Programación del Z80 con ensamblador

PARANINFO S.A.

Z80

Olivier LEPAPE

Programación del Z80 con ensamblador

Olivier LEPAPE

Programación del Z80 con ensamblador

1985

PARANINFO S.A.

MADRID

Traducido por:
FELISA MATEO GARCIA
y LUIS CAMPOY GUILLEN

Esta obra es traducción del libro francés
de OLIVIER LEPAPE "L'ASSEMBLEUR FACILE DU Z 80"
publicado por Editions Eyrolles, París (Francia)

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 84-283-1376-8

Depósito legal: M. 6.674.—1985

PARANINFO S.A. Magallanes, 25 - 28015 MADRID (3-3400)

ARTES GRÁFICAS BENZAL, S. A. Virtudes, 7. 28010 MADRID

Indice de materias

Al borde del precipicio

Este libro intenta ser una introducción a la programación en lenguaje máquina para todos los poseedores o futuros poseedores de un micro-ordenador concebido sobre un micro-procesador Z 80. Les podremos acompañar en sus primeros pasos hacia el descubrimiento de este nuevo lenguaje que enriquecerá en gran medida las posibilidades de su máquina.

Es posible que Vd. no posea actualmente ningún conocimiento sobre el lenguaje máquina; no se asuste, no es más que una impresión. En efecto, tachemos provisionalmente la palabra "máquina", no queda más que la palabra "lenguaje" y aquí nos situamos en un terreno ya conocido, ya que todos Vds. utilizan un lenguaje de programación que es el BASIC. Entonces empecemos por aquello que conocemos y tratemos de ver en qué se puede asemejar el lenguaje máquina, también llamado "ensamblador" con el BASIC, a pesar de las apariencias. No hay pues razón para asustarse, no vamos a lanzarnos de golpe a la fosa de los "LD A, (8A4 H), "DEC HL", "CALL SUBR" con el riesgo de ser devorados. Utilizaremos mejor una escalera para bajar por ella progresivamente; ya verá como a fin de cuentas las pequeñas fieras antes citadas no son tan antipáticas como parecen.

1

Ensamblador y Basic

Por curioso que pudiera parecer en un libro sobre el ensamblador, empezaremos por ver el BASIC. Esto no es absurdo si se considerar que el BASIC y el ensamblador son dos lenguajes de programación y que por lo tanto debe existir algo en común. Para descubrir los puntos comunes tratemos de resolver el mismo problema en BASIC y en ensamblador.

Realicemos un programa para calcular la cantidad:

$$1 + 2 + 4 + 8 + 16 + 32 + 64 \ldots = \sum_{i=0}^{N} 2^i$$

Este problema se resuelve con el pequeño programa:

```
 10  R = 0
 20  X = 1
 30  N = 5
 40  I = 0
 50  IF I = N + 1 THEN GOTO 90
 60  GOSUB 100
 70  I = I + 1
 80  GOTO 50
 90  END
100  R = R + X
110  X = X + X
120  RETURN
```

Esta solución no es necesariamente la mejor, pero nos ayudará a comprender lo que es un lenguaje de programación. El programa se presenta como una sucesión de líneas escritas en un lenguaje especial. Se pueden separar dichas líneas en diferentes grupos. Para empezar, tenemos una serie completa para la definición e inicialización de las variables necesarias para resolver el problema. En este caso, estas variables son números enteros. A continuación vienen unas líneas que realizan el tratamiento solicitado. Podemos diferenciar las líneas que efectúan un cálculo de las líneas que son órdenes de bifurcación y que componen el "esqueleto" del programa.

Podemos resumir el BASIC en tres grupos fundamentales de instrucciones:

Las instrucciones de bifurcación o de decisión	GOTO GOSUB RETURN IF THEN ELSE FOR NEXT
Las instrucciones aritméticas y lógicas	+ − * / SIN COS LOG EXP AND OR NOT etc...
Las instrucciones de entrada-salida	INPUT PRINT

Estos tres tipos de instrucciones son la base de un lenguaje de programación. Las instrucciones de bifurcación forman el "esqueleto" del programa, las instrucciones aritméticas y lógicas realizan las "funciones orgánicas" del programa y las entradas-salidas son el oído y la voz del programa.

Ya es hora de poner el pie en nuestro primer escalón, diciendo que estos tres tipos de instrucciones se encuentran en el lenguaje máquina o ensamblador con un vocabulario distinto. Así el GOSUB se llamará CALL, el GOTO, JP, las operaciones + y − serán ADD y SUB, etc... Esto significa que el análisis y la concepción de un programa informático son los mismos, bien sea en BASIC o en ensamblador, las divergen-

cias aparecen en el curso de la realización final, es decir, durante la escritura de las líneas. El ejemplo que hemos escrito en BASIC puede escribirse en ensamblador de la siguiente forma:

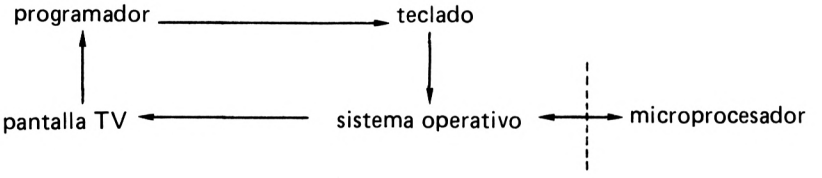
```
        LD    A, 0        |  10  R = 0
        LD    (R), A      |
        LD    A,1         |  20  X = 1
        LD    (X), A      |
        LD    A, 5        |  30  N = 5
        LD    (N), A      |
        LD    A, 0        |  40  I = 0
        LD    (I), A      |
TEST    LD    A, (N)      |  50  IF I = N + 1
        INC   A           |
        LD    HL, I       |
        CP    (HL)        |
        JP    Z, FIN      |                    THEN GOTO 90
        CALL  CALCUL      |  60  GOSUB 100
        INC   (HL)        |  70  I = I + 1
        JP    TEST        |  80  GOTO 50
        JP    END         |  90  END
CALCUL  LD    A, (R)      | 100  R = R + X
        ADD   A, (X)      |
        LD    (R), A      |
        LD    A, (X)      | 110  X = X + X
        ADD   A, (X)      |
        LD    (X), A      |
        RET               | 120  RETURN
```

La colocación en paralelo de estos dos programas escritos en dos lenguajes diferentes permite realizar varias observaciones. Lo primero que salta a la vista es que la versión ensamblador necesita muchas más líneas. En general, los programas escritos en ensamblador son más largos, lo cual no tiene por qué ser necesariamente un hándicap, ya que el programa ensamblador a menudo puede descomponerse en pequeños módulos que reagrupan una serie de instrucciones. Por ejemplo:

```
R = R + X    ⇔    LD   A, (R)
                  ADD  A, (X)
                  LD   (R), A
```

Con la práctica, cada uno se va creando su propia biblioteca de módulos adaptados a cada problema. Si la longitud de los programas es la ventaja del BASIC, el tiempo de ejecución es, por el contrario, claramente la ventaja del ensamblador. La instrucción BASIC R = R + X necesita alrededor de 1 ms. para su ejecución, mientras que las tres instrucciones en lenguaje máquina equivalentes necesitan solamente una decena de microsegundos; existe pues una relación de 100 respecto a la velocidad de ejecución. Esto es muy conveniente para los programas que utilizan las posibilidades gráficas de su ordenador.

Ahora que hemos puesto en evidencia un cierto parecido en lo que concierne al aspecto puramente informático entre el BASIC y el ensamblador, tratemos de ver si no existirían también algunas bases comunes en el lado máquina. El ordenador funciona de la siguiente forma:



Se deduce de este diagrama que existe una barrera entre el programador y el microprocesador. Al programador le resulta imposible decir al microprocesador qué es exactamente lo que debe hacer. El microprocesador está ligado al mundo exterior por unos cables eléctricos, siendo evidente que por sí solo es incapaz de comprender una palabra BASIC. El intermedio entre el teclado manipulado por el programador y el microprocesador es el "sistema operativo" (sistema de explotación).

El sistema operativo es un programa en lenguaje máquina que reside físicamente en ROM en el ordenador. Este programa interpreta las señales eléctricas provenientes del teclado, envía mensajes legibles a la pantalla, y sobre todo traduce una instrucción BASIC en una serie de instrucciones máquina materializadas en el ordenador mediante señales eléctricas con dos estados: 1 y 0. Una instrucción máquina se parece, por lo tanto, a esto:

```
1 0 1 1 0 0 1 1
```

Existe una enorme diferencia entre esto y:

```
R = R + X      o      LD A, (R)
```

En los dos casos el paso "texto" → "código máquina" necesita de un programa especial que se llama ensamblador o intérprete en el caso del BASIC. La diferencia reside en el hecho de que el ensamblador genera un solo código de máquina por línea mientras que el intérprete genera una serie de códigos máquina que realizan la instrucción (alrededor de 1000).

Cabría el preguntarse entonces por qué programar en ensamblador en lugar de seguir beneficiándose de las ventajas del BASIC que condensa una sucesión de códigos máquina en una sola instrucción.

Volviendo al ejemplo anterior, el ensamblador generará exactamente un código máquina por línea, es decir, 24 códigos; en cuanto al BASIC, efectuará, para cada línea, las siguientes operaciones:

a) Leer la instrucción,
b) Verificar la sintaxis,
c) Transformar esta instrucción en una serie de códigos máquina,
d) Ejecutar la instrucción.

El conjunto de las operaciones a, b, c y d necesita cerca de 1000 códigos máquina, lo cual da un total de 12000 códigos máquina. Esta diferencia enorme entre el BASIC y el ensamblador proviene de la no existencia de las operaciones a y b en el caso de un programa en ensamblador y de que la operación c es pensada y realizada por el programador. No queda más que la fase d. En BASIC el programador escribirá R = = R + X y no se planteará más problemas. En ensamblador en el punto R = R + X el programador debe preguntarse cómo hacer esto; realiza entonces la operación c y escribe:

```
LD A, (R)
ADD A, (X)
LD (R), A
```

Las principales ventajas del lenguaje máquina son:

– *Ejecución más rápida del programa.*
– *Utilización más eficaz del espacio de memoria.*
– *Programas con menos requerimientos de memoria.*
– *Libertad en relación al sistema operativo.*

Las principales desventajas del lenguaje máquina son:

– *Programas difíciles de leer y de poner a punto.*
– *Programas no adaptables a otros ordenadores.*

– *Programas que necesitan más líneas.*
– *Dificultades para efectuar cálculos aritméticos complicados.*

Hay que hacer, por lo tanto, una elección sobre el lenguaje que se va a emplear para resolver un problema.

El Basic resulta apropiado para la resolución de problemas matemáticos o de gestión sencilla.

El ensamblador se adapta muy bien a los problemas de juegos con animación gráfica.

Hasta ahora hemos mantenido una confusión entre el lenguaje máquina y el lenguaje ensamblador. Realmente el lenguaje máquina sólo se compone de 1's y 0's. No es fácil de comprender:

```
1 0 0 0 0 0 1 0
```

Por esto ha habido que crear un intermediario entre el hombre y la máquina; este intermediario es el ensamblador. Cada código máquina se sustituye por un nemónico. Resulta más fácil de leer:

```
ADD A, D
```

Este libro está aquí para explicarle que ADD significa "addition" (suma) y que A y D son dos variables de las que volveremos a hablar. El ensamblador se encargará de realizar la traducción.

```
ADD A, D → 1 0 0 0 0 0 0 1 0
```

El número binario 10000010 puede también representarse con un equivalente hexadecimal:

```
82
```

Esta representación es bastante más legible pero tampoco describe lo que hace la instrucción. No obstante algunos KITS microprocesadores utilizan únicamente este intermediario hexadecimal para programar en lenguaje máquina.

1.1. ENSAMBLADOR CONTRA BASIC

Ahora que hemos puesto los pies en el primer escalón, veamos el camino recorrido. Queremos escribir un programa para nuestro ordena-

dor; los pasos que hay que seguir son los siguientes:

– *Análisis del problema:*

Esta fase entraña la disección del gran problema primario en una multitud de pequeños problemas secundarios que llamaremos "módulos".

– *Composición de los distintos módulos:*

Se trata de enlazar estos módulos para formar el esqueleto del programa. Cada módulo es una función concreta cuyo esquema es:

variables de entrada → [tratamiento] → variables de salida

– *Realización del programa:*

En este momento hay que elegir entre BASIC y ensamblador siguiendo los criterios ya enunciados:

1.1.1. Ensamblador

Tratamientos que necesitan poco cálculo.
Tratamientos que manejen muchos datos gráficos.
Tratamientos rápidos.

1.1.2. BASIC

Muchos cálculos matemáticos.
Velocidad de ejecución poco crítica.
Solamente datos numéricos.

En cuanto a esto último, un programa BASIC lleno de sentencias tipo PEEK y POKE es, en general, más fácil de realizar en ensamblador.

Así pues, Vd ya sabe realizar las dos primeras etapas gracias al basic, déjese tentar ahora por la solución ensamblador de la tercera etapa. Al principio esto le exigirá un pequeño esfuerzo, pero le será plenamente recompensado cuando descubra las ilimitadas posibilidades que ofrece el ensamblador.

2

El microprocesador

Antes de aprender con detalle el lenguaje de este curioso individuo es importante conocerle para saber lo que hace y cómo lo hace.

Es posible que le hayan presentado al microprocesador como el cerebro, el "pequeño genio" que da vida a su ordenador. Debe Vd., desde ahora, desembarazarse de esta idea y considerar al microprocesador como un sencillo individuo al que se le pide que realice un gran número de tareas lo más rápidamente posible. Si su ordenador es un buen ajedrecista no es debido a la inteligencia del microprocesador sino a la del programador que ha sabido dividir el problema en un gran número de tareas elementales que son ejecutadas por el microprocesador. Estas tareas son, esencialmente, cálculos.

Nuestro microprocesador es por lo tanto, un individuo que puede ejecutar un cierto número de cálculos sucesivos rápidamente sin preocuparse en saber para qué van a servir. Esto es el problema del programador. Veamos un ejemplo para entrar más en detalle. Vd. quiere que su microprocesador calcule la cantidad que le queda sabiendo que Vd. tenía 57 Ptas. y que se ha gastado 23 Pts. Descomponga el problema para entregárselo al microprocesador:

1) ponga el número 57 en la casilla 1.
2) ponga el número 23 en la casilla 2.
3) tome el contenido de la casilla 1.
4) efectúe una sustracción con el contenido de la casilla 2.
5) ponga el resultado en la casilla 3.

Ahora el microprocesador puede efectuar las operaciones 1, 2, 3, 4 y 5 dando el resultado 34 en la casilla 3. Todas estas manipulaciones de

números, casillas y operaciones aritméticas serían muy penosas si el microprocesador tuviese que ejecutarlas de memoria. Por esta razón, utilizará los dedos para contar. ¡Un microprocesador utiliza las manos para contar!

Como veremos más tarde el Z80 posee un gran número de manos a las que científicamente se les llama registros. El microprocesador posee una mano especial con la que efectúa los cálculos aritméticos y lógicos: la "mano A". El microprocesador interpreta las líneas 3, 4 y 5 de la siguiente forma:

3) Contar con la mano A el valor inscrito en la casilla 1.

4) Restar de los dedos de la mano A el valor inscrito en la casilla 2.

5) Colocar en la casilla 3 el valor contenido en la mano A.

Se pueden hacer ya algunas observaciones sobre el microprocesador a partir de este ejemplo:

- El microprocesador no puede hacer cálculos con números decimales como 13,25. Sólo se pueden contar números enteros con las manos.
- En cuanto al tamaño de los enteros a tratar, el microprocesador está limitado por el número de dedos de una mano. Las manos del microprocesador tienen 8 dedos, lo cual le permite contar hasta 255 como veremos más adelante.
- El microprocesador no sabe que el número 57 representa una cantidad de dinero. No se preocupa en absoluto por saber qué representa el dato que está manipulando.

El concepto de variable que Vd. aprendió con el BASIC es muy diferente en ensamblador. En BASIC una variable es únicamente un número entero o decimal o un carácter alfanumérico. Cada variable tiene un nombre y el programador ya no se preocupa de saber cómo tratará el ordenador a esas variables. La gran diferencia con el ensamblador reside en el hecho de que ahora el programador debe preocuparse de la forma en la que va a "codificar" los datos del problema para que puedan ser tratados por el microprocesador. Como ya hemos dicho, el microprocesador calcula con los dedos, por lo tanto hay que adaptar la representación de las variables a este método de cálculo.

Una de las grandes diferencias entre el BASIC y el ensamblador consiste en la ausencia de variables definidas de antemano.

El ensamblador presentará pues un problema suplementario que es la codificación de los datos. Para el ejemplo que hemos enunciado anteriormente esta codificación era evidente. En efecto, sabiendo que el microprocesador puede contar hasta 255 con una mano, las cantidades 57 Pts. y 23 Pts. han sido codificadas mediante los números 57 y 23. Habría sido necesario encontrar otra solución si hubiésemos tenido 450 Ptas. porque el número 450 no cabe en una sola mano del microprocesador. Pueden presentarse otros problemas; por ejemplo, ¿cómo representaremos el alfabeto utilizando los dedos del microprocesador? Así pues, la casilla 1 que en principio contiene el número 57 representa una cantidad de dinero. Más tarde podemos perfectamente decidir que esta casilla contenga un código que represente la letra F y un poco más tarde, esta casilla representará un código indicado que se ha pulsado una tecla desde el teclado, etc... El programador es el único que decide sobre lo que representa el contenido de la casilla a lo largo del programa.

Vemos que el concepto de variable es más complicado en ensamblador, *pero también mucho más rico.* Podemos personalizar muy bien los dedos de una mano. Por ejemplo, podemos imaginar que el primer dedo representa el signo + ó el – y los 7 dedos restantes un valor numérico, o también que los 4 primeros representan la parte entera de un número decimal y los 4 últimos dedos representan la parte fraccionaria. Luego... ¿es posible entonces trabajar con números negativos y números decimales? Sí, pero no hay que olvidar que solamente el programador tiene conocimiento de cuál es su codificación, debiendo obrar en consecuencia en cuanto a su tratamiento. En cuanto al microprocesador, no ve más que los números que puede representar con los dedos de una de sus manos, es decir, los comprendidos entre 0 y 255.

Igual que ha sido posible edificar la torre Eiffel con solamente trozos de metal y remaches, es posible hacer un ajedrecista de un microprocesador que sólo posee la facultad de saber contar con sus manos de 8 dedos. Se encuentra Vd de frente a una persona que no sabe hacer más que sumas.

Problema: Vd. quiere que microprocesador le realice la operación 10 x 25.

Solución: Hágale sumar 10 veces el número 25.

La persona en cuestión habrá efectuado una multiplicación sin saberlo y le dará un resultado correcto. Su interés aumentará si consideramos que Vd. ha necesitado 0,1 s para realizar esta multiplicación,

mientras que la persona a la que Vd. se ha dirigido efectúa la suma en 3 μs. Necesitará pues 30 μs para efectuar la multiplicación. Esto explica por qué el microprocesador puede jugar al ajedrez: *es rápido y no comete errores de cálculo.*

Ahora que conoce un poco mejor las posibilidades de un microprocesador comprenderá por qué los programas en ensamblador son largos pero rápidos. Pero volvamos al lenguaje. El ejemplo citado antes se escribe:

```
INICIALIZACION

        LD    A, 57
        LD    (CASE 1), A
        LD    A, 23
        LD    (CASE 2), A

CALCULO

        LD    A, (CASE 1)
        SUB   A, (CASE 2)
        LD    (CASE 3), A
```

A es el nombre asignado a la mano privilegiada del Z 80. LD es un nemónico que significa LOAD (cargar). Las palabras entre paréntesis significan "el contenido de".

```
(CASILLA 1) = el contenido de CASILLA 1
```

Ya ve Vd. que el ensamblador no es tan complicado. Queda una pequeña duda:

Nosotros tenemos 10 dedos y contamos hasta 10 con las dos manos. Entonces ¿cómo puede contar el microprocesador hasta 255 con solamente 8 dedos? Encontraremos la solución un escalón más abajo.

3

¿Cómo efectua los cálculos el microprocesador?

El microprocesador cuenta con los dedos de una manera mucho más organizada que nosotros. Si un dedo extendido representa un 1 y un dedo doblado representa un 0 no existe para nosotros ninguna diferencia entre las dos representaciones siguientes:

1 0 0 0 0 1

El microprocesador en cambio establece una diferencia entre el pulgar y el índice cuando cuenta con los dedos. Ordena sus ocho dedos y con solamente dos dedos puede contar hasta tres.

00 = 0

01 = 1

10 = 2

11 = 3

El sistema para contar es el siguiente:

El primer dedo cambia de estado (doblado, extendido) con cada incremento; cada dedo cambia de estado si el dedo anterior pasa del estado extendido al estado doblado. De esta forma se obtienen 256 configuraciones posibles de los 8 dedos de cada mano del microprocesador, permitiéndole contar de 0 a 255. Pruebe con 3 dedos Vd. podrá contar de 0 a 7.

Esta forma de contar con los dedos está en relación directa con la numeración binaria. Una mano de Z 80 está materializada por un conjunto de 8 casillas que pueden registrar dos valores: 0 ó 1.

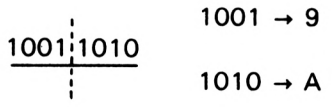
La posición de una mano se representa por una secuencia de ocho dígitos binarios (0 ó 1) llamados bits:

⇔ 10011010

Para simplificar esta representación hemos partido la mano en dos mitades de 4 dedos. Dado que se puede contar de 0 a 15 con 4 dedos las 16 posiciones posibles están representadas por una cifra:

0000	0	→ 0
0001	1	→ 1
0010	2	→ 2
0011	3	→ 3
0100	4	→ 4
0101	5	→ 5
0110	6	→ 6
0111	7	→ 7
1000	8	→ 8
1001	9	→ 9
1010	10	→ A
1011	11	→ B
1100	12	→ C
1101	13	→ D
1110	14	→ E
1111	15	→ F

así:



está representado por el número haxadecimal:

9A

La representación haxadecimal se utiliza mucho en ensamblador, ya que la unidad de base que sirve para todos los cálculos es la mano de 8 dedos. El programa en ensamblador se encargará de convertir los números hexadecimales en binario. La representación del sistema binario mediante dígitos hexadecimales es el método más utilizado por varias razones:

1. Conversión hexadecimal binario sencilla.
2. Distinción entre los números de 8 y 16 bits.
   8 bits = 2 dígitos hexadecimales
   16 bits = 4 dígitos hexadecimales

Este sistema permite representar cualquier registro o cualquier casilla (celda) de memoria por dos dígitos hexadecimales. Todo el ambiente del Ẓ 80 está organizado en bloques de 8 bits ó 16 bits:

- registro: 8 bits
- memoria: 8 bits
- dirección: 16 bits

3.1. LA ARITMETICA DE 4 BITS

Es importante familiarizarse con esta aritmética para comprender cómo calcula un microprocesador como el Z80. Hay que empezar por saber pasar de la representación binaria a la representación hexadecimal o decimal.

1111 = 8 + 4 + 2 + 1 = 15 decimal

F hexadecimal

Observe que cada dedo representa las potencias crecientes de 2. Para obtener el valor decimal, hay que sumar las potencias de 2, correspondientes a cada dedo extendido:

dedo 1: $2^0 = 1$
dedo 2: $2^1 = 2$
dedo 3: $2^2 = 4$
dedo 4: $2^3 = 8$

así la mano siguiente representa el valor 10.

= 1010 binario
= 8 + 0 + 2 + 0 = 10 decimal
= A hexadecimal

Para evitar confusiones entre los números decimales y los números hexadecimales, estos últimos van seguidos de la letra H.

10, 38 decimal

AH, 38H hexadecimal

¡¡*Atención!!* 38H no representa el mismo número que 38.

Es muy importante saber trabajar con las diferentes representaciones numéricas, por eso le proponemos este pequeño ejercicio antes de continuar:

Realice la conversión "binario → decimal" y "binario → hexadecimal"

	decimal	hexadecimal
0010:		
0110:		
1001:		
1010:		
1100:		

por ejemplo:

1101 = 8 + 4 + 0 + 1 = 13 (decimal)
DH (hexadecimal)

3.2. LA ARITMETICA DE 8 BITS

La aritmética de 4 bits sólo puede tratar números comprendidos entre 0 y 15. Si queremos representar el número 16 hay que añadir dedos a la mano. En la mano de 8 dedos del Z 80:

= 16 decimal
= 10 H hexadecimal

Se observará que $16 = 2^8$ con lo que se extiende el método expuesto para 4 bits:

0001 000 = 0 + 0 + 0 + 16 + 0 + 0 + 0 + 0 = 16
0001 000 = 10 H

En hexadecimal basta con cortar la manos en dos y representar cada media-mano de 4 dedos por su equivalente hexadecimal. Por ejemplo:

$$0101\ 0011 = 0 + 2^6 + 0 + 2^4 + 0 + 0 + 2^1 + 2^0$$

$$0 + 64 + 0 + 16 + 0 + 0 + 2 + 1 = 83 \quad \text{decimal}$$

0101 0011 = 53 H — hexadecimal

Ahora se comprende por qué es más fácil de manejar el hexadecimal que el decimal. Se plantea ahora el problema de la conversión hexadecimal en decimal.

Como estamos acostumbrados al sistema decimal, cuando vemos el número 83 efectuamos inmediatamente la operación:

$$83 = (8 * 10) = 3$$

Se produce exactamente la misma situación en hexadecimal, de manera que el valor 53 representa:

$$53H = (5 * 16) + 3 = 80 + 3 = 83$$

No existe por lo tanto ningún problema para convertir de hexadecimal a decimal. Si se mira la capacidad de una mano de ocho dedos:

= 1111 1111
= F F H
= (15 * 16) + 15 = 255

Practique Vd. ahora, convirtiendo cualquier número comprendido entre 0 y 255 hexadecimales, en binario y en decimal. Es importante saber desenvolverse bien con estos sistemas de numeración.

decimal	27	28	29	30
hexadecimal	27 H	28 H	29 H	2 AH 2 BH 2 CH 2 DH 2 EH 2 FH 30 H

El valor que sigue al 29H es 2AH y no 30H. Por otra parte 28H no representa el mismo número que 28.

3.3. LA SUMA

La suma binaria es extremadamente sencilla; basta recordar las siguientes fórmulas:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

tabla de suma binaria

La suma se efectúa empleando la tabla de suma binaria de la misma manera que en numeración decimal. El caso 1 + 1 = 10 entraña el paso al rango superior exactamente igual que con 8 + 9 = 17 (7 + 10) en numeración decimal. La suma de dos números de 8 bits se hace así:

00101101	=	45	2DH
01110100	=	116	74H
10100001	=	161	A1H

3.4. NUMEROS NEGATIVOS

Ya hemos dado una idea sobre la representación de los números negativos cuando dijimos que el bit situado más a la izquierda podía representar el bit de signo y los 7 bits restantes y el valor absoluto del número. Con este sistema hay 128 posiciones de los dedos, para los cuales el dedo de la izquierda está extendido y 128 posiciones para los cuales está doblado. Esto da 128 números negativos y 128 números positivos. Admitiendo que 0 es positivo (bit 7 = 0) se puede contar en el intervalo:

−128 a + 127

Como ya hemos dicho, el microprocesador no sabe si el dedo de la izquierda representa el signo del número o no. Le corresponde al programador la decisión de considerar un cierto código binario como un número con signo (bit 7 = signo + ó −) o como un valor positivo de 8 bits. Todas las instrucciones del Z 80 son compatibles con una representación con signo siempre y cuando se elija correctamente una representación para los números negativos.

Es lógico codificar el valor 0 como:

00000000

La representación de los números negativos ha de ser tal que la suma de un número y su opuesto debe dar un resultado nulo. Para esto no basta con invertir el bit 7:

```
0 0100100 = + 36          0 = signo +
1 0100100 = − 36 ???      1 = signo −
-----------------
1 1001000 = − 72 !!!
```

Esta representación no es buena. Para representar un número negativo hay que pasar por las siguientes etapas:

1) 36 = 0|0100100

2) se invierte cada bit, lo cual da:

11011011

3) se le suma 1:

```
  11011011
  00000001
  --------
  11011100
−36 = 1|1011100
```

Si ahora realizamos la operación (+ 36) + (−36) se obtiene:

```
          00100100
        ←11011100
(acarreo)  ------
(perdido) * 00000000
```

El acarreo del nivel 8 se pierde porque las manos del microprocesador sólo tienen 8 dedos. El resultado final obtenido es 0. Con esta representación el mayor número positivo es:

$$01111111 = +127$$

El número negativo más pequeño es:

$$10000000 = -128$$

El método que hemos descrito también es válido para encontrar el opuesto de un número negativo:

1) $-36 = 11011100$

2) se invierte cada bit:

00100011

3) se le suma 1:

```
  00100011
  00000001
  --------
  00100100
```

$36 = 00100100$

A esta representación de los números negativos se le llama el complemento a 2. El método enunciado es válido cualquiera que sea el número

de bits elegidos. En todos los casos el bit situado más a la izquierda representa el signo:

0 = +

1 = −

No debe olvidar que esto no es más que una convención y que el programador decide por sí mismo cuando un número codificado representa un valor positivo (0,255) o un valor con signo (−128, + 127).

Con la convención elegida (complemento a 2) el microprocesador dará siempre el resultado correcto:

$$\begin{cases} 00101101 = 45 \\ 10001000 = 136 \end{cases} \qquad 45 + 136 = 181$$

o

$$\begin{cases} 0\,0101101 = +45 \\ 1\,0001000 = -120 \end{cases} \qquad (+45) + (-120) = -75$$

En los dos casos, durante la suma el microprocesador efectúa:

$$\begin{array}{r} 00101101 \\ 10001000 \\ \hline 10110101 \end{array}$$

10110101 = 181

o

1 0110101 = − 75

Como ejercicio, calcule el complemento a 2 de −75 y verifique que se obtiene 75.

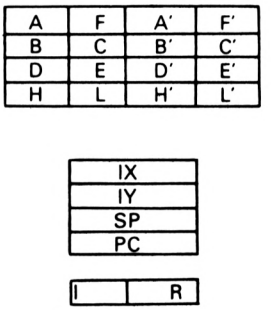
Esto nos demuestra que el programador tiene la libre elección de la convención adoptada para representar los números. En los dos casos, el microprocesador calcula de la misma forma y da un resultado correcto.

Es importante tener las ideas claras sobre estas convenciones. Le aconsejamos encarecidamente que tome un lápiz y un papel y repita todos los cálculos que le hemos expuesto. No será muy divertido, pero le proporcionará una buena formación.

4

Los registros del Z 80

La configuración de los registros del Z80 es la siguiente:



Podemos clasificar estos registros en dos categorías. Los registros de uso general y los registros dedicados.

4.1. REGISTROS DE USO GENERAL

Son los registros B, C, D, E, H, L, B', C', D', E', H' y L'. Cada uno puede contener un octeto. Podemos considerar a estos registros como simples celdas (casillas) de memoria directamente accesibles por el microprocesador sin tener que pasar a través de una dirección. Estas celdas

no tienen dirección y se distinguen unas de otras atribuyéndolas un nombre. Así una instrucción del tipo:

"hacer la operación X con el contenido de la celda de memoria de dirección Y" se convierte en:

"hace la operación X con el contenido del registro B" por ejemplo.

Estos registros pueden estar asociados dos a dos para formar un solo registro de 16 bits. En este caso ya no se puede considerar a un par de registros como una casilla normal de memoria. Esta asociación de dos registros hace que el Z80 posea un cierto número de instrucciones que operan con datos de 16 bits. Estos pares de registros están ya establecidos. Se pueden asociar dos a dos los registros "BC", "DE", "HL", "B'C'" y "H'L' ". El octeto con más peso está contenido en el primer registro (B, D, H, B', D' y H'), el octeto con menos peso está en el segundo registro (C, E, L, C', E' y L').

El microprocesador no puede acceder directamente al conjunto de estos registros. Podríamos considerar que existen dos juegos intercambiables de registros. Una vez seleccionado un juego, el otro se coloca al lado conservando, evidentemente, su información. Se pueden comparar a estos dos juegos con unas pizarras de dos caras girando alrededor de un eje central. La cara activa sobre la cual se escribe corresponde al juego de registros (B, C, D, E, H, L), la cara oculta corresponde al juego puesto a continuación (B', C', D', E', H', L'). El microprocesador posee una instrucción del tipo "hacer girar la pizarra 180". La cara sobre la que se ha estado escribiendo se retira, conservando lo que se hubiese escrito en ella y ahora se trabaja sobre la cara que antes estaba oculta. Este tipo de pizarras es muy práctico para un profesor que por ejemplo conserve los resultados principales en una de las caras y efectúe los cálculos intermedios en la otra. Este método se emplea a menudo en programación; volveremos sobre este punto bajo el nombre más científico de "salvaguarda del contexto". Por el momento, quédese con la idea clara de que el Z 80 posee dos juegos de registros, uno activo y otro a la espera, pudiendo intercambiar estos dos juegos mediante la instrucción apropiada en cualquier momento.

4.2. REGISTROS DEDICADOS

Son los registros A, F, A', F', IX, IY, SP, PC I y R. Se puede observar que algunos registros contienen palabras de 8 bits (aquellos cuyo nom-

bre es de una sola letra) y otros contienen palabras de 16 bits (aquellos cuyo nombre tiene dos letras).

4.2.1. Registros A y A'

A inicial de acumulador. El microprocesador tiene dos acumuladores pero sólo utiliza uno a la vez, el acumulador A. A' se mantiene inactivo. Cuando el acumulador se utiliza para realizar operaciones combinatorias (A = A □ dato; donde □ representa una operación cualquiera) se le asocia un registro especial llamado "Flag" o F (y F'). Este "Flag" contiene datos binarios relacionados con el resultado de la operación realizada. Los registros A y F (A' y F'), en ningún caso pueden asociarse para componer un único registro de 16 bits.

Los datos contenidos en A y F están relacionados pero son de naturaleza totalmente diferente. Los dos juegos "acumulador" "flag" AF y A'F' forman también una pequeña pizarra de dos caras que el microprocesador puede hacer girar independiente de los registros de uso general (B, C, D, E, H, L, B', C', D', E', H', L').

4.2.2. Registro PC

Este registro es el contador de programa (Program Counter). Contiene una palabra de 16 bits que representa la dirección de la siguiente instrucción. Para cada instrucción se incrementa el contenido del registro en el número de octetos necesarios para la codificación de la instrucción incluídos sus operandos. Se puede modificar por programa el valor del PC, lo cual permite modificar la dirección de la siguiente instrucción y saltar a distintos puntos en el programa. Es el equivalente de los GOTO y GOSUB del BASIC.

4.2.3. Registro IX

Este registro de 16 bits se llama registro de índice. El microprocesador lo utiliza para un modo de direccionamiento especial llamado direccionamiento indexado (o con base). El dato contenido en IX es la dirección de una casilla de memoria específica, elegida por el programador. El contenido de IX puede apuntar a todas las celdas de la memoria central debido a sus 16 bits de longitud. Este registro no es indispensable como lo son los registros A, F y PC pero facilita mucho la resolución de los problemas que se presentan en el tratamiento de tablas, por ejemplo.

4.2.4. Registro IY

Es otro registro de índice siendo su utilización idéntica a la del registro IX.

4.2.5. Registros I y R

Estos dos registros no son de ninguna utilidad para el programador. Son utilizados únicamente por algunas configuraciones "hardware". Solamente diremos, a título de cultura general, que el registro R se utiliza para un tipo de memoria llamada RAM dinámica.

4.2.6. Registro SP

Este registro es el apuntador de la pila (Stack Pointer). Es muy importante comprender bien el funcionamiento de este registro que vamos a describir. Veremos a continuación cómo es utilizado por el microprocesador y por el programador. El registro SP contiene una palabra de 16 bits que representa una dirección de la memoria central. Como su nombre indica, el registro SP sirve para gestionar una pila.

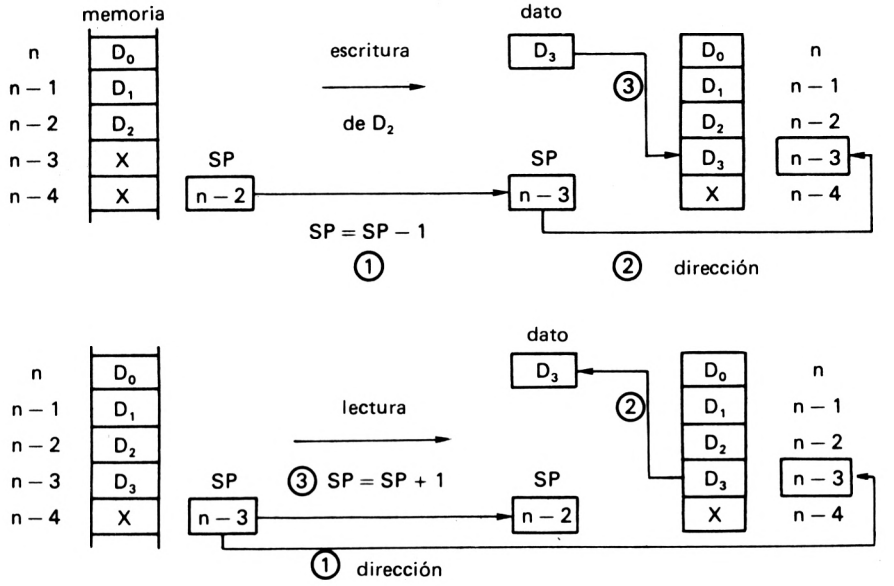
4.3. DEFINICION DE LA PILA

Una pila es una serie de celdas contiguas de memoria cuyo acceso está gestionado por el registro SP. Se puede comparar este tipo de acceso, algo especial, con el apilamiento de platos en un armario. Cada dato a escribir en la pila sería el equivalente al final de la pila. Cada escritura realizada por el microprocesador en la pila aumenta el número de datos apilados, conservándolos todos. Por el contrario, cada lectura de la pila equivale a retirar el plato superior que es el único accesible sin peligro de roturas. Se deduce entonces que el primer dato leído de la pila será el último que se escribió. Y con cada lectura se va accediendo a los datos más profundos. Este tipo de acceso se llama pila "último que entra, primero que sale" o "Last In, First Out" o más corrientemente pila LIFO.

La gestión de esta pila por el registro SP es muy sencilla. Al comenzar, cuando la pila está vacía, SP contiene la dirección de una casilla de memoria. Para cada escritura en la pila, el registro SP es decrementado

en 1, escribiéndose a continuación, el dato en la celda de memoria cuya dirección viene indicada por el contenido del registro SP. De esta manera los datos de la pila se colocan en direcciones contiguas decrecientes. Para cada lectura se tiene el camino inverso. Es decir, se lee la casilla de memoria apuntada por SP y después se incrementa en 1 este registro.

Estas operaciones aparecen resumidas en el esquema siguiente: Las diferentes operaciones están anotadas cronológicamente 1, 2, 3.



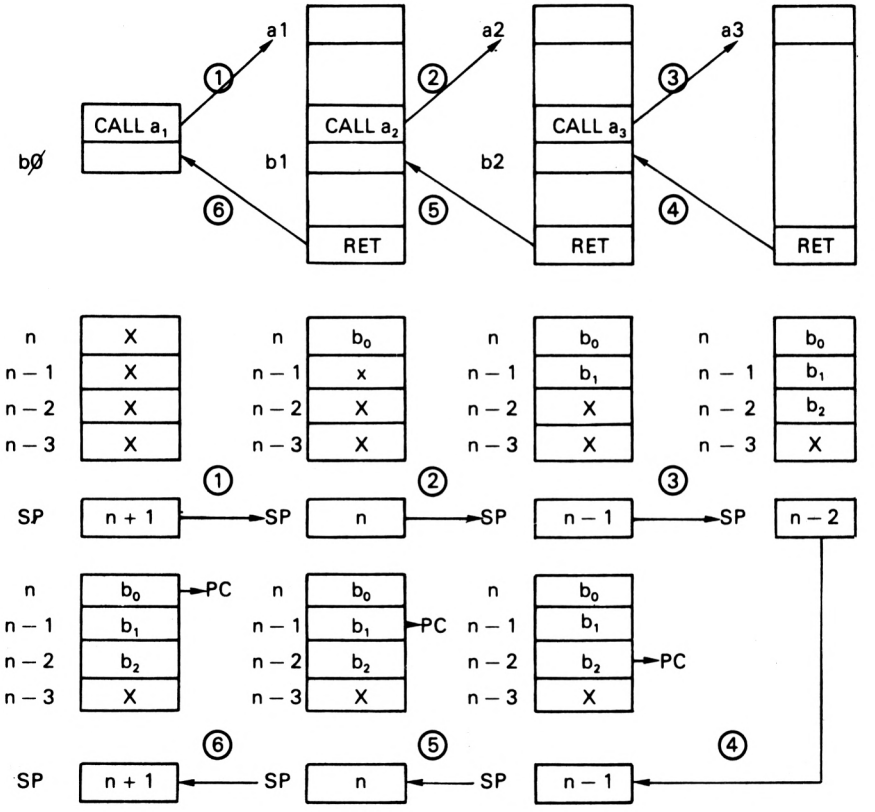
Observe que la lectura no afecta a los valores guardados en la pila. Sólo queda modificado el registro SP.

La pila es un elemento esencial para el funcionamiento del microprocesador. Es pues primordial, al comienzo de cada programa, tener reservada una zona de memoria para la pila e inicializar el registro SP con la dirección superior de esta zona de memoria. No le describiremos el desastre que se produciría si como consecuencia de este olvido el registro SP apuntase a la zona de memoria donde reside el programa o los datos.

Utilización de la pila por el microprocesador:

La pila se utiliza para la llamada a las rutinas. Vd. ya conoce el GOSUB del BASIC y su RETURN asociado. Estas dos instrucciones tie-

nen un equivalente en el lenguaje máquina con CALL nn y RET. Cuando el microprocesador encuentra un CALL nn bifurca a la dirección nn y continúa el programa hasta que encuentra la instrucción RET. En ese momento hay que volver a la instrucción siguiente el CALL nn. Esto significa que el microprocesador ha conservado en algún sitio esta dirección. Este sitio es la pila. El funcionamiento de la pila permite imbricar (anidar) las rutinas de forma casi ilimitada. En efecto, la última dirección de retorno apilada corresponde al primer RET encontrado. Recordemos de nuevo el conocido LIFO y el siguiente esquema le resultará claro:



La dirección de la instrucción siguiente al CALL está en el PC. Cada CALL pone en la pila el PC y a continuación lo carga con la dirección de comienzo de la rutina. Para cada RET el PC saca la dirección de la

pila. Observe que son necesarias dos operaciones con la pila para colocar los 16 bits del PC después de un CALL.

$$(SP - 1) = PC_H \quad y \quad (SP - 2) = PC_L \qquad SP = SP - 2$$

después de un RET:

$$PCL = (SP) \qquad y \qquad PC_H = (SP + 1) \qquad SP = SP + 2$$

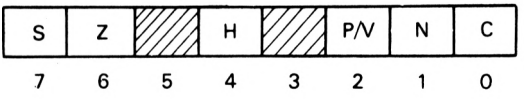
La notación (SP) significa "contenido de la casilla de memoria, cuya dirección está contenida en SP".

Utilización de la pila por el programador:

La pila resulta práctica a menudo para almacenar datos temporales. El programador dispone de varias instrucciones para leer o escribir en la pila los datos contenidos en los registros del Z80. Veremos cómo esta pila puede ser utilizada para salvar el contexto y para pasar parámetros en las llamadas a las rutinas.

4.4. REGISTRO F

Este registro de 8 bits contiene unos indicadores (flags) binarios cuyo valor (0 ó 1) depende de la última instrucción ejecutada por el microprocesador. Cada uno de los bits de este registro tiene un significado determinado que vamos a describir. La estructura de este registro es la siguiente:



4.4.1. Bit 0 ó C

Este bit es el indicador de acarreo. Es posicionado por las operaciones aritméticas, de desplazamiento y de comparación. Se puede poner a 1 mediante la instrucción especial (SCF). Las operaciones lógicas (AND, OR, XOR) ponen este bit a 0.

SUMA:

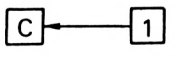
El Z80 puede efectuar la suma de dos octetos. El resultado es un octeto. Esto significa que los datos y el resultado tienen un valor comprendido entre 0 y 255. Hablemos en numeración decimal para entendernos mejor. Supongamos que tenemos un microprocesador ficticio que trabaja con números comprendidos entre 0 y 99 en númeración decimal.

```
  70
+ 80
-----
 150
```

El resultado 150 es mayor que 99. En ese caso, el microprocesador da un resultado igual a 50 y un acarreo para las centenas, luego C = 1. La suma de 60 y 25 da 85. En este caso no hay acarreo para las centenas y C = 0. Con el Z80 sucede exactamente lo mismo, aunque sólo trabaje en binario.

00100011	23 H	35
01001001	49 H	73
01101100	6 CH	108

El resultado cabe en 8 bits (< 256), por lo tanto no hay acarreo y C = 0:



10100011	A3H	163
10101110	AEH	174
01010001	151 H	337

El resultado ocupa 9 bits (≥ 256) por lo tanto hay acarreo para el nivel 2^8 y C = 1.

RESTA:

El caso de la resta es más complejo y hay que recurrir al concepto de complemento a 2. La resta es tratada por el microprocesador como

una suma del primer número y el complemento a 2 del número sustraendo. En este caso C = 0 significa que el resultado es positivo. El caso C = 1 significa que el resultado es negativo y viene expresado por su complemento a 2. Dado que lo que se suma es el complemento a 2, el acarreo C registrará el complemento del acarreo real de la suma.

Hagamos la operación 30 – 25. Lo que realmente hace el microprocesador es: 30 + comp (25) que podemos descomponer en:

(30 – 25) + (25 + comp (25))

Como 25 + comp (25) = 256 = 100H;

(30 – 25) + (25 + comp (25)) = 5 + 256 = 5H + 100H = 105H

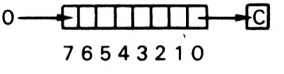
El acarreo real es 1, pero como lo que se registró es el complemento de este acarreo C = 0 y el resultado de 30 – 25 es positivo. Hagamos ahora 30 – 35. El microprocesador realiza la suma:

30 + comp (35)
(30 – 35) + (35 + comp (35))
– 5 + 256 = 251

El resultado es menor que 256 y no conlleva acarreo, pero como se registró el complemento de este acarreo C = 1. El resultado obtenido es efectivamente el complemento a 2 de 5 (comp (5) = 256-5), es decir, (–5).

DESPLAZAMIENTO:

El bit de acarreo C queda afectado por numerosos tipos de desplazamiento debidos a las instrucciones RLA, RLCA, RRA, RRCA, RL, RLC, RR, RRC, SLA, SLA, SRA y SRL de las que hablaremos con más detalle más adelante. Citemos, por ejemplo, el desplazamiento lógico a la derecha SRL. El bit 0 del octeto sobre el que se realiza el desplazamiento pasa al indicador C:



Todas estas rotaciones efectúan circuitos diferentes pasando por el indicador C.

COMPARACIONES:

De hecho, cuando el microprocesador compara dos números A y B efectúa de forma ficticia la operación A − B sin dar el resultado, con el único objeto de poder posicionar el flag correspondiente en el indicador C. El acumulador se carga siempre con el valor A. El valor B se carga en un registro o en una casilla de memoria definida por el operando de la instrucción de comparación. Aplicando todo lo dicho para la resta:

Si $A \geqslant B$ resultado de $A - B \geqslant 0$ luego $C = 0$

Si $A < B$ resultado de $A - B < 0$ luego $C = 1$

4.4.2. Bit 1 ó N

El flag N indica si la última operación efectuada por el Z80 ha sido una resta. En ese caso N = 1. Este indicador se utiliza con el indicador H y la operación de ajuste decimal.

4.4.3. Bit 2 ó P/V

El significado de este flag depende de la operación que acaba de realizarse. Sirve como indicador de desbordamiento de capacidad (overflow) V en el caso de una operación aritmética, de incremento o de decremento. En el caso de operaciones lógicas o de desplazamiento, sirve como indicador de paridad P.

PARIDAD:

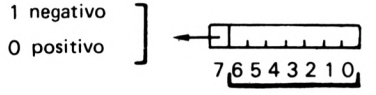
La paridad de un octeto viene dada por el número de 1 contenidos en el octeto. Si este número es par, la paridad es par y P = 1. En caso contrario, la paridad es impar y P = 0.

A = 01100100	3 (1) impar	P = 0
A = 10011010	4 (1) par	P = 1

DESBORDAMIENTO:

El bit V sólo es útil cuando se trabaja con octetos con signo. Un octeto con signo permite representar números positivos y negativos. Los nú-

meros positivos están codificados en binario natural, y los negativos con el complemento a 2 de su valor absoluto. (Para un octeto, el complemento a 2 de 11 es el valor 256 – 11 = 245). En este caso los números que se pueden codificar en un octeto son los comprendidos entre –128 y + 127. El bit 7 del octeto representa el bit de signo. Para los números positivos bit 7 = 0 (normal ya que son + 127), y para los números negativos bit 7 = 1 (de –1 a –128 están representados por 256 – 1 = 255 a 256 – 128 = 128, luego bit 7 = 1).



binario natural si bit 7 = 0

complemento a 2 si bit 7 = 1

El valor del número está codificado por lo tanto en 7 bits. Durante las operaciones aritméticas si el resultado necesita más de 7 bits para su codificación existe desbordamiento (overflow). Este desbordamiento tiene lugar en la suma de dos números del mismo signo o en la resta de dos números de signo opuesto. En la práctica se obtiene V = 1 cuando el resultado es incoherente. Por ejemplo:

```
   79      01001111
+ 119      01110111
           --------
          11000110
          ↓
```

¿¿el resultado es negativo?? → V = 1

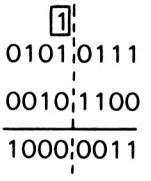
La suma de dos números positivos nunca puede dar un número negativo. Pero para el microprocesador 119 + 79 = 198 ⩾ 127 no se puede codificar con 7 bits.

El siguiente es un método sencillo para predecir el desbordamiento: existe desbordamiento cuando el bit 7 del resultado y el bit de acarreo son complementarios:

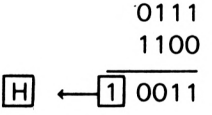
bit 7 = 0 y C = 1
bit 7 = 1 y C = 0

4.4.4. Bit 4 ó H

Este flag es el indicador de acarreo en el nivel 2^4. Es decir, que funciona exactamente igual que el C pero limitándose a los 4 primeros bits 0, 1, 2 y 3.



Hay un acarreo en el nivel 2^4 porque:



y H = 1

4.4.5. Bit 6 ó Z

Este indicador señala si el resultado de la última operación realizada ha sido nulo. En ese caso Z = 1.

Para las operaciones aritméticas y lógicas Z = 1 si el resultado es nulo. Para las operaciones de comparación Z = 1 si los dos valores comparados son iguales. Durante las comprobaciones de bit, Z = 1 si el bit comprobado es 0. Este flag queda también afectado por algunas instrucciones de entrada-salida en bloque cuando el decremento de B da un resultado nulo (alcanza el valor 0).

4.4.6. Bit 7 ó S

Este flag indica el signo del octeto contenido en el acumulador, es decir, toma el valor del bit 7 del octeto contenido en el acumulador.

5

Los modos de direccionamiento del Z 80

Los modos de direccionamiento conciernen a todos los medios puestos a disposición del microprocesador para acceder a los datos a partir de los operandos de una instrucción. El microprocesador accede siempre a la memoria central generando una dirección; no hay pues nada nuevo a este respecto. Sin embargo, los operandos de una instrucción no tienen por qué ser una dirección real, sino un número, un indicativo a partir del cual el microprocesador calcula la dirección real de la casilla de memoria donde se encuentra el dato a tratar. El siguiente ejemplo demuestra lo interesante que resulta el disponer de varios modos de direccionamiento.

Sean dos variables A y B, A es un número codificado en un octeto y B es una tabla de diez números, codificados cada uno en un octeto. Estos diez números ocupan posiciones contiguas de la memoria central. Las instrucciones que traten la variable A tendrán, normalmente, como operando la dirección real de A, sin embargo, para la variable B sería poco práctico dar explícitamente una de las diez direcciones correspondientes a los elementos de la tabla.

Es más lógico decirle al microprocesador: "La operación actual afecta al octavo elemento de la tabla B". Esto es posible utilizando un modo de direccionamiento apropiado, para el cual el operando de la instrucción no es la dirección real del octavo elemento, sino el dígito 8. Para ello, necesita que la dirección del primer elemento de la tabla B sea conocida por el microprocesador, almacenándola en un registro especial.

Vamos a describir los ocho modos de direccionamiento que tiene el Z80. La descripción de cada uno de estos modos vendrá completada con una instrucción ilustrativa del Z 80.

Direccionamiento extendido:

Este modo de direccionamiento es el más sencillo, ya que el operando es la dirección real de la casilla de memoria afectada. Se designa con la palabra "extendido" para indicar que este modo de direccionamiento permite acceder a la totalidad del espacio de memoria. El operando es, por lo tanto, una palabra de 16 bits (2 octetos).

La instrucción:

"cargar en el acumulador el contenido de la casilla de memoria de dirección OC745H" se escribe:

LD A, (OC745H)

Tenemos pues:

dirección efectiva (DE) = operando

5.1. DIRECCIONAMIENTO INDIRECTO A TRAVES DE REGISTRO

Este modo de direccionamiento no necesita operando. La dirección de la celda de memoria es el contenido de uno de los pares de registros del Z80 HL, DE o BC. Al estar codificada la dirección en 16 bits, se puede acceder a la totalidad del espacio de memoria. Este modo de direccionamiento es muy útil para pasar parámetros a las rutinas.

La instrucción:

"cargar en el acumulador el contenido de la casilla de memoria, cuya dirección está en el registro DE" se escribe:

LD A, (DE)

Tenemos pues:

DE = contenido de HL o BC o DE

5.2. DIRECCIONAMIENTO INDEXADO

Este modo de direccionamiento también se llama "relativo a una base" y corresponde al ejemplo de la introducción en el que se hace referencia a la tabla B. El operando es un octeto con signo (−128 a + 127). El microprocesador calcula la dirección real mediante la suma del operando (llamado desplazamiento) más el contenido de uno de los registros de índice (IX o IY).

La instrucción:

"cargar en el acumulador el contenido de la casilla de memoria del elemento 8 de la tabla apuntada por el registro IX" se escribe:

```
LD A, (IX + 8)
```

Tenemos pues:

DE = *contenido de IX o IY + desplazamiento*

Estos tres modos de direccionamiento que acabamos de ver constituyen el único sistema que posee el Z80 para acceder a los datos de la memoria central.

5.3. DIRECCIONAMIENTO A TRAVES DE REGISTRO

Este modo de direccionamiento está reservado para las operaciones internas del microprocesador que no necesitan ningún acceso a memoria. Las manipulaciones de registros utilizan este modo de direccionamiento, ya que un registro es una casilla de memoria directamente accesible sin dirección.

La instrucción:

"cargar en el registro B el contenido del registro D" se escribe:

```
LD B, D
```

operando = nombre del registro

5.4. DIRECCIONAMIENTO IMPLICITO

Algunas instrucciones del Z80 sólo pueden ejecutarse a través del acumulador. En ese caso no hay necesidad de precisar implícitamente que el operando es el acumulador.

La instrucción:

"rotación a la izquierda del acumulador a través del acarreo" se escribe:

·RLCA

sin operando

5.5. DIRECCIONAMIENTO INMEDIATO

No se trata realmente de un modo de direccionamiento aunque se le considere como tal por deformación del lenguaje. En efecto, el operando no es ni una dirección, ni un desplazamiento, ni un nombre de registro, sino el dato numérico a tratar por la instrucción. En un modo de direccionamiento auténtico, el operando indica dónde se encuentra el dato.

La instrucción:

"cargar en el acumulador el valor 133" se escribe:

LD A, 133D

Operando = dato (8 bits)

5.6. DIRECCIONAMIENTO INMEDIATO EXTENDIDO

Este modo de direccionamiento es idéntico al anterior pero en este caso el dato está codificado en dos octetos. Se utiliza para cargar los registros de 16 bits HL, BC, DE, SP, IX, IY.

La instrucción:

"inicializar el puntero de pila con la dirección 8000H" se escribe:

```
LD SP, 8000H
```

operando = dato (16 bits)

Existe otro registro de 16 bits que no hemos nombrado, el contador PC. De hecho, una bifurcación no es más que la carga inmediata en el registro PC de un dato de 16 bits. En lugar de escribir:

```
LD PC, XXXXH
```

se utiliza la forma más expresiva:

```
JP XXXXH
```

donde JP (Jump) indica un salto a la dirección XXXXH.

5.7. DIRECCIONAMIENTO RELATIVO AL PC

Ya que estamos en las bifurcaciones nos quedaremos en ellas. El operando es un desplazamiento (octeto con signo). Este modo de direccionamiento sólo es utilizado por el contador PC y permite efectuar bifurcaciones modificando su valor. El microprocesador realiza la operación PC = PC + desplazamiento. La bifurcación relativa al PC resulta interesante porque es independiente del lugar de la memoria central en el que se cargue el programa.

La instrucción:

"volver 30 octetos hacia atrás" se escribe:

```
JR – 30 D
```

operando = desplazamiento

5.8. DIRECCIONAMIENTO A TRAVES DE LA PAGINA CERO

Este modo de direccionamiento es utilizado por el Z80 para una llamada especial a rutinas llamada RESTART. Esta instrucción fuerza en el PC una dirección de la página cero (los bits con mayor peso se ponen a cero). El Z80 tiene 8 restarts diferentes bifurcando a ocho rutinas que empiezan en las direcciones 0, 8H, 10H, 18H, 28H, 30H, 38H. Las instrucciones:

```
CALL 10 H     (llamada a rutina)

y

RST 10H       (RESTART)
```

son totalmente equivalentes, aunque el CALL y su operando se codifican en tres octetos mientras que el RST sólo necesita uno.

El RESTART es utilizado para rutinas llamadas con mucha frecuencia debido a la rapidez que proporciona.

Hay que observar que el empleo de la instrucción RST está reservado a menudo para el sistema operativo del microprocesador y corresponde a determinadas rutinas en ROM. Su utilización no resulta cómoda y no es aconsejable para principiantes.

6

El ensamblador del Z 80

En el capítulo de introducción le hemos mostrado lo útil que resulta disponer de un intermediario entre el programador en lenguaje máquina y el microprocesador que sólo entiende de unos y ceros. Este intermediario es el ensamblador. Un ensamblador es un programa que realiza una traducción de un texto llamado fuente en una serie de octetos llamada código objeto, que es directamente comprensible por el microprocesador, en el caso de pequeños ordenadores individuales. Veremos más adelante, que en esta traducción pueden intervenir otros intermediarios. En realidad el ensamblador es algo más que un traductor que se contente con transformar la línea:

```
LD A, B
```

en el octeto 78H. El ensamblador ofrece otras posibilidades que permiten simplificar en gran medida la escritura de los programas. Todo ello se resume en el hecho de que se puede asignar un valor numérico a un nombre. De aquí se desprende la posibilidad de asignar un nombre a una dirección de variable o de instrucción, o a una constante numérica. Es más agradable leer la línea:

```
CALL TECLADO
```

en lugar de

```
CALL 3BF7H
```

En el primer caso nos damos cuenta inmediatamente de que se trata de una llamada a una rutina de lectura de teclas desde el teclado alfa-

numérico. El segundo caso nos deja perplejos. Pero volvamos al principio y descubramos la "gramática" extremadamente sencilla del ensamblador.

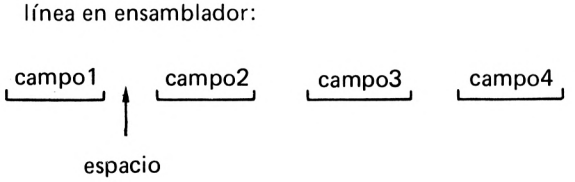
DOS REGLAS PRINCIPALES:

– *Las reglas absolutas:* Son aquellas impuestas por el programa ensamblador que sólo admite líneas formateadas y una sintaxis correcta. Encontramos la misma situación en BASIC. Si no se cumplen estas reglas el ensamblador genera un código de error e indica la línea causante, de la misma forma que su intérprete BASIC.

– *Las reglas recomendadas:* Son las reglas que debe imponerse el programador para conseguir una programación rápida, minimizando al máximo el tiempo de puesta a punto de los programas, que raramente funcionan a la primera. Por ejemplo, deberán utilizarse preferentemente nombres de variables significativas del tipo: CONTADOR, BUCLE, INDICE, etc., en lugar de X, Y, Z,... Los programas serán claros y las rutinas se escribirán separadas y claramente definidas. Estas reglas son muy importantes porque la utilización de nemónicos no clarifica los programas.

6.1. LA SINTAXIS DEL ENSAMBLADOR

Un programa en ensamblador se presenta como una sucesión de líneas. Cada línea representa una instrucción ejecutable por el microprocesador o una directiva de las que hablaremos más tarde. La línea en ensamblador tiene una estructura claramente definida. Está compuesta por varias zonas llamadas "campos", separadas por un delimitador que normalmente es un espacio. Estos campos son cuatro y cada uno de ellos tiene un papel definido:



6.1.1. Campo 1 o zona de etiqueta

Una etiqueta es un nombre especial asignado a una línea. Este nombre sólo puede aparecer una vez en la zona de etiqueta; en caso contrario, el ensamblador genera un código de error (Etiqueta asignada dos veces). La presencia de etiqueta no es obligatoria. En ese caso el campo 1 es reemplazado por una tabulación (CTL I) o TAB.

La presencia de etiqueta es muy útil para las instrucciones de bifurcación o de llamada a rutinas. Es más sencillo escribir:

```
JP SEGUIR
```

que es una bifurcación a la línea cuya etiqueta es SEGUIR, que:

```
JP 034AFH
```

Esta segunda opción presupone que el programador sabe de antemano que la línea SEGUIR corresponde a una instrucción que ocupa la dirección 034AFH y éste no suele ser nunca el caso.

Aquí también se deben utilizar etiquetas significativas.

Aproveche esta posibilidad que representa una ventaja sobre el Basic que sólo reconoce etiquetas numéricas (GOSUB 380). Evite sobre todo las etiquetas A, B, C..., ya que son nombres reservados para los registros del Z80.

El campo etiqueta está a menudo limitado por un número máximo de caracteres. Para el ensamblador Z80 MOSTEX, que es el más habitual, este límite es de seis caracteres.

6.1.2. Campo 2 o zona de operación

Se reserva este campo para el nemónico de una instrucción o de una directiva. Todos los nemónicos del Z80 se encuentran relacionados más adelante. Un nemónico es un conjunto de 2, 3 ó 4 letras que define una instrucción ejecutable por el microprocesador. Este conjunto es una contracción de una palabra o de una frase escrita en inglés:

```
LoaD    LD      carga
Decrement and Jump relative if Not Zero      DJNZ
```

Al contrario de los nemónicos de las instrucciones, que son fijados por el constructor del Z80, los nemónicos de las directivas dependen del ensamblador utilizado.

6.1.3. Campo 3 o zona operando

Este campo es el más complicado. Corresponde al operando de la instrucción escrita en el campo 2. Este operando puede ser simple o doble. Si es doble podemos distinguir entre operando origen (fuente) y operando destino. Los operandos pueden ser:

- números
- nombres de variables
- nombres de etiquetas
- nombres de registros
- expresiones aritméticas o lógicas

Un operando escrito entre parántesis indica que representa una dirección, así:

```
LD A, 037H
```

significa: cargar el acumulador con el valor 037H.

```
LD A, (037H)
```

significa: cargar el acumulador con el contenido de la casilla de memoria de dirección 037H.

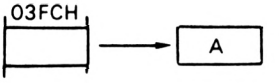
(Expresión) significa siempre: contenido de la casilla de memoria de dirección "expresión".

La instrucción de carga es un caso de operando doble. Los dos operandos, origen y destino, están separados por una coma (,). El primer operando es el destino, el segundo es el origen. Las instrucciones con operando doble tienen siempre el siguiente formato:

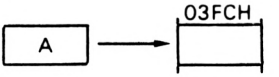
```
nemónico      destino, origen
(campo2)        (campo3)
```

así las instrucciones:

LD A, (03FCH)



LD (03FCH), A

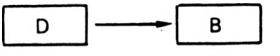


no son equivalentes. Podemos definir origen y destino como:

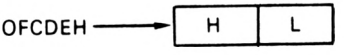
Destino: lugar donde es colocado el resultado de la operación ejecutada.
Origen: lugar donde se encuentra el dato que va a ser tratado por la instrucción.

Cuando el operando es un registro, éste viene indicado por su nombre:

LD B, D



LD HL, OFCDEH



La forma de escribir el operando está ligada al modo de direccionamiento empleado; podemos distinguir:

– direccionamiento extendido:

```
operando = (expresión numérica)
LD A, (03FCH)
```

– direccionamiento indirecto a través de los registros:

```
operando = (par de registros)
LD A, (BC)
```

– direccionamiento indexado:

```
operando = (IX o IY + expresión numérica)
LD A, (IX + 10)
```

– direccionamiento por registro:

```
operando = nombre de registro
LD A, C
```

– direccionamiento inmediato:

```
operando = expresión numérica
LD A, 30
LD HL, 0D034H
```

– direccionamiento relativo al PC:

```
operando = etiqueta de bifurcación
JP BUCLE
```

En este caso el ensamblador sólo calcula el valor del desplazamiento.

Expresiones aritméticas y lógicas

La parte numérica de un operando puede ser una expresión conteniendo sumas, restas, el Y lógico o el desplazamiento lógico. Son válidas las siguientes expresiones:

```
LD A, 30 + 40
LD A, 25 – 6
LD A, – 10H
LD A, 11H & 01H
```

El desplazamiento lógico se expresa a través de la combinación:

```
valor < número de desplazamientos
3C00H < +2  el valor 3C00H se ha desplazado 2 veces a la
izquierda → ED00H
3C00H <–2  el valor 3C00H se ha desplazado 2 veces a la
derecha → 0ED0H
```

Se tiene, por ejemplo:

```
LD HL, 3C00H < +2
```

Representación de los números

El ensamblador acepta varias bases de numeración, pero las más usuales son:

decimal	base 10
hexadecimal	base 16
octal	base 8

Cualquier número expresado en una de estas bases va seguido de una letra identificativa de la base:

decimal	letra D o nada	34D ó 34
hexadecimal	letra H	0A3F4H
octal	letra O	430

La letra D del sistema decimal es opcional. Cuando el ensamblador encuentra un número como 12 ó 12D lo interpreta como una constante decimal. El ensamblador sólo admite números enteros.

El sistema hexadecimal (base 16) consta de 16 símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. Una constante numérica debe empezar necesariamente por un dígito (0 a 9). La escritura F2H es incorrecta, sin embargo 0F2H, que representa el mismo valor, es correcto.

El sistema octal (base 8) necesita 8 símbolos: 0, 1, 2, 3, 4, 5, 6 y 7. Una constante octal se escribe, por ejemplo, 670. Recordemos la fórmula de transformación de una base cualquiera a decimal:

$$3C2FH = F \times 1 + 2 \times 16 + C \times (16)^2 + 3 \times (16)^3$$
$$= 15 \times 1 + 2 \times 16 + 12 \times (16)^2 + 3 \times (16)^3 = 15407$$

$$4320 = 2 \times 1 + 3 \times 8 + 4 \times (8)^2 = 282$$

En la práctica se emplean a menudo los sistemas decimal y hexadecimal. El octal se utiliza para expresar un código de operación. Permite distinguir tres campos en un código de operación. El decimal se emplea a menudo para inicializar los contadores o para los desplazamientos durante la búsqueda en una tabla. El hexadecimal es práctico para expresar las direcciones de las variables o de las rutinas externas en memoria central.

6.1.4. Campo o zona de comentarios

Este campo está separado del resto de la línea por un espacio y un ; . Después del ; se puede escribir lo que se quiera. Con ello se permite documentar los programas haciéndolos más claros y asequibles. Un buen comentario no debe completar el nemónico empleado en la línea. Muy por el contrario, debe aportar una información suplementaria sobre el papel que dicha línea juega en el programa.

```
LD B, 10    ; cargar el valor 10 en B      coment. malo
LD B, 10    ; hacer 10 veces el bucle      coment. bueno
```

si, por ejemplo, el registro B se ha utilizado como contador de bucle.

Algunas líneas no contienen más que un comentario. En ese caso deben comenzar por ; . Esto hace que los programas no sean tan compactos, que se puedan poner títulos, etc,...

```
; COMENTARIO
BUCLE LD B, 10      ; hacer el bucle 10 veces.
```

6.2. FUNCIONAMIENTO DE UN ENSAMBLADOR

Como ya hemos dicho, el ensamblador es un programa que efectúa un tratamiento sobre un texto escrito en ensamblador llamado fuente y que genera un código objeto ejecutable por el microprocesador.

La línea de ensamblador se compone de varios campos. La zona etiqueta contiene palabras desconocidas, a priori, por el ensamblador. Sin embargo, la zona operación utiliza un nemónico conocido por el ensamblador, ya que corresponde a un código de operación del Z80 o a una directiva. La zona operando contiene un cierto número de símbolos, conocidos por el ensamblador que son los paréntesis, los nombres de registros, las letras D, H, O, los símbolos aritméticos y lógicos (+, –, &, <) y unos símbolos desconocidos a priori que son las etiquetas utilizadas por el programador. El ensamblador ignora la zona comentario.

Una etiqueta (label) no es más que la asignación de una palabra a un valor numérico, por consiguiente, el ensamblador debe, en un primer paso, hacer una recapitulación de estas asignaciones, dando lugar a lo

que se llama la tabla de símbolos. Constituye su propio diccionario, permitiéndole analizar todo el programa. Esto necesita de una primera lectura completa del texto fuente para no dejar pasar ninguna etiqueta. Durante esta primera pasada el ensamblador también verifica la sintaxis de las líneas y visualiza los errores que va encontrando. Por ejemplo, en el campo operación el ensamblador debe reconocer el nemónico, si no es así, visualizará un código de error (operación desconocida) y la línea afectada. Mientras el ensamblador configura la tabla de etiquetas, no debe encontrar más que una asignación por etiqueta.

Pero ¿cuál es el valor numérico que corresponde a una etiqueta dada? Será necesario que "profundicemos" algo más en el ensamblador si queremos comprender más adelante el papel de algunas directivas.

Cuando el ensamblador encuentra una etiqueta en el campo 1 le asigna el valor de la dirección que ocupará el código de operación de la instrucción definida por el nemónico de la línea cuando el programa sea ensamblado y esté en código ejecutable por el microprocesador. El ensamblador debe ingeniárselas para conocer de antemano esta dirección. Para esto utiliza un puntero (indicador de dirección) inicializado con un valor fijo por el programador al comienzo del programa (directiva ORG). Cada vez que el ensamblador encuentra una instrucción del repertorio del Z80, incrementa el puntero en un número igual al número de octetos necesario para codificar la instrucción y su operando (1, 2, 3 ó 4). Este puntero traduce el valor del contador de programa, cuando el programa se está ejecutando. Este contador contiene la dirección del código de operación de la siguiente instrucción. Así pues, cuando el ensamblador encuentra una etiqueta le asigna el valor del puntero de dirección.

Una vez finalizada esta primera pasada, el ensamblador ejecuta una segunda pasada durante la cual tiene lugar el ensamblaje propiamente dicho. Supongamos que el ensamblador encuentra la línea:

```
ETIQ    LD A, (VARIAB)
```

- Reconoce LD: Instrucción de carga.
- Reconoce A: cargar en el acumulador.
- Reconoce ( ) y VARIAB no es un registro: direccionamiento extendido.

El ensamblador busca VARIAB en la tabla de símbolos y encuentra el valor 3C45 que le fue asignado; por otra parte, sabe que el código de

operación de "cargar el acumulador con direccionamiento extendido" es 3A. La línea dada, una vez ensamblada, quedará:

```
3A453C
```

Observe que se coloca primero el octeto con menor peso (octeto débil). Viene impuesto por el microprocesador.

Durante esta segunda pasada se detectan algunos errores. Por ejemplo: el cálculo de un desplazamiento (salto relativo al PC) > 127 ó < −128 o una etiqueta sin definir.

6.3. DIRECTIVAS DE LOS ENSAMBLADORES

Una directiva (o pseudo-operación) es una instrucción especial utilizada solamente por el ensamblador y que no da lugar a la generación de un código de operación del Z80. Describiremos a continuación las directivas más usuales de los ensambladores Z80.

ORG nn:

Esta directiva debe colocarse al principio del programa.

Establece la dirección para la primera instrucción encontrada, inicializando el puntero de dirección.

```
        ORG     3C00H
COMIEN  LD      SP, 3000H  ; primera instrucción
        |
        |
        |
```

El código de operación de esta primera instrucción es colocado en la dirección 3C00H. Después del ensamblaje tendremos en la memoria central:

3BFF	XX	
3C00	31	carga inmediata del SP
3C01	00	
3C02	30	

END nn

Esta directiva debe encontrarse obligatoriamente al final del programa. Indica al ensamblador que el texto fuente ha terminado.

Si se escribe:

```
END COMIEN
```

La etiqueta COMIEN será la dirección de comienzo del programa después de su carga en memoria. Esta dirección no siempre coincide con la de la primera instrucción del programa.

etiq. EQ nn

Esta directiva asigna explícitamente un valor numérico nn a la etiqueta. Permite dar nombres a las variables o a las rutinas externas asignándoles la dirección al nombre:

```
VARIAB  EQU 4000H    ; variable
RUTIN   EQU 27BCH    ; rutina en ROM
```

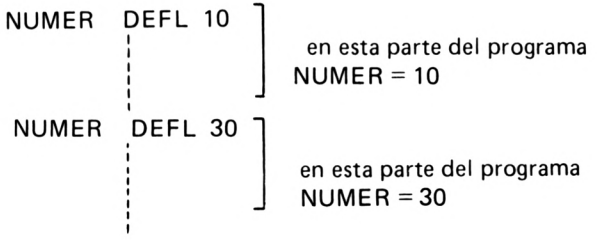
Más adelante podremos encontrarnos instrucciones como:

```
LD      A, (VARIAB)
CALL    RUTIN
```

Una etiqueta definida con EQU no puede volver a ser definida más tarde. Esta directiva sirve para definir las referencias externas del programa, por eso, el valor asignado a la etiqueta no es el del puntero de dirección, como ocurre con una línea de instrucción. En el caso del ejemplo, la rutina RUTIN, que se encuentra en ROM, no se puede definir mediante una línea de programa con el puntero de dirección, hay que utilizar por lo tanto la directiva EQU.

etiq. DEFL nn

Esta etiqueta tiene la misma característica que EQU pero en este caso la etiqueta puede redefinirse:



6.3.1. Directivas de reserva de memoria

Las directivas que relacionamos a continuación, a diferencia de las anteriores, tienen una influencia directa sobre el espacio de memoria donde se cargará el programa. Estas directivas modifican el valor del puntero de dirección.

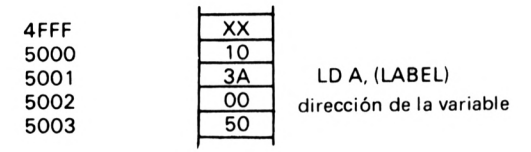
etiq. DEFB n

Esta directiva inicializa el contenido de la casilla de memoria cuya dirección es el valor del puntero de dirección. El puntero se incrementa en 1. Analicemos dos ejemplos para ver la diferencia que existe con EQU:

1)

```
        ORG     5000H
LABEL   DEFB    10H
        LD      A, (LABEL)
        :
        :
        :
```

El ensamblador dará:

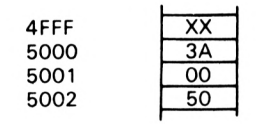


Se ha reservado una celda de memoria en la dirección 5000H para LABEL y se ha inicializado con 10H.

2)

```
        ORG     5000H
LABEL   EQU     5000H
        LD      A, (LABEL)
        :
        :
        :
```

El ensamblaje dará:



Esta vez no ha habido reserva de memoria central.

etiq. DEFB "S"

Esta directiva es idéntica a la anterior inicializando la casilla de memoria con el código ASCII del carácter escrito entre comillas.

```
        ORG     5000H
                        4FFF    XX
CARACT  DEFB    "A"     5000    41      código ASCII de A
```

etiq. DEFS nn

Esta directiva reserva nn celdas de memoria a partir del valor del puntero de dirección. Sirve para reservar una zona de memoria para las tablas, por ejemplo:

Supongamos que el puntero de dirección PD = 53F0:

```
TABLA DEFS 3

LABEL LD   HL, TABLA
```

53F0	00	3 octetos reservados
53F1	00	
53F2	00	
53F3	21	LD HL, inmediato
53F4	F0	valor asignado a TABLA
53F5	53	= 53F0

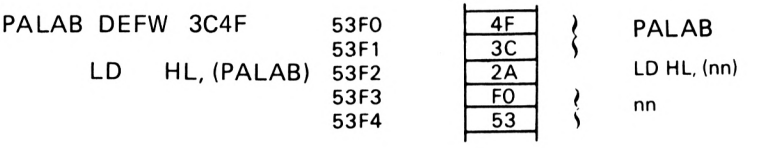
etiq. DEFM "ssss"

Esta directiva permite inicializar cadenas de caracteres ASCII a partir de la dirección del PD:

```
CADENA      DEFM  "TEXTO"       53F0    54      T
                                53F1    45      E
            LD    HL, CADENA    53F2    58      X
                                53F3    54      T
                                53F4    49      O
                                53F5    21      LD HL, nn
                                53F6    F0      }
                                53F7    53      }  nn
```

etiq. DEFW nn

Esta directiva reserva dos octetos para escribir en ellos la palabra nn El octeto con menor peso se escribe en primer lugar:



Existen otras directivas que influyen sobre la presentación del listado del ensamblaje:

TITLE	TITULO	da un nombre al listado
PAGE		salto de página
NOLIST		interrumpe el listado (no el ensamblaje)
LIST		obtención de listado

8

El juego de instrucciones del Z 80

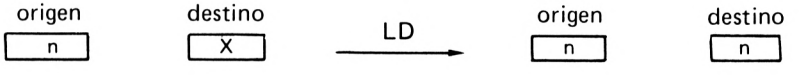
7.1. INSTRUCCION DE CARGA 8 BITS

El conjunto de estas instrucciones tiene como único efecto el de volver a copiar el dato numérico definido por el operando origen en el operando destino sin que quede alterado su valor. Estas instrucciones no efectúan ninguna operación aritmética, lógica o de desplazamiento sobre el dato manipulado, no afectando a los indicadores del registro F (flags).

La escritura de estas instrucciones es:

```
LD    destino, origen
```

El esquema de la instrucción es el siguiente:



Podemos dividir este conjunto de instrucciones en tres subconjuntos dependiendo de la naturaleza del origen y del destino.

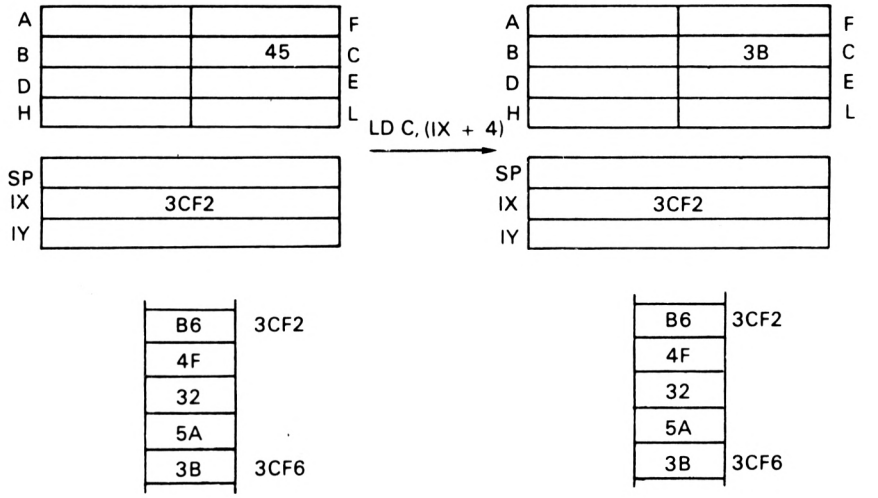
```
cargar registro      : memoria   ——→  registro
almacenar registro : registro   ——→  memoria
transferir           : registro   ——→  registro
```

Tanto el origen como el destino están definidos por uno de los modos de direccionamiento que hemos visto. Daremos ahora la lista ilustrada con ejemplos, de todas las instrucciones de carga.

CARGAR REGISTRO:

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
LD A, (nn)	extendido	ninguno
LD A, (BC)	indirecto a través de registro	"
LD A, (DE)	"	"
LD A, (HL)	"	"
LD B, (HL)	"	"
LD C, (HL)	"	"
LD D, (HL)	"	"
LD E, (HL)	"	"
LD H, (HL)	"	"
LD L, (HL)	"	"
LD A, (IX + d)	indexado con IX	"
LD B, (IX + d)	"	"
LD C, (IX + d)	"	"
LD D, (IX + d)	"	"
LD E, (IX + d)	"	"
LD H, (IX + d)	"	"
LD L, (IX + d)	"	"
LD A, (IY + d)	indexado con IY	"
LD B, (IY + d)	"	"
LD C, (IY + d)	"	"
LD D, (IY + d)	"	"
LD E, (IY + d)	"	"
LD H, (IY + d)	"	"
LD L, (IY + d)	"	"

La utilización de estas instrucciones no plantea ninguna dificultad. Recordemos con un ejemplo el funcionamiento de direccionamiento indexado:



El desplazamiento es 4, luego la dirección de la palabra que hay que cargar en C es:

3CF2 + 4 = 3CF6

El contenido de esta celda de memoria es 3B y éste es el valor que se carga en el registro C.

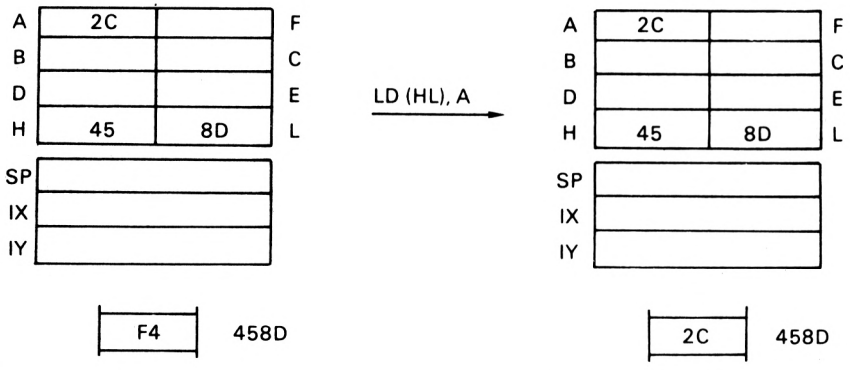
ALMACENAR REGISTRO:

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
LD (nn), A	extendido	ninguno
LD (BC), A	indirecto a través de registro	"
LD (DE), A	"	"
LD (HL), A	"	"
LD (HL), B	"	"
LD (HL), C	"	"
LD (HL), D	"	"
LD (HL), E	"	"
LD (HL), H	"	"
LD (HL), L	"	"

Instrucción	*Modo de direccionamiento del destino*	*Flags afectados*
LD (IX + d), A	indexado con IX	ninguno
LD (IX + d), B	"	"
LD (IX + D), C	"	"
LD (IX + d), D	"	"
LD (IX + d), E	"	"
LD (IX + d), H	"	"
LD (IX + d), L	"	"
LD (IY + d), A	indexado con IY	"
LD (IY + d), B	"	"
LD (IY + d), C	"	"
LD (IY + d), D	"	"
LD (IY + d), E	"	"
LD (IY + d), H	"	"
LD (IY + d), L	"	"

Este conjunto de instrucciones es idéntico al de cargar registro pero los orígenes y los destinos están ahora invertidos.

Recordemos con el siguiente ejemplo el direccionamiento indirecto a través de registro:



La dirección en la que se almacenará el registro A viene dada por el par de registros HL:

458D

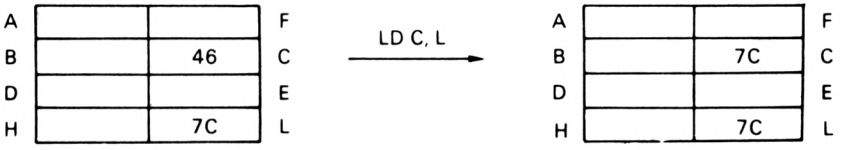
El contenido de A se escribe en esta dirección.

TRANSFERIR:

Para este conjunto de instrucciones el origen y el destino son un nombre de registro (direccionamiento a través de registro).

```
LD   R1, R2                    ningún flag afectado
R1 = A, B, C, D, E, H, L
R2 = A, B, C, D, E, H, L
```

Por ejemplo:



El contenido del registro L se copia en el registro C.

Existen cuatro instrucciones más que sí modifican los flags. Sólo las citaremos de pasada porque en la práctica no se utilizan nunca:

```
LA   A, R
LD   R, A
LD   A, I
LD   I, A
```

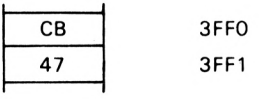
¡Es mejor que se olvide de que existen estas dos instrucciones!

7.2. INSTRUCCIONES DE CARGA 16 BITS

El Z 80 puede agrupar dos registros de 8 bits para formar un solo registro de 16 bits, pero también cuenta con registros de 16 bits (IX, IY, SP). Existen instrucciones de carga para estos registros de 16 bits. Dado que la memoria está organizada en palabras de 8 bits, este tipo de carga precisa de dos octetos de memoria. Una palabra de 16 bits de la memoria central contiene el octeto de menor peso en la dirección de

la palabra y el octeto de mayor peso en la dirección siguiente. Por ejemplo:

47CBH escrito en la dirección 3FF0:



Esto explica por qué la directiva DEFW coloca el octeto débil en primer lugar.

Este conjunto de instrucciones tiene una estructura idéntica al de carga de 8 bits. Diferenciaremos:

cargar registro de 16 bits : memoria ⟶ registro
almacenar registro de 16 bits : registro ⟶ memoria
transferir : registro ⟶ registro

CARGAR REGISTRO 16 BITS

Instrucción	*Modo de direccionamiento del destino*	*Flags afectados*
LD HL, (nn)	extendido	ninguno
LD BC, (nn)	"	"
LD DE, (nn)	"	"
LD IX, (nn)	"	"
LD IY, (nn)	"	"
LD SP, (nn)	"	"

ALMACENAR REGISTRO 16 BITS

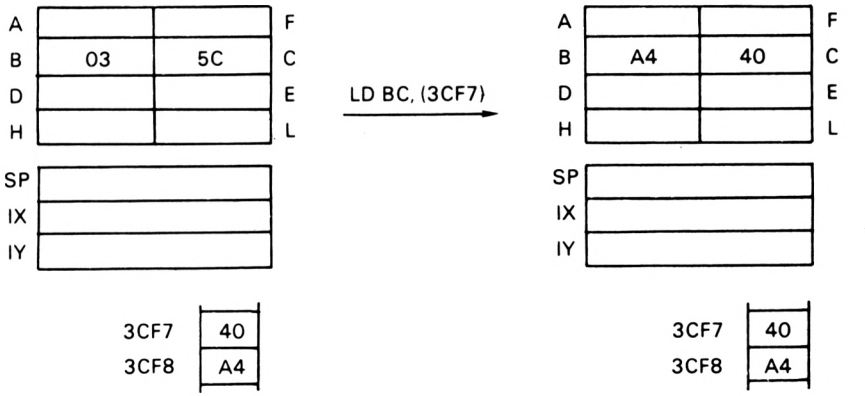
Instrucción	*Modo de direccionamiento del destino*	*Flags afectados*
LD (nn), HL	extendido	ninguno
LD (nn), BC	"	"
LD (nn), DE	"	"
LD (nn), IX	"	"
LD (nn), IY	"	"
LD (nn), SP	"	"

TRANSFERIR:

Instrucción	*Modo de direccionamiento del destino*	*Flags afectados*
LD SP, HL	a través de registro	ninguno
LD SP, IX	"	"
LD SP, IY	"	"

En este caso sólo se puede usar el registro SP (puntero de pila) como destino.

Ejemplo de carga del BC:



El octeto colocado en la dirección 3CF7 se copia en C (octeto débil), y el octeto de la dirección siguiente se copia en B (octeto fuerte).

7. 3. INSTRUCCIONES DE CARGA INMEDIATA

Sería más correcto llamar a este conjunto de instrucciones como "carga de una constante". El dato cargado en un registro o en memoria no es una variable sino una constante impuesta por el programa y que por lo tanto, forma parte de él. Esta constante sólo puede ser leída, es

decir, cargada en un registro o en una celda de memoria (ver direccionamiento inmediato). El esquema de estas instrucciones es el siguiente:

constante ⟶ destino

Su escritura es:

LD destino, constante

La escritura de "constante" corresponde al modo de direccionamiento inmediato. El destino puede ser un registro de 8 bits, un octeto en memoria central o un registro de 16 bits.

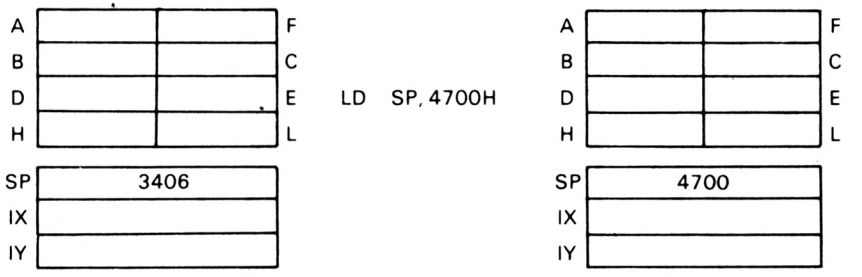
CARGA INMEDIATA 8 BITS

Instrucción	*Modo de direccionamiento del destino*	*Flags afectados*
LD A, n	a través del registro	ninguno
LD B, n	"	"
LD C, n	"	"
LD D, n	"	"
LD E, n	"	"
LD H, n	"	"
LD L, n	"	"
LD (HL), n	indirecto a través de registro	"
LD (IX + d), n	indexado con IX	"
LD (IY + d), n	indexado con IY	"

CARGA INMEDIATA 16 BITS

Instrucción	*Modo de direccionamiento del destino*	*Flags afectados*
LD BC, nn	a través de registro	ninguno
LD DE, nn	"	"
LD HL, nn	"	"
LD SP, nn	"	"
LD IX, nn	"	"
LD IY, nn	"	"

La inicialización del SP, por ejemplo, utiliza una carga inmediata:



Ya hemos terminado con el LD. Recordemos una vez más que el LD sólo copia un dato sin modificarlo. Es una instrucción fundamental; estableciendo un paralelismo con el BASIC, el LD sería el equivalente de LET — = —.

LD destino, origen
LET destino = origen

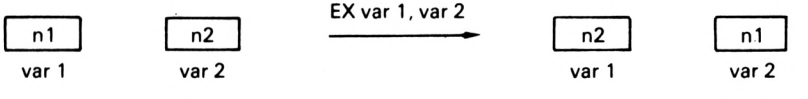
Siéntese ahora adelante de su teclado y pruebe estos LD; así se familiarizará con los modos de direccionamiento analizando en cada caso el camino que sigue el dato.

7.4. INSTRUCCION DE INTERCAMBIO

Estas instrucciones efectúan un desplazamiento de datos pero, al contrario que con el LD, el desplazamiento es bidireccional; ya no se puede hablar de origen y destino sino de dos variables que intercambian sus contenidos. La escritura de estas instrucciones es:

EX variable1, variable2

El esquema es el siguiente:



Las instrucciones de carga son:

Instrucción	*Modo de direccionamiento de var 1*	*Flags afectados*
EX DE, HL	par registre	ninguno
EX AF, AF'	"	F intercambiado con F'
EX (SP), HL	indirecto a través de registro	ninguno
EX (SP), IX	"	"
EX (SP), IY	"	"

¿Recuerda Vd. los dos juegos de registros del Z 80?

Ahora tenemos una instrucción para intercambiar A y F con A' y F'.

La otra instrucción de intercambio entre los dos juegos de registros de uso general es:

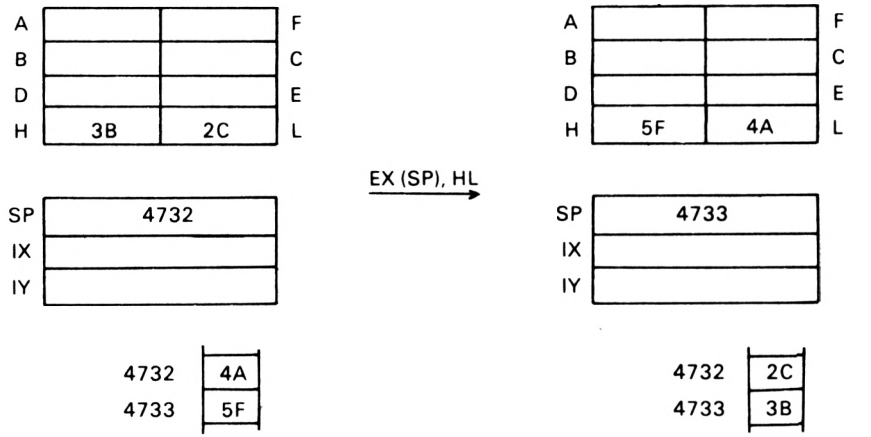
EXX

que intercambia:

BC con B'C'
DE con D'C'
HL con H'L'

Para volver a obtener nuestra imagen, hacer girar la pizarra 180° se escribe EXX.

Veamos un ejemplo con EX (SP), HL.



Una palabra de 16 bits se almacena siempre con el octeto de menor peso en primer lugar.

7.5. INSTRUCCIONES ARITMETICAS 8 BITS

La aritmética del Z 80 es pobre, ya que sólo consta de la suma y la resta. El Z 80 efectúa estas operaciones sobre palabras de 8 bits. En este caso el indicador P/V indica un desbordamiento y no la paridad.

Todas las operaciones utilizan el acumulador, es decir, que el esquema de una operación aritmética es:

$$A = A \Delta \text{ origen}$$

Δ representa una operación que el Z 80 puede ejecutar
"origen" está definido por el operando.

La escritura de una operación aritmética es como sigue:

```
OPR A, origen
```

OPR es un nemónico ficticio para designar la suma o la resta. El origen está definido por uno de los modos de direccionamiento disponibles, exactamente igual que con las instrucciones de carga.

SUMA – 8 BITS: A = A + 0

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
ADD A, γ*	a través de registro	S, Z, H, V, C, N = 0
ADD A, n	inmediato	"
ADD A, (HL)	indirecto a través de registro	"
ADD A, (IX + d)	indexado con IX	"
ADD A, (IY + d)	indexado con IY	"

* la letra γ representa uno de los registros A, B, C, D, E, H, L.

SUMA – 8 BITS CON ACARREO: A = A + o + CY

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
ADC A, γ	a través de registro	S, Z, H, V, C, N = 0
ADC A, n	inmediato	"
ADC A, (HL)	indirecto a través de registro	"
ADC A, (IX + d)	indexado con IX	"
ADC A, (IY + d)	indexado con IY	"

El bit contenido en el flag CY es sumado, con esto se consigue propagar el acarreo en las sumas sucesivas de varios octetos.

RESTA – 8 BITS: A = A – o

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
SUB A, γ	a través de registro	S, Z, H, V, C, N = 1
SUB A, n	inmediato	"
SUB A, (HL)	indirecto a través de registro	"
SUB A, (IX + d)	indexado con IX	"
SUB A, (IY + d)	indexado con IY	"

RESTA – 8 BITS CON ACARREO: A = A – o – CY

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
SBC A, γ	a través de registro	S, Z, H, V, C, N = 1
SBC A, n	inmediato	"
SBC A, (HL)	indirecto a través de registro	"
SBC A, (IX + d)	indexado con IX	"
SBC A, (IY + d)	indexado con IY	"

INCREMENTO – 8 BITS: o = o + 1

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
INC γ	a través de registro	S, Z, H, V, N = 0
INC (HL)	indirecto a través de registro	"
INC (IX + d)	indexado con IX	"
INC (IY + d)	indexado con IY	"

Esta serie de instrucciones no utiliza el acumulador, de ahí que su sintaxis sea algo diferente, no obstante, podemos considerar a esta instrucción como un compendio de tres instrucciones que sí utilizan el acumulador:

```
INC  (HL)  ⇔  LD   A, (HL)
              ADD  A, 1
              LD   (HL), A
```

La única diferencia estriba en que la instrucción INC no modifica el contenido del acumulador (excepto INC A).

DECREMENTO – 8 BITS: o = o – 1

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
DEC γ	a través de registro	S, Z, H, V, N = 1
DEC (HL)	indirecto a través de registro	"
DEC (IX + d)	indexado con IX	"
DEC (IY + d)	indexado con IY	"

COMPARACION: A – o

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
CP γ	a través de registro	S, Z, H, V, N = 1, C
CP n	inmediato	"
CP (HL)	indirecto a través de registro	"
CP (IX + d)	indexado con IX	"
CP (IX + d)	indexado con IY	"

La comparación es una resta en la que no se conserva el resultado. Sólo se posicionan los flags correspondientes.

CY	= 0	A >=	operando
CY	= 1	A <	operando
Z	= 0	A ≠	operando
Z	= 1	A =	operando

OPUESTO A = – A

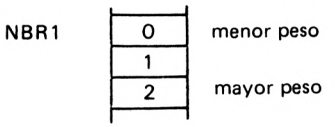
Instrucción	*Modo de direccionamiento*	*Flags afectados*
NEG	implícito	S, Z, H, V, N = 1, C

Esta instrucción calcula el opuesto de un octeto con la convención citada al comienzo del libro para representar los números negativos.

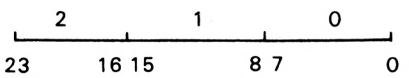
SUMA DE DOS NUMEROS CODIFICADOS EN 24 BITS

```
LD    IX, NBR1      carga de las direcciones
LD    IY, NBR2      en los punteros
                    ---------------------------
LD    A, (IX + 0)   suma de los pesos débiles
ADD   A, (IY + 0)   sin tener en cuenta el acarreo
LD    (IY + 0), A   almacenamiento del resultado
                    ---------------------------
LD    A, (IX + 1)   suma de los pesos fuertes
ADC   A, (IY + 1)   propagando el acarreo
LD    (IY + 1), A
                    ---------------------------
LD    A, (IX + 2)
ADC   A, (IY + 2)
LD    (IY + 2), A
```

Este pequeño programa permite mostrar la diferencia entre las instrucciones ADD y ADC. Las variables se han codificado en 24 bits, ocupan, por lo tanto, 3 octetos colocados de la siguiente forma:



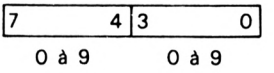
El número se lee así:



La suma debe hacerse empezando por el octeto de peso débil (0) continuando con los octetos (1) y (2) propagando el acarreo. Esto explica la elección de la instrucción ADD para el octeto (0): no hay que tener en cuenta el bit C para esta primera suma ya que un eventual acarreo falsearía el resultado. A continuación el acarreo se propaga con la instrucción ADC.

7.6. INSTRUCCION DE AJUSTE DECIMAL

Esta instrucción posibilita la aritmética decimal con el Z 80. El octeto se divide en dos palabras de 4 bits. Cada palabra codifica un dígito de 0 a 9 con lo cual se puede contar de 0 a 99 con un solo octeto.



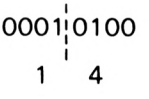
codificación BCD

Esta representación no es compatible con la aritmética binaria del Z 80. La instrucción de ajuste decimal (DAA) permite el paso de la convención binaria a la BCD:

Supongamos que sumanos 8 y 6:

8:	00001000	=	0 8	
6:	00000110	=	0 6	
14:	00001110	=	0 EH	código inválido en BCD

El resultado es OEH. Si se aplica la instrucción DAA a este resultado se obtiene:



Se obtiene el resultado codificado en BCD.

La instrucción DAA funciona únicamente sobre el acumulador (direccionamiento implícito).

Cuando los bits 0 a 3 tienen un valor superior a 9 o el bit H es 1, DAA suma 6 al acumulador. A continuación si los bits 4 a 7 tienen un valor superior a 9 o el bit C es 1, se suma 6 a estos 4 bits.

Instrucción	*Modo de direccionamiento*	*Flags afectados*
DAA	implícito	S, Z, H, P, C

7.7. INSTRUCCIONES LOGICAS 8 BITS

El conjunto de estas instrucciones es similar a las instrucciones aritméticas. Las operaciones efectuadas forman parte de una "aritmética" especial aplicable solamente al sistema binario (álgebra de Boole).

En este caso el indicador P/V indica la paridad del resultado. Todas estas instrucciones utilizan el acumulador y se escriben:

OPR A, origen

Antes de presentarle el conjunto de instrucciones recordemos las operaciones lógicas. Para este conjunto de instrucciones consideraremos al octeto como un conjunto de 8 bits independientes. Las operaciones lógicas se realizan entre dos bits del mismo peso.

Y lógico (notación Λ)

$0 \Lambda 0 = 0$
$0 \Lambda 1 = 0$
$1 \Lambda 0 = 0$
$1 \Lambda 1 = 1$

$$\begin{array}{r} 01001110 \\ 00111010 \\ \hline 00001010 \end{array}$$

O lógico (notación V)

$0 V 0 = 0$
$0 V 1 = 1$
$1 V 0 = 1$
$1 V 1 = 1$

$$\begin{array}{r} 01001110 \\ 00111010 \\ \hline 01111110 \end{array}$$

O exclusivo lógico (notación $\oplus$)

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0$$

$$\begin{array}{r} 01001110 \\ \underline{00111010} \\ 01110100 \end{array}$$

NO lógico (notación $^{-}$)

$$\overline{1} = 0$$
$$\overline{0} = 1$$

$$\overline{01001110} = 10110001$$

Observe que:

$$A \oplus B = (\overline{A} \wedge \overline{B}) \vee (A \wedge \overline{B})$$

Y LOGICO: A = A ∧ o

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
AND A, γ	a través de registro.	S, Z, H = 1, P, N = 0, C = 0
AND A, n	inmediato	"
AND A, (HL)	indirecto a través de registro	"
AND A, (IX + d)	indexado con IX	"
AND A, (IY + d)	indexado con IY	"

O LOGICO: A = A ∨ o

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
OR A, γ	a través de registro	S, Z, H = 0, P, N = 0, C = 0
OR A, n	inmediato	"
OR A, (HL)	indirecto a través de registro	"
OR A, (IX + d)	indexado con IX	"
OR A, (IY + d)	indexado con IY	"

O EXCLUSIVO LOGICO: A = A ⊕ o

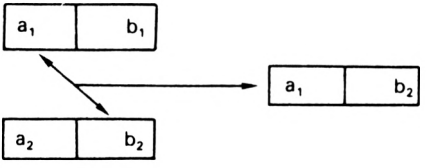
Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
XOR A, γ	a través de registro	S, Z, H = 0, P, N = 0, C = 0
XOR A, n	inmediato	"
XOR A, (HL)	indirecto a través de registro	"
XOR A, (IX + d)	indexado con IX	"
XOR A, (IY + d)	indexado con IY	"

NO LOGICO: A = $\bar{A}$

Instrucción	*Modo de direccionamiento*	*Flags afectados*
CPL	implícito	H = 1, N = 1

Esta instrucción sólo se realiza sobre el acumulador.

CONCATENACION DE DOS PALABRAS DE 4 BITS




```
LD      BC, VAR 1     carga de las direcciones
LD      DE, VAR 2     de las dos variables
LD      HL, VAR 3     dirección del resultado
-------------------------------------------------
LD      A, (BC)       máscara para los 4 bits con menor peso
AND     A, OFOH       de VAR1
LD      (HL), A       almacenamiento provisional
-------------------------------------------------
LD      A, (DE)       máscara para los 4 bits de mayor peso
AND     A, OFH        de VAR2
OR      A, (HL)       unión de los dos semi-octetos
-------------------------------------------------
LD      (HL), A       almacenamiento definitivo del resultado
```

Este pequeño problema permite crear un octeto a partir de dos medios octetos. La utilización del AND permite enmascarar los bits, mientras que el OR, por el contrario, permite poner a 1 determinados bits inicialmente puestos a 0.

Las instrucciones AND y OR utilizan aquí un direccionamiento indirecto a través de registro; hay pues que inicializar los registros BC, DE y HL con las direcciones de 3 variables tratadas por el programa.

7.8. INSTRUCCIONES ARITMETICAS 16 BITS

Algunas instrucciones aritméticas del Z 80 se pueden ejecutar utilizando los registros de 16 bits o los pares de registros de 8 bits. Los operandos tienen entonces una capacidad de 16 bits. Este conjunto de instrucciones se asemeja a las instrucciones aritméticas de 8 bits. El indicador P/V indica un desbordamiento.

El acumulador no es utilizado por estas instrucciones, ya que sólo tiene 8 bits, es decir, que la aritmética de 16 bits define los dos operandos: origen y destino.

OPR origen, destino

SUMA – 16 BITS: PP = PP + OO

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
ADD HL, BC	a través de registro	N = 0, C
ADD HL, DE	"	"
ADD HL, HL	"	"
ADD HL, SP	"	"
ADD IX, BC	"	"
ADD IX, DE	"	"
ADD IX, IX	"	"
ADD IX, SP	"	"
ADD IY, BC	"	"
ADD IY, DE	"	"
ADD IY, IY	"	"
ADD IY, SP	"	

SUMA – 16 BITS CON ACARREO: HL = HL + oo + CY

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
ADC HL, BC	a través de registro	S, Z, V, N = 0, C
ADC HL, DE	"	"
ADC HL, HL	"	"
ADC HL, SP	"	"

RESTA – 16 BITS CON ACARREO: HL = HL – oo – CY

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
SBC HL, BC	a través de registro	S, Z, V, N = 1, C
SBC HL, DE	"	"
SBC HL, HL	"	"
SBC HL, SP	"	"

INCREMENTO – 16 BITS: oo = oo + 1

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
INC BC	a través de registro	ninguno
INC DE	"	"
INC HL	"	"
INC SP	"	"
INC IX	"	"
INC IY	"	"

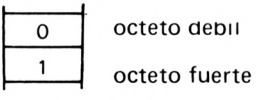
DECREMENTO – 16 BITS:oo=oo– 1

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
DEC BC	a través de registro	ninguno
DEC DE	"	"
DEC HL	"	"
DEC SP	"	"
DEC IX	"	"
DEC IY	"	"

RESTA – 16 BITS

```
LD   HL, (VAR1)      inicialización
LD   DE, (VAR1)      de los registros
OR   A, A            pone el flag C a 0
SBC  HL, DE          resta: VAR1 – VAR2
LD   (RESULT), HL    almacenamiento del resultado
```

Las operaciones de 16 bits sólo se realizan entre registros. Es pues necesario cargar previamente estos registros con el contenido de las variables de la memoria central. Estas variables codificadas en 16 bits se almacenan así:



Es interesante resaltar que la instrucción OR A, A no afecta al acumulador y pone a cero el bit C.

7.9. INSTRUCCIONES DE SALTO

Estas instrucciones le son ya familiares porque son el equivalente de los GOTO e IF THEN GOTO del BASIC. Como ya hemos dicho, estas instrucciones van a formar el esqueleto del programa, es pues importante conocer las posibilidades que ofrece este conjunto de instrucciones.

Estas instrucciones utilizan tres tipos de bifurcación que ya hemos descrito:

- relativo,
- indirecto,
- página cero.

El salto puede ser condicional (equivalente de IF THEN GOTO) o incondicional (GOTO). Los saltos condicionales utilizan la posición de los flags para decidir si la bifurcación se debe realizar o no. Hay ocho condiciones posibles identificadas por un nemónico:

NZ	flag Z = 0	resultado no nulo
N	flag Z = 1	resultado nulo
NC	flag C = 0	no hay acarreo
C	flag C = 1	hay acarreo
PO	flag P/V = 0	paridad impar
PE	flag P/V = 1	paridad par
P	flag S = 0	signo positivo
M	flag S = 1	signo negativo

La sintaxis de una bifurcación condicional es la siguiente:

```
SALTO  CONDICION, ETIQUETA
```

SALTO puede ser una bifurcación directa o una bifurcación relativa. La diferencia entre las dos es:

directa	instrucción 3 octetos bifurcación sobre toda la zona de memoria
relativa	instrucción 2 octetos bifurcación en un espacio limitado a 256 direcciones: + 127, −128

CONDICION es uno de los nemónicos citados.

ETIQUETA es una de las etiquetas de bifurcación del programa situada en la zona de etiquetas o definida por una directiva.

BIFURCACION INMEDIATA

Instrucción	*Condición*
JP ETIQ	incondicional
JP NZ, ETIQ	flag Z = 0
JP Z, ETIQ	flag Z = 1
JP NC, ETIQ	flag C = 0
JP C, ETIQ	flag C = 1
JP PO, ETIQ	flag P/V = 0
JP PE, ETIQ	flag P/V = 1
JP P, ETIQ	flag S = 0
JP M, ETIQ	flag S = 1

BIFURCACION RELATIVA

Instrucción	*Condición*
JR ETIQ	incondicional
JR C, ETIQ	flag C = 1
JR NC, ETIQ	flag C = 0
JR Z, ETIQ	flag Z = 1
JR NZ, ETIQ	flag Z = 0

BIFURCACION INDIRECTA

La dirección de bifurcación es el contenido de uno de los registros de 16 bits HL, IX o IY:

Instrucción	*Condición*
JP (HL)	incondicional
JP (IX)	"
JP (IX)	"

BIFURCACION Y DECREMENTO:

El Z80 tiene una instrucción especial para hacer bucles. Esta instrucción es la asociación de un salto y de un decremento del registro B. Se la puede resumir así:

– decremento de B.
– si B = 0 continuar
si no, bifurcar a la etiqueta especificada

La bifurcación es relativa y se efectúa en un entorno de 256 direcciones.

La instrucción que efectúa esto es:

```
DJNZ  ETIQ
```

Esta instrucción nos recuerda el bucle FOR ... NEXT del BASIC:

```
FOR  I = 1 TO 10                    LD     B, 10
     ¦                      BUCLE    ¦
     ¦                               ¦
     ¦                               ¦
NEXT ¦                      DJNZ  BUCLE
```

en ambos casos la secuencia se ejecuta 10 veces.

La versión ensamblador utiliza el registro B como contador de bucle.

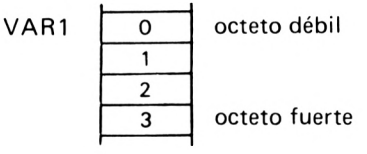
```
50 IF VAR1 = VAR2 THEN GOTO 60 ELSE VAR1 = VAR1 + 1
60 . . .

   LD   A, (VAR1)        carga del contenido de VAR!
   LD   HL, VAR2         carga de la dirección de VAR2
   CP   (HL)             IF; comparación
   JR   Z, CONTIN        THEN: test positivo
   INC  A                ELSE: test negativo
   LD   (VAR1), A

CONTIN
```

El empleo de la instrucción CP asociada a las bifurcaciones condicionales permite establecer el equivalente en ensamblador de la instrucción BASIC IF ... THEN ... ELSE.

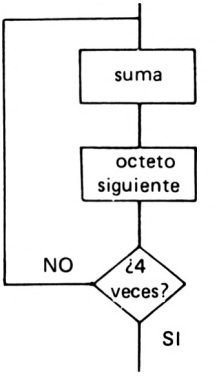
SUMA DE DOS NUMEROS CODIFICADOS EN 32 BITS

```
        LD    B, 4          hacer 4 veces el bucle
        LD    DE, VAR1      carga de las direcciones
        LD    HL, VAR2
        OR    A, A          se pone el flag C a 0
BUCLE   LD    A, (DE)       suma de los octetos
        ADC   A, (HL)
        LD    (HL), A       almacenar el resultado
        INC   HL            octeto siguiente
        INC   DE
        DJNZ  BUCLE         volver a empezar 4 veces
```

Este programa efectúa la suma de dos números codificados en 32 bits:



Contrariamente al ejemplo anterior, la suma sucesiva de los octetos se hace mediante un bucle:



El contador de bucle es el registro B cargado con el valor 4. Esto permite utilizar la instrucción DJNZ.

7.10. RUTINAS

7.10.1. Llamadas a las rutinas

Estas instrucciones funcionan exactamente como el GOSUB del BASIC. Existen, por lo tanto, dos instrucciones emparejadas:

```
llamada:  CALL   (GOSUB)
retorno:  RET    (RETURN)
```

La llamada a las rutinas puede ser condicional (IF ... THEN GOSUB). En este caso las condiciones son las mismas que para las instrucciones de salto.

Así mismo el retorno de la rutina puede ser también condicional (IF ... THEN RETURN).

La sintaxis de estas instrucciones es:

```
CALL    CONDICION, ETIQUETA      llamada
RET     CONDICION                retorno
```

CONDICION es uno de los nemónicos citados en las instrucciones de salto.

ETIQUETA es el label asignado a la rutina de llamada.

Instrucción	*Condición*
CALL ETIQ	incondicional
CALL NZ, ETIQ	flag Z = 0
CALL Z, ETIQ	flag Z = 1
CALL NC, ETIQ	flag C = 0
CALL C, ETIQ	flag C = 1
CALLPO, ETIQ	flag P/V = 0
CALL PE, ETIQ	flag P/V = 1
CALL P, ETIQ	flag S = 0
CALL M, ETIQ	flag S = 1

7.10.2. Instrucciones de retorno de la rutina

Instrucción	*Condición*
RET	incondicional
RET NZ	flag Z = 0
RET Z	flag Z = 1
RET NC	flag C = 0
RET C	flag C = 1
RET PO	flag P/V = 0
RET PE	flag P/V = 1
RET P	flag S = 0
RET M	flag S = 1

LLAMADA A RUTINAS EN PAGINA CERO

El Z 80 dispone de una instrucción especial llamada RESTART que permite efectuar una bifurcación a una rutina residente en una dirección especial de la página cero. El interés de este RESTART reside en que utiliza un solo octeto en lugar de los tres que necesita el CALL. Hay ocho direcciones posibles de bifurcaciones en la página cero:

00H, 08H, 10H, 18H, 20H, 28H, 30H, 38H

La sintaxis es la siguiente:

RST p

el resultado es el mismo que con CALL p.

Instrucción	*Condición*
RST 00H	incondicional
RST 08H	"
RST 10H	"
RST 18H	"
RST 20H	"
RST 28H	"
RST 30H	"
RST 38H	"

¡¡Atención!!

Las instrucciones RST p son frecuentemente utilizadas por el Sistema Operativo siendo por consiguiente delicadas de manejar.

LLAMADA A RUTINAS DE SUMAR 32 BITS

```
        LD    DE, VAR1      carga de las direcciones
        LD    HL, VAR2      de las variables a sumar
        CALL  SUMA          llamada a la rutina de sumar

SUMA    OR    A, A          rutina de sumar
        LD    B, 4
BUCLE   LD    A, (DE)
        ADC   A, (HL)
        LD    (HL), A
        INC   HL
        INC   DE
        DJNZ  BUCLE
        RET
```

El hecho de utilizar una rutina de suma permite tratar varias variables con el mismo programa de suma. En el ejemplo anterior esto no era posible, había que repetir cada vez la secuencia de la suma. Ahora basta cargar los dos registros DE y HL con las direcciones de las dos variables que deseamos sumar y llamar a la rutina SUMA:

```
LD    DE, X1      son necesarias 3 instrucciones
LD    HL, X2      para realizar la suma
CALL  SUMA        mediante la rutina
```

7.11. INSTRUCCIONES PARA MANIPULAR LA PILA

No insistiremos aquí sobre el funcionamiento de la pila que ya hemos visto. El Z 80 posee un conjunto de instrucciones que le permite colocar y retirar sus registros de la pila. Esta salvaguarda en la pila se hace siempre por pares de registros o por registro de 16 bits.

La acción de apilar un registro se llama PUSH; la acción de desempilar se llama POP.

Instrucción		*Modo de direccionamiento del origen*	*Flags afectados*
PUSH	BC	a través de registro	ninguno
PUSH	DE	"	"
PUSH	HL	"	"
PUSH	AF	"	"
PUSH	IX	"	"
PUSH	IY	"	"
POP	BC	"	"
POP	DE	"	"
POP	HL	"	"
POP	AF	"	"
POP	IX	"	"
POP	IY	"	"

LLAMADA A RUTINAS CONDICIONAL SALVAGUARDA DE CONTEXTO:

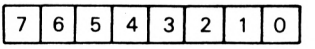
```
        LD      DE, VAR1
        LD      HL, VAR2
        CALL    SUMA

SUMA    PUSH AF,                salvaguarda de los
        PUSH    BC              registros AF y BC
        OR      A, A
        LD      B, 4
BUCLE   LD      A, (DE)
        ADC     A, (HL)
        LD      (HL), A
        INC     HL
        INC     DE
        DJNZ    BUCLE
        POP     BC              restauración de los registros
        POP     AF              AF y BC
        RET
```

Este ejemplo es idéntico al anterior, pero ahora cuando se retorna de la suma los registros AF y BC no están modificados. Observe el orden de los PUSH y POP. Hay que respetar la regla: El primer PUSH corresponde al último POP.

7.12. INSTRUCCIONES SOBRE LOS BITS SET y RESET

Hasta ahora todas las instrucciones que hemos presentado trabajan sobre octetos (8 bits). El conjunto de instrucciones SET y RESET permite manipular un solo bit de un registro o de una celda de memoria. Las operaciones que se pueden ejecutar sobre un bit son su puesta a 1 (SET) o su puesta a 0 (RESET) o su comprobación (test).

El bit sobre el cual opera la instrucción viene identificado por un número de 0 a 7:



La sintaxis de estas instrucciones es:

OPR número de bit, origen

TEST DE BIT: Z = $\overline{\text{BIT COMPROBADO}}$

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
* BIT b, r	a través de registro	Z, H = 1, N = 0
BIT b, (HL)	indirecto a través de registro	"
BIT b, (IX + d)	indexado con IX	"
BIT b, (IY + d)	indexado con IY	"

(* b = número de 0 a 7, por ejemplo BIT 6, C)

El test de bif afecta únicamente al flag Z de forma significativa:

Z = 0 si el bit comprobado = 1
Z = 1 si el bit comprobado = 0

PUESTA A 1 DEL BIT: BIT = 1

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
SET b, γ	a través de registro	ninguno
SET b, (HL)	indirecto a través de registro	"
SET b, (IX + d)	indexado con IX	"
SET b, (IY + d)	indexado con IY	"

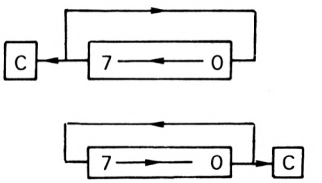
PUESTA A 0 DEL BIT: BIT = 0

Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
RES b, γ	a través de registro	ninguno
RES b, (HL)	indirecto a través de registro	"
RES b, (IX + d)	indexado con IX	"
RES b, (IY + d)	indexado con IY	"

7.13. INSTRUCCIONES DE DESPLAZAMIENTO

Hay un gran número de desplazamientos posibles. Resulta difícil no perderse entre ellos. La mayoría de estos desplazamientos utilizan el bit de acarreo C que por este motivo puede ser considerado como un noveno bit participando en el desplazamiento.

7.13.1. Rotación circular



a la izquierda RLC

a la derecha RRC

El bit que sale por un lado entra por el otro y se copia en el flag C.

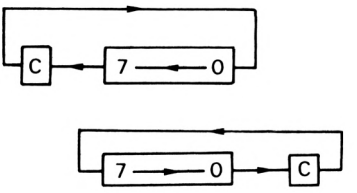
Instrucción		*Modo de direccionamiento del origen*	*Flags afectados*
RLC	r	a través de registro	S, Z, H = 0, P, N = 0, C
RLC	(HL)	indirecto a través de registro	"
RLC	(IX + d)	indexado con IX	"
RLC	(IY + d)	indexado con IY	"
RRC	r	a través de registro	"
RRC	(HL)	indirecto a través de registro	"
RRC	(IX + d)	indexado con IX	"
RRC	(IY + d)	indexado con IY	"

Existen dos instrucciones distintas que producen el mismo efecto. Se trata de rotaciones circulares del acumulador:

RLC A = RLCA
RRC A = RRCA

Esto es una reminiscencia del juego de instrucciones del 8080. Las instrucciones RRCA y RLCA ocupan un solo octeto y no afectan a los flags S, Z y P. Las instrucciones RRC A y RLC A ocupan dos octetos.

7.13.2. Rotación circular a través del acarreo



a la izquierda RL

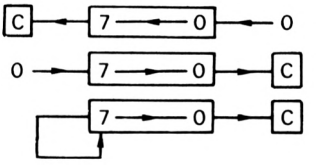
a la derecha RR

El bit que sale se vuelve a copiar en el indicador C. El bit contenido en el indicador C entra en el octeto.

Instrucción		*Modo de direccionamiento del origen*	*Flags afectados*
RL	γ	a través de registro	S, Z, H = 0, P, N = 0, C
RL	(HL)	indirecto a través de registro	"
RL	(IX + d)	indexado con IX	"
RL	(IY + d)	indexado con IY	"
RR	γ	a través de registro	"
RR	(HL)	indirecto a través de registro	"
RR	(IX + d)	indexado con IX	"
RR	(IY + d)	indexado con IY	"

También existen las dos instrucciones del 8080 RLA Y RRA idénticas a RL A y RR A con la salvedad de que no afectan a los flags S, Z y P y que sólo ocupan un octeto.

7.13.3. Desplazamientos lógicos y aritméticos.



En el desplazamiento aritmético a la derecha, el bit 7 vuelve a copiarse sobre él mismo sin ponerse a 0. El término aritmético proviene del hecho de que este desplazamiento conserva el signo del octeto (bit 7).

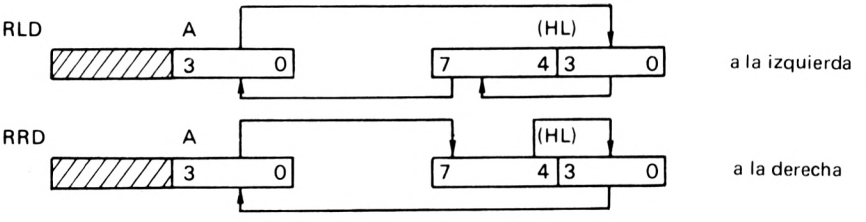
Instrucción		*Modo de direccionamiento del origen*	*Flags afectados*
SLA	γ	a través de registro	S, Z, H = 0, P, N = 0, C
SLA	(HL)	indirecto a través de registro	"
SLA	(IX + d)	indexado con IX	"
SLA	(IX + d)	indexado con IY	"
SRL	γ	a través de registro	"
SRL	(HL)	indirecto a través de registro	"
SRL	(IX + d)	indexado con IX	"

Instruction		*Modo de direccionamiento del origen*	*Flags affectés*
SRL	(IY + d)	indexado con IY	"
SRA	γ	a través de registro	"
SRA	(HL)	indirecto a través de registro	"
SRA	(IX + d)	indexado con IX	"
SRA	(IY + d)	indexado con IY	"

7.13.4. Rotación circular BCD

Esta rotación está adaptada a la aritmética decimal. En aritmética decimal el octeto está dividido en dos palabras de 4 bits. Cada una de estas palabras representa el código binario de los dígitos 0 a 9. Las rotaciones binarias operan, por lo tanto, con palabras de 4 bits.

El destino es el acumulador; la segunda palabra origen viene designada por un direccionamiento indirecto a través de registro:

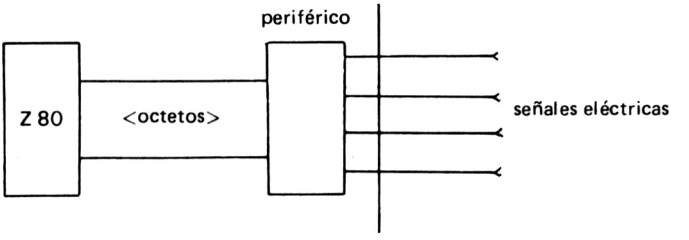


Instrucción	*Modo de direccionamiento del origen*	*Flags afectados*
RLD	indirecto a través del registro HL	S, Z, H = 0, P, N = 0, C
RRD	indirecto a través del registro HL	"

7.14. INSTRUCCIONES DE ENTRADA–SALIDA

El Z 80 posee dos instrucciones que le permiten dialogar con los periféricos: IN y OUT. Un periférico es un componente electrónico capaz

de preparar los octetos enviados por el microprocesador para ser enviados al exterior o de recibir las señales eléctricas externas transformándolas en octetos accesibles por el Z 80.



De esta forma el Z 80 se puede comunicar con el exterior, pudiendo accionar una impresora, gestionar un magnetofón de casetes o enviar mandatos a cualquier máquina. Con estos periféricos el microprocesador puede efectuar varias tareas diferentes; cada tarea tiene su correspondiente periférico de entrada-salida. El Z 80 organiza todos estos "brazos" extendidos hacia el exterior asignando a cada uno de ellos una dirección, exactamente igual que con la memoria central. Esta dirección está contenida en un octeto, lo cual proporciona 256 "brazos" posibles al microprocesador.

No hay que confundir una dirección de periférico con una dirección de memoria central. Ambas son completamente distintas. El diálogo microprocesador – periférico se limita a la transferencia de octetos. En el Z 80 hay dos modos de direccionamiento para la comunicación con sus periféricos:

– directo
– indirecto a través del registro C.

Las dos instrucciones son:

IN transferir un octeto desde el periférico al Z 80

OUT transferir un octeto desde el Z 80 al periférico

En el direccionamiento directo, la transferencia se realiza entre el acumulador A y el periférico. El operando es la dirección del periférico llamado. En el direccionamiento indirecto la transferencia se realiza entre uno de los registros A, B, C, D, E, H, L y el periférico.

Instrucción		*Modo de direccionamiento de los periféricos*	*Flags afectados*
IN	A, (n)	directo	ninguno
IN	γ, (C)	indirecto a través del registro C	S, Z, H, P, N = 0
OUT	(n), A	directo	ninguno
OUT	(C), γ	indirecto a través del registro C	S, Z, H, P, N = 0

por ejemplo:

```
IN A, (DIRP)
```

Se ha cargado el acumulador con un octeto enviado por el periférico cuya dirección, definida por 8 bits, es DIRP.

7.15. INSTRUCCIONES DE CADENAS

Estas instrucciones del Z 80 son muy potentes y trabajan en un espacio de memoria de varios octetos contiguos. Se dividen en tres grupos:

- instrucciones de transferencia de cadena,
- instrucciones de búsqueda de un octeto en una cadena,
- instrucciones de entrada–salida de cadena.

7.15.1. Transferencia de cadena.

$$\text{LDI}: \begin{cases} (DE) = (HL) \\ DE = DE + 1 \\ HL = HL + 1 \\ BC = BC - 1 \end{cases}$$

La casilla de memoria cuya dirección está contenida en HL se copia en la dirección contenida en DE: a continuación se incrementan los dos pares de registros DE y HL. El par de registros BC se decrementa.

LDIR :
- (DE) = (HL)
- DE = DE + 1
- HL = HL + 1
- BC = BC − 1

La instrucción se repite hasta que BC = 0

La letra R que hay detrás de LDI significa repetición. Así pues, las instrucciones que terminan en R son instrucciones que operan sobre una cadena de octetos.

LDD :
- (DE) = (HL)
- DE = DE − 1
- HL = HL − 1
- BC = BC − 1

LDDR :
- (DE) (HL)
- DE = DE − 1
- HL = HL − 1
- BC = BC − 1

La instrucción se repite hasta que BC = 0

7.15.2. Búsqueda de un octeto en una cadena

CPI :
- A − (HL)
- HL = HL + 1
- BC = BC − 1

El Z 80 realiza una comparación entre el contenido del acumulador y la celda de memoria (HL).

CPIR :
- A − (HL)
- HL = HL + 1
- BC = BC − 1

La instrucción se repite hasta que BC = 0 o A = (HL)

CPD :
$$\begin{cases} A - (HL) \\ HL = HL - 1 \\ BC = BC - 1 \end{cases}$$

CPDR :
$$\begin{cases} A - (HL) \\ HL = HL - 1 \\ BC = BC - 1 \end{cases}$$

La instrucción se repite hasta que BC = 0

7.15.3. Entrada–salida de cadena.

INI :
$$\begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL + 1 \end{cases}$$

INIR :
$$\begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL + 1 \end{cases}$$

La instrucción se repite hasta que B = 0

IND :
$$\begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL - 1 \end{cases}$$

INDR :
$$\begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL - 1 \end{cases}$$

La instrucción se repite hasta que B = 0

OUTI :
$$\begin{cases} (C) = (HL) \\ HL = HL + 1 \\ B = B - 1 \end{cases}$$

OTIR :
$$\begin{cases} (C) = (HL) \\ HL = HL + 1 \\ B = B - 1 \end{cases}$$

La instrucción se repite hasta que B = 0

OUTD : $\begin{cases} (C) = (HL) \\ HL = HL - 1 \\ B = B - 1 \end{cases}$

OTDR : $\begin{cases} (C) = (HL) \\ HL = HL - 1 \\ B = B - 1 \end{cases}$ La instrucción se repite hasta que $B = 0$

C contiene la dirección del periférico a donde se dirige la transferencia.

Instrucción	*Flags afectados*
LDI	H = 0, N = 0, P/V = 0 si BC − 1 = 0
LDIR	H = 0, P/V = 0, N = 0
LDD	H = 0, N = 0, P/V = 0 si BC − 1 = 0
LDDR	H = 0, P/V = 0, N = 0
CPI	S, H, N = 1, Z = 1 si A = (HL), P/V = 0 si BC − 1 = 0
CPIR	"
CPD	"
CPDR	"
INI	N = 1, Z = 1 si B − 1 = 0
INIR	N = 1, Z = 1
IND	N = 1, Z = 1 si B − 1 = 0
INDR	N = 1, Z = 1
OUTI	N = 1, Z = 1 si B − 1 = 0
OTIR	N = 1, Z = 1
OUTD	N = 1, Z = 1 si B − 1 = 0
OTDR	N = 1, Z = 1

7.16. INSTRUCCIONES DE USO GENERAL

Las instrucciones que damos aquí son de una utilización muy especial y no siempre son fácilmente utilizables. Daremos la relación sin entrar en detalles.

7.17. INSTRUCCIONES PARA LAS INTERRUPCIONES

DI	inhibe las interrupciones
EI	autoriza las interrupciones
IMO	pone al Z 80 en modo de interrupción 0
IM1	pone al Z 80 en modo de interrupción 1
IM2	pone al Z 80 en modo de interrupción 2
RETI	retorno de interrupción
RETN	retorno de interrupción sin máscara

7.18. INSTRUCCIONES DE CONTROL

El Z 80 tiene dos instrucciones que le permiten "descansar".

NOP	no hacer nada
HALT	parada del microprocesador

La instrucción NOP no ejecuta ninguna operación y no modifica absolutamente nada. No por ello deja de ser interesante. Por una parte el NOP permite la supresión de instrucciones directamente del programa objeto sin tener que pasar por un nuevo ensamblaje. Basta reemplazar la instrucción que se desea suprimir por una serie de NOP. Por otra parte, el NOP hace que el Z 80 pueda perder tiempo. Esto puede resultar útil durante el diálogo del Z 80 con periféricos más lentos que él. El NOP hace que el Z 80 pueda esperar un poco, sin hacer nada, a que el periférico le responda.

En cuanto a la instrucción HALT, interrumpe completamente al Z 80. Este no puede rearrancar más que a través de una acción hardware proveniente del exterior: RESET o interrupción. En la práctica HALT no se utiliza nunca más que para configuraciones hardware especiales y éste no es el caso de los micro-ordenadores.

ANEXO 1

Relación de los códigos de operación por orden numérico

0000	00	1	NOP
0001	018405	2	LD BC,NN
0004	02	3	LD (BC),A
0005	03	4	INC BC
0006	04	5	INC B
0007	05	6	DEC B
0008	0620	7	LD B,N
000A	07	8	RLCA
000B	08	9	EX AF,AF'
000C	09	10	ADD HL,BC
000D	0A	11	LD A,(BC)
000E	0B	12	DEC BC
000F	0C	13	INC C
0010	0D	14	DEC C
0011	0E20	15	LD C,N
0013	0F	16	RRCA
0014	102E	17	DJNZ DIS
0016	118405	18	LD DE,NN
0019	12	19	LD (DE),A
001A	13	20	INC DE
001B	14	21	INC D
001C	15	22	DEC D
001D	1620	23	LD D,N
001F	17	24	RLA
0020	182E	25	JR DIS
0022	19	26	ADD HL,DE
0023	1A	27	LD A,(DE)
0024	1B	28	DEC DE
0025	1C	29	INC E
0026	1D	30	DEC E
0027	1E20	31	LD E,N
0029	1F	32	RRA
002A	202E	33	JR NZ,DIS
002C	218405	34	LD HL,NN
002F	228405	35	LD (NN),HL
0032	23	36	INC HL
0033	24	37	INC H
0034	25	38	DEC H
0035	2620	39	LD H,N
0037	27	40	DAA
0038	282E	41	JR Z,DIS
003A	29	42	ADD HL,HL
003B	2A8405	43	LD HL,(NN)
003E	2B	44	DEC HL
003F	2C	45	INC L
0040	2D	46	DEC L
0041	2E20	47	LD L,N
0043	2F	48	CPL
0044	302E	49	JR NC,DIS
0046	318405	50	LD SP,NN
0049	328405	51	LD (NN),A
004C	33	52	INC SP
004D	34	53	INC (HL)
004E	35	54	DEC (HL)
004F	3620	55	LD (HL),N
0051	37	56	SCF
0052	382E	57	JR C,DIS
0054	39	58	ADD HL,SP
0055	3A8405	59	LD A,(NN)
0058	3B	60	DEC SP
0059	3C	61	INC A
005A	3D	62	DEC A
005B	3E20	63	LD A,N
005D	3F	64	CCF
005E	40	65	LD B,B
005F	41	66	LD B,C
0060	42	67	LD B,D
0061	43	68	LD B,E
0062	44	69	LD B,H,NN
0063	45	70	LD B,L
0064	46	71	LD B,(HL)
0065	47	72	LD B,A
0066	48	73	LD C,B
0067	49	74	LD C,C
0068	4A	75	LD C,D
0069	4B	76	LD C,E
006A	4C	77	LD C,H
006B	4D	78	LD C,L
006C	4E	79	LD C,(HL)
006D	4F	80	LD C,A
006E	50	81	LD D,B
006F	51	82	LD D,C
0070	52	83	LD D,D
0071	53	84	LD D,E
0072	54	85	LD D,H
0073	55	86	LD D,L
0074	56	87	LD D,(HL)
0075	57	88	LD D,A
0076	58	89	LD E,B
0077	59	90	LD E,C
0078	5A	91	LD E,D
0079	5B	92	LD E,E
007A	5C	93	LD E,H
007B	5D	94	LD E,L
007C	5E	95	LD E,(HL)
007D	5F	96	LD E,A

007E	60	97	LD H,B
007F	61	98	LD H,C
0080	62	99	LD H,D
0081	63	100	LD H,E
0082	64	101	LD H,H
0083	65	102	LD H,L
0084	66	103	LD H,(HL)
0085	67	104	LD H,A
0086	68	105	LD L,B
0087	69	106	LD L,C
0088	6A	107	LD L,D
0089	6B	108	LD L,E
008A	6C	109	LD L,H
008B	6D	110	LD L,L
008C	6E	111	LD L,(HL)
008D	6F	112	LD L,A
008E	70	113	LD (HL),B
008F	71	114	LD (HL),C
0090	72	115	LD (HL),D
0091	73	116	LD (HL),E
0092	74	117	LD (HL),H
0093	75	118	LD (HL),L
0094	76	119	HALT
0095	77	120	LD (HL),A
0096	78	121	LD A,B
0097	79	122	LD A,C
0098	7A	123	LD A,D
0099	7B	124	LD A,E
009A	7C	125	LD A,H
009B	7D	126	LD A,L
009C	7E	127	LD A,(HL)
009D	7F	128	LD A,A
009E	80	129	ADD A,B
009F	81	130	ADD A,C
00A0	82	131	ADD A,D
00A1	83	132	ADD A,E
00A2	84	133	ADD A,H
00A3	85	134	ADD A,L
00A4	86	135	ADD A,(HL)
00A5	87	136	ADD A,A
00A6	88	137	ADC A,B
00A7	89	138	ADC A,C
00A8	8A	139	ADC A,D
00A9	8B	140	ADC A,E
00AA	8C	141	ADC A,H
00AB	8D	142	ADC A,L
00AC	8E	143	ADC A,(HL)
00AD	8F	144	ADC A,A
00AE	90	145	SUB B
00AF	91	146	SUB C
00B0	92	147	SUB D
00B1	93	148	SUB E
00B2	94	149	SUB H
00B3	95	150	SUB L
00B4	96	151	SUB (HL)
00B5	97	152	SUB A
00B6	98	153	SBC A,B
00B7	99	154	SBC A,C
00B8	9A	155	SBC A,D
00B9	9B	156	SBC A,E
00BA	9C	157	SBC A,H
00BB	9D	158	SBC A,L
00BC	9E	159	SBC A,(HL)
00BD	9F	160	SBC A,A
00BE	A0	161	AND B
00BF	A1	162	AND C
00C0	A2	163	AND D
00C1	A3	164	AND E
00C2	A4	165	AND H
00C3	A5	166	AND L
00C4	A6	167	AND (HL)
00C5	A7	168	AND A
00C6	A8	169	XOR B
00C7	A9	170	XOR C

00C8	AA	171	XOR D
00C9	AB	172	XOR E
00CA	AC	173	XOR H
00CB	AD	174	XOR L
00CC	AE	175	XOR (HL)
00CD	AF	176	XOR A
00CE	B0	177	OR B
00CF	B1	178	OR C
00D0	B2	179	OR D
00D1	B3	180	OR E
0002	B4	181	OR H
00D3	B5	182	OR L
00D4	B6	183	OR (HL)
00D5	B7	184	OR A
00D6	B8	185	CP B
00D7	B9	186	CP C
00D8	BA	187	CP D
00D9	BB	188	CP E
00DA	BC	189	CP H
00DB	BD	190	CP L
00DC	BE	191	CP (HL)
00DD	BF	192	CP A
00DE	C0	193	RET NZ
00DF	C1	194	POP BC
00E0	C28405	195	JP NZ, NN
00E3	C38405	196	JP NN
00E6	C48405	197	CALL NZ,NN
00E9	C5	198	PUSH BC
00EA	C620	199	ADD A,N
00EC	C7	200	RST 0
00ED	C8	201	RET Z
00EE	C9	202	RET
00EF	CA8405	203	JP Z,NN
00F2	CC8405	204	CALL Z,NN
00F5	CD8405	205	CALL NN
00F8	CE20	206	ADC A,N
00FA	CF	207	RST 8
00FB	D0	208	RET NC
00FC	D1	209	POP DE
00FD	D28405	210	JP NC,NN
0100	D320	211	OUT N,A
0102	D48405	212	CALL NC,NN
0105	D5	213	PUSH DE
0106	D620	214	SUB N
0108	D7	215	RST 10H
0109	D8	216	RET C
010A	D9	217	EXX
010B	DA8405	218	JP C,NN
010E	DB20	219	IN A,N
0110	DC8405	220	CALL C,NN
0113	DE20	221	SBC A,N
0115	DF	222	RST 18H
0116	E0	223	RET PO
0117	E1	224	POP HL
0118	E28405	225	JP PO,NN
011B	E3	226	EX (SP),HL
011C	E48405	227	CALL PO,NN
011F	E5	228	PUSH HL
0120	E620	229	AND N
0122	E7	230	RST 20H
0123	E8	231	RET PE
0124	E9	232	JP (HL)
0125	EA8405	233	JP PE,NN
0128	EB	234	EX DE,HL
0129	EC8405	235	CALL PE,NN
012C	EE20	236	XOR N
012E	EF	237	RST 28H
012F	F0	238	RET P
0130	F1	239	POP AF
0131	F28405	240	JP P,NN
0134	F3	241	DI
0135	F48405	242	CALL P,NN
0138	F5	243	PUSH AF
0139	F620	244	OR N

013B	F7	245	RST 30H
013C	F8	246	RET M
013D	F9	247	LD SP,HL
013E	FA8405	248	JP M,NN
0141	FB	249	EI
0142	FC8405	250	CALL M,NN
0145	FE20	251	CP N
0147	FF	252	RST 38H
0148	CB00	253	RLC B
014A	CB01	254	RLC C
014C	CB02	255	RLC D
014E	CB03	256	RLC E
0150	CB04	257	RLC H
0152	CB05	258	RLC L
0154	CB06	259	RLC (HL)
0156	CB07	260	RLC A
0158	CB08	261	RRC B
015A	CB09	262	RRC C
015C	CB0A	263	RRC D
015E	CB0B	264	RRC E
0160	CB0C	265	RRC H
0162	CB0D	266	RRC L
0164	CB0E	267	RRC (HL)
0166	CB0F	268	RRC A
0168	CB10	269	RL B
016A	CB11	270	RL C
016C	CB12	271	RL D
016E	CB13	272	RL E
0170	CB14	273	RL H
0172	CB15	274	RL L
0174	CB16	275	RL (HL)
0176	CB17	276	RL A
0178	CB18	277	RR B
017A	CB19	278	RR C
017C	CB1A	279	RR D
017E	CB1B	280	RR E
0180	CB1C	281	RR H
0182	CB1D	282	RR L
0184	CB1E	283	RR (HL)
0186	CB1F	284	RR A
0188	CB20	285	SLA B
018A	CB21	286	SLA C
018C	CB22	287	SLA D
018E	CB23	288	SLA E
0190	CB24	289	SLA H
0192	CB25	290	SLA L
0194	CB26	291	SLA (HL)
0196	CB27	292	SLA A
0198	CB28	293	SRA B
019A	CB29	294	SRA C
019C	CB2A	295	SRA D
019E	CB2B	296	SRA E
01A0	CB2C	297	SRA H
01A2	CB2D	298	SRA L
01A4	CB2E	299	SRA (HL)
01A6	CB2F	300	SRA A
01A8	CB38	301	SRL B
01AA	CB39	302	SRL C
01AC	CB3A	303	SRL D
01AE	CB3B	304	SRL E
01B0	CB3C	305	SRL H
01B2	CB3D	306	SRL L
01B4	CB3E	307	SRL (HL)
01B6	CB3F	308	SRL A
01B8	CB40	309	BIT 0,B
01BA	CB41	310	BIT 0,C
01BC	CB42	311	BIT 0,D
01BE	CB43	312	BIT 0,E
01C0	CB44	313	BIT 0,H
01C2	CB45	314	BIT 0,L
01C4	CB46	315	BIT 0,(HL)
01C6	CB47	316	BIT 0,A
01C8	CB48	317	BIT 1,B
01CA	CB49	318	BIT 1,C
01CC	CB4A	319	BIT 1,D
01CE	CB4B	320	BIT 1,E
01D0	CB4C	321	BIT 1,H
01D2	CB4D	322	BIT 1,L
01D4	CB4E	323	BIT 1,(HL)
01D6	CB4F	324	BIT 1,A
01D8	CB50	325	BIT 2,B
01DA	CB51	326	BIT 2,C
01DC	CB52	327	BIT 2,D
01DE	CB53	328	BIT 2,E
01E0	CB54	329	BIT 2,H
01E2	CB55	330	BIT 2,L
01E4	CB56	331	BIT 2,(HL)
01E6	CB57	332	BIT 2,A
01E8	CB58	333	BIT 3,B
01EA	CB59	334	BIT 3,C
01EC	CB5A	335	BIT 3,D
01EE	CB5B	336	BIT 3,E
01F0	CB5C	337	BIT 3,H
01F2	CB5D	338	BIT 3,L
01F4	CB5E	339	BIT 3,(HL)
01F6	CB5F	340	BIT 3,A
01F8	CB60	341	BIT 4,B
01FA	CB61	342	BIT 4,C
01FC	CB62	343	BIT 4,D
01FE	CB63	344	BIT 4,E
0200	CB64	345	BIT 4,H
0202	CB65	346	BIT 4,L
0204	CB66	347	BIT 4,(HL)
0206	CB67	348	BIT 4,A
0208	CB68	349	BIT 5,B
020A	CB69	350	BIT 5,C
020C	CB6A	351	BIT 5,D
020E	CB6B	352	BIT 5,E
0210	CB6C	353	BIT 5,H
0212	CB6D	354	BIT 5,L
0214	CB6E	355	BIT 5,(HL)
0216	CB6F	356	BIT 5,A
0218	CB70	357	BIT 6,B
021A	CB71	358	BIT 6,C
021C	CB72	359	BIT 6,D
021E	CB73	360	BIT 6,E
0220	CB74	361	BIT 6,H
0222	CB75	362	BIT 6,L
0224	CB76	363	BIT 6,(HL)
0226	CB77	364	BIT 6,A
0228	CB78	365	BIT 7,B
022A	CB79	366	BIT 7,C
022C	CB7A	367	BIT 7,D
022E	CB7B	368	BIT 7,E
0230	CB7C	369	BIT 7,H
0232	CB7D	370	BIT 7,L
0234	CB7E	371	BIT 7,(HL)
0236	CB7F	372	BIT 7,A
0238	CB80	373	RES 0,B
023A	CB81	374	RES 0,C
023C	CB82	375	RES 0,D
023E	CB83	376	RES 0,E
0240	CB84	377	RES 0,H
0242	CB85	378	RES 0,L
0244	CB86	379	RES 0,(HL)
0246	CB87	380	RES 0,A
0248	CB88	381	RES 1,B
024A	CB89	382	RES 1,C
024C	CB8A	383	RES 1,D
024E	CB8B	384	RES 1,E
0250	CB8C	385	RES 1,H
0252	CB8D	386	RES 1,L
0254	CB8E	387	RES 1,(HL)
0256	CB8F	388	RES 1,A
0258	CB90	389	RES 2,B
025A	CB91	390	RES 2,C
025C	CB92	391	RES 2,D

025E	CB93	392	RES 2,E
0260	CB94	393	RES 2,H
0262	CB95	394	RES 2,L
0264	CB96	395	RES 2,(HL)
0266	CB97	396	RES 2,A
0268	CB98	397	RES 3,B
026A	CB99	398	RES 3,C
026C	CB9A	399	RES 3,D
026E	CB9B	400	RES 3,E
0270	CB9C	401	RES 3,H
0272	CB9D	402	RES 3,L
0274	CB9E	403	RES 3,(HL)
0276	CB9F	404	RES 3,A
0278	CBA0	405	RES 4,B
027A	CBA1	406	RES 4,C
027C	CBA2	407	RES 4,D
027E	CBA3	408	RES 4,E
0280	CBA4	409	RES 4,H
0282	CBA5	410	RES 4,L
0284	CBA6	411	RES 4,(HL)
0286	CBA7	412	RES 4,A
0288	CBA8	413	RES 5,B
028A	CBA9	414	RES 5,C
028C	CBAA	415	RES 5,D
028E	CBAB	416	RES 5,E
0290	CBAC	417	RES 5,H
0292	CBAD	418	RES 5,L
0294	CBAE	419	RES 5,(HL)
0296	CBAF	420	RES 5,A
0298	CBB0	421	RES 6,B
029A	CBB1	422	RES 6,C
029C	CBB2	423	RES 6,D
029E	CBB3	424	RES 6,E
02A0	CBB4	425	RES 6,H
02A2	CBB5	426	RES 6,L
02A4	CBB6	427	RES 6,(HL)
02A6	CBB7	428	RES 6,A
02A8	CBB8	429	RES 7,B
02AA	CBB9	430	RES 7,C
02AC	CBBA	431	RES 7,D
02AE	CBBB	432	RES 7,E
0280	CBBC	433	RES 7,H
0282	CBBD	434	RES 7,L
0284	CBBE	435	RES 7,(HL)
0286	CBBF	436	RES 7,A
0288	CBC0	437	SET 0,B
02BA	CBC1	438	SET 0,C
02BC	CBC2	439	SET 0,D
02BE	CBC3	440	SET 0,E
02C0	CBC4	441	SET 0,H
02C2	CBC5	442	SET 0,L
02C4	CBC6	443	SET 0,(HL)
02C6	CBC7	444	SET 0,A
02C8	CBC8	445	SET 1,B
02CA	CBC9	446	SET 1,C
02CC	CBCA	447	SET 1,D
02CE	CBCB	448	SET 1,E
02D0	CBCC	449	SET 1,H
02D2	CBCD	450	SET 1,L
02D4	CBCE	451	SET 1,(HL)
02D6	CBCF	452	SET 1,A
02D8	CBD0	453	SET 2,B
02DA	CBD1	454	SET 2,C
02DC	CBD2	455	SET 2,D
02DE	CBD3	456	SET 2,E
02E0	CBD4	457	SET 2,H
02E2	CBD5	458	SET 2,L
02E4	CBD6	459	SET 2,(HL)
02E6	CBD7	460	SET 2,A
02E8	CBD8	461	SET 3,B
02EA	CBD9	462	SET 3,C
02EC	CBDA	463	SET 3,D
02EE	CBDB	464	SET 3,E
02F0	CBDC	465	SET 3,H
02F2	CBDD	466	SET 3,L
02F4	CBDE	467	SET 3,(HL)
02F6	CBDF	468	SET 3,A
02F8	CBE0	469	SET 4,B
02FA	CBE1	470	SET 4,C
02FC	CBE2	471	SET 4,D
02FE	CBE3	472	SET 4,E
0300	CBE4	473	SET 4,H
0302	CBE5	474	SET 4,L
0304	CBE6	475	SET 4,(HL)
0306	CBE7	476	SET 4,A
0308	CBE8	477	SET 5,B
030A	CBE9	478	SET 5,C
030C	CBEA	479	SET 5,D
030E	CBEB	480	SET 5,E
0310	CBEC	481	SET 5,H
0312	CBED	482	SET 5,L
0314	CBEE	483	SET 5,(HL)
0316	CBEF	484	SET 5,A
0318	CBF0	485	SET 6,B
031A	CBF1	486	SET 6,C
031C	CBF2	487	SET 6,D
031E	CBF3	488	SET 6,E
0320	CBF4	489	SET 6,H
0322	CBF5	490	SET 6,L
0324	CBF6	491	SET 6,(HL)
0326	CBF7	492	SET 6,A
0328	CBF8	493	SET 7,B
032A	CBF9	494	SET 7,C
032C	CBFA	495	SET 7,D
032E	CBFB	496	SET 7,E
0330	CBFC	497	SET 7,H
0332	CBFD	498	SET 7,L
0334	CBFE	499	SET 7,(HL)
0336	CBFF	500	SET 7,A
0338	DD09	501	ADD IX,BC
033A	DD19	502	ADD IX,DE
033C	DD218405	503	LD IX,NN
0340	DD228405	504	LD (NN),IX
0344	DD23	505	INC IX
0346	DD29	506	ADD IX,IX
0348	DD2A8405	507	LD IX,(NN)
034C	DD2B	508	DEC IX
034E	DD3405	509	INC (IX+IND)
0351	DD3505	510	DEC (IX+IND)
0354	DD360520	511	LD (IX+IND),N
0358	DD39	512	ADD IX,SP
035A	DD4605	513	LD B,(IX+IND)
035D	DD4E05	514	LD C,(IX+IND)
0360	DD5605	515	LD D,(IX+IND)
0363	DD5E05	516	LD E,(IX+IND)
0366	DD6605	517	LD H,(IX+IND)
0369	DD6E05	518	LD L,(IX+IND)
036C	DD7005	519	LD (IX+IND),B
036F	DD7105	520	LD (IX+IND),C
0372	DD7205	521	LD (IX+IND),D
0375	DD7305	522	LD (IX+IND),E
0378	DD7405	523	LD (IX+IND),H
037B	DD7505	524	LD (IX+IND),L
037E	DD7705	525	LD (IX+IND),A
0381	DD7E05	526	LD A,(IX+IND)
0384	DD8605	527	ADD A,(IX+IND)
0387	DD8E05	528	ADC A,(IX+IND)
038A	DD9605	529	SUB (IX+IND)
038D	DD9E05	530	SBC A,(IX+IND)
0390	DDA605	531	AND (IX+IND)
0393	DDAE05	532	XOR (IX+IND)
0396	DDB605	533	OR (IX+IND)
0399	DDBE05	534	CP (IX+IND)
039C	DDE1	535	POP IX
039E	DDE3	536	EX (SP),IX
03A0	DDE5	537	PUSH IX
03A2	DDE9	538	JP (IX)

```
03A4  DDF9          539   LD SP,IX
03A6  DDCB0506      540   RLC (IX+IND)
03AA  DDCB050E      541   RRC (IX+IND)
03AE  DDCB0516      542   RL (IX+IND)
03B2  DDCB051E      543   RR (IX+IND)
03B6  DDCB0526      544   SLA (IX+IND)
03BA  DDCB052E      545   SRA (IX+IND)
03BE  DDCB053E      546   SRL (IX+IND)
03C2  DDCB0546      547   BIT 0,(IX+IND)
03C6  DDCB054E      548   BIT 1,(IX+IND)
03CA  DDCB0556      549   BIT 2,(IX+IND)
03CE  DDCB055E      550   BIT 3,(IX+IND)
03D2  DDCB0566      551   BIT 4,(IX+IND)
03D6  DDCB056E      552   BIT 5,(IX+IND)
03DA  DDCB0576      553   BIT 6,(IX+IND)
03DE  DDCB057E      554   BIT 7,(IX+IND)
03E2  DDCB0586      555   RES 0,(IX+IND)
03E6  DDCB058E      556   RES 1,(IX+IND)
03EA  DDCB0596      557   RES 2,(IX+IND)
03EE  DDCB059E      558   RES 3,(IX+IND)
03F2  DDCB05A6      559   RES 4,(IX+IND)
03F6  DDCB05AE      560   RES 5,(IX+IND)
03FA  DDCB05B6      561   RES 6,(IX+IND)
03FE  DDCB05BE      562   RES 7,(IX+IND)
0402  DDCB05C6      563   SET 0,(IX+IND)
0406  DDCB05CE      564   SET 1,(IX+IND)
040A  DDCB05D6      565   SET 2,(IX+IND)
040E  DDCB05DE      566   SET 3,(IX+IND)
0412  DDCB05E6      567   SET 4,(IX+IND)
0416  DDCB05EE      568   SET 5,(IX+IND)
041A  DDCB05F6      569   SET 6,(IX+IND)
041E  DDCB05FE      570   SET 7,(IX+IND)
0422  ED40          571   IN B,(C)
0424  ED41          572   OUT (C),B
0426  ED42          573   SBC HL,BC
0428  ED438405      574   LD (NN),BC
042C  ED44          575   NEG
042E  ED45          576   RETN
0430  ED46          577   IM 0
0432  ED47          578   LD I,A
0434  ED48          579   IN C,(C)
0436  ED49          580   OUT (C),C
0438  ED4A          581   ADC HL,BC
043A  ED4B8405      582   LD BC,(NN)
043E  ED4D          583   RETI
0440  ED50          584   IN D,(C)
0442  ED51          585   OUT (C),D
0444  ED52          586   SBC HL,DE
0446  ED538405      587   LD (NN),DE
044A  ED56          588   IM 1
044C  ED57          589   LD A,I
044E  ED58          590   IN E,(C)
0450  ED59          591   OUT (C),E
0452  ED5A          592   ADC HL,DE
0454  ED5B8405      593   LD DE,(NN)
0458  ED5E          594   IM 2
045A  ED60          595   IN H,(C)
045C  ED61          596   OUT (C),H
045E  ED62          597   SBC HL,HL
0460  ED67          598   RRD
0462  ED68          599   IN L,(C)
0464  ED69          600   OUT (C),L
0466  ED6A          601   ADC HL,HL
0468  ED6F          602   RLD
046A  ED72          603   SBC HL,SP
046C  ED738405      604   LD (NN),SP
0470  ED78          605   IN A,(C)
0472  ED79          606   OUT (C),A
0474  ED7A          607   ADC HL,SP
0476  ED7B8405      608   LD SP, (NN)
047A  EDA0          609   LDI
047C  EDA1          610   CPI
047E  EDA2          611   INI

0480  EDA3          612   OUTI
0482  EDA8          613   LDD
0484  EDA9          614   CPD
0486  EDAA          615   IND
0488  EDAB          616   OUTD
048A  EDB0          617   LDIR
048C  EDB1          618   CPIR
048E  EDB2          619   INIR
0490  EDB3          620   OTIR
0492  EDB8          621   LDDR
0494  EDB9          622   CPDR
0496  EDBA          623   INDR
0498  EDBB          624   OTDR
049A  FD09          625   ADD IY,BC
049C  FD19          626   ADD IY,DE
049E  FD218405      627   LD IY,NN
04A2  FD228405      628   LD (NN),IY
04A6  FD23          629   INC IY
04A8  FD29          630   ADD IY,IY
04AA  FD2A8405      631   LD IY,(NN)
04AE  FD2B          632   DEC IY
04B0  FD3405        633   INC (IY+IND)
04B3  FD3505        634   DEC (IY+IND)
04B6  FD360520      635   LD (IY+IND),N
04BA  FD39          636   ADD IY,SP
04BC  FD4605        637   LD B,(IY+IND)
04BF  FD4E05        638   LD C,(IY+IND)
04C2  FD5605        639   LD D,(IY+IND)
04C5  FD5E05        640   LD E,(IY+IND)
04C8  FD6605        641   LD H,(IY+IND)
04CB  FD6E05        642   LD L,(IY+IND)
04CE  FD7005        643   LD (IY+IND),B
04D1  FD7105        644   LD (IY+IND),C
04D4  FD7205        645   LD(IY+IND),D
04D7  FD7305        646   LD (IY+IND),E
04DA  FD7405        647   LD (IY+IND),H
04DD  FD7505        648   LD (IY+IND),L
04E0  FD7705        649   LD (IY+IND),A
04E3  FD7E05        650   LD A,(IY+IND)
04E6  FD8605        651   ADD A,(IY+IND)
04E9  FD8E05        652   ADC A,(IY+IND)
04EC  FD9605        653   SUB (IY+IND)
04EF  FD9E05        654   SBC A,(IY+IND)
04F2  FDA605        655   AND (IY+IND)
04F5  FDAE05        656   XOR (IY+IND)
04F8  FDB605        657   OR (IY+IND)
04FB  FDBE05        658   CP (IY+IND)
04FE  FDE1          659   POP IY
0500  FDE3          660   EX (SP),IY
0502  FDE5          661   PUSH IY
0504  FDE9          662   JP (IY)
0506  FDF9          663   LD SP,IY
0508  FDCB0506      664   RLC (IY+IND)
050C  FDCB050E      665   RRC (IY+IND)
0510  FDCB0516      666   RL (IY+IND)
0514  FDCB051E      667   RR (IY+IND)
0518  FDCB0526      668   SLA (IY+IND)
051C  FDCB052E      669   SRA (IY+IND)
0520  FDCB053E      670   SRL (IY+IND)
0524  FDCB0546      671   BIT 0,(IY+IND)
0528  FDCB054E      672   BIT 1,(IY+IND)
052C  FDCB0556      673   BIT 2,(IY+IND)
0530  FDCB055E      674   BIT 3,(IY+IND)
0534  FDCB0566      675   BIT 4,(IY+IND)
0538  FDCB056E      676   BIT 5,(IY+IND)
053C  FDCB0576      677   BIT 6,(IY+IND)
0540  FDCB057E      678   BIT 7,(IY+IND)
0544  FDCB0586      679   RES 0,(IY+IND)
0548  FDCB058E      680   RES 1,(IY+IND)
054C  FDCB0596      681   RES 2,(IY+IND)
0550  FDCB059E      682   RES 3,(IY+IND)
0554  FDCB05A6      683   RES 4,(IY+IND)
0558  FDCB05AE      684   RES 5,(IY+IND)
055C  FDCB05B6      685   RES 6,(IY+IND)
```

```
0560 FDCB05BE 686      RES 7,(IY+IND)
0564 FDCB05C6 687      SET 0,(IY+IND)
0568 FDCB05CE 688      SET 1,(IY+IND)
056C FDCB05D6 689      SET 2,(IY+IND)
0570 FDCB05DE 690      SET 3,(IY+IND)
0574 FDCB05E6 691      SET 4,(IY+IND)
0578 FDCB05EE 692      SET 5,(IY+IND)
057C FDCB05F6 693      SET 6,(IY+IND)
0580 FDCB05FE 694      SET 7,(IY+IND)
0584          695 NN   DEFS 2
              696 IND  EQU 5
              697 M    EQU 10H
              698 N    EQU 20H
              699 DIS  EQU 30H
              700      END
```

ANEXO 2

Relación de los códigos de operación por orden alfabético

0000	8E	1	ADC	A, (HL)
0001	DD8E05	2	ADC	A, (IX+IND)
0004	FD8E05	3	ADC	A, (IY+IND)
0007	8F	4	ADC	A, A
0008	88	5	ADC	A, B
0009	89	6	ADC	A, C
000A	8A	7	ADC	A, D
000B	8B	8	ADC	A, E
000C	8C	9	ADC	A, H
000D	8D	10	ADC	A, L
000E	CE20	11	ADC	A, N
0010	ED4A	12	ADC	HL, BC
0012	ED5A	13	ADC	HL, DE
0014	ED6A	14	ADC	HL, HL
0016	ED7A	15	ADC	HL, SP
0018	86	16	ADD	A, (HL)
0019	DD8605	17	ADD	A, (IX+IND)
001C	FD8605	18	ADD	A, (IY+IND)
001F	87	19	ADD	A, A
0020	80	20	ADD	A, B
0021	81	21	ADD	A, C
0022	82	22	ADD	A, D
0023	83	23	ADD	A, E
0024	84	24	ADD	A, H
0025	85	25	ADD	A, L
0026	C620	26	ADD	A, N
0028	09	27	ADD	HL, BC
0029	19	28	ADD	HL, DE
002A	29	29	ADD	HL, HL
002B	39	30	ADD	HL, SP
002C	DD09	31	ADD	IX, BC
002E	DD19	32	ADD	IX, DE
0030	DD29	33	ADD	IX, IX
0032	DD39	34	ADD	IX, SP
0034	FD09	35	ADD	IY, BC
0036	FD19	36	ADD	IY, DE
0038	FD29	37	ADD	IY, IY
003A	FD39	38	ADD	IY, SP
003C	A6	39	AND	(HL)
003D	DDA605	40	AND	(IX+IND)
0040	FDA605	41	AND	(IY+IND)
0043	A7	42	AND	A
0044	A0	43	AND	B
0045	A1	44	AND	C
0046	A2	45	AND	D
0047	A3	46	AND	E
0048	A4	47	AND	H
0049	A5	48	AND	L
004A	E620	49	AND	N
004C	CB46	50	BIT	0, (HL)
004E	DDCB0546	51	BIT	0, (IX+IND)
0052	FDBC0546	52	BIT	0, (IY+IND)
0056	CB47	53	BIT	0, A
0058	CB40	54	BIT	0, B
005A	CB41	55	BIT	0, C
005C	CB42	56	BIT	0, D
005E	CB43	57	BIT	0, E
0060	CB44	58	BIT	0, H
0062	CB45	59	BIT	0, L
0064	CB4E	60	BIT	1, (HL)
0066	DDCB054E	61	BIT	1, (IX+IND)
006A	FDCB054E	62	BIT	1, (IY+IND)
006E	CB4F	63	BIT	1, A
0070	CB48	64	BIT	1, B
0072	CB49	65	BIT	1, C
0074	CB4A	66	BIT	1, D
0076	CB4B	67	BIT	1, E
0078	CB4C	68	BIT	1, H
007A	CB4D	69	BIT	1, L
007C	CB56	70	BIT	2, (HL)
007E	DDCB0556	71	BIT	2, (IX+IND)
0082	FDCB0556	72	BIT	2, (IY+IND)
0086	CB57	73	BIT	2, A
0088	CB50	74	BIT	2, B
008A	CB51	75	BIT	2, C
008C	CB52	76	BIT	2, D
008E	CB53	77	BIT	2, E
0090	CB54	78	BIT	2, H
0092	CB55	79	BIT	2, L
0094	CB5E	80	BIT	3, (HL)
0096	DDCB055E	81	BIT	3, (IX+IND)
009A	FDCB055E	82	BIT	3, (IY+IND)
009E	CB5F	83	BIT	3, A
00A0	CB58	84	BIT	3, B
00A2	CB59	85	BIT	3, C
00A4	CB5A	86	BIT	3, D
00A6	CB5B	87	BIT	3, E
00A8	CB5C	88	BIT	3, H
00AA	CB5D	89	BIT	3, L
00AC	CB66	90	BIT	4, (HL)
00AE	DDCB0566	91	BIT	4, (IX+IND)
00B2	FDCB0566	92	BIT	4, (IY+IND)
00B6	CB67	93	BIT	4, A
00B8	CB60	94	BIT	4, B
00BA	CB61	95	BIT	4, C
00BC	CB62	96	BIT	4, D

00BE	CB63	97	BIT	4, E
00C0	CB64	98	BIT	4, H
00C2	CB65	99	BIT	4, L
00C4	CB6E	100	BIT	5, (HL)
00C6	DDCB056E	101	BIT	5, (IX+IND)
00CA	FDCB056E	102	BIT	5, (IY+IND)
00CE	CB6F	103	BIT	5, A
00D0	CB68	104	BIT	5, B
00D2	CB69	105	BIT	5, C
00D4	CB6A	106	BIT	5, D
00D6	CB6B	107	BIT	5, E
00D8	CB6C	108	BIT	5, H
00DA	CB6D	109	BIT	5, L
00DC	CB76	110	BIT	6, (HL)
00DE	DDCB0576	111	BIT	6, (IX+IND)
00E2	FDCB0576	112	BIT	6, (IY+IND)
00E6	CB77	113	BIT	6, A
00E8	CB70	114	BIT	6, B
00EA	CB71	115	BIT	6, C
00EC	CB72	116	BIT	6, D
00EE	CB73	117	BIT	6, E
00F0	CB74	118	BIT	6, H
00F2	CB75	119	BIT	6, L
00F4	CB7E	120	BIT	7, (HL)
00F6	DDCB057E	121	BIT	7, (IX+IND)
00FA	FDCB057E	122	BIT	7, (IY+IND)
00FE	CB7F	123	BIT	7, A
0100	CB78	124	BIT	7, B
0102	CB79	125	BIT	7, C
0104	CB7A	126	BIT	7, D
0106	CB7B	127	BIT	7, E
0108	CB7C	128	BIT	7, H
010A	CB7D	129	BIT	7, L
010C	DC8405	130	CALL	C, NN
010F	FC8405	131	CALL	M, NN
0112	D48405	132	CALL	NC, NN
0115	CD8405	133	CALL	NN
0118	C48405	134	CALL	NZ, NN
011B	F48405	135	CALL	P, NN
011E	EC8405	136	CALL	PE, NN
0121	E48405	137	CALL	PO, NN
0124	CC8405	138	CALL	Z, NN
0127	3F	139	CCF	
0128	BE	140	CP	(HL)
0129	DDBE05	141	CP	(IX+IND)
012C	FDBE05	142	CP	(IY+IND)
012F	BF	143	CP	A
0130	B8	144	CP	B
0131	B9	145	CP	C
0132	BA	146	CP	D
0133	BB	147	CP	E
0134	BC	148	CP	H
0135	BD	149	CP	L
0136	FE20	150	CP	N
0138	EDA9	151	CPD	
013A	EDB9	152	CPDR	
013C	EDA1	153	CPI	
013E	EDB1	154	CPIR	
0140	2F	155	CPL	
0141	27	156	DAA	
0142	35	157	DEC	(HL)
0143	DD3505	158	DEC	(IX+IND)
0146	FD3505	159	DEC	(IY+IND)
0149	3D	160	DEC	A
014A	05	161	DEC	B
014B	0B	162	DEC	BC
014C	0D	163	DEC	C
014D	15	164	DEC	D
014E	1B	165	DEC	DE
014F	1D	166	DEC	E
0150	25	167	DEC	H
0151	2B	168	DEC	HL
0152	DD2B	169	DEC	IX
0154	FD2B	170	DEC	IY
0156	2D	171	DEC	L
0157	3B	172	DEC	SP
0158	F3	173	DI	
0159	102E	174	DJNZ	DIS
015B	FB	175	EI	
015C	E3	176	EX	(SP), HL
015D	DDE3	177	EX	(SP), IX
015F	FDE3	178	EX	(SP), IY
0161	08	179	EX	AF, AF'
0162	EB	180	EX	DE, HL
0163	D9	181	EXX	
0164	76	182	HALT	
0165	ED46	183	IM	0
0167	ED56	184	IM	1
0169	ED5E	185	IM	2
016B	ED78	186	IN	A, (C)
016D	DB20	187	IN	A, N
016F	ED40	188	IN	B, (C)
0171	ED48	189	IN	C, (C)
0173	ED50	190	IN	D, (C)
0175	ED58	191	IN	E, (C)
0177	ED60	192	IN	H, (C)
0179	ED68	193	IN	L, (C)
017B	34	194	INC	(HL)
017C	DD3405	195	INC	(IX+IND)
017F	FD3405	196	INC	(IY+IND)
0182	3C	197	INC	A
0183	04	198	INC	B
0184	03	199	INC	BC
0185	0C	200	INC	C
0186	14	201	INC	D
0187	13	202	INC	DE
0188	1C	203	INC	E
0189	24	204	INC	H
018A	23	205	INC	HL
018B	DD23	206	INC	IX
018D	FD23	207	INC	IY
018F	2C	208	INC	L
0190	33	209	INC	SP
0191	EDAA	210	IND	
0193	EDBA	211	INDR	
0195	EDA2	212	INI	
0197	EDB2	213	INIR	
0199	E9	214	JP	(HL)
019A	DDE9	215	JP	(IX)
019C	FDE9	216	JP	(IY)
019E	DA8405	217	JP	C, NN
01A1	FA8405	218	JP	M, NN
01A4	D28405	219	JP	NC, NN
01A7	C38405	220	JP	NN
01AA	C28405	221	JP	NZ, NN
01AD	F28405	222	JP	P, NN
01B0	EA8405	223	JP	PE; NN
01B3	E28405	224	JP	PO, NN
01B6	CA8405	225	JP	Z, NN
01B9	382E	226	JR	C, DIS
01BB	182E	227	JR	DIS
01BD	302E	228	JR	NC, DIS
01BF	202E	229	JR	NZ, DIS
01C1	282E	230	JR	Z, DIS
01C3	02	231	LD	(BC), A
01C4	12	232	LD	(DE), A
01C5	77	233	LD	(HL), A
01C6	70	234	LD	(HL), B
01C7	71	235	LD	(HL), C
01C8	72	236	LD	(HL), D
01C9	73	237	LD	(HL), E
01CA	74	238	LD	(HL), H
01CB	75	239	LD	(HL), L
01CC	3620	240	LD	(HL), N
01CE	DD7705	241	LD	(IX+IND), A
01D1	DD7005	242	LD	(IX+IND), B
01D4	DD7105	243	LD	(IX+IND), C
01D7	DD7205	244	LD	(IX+IND), D

01DA	DD7305	245	LD	(IX+IND), E
01DD	DD7405	246	LD	(IX+IND), H
01E0	DD7505	247	LD	(IX+IND), L
01E3	DD360520	248	LD	(IX+IND), N
01E7	FD7705	249	LD	(IY+IND), A
01EA	FD7005	250	LD	(IY+IND), B
01ED	FD7105	251	LD	(IY+IND), C
01F0	FD7205	252	LD	(IY+IND), D
01F3	FD7305	253	LD	(IY+IND), E
01F6	FD7405	254	LD	(IY+IND), H
01F9	FD7505	255	LD	(IY+IND), L
01FC	FD360520	256	LD	(IY+IND), N
0200	328405	257	LD	(NN), A
0203	ED438405	258	LD	(NN), BC
0207	ED538405	259	LD	(NN), DE
020B	228405	260	LD	(NN), HL
020E	DD228405	261	LD	(NN), IX
0212	FD228405	262	LD	(NN), IY
0216	ED738405	263	LD	(NN), SP
021A	0A	264	LD	A, (BC)
021B	1A	265	LD	A, (DE)
021C	7E	266	LD	A, (HL)
021D	DD7E05	267	LD	A, (IX+IND)
0220	FD7E05	268	LD	A, (IY+IND)
0223	3A8405	269	LD	A, (NN)
0226	7F	270	LD	A, A
0227	78	271	LD	A, B
0228	79	272	LD	A, C
0229	7A	273	LD	A, D
022A	7B	274	LD	A, E
022B	7C	275	LD	A, H
022C	ED57	276	LD	A, I
022E	7D	277	LD	A, L
022F	3E20	278	LD	A, N
0231	46	279	LD	B, (HL)
0232	DD4605	280	LD	B, (IX+IND)
0235	FD4605	281	LD	B, (IY+IND)
0238	47	282	LD	B, A
0239	40	283	LD	B, B
023A	41	284	LD	B, C
023B	42	285	LD	B, D
023C	43	286	LD	B, E
023D	44	287	LD	B, H, NN
023E	45	288	LD	B, L
023F	0620	289	LD	B, N
0241	ED4B8405	290	LD	BC, (NN)
0245	018405	291	LD	BC, NN
0248	4E	292	LD	C, (HL)
0249	DD4E05	293	LD	C, (IX+IND)
024C	FD4E05	294	LD	C, (IY+IND)
024F	4F	295	LD	C, A
0250	48	296	LD	C, B
0251	49	297	LD	C, C
0252	4A	298	LD	C, D
0253	4B	299	LD	C, E
0254	4C	300	LD	C, H
0255	4D	301	LD	C, L
0256	0E20	302	LD	C, N
0258	56	303	LD	D, (HL)
0259	DD5605	304	LD	D, (IX+IND)
025C	FD5605	305	LD	D, (IY+IND)
025F	57	306	LD	D, A
0260	50	307	LD	D, B
0261	51	308	LD	D, C
0262	52	309	LD	D, D
0263	53	310	LD	D, E
0264	54	311	LD	D, H
0265	55	312	LD	D, L
0266	1620	313	LD	D, N
0268	ED5B8405	314	LD	DE, (NN)
026C	118405	315	LD	DE, NN
026F	5E	316	LD	E, (HL)
0270	DD5E05	317	LD	E, (IX+IND)
0273	FD5E05	318	LD	E, (IY+IND)
0276	5F	319	LD	E, A
0277	58	320	LD	E, B
0278	59	321	LD	E, C
0279	5A	322	LD	E, D
027A	5B	323	LD	E, E
027B	5C	324	LD	E, H
027C	5D	325	LD	E, L
027D	1E20	326	LD	E, N
027F	66	327	LD	H, (HL)
0280	DD6605	328	LD	H, (IX+IND)
0283	FD6605	329	LD	H, (IY+IND)
0286	67	330	LD	H, A
0287	60	331	LD	H, B
0288	61	332	LD	H, C
0289	62	333	LD	H, D
028A	63	334	LD	H, E
028B	64	335	LD	H, H
028C	65	336	LD	H, L
028D	2620	337	LD	H, N
028F	2A8405	338	LD	HL, (NN)
0292	218405	339	LD	HL, NN
0295	ED47	340	LD	I, A
0297	DD2A8405	341	LD	IX, (NN)
029B	DD218405	342	LD	IX, NN
029F	FD2A8405	343	LD	IY, (NN)
02A3	FD218405	344	LD	IY, NN
02A7	6E	345	LD	L, (HL)
02A8	DD6E05	346	LD	L,(IX+IND)
02AB	FD6E05	347	LD	L,(IY+IND)
02AE	6F	348	LD	L,A
02AF	68	349	LD	L,B
02B0	69	350	LD	L,C
02B1	6A	351	LD	L,D
02B2	6B	352	LD	L,E
02B3	6C	353	LD	L,H
02B4	6D	354	LD	L,L
02B5	2E20	355	LD	L,N
02B7	ED7B8405	356	LD	SP,(NN)
02BB	F9	357	LD	SP,HL
02BC	DDF9	358	LD	SP,IX
02BE	FDF9	359	LD	SP,IY
02C0	318405	360	LD	SP,NN
02C3	EDA8	361	LDD	
02C5	EDB8	362	LDDR	
02C7	EDA0	363	LDI	
02C9	EDB0	364	LDIR	
02CB	ED44	365	NEG	
02CD	00	366	NOP	
02CE	B6	367	OR	(HL)
02CF	DDB605	368	OR	(IX+IND)
02D2	FDB605	369	OR	(IY+IND)
02D5	B7	370	OR	A
02D6	B0	371	OR	B
02D7	B1	372	OR	C
02D8	B2	373	OR	D
02D9	B3	374	OR	E
02DA	B4	375	OR	H
02DB	B5	376	OR	L
02DC	F620	377	OR	N
02DE	EDBE	378	OTDR	
02E0	EDB3	379	OTIR	
02E2	ED79	380	OUT	(C),A
02E4	ED41	381	OUT	(C),B
02E6	ED49	382	OUT	(C),C
02E8	ED51	383	OUT	(C),D
02EA	ED59	384	OUT	(C),E
02EC	ED61	385	OUT	(C),H
02EE	ED69	386	OUT	(C),L
02F0	D320	387	OUT	N,A
02F2	EDAB	388	OUTD	
02F4	EDA3	389	OUTI	
02F6	F1	390	POP	AF
02F7	C1	391	POP	BC
02F8	D1	392	POP	DE

02F9	E1	393	POP	HL
02FA	DDE1	394	POP	IX
02FC	FDE1	395	POP	IY
02FE	F5	396	PUSH	AF
02FF	C5	397	PUSH	BC
0300	D5	398	PUSH	DE
0301	E5	399	PUSH	HL
0302	DDE5	400	PUSH	IX
0304	FDE5	401	PUSH	IY
0306	CB86	402	RES	0,(HL)
0308	DDCB0586	403	RES	0,(IX+IND)
030C	FDCB0586	404	RES	0,(IY+IND)
0310	CB87	405	RES	0,A
0312	CB80	406	RES	0,B
0314	CB81	407	RES	0,C
0316	CB82	408	RES	0,D
0318	CB83	409	RES	0,E
031A	CB84	410	RES	0,H
031C	CB85	411	RES	0,L
031E	CB8E	412	RES	1,(HL)
0320	DDCB058E	413	RES	1,(IX+IND)
0324	FDCB058E	414	RES	1,(IY+IND)
0328	CB8F	415	RES	1,A
032A	CB88	416	RES	1,B
032C	CB89	417	RES	1,C
032E	CB8A	418	RES	1,D
0330	CB8B	419	RES	1,E
0332	CB8C	420	RES	1,H
0334	CB8D	421	RES	1,L
0336	CB96	422	RES	2,(HL)
0338	DDCB0596	423	RES	2,(IX+IND)
033C	FDCB0596	424	RES	2,(IY+IND)
0340	CB97	425	RES	2,A
0342	CB90	426	RES	2,B
0344	CB91	427	RES	2,C
0346	CB92	428	RES	2,D
0348	CB93	429	RES	2,E
034A	CB94	430	RES	2,H
034C	CB95	431	RES	2,L
034E	CB9E	432	RES	3,(HL)
0350	DDCB059E	433	RES	3,(IX+IND)
0354	FDCB059E	434	RES	3,(IY+IND)
0358	CB9F	435	RES	3,A
035A	CB98	436	RES	3,B
035C	CB99	437	RES	3,C
035E	CB9A	438	RES	3,D
0360	CB9B	439	RES	3,E
0362	CB9C	440	RES	3,H
0364	CB9D	441	RES	3,L
0366	CBA6	442	RES	4,(HL)
0368	DDCB05A6	443	RES	4,(IX+IND)
036C	FDCB05A6	444	RES	4,(IY+IND)
0370	CBA7	445	RES	4,A
0372	CBA0	446	RES	4,B
0374	CBA1	447	RES	4,C
0376	CBA2	448	RES	4,D
0378	CBA3	449	RES	4,E
037A	CBA4	450	RES	4,H
037C	CBA5	451	RES	4,L
037E	CBAE	452	RES	5,(HL)
0380	DDCB05AE	453	RES	5,(IX+IND)
0384	FDCB05AE	454	RES	5,(IY+IND)
0388	CBAF	455	RES	5,A
038A	CBA8	456	RES	5,B
038C	CBA9	457	RES	5,C
038E	CBAA	458	RES	5,D
0390	CBAB	459	RES	5,E
0392	CBAC	460	RES	5,H
0394	CBAD	461	RES	5,L
0396	CBB6	462	RES	6,(HL)
0398	DDCB05B6	463	RES	6,(IX+IND)
039C	FDCB05B6	464	RES	6,(IY+IND)
03A0	CBB7	465	RES	6,A
03A2	CBB0	466	RES	6,B
03A4	CBB1	467	RES	6,C
03A6	CBB2	468	RES	6,D
03A8	CBB3	469	RES	6,E
03AA	CBB4	470	RES	6,H
03AC	CBB5	471	RES	6,L
03AE	CBBE	472	RES	7,(HL)
03B0	DDCB05BE	473	RES	7,(IX+IND)
03B4	FDCB05BE	474	RES	7,(IY+IND)
03B8	CBBF	475	RES	7,A
03BA	CBB8	476	RES	7,B
03BC	CBB9	477	RES	7,C
03BE	CBBA	478	RES	7,D
03C0	CBBB	479	RES	7,E
03C2	CBBC	480	RES	7,H
03C4	CBBD	481	RES	7,L
03C6	C9	482	RET	
03C7	D8	483	RET	C
03C8	F8	484	RET	M
03C9	D0	485	RET	NC
03CA	C0	486	RET	NZ
03CB	F0	487	RET	P
03CC	E8	488	RET	PE
03CD	E0	489	RET	PO
03CE	C8	490	RET	Z
03CF	ED4D	491	RETI	
03D1	ED45	492	RETN	
03D3	CB16	493	RL	(HL)
03D5	DDCB0516	494	RL	(IX+IND)
03D9	FDCB0516	495	RL	(IY+IND)
03DD	CB17	496	RL	A
03DF	CB10	497	RL	B
03E1	CB11	498	RL	C
03E3	CB12	499	RL	D
03E5	CB13	500	RL	E
03E7	CB14	501	RL	H
03E9	CB15	502	RL	L
03EB	17	503	RLA	
03EC	CB06	504	RLC	(HL)
03EE	DDCB0506	505	RLC	(IX+IND)
03F2	FDCB0506	506	RLC	(IY+IND)
03F6	CB07	507	RLC	A
03F8	CB00	508	RLC	B
03FA	CB01	509	RLC	C
03FC	CB02	510	RLC	D
03FE	CB03	511	RLC	E
0400	CB04	512	RLC	H
0402	CB05	513	RLC	L
0404	07	514	RLCA	
0405	ED6F	515	RLD	
0407	CB1E	516	RR	(HL)
0409	DDCB051E	517	RR	(IX+IND)
040D	FDCB051E	518	RR	(IY+IND)
0411	CB1F	519	RR	A
0413	CB18	520	RR	B
0415	CB19	521	RR	C
0417	CB1A	522	RR	D
0419	CB1B	523	RR	E
041B	CB1C	524	RR	H
041D	CB1D	525	RR	L
041F	1F	526	RRA	
0420	CB0E	527	RRC	(HL)
0422	DDCB050E	528	RRC	(IX+IND)
0426	FDCB050E	529	RRC	(IY+IND)
042A	CB0F	530	RRC	A
042C	CB08	531	RRC	B
042E	CB09	532	RRC	C
0430	CB0A	533	RRC	D
0432	CB0B	534	RRC	E
0434	CB0C	535	RRC	H
0436	CB0D	536	RRC	L
0438	0F	537	RRCA	
0439	ED67	538	RRD	
043B	C7	539	RST	0
043C	D7	540	RST	10H

```
043D  DF        541  RST  18H
043E  E7        542  RST  20H
043F  EF        543  RST  28H
0440  F7        544  RST  30H
0441  FF        545  RST  38H
0442  CF        546  RST  8
0443  9E        547  SBC  A,(HL)
0444  DD9E05    548  SBC  A,(IX+IND)
0447  FD9E05    549  SBC  A,(IY+IND)
044A  9F        550  SBC  A,A
044B  98        551  SBC  A,B
044C  99        552  SBC  A,C
044D  9A        553  SBC  A,D
044E  9B        554  SBC  A,E
044F  9C        555  SBC  A,H
0450  9D        556  SBC  A,L
0451  DE20      557  SBC  A,N
0453  ED42      558  SBC  HL,BC
0455  ED52      559  SBC  HL,DE
0457  ED62      560  SBC  HL,HL
0459  ED72      561  SBC  HL,SP
045B  37        562  SCF
045C  CBC6      563  SET  0,(HL)
045E  DDCB05C6  564  SET  0,(IX+IND)
0462  FDCB05C6  565  SET  0,(IY+IND)
0466  CBC7      566  SET  0,A
0468  CBC0      567  SET  0,B
046A  CBC1      568  SET  0,C
046C  CBC2      569  SET  0,D
046E  CBC3      570  SET  0,E
0470  CBC4      571  SET  0,H
0472  CBC5      572  SET  0,L
0474  CBCE      573  SET  1,(HL)
0476  DDCB05CE  574  SET  1,(IX+IND)
047A  FDCB05CE  575  SET  1,(IY+IND)
047E  CBCF      576  SET  1,A
0480  CBC8      577  SET  1,B
0482  CBC9      578  SET  1,C
0484  CBCA      579  SET  1,D
0486  CBCB      580  SET  1,E
0488  CBCC      581  SET  1,H
048A  CBCD      582  SET  1,L
048C  CBD6      583  SET  2,(HL)
048E  DDCB05D6  584  SET  2,(IX+IND)
0492  FDCB05D6  585  SET  2,(IY+IND)
0496  CBD7      586  SET  2,A
0498  CBD0      587  SET  2,B
049A  CBD1      588  SET  2,C
049C  CBD2      589  SET  2,D
049E  CBD3      590  SET  2,E
04A0  CBD4      591  SET  2,H
04A2  CBD5      592  SET  2,L
04A4  CBDS      593  SET  3,B
04A6  CBDE      594  SET  3,(HL)
04A8  DDCB05DE  595  SET  3,(IX+IND)
04AC  FDCB05DE  596  SET  3,(IY+IND)
04B0  CBDF      597  SET  3,A
04B2  CBD9      598  SET  3,C
04B4  CBDA      599  SET  3,D
04B6  CBDB      600  SET  3,E
04B8  CBDC      601  SET  3,H
04BA  CBDD      602  SET  3,L
04BC  CBE6      603  SET  4,(HL)
04BE  DDCB05E6  604  SET  4,(IX+IND)
04C2  FDCB05E6  605  SET  4,(IY+IND)
04C6  CBE7      606  SET  4,A
04C8  CBE0      607  SET  4,B
04CA  CBE1      608  SET  4,C
04CC  CBE2      609  SET  4,D
04CE  CBE3      610  SET  4,E
04D0  CBE4      611  SET  4,H
04D2  CBE5      612  SET  4,L
04D4  CBEE      613  SET  5,(HL)
04D6  DDCB05EE  614  SET  5,(IX+IND)
04DA  FDCB05EE  615  SET  5,(IY+IND)
04DE  CBEF      616  SET  5,A
04E0  CBE8      617  SET  5,B
04E2  CBE9      618  SET  5,C
04E4  CBEA      619  SET  5,D
04E6  CBEB      620  SET  5,E
04E8  CBEC      621  SET  5,H
04EA  CBED      622  SET  5,L
04EC  CBF6      623  SET  6,(HL)
04EE  DDCB05F6  624  SET  6,(IX+IND)
04F2  FDCB05F6  625  SET  6,(IY+IND)
04F6  CBF7      626  SET  6,A
04F8  CBF0      627  SET  6,B
04FA  CBF1      628  SET  6,C
04FC  CBF2      629  SET  6,D
04FE  CBF3      630  SET  6,E
0500  CBF4      631  SET  6,H
0502  CBF5      632  SET  6,L
0504  CBFE      633  SET  7,(HL)
0506  DDCB05FE  634  SET  7,(IX+IND)
050A  FDCB05FE  635  SET  7,(IY+IND)
050E  CBFF      636  SET  7,A
0510  CBF8      637  SET  7,B
0512  CBF9      638  SET  7,C
0514  CBFA      639  SET  7,D
0516  CBFB      640  SET  7,E
0518  CBFC      641  SET  7,H
051A  CBFD      642  SET  7,L
051C  CB26      643  SLA  (HL)
051E  DDCB0526  644  SLA  (IX+IND)
0522  FDCB0526  645  SLA  (IY+IND)
0526  CB27      646  SLA  A
0528  CB20      647  SLA  B
052A  CB21      648  SLA  C
052C  CB22      649  SLA  D
052E  CB23      650  SLA  E
0530  CB24      651  SLA  H
0532  CB25      652  SLA  L
0534  CB2E      653  SRA  (HL)
0536  DDCB052E  654  SRA  (IX+IND)
053A  FDCB052E  655  SRA  (IY+IND)
053E  CB2F      656  SRA  A
0540  CB28      657  SRA  B
0542  CB29      658  SRA  C
0544  CB2A      659  SRA  D
0546  CB2B      660  SRA  E
0548  CB2C      661  SRA  H
054A  CB2D      662  SRA  L
054C  CB3E      663  SRL  (HL)
054E  DDCB053E  664  SRL  (IX+IND)
0552  FDCB053E  665  SRL  (IY+IND)
0556  CB3F      666  SRL  A
0558  CB38      667  SRL  B
055A  CB39      668  SRL  C
055C  CB3A      669  SRL  D
055E  CB3B      670  SRL  E
0560  CB3C      671  SRL  H
0562  CB3D      672  SRL  L
0564  96        673  SUB  (HL)
0565  DD9605    674  SUB  (IX+IND)
0568  FD9605    675  SUB  (IY+IND)
056B  97        676  SUB  A
056C  90        677  SUB  B
056D  91        678  SUB  C
056E  92        679  SUB  D
056F  93        680  SUB  E
0570  94        681  SUB  H
0571  95        682  SUB  L
0572  D620      683  SUB  N
0574  AE        684  XOR  (HL)
0575  DDAE05    685  XOR  (IX+IND)
0578  FDAE05    686  XOR  (IY+IND)
057B  AF        687  XOR  A
057C  A8        688  XOR  B
```

ANEXO 2

```
057D  A9      689  XOR  C
057E  AA      690  XOR  D
057F  AB      691  XOR  E
0580  AC      692  XOR  H
0581  AD      693  XOR  L
0582  EE20    694  XOR  N
```

```
0584
```

```
695 NN    DEFS  2
696 IND   EQU   5
697 M     EQU   10H
698 N     EQU   20H
699 DIS   EQU   30H
700       END
```

Libros sobre INFORMATICA publicados por

PARANINFO S.A.

Generalidades

ABRAMSON.– Teoría de la información y codificación. 5ª edición.
FLORES.– Estructuración y proceso de datos. 4ª edición.
GARCIA SANTESMASES.– Cibernética. Aspectos y tendencias actuales.
GOSLING.– Códigos para ordenadores y microprocesadores.
LEWIS y SMITH.– Estructuras de datos. Programación y aplicaciones.
NANIA.– Diccionario de informática.
OLIVETTI.– Diccionario de Informática. Inglés-Español y Español-Inglés. 5ª edición.
PUJOLLE.– Telemática.
SCHMIDT y MEYERS.– Introducción a los ordenadores y al proceso de datos. 5ª edición.
URMAIEV.– Calculadores analógicos. Elementos de simulación.

Hardware (Equipo físico)

ANGULO.– Electrónica digital moderna. 5ª edición.
ANGULO.– Memorias de Burbujas magnéticas.
ANGULO.– Microprocesadores. Arquitectura, programación y desarrollo de sistemas. 3ª edición.
ANGULO.– Microprocesadores. Curso sobre aplicaciones en sistemas industriales. 3ª edición.
ANGULO.– Microprocesadores. Diseño práctico de sistemas. 2ª edición.
ANGULO.– Microprocesadores. Fundamentos, diseño y aplicaciones en la industria y en los microcomputadores. 3ª edición.
ANGULO.– Microprocesadores de 16 Bits. El 68000 y el 8086/8088.
GARLAND.– Diseño de sistemas microprocesadores. 2ª edición.
HALSALL.– Fundamentos de microprocesadores.
ROBIN y MAURIN.– Interconexión de microprocesadores.
RONY.– El microprocesador 8080 y sus interfases.
SHELLEY.– Microelectrónica.

Lenguajes

BELLIDO y SANCHEZ.– BASIC para maestros.
CHECROUN.– BASIC. Programación de microordenadores. 4ª edición.
DELANOY.– Ficheros en BASIC.
GALAN PASCUAL.– Programación con el lenguaje COBOL. 3ª edición.
LARRECHE.– BASIC. Introducción a la programación. 4ª edición.
MARSHALL.– Lenguajes de programación para micros.
MONTEIL.– Primeros pasos en LOGO.
ROSSI.– BASIC. Curso acelerado. 4ª edición.
SANCHIS LLORCA y MORALES LOZANO.– Programación con el lenguaje PASCAL. 5ª edición.
WATT y MANGADA.– BASIC para niños. 5ª edición.
WATT y MANGADA.– BASIC avanzado para niños. 2ª edición.
WATT y MANGADA.– BASIC para niños con el microordenador DRAGON.
WILLIE.– Diccionario de BASIC.

Aplicaciones e Informática Profesional

ANGULO.– Curso de Robótica.
ANGULO.– Robótica práctica. Teoría y aplicaciones.
ASPINALL.– El microprocesador y sus aplicaciones.
BANKS.– Microordenadores. Cómo funcionan. Para qué sirven.
BELLIDO.– ZX81. Curso de programación BASIC. 3ª edición.
BELLIDO.– Cómo programar su Spectrum y Timex 2068. 7ª edición.
BELLIDO.– Cómo usar los colores y los gráficos en el Spectrum. 3ª edición. (libro y casete)
BISHOP.– Software 64.
ELLERSHAW y SCHOFIELD.– Las primeras 15 lecciones con el Spectrum.
ESCUDERO.– (Centro de Investigación UAM-IBM). Reconocimiento de Patrones. (Fundamentos teóricos, algorítmos y aplicaciones de la moderna técnica denominada "Pattern Recognition").
GAUTHIER y PONTO.– Diseño de programas para sistemas. 4ª edición.
HARTMAN, MATTHES y PROEME.– Manual de los sistemas de información. 2 tomos. 6ª edición.
LUCAS, Jr.– Sistemas de información. Análisis. Diseño. Puesta a punto.
MARTINEZ VELARDE.– El libro de código máquina del Spectrum.
MONTEIL.– Cómo programar su Commodore 64. T/1. BASIC. Gráficos. Sonidos.
PANNELL, JACKSON y LUCAS.– El microordenador en la pequeña Empresa.
PLOUIN.– IBM-PC. Características. Programación. Manejo.
VARIOS.– 60 Programas para el Sinclair ZX Spectrum.
WILLIAMS.– Programación paso a paso con el Spectrum.

PROGRAMACION DEL Z80 CON ENSAMBLADOR

Es una introducción a la programación en lenguaje máquina dirigida a todos los poseedores o futuros poseedores de un micro-ordenador concebido sobre un micro-procesador Z 80.

Con esta obra descubrirá este nuevo lenguaje, que enriquecerá en gran medida las posibilidades de su máquina.

EDITORIAL PARANINFO S.A

Magallanes, 25 - 28015 Madrid

ISBN: 84-283-1376-8

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/