GRAN BIBLIOTECA

AMSTRAD

PRINCIPIOS EN CÓDIGO MÁQUINA

ADIÓS A LOS TABÚES

EDICIONES INGELEK

GRAN BIBLIOTECA AMSTRAD

10

PRINCIPIOS EN CÓDIGO MÁQUINA

Director editor:
Antonio M.ª Ferrer Abelló

Director de producción:
Vicente Robles

Dírector de la obra:
Fernando López Martínez

Redactor técnico:
Carlos de la Ossa Villacañas

Colaboradores:
Data-3 Informática, S. A.
Pilar Manzanera Amaro
Amelia Polo García

Diseño:
Bravo/Lofish

Maquetación:
Carlos González Amezúa

Dibujos:
José Ochoa

Fotografía
Grupo Gálata

ISBN del tomo: 84-7708-047-X
ISBN de la obra: 84-7708-004-6
Fotocomposición: Andueza, S. A.
Imprime: Héroes, S. A.
Depósito Legal: M-3625-1987
Precio en Canarias, Ceuta y Melilla: 435 ptas.
Marzo 1987

PRINCIPIOS EN CÓDIGO MÁQUINA

INTRODUCCIÓN

Los próximos capítulos, es decir, todos los que componen el décimo tomo de la presente obra de la GRAN BIBLIOTECA AMSTRAD, estarán íntegramente dedicados al código máquina o lenguaje máquina del microprocesador Z80.

Por tratarse de un lenguaje completamente diferente al BASIC, al menos en el formato de sus instrucciones, potencia, posibilidades y técnicas de construcción de programas, en principio no es necesario conocer a fondo este último, pues parten de bases distintas, aunque ambos persiguen un objetivo común: informar claramente al Amstrad sobre cuál es el trabajo que debe ejecutar.

El lenguaje máquina es lo más próximo a la forma de hablar de nuestro micro. Por ello su conocimiento y comprensión nos descubrirá sus auténticas posibilidades y la manera de funcionar de nuestro pequeño «cerebro» electrónico.

No desesperemos si el aluvión de nuevos conceptos desborda nuestras previsiones. En poco tiempo y con alguna práctica seremos

capaces de manejarlos a nuestro antojo. Todos los programas y ejemplos estarán en forma clara y debidamente documentados para que podamos detectar cualquier posible equivocación de carga.

No olvidemos que los errores de programación en C/M no pueden estropear nuestro ordenador. Todo lo más, el «desastre» provocará la pérdida del control sobre nuestro micro y el problema quedará solucionado desconectando la alimentación. Por ello, antes de ejecutar un programa conviene grabarlo como seguro del tiempo empleado introduciéndolo a través del teclado.

Una última advertencia. Para la correcta interpretación de las tablas de comandos presentados a lo largo de las páginas de este libro, debemos recurrir al último cuadro del apéndice.

LOS PRIMEROS PASOS

omo ocurre en el caso de los programas escritos en BASIC, los de C/M deben estar almacenados en la memoria del ordenador antes de ser ejecutados. No olvidemos que la ROM, a fin de cuentas, es también una memoria, aunque no podamos alterarla, y por tanto, tendremos la posibilidad, si lo creemos necesario, de leer los 32 Kbytes en C/M (en el caso de los CPC) almacenados allí y utilizarlos en nuestros programas.

Un programa en C/M es una secuencia de instrucciones encargadas de procesar ciertos datos, las cuales permiten realizar al ordenador una serie de operaciones que completen una determinada tarea, pero con una diferencia sustancial respecto a otros lenguajes de alto nivel, como el BASIC: son directamente interpretadas por el ordenador sin necesidad de emplear un traductor intermedio. De ahí, su extremada velocidad de proceso y a la par la complejidad de su construcción.

De lo anterior deducimos que en la memoria de nuestro Amstrad (RAM o ROM) podemos encontrar tanto instrucciones como datos,

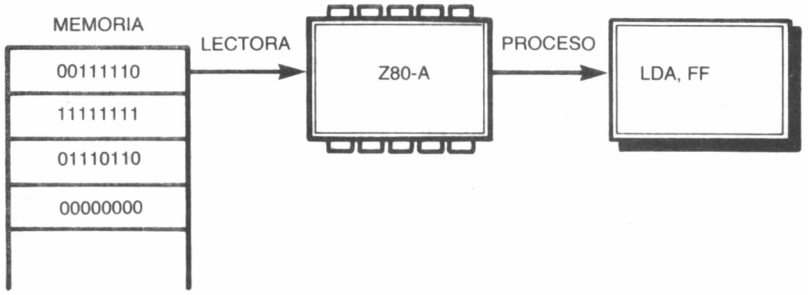
pero almacenados no de cualquier manera, sino en forma de dígitos binarios o bits (0 y 1), correspondientes, como sabemos, a dos niveles eléctricos diferentes, agrupados de ocho en ocho en unidades mayores, denominadas bytes.

Pero ¿cómo se ejecuta un programa? Pues bien, el Z80-A, microprocesador del Amstrad, lee de la memoria una instrucción, analiza el byte o los bytes que la componen y en función de su contenido fielmente realiza la operación indicada, y así sucesivamente hasta completar su trabajo.

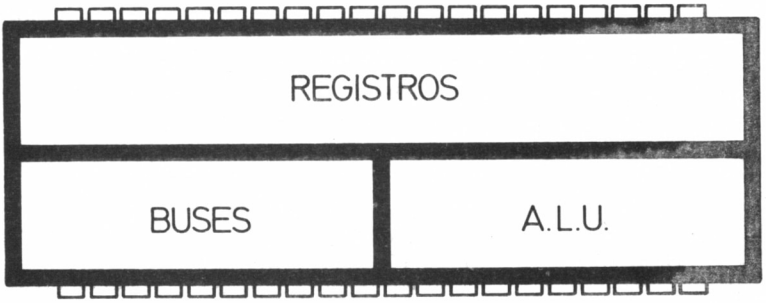
Como vemos, el verdadero «cerebro» en todas las tareas que efectúa nuestro ordenador es su microprocesador, físicamente constituido por miles de pequeños componentes electrónicos encapsulados en una única pastilla o chip.

El cómo funcionan y su interconexión eléctrica con el resto del sistema es algo que, por el momento, para el estudio del C/M no nos interesa. Sin embargo, sí es importante conocer cuáles son los componentes lógicos alojados dentro de este chip, pues son la razón fundamental que justifica el gran número de micros que lo incorporan.

Antes de seguir, una aclaración: el sufijo A en Z80-A indica que se trata de una versión mejorada del antiguo modelo, la cual es capaz de funcionar mucho más rápidamente que su predecesor, aunque los juegos de instrucciones que son capaces de manipular son totalmente equivalentes. Por ello, siempre que nos refiramos a él en adelante lo haremos bajo la denominación Z80.



10.1.1. El Z-80 A lee de la memoria una instrucción y ejecuta la operación indicada por ésta.



10.1.2. Registros del Z-80.

LA CPU

En la unidad central de proceso de nuestro Amstrad podemos encontrar los siguientes tres bloques lógicos principales:

— Los buses.
— La unidad aritmética y lógica (ALU).
— Los registros.

Como ocurre en el resto del ordenador, la información que circula a través de los diferentes componentes del interior del Z80 lo hace a través de los buses o conjunto de conductores eléctricos, uno por bit, habilitados a tal efecto.

Son tres los presentes: el de datos de 8 bits, el de control de 13 bits para sincronizar la CPU con los dispositivos exteriores a ella y el de direcciones de 16 bits, por lo cual el Z80 es capaz de direccionar $2 \uparrow 16 = 65536$ (64 Kbytes) posiciones diferentes de memoria de un byte cada una.

La ALU *(Arithmetic Logic Unit)* es la calculadora interna de la CPU. Las operaciones que puede ejecutar son las más básicas: sumas, restas, comparaciones, incrementar o decrementar una cantidad en uno, pero no multiplica ni divide directamente.

LOS REGISTROS

Un registro es un circuito del interior del Z80 utilizado como almacenamiento intermedio de información. Los registros disponibles en

este microprocesador podemos clasificarlos en los siguientes grupos:

— Registros de uso general.
— Registros de uso específico.
— Registros de control.

Los registros de uso general son utilizados a voluntad por el usuario e incluyen un acumulador A y los registros B, C, D, E, H y L. Cada uno de ellos tiene una capacidad de 8 bits y pueden emplearse solos o por parejas, BC, DE y HL (constituyéndose en unidades de 16 bits).

También está presente un registro de indicadores F *(Flags),* con capacidad para 8 bits, siendo su contenido variable según el resultado de determinadas operaciones que más adelante discutiremos en detalle.

El significado de cada uno de los bits que lo componen está íntimamente relacionado con el funcionamiento interno del microprocesador y su conocimiento es determinante para la correcta programación. Hemos de señalar también que algunas instrucciones utilizan la pareja AF para efectuar su cometido.

Como comprobaremos en su momento, entre los registros de uso general el microprocesador utiliza algunos de forma preferente, tales como el acumulador (A) o la pareja HL, pues determinadas operaciones sólo pueden ser ejecutadas con su concurso.

El Z80 dispone, además, de otro grupo equivalente, denominado ARS *(Alternate Register Set,* juego de registros alternativo), formado por los registros A', F', B', C', D', E', H' y L', el cual puede intercambiarse con el principal en cualquier momento ejecutando las instrucciones adecuadas.

Los de uso específico son registros que cumplen misiones especiales durante la ejecución de un programa. El Z80 cuenta con cuatro de 16 bits y dos de 8 bits. Efectuemos un repaso sobre ellos y veamos cuál es su cometido:

— PC *(Program Counter,* contador de programa). En él queda anotada la dirección de memoria de la instrucción en curso durante la ejecución del programa. Su capacidad son 16 bits.

— IX e IY son dos registros con capacidad cada uno de 16 bits, denominados registros INDICE X e Y, respectivamente. Son manejados por el microprocesador durante las operaciones de direccionamiento indexado (ya lo discutiremos en su momento), siendo especialmente útiles para el acceso a tablas de datos y la transferencia de parámetros entre BASIC y código máquina.

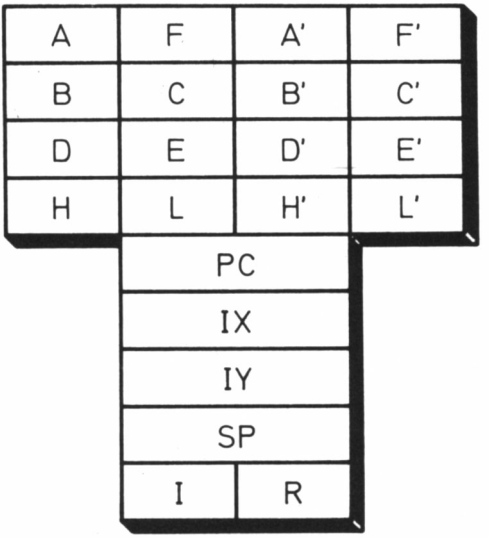
— SP *(Stack Pointer,* puntero de la pila) es un registro de 16 bits

cuya función consiste en indicar al microprocesador dónde se encuentra una zona especial de memoria RAM denominada STACK o pila, en la cual son almacenados determinados datos para su uso posterior.

— I *(Interrupt Page Address Register,* registro de dirección de página de interrupción), empleado cuando una solicitud exterior es efectuada al microprocesador por un periférico de alta prioridad, y al objeto de ser atendida ésta se rompe la secuencia normal del programa en ejecución, continuándose posteriormente el trabajo desde el punto en el cual se produjo la interrupción. Se trata de registro de 8 bits.

— R *(Memory Refresh register),* cuyo contenido es cíclicamente incrementado para que el Z80 dé la orden de regenerar las memorias dinámicas del sistema, evitando la pérdida de información que de otro modo se produciría en ellas.

Además de todos los anteriores, la CPU cuenta con otros registros no accesibles por el programador, como, por ejemplo, el IR *(Instruction Register,* registro de instrucciones), el cual contiene la instrucción que va a ejecutarse mientras es decodificada (interpretada) por el microprocesador, o registros de control para uso temporal durante las operaciones que se efectúen.



10.1.3. Z-80: bloques lógicos.

EJECUTANDO PROGRAMAS

Como hemos expuesto anteriormente, la información necesaria para que el programa en C/M pueda ejecutarse (datos + instrucciones) debe encontrarse previamente almacenada en la memoria del ordenador.

Algunas instrucciones ocupan un único byte, pero también las hay de dos, tres y cuatro bytes, siendo su contenido variable en función de la tarea que realizan. Entonces, ¿cómo reconoce el Z80 los diferentes tipos de información presentes en la memoria sin dar lugar a ambigüedades?

Es precisamente el orden en que el microprocesador encuentra la información el que determina el tipo de que se trata. Por esto los formatos de las instrucciones están sujetos a determinadas normas que los hacen inconfundibles para la CPU, y les prestaremos especial atención para ceñirnos a las «leyes» de programación del lenguaje máquina del Z80.

En las tablas de los apéndices encontraremos el juego completo de instrucciones del microprocesador Z80, sus códigos de operación y la representación simbólica (mnemónicos) equivalente.

INSTRUCCIONES Y OPERACIONES

l conjunto de instrucciones del Z80 es una ampliación del presente en el microprocesador 8080 de la firma INTEL. Por ello la mayoría de los programas escritos para este último pueden ser ejecutados en el desarrollado por los técnicos de ZILOG en el año 1976.

Es más, la compatibilidad podría hacerse efectiva a la inversa, siempre y cuando no incluyéramos en nuestros programas las nuevas instrucciones implementadas. No es éste, en principio, nuestro objetivo, pues son precisamente los nuevos grupos los que confieren una potencia fuera de lo común a este singular microprocesador de 8 bits.

En el capítulo anterior, una serie de nuevos conceptos fueron introducidos, y ya es hora de formalizar claramente su significado. Continuemos, por tanto, con algunas definiciones útiles.

A nadie se le oculta que cuando hablamos de programación cualquier instrucción está formada por un grupo de caracteres que definen una operación que el ordenador debe realizar. Un carácter es cualquier símbolo de los que nosotros consideramos elementales, como

una letra del alfabeto, un número del 0 al 9 o los especiales: dólar ($), asterisco (*), entre otros.

Una operación es la acción específica que el ordenador ejecuta cuando una instrucción se lo exige. Por tanto, cualquier instrucción en C/M estará compuesta, entre otras cosas que en seguida comentaremos, por uno o varos códigos, los cuales especifican al Z80 lo que debe hacer en todo momento y con toda precisión.

Los Formatos

Las tablas que acompañan al libro, presentes en los apéndices, contienen el conjunto completo de instrucciones que es posible implementar manejando el Z80, ordenadas según su función. Son 158 tipos diferentes (696 de distinto código de operación) y no debemos preocuparnos si encontramos algunos símbolos o datos que todavía no hayamos estudiado. Todo llegará en su momento.

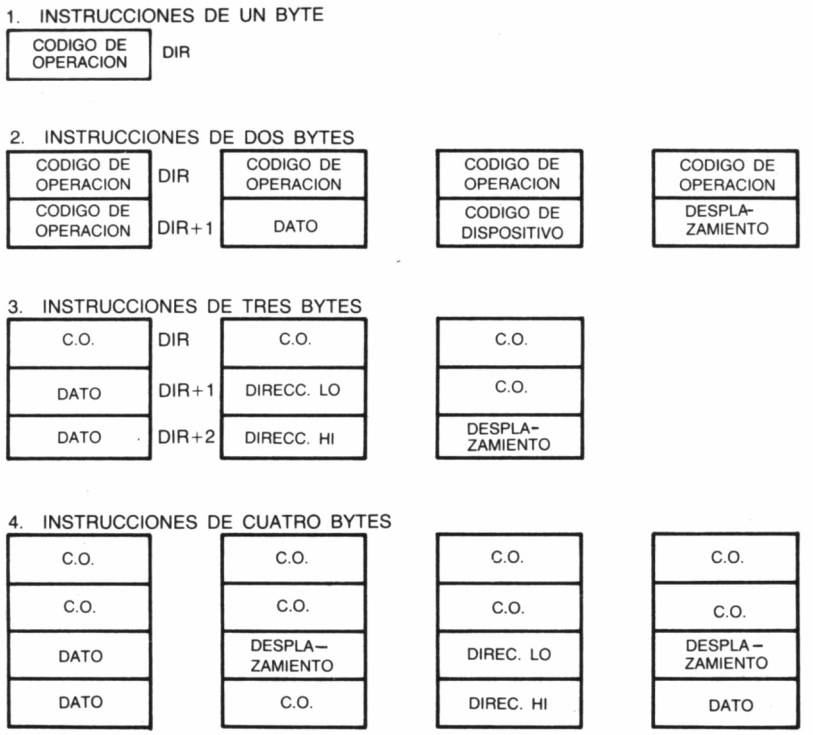
Por otra parte, en la figura se muestran los formatos de las instrucciones, atendiendo al número de octetos necesarios para su correcta construcción. Como observamos, pueden estar compuestas por uno, dos, tres o cuatro bytes, extremo que podemos confirmar ojeando las tablas principales. Pero ¿cómo es capaz de reconocer el microprocesador su longitud?

Si nos fijamos con detalle, el primer byte de todas las instrucciones es siempre un código de operación, sea cual sea el tamaño de ésta. El proceso seguido por la CPU de nuestro Amstrad es el siguiente:

— El registro PC le indica la dirección de memoria donde encontrar la siguiente instrucción. Allí lee un código de operación, y en función de éste reconoce el número de bytes de la instrucción, cantidad que es añadida al contador de programa PC. Finalmente, la instrucción es procesada y el Z80 busca un nuevo código de operación en la dirección señalada por el registro PC.

Algunas instrucciones en código máquina, las cuales provocan saltos o bifurcaciones en el proceso secuencial de un programa (algo parecido a lo que ocurre con los mandatos GOTO o GOSUB del BASIC), pueden modificar bruscamente el contenido del registro PC, pero siempre quedará almacenado en éste la dirección de memoria en donde la CPU debe buscar la siguiente instrucción a ejecutar.

Los octetos que siguen al primer código de operación, si los hubiere, pueden contener información de diversa índole, que a continuación detallamos:



10.2.1. Formato de las instrucciones del Z-80.

— Otro código de operación de 8 bits, el cual indica a la CPU el tipo de instrucción de que se trata (transferencia de datos, entrada/salida, lógica, manejo del STACK, etc.).

— Un dato de 8 ó 16 bits, codificado en binario, ASCII, binario codificado en decimal (BCD), etc.

— Un código de dispositivo de 8 bits. Teóricamente el Z80 es capaz de direccionar hasta $2 \uparrow 8 = 256$ periféricos, pero esta cuestión está íntimamente relacionada con el diseño de los circuitos exteriores al microprocesador que en su momento trazaron sus fabricantes, quedando habilitadas sólo algunas líneas, las cuales soportan, por ejemplo, el casete, la unidad de disco, el teclado, la impresora o las memorias ROM exteriores al sistema.

— Un byte de desplazamiento, es decir, un número en comple-

mento a dos con signo, empleado en las instrucciones que manejan direccionamiento indexado (pronto aprenderemos a calcular el complemento a dos de un número).

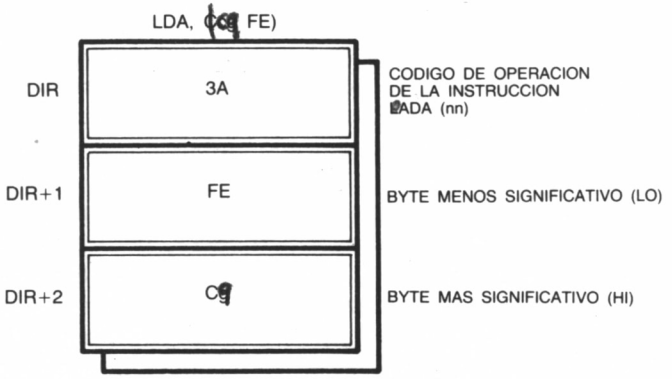
— Una dirección, determinada por dos bytes, donde el microprocesador siempre interpretará como byte de menor peso, LO (abreviatura de LOw, bajo), el presente en primer lugar, siendo el segundo el más significativo, HI (abreviatura de HIgh, alto).

Para todos ha de estar claro de ahora en adelante que el número en cuestión es interpretado por el Z80 de la siguiente manera:

VALOR TOTAL = VALOR DEL BYTE LO + 256 × VALOR DEL BYTE HI

El punto anterior, y no nos cansaremos de recalcarlo, es fundamental tenerlo presente, puesto que suele ser el causante de buena cantidad de errores de programación, dado que pasa normalmente desapercibido al intentar encontrar el fallo en una rutina, lo cual conduce inevitablemente hacia el dolor de cabeza y las horas de trabajo perdidas.

Por ejemplo, consideremos la instrucción LD A (C9FE), la cual «carga» el acumulador con el valor almacenado en la dirección de memoria C9FE Hex. En la figura se muestra el formato en que el ordenador debe encontrarla para su correcta interpretación. En el próximo capítulo encontraremos las indicaciones necesarias para obtener el máximo rendimiento de las tablas y realizaremos un recorrido a través de los diferentes grupos que las componen.



10.2.2. Cuando la CPU interpreta una dirección supone que ha encontrado los bytes que la definen en el orden.

ANALIZANDO LOS GRUPOS

uchos quizá nos estemos preguntando cuáles son las auténticas posibilidades del microprocesador de nuestro Amstrad. Somos bombardeados con una avalancha de nuevos conceptos y todavía no se ha justificado su utilidad.

Paciencia, es necesario sentar unas sólidas bases antes de proseguir nuestro estudio del C/M. Efectuemos un recorrido a través de los diferentes grupos de instrucciones para hacernos una idea bastante aproximada de las operaciones que puede ejecutar el Z80.

RECORRIENDO LAS TABLAS

Volvamos a las tablas de los apéndices. Allí encontraremos el juego de instrucciones completo del Z80 (casi 700 instrucciones diferentes), las cuales hemos subdividido en los siguientes bloques:

1) Carga de 8 bits: en este grupo están encuadradas todas las

instrucciones de intercambio de información entre registros individuales y entre registros y posiciones de memoria.

2) Carga de 16 bits: como podemos observar, muy pocas transferencias de información pueden efectuarse entre pares de registros. Sin embargo, a este grupo pertenecen las instrucciones PUSH y POP, fundamentales en el manejo del STACK o pila.

3) Aritmético de 8 bits: las operaciones de suma y resta entre cantidades de 8 bits son ejecutadas por las instrucciones de este grupo. Es posible también incrementar o decrementar en uno el contenido de un determinado registro o dirección de memoria.

4) Lógico: pertenecen a este bloque las instrucciones que hacen uso de los operadores AND, OR y XOR, discutidos en el primer volumen de esta misma colección dentro del capítulo dedicado al Algebra de Boole, así como las comparaciones lógicas.

5) Aritmético de 16 bits: se trata de un grupo análogo al anterior, pero efectuándose las operaciones entre pares de registros.

6) Aritmético y de control: comprende ciertas instrucciones de uso exclusivo del acumulador y el indicador de arrastre o acarreo, así como el manejo de las interrupciones.

7) Saltos y llamadas: permiten que el control del programa sea transferido a otra zona de éste, incondicionalmente o en determinados supuestos establecidos previamente, volviéndose al punto donde se efectuó la bifurcación, en el caso de que la instrucción fuera de llamada.

Hemos incluido en este grupo las instrucciones de RESTART (RST), llamadas incondicionales a determinadas direcciones de la ROM, que tan sólo ocupan un byte de memoria, a diferencia de las clásicas llamadas que necesitan tres octetos para ser procesadas.

8) Retornos: las instrucciones de retorno permiten que el programa continúe su ejecución a partir del punto donde fue efectuada la última llamada. De la misma manera que en el grupo anterior, el regreso puede producirse según se cumplan o no algunas condiciones especiales.

9) Intercambio: en este grupo encontramos las únicas instrucciones del microprocesador Z80 que permiten el manejo del grupo de registros alternativos (ARS).

10) Transferencia: facilitan el movimiento de bloques de datos desde unas posiciones de memoria a otras.

11) Búsqueda: permiten examinar un bloque de memoria con la intención de hallar un byte con el mismo contenido que el almacenado en el acumulador.

12) Tratamiento de bits: 240 nuevas instrucciones que permiten manipular individualmente cada uno de los bits almacenados en cualquier registro o posición de memoria.

13) Rotación y desplazamiento: al igual que las anteriores, posibilitan la ejecución de operaciones a nivel de bit dentro de cada octeto (byte).

14) Entrada/salida: estas instrucciones habilitan al microprocesador para la comunicación con los dispositivos exteriores a él, recogiendo o enviando información desde o hacia ellos.

Nuestra curiosidad debe estar ahora satisfecha. No dudaremos que con semejante volumen de instrucciones las posibilidades de programación que ofrece nuestro microprocesador son francamente fabulosas.

Pero no hemos hecho sino comenzar, y todavía hemos de seguir ampliando conocimientos antes de pasar al análisis de cada instrucción en particular. El camino puede resultar largo y complicado al principio, pero pronto estaremos en condiciones de obtener los primeros resultados.

MANEJANDO LAS TABLAS

En las tablas podemos encontrar de manera rápida y clara los códigos de operación (en hexadecimal) correspondientes a todas las instrucciones implementables en el microprocesador de nuestro Amstrad.

Quizá nos encontremos algo sorprendidos por la anterior afirmación («... de manera rápida y clara...») y no seamos capaces de ver en ellas otra cosa que un maremágnum de símbolos aparentemente sin sentido.

No es posible dar una norma general, la cual permita el acceso común a todos los grupos, pues, como descubriremos cuando sean analizados cada uno en particular, su utilidad se hará patente cuando estudiemos su forma de operar.

Por ejemplo, tomemos el grupo de carga de 8 bits. Entre paréntesis está señalado, que en lenguaje ensamblador, todas estas instrucciones comienzan por LD (abreviatura de *LoaD*, cargar).

Supongamos que buscamos la codificación de la instrucción LD B, A, la cual significa que introduzcamos en el registro B el contenido del registro A. En la parte superior de la tabla está señalada la palabra ORIGEN, y a la izquierda, DESTINO.

Por tanto, el destino es B y el remitente o fuente de la información es A, el acumulador. Entrando en la tabla por la fila señalada con B buscamos la intersección con la columna marcada con A y en la casilla correspondiente encontramos que el código de operación asociado al ensamblador LD B, A es 47h. Este es el calor que el ordenador deberá encontrar almacenado en su memoria para ejecutar la mencionada instrucción.

En otros grupos se define directamente el mnemónico de la instrucción, como en el caso de los saltos y llamadas. Supongamos que la instrucción a codificar es CALL 7FFFh, es decir, una llamada a la subrutina ubicada a partir de la dirección 7FFF hexadecimal o 32767 decimal.

En la columna de la izquierda están los mnemónicos correspondientes a estas instrucciones. En la sexta fila encontramos CALL. En la cabecera de la primera columna comprobamos que la llamada ha sido «incondicional» a la dirección de memoria especificada. Por tanto, siguiendo en la tabla, a continuación aparece el formato CALL nn, siendo nn los símbolos representativos de los dos bytes que definen dicha dirección.

Luego la codificación correcta será CD FF 7F. No debería sorprendernos el encontrar los dos bytes que determinan la dirección afectada con su posición intercambiada. Ya aclaramos en su momento que el microprocesador, cuando se trata de una dirección, espera encontrar en primer lugar siempre el byte de menor peso.

Otros símbolos utilizados son, por ejemplo, la «d», que representa el byte de desplazamiento en las instrucciones que manejan direccionamiento indexado, o la «e», la cual representa la extensión del salto en el direccionamiento relativo (los modos de direccionamiento, fundamentales para programar correctamente en código máquina, serán el tema principal de nuestro próximo capítulo).

CARGADOR DE C/M

Para terminar el presente capítulo hemos preparado un pequeño programa, el cual actuará de cargador hexadecimal para todas las rutinas en C/M, con las que pondremos en práctica los ejemplos necesarios para explicar los distintos grupos de instrucciones.

La elección del sistema hexadecimal nos ha parecido la más adecuada, dada la extensa bibliografía existente que lo utiliza, en especial la literatura dedicada a la descripción de las rutinas almacenadas en la ROM. Además, cuando nos enfrentamos con la ardua labor de te-

clear un programa en código máquina sin la ayuda de un ensamblador, este sistema ofrece la ventaja adicional de un ahorro considerable de pulsaciones frente al decimal.

Los códigos hexadecimales correspondientes a las instrucciones que configuren nuestros programas estarán incluidos en las sentencias DATA que situaremos al final del mismo, y efectuando RUN quedarán almacenados en la memoria de nuestro ordenador para su posterior ejecución.

En caso de cometer algún error se nos indicará la sentencia donde ocurrió para modificarla mediante un mensaje intermitente que aparecerá sobre la pantalla del monitor.

Al comienzo se nos preguntará sobre la posición de memoria donde deseamos introducir la rutina en C/M. En principio, toda la RAM podría utilizarse, pero para evitar posibles conflictos con las variables del sistema y otras zonas delicadas de éste elegiremos posiciones de memoria cercanas al RAMTOP (final del área de memoria destinada al BASIC), donde ejecutar nuestros programas, salvo que no se indique lo contrario.

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

A continuación encontraremos el listado del programa cargador. Tecleémoslo cùidadosamente y grabémoslo antes de ponerlo en funcionamiento. Los poseedores de un ensamblador encontrarán más cómoda su utilización, aunque recomendamos que también utilicen las tablas para practicar al máximo con la correcta decodificación de cada instrucción.

DIRECCIONANDO

on el presente capítulo completamos la exposición de los conceptos previos necesarios para la correcta comprensión del lenguaje máquina antes de adentrarnos en el estudio detallado de cada instrucción en particular.

Serán tratados temas fundamentales, como la representación de un número en complemento a dos, el cálculo de operaciones aritméticas utilizando este formato y los diferentes modos de direccionamiento implementados en el microprocesador Z80.

A partir de este momento ya es posible considerar que la introducción quedó atrás, y por tanto es esencial que prestemos una atención especial a todo lo que sigue, pues de su correcta comprensión depende en buen grado la operatividad de los programas que construyamos.

EL COMPLEMENTO A DOS

Hasta ahora sabemos que toda información almacenada en la memoria de nuestro ordenador, sea cual sea su naturaleza, se encuentra

en forma de dígitos binarios o bits. Como el Z80 es un microprocesador de 8 bits, en cada posición de memoria tendremos ocho de estos dígitos.

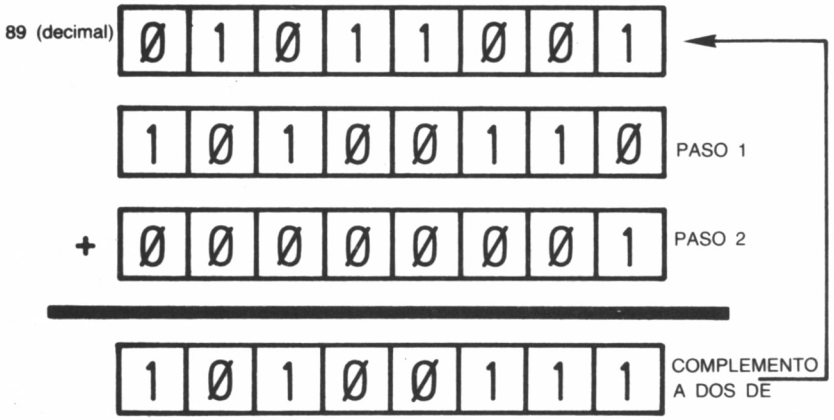
Los números siguen, por supuesto, el mismo formato, y sabemos que con un octeto es posible definir 2↑8=256 cantidades distintas comprendidas entre 0 y 255, es decir, entre 00000000 y 11111111 en binario.

Pero, obviamente, un ordenador es capaz de manejar números negativos (de no ser así, menudo desastre). Entonces, ¿cómo los reconoce? La respuesta la proporciona una técnica de codificación de números enteros (positivos y negativos) parecida a la binaria que todos conocemos, denominada *complemento a dos*.

Si un número está codificado según dicha técnica el microprocesador sólo tiene que «ojear» el bit 7 (el de mayor peso, o si lo preferimos, el primero comenzando por la izquierda) del byte correspondiente para identificar su signo. Entonces, si allí aparece un 0, el número será positivo, mientras que si encuentra un 1, lo considerará negativo.

Posiblemente estemos pensando que sólo nos quedan 7 bits con los que efectuar combinaciones y, por tanto, representar con ellos un número como el 255 puede resultar problemático.

Efectivamente, esto es así, y para no llevarnos a engaño vamos a definir los márgenes sobre los cuales es posible trabajar con 8 bits en



10.4.1. Cálculo del complemento a dos de un número.

DECIMAL	COMPLEMENTO A DOS
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
−1	1111
−2	1110
−3	1101
−4	1100
−5	1011
−6	1010
−7	1001
−8	1000

10.4.2. Representación en complemento a dos de los números con cuatro bits.

complemento a dos. Despreciado el de signo, podremos formar $2^7=128$ combinaciones diferentes, entre 0000000 y 1111111.

Es decir, si a los 7 bits anteriores les añadimos en primer lugar un 0 indicativo del signo más, los números positivos en este formato estarán comprendidos entre 0 y 127 expresados en decimal. ¿Y los negativos?

CALCULANDO EL COMPLEMENTO A DOS

Para hallar el complemento a dos de un número representado en binario seguiremos los siguientes pasos:

1) Cambiamos sus dígitos 1 a 0, y los que estén a 0 los situamos a 1.

2) Al número resultante le sumamos 1 expresado en binario, es decir, 00000001 (ver la tabla correspondiente a la suma de la aritmética binaria). Pongámoslo en práctica con algunos ejemplos. Calculemos el complemento a dos del número 89, 01011001 en binario. Al

efectuar el paso primero obtenemos 10100110. A continuación le sumamos 00000001:

```
  10100110
+ 00000001
  10100111 = -39 decimal
```

(observemos que el bit 7 es 1)

Del presente ejemplo y de los que muestran las figuras (ejercitémonos con ellos y comprobemos que llegamos a los mismos resultados) podemos extraer importantes consecuencias:

— Siempre debemos especificar el número de bits, cuando trabajamos en complemento a dos, con el que estamos operando.

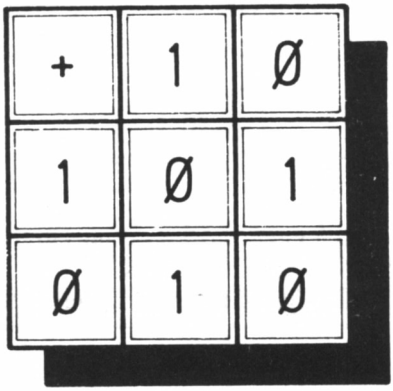
— El complemento a dos de un número positivo es siempre negativo, y viceversa.

Utilizando 8 bits es posible representar números enteros entre -128 y $+127$. En general, si operamos con n bits el complemento a dos permite cantidades entre $-(2\uparrow(n-1))$ y $+(2\uparrow(n-1))-1$.

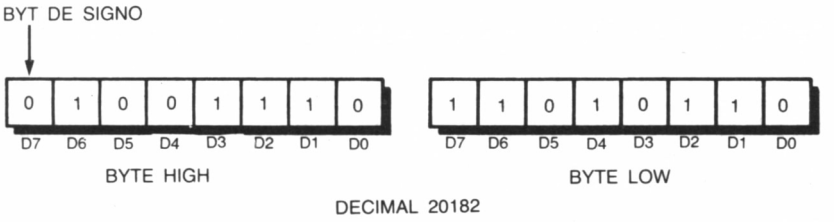
— El complemento a dos de un número decimal negativo es su código binario.

— Cuando utilizamos números de 16 bits en complemento a dos, el bit 7 del byte de mayor peso contiene el signo de éste, siguiendo el mismo criterio anterior.

Dos números en complemento a dos «se complementan», es decir,



10.4.3. La suma en la aritmética binaria.



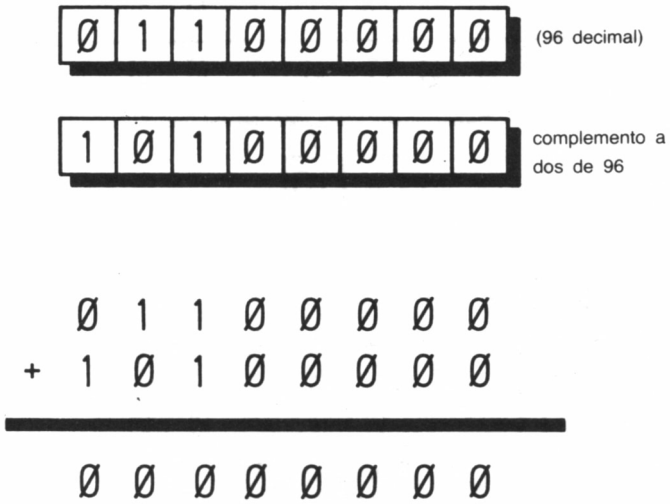
10.4.4. Signo en los números de 16 bits en complemento a dos.

si los sumamos obtenemos cero (recordemos que ha de estar claro el número de bits con el que estamos trabajando).

— Para sumar dos números en complemento a dos se siguen las mismas reglas que si estuvieran en su codificación binaria.

— La resta se efectúa complementando el sustraendo y sumándolo al minuendo.

— Cuando la suma de dos números positivos da uno negativo, o la de dos negativos es positiva, se ha producido el denominado sobrepasamiento o desbordamiento. En tal caso podremos tener constancia de dicha situación analizando el contenido del bit 2 del registro F (indicadores), el cual es situado a 1 cuando sucede tal eventualidad.



10.4.5. Dos números en complemento a dos «se complementan».

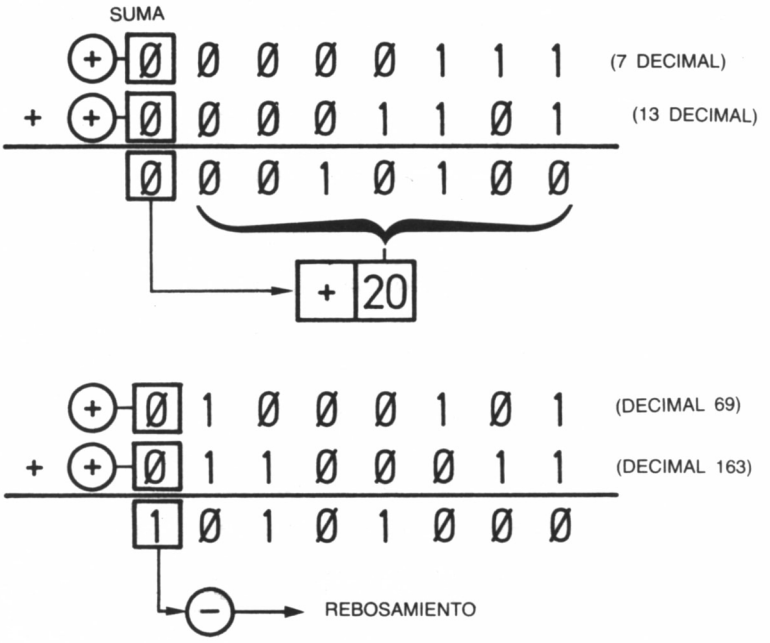
MODOS DE DIRECCIONAMIENTO

Bajo la denominación direccionamiento queremos hacer referencia a la forma en que el microprocesador Z80 accede a los datos almacenados en los registros o posiciones de memoria. Son posibles 10 modos diferentes, y aunque la mayoría de las instrucciones utilizan uno solo, algunas combinan varios de ellos para acceder a la información.

1) INMEDIATO. El byte que sigue al código de operación contiene al operando cuando utilizamos este modo. Por ejemplo, se emplea en la carga de registros con cantidades constantes:

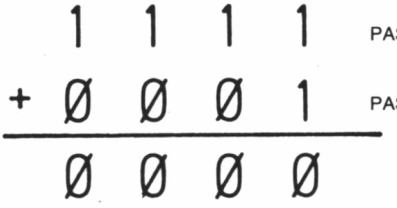
```
                    LD A,FFh
Código de operación:  3E
Operando:             FF
```

2) INMEDIATO EXTENDIDO. Básicamente es idéntico al anterior, salvo que el operando ocupa dos bytes (de ahí la denominación

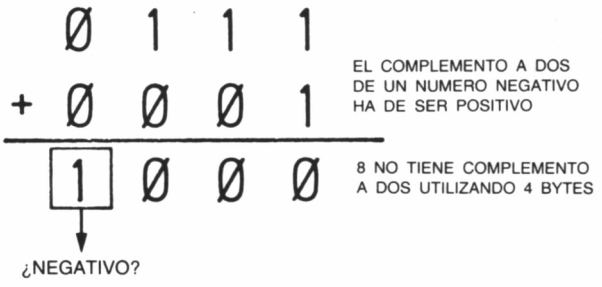


10.4.6. Suma en complemento a dos.

1) COMPLEMENTO A DOS 0000 (0 decimal)



2) COMPLEMENTO A DOS DE 1000 (−8 decimal)



10.4.7. Para poder calcular el complemento a dos de un número debemos especificar la cantidad de bits con la que estamos operando.

extendido). En la carga de registros dobles es empleado, por ejemplo, este modo:

```
                    LD HL,FFA9h
Código de operación:    21
Operandos:              A9 (byte LO) y FF (byte HI)
```

3) EXTENDIDO. Las instrucciones que lo manejan contienen siempre en sus dos últimos bytes una dirección de memoria. Esta puede ser el lugar donde recoger o almacenar el dato o donde ha de saltar el programa para seguir ejecutándose. Genéricamente podría representarse del siguiente modo:

```
LD rs (dirección)
LD (dirección), rs
JP dirección
```

En el primero, el byte contenido en «dirección» se transfiere al registro s, y el almacenado en «dirección»+1 se envía al r. En el se-

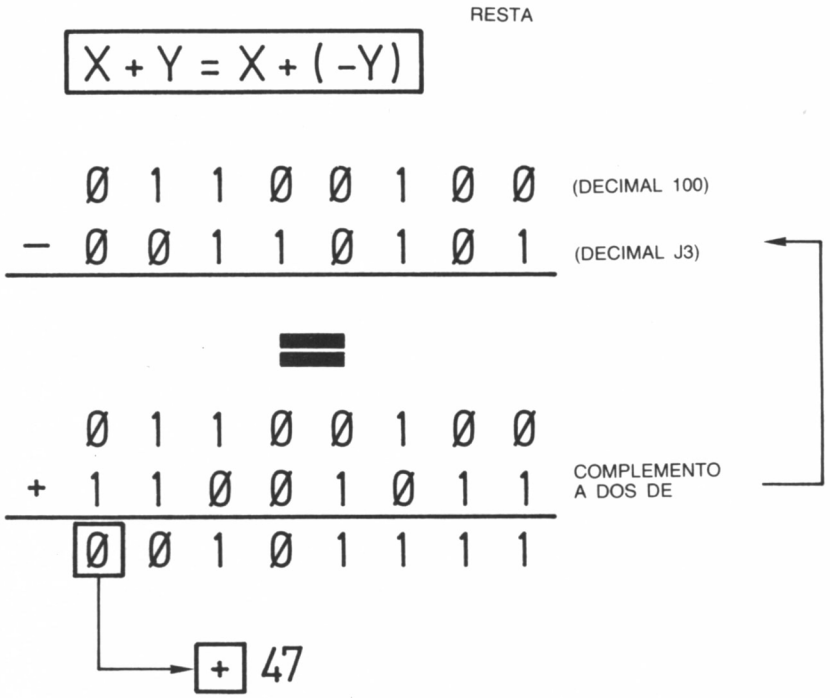
gundo, el proceso es inverso: se almacena en «dirección» el byte contenido en s y en «dirección»+1 el contenido en r.

En la tercera construcción se produce un salto a la posición de memoria especificada por «dirección», para continuar desde allí la ejecución del programa. Obsérvese que no hay transferencia de datos. Por ello, «dirección» no se expresa entre paréntesis.

4) POR REGISTRO. Dentro de los códigos de operación (expresados en binario) de muchas instrucciones algunos bits son variables en función del registro al que se refiere la instrucción, permaneciendo los otros inalterables. Tal es el caso de la transferencia de datos entre registros; por ejemplo:

LD A,B (cargar en el acumulador el byte contenido en el registro B)

5) INDIRECTO. Su forma de operar es similar al direccionamiento extendido, sólo que en este caso se emplea un par de registros



10.4.8. Resta en complemento a dos.

para señalar a la posición de memoria de donde recoger el dato. Por ejemplo:

LD A (HL)

Que significa: «cargar en A el byte contenido en la posición de memoria a la que apunta el registro doble HL».

6) IMPLICITO. Las instrucciones que lo manejan conocen a priori la posición de los datos implicados. Este es el caso de las operaciones aritméticas y lógicas de 8 bits, porque todas ellas realizan sus operaciones con el acumulador; por ejemplo:

ADD A,B

La cual suma los contenidos de los registros A y B, dejando el resultado en A.

7) INDEXADO. La mayoría de las instrucciones que manejan los registros índice IX e IY lo implementan. Todas ellas contienen tras el código de operación un byte de desplazamiento (d), que se suma al contenido actual del registro índice para señalar a una determinada posición de memoria.

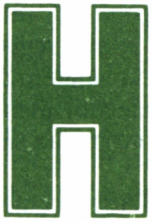
8) RELATIVO. Es el modo de direccionamiento utilizado en las instrucciones de salto relativo. Consideremos, por ejemplo, la instrucción JR 1Ah. Cuando el programa la encuentra efectúa un salto incondicional relativo a una posición de memoria situada 26 bytes más adelante de la que normalmente se habría ejecutado a continuación.

9) DIRECCIONAMIENTO DE BIT. Es el empleado en todas las instrucciones de manipulación de bits individuales dentro de un octeto determinado (BIT, SET, RESET).

10) PAGINA CERO O MODIFICADO. Son ocho las instrucciones que lo manejan, denominadas de RESTART. Algunas de estas instrucciones se utilizan para mejorar las posibilidades del Z80, mientras que otras se encuentran reservadas para su uso exclusivo por parte del sistema.

Las rutinas que habilitan, en el caso del sistema operativo de los Amstrad CPC, pertenecen a la ROM y son copiadas a la RAM durante el proceso de reinicialización, siendo el principio de importantes subrutinas de uso general.

LOS INDICADORES



asta el presente capítulo los nuevos conceptos se acumularon uno tras otro sin una explicación que clarificara su aparición. Como ya dijimos, no se trataba de un capricho: todo lo contrario.

Sin ellos será imposible obtener provecho de todo lo siguiente. Es más, tengamos en cuenta que el significado de muchas instrucciones está relacionado con varios de estos conceptos, y sería complicado, además de repetitivo, establecer sin su conocimiento previo un método que expusiera claramente qué es lo que hace determinada orden.

Por estas razones, nuestro esfuerzo inicial, como en adelante comprobaremos, quedará sobradamente recompensado y no nos cansaremos de recordar, aun a costa de resultar redundantes, la importancia fundamental de la lectura atenta y detallada de los capítulos anteriores antes de seguir adelante.

Como normal general, en lo sucesivo, además de discutir el objetivo de una determinada instrucción y su funcionamiento, encontraremos otros datos de interés, como son los referentes al tiempo de ope-

ración y las posibles inferencias de ésta con el registro de indicadores F.

Aunque algunos temas fueron tratados también anteriormente, no está de más profundizar sobre ellos con tal de familiarizarnos con su significado, pues su participación en cualquier rutina en C/M es de importancia capital.

CUESTIÓN DE TIEMPO

El factor «tiempo», cuando se trata de programas escritos en lenguaje máquina, es fundamental. Por dos razones: la primera justifica si nuestra inversión de trabajo al construir una rutina en concreto ha sido acertada, es decir, si ésta es lo suficientemente rápida como para emplearla en lugar de una idéntica codificada en BASIC.

Si el proceso seguido es correcto, la consideración anterior se cumplirá en la inmensa mayoría de los casos, pero a veces no siempre constituirá una ventaja, sino tal vez todo lo contrario.

Imaginemos un programa que necesite recibir del teclado determinada información antes de seguir adelante. Si no fijamos un retardo en la rutina para que esta operación pueda realizarse lo más seguro es que pase de largo ante nosotros a tal velocidad que ni siquiera apreciemos si fue ejecutada, tomando el ordenador indefectiblemente la no pulsación de tecla alguna como resultado, al no darnos ni tan siquiera unas décimas de segundo para reaccionar.

Recordemos que instrucciones tan sencillas en BASIC como INPUT o PRINT no existen en lenguaje máquina, y para simularlas debemos construir la rutina correspondiente o hacer uso de las implementadas en el FIRMWARE, considerando toda la gama de parámetros y posibles consecuencias que lleven asociadas.

Para que el microprocesador pueda ejecutar todas sus funciones necesita disponer de un patrón de tiempos, de tal forma que cada operación se ejecute sincronizadamente, en su momento preciso; ni antes ni después.

Es algo así como el director de una orquesta y bajo su batuta todos los componentes realizan la labor para la cual fueron construidos sin dar lugar a errores y, por supuesto, sin desafinar.

Este cronometraje lo efectúa un oscilador de cuarzo, el cual entrega, en el caso de los Amstrad CPC, cuatro millones de señales cada segundo (técnicamente se dice que está regulado a 4 MHz), denomi-

nándose cada una ciclo de reloj o estado, recibida en el Z80-A a través de su pin número 6 (CLOCK, reloj).

Esta frecuencia es la máxima teórica a la cual puede funcionar el Z80-A. Los diseñadores de Amstrad la eligieron en virtud de la calidad de los componentes que complementan el resto de la circuitería interna del equipo, sin que dicha circunstancia pudiera constituir una posible fuente de errores al forzar al máximo el rendimiento del microprocesador.

Una sencilla división nos muestra el tiempo empleado por un ciclo de reloj:

$$\frac{1}{4000000} = 0.00000025$$

Es decir, 250 nanosegundos (segundos divididos por 10↑9).

Se conoce como ciclo de instrucción al tiempo utilizado por el microprocesador para ejecutarla completamente. Dicha operación está compuesta por otras más elementales, denominándose al tiempo empleado en cada una de ellas ciclo de máquina.

Ha de quedar claro para todos que estos tiempos son diferentes según la instrucción particular o la operación elemental en cuestión. Así, por ejemplo, el ciclo de máquina durante el cual el microprocesador busca el código de operación de la instrucción toma 4 ciclos de reloj, mientras que las operaciones de lectura o escritura en la memoria (ciclos de memoria) ocupan 3 de reloj.

Así pues, para determinar el tiempo de ejecución de una rutina dada bastará con conocer el número de ciclos de reloj de cada instrucción y las veces que es ejecutada. Multiplicando la cantidad resultante por 250 obtendremos el resultado en nanosegundos (no nos preocupemos si no lo vemos muy claro. En seguida lo comprobaremos con un ejemplo).

El Registro F

Entre los registros de uso general disponibles en la CPU de nuestro Amstrad, el F (abreviatura de Flags, indicadores o banderas) tiene como misión revelarnos determinados sucesos que se producen durante la ejecución de algunas instrucciones, las cuales aparentemente no tienen ninguna relación con él.

Se trata de un registro de ocho bits y, como podemos comprobar

en la figura, cada uno de sus bits individuales tiene un significado especial. Analicémoslos con detenimiento:

— Indicador de arrastre o acarreo C (abreviatura del inglés *Carry):* su contenido varía en operaciones de suma y resta cuando se ha producido acarreo (sobrepasamiento o desbordamiento) en el bit más significativo del acumulador.

Por ejemplo, en la aritmética de 8 bits se comporta como un testigo colocado al lado de un cuentakilómetros que sólo puede marcar de 0 a 255. ¿Y si aumentamos la velocidad hasta 256 kilómetros por hora?

Este imaginario cuentakilómetros volvería a señalar 0, pero el indicador C estaría encendido señalando que le hemos dado la vuelta; es decir, que a la cantidad que ahora marca le debemos sumar 256.

— Indicador de cero Z (del inglés *Zero):* son muchas las instrucciones que afectan a su valor, por ejemplo, las que modifiquen el contenido del acumulador, las operaciones sobre bits individuales, las comparaciones, etc.

Su forma de trabajar resulta un tanto confusa si consideramos que cuando encontramos en su lugar correspondiente del registro F un 0 quiere decir que el resultado de la operación efectuada NO FUE CERO.

— Indicador de signo S *(Sign,* signo en inglés): el propósito de este bit es señalar si el resultado de una operación que involucre número en complemento a dos es positivo (bandera a 0) o negativo (bandera a 1).

Por tanto, no es otra cosa que una copia del bit más significativo de un número expresado en complemento a dos.

— Indicador de paridad/desbordamiento P/V *(Parity/Verflow):* a este bit le están encomendadas dos misiones diferentes. La primera se encarga de señalar si, por ejemplo, tras una instrucción de rotación, desplazamiento o lógica el número de bits que están en 1 lógico es par (indicador a 1) o impar (indicador a 0).

La segunda consiste en revelar si se ha producido desbordamiento en operaciones aritméticas con números expresados en complemento a dos. Tal es el caso que se presenta cuando la suma de dos números positivos es negativa o la de dos negativos resulta positiva.

En estas circunstancias el indicador V se coloca a 1, y en caso de no existir sobrepasamiento, el valor encontrado en la posición correspondiente del registro F será 0.

— Indicador de semiacarreo H *(Half carry):* se utiliza solamente en la aritmética codificada en decimal (BCD), la cual será comentada

en su momento. Su funcionamiento es similar al de arrastre (C), pero con la salvedad de que nuestro cuentakilómetros en este caso tendrá solamente 4 bits a diferencia con el anterior, el cual disponía de 8.

— Indicador de resta N: al igual que el anterior, solamente se emplea en la aritmética BCD, situándose a 1 cuando la última operación efectuada fue una resta.

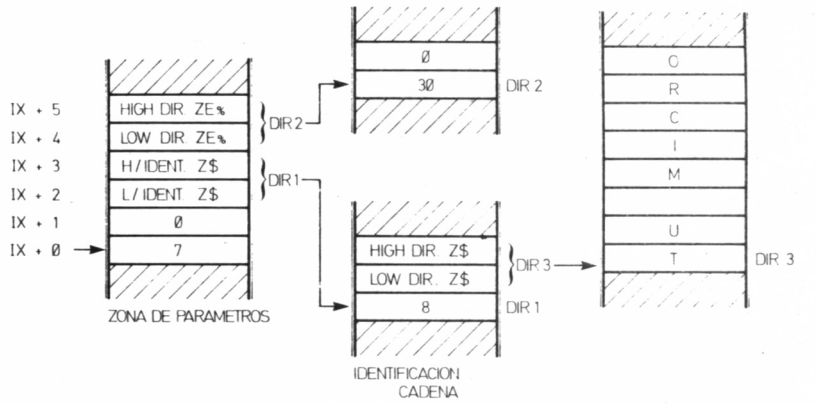
Como observaremos, sólo 6 de los 8 bits que componen el registro F tienen significado para nosotros. Los otros dos no tienen ningún interés para el programador.

No quiere ello decir que su valor no sea modificado durante la ejecución de una rutina determinada. Al contrario, la CPU los utiliza como apoyo para completar determinados procesos internos.

Antes de adentrarnos definitivamente en los diferentes grupos de instrucciones hemos de recalcar la importancia que como nexo de unión entre BASIC y C/M tienen los pares de registros índices IX e IY y la función CALL de BASIC dentro del sistema operativo de Amstrad.

CALL: Un Comando Sin Explotar

Si nos dejamos guiar por las sucintas explicaciones que del comando CALL se nos ofrecen en el manual de consulta de nuestro AMSTRAD («Permite invocar desde BASIC una rutina escrita en código máquina») pensaremos que sus posibilidades se reducen simplemente



10.5.1. Cronograma de los ciclos de máquinas típicos en el Z-80 A.

a indicar tras él una dirección de memoria, y el programa almacenado allí se ejecutará sin necesidad de nada más.

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

CALL dirección, param1, param2, param3...

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

CALL dir, valor o CALL dir, variable %

Donde valor, como hemos indicado, debe tratarse de un número entero.

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

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

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

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

Z$ = «TU MICRO»;: CALL dir, @Z$

Ahora el problema reside en recoger los parámetros transferidos

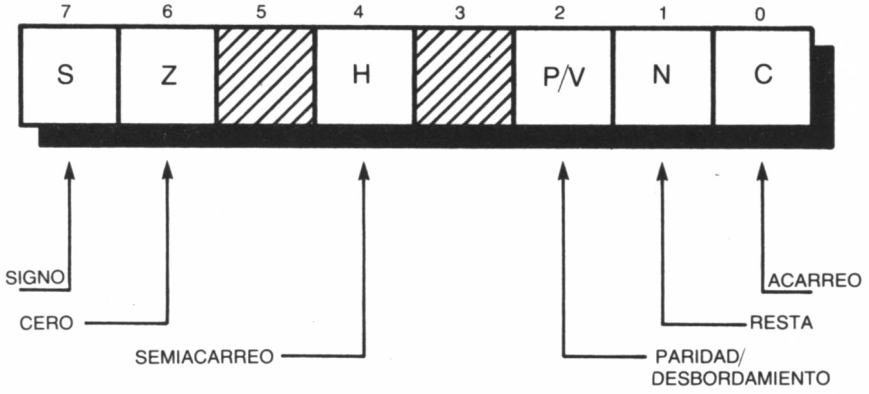
al programa en C/M. Nada más fácil. Cuando se invoca una subrutina mediante CALL el intérprete de BASIC se encarga de modificar el contenido de algunos registros de forma que sea fácil acceder a ellos.

De esta manera, el acumulador (A) contiene el número de parámetros, mientras que el registro índice IX apunta a un área denominada «zona de parámetros», la cual emplea dos bytes para cada uno de los que fueron enviados y cuyo significado depende del tipo de información manejada en la orden CALL.

La Zona De Parámetros

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

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



10.5.2. El registro F.

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

Por el momento no debemos preocuparnos si no comprendemos el significado de las instrucciones en ensamblador. Se trata simplemente de ilustrar, por ahora, la participación de los registros índice.

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

Vamos a desarrollar una pequeña rutina, la cual demuestre la forma de trabajar de la función CALL, a la vez de servirnos de muestra para ilustrar el cálculo del tiempo de ejecución de una rutina en concreto. Se trata, por ejemplo, de realizar en código máquina un programa similar a lo que haría la instrucción X%=45 de BASIC. Almacenaremos la rutina a partir de la posición 30.000 y la ejecutaremos mediante CALL 30000,45,@X%.

El lenguaje ensamblador, todas las instrucciones de carga, como podemos comprobar en la tabla correspondiente del apéndice, tienen código mnemónico LD. Sea, pues, nuestro programa, almacenado a partir de la dirección 30000, el siguiente:

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

¿Qué es lo que hace? Las dos primeras instrucciones almacenan en el par HL la dirección de memoria donde se encuentra el valor de la variable X%. La tercera instrucción recoge el número 45 en el acumulador y la cuarta se encarga de transferirlo a la posición en la que el sistema operativo espera encontrar el valor para X%. Final-

mente, mediante la instrucción RET se devuelve el control desde el C/M al BASIC.

Para almacenarla en la memoria del ordenador podemos utilizar el cargador de código máquina aparecido anteriormente en este volumen. Carguemos éste y añadamos la siguiente línea:

```
1000 DATA «DD6E00DD6601DD7E0277C900000000000000000000»,52C
```

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

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

Para terminar este capítulo calculemos el tiempo empleado en ejecutarse esta pequeña rutina. En las tres primeras instrucciones el microprocesador necesita para completar cada instrucción 19 ciclos de reloj, según se desprende de la tabla titulada «CARGA DE 8 BITS».

En la siguiente le bastan 7 ciclos para ejecutarla, 4 para encontrar el código de operación de la instrucción y tres más para escribir en una posición de memoria determinada. La instrucción RET, la cual será discutida más extensamente cuando tratemos las instrucciones de retorno, toma otros 10 ciclos en su ejecución.

En definitiva, $3\times19+7+10=74$ ciclos de reloj, o, lo que es lo mismo, 18500 nanosegundos. ¡Algo menos de 19 millonésimas de segundo en efectuar todo lo anterior!

DE 8 EN 8 BITS

entadas ya las bases previas necesarias para la comprensión de los distintos grupos de instrucciones comenzamos en este capítulo a realizar un análisis detallado de cada uno de ellos.

Aparte de la misión concreta de una determinada instrucción (discutiremos siempre en el texto su funcionamiento) encontraremos otros datos de interés para la programación en C/M en las tablas que presentamos como desglose de cada grupo particular, cuyo significado y utilidad discutíamos en el bloque anterior, tales como los ciclos de reloj necesarios para su ejecución o el estado del registro de indicadores F.

LD DESTINO, ORIGEN

Regresemos a las tablas de los apéndices. En ellas, bajo la denominación «carga de ocho bits», encontramos una colección de instruc-

REGISTRO	DECODIFICACION r ó r'
A	111
B	000
C	001
D	010
E	011
H	100
L	101

ciones, todas ellas de código mnemónico LD (abreviatura del inglés *L*oaD, cargar), encargadas de tramitar los movimientos de información que involucran a registros, posiciones de memoria y datos individuales.

Todas ellas responden al formato

LD Destino, Origen o LD Destino, Dato

Donde tanto Origen como Destino pueden ser registros o posiciones de memoria individuales y Dato un valor numérico almacenado en un octeto, y por tanto en el rango de 0 a 255.

Para determinar el código de operación de una de estas instrucciones entraremos en la tabla por la fila marcada con DESTINO y buscaremos la intersección con la columna ORIGEN, lugar en el cual encontraremos los códigos hexadecimales correspondientes.

Si allí aparece un cuadro en blanco la instrucción planteada no está contemplada dentro del juego presente en el microprocesador Z80 (no todo podía ser perfecto).

El resultado de una instrucción de «carga» trae consigo la «copia» del contenido de la posición de memoria o registro Origen en el registro o posición de memoria Destino, manteniéndose SIEMPRE inalterado el primero por la ejecución de la instrucción (en su momento comentaremos una excepción, la cual confirma la regla).

Efectuando un recorrido por la tabla correspondiente es factible la siguiente clasificación:

1. Carga en registros.

1.1. Entre registros: LD r,r'.

Con r y r' representamos cualquiera de los registros A, B, C, D, E, H o L y en la figura encontramos la decodificación seguida por el

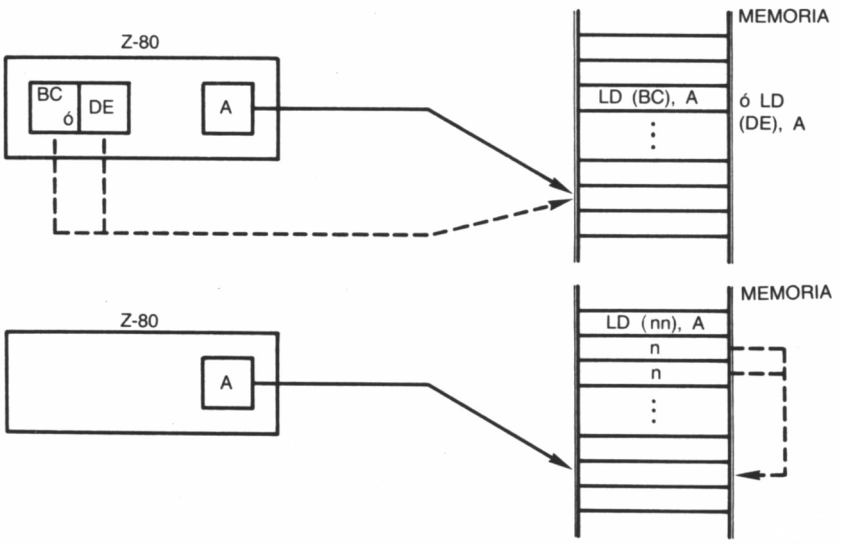
Z80 para reconocerlos, siendo el formato general de esta instrucción en código objeto (código máquina):

$$01(\text{-r-})(\text{-r'-})$$

Consideremos, por ejemplo, la instrucción LD A, B perteneciente a este grupo. En las tablas encontramos su código de operación asociado: 78h. Utilizando la decodificación dada en la figura obtenemos el byte 0111100, el cual no es otra cosa que la representación binaria del código anterior. Para su ejecución, el microprocesador necesita realizar una sola lectura de la memoria, pues toda la información que precisa está en el propio código de operación. Por tanto, empleará 4 ciclos de reloj, no siendo alterado por esta instrucción el registro de indicadores.

1.2. Carga inmediata: LD r, n.

EMSAMBLADOR	CODIGO MAQUINA	N.° BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD (BC), A	0 0 0 0 0 0 1 0	1	7	2
LD (DE), A		1	7	2
LD (nn), A	0 0 0 1 0 0 1 0	3	13	4
(BAJO)	0 0 1 1 0 0 1 0			
(ALTO)	n n n n n n n n			
	n n n n n n n n			



Mediante esta instrucción se almacena en el registro r, el valor numérico especificado en los 8 bits del dato n. Para ello son necesarios dos ciclos máquina: uno para localizar el código de operación (4 ciclos de reloj) y otro para leer el dato (3 ciclos) sin cambio en los indicadores de estado. Los dos bytes que la componen ofrecen el siguiente aspecto:

00(-r-)110
(- - -n- - -)

1.3. Carga indirecta: LD r, (HL).

Estas instrucciones recurren al par de registros HL como puntero de una determinada posición de memoria, donde leer el byte a ser copiado en el registro r.

La ejecución del único octeto que la conforma

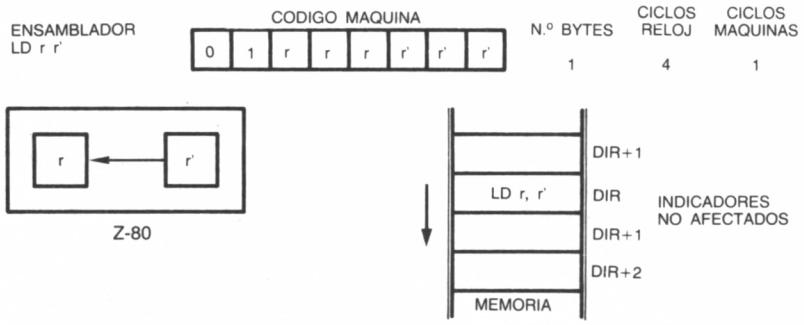
01(-r-)110

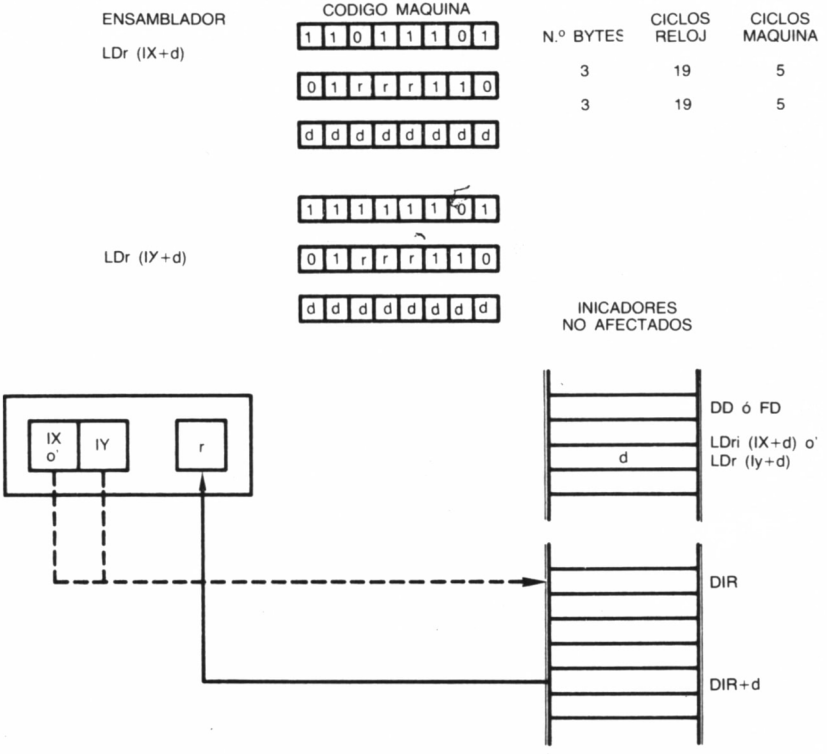
no influye en los indicadores de condición, siendo necesarios 2 ciclos de memoria, uno para la búsqueda del código de operación y otro para la lectura del dato señalado por HL. En total 7 ciclos de reloj, como ya sabemos calcular.

Idéntico comportamiento y comentarios son factibles sobre las instrucciones LD A, (BC) y LD A, (DE). Como observaremos, en ellas r debe ser ineludiblemente el acumulador A, pues de otra manera no están contempladas en el juego del Z80.

En este mismo grupo podríamos incluir una instrucción, la cual exclusivamente trabaja sobre el acumulador sirviéndose de direccionamiento extendido: LD A, (nn).

Mediante ella se almacena en éste el contenido de la posición de





memoria definida por los dos octetos nn, debiéndose especificar en primer lugar el byte bajo de tal dirección (recordemos que cuando aparecen los paréntesis en una instrucción escrita en ensamblador debe leerse «el contenido de...»).

Emplea 13 ciclos de reloj repartidos, 4 para buscar el código de operación, 3 más 3 para leer los dos octetos nn y finalmente otros 3 al buscar el dato contenido en la posición nn. Los indicadores no se ven afectados.

1.4. Carga Indexada: LD r, (IX+d) y LD r, (IY+d).

El objetivo de estas instrucciones es situar en el registro r el contenido de la posición de memoria obtenida al sumar el valor actual del par de registros índice IX o IY (según empleemos la primera o la segunda) y un byte de desplazamiento d expresado en complemento a dos, y, por tanto, en el rango −128 a +127.

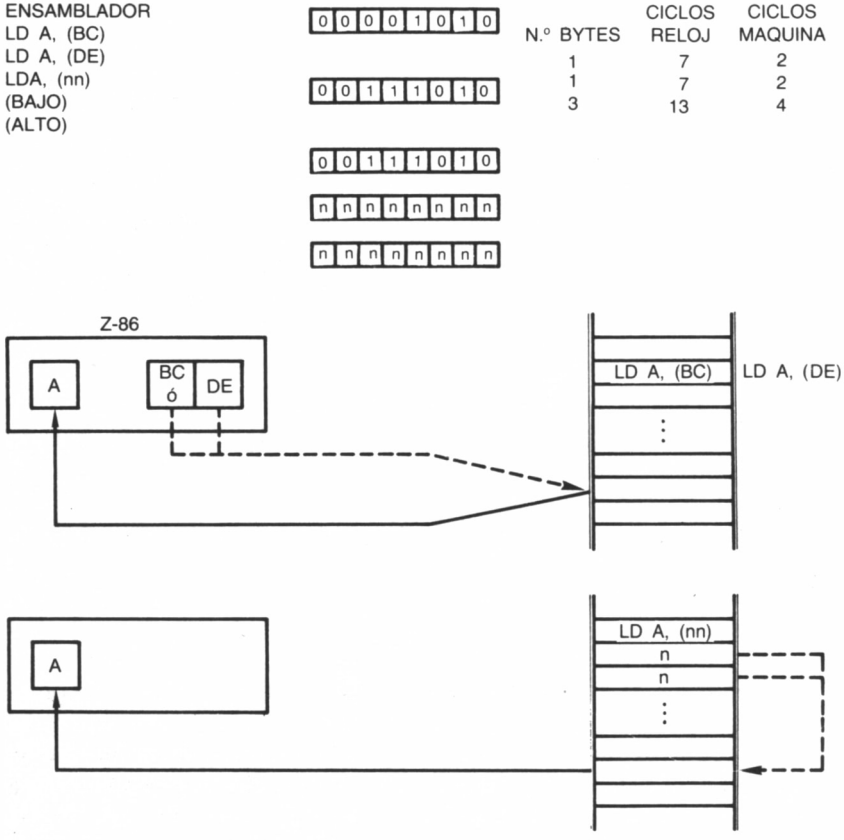
Aclarémoslo con un ejemplo: supongamos que deseamos ejecutar la instrucción LD B, (IX-10) y que en ese momento el par IX contiene

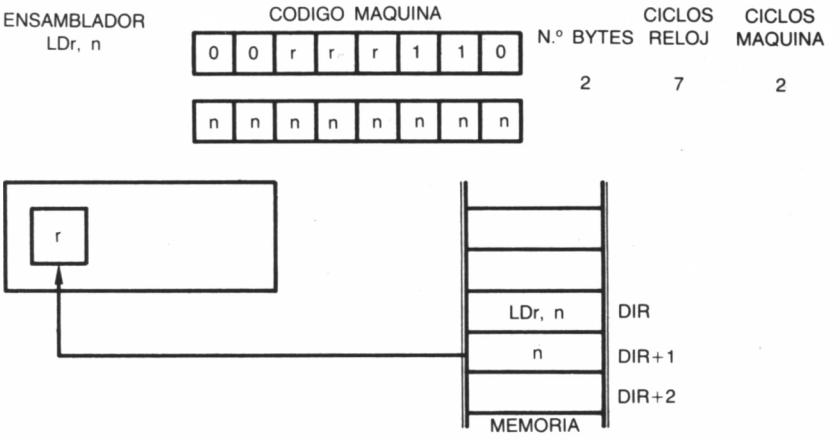
32767 (7FFFh). Busquemos en la tabla los códigos hexadecimales asociados: DD 46 d. Nos queda, por tanto, sustituir d por su valor en dicho sistema de numeración.

Pues su valor es F6. ¿Por qué? Recordemos que el desplazamiento hemos de expresarlo en complemento a dos, y precisamente F6 es la representación en este sistema del número decimal −10. Con ello la instrucción completa es DD 46 F6. ¿Qué ocurre al ejecutarla?

El microprocesador buscará el valor del par de registros IX (32767 en el ejemplo). A continuación, restará 10 unidades de éste, obteniendo 32757. Finalmente, extraerá los ocho bits almacenados en esta posición de memoria y los cargará en el registro B.

Podemos comprobar en las tablas que el primer byte de todas las





instrucciones que utilizan el registro IX es DD, y el de las que manejan el IY, FD.

2. Carga en memoria.

En cierto modo, las instrucciones del presente grupo pueden considerarse como inversas de las anteriores, al menos si tenemos en cuenta que el papel desempeñado por los registros o posiciones de memoria Origen y Destino afectados se intercambia respecto al sentido en que se efectuaba la transferencia de datos.

2.1. Carga indirecta: LD (HL), r y LD (HL), n.

Al ejecutar estas instrucciones se consigue almacenar en el octeto de memoria especificado por el par de registros HL el valor actual del registro r o del dato n. Volvemos a insistir en que, tras el proceso, tanto HL como r o n se mantienen inalterados.

Dentro de este mismo apartado es posible considerar tres instrucciones, las cuales sólo tienen sentido cuando el registro Origen es el acumulador A:

LD (BC), A
LD (DE), A
LD (nn), A

El funcionamiento es idéntico a las anteriores, pero el contenido de A es transferido a la dirección de memoria señalada por BC, DE o los dos bytes nn. Recordemos que al utilizar la última los octetos nn deberán especificarse en el orden bajo-alto.

2.2. Carga indirecta indexada:

LD (IX+d),r
LD (IX+d),n
LD (IY+d),r
LD (IY+d),n

Para establecer la posición de memoria donde almacenar el contenido del registro r o el dato n el microprocesador sigue el mismo sistema que explicábamos cuando nos referíamos, en el grupo 1.4, a las instrucciones que manejan direccionamiento indexado.

3. Carga sobre registros especiales:

LD A,I
LD A,R
LD I,A
LD R,A

Manejando las dos primeras conseguimos trasladar al acumulador el contenido del vector de página de interrupción I o el valor del registro de regeneración de memoria R, respectivamente. Son las únicas instrucciones dentro del grupo de carga de 8 bits que alteran el contenido del registro de indicadores. Veamos cuál es el efecto sobre ellos:

C: No es afectado por estas instrucciones.

Z: Se sitúa a 0 si I (R) vale 0.

S: Encontraremos un 1 cuando el bit más significativo de I (R) sea 1.
1.

N: Se pone a 0.

H: Contendrá 0.

P/V: Copia el estado de IFF2, es decir, 0 cuando las interrupciones están inhibidas y 1 cuando están habilitadas.

¿Qué quiere decir IFF? Entre los múltiples circuitos presentes en el Z80 dos son los encargados de señalarle cuando están habilitadas las interrupciones. Son denominados IFF1, el cual las permite o no, e IFF2, que sirve de almacenamiento temporal del estado de IFF1. Las siglas IFF corresponden, según su denominación inglesa, a *Interruption enable Flip-Flop,* y de ahí que en castellano normalmente se conozca como «báscula de habilitación de interrupciones».

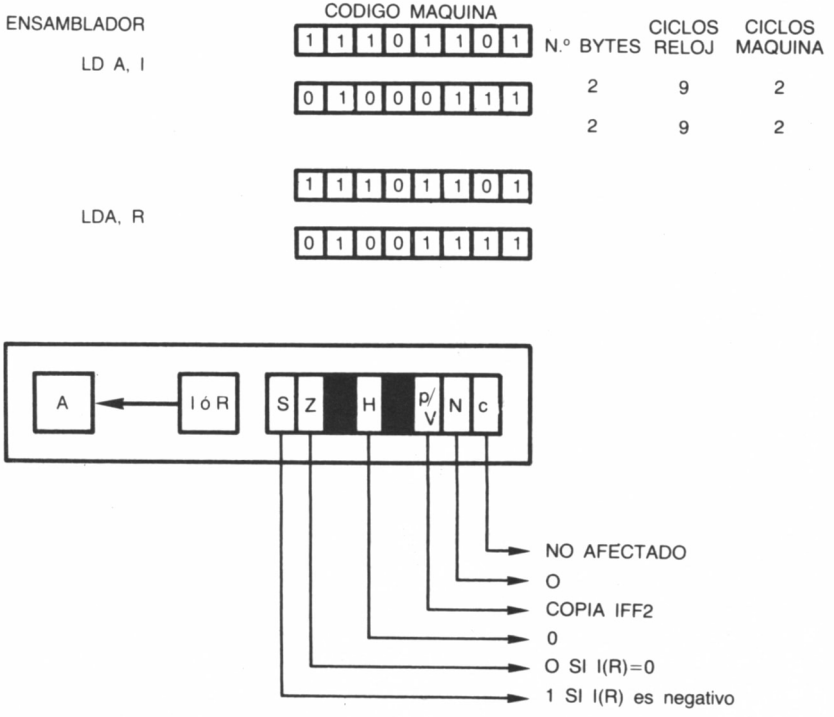
Cuando ejecutamos una instrucción EI (habilitar interrupción), IFF1 e IFF2 son puestos al unísono a 1. Al ser tratada ésta, simultáneamente pasan a ser 0, y deberemos definirlos por programa nuevamente a 1, si queremos tratar otra interrupción. Todo esto será comentado con detalle en el capítulo dedicado a las interrupciones.

Otra característica a tener en cuenta se refiere a la periódica variación a que está sujeto el registro R (recordemos la excepción comentada anteriormente sobre lo inalterable, tras procesar una instrucción, que se mantiene el contenido de los registros afectados). Esta circunstancia puede aprovecharse en los programas en que necesitemos generar números aleatorios.

Las instrucciones LD I,A y LD R,A introducen en los registros I o R, respectivamente, el contenido actual del acumulador sirviéndose de direccionamiento implícito. Con ello completamos el recorrido efectuado a través del grupo de carga de 8 bits.

EJERCICIOS

En este primer bloque de ejercicios pretendemos ilustrar las diferentes transferencias de información dentro del grupo de carga de 8 bits, haciendo uso de sus múltiples tipos de direccionamiento.



Para ello construiremos un programa, cuya primera parte será la carga de un dato, variable según el sistema de direccionamiento que queramos utilizar en cada caso. El final de la rutina será común en todos los ejercicios y simplemente servirá para visualizar en la pantalla el carácter con que hemos cargado el acumulador (A) en la fase anterior.

No debemos preocuparnos si desconocemos el modo de funcionamiento de esta última parte de la rutina. Nos conformaremos por el momento con saber que las instrucciones de ensamblador CALL #BB5A y RET simplemente se utilizan para presentar el carácter en la pantalla y devolver el control al BASIC, respectivamente.

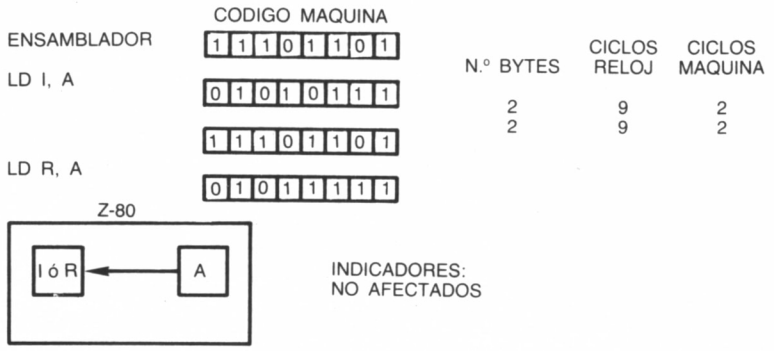
El carácter escogido en un principio ha sido la A mayúscula (código ASCII 65), aunque podemos alterarlo a nuestro gusto; de ser así, tengamos en cuenta que los caracteres por debajo del ASCII 32 son códigos de control, y por tanto en la mayoría de los casos no podremos apreciar el efecto que causa su emisión. En este sentido añadiremos que el código 7 es de especial interés, dado que permite emitir un pitido por el altavoz.

Para todos los ejemplos podremos utilizar como dirección de carga 30000.

CARGA INMEDIATA

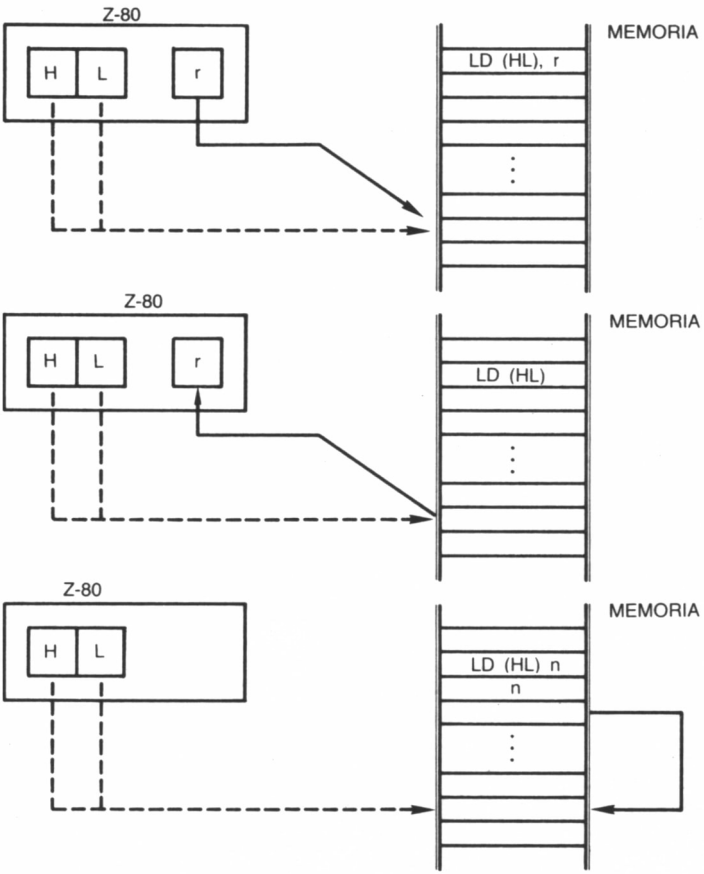
Dado que en nuestro supuesto, conocemos de antemano todos los datos, el sistema más rápido y corto para la codificación de la rutina será el de carga inmediata del dato a escribir en el acumulador.

```
LD A,65        ;Carga inmediata del acumulador con 65.
CALL #BB5A     ;Imprime el acumulador.
RET            ;Vuelve al BASIC.
```



ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD (HL), r	0 1 1 1 0 r r r	1	7	2
LD r, (HL)	0 1 r r r 1 1 0	1	7	2
LD (HL), n	0 0 1 1 0 1 1 0 n n n n n n n n	2	10	3

INDICADORES NO AFECTADOS



Si queremos utilizar el cargador BASIC...

```
1000 DATA 3E41CD5ABBC900000000000000000000000000,32A
```

CARGA ENTRE REGISTROS

Supongamos ahora que el dato a imprimir estuviera almacenado en otro registro. En nuestro caso utilizaremos para ello una carga inmediata del registro B. Así pues, sería necesaria una transferencia entre registros, del B al A, antes de efectuar la escritura del dato.

```
LD B,65        ;Carga el registro B con 65 para simular que el
                dato estaba contenido en éste.
LD A,B         ;Carga entre registros de B a A.
CALL #BB5A     ;Imprime el acumulador.
RET            ;Vuelve al BASIC.
```

Si queremos utilizar el cargador BASIC...

```
1000 DATA 064178CD5ABBC900000000000000000000000000,36A
```

En lo referente al uso de los registros especiales (R e I), el sistema empleado es el mismo, con la particularidad de que sólo puede emplearse el acumulador para la transferencia de información.

CARGA INDIRECTA

Para este ejemplo el dato se debe encontrar en una determinada dirección de memoria. Supondremos que dicha dirección es la 40000 (9C40 hex.); así pues, antes de efectuar la llamada a la rutina (CALL 30000) deberemos depositar el valor a escribir en la dirección 40000 mediante un POKE. Por ejemplo: POKE 40000,65: CALL 30000.

```
LD HL,#9C40    ;Carga el par de registros HL con la dirección en
                la que se encuentra el dato.
LD A,(HL)      ;Carga indirecta de A a través de HL.
CALL #BB5A     ;Imprime el acumulador.
RET            ;Vuelve al BASIC.
```

Si queremos utilizar el cargador BASIC...

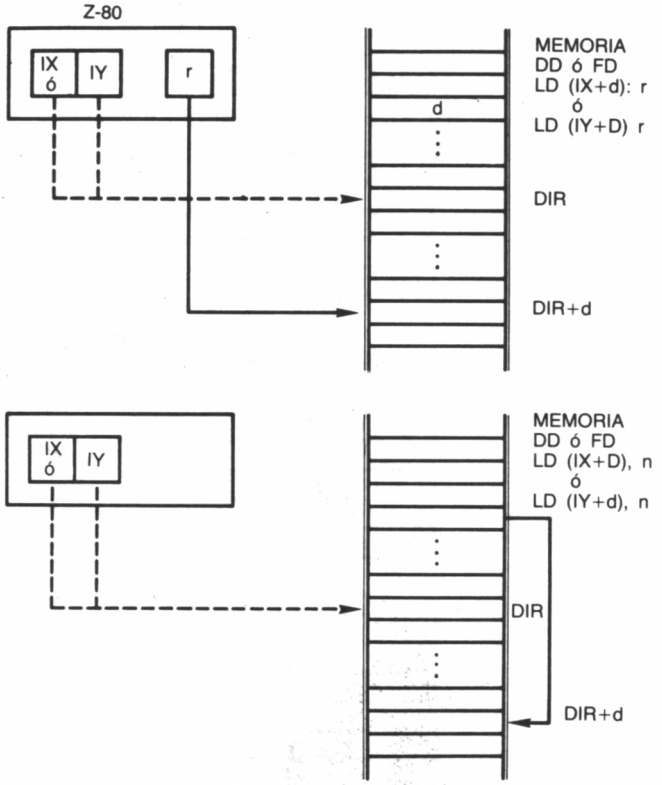
```
1000 DATA 21409C7ECD5ABBC900000000000000000000000000,426.
```

CARGA INDEXADA

Análogamente al ejemplo anterior deberemos introducir antes de efectuar la llamada a la rutina el dato en la posición 40005 (9C45 Hex.). Para acceder a dicha dirección nos serviremos del par de registros índice IX, el cual señalará a 40000 (9C40 Hex.), y un desplazamiento de 5, para finalmente tomar el dato e imprimirlo.

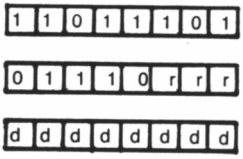
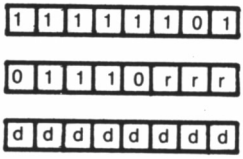
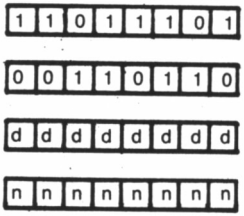
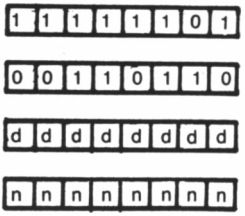
```
LD IX,#9C40   ;El par de registros IX apunta a la dirección 40000.
LD A,(IX+5)   ;Carga indexada del acumulador con el dato alma-
               cenado en IX+5 (40005).
CALL #BB5A;   ;Imprime el acumulador.
RET           ;Vuelve al BASIC...
```

```
1000 DATA DD21409CDD7E05CD5ABBC900000000000000000000,5E5
```



CARGA EN MEMORIA

Para ilustrar las técnicas de carga en memoria variaremos nuestro ejemplo anterior. Ahora, cargaremos el acumulador con una determinada cantidad (65) que depositaremos en una dirección de memoria, por ejemplo, 40000. Para comprobar la acción correcta de la rutina, una vez finalizada ésta, podremos averiguar el contenido de la posición mediante la función PEEK (PRINT PEEK (40000)). Con el fin de asegurarnos de que el dato ha sido correctamente alterado sería conveniente que antes de llamar la rutina almacenáramos un contenido determinado en la posición: POKE 40000,0: CALL 30000: PRINT PEEK (40000).

ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD (IX + d), r		3	19	5
LD (IY + d), r		3	19	5
LD (IX + d), n		4	19	5
LD (IY + d), n		4	19	5

Para la carga indirecta procederemos mediante direccionamiento inmediato a almacenar en el acumulador el dato (65) y acto seguido, a través del par HL, a depositarlo en la dirección 40000.

```
LD A,65          ;Carga el acumulador con el dato.
LD HL,#9C40      ;Carga HL con la dirección de memoria en la
                  que depositaremos el dato.
LD (HL),A        ;Transfiere el dato a la dirección marcada por HL.
RET              ;Vuelve al BASIC.
```

Si queremos utilizar el cargador BASIC..

```
1000 DATA 3E4121409C77C900000000000000000000000000,2BC
```

Las modificaciones para la carga indirecta indexada en memoria son bien pocas. Simplemente utilizaremos IX en vez de HL, y la dirección 40005 en lugar de 40000, para que el registro índice señale 40000 y se emplee un desplazamiento de 5.

```
LD A,65          ;Carga el acumulador con el dato.
LD IX,#9C40      ;Carga IX con la dirección base de memoria para
                  depositar el dato (40000).
LD (IX+5),A      ;Transfiere el dato (acumulador) a la dirección
                  IX+5 (400005).
RET              ;Vuelve al BASIC.
```

Si queremos utilizar el cargador BASIC...

```
1000 DATA 3E41DD21409CDD7705C900000000000000000000,47B
```

MOVIENDO 16 BITS

En el capítulo anterior discutíamos en detalle el grupo de carga de 8 bits. Nuestro próximo objetivo consistirá en efectuar otro tanto con las instrucciones encargadas de gestionar la transferencia de 16 bits entre registros dobles y la memoria de nuestro ordenador.

A ellas uniremos un grupo muy especial, el cual actúa sobre una importante zona de la RAM del Amstrad denominada stack o pila, encargada de conservar ciertos datos de interés durante la ejecución de un programa para su posterior utilización.

CARGA DE 16 BITS

Si observamos la tabla que lleva este nombre la encontraremos mucho menos poblada de lo que estaba la de carga de 8 bits, sobre todo en lo referente a transferencia de datos entre registros dobles.

Solamente cuando el destinatario de la información es el stack pointer SP es posible seleccionar algunos pares de registros como origen de ésta.

La última fila y la última columna están señaladas con PUSH y POP, respectivamente, instrucciones que discutiremos en breve. Pero concentrémonos en el resto de la tabla. Esta zona constituye el grupo de carga propiamente dicho y, por supuesto, el código mnemónico asociado es LD.

El formato es idéntico al grupo anterior:

LD DESTINO, ORIGEN o LD DESTINO, DATO

pero con la diferencia, ya advertida, de que tanto ORIGEN, DESTINO, así como DATO son registros, posiciones de memoria o cantidades de 16 bits. Lo que es lo mismo: abarcan dos octetos.

Para encontrar el código hexadecimal correspondiente a una instrucción entraremos en la tabla por la fila DESTINO y buscaremos su intersección con la columna ORIGEN. De nuevo, si en este lugar se haya un cuadro en blanco, la instrucción planteada no está contemplada en el juego del Z80. Efectuemos la siguiente clasificación:

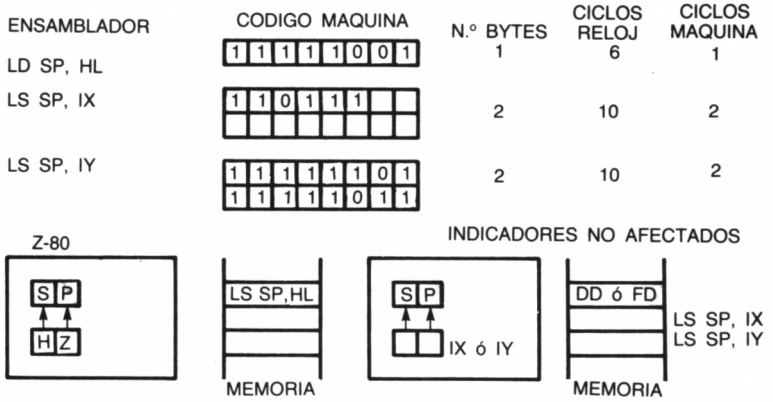
1. Carga en registros.

1.1. Inmediata extendida: LD dd, nn.

Con dd representamos cualquiera de los pares de registros BC, DE, HL o SP. En la figura se muestra la decodificación seguida por la CPU de nuestro Amstrad para reconocerlos. Tras ejecutarla, el valor entero determinado por los dos bytes nn, es decir, una cantidad comprendida entre 0 y 65535, se «copia» en el par de registros dd. Hemos

PAR	dd	qq
BC	00	00
DE	01	01
HL	10	10
SP	11	—
AF	—	11

10.7.1. Decodificación seguida por el Z-80 para reconocer los pares de registro indicados.



de tener presente el orden de los octetos nn: primero el de menor peso y en segundo lugar el más significativo.

A los registros dd es posible añadir los índice IX e IY, pues es factible almacenar directamente en ellos un entero nn mediante las instrucciones LD IX, nn y LD IY, nn.

Como en el capítulo anterior, en las tablas correspondientes está recogida toda la información accesoria sobre cada grupo particular de instrucciones.

1.2. Carga extendida LD dd (nn).

Para analizar el comportamiento de esta instrucción, lo mejor es recurrir a un ejemplo en el cual consideraremos que el par de registros dd es el BC. Supongamos, por tanto, que deseamos procesar LD BC (5FA6h) o en su correspondiente expresión decimal LD BC (24486d). Efectuemos también el supuesto de que la posición de memoria 24486 almacena el valor 176 y la 24487 conserva un 209.

Pues bien, tras leer el código de operación de dicha instrucción, el microprocesador procederá de la manera siguiente: buscará la posición de memoria 24486 y trasladará el dato allí almacenado (176) al octeto menos significativo del par de registros destino.

A continuación volverá a acceder a la memoria para recoger el dato situado en la dirección siguiente, es decir, en la 24487 y lo copiará en el byte de mayor peso del par BC, el cual tomará el valor 176 + 209 * 256 = 53680.

Las instrucciones de este tipo ocupan cuatro bytes, pero dentro del juego del Z80 sus fabricantes decidieron dar un trato preferencial a la pareja HL, de modo tal que cuando dd coincide con el par anterior (LD HL (nn) puede implementarse en sólo tres bytes. Por tanto,

es ejecutable en dos formatos distintos, pero en este último, consecuentemente, se ahorra memoria y tiempo.

También es factible que el destinatario de la información sea cualquiera de los registros índice empleando las instrucciones LD IX (nn) o LD IY (nn), cuyo funcionamiento es idéntico al de las consideradas en este mismo subgrupo.

2. Carga en memoria.

2.1. Carga extendida: LD (nn), dd.

Este grupo puede considerarse inverso del 1.2, el cual manejaba direccionamiento extendido para copiar la información almacenada en una pareja de registros en la memoria de nuestro ordenador.

Ahora el proceso es el contrario, es decir, la CPU toma el byte de menor peso contenido en el par dd y lo duplica en la celda de memoria direccionada por nn. A continuación, lee el octeto más significativo conservado en dd y lo copia en la siguiente posición a nn.

La misma precisión efectuada anteriormente cabe realizar cuando la instrucción a procesar es LD (nn), HL, pues como podemos comprobar en la tabla del grupo de carga de 16 bits es implementable en sólo tres bytes, aparte del formato previo que lo hace en cuatro.

Los registros índice IX e IY también aceptan la inversa, es decir, es factible duplicar su contenido en las posiciones de memoria definidas por nn y nn + 1.

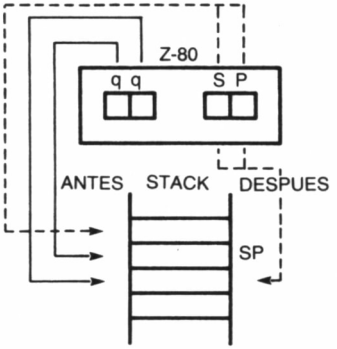
Conservando Los Datos

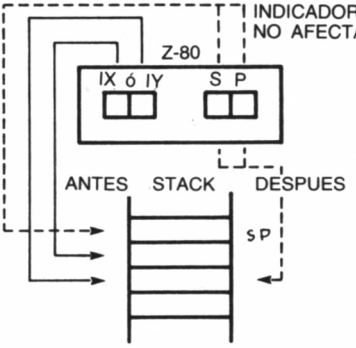
Quizá habremos advertido que las instrucciones relacionadas con el registro SP todavía no han sido discutidas (las señaladas en la tabla con PUSH y POP). Deliberadamente hemos retrasado su análisis hasta este punto, pues como comentábamos en la introducción del capítulo están ligadas con una zona de la memoria RAM de nuestro Amstrad denominada stack o pila, de la cual es muy importante conocer su particular modo de funcionamiento.

Durante la rutina de inicialización, es decir, cuando conectamos a la red el ordenador o tras pulsar CONTROL + MAYS + ESC, el sistema operativo mediante las instrucciones almacenadas en la ROM, cargan en el registro SP *(Stack Pointer* o puntero de la pila) el valor contenido en una de las variables del sistema, quedando situado en la zona alta de memoria.

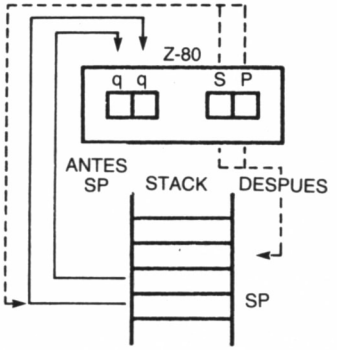
Pero a medida que el microprocesador o nosotros mismos vamos introduciendo datos en él, «crece» desde allí hacia las direcciones de

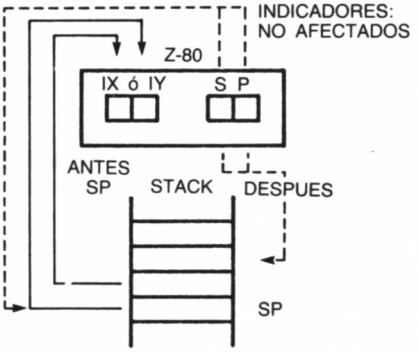
ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
POP qq	1 1 q q 0 0 0 1	1	10	3
POP IX	1 1 0 1 1 1 0 1 1 1 1 0 0 0 0 1	2	14	4
POP IY	1 1 1 1 1 1 0 1 1 1 1 0 0 0 0 1	2	14	4





ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
PUSH qq	1 1 q q 0 1 0 1	1	11	3
PUS IX	1 1 0 1 1 1 0 1 1 1 1 0 0 1 0 1	2	15	4
PUSH IY	1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1	2	15	4





memoria menores, es decir, el valor del registro doble SP disminuye a medida que el stack aumenta de tamaño y viceversa.

Esta circunstancia debe ser tenida muy en cuenta en la construcción de programas, pues un aumento incontrolado del stack puede traer como consecuencia la invasión por su parte del área de BASIC u otras zonas que en principio no hemos previsto para él, quedando adulterados por este motivo los resultados previstos.

Sobre la pila es posible efectuar dos operaciones: INTRODUCIR bytes (PUSH) o EXTRAERLOS (POP). Para ejecutarlas es necesario especificar siempre un par de registros como origen o destino de los datos intercambiados. Por tanto (es imprescindible tenerlo en cuenta) la transferencia de información entre registros y la pila o en sentido contrario es siempre de DOS en DOS bytes.

En la figura podemos encontrar los formatos bajo los cuales es posible implementar las instrucciones PUSH y POP. Veamos las operaciones llevadas a cabo por el Z80 al ejecutarlas. Supongamos que deseamos introducir (PUSH) un par de datos en la pila:

1. Se disminuye en 1 el contenido del par SP.
2. Se copia el contenido del byte de orden alto del par de registros origen en la posición de memoria indicada por SP.
3. Se decrementa SP.
4. Finalmente, el valor almacenado en el octeto de menor peso del par origen se duplica en la dirección apuntada por SP.

Ahora supongamos que queremos realizar el proceso inverso, es decir, extraer (POP) información de la pila para ser almacenada en un par de registros:

1. El contenido de la posición de memoria direccionada por SP se carga en el octeto de orden bajo del par de registros elegido.
2. Se le añade 1 al contenido del SP.
3. El byte indicado ahora por SP se transfiere al octeto más significativo del par correspondiente.
4. Nuevamente es incrementado en 1 el contenido del registro SP.

En la jerga informática esta forma de tratar los datos se denomina LIFO *(Last In, First Out:* último en entrar, primero en salir) y su utilidad es obvia si consideramos una cierta cantidad de información, la cual desborda la capacidad de los registros, y deseamos mantener estructurada de manera tal que tengamos fácil y rápido acceso al recuperarla.

Además, el microprocesador también necesita almacenar ciertos valores de interés para él, por ejemplo, la dirección a donde regresar

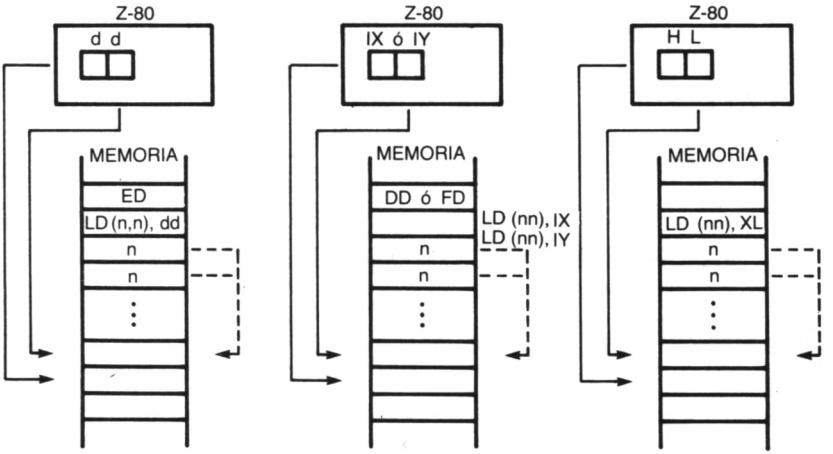
cuando termine de ejecutar una subrutina. Por ello, cuando hagamos uso de la pila debemos asegurarnos que el número de instrucciones PUSH y POP ejecutadas es el mismo.

En otro caso, el Z80 tomará los dos bytes indicados por el registro SP como dirección de retorno y el resultado puede convertirse en imprevisible (normalmente, el ordenador queda bloqueado y es necesario desconectarlo para poder continuar).

Por otra parte, en C/M es posible desplazar el puntero del stack SP a la posición de memoria determinada por un par de registros.

ENSAMBLADOR	CODIGO MAQUINA	BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD (nn), dd	1 1 1 0 1 1 0 1 0 1 d d 0 0 1 1 n n n n n n n n n n n n n n n n	4	20	6
LD (nn), IX	1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 n n n n n n n n n n n n n n n n	4	20	6
LD (nn), IY	1 1 1 1 1 1 0 1 0 0 1 0 0 0 1 0 n n n n n n n n n n n n n n n n	4	20	6
LD (nn), HL	0 0 1 0 0 0 1 0 n n n n n n n n n n n n n n n n	3	16	5

INDICADORES NO AFECTADOS



Para ello basta ejecutar cualquiera de las instrucciones siguientes: LD SP, HL, LD SP, IX o LD SP, IY.

La primera, como podemos comprobar en la tabla correspondiente, es una instrucción rápida que tan sólo ocupa un byte de memoria. Tras ejecutarla, el SP señalará a la posición de memoria definida por el contenido del par HL. Con esto cerramos la discusión emprendida en el capítulo anterior sobre las instrucciones de carga.

EJERCICIOS

Como la carga inmediata de pares de registros ya ha sido utilizada en los ejemplos del capítulo anterior (LD HL, #9C40, etc.), pasaremos directamente al ensayo del direccionamiento inmediato extendido. A tal fin, efectuaremos una rutina similar a las anteriores: visualizaremos un determinado carácter en la pantalla, con la particularidad de que el código a representar se encuentra almacenado en una dirección de memoria específica, que a su vez nos viene indicada desde otra posición.

En nuestro caso concreto, las direcciones 40000 y 40001 contendrán respectivamente 45 y 9C, cuyo significado en bajo-alto es 40005 (decimal); en esta dirección, y antes de la llamada a la rutina, habremos depositado mediante POKE el dato a escribir: POKE 40000, & 45: POKE 40001, &9C: POKE 40005, 65: CALL 30000. Así, a la entrada en la rutina las posiciones 40000 y 40001 señalan la dirección en la cual leer el dato (40005).

```
LD HL, (#9C40)  ;Carga inmediata extendida de HL.
LD A, (HL)      ;Almacena el dato en el acumulador.
CALL #BB5A      ;Escribe el dato.
RET             ;Regresa a BASIC.
```

Si queremos utilizar el cargador BASIC...

```
1000 DATA
2A409C7ECD5ABBC900000000000000000000000000,42F.
```

MANEJO DEL STACK

Como ejercicio sobre las técnicas de manejo de la pila recurriremos al diseño de una rutina de cierta utilidad, ya que se encuentra implementada como comando en los BASIC de algunos ordenadores:

SWAP. Su misión es intercambiar el contenido de dos variables.

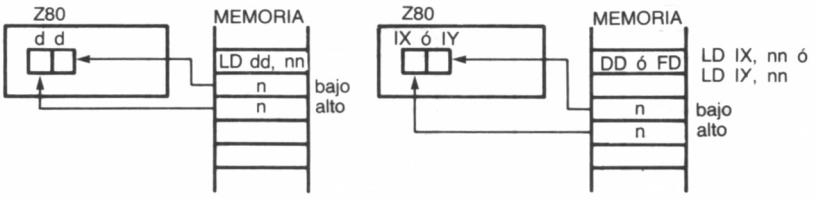
En nuestro caso concreto, las variables deberán ser numéricas enteras, y es necesario que se encuentren definidas previamente, dado que de no ser así aparecerá el mensaje *Improper Argument,* al no aceptarse el cero como valor por defecto.

Así, por ejemplo, suponiendo que ubicáramos la rutina a partir de la dirección 30000 y que quisiéramos intercambiar los contenidos de dos supuestas variables X% e Y%, que hubieran sido utilizadas con anterioridad, haríamos uso de la siguiente instrucción: CALL 30000,@X%,@Y%.

```
LD L,(IX+2)   ;Carga L con el byte bajo de la dirección de la primera variable.
LD H,(IX+3)   ;Carga H con el byte alto de la dirección de la primera variable.
PUSH HL       ;Guarda la dirección de la primera variable en el stack.
POP DE        ;Recupera la dirección de la primera variable en DE. DE = HL.
LD C,(HL)     ;Carga C con el byte bajo el contenido de la primera variable.
```

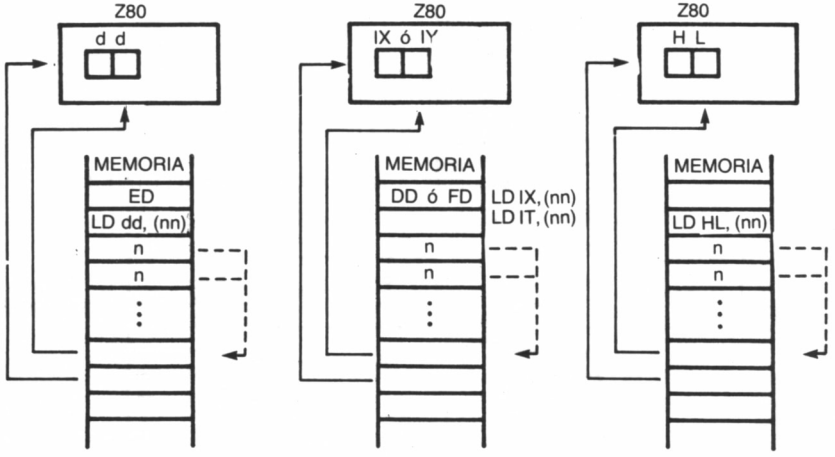
ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINAS
LD dd, nn	0 0 0 0 0 0 0 1 n n n n n n n n n n n n n n n n	3	10	3
LD IX, nn	1 1 0 1 1 1 0 1 0 0 1 0 0 0 0 1 n n n n n n n n n n n n n n n n	4	14	4
LD IY, nn	1 1 1 1 1 1 0 1 0 0 1 0 0 0 0 1 n n n n n n n n n n n n n n n n	4	14	4

INDICADORES: NO AFECTADOS



```
INC HL        ;Incrementa en uno el contenido de HL (HL=
                =HL+).
LD B,(HL)     ;Carga B con el byte alto del contenido de la pri-
                mera variable.
PUSH BC       ;Guarda el contenido de la primera variable en el
                stack.
LD L,(IX)     ;Carga L con el byte bajo de la dirección de la se-
                gunda variable.
LD H,(IX+1)   ;Carga H con el byte alto de la dirección de la
                segunda variable.
```

ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD dd (nn)	1 1 1 0 1 1 0 1 0 1 d d 1 0 1 1 n n n n n n n n n n n n n n n n	4	20	6
LD IX, (nn)	1 1 0 1 1 1 0 1 0 0 1 0 1 0 1 0 n n n n n n n n n n n n n n n n	4	20	6
LD IY, (nn)	1 1 1 1 1 1 0 1 0 0 1 0 1 0 1 0 n n n n n n n n n n n n n n n n	4	20	6
LD HL, (nn)	0 0 1 0 1 0 1 0 n n n n n n n n n n n n n n n n	3	15	5
		INDICADORES NO AFECTADOS		



```
PUSH HL     ;Guarda la dirección de la segunda variable.
LD A,(HL)   ;Carga A con el byte bajo del contenido de la se-
              gunda variable.
LD (DE),A   ;Deposita A en el byte bajo del contenido de la
              primera variable.
INC DE      ;Incrementa en uno la dirección a la que apun-
              ta DE.
INC HL      ;Incrementa en uno la dirección a la que apun-
              ta HL.
LD A,(HL)   ;Carga A con el byte alto del contenido de la se-
              gunda variable.
LD (DE),A   ;Deposita A en el byte alto del contenido de la
              primera variable.
POP HL      ;Recupera la dirección inicial de la segunda va-
              riable.
POP DE      ;Recupera el contenido original de la primera va-
              riable, que fue almacenado con PUSH BC.
LD (HL),E   ;Copia el byte bajo del contenido de la primera
              variable en la segunda.
INC HL      ;Incrementa en uno el contenido de HL, para
              apuntar al byte alto del contenido de la segunda
              variable.
LD (HL),D   ;Copia el byte alto del contenido de la primera
              variable en la segunda.
RET         ;Retorna al BASIC.
```

Si queremos utilizar el cargador BASIC.

```
1000 DATA
DD6E02DD6603E5D14E2346C5DD6E00DD6601E57E,9B7
1010 DATA
1213237E12E1D1732372C900000000000000000000,45B
```

MOVIMIENTOS DE BLOQUES

En las próximas líneas discutiremos tres nuevos grupos de instrucciones denominados en las tablas del apéndice «intercambio, transferencia y búsqueda de bloques», cuya misión consiste en evitarnos la construcción de complicadas subrutinas cuando pretendamos realizar determinadas tareas repetitivas.

EL GRUPO DE INTERCAMBIO

Las instrucciones que lo componen provocan el intercambio de la información contenida en los registros implicados. No tiene sentido hablar de origen y destino, pues el intercambio es biyectivo.

Consideremos la primera de ellas: EX DE, HL (el mnemónico EX responde a la abreviatura de la palabra inglesa *EXchange,* canje o intercambio). Mediante ella, el valor almacenado en el par DE se transfiere al par HL, y simultáneamente el contenido de HL pasa a DE.

Las instrucciones de intercambio habilitan el uso del juego de registros alternativos, el cual se utiliza habitualmente en C/M para conservar valores o direcciones de interés que requieran una fácil y rápida recuperación, a la par de seguridad ante una posible corrupción como puede ocurrir, por ejemplo, con los datos almacenados en el *stack*.

EX, AF, A'F', efectúa el canje de los valores contenidos en los pares AF y A'F'. Finalmente, completa el grupo de intercambio una potente instrucción: EXX, la cual de una sola vez intercambia los datos almacenados en los registros BC, DE y HL con los contenidos de B'C', D'E' y H'L', respectivamente.

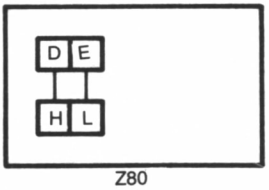
MANEJANDO BLOQUES

Entre las instrucciones desarrolladas para el microprocesador Z80, ocho fueron las previstas para gestionar la transferencia y búsqueda de bloques. Las primeras permiten al programador trasladar un área de memoria de una zona a otra de ésta. Las segundas examinan si una determinada configuración se haya presente dentro de un bloque de datos.

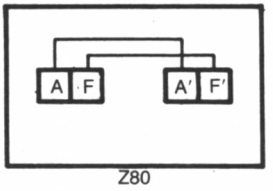
Cuando efectuemos la transferencia de un bloque, la dirección de origen de éste debemos definirla previamente en el par HL, y la dirección de destino, en el par DE. Por su parte, el par de registros BC debe contener el número de bytes a ser transferidos.

ENSAMBLADOR	CODIGO MAQUINA	N.° BYTES	CICLOS RELOJ	CICLOS MAQUINA
EX DE, HL	1 1 1 0 1 0 1 1	1	4	1
EX AF, AF'	0 0 0 0 1 0 0 0	1	4	1

INDICADORES: NO AFECTADOS

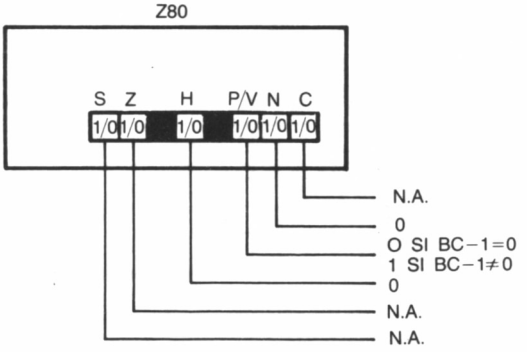


Z80



Z80

ENSAMBLADOR	CODIGO MAQUINA	N.° BYTES	CICLOS RELOJ	CICLOS MAQUINA
LDI	1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 0	2	16	4
LDD	1 1 1 0 1 1 0 1 1 0 1 0 1 0 0 0	2	16	4



Si nuestro objetivo es examinar un área determinada de memoria al objeto de confirmar o no si un valor particular está presente en ella, situaremos el origen de ésta en el par HL, el número de bytes a comparar en BC, y en el acumulador A, el valor particular a ser contrastado.

Podemos clasificarlas en dos subgrupos:

1) Funcionamiento automático (LDIR, LDDR, CPIR, CPDR).

Estas instrucciones ejecutan cíclicamente su trabajo hasta que una determinada condición les indica que ha finalizado éste. Por ejemplo, supongamos que deseamos transferir un bloque de una zona a otra mediante la orden LDIR (LD-cargar, I-incrementar, R-repetir). Internamente, el microprocesador efectúa las siguientes operaciones:

— T1: Carga el contenido de la posición de memoria direccionada por DE (como sabemos se trata de la dirección destino) con el octeto señalado por HL (dirección origen o fuente).

— T2: Incrementa en 1 el par DE, así como el HL, al objeto de transferir el siguiente dato a la próxima posición de destino.

— T3: Decrementa en 1 el contador de bytes BC.

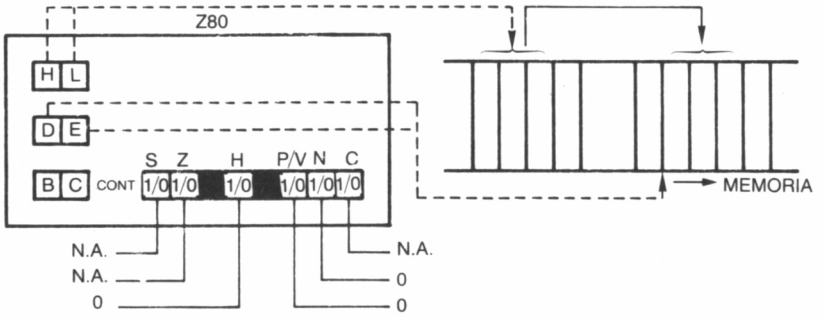
— T4: El mismo proceso es ejecutado una y otra vez, hasta que por fin el par BC es 0, es decir, todos los datos han sido transferidos, momento en el cual termina la instrucción.

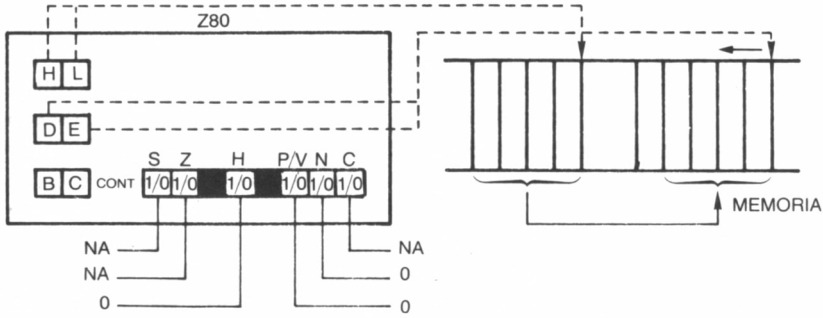
Idénticas consideraciones es factible efectuar sobre LDDR (LD-cargar, D-decrementar, R-repetir): los pasos T1, T3 y T4 son los mismos y la diferencia estriba en que los registros DE y HL son decrementados en 1, en vez de incrementados en el paso T2.

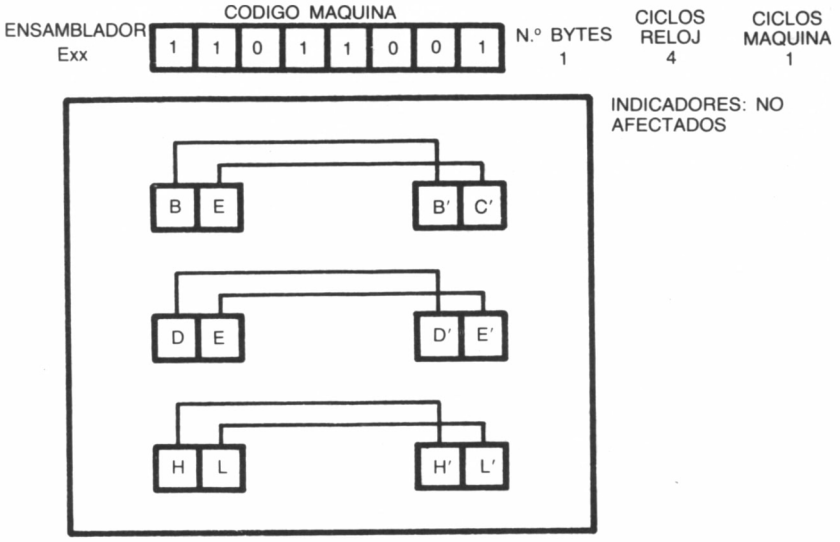
Las instrucciones de búsqueda actúan de manera similar. Sea, por ejemplo, nuestro objeto ejecutar CPIR (CP-comparar, I-incrementar, R-repetir) sobre un bloque de memoria determinado. Para ello, el Z80 llevará a cabo las operaciones siguientes:

— B1: Compara el contenido del acumulador con el byte direccionado por HL.

ENSAMBLADOR	CODIGO MAQUINA	N.° BYTES	CICLOS RELOJ	CICLOS MAQUINA
LDIR	1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 0	2	21BC≠0 16BC=0	5 4
LDDR	1 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0	2	21BC≠0 16BC=0	5 4







— B2: Incrementa en 1 el par de registros HL.

— B3: Decrementa en 1 el contador de bytes a ser examinados (BC).

— B4: Repite el proceso anterior hasta que, o bien se encuentra en la zona escrutada un valor idéntico al del acumulador, o bien, no quedan más bytes a ser comparados, circunstancia satisfecha cuando BC es 0.

CPDR (CP-comparar, D-decrementar, R-repetir) es a las instrucciones de búsqueda lo que LDDR es a las de transferencia de bloques. La única diferencia en cuanto a funcionamiento con CPIR reside en el paso B2, donde el par HL es decrementado en vez de incrementado en 1.

2) Mecanismo no automático (LDI, LDD, CPI, CPD).

Estas cuatro instrucciones tienen en común con las del grupo anterior los tres primeros pasos indicados, pero carecen de la autorrepetición. Por tanto, al construir un programa que las implemente deberemos hacer uso de ellas, una vez por cada byte a ser transferido o comparado.

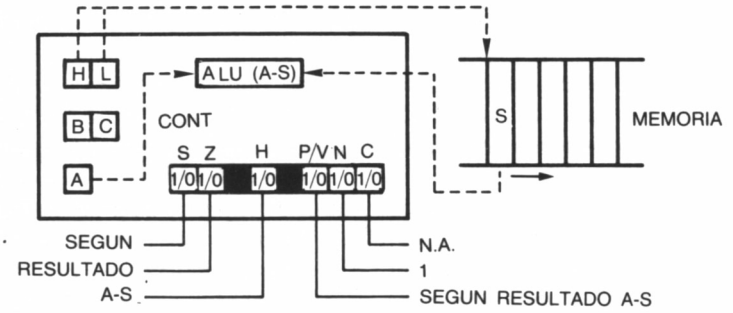
EJERCICIOS

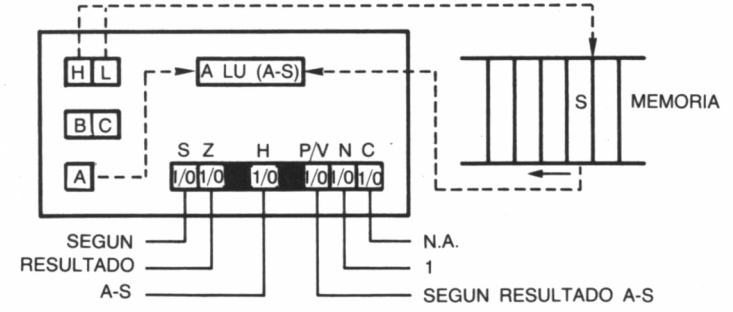
El desplazamiento de bloques tendrá, gracias al ejemplo siguiente, una demostración palpable en la pantalla. Para apreciar su efecto,

antes de ejecutar la rutina deberemos rellenar la fila superior de la pantalla con parejas de caracteres 143 (bloque que ocupa la totalidad de un carácter) y espacio (código 32). A tal fin podremos utilizar la siguiente rutina: MODE 0: FOR I = 0 TO 39: PRINT CHR$(143)CHR$(32);: NEXT: CALL 30000

Nuestra subrutina efectuará un scroll horizontal de la primera línea de los caracteres impresos. Para ello hará uso de la instrucción LDIR. El resto de los códigos de operación de bloques se utilizan de forma similar.

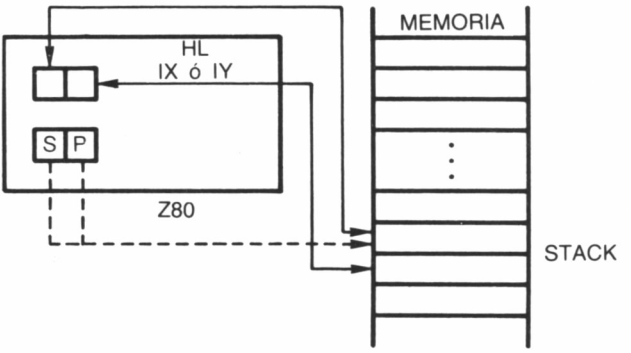
ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
CPIR	1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1	2	21BC≠0 y A≠(HL) 16BC=0 ó A=(HL)	5 4
CPDR	1 1 1 0 1 1 0 1 1 0 1 1 1 0 0 1	2	21BC≠0 y A≠(HL) 16BC=0 ó A=(HL)	5 4





EMSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
EX (SP), HL	1 1 1 0 0 0 1 1	1	19	5
EX (SP), IX	1 1 0 1 1 1 0 1 1 1 1 0 0 0 1 1	2	23	6
EX (SP), IY	1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1	2	23	6

INDICADORES: NO AFECTADOS




```
LD HL,#C001   ;Primera dirección de pantalla.
LD DE,#C000   ;DE = HL - 1.
LD BC,#4F     ;Número de bytes a desplazar por línea (80).
LDIR          ;Efectúa la transferencia.
RET           ;Regresa a BASIC.
```

Si utilizamos el cargador BASIC...

```
1000DATA
2101C01100C0014F00EDB0C90000000000000000,469
```

LAS INSTRUCCIONES LÓGICAS

Llegado este apartado, recomendamos antes de proseguir con él repasar el capítulo dedicado al Álgebra de Boole (volumen 1 de esta misma colección, capítulo 5), punto de partida de todo lo que seguirá a continuación.

Las instrucciones lógicas enfrentan dos octetos entre sí, efectuando la operación (AND, OR, XOR) entre sus bits individuales, tomándolos por parejas formadas por aquellos que ocupan el mismo lugar dentro de los bytes considerados.

El primero de los bytes a operar ha de estar contenido en el acumulador A, y el segundo en un registro o localización de memoria específica. El resultado siempre se recoge en A.

EL OPERADOR AND

En la figura está representada su tabla de verdad. Como observaremos, efectúa el producto lógico entre los bits implicados, es decir,

el resultado es 1 si, y sólo si, los dos son 1, siendo 0 si al menos uno de ellos es 0.

Una de las operaciones más habituales en los programas en C/M consiste en aplicarlo sobre el mismo acumulador, es decir, implementar la instrucción AND A. Con ello, el contenido de este registro se mantiene inalterado, pero modifica algunos bits del registro F, de tal manera que es posible identificar si el valor almacenado en A es 0, negativo o tiene un número par de unos (paridad par).

EL OPERADOR OR

Al ejecutar la instrucción OR s, donde s es cualquier valor almacenado en los registros o posiciones de memoria que se detallan en las figuras, se efectúa la suma lógica entre los bits correspondientes de los dos octetos implicados.

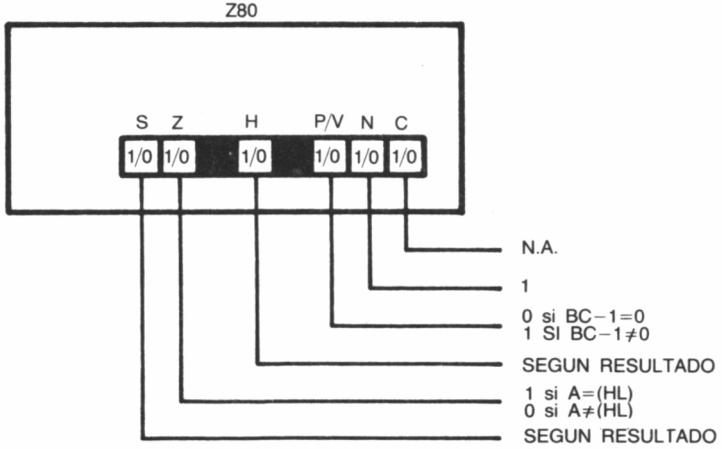
Hemos de tener presente que en las operaciones lógicas no se producen acarreos, es decir, el resultado obtenido tras aplicar cualquiera de los operadores sobre una pareja de bits no tiene ninguna influencia sobre los siguientes, siendo tratadas de forma independiente.

Al igual que antes analizando el contenido del registro F tras implementar la instrucción OR A, estamos en condiciones de precisar si el dato almacenado en el acumulador es negativo, cero o tiene paridad par.

ORs

OPERANDO	HEX	DEC	INDICADOR	
n	F6,n	246,N	S	1 si resultado − 0 si resultado +
A	B7	183	Z	1 si resultado 0 0 en caso contrario
B	B0	176		
C	B1	177	H	0
D	B2	178		
E	B3	179	P/V	1 paridad par 0 paridad impar
H	B4	180	N	0
L	B5	181		
(HL)	B6	182	C	0
(IX+d)	DD,B6,d	221,182,d		
(IY+d)	FD,B6,d	253,182,d	S OPERANDO	

ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
CPI	1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 1	2	16	4
CPD	1 1 1 0 1 1 0 1 1 0 1 0 1 0 0 1	2	16	4



ANDs				
OPERANDO	HEX	DEC	INDICADOR	
n	E6,n	230,n	S	1 si resultado − 0 si resultado +
A	A7	167	Z	1 si resultado 0 0 en caso contrario
B	A0	160	H	1
C	A1	161	P/V	1 paridad par 0 paridad impar
D	A2	162	N	0
E	A3	163	C	0
H	A4	164	S OPERANDO	
L	A5	165		
(HL)	A6	166		
(IX+d)	DD,A6,d	222,166,d		
(IY+d)	FD,A6,d	253,166,d		

OR EXCLUSIVO, XOR

El resultado de este operador aplicado sobre dos octetos determinados conduce a la comparación lógica de ambos, es decir, cuando enfrentamos dos bits se obtiene 1 si son diferentes, y 0 en el caso de ser iguales.

Muchos programas utilizan la instrucción XOR A como sustituto de LD A, 0, siempre y cuando no tenga importancia modificar el contenido del registro F, puesto que se consigue ahorrar un byte por dicho sistema.

MÁSCARAS

Una de las utilidades más importantes ligada al empleo de las instrucciones lógicas es la de permitir una técnica de programación basada en las máscaras (en inglés *mask).* En la tabla podemos encontrar algunos ejemplos relacionados con ellas.

Quizá fuera más preciso utilizar el término «filtro», pues el objetivo final de dicha técnica consiste en seleccionar ciertos bits de interés para nosotros dentro de un octeto, obviando o inhibiendo los restantes.

COMPARACIONES

Para completar el grupo lógico de 8 bits nos queda analizar las instrucciones de código mnemónico CP *(ComPare,* comparar), las cuales confrontan el contenido del acumulador con un byte s determinado por el valor almacenado en un registro o posición de memoria específica.

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

XORs				
OPE-RANDO	HEX	DEC	INDICADOR	
n	EE,n	238,n	S	1 si resultado − 0 si resultado +
A	AF	175	Z	1 si resultado 0 0 en caso contrario
B	A8	168	H	0
C	A9	169		
D	AA	170	P/v	1 si paridad par 0 si paridad impar
E	AB	171	N	0
H	AC	172		
L	AD	173	C	0
(HL)	AE	174		
(IX+d)	DD,AE,d	221,174,d	S OPERANDO	
(IY+d)	FD,AE,d	253,174,d		

Como sabemos, el resultado de una comparación solamente ofrece dos posibilidades: o las dos cantidades son iguales o son diferentes. Cuando se trata del segundo caso puede interesarnos conocer cuál es la mayor (o cuál es la menor).

El Z80 soluciona la papeleta con habilidad ejecutando internamente la resta A−s y activando algunos indicadores, los cuales delatan sin duda el resultado de la comparación, manteniendo inalterados los contenidos tanto del acumulador como del byte s. Comprobémoslo:

— Si la bandera de cero Z está alzada, es decir, contiene un 1, entonces la resta A−s fue 0, y, por tanto, A y s son iguales.

— En caso contrario Z contendrá un 0. Nos fijamos ahora en el indicador de acarreo C. Si está a 0, quiere decir que no hubo arraste en el bit más significativo del acumulador durante la resta, condición que se cumple cuando A es mayor que s.

— De no ser así, la bandera C estará levantada señalando que s es mayor que el valor almacenado en el acumulador.

Como siempre, en las tablas se presenta toda la información necesaria para la correcta interpretación de cada instrucción en particular.

EJERCICIOS

Como ejercicio de este capítulo haremos especial hincapié en el

1. Eliminar ciertos bits dentro de un byte (ej.: los seis superiores)

0	1	1	0	0	1	1	0	
0	0	0	0	0	0	1	1	MASCARA AND
0	0	0	0	0	0	1	0	RESULTADO

2. Paso de minúscula a mayúscula

0	1	0	0	0	0	0	1	A.S.C.I.I. de la «A»
0	0	1	0	0	0	0	0	MASCARA OR
0	1	1	0	0	0	0	1	RESULTADO (A.S.C.I.I. de la «a»)

uso de las instrucciones lógicas para la creación de máscaras. Para nuestro ejemplo tomaremos como área de datos las direcciones comprendidas entre 40000 y 40002, ambas inclusive. En las dos primeras, y antes de efectuar la llamada a la rutina, deberemos depositar el valor inicial y la máscara, respectivamente. A la vuelta al BASIC podremos consultar mediante PEEK (40002) el resultado del enmascaramiento.

Para apreciar mejor su efecto es conveniente que empleemos el sistema binario para la introducción de los datos y el análisis del resultado. Así, por ejemplo, podremos comprobar el efecto que produce un enmascaramiento (en el ejercicio, con AND) de 170 (10101010)

CPs				
OPERANDO	HEX	DEC	INDICADORES	
n	FE, n	254, n	S	1 si resultado − 0 si resultado +
A	BF	191	Z	1 si resultado 0 0 en caso contrario
B	B8	184	H	1 si acarreo del BIT3
C	B9	185	P/V	1 si hay sobrepasamiento
D	BA	186	N	1
E	BB	187	C	1 si hay acarreo
H	BC	188		
L	BD	189		
(HL)	BE	190		
(IX+d)	DD,BE,d	221,190,d	S OPERANDO	
(IY+d)	FD,BE,d	223,190,d		

con 15 (00001111). Así, suponiendo 30000 como dirección de carga de la rutina: POKE 40000, &X10101010: POKE 40001, &X00001111: CALL 30000: PRINT RIGHT$(«0000000» + BIN$ (PEEK (40002)), 8).

En el diseño inicial de la rutina se ha empleado para el enmascaramiento la función AND, aunque sin ningún problema podremos sustituirla por OR o XOR para observar su efecto. En este último caso será de especial importancia que comprobemos el resultado de XOR sobre dos datos iguales: cero. Por ello, una forma habitual de conseguir poner a cero el acumulador, utilizando un solo byte, es XOR A, puesto que cualquiera que sea el valor anterior del acumulador siempre se hará cero, al someterse a un XOR consigo mismo.

```
LD A,(40000)   ;Carga el acumulador con el dato inicial.
LD B,A         ;Deposita en B el dato inicial.
LD A,(40001)   ;Carga el acumulador con la máscara.
AND B          ;Aplica la máscara A sobre el dato contenido en B.
                y deja su resultado en el acumulador.
LD (40002),A   ;Deposita el resultado de la operación (A) en la
                dirección 40002.
RET            ;Retorna al BASIC.
```

Si utilizamos el cargador BASIC...

```
1000 DATA
3A409C473A419CA032429CC90000000000000000,4ED.
```

MICROARITMÉTICA

La ALU (unidad aritmética y lógica) situada en el interior del Z80 es la encargada de gestionar, entre otros trabajos, aquellos que implican la ejecución de sumas, restas y manejo de los operadores lógicos AND, OR y XOR.

Estos últimos ya fueron discutidos en el capítulo anterior, por lo que en éste nos centraremos en las instrucciones aritméticas previstas por los fabricantes del microprocesador Z80.

ARITMÉTICO DE 8 BITS

Pertenecen a este grupo todas aquellas instrucciones que efectúan la suma (ADD) o resta (SUB) de un valor contenido en un registro o posición de memoria con el almacenado en el acumulador, quedando el resultado en este último y manteniéndose inalterado el valor del registro o posición de memoria implicada tras la operación aritmética.

Genéricamente podríamos representar estas operaciones de la manera siguiente:

SUMA A+n->A.
RESTA A-n->A.

donde A representa el valor almacenado en el acumulador y n el contenido del registro o posición de memoria especificado en la instrucción.

MNEMONICO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	N.º BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
ADD HL, SS	0 0 s s 1 0 0 1	•	•		X		•	0	↕	1	3	11	
ADCHL, SS	1 1 1 0 1 1 0 1 0 1 s s 1 0 1 0	↕	↕		X		V	0	↕	2	4	15	
SBCHL, SS	1 1 1 0 1 1 0 1 0 1 s s 0 0 1 0	↕	↕		X		V	1	↕	2	4	15	
ADDIX,pp	1 1 0 1 1 1 0 1 0 0 p p 1 0 0 1	•	•		X		•	0	↕	2	4	15	
ADDIY,rr	1 1 1 1 1 1 0 1 0 0 r r 1 0 0 1	•	•		X		•	0	↕	2	4	15	
INC,SS	0 0 s s 0 0 1 1	•	•		•		•	•	•	1	1	6	
INC IX	1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1	•	•		•		•	•	•	2	2	10	
INC IY	1 1 1 1 1 1 0 1 0 0 1 0 0 0 1 1	•	•		•		•	•	•	2	2	10	
DEL SS	0 0 s s 1 0 1 1	•	•		•		•	•	•	1	1	6	
DEC IX	1 1 0 1 1 1 0 1 0 0 1 0 1 0 1 1	•	•		•		•	•	•	2	2	10	
DEC IY	1 1 1 1 1 1 0 1 0 0 1 0 1 0 1 1	•	•		•		•	•	•	2	2	10	

REG	SS
BC	00
DE	01
HL	10
SP	11

REG	PP
BC	00
DE	01
IX	10
SP	11

REG	rr
BC	00
DE	01
IY	10
SP	11

10.10.1. Grupo aritmético de 16 bits.

Además es posible incrementar (INC) o decrementar (DEC) en 1 la cantidad almacenada en un determinado registro. Veamos el efecto que sobre los indicadores tienen estas operaciones:

S.—Cuando el resultado en A es negativo, es decir, si el bit más

MN EMONICO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	N.º BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
ADDr	1 0 0 0 0 r r r	↕	↕		↕		V	0	↕	1	1	4	
ADDn	1 1 0 0 0 1 1 0 n n n n n n n n	↕	↕		↕		V	0	↕	2	2	7	
ADD(HL)	1 0 0 0 0 1 1 0	↕	↕		↕		V	0	↕	1	1	4	
ADD(IY+d)	1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 1 0 d d d d d d d d	↕	↕		↕		V	0	↕	3	5	19	
ADD(IX+d)	1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 0 d d d d d d d d	↕	↕		↕		V	0	↕	3	5	19	
ADCs	0 0 1	↕	↕		↕		V	0	↕	(a)	(b)	(c)	
SUBs	0 1 0	↕	↕		↕		V	1	↕				
SBCS	0 1 1	↕	↕		↕		V	1	↕				
INCr	0 0 r r r 1 0 0	↕	↕		↕		V	0	•	1	1	4	
INC(HL)	0 0 1 1 0 1 0 0	↕	↕		↕		V	0	•	1	3	11	
INC(IX+d)	1 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 0 d d d d d d d d	↕	↕		↕		V	0	•	3	6	23	
	1 1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 0 d d d d d d d d	↕	↕		↕		V	0	•	3	6	23	
DECs	1 0 1	↕	↕		↕		V	1	•				

Reg	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

s
r
n*
(HL)
(IX+d)
(IY+d)

* salvo en DECS

10.10.2. Grupo aritmético de 8 bits.

significativo del acumulador es 1, la bandera de signo contendrá un 1. En caso contrario será 0.

Z.—Tras una operación aritmética, su valor será 1 si el resultado de ésta fue 0. De lo contrario, contendrá un 0.

E.—El indicador de resta se sitúa a 1 en todas las operaciones relacionadas con esta operación, mientras que en las sumas permanece a 0.

P/V.—Señala si hubo sobrepasamiento (1) o no (0) en las operaciones aritméticas de dos números expresados en complemento a dos.

C.—Actúa de manera similar al anterior, pero con números expresados en su correspondiente formato binario.

H.—Su importancia se pone de manifiesto cuando ejecutamos operaciones aritméticas con números en formato BCD (decimal codificado en binario), los cuales serán discutidos próximamente al analizar la instrucción DAA.

En el apéndice dedicado a la presentación de las tablas de instrucciones se señalan los operandos sobre los cuales actúan las de mnemónico ADD, SUB, INC y DEC. Como podemos comprobar, además están contempladas dos operaciones especiales para la suma y la resta, las cuales tienen presente el valor de la bandera de arrastre C. Se trata de aquellas cuyo mnemónico es ADC (sumar con acarreo) y SBC (restar con acarreo).

Implementar estas instrucciones obliga al microprocesador a seguir los siguientes pasos basados en el esquema:

SUMA con acarreo A+n+C->A.
RESTA con acarreo A-n-C->A.

1) Suma o resta al/del acumulador el byte especificado en la instrucción como si se tratara, al igual que antes, de ADD o SUB, pero sin modificar ningún bit del registro de indicadores.

2) El valor del bit de acarreo es sumado o restado al/del acumulador, es decir, si previamente a la ejecución de la instrucción estaba alzado (1), se suma o resta 1 de A.

3) Finalmente el Z80 ajusta las banderas del registro F, basándose en el último resultado recogido en A.

El Grupo Aritmético De 16 Bits

Las instrucciones aritméticas anteriormente comentadas ADD,

ADC, SBC, INC y DEC, que actúan sobre registros o posiciones de memoria de 8 bits, tienen sus análogas cuando se trata de operar con 16 bits, pero solamente trabajan con cantidades almacenadas en registros dobles, no siendo posible recogerlas de una celda de memoria determinada.

En el caso de las sumas y restas con 8 bits, el primer operando no era necesario especificarlo en la instrucción, pues el microprocesador tomaba siempre por defecto el valor almacenado en el acumulador.

Cuando se trata de sumar o restar números de dos bytes, el primer operando ha de estar contenido en el par HL, salvo en las instrucciones ADD IX, pp y ADD IY, rr donde se toman los registros índice con este fin.

El segundo operando será el valor almacenado en cualquier par de registros de los autorizados en la instrucción en concreto de que se trate (ss, pp o rr), siguiéndose la decodificación expuesta en la tabla, sin que sea posible trabajar con parejas BL o HC.

Tras efectuar la suma o resta, el resultado queda almacenado en HL (en IX o IY si fuera el caso), manteniéndose inalterado el contenido del par de registros utilizado como segundo operando.

Aclaremos su funcionamiento con algunos ejemplos: supongamos que deseamos ejecutar la instrucción ADD HL, DE siendo el valor contenido en HL 52277 y en DE 23758:

ANTES

H 11001100
L 00110101
D 01011100
E 11001110

DESPUÉS

00101001
00000011
01011100
11001110

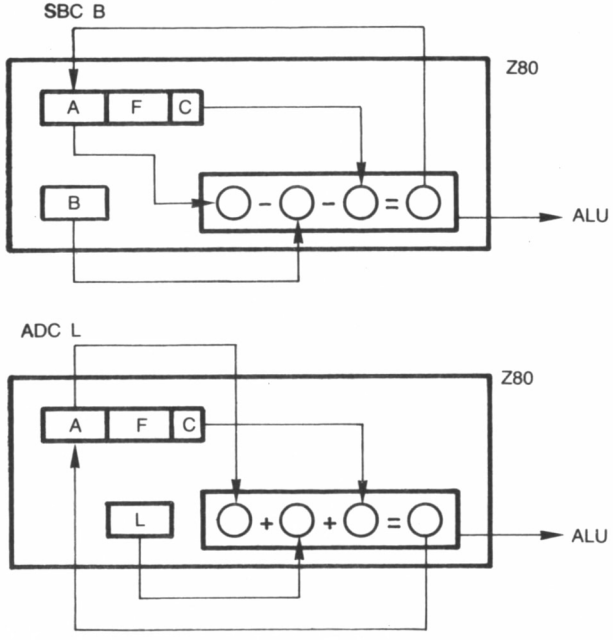
Es decir, tras sumar 52277 y 23758 ¡hemos obtenido como resultado 10499! en vez de 76035 como sería de esperar. Podríamos pensar que el Z80 no tiene ni idea de lo que es una suma. Si nos fijamos en

la tabla donde se detallan los indicadores afectados observaremos que sólo tres de ellos pueden haber cambiado.

La bandera de semiacarreo (H) se alzará en estas operaciones, cuando al ejecutar la suma binaria nos vayamos «llevando una» desde el bit 11 hasta el final. En nuestro ejemplo no ocurre esto y, por tanto, permanecerá a 0. El indicador de resta N contendrá 1 cuando efectuemos, por ejemplo, una instrucción SBC, es decir, una resta. En nuestro caso, la operación implementada fue una suma y contendrá 0, por tanto.

Hasta ahora no hemos conseguido detectar ninguna condición que permita justificar tan sorprendente resultado y sólo nos queda uno de los bits del registro F: el representativo del acarreo. Esta bandera se alza cuando se produce acarreo en el bit 15 (si me «llevo una» a partir del bit 15 al realizar la suma). Y eso precisamente es lo que ha ocurrido en nuestro ejemplo (comprobémoslo). Entonces el bit de acarreo C del registro F contendrá 1.

¿Cómo interpretamos entonces el resultado? Todos sabemos que con 16 bits podemos representar números decimales entre 0 y 65535,



10.10.3. Suma y resta con acarreo.

MNEMONICO	CODIGO MAQUINA	REGISTRO F								N.º BYTES	CICLOS		NOTAS
		7	6	5	4	3	2	1	0		MAQ.	RE-LOJ	
		S	Z		H		P/V	N	C				
DAA	0 0 1 0 0 1 1 1	↕	↕		↕		1	●	↕	1	1	4	
CPL	0 0 1 0 1 1 1 1	●	●		1		●	1	●	1	1	4	
NEG	1 1 1 0 1 1 0 1 0 1 0 0 0 1 0 0	↕	↕		↕		V	1	↕	2	2	4	
CCF	0 0 1 1 1 1 1 1	●	●		X		●	0	↕	1	1	8	
SCF	0 0 1 1 0 1 1 1	●	●		0		●	0	1	1	1	4	
NOP	0 0 0 0 0 0 0 0	●	●		●		●	●	●	1	1	4	
HALT	0 1 1 1 0 1 1 0	●	●		●		●	●	●	1	1	4	

10.10.4. Grupo aritmético de la CPU.

es decir, 65536 combinaciones distintas. Pues 10499 es exactamente la cantidad resultante de efectuar la operación 52277 + 23758 − 65536.

Como el par HL sólo puede contener números de 16 bits, es decir, inferiores a 65535, el 76035 se le hace demasiado grande y por ello le resta 65536, circunstancia que detectamos al comprobar que el banderín de acarreo está alzado.

Las instrucciones de incremento y decremento de 16 bits son completamente análogas en su funcionamiento a sus correspondientes de 8 bits, sólo que ahora actúan sobre registros dobles.

Al ejecutarlas se suma/resta 1 del contenido del byte de menor peso dentro del par especificado en la instrucción. Si esta operación provoca acarreo, éste es transmitido automáticamente al octeto superior.

En los programas en C/M se utilizan para realizar bucles de más de 256 repeticiones, donde puede resultar interesante conocer cuándo hemos llegado, a base de decrementar un valor prefijado, a 0.

Como estas instrucciones no afectan al contenido del registro de indicadores podríamos pensar que la anterior circunstancia no es detectable. Sin embargo, podemos servirnos del siguiente artificio: su-

pongamos que hemos estado decrementando el par BC. Tras ejecutar las instrucciones

```
LD A,B
OR C
```

la bandera de cero Z contendrá un 1, solamente cuando tanto B como C sean 0.

El Grupo Aritmético De La CPU

Pertenecen a él algunas de las instrucciones incluidas en el apéndice bajo la denominación «aritmético y de control de la CPU», las cuales vienen a complementar a las anteriores. Comentemos cuál es exactamente su cometido:

— CPL *(ComPlement,* complementar) efectúa el complemento a 1 del valor almacenado en el acumulador, es decir, cambia los 1 por 0 y los 0 por 1.

— NEG *(NEGative,* negativo) realiza la negación del acumulador, operación que ha de entenderse como encontrar su complemento a 2. En realidad, el microprocesador lo que hace es restar de 0 el contenido de A (0−A), y para ello, como en todas las operaciones de resta, complementa a 2 el sustraendo y suma después el minuendo.

— CCF *(Complement Carry Flag)* complementa el valor actual de la bandera de acarreo C, es decir, si contenía 1 la sitúa a 0 y viceversa.

— SCF *(Set Carry Flag)* alza (1) el indicador de acarreo C.

— DAA *(Decimal Adjustment Accumulator)* se encarga de efectuar el ajuste decimal del acumulador tras una operación aritmética de dos números en formato BCD.

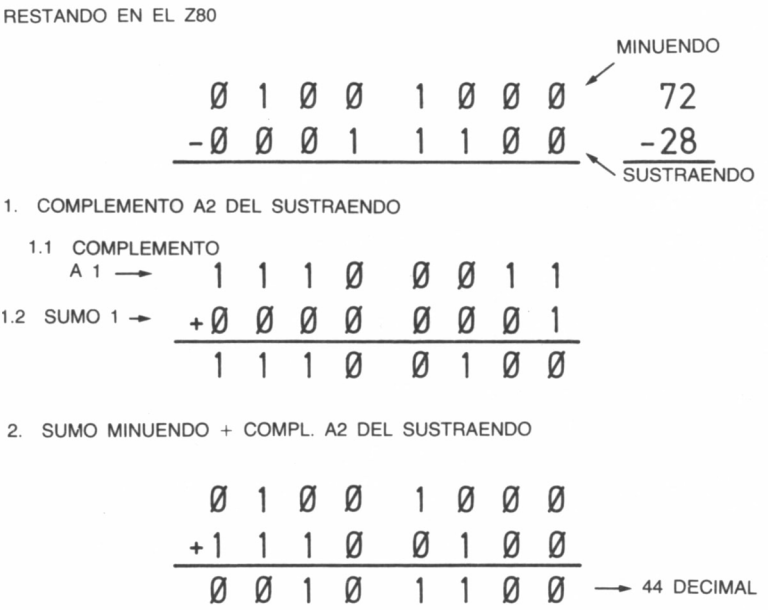
El BCD es un sistema de codificación encargado de representar cada dígito correspondiente a un número decimal, mediante cuatro dígitos binarios. Por ejemplo, el número decimal 76 tendría el siguiente aspecto en BCD: 01110110. Si quisiéramos representar un número mayor, por ejemplo, 6592 necesitaríamos dos octetos: 01000101 y 10010010.

Al realizar una operación aritmética con números en este formato, lo normal es que el resultado recogido en el acumulador no esté en BCD y para transformarlo a éste se utiliza la instrucción DAA, de manera que no se produzcan errores de interpretación entre cantidades en binario y BCD.

A la anterior podemos añadir dos instrucciones, las cuales afectan directamente a la CPU. Se trata de NOP *(No OPeration)* la cual simplemente indica al Z80 que no haga nada. Esto no quiere decir que ejecutar esta instrucción no tome tiempo, pues de hecho se requieren cuatro ciclos de reloj para llevarla a cabo. Su utilidad es reservar espacio dentro de un programa al objeto de añadir luego otras instrucciones, o emplearlo como zona de trabajo y almacenamiento temporal de informaciones.

Finalmente, la orden HALT provoca que el microprocesador pare cualquier trabajo que esté efectuando.

Con esto cerramos el análisis de los grupos aritmético y lógico del Z80 emprendido en el capítulo anterior. En el próximo discutiremos las instrucciones que provocan bifurcaciones dentro de la secuencia de un programa.



10.10.5. Restando en el Z-80.

EJERCICIOS

De todas las instrucciones aritméticas, probablemente las más utilizadas sean INC y DEC, cuyo funcionamiento, por otra parte de extrema simplicidad, ya fue puesto en práctica en el ejercicio de PUSH y POP, tomando parte en una rutina SWAP.

Además de estas instrucciones, seguramente el punto más importante de este capítulo sea la diferencia de resultado que se produce en sumas y restas según se tenga en cuenta o no el acarreo. El siguiente programa utiliza la misma zona de datos que el del capítulo anterior; en 40000 y 40001 se deben suministrar los bytes a sumar y en 40002 se podrá recoger el resultado.

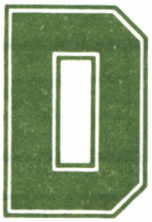
La primera instrucción de la rutina se asegura de fijar adecuadamente el estado de la bandera *carry*. Así, si la primera instrucción (primeros dos caracteres de la DATA en BASIC) es SCF (37 en la DATA) el acarreo quedará a uno; por el contrario, si es XOR A (AF en la DATA), el valor de la bandera C quedará a cero. Puesto que la instrucción empleada para la suma es ADC (suma con acarreo), podremos comprobar el efecto producido por los dos estados de la bandera de *carry*.

```
SCF             ;Fija a 1 la bandera de carry. Si empleamos XOR A
                 en lugar de SCF, el valor de C será 0.
LD A,(40000)    ;Carga el acumulador con el primer dato.
LD B,A          ;Deposita A en B (primer dato).
LD A,(40001)    ;Carga el acumulador con el segundo dato.
ADC A,B         ;Suma A y B teniendo en cuenta el valor de carry.
LD (40002),A    ;Deposita el resultado de la operación en la direc-
                 ción 40002.
RET             ;Retorna al BASIC.
```

Si utilizamos el cargador BASIC...

```
1000 DATA
373A409C473A419C8832429CC900000000000000,50C.
```

BIFURCACIONES



urante los capítulos precedentes han ido apareciendo los diferentes grupos de instrucciones que modificaban el contenido de los indicadores F.

Pues bien, nuestro objetivo durante las próximas líneas será describir las instrucciones que modifican la secuencia de un programa. En ocasiones necesitaremos que ésta se rompa tan sólo cuando determinada condición se cumpla, y lo «averiguaremos» analizando el contenido del bit especificado en el código de operación de la instrucción, dentro del registro de indicadores F.

EL PROGRAMA COUNTER (PC)

Por lo que hasta ahora sabemos, un programa no es otra cosa que una serie de bytes almacenados en la memoria del ordenador, que la CPU interpreta al objeto de llevar a cabo una determinada tarea.

En todo momento, «el cerebro electrónico» debe conocer la direc-

ción a donde dirigirse y recoger la información, pues de otra manera el caos sería inevitable y la ejecución de cualquier rutina por sencilla que fuera sería algo más que misión imposible.

Con la saludable intención de no armarse un fenomenal lío, el Z80 conserva en su registro PC (contador de programa) la situación de la próxima instrucción a procesar y repite cíclicamente los siguientes pasos:

1) Lee el contenido del registro PC.
2) Se posiciona en la dirección de memoria indicada por éste.
3) Recoge allí el byte almacenado y averigua si es necesario leer octetos adicionales (recuerda que el formato de las instrucciones del Z80 varía de 1 a 4 bytes).
4) Le añade al contador de programa PC el número de bytes de la instrucción actual, es decir, antes de procesarla, PC ya está señalando a la siguiente.
5) Finalmente, implementa la instrucción y vuelve al paso número 1.

Evidentemente, siguiendo este sistema los programas sólo podrían ejecutarse secuencialmente, es decir, recorriendo un byte tras otro hasta «chocar» con el final de la memoria. Pero tengamos en cuenta que aunque parezca que el ordenador no está haciendo nada mientras

MNEMONICO	CODIGO MAQUINA
CALL nn	1 1 0 0 1 1 0 1 n n n n n n n n n n n n n n n n
CALL cc, nn	1 1 x cc x 1 0 0 n n n n n n n n n n n n n n n n
RET	1 1 0 0 1 0 0 1
RET cc	1 1 x cc x 0 0 0

está conectado y no ejecutamos programa alguno, en el peor de los casos estará inmerso en alguna rutina de la ROM como, por ejemplo, la de análisis del teclado, en espera de alguna pulsación, repitiéndola una y otra vez.

La clave está, por tanto, en el registro PC, y si pudiéramos modificar a nuestro antojo su contenido, sería posible alterar la secuencia de un programa de la misma manera que en BASIC lo hacemos mediante el comando GOTO.

Pero entre las instrucciones de carga de registros vistas anteriormente, no disponemos de ninguna del tipo LD PC, nn o algo parecido. Entonces, ¿son posibles las bifurcaciones en C/M? La respuesta es sí, y son las instrucciones de salto *(Jump,* en inglés) las encargadas de llevarlas a cabo.

CAMBIOS DE SECUENCIA

Las instrucciones de salto provocan que el control del programa sea transferido a una dirección de memoria especificada en el propio código de operación. La manera de realizar la bifurcación depende

REGISTRO F								N.º BYTES	CICLOS		NOTAS
		5	4	3	2	1	0		MAQ.	RELOJ	
P/V	N		H		P/V	N	C				
•	•	•	•	•	•	•	•	3	5	17	ciclos si CC es falsa
											A/B
•	•	•	•	•	•	•	•	3	3/5	10/17	
•	•	•	•	•	•	•	•	1	3	10	ciclos si CC es verdadera
•	•	•	•	•	•	•	•	1	1/3	5/11	

del tipo de instrucción elegida, es decir, se pueden ejecutar saltos absolutos, relativos o por direccionamiento indirecto.

Tanto los absolutos como los relativos es posible efectuarlos de manera incondicional o tan sólo cuando una determinada condición se verifique (saltos condicionales), mientras que los indirectos siempre son incondicionales.

SALTOS INCONDICIONALES

Cuando la CPU accede a la memoria y lee una instrucción de salto incondicional, tras decodificar el primer octeto, reconoce que se trata de una de este tipo. A continuación, cuando el salto es absoluto (JP nn), escruta los dos bytes siguientes donde encuentra la dirección a la cual saltar. Finalmente, carga en el registro PC este valor y continúa con la ejecución del programa a partir de esta última dirección.

El salto, aunque también incondicional, puede efectuarse de forma relativa, es decir, avanzando o retrocediendo en la memoria un determinado número de posiciones a partir de la señalada por el contador del programa PC.

En estos casos, la instrucción tan sólo precisa de dos bytes, uno para el código de operación y otro para indicar el desplazamiento, el cual será un número en complemento a dos y, por tanto, en el rango de −128 a +127.

El mecanismo seguido por el Z80 al implementarla se describe en la figura, y como podemos comprobar es de capital importancia tener presente que el desplazamiento se añade al PC suponiendo que éste se habría actualizado normalmente, como si la instrucción no hubiera sido de salto.

MNEMONICO	CODIGO MAQUINA								REGIST			
									7	6	5	
									S	Z		
RSTp	1	1	x	t	x	1	1	1	•	•	•	

Dada esta circunstancia, los saltos permitidos al procesar una instrucción JR e, siendo e el desplazamiento, están comprendidos entre −126 y +129 posiciones de memoria, a partir de la ocupada por el primer byte de la instrucción de salto relativo.

Una de las cualidades de las instrucciones de salto relativo es que son reubicables, es decir, actúan de la misma forma sea cual sea la zona de memoria donde hayamos almacenado nuestro programa en C/M, mientras que con las absolutas, generalmente, no ocurre lo mismo y nos acarrearán problemas de adaptación.

Sin embargo, con ellas sólo se pueden efectuar desplazamientos en los márgenes señalados, mientras que los saltos absolutos pueden recorrer toda la memoria a costa, por supuesto de ocupar un byte más.

Finalmente, tres instrucciones JP (HL), JP (IX) y JP (IY) se sirven del contenido de los registros indicados para calcular la posición de memoria a la que debe apuntar el PC. Obviamente en todas ellas el desplazamiento se efectúa de forma absoluta.

LOS SALTOS CONDICIONALES

En el grupo anterior, cuando el microprocesador «tropezaba» en la memoria con una instrucción de salto, fuera cual fuera el estado del registro de indicadores F, SIEMPRE transfería el control del programa a la posición de memoria especificada en la instrucción.

Ahora analizaremos un nuevo tipo de instrucciones, las cuales ejecutan o no el salto (absoluta o relativamente, como antes) basándose en el estado de alguna de las banderas presentes en el registro F.

Para los indicadores de signo S, cero Z, paridad/desbordamiento P/V y arrastre C es posible plantear la instrucción de salto absoluto para que en función de su contenido el programa bifurque o no.

F				N.º BYTES	CICLOS		NOTAS
3	2	1	0		MAQ.	RELOJ	
	P/V	N	C				
•	•	•	•	1	3	11	

JP NZ, nn: producirá salto a la dirección nn cuando el indicador de cero esté a 0.

JP Z, nn: salta si la bandera de cero está alzada. Aparentemente existe un contrasentido entre estas dos instrucciones, pero recordemos que el indicador de cero se sitúa a 1 cuando el resultado de la operación previa fue 0.

JP NC, nn: bifurca en el caso de comprobar que la bandera de acarreo está bajada.

JP C, nn: es la condición contraria a la anterior.

JP PO, nn: el salto es efectivo si en la última operación que afectó al indicador P/V se produjo paridad impar o no hubo sobrepasamiento.

JP PE, nn: salta cuando se dan las condiciones inversas a las anteriores.

JP P, nn: tras escrutar el indicador de signo S, si éste contiene un 1 se produce el salto.

JP M, nn: es la opuesta de la anterior.

Los saltos condicionales relativos solamente pueden efectuarse dependiendo del estado de dos de los indicadores: el de acarreo y el de cero. El formato de estas instrucciones es el siguiente:

JR C,e JR NC,e JR Z,e JR NZ,e

t	p	LLAMADA A LA RUTINA
000	00 H	INICIALIZACION DEL SISTEMA
001	08 H	GESTION ERRORES
010	10 H	IMPRESION DE UN CARACTER
011	18 H	RECOGIDA DE UN CARACTER
100	20 H	RECOGIDA DEL SIGUIENTE CARACTER
101	28 H	ENTRADA AL CALCULADOR
110	30 H	CREACION DE ESPACIO LIBRE
111	38 H	EXAMEN DEL TECLADO

donde como antes, se representa el byte de desplazamiento a sumar al PC, caso de cumplirse la condición fijada.

Para terminar entre las instrucciones de salto existe una de funcionamiento especial (DJNZ,e) la cual utiliza el registro B como condición. Cuando el Z80 la encuentra decrementa en 1 el contenido de B, comprueba si éste es 0 y de no serlo efectúa el salto relativo.

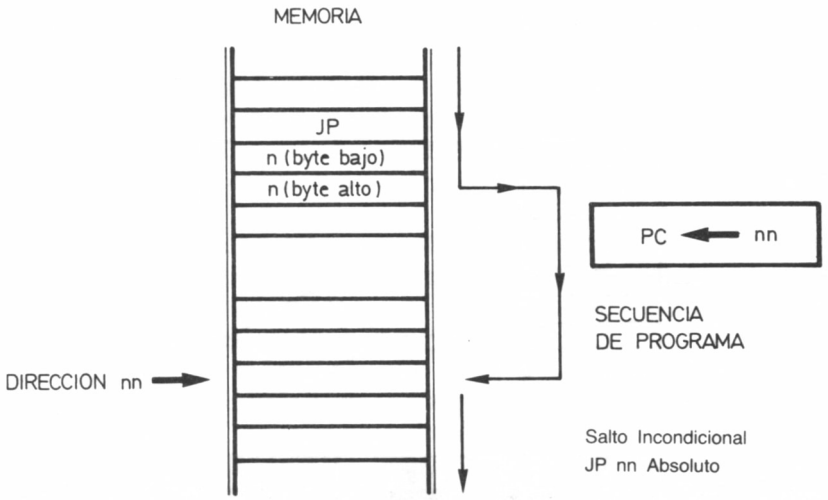
En la figura podemos comprobar como su manejo es una buena técnica para efectuar repetidas pasadas por un grupo de instrucciones, durante el número de veces con que hayamos cargado el registro B.

LLAMADAS Y RETORNOS

Las llamadas (CALL) son instrucciones que modifican la ejecución secuencial de un programa, pero a diferencia de los saltos toman la precaución de anotar en el *stack* la dirección de memoria de la siguiente instrucción almacenada tras ella.

Como podemos comprobar en la tabla correspondiente del apéndice destinado a la presentación del conjunto de ellas (Saltos y Llamadas) es factible ejecutarla incondicionalmente o sujeta a alguna restricción de las comentadas anteriormente.

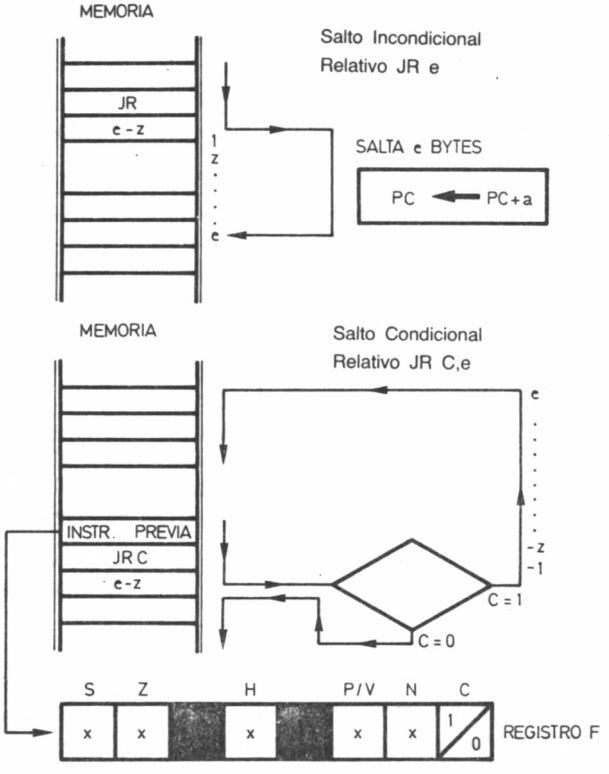
Su funcionamiento es parecido a lo que en BASIC conseguimos mediante la sentencia GOSUB, es decir, cuando el Z80 la encuentra,

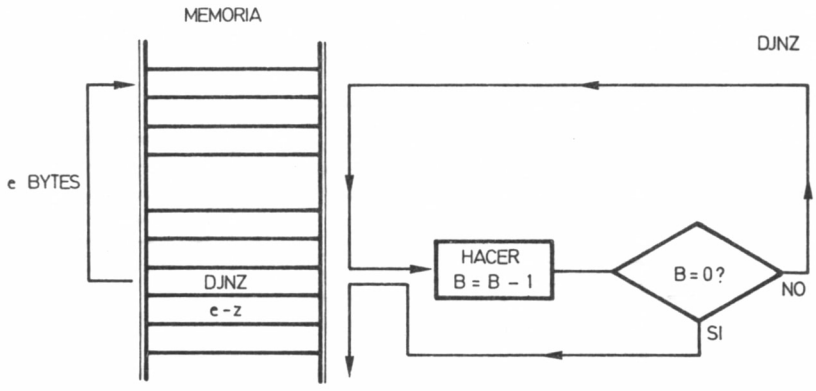


carga en el registro PC el contenido de los dos bytes siguientes determinando de esta manera la posición de memoria a donde transferir el control del programa, al objeto de ejecutar allí una determinada subrutina.

Pero antes de comenzar a procesar las instrucciones almacenadas a partir de dicha dirección, anota en el *stack* la posición de la instrucción siguiente a la llamada CALL. Tras ello, la subrutina es procesada hasta que aparece una instrucción de retorno incondicional RET o una condicionada RET cc, devolviéndose entonces el control a la rutina principal, si es que se da la situación cc estipulada (mecanismo similar al RETURN del BASIC).

En estas circunstancias el microprocesador carga el PC con los dos bytes de lo alto del *stack* y regresa a la dirección especificada por éstos, los cuales, siempre que no manipulemos indebidamente la pila, han de ser los correspondientes a la siguiente posición de memoria tras la instrucción CALL.





Todas las instrucciones de llamada precisan de tres bytes para ser implementadas. Sin embargo, existe otro tipo de llamadas incondicionales de un byte, cuyo mnemónico es RST denominadas instrucciones de **ReSTart.**

Estas transfieren el control a la rutina almacenada a partir de la dirección especificada en su código de operación. En el caso del Amstrad, éstas se corresponden con ciertas subrutinas de interés incluidas en la ROM, cuyo cometido podemos encontrarlo en la tabla adjunta.

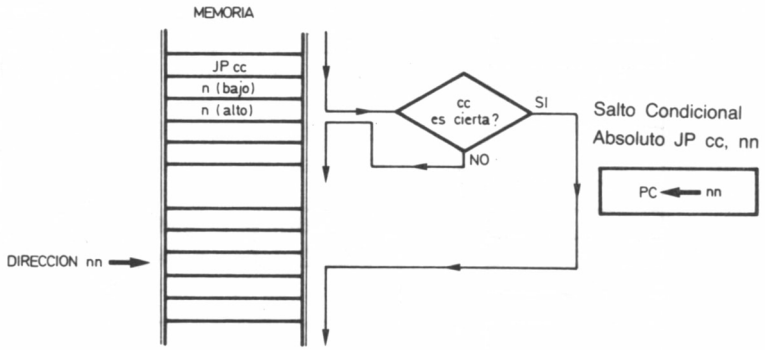
Entre las posibilidades de las instrucciones de llamada se encuentra el aprovechar las utilidades programadas en la ROM del sistema, ahorrándose de esta manera la construcción de muchas subrutinas que de otra manera sería imprescindible diseñar.

EJERCICIOS

Son muchísimas las rutinas en las que se pueden hacer uso de algún tipo de bifurcación, y se convierten en prácticamente imprescindibles en cuanto el programa tiene una cierta entidad dado que el lenguaje máquina carece casi totalmente (salvo DJNZ) de estructuras de tipo bucle.

Servirá como ejemplo un scroll horizontal completo, cuya zona principal ya fue citada para ilustrar el movimiento de bloques. La rutina efectuará un desplazamiento de un carácter hacia la izquierda en las pantallas en modo 2.

```
LD A,25          ;Carga A con el número de filas a
                  desplazar.
LD HL,#C001      ;Primera dirección de la pantalla a
                  desplazar.
```




```
BUCLE 2  PUSH AF          ;Guarda A en la pila.
         LD B,8           ;Número de líneas por fila.
BUCLE 1  PUSH BC          ;Guarda B en la pila.
         PUSH HL          ;Guarda HL en la pila.
         LD D,H           ;
         LD E,L           ;
         DEC E            ;Hace DE=HL-1 para conseguir un
                           scroll hacia la izquierda.
         LD BC,79         ;Número de bytes por fila menos
                           uno.
         LDIR             ;Efectúa el desplazamiento.
         XOR A            ;Borra el acumulador.
         LD (DE),A        ;Deja vacía la última columna.
         POP HL           ;Recupera HL.
         LD A,8           ;Carga A con 8 para sumárselo al
                           byte alto de HL,
         ADD A,H          ;lo cual equivale a sumar 2048
         LD H,A           ;para que HL señale a la próxima fila
                           a transferir (2048 bytes más ade-
                           lante).
         POP BC           ;Recupera el número de bytes por
                           fila.
         DJNZ BUCLE 1     ;Ejecuta el proceso a partir de BU-
                           CLE 1 «B» veces.
         LD DE,#C050      ;Prepara el salto de una línea.
         ADD HL,DE        ;HL apunta una línea más adelante.
         POP AF           ;Recupera el número de filas a trans-
                           ferir.
         DEC A            ;Decrementa el número de filas a
                           transferir.
```

```
JR NZ BUCLE 2   ;Salta a BUCLE 2 si todavía queda
                 alguna fila por desplazar.
RET             ;En caso contrario, vuelve al BASIC.
```

Si utilizamos el cargador BASIC...

```
1000 DATA
3E192101C0F50608C5E5545D1D014F00EDB0AD12,762
1010 DATA
E13E088467C1150C019F13D20E1C9000000,741
```

MNEMONICO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	Nº BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
JP nn	1 1 0 0 0 0 1 1 n n n n n n n n n n n n n n n n	•	•	•	•	•	•	•	•	3	3	10	
JP cc, nn	1 1 x cc x 0 1 0 n n n n n n n n n n n n n n n n	•	•	•	•	•	•	•	•	3	3	10	
JR e	0 0 0 1 1 0 0 0 x e z x	•	•	•	•	•	•	•	•	2	3	12	
JR C, e	0 0 1 1 1 0 0 0 x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12	
JR NC, e	0 0 1 1 0 0 0 0 x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12	
JR Z, e	0 0 1 0 1 0 0 0 x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12	
JR NZ, e	0 0 1 0 0 0 0 0 x e z x	•	•	•	•	•	•	•	•	2	2/3	7/12	
JP (HL)	1 1 1 0 1 0 0 1	•	•	•	•	•	•	•	•	1	1	4	
JP (IX)	1 1 0 1 1 1 0 1 1 1 1 0 1 0 0 1	•	•	•	•	•	•	•	•	2	2	8	
JP (IY)	1 1 1 1 1 1 0 1 1 1 1 0 1 0 0 1	•	•	•	•	•	•	•	•	2	2	8	
DJNZ, e	0 0 0 1 0 0 0 0 x e z x	•	•	•	•	•	•	•	•	2	2/3	8/13	

CONDICION	CC
NZ No cero	000
z Cero	001
NC No arratre	010
c Arrastre	011
PO Pondad impar	100
PE Pondad par	101
p signo positivo	110
n signo negativo	111

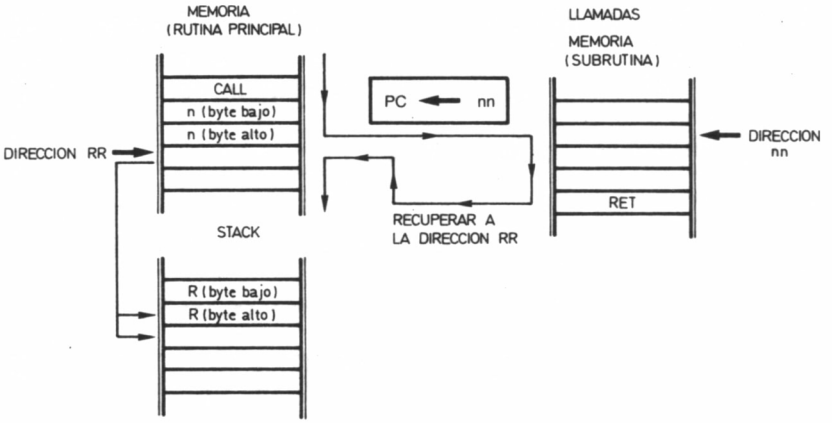
• Indicador no afectado

ciclos si no se ha alcanzado la condición

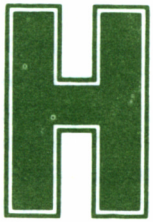
A/B

ciclos si se ha alcanzado la condición

e-z proporciona la dirección efectiva de Pc+e, puesto que, PC es incrementado en 2 antes de la suma del desplazamiento.


MEMORIA
(RUTINA PRINCIPAL)
CALL
n (byte bajo)
n (byte alto)
DIRECCION RR
STACK
R (byte bajo)
R (byte alto)
PC ← nn
RECUPERAR A
LA DIRECCION RR
LLAMADAS
MEMORIA
(SUBRUTINA)
RET
DIRECCION
nn

MANIPULANDO BITS



asta el presente capítulo, todas las instrucciones discutidas mantenían el objetivo común de efectuar una determinada operación sobre un octeto o grupo de ellos, pero no podíamos actuar sobre un bit concreto, el cual, por cualquier razón, fuera preciso escrutar o modificar en contenido. Precisamente, una de las mejoras que el microprocesador Z80 incorpora frente a su predecesor el 8080, la constituye el numeroso grupo de instrucciones encargadas de gestionar información a nivel de bit individual (312 nuevas instrucciones).

BIT, SET Y RESET

En el apéndice bajo la denominación «Manipulación de bits» están recogidas todas las instrucciones encargadas de examinar (BIT) el contenido del bit señalado dentro de un registro o posición de memoria determinada, situarlo a 1 (SET) fuera cual fuera su estado, o fijarlo a 0 (RESET).

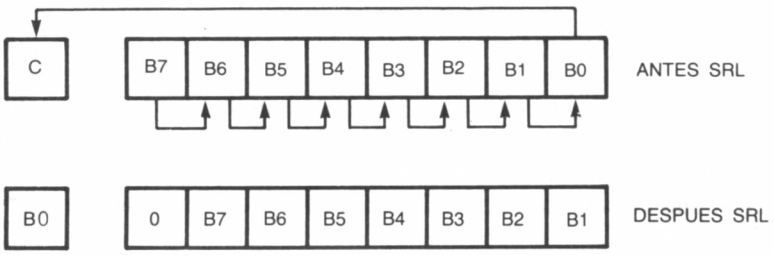
Los mnemónicos asociados con estas operaciones son BIT b,s SET b,s y RES b,s, respectivamente, donde s es el contenido del registro o posición de memoria (8 bits) a la que acceder y b, el lugar ocupado por el bit individual, sobre el cual vamos a actuar.

Recordemos el convenio seguido en toda la literatura informática, según el cual dentro de un octeto los bits que lo constituyen van numerados de 0 a 7 comenzando por el de la derecha.

Supongamos, por ejemplo, que el acumulador contiene en el momento previo a efectuar una de estas instrucciones 10110011. Comprobemos el efecto que sobre el valor almacenado en A obran la ejecución sucesiva de algunas de estas órdenes:

INSTRUCCION	ACUMULADOR
SET 2,A	10110111
SET 4,A	10110111
RES 7,A	00110111
RES 6,A	00110111

Fácilmente se comprueba que tanto las instrucciones SET como las de RESET son independientes del valor original del bit afectado, es decir, fijan su contenido a 1(9SET) o a 0 (SET) sin atender a su estado anterior.



MNE-MONICO	CODIGO MAQUINA	REGISTRO F								N.º BYTES	CICLOS		NOTAS
		7	6	5	4	3	2	1	0		MAQ.	RELOJ	
		S	Z		H		P/V	N	C				
BIT b,r	1 1 0 0 1 0 r r												
	0 1 x b x x r r	x	↕		1		x	0	●	2	2	8	
BIT b, (IX+d)	1 1 0 1 1 1 0 1												
	1 1 0 0 1 0 1 1												
	x x x d x x x x												
	0 1 x b x 1 1 0	x	↕		1		x	0	●	4	5	20	
BIT b, (IY+d)	1 1 1 1 1 1 0 1												
	1 1 0 0 1 0 1 1												
	x x x d x x x x												
	0 1 x b x 1 1 0	x	↕		1		x	0	●	4	5	20	
SET b,s													
	1 1 x x x x x x	●	●	●	●	●	●	●	●				
RES b,s													
	1 0 x x x x x x	●	●	●	●	●	●	●	●				

	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111
(HL)	110

S
r
(IX+d)
(IY+d)

NOTA: Las instrucciones SET y RES siguen idéntica decodificación a BIT, pero sustituyendo los bits B7 y B6 por las parejas 11 y 10, respectivamente.

Otra circunstancia a tener en cuenta es que implementarlas no provoca cambio en el registro de indicadores F. Sin embargo, diferente es lo que ocurre cuando nuestro objetivo es escrutar (BIT) un bit específico.

Supongamos ahora que el acumulador mantiene el valor obtenido tras la instrucción RES 6,A. Efectuemos algunas opciones de BIT sobre algunos bits del byte almacenado en A:

INSTRUCCION

BIT 0,A
BIT 6,A

ACUMULADOR

00110111
00110111

INDICADOR
DE CERO

0
1

La primera consecuencia que extraemos es que tras una instrucción BIT, el contenido del registro o posicion de memoria implicada no varía. En nuestro ejemplo el acumulador no ha experimentado cambio alguno.

La segunda es que el resultado se pone de manifiesto en función de la bandera de cero del registro F, es decir, si el bit es 0, el indicador Z contendrá 1 y 0, en caso contrario.

Estas circunstancias son en particular muy útiles cuando tratemos de determinar el valor de un bit concreto sin necesidad de modificar el contenido de A. Por ejemplo, las dos parejas de instrucciones siguientes conducen al mismo objetivo: determinar si el bit 6 del byte almacenado en el registro B es 1.

```
LD A, B
AND 40h

:    LD A,B
:    BIT 6,A
```

La primera secuencia se vale del byte 40 como máscara para detectar si efectivamente el bit considerado es 1, modificándose para ello el contenido de A. La segunda realiza el mismo trabajo, pero A no se ve alterado. Ambas precisan 5 bytes para ser implementadas, y según las circunstancias nos decidiremos por una u otra técnica.

ROTACIONES

En el apéndice están recogidas todas las instrucciones de rotación y desplazamiento. Centrándonos en las de rotación, las cuatro primeras líneas contienen los códigos asociados con estas operaciones.

El primer grupo de instrucciones es el de rotaciones circulares hacia la izquierda (RLC, *Rotate Left Circular).* Su mecanismo queda descrito en el diagrama de la figura, es decir, todos los bits del octeto considerado giran una posición hacia la izquierda, y el contenido del último se duplica en el lugar dejado vacante por primero y en el indicador de arrastre C.

La rotación derecha circular (RRC, *Rotate Right Circular)* es, en esencia, idéntica a la anterior, pero ahora el giro se produce en sentido contrario, es decir, hacia la derecha y como el bit sobrante sería el B0, éste pasa a ocupar el lugar B7 y el de la bandera de arrastre.

MNE-MONICO	CODIGO MAQUINA	REGISTRO F 7 6 5 4 3 2 1 0								N.º BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
		S	Z		H		P/V	N	C				
RLCA	0 0 0 0 0 1 1 1	●	●		0		●	0	↕	1	1	4	
RLA	0 0 0 1 0 1 1 1	●	●		0		●	0	↕	1	1	4	
RRCA	0 0 0 0 1 1 1 1	●	●		0		●	0	↕	1	1	4	
RRA	0 0 0 1 1 1 1 1	●	●		0		P	0	↕	1	1	4	
RLCr	1 1 0 0 1 0 1 1 0 0 0 0 0 x r x	↕	↕		0		P	0	↕	2	2	8	
RLC(IX+d)	1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 x d x 0 0 0 0 0 1 1 0	↕	↕		0		P	0	↕	4	6	23	
RLC(IY+d)	1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 x d x 0 0 0 0 0 0 1 1	↕	↕		0		P	0	↕	4	6	23	
RLS	0 1 0	↕	↕		0		P	0	↕				
RRCS	0 0 1	↕	↕		0		P	0	↕				
RRs	0 1 1	↕	↕		0		P	0	↕				
SLAs	1 0 0	↕	↕		0		P	0	↕				
SRAs	1 0 1	↕	↕		0		P	0	↕				
SRLs	1 1 1	↕	↕		0		P	0	↕				
RLD	1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1	↕	↕		0		P	0	●	2	5	18	
RRD		↕	↕		0		P	0	●	2	5	18	

	R
B	000
C	001
D	010
E	011
H	100
L	101
A	111
(HL)	110

S
r
(IX+d)
(IY+d)

NOTA: Las instrucciones donde en su código máquina sólo se dan tres bits siguen idéntica decodificación que los anteriores.

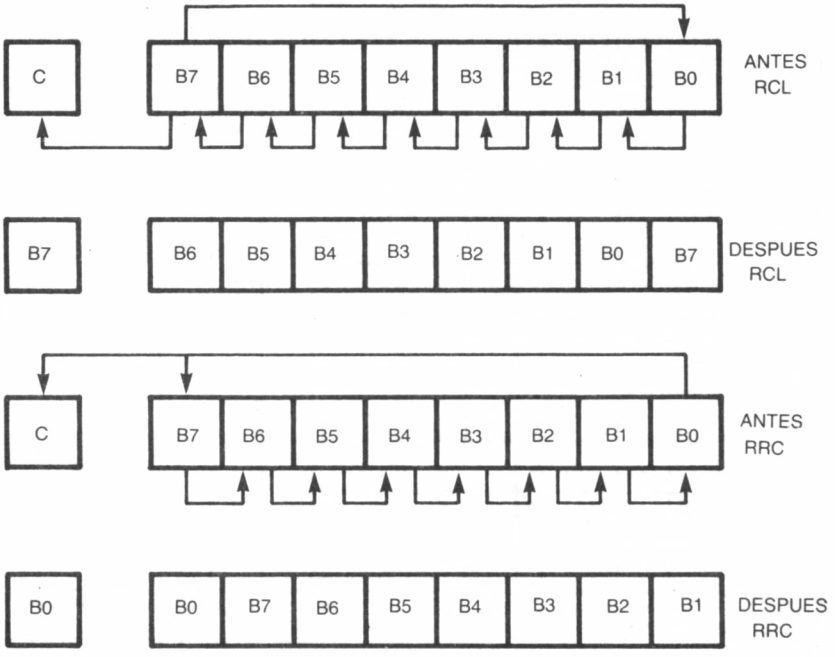
Para averiguar, por ejemplo, el bit de mayor orden que no es cero dentro de un determinado octeto, se emplean con mucha frecuencia las instrucciones de rotación izquierda (RL, *Rotate Left)*. Se trata de un giro completo de los bits del octeto especificado al que añadimos un noveno bit: el indicador de acarreo.

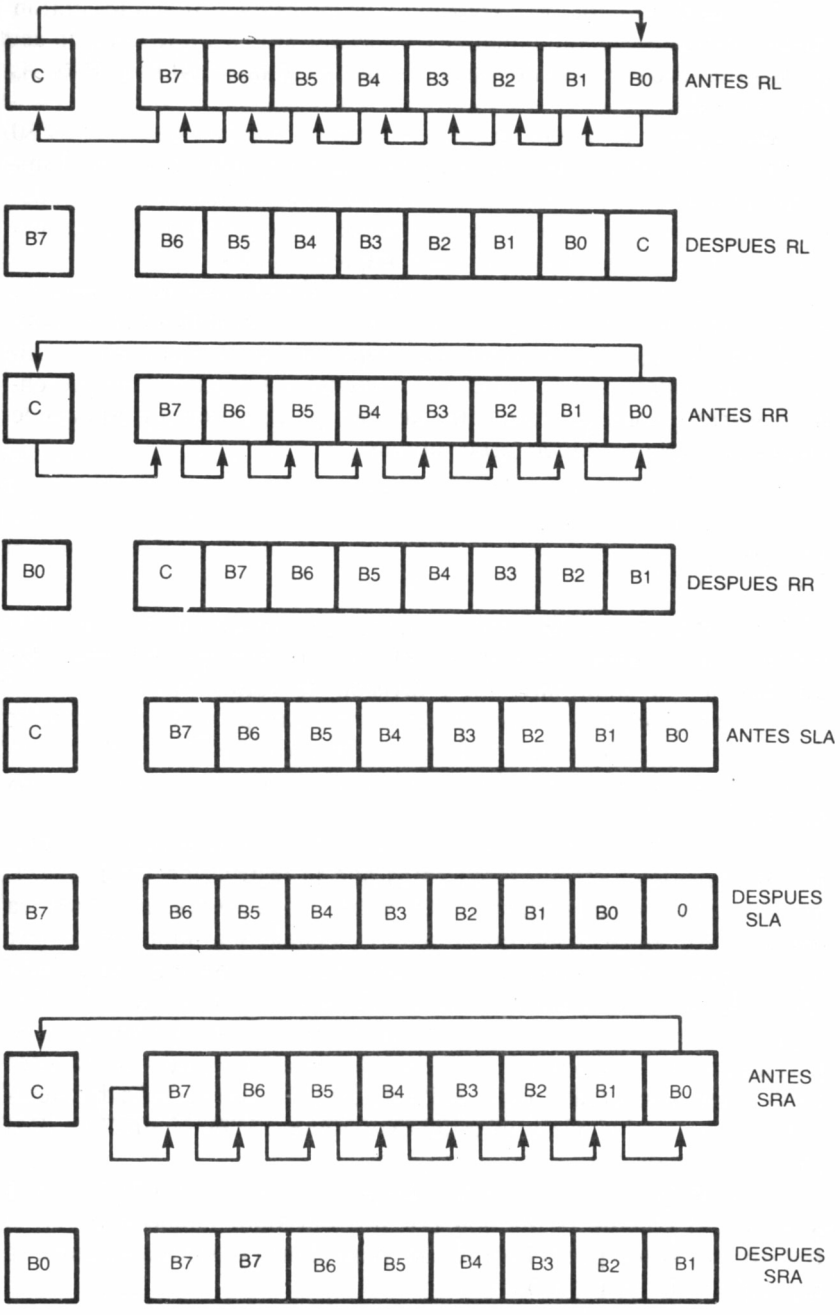
Puesto que B7 es transferido al indicador C cada vez que implementamos una instrucción de rotación izquierda, bastará escrutar el contenido de éste, hasta que finalmente almacene un 1.

La rotación derecha (RR, *Rotate Right)* sigue el mismo principio de efectuar el giro de los nueve bits obtenidos al añadir el bit C al octeto sobre el cual operamos, aunque por supuesto el desplazamiento se realiza en sentido contrario a las agujas del reloj.

Su aplicación resulta inmediata en programas en los que sólo se necesite acceder a determinadas subrutinas dentro de un bloque de éstas, y en un orden prefijado, de tal forma que no siempre se ejecuten todas, sino tan sólo las que necesitemos.

En el diagrama hemos representado un ejemplo de cómo se implementaría esta técnica manejando una instrucción de rotación derecha. Observaremos que es posible fijar 256 combinaciones diferentes según nuestras necesidades.




C
B7
B6
B5
B4
B3
B2
B1
B0
ANTES RL
B7
B6
B5
B4
B3
B2
B1
B0
C
DESPUES RL
C
B7
B6
B5
B4
B3
B2
B1
B0
ANTES RR
B0
C
B7
B6
B5
B4
B3
B2
B1
DESPUES RR
C
B7
B6
B5
B4
B3
B2
B1
B0
ANTES SLA
B7
B6
B5
B4
B3
B2
B1
B0
0
DESPUES SLA
C
B7
B6
B5
B4
B3
B2
B1
B0
ANTES SRA
B0
B7
B7
B6
B5
B4
B3
B2
B1
DESPUES SRA

Al final de la tabla de «Rotación y desplazamiento» se encuentran cuatro instrucciones redundantes con las anteriores y que tan sólo actúan sobre el contenido del acumulador. Se trata de RLCA, RRCA, RLA y RRA.

Aun a pesar de poder resultar repetitivas, los fabricantes del Z80 prefirieron mantenerlas para conservar la compatibilidad con el microprocesador 8080 y bajo determinadas condiciones nosotros podemos emplearlas con cierta ventaja sobre sus equivalentes, pues tan sólo precisan de un byte para ser implementadas.

Estas cuatro instrucciones sólo afectan al indicador de arrastre C, a diferencia de todas las anteriores que además modifican el contenido de los de signo, paridad/sobrepasamiento y cero. Por tanto, cuando tras una operación de rotación solamente sea preciso tener presente el acarreo, conseguiremos ahorrar tiempo de proceso y espacio de memoria implementando estas últimas.

DESPLAZAMIENTOS

Las instrucciones de desplazamiento constituyen un arma de inestimable valor cuando se trata de efectuar rutinas que resuelvan las operaciones de multiplicación y división de varios bytes.

En su mecanismo interviene de manera protagonista el indicador de arrastre C, aunque tras procesarlas, las banderas Z, S y P/V serán afectadas según el resultado obtenido, al igual que ocurría con las instrucciones de rotación.

En los gráficos correspondientes se representa de forma esquemática su funcionamiento. Observemos que la instrucción de desplazamiento aritmético a la izquierda SLA *(Shift Left Arithmetic),* provoca que el valor almacenado en el octeto afectado sea multiplicado por dos, proporcionando una señal de sobrepasamiento (se activa el indicador de arrastre) cuando el resultado sea mayor o igual de 256.

Por el contrario, la instrucción SRA *(Shift Right Arithmetic,* desplazamiento aritmético a la derecha) divide el contenido del registro dado entre dos, situándose el resto de esta división (el microprocesador supone que los números están expresados en complemento a dos) en el indicador C.

Finalmente, SRL *(Shift Right Logic,* desplazamiento lógico hacia la derecha) efectúa la división por dos del valor almacenado en el registro implicado por la instrucción, pero ahora considerando que está expresado en formato binario.

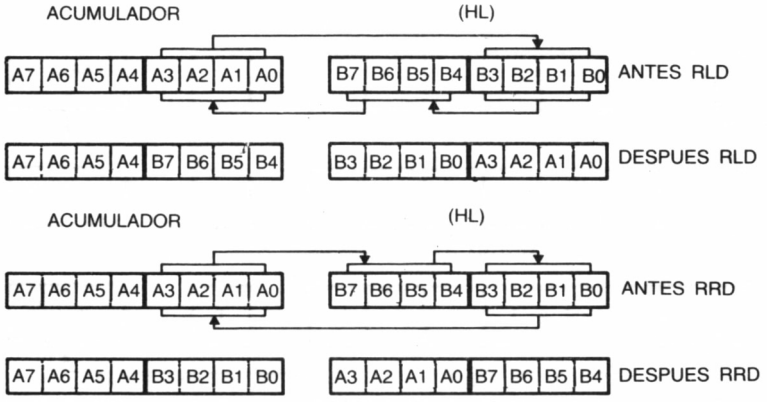
Resulta útil considerar el uso conjunto de las instrucciones de rotación y las de desplazamiento al objeto de conseguir trasladar a la izquierda o derecha un grupo de varios bytes. Por ejemplo, la siguiente secuencia desplaza a la izquierda bit a bit el contenido del par HL, colocando en el lugar vacante un 0, lo cual equivale a la multiplicación por 2 del valor almacenado en dicho par:

```
SLA  L
RL   H
```

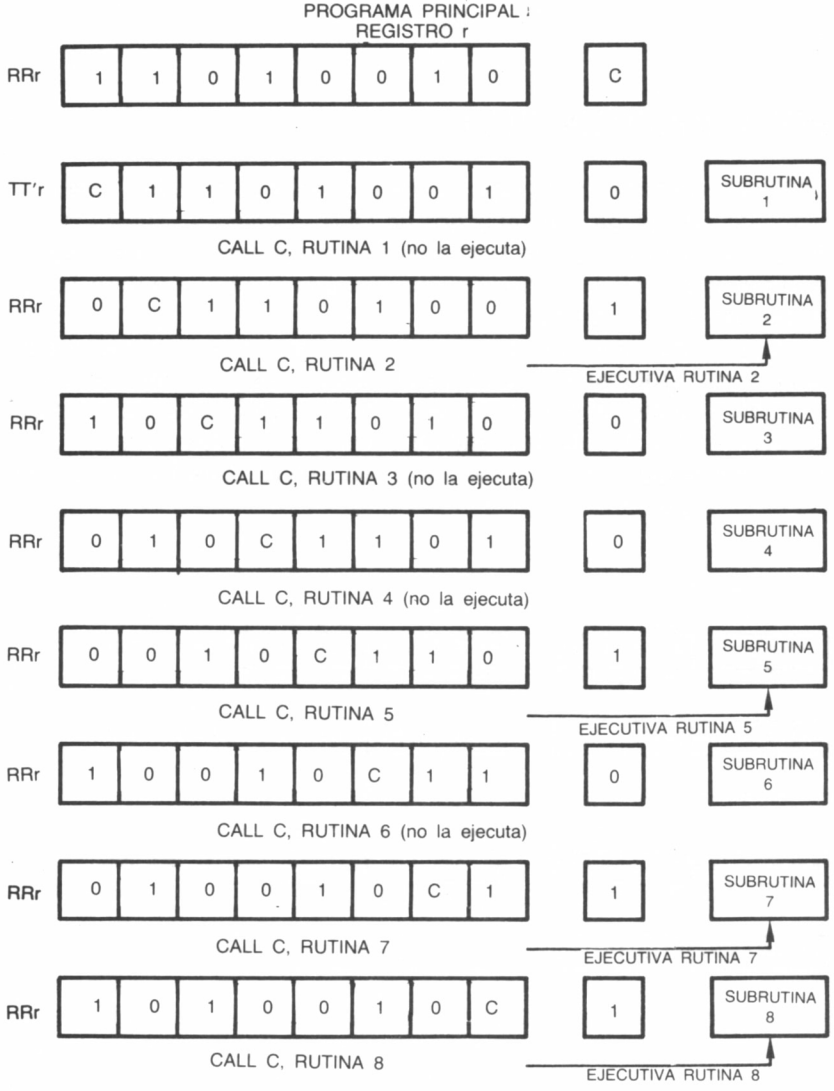
ROTACIONES EN BCD

Para terminar con el grupo de instrucciones de rotación y desplazamiento nos queda comentar dos de ellas, las cuales sirviéndose de direccionamiento indirecto (efectúan su trabajo sobre el octeto señalado por el par HL), ejecutan rotaciones a la izquierda o derecha de grupos de 4 bits *(nibbles)*.

Al igual que antes, para no perder la información sobrante al ejecutar el giro (rotábamos bit a bit y el saliente pasaba al indicador C), el nibble saliente, ahora se recoge en los cuatro bits de menor peso del acumulador A, mientras que los previamente contenidos allí quedan almacenados en el nibble vacío del octeto considerado. En la figura se representa el proceso seguido.



Por estas razones, las instrucciones RLD *(Rotate Left Decimal,* rotación decimal hacia la izquierda) y RRD *(Rotate Right Decimal,* rotación decimal hacia la derecha) son de especial utilidad cuando estemos manejando números expresados en formato binario codificado en decimal (BCD), pues efectúan una rotación entre los dígitos decimales almacenados en la posición de memoria indicada por HL y el acumulador.



EJERCICIOS

Al contrario de lo que sucedía con el tema expuesto en el capítulo anterior, las instrucciones de manipulación de bits se emplean en programas relativamente especializados, que se escapan del propósito de este libro. No obstante, propondremos un brevísimo ejemplo, gracias a la instrucción SLA, que nos permitirá multiplicar por dos el byte proporcionado como dato en la dirección 40000; la recogida del resultado, para comprobar la correcta ejecución de la rutina, se conseguirá en la dirección 40001. Así, por ejemplo, supuesto 30000 sea la dirección de carga del programa, si queremos obtener el resultado de duplicar 65, ejecutaremos: POKE 40000,65: CALL 30000: PRINT PEEK (40001).

```
LD A,(40000) ;Carga el acumulador con el dato.
SLA A        ;Multiplica por dos el valor del acumulador.
LD (40001),A ;Deposita A en la dirección 40001.
RET          ;Retorna al BASIC.
```

Si utilizamos el cargador BASIC...

```
1000DATA
3A409CCB2732419CC90000000000000000000000,3E0
```

ENTRADA Y SALIDA

El sistema de comunicaciones que un microprocesador debe mantener con los periféricos que le rodean es uno de los más importantes factores considerado por los fabricantes de un ordenador, cuando han de tomar la decisión de incorporarlo a sus equipos.

En el caso de Z80, la interacción con el teclado, el casete, las impresoras, la unidad de disco, etc., se encuentra satisfactoriamente gestionada a través de las rutinas almacenadas en la ROM principal o mediante memorias exteriores (como es el caso del controlador de disco) conectadas con el microprocesador a través de la placa de circuitos integrados del interior del sistema.

Pero no olvidemos que cuando un microprocesador necesita enviar o recoger información destinada o procedente de un dispositivo exterior, lo hace siempre byte a byte a través de los canales habilitados a tal efecto del bus de datos.

Este, como sabemos, está formado por 8 líneas, y a través de cada una de ellas puede circular un bit cada vez, es decir, hablamos de un

bus de datos de 8 bits o un byte, el cual podríamos decir, sin entrar en consideraciones de índole electrónica, que comienza en el microprocesador y termina en el puerto *(port)* habilitado para el periférico.

IN Y OUT

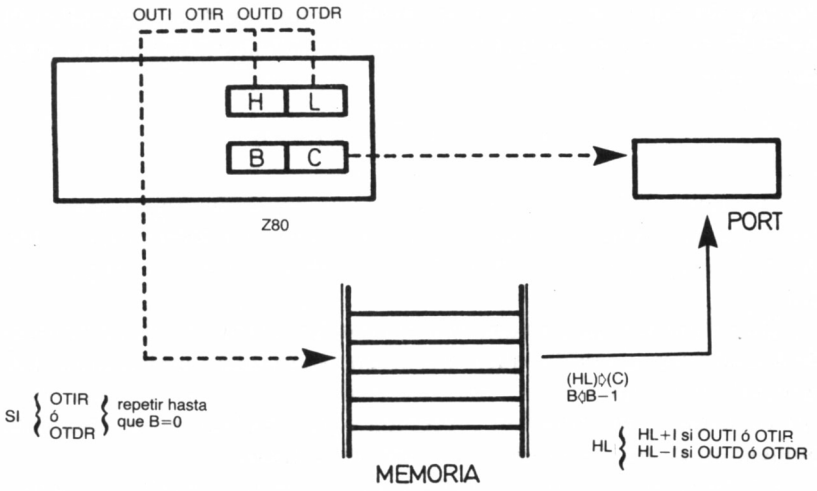
IN y OUT son las dos instrucciones preparadas por los fabricantes del Z80 encargadas de gestionar cualquier transferencia de datos entre microprocesador y periférico. La CPU no necesita saber cómo funciona un dispositivo exterior a ella en detalle, sino tan sólo de dónde (IN) o a dónde (OUT) recoger o enviar la información que con él intercambia.

Este lugar es lo que antes hemos denominado puerto, o *port* en su acepción inglesa, y puesto que el Z80 no le interesa como llega o sale de allí cada byte desde/hacia el periférico, envía o recoge un octeto, de uno en uno, cada vez que sea necesario. Y éste es, precisamente, el trabajo encomendado a las instrucciones IN y OUT. El formato más sencillo de las instrucciones de entrada (IN) de datos hacia el microprocesador es:

IN DESTINO, PORT

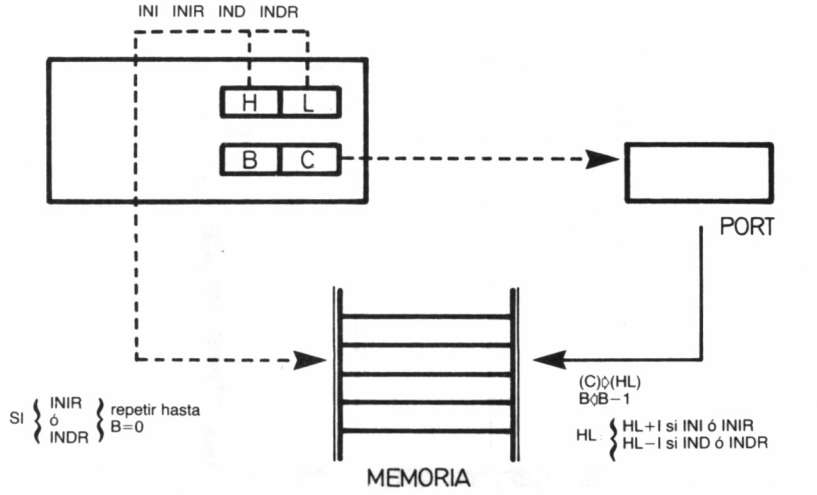
MNEMONICO	CODIGO MAQUINA	REGISTRO F								N.º BYTES	CICLOS		NOTAS
		7	6	5	4	3	2	1	0		MAQ.	RELOJ	
		S	Z		H		P/V	N	C				
DI	1 1 1 1 0 0 1 1	•	•	•	•	•	•	•	•	1	1	4	
EI	1 1 1 1 1 0 1 1	•	•	•	•	•	•	•	•	1	1	4	
IMO	1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0	•	•	•	•	•	•	•	•	2	2	8	
IM1	1 1 1 0 1 1 0 1 0 1 0 1 0 1 1 0	•	•	•	•	•	•	•	•	2	2	8	
IM2	1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0	•	•	•	•	•	•	•	•	2	2	8	
RETI	1 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1	•	•	•	•	•	•	•	•	2	4	14	
RETN	1 1 1 0 1 1 0 1 0 1 0 0 0 1 0 1	•	•	•	•	•	•	•	•	2	4	14	

10.13.1. Instrucciones de control de interrupciones.



donde DESTINO puede ser cualquiera de los registros de 8 bits A, B, C, D, E, H o L y PORT, el puerto del cual nosotros necesitemos recabar la información definido por el contenido del par BC en las instrucciones IN r,(C) y por el par An en IN A,(n).

Respecto de las instrucciones de salida (OUT) podemos efectuar idénticas apreciaciones, pero considerando ahora que la operación es



10.13.2. Mecanismo de las instrucciones de entrada y salida.

inversa, es decir, enviamos a un puerto determinado el byte almacenado en el registro ORIGEN del microprocesador. El formato será, por tanto:

OUT (PORT), ORIGEN

donde, tanto PORT como ORIGEN, quedan definidos exactamente igual que en las instrucciones de entrada pero, por supuesto, el sentido de la transferencia de información es opuesto (del Z80 al periférico).

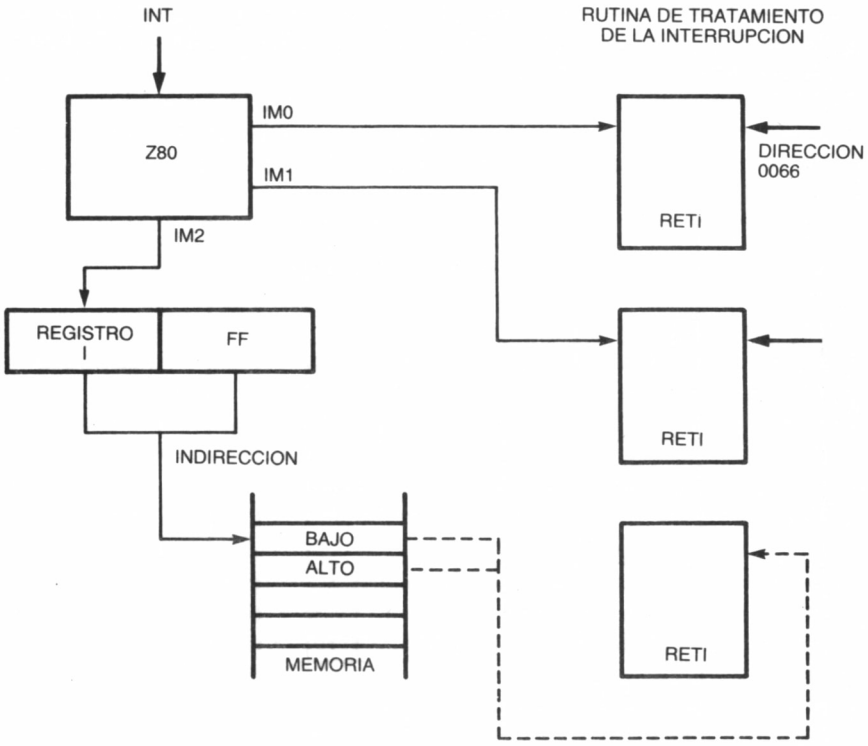
De forma similar a lo que ocurría con las instrucciones de búsqueda y transferencia de bloques, las de entrada/salida pueden funcionar semiautomáticamente o de manera totalmente automática para recoger o enviar la información contenida o destinada a un determinado bloque de memoria desde/a un port específico.

En estos casos el par HL debe señalar al primer octeto del bloque (instrucciones INI, INIR, OUTI y OTIR) o al último (caso de IND, INDR, OUTD y OTDR) de dónde o hacia dónde recoger/enviar la información, puesto que en las primeras se incrementa de uno en uno y en las segundas se decrementa en la misma cantidad cada vez que se realiza una lectura/escritura del port correspondiente.

El registro B es utilizado como contador, y en todas estas instrucciones se decrementa en uno por cada lectura o escritura que se realiza del puerto especificado por el registro C.

MNEMONICO	HEXA-DECIMAL	DECIMAL
DI	F3	243
EI	FB	251
IMNO	ED 46	237,70
IM1	ED 56	237,86
IM2	ED, JE	237,94

10.13.3. Códigos de las instrucciones de interrupción.



10.13.4. Los modos de interrupción.

Si las instrucciones son de mecanismo automático, la transferencia de datos entre ordenador y periférico se ejecuta una y otra vez hasta que el contenido del registro B sea 0. En las figuras se han representado esquemáticamente todos estos procesos.

INTERRUPCIONES

Una interrupción es una señal enviada por un periférico hacia el microprocesador con la intención de que éste abandone cualquier trabajo que estuviera realizando y atienda su solicitud.

Cuando esta circunstancia se produce, la CPU completa la ejecución de la instrucción en curso en ese instante y salta a una dirección

de memoria donde encontrará almacenada la subrutina de tratamiento de la interrupción, anotando previamente en el stack dónde paralizó el programa principal.

Una vez concluida la rutina de interrupción retorna al punto de salto, siempre que al final de ésta hayamos incluido una instrucción de retorno, y continúa implementando el resto del programa. Sobre

MNEMONICO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	N.º BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
INA, (n)	1 1 0 1 1 0 1 1 x n x	•	•	x	•	x	•	•	•	2	3	11	
INr, (c)	1 1 1 0 1 1 0 1 0 1 r r r 0 0 0	↕	↕	x	↕	x	p	0	•	2	3	12	
INI	1 1 1 0 1 1 0 1 1 0 1 0 0 0 1 0	x	↕	x	x	x	x	1	•	2	4	16	
INIR	1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0	x	1	x	x	x	x	1	•	2	5/4	21/16	
IND	1 1 1 0 1 1 0 1 1 0 1 0 1 0 1 0	x	↕	x	x	x	x	1	•	2	4	16	
INDR	1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 0	x	1	x	x	x	x	1	•	2	5/4	21/16	
OUT (n), A	1 1 0 1 0 0 1 1 x n x	•	•	x	•	x	•	•	•	2	3	11	
OUT (c), r	1 1 1 0 1 1 0 1 0 1 r r r 0 0 1	•	•	x	•	x	•	•	•	2	3	12	
OUTI	1 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1	x	↕	x	x	x	x	1	•	2	4	16	
OTIR	1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1	x	1	x	x	x	x	1	•	2	5/4	21/16	
OUTD	1 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1	x	↕	x	x	x	x	1	•	2	4	16	
OTDR	1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1	x	1	x	x	x	x	1	•	2	5/4	21/16	

	r
B	000
C	001
D	010
E	100
H	101
L	111
A	

1 significa que el indicador Z será colocado a 1, si 3-1 es cero; en caso contrario será puesto a 0. Cuando en CICLOS se dan dos cantidades, la primera indica ciclos si B≠0 y la segunda ciclos cuando B=0.

10.13.5. El grupo de entrada/salida.

este particular conviene puntualizar que, en principio, bastaría colocar al final de la subrutina una instrucción de retorno de las ya conocidas, por ejemplo, RET. Pero solamente las de retorno de interrupción (RETI y RETN) garantizan que el microprocesador encontrará restablecidas las condiciones de partida previas, al tratamiento de la solicitud del periférico en cuestión.

MODOS DE INTERRUPCIÓN

En el juego del Z80 existen tres instrucciones IM *(Interruption Mode)* que fijan otros tantos modos de interrupción: IM 0, IM 1 e IM 2.

Cuando un periférico interrumpe en modo 0 coloca en el bus de datos un byte el cual provoca que el microprocesador salte a alguna de las primeras direcciones de la página cero.

El modo 1 es similar al anterior, pero en este caso la dirección de salto es constante (0038h).

Finalmente, mediante la instrucción IM 2, se accede al modo de interrupción 2 el cual, desde el punto de vista de la programación, es el más interesante para nosotros. Cuando una solicitud es efectuada al microprocesador por este sistema, el Z80 conforma una dirección absoluta de salto entre el contenido del registro I (parte alta) y el byte entregado por el periférico (parte baja) al bus de datos.

La dirección así formada (byte colocado en el bus de datos + 256 × contenido de 1) señala a una posición de memoria determinada. Ahora la CPU lee el contenido de ésta y de la siguiente celda, y con estos dos octetos forma la dirección absoluta donde encontrar la subrutina de tratamiento de la interrupción (1 posición + 256 × siguiente), método conocido técnicamente como de «indirección».

Por último, la atención a las interrupciones puede ser deshabilitada mediante DI *(Disable Interrupts)* y habilitada nuevamente con EI *(Enable Interrupts)*.

TABLAS DE COMANDOS

ENTRADA/SALIDA

INA,(n)	DB n
OUT(n),A	D3 n
INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA
OUTI	ED A3
OTIR	ED B3
OUTD	ED AB
OTDR	ED BB

TRANSFERENCIA

DESTINO (DE)	ORIGEN (HL)		INSTRUCCION
	ED A0	LDI	
	ED B0	LDIR	
	ED A8	LDD	
	ED B8	LDDR	

BUSQUEDA

DIRECCION DE BUSQUEDA (HL)	
ED A1	CPI
ED B1	CPIR
ED A9	CPD
ED B9	CPDR

INTERCAMBIO (EX y EXX)

	AF'	BC' DE' HL'	HL	IX	
AF	08				
BC DE HL		D9			
DE			EB		
(SP)			E3	DD E3	FD E3

ENTRADA/SALIDA

	A	B	C	D	E	H	L
IN	ED 78	ED 40	ED 48	ED 50	ED 58	ED 60	ED 68
OUT	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69

ARITMETICO DE 16 BITS

		BC	DE	HL	SP	IX	IY
ADD	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
ADC	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SBC	HL	ED 42	ED 52	ED 62	ED 72		
INC		03	13	23	33	DD 23	FD 23
DEC		0B	1B	2B	3B	DD 2B	FD 2B

MANIPULACION DE BITS

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
8	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
9	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E

RES	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

RETORNO

		CONDICION								
		INCOND.	ARRASTRE	NO ARRASTRE	CERO	NO CERO	PARIDAD PAR	PARIDAD IMPAR	SIGNO NEG.	SIGNO POS.
RET	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0
RETI	(SP) (SP+1)	ED 4D								
RETN	(SP) (SP+1)	ED 45								

SALTOS Y LLAMADAS

		CONDICION									
		INCOND.	ARRASTRE	NO ARRASTRE	CERO	NO CERO	PARIDAD PAR	PARIDAD IMPAR	SIGNO NEG.	SIGNO POS.	B≠0
JP	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JR	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JP	(HL)	E9									
JP	(IX)	DD E9									
JP	(IY)	FD E9									
CALL	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DJNZ	PC+e										10 e-2

DIRECCION DE LLAMADA	00 h	08 h	10 h	18 h	20 h	28 h	30 h	38 h
INSTRUCCION	RST 0	RST 8	RST 10	RST 18	RST 20	RST 28	RST 30	RST 38
CODIGO DE OPERACION	C7	CF	D7	DF	E7	EF	F7	FF

CARGA DE 8 BITS (LD)

DESTINO \ ORIGEN	AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
AF										F1
BC								01 n n	ED 4B n n	C1
DE								11 n n	ED 5B n n	D1
HL								21 n n	2A n n	E1
SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
IX								DD 21 n n	DD 2A n n	DD E1
IY								FD 21 n n	FD 2A n n	FD E1
(n)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
(SP)	F5	C5	D5	E5		DD E5	FD E5			

LOGICO 8 BITS

ORIGEN	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
AND	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
XOR	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
OR	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
CP	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n

CARGA DE 8 BITS (LD)

DESTINO \ ORIGEN	I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+0)	(nn)	n
A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
(HL)			77	70	71	72	73	74	75							36 n
(BC)			02													
(DE)			12													
(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
(nn)			32 n n													
I			ED 47													
R			ED 4F													

ROTACION Y DESPLAZAMIENTO

	ORIGEN Y DESTINO									
	A	B	C	D	E	H	L	(HL)	(IX+d)	(YI+d)
RLC	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR	CB IF	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
SRA	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d	FD CB d
RLD								ED 6F		
RRD								ED 67		
RLCA	07									
RRCA	OF									
RLA	17									
RRA	IF									

ARITMETICO 8 BITS

	ORIGEN										
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADC	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUB	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SBC	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
INC	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DEC	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

ARITMETICO Y DE CONTROL DE LA C.P.U.

DAA	27
CPL	2F
NEG	ED 44
CCF	3F
SCF	37
NOP	00
HALT	76
DI	F3
EI	FB
IM0	ED 46
IM1	ED 56
IM2	ED 5E

LISTA DE COMANDOS

MNEMONICOS Z-80

Dec	Hex.	Bin.	Directo	Por CB	Por ED	Por DD	Por FD	Por DDCB	Por FDCB
0	00	0	nop	rlc b					
1	01	1	ld bc,NN	rlc c					
2	02	10	ld (bc),a	rlc d					
3	03	11	inc bc	rlc e					
4	04	100	inc b	rlc h					
5	05	101	dec b	rlc l					
6	06	110	ld b,N	rlc (hl)				rlc (ix+N)	rlc (iy+N)
7	07	111	rlca	rlc a					
8	08	1000	ex af,af	rrc b					
9	09	1001	add hl,bc	rrc c		add ix,bc	add iy,bc		
10	0A	1010	ld a,(bc)	rrc d					
11	0B	1011	dec bc	rrc e					
12	0C	1100	inc c	rrc h					
13	0D	1101	dec c	rrc l					
14	0E	1110	ld c,N	rrc (hl)				rrc (ix+N)	rrc (iy+N)
15	0F	1111	rrca	rrca					
16	10	10000	djnz DIS	rl b					
17	11	10001	ld de,NN	rl c					
18	12	10010	ld (de),a	rl d					
19	13	10011	inc de	rl e					
20	14	10100	inc d	rl h					
21	15	10101	dec d	rl l					
22	16	10110	ld d,N	rl (hl)				rl (ix+N)	rl (iy+N)
23	17	10111	rla	rl a					
24	18	11000	jr DIS	rr b					
25	19	11001	add hl,de	rr c		add ix, de	add iy,de		

26	1A	11010	ld a,(de)	rr d					
27	1B	11011	dec de	rr e					
28	1C	11100	inc e	rr h					
29	1D	11101	dec e	rr l					
30	1E	11110	ld e,N	rr (hl)				rr (ix+N)	rr (iy+N)
31	1F	11111	rra	rr a					
32	20	100000	jr nz,DIS	sla b					
33	21	100001	ld hl,NN	sla c					
34	22	100010	ld (NN),hl	sla d		ld (NN),ix	ld (NN),iy		
35	23	100011	inc hl	sla e		inc ix	inc iy		
36	24	100100	inc h	sla h					
37	25	100101	dec h	sla l					
38	26	100110	ld h,N	sla (hl)				sla (ix+N)	sla (iy+N)
39	27	100111	daa	sla a					
40	28	101000	jr z,DIS	sra b					
41	29	101001	add hl,hl	sra c		add ix,ix	add iy,iy		
42	2A	101010	ld hl,(NN)	sra d		ld ix,(NN)	ld iy,(NN)		
43	2B	101011	dec hl	sra e		dec ix	dec iy		
44	2C	101100	inc l	sra h					
45	2D	101101	dec l	sra l					
46	2E	101110	ld l,N	sra (hl)				sra (ix+N)	sra (iy+N)
47	2F	101111	cpl	sra a					
48	30	110000	jr nc,DIS						
49	31	110001	ld sp,NN						
50	32	110010	ld (NN),a						
51	33	110011	inc sp						
52	34	110100	inc (hl)			ine (ix+N)	ine (iy+N)		
53	35	110101	dec (hl)			dec (ix+N)	dec (iy+N)		
54	36	110110	ld (hl),N			ld (ix+N),N	ld (iy+N),N		
55	37	110111	scf						
56	38	111000	jr c,DIS	srl b					
57	39	111001	add hl,sp	srl c		add ix,sp	add ix,sp		
58	3A	111010	ld a,(NN)	srl d					
59	3B	111011	dec sp	srl e					
60	3C	111100	inc a	srl h					
61	3D	111101	dec a	srl l					
62	3E	111100	ld a,N	srl (hl)				srl (ix+N)	srl (iy+N)
63	3F	111111	ccf	srl a					
64	40	1000000	ld b,b	bit 0,b	in b,(c)				
65	41	1000001	ld b,c	bit 0,c	out (c),b				
66	42	1000010	ld b,d	bit 0,d	sbc hl,bc				
67	43	1000011	ld b,e	bit 0,e	ld (NN),bc				
68	44	1000100	ld b,h	bit 0,h	neg				
69	45	1000101	ld b,l	bit 0,l	retn				
70	46	1000110	ld b,(hl)	bit 0,(hl)	im 0	ld b, (ix+N)	ld b,(iy+N)	bit 0, (ix+N)	bit 0,(iy+N)
71	47	1000111	ld b,a	bit 0,a	ld i,a				
72	48	1001000	ld c,b	bit 1,b	in c,(c)				
73	49	1001001	ld c,c	bit 1,c	out (c),c				
74	4A	1001010	ld c,d	bit 1,d	adc hl,bc				
75	4B	1001011	ld c,e	bit 1,e	ld bc,(NN)				
76	4C	1001100	ld c,h	bit 1,h					
77	4D	1001101	ld c,l	bit 1,l	reti				
78	4E	1001110	ld c,(hl)	bit 1,(hl)		ld c,(ix+N)	ld c,(iy+N)	bit 1,(ix+N)	bit 1,(iy+N)
79	4F	1001111	ld c,a	bit 1,a	ld r,a				
80	50	1010000	ld d,b	bit 2,b	in d,(c)				
81	51	1010001	ld d,c	bit 2,c	out (c),d				
82	52	1010010	ld d,d	bit 2,d	sbc hl,de				
83	53	1010011	ld d,e	bit 2,e	ld (NN),de				
84	54	1010100	ld d,h	bit 2,h					
85	55	1010101	ld d,l	bit 2,l					
86	56	1010110	ld d,(hl)	bit 2,(hl)	im 1	ld d,(ix+N)	ld d,(iy+N)	bit 2, (ix+N)	bit 2,(iy+N)
87	57	1010111	ld d,a	bit 2,a	ld a,i				
88	58	1011000	ld e,b	bit 3,b	in e,(c)				
89	59	1011001	ld e,c	bit 3,c	out (c),e				
90	5A	1011010	ld e,d	bit 3,d	adc hl,de				
91	5B	1011011	ld e,e	bit 3,e	ld de,(NN)				
92	5C	1011100	ld e,h	bit 3,h					
93	5D	1011101	ld e,l	bit 3,l					
94	5E	1011110	ld e,(hl)	bit 3,(hl)	im 2	ld e,(ix+N)	ld e,(iy+N)	bit 3,(ix+N)	bit 3,(iy+N)
95	5F	1011111	ld e,a	bit 3,a	ld a,r				
96	60	1100000	ld h,b	bit 4,b	in h,(c)				
97	61	1100001	ld hc,	bit 4,c	out (c),h				
98	62	1100010	ld h,d	bit 4,d	sbc hl,hl				
99	63	1100011	ld h,e	bit 4,e	ld (NN),hl				
100	64	1100100	ld h,h	bit 4,h					
101	65	1100101	ld h,l	bit 4,l					
102	66	1100110	ld h,(hl)	bit 4,(hl)		ld h,(ix+N)	ld h,(iy+N)	bit y,(ix+N)	bit 4,(iy+N)
103	67	1100111	ld h,a	bit 4,a	rrd				
104	68	1101000	ld l,b	bit 5,b	in l,(c)				
105	69	1101001	ld l,c	bit 5,c	out (c),l				
106	6A	1101010	ld l,d	bit 5,d	adc hl,hl				
107	6B	1101011	ld l,e	bit 5,e	ld hl,(NN)				
108	6C	1101100	ld l,h	bit 5,h					
109	6D	1101101	ld l,l	bit 5,l					
110	6E	1101110	ld l,(hl)	bit 5,(hl)		la l,(ix+N)	ld l,(iy+N)	bit 5,(ix+N)	bit 5,(iy+N)
111	6F	1101111	ld l,a	bit 5,a	rld				
112	70	1110000	ld (hl),b	bit 6,b	in f,(c)	ld (ix+N)	ld (iy+N),b		
113	71	1110001	ld (hl),c	bit 6,c		ld (ix+N),c	ld (iy+N),c		
114	72	1110010	ld (hl),d	bit 6,d	sbc hl,sp	id (ix+N),d	ld (iy+N),d		
115	73	1110011	ld (hl),e	bit 6,e	ld (NN),sp	ld (ix+N),e	ld (iy+N),e		
116	74	1110100	ld (hl),h	bit 6,h		ld (ix+N),h	ld (iy+N),h		
117	75	1110101	ld (hl),l	bit 6,l		ld (ix+N),l	ld (iy+N),l		
118	76	1110110	halt	bit 6,(hl)				bit 6,(ix+N)	bit 6,(iy+N)
119	77	1110111	ld (hl),a	bit 6,a		ld (ix+N), a	ld (iy+N),a		
120	78	1111000	ld a,b	bit 7,b	in a,(c)				
121	79	1111001	ld a,c	bit 7,c	out (c),a				
122	7A	1111010	ld a,d	bit 7,d	adc hl,sp				
123	7B	1111011	ld a,e	bit 7,e	ld sp,(NN)				
124	7C	11111100	ld a,h	bit 7,h					
125	7D	1111101	ld a,l	bit 7,l					

126	7E	11111110	ld a,(hl)	bit 7,(hl)		ld a,(ix+N)	ld a,(iy+N)	bit 7,(ix+N)	bit 7,(iy+N)
127	7F	11111111	ld a,a	bit 7,a					
128	80	10000000	add a,b	res 0,b					
129	81	10000001	add a,c	res 0,c					
130	82	10000010	add a,d	res 0,d					
131	83	10000011	add a,e	res 0,e					
132	84	10000100	add a,h	res 0,h					
133	85	10000101	add a,l	res 0,l					
134	86	10000110	add a,(hl)	res 0,(hl)		add a,(ix+N)	add a,(iy+N)	res o,(ix+N)	res o,(iy+N)
135	87	10000111	add a,a	res 0,a					
136	88	10001000	adc a,b	res 1,b					
137	89	10001001	adc a,c	res 1,c					
138	8A	10001010	adc a,d	res 1,d					
139	8B	10001011	adc a,e	res 1,e					
140	8C	10001100	adc a,h	res 1,h					
141	8D	10001101	adc a,l	res 1,l					
142	8E	10001110	adc a,(hl)	res 1,(hl)		adc a,(ix+N)	adc a,(iy+N)	res 1,(ix+N)	res 1,(iy+N)
143	8F	10001111	adc a,a	res 1,a					
144	90	10010000	sub b	res 2,b					
145	91	10010001	sub c	res 2,c					
146	92	10010010	sub d	res 2,d					
147	93	10010011	sub e	res 2,e					
148	94	10010100	sub h	res 2,h					
149	95	10010101	sub l	res 2,l					
150	96	10010110	sub (hl)	res 2,(hl)		sub (ix+N)	sub (iy+N)	res 2,(ix+N)	res 2,(iy
151	97	10010111	sub a	res 2,a					
152	98	10011000	sbc a,b	res 3,b					
153	99	10011001	sbc a,c	res 3,c					
154	9A	10011010	sbc a,d	res 3,d					
155	9B	10011011	sbc a,e	res 3,e					
156	9C	10011100	sbc a,h	res 3,h					
157	9D	10011101	sbc a,l	res 3,l					
158	9E	10011110	sbc a,(hl)	res 3,(hl)		sbc a,(ix+N)	sbc a,(iy+N)	res 3,(ix+N)	res 3,(iy+
159	9F	10011111	sbc a,a	res 3,a					
160	A0	10100000	and b	res 4,b	ldi				
161	A1	10100001	and c	res 4,c	cpi				
162	A2	10100010	and d	res 4,d	ini				
163	A3	10100011	and e	res 4,e	outi				
164	A4	10100100	and h	res 4,h					
165	A5	10100101	and l	res 4,l					
166	A6	10100110	and (hl)	res 4,(hl)		and (ix+N)	and (iy+N)	res 4,(ix+N)	res 4,(iy+
167	A7	10100111	and a	res 4,a					
168	A8	10101000	xor b	res 5,b	ldd				
169	A9	10101001	xor c	res 5,c	cpd				
170	AA	10101010	xor d	res 5,d	ind				
171	AB	10101011	xor e	res 5,e	outd				
172	AC	10101100	xor h	res 5,h					
173	AD	10101101	xor l	res 5,l					
174	AE	10101110	xor (hl)	res 5,(hl)		xor (ix+N)	xor (iy+N)	res 5,(ix+N)	res 5,(iy+
175	AF	10101111	xor a	res 5,a					
176	B0	10110000	or b	res 6,b	ldir				
177	B1	10110001	or c	res 6,c	cpir				
178	B2	10110010	or d	res 6,d	inir				
179	B3	10110011	or e	res 6,e	otir				
180	B4	10110100	or h	res 6,h					
181	B5	10110101	or l	res 6,l					
182	B6	10110110	or (hl)	res 6,(hl)		or (ix+N)	or (iy+N)	res 6,(ix+N)	res 6,(iy+
183	B7	10110111	or a	res 6,a					
184	B8	10111000	cp b	res 7,b	lddr				
185	B9	10111001	cp c	res 7,c	cpdr				
186	BA	10111010	cp d	res 7,d	indr				
187	BB	10111011	cp e	res 7,e	otdr				
188	BC	10111100	cp h	res 7,h					
189	BD	10111101	cp l	res 7,l					
190	BE	10111110	cp (hl)	res 7,(hl)		cp (ix+N))	cp (iy+N)	res 7,(ix+N)	res 7,(iy+
191	BF	10111111	cp a	res 7,a					
192	C0	11000000	ret nz	set 0,b					
193	C1	11000001	pop bc	set 0,c					
194	C2	11000010	jp nz,NN	set 0,d					
195	C3	11000011	jp NN	set 0,e					
196	C4	11000100	call nz,NN	set 0,h					
197	C5	11000101	push bc	set 0,l				set o,(ix+N)	set o,(iy+N
198	C6	11000110	add a,N	set 0,(hl)					
199	C7	11000111	rst 0	set 0,a					
200	C8	11001000	ret z	set 1,b					
201	C9	11001001	ret	set 1,c					
202	CA	11001010	jp z,NN	set 1,d					
203	CB	11001011		set 1,e					
204	CC	11001100	call z,NN	set 1,h					
205	CD	11001101	call NN	set 1,l				set 1,(ix+N)	set 1,(iy+N
206	CE	11001110	adc a,N	set 1,(hl)					
207	CF	11001111	rst 8	set 1,a					
208	D0	11010000	ret nc	set 2,b					
209	D1	11010001	pop de	set 2,c					
210	D2	11010010	jp nc,NN	set 2,d					
211	D3	11010011	out (N),a	set 2,e					
212	D4	11010100	call nc,NN	set 2,h					
213	D5	11010101	push de	set 2,l					
214	D6	11010110	sub N	set 2,(hl)				set 2,(ix+N)	set 2,(iy+N
215	D7	11010111	rst 16	set 2,a					
216	D8	11011000	ret c	set 3,b					
217	D9	11011001	exx	set 3,c					
218	DA	11011010	jp c,NN	set 3,d					
219	DB	11011011	in a,(N)	set 3,e					
220	DC	11011100	call c,NN	set 3,h					
221	DD	11011101		set 3,l					
222	DE	11011110	sbc a,N	set 3,(hl)				set 3,(ix+N)	set 3,(iy+N)
223	DF	11011111	rst 24	set 3,a					
224	E0	11100000	ret po	set 4,b					
225	E1	11100001	pop hl	set 4,c		pop ix	pop iy		

226	E2	11100010	jp po,NN	set 4,d					
227	E3	11100011	ex (sp),hl	set 4,e		ex (sp),ix	ex (sp),iy		
228	E4	11100100	call po,NN	set 4,h					
229	E5	11100101	push hl	set 4,l		pus ix	push iy		
230	E6	11100110	and N	set 4,(hl)				set 4,(ix+N)	set 4,(iy+N)
231	E7	11100111	rst 32	set 4,a					
232	E8	11101000	ret pe	set 5,b					
233	E9	11101001	jp (hl)	set 5,c		jp (ix)	jp (iy)		
234	EA	11101010	jp pe,NN	set 5,d					
235	EB	11101011	ex de,hl	set 5,e					
236	EC	11101100	call pe,NN	set 5,h					
237	ED	11101101		set 5,l					
238	EE	11101110	xor N	set 5,(hl)				set 5,(ix+N)	set 5,(iy+N)
239	EF	11101111	rst 40	set 5,a					
240	F0	11110000	ret p	set 6,b					
241	F1	11110001	pop af	set 6,c					
242	F2	11110010	jp p,NN	set 6,d					
243	F3	11110011	di	set 6,e					
244	F4	11110100	call p,NN	set 6,h					
245	F5	11110101	push af	set 6,l					
246	F6	11110110	or N	set 6,(hl)				set 6,(ix+N)	set 6,(iy+N)
247	F7	11110111	rst 48	set 6,a					
248	F8	11111000	ret m	set 7,b					
249	F9	11111001	ld sp,hl	set 7,c		ld sp, ix	ld sp, iy		
250	FA	11111010	jp m,NN	set 7,d					
251	FB	11111011	ei	set 7,e					
252	FC	11111100	call m,NN	set 7,h					
253	FD	11111101		set 7,l					
254	FE	11111110	cp N	ser 7,(hl)				set 7,(ix+N)	set 7,(iy+N)
255	FF	11111111	rst 56	set 7,a					

INTERPRETACION SIMBOLICA DE TABLAS

- Indicador no afectado. Indicador modificado según el resultado de la operación.

p El bit 2 del registro F actúa como indicador de pondad (1–pondad par/0–pondad impar).

V El bit 2 del registro F actúa como indicador de sobrepasamiento (1–sobrepasamiento/0–sobrepasamiento).

X Estado indeterminado (a) (b) (c). En las instrucciones donde se utiliza la abreviatura «s» el número de bytes, ciclos máquina y de reloj, son los mismos que sus correspondientes anteriores.

NOTAS

El lenguaje máquina es lo más próximo a la forma de hablar de nuestro micro. Por ello su conocimiento y comprensión nos descubrirá sus auténticas posibilidades y la manera de funcionar de nuestro pequeño «cerebro» electrónico. No desesperemos si el aluvión de nuevos conceptos desborda nuestras previsiones. En poco tiempo y con alguna práctica seremos capaces de manejarlos a nuestro antojo. Todos los programas y ejemplos estarán en forma clara y debidamente documentados para que podamos detectar cualquier posible equivocación de carga.

GRAN BIBLIOTECA AMSTRAD

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