente 1 segundo y, luego, el bloque de información, propiamente dicho, precedido de un tono guía de 2 segundos de duración. El último byte de este bloque es un byte de control que contiene el resultado de «XORear» entre sí todos los bytes del bloque (otro tanto ocurre en la cabecera). Este byte de control es añadido de forma automática por la rutina «SA-BYTES» y es leído y chequeado por la «LD-BYTES» de forma que el programador no se tiene que preocupar, en absoluto, por él.

El «Flag» del registro «A» le indica a la rutina de carga «LD-BYTES» si se trata de una cabecera o de un bloque de información. En el primer caso, este byte contendrá «00» (cabecera) y en caso de ser un bloque de información, contendrá «FF». En realidad, nosotros podemos hacer que este byte contenga cualquier dato. Cuando utilizamos la rutina «LD-BYTES» para cargar el bloque, el dato contenido por «A» deberá ser igual a aquel que utilizamos para salvar el blo-

que. De esta forma, es posible proteger un bloque de bytes de forma que solo se pueda cargar si se conoce el flag identificador.

La rutina añadirá un tono guía de 5 segundos si el flag identificador tiene su bit de más peso a «0» y añadirá un tono de 2 segundos si el flag identificador tiene el byte de más peso a «1». No obstante, existe un punto de entrada alternativo en el cual podemos fijar la longitud del tono guía. Este punto de entrada es «SA-FLAG» en la dirección 04D0h (1232). Las condiciones de entrada en este punto son las mismas que en «SA-BYTES», pero además, deberemos tener el número 053Fh en la parte alta de la pila de máquina y una constante en «HL» que va a indicar la duración del tono guía. Esta constante será igual (aproximadamente) a «t» \* 1612 donde «t» es el número de segundos que durará el tono guía.

```
LD IX, 16384
LD DE, 6912
LD A, #AA
LD HL, #053F
PUSH HL
LD HL, 4836
CALL #04D0
```

> EJEMPLO: Vamos a salvar una pantalla sin cabecera, con un flag identificador «AA» (170) y con un tono guía de 3 segundos de duración:

- 1.º) La dirección de inicio de la pantalla es 16384.
- 2.º) Su longitud es 6912.
- 3.º) La constante para el tono guía es 3\*1612 + 4836



porización de señales no se vea afectada por la respuesta a perturbaciones de interrupción.

- > ENTRADA: Si llamamos «f» a la frecuencia de la nota en hercios (ciclos por segundo) y «t» a su duración en segundos, los valores de «HL» y «DE» serán los siguientes:

$$\begin{aligned} \text{«HL»} &= \text{INT} \left( (1.5 * 6689/4) * 30.125 \right) \\ \text{«DE»} &= \text{INT} \left( f * t \right) \end{aligned}$$

La tabla de frecuencias para la octava central es la siguiente:

	(C)	
D0	261.63	
D0 #	277.18	
RE	293.66	
RE #	311.13	
MI	329.63	
FA	349.23	
FA #	369.99	
SOL	392.00	
SOL #	415.30	
LA	440.00	
LA #	466.16	
SI	493.88	

La primera columna representa la notación clásica, la segunda columna es la notación americana y la tercera representa la frecuencia expresada en hercios. El punto indica los decimales (no los «millares»). Para subir una octava, se multiplicarán todas las frecuencias por 2 y se dividirán por 2 para bajar una octava. Estas frecuencias corresponden a la «afinación» del Spectrum, pero pueden modificarse ligeramente para conseguir cualquier afinación.

- > SALIDA: Ninguna.
- > EJEMPLO: Vamos a interpretar la nota FA de la segunda octava durante 1.5 segundos:

La frecuencia de FA es 349.23, la multiplicamos por 2 para la segunda octava:  $349.23 * 2 = 698.46$ .

Multiplicamos este valor por 1.5 segundos para hallar el contenido de «DE»:  $698.46 * 1.5 = 1047.69$  y tomamos su parte entera: «DE» = 1047 = 0417h.

El contenido de «HL» será: «HL» = INT  $(1.5 * 6689/4) - 30.125 = 2478 = 09AEh$ .

Para llamar a la rutina haremos:

```
LD HL, 2478
LD DE, 1047
CALL #3B5
```

## SA-BYTES: 04C2h (1218)

- > DESCRIPCION: Es la subrutina que utiliza el comando «SAVE» dos veces, una para la cabecera y otra para el bloque de datos propiamente dicho. Podemos utilizarla en nuestros programas para salvar bloques sin cabecera.

- > ENTRADA: «IX» = Dirección inicial del bloque.  
«DE» = Longitud del bloque (n.º de bytes).  
«A» = Flag identificador (ver FUNCIONAMIENTO).

- > SALIDA: «IX» = Final del bloque más 2.

- > REGISTROS ALTERADOS: «AF», «BC», «HL», «DE», «IX», «AF».

- > FUNCIONAMIENTO: Cada bloque de bytes salvado en el casette, con la instrucción «SAVE» del Basic, consta de una cabecera y el bloque propiamente dicho. La cabecera consta de 17 bytes con el siguiente significado:

- 1.º): Tipo de bloque:

# SUBROUTINAS DE LA ROM

Una vez vistas todas las instrucciones y la forma de manejar un ensamblador, lo más probable es que el lector se lance, como loco, a escribir rutinas en código máquina para incorporar en sus programas.

En muchos casos, habrá rutinas que no será necesario escribir, ya que los señores de SINCLAIR lo han hecho por nosotros. Estas rutinas se encuentran en la ROM. Forman parte del Sistema Operativo y son utilizadas por éste para realizar ciertas tareas. Pero también podemos utilizarlas nosotros, llamándolas desde nuestros programas. Lo único que se necesita saber, para esto, es:

- 1.º Qué hace la rutina.
- 2.º Dónde está ubicada.
- 3.º Condiciones de entrada (contenido de registros, flags y variables).
- 4.º Condiciones de salida.

Estos datos serán los que daremos en el presente capítulo. La lista de rutinas no es exhaustiva, simplemente, hemos seleccionado aquellas que consideramos más útiles.

Al final del capítulo, pasaremos, también, revista al calculador de la ROM.

El formato que vamos a emplear para cada rutina es el siguiente:

**NOMBRE:** XXXXh (xxxx)

- > DESCRIPCION: ...
- > ENTRADA: ...
- > SALIDA: ...
- > REGISTROS ALTERADOS: ...
- > FUNCIONAMIENTO: ...
- > EJEMPLO: ...

La primera línea es el nombre de la rutina (el que tiene en el código fuente del Sistema Operativo). A continuación viene su dirección, tanto en hexadecimal como en decimal.

La segunda línea es una breve descripción de lo que hace la rutina. Esta descripción puede servir como referencia rápida para elegir la rutina más adecuada a determinada tarea.

La tercera línea indica las condiciones de entrada a la rutina. El estado de los registros que no se mencionen no afecta al funcionamiento de la rutina.

La cuarta línea indica las condiciones de salida. Los registros que no se mencionen, o bien no sufren modificación, o bien salen con un contenido indeterminado.

La quinta línea indica los registros que es conveniente preservar, antes de llamar a la rutina, si se desea que su valor no resulte alterado. En caso de que no exista esta línea, deberá asumirse que la rutina altera el contenido de todos los registros del «set» principal («A», «F», «B», «C», «D», «E», «H» y «L»).

En el apartado «FUNCIONAMIENTO», se dará una descripción más amplia de cómo funciona la rutina. Cuando la longitud de la misma lo permita, procuraremos incluir su listado.

Por último, en el apartado «EJEMPLO», se dará, cuando sea posible, un breve ejemplo de utilización de la rutina. Procuraremos, siempre, que el ejemplo ilustre al lector sobre la forma de usarla.

Para utilizar estas rutinas en sus pro-



gramas, deberá llamarlas con «CALL». No incluimos aquí los «RESTARTs» de página «0» ya que se vieron en un capítulo anterior.

Para evitar confusiones, utilizaremos la palabra «pila» para referirnos a la pila de máquina y usuario, y la palabra «stack» para referirnos a la pila del calculador. Esta pila se utiliza para almacenar datos en los cálculos, cada elemento de la misma tiene 5 bytes que pueden representar un número en coma flotante o los parámetros de una cadena. La veremos con más detenimiento cuando estudiemos el calculador de la ROM.

### Rutinas de control de pantalla

#### PLOT-SUB: 22E5h (8933)

- > DESCRIPCION: Equivalente al comando «PLOT» del Basic.
- > ENTRADA: «B» = Coordenada «Y»  
«C» = Coordenada «X»
- > SALIDA: Ajusta coordenadas en la Variable «COORDS».

> FUNCIONAMIENTO: Se trata de una entrada alternativa a la rutina del comando «PLOT». La verdadera entrada es por 22DCh y exige que las coordenadas se hallen presentes en la parte alta del stack del calculador. Primero actualiza la variable «COORDS», luego halla la dirección del byte de pantalla que contiene el pixel que hay que poner a «1» y, finalmente, altera el bit en cuestión, tomando en cuenta el estado de «INVERSE» y «OVER» (Bit 0 de «P-FLAGS» es «1» para «OVER 1» y Bit 2 de «P-FLAGS» es «1» para «INVERSE 1»).

- > EJEMPLO: Supongamos que queremos poner a «1» el pixel cuyas coordenadas son: x = 112 e y = 93. El procedimiento sería:

```
PUSH BC
LD BC, #5D70
PUSH HL
PUSH AF
PUSH DE
CALL #22E5
POP DE
POP AF
POP HL
POP BC
```

Esto sería equivalente a la sentencia Basic: PLOT 112,93.

#### POINT: 22CEh (8910)

- > DESCRIPCION: Equivalente a la función «POINT» del Basic.
- > ENTRADA: «B» = Coordenada «Y»  
«C» = Coordenada «X»

> SALIDA: Coloca el resultado en la parte alta del stack del calculador. Este resultado será «0» si el pixel tiene color de papel o «1» si lo tiene de tinta.

> FUNCIONAMIENTO: En realidad, se trata de un punto de entrada alternativo a la subrutina de la función «POINT». El otro punto de entrada es por 22CBh y exige que las coordenadas se hallen en la parte alta del stack. Al igual que la anterior, emplea por hallar la dirección del byte que contiene el pixel y, luego, comprueba si este es «1» o «0» terminando por meter este valor en el stack.

#### CL-SCROLL: 0E00h (3584)

- > DESCRIPCION: Hace un «Scroll» de pantalla hacia arriba, tantas líneas como indique el contenido del registro «B».
- > ENTRADA: «B» = N.º de líneas a desplazar.

> SALIDA: Ninguna.

> FUNCIONAMIENTO: Empieza por hallar la dirección de pantalla de la línea que va a quedar en la parte alta, a continuación, entra un bucle donde copia cada línea en su nueva ubicación. Finalmente, sale por «CL-LINE» (siguiente rutina) para borrar tantas líneas, desde abajo, como se hubieran desplazado hacia arriba. Un punto de entrada alternativo sería por 0DFEH para desplazar toda la pantalla. Esto sería equivalente a:

```
LD B, 23
CALL #0E00
```

De hecho, la instrucción ensamblada en 0DFEH es, precisamente, LD B,23.

#### CL-LINE: 0E44h (3652)

- > DESCRIPCION: Borra tantas líneas de la pantalla, contadas desde abajo, como indique el contenido del registro «B».
- > ENTRADA: «B» = N.º de líneas a borrar.

> SALIDA: Ninguna.

> FUNCIONAMIENTO: Empieza por hallar la dirección de la línea más alta que habrá de borrar. A continuación, entra en un bucle donde borra todos los scans hasta el final de la pantalla. Finalmente, copia los atributos permanentes en las líneas que ha borrado. Un punto de entrada alternativo para hacer la función equivalente a «CLS» sería por 0D6Bh que reseta, también, las posiciones de PRINT y PLOT. En realidad, esta última es la rutina de reseta al comando «CLS».

#### BORD-1: 229Bh (8859)

- > DESCRIPCION: Pone el borde del color especificado por el contenido del registro «A».
- > ENTRADA: «A» = Nuevo color del borde.
- > SALIDA: Altera la variable «BORDCR».
- > REGISTROS ALTERADOS: «AF».

> FUNCIONAMIENTO: Se trata de una entrada alternativa a la rutina del comando «BORDER». Esta última tiene su entrada por 2294h y requiere que el nuevo color se encuentre en el stack. BORD-1 empieza por hacer un «OUT (FE),A» para cambiar el color del borde, a continuación, halla un color de tinta que contraste con este y, por último, mete estos datos en la variable «BORDCR» para fijar los atributos de la parte inferior de la pantalla.

- > EJEMPLO: Para poner el borde de color rojo, podríamos hacer:

```
LD A, 2
CALL #229B
```

### Rutinas de cassette y sonido

#### BEEPER: 03B5h (949)

- > DESCRIPCION: Se trata de la subrutina que utiliza el comando «BEEP» del Basic para producir un tono de una frecuencia y duración determinadas. Hay que poner sumo cuidado con los datos de entrada ya que estos no son, exactamente, como en el Basic. La rutina deshabilita las interrupciones al principio y las vuelve a habilitar al final; lo hace así para que la tem-



El stack del calculador está situado por encima del área de trabajo. Su dirección base está apuntada por la variable del Sistema «STKBOT» y el último dato introducido es apuntado por «STKEND». Esta última variable será, por tanto, el puntero del stack. Cuando no hay ningún dato introducido, «STKEND» apunta al mismo sitio que «STKBOT». El stack del calculador, a diferencia de la pila de máquina, crece hacia arriba. Cada dato ocupa 5 bytes, por tanto, cada vez que se introduzca un dato en el stack, la variable «STKEND» se incrementa 5 veces; y se decrementa 5 veces cada vez que se saca un dato del stack.

Los datos almacenados en el stack serán los operando que utilice el calculador. También, los resultados de los cálculos realizados por éste irán a parar al stack. Dado que el calculador puede operar tanto con cadenas como con números, cada dato del stack podrá ser un número o los parámetros de una cadena. Veamos primero el segundo caso.

## CADENAS

Las cadenas con las que opera el calculador podrán estar situadas en cualquier lugar de la memoria. Al calculador sólo le interesan sus parámetros. Cuando una operación entre cadenas (por ejemplo, una concatenación) dé como resultado otra cadena, el calculador la construirá en el área de trabajo y dejará sus parámetros como última entrada en el stack. Veamos cuáles son esos parámetros.

Los parámetros de una cadena (como cualquier dato del calculador) constan de 5 bytes, el primero de ellos es el que ocupa la dirección de memoria más baja en el stack. Este primer byte es un flag que indica si la cadena es nueva o procede de una asignación. Este flag es utilizado por el comando «LET» para saber si tiene que borrar una cadena anterior con el mismo nombre. Suponga la operación: LET A\$ = A\$ + B\$ al colocar el resultado de la operación, como A\$, en el área de variables, es necesario borrar la anterior variable A\$. Esto lo sabe la rutina de LET, porque el flag será «1». En caso contrario, este flag será «0». Los dos bytes siguientes contienen la dirección de memoria a partir de donde está colocada la cadena. Finalmente, los dos últimos bytes contienen la longitud de ésta.

Observe que, cuando hagamos operaciones con cadenas, deberemos pasar estos datos al stack. Si la cadena de entrada está en una variable, podemos buscarla con la ayuda de «NEXT-ONE» que nos dará su dirección y longitud. La cadena resultante la podríamos leer del área de trabajo, teniendo en cuenta los parámetros que el calculador nos ha dejado en el stack. De esta forma, podríamos implementar funciones inexistentes en el Basic del Spectrum, por ejemplo, la función: STRING\$(n,c) que nos devuelve una cadena de «n» caracteres cuyo código ASCII es «c».

Para ayudarnos en la manipulación de cadenas con el calculador, tenemos dos subrutinas. La primera nos va a permitir meter en el stack los parámetros de una cadena, la segunda, lógicamente, nos va a permitir sacarlos.

### STK-STO-\$: 2AB2h (10930h)

- > DESCRIPCION: Introduce, en el stack, los parámetros de una cadena.
- > ENTRADA: «A» = Flag.  
«DE» = Dirección de inicio.  
«BC» = Longitud de la cadena.
- > SALIDA: La variable «STKEND» resulta incrementada cinco veces. Los parámetros quedan en el stack.
- > FUNCIONAMIENTO: Su listado es el siguiente:

Si quisiéramos hacer lo mismo, pero sin que se imprimiera el mensaje y se esperara la pulsación de una tecla, la rutina tendría que ser:

```
LD HL,16384
LD IX,CABEC.
CALL #0970
.....
CABEC. DEFB #03
DEFB "PANTALLA"
DEFB 0,0
DEFB 0912
DEFB 16384
DEFB 0
```

### LD-BYTES: 0556h (1366)

- > DESCRIPCION: Sirve tanto para cargar como para verificar un bloque de datos o una cabecera.
- > ENTRADA: «IX» = Dirección de comienzo.  
«DE» = Longitud.  
«A» = Flag identificador (deberá ser igual que aquel con el que se salvó el bloque, de lo contrario, éste no será cargado).  
«Carry» = (Indicador de acarreo). Deberá estar a «1» para cargar y a «0» para verificar.
- > SALIDA: «IX» = Dirección del último byte cargado más uno.  
«DE» = «0» si la carga ha sido correcta.  
«Carry» = «1» si la carga ha sido correcta, «0» si ha habido error de carga o de verificación.
- > REGISTROS ALTERADOS:

«AF», «BC», «HL», «DE», «IX» y «AF».

- > FUNCIONAMIENTO: Se empieza por esperar un tono guía de, al menos, un segundo de duración. A continuación, se carga el flag y se retorna si no coincide con el especificado al llamar a la rutina. Después se van cargando o verificando, todos los bytes uno por uno; se retorna si se produce algún error (por ejemplo, si se interrumpe la señal entrante) con el acarreo a «0». Finalmente, se carga el último byte (el de control) y se comprueba si es correcto; de no serlo, se retorna con el acarreo a «0», en otro caso, se retorna con el acarreo a «1».
- > EJEMPLO: Vamos a cargar la pantalla que habíamos salvado, anteriormente, sin cabecera y con un flag «AA». La rutina será:

```
LD HL,16384
LD IX,CABEC.
PUSH HL
CALL #0984
.....
```

Llamaremos a la rutina con un «CALL LOAD» si es desde código máquina o un «RANDOMIZE USR ...» si es desde Basic. En cualquier caso, la rutina retorna si la carga ha sido correcta e imprime el mensaje. «Tape loading error» si se ha producido algún error.

### Rutinas de uso general

#### BREAK-KEY: 1F54h (8020h)

- > DESCRIPCION: Comprueba si están pulsadas las teclas «Caps



- > shift» y «Space» de forma simultánea; o la tecla «BREAK» o «PARAR» en el Plus.
- > ENTRADA: Ninguna.
- > SALIDA: Indicador de acarreo a «1» si no está pulsada y a «0» si lo está.
- > REGISTROS ALTERADOS: «AF».
- > FUNCIONAMIENTO: Primero lee la tecla «Space» y retorna con el carry a «1» si no está pulsada. Si lo está, lee la tecla «Caps Shift» y retorna con el carry a «0» si está pulsada y a «1» si no lo está. En los ejercicios del capítulo relativo a las instrucciones de entrada/salida, se puede ver el listado de esta rutina.

```

LOAD LD IX,16384
LD DE,6912
LD A,#AA
SCF
CALL #0556
RET C
RST #08
DEFB #1A

```

#### PAUSE-1: 1F3Dh (7991)

- > DESCRIPCIÓN: Entra en un bucle donde espera el tiempo indicado por el registro «BC» en cincuenta-avos de segundo. También sale del bucle si se pulsa una tecla (equivalente al comando «PAUSE» del Basic).
- > ENTRADA: «BC» = Tiempo de la pausa en 1/50 de segundo. Si vale «0», la pausa es indefinida hasta que se pulse una tecla.
- > SALIDA: Bit 5 de «FLAGS» a «1».

- > REGISTROS ALTERADOS: «AF» y «BC».
- > FUNCIONAMIENTO: La primera instrucción es «HALT» donde se detiene la ejecución hasta que se reciba una interrupción, momento en el que se decrementa «BC» y se retorna si ha llegado a «0»; en caso contrario, se lee el bit 5 de «FLAGS» para ver si se ha pulsado alguna tecla. En caso afirmativo, se retorna; si no, se vuelve al principio. Es imprescindible que las interrupciones estén habilitadas, de lo contrario, el microprocesador no saldría nunca del estado «HALT».

#### FREE-MEM: 1F1Ah (7962)

- > DESCRIPCIÓN: Devuelve, en «BC», el número de bytes de memoria ocupados. Si se resta este número de 65536 se obtiene el número de bytes que quedan libres para el Basic. Operación equivalente al comando «FRE» de algunos ordenadores. Para utilizarlo desde Basic, podríamos hacer: PRINT 65536-USR 7962.
- > ENTRADA: Ninguna.
- > SALIDA: «BC» = n.º de bytes ocupados.

#### DE, (DE + 1): 2AEH (10990)

- > DESCRIPCIÓN: Realiza la operación equivalente a la instrucción «LD DE, (DE + 1)» inexistente.
- > ENTRADA: «DE» = Dirección de memoria.
- > SALIDA: «DE» = Contenido de los dos bytes siguientes al señalado por «DE» al entrar en la rutina.
- > REGISTROS ALTERADOS: «DE» y «HL».

- > FUNCIONAMIENTO: Es la dirección contenida en el elemento de la pila de máquina apuntado por «ERRSP», por tanto, es la dirección habitual de retorno de error. Su funcionamiento es el mismo que el de «RST 8» salvo que no limpia la pila de máquina ni el stack del calculador.

#### LINADD: 196Eh (6510)

- > DESCRIPCIÓN: Busca la dirección de una determinada línea de programa Basic.
- > ENTRADA: «HL» = Número de línea a buscar.
- > SALIDA: «HL» = Dirección de la línea buscada o de la siguiente si esa no existiera.
- «DE» = Dirección de la línea anterior a la buscada. Indicador de «cero» a «1» si la línea buscada existe y a «0» si no existe.

- base de la tabla. Es necesario que se haya abierto un stream con «CHAN-OPEN».

- > FUNCIONAMIENTO: La estructura de la tabla ha de ser la siguiente: El primer byte debe ser «80h» (128). A partir de ahí siguen los mensajes que pueden tener cualquier longitud pero el último carácter de cada uno ha de tener el bit 7 a «1» para indicar fin de mensaje.

#### WAIT-KEY: 15DEh (5598)

- > DESCRIPCIÓN: Espera la pulsación de una tecla y retorna cuando se haya pulsado. No retorna si se pulsa sólo «CAPS SHIFT» o «SIMBOL SHIFT».
- > ENTRADA: Ninguna.
- > SALIDA: «A» = Código de la tecla pulsada. Indicador de acarreo a «0».

#### Rutinas para manejar el stack del calculador

Un stack (en castellano, «pila») es un lugar de la memoria donde los datos se guardan de forma que, de cada vez, sólo puede leerse el último que ese metió. Suponga que guarda sus ejemplares de MICROHOBBY en una caja y colocados uno encima de otro. Su caja será una pila. Para acceder al penúltimo ejemplar que metió, deberá retirar primero el último.

El Spectrum utiliza tres pilas. La pila de máquina, que ya se trató ampliamente durante el curso. La pila de GOSUB que almacena el lugar de retorno cuando se hacen llamadas a subrutinas desde Basic. Y finalmente, la pila del calculador (a la que, para evitar confusiones, llamaremos «stack»).



La rutina empieza llamando a «NUMERIC» y retorna si se trata de un número, de lo contrario, continúa en «ALPHA».

#### CHAN-OPEN: 1601h (5633)

- > DESCRIPCION: Hace que la corriente indicada por el contenido de «A» sea la corriente en curso.
- > ENTRADA: «A» = Número de la corriente a abrir.
- > SALIDA: La variable del Sistema «CURCHL» sale apuntando a los datos del canal asignado a esa corriente. Se produce el error «O Invalid Stream» si la corriente no tiene canal asignado.
- > FUNCIONAMIENTO: Comprueba, en la tabla de corrientes (STRMS), si la corriente solicitada tiene canal asignado. Si es así, carga la dirección de ese canal en la variable «CURCHL» (canal en curso) y fija los flags de acuerdo con el canal de que se trate. Los flags que se fijan son los siguientes:

```
CANAL "K": SET 0, (TV-FLAG)
RES 5, (FLAGS)
SET 4, (FLAGS2)
RES 1, (FLAGS)
CANAL "S": RES 0, (TV-FLAG)
RES 4, (FLAGS2)
RES 1, (FLAGS)
CANAL "P": SET 1, (FLAGS)
RES 4, (FLAGS2)
```

- > EJEMPLO: Utilizaremos esta rutina para abrir una corriente antes de realizar cualquier impresión con «RST #10». Para imprimir «MICROHOBBY» en la parte alta de la pantalla, se haría:

```
START LD A, 2
CALL #1601
LD B, 10
LD HL, TEXT0
LD A, (HL)
BUCLE PUSH HL
PUSH BC
RST #10
POP BC
POP HL
INC HL
DJNZ BUCLE
RET
TEXT0 DEFM "MICROHOBBY"
```

#### STMTR1: 1B7Dh (7037)

- > DESCRIPCION: Pone el Basic en ejecución a partir de la línea apuntada por «NEWPPC».
- > ENTRADA: Bit 7 de «FLAGS» a «1» para indicar ejecución.
- «NEWPPC»: Número de línea donde iniciar la ejecución.
- «NSPPC»: Número de comando dentro de la línea donde iniciar la ejecución.
- > FUNCIONAMIENTO: Salta al bucle principal del intérprete de Basic.

#### MAIN 4: 1303h (4867)

- > FUNCIONAMIENTO: Termina la ejecución del Basic imprimiendo un mensaje de error.
- > ENTRADA: El código del mensaje a imprimir ha de estar en la variable del sistema «ERRNR».
- > SALIDA: Detiene la ejecución, imprime el mensaje y entra en el editor de Basic.

- > FUNCIONAMIENTO: Su listado es el siguiente:  
Este es un ejemplo de cómo se pueden utilizar subrutinas para simular instrucciones inexistentes.

```
EX DE, HL
INC HL
LD E, (HL)
INC HL
LD D, (HL)
RET
```

#### HL = HL \* DE: 30A9h (12457)

- > DESCRIPCION: Multiplica «HL» por «DE» y devuelve el resultado en «HL». Simula la instrucción de multiplicar inexistente en el Z-80.
- > ENTRADA: «HL» = Multiplicando.  
«DE» = Multiplicador.
- > SALIDA: «HL» = Resultado. Si este excede de 65535, el contenido de «HL» será indeterminado.
- > REGISTROS ALTERADOS: «AF», «DE» y «HL».
- > FUNCIONAMIENTO: Su listado es el siguiente:

```
PUSH BC
LD B, 16
LD A, H
LD C, L
LD HL, 0
ADD HL, HL
JR C, END
RL C
RLA
JR NC, AGAIN
ADD HL, DE
JR C, END
DJNZ LOOP
AGAIN
END
POP BC
RET
```

Puede constituir un magnífico ejemplo de cómo crear una rutina de multiplicación.

#### KEYBOARD: 02BFh (703)

- > DESCRIPCION: Lee el teclado y devuelve, en «A» y en la variable «LAST-K» el valor de la tecla pulsada, si hubiera alguna. Se decodifica el teclado y se tienen en cuenta los valores de «REPER» y «REPDEL».
- > ENTRADA: Ninguna.
- > SALIDA: «A» y «LAST-K» = Última tecla pulsada. Bit 5 de «FLAGS» = «1» si ha habido pulsación.

#### MAKE-ROOM: 1655h (5717)

- > DESCRIPCION: Hace sitio en memoria desplazando los bloques que sea necesario y actualizando los punteros del Sistema.
- > ENTRADA: «HL» = Dirección siguiente a aquella donde debe empezarse la nueva zona.
- «BC» = Número de bytes que deberá tener de longitud la nueva zona.
- > SALIDA: «HL» = Posición anterior a aquella donde la nueva zona empieza.
- «DE» = Última dirección de la nueva zona.

- > FUNCIONAMIENTO: Empieza por comprobar si hay suficiente sitio en memoria, para lo cual llama a la subrutina «TEST-ROOM» (1F05h). Si es así, actualiza los punteros llamando a la subrutina «POINTERS» (1664h) que sumará «BC» a todos los punteros del Basic que estén apuntando por encima de «HL». Finalmente, utiliza la ins-



trucción LDDR para desplazar la memoria hacia arriba el número de bytes que indique «BC».

#### NEXT-ONE: 19B8h (6584)

> DESCRIPCIÓN: Halla la dirección de comienzo de la línea o variable siguiente a la que esté apuntada por «HL». Por añadidura también halla la longitud de la línea o variable actualmente apuntada.

> ENTRADA: «HL» = Apuntando a la dirección de inicio de una determinada línea o variable.

> SALIDA: «HL» = No sufre modificación.

«DE» = Dirección inicial de la siguiente línea o variable.

«BC» = Longitud de la línea o variable apuntada por «HL».

> FUNCIONAMIENTO: La rutina se limita a leer la longitud de la línea o variable actual y sumársela a «HL» sacando el resultado en «BC». En el caso de variables, se leen los bits identificadores para saber de qué tipo de variable se trata y, por tanto, cuál es su longitud.

> EJEMPLO: La mayor utilidad de esta rutina es para buscar una variable determinada en el área de variables. Para ello, deberemos saber cuál es el byte identificador. Este está compuesto por los tres bits identificadores del tipo de variable más los cinco bits inferiores de la letra que le da nombre. Los bits identificadores son:

010 = Variable numérica cuyo nombre es una sola letra.

101 = Primera letra de una variable numérica cuyo nombre son varias letras.  
111 = Última letra de una variable del tipo anterior.

100 = Matriz de números.  
110 = Variable de control de un bucle FOR-NEXT.  
010 = Variable de cadena.  
110 = Matriz de caracteres.

Según esto, sabemos que, por ejemplo, la variable «H\$» tendrá un byte identificador que será: 010 (porque es una variable de cadena) + 01000 (que son los 5 bits inferiores de la letra «H»), por tanto: H\$ = 01001000 es decir, 72.

Supongamos que queremos buscar la variable H\$ en el área de variables. Empezamos por cargar «HL» con la dirección contenida en «VARS» que nos indica el inicio de la zona de variables. Luego, vamos llamando a «NEXT-ONE» hasta que encontremos la variable que estamos buscando:

100	START	LD	HL, (VARS)
110	BUCLE	LD	A, (HL)
120		CP	72
130		RET	Z
140		CALL	#19B8
150		EX	DE, HL
160		LD	A, (HL)
170		CP	128
180		JR	NZ, BUCLE
190		RST	8
200		DEFB	1
210	VARS	EQU	23627

Empezamos por cargar en «HL» la dirección de la primera variable y entramos en un bucle comprendido entre las líneas 110 y 180. En el bucle empezamos por comparar el contenido de la dirección apuntada por «HL» con 72 que es el identificador de la variable que estamos buscando. Si la comparación da «0», retornamos sin más. En caso contrario, pasamos a «HL» la dirección de

> FUNCIONAMIENTO: Su listado es el siguiente:

ALPHA	CP	#41
	CCF	
	RET	NC
	CP	#5B
	RET	C
	CP	#61
	CCF	
	RET	NC
	CP	#7B
	RET	

El mismo sistema que en el caso anterior, pero comprobando dos intervalos. Obsérvese que el indicador retorna con valores contrarios a los del caso anterior.

#### ALPHANUM: 2C88h (11400)

> DESCRIPCIÓN: Es una combinación de las anteriores. Comproba si el contenido de «A» es el código ASCII de un dígito o de una letra.

> ENTRADA: Código en «A».

> SALIDA: Indicador de acarreo a «1» si el código corresponde a un número o a una letra, y a «0» en caso contrario.

> REGISTROS ALTERADOS: Ninguno.

> FUNCIONAMIENTO: Su listado es el siguiente:

ALPHANUM	CALL	NUMERIC
	CCF	
	RET	C
		.....
		.....
ALPHA		

la siguiente variable llamando a «NEXT-ONE» e intercambiando «HL» con «DE». Antes de cerrar el bucle, comprobamos si hemos alcanzado el final del área de variables, lo que se comprueba comparando el contenido de la dirección apuntada por «HL» con «128» que es el indicador de fin de la zona de variables. Si esta comparación diera «0», se detiene la ejecución y se imprime el mensaje «Variable not found».

#### NUMERIC: 2D1Bh (11547)

> DESCRIPCIÓN: Comprueba si el contenido del acumulador es el código ASCII de un carácter numérico.

> ENTRADA: Código en «A».

> SALIDA: Indicador de acarreo a «0» si el código corresponde a un carácter numérico, y a «1» si no es así.

> REGISTROS ALTERADOS: Ninguno.

> FUNCIONAMIENTO: Su listado es el siguiente:

NUMERIC	CP	#30
	RET	C
	CP	#3A
	CCF	
	RET	

Más sencillo, imposible.

#### ALPHA: 2C8D (11405)

> DESCRIPCIÓN: Comprueba si el contenido del acumulador es el código ASCII de una letra del alfabeto.

> ENTRADA: Código en «A».

> SALIDA: Indicador de acarreo a «1» si el código corresponde a una letra, y a «0» si no es así.

> REGISTROS ALTERADOS: Ninguno.



#### «08h» NO-&NO

Opuesto al anterior, realiza un «AND» lógico entre los dos números y devuelve el primero si el segundo no es cero y el valor «0» en caso contrario.

#### «09h» NO-L-EQL

#### «0Ah» NO-GR-EQ

#### «0Bh» NOS-NEQ

#### «0Ch» NO-GRTR

#### «0Dh» NO-LESS

#### «0Eh» NOS-EQL

Estos seis literales realizan las seis comparaciones binarias posibles entre números (menor/igual, mayor/igual, diferente, mayor, menor e igual) devolviendo un valor de verdad (cierto o falso) que será usado por los saltos condicionales.

#### «0Fh» ADDITION

Realiza la suma de dos números devolviendo el resultado como última entrada del stack.

#### «10h» STR-&NO

Realiza un «AND» lógico entre una cadena (sus parámetros serán los que deban estar en el stack) y un número. Devuelve la misma cadena si el número es distinto de cero y la cadena nula en caso contrario. Los parámetros de la cadena son la última entrada en el stack.

#### «11h» STR-L-EQL

#### «12h» STR-GR-EQ

#### «13h» STRS-NEQL

#### «14h» STR-GRTR

#### «15h» STR-LESS

#### «16h» STRS-EQL

En este caso, se realizan las seis comparaciones binarias posibles entre cadenas (las mismas que en el caso de los números). Al igual que en todas las operaciones con cadenas, serán los parámetros de éstas los que deberán estar presentes en el stack.

#### «17h» STRS-ADD

Realiza la concatenación de las dos cadenas cuyos parámetros se encuentren en el stack. El resultado es el último dato del stack que representa los parámetros de una cadena construida en el área de trabajo.

#### «18h» VAL\$

Realiza la operación equivalente a «VAL\$» en Basic, es decir, evalúa una expresión de cadena, devolviendo el resultado como otra cadena construida en el área de trabajo y cuyos parámetros están en lo alto del calculador.

#### «19h» USR-\$

Devuelve la dirección en memoria del UDG correspondiente al primer carácter de una cadena.

#### «1Ah» READ-IN

Equivalente a la función «INKEY\$» del Basic, devuelve una cadena (en realidad, un solo carácter) que ha sido leída desde la corriente (stream) que estuviese abierta.

#### «1Bh» NEGATE

Cambia el signo del número que se halle presente en la parte alta del stack. Si éste es «0», no resulta afectado.

#### «1Ch» CODE

Devuelve el código ASCII del primer carácter de una cadena cuyos parámetros estén en el stack. Si se tratase de una cadena nula, se devuelve un cero.

#### «1Dh» VAL

Al igual que la función «VAL» del Basic, evalúa una cadena devolviendo un resultado numérico si ello fuera posible.

#### «1Eh» LEN

Devuelve un número que es la longitud de la cadena cuyos parámetros se encuentren en el stack.

```
STK_STO_$      RES 6, (FLAGS)
STK_STORE      PUSH BC
                CALL TEST_5_SP
                POP  BC
                LD  HL, (STKEND)
                LD  (HL), A
                INC HL
                LD  (HL), E
                INC HL
                LD  (HL), D
                INC HL
                LD  (HL), C
                INC HL
                LD  (HL), B
                INC HL
                LD  (STKEND), HL
                RET
TEST_5_SP      EQU  #33A9
```

El bit 6 de «FLAGS» se pone «0» para indicar que el dato alto del stack es una cadena. Si no se desea así, se puede entrar por «STK-STORE». Llamando a «TEST-5-SP» se comprueba que exista sitio en memoria para 5 bytes más. Esta rutina se llama cada vez que se necesite añadir un nuevo dato al stack. Hace uso de la subrutina «TEST-ROOM» que vimos antes, y su listado es el siguiente:

```
TEST_5_SP      PUSH DE
                PUSH HL
                LD  BC, 5
                CALL TEST_ROOM
                POP  HL
                POP  DE
                RET
TEST_ROOM      EQU  #1F05
```

Por lo demás, la rutina es de sencilla comprensión.

#### STK-FETCH: 2BF1h (11249)

- > DESCRIPCION: Saca del stack los parámetros de una cadena.
- > ENTRADA: Parámetros en el stack.
- > SALIDA: «A» = Flag.  
«BC» = Longitud.  
«DE» = Dirección.  
La variable «STKEND» resulta decrementada cinco veces.
- > FUNCIONAMIENTO: Su listado es el siguiente:

```
STK_FETCH      LD  HL, (STKEND)
                DEC HL
                LD  B, (HL)
                DEC HL
                LD  C, (HL)
                DEC HL
                LD  D, (HL)
                DEC HL
                LD  E, (HL)
                DEC HL
                LD  A, (HL)
                LD  (STKEND), HL
                RET
```

El listado es tan sencillo que no creamos que requiera comentario alguno.

#### NUMEROS

Los números que se almacenen en el stack pueden ser de dos tipos, enteros o números en coma flotante. En Assembler no tienen mucho sentido los números no enteros, por lo que, habitualmente, utilizaremos números enteros en nuestros cálculos. No obstante, el Sistema Operativo está preparado para trabajar con ambos tipos de números y nosotros podremos aprovecharnos de esa posibilidad.

Un número en coma flotante (en inglés, «FP» abreviatura de «Floating



Point») se almacena como una mantisa elevada a un exponente. Tanto la mantista como el exponente están en binario y en complemento a 2, por lo que su primer bit, en cada caso, es el bit de signo. Se utilizan 8 bits (1 byte) para el exponente y 32 bits (4 bytes) para la mantisa. El primer byte es el exponente. Este byte nunca puede ser «0» ni «FF» y de esta forma, el calculador diferencia si se trata de un entero o de un número en coma flotante (si el primer byte fuera «0» el número sería un entero positivo y si fuera, «FF» se trataría de un entero negativo).

Un número entero (en lo que respecta al calculador) es aquel cuyo valor absoluto es entero y menor de 65536. Es decir, es número entero todo aquel cuyo valor absoluto cabe en dos bytes. En estos casos, el primer byte es el signo, el segundo es «0», los bytes tercero y cuarto contienen el número en el formato habitual del Z-80 (el menos significativo primero) y, finalmente, el último byte es también «0». El byte de signo es «0» para positivo y «FF» para negativo.

Como ejemplo, veamos la representación de las constantes usadas por el calculador:

```
"0" = 00 00 00 00 00 00 (ENTERO)
"1" = 00 00 01 00 00 00 (ENTERO)
"1/2" = 00 00 00 00 00 00 (COMA FLOTANTE)
"PI/2" = 81 49 0F DA A2 (COMA FLOTANTE)
"10" = 00 00 0A 00 00 00 (ENTERO)
```

Hay una serie de subrutinas que nos van a permitir meter y sacar números del stack.

#### STK-DIGIT: 2D22h (11554)

- > DESCRIPCION: Si el contenido del acumulador es el código ASCII de un dígito, se mete en el stack. Si no, se retorna sin más.
- > ENTRADA: Código ASCII de un dígito en el acumulador.

- > SALIDA: Dato en el stack.  
«HL» = apuntado a la entrada anterior (STKEND-5).

#### STACK-A 2D28h (11560)

- > DESCRIPCION: Mete el contenido de «A» en el stack.
- > ENTRADA: Dato en el acumulador.
- > SALIDA: Dato en el stack.  
«HL» = apuntando a la entrada anterior (STKEND-5).
- > FUNCIONAMIENTO: Veamos el listado de esta rutina y las dos anteriores, ya que trabajan en cadena:

#### STKACK-BC: 2D2Bh (11563)

- > DESCRIPCION: Mete el contenido de «BC» en el stack.
- > ENTRADA: Dato en «BC».
- > SALIDA: Dato en el stack.  
«HL» = apuntando a la entrada anterior (STKEND-5).

- > FUNCIONAMIENTO: Veamos el listado de esta rutina y las dos anteriores, ya que trabajan en cadena:

```
STK_DIGIT  CALL NUMERIC
            RET C
            SUB #30
            EQU #2D1B
            LD C,A
            LD B,0
            LD IY,#5C3A
            XOR A
            LD E,A
            LD D,C
            LD C,B
            LD B,A
            CALL STK_STORE
            RST #28
            DEFB #3B
            AND A
            RET
            EQU #2AB6
```

función del contenido del stack, por ello, se puede decir que el calculador de la ROM es una auténtica «calculadora programable», ya que la secuencia de literales se comporta como un verdadero «programa».

Las operaciones que puede realizar el calculador son de tres tipos: Binarias, Unitarias y de Manipulación.

Las operaciones binarias son aquellas que se realizan entre los dos elementos superiores del stack. Recuerde que cada elemento puede ser un número o los parámetros de una cadena. Un ejemplo de operación binaria es la multiplicación que veíamos anteriormente. (Se llaman operaciones binarias porque se realizan entre dos elementos, no porque se realicen en sistema de numeración binario. Todas las operaciones del calculador se realizan en decimal y en coma flotante).

Las operaciones unitarias son aquellas que se realizan sobre un solo elemento del stack, por ejemplo, la operación «SIN» que halla el seno de número que se encuentre en la parte alta del stack.

Por último, las operaciones de manipulación son aquellas que no implican cálculos, por ejemplo: copiar un número del stack en una de las memorias, realizar un salto, tomar una decisión (salto condicional) o meter un determinado valor en el stack.

En caso de querer realizar una sola operación, puede utilizarse el literal «3B» colocando el literal de la operación a realizar en el registro «B».

A continuación, estudiaremos los literales del calculador, uno por uno.

#### Literales del calculador

##### «00h» JUMP-TRUE

Se trata de un salto relativo. Se ejecuta si el tercer byte del número almacenado en el stack es distinto de cero.

El literal que le siga deberá ser el desplazamiento del salto, al igual que en las instrucciones de salto relativo del Z-80.

##### «01h» EXCHANGE

Intercambia entre sí los dos números de la parte alta del stack. El primero pasa a ser el segundo y el segundo pasa a ser el primero.

##### «02h» DELETE

Borra el dato superior del stack. En realidad, lo único que hace es restar 5 del puntero del stack.

##### «03h» SUBSTRACT

Resta el segundo número del stack (minuyendo) del primero (sustraendo). En realidad, se limita a cambiar el signo del minuendo y sumarlos. El resultado se obtiene —como siempre— en lo alto del stack.

##### «04h» MULTIPLY

Multiplica, entre sí, los dos números de la parte alta del stack y deja el resultado en lo alto de éste.

##### «05h» DIVISION

Divide el primer número del stack (dividendo) entre el segundo (divisor) y deja el resultado en la parte alta del stack. Se produce «overflow» si el divisor es cero.

##### «06h» TO-POWER

Eleva el primer número a la potencia indicada por el segundo. Al igual que en todas las calculadoras, esta operación se realiza como el antilogaritmo del resultado de multiplicar el exponente por el logaritmo de la base. Por lo cual, no admite exponentes negativos.

##### «07h» OR

Realiza un «OR» lógico entre el primer número y el segundo, devolviendo el primer número si el segundo es «0» y el valor «1» en caso contrario.



INT-2» trabajan encadenadas y su listado es el siguiente:

```

FIND_INT_1 CALL FP_TO_A
JR FIND
FIND_INT_2 CALL FP_TO_BC
FIND JR C, REP_B
RET Z
REP_B RST B
DEFB #0A

```

Si se entra por «FIND-INT-1», se llama a «FP-TO-A» y si se hace por «FIND-INT-2», se llama a «FP-TO-BC». En ambos casos, se continúa en «FIND». Si hay acarreo, se salta a «REP-B» para producir el error «B». A continuación se retorna si el número es positivo. Si fuera negativo (indicador de cero a «0»), se continuaría en «REP-B» con lo que también se produce el error «B».

### El calculador de la ROM

El calculador del Spectrum puede considerarse como un programa aparte dentro del Sistema Operativo del ordenador. Su funcionamiento es similar al de una calculadora programable, salvo que, en lugar de pantalla tiene un stack y en lugar de teclas, números colocados en la memoria del ordenador que actúan a modo de instrucciones, diciéndole las operaciones que debe realizar con los datos del stack. Estos datos podrán introducirse en el stack utilizando las rutinas vistas hasta ahora. Los resultados de las operaciones quedarán en la parte alta del stack y podrán ser leídos de ahí utilizando, también, estas rutinas.

Para entrar en el calculador, utilizaremos la instrucción «RST #28» y a continuación de ella irán una serie de literales que servirán para indicarle al calculador las operaciones que deberá realizar. El último de estos literales deberá

ser «38h» que significa «salir del calculador». Veámoslo más claro con un ejemplo:

Supongamos que tenemos dos datos en lo alto del stack y queremos multiplicarlos. El resultado lo obtendremos, de nuevo, en lo alto del stack. Al entrar tenemos dos datos y al salir tendremos uno, por lo que el stack se habrá decrementado una vez, es decir, cinco bytes. La secuencia de instrucciones sería:

```

RST #28 ; Calculador.
DEFB #04 ; Multiplicar.
DEFB #38 ; Terminar.

```

Como ya habrá adivinado más de un lector, el literal «04h» es, precisamente, el de multiplicar. Hay un total de 66 literales (desde el «00h» hasta el «41h») para las distintas operaciones que puede realizar el calculador. Luego los veremos todos.

Además del stack, el calculador utiliza 6 memorias numeradas de «0» a «5» para almacenamiento temporal de algunos datos, y una posición de memoria que actúa como contador en los bucles y que se denomina «BREG» ya que cumple la misma función que el registro «B» del microprocesador.

Por otro lado, el calculador posee 5 constantes que pueden ser introducidas en el stack mediante el uso de un literal. Estas constantes son: «0», «1», «1/2», «1/2» y «10».

Algunos literales llevan parámetros, por ejemplo, en los literales que implican un salto, el siguiente literal se tomará como un parámetro que indicará el número de literales a saltar en la secuencia de instrucciones. Su funcionamiento es similar al de los saltos relativos en el Z-80. También existen saltos condicionales que se ejecutan o no en

«STK-DIGIT» utiliza la subrutina «NUMERIC» (vista anteriormente) para comprobar si se trata de un dígito. Si es así, continúa en «STACK-A», si no, retorna con el indicador de acarreo a «1».

«STACK-A» se limita a copiar el contenido de «a» en «BC» y continuar en «STACK-BC».

«STACK-BC» empieza por restaurar el valor de «Y» para que apunte a la variable «ERRNR». Esto se hace así porque «STACK-BC» es el punto de retorno de las llamadas a rutinas de usuario (USR) y se prevé la posibilidad de que este valor haya sido corrompido. Resulta bastante absurdo, ya que si una subrutina de usuario corrompiera el valor de «Y», la rutina de respuesta a las interrupciones no podría actualizar «FRAMES» adecuadamente. Por el contrario, no se restablece el valor de «HL» que también es importante que no sea corrompido en una subrutina de usuario. Este registro apunta, siempre, al siguiente literal a ejecutar por el calculador. Recuerde que, cuando hemos utilizado «HL» en un ejemplo, nos hemos cuidado mucho de preservar su valor y recuperarlo antes de retornar. El registro «HL» puede utilizarse en una subrutina de usuario siempre que se preserve su valor y se recupere antes de retornar. Por el contrario, el registro «Y» no debe alterarse, a menos que se cambie el vector de interrupción. Si se retorna con RET, el contenido de «Y» será reinicializado. No así si se retorna de otro modo (por ejemplo, con RST #08). Sigamos con «STACK-BC». Lo siguiente que se hace es poner «A», «E» y «B» a «0» y copiar «B» en «C» y «C» en «D» para terminar llamando a la rutina «STK-STORE» que almacenará todos estos registros en el orden correcto (observe que «A» será siempre «0» por lo que el número almacenado será siempre un entero positivo). La llamada a RST #28 con el literal «38h» es un «truco» para conseguir que «HL» salga

apuntando a (STKEND-5). Lo que ese hace es llamar al calculador (con RST #28) y salir de él con el literal «38h»; no se realiza ninguna operación, pero se consigue el efecto deseado. Hubiera sido más rápido cargar «STKEND» en «HL» y restarle 5, pero hubiera ocupado más memoria y el Sistema Operativo del Spectrum está escrito pensando más en el ahorro de memoria que en la velocidad de proceso —lo curioso es que les ha sobrado más de 1K de memoria ROM (entre 386Eh y 3CFFh está toda a «FF») y les ha quedado un S.O. bastante lento—. Lo último que se hace antes de retornar es poner el indicador de acarreo a «0» con «AND A». De esta forma, cuando entremos por «STK-DIGIT», sabremos si el dígito ha sido almacenado, ya que en ese caso, el indicador de acarreo retorna a «0» y en caso contrario retorna a «1».

### FP-TO-BC: 2DA2h (11682)

- > DESCRIPCION: Coge un número del stack, lo redondea a su valor entero más próximo y copia el resultado en «BC». El octeto inferior del número se copia, también en «A». Si el resultado es mayor —en valor absoluto— de 65535, se retorna con el indicador de acarreo a «1». Si el resultado es negativo, se retorna con el indicador de cero a «0».
- > ENTRADA: Dato en el stack del calculador.
- > SALIDA: «BC» = Número leído del stack.  
«A» = Octeto inferior de este número.  
Indicador de acarreo a «1» si el número está fuera de rango. Indicador de cero a «1» si el número es positivo.



#### FP-TO-A: 2DD5h (11733)

- > DESCRIPCION: Hace lo mismo que «FP-TO-BC» salvo que, esta vez, el número tiene que ser menor —en valor absoluto— de 256.
  - > ENTRADA: Dato en el stack del calculador.
  - > SALIDA: «A» = Número leído del stack.
- Indicador de acarreo a «1» si el número está fuera de rango. Indicador de cero a «1» si el número es positivo.
- > FUNCIONAMIENTO: Utiliza «FP-TO-BC» como subrutina. Su listado es el siguiente:

```
FP_TO_A CALL FP_TO_BC
RET C
PUSH AF
DEC B
INC B
JR Z, END
POP AF
SCF
RET
POP AF
RET
```

Empieza por llamar a «FP-TO-BC» para obtener una copia del octeto menos significativo en «A» y retorna en caso de que hubiera acarreo indicando que el número es mayor de 65535. A continuación preserva el registro «A» y los indicadores, y comprueba si «B» es cero, para ver si el número es menor de 256. Si es así, salta a «END» donde recupera «A» y los indicadores y retorna. En caso contrario, también recupera «A» y los indicadores, pero pone a «1» el indicador de acarreo antes de retornar, para indicar que el número está fuera de rango.

#### STK-TO-A: 2314h (8980)

- > DESCRIPCION: Haciendo uso de las rutinas anteriores, lee del stack del calculador, un número cuyo valor absoluto no exceda de 255, y lo devuelve en el registro «A». El registro «C» retorna con «1» si el número es positivo y con «FF» si es negativo (a estos efectos, el número «0» se considera positivo). Se produce el error: «B Integrar out of range» si el número está fuera de rango.
- + ENTRADA: Dato en el stack del calculador.
- > SALIDA: «A» = Número leído del stack.  
«C» = Signo del número.
- > FUNCIONAMIENTO: Utiliza la subrutina «FP-TO-A» por lo que puede constituir un magnífico ejemplo de cómo usar esta rutina. Su listado es el siguiente.

```
STK_TO_A CALL FP_TO_A
JP C, ERR_B
LD C, 1
RET Z
LD C, #FF
RET
*****
ERR_B RST B
DEFB #0A
```

Empieza por llamar a «FP-TO-A» y salta a «ERR-B» si hay acarreo indicando que el número está fuera de rango. A continuación, carga «C» con «FFh» para indicar signo negativo y retorna. La etiqueta «ERR-B» se encuentra en la dirección 24F9h (9465), ya que este mensaje de error es usado por varias subrutinas.

#### STK-TO-BC: 2307h (8967)

- > DESCRIPCION: Complementaria de la anterior y haciendo uso de ella, lee dos números del stack —cuyo valor absoluto deberá ser menor de 256— y los coloca en «B» y «E». El número que ocupaba el lugar más alto del stack (última entrada) será el que se cargue en «B» y el siguiente, el que se cargue en «C». La utilidad de esta rutina es para leer los parámetros de aquellos comandos que necesitan dos argumentos, por ejemplo: PLOT, DRAW, etc.
- ENTRADA: Dos datos en el stack del calculador.
- > SALIDA: «B» = Número más alto del stack.  
«D» = Signo de «B»  
 («1» = posit.  
 «FF» = negat.)  
«C» = Significante número del stack.  
«E» = Signo de «C»  
 («1» = posit.  
 «FF» = negat.)  
Error «B» si alguno de los dos números está fuera de rango.
- > FUNCIONAMIENTO: Como era de esperar, utiliza dos llamadas a «STK-TO-A». Su listado es el siguiente:

```
STK_TO_BC CALL STK_TO_A
LD B, A
PUSH BC
CALL STK_TO_A
LD E, C
POP BC
LD D, C
LD C, A
RET
```

Para entender el listado, tenga en cuenta que en cada llamada a «STK-TO-A» se retorna con el número «A» y su signo en «C». La rutina se limita a hacer dos de estas llamadas y transferir los datos para que queden colocados en «BC» y «DE». Por añadidura, el registro «A» contendrá, a la salida, lo mismo que el «C».

#### FIND-INT-1: 1E94h (7828)

- > DESCRIPCION: Al igual que «FP-TO-A», lee un número del stack. Pero esta vez, se produce el error «B» tanto si el número está fuera de rango como si es negativo. Por tanto, el número leído no puede ser menor de «0» ni mayor de «225».
- + ENTRADA: Dato en el stack del calculador.
- > SALIDA: «A» = Número que ha sido leído del stack.  
Error «B» si es mayor de 255 o menor de «0».
- > FUNCIONAMIENTO: (Ver «FIND-INT-2»).

#### FIND-INT-2: 1E99 (7833)

- > DESCRIPCION: De la misma forma que «FP-TO-BC», lee un número del stack (cuyo valor absoluto sea menor de 65536). Pero esta vez, se produce el error «B» tanto si el número está fuera de rango como si es negativo. El número leído deberá ser menor de «65536» y mayor de «0».
- > ENTRADA: Dato en el stack del calculador.
- > SALIDA: «BC» = Número que ha sido leído del stack.  
Error «B» si es mayor de «65535» o menor de «0».
- > FUNCIONAMIENTO: Las dos rutinas «FIND-INT-1» y «FIND-



## Capítulo 10

### INSTRUCCIONES DE MANEJO DE BITS

Introducción .....	258
Instrucciones de prueba de bits (BIT b,r; BIT b,(IX+d); BIT b,(IY+d)) .....	258
Instrucciones de activar bits (SET b,r; SET b,(HL); SET b,(IX+d); SET b,(IY+d)) .....	260
Instrucciones de borrar bits (RES b,r; RES b,(HL); RES b,(IX+d); RES b,(IY+d)) .....	262
Tablas de codificación .....	263
Ejemplos .....	269
Ejercicios .....	279
Soluciones a los ejercicios .....	280

## Capítulo 11

### GRUPO DE INSTRUCCIONES DE LLAMADA Y RETORNO

Subrutinas .....	281
Instrucciones de llamada (CALL nn; CALL cc,nn) .....	286
Instrucciones de retorno (RET; RET cc; RETI; RETN) .....	288
Reinicios de página cero (RST p) ..	291
Tablas de codificación .....	293
Ejemplos .....	293
Ejercicios .....	328
Soluciones a los ejercicios .....	329

## Capítulo 12

### GRUPO DE INSTRUCCIONES DE ENTRADA Y SALIDA

Introducción .....	330
Instrucciones de entrada (IN A,(n); IN r,(C); INI; INIR; INDR) .....	332
Instrucciones de salida (OUT (n),A; OUT (C),r; OTIR; OTDR; OTDR) ..	337
Tablas de codificación .....	342
El teclado del Spectrum .....	342
Ejemplos .....	346
Ejercicios .....	355
Soluciones a los ejercicios .....	355

## Capítulo 13

### GRUPO DE INSTRUCCIONES DE CONTROL DE CPU

Instrucciones de control (NOP; HALT) ..	357
Las interrupciones .....	358
Instrucciones relativas a las interrupciones (DI; EI; IM 0; IM 1; IM 2) ..	360
Tablas de codificación .....	362
Ejemplos .....	362
Ejercicios .....	370
Soluciones a los ejercicios .....	371
REPERTORIO ALFABETICO DE INSTRUCCIONES .....	372

## Capítulo 14

### MANEJO DE ENSAMBLADORES

Introducción .....	380
Etiqueta .....	380
Contador de posiciones .....	381
Expresiones .....	381
Directivos del ensamblador (EQU; DEFB; DEFW; DEFS; DEFM; IF; ELSE; END) .....	381
Comandos del ensamblador (E; Hs; S; L-; L+; D-; D+; C-; C+; F) ..	382
Editor .....	382
Comandos del editor .....	383
Ensamblaje y puesta en marcha ..	384
Comandos de cinta .....	384
Comandos de microdrive .....	384
Otros comandos .....	384
TABLAS DE MANEJO DEL GENS-3 .....	385

## Capítulo 15

### SUBROUTINAS DE LA ROM

Introducción .....	387
Rutinas de control de pantalla .....	388
Rutinas de cassette y sonido .....	389
Rutinas de uso general .....	393
Rutinas para manejar el stack del Calculador .....	399
El Calculador de la ROM .....	406
Literales del Calculador .....	407
APENDICE INSTRUCCIONES ESPECIALES .....	412
INDICE .....	414

«1Fh» SIN
«20h» COS
«21h» TAN
«22h» ASN
«23h» ACS
«24h» ATN

Se trata de las funciones trigonométricas del calculador (seno, coseno, tangente, arco-seno, arco-coseno y arco-tangente respectivamente). En el caso de las tres primeras, la entrada ha de ser en radianes. En el caso de las tres últimas, la salida es, asimismo, en radianes.

#### «25h» LN

Devuelve el logaritmo neperiano del número que se encuentre en el stack.

#### «26h» EXP

Devuelve el «antilogaritmo» o función exponencial ( $e^x$ ) del número que se encuentre en el stack.

#### «27h» INT

Devuelve la parte entera del número que se encuentre en el stack.

#### «28h» SQR

Devuelve la raíz cuadrada del número presente en el stack.

#### «29h» SGN

Devuelve el signo ( $-1, 0$  ó  $+1$ ) del número presente en el stack.

#### «2Ah» ABS

Elimina el signo (lo cambia a positivo) del número presente en el stack. Es decir, devuelve su valor absoluto.

#### «2Bh» PEEK

Devuelve el contenido de la posición de memoria cuya dirección es el último valor del stack.

#### «2Ch» IN

Realiza un «IN» a nivel de micropro-

cesador al port direccionado por el dato del stack y devuelve el valor que tuviera el port en ese momento.

#### «2Dh» USR-NO

Realiza un «CALL» a la dirección apuntada por el dato del stack. Devuelve el valor que contenga el registro «BC» en el retorno. Es el sistema por el que, habitualmente, entramos a nuestros programas en C/M desde Basic. La dirección a la que se ha hecho el «CALL» estará presente en el registro «BC» en el momento de entrar en la subrutina. De forma que, si ésta fuera un simple «RET», el dato del stack no se modificaría.

#### «2Eh» STRS

Devuelve la cadena que representa, en ASCII, el número que contuviera el stack. Como siempre, la cadena se construirá en el área de trabajo y sus parámetros serán el resultado que se obtenga en el stack. Puede resultar muy útil para imprimir en ASCII un determinado dato del stack.

#### «2Fh» CHRS

Devuelve una cadena de un solo carácter, que será aquél cuyo código ASCII contuviera el stack.

#### «30h» NOT

Devuelve un valor de «1» si el contenido del stack fuera «0», y un valor de «0» en caso contrario.

#### «31h» DUPLICATE

Duplica el número que se halle en el stack. Por tanto, el stack crece en un elemento (cinco bytes).

#### «32h» N-MOD-M

Halla el módulo «M» del número «N». «M» será la última entrada en el stack y «N» la penúltima. En realidad, lo que hace es dividir «N» entre «M» sin sacar decimales y dar el cociente como últi-



ma entrada en el stack y el resto como penúltima.

### «33h» JUMP

Salta tantos literales (hacia delante o hacia atrás) como indique el literal que lo sigue (en complemento a 2) que se considera como un parámetro. De hecho, se comporta igual que las instrucciones de salto relativo del microprocesador. En este caso, el salto es incondicional.

### «34h» STK-DATA

Sirve para almacenar un dato en el stack. El dato vendrá dado por los literales que le sigan y que serán considerados como parámetros. El primero de ellos será dividido por «40h» (64) y el cociente más 1 determina si siguen 1, 2, 3 ó 4 literales que formarán la mantisa del número, rellenándose los espacios vacíos con ceros. El primer literal se utiliza, también, como exponente tras dividirlo por «40h», a menos que el resto sea «cero», en cuyo caso, se le añade «50h» para formar el verdadero exponente.

### «35h» DEC-JR-NZ

Funciona de la misma forma que la instrucción «DUNZ» del Z-80, salvo que el contador es el pseudo-registro «B» del calculador, es decir, la posición de memoria 23655. La longitud del salto viene indicada, en complemento a 2, por el siguiente literal.

### «36h» LESS-0

Devuelve un valor de «1» si el valor del stack es menor de cero y un valor de «0» en caso contrario.

### «37h» GREATER-0

Devuelve un valor de «1» si el valor del stack es mayor de cero y un valor de «0», en caso contrario.

## 410 CODIGO MAQUINA

### «39» END-CALC

Termina el cálculo y devuelve el control a la secuencia de programa. En este momento, el resultado del cálculo deberá estar en el stack para poder ser leído desde allí.

### «39h» GET-ARGT

Este literal realiza una complicada reducción con el argumento «X» de «SIN X» o «COS X». Devolviendo un valor «V» que cumpla las dos propiedades siguientes:

$$\begin{aligned} -1 < V &\leq 1 \\ \text{SIN} (PI * V / 2) &= \text{SIN } X \end{aligned}$$

La reducción se hace a través de un valor «Y» intermedio de la siguiente forma:

$$\begin{aligned} Y &= X / (2 * PI) - \text{INT} (X / (2 * PI) + 0.5) \\ \text{Si } -1 < Y &\leq 1 \text{ Entonces } V = 4 * Y \\ \text{Si } 1 < 4 * Y &< 2 \text{ Entonces } V = 2 - 4 * Y \\ \text{Si } -2 < 4 * Y &< -1 \text{ Entonces } \\ V &= -4 * Y - 2 \end{aligned}$$

### «3Ah» TRUNCATE

Redondea un número a su entero más próximo.

### «3Bh» FP-CALC-2

Se utiliza para realizar una operación aritmética única. El literal que indica la operación a realizar, se pasa a través del pseudo-registro «B» del calculador (variable «BREG» del Sistema).

### «3Ch» E-TO-FP

Devuelve un valor que es el resultado de convertir un número en notación exponencial (científica) a coma flotante. La notación exponencial es «xEm» donde «x» es la última entrada del stack y «m» está en el registro «A».

### «3Dh» RE-STACK

Transforma un número que se en-

## Capítulo 6

### INSTRUCCIONES ARITMETICAS Y LOGICAS

Introducción ..... 76

Grupo de instrucciones aritméticas para 8 bits (ADD A,r; ADD A,n; ADD A,(HL); ADD A,(IX+d); ADD A,(IY+d); ADC A,r; ADC A,n; ADC A,(HL); ADC A,(IX+d); ADC A,(IY+d); SUB r; SUB n; SUB (HL); SUB (IX+d); SUB A,(IY+d); SBC A,r; SBC A,n; SBC A,(HL); SBC A,(IX+d); SBC A,(IY+d)) ..... 77

Ejemplos ..... 94  
Grupo de incremento y decremento para 8 bits (INC r; INC (HL); INC (IX+d); INC (IY+d); DEC r; DEC (HL); DEC (IX+d); DEC (IY+d)) ..... 97

Grupo de instrucciones lógicas (AND r; AND n; AND (HL); AND (IX+d); AND (IY+d)) ..... 103  
Control de paridad ..... 107

Grupo de instrucciones lógicas (cont.) (OR r; OR n; OR (HL); OR (IX+d); OR (IY+d); XOR r; XOR n; XOR (HL); XOR (IX+d); XOR (IY+d)) ..... 113

Máscaras ..... 122  
Grupo de instrucciones de comparación (CP r; CP n; CP (HL); CP (IX+d); CP (IY+d)) ..... 123

Grupo aritmético de 16 bits (ADD HL,ss; ADC HL,ss; SBC HL,ss; ADD IX,pp; ADD IY,rr) ..... 126

Grupo de incremento y decremento para 16 bits (INC ss; INC IX; INC IY; DEC ss; DEC IX; DEC IY) ..... 130

Grupo de instrucciones aritméticas de uso general (CPL; NEG; CCF; SCF; DAA) ..... 133

Ejemplos ..... 140

Ejercicios ..... 145

Soluciones a los ejercicios ..... 147

## Capítulo 7

### INSTRUCCIONES DE CAMBIO DE SECUENCIA

Introducción ..... 149

Instrucciones de salto absoluto (JP n; JP cc,nn) ..... 149

Instrucciones de salto relativo (JR e; JR c,e; JR NC,e; JR Z,e; JR NZ,e; DJNZ e) ..... 151

Bucles ..... 155  
Instrucciones de salto indirecto (JP (HL); JP (IX); JP (IY)) ..... 157  
Ejemplos ..... 159  
Ejercicios ..... 171  
Soluciones a los ejercicios ..... 172

## Capítulo 8

### INSTRUCCIONES DE INTERCAMBIO DE TRANSFERENCIA Y BUSQUEDA

Grupo de instrucciones de intercambio (EX DE,HL; EX AF,AF; EX; EX (SP),HL; EX (SP),IX; EX (SP),IY) ..... 173

Grupo de instrucciones de transferencia (LDI; LDIR; LDD; LDDR) ..... 177

Grupo de instrucciones de búsqueda (CPI; CPIR; CPD; CPDR) ..... 182

Tablas de codificación ..... 188

Métodos de búsqueda ..... 189

Ejemplos ..... 190

Ejercicios ..... 203

Soluciones a los ejercicios ..... 204

## Capítulo 9

### INSTRUCCIONES DE ROTACION Y DESPLAZAMIENTO

Introducción ..... 206

Instrucciones de rotación (RLCA; RL; RRCA; RRA; RLC r; RLC (HL); RLC (IX+d); RLC (IY+d); RL r; RL (HL); RL (IX+d); RL (IY+d); RRC r; RRC (HL); RRC (IX+d); RRC (IY+d); RR r; RR (HL); RR (IX+d); RR (IY+d)) ..... 206

Tablas de codificación ..... 219

Instrucciones de desplazamiento (SLA r; SLA (HL); SLA (IX+d); SLA (IY+d); SRA r; SRA (HL); SRA (IX+d); SRA (IY+d); SRL r; SRL (HL); SRL (IX+d); SRL (IY+d); RLD; RRD) ..... 219

Tablas de codificación ..... 230

Multiplicación y división con instrucciones de rotación y desplazamiento ..... 230

Los archivos de pantalla y atributos ..... 236

Ejemplos ..... 241

Los canales de comunicación ..... 251

Ejercicios ..... 256

Soluciones a los ejercicios ..... 257

## CODIGO MAQUINA 415



# INDICE

I	Introducción
III	Manejando una calculadora
V	Algebra de Boole
VIII	Ejercicios
2	PRESENTACION

## Capítulo 1

### CODIGO MAQUINA Y ASSEMBLER

3	Lenguaje de máquina
3	Microprocesador imaginario
4	Supuesto
4	Codificación
6	Codificación del supuesto en lenguaje simbólico
6	Intérpretes y ensambladores
6	Ejecución

## Capítulo 2

### SISTEMAS DE NUMERACION

8	Sistema decimal
8	Sistema binario
8	Operaciones aritméticas en binario
10	Números negativos
11	Sistema hexadecimal
13	Conversión entre bases
16	Ejercicios

## Capítulo 3

### EL MICROPROCESADOR Z-80

17	Qué es un microprocesador
18	Registros
18	Registros especiales de 16 bits
19	Registros especiales de 8 bits
19	Registros alternativos
20	Unidad aritmética-lógica
21	Registro de instrucciones
21	Buses
23	Las interrupciones en el Z-80
24	Palabra de datos del Z-80
24	Ciclos y tiempos
26	Modos de direccionamiento
28	Instrucciones del Z-80

## Capítulo 4

### PROGRAMACION EN ASSEMBLER

30	Introducción
----	--------------

30	Realización de un programa
31	Formatos de instrucción en código máquina
32	Necesidad de conocer el código de máquina
32	Formatos de instrucción en lenguaje simbólico
34	Contador de posición
34	Generación de palabras de datos
35	Diagramas de flujo
37	Presentación de las instrucciones
39	Ejecución de código máquina en el Spectrum
40	Codificación hexadecimal
41	Dónde ubicar un programa en código máquina

## Capítulo 5

### INSTRUCCIONES DE CARGA

42	Introducción
42	Grupo de instrucciones de carga en registros (LD r,r; LD r,n; LD r,(HL); LD r,(IX+d); LD r,(IY+d))
45	Grupo de instrucciones de carga en memoria (LD (HL),r; LD (IX+d),r; LD (IY+d),r; LD (HL),n; LD (IX+d),n; LD (IY+d),n)
47	Grupo de instrucciones de carga en registro acumulador (LD A,(BC); LD A,(DE); LD A,(nn); LD A,I; LD A,R) ..
49	Grupo de instrucciones para salvar el registro acumulador (LD (BC),A; LD (DE),A; LD (nn),A; LD I,A; LD R,A) ..
51	Grupo de instrucciones de carga en registros de 16 bits (LD dd,nn; LD IX,nn; LD IY,nn; LD HL,(nn); LD dd,(nn); LD IX,(nn); LD IY (nn))
54	Grupo de instrucciones de carga en memoria, 16 bits (LD (nn),HL; LD (nn),dd; LD (nn),IX; LD (nn),IY) ..
56	Grupo de instrucciones de carga en registro SP (LD SP,HL; LD SP,IX; LD SP,IY) ..
57	Grupo de instrucciones de manejo de pila (PUSH qq; PUSH IX; PUSH IY; POP qq; POP IX; POP IY) ..
61	Una mirada gráfica a la pila
66	Tablas de codificación
70	Carga del registro «PC»
71	Ejemplos
75	Ejercicios

cuentre en formato de entero, en ese mismo número pero en formato de coma flotante.

«86h» SERIES-06  
«88h» SERIES-08  
«8Ch» SERIES-0C

Estos tres literales se dirigen a la misma subrutina que se encarga de realizar los desarrollos en serie necesarios para las funciones logarítmicas y trigonométricas. La rutina resulta imprescindible para el Sistema Operativo, pero de escasa utilidad para el programador.

«A0h» STK-ZERO  
«A1h» STK-ONE  
«A2h» STK-HALF  
«A3h» STK-PI/2  
«A4h» STK-TEN

Estos cinco literales sirven para almacenar en el stack las cinco constantes de que dispone el calculador. Respectivamente: «0», «1», «1/2», «PI/2» y «10».

«C0h» ST-MEM-0  
«C1h» ST-MEM-1  
«C2h» ST-MEM-2

«C3h» ST-MEM-3  
«C4h» ST-MEM-4  
«C5h» ST-MEM-5

Estos seis literales sirven para transferir el dato que se encuentre en la parte alta del stack, a una de las seis memorias de que dispone el calculador.

«E0h» GET-MEM-0  
«E1h» GET-MEM-1  
«E2h» GET-MEM-2  
«E3h» GET-MEM-3  
«E4h» GET-MEM-4  
«E5h» GET-MEM-5

Opuestos a los anteriores, estos literales sirven para recuperar un dato desde una de las memorias y pasarlo al stack, dejándolo en la parte superior.

Hasta aquí, hemos visto las principales rutinas de la ROM del Spectrum que pueden ser utilizadas por el programador. Para aquellos que deseen profundizar en el funcionamiento del Sistema Operativo, puede ser de gran ayuda el libro de los doctores Ian Logan y Frank O'Hara: «The Complete Spectrum ROM Disassembly», Ed. Melbourne House, 1983.



# INSTRUCCIONES ESPECIALES

A lo largo de los anteriores capítulos, hemos ido viendo todo el juego de instrucciones del microprocesador, Z-80, al menos el juego de instrucciones que figura en los manuales del fabricante. No obstante, el Z-80 puede hacer algo más de lo que hemos visto hasta ahora.

Este microprocesador fue concebido como una versión mejorada del, entonces, más popular microprocesador de 8 bits: el 8080. Para ello, se le añadió un set de registros alternativos y algunas instrucciones más. También se le añadieron los registros índice «IX» e «IY» para darle la posibilidad de trabajar con direccionamiento indexado. En este modo de direccionamiento, se utilizan los mismos códigos de operación que para el direccionamiento indirecto a través del registro «HL», pero precedidos de «DDh» si utilizan el índice «IX», o

de «FDh» si utilizan el «IY». Probablemente más de un lector ya se haya dado cuenta de esta particularidad. Si desea comprobarlo, elija una instrucción al azar, por ejemplo, «LD A, (HL)» cuyo código de operación es «7Eh». Si le añadimos «DDh» se convierte en «DDh 7Eh» que es, precisamente, el código de operación de «LD A, (IX + d)». El valor de «d» se añade, en complemento a 2, a continuación del código de operación.

Todo esto se encuentra en cualquier manual, así como todos los códigos de las instrucciones que utilizan direccionamiento indexado. Pero lo que no dice ningún manual (al menos, ninguno que nosotros conozcamos), es que los registros «IX» e «IY» pueden ser utilizados como cualquier otro registro del microprocesador. No solamente como punteros, sino para realizar operaciones dentro de ellos mismos. Por ejem-

plo, hemos visto que la instrucción «INC (HL)», cuyo código de operación es «34h» se transforma en «INC (IX + d)» con código de operación «DDh 34h» o en «INC (IY + d)» con código «FDh 34h». Pero lo que no hemos visto, y veremos ahora, es que la instrucción «INC HL» cuyo código de operación es «23h» se transforma en «INC IX» si le ponemos delante «DDh» quedando, como código de operación, «DDh 23h». Esto es válido para todas las instrucciones que utilicen el registro «HL» y es una consecuencia del sistema de decodificación que utiliza, internamente, el Z-80. Cada vez que se recibe un «DDh» o un «FDh», se sabe que el siguiente código de operación que se reciba no irá referido al registro «HL», sino al «IX» o al «IY».

No queda ahí la cosa. Los registros «IX» e «IY», que son registros dobles (de 16 bits), pueden «par-

tirise» y convertirse en registros sencillos (de 8 bits). Para evitar confusiones, llamaremos a los registros de la siguiente forma:

«IX» = Mitad alta de «IX»  
«IX» = Mitad baja de «IX»  
«IY» = Mitad alta de «IY»  
«IY» = Mitad baja de «IY»

La letra que esté en mayúscula indicará a cuál de las dos mitades nos estamos refiriendo.

Cualquier instrucción que actúe sobre el registro «H», podrá actuar sobre la mitad alta de los registros «IX» e «IY» si le anteponeamos «DDh» o «FDh». De la misma forma, cualquier instrucción que actúe sobre el registro «L», lo hará sobre la mitad baja del «IX» o «IY» si le anteponeamos «DDh» o «FDh». Por ejemplo: la instrucción «LD A,L» cuyo código de operación es «7Dh» se transforma en «LD A,IX» con código «DDh 7Dh».

Existen un total de 126 instrucciones que actúan sobre «H», sobre «L» o sobre «HL», por lo que podemos obtener 252 instrucciones extra que actúen sobre «IX», sobre «IY» o sobre cualquiera de sus mitades.

Estas instrucciones no serán reconocidas por ningún ensamblador, por lo que la forma de teclearlas es utilizar la correspondiente instrucción referida a «HL», pero anteponiéndole un «DEFB #DD» o un «DEFB #FD». Por ejemplo: si queremos teclear en nuestro ensamblador la instrucción «LD A,IX» podremos hacerlo de la siguiente forma:

DEFB #DD  
LD A, L

Aunque recomendamos añadir un comentario para hacer más legible el código fuente. Algunos desensambladores (por ejemplo el MONS-3) de-

sensamblan como «NOP» aquellas instrucciones que no identifican, pero señalándolo con un asterisco a la derecha del código objeto. Por ejemplo, si un MONS-3 se encuentra con la instrucción que acabamos de introducir, la desensamblaría de la siguiente forma:

DD\* NOP  
7D LD A, L

Tenga esto en cuenta cada vez que desensamble un programa comercial, ya que últimamente, aparecen este tipo de instrucciones con bastante frecuencia.

Con esto, damos por terminado el CURSO DE CODIGO MAQUINA, no sin antes recordar al lector que puede remitir cualquier duda que le surja, a la sección CONSULTORIO de MICROHOBBY Semanal, donde tendremos sumo gusto en resolverla... ¡Si sabemos!



# INTRODUCCION AL CODIGO MAQUINA

Prácticamente cualquier usuario de Spectrum ha tenido alguna vez contacto con el código máquina. En este lenguaje están escritos los mejores programas comerciales, y algunas veces lo hemos utilizado en las páginas de nuestra revista.

El código máquina no es sólo otro lenguaje más de programación; se trata de hablarle al ordenador directamente en el lenguaje que él entiende. De esta forma, no estamos sujetos a las restricciones del Basic, que son muchas, y tenemos un dominio completo sobre nuestra máquina.

Normalmente, nadie programa directamente en código máquina, este lenguaje está compuesto únicamente por sucesiones de números y sobre todo, para qué sirve. A lo largo de este curso, vamos a dar respuesta a estas y otras preguntas.

En los cuatro primeros capítulos del curso, se estudian algunas nociones previas que serán necesarias en los capítulos posteriores, por lo que no es recomendable pasar a estudiar un capítulo sin haber comprendido totalmente el anterior.

Es muy probable que el

se suelen escribir los programas en Assembler y después, traducirlos a código máquina. Esta última tarea, se conoce por el nombre de "ensamblado", y habitualmente, se realiza con la ayuda de un programa llamado "Ensamblador".

En los cuatro primeros capítulos del curso, se estudian algunas nociones previas que serán necesarias en los capítulos posteriores, por lo que no es recomendable pasar a estudiar un capítulo sin haber comprendido totalmente el anterior.

Es muy probable que el



XOR	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	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	A	B	C	D	E	F
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	F	E	D	C	B	A	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	8	9
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Figura 6. Tabla hexadecimal de "EXOR".

AND	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	0	0	2	0	2	0	0	2	0	2	0	0	2	0	2	0
3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	0	0	0	4	4	0	0	4	0	0	4	0	0	4	0	0
5	0	1	0	1	4	5	0	1	0	1	4	5	0	1	0	1
6	0	0	2	2	4	6	0	2	4	6	0	2	4	6	0	2
7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	0	0	0	0	0	0	0	0	8	8	0	0	0	0	0	0
9	0	1	0	1	0	1	0	1	8	9	0	8	9	0	8	9
A	0	0	2	2	0	2	0	2	8	A	0	8	A	0	8	A
B	0	1	2	3	0	1	2	3	8	B	0	8	B	0	8	B
C	0	0	0	4	4	0	0	4	8	C	0	8	C	0	8	C
D	0	1	0	1	4	5	0	1	8	D	0	8	D	0	8	D
E	0	0	2	2	4	6	0	2	8	E	0	8	E	0	8	E
F	0	1	2	3	4	5	6	7	8	F	0	8	F	0	8	F

Fig. 5. Tabla hexadecimal de "AND".

una operación "EXOR" de un número consigo mismo, el resultado es siempre "cero"; lo cual es muy útil, ya que si se quiere cargar el número "cero" en el acumulador, se puede utilizar la instrucción "XOR A" ("EXOR" del acumulador consigo mismo) que ocupa la mitad de memoria y es el doble de rápida que "LD A,0".

acarreo sin que varíe el contenido del acumulador.

El empleo de operadores lógicos nos va a permitir movernos por tablas, calcular direcciones de memoria, poner "máscaras" a algunos bits, y un sinfín de utilidades que justifican la necesidad de adquirir el mayor dominio posible del álgebra de Boole.

## EJERCICIOS

- Realizar un "AND", un "OR" y un "EXOR" entre los siguientes pares de números:
- |                 |   |                 |
|-----------------|---|-----------------|
| 1 1 1 0 0 1 0 0 | Y | 0 1 1 0 1 0 0 1 |
| 0 1 0 1 0 0 0 0 | Y | 1 0 1 0 1 1 1 1 |
| 0 0 0 1 1 0 1 0 | Y | 1 0 1 1 0 1 0 0 |
| 2Ah             | Y | Bfh             |
| B1h             | Y | 58h             |
- Repetir el ejercicio anterior, utilizando los complementarios de los números propuestos.



gran volumen de lectores nos haga imposible mantener una correspondencia personalizada, pero aún así, nos agradecería que quienes sigan el curso nos escriban contándonos sus progresos o las dificultades que encuentran. Estas cartas nos permitirán ir adaptando las explicaciones a un nivel que satisfaga a todos y permita que nadie se quede "descolgado".

En este mismo capítulo incluímos dos rutinas de utilidad: una que permite hacer "Scroll" lateral de la pantalla, a derecha e izquierda y pixel a pixel.

Las dos rutinas se han ensablado una a continuación de la otra y son reubicables, es decir, se pueden colocar en cualquier parte de la memoria. Nosotros las hemos ensablado a partir de la dirección 55000, pero quien disponga sólo de 16 K, puede colocarlas en otra dirección, haciendo unas pequeñas modificaciones en el programa cargador, que explicaremos un poco más adelante.

En la FIGURA 1, reproducimos fotográficamente el listado en Assembler de las dos rutinas. No se preocupe el lector si le suena a "chino", un listado en Assembler no es más difícil de entender que uno en Basic, cuando se conoce. Al final del curso, más de uno será capaz de mejorarlo.

El PROGRAMA 1 sirve, lógicamente, para cargar estas rutinas en memoria sin necesidad de Ensamblador. De esta forma, no es necesario saber código máquina para usarlas. Una vez estén en memoria, basta teclear:

RANDOMIZE USR 55000

## II CODIGO MAQUINA

\*HISOFT GEN3M ASSEMBLER\*  
ZX SPECTRUM

Copyright HISOFT 1983  
CURSO C/M MICROHOBBY

Pass 1 errors: 00

```

10 *C-
20 *D+
30 ORG 55000
40 ;
50 ; SCROLL_IZQUIERDA
60 ;
70 LD HL, 22527
80 LD C, 192
90 LD BUC_2, B, 32
100 AND A
110 BUC_1 RL (HL)
120 DEC HL
130 DJNZ BUC_1
140 JR NC, NOCA_1
150 LD (VAR), HL
160 LD IX, (VAR)
170 SET O, (IX+32)
180 NOCA_1 DEC C
190 JR NZ, BUC_2
200 RET
210 ;
220 ; SCROLL_DERECHA
230 ;
240 LD HL, 16384
250 LD C, 192
260 BUC_4 LD B, 32
270 AND A
280 BUC_3 RR (HL)
290 INC HL
300 DJNZ BUC_3
310 JR NC, NOCA_2
320 LD (VAR), HL
330 LD IX, (VAR)
340 SET 7, (IX-32)
350 NOCA_2 DEC C
360 JR NZ, BUC_4
370 RET
380 VAR EQU 23728

```

Pass 2 errors: 00

Table used: 97 from 160

Fig. 1. Listado Assembler de las rutinas de "Scroll" lateral.

### OPERADOR AND

En cierto sentido, se puede considerar que este operador es el opuesto del anterior. Equivale al producto, en cuanto a las propiedades algebraicas que satisface.

Cuando operamos dos elementos de nuestro conjunto ("unos" o "ceros") obtenemos un "1" solamente si ambos elementos son "1"; y un "0" en cualquier otro caso. La Tabla de verdad del operador AND es la siguiente:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Vamos a ver que ocurre si, con los números del ejemplo anterior, aplicamos la operación AND:

01000110 (46h)

AND

11100011 (E3h)

=

01000110 (42h)

Esta vez, hemos puesto un "1" sólo en los lugares donde ambos números tenían un "1", y hemos puesto "0" en todos los demás lugares. La tabla del operador "AND" en Hexadecimal, está representada en la FIGURA 5.

### OPERADOR EXOR

No se trata propiamente, de una operación del álgebra de Boole, pero es necesario describirlo dado el gran uso que se hace de él cuando se programa en Assembler. El operador

NOT	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Fig. 3. Tabla hexadecimal de "NOT".

OR	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	A	B	C	D	E	F
1	1	1	3	3	5	5	7	7	9	9	B	B	D	D	F	F
2	2	3	2	3	6	8	6	7	A	B	A	B	E	F	E	F
3	3	3	3	3	7	7	7	7	B	B	B	B	F	F	F	F
4	4	5	6	7	4	5	6	7	C	D	E	F	C	D	E	F
5	5	5	8	7	5	5	7	7	D	D	F	F	D	D	F	F
6	6	7	6	7	6	7	6	7	E	F	E	F	E	F	E	F
7	7	7	7	7	7	7	7	7	F	F	F	F	F	F	F	F
8	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	F
9	9	9	B	B	D	D	F	F	9	9	B	B	D	D	F	F
A	A	B	A	B	E	F	E	F	A	B	A	B	E	F	E	F
B	B	B	B	B	F	F	F	F	B	B	B	B	F	F	F	F
C	C	D	E	F	C	D	E	F	C	D	E	F	C	D	E	F
D	D	D	F	D	D	F	D	D	D	F	D	D	D	F	D	D
E	E	F	E	F	E	F	E	F	E	F	E	F	E	F	E	F
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Fig. 4. Tabla hexadecimal de "OR".

rador "EXOR" es, en cierta forma, una mezcla de los operadores "AND" y "OR".

Cuando operamos dos elementos de nuestro conjunto mediante este operador, obtenemos un "1", sólo si uno de los dos elementos es "1"; y obtenemos un "0" tanto si ambos son "ceros", como si ambos son "unos". La Tabla de verdad del operador "EXOR" es la siguiente:

0 EXOR 0 = 0
0 EXOR 1 = 1
1 EXOR 0 = 1
1 EXOR 1 = 0

Hemos puesto un "0" en los lugares donde ambos números eran iguales (dos "unos" o dos "ceros"), y un "1" donde eran distintos ("cero" y "uno" o "uno" y "cero").

La tabla del operador "EXOR" en hexadecimal, está representada en la FIGURA 6. Como curiosidad importante, cabe señalar que si se realiza

## CODIGO MAQUINA VII







quina". Pero también podemos decirle: "pulsas la tecla 5, luego la tecla «por» y luego la tecla 7"; esto sería "Assembler".

Vamos a "programar" en "Assembler" a nuestro amigo, para que nos calcule el cuadrado de 5 por 7 y no escriba el resultado en un recuadro de la hoja de papel al que denominamos "archivo de representación visual". La calculadora puede ser la representada en la FIGURA 2.

El programa podría quedar más o menos así:

```

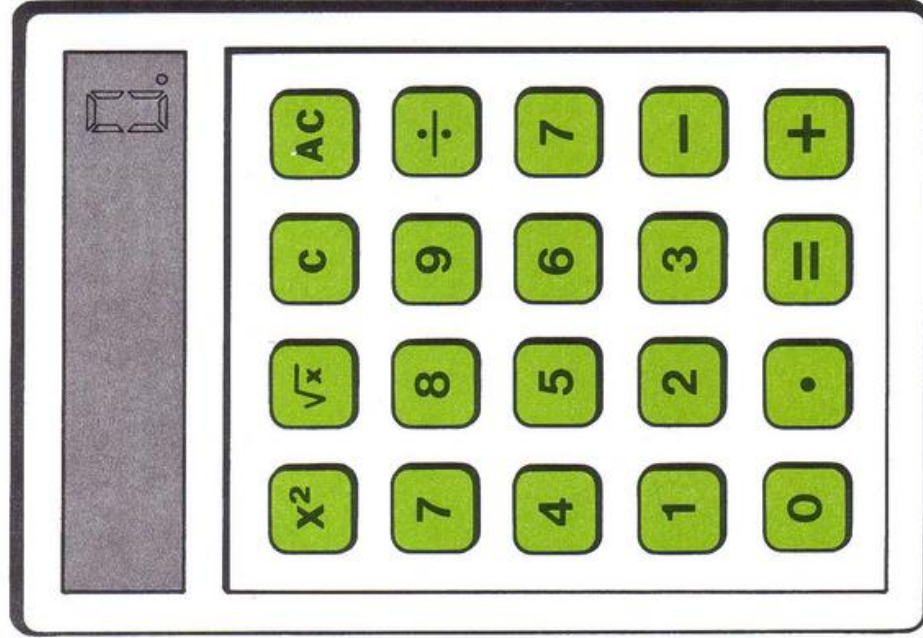
10 Pulsa "AC"
20 Pulsa "5"
30 Pulsa "POR"
40 Pulsa "7"
50 Pulsa "="
60 Pulsa "CUADRADO"
70 Escribe Resultado
80 Fin

```

Este programa se lo anotamos en el papel, y le damos la orden de que lo ejecute. Al final, él nos escribe el resultado en el papel.

Podemos sacar aún más partido a nuestro ejemplo. Supongamos que nuestro amigo supiera manejar perfectamente la calculadora, en ese caso, nos bastaría con decirle: "Calcula el cuadrado de 5 por 7 y anota el resultado". En este caso, estaríamos usando un "lenguaje de alto nivel". El Basic es un lenguaje de alto nivel, y lo podemos usar gracias a que nuestro ordenador tiene un "intérprete", lo que hace que "sepa" manejar perfectamente el microprocesador.

Vamos a estudiar detenidamente el proceso. Primero pulsamos "AC", con lo que se borran los anteriores contenidos de la calculadora. A



La calculadora de nuestro amigo el microprocesador.

continuación, pulsamos la tecla "5", con lo que aparece en pantalla el número cinco. La pantalla de la calculadora es un registro, y lo que hemos hecho ha sido cargar este registro con el número cinco. Luego definimos la operación a realizar, y cargamos el registro con el segundo operando (siete). Al pulsar "=" se realiza la operación y el resultado aparece de nuevo en ese registro. Finalmente, al pulsar "CUADRADO", elevamos al cuadrado el contenido del registro, y el resultado nos vuelve

ve a aparecer en el mismo. La pantalla de una calculadora va acumulando los resultados de todas las operaciones que vamos realizando, por eso, podemos llamarla "registro ACUMULADOR". Todos los microprocesadores tienen un registro acumulador, pero a diferencia de las calculadoras, tienen también otros registros que nos pueden servir para diversos fines. A pesar de tener muchos registros, no son suficientes para almacenar el enorme volumen de datos que maneja

dremos utilizar números en base 16 (hexadecimales).

Es muy importante adquirir cierta soltura en el manejo de la numeración hexadecimal, por ello, hemos dedicado un capítulo entero a este tema. Con bastante frecuencia, tendremos que convertir números decimales a hexadecimales o viceversa. Para esto se pueden usar los métodos descritos en el citado capítulo, pero resulta bastante tedioso, así que hemos desarrollado un programa que hace ese trabajo por nosotros. Este programa se encuentra en la página 11 del curso (MICROHOBBY número 43).

Existen también, calculadoras de bolsillo capaces de operar en estas bases, y representan una gran ayuda a la hora de programar en código máquina. También hemos dedicado un capítulo a describir el microprocesador Z-80 con el mayor detalle posible, ya que su conocimiento es imprescindible para programarlo. Sería algo así como el "manual" de la calculadora. Finalmente, y antes de empezar a estudiar las instrucciones, hemos dedicado un capítulo a describir la forma en la que se debe elaborar un programa, independientemente del lenguaje utilizado.

## Algebra de Boole

Prácticamente todos los instrumentos matemáticos que se utilizan en la programación de un pequeño ordenador como el Spectrum, forman parte del bagaje cultural de cualquier persona medianamente formada. Excepto, quizá, el álgebra de Boole.

Tal vez por ser de aparición relativamente reciente, tal vez por su escasa utilidad práctica en la realidad habitual, el caso es que el álgebra de Boole no ha sido incluida en el programa de estudios hasta fecha reciente; y lo ha sido dentro de la asignatura de "matemáticas comunes" del C.O.U. desgraciadamente, una de las "Marias".

Programando en Basic, hemos hecho uso de algunos conceptos provenientes del álgebra de Boole; cuando utilizábamos en las sentencias IF... THEN, los operadores OR, AND y NOT para expresar conjunciones o disyunciones lógicas. Al programar en Assembler o código máquina, haremos un uso mucho más profundo y preciso de estos operadores, así como del operador EXOR que no se utiliza en Basic.

Es tan frecuente (y útil) utilizar operadores lógicos en Assembler, como lo puede ser utilizar la suma y la resta en una calculadora de bolsillo. Por ello, es imprescindible tener un cierto conocimiento del álgebra de Boole; que por otro lado, es sumamente sencilla de aprender.

Dado que éste no pretende ser un manual de matemática moderna, no entraremos a definir formalmente lo que constituye un álgebra de Boole. Bastenos saber que un álgebra de Boole se puede construir allá donde tengamos un conjunto de elementos que puedan tomar dos valores (en nuestro caso, "0" y "1") y definamos una relación de equivalencia ("ser igual a") y dos operaciones internas al conjunto, que cumplan una serie de propiedades, similares a las que cumplen la suma