Dullin - Strassenburg

MSX
Lenguaje Máquina

UN LIBRO DATA BECKER
EDITADO POR FERRE MORET, S.A.

Los programas y utilidades que se encuentran en este libro pueden ser adquiridos en Cassette solicitándolos directamente a FERRE MORET, S.A., o en sus distribuidores.

Este libro ha sido traducido por Dña. Elisabeth Flores Sies, traductora diplomada por la Universidad Autónoma de Barcelona y entusiasta del sistema MSX.

Imprime APSSA, ROCA UMBERT, 26 - L'HOSPITALET DE LL. (Barcelona)

ISBN 84-86437-27-X

Depósito legal B.32.101-1985

Advertencia Importante

Los circuitos, procedimientos y programas reproducidos en este libro, son divulgados sin tener en cuenta el estado de las patentes. Están destinados exclusivamente al uso amateur o docente, y no pueden ser utilizados para fines comerciales.

Todos los circuitos, datos técnicos y programas de este libro, han sido eleborados o recopilados con el mayor cuidado por el autor y reproducidos utilizando medidas de control eficaces. No obstante, es posible que exista algún error. FERRE MORET,S.A. se ve por tanto obligada a advertirles, que no puede asumir ninguna garantía, ni responsabilidad jurídica, ni cualquier otra responsabilidad sobre las consecuencias atribuibles a datos erróneos. El autor les agradecerá en todo momento la comunicación de posibles fallos.

AGRADECIMIENTO

En cierto modo, el escribir un libro de este tipo generalmente sólo divierte a los autores. Por desgracia, a veces los conciudadanos, amigos y amigas pueden llegar a sentirse "frustados", pues nosotros, los autores, trabajamos inmersos en una especie de "embriaguez biblio-laboral" y en un estado de delirio por el ordenador, para poder así finalizar un libro. Por ello, queremos agradecer a todas aquellas personas que, con su paciencia y apoyo, han hecho posible la publicación de este libro. Especialmente, damos las gracias a Birgitt Mikutta, Kristin Gruenewald y a Andreas Bethmann por la vallosa ayuda aportada en la corrección del manuscrito. Agradecemos también a Ralph Dullin la elaboración de los diagramas, dibujos y de algunas de las tablas que aparecen en el libro.

Por otro lado, damos las gracias asímismo a la empresa Zilog Inc,USA, que puso a nuestra disposición las listas de comandos para el Z80.

PREFACIO

Desde el momento en que pudimos disponer de uno de los tan prometidos ordenadores MSX, nos quedamos maravillados con esta prodigiosa máquina. El MSX BASIC es realmente extraordinario.
En esta nueva generación de ordenadores salta a la vista un hecho notable: "Compatibilidad" es la palabra clave. Con ello, nos referimos a que, una vez realizadas las necesarias solicitudes de información y revisiones, comprobamos que los programas que habían funcionado con el ordenador MSX, también podían funcionar con cualquier otro ordenador de igual nivel.
La programación en Lenguaje Máquina ofrece algunas ventajas decisivas frente al Lenguaje Basic en cuanto a velocidad y necesidad de espacio para memorizar datos. El objetivo de este libro consiste en facilitar al usuario del MSX el acceso al lenguaje máquina y, de ese modo, permitirle aprovechar para sus programas las ventajas antes mencionadas.
Sin embargo, el aprendizaje del lenguaje máquina no es tan sencillo, porque, ¿quién es capaz de entender lo siguiente así, de entrada?:

3E,2A,21,00,00,01,C0,03,CD,15,08,C9

Pero no se desanime tan pronto. Le resultará fácil aprender el lenguaje máquina si maneja el libro de la siguiente manera:

-Trabaje el libro a fondo capítulo a capítulo
-Intente resolver los ejercicios
-Si ello le resulta difícil, revise tranquilamente el capítulo de nuevo.

Pero ya basta de buenos consejos. Desde ahora, le invitamos a entrar en la aventura del LENGUAJE MAQUINA.

Los autores

INDICE

CAPITULO I : I N T R O D U C C I O N

1.1 ¿Qué es el LENGUAJE MAQUINA?

El Lenguaje Máquina es el lenguaje de programación que el ordenador es capaz de tratar directamente. ¿Qué se entiende por ésto?. Como usted seguramente sabrá, cada ordenador posee un microprocesador que podemos considerar como el "cerebro" del ordenador. Este IC (circulto integrado) se denomina CPU (Central Prozessing Unit) o Unidad central de Procesado (UCP). La CPU ejecuta las órdenes de la máquina, dirige el funcionamiento del ordenador y el de los periféricos. La Unidad Central es la pieza clave de todo el ordenador. Cuando programamos en lenguaje máquina, utilizamos comandos dirigidos directamente a la CPU, la cual puede ejecutarlas de inmediato. Con ello, el lenguaje máquina depende de cada uno de los distintos tipos de procesador.
Los ordenadores MSX poseen un procesador Z80, que también halla aplicación en muchos otros microordenadores. El Z80 es una Unidad Central de gran rendimiento; entiende más de 600 comandos tratados a gran velocidad por el MSX.

¿Por qué en realidad Lenguaje Máquina?

La mayoría de los ordenadores domésticos están equipados con BASIC. Como seguramente ya habrá observado, el aprendizaje de este lenguaje no presenta gran dificultad. El BASIC MSX llama la atención especialmente por su variedad de comandos. Da la impresión como si con este BASIC no quedaran nunca deseos sin cumplir, pudiendo quedar siempre bien resueltos todos los problemas de programación.

Imagínese usted:

El Ministro de Asuntos Exteriores, Sr.Basic, negocia con su colega, el Sr.CPU, en el pais del Lenguaje Máquina. Desgraciadamente, sus conocimientos de ese idioma son muy escasos, de modo que solicita los servicios de la intérprete, la Srta.interpreter, quien traduce sus frases al lenguaje máquina. Como es de suponer, la Srta. Interpreter, a pesar de ser una excelente traductora, siempre es un poco más lenta en la traducción que el político en su discurso, y debido a ello, se prolonga innecesariamente el diálogo de la negociación.

Al programar en BASIC encontramos exactamente el mismo problema. A través del Interpretador, el ordenador ha de interpretar primero el BASIC escrito por el programador. El Interpreador de BASIC es una parte de los programas ya incorporados, que interpreta el programa siguiendo uno a uno cada comando dado. Acto seguido, provoca la inmediata ejecución de los mismos. Para ser más exactos: el Interpreter reconoce el comando BASIC y luego, resuelve la ejecución de dicho comando BASIC a través de la llamada a las rutinas en Lenguaje Máquina de cada comando concreto.

Por ejemplo:

SCREEN 0

En primer lugar, el Interpretador lee el comando carácter a carácter, donde por ejemplo, los espacios, dos puntos, paréntesis y comas le indican cuando ha finalizado una palabra. La palabra (SCREEN) la compara con las entradas de la tabla de comandos en BASIC en ROM. Si no la encuentra, intenta interpretar la palabra como Variable. Si ello tampoco funciona, saca un mensaje de error.
Si el Interpreter encuentra la palabra, bifurca hacia la dirección de salto correspondiente a la palabra. Allí se lee el siguiente valor, en nuestro ejemplo: 0; se verifica la admisibilidad de ese argumento y se ejecuta el comando. Luego, se vuelve de nuevo al Interpretador y el proceso descrito arriba comienza de nuevo. La labor que, en nuestro ejemplo desempeña la Srta. Interpreter, requiere algo de tiempo. Este tiempo se ahorra si programamos directamente en

lenguaje máquina.
Por desgracia, el lenguaje máquina presenta el inconveniente de ser muy abstracto. Básicamente, al hombre le resulta difícil imaginarse cifras. Esta dificultad es la causa de la creación de los denominados "Lenguajes Superiores de Programación", tales como el LOGO, BASIC, etc., que operan con conceptos y no con cifras. Estos lenguajes presentan un compromiso en la comunicación entre el hombre y la máquina. Sin embargo, desgraciadamente ello trae consigo importantes inconvenientes en cuanto a la velocidad, a la necesidad de espacio en la memoria y, a menudo también, en cuanto a la posibilidad de programación.

Todos los lenguajes de programación, tales como el Cobol, Pascal, Fortran, etc, han de ser traducidos antes de ser ejecutados por el ordenador. Aquí se distingue entre Interpretador y compilador:
Un interpretador, como por ejemplo el del ordenador MSX, traduce paso a paso todos los comandos del programa, ejecutándolos simultáneamente. El interpretador es, por consiguiente, un traductor simultáneo; es decir, que durante el desarrollo del programa, cada comando se interpreta de nuevo otra vez. Por ello, las modificaciones en BASIC no plantean problemas.
Contrariamente a ello, un compilador traduce cada programa una sola vez, creando al mismo tiempo otro equivalente en lenguaje máquina. Sólo entonces puede ejecutarse el programa en lenguaje máquina. Por lo general, el proceso del compilador dura bastante, pero una vez realizado, el programa en lenguaje máquina se ejecuta a gran velocidad. Si se desea modificar el programa, deberá volver a compilarse la versión nueva. Esto hace que las modificaciones en tales programas sean de larga duración. En este libro le presentamos un compilador que traduce de lenguaje ensamblador a Código Máquina o Lenguaje Máquina. A este compilador se le llama ENSAMBLADOR.

Aquí, usted ya puede reconocer una ventaja fundamental del lenguaje máquina: los programas en lenguaje máquina alcanzan una velocidad de ejecución hasta 1000 veces mayor que los programas en BASIC.

Asimismo, los programas en lenguaje máquina escritos a mano para solucionar un problema especial son más rápidos que los programas en lenguaje máquina confeccionados mediante compilador. El comando RETURN en BASIC tiene un tiempo de ejecución de aprox. 0.6 milisegundos; el comando correspondiente RET en lenguaje máquina dura sólo 2.5 microsegundos. Ello hace que la orden RET en lenguaje máquina sea casi 240 veces más rápida, y en su equivalente para el comando Poke en lenguaje Máquina, hasta casi 1000 veces. Tales diferencias son importantes a la hora de, por ejemplo, ordenar y buscar en grandes cantidades de datos, desplazar contenidos en la memoria, como es necesario para el Scrolling o también para programas de textos. Además, la programación de gráficas complejas en BASIC es demasiado lenta; es decir, el lenguaje máquina se hace indispensable para programas de juegos y gráficas profesionales.

También existen otras ventajas:

Por regla general, los programas en lenguaje máquina son más cortos que los programas en BASIC, con lo cual se ahorra un importante espacio en la memoria. Tan pronto como usted haya escrito sus primeros programas en lenguaje máquina comprobará que un programa en lenguaje máquina de más de 500 Bytes ya es muy largo y que se pueden hacer muchas cosas con él. Por el contrario, para un programa en BASIC de características similares, se necesitaría mucho más espacio de memoria.

Otra ventaja del lenguaje máquina estriba en que unicamente con él pueden aprovecharse al máximo todas las posibilidades del ordenador. En primer lugar, con lenguaje máquina es más rápido programar, por ejemplo, datos de entrada o salida. Asimismo, con ayuda de programas propios se puede controlar a periféricos de entrada o salida, o también recibir datos de ellos. También, sólo en lenguaje máquina es posible desarrollar estructuras de datos propias, las cuales, generalmente, ahorran mucho más espacio que las dadas en Lenguaje BASIC. Grandes cantidades de datos, como las que aparecen -entre otras- en el tratamiento de textos, pueden de este modo ser almacenadas mejor en la memoria disponible.

Estos ejemplos deberían ser suficientes para dejar de manifiesto la necesidad del lenguaje máquina incluso en el caso de un ordenador MSX con muy buen lenguaje BASIC. Sin embargo, hay que decir que la programación en lenguaje máquina presenta un gran inconveniente.
El lenguaje máquina es el lenguaje de la CPU del ordenador y por lo tanto, el lenguaje más orientado hacia la máquina. Ello supone que el programador deberá pensar de un modo muy abstracto para llegar a entender este lenguaje. El ser humano tiende a pensar en palabras y asociaciones; es decir, que un lenguaje orientado hacia el hombre utiliza estructuras y conceptos claros. Ese no es el caso en el lenguaje máquina. Principalmente, la CPU sólo entiende cifras; o sea, que un programa para la máquina se reduce simplemente a una serie de números y no a una consecuencia de conceptos. En tal forma, la programación en lenguaje máquina para programas extensos sería casi imposible. Por ello, los ''Pioneros de la Informática'' ya desarrollaron una especie de lenguaje intermedio, capaz de hacer más inteligibles y claros los programas en lenguaje máquina. A este lenguaje se le denominó ENSAMBLADOR. El lenguaje ensamblador ordena a cada código de la máquina (o sea, a una cifra) una serie de símbolos. Tales símbolos se componen de:

1. Las palabras del comando: consisten generalmente en una abreviación de la palabra inglesa de comando, denominada también mnemónico.

2. El Operando que, por ejemplo, especifica las direcciones, constantes o similares (concernientes al comando).

De este modo, la confección de un programa en lenguaje máquina se simplifica al escribirlo en lenguaje ensamblador. Luego, este lenguaje ensamblador es traducido automaticamente al lenguaje máquina por el denominado "Programa Ensamblador". Un ensamblador de este tipo (un compilador para lenguaje ensamblador) es el que vamos a presentarle en este libro, para que usted pueda programar en ensamblador (aquí se refiere al lenguaje ensamblador).

Por este motivo sólo programaremos en lenguaje máquina de forma rápida y a modo de ejemplo, pasando a continuación a la programación en ensamblador y dejándole al Ensamblador (compilador) el trabajo de la traducción.

¡¡Ahora sí que empezamos de verdad!!

1.2 EL PRIMER PROGRAMA EN LENGUAJE MAQUINA

Para demostrarle que vale la pena aprender a programar en lenguaje máquina, le ofrecemos a continuación una comparación entre un programa en BASIC y su primer PROGRAMA EN CODIGO MAQUINA.
Por favor, introduzca las siguientes líneas de BASIC:

```
10 HL=0
20 A=42
30 VPOKE HL,A
40 HL=HL+1
50 IF HL<960 THEN 20
60 RETURN
```

Introduzca ahora en modo directo >SCREEN 0< y a continuación >GOSUB 10< y, ¡vea lo que ocurre!

El siguiente programa carga el programa en lenguaje máquina con la misma misión que el programa en BASIC:

```
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF00E
30 READ A
40 POKE I,A
50 NEXT I
60 DEF USR1=&HF000
70 END
80 DATA &H21,&H00,&H00,&H3E,42,&HCD,&HCD
```

```
90 DATA &H07,&H23,&H3E,04,&HBC,&H20,&HF5,&HC9
```

Ahora, vuelva a introducir >SCREEN 0< en el modo directo, cargue el programa en código máquina con >RUN<, llame al programa así cargado con >X=USR1(1)< y, ¡maravíllese!

Como habrá podido comprobar, tardan:

- Programa BASIC : aprox. 11 segundos
- Programa MAQUINA : aprox. 0.07 segundos

La longitud supone para:

- Programa BASIC : 76 Bytes
- Programa MAQUINA : 15 Bytes
  desde &HF000 hasta &HF00E.

Esperamos que no haya sufrido un shock demasiado fuerte con todas estas novedades. En los capítulos siguientes se lo explicaremos todo paso por paso.

Para la analogía de los programas:

BASIC	Lenguaje ENSAMBLADOR
10 HL=0	- LD HL,0000
20 A=42	- LD A,42
30 VPOKE HL,A	- CALL &H7CD
40 HL=HL+1	- INC HL
50 IF HL<960 THEN 20	- LD A,4
	- CP H
	- JR NZ,-11
60 RETURN	- RET

ACLARACION:

Línea 10: Aquí se coloca el valor de la VARIABLE HL o bien del REGISTRO HL del principio de la memoria de pantalla. (LD=Ingl.load=cargar)

Línea 20: En esta Línea se almacena o graba en A el ASCII del signo a proyectar ("*").

Línea 30: Aquí se escribe en la pantalla el valor de A, lo que provoca la aparición del signo.

Pruebe usted mismo a dar distintos valores en el modo directo para la dirección HL en la memoria de la pantalla (¡¡HL puede estar entre 0 y 1023!!) y valores entre 0 y 255 para A (A el el código ASCII). Por ejemplo: >POKE 10,65<.

Línea 40: Aquí, la variable HL, que contiene la dirección en la memoria de la pantalla, se aumenta en uno, para que poco a poco se vaya llenando toda la pantalla. (Ingl.:INCrease: aumentar).

Línea 50: Se pregunta si HL es mayor que 1023,; o sea, si se ha llegado al final de la memoria de pantalla Esta pregunta ha de dividirse en dos comandos en lenguaje máquina: LD (Ingl.load=cargar); carga de A con el valor comparativo. CP H: Compara con H (CP:Ingl.compare:comparar).JR (Ingl.jump relativ:salto relativo); NZ (Ingl.non zero:no cero). Así pues, se puede decir: ''Salta, cuando no cero''; aunque tal expresión no es del todo cierta. Más adelante se dará una aclaración más exacta.

Línea 60: RETURN o bien RET finaliza el subprograma.

A continuación le mostramos el listado en lenguaje ensamblador para darle un ejemplo:

LISTADO ENSAMBLADOR para el programa en lenguaje máquina

```
Direcc.Code  Núm.Línea  comando ensamblador   Comentario

F000 2100C0      10        LD  HL,0        ;  Inicio memoria
pantalla
F003 3E2A        20        LD  A,&H2A      ; =42 es el ASCII de
"*"
F005 CDCD07      30        CALL &H7CD      ; corresponde a la
Rutina VPOKE
F008 23          40        INC HL          ; HL=HL+1
F009 3E04        50        LD  A,4
F00B BC          60        CP  H           ; ¿H>4?
F00C 20F5        70        JR  NZ,&HF003; ¡no, pues, de
nuevo!
F00E C9          80        RET             ; Vuelta al Basic
```

Esperamos haber despertado su curiosidad, ya que a partir de ahora, vamos a pasar al tratamiento sistemático del lenguaje máquina y a explicar los ejemplos dados arriba.

1.3 SISTEMAS NUMERICOS

En el capítulo anterior utilizamos el signo "&H" como indicador de un número en el sistema Hexadecimal (Hexadecimal - 16). ¿Qué significa eso?
Para la realización de dispositivos electrónicos de cálculo existían dos posibilidades de representación de las cifras.

Analogía: En un ordenador analógico, una cifra se presenta mediante la correspondiente alta tensión, p.ej. 1=1 voltio y 100=100 voltio. Según esto, un reloj de muñeca con manecillas sería un reloj analógico. El aumento continuado del tiempo corresponde -es analógico- al número de vueltas de las agujas.

Digital: En los ordenadores digitales, la idea no se basa en la medida de la tensión, sinó en observar solamente los dos estados: corre flujo y no corre flujo. Digital significa: Exposición de tamaños con ayuda de cifras. Los estados ENCENDIDO y APAGADO corresponden por lo tanto a las cifras 1 y 0.

Con esto, un ordenador digital sólo dispone de dos cifras. Con ayuda de ambas se lleva a cabo la representación numérica en el ordenador.
Para resolver según que problemas establecidos, tiene mayor sentido, bajo ciertas condiciones, trabajar con un ordenador analógico (p.ej. control de la máquina). Sin embargo, si han de resolverse distintos problemas en el ordenador, el ordenador digital desplaza de inmediato al ordenador analógico, ya que nos es imposible programar un ordenador analógico con nuestros conocimientos de programación habitual. Esto significa que los ordenadores domésticos y personales son ordenadores digitales y que, por lo tanto, tratan datos en el sistema binario con las cifras 1 y 0.
Para el programador es importante conocer los siguientes sistemas numéricos:

1. Sistema Decimal
2. Sistema Binario
3. Sistema Hexadecimal

Los sistemas numéricos son esquemas de cifras ordenadas según un determinado principio. Cada cifra puede ser convertida a otro sistema numérico. En todos los sistemas numéricos, el valor posicional de una cifra aumenta de derecha a izquierda.
Para explicar los otros sistemas numéricos, partiremos del conocido sistema decimal.

El Sistema Decimal

```
Millar   Centena   Decena   Unidad   -   Valor Posicional
  7        3         5        6     -   Cifras
 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
  ¡El valor posicional aumenta de derecha a izquierda!
```

Potencia	Número	Denominación
10^0	1	U - nidad
10^1	10	D - ecena
10^2	100	C - entena
10^3	1000	M - illar
10^4	10000	D - ecena de millar
10^6	1000000	Millón

El número decimal 1335 también se puede escribir del siguiente modo:

1335 significa : 1M + 3C + 3D + 5U - La posición del valor más bajo (Unidad)
435 significa : 4C + 3D + 5U - está a la derecha.

1335 es : $1*1000 + 3*100 + 3*10 + 5*1$

$$1335 = 1*10^3 + 3*10^2 + 3*10^1 + 5*10^0$$

Una potencia con exponente 0 se define como 1.

p.ej.: $10^0 = 1$, $2^0 = 1$, $x^0 = 1$

El Sistema Binario

El sistema Binario se basa en el mismo principio. La única diferencia radica en que el valor de posición de cada cifra no se expresa por potencias de diez, sinó por potencias de dos.
La base del sistema Binario es 2.

Binario 10101101 = Decimal 173

7	6	5	4	3	2	1	0	
2	2	2	2	2	2	2	2	- valor posicional
1	0	1	0	1	1	0	1	- Cifra

$$173 = 1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$$

$$173 = 1*128 + 0*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*1$$

De momento, usted ya ha aprendido a pasar del Sistema Binario al Sistema Decimal. Naturalmente, este proceso también se puede hacer a la inversa. Para aclarar el proceso inverso, observemos la cifra decimal 173 antes calculada. Pensemos qué potencia de dos hay aún en esta cifra. Una ayuda: En principio, el Sistema Binario puede aplicarse a cifras de n-posiciones. Sin embargo, en el ámbito del ordenador sólo se utilizan cifras binarias de 8-posiciones. Pueden aparecer las siguientes potencias de dos.

	7	6	5	4	3	2	1	0
Potencias de dos	2	2	2	2	2	2	2	2
valor decimal	128	64	32	16	8	4	2	1

En este caso, 2^7=128 es la potencia de dos más alta. Ahora calculamos la diferencia entre 173 y 128. El resultado es 45. Con este Resto se procederá igual que arriba. Volvemos a buscar la potencia de dos más alta contenida en ese valor. Utilizando la tabla, se encuentra facilmente y resulta 2^5=32. Finalmente, volvemos a calcular la diferencia: (45-32=13).

El proceso descrito se sigue aplicando hasta que el Resto da cero.

2^3=8 (13-8=5)
2^2=4 (5-4=1)
2^0=1 (1-1=0)

Así hemos obtenido las siguientes potencias de dos:
2^7 , 2^5 , 2^3 , 2^2 y 2^0
Debajo de cada potencia obtenida escribiremos un uno y debajo de la potencia inexistente, un cero:

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

  1  0  1  0  1  1  0  1  =  173
```

La cifra decimal 173 viene expresada por lo tanto en el Sistema Binario por 10101101. En lo sucesivo, representaremos las cifras binarias anteponiendo el signo &B.

p.ej. 173= &B 10101101

Bit y Byte

Un BIT es la unidad de información más pequeña a partir de la cual se componen todas las demás informaciones. BIT es la abreviación de "binary digit", lo que equivale a decir: cifra binaria. Se habla de un BIT activado cuando el BIT tiene el estado 1, o de un BIT desactivado cuando tiene el estado 0.
Los ordenadores MSX tienen un procesador de 8-BITS; es decir, puede tratar largas cifras binarias de 8-BITS, lo cual comprende los valores decimales entre 0 y 255.
Número binario:

```
1 0 1 1 0 1 1 1

a d a a d a a a   a=BIT activado; d=BIT desactivado

7 6 5 4 3 2 1 0   número de BITs.
```

A cada BIT y a cada cifra se le asigna un número de BIT. El BIT con el valor posicional más bajo, es decir, el que está más a la derecha, tiene el número 0. La numeración aumenta de derecha a izquierda. El número de BIT corresponde al exponente de la potencia de dos, que determina el correspondiente valor posicional.

En el ordenador, a veces tiene sentido imaginarse los estados del BIT como un interruptor.

INTERRUPTOR ABIERTO = 1
INTERRUPTOR CERRADO = 0

En un número de 8 interruptores se presentan valores de 0-255, o sea, 256 estados de interruptor.
El conjunto de ocho interruptores (BITs) recibe el nombre de BYTE. Un BYTE puede ser colocado por el ordenador en una posición de la memoria. ¿Pero, cómo se memorizan o graban números superiores a 255?. Para ello se dividirá el número en dos mitades: el LOW Byte (engl.low: bajo; Byte de valor bajo) y el HIGH Byte (engl.high: alto; Byte de valor alto). Estos Bytes son entonces colocados en dos posiciones de la memoria contiguas.

EL HIGH Byte y el LOW Byte se calculan de la siguiente forma:

Cifra dividida por 256=(HIGH Byte)+Resto
El resto de la división será el LOW Byte.

Recordemos: El número 255 es el máximo valor expresable en un Byte, ya que se compone de 8 BITs seguidos.

Ejemplo: El número 34065 ha de ser descompuesto en un LOW y en un HIGH Byte.

34065 / 256=133 Resto 17
34065 =133*256+17

133=High Byte
17=Low Byte

Las fórmulas generales escritas en BASIC serían:

1. HB=INT(NÚmero/256) HB=High Byte
   LB=NÚmero-HB*256 LB=Low Byte

Esta fórmula se puede aplicar a números de valor cualquier valor.

2. HB=Número / 256 HB=High Byte
   LB=Número MOD 256 LB=Low Byte

La segunda fórmula es aplicable a números de valores comprendidos entre -32768 y 32767.

Con ello, un número comprendido entre 256 y 65535 y colocada en la memoria, necesitará 2 Bytes.
Para simplificar la representación de números, colocados de esta forma en la memoria, es mejor adoptar otro sistema de cifras.

El Sistema Hexadecimal

La base del sistema Hexadecimal es 16.
Recordemos:

La base del sistema Decimal es 10
La base del sistema Binario es 2

Para la representación de cifras cuyo valor es superior a 10, se utilizan en el sistema hexadecimal las letras A hasta F.

Sistema Decimal:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,...

Sistema Hexadecimal:
0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F,10,11,12,...

Primero, convertimos números hexadecimales en números decimales:

Potencia	Valor
0	
16	1
1	
16	16
2	
16	256
3	
16	4096

```
&H3ABF=3*16^3 + 10*16^2 + 11*16^1 + 15*16^0
&H3ABF=3*4096 + 10*256  + 11*16    + 15*1
&H3ABF=12288  + 176     + 15
&H3ABF=15039
```

Otro ejemplo:

```
&H1A3E=1*16^3 + 10*16^2 + 3*16^1 + 14*16^0
&H1A3E=1*4096 + 10*256  + 3*16    + 14*1
&H1A3E=4096   + 2560    + 48      + 14
&H1A3E=6718
```

Ahora efectuaremos la conversión de números decimales a números hexadecimales:

El proceso de la conversión es igual que el descrito en las páginas anteriores. Supongamos que queremos pasar la cifra decimal 45380 al sistema hexadecimal:

1. paso: Pensamos que potencia mayor de 16 puede estar comprendida en este número (tenemos a nuestra disposición las cifras de las anteriores tablas de conversión: hexadecimal a números decimales).

2. paso: Dividimos la cifra antes mencionada (45380) por este valor (4096) y convertimos la cifra decimal así obtenida en cifra hexadecimal.

45380/4096=11 Resto 324
decimal=11 => hexadecimal B

3. paso: A continuación se realiza el mismo proceso con el Resto (324). Este lo dividimos por la siguiente cifra más pequeña y adecuada de la tabla, o sea, por 256.

324/256=1 Resto 68
decimal=1 => hexadecimal 1

Los cálculos descritos arriba se continúan hasta que el resultado de la división dé como Resto 0.

68/16=4 Resto 4
decimal=4 => hexadecimal 4

4/1=4 Resto 0
decimal=4 => hexadecimal 4

Nuestra cifra convertida se llama &HB144

La ventaja del sistema hexadecimal estriba en que se pueden leer directamente el Byte alto y el Byte bajo.
Para &H3ABF será:

- El Byte alto se compone de las dos primeras cifras hexadecimales (3 y A). Tiene el valor decimal de (3*16^1+10*16^0)=58

-El Byte bajo se compone de las dos últimas cifras hexadecimales (B y F). Tiene el valor decimal de (11*16^1 + 15*16^0)=191

Ahora, introduzca los siguientes datos:

```
PRINT PEEK(&H1D),PEEK(&H1E)
```

En las dos direcciones &H1D y &H1E está la dirección de salto, a la que bifurca el sistema operativo, cuando una rutina, situada por ejemplo en un módulo conectable, ha de ser llamada. Para una dirección de salto es posible un valor de 0 a 65535 (o sea, hasta &HFFFF). Esta cifra se memoriza con la ayuda del Byte alto y del Byte bajo. Ahora, queremos calcular la dirección de salto. Con el comando anterior de BASIC, obtenemos de la dirección &H1D el valor 23 y de la dirección &H1E, el valor 2. En cifras decimales, la dirección de salto resulta por lo tanto de 2*256+23=535.
Ahora queremos realizar el mismo cálculo en el sistema hexadecimal:
23=&H17 y 2=&H2, como podrá comprobar con facilidad. El valor de la dirección de salto lo obtenemos escribiendo uno a continuación del otro el Byte alto y el Byte bajo: 535=&H217.
Así pues, resulta igual de fácil separar una cifra hexadecimal Byte alto y bajo, así como componerla a partir de Bytes alto y bajo. Por regla general, el Byte bajo de una cifra se halla en la dirección más baja de la memoria, y a continuación le sigue el Byte alto.
Con ésto, usted ya ha aprendido la primera ventaja del Sistema Hexadecimal. Además, también es muy sencillo pasar del Sistema Decimal al Sistema Hexadecimal. Para ello, se subdivide una cifra binaria en dos bloques, cada uno de 4 BITs. Al bloque del 0 al 3er Bit se le llama Low Nibble y al otro bloque, del 4o al 7o Bit, High Nibble. Cada Nibble corresponde exactamente a una cifra hexadecimal. Ello es fácil de comprender, pues una cifra binaria de cuatro Bits puede coger como máximo el valor 15 (15=8+4+2+1). Todos los valores de 0 a 15 también pueden ser representados por una cifra hexadecimal (0, 1,...9, A, B, C, D, E, F). Observemos un ejemplo:

```
1 1 0 1  1 0 0 1
High N.  Low Nibble
8+4+1    8+1
  13       9
  &HD      &H9
```

Así pues: &B11011001=&HD9

Con un poco de práctica, usted podrá leer directamente de una cifra de 4-Bits su cifra hexadecimal correspondiente y viceversa. Para ello le ayudará la siguiente tabla:

Sistema Binario	Sistema Hexadecimal	Sistema Decimal
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	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

De igual modo funciona la conversión del Sistema Hexadecimal al Sistema Binario. Cada número hexadecimal se sustituye por la combinación de 4-Bits, p.ej., &HC7=&B1100 0111.

La comprensión de la conversión de los distintos sistemas numéricos es una base para la programación en lenguaje máquina.

Ejercicios:

1. Complete la siguiente tabla:

Decimal	Binario	Hexadecimal
130	?	?
?	&B1001001	?
57312	---	?
?	---	&HC0B6
?	?	&H37

2. El valor 37315 ha de ser memorizado a partir de la posición de memoria &HF000. Calcule el Byte alto y el Byte bajo y dé los comandos en BASIC, con los cuales podrá ser memorizado el número.

3. Hay una dirección de salto importante a partir de la posición de memoria &H0009 ¿Qué valor tiene?

Soluciones:

1.

Decimal	Binario	Hexadecimal
130	&B10000010	&H82
147	&B10010011	&H93
57312	---	&HDFE0
49334	---	&HC0B6
55	&B00110111	&H37

2. Byte alto=145=&H91; Byte bajo=195=&HC3
   POKE &HF000,&HC3: POKE &HF001,&H91

3. Byte bajo=PEEK(&H0009), Byte alto=PEEK(&H000A)
   Dirección de salto=&H2683

En el Epílogo del libro, encontrará una tabla compuesta por números desde 0-255 (1-Byte) en los tres sistemas numéricos.

1.4 ESTRUCTURA DEL ORDENADOR

Si queremos dedicarnos a la programación en lenguaje máquina, antes deberemos hacernos una idea de la estructura y de la organización interna del ordenador. En este capítulo intentaremos desarrollar una imagen que se adapte a nuestras necesidades.

Usted ya sabe que posee un ordenador de 64K (o de 48K, 32K, 16K) (K-Kilobyte=1024 Bytes). Ello significa que la capacidad de memoria del ordenador es de 64*1024=65536 Bytes. Como un Byte se compone de 8 Bits unidos, y con ello se representan los datos de la memoria interna, resulta que su ordenador se compone de casi 64*1024*8 Bits, o sea, aprox. 0.5 millones de interruptores, que pueden estar tanto encendidos como apagados. Sin embargo, tal expresión no tiene sentido para el trabajo concreto del ordenador. Por ello, se han unido 8 bits en un Byte. Estos 64*1024 Bytes se hallan en la memoria RAM del ordenador. RAM significa: Random Access Memory, en español, memoria de lectura y escritura o también, memoria de trabajo. Los 65536 Bytes de la RAM están numerados desde &H0000 hasta &HFFFF. El número correspondiente al Byte es su dirección. Esta dirección viene dada normalmente en una cifra hexadecimal. Podemos acceder directamente a la RAM a partir del BASIC. Aquí sirven los comandos >PEEK< y >POKE<. >PEEK(Dirección)< lee el valor de los Bytes situados en la dirección dada y >POKE Dirección, valor< pone el valor indicado la dirección dada. Como a cada dirección le corresponde un Byte, y como un Byte se compone de 8 BITs, o sea, que está entre 0 y 255 (&H00-&HFF), el valor a memorizar también deberá estar dentro de este ámbito. Naturalmente, la dirección también deberá hallarse entre &H0000 y &HFFFF.

La memoria RAM sirve para almacenar los programas introducidos por usted en el ordenador. Además, también hay a su disposición espacio en la memoria para almacenar programas en lenguaje máquina de aquellas versiones que

poseen más de 32K de RAM. Sin embargo, no se puede acceder a este espacio adicional de RAM simplemente a partir del BASIC. En la RAM superior también se encuentran algunas rutinas del sistema operativo así como información importante, como por ejemplo, colores actuales, teclas de función, buffer de entrada, etc.. Por este motivo, los POKEs incontrolados, que varían el contenido de la RAM, podrían colapsar el ordenador. Por ejemplo, jamás intente >POKE &HFF0C,199<.

Por consiguiente, la distribución de la RAM es la siguiente:

&H0000 - &H7FFF para programas máquina (Versión 64K)
&H8000 - &HBFFF para programas BASIC (Versión 32K)
&HC000 - &HF37F para programas BASIC (todas versiones)
&HF380 - &HFFFF utilizado por el sist.(todas versiones)

Debido a que la distribución de la RAM depende de su tamaño, consulte el manual para la exacta distribución de la memoria de su ordenador.
Podemos limitar el espacio reservado para programas en BASIC mediante el comando >CLEAR tamaño del área de cadenas, dirección<. Así, disponemos del espacio de la dirección dada en el comando >CLEAR< hasta &HF37F para almacenar nuestros programas en código máquina. En nuestrtro ejemplo hemos reservado mediante >CLEAR 200,&HEFFF< el espacio de &HF000 a &HF37F para nuestro programa máquina, grabándolo luego desde &HF000 con ayuda de los comandos >POKE<.
Usted se extrañará al ver que sólo se utilizan algo más de 3K de la RAM para rutinas del sistema:
¿Dónde están el Interpretador y el Sistema operativo, que nos hacen posible el programar en BASIC?

Supone bien:

Existe además otra memoria importante, la ROM (Read Only Memory=Memoria sólo de lectura, o memoria de valor fijo). En la ROM se hallan todos los datos y programas que nos permiten programar en BASIC sin dificultad. Puesto que la ROM es una memoria de valor fijo, se graba con datos y programas (en lenguaje máquina) y se instala en el ordenador

antes de salir de fábrica.
Los ordenadores MSX poseen dos ROMs de 16K (a veces 3, en un programa adicional incorporado, p.ej, Sony), cuyas direcciones se superponen con las de la RAM. Ello es necesario, pues el Procesador Z80 posee unicamente 16 líneas para direccionamiento; es decir, que la dirección de un Byte no puede ser más larga que 16 Bits. Con 16 Bits queda perfectamente cubierto el espacio que va desde &H0000 hasta &HFFFF.
Al conectar el ordenador se activa la ROM en el espacio de memoria desde &H0 hasta &H7FFF. La RAM se encuentra entre las direcciones &H8000 y &HBFFF. En algunas versiones, en ese espacio se halla la ROM con los programas incorporados. En el bloque superior de 16K, &HC000 hasta &HFFFF, en principio, siempre se ubica la RAM. Para leer la RAM inferior, habrá que comunicar antes a la CPU que lea la RAM, luego, ya pueden utilizarse las mismas direcciones que para la ROM.
Las ROMs ocupan los siguientes espacios:

1. ROM I	&H0000 - &H3FFF	Sistema operativo
2. ROM II	&H4000 - &H7FFF	BASIC
3. ROM III	&H8000 - &HBFFF	Programa adicional (en el caso de existir)

El sistema operativo contiene, como su nombre indica, las rutinas, basicamente necesarias para que el ordenador trabaje. Su función es dirigir los periféricos, administrar datos, mover datos, etc. En el área de la ROM también se encuentran las copias de las rutinas del sistema existentes en la RAM. Al conectar o hacer un Reset del ordenador, tales rutinas son copiadas de la ROM a la RAM. Además, en la ROM se halla la memoria de signos o caracteres (&H1BBF-&H23BC), donde cada signo del ordenador viene representado en una matriz de bits (o sea, 0-ningún punto, 1-punto).
Los comandos BASIC programados por nosotros se ejecutan por los programas existentes en la ROM de BASIC. La tabla de palabras de comandos está, p.ej, a partir de &H3A72.
El procedimiento que permite al ordenador acceder a más de los 64K de la memoria realmente direccionables, se denomina

Bank-Switching (significa algo así como: conmutar áreas). Los ordenadores MSX poseen 4 "Bancos", descritos como Slots en el manual. La denominación "Slot" viene dada debido a que estos módulos conectables también pueden ser direccionados mediante este procedimiento.
El Slot 0 se asigna a las ROMs incorporadas (sistema operativo, BASIC y programa adicional). En las versiones de 16K, las 16K de RAM también se cuentan como Slot 0. Con ello, el Slot 0 queda lleno, pues cada Slot puede contener sólo 64K. Normalmente, el Slot 2 se asigna a la RAM. Finalmente, los Slots 1 y 3 corresponden a las dos módulos enchufables.
A cada bloque de 16K de direcciones de memoria unicamente se le puede ordenar uno de estos cuatro Slots.

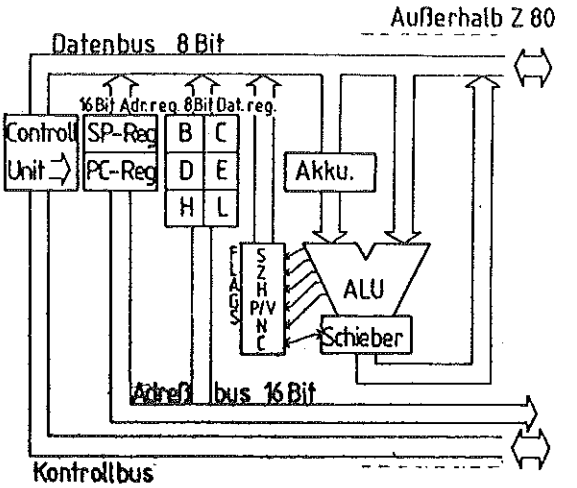
Dirección	Slot 0	Slot 1	Slot 2	Slot 3
0-&H3FFF	ROM Sist. operativo*	Lugar conector 1	(RAM)	Lugar conect 2
&H4000-&H7FFF	ROM BASIC *		(RAM)	
&H8000-&HBFFF	ROM Programa personal		RAM *	
&HC000-&HFFFF	(RAM)		RAM *	

La asignación de la RAM depende de la versión del ordenador. Si usted programa en BASIC, se seleccionan los espacios indicados con "*" (esto también depende de cada versión).
Además de la ya mencionado RAM, su MSX también posee un espacio denominado RAM de Video de 16K. Esta RAM de Video (=VRAM) contiene, para cada modo gráfico, el contenido de la pantalla, informaciones sobre Sprites y colores. El procesador no tiene acceso directo a este espacio RAM. Es administrado por el Video Chip (VDP-Video Display Prozessor). Con ayuda de los comandos >VPOKE< y >VPEEK< se puede acceder directamente a este espacio desde el BASIC. Internamente, se efectúa una lectura o escritura de ese espacio mediante elementos de entrada y salida, en BASIC, >INP< y >OUT<. ¡Si todos los Slots están cubiertos, usted tiene la posibilidad de disponer de un espacio de memoria de 272K (incl.VRAM)!. Teóricamente, incluso hasta 1 Megabyte)

Hasta aquí, todo sobre las memorias del ordenador MSX.

Naturalmente, nuestro ordenador también contiene otros muchos ICs, como el Procesador Z80 o el Chip de Sonido (PSG-Programmable Sound Generator). En el siguiente capítulo describiremos detalladamente el Procesador Z80. Si tiene interés en conseguir más información sobre cada uno de los Chips y su utilización, le aconsejo que lea el libro "Consejos y Trucos para el MSX".

2.1 ESTRUCTURA DE LA CPU



Datenbus: bus de datos
Ausserhalb Z 80: fuera del Z80
16Bit Adr.reg: registro direc.16Bit
8Bit Adr.reg: reg.direcc.8Bit
Schieber:Cursor
Kontrollbus: Bus de control

Los ordenadores MSX poseen una CPU Z80 (Unidad Central). Recordemos que se puede calificar a la CPU como "el cerebro" del ordenador. De este modo, es correcta la denominación del MPU (MPU: Ingl. Micro Prozessing Unit - Microprocesador).

En este capítulo nos ocuparemos de la estructura y de la función de cada uno de los elementos que componen la CPU. El gráfico de la página anterior nos ayudará a comprender la vida interna de la Unidad Central. Observemos el diagrama de izquierda a derecha:

1. CU (CU: Ingl. Control Unit- Unidad de Control)
   Todos los movimientos del ordenador son controlados y dirigidos por la CU.

2. Bus de Control
   El Bus de Control es el "brazo largo" del CU. A través de él se dirigen y supervisan los elementos fuera de la CPU.

3. Indicador de la pila SP (SP: engl. Stack Pointer)
   Con ayuda del SP se pueden guardar en la RAM datos y direcciones de retorno de los subprogramas.
   Puesto que en el SP se almacenan direcciones, éste es un registro de 16-Bits.

4. Contador de Programa PC (PC:engl.Programm Counter).
   El PC indica la dirección de memoria en la que se halla cada comando que ha de ser ejecutado.

5. Registro B hasta L (registro de Registros)
   La CPU posee varios Registros, donde son almacenados los datos.

6. Flags (Flag: engl. flag - Bandera; aquí, mejor Indicador característico). Los Flags sirven como indicadores de determinados incidentes que acontecen en la CPU durante las operaciones del ordenador. Los Flags pueden estar activados (flecha arriba), o no; es decir, desactivados (flecha abajo).

7. Bus de direcciones (se halla fuera de la CPU)

El Bus de direcciones realiza el enlace con otros MPU del ordenador. Indica el lugar de memoria en la ROM o en la RAM, cuyo contenido ha de ser leído o escrito. El Bus de direcciones tiene una extensión de 16-Bits. Ello es necesario para direccionar el espacio de memoria de 64K.

8. Bus de Datos (se halla fuera de la CPU)
   Los Buses de datos "trasladan" los datos que han de ser leídos o escritos. El Bus de datos apunta así hacia la dirección de los datos. El Bus de Datos tiene una extensión de 8-Bits.

9. Acumulador (lat.Akkumulator)
   El Acumulador (Acu) es el Registro más importante de la CPU. También se le puede calificar como el Registro de cálculo.

10. ALU (ALU: Ingl.Arithmetical Logical Unit -Unidad aritmética lógica, Unidad de cálculo).
    La ALU realiza diversas operaciones aritméticas y lógicas. Los Flags son influenciados en función del resultado de las operaciones.

11. Cursor
    El cursor dirige las rutinas de rotación y traslación

Como ya hemos mencionado en el punto 5., la CPU contiene varios Registros. Para comprender mejor sus funciones, los hemos dividido en cinco grupos.

1. El Acumulador
2. Los Flags
3. Los "seis asociables" registros de 8 Bits
4. Los "cuatro inseparables" registros de 16 Bits
5. Registro Interrupt/refresh

2.2 EL ACUMULADOR

El Acu, o bien, el registro A es el registro más importante del Z80. La mayoría de los comandos aritméticos y lógicos utilizan este registro. Cuando se ejecuta un comando de comparación, se efectúa fundamentalmente con el contenido del Acu. Al igual que todos los registros, excepto SP, PC, IX y IY, el registro A es un registro de 8-Bits.

2.3 LOS FLAGS

El registro Flag o registro F tiene una extensión de 8-Bits (como A,B,C,D,E,H y L). Sin embargo, también tiene otras funciones>. En el registro Flag se utilizan los distintos Bits como indicadores para determinados sucesos originados durante las operaciones de la ALU (Unidad de cálculo). Cada uno de los Bits del registro F tienen los siguientes significados:

```
S  Z     H     P/V  N  C  - Denominación del Flag
7  6  5  4  3  2     1  0  - Número de Bit
```

```
C   - Carry-acarreo
N   - Sustracción
P/V - Paridad/Desbordamiento
H   - media transferencia
Z   - Zero-Cero
S   - Sign-Signo
```

Flag C (Bit 0)
Si en el transcurso de una suma o resta se produce un acarreo, se coloca este Bit, sinó, se borra.

Flag N y H (Bit 1, Bit 4)
Estos Flags son utilizados por el Z80 internamente. Para nuestros fines, carecen de importancia.

Flag P/V (Bit 2)

Este Flag tiene una doble función:
Se activa cuando tiene lugar un desbordamiento (V) (ingl.:overflow), de lo contrario, se borra. Además, indica la paridad (P) de un Byte.

Flag 2 (Bit 6)
Este Flag se activa cuando el resultado de una resta da cero, sinó, se borra. también se coloca este Bit en una comparación, si existe igualdad.

Flag S (Bit 7)
Este Bit se coloca cuando el resultado de una suma o resta es mayor que 127. Como veremos más adelante, en la aritmética de la CPU, los Bytes mayores que 127 significan números negativos.

los Bits 3 y 5 del registro Flag no se utilizan.

2.4 LOS "SEIS ASOCIABLES" REGISTROS DE 8-BITs

Pertenecen a este grupo seis registros de 8-Bits:
B, C, D, E, H, L
Estos registros tienen la facultad de formar pares de registros, con el fin de crear un registro de 16-Bits de estensión. En C, E, y L se memoriza el Low Byte, y en B, D, H, el High byte.

B/C (Byte Counter)
El Registro B, o bien, par de registros-BC, se utiliza a menudo como contador, p.ej.,en los bucles.

El par de registros-DE es de libre utilización.
Este par de registros se utiliza a menudo para guardar temporalmente direcciones y datos.

H/L (High/Low)
El par de registros-HL se utiliza a menudo para guardar direcciones

Tiene sentido el habituarse a denominar los registros de esta manera, ya que algunos comandos utilizan los registros siguiendo la nomenclatura antes citada. Principalmente, también se pueden utilizar los registros L o E como contadores.
Una particularidad del Z80 radica en que todos los registros antes citados se presentan de nuevo con la misma función. Este doble juego de registros está a nuestra disposición. De todos modos, sólo se puede utilizar un juego de registros al mismo tiempo.

2.5 "LOS CUATRO INSEPARABLES" REGISTTROS DE 16-BITs

A este grupo pertenecen cuatro registros de 16-Bits:
SP, PC, IX, IY.
El Registro SP es un registro fijo; es decir, no puede descomponerse en dos registros de extensión 8-Bits. El Puntero de pila (SP) indica cada una de las direcciones en la memoria, en las cuales hay direcciones de retorno o datos de memoria intermedia. La dirección se refiere a una posición de memoria situada en un espacio de la RAM, denominada Stack o pila. La utilización de los Stacks para el almacenamiento de datos se lleva a cabo de la siguiente manera:
Al conectar el ordenador se coloca el SP en la dirección más elevada de la pila (&HF000). Si hay que colocar un Byte en la pila, entonces el SP será reducido automaticamente en 1, almacenándose así ese Byte en la dirección indicada por el SP. Así pues, el SP siempre indica el último apunte en la pila. Al "recoger de la pila", el proceso se realiza de forma inversa. Primero, se lee el Byte en la dirección indicada por el SP, a continuación se incrementa el SP en uno. De este modo, es posible ensertar llamadas a los subprogramas.

El PC es un registro especial. Desde un programa no puede ser grabado ni modificado. El PC se administra internamente y siempre indica la dirección del comando actual.

Los registros IX/IY se utilizan fundamentalmente para almacenar direcciones o direcciones relativas. Al igual que todos los registros descritos en 2.5, estos dos registros también pertenecen a los registros de 16-Bits. En éstos no es posible acceder por separado al Byte alto y bajo (como ocurre con BC, DE, HL). El uso del Registro Indice se asemeja al par de registros-HL. Veremos la diferencia cuando hablemos del direccionamiento Indexado.

2.6 REGISTRO INTERRUPT / REFRESH

Estos dos registros corresponden a la CU.

I- o bien, registro Interrupt
(engl.Interrupt: Interrupción)
SI tiene lugar una interrupción; es decir, una interrupción del programa, este registro de 8-Bits contiene la parte superior de la dirección a la cual ha de realizarse la bifurcación. La parte inferior viene suministrada por el elemento del ordenador que ha ocasionado la interrupción.

R-o bien, registro Refresh (ingl.:refresh:refresco)
Este registro es utilizado por el Hardware como contador, con el fin de refrescar, a intervalos regulares, el contenido de la memoria dinámica. Con ello se impide que se pierdan las informaciones memorizadas. La pérdida de datos se impide mediante la continua recarga del mismo contenido de la memoria.

La CPU ejecuta un comando de la siguiente forma:
Se lee el Byte en la dirección que indica el PC, y el PC aumenta en uno; es decir, indica hacia el Byte siguiente. El Byte leído es interpretado como un comando. A continuación, se procede a la lectura eventual de los datos pertenecientes al comando (el PC vuelve a aumentar). Acto seguido se ejecuta el comando y el proceso comienza de nuevo.

Una vez que ya conocemos la CPU Z80, pasaremos a ocuparnos de los verdaderos comandos de la máquina.

CAPITULO III: E L J U E G O DE C A R A C T E R E S DEL Z 8 0

3.1 INTRODUCCION: ENTRADA DE PROGRAMAS EN LENGUAJE MAQUINA

Para poder empezar enseguida a probar comandos con el Z80, deberemos antes hacernos una idea de como se introduce y memoriza un programa en lenguaje máquina a partir del BASIC. Al igual que en BASIC, a cada número de línea le corresponde un comando, en lenguaje máquina, a cada comando le corresponderá una dirección.

BASIC		LENGUAJE MAQUINA		
Núm. Línea	Comando	Dirección	Comando	Código
9	HL=HL+1	&HF009	INC HL	&H23
10	RETURN	&HF00A	RET	&HC9

-En BASIC, a cada núm. de línea le corresponde un comando.

-En CODIGO MAQUINA, a cada comando le corresponde una dirección.

Así pues, un programa máquina es una sucesión de códigos de comandos, situados en la memoria en direcciones sucesivas. Mediante el BASIC, y con ayuda de los comandos >POKE<, podemos escribir los códigos en las correspondientes direcciones. De este modo, los programas son llamados con el comando >USR<. Antes, debe ser fijada la dirección de partida del programa máquina con >DEFUSR<. Generalmente, la dirección de partida es la dirección del espacio de la memoria, que contiene el primer código máquina. Para que no se escriba nada por equivocación en nuestro programa máquina, deberemos reservar su espacio de memoria con el comando >CLEAR<. Reservaremos siempre el espacio desde &HF000 hasta &HF37F mediante >CLEAR 200,&HEFFF<. De este

modo dispondremos de &H380 Bytes (aproximadamente 1K) para programas máquina. Un programa BASIC típico para cargar programas en lenguaje máquina tiene la siguiente estructura:

```
10 CLEAR 200,&HEFFF
20 FOR I=Dirección inicio TO Dirección final
30 READ A
40 POKE I,A
50 NEXT I
60 DEFUSR 1=Dirección inicio
70 END
80 DATA...
90 DATA...
    .
    .
    .
```

En las líneas DATA estan los códigos que compondrán el verdadero programa máquina. La Dirección final (V=Variable; en lo sucesivo, escribiremos siempre esta abreviatura detrás de aquellas palabras que sean Variables) deber ser naturalmente mayor que &HEFFF, y la Dirección de inicio (V), ser menor que &HF380. La llamada del programa cargado se realiza mediante >X=USR1 (Parámetro)<.
Por regla general utilizaremos &HF000 como dirección de inicio. La dirección Final (V) resulta de la dirección de inicio (V) más la longitud del programa en Bytes, menos 1. La longitud del programa corresponde al número de entradas en las líneas DATA.

Con ayuda de los comandos >DEFUSR N.<y >USR N.(Parámetro)<,es posible definir más de 10 llamadas de programa máquina distintas en el MSX BASIC. A partir de ahora, siempre utilizaremos >USR1<. Más adelante daremos a conocer el significado del parámetro entre paréntesis que aparece siempre detrás del comando >USR<. Hasta entonces, puede colocarse en su lugar cualquier número o Variable. Siguiendo esta relación, también es discutible, hasta que punto el comando >USR< es realmente una función; o sea, por qué se lleva a cabo la llamada a través de >Variable=USR1(Parámetro)<, o bien, >PRINT USR1(Parámetro)<.

Para la entrada de programas pequeños es práctico el siguiente programa en BASIC:

```
10 CLEAR 200,&HEFFF:CLS
20 INPUT "Direccion Inicio &H";A$
30 ST=VAL("&H"+A$):IF ST<&HEFFF THEN BEEP:GOTO 20
40 AD=ST
50 IF AD>=&HF380 THEN BEEP: PRINT
"Espacio sobrepasado":END
60 PRINT"Direccion &H";HEX$(AD);" : ";
70 INPUT"VALOR &H" ;A$
80 IF A$=" " THEN DEFUSR1=ST:END
90 W=VAL("&H"+A$):A$="":IF W<0 OR W>255 THEN
BEEP:GOTO 50
100 POKE AD,W:AD=AD+1:GOTO 50
```

Introduzca los códigos hexadecimales, y el programa resolverá el >POKE<. No será necesario entrar el signo hexadecimal.(&H).

Una vez que ya hemos aprendido a entrar programas en lenguaje máquina, veamos los comandos del Z80.

Observación: En las aclaraciones de comandos trabajaremos a menudo con analogías de los comandos en BASIC. Para ello, nos imaginaremos un registro en BASIC como una Variable con el mismo nombre (Registro HL en lenguaje máquina corresponde a la Variable HL en BASIC).

Los comandos del Z80 se subdividen en cinco grupos:

1. Transferencia de datos
2. Tratamiento de datos y comparaciones
3. Saltos
4. Comandos de control
5. Entrada y Salida

3.2 TRANSFERENCIA DE DATOS

Estos comandos sirven para transferir datos.
Los datos pueden ser transferidos de:

a) Registro a registro
   Ello supone una designación en BASIC; como p.ej.,A=B, o SP=HL. El comando en l. máquina presenta el siguiente formato: LD A,B (LD-ingl.load: carga)

b) Registro a posición de memoria
   En el traspaso de datos del registro a la posición de memoria, el comando en BASIC >POKE Dirección de memoria, Variable<, p.ej, >POKE &HF000,HL< corresponde al comando máquina LD (&HF000),HL.

c) Posición de memoria a registro
   El traspaso de datos de la memoria a un registro, p.ej, LD H,(&HF005), corresponde al comando en BASIC: >H=PEEK (&HF005)<.

3.3 TRATAMIENTO DE DATOS Y COMPARACIONES

Los comandos para el tratamiento de datos pueden volver a dividirse en cinco grupos:

- Operaciones aritméticas (p.ej. ADición, SUBstracción)
- Operaciones lógicas (p.ej. AND OR)

- Comandos contadores (INCrease=aumentar, DECrease= disminuir)
- Manipulación de Bits (SET, RESet)
- Intercambios y desplazamientos de Bits (Rotate = rotar, Shift = desplazar)

Durante la ejecución de estos comandos se modifican los contenidos de los registros o de la memoria (en RAM). Muchos de estos comandos son similares a los del BASIC:

Ensamblador	BASIC
SUB A,B (SUBstracción)	A=A-B
ADD HL,BC (ADDición)	HL=HL+BC
AND C	A=A AND C
OR HL	A=A OR PEEK(HL)

Las comparaciones se realizan, o bien de Bits sueltos en el registro, o en las posiciones de memoria (comando BIT), o bien, se comparan los contenidos de los registros o de la memoria con el Acu (comando CP=compare). Según los resultados de estas comparaciones, la ALU activará o dasactivará cada uno de los Flags en el registro F.

3.4 SALTOS

Con ayuda de estos comandos es posible incluir bifurcaciones en programas máquina.
Se distingue entre tres tipos de saltos:

- salto directo a una dirección de 16-Bits (JP=Jump)
- salto relativo a la dirección actual (JR=Jump relativ)
- saltos a subprograma (CALL y saltos de retorno RET)

Un salto es condicional cuando la decisión de si ha de efectuarse el salto o no, depende del estado de un Flag.

Un salto condicional, es decir, aquel que depende del estado de un Flag, sería p.ej. JR NZ,&HF003.

Analogías:

Ensamblador	BASIC
JP	GOTO
CALL	GOSUB
RET	RETURN
JR	--- (parecido al bucle FOR-NEXT)

3.5 COMANDOS DE CONTROL

Con estos comandos puede ser interrumpido un programa. El control con interrupciones también es posible con estos comandos.

3.6 COMANDOS DE ENTRADA / SALIDA (Input/Output)

Los comandos I/O sirven para la comunicación con los periféricos de entrada y salida. Las más diversas actividades se llevan a cabo con estos comandos, dependiendo de cada una de las direcciones del port de I/O activadas. Los ICs frecuentemente activados por comandos de I/O son:

PPI (Programmable Peripheral Interface),
PSG (Programmable Sound Generator)
VDP (Video Display Controller), y con ello, los periféricos: teclado, altavoz, monitor, impresora y grabadora.

CAPITULO IV : L O S C O M A N D O S

4.1 COMANDOS DE TRANSFERENCIA DE 8-BITs

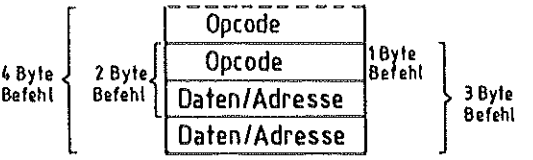
Todos los comandos de transferencia de este tipo se representan por el comando de carga LD (load).
Un comando de carga presenta el siguiente formato:

LD Destino, Origen

En los comandos de transferencia de 8-Bits, se cargan cada 8-Bits del origen al destino. Como ejemplo de estos comandos le presentaremos los tipos de direccionamiento del Z80.

Cada comando máquina se compone fundamentalmente de un código de operación (Opcode), que puede ser seguido de un campo de Operando o de Dirección. El Opcode determina que operación debe ser efectuada. A veces, un Opcode contiene Bits que son utilizados como indicadores de un registro. En realidad, estos Bits no pertenecen al Opcode. Para simplificar el ejemplo, contaremos estos indicadores eventuales como pertenecientes al Opcode. En algunos comandos, el Opcode viene seguido de Bytes de datos y de dirección. Además, también existen comandos cuyo Opcode tiene dos Bytes. De este modo, un comando puede llegar a tener una longitud de 1 a 4 Bytes.

Esq. 2 · 4.1



4 Byte befehl=comando de 4 Bytes
etc.
Daten/Adresse=Datos/dirección

Para interpretar los datos o direcciones que siguen a un comando, es necesario conocer los distintos tipos de direccionamiento.

Direccionamiento Inmediato (Immediately Adressing)

Este es el tipo más sencillo de direccionamiento.

Formato:

LD r,n

En este comando, "r" representa un registro (A,B,C,D,E,H o L) y "n", un número de 8-bits; es decir, el registro r dado se carga con la constante "inmediata" que le sigue. Una constante de este tipo también se la califica de Literal. El direccionamiento inmediato viene representado en el esquema 3. Al Opcode de 8-Bits le sigue una Literal (constante) de 8- ó 16-Bits.

Ejemplo:

LD C,&H7F BASIC: C=&H7F
(significa: cargar registro C con &H7F)

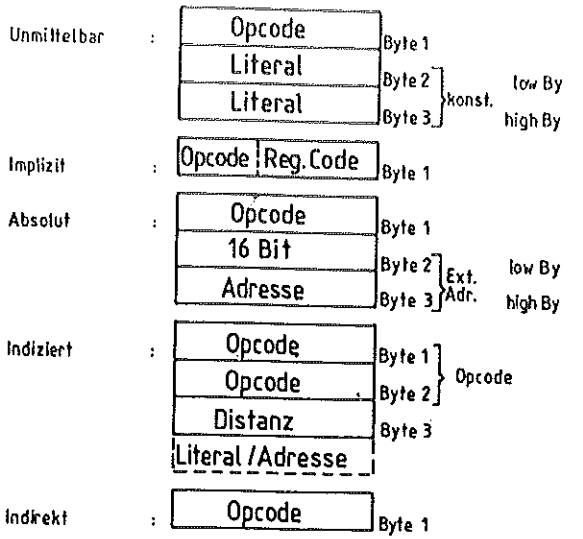


Abb. 3 4.1

Unmittelbar=Inmediato
Implizit=Implícito
absolut=absoluto
Indiziert=Indexado
Indirekt=Indirecto
Distanz=distancia

Direccionamiento implícito y de registro (ingl.: Implied Register Adressing)

Los comandos que trabajan exclusivamente con registros, utilizan el direccionamiento implícito (ingl.: Implied: implícito)

Formato:

LD r,r'

Traslada el contenido del registro origen r' a r, o carga r de r'. Los registros pueden ser A, B, C, D, E, H o L.
El nombre de este tipo de direccionamiento resulta del hecho de que el Operando (es decir, los dos registros afectados) no se indican expresamente. En realidad, el Opcode de la orden contiene los registros afectados, (los implica).
En forma binaria, el código de operación de esta orden es:

01DDD000

Cada una de las letras D o O corresponden aqu a un Bit. además, las tres D representan el registro destino r, y las tres O, el registro origen r'. El código para los registros es:

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

Ejemplo: LD B,C = 01 000 001 = &H41
LD B C

Con ello, es posible representar los comandos direccionados implicitamente como un código de operación de 1-Byte. Por este motivo, es muy reducida la duración de su ejecución.

Ejemplo:

LD A,B BASIC: A=B

Significa: transfiere el contenido de B a A, o carga A de B.

Zilog Inc. (El "Inventor" del Z80) califica el anterior tipo de direccionamiento como direccionamiento de registro,

y discrepa en la definición del direccionamiento implícito. Según ello, solamente serían direccionados implícitamente los comandos LD I,A ; LD R,A ; LD A,R y LD A,I. Nosotros, sin embargo, no haremos tal diferencia, y utilizaremos como sinónimos ambos conceptos: direccionamiento implícito y direccionamiento de registro.

Direccionamiento absoluto o "externo" (External Adressing)

(Ingl.:external=externo)
Como direccionamiento absoluto se describe al proceso de sacar datos de la memoria, o de almacenarlos en ella. Mediante este procedimiento, la dirección de 16-bits de la posición de memoria se indica por completo (la dirección "absoluta").

Formato:

LD (nn),r o LD r,(nn)
(nn: es la dirección de la posición de memoria).

El registro r dado es cargado con el contenido de la posición de memoria nn, y viceversa. Del esquema 3, puede usted deducir que la dirección sigue al código de operaciones. El direccionamiento absoluto requiere tres Bytes, por lo que la ejecución de los comandos de esta clase resulta ser relativamente larga.

Ejemplo:

```
LD A, (&HBF93)        BASIC:A=PEEK(&HBF93)
LD (&HF001), A        BASIC:POKE &HF001,A
```

Direccionamiento indexado (Indexed Adressing)

(Ingl. Index: índice)

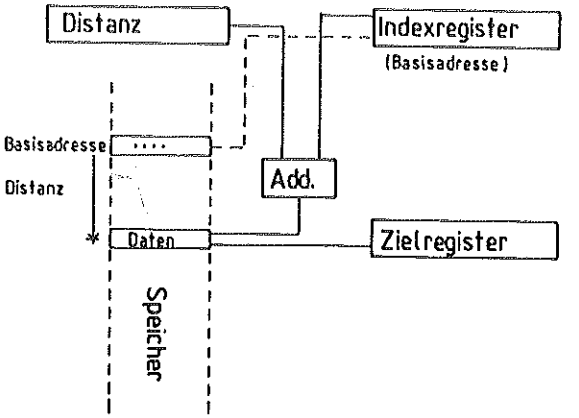
En el direccionamiento indexado no queda indicada absolutamente la dirección de la posición de memoria, sinó que ésta se calcula a partir del contenido de un registro índice y de un desplazamiento indicado.

Formato:

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

(d=distancia)(IX o IY son registros)

Carga del registro r con la posición de memoria que posee la siguiente dirección (y viceversa): La dirección resulta del contenido del registro índice y del desplazamiento indicado.



Indexregister=registro indice
basisadresse=dirección de base
Speicher= memoria
Zielregister=registro destino
Daten=datos
Distanz=desplazamiento

Esq.4 Direccionamiento indexado/ reg.(XY+dis),LD

Los comandos indexados poseen un código de operación de 2-Bytes, tras el cual, viene indicado el desplazamiento. El primer Byte del código de operación es:

&HDD - cuando se refiere al registro IX
&HFD - cuando se refiere al registro IY

El resto de los Bytes del código son idénticos, independientemente de si se alude al registro IX o al IY. La técnica del direccionamiento indexado se utiliza para acceder, uno a continuación del otro, a los elementos de un bloque de datos. El desplazamiento puede ser positivo o negativo; es decir, el Byte del desplazamiento se indica como complemento a dos.

Ejemplo:

```
LD E, (IX+&H32)       BASIC: E=PEEK (IX+&H32)
LD (IY+&H12), A       BASIC: POKE IY+&H12,A
```

Direccionamiento Indirecto (Registro Indirecto)

Este tipo de direccionamiento se asemeja al direccionamiento indexado y al absoluto, sólo que aquí, la posición de memoria se direcciona mediante el contenido de uno de los registros par HL, BC o DE.

Formato:

```
LD r,(HL)     o bien,     LD (HL),r
LD A,(BC)     o bien,     LD (BC),A
LD A,(CD)     o bien,     LD (DE),A
```

Carga el registro r, o bien, A con el contenido de la posición de memoria, la cual está direccionada mediante el contenido del registro par HL, BC o DE. Esta técnica de direccionamiento presenta, frente al direccionamiento indexado y al absoluto, la ventaja de que sólo necesita comandos de 1-Byte de longitud; es decir, que el registro r y el registro par HL, o el BC,DE, están contenidos en el código de operaciones no necesitan ser indiciados expresamente. De este modo, este comando es más rápido, y sin embargo, también ofrece la posibilidad de acceder a todas las 64K.

Ejemplo:

```
LD B, (HL)        BASIC: B=PEEK (HL)
LD (BC), A        BASIC: POKE BC,A
```

Con ésto, ya hemos estudiado todos los tipos de direccionamiento que aparecen en los comandos de transferencia de 8-Bits. En el curso de este capítulo, aún daremos a conocer algunos otros tipos de direccionamiento, así como también aplicaremos los ya conocidos a otros comandos.

En el apéndice, usted hallará unas tablas, donde podrá encontrar todos los comandos, clasificados por funciones (transferencia, saltos, etc), y por tipos de direccionamiento. En estas tablas podrá consultar los códigos de operación de todos los comandos. A continuación le presentamos una tabla que reune todos los comandos de carga de 8-Bits. En el apéndice encontrará una tabla para los símbolos de la influencia de los Flags. Las dos últimas columnas: NO. of M Cycles, y NO. of T Cycles las explicaremos más adelante.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	OP-Code 76	543	210	No. of Bytes	No. of M Cycles	No. of T Cycles	Comments
LD r, r'	r ← r'	•	•	•	•	•	•	01	r	r'	1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	•	•	•	•	00	r	110	2	2	7	000 B
								←	n	→				001 C
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	01	r	110	1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	•	•	•	•	11	011	101	3	5	19	011 E
								01	r	110				100 H
								←	d	→				101 L
LD r, (IY+d)	r ← (IY+d)	•	•	•	•	•	•	11	111	101	3	5	19	111 A
								01	r	110				
								←	d	→				
LD (HL), r	(HL) ← r	•	•	•	•	•	•	01	110	r	1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	•	•	•	•	11	011	101	3	5	19	
								01	110	r				
								←	d	→				
LD (IY+d), r	(IY+d) ← r	•	•	•	•	•	•	11	111	101	3	5	19	
								01	110	r				
								←	d	→				
LD (HL), n	(HL) ← n	•	•	•	•	•	•	00	110	110	2	3	10	
								←	n	→				
LD (IX+d), n	(IX+d) ← n	•	•	•	•	•	•	11	011	101	4	5	19	
								00	110	110				
								←	d	→				
								←	n	→				
LD (IY+d), n	(IY+d) ← n	•	•	•	•	•	•	11	111	101	4	5	19	
								00	110	110				
								←	d	→				
								←	n	→				
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	00	001	010	1	2	7	
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	00	011	010	1	2	7	
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	00	111	010	3	4	13	
								←	n	→				
								←	n	→				
LD (BC), A	(BC) ← A	•	•	•	•	•	•	00	000	010	1	2	7	
LD (DE), A	(DE) ← A	•	•	•	•	•	•	00	010	010	1	2	7	
LD (nn), A	(nn) ← A	•	•	•	•	•	•	00	110	010	3	4	13	
								←	n	→				
								←	n	→				
LD A, I	A ← I	•	‡	IFF	‡	0	0	11	101	101	2	2	9	
								01	010	111				
LD A, R	A ← R	•	‡	IFF	‡	0	0	11	101	101	2	2	9	
								01	011	111				
LD I, A	I ← A	•	•	•	•	•	•	11	101	101	2	2	9	
								01	000	111				
LD R, A	R ← A	•	•	•	•	•	•	11	101	101	2	2	9	
								01	001	111				

Notes: r, r' means any of the registers A, B, C, D, E, H, L

IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,

‡ = flag is affected according to the result of the operation.

Ejercicio:

Ahora, usted ya conoce los comandos de carga de 8-Bits. Como aplicación, cambiaremos, con ayuda de estos comandos, algunas posiciones de memoria del sistema operativo a partir del lenguaje máquina. Observemos primero la posición de memoria &HF3B0. En esta dirección está memorizado el número actual de columnas en pantalla, que puede darse por el comando en BASIC >WIDTH<. Queremos fijar el número de columnas en 20. A partir del BASIC, ésto se consigue con el comando >WIDTH 20<. Pero también es posible escribir directamente en la posición de memoria &HF3B0 el valor deseado utilizando un comando >POKE<.
Así pues, pruebe >POKE &HF3B0,20<, y el ancho de la pantalla quedará limitado a 20 columnas. La diferencia con el comando >WIDTH< consiste en que, al modificar el valor, no se reliza automáticamente un >CLS<. ¡Ello puede ser muy útil en según que casos !
Pruebe a escribir el comando BASIC
>POKE &HF3B0,20<
en lenguaje máquina. Finalice su programa con el comando RET (code &HC9).

Discusión de la solución

Analicemos primero el ejercicio propuesto:
Una posición de memoria (Dirección &HF3B0) ha de ser cargada con un valor de 1-Byte, o sea, con un número entre 0 y 255. Puesto que el valor es un número de 1-Byte, utilizaremos un comando de carga de 8-Bits.
A partir de aquí, existen varias posibilidades de solución que dependerán del tipo de direccionamiento elejido. Para cargar una posición de memoria con un Byte, deberá fijarse de antemano la dirección de la posición de memoria. La posibilidad más próxima es el direccionamiento absoluto, por el cual se obtiene la dirección completa:

```
LD (&HF3B0),A  (direccionamiento absoluto)
```

es decir: carga la posición de memoria &HF3B0 con el valor contenido en el Acu. Ello significa que el Acu deberá haber sido cargado anteriormente con el valor deseado:

```
LD A,20        (direccionamiento inmediato)
```

Ambos comandos, ejecutados uno a continuación del otro, llevan al resultado deseado.

```
LD A,20
LD (&HF3B0),A
RET
```

El comando RET ha de estar al final de cada programa en lenguaje máquina. Provoca que la ejecución del programa continúe en BASIC.
Hasta ahora hemos presentado el programa máquina sólo en lenguaje Ensamblador; es decir, en un modo de escritura simbólico. Para que el procesador pueda ejecutar nuestros comandos, éstos, deberán ser antes traducidos en cifras. Las listas de comandos o las tablas contenidas al final del libro, le darán los códigos &H3E,20 para el comando de carga de 8-Bits LD A,20 direccionamiento inmediatamente, donde el segundo código es el valor que nosotros deseamos. Para el

comando LD (&HF3B0),A, de direccionamiento absoluto, obtenemos &H32, &HB0, &HF3. También aquí, los dos últimos códigos representan la dirección dada por nosotros. Observe que primero se indica el Byte bajo (&HB0) y luego, el Byte alto.
Como ya hemos indicado antes, el código para RET es &HC9.

A partir de estos códigos, resulta la línea DATA para el cargador en BASIC de la página 42 :

DATA &H3E,20,&H32,&HB0,&HF3,&HC9

Nuestro programa tiene una longitud de 6 Bytes; es decir, el bucle FOR-NEXT deberá funcionar desde &HF000 hasta &HF005. Una vez activado el cargador BASIC con >RUN<, tenemos en la memoria nuestro primer programa máquina. Mediante el comando DEFUSR1=&HF000, la dirección de partida del comando >USR1< estará colocada al inicio de nuestro programa. Ahora, usted ya puede iniciar su primer programa, dando >X=USR1(1)<. Si ha seguido correctamente todos los pasos, su pantalla debería quedar reducida a 20 columnas. Si desea cambiar de nuevo el valor de la amplitud de pantalla, simplemente deberá cambiar el segundo número de la línea DATA (el "20"), y reescribir de nuevo en la memoria el programa mediante >RUN<.

4.2 COMANDOS DE TRANSFERENCIA DE 16-BITs

Los comandos de carga de 16-Bits también presentan el formato general:

LD Destino,Origen

Sin embargo, aquí se transfieren 16-Bit. Mediante estos comandos, se activan los registros pares BC,DE,HL,SP,IX y IY.

Direccionamiento Inmediato

Como aquí solamente se cargan registros de 16-Bits, la constante que sigue al código de operaciones deberá tener una longitud de 16-Bits. De este modo, los dos Bytes siguientes al código de operaciones, contienen el Byte bajo y el Byte alto de la constante (¡en este orden!). En contraposición al direccionamiento inmediato con constantes de 1-BYTE, esta técnica recibe el nombre de direccionamiento de extensión inmediata (Ingl. Inmediately extended).

Formato:

LD x,nn
(x: Uno de los registros de 16-Bits SP,BC,DE,HL,IX,IY)
(nn: consonante de 16-Bits)

Mediante este comando, el registro x se carga con la constante nn.

Ejemplo:

```
LD HL,&HF000                    BASIC: HL=&HF000
```

Direccionamiento Implícito

Para los comandos de carga de 16-Bits sólo existen tres comandos de este tipo, y todos ellos afectan al registro SP:

LD SP,HL LD SP,IX LD SP,IY

Estos comandos significan:
Carga del apuntador de la pila con el contenido del registro HL, IX o IY.

Analogía en BASIC:
SP=HL SP=IX SP=IY

Direccionamiento Absoluto

El direccionamiento absoluto en los comandos de 16-Bits deberemos explicarlo de nuevo con más detalle:

Formato:

LD x,(nn) o LD (nn),x

(X: BL, DE, HL, SP, IX o IY)
Puesto que nn indica a una dirección, o sea, que solamente direcciona un Byte, y puesto que sin embargo, x es un registro de 16-Bits, se ha tomado la siguiente convención:
Primero se carga en el registro el Byte bajo de la dirección adr, luego, el Byte alto de la dirección nn+1.

p.ej: LD HL,(&HF280) significa:
Registro L = Low-Byte de la dirección &HF280
Registro H = High-Byte de la dirección &HF281

En el comando inverso de la forma LD (nn),x, el Byte bajo se memoriza en la correspondiente dirección nn, y el Byte alto en la dirección nn+1.

p.ej: LD (&HF000),IX

Dirección &HF000 = Low Byte de IX
Dirección &HF001 = High Byte de IX

Un comando de este tipo corresponde, por lo tanto, a dos comandos de carga de 8-Bits.

Comando de 16-Bit:		Comandos de 8-Bit:
LD BC,(&HF005)	corresponde a	LD C,(&HF005) (Low Byte)
		LD B,(&HF006) (High Byte)

Como ya sabe, se puede representar un número de 16-Bit a partir del Byte alto y del Byte bajo, de la siguiente manera:

Número=256*(High Byte)+(Low Byte)

Con ello, el equivalente BASIC para:

Lenguaje Máquina:	BASIC:
LD DE,(&HF000)	DE=256*PEEK(&HF001)+PEEK(&HF000)

Tenga en cuenta, que utilizando el sistema hexadecimal, también se puede escribir lo siguiente:

```
DE=VAL("&H"+HEX$(PEEK(&HF001))+HEX$(PEEK(&HF000)))
```

Para escribir en BASIC el comando inverso; es decir, p.e. LD (&HF100),IY, son necesarios dos comandos:

```
POKE &HF100,  IY-INT (IY/256)*256     (Low Byte)
POKE &HF101,     INT (IY/256)         (High Byte)
```

Si no le resultan claras estas analogías, vuelva a repasar el capítulo relativo a la representación de números. Coloque entonces números para cada DE e IY, y luego, ¡realice el cálculo real!
Todas las listas de comandos de este libro han sido cedidas por la empresa Zilog.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD *dd*, *nn*	*dd* ← *nn*	•	•	•	•	•	•	00 *dd*0 001 ← *n* → ↔ *n* →	3	3	10	*dd* Pair 00 BC 01 DE 10 HL 11 SP
LD IX, *nn*	IX ← *nn*	•	•	•	•	•	•	11 011 101 00 100 001 ← *n* → ← *n* →	4	4	14	
LD IY, *nn*	IY ← *nn*	•	•	•	•	•	•	11 111 101 00 100 001 ← *n* → ← *n* →	4	4	14	
LD HL, (*nn*)	H ← (*nn*+1) L ← (*nn*)	•	•	•	•	•	•	00 101 010 ← *n* → ← *n* →	3	5	16	
LD *dd*, (*nn*)	dd_H ← (*nn*+1) dd_L ← (*nn*)	•	•	•	•	•	•	11 101 101 01 *dd*1 011 ← *n* → ← *n* →	4	6	20	
LD IX, (*nn*)	IX_H ← (*nn*+1) IX_L ← (*nn*)	•	•	•	•	•	•	11 011 101 00 101 010 ← *n* → ← *n* →	4	6	20	
LD IY, (*nn*)	IY_H ← (*nn*+1) IY_L ← (*nn*)	•	•	•	•	•	•	11 111 101 00 101 010 ← *n* → ← *n* →	4	6	20	
LD (*nn*), HL	(*nn*+1) ← H (*nn*) ← L	•	•	•	•	•	•	00 100 010 ← *n* → ← *n* →	3	5	16	
LD (*nn*), *dd*	(*nn*+1) ← dd_H (*nn*) ← dd_L	•	•	•	•	•	•	11 101 101 01 *dd*0 011 ← *n* → ← *n* →	4	6	20	
LD (*nn*), IX	(*nn*+1) ← IX_H (*nn*) ← IX_L	•	•	•	•	•	•	11 011 101 00 100 010 ← *n* → ← *n* →	4	6	20	
LD (*nn*), IY	(*nn*+1) ← IY_H (*nn*) ← IY_L	•	•	•	•	•	•	11 111 101 00 100 010 ← *n* → ← *n* →	4	6	20	
LD SP, HL	SP ← HL	•	•	•	•	•	•	11 111 001	1	1	6	
LD SP, IX	SP ← IX	•	•	•	•	•	•	11 011 101 11 111 001	2	2	10	
LD SP, IY	SP ← IY	•	•	•	•	•	•	11 111 101 11 111 001	2	2	10	
PUSH *qq*	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	•	•	•	•	11 *qq*0 101	1	3	11	*qq* Pair 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	•	•	•	•	11 011 101 11 100 101	2	4	15	
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	•	•	•	•	11 111 101 11 100 101	2	4	15	
POP *qq*	qq_H ← (SP+1) qq_L ← (SP)	•	•	•	•	•	•	11 *qq*0 001	1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	•	•	•	•	11 011 101 11 100 001	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	•	•	•	•	11 111 101 11 100 001	2	4	14	

Notes: *dd* is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
E.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ flag is affected according to the result of the operation.

Ejercicio:

Con ayuda de los comandos de carga de 16-Bits, usted ya puede reconstruir el programa del capítulo anterior. Utilice para ello un comando de carga de 16-Bits del direccionamiento indirecto.

Solución:

```
LD HL,&HF3B0
LD (HL),20
RET
```

Primero, se carga en el registro HL la dirección con un comando de carga de 16-Bits del direccionamiento de extensión inmediata. A continuación, se escribe el valor en la posición de memoria, ahora direccionada indirectamente. La línea DATA del programa traducido es la siguiente:

```
DATA &H21,&HB0,&HF3,&H36,20,&HC9
```

En principio, también es posible aplicar el direccionamiento indexado. Sin embargo, este procedimiento tiene poco sentido en esta aplicación, ya que sólo prolongaría innecesariamente el programa.
Pruebe también el mismo programa con la dirección &HF3DB. Esta dirección es la responsable del ruído de activación del teclado. Si tiene el valor 0, no se produce ningún ruido. Todos los demás valores provocan que, al ser activado el teclado, se produzca un crujido.

Observemos otro ejemplo para los comandos de carga de 16-Bits de direccionamiento absoluto.
Estos comandos siempre afectan a 2 Bytes. Con ayuda de estos comandos, vamos a cambiar la posición del cursor. La posición queda perfectamente descrita por 2 Bytes: la coordenada-X y la coordenada-Y. Estos dos valores están en la dirección &HF3DC (Y) y &HF3DD (X). Escriba un programa que coloque la posición-X en 5, y la posición-Y, en 10.

Solución:

Como siempre, para este problema también se dan varias soluciones. Primero ha de ser cargado el registro HL con los valores correspondientes. La coordenada-Y ha de ser colocada en el registro L, y la coordenada-X, en el registro H, y ello, debido a que Y debe ser colocada en la posición más baja de la memoria (&HF3DC). Ahora, o bien podemos cargar cada registro por separado

```
LD L, 10
LD H,5
```

o bien, podemos cargar el registro doble HL de una sola vez. Para ello, antes deberemos unir en L (=Y) y H (=X) a un número de 16-Bits:

```
L=10= &H0A
H= 5= &H05
```

HL=256*5+10= &H050A

HL se cargará con este número:

```
LD HL,&H050A
```

Como esta es la solución más corta, la escogeremos. El código para el comando LD HL,nn es &H21.

Los dos códigos siguientes expondrán en el orden Byte bajo-Byte alto, el número &H050A. El comando completo en Código Máquina es el siguiente:

```
&H21,&H0A,&H05
```

Ahora, este valor debe ser cargado en la dirección &HF3DC/D:

```
LD (&HF3DC),HL
```

Este comando, al igual que arriba, se traduce en:

```
&H22,&HDC,&HF3
```

La línea DATA al finalizar el comando RET, sería la siguiente:

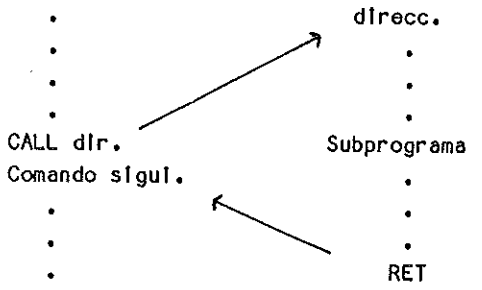
```
DATA &H21,&H0A,&H05,&H22,&HDC,&HF3,&HC9
```

Para poder reconocer en pantalla la posición actual del cursor, tiene sentido añdir lo siguiente al programa:

```
11 X=USR1(1):PRINT"*"
```

4.3 COMANDOS DE LA PILA

Para comprender mejor el modo de funcionamiento de los apuntadores, es necesario saber lo que ocurre en el interior del Z80 cuando se salta a un subprograma. El comando Ensamblador necesario para ello, es: CALL dirección. El problema básico consiste en que la CPU ha de "anotarse" la dirección de los comandos siguientes a ésta, ya que cuando se retorne al programa principal (RET), es allí donde continuará la ejecución del programa.



Esq. 5 Llamada al Subprograma

Puesto que los registros son necesarios para otras operaciones importantes, las direcciones de retorno han de ser memorizadas fuera de la CPU, o sea, en la RAM. Sin embargo, siguiendo este procedimiento, sólo se podría memorizar una dirección de retorno. Ello significa que no sería posible un anidamiento de los subprogramas. Este es el motivo por el cual se ha reservado un espacio de la RAM para esta operación. A este espacio se le denomina stack o pila. Imaginémonos a esta pila como un apilamiento de platos:

Una dirección de retorno se registrará siendo anotada en un plato. El plato así "direccionado", se coloca encima de la pila de platos. De este modo pueden tener lugar muchas llamadas a subprogramas: así va aumentando el apilamiento. Cuando se realiza un retorno, se quita el plato de encima y se bifurca en la dirección que lleva escrita. De este modo se suceden, uno a continuación del otro, los saltos hacia atrás, hasta que le pila de platos queda deshecha; es decir, hasta que se llega de nuevo al programa principal. Es importante que el último plato colocado en la pila, sea siempre el primero que se quite (de lo contrario, toda la pila se vendría abajo).
Puesto que en el ordenador no son platos los que se apilan, deberá utilizarse un registro del Z80 como "medidor de la cima" de la pila. Este registro guarda siempre en memoria la dirección de la cima de la pila. Se le denomina Stack Pointer (SP). De todos modos, nuestra pila en el ordenador "cuelga" del techo, es decir, el primer plato es colocado en la dirección más alta en la pila, y el último, en la más baja. Este espacio (de la pila) está situado en el espacio superior de la RAM en el ordenador MSX. Su situación exacta depende de cada una de las divisiones fijadas por los comandos >CLEAR< y >MAXFILES<. Con todo ésto, la ejecución del comando CALL sería la siguiente:
Sección de la pila:

Posición de partida:

```
                .          .
                .          .
                .          .
          Pila &HEBF4 : (entrada anterior)
               &HEBF3 : (entrada anterior)
               &HEBF2 : (entrada anterior)
               &HEBF1 : (última entrada)
               &HEBF0 : (lugar p. nueva entada)
               &HEBEF : (lugar p. nueva entrada)
                   .          .
                   .          .
```

Apuntador de la pila (SP):
    &HEBF1

El registro SP indica la última entrada en la pila. Supongamos: en la ejecución del programa, el Procesador topa con la orden CALL &H077E en la dirección &H052D.

&H052D CALL &H077E
&H0530 comando siguiente

Después de leer el comando, el PC está en &H530. Esta es la dirección de retorno que ha de ser registrada. La dirección es colocada en la pila como Byte bajo y Byte alto. O sea, el SP disminuye, el Byte alto se registra en la dirección SP, el SP disminuye de nuevo, y el Byte bajo es registrado en la nueva dirección. A continuación, se carga el PC con la dirección de' inicio del subprograma antes dada (&H077E); es decir, ahí es donde se continúa la ejecución del programa. Resulta de ello la siguiente constelación:

```
        .       .
        .       .
        .       .
Pila: &HEBF0    &H05
      &HEBEF    &H30    : (Última entrada)
        .       .
        .       .
SP:&HEBEF
```

Como usted ve, el SP indica de nuevo la última entrada. Con el comando RET, todo el proceso se realiza a la inversa:

El Byte de la posición de memoria, indicada por el SP, es cargado como Byte bajo en el PC. Se eleva en uno el SP y se carga en el PC el Byte alto de la dirección de retorno. Luego, se eleva otra vez en uno el SP, es decir, indica otra vez la dirección de retorno actual en la pila. Ahora, se continúa con le ejecución del programa en la posición PC, o sea, en la dirección de retorno correcta.

```
        .           .
        .           .
Pila: &HEBF1        ...         SP: &HEBF1
      &HEBF0        &H07
      &HEBEF        &H83
        .           .
```

Los procesos descritos tienen lugar de forma automática en el Z80, siempre que se utiliza un comando CALL o RET. Esto determina que la sucesión en la pila sea siempre correcta y que el SP indique siempre la posición correcta. Si usted modifiqua el SP directamente a partir del programa, puede producirse facilmente un desorden en la sucesión y el consiguiente bloqueo del ordenador. Así pues, utilice con cuidado los comandos LD SP,x.
Por otro lado, también pueden ser colocados datos en la pila y ser llamados desde ahí. Para ello sirven los comandos:

```
PUSH (colocar en la pila)
 y
POP  (coger de la pila)
```

El comando PUSH tiene un funcionamiento análogo al del comando CALL. Los datos que han de ser memorizados son escritos en la pila tras la disminución del SP. Con POP se procede a la lectura de los datos y el SP se eleva automaticamente. La CPU también se hace cargo aquí de varias operaciones. Con PUSH y POP pueden "apilarse" varios registros (pares) de 16-Bits, a excepción del propio SP.

Formato:

PUSH qq	POP qq
PUSH IX	POP IX
PUSH IY	POP IY

(qq: BC, DE, HL,AF)

Puesto que el Acumulador es siempre un registro de 8-Bits, y como tiene sentido salvar el registro Flag (F) en la pila, A y F son tratados conjuntamente.
La técnica del almacenamiento intermedio en la pila tendrá sentido cuando los registros para el almacenamiento ya no sean suficientes.

Ejemplo:

HL contiene un sumando
BC contiene otro sumando

Ahora, se llama a un Subprograma, que sumará HL y BC. El resultado de la suma se registrará en HL. Si aún se necesitase el primer sumando, éste debería ser colocado a tiempo en la pila.

```
LD HL,sumando-uno
LD BC,sumando-dos
PUSH HL
CALL Dirección-suma
  •
  •
  •
POP HL
  •
  •
```

Si se necesita este sumando, puede ser recogido de la pila con POP HL.
Hay que tener en cuenta que el comando POP perteneciente a un comando PUSH ha de estar siempre en el mismo subprograma. De lo contrario, los datos almacenados mediante PUSH, se

interpretan como la dirección de retorno para el comando RET, lo que, probablemente, ocasionaría el bloqueo del ordenador. Los comandos PUSH o POP no poseen ningún comando similar directo en el MSX BASIC. Estos comandos pueden escribirse en BASIC de la siguiente forma:

Ejemplo BASIC:

```
PUSH AF         BASIC: POKE SP-1,A:(High Byte)
                       POKE SP-2,F
                       SP=SP-2

POP BC          BASIC: BC=PEEK(SP)+256*PEEK(SP+1)
                       SP=SP+2
```

Puesto que PUSH y POP utilizan el registro SP como indicador de dirección, ambos se cuentan como direccionamiento indirecto.

Ejemplo:

```
PUSH HL                SP=&HEB05
                       HL=&H1234
```

Después de la ejecución: posición de memoria &HEB04:&H12
posición de memoria &HEB03:&H34

```
SP = &HEB03
HL = &H1234
```

Ejemplo:

```
POP HL                 SP=&HEB03
                       HL=&HFFFF
```

Después de la ejecución:

SP = &HEB05

HL = &H1234

La lista de comandos para los comandos de pila se halla al final del capítulo 4.2: Comandos de carga de 16-Bits.

4.4 COMANDOS DE INTERCAMBIO

Además de los comandos sencillos de transferencia de datos (LD), en el Z80 también se da otro comando que intercambia el contenido de dos áreas. Estos comandos vienen representados por EX (ingl.exchange: intercambiar).
Un comando de este tipo, EX DE,HL intercambia, por ejemplo, el contenido del registro DE con el del registro HL. El comando EX con direccionamiento indirecto, intercambia el contenido de los registros HL, IX o IY con el elemento superior de la pila (sin que varíe el SP).

Formato:

EX (SP),HL
EX (SP),IX
EX (SP),IY

Además, también existen otros comandos de intercambio que intercambian entre sí los contenidos de las series de registros pares. Como ya hemos mencionado, cada registro A, BC, DE, HL, F tiene su correspondiente registro A', BC', DE', y F'. Sin embargo, sólo se trabaja a la vez con un par de registros pares. Pero si es necesario, se puede intercambiar entre sí el contenido de ambos pares.
El comando EX AF,AF' intercambia el contenido del Acumulador y el del registro Flag con los registros correspondientes A' y F'. El comando EXX intercambia los otros registros pares BC, DE y HL con BC', DE' y HL'.
Estos comandos están direccionados implícitamente.

Ejemplo:

```
EX DE,HL        BASIC:SWAP DE,HL

EX (SP),HL      BASIC:ZWI=HL:HL=256*PEEK(SP+1)+PEEK(SP):
                      POKE SP+1,INT(ZWI/256):POKE SP,
                      ZWI-INT(ZWI/256)*256
```

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	$\begin{pmatrix}BC\\DE\\HL\end{pmatrix} \leftrightarrow \begin{pmatrix}BC'\\DE'\\HL'\end{pmatrix}$	•	•	•	•	•	•	11 011 001	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP+1) L ↔ (SP)	•	•	•	•	•	•	11 100 011	1	5	19	
EX (SP), IX	$IX_H \leftrightarrow (SP+1)$ $IX_L \leftrightarrow (SP)$	•	•	•	•	•	•	11 011 101 11 100 011	2	6	23	
EX (SP), IY	$IY_H \leftrightarrow (SP+1)$ $IY_L \leftrightarrow (SP)$	•	•	•	•	•	•	11 111 101 11 100 011	2	6	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	① 1	•	0	0	11 101 101 10 100 000	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	① 0	•	0	0	11 101 101 10 110 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	① 1	•	0	0	11 101 101 10 101 000	2	4	16	
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	① 0	•	0	0	11 101 101 10 111 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A - (HL) HL ← HL+1 BC ← BC - 1	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 100 001	2	4	16	
CPIR	A - (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 110 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A - (HL) HL ← HL-1 BC ← BC-1	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 101 001	2	4	16	
CPDR	A - (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 111 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1
② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

4.5 COMANDOS DE TRANSFERENCIA DE BLOQUES Y DE BUSQUEDA

A diferencia del comando LD, que sólo transfiere uno o dos Bytes, los comandos de transferencia de bloques transfieren todo un bloque de datos. Presentan una particularidad del Z80. Por lo general, no se puede disponer de estos comandos en los microprocesadores, pues son muy laboriosas de realizar para los constructores de ordenadores. Sin embargo, estos comandos sí son de gran utilidad para el programador, ya que aumentan la capacidad de rendimiento de un programa. Un bloque de datos se caracteriza por las siguientes indicaciones:

- La dirección de inicio y de final del bloque. Se almacena en el registro HL.
- La longitud del bloque en Bytes. Se almacena en el registro BC (Byte Counter)

Con estos dos valores es posible definir bloques de hasta 64K de longitud, que comienzan en cualquier lugar (HL) de la memoria. Puesto que el bloque así definido ha de ser transferido, deberá ser dada antes la dirección de inicio o la dirección final del bloque a transferir. Se almacenará en DE. Una vez colocados estos datos en los registros, puede efectuarse el verdadero comando de transferencia de bloques. Existen cuatro comandos de transferencia de bloques:

LDD, LDDR, LDI, LDIR

Cada comando de transferencia de bloque decrementa, (disminuye) el contador BC después de cada transferencia de un Byte. Dos de ellas, LDI y LDIR, incrementan los indicadores HL y DE, que indicarán la dirección origen y la dirección destino del siguiente Byte a transferir.
Contrariamente a ello, LDD y LDDR decrementan los indicadores; es decir, el bloque se transfiere, por así decirlo, "empezando por arriba". Para estos comandos, HL y DE también deberán estar cargados al principio con la dirección de inicio, o bien, la dirección de destino del bloque. La R que aparece al final del comando corresponde a

Repeat (ingl.:repetir). Estos comandos se repiten de forma automática hasta que BC=0; es decir, hasta que queda transferido todo el bloque. Para cada comando en particular es válido lo siguiente:

LDI : Carga e (I)ncrementa

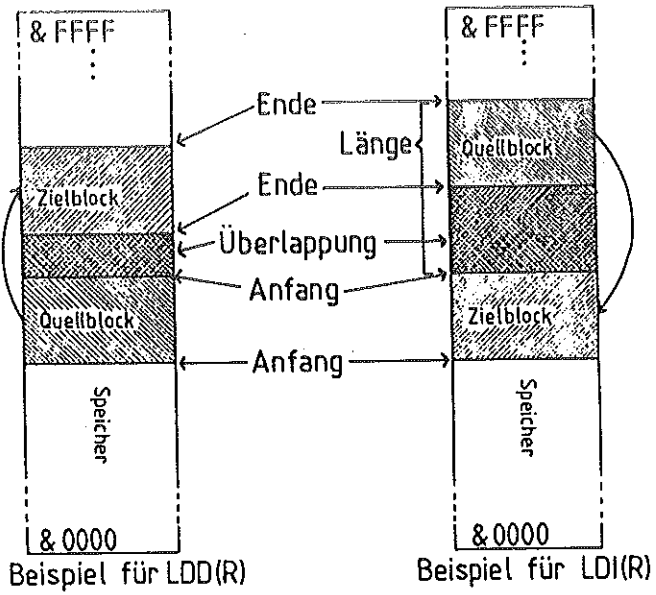
Este comando transfiere un Byte de la dirección HL a la dirección DE. A continuación, se decrementa BC. Los indicadores de dirección HL y DE se incrementan, de modo que todo queda dispuesto para una eventual continuación de la transferencia. Para ello, se deberá saltar de nuevo a este comando

LDIR: carga, incrementa y repite

El proceso de la transferencia se realiza de igual modo que en LDI. A continuación, el PC apunta de nuevo automáticamente a este comando. Luego, se vuelve a ejecutar hasta que BC=0. Finalmente, continúa el programa con el comando siguiente.

LDD : Carga y (D)ecrementa

Similar a LDI, sólo que aquí, el bloque se transfiere empezando por la dirección final; es decir, HL y DE se decrementan. Esta diferencia es importante cuando se entrecruzan el bloque de destino y el bloque de origen. Si se utilizase aquí un comando equivocado, en según que condiciones, serían modificados datos del bloque original antes de haber sido transferidos.



Ende=fin
Zielblock=bloque de destino , Anfang=inicio
Quellblock=bloque origen , Laenge=longitud
Überlappung=solapamiento , Beispiel fuer=ejemplo para
Speicher=Mnemonic
Esq. 6 Comandos de transferencia de bloques

LDD : carga, decrementa y repite

Similar a LDD, sólo que aquí, al igual que en LDIR, se repite el comando, hasta que todo el bloque queda transferido.

Ejemplo:

```
LDIR     BASIC: 10 POKE DE,PEEK(HL)
```

```
20 HL=HL+1
30 DE=DE+1
40 BC=BC-1
50 IF BC<>0 THEN 10
```

LDD BASIC: POKE DE,PEEK(HL)
DE=DE-1:HL=HL-1:BC=BC-1

Piense usted mismo como sería la analogía en BASIC para LDDR y LDI.

En el tamaño del programa BASIC, usted puede observar que se trata de un comando de gran capacidad.
Influencia de Flags: Cuando tras la ejecución, BC=0,P/V=0.
Los comandos Repeat LDDR y LDIR siempre activan los Flag P/V a 1.

Comandos de búsqueda de bloque

Con ayuda de los comandos de búsqueda de bloque, un bloque de datos puede ser buscado por un determinado Byte. El Byte buscado se almacena antes en el Acumulador. Si durante la búsqueda, el comando topa con un Byte, que es igual al contenido del Acumulador, se activa el Flag Z, y se detiene o bloquea la repetición de los comandos Repeat. Los registros se utilizan igual que en los comandos de transferencia de bloque.

HL- Dirección inicio o final del bloque
BC- Byte Counter: Longitud del bloque
DE- no tiene función
A- El Acumulador contiene el Byte que debe ser buscado

CPIR compara en cada transferencia el contenido de la posición de memoria HL con el contenido del Acumulador. Luego, se incrementa HL y se decrementa BC. Si B=0, el Flag P/V se sitúa en 0, y sinó, en uno. Si en la comparación de A y (HL) se presenta alguna igualdad, se activa el Flag Z, y sinó, se desactiva.
El Flag S corresponde, como en CP, al séptimo Bit del resultado de la resta A-(HL). El Carry no queda influído.
Hay cuatro comandos de búsqueda de bloque:

CPI, CPIR, CPD, CPR

Su modo de funcionamiento corresponde al de cada comando de transferencia de bloque.
Todos los comandos de bloque son comandos de 2 Bytes, y su primer Byte de código de operación es &HED. Al igual que en los comandos de transferencia de bloque, con los comandos de búsqueda, la programación se hace más sencilla y más rápida en muchos campos.
La lista de comandos para los comandos de transferencia de bloques y de búsqueda de bloques se encuentran al final del capítulo anterior.

Ejercicio:

Para comprender del todo el comando LDIR, lo probaremos enseguida. Queremos volver a componer la programación de las teclas de función que aparece al conectar el ordenador (si han sido cambiadas con el comando >KEY<). Como podemos cambiar la función de las teclas, la función actual de las teclas deberá estar registrada en la memoria RAM. Además, la programación de las teclas deberá estar contenida una vez en la ROM, para que esté presente al ser conectado el ordenador. La copia de la ROM está a partir de &H13A9; la programación actual está registrada a partir de la dirección &HF87F. Puesto que para cada tecla de función es posible tener una programación de una longitud máxima de 16 caracteres, la longitud del bloque será de 10*16=&HA0 Bytes. Escriba ahora un programa que transfiera la programación de las teclas de función de la ROM a la RAM.

Solución:

```
Bloque origen     a partir de &H13A9  (=registro HL)
Bloque destino    a partir de &HF87F  (=registro DE)
longitud de bloque      :&HA0         (=registro BC)
```

Puesto que conocemos la dirección de inicio de los bloques a transferir, utilizaremos el comando LDIR. Así se obtiene el siguiente listado Ensamblador:

```
   LD HL,&H13A9
   LD DE,&HF87F
   LD BC,&H0010
   LDIR
   RET
```

Una vez traducido este programa, las líneas DATA del cargador BASIC serán:

```
DATA &H21,&HA9,&H13,&H11,&H7F,&HF8
DATA &H01,&H00,&HA0,&HED,&HB0,&HC9
```

(La dirección de inicio es &HF000 y la dirección final es &HF00B)
Ahora, cambie también la programación de las teclas de función con el comando BASIC >KEY<, para que pueda detectar una modificación. Luego, cargue el programa con >RUN< e inícielo con >X=USR1(1)<. Para ver la nueva programación, aún deberá ejecutar el comando >CLS<.

Los comandos de carga de bloque también pueden ser utilizados para borrar un bloque de datos. Para ello, deberemos utilizarlos erróneamente a propósito:
Primero, almacenaremos en la posición &HF87F el Byte Cero o Nulo. Luego, trasladaremos el bloque de &HF87F a &HF87F+&HA0=&HF91F hasta &HF880. Ya que las áreas se solapan a la dirección Final del bloque de origen, tendríamos que utilizar en realidad LDDR. Sin embargo, utilizamos LDIR, HL=&HF87F, DE=&HF880, BC=&HA0, de este modo, la posición de memoria que ha de ser transferida a continuación, siempre se escribirá con el valor de la que acaba de ser transferida. ¡¡Si &HF87F tiene el valor 0, significa que todos los Bytes de ese bloque tienen el valor cero!!

El prgrama completo tiene la siguiente forma:

```
Direcc/Codigo/ N.Lin.-BASIC/Comando Ensamblador

F000   217FF8      10      LD HL,&HF87F
F003   3600        20      LD (HL),0
F005   1180F8      30      LD DE,&HF880
F008   01A000      40      LD BC,&HA0
F00B   EDB0        50      LDIR
F00D   C9          60      RET
```

Explicación del listado Ensamblador:
La dirección es numerada consecutivamente según la cantidad de Bytes en el código. Puesto que un Byte sólo puede ser indicado por dos cifras hexadecimales, tiene lugar el salto, de momento incomprensible, de &HF000 a &HF003.

Aquí, el código está compuesto por 3 Bytes: &H21, &H7F, &HF8. Puesto que cada Byte eleva la dirección el valor en uno, la dirección de inicio del Byte siguiente será: &HF003 (&HF000+3=&HF003). La longitud de los comandos se obtiene facilmente de la cantidad de códigos. Los comandos Ensamblador están detrás de los códigos.

4.6 COMANDOS ARITMETICOS

Los primeros ordenadores digitales que aparecieron en los aõs 50 eran considerados fundamentalmente como máquinas calculadoras. Aunque aquellas primeras computadoras poco tienen que ver con los ordenadores actuales, los comandos para la aritmética sí que son similares. Existen dos operaciones aritméticas básicas, la adición y la substracción, que corresponden a los comandos en lenguaje máquina ADD y SUB. Como el ordenador calcula en el sistema binario, observemos de momento, como se realizan estas operaciones en el sistema numérico.

Adición:

En el sistema decimal se suman dos cifras colocadas una encima de otra. La posición de la unidad es anotada y las eventuales posiciones de las decenas (el resto) son memorizadas para la adición de las siguientes cifras.
Ejemplo:

```
   3573
+  7154      (* Aquí debería usted memorizar un uno
-------       al acarreo. Esta cifra corresponde
  10727
  * *
```

Un acarreo aparece siempre que la suma de dos cifras es mayor que 9 (10-1). En el Sistema binario aparece un acarreo cuando la suma de dos cifras es mayor que 1 (2-1).

Reglas:

```
0 + 1 = 1
1 + 0 = 1
0 + 0 = 1
1 + 1 = 0 <--(¡en la última suma usted "llevarse
            uno"!)
```

Aplicación:

```
  1 0 0 1 0 1 1 0  = &H96 = 150
+ 0 0 1 1 1 0 0 1  = &H39 =  57
  ---------------    -----   ---
  1 1 0 0 1 1 1 1  = &HCF = 207
    * *
  (* significa: ¡ 1 de acarreo!)
```

En el sistema hexadecimal se obtiene lo mismo (v.a.):

Un acarreo aparece cuando el resultado es mayor que 15

```
  0 0 1 0 1 1 1 0  = &H2E =  46
+ 0 0 0 1 0 1 1 1  = &H17 =  23
  ---------------    -----   ---
  0 1 0 0 0 1 0 1  = &H45 =  69
```

&HE+&H7= 14+7= 21 =&H15

o sea: ¡anotar 5, acarrear 1!

Además, en el ejemplo anterior aún se da otro caso en la adición binaria:

```
  &H11
 +&H11
 -----
 &H110
```

En la segunda posición es válida la siguiente regla:

¡ 1 + 1 + 1 = 1, y 1 de acarreo!

Ejercicios:

```
1)    1 0 1 0 1 1 1 0  = &H ?  =  ?
    + 0 0 1 0 1 1 1 1  = &H ?  =  ?
      ----------------    -----   ---
            ?          = &H ?  =  ?

2)    0 0 1 1 1 1 1 1  = &H ?  =  ?
    + 0 0 1 0 1 1 1 1  = &H ?  =  ?
      ----------------    -----   ---
            ?          = &H ?  =  ?

3)    1 1 1 1 1 1 1 1  = &H ?  =  ?
    + 1 1 0 0 1 0 1 0  = &H ?  =  ?
      ----------------    -----   ---
            ?          = &H ?  =  ?
```

Solución:

```
1)   1 0 1 0 1 1 1 0  = &HAE  = 174
   + 0 0 1 0 1 1 1 1  = &H2F  =  47
   -------------------   ----   ---
     1 1 0 1 1 1 0 1  = &HDD  = 221


2)   0 0 1 1 1 1 1 1  = &H3F  =  63
   + 0 0 1 0 1 1 1 1  = &H2F  =  47
   -------------------   ----   ---
     0 1 1 0 1 1 1 0  = &H6E  = 110


3)   1 1 1 1 1 1 1 1  = &HFF  = 255
   + 1 1 0 0 1 0 1 0  = &HCA  = 202
   -------------------   ----   ---
   1 1 1 0 0 1 0 1 1  =&H1C9  = 457
```

Concerniente a 3). En esta adición aparece un acarreo del lugar 8 (Bit 7) al lugar 9 (Bit 8). Sin embargo, un Byte sólo tiene 8 posiciones (8 Bits). Por eso el "Bit de acarreo", el Carry, es almacenado en el Bit 0 del registro Flag. En principio, también se pueden sumar números de varias cifras. Pero para ello hay que proceder de otro modo en el ordenador.

Resta

La resta en el sistema binario es análoga a la del sistema decimal.
Existen las siguientes reglas:

```
0-1=1   memorizar 1
1-0=1
```

```
0-0=0
1-1=0
```

Observemos un ejemplo:

```
  01101110=&H6E=110
- 00110101=&H35= 53
  -------- ---- ---
  00111001 &H39  57
    **   *
```

Reconoceremos las reglas especiales para la continuación del cálculo con el acarreo:

```
1-(1+1)=1    acarrear
0-(1+1)=0    acarrear
```

Ejercicios:

Realice los mismos ejercicios de la suma para la resta. Compruebe usted mismo los resultados partiendo de la conversión al sistema decimal.

Concerniente a 2). Tras finalizar el cálculo, el resultado da una cifra negativa. El resultado correcto sería 63-157=-84. En forma binaria sería:

```
  00111111
- 10011101
  --------
 110100010=&H1A2
```

Evidentemente, este resultado es erróneo. En la resta binaria con el ordenador se presenta el problema de que aparecen números negativos. Para ello, se ha creado la siguiente convención:

El Bit 7 de un número binario se utiliza como Bit de signo. 0 significa cifras positivas y 1 significa cifras negativas. Con ésto se limita, de -128 hasta +127, el margen de números codificados por un Byte. La resta de números binarios lleva, de este modo, a la suma de números con signo (5-2=5+(-2)). A la representación con signo que se utiliza en la sustracción se le denomina complemento a dos.

¿Qué es el complemento a dos?

En la representación del complemento a dos, los números positivos se representan igual que hasta ahora. P.ej.: 5=&B00000101, 126=&B01111110.
Un úmero negativo viene representado calculando primero su complemento. El complemento es el número binario, en el que todos los Bits tienen el valor opuesto, 0 se convierte en 1 y 1 se convierte en 0. El número binario obtenido se llama "Complemento a uno" o simplemente Complemento.

Ejemplo:

```
Número      : 7=&B00000111
Complemento:   &B11111000
```

Para obtener el complemento a dos del número, se deberá sumar 1.

Ejemplo:

```
Complemento          &B11111000
más  1             +         1
                     ----------
Complemento a dos    &B11111001
```

Esta es la representación de -7 en el complemento a dos.

Así pues, el complemento a dos se forma de la siguiente manera:

- un número positivo permanece invariable
- en los números negativos se forma el complemento y se le suma 1

Representación del complemento a dos:

Decimal	Complemento de dos
+127	&B01111111
+126	&B01111110
+125	&B01111101
.	.
.	.
.	.
+ 2	&B00000010
+ 1	&B00000001
0	&B00000000
- 1	&B11111111
- 2	&B11111110
- 3	&B11111101
.	.
.	.
.	.
-126	&B10000010
-127	&B10000001
-128	&B10000000

Para conservar el valor de un número negativo representado en complemento a dos, se forma de nuevo a partir de el número el complemento a dos.

Ejemplo:

```
&B00000111   Complemento
+        1   más  1
-----------
&B00001000
```

&B00001000=8

¡Esto significa que el valor de &B11111000 es -8!
Una doble aplicación de complemento a dos lleva de nuevo al número de partida.

El Z80 pone a nuestra disposición comandos para la conversión del contenido del Acumulador en el complemento (CPL) y en el complemento a dos (NEG). A continuación, vamos a ejecutar la función de estos comandos en BASIC:

Observemos primero la formación del complemento:
A contiene un número entre 0 y 255 (1Byte). La aorden >BIN$< convierte un número en un string, que corresponde al número binario. Esta cadena la "complementaremos" Bit a Bit.

```
10 A=&B11011
20 AB$=RIGHT$("0000000"+BIN$(A),8)
30 PRINT "Num. Binario :";AB$
40 FOR I=0 TO 7
50 BI$=MID$(AB$,8-I,1):REM Bit N.I
60 IF BI$="1" THEN BI$="0" ELSE BI$="1"
70 AK$=BI$+AK$ : REM AK$ es el complemento
80 NEXT
90 PRINT "Complemento:";AK$
100 A=VAL("&B"+AK$)
```

La línea 50 estrae, si se da el caso, el Bit-I de AB$. En la línea 60 se forma el complemento del Bit; o sea, de 0 pasa a 1, y de 1 pasa a 0. En la línea 70 se reunen los Bits complementados en AK$. De todos modos, este programa es bastante lento. El comando XOR ejecuta la complementación más ráplido en BASIC. Aquí, sólo le damos el programa; el modo de funcionamiento de este comando lógico se lo explicaremos en el capítulo siguiente.

```
10 A=&B11011
20 AB$=RIGHT$("0000000"+BIN$(A),8)
30 PRINT "Num. Binario :";AB$
40 A=A XOR 255
50 AK$=RIGHT$("0000000"+BIN$(A),8)
60 PRIN "Complemento:";AK$
```

La línea 40 reliza la verdadera formación del complemento.

El comando NEG convierte un número positivo en uno negativo en codificación de complemento a dos. En BASIC obtendremos el mismo resultado añadiendo la línea 45:

```
45 A=A+1
```

tras realizar la prueba, añada también la siguiente línea:

```
100 GOTO 40
```

Tras la interrupción de este programa sin fin, comprobará que una doble realización de la formación del complemento a dos lleva de nuevo al punto de partida.
Con la representación del complemento de dos, se considera una sustracción de dos números como adición de uno de ellos con el complemento a dos del otro. Por otro lado, cuando se coloca el Bit 7, el resultado de una sustracción se considera como número negativo (en representación de complemento a dos).

Ejemplo:

```
120-63=57
120=&B01111000
 63=&B00111111
```

El complemento a dos de 63 es &B11000001
Ahora, sumaremos:

```
  01111000 =120
+ 11000001 =complemento a dos de 63
  ----------
  100111001
```

No tengamos en cuenta, de momento, el acarreo del Bit 7 al Bit 8 (Carry). Nuestro resultado es correcto:&B00111001=57

El Bit 7 no ha sido activado, es decir que el resultado es positivo. Según ésto, no debería ser activado el Carry.

Puesto que estamos calculando con el complemento a dos, el acarreo también queda, por así decirlo, complementado. En este caso no es necesario tenerlo en cuenta. Nuestro resultado es correcto de todos modos. La observación exacta de la aritmética con números con signo demuestra que deben ser tenidos en cuenta algunos casos especiales. Aquí es importante el efecto conjunto de los Flags.

Ejercicio:

Calcule el complemento a dos de:

1) -60
2) -120
3) +5
4) -6

Soluciones:

1) &B11000100(=196)
2) &B10001000(=136)
3) &B00000101(=5)
4) &B11111010(=250)

Comandos aritméticos y de cálculo de 8-Bits

Existen dos comandos para la suma y la resta:

ADD;ADC y SUB;SBC

Para los comandos que finalizan en C (-Carry), se deberá tener en cuenta en la operación el Flag de acarreo en la forma correspondiente. Al utilizar uno de estos dos comandos, se sumará o restará el Bit 0 del registro F (¡¡el carry!!).
Los operandos de estos comandos presentan el formato:

A,x donde x corresponde a r, n, (HL), (IX+d) o (IY+d)

De ahí se obtienen los siguientes tipos de instrucciones:

A,r	- Implícito
A,n	- Inmediato
A,(HL)	- Indirecto
A,(IX+d)	- Indexado
A,(IY+d)	- Indexado

Con el comando SUB sólo se indican como operandos: r, n, (HL), (IX+d) o (IY+d). "A" queda excluído debido a que todos los comandos de este tipo se refieren al Acumulador.
Estos comandos son operaciones de 8-Bits. El Z80 contiene además comandos aritméticos de 16-Bits.

Los siguientes Flags son influídos al ejecutar los comandos que manipulan datos:

Flag de acarreo

El Carry es activado cuando tiene lugar un acarreo del Bit 7 al Bit 8. Puesto que un Byte se compone sólo de los Bits 0 a 7, el acarreo se almacena en el Flag C. En caso contrario, el Flag de acarreo es retirado.

Flag N y H

Estos Flags son influenciados en la aritmética BCD, pero carecen de importancia para el programador.

Flag P/V Overflow

Un desbordamiento se define del siguiente modo:

- Cuando hay un acarreo interno del Bit 6 al Bit 7, y no hay acarreo del Bit 7 al Bit 8 (el denominado acarreo externo se indica mediante el Carry)

- Cuando no hay ningún acarreo interno, pero sí uno externo.

No precisaremos aquí como han sido formuladas estas definiciones. Lo importante es saber que este Flag es activado, cuando en una operación aritmética, el signo del resultado (Bit 7) se modificó con error. El Flag V es activado cuando tiene lugar un desbordamiento, de lo contrario, es retirado.

Flag Cero

Este Flag es activado cuando el resultado de la operación es 0, de lo contrario, es retirado.

Flag de signo

Este Flag corresponde al Bit 7 del resultado. En la representación numérica con signo, éste corresponde al signo, de ahí su nombre Flag de signo.

En el apéndice encontrará una tabla para la influencia de los Flags.

En lo sucesivo, para la explicación de los comandos escribiremos lo siguiente para el estado de un Flag después de una operación:

1- El Flag es activado tras la operación
0- El Flag es retirado tras la operación
↕- El Flag es activado o retirado, según resultado de la operación.
P- El flag P/V indica paridad
V- El flag P/V indica Overflow
- No hay influencia
X- El Flag es desconocido tras la operación

Ejemplo: Flags S Z V C
X ↕ 1
significa:
S - desconocido
Z - cuando 0, entonces 1, y viceversa
P/V - 1 Overflow
C - No hay influencia

Analogías en BASIC para los comandos:

ADD A,H BASIC: A=A+H

ADC A,&HA9 BASIC: A=A+&HA9+CF

CF es el Flag de acarreo, su valor se suma adicionalmente.

SUB A,(HL) BASIC: A=A-PEEK(HL)
SBC A,L BASIC: A=A-L-CF

Ejemplos:

ADD A,(HL) A =&H1F
HL=&HB1C9

```
Posición de memoria &HB1C9: &H43

   &H1F =   0 0 0 1 1 1 1 1
 + &H43 =   0 1 0 0 0 0 1 1
            ---------------
            0 1 1 0 0 0 1 0
          8 7 6 5 4 3 2 1 0 - Número Bit

Bit 8= 0 => Flag de accarreo =0
Bit 7= 0 => Flag de signo =o
Resultado <>0 => Flag de Cero=0
Acarreo externo. = 0 y Acarreo interno.= 0 => Desbordamiento
Flag (P/V) acumulador = 0

Contenido del Acumulador tras la operación:&B01100010=&H62

ADD A,D      A contiene &HE1
             D contiene &HA2

     &HE1 =   1 1 1 0 0 0 0 1
   + &HA2 =   1 0 1 0 0 0 1 0
   ------     ---------------
    &H183 = 1 1 0 0 0 0 0 1 1
            8 7 6 5 4 3 2 1 0 - Número Bit

Bit 8=1 => Flag de acarreo = 1
Bit 7=1 => Flag de signo = 1
Resultado no nulo          => Flag de cero = 0
Acarreo externo e interno => Flag (P/V) de desbordamiento=0

Contenido del acumulador tras la ejecución: &B10000011=&H83

Como usted ve, el acumulador no contiene el resultado
correcto. Sólo obtenemos el resultado correcto cuando se
añade el Flag de acarreo como octavo Bit. Por este motivo
es importante comprobar el estado de los Flags tras las
operaciones
aritméticas, y corregir así los eventuales resultados
incorrectos.
```

Observe además, que en una operación cuyo resultado es exactamente 256, se activa el Flag de Cero, aunque el resultado no sea nulo.

```
ADC  A,&H19          A=&H5A
                   Flag de acarreo = 1 (activado)

A     &H5A = 0 1 0 1 1 0 1 0
   +  &H19 = 0 0 0 1 1 0 0 1
             ---------------
      &H74 = 0 1 1 1 0 1 0 0

   Flags:  S Z V C          Acumulador = &B 01110100 = &H74
           0 0 0 0
```

Recuerde: Si se borra el Carry antes de un comando ADC, éste corresponderá exactamente al comando ADD.

```
SUB A,(HL)

        A   contiene &H3C
       HL   contiene &HBC19
    &HBC19  contiene &H15

    0 0 1 1 0 1 1 0    &H36
    1 1 1 0 1 0 1 1    compl. a dos de &H15
   ----------------
  1 0 0 1 0 0 0 0 1

  Bit 7=0 => Flag de signo = 0
  Bit 8=1 => Flag de acarreo = 0
```

Observe que aquí se ha tomado el complemento del verdadero Carry (¡caso especial!)

Ningún desbordamiento V=0
Resultado <> 0 =>Z=0

Contenido del Acumulador tras la ejecución &B00100001=&H21

```
SBC A,B
    A=&H57
    B=&H73

      CF=1

   0 1 0 1 0 1 1 1 = &H57
 + 1 0 0 0 1 1 0 1 = Compl. a dos de &H73
 + 1 1 1 1 1 1 1 1 = compl. a dos de &H1(CF)
 -----------------
 1 1 1 1 0 0 0 1 1

 Flags: S Z V C
        1 0 0 1
```

Contenido del Acumulador B11100100 = &HE4
es el Complemento a dos de 29
es decir, el resultado es -29 (87-115-1=-29)

Además de la aritmética binaria existe tambin otra posibilidad de operar con números en el ordenador:

Aritmética BCD
Aquí se representa cada cifra del sistema decimal por un bloque de 4 Bits. Esta aplicación es importante para el tratamiento de problemas comerciales, que exige mantener una perfecta exactitud en cifras decimales. En el BASIC MSX, los números son almacenados con simple o doble precisión en el formato BCD. Para las operaciones BCD existe el comando especial DAA, que prepara el contenido del Acumulador para tales operaciones (ver capítulo 5.4).

Además, también están los comandos especiales CPL y NEG, ya descritas.
CPL complementa el contenido del Acumulador y NEG lo niega; es decir, lo transforma en un complemento a dos.
Esta aritmética también convierte algunos comandos "normales" en comandos especiales; p.ej., se puede utilizar SUB A para borrar el Acumulador. Este comando lo se efectúa casi el doble de rápido y es la mitad de corto que el comando LD A, 0.

A este grupo de comandos corresponden también los comandos de contador, que aumentan o disminuyen el valor de una memoria. Para los comandos de contador se dispone del direccionamiento implícito, indexado, y del direccionamiento de registro. Los comandos de este tipo se utilizan a menudo para la programación de bucles. Su método de funcionamiento es sencillo:

INC x aumenta x, y
DEC x disminuye x, pudiendo ser x lo siguiente:

r, (HL), (IX+d), (IY+d)

INC r	BASIC: r=r+1
DEC (HL)	BASIC: POKE HL,PEEK(HL)-1

Cada uno de los Flags de signo, de cero y V se activan o retiran, según el resultado de la operación. El Carry permanece inalterado. Es importante el hecho de que sólo los comandos de contador de 8-Bits influencian los Flags. En los comandos de contador de 16-Bits hay que hacer adicionalmente una comparación.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	V	↕	0	↕	10 [000] r	1	1	4	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, n	A ← A + n	↕	↕	V	↕	0	↕	11 [000] 110 ← n →	2	2	7	
ADD A, (HL)	A ← A + (HL)	↕	↕	V	↕	0	↕	10 [000] 110	1	2	7	
ADD A, (IX+d)	A←A + (IX+d)	↕	↕	V	↕	0	↕	11 011 101 10 [000] 110 ← d →	3	5	19	
ADD A, (IY+d)	A←A+(IY+d)	↕	↕	V	↕	0	↕	11 111 101 10 [000] 110 ← d →	3	5	19	
ADC A, s	A ← A + s + CY	↕	↕	V	↕	0	↕	[001]				s is any of r, n, (HL), (IX+d), (IY+d) as shown for ADD instruction
SUB s	A ← A − s	↕	↕	V	↕	1	↕	[010]				
SBC A, s	A ← A − s − CY	↕	↕	V	↕	1	↕	[011]				
AND s	A ← A ∧ s	0	↕	P	↕	0	1	[100]				
OR s	A ← A ∨ s	0	↕	P	↕	0	0	[110]				The indicated bits replace the 000 in the ADD set above.
XOR s	A ← A ⊕ s	0	↕	P	↕	0	0	[101]				
CP s	A − s	↕	↕	V	↕	1	↕	[111]				
INC r	r ← r + 1	•	↕	V	↕	0	↕	00 r [100]	1	1	4	
INC (HL)	(HL) ← (HL)+1	•	↕	V	↕	0	↕	00 110 [100]	1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	•	↕	V	↕	0	↕	11 011 101 00 110 [100] • d •	3	6	23	
INC (IY+d)	(IY+d) ← (IY+d) + 1	•	↕	V	↕	0	↕	11 111 101 00 110 [100] ← d →	3	6	23	
DEC m	m←m-1	•	↕	V	↕	1	↕	[101]				m is any of r, (HL), (IX+d), (IY+d) as shown for INC. Same format and states as INC. Replace 100 with 101 in OP code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow. P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

En principio, los comandos aritméticos de 16-Bits se asemejan a los de 8-Bits. Los comandos de 16-Bits son más limitados. Unicamente los comandos ADD,ADC y SUB se presentan para algunos registros pares. El resultado de una operación se coloca basicamente en el registro par HL (no en el acumulador, como en los comandos de 8-Bits). Con el comando ADD existe la posibilidad de almacenar, así mismo, resultados en los registros índice.
Los comandos de 16-Bits corresponden a varias ejecuciones sucesivas de comandos de 8-Bits. Como la asociación de estos comandos se produce de manera automática, los comandos de 16-Bits son más rápidos y más cortos.

16-Bits	8-Bits
ADD HL,BC	LD A,L
	ADD A,C
	LD L,A
	LD A,H
	ADC A,B
	LD H,A

Algunos comandos aritméticos de 16-Bits utilizan el direccionamiento implícito. La influencia de Flags en ADC y SBC es análoga a la de los comandos de 8-Bits. Con ADD sólo se influencia el Carry, ¡y en los comandos de 16-Bits INC y DEC, no se tiene en cuenta para nada los Flags!

```
ADD IX,DE        BASIC: IX=IX+DE
ADC HL,BC        BASIC: HL=HL+BC+CF
SBC HL,SP        BASIC: HL=HL-SP-CF
```

Ejemplo:

```
HL=&HC000
DE=&H0800
```

ADD HL,DE

```
  &HC000 = 1100 0000 0000 0000
+ &H0800 = 0000 1000 0000 0000
  -------  --------------------
  &HC800 = 1100 1000 0000 0000
```

Flag: S Z V C
0
Los Flags S, Z, V no son afectados.

HL=&HF800
DE=&H0800

ADC HL,DE

```
  &HF800 =  1111 1000 0000 0000
+ &H0800 =  0000 1000 0000 0000
  -------   --------------------
 &H10000 = 1 0000 0000 0000 0000
```

Flag: S Z V C
0 0 1 1

También aquí, HL no contiene el resultado correcto &H10000, sinó que contiene 0. El Flag de acarreo indica este error. En las operaciones de 16-Bits, es el Bit 16 el que pone de manifiesto este error.

Todas los comandos de cálculo de 16-Bits son de direccionamiento implícito. Pueden referirse a los registros de 16-Bits: BC, DE, HL, SP, IX e IY. Contrariamente a los comandos de cálculo de 8-Bits, estos comandos no (!) influyen sobre los Flags.

Mnemonic	Symbolic Operation	Flags						Op-Code	No. of Bytes	No. of M Cycles	No. of T States	Comments	
		C	Z	P/V	S	N	H	76 543 210					
ADD HL, ss	HL ← HL+ss	↕	•	•	•	0	X	00 ss1 001	1	3	11	ss	Reg.
												00	BC
ADC HL, ss	HL←HL+ss+CY	↕	↕	V	↕	0	X	11 101 101	2	4	15	01	DE
								01 ss1 010				10	HL
SBC HL, ss	HL←HL-ss-CY	↕	↕	V	↕	1	X	11 101 101	2	4	15	11	SP
								01 ss0 010					
ADD IX, pp	IX ← IX + pp	↕	•	•	•	0	X	11 011 101	2	4	15	pp	Reg.
								00 pp1 001				00	BC
												01	DE
												10	IX
												11	SP
ADD IY, rr	IY←IY+rr	↕	•	•	•	0	X	11 111 101	2	4	15	rr	Reg.
								00 rr1 001				00	BC
												01	DE
												10	IY
												11	SP
INC ss	ss ← ss + 1	•	•	•	•	•	•	00 ss0 011	1	1	6		
INC IX	IX ← IX + 1	•	•	•	•	•	•	11 011 101	2	2	10		
								00 100 011					
INC IY	IY ← IY + 1	•	•	•	•	•	•	11 111 101	2	2	10		
								00 100 011					
DEC ss	ss ← ss - 1	•	•	•	•	•	•	00 ss1 011	1	1	6		
DEC IX	IX ← IX - 1	•	•	•	•	•	•	11 011 101	2	2	10		
								00 101 011					
DEC IY	IY ← IY - 1	•	•	•	•	•	•	11 111 101	2	2	10		
								00 101 011					

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

Ejercicio:

Una vez dejado atrás el tramo más árido, vamos, por fin, a aplicar por primera vez los nuevos comandos. Escriba un programa pequeño para la suma de dos cifras de 8-Bits. Las cifras serán almacenadas en la memoria RAM mediante los comandos >POKE< en BASIC. El resultado de la suma deberá ser registrado de nuevo en el RAM. Después de retroceder de nuevo al BASIC, el resultado podrá ser leído y proyectado utilizando >PEEK<.

Solución:

Puesto que las sumas de 8-Bits utilizan basicamente el Acumulador, el primer sumando deberá ser almacenado en el Acumulador:

```
LD A, sumando1
```

El segundo sumando se almacenará en uno de los registros de 8-Bit:

```
LD H,sumando2
```

Ahora, realicemos la suma:

```
ADD A,H
```

El resultado deberá ser colocado en la posición de memoria &HF100:

```
LD (&HF100),A
```

Si elegimos como dirección de inicio &HF000, obtenemos el siguiente cuadro:

```
F000 3E10     10   LD   A,&H10
F002 2620     20   LD   H,&H20
F004 87       30   ADD  A,H
F005 3200A1   40   LD   (&HF100),A
F008 C9       50   RET
```

La línea DATA del cargador será:

```
60 DATA &H38,&H10,&H26,&H20,&H84,&H32,&H00,&HA1,&HC9
```

Del listado Ensamblador se deduce que el primer sumando es almacenado en la dirección &HF001, y el segundo, en la dirección &HF003. En nuestro caso, hemos elegido aquí &H10 y &H20. El programa BASIC, que establece estos valores, ejecuta el programa y extrae el resultado, tiene el siguiente aspecto:
(Naturalmente, antes deber ser cargado el programa en lenguaje máquina con el cargador BASIC.)

```
10 POKE &HF001,S1
20 POKE &HF003,S1
30 X=USR1(1)
40 PRINT PEEK (&HF100)
```

Gran parte de los saltos son condicionales; es decir que dependen del estado de los Flags. Por eso, volveremos a describir aquí, de forma resumida, el papel que juega cada Flag.
Los dos Flags H y N son utilizados en la aritmética BCD. No pueden ser comprobados. Los demás Flags (C, P/V, Z, S) sí pueden ser comprobados en los saltos.

Flag de acarreo, C

El Flag de acarreo tiene dos funciones.
- Indica si se ha producido un acarreo en una adición o sustracción.
- Los comandos SRL, SRA, SLA, RR, RL, RRC, RLC, RRA, RLA, RRCA y RLCA utilizan el Carry como noveno Bit.

Una excepción la constituyen los comandos de rotación RLD, RRD en BCD. Estos no influyen sobre el Carry.
Los comandos lógicos AND, OR y XOR ponen el Carry siempre a 0. Pueden ser utilizadas para borrar el Carry. Los siguientes comandos originan además una modificación del Carry.

NEG: El Flag C es activado, si antes de ejecutarse el comando, A era 0.

DAA: La influencia de este comando es compleja. Sirve para la realización de cálculos en formato BCD (ver cap.5.4).

SCF: Set Carry Flag
Este comando coloca el Carry=1

CCF: Complement Carry Flag
Este comando complementa el Carry.

¡Todos los demás comandos no influencian el Carry!

Parity/Overflow (Paridad/desbordamiento-P/V)

Este Flag presenta varias funciones, que dependen del comando ejecutado.

- Desbordamiento

En las operaciones aritméticas ADD de 8-Bits, ADC, SUB, SBC, INC de 8-Bits, DEC de 8 Bits, NEG y en CP, se indica un desbordamiento. Esto significa que el signo de un Número ha sido modificado con error.
Excepciones: ADD de 16-Bits, INC de 16-Bits, DEC de 16-Bits
Estos comandos no influyen sobre V.

- Paridad

En los comandos Input (IN), en los de rotación y de en los de traslado RR, RL, RRC, RLC, RLD, RRD, SLA, SRA y SRL, en los comandos lógicos AND, OR, XOR, y en DAA, este Flag es utilizado como Flag P. P es 1 cuando la cantidad de unos de un Byte es par, y es 0, cuando el número de Bits activados es impar.
Excepción: ¡RLA, RRA, RLCA, RRCA no influencian P!

- En los comandos de bloque LDD, LDI, CPD, CPI, CPDR y CPIR, se retira P/V, si BC=0 (BC es el registro contador), en caso contrario, se activa.
Por esta razón, P/V siempre es retirado por LDDR y por LDIR.

-Flag de Interrupción

P/V presenta el valor del Interrupt Enable Flip Flop (IFF) en LD A,I y LD A,R. Este pasa a 0 cuando no están permitidas las interrupciones enmascarables, y a 1, cuando están permitidas.

ATENCION: El comando BIT y todos los comandos de entrada/salida de bloques activan este Flag a voluntad, es decir que, en según que casos, modifican el valor anterior. Otros comandos no influyen sobre este Flag.

Flag de cero (Cero,Z)

El Flag Z indica si el valor de un Byte es nulo. Si es 0, se activa Z, de lo contrario, se retira.
En los comandos de comparación, Z se activa a 1 cuando existe una igualdad, sinó, se retira.
En el comando BIT, el Flag de cero se activa a 1 cuando el Bit comprobado =0, sinó, es retirado.
Los siguientes comandos influencian el Flag Z:

```
aritméticos           : ADD,ADC,SUB,SBC,INC,DEC,NEG,DAA
      ATENCION         : ¡no influyen: ADD de 16- Bits;
                       : INC de 16-Bits y DEC de 16-Bits
comparación          : CP : Z=1 en una igualdad, sinó, Z=0
BIt                    : BIT
rotación/traslación : RR,RL,RRC,RLC,SRL,SRA,SLA,RLD,RRD
       ATENCION       : no influyen:RRA,RLA,RRCA,RLCA
búsqueda/Bloque    : CPI,CPIR,CPD,CPDR: Z=1 en igualdad
Entrada             : IN
Comandos de carga  : LD A,I o bien, LD A,R
```

Entrada/salida de bloques: Z es activado cuando, tras la ejecución, B=0; es decir, INI,IND,OUTI,OUTD influencian Z de este modo, y INIR;INDR;OTIR;OTDR activan Z a 1.
¡Todos los demás comandos no influyen sobre Z!

Flag de signo (Signo, S)

El Flag de signo contiene el valor del Bit más alto de un Byte. Este Bit corresponde al signo en la aritmética con signo.

Los siguientes comandos influyen sobre el Flag S:

Todos los comandos aritméticos o lógicos:
ADD,ADC,SUB,SBC,INC,DEC,NEG,DAA,AND,OR,XOR,CP

Los comandos de rotación y de traslación:
RL,RR,RLC,RRC,SRL,SRA,SLA,RLD,RRD

Los comandos de búsqueda de bloque CPD,CPI,CPDR,CPIR

Los comandos de entrada IN y los comandos de carga LD A,I y LD A,R

ATENCION: ADD de 16-Bits, INC de 16-Bits, DEC de 16-Bits, RLA,RRA,RLCA,RRCA: ¡no influyen!

El comando BIT y los comandos de entrada/salida de bloques, INI, IND, OUTI, OUTD, INIR, INDR, OTIR, OTDR ponen el Flag S a voluntad en un estado indeterminado.
En el apéndice encontrará para cada comando, la influenciación de los Flags.

En el Z80 existen cinco tipos distintos de saltos.

- Saltos dentro del programa principal (JUMP), que corresponden al comando BASIC >GOTO<.
- Bifurcaciones a subprogramas (CALL y RET), que corresponden a los comandos de BASIC >GOSUB< y >RETURN<
- Saltos relativos (JUMP RELATIVE) semejantes al comando BASIC >FOR-NEXT<.
- Comandos Restart (RST), que realizan una Bifurcación a una dirección fijada de antemano. El comando RST no posee análogo en BASIC.
- Saltos de interrupción (ver comandos de control)

Los tres primeros tipos de bifurcación existen como incondicionales y condicionales en el Z80, dependiendo del estado de un Flag.

En los saltos condicionales, se puede saltar debido a los Flags Z, C, P/V y S. Cada Flag puede ser comprobado por el valor 0 o 1.
En lenguaje ensamblador se dan las siguientes abreviaciones:

Z=	Salta cuando nulo	(Z=1)
NZ=	Salta cuando no nulo	(Z=0)
C=	Salta cuando hay acarrep	(C=1)
NC=	Salta cuando no hay acarreo	(C=0)
PO=	Salta cuando paridad impar	(P/V=0)
PE=	Salta cuando paridad par	(P/V=1)
P=	Salta cuando positivo (+)	(S=0)
M=	salta cuando negativo (-)	(S=1)

Además, el Z80 conoce un comando de bucle que decrementa el registro B y ejecuta luego un salto relativo, siempre y cuando B<>0. Este comando recibe el nombre de DJNZ (Decrementa Jump Non Zero).

JUMP/JP

Las bifurcaciones en el programa principal son ejecutadas por el comando JP. La dirección de salto puede estar direccionada por dos métodos.

Direccionamiento Absoluto:

Formato:

JP nn · o bien, JP cc,nn

(cc representa una condición (Condition), o sea, para Z, NZ, C, NC, PO, PE, P o M)

JP nn - salto "incondicional" a la dirección dada.
JP cc,nn - salta a la dirección dada
cuando se cumple la condición. Si la condición no se cumple, se ejecuta el commando siguiente.

Analogía

JP nn	BASIC: GOTO Núm. línea
JP NZ,nn	BASIC: IF Z=0 THEN GOTO Núm. línea
JP Z,nn	BASIC: IF Z=1 THEN GOTO Núm. línea

El procesador realiza un salto leyendo la dirección dada en el PC. Entonces, se lee y se ejecuta en esta posición el siguiente código de comando.
En el direccionamiento absoluto, al código de operaciones de 1-Byte le sigue la correspondiente dirección de salto en la forma Byte bajo, Byte alto. Puesto que todos los comandos de 3-Byte son relativamente lentos, se hicieron posibles los saltos relativos debido a que éstos son comandos de sólo 2-Bytes. Los saltos de direccionamiento indirecto tienen un código de operaciones de 1-Byte.

Direccionamiento Indirecto

Formato:

JP (x)

x: HL, IX o IY

JP (x) salta a la dirección indicada en el registro x.

CALL/RET

Ya hemos descrito como son almacenadas o leídas las direcciones de retorno en CALL y RET. Una llamada a un subprograma puede ser condicional o incondicional. La dirección de salto (dirección de inicio de un Subprograma) se indica de forma absoluta.

Formato:

CALL nn o bien, CALL cc,nn

En la ejecución, todas las operaciones necesarias se realizan en la pila, en SP y en PC. El funcionamiento es el siguiente:
Una vez completada la lectura del comando, PC indica el siguiente comando a ejecutar. A continuación, se realizan las operaciones:

(SP-1)=PC -(High Byte)
(SP-2)=PC -(Low byte)
SP=SP-2
PC=nn

El siguiente comando es leído de la dirección que indica el PC. Para finalizar un Subprograma se coloca un comando RET. El RETURN también puede ser condicional o incondicional.

Formato:

RET o bien, RET cc

Al ejecutar el comando RET ocurre lo siguiente:

PC -(Low Byte)=(SP)
PC -(High Byte)=(SP+1)
SP=SP+2

La ejecución del programa se continúa en la dirección recogida de la pila.
El comando RET, contrariamente al comando CALL, sólo tiene 1-Byte de longitud. En CALL se deberá indicar la dirección de 16-Bits, es decir que este comando tiene una longitud de 3 Bytes.
Existen dos saltos especiales de retorno RETI y RETN, que son descritos en el capítulo relativo a los comandos de control.

RESTART-REST

Este tipo de comandos de salto tiene una longitud de un Byte y por eso se ejecuta a mayor velocidad que todos los demás comandos de salto (sin contar el comando RET). El comando RST, que en lo sucesivo denominaremos Restart, ocasiona un salto de un subprograma a una dirección localizada en la parte inferior de la memoria. Existen ocho comandos Restart. Las direcciones de bifurcación del restart son &H0, &H8, &H10, &H18, &H20, &H28, &H30 y &H38.

Formato:

REST p
p: una de las direcciones de 8-Bits antes citadas.

Puesto que Restart es el comando de salto más rápido, en la parte inferior de la memoria (&H0-&H40) se localizan importantes rutinas o saltos a estas rutinas, que son utilizadas muy a menudo. Más adelante estudiaremos la función exacta de cada comando Restart.

JUMP RELATIV-JR

Los saltos relativos saltan en relación a la dirección actual. La distancia de salto deberá ser indicada. El primer Byte es el Código de operaciones, el segundo indica la distancia con signo (en complemento a dos). Este proceso se caracteriza como direccionamiento relativo. En este caso, la distancia recibe la denominación de Offset de desplazamiento.

Formato:

JR e o bien, JR x,e
e: Offset
x: Z,NC,C,NZ

Los saltos relativos condicionales sólo son posibles mediante los Flags C y F. ¿Cómo se calcula el Offset?

Observemos el primer programa del capítulo 1.2. En la dirección &HF00C se halla el comando JR. La dirección de destino es el comando LD A,&H2A en la dirección &HF003. Así pues, la diferencia es &HF00C menos &HF003=9. Como se trata de un ''Salto hacia atrás'' (La dirección de destino es menor que la ''dirección de partida del salto''(V)), el Offset es -9. Para conservar el segundo Byte del comando, deberemos restar dos del Offset.

¿Por qué es necesaria esta resta?

El procesador siempre lee primero el comando completo, en este caso, el código de operaciones (Byte 1) y el Offset (Byte 2). Tras cada "lectura", PC es aumentado en uno. Una vez leído el comando por completo, el PC se hallará en la dirección de inicio del siguiente comando. Como consecuencia de ello, el indicador de programa será mayor en dos que la dirección del comando de salto. El Z80 ejecuta el salto sumando la distancia al PC. Por este motivo deberemos tener en cuenta el aumento del PC en 2. En un ''salto hacia atrás'' será necesario saltar por encima de estos dos Bytes. La distancia que ha de ser memorizada se calcula a partir de:

-9-2=-11= &HF5 en el complemento a dos.

Este Byte se indica en el listado Ensamblador en la dirección &HF00D, a continuación del Opcode en la dirección &HF00C. En lenguaje ensamblador no se indica esta diferencia de 2. La orden es JR NZ,&HF003. El programa Ensamblador calcula automaticamente la diferencia de salto y realiza la resta de 2 y la conversión al complemento a dos. Aunque en el comando ensamblador se indica el comando de 16-Bits, aquí se trata de un salto relativo. Teniendo en cuenta la resta, es posible efectuar saltos relativos a la dirección actual desde +129 hasta -126 Bytes.

Resumamos el método de cálculo del Byte-Offset:

El comando de salto está en la dirección ADR
La dirección del salto está en la dirección ADRZ
Offset = ADRZ-ADR
Byte a memorizar: (Offset-2) en complemeto a dos.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	•	•	•	•	11 000 011 ← n → ← n →	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	•	•	•	•	11 cc 010 ← n → ← n →	3	3	10	cc: 000 NZ non zero; 001 Z zero; 010 NC non carry; 011 C carry; 100 PO parity odd; 101 PE parity even; 110 P sign positive; 111 M sign negative
JR e	PC ← PC + e	•	•	•	•	•	•	00 011 000 ← e-2 →	2	3	12	
JR C, e	If C = 0, continue	•	•	•	•	•	•	00 111 000 ← e-2 →	2	2	7	If condition not met
	If C = 1, PC ← PC+e								2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	•	•	•	•	00 110 000 ← e-2 →	2	2	7	If condition not met
	If C = 0, PC ← PC + e								2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	•	•	•	•	00 101 000 ← e-2 →	2	2	7	If condition not met
	If Z = 1, PC ← PC + e								2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	•	•	•	•	00 100 000 ← e-2 →	2	2	7	If condition not mc
	If Z = 0, PC ← PC + e								2	3	12	If condition met
JP (HL)	PC ← HL	•	•	•	•	•	•	11 101 001	1	1	4	
JP (IX)	PC ← IX	•	•	•	•	•	•	11 011 101 11 101 001	2	2	8	
JP (IY)	PC ← IY	•	•	•	•	•	•	11 111 101 11 101 001	2	2	8	
DJNZ,e	B ← B-1 If B = 0, continue	•	•	•	•	•	•	00 010 000 ← e-2 →	2	2	8	If B = 0
	If B ≠ 0, PC ← PC + e								2	3	13	If B ≠ 0

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <-126, 129>

e-2 in the op-code provides an effective address of pc +e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC\leftarrow nn$	•	•	•	•	•	•	11 001 101 ← n → ← n →	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	•	•	•	•	11 cc 100 ← n → ← n →	3 3	3 5	10 17	If cc is false If cc is true
RET	$PC_L\leftarrow(SP)$ $PC_H\leftarrow(SP+1)$	•	•	•	•	•	•	11 001 001	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	•	•	•	•	11 cc 000	1 1	1 3	5 11	If cc is false If cc is true
RETI	Return from interrupt	•	•	•	•	•	•	11 101 101 01 001 101	2	4	14	
RETN	Return from non maskable interrupt	•	•	•	•	•	•	11 101 101 01 000 101	2	4	14	
RST p	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC_H\leftarrow 0$ $PC_L\leftarrow P$	•	•	•	•	•	•	11 t 111	1	3	11	

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	P
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown
t = flag is affected according to the result of the operation.

Aplicación:

Los comandos de salto son comandos muy importantes. Con los saltos relativos, podemos, por ejemplo, programar bucles. Las llamadas al subprograma son de especial importancia para el programador:
Además de los programas escritos por uno mismo, también pueden utilizarse los ya existentes en la ROM (las denominadas rutimas del sistema; ver cap. 6 y siguientes). De este modo, disponemos de fuertes elementos de programación. Más adelante, explicaremos detalladamente la utilización de las rutinas del sistema. Pero usted ya podrá conocer antes algunas rutinas interesantes. Una de ellas sería, por ejemplo, la rutina VPOKE utilizada en el programa de demostración del capítulo 1.2. Equivale principalmente al comando LD (HL),A; sin embargo, los datos no se escriben en la RAM normal, sinó en la RAM de video. Tales rutinas del sistema finalizan siempre con el comando RET, de manera que, tras efectuar la acción deseada, nuestro propio programa sigue su curso. La dirección de llamada a la rutina VPOKE es &H7CD.
Utilizando esta rutina, escriba un programa que reproduzca en pantalla toda la serie de caracteres (códigos-ASCII 0-255) en >SREEN 0<. El área de la RAM de video donde queda registrado el contenido de la pantalla está entre la dirección &H0 y &H3C0. Escriba primero un programa BASIC para este ejercicio. ¡ Ponga entonces especial atención en el programa en lenguaje máquina, sobre todo, en el cálculo correcto del Offset para el bucle!

Solución:

Primero, el programa Basic:

```
10 FOR I=0 TO 255
20 VPOKE I,I
30 NEXT
```

Y a continuación, el listado Ensamblador del programa en lenguaje máquina:

```
F000  3E00     10      LD   A,0
F002  210000   20      LD   HL,0
F005  CDCD07   30      CALL &H7CD
F008  23       40      INC  HL
F009  3C       50      INC  A
F00A  20F9     60      JR   NZ,&HF005
F00C  C9       70      RET
```

La línea 10 coloca el valor a escribir contenido en el Acumulador en 0.
La línea 20 almacena un 0 en el contador de direcciones HL.
La línea 30 es el comienzo del bucle: Aquí, es llamada la rutina de sistema para que escriba en la RAM de vídeo el valor del Acumulador en la dirección HL.
En las líneas 40 y 50, el valor (A) y la dirección (HL) se incrementan cada una en uno. Finalmente, el comando de salto en la línea 60 decide si el programa ha finalizado o no: el Acumulador es incrementado directamente por el comando JR, influenciando a su vez a los correspondientes Flags. La pregunta será la siguiente: ''¡Continúa el Acumulador, tras el aumento, distinto de 0?''
Este caso se dará sólo hasta que el Acumulador tome el valor 255 al inicio del bucle. Si se realiza entonces el comando INC A, A tendría que ser en realidad 256. Puesto que, sin embargo, el Acumulador sólo trabaja con cifras de 1-Byte, su valor será 0 (255+1=00+Carry=1). En este caso, el bucle no se repite y el comando RET finaliza el programa.

4.8 COMANDOS LOGICOS

En los comandos para el tratamiento de datos también se incluyen los comandos lógicos.
El Z80 posee los comandos lógicos AND,OR y XOR (exclusive OR), así como el comando de comparación CP. Todos estos comandos trabajan con datos de 8-Bits. El Acumulador es siempre el registro con el que se realiza la operación lógica. Por eso, el Acumulador no se indica en el Operando del comando ensamblador (como p.ej., en ADD A,B) (p.ej. AND B).
Los cuatro comandos AND, OR, XOR y CP pueden aparecer con los siguientes tipos de direccionamiento:

- Implícito (registro A, B, C, D, E, H, L)
- Indirecto : registro (HL)
- Indexado
- Inmediato

Observemos las funciones de los comandos lógicos. Todo el mundo entiende la siguiente sentencia lógica:

"Cuando llueve, (entonces) la calle se moja."

Esta expresión es una deducción de la forma <cuando...,entonces...>.
Observemos la siguiente sentencia:

"Cuando llueve, Y yo estoy en la calle, (entonces) me mojo".

En este caso hay dos expresiones unidas por la conjunción Y. El Y lógico (Ingl.AND) expresa que ambas expresiones, es decir, "llueve" (1.expresión) y "yo estoy en la calle" (2.expresión) han de tener lugar para que se produzca el resultado. Si no llueve (no se realiza la 1. expresión), yo no me mojo. Si estoy en una casa (no se realiza la 2. expresión), tampoco me mojo. Para que la consecuencia se produzca (sea cierta), ambas expresiones han de ser

ciertas. Esta es precisamente la particularidad de la conexión AND (Y). Puesto que el ordenador trabaja con 0 y 1, se adopta la siguiente convención:

1 corresponde a ''expresión cierta''
0 corresponde a ''expresión falsa''

Con ello se obtiene:

```
1 AND 1= 1 ambas expres. son ciertas =>resultado cierto
1 AND 0= 0 una expres. es falsa      =>Resultado falso
0 AND 1= 0 una expres. es falsa      =>resultado falso
0 AND 0= 0 ambas expres. son falsas  =>resultado falso
```

El BASIC MSX contiene los comandos lógicos. Ejecútelos usted mismo:

```
PRINT 1 AND 1
PRINT 1 AND 0 etc...
```

Las operaciones lógicas son de máxima importancia para la técnica de ordenadores. Son fáciles de ejecutar electrónicamente. Para ello hay dos direcciones de entrada, que llevan corriente (=1), o que no la llevan (=0), conectadas a un circuito eléctrico, cuya dirección de salida conduce o no corriente (es 1 o 0) según las necesidades de entrada. Tales circuitos se comprenden con ayuda de la Algebra de Boole. Un microprocesador esta compuesto por múltiples puertas lógicas cerradas, situadas una después de otra. La suma en el MPU se forma. p.e., a partir de diversas operaciones lógicas.
Nosotros, como programadores que somos, no entramos en contacto con estas estructuras. Nosotros aplicamos las operaciones lógicas a datos (8-bits). Para ello se unen en operaciones lógicas cada uno de los correspondientes Bits de los dos Bytes.

```
        11111000
AND     01010011
        --------
        01010000

Bit 0: 0 AND 1=0
Bit 1: 0 AND 1=0
Bit 3: 0 AND 0=0
Bit 4: 1 AND 0=0
Bit 5: 1 AND 1=1
         .
         .
```

Una de las aplicaciones más importantes del comando AND consiste en borrar o anular determinados Bits.

```
A=&B10111001
```

Supongamos que sólo queremos observar los Bits 0 a 3, es decir, han de ser anulados desde el Bit 4 hasta el Bit 7. Para conseguirlo unimos (con Y) A con &B00001111.

```
        10111001        :A
AND     00001111        :Máscara
        --------
        00001001
```

La Máscara utilizada para anular los Bits, contiene un 0 para un Bit a anular, y un 1 para un Bit significativo. Formule en BASIC:

```
A=&B10111001
A=A AND &B00001111
```

En Lenguaje Máquina obtenemos:

```
LD A,&B10111001
AND  &B00001111
```

Observe las siguientes expresiones:

''Cuando llueve, O me baño, (entonces) me mojo.''

El resultado será cierto cuando, como mínimo, una de las dos expresiones sea cierta. Con ello, obtenemos los siguiente para la conjunción O (OR).

```
0 OR 0= 0
0 OR 1= 1
1 OR 1= 1
1 OR 1= 1
```

Con la conjunción OR es posible activar determinados Bits de un Byte.
A contiene &B10001011.
Ahora han de ser colocados los 3 primeros Bits (5, 6, 7) a 1:

```
   10001011    :A
OR 11100000    : Máscara
   --------
   11101011
```

Para cada Bit que ha de ser colocado a 1, la máscara contiene un 1, y para los Bits que no han de ser modificados, un 0.

```
LD A,&B10001011        BASIC: A=&B10001011
OR   &B11100000        BASIC: A=A OR &B11100000
```

El comando XOR o "O exclusivo", sólo se diferencia en un punto del O normal o inclusivo. Si ambos Bits de salida estan a 1, la salida será 0. El comando OR exclusivo suministra un 1 para entradas distintas y un 0 para entradas iguales:

```
0 XOR 0= 0
1 XOR 0= 1
0 XOR 1= 1
1 XOR 1= 0
```

Existen dos aplicaciones para XOR, la comparación y la complementación. Los Bytes que han de ser comparados son unidos mediante XOR. Si el resultado es 0, es que todos los Bytes eran iguales. Si se produce desigualdad, están activados los bits desiguales en el resultado.

```
     10101010
XOR  10101010    ¡Comparación!
     --------
     00000000

     10101010
XOR  10101100    ¡Comparación!
     --------
     00000110
```

=> Bit 1 y Bit 2 son distintos.

Para la complementación se unen de nuevo los Bits con una máscara. Esta contiene un 1 para un Bit que ha de ser complementado y un 0 para un Bit que permanece invariable.

Bit 4-7 han de ser complementados.

```
     10101111    :A
XOR  11110000    :Máscara
     --------
     01011111
```

Analogías:

Lenguaje Máquina	BASIC
AND H	A=A AND H
OR (HL)	A=A OR PEEK(HL)
XOR &HFF	A=A XOR &HFF

En los comandos lógicos, el Carry siempre es colocado a 0. Los Flags Z y S, como es habitual, son influenciados. El Flag P/V indica en estos comandos la paridad del resultado. La paridad será 1 cuando la cantidad de unos sea par en el Byte, y será cero, cuando sea impar.

Ejercicios:

1. ¿Qué provoca un:

   - OR con &HFF?
   - OR con &H0?
   - AND con &HFF?
   - AND con &H0?
   - XOR con &HFF?
   - XOR con &H0?

2. En BASIC existe el comando NOT. Convierta este comando a Lenguaje Máquina siguiendo dos métodos distintos (respecto al Acumulador).

Solución:

Para 1.

```
 OR &HFF => &HFF, es decir, todos los Bits están colocados
 OR &H0  => ninguna modificación
AND &HFF => ninguna modificación
AND &H0  => &H0, o sea, todos los Bits han sido retirados
XOR &HFF => todos los Bits han sido complementados
XOR &H0  => ninguna modificación
```

Para 2.

```
Orden XOR : XOR &HFF
Orden CPL : CPL
```

El comando de comparación CP

El comando CP sirve para compararar el contenido del Acumulador con un Byte. Este Byte puede tener los siguientes direccionamientos:

- Implícito : registros A, B, C, D, E, H, L
- Indirecto : registro par (HL)
- Indexado
- Inmediato

Mediante el comando CP se resta del Acumulador el Byte direccionado, y cada uno de los Flags será influenciado según el resultado de la operación. Contrariamente al comando SUB, el resultado no es almacenado en el Acumulador, es decir, el contenido del Acumulador no es afectado por el comando. Dependiendo del estado de los Flags, puede efectuarse un salto condicional después de este comando. Observemos los posibles casos que se presentan en la comparación:

El contenido del Acumulador es mayor:
- En este caso, el Carry siempre es 0, pues el resultado no puede ser mayor que 255.

El contenido del Acumulador es igual:
- en este caso, Z=1, debido a que el resultado de la sustracción es 0. También aquí, C=0 debido a que no se realiza ningún acarreo.

El contenido del Acumulador es menor:
- En este caso, el Flag de acarreo siempre está activado, pues se realiza un acarreo negativo.

Reglas:

C=0 significa >=
Z=0 significa =
C=1 significa <

además, se obtiene:

Z=1 significa <>
C=0 y Z=1 significa >
C=1 o Z=0 significa =<

Estas reglas serán válidas sólo cuando los Bytes que han de ser comparados se consideren como números sin signo, situados entre 0 y 255.

Si ambos Bytes presentan números con signo en complemento a dos, adquieren validez unas reglas más complicadas que resultan de las reglas a las que están sujetos los Flags para la aritmética con signos. En la mayoría de los casos no es necesaria esta aplicación.
Para la decisión de la comparación de igualdad se utiliza el Flag Z. Según el estado de los Flags S y V se decidirá si es mayor o menor.

Los Flags S y V se unen lógicamente mediante la orden XOR; es decir,si se activa V (tiene lugar un desbordamiento), se complementa S, de lo contrario, S permenece en su estado anterior.

```
S XOR V =0   significa >=
S XOR V =1   significa <
```

En lo sucesivo, partiremos de la premisa de que los Bytes serán interpretados como números carentes de signo.

Ejemplo:

```
A =&H35
B =&H21

CP B

suministra a S Z V C
             0 0 0 0   debido a

 00110101  :A
- 00100001  :B (ningún (!) complemento a dos)
 --------
 00010100

No hay acarreo:          => C=0
Bit=0                    => S=0
 <>0                     => Z=0
no hay desbordamiento    => V=0
```

El Flag de acarreo es igual a 0. De ésto se deduce que el contenido del Acumulador es mayor que el del Byte comparable (contenido del registro B).

```
C =&H81

CP C    suministra a
```

```
 los Flags:S Z V C
           1 0 1 1    debido a:

  00000001  :registro A
- 10000001  :registro C
  --------
 110000000

 acarreo de 7 a 8        => C=1
 Bit 7=1                 => S=1
      <>                 => Z=0
 acarreo de 7 a 8 y ningún
 acarreo de 6 a 7  => V=1
```

Consecuentemente, C=1. De ahí se deduce que el valor con el que se ha comparado (el contenido del registro C), era mayor que el contenido del Acumulador.

Más adelante utilizaremos a menudo los comandos CP en relación con los comandos para comparaciones y saltos.
Los comandos lógicos se encuentran en la lista de los comandos aritméticos de 8-Bits (capítulo 4.6).

Ejercicio:

Mediante los comandos lógicos nos es posible convertir las letras mayúsculas en letras minúsculas (y viceversa). Observe para ello los códigos ASCII de las letras.

```
A:&H41=&B01000001          Z:&H5A=&B01011010
a:&H61=&B01100001          z:&H7A=&B01111010
```

En las letras minúsculas, el Bit 5 siempre es igual a 1, en las mayúsculas, siempre es igual a 0.
Escriba un programa que convierta en minúsculas todas las mayúsculas que aparecen en >SCREEN 0< (espacio de dirección de pantalla: &H0 - &H3C0). Para escribir en el RAM de video utilice la rutina del capítulo anterior (&H7CD). La dirección de la rutina que lee el RAM de video es &H7D7. Esta rutina sustituye el comando LD A,(HL) cuando HL presenta una dirección de la RAM de video.

Solución:

```
F000 210000     10   LD    HL,0
F003 CDD770     20   CALL  &H7D7
F006 FE5B       30   CP    &H5B
F008 30FE       40   JR    NC,&HF013
F00A FE41       50   CP    &H41
F00C 38FE       60   JR    C,&HF013
F00E F620       70   OR    &B00100000
F010 CDCD70     80   CALL  &H7CD
F013 23         90   INC   HL
F014 7C        100   LD    A,H
F015 FE04      110   CP    4
F017 20EA      120   JR    NZ,&HF003
F019 C9        130   RET
```

Explicación:

10: Indicador de dirección HL en 0
20: cargar en el acumulador el contenido de la RAM de video de la dirección HL.
30: ¿Es A >= ASC("Z")+1?
40: Si afirmativo, no modificar
50: ¿Es A < ASC("A")?
60: Si afirmativo, no modificar
70: activar Bit 5
80: escribir en la RAM de video el valor nuevo
90: aumentar la dirección
100: cargar en el Acumulador el Byte alto para la comparación
110: Es la dirección del Byte alto >=4?
120: si negativo, comenzar de nuevo
130: Si afirmativo, FIN.

Si las minúsculas han de ser convertidas en mayúsculas, se habrán de sustituir los valores de los comandos CP por &H7B (en la línea 30) y &H61 (en la línea 50), y luego, el comando OR &B00100000 por el comando AND &B11011111

El sistema operativo, p.ej., también efectúa la conversión de minúsculas a mayúsculas: Mediante la introducción de programas en BASIC, usted puede entrar letras mayúsculas y minúsculas; sin embargo, en el tratamiento interno, muchas entradas son transformadas a letras mayúsculas.

4.9 EL ENSAMBLADOR

Para que ya no tengamos que traducir a mano los programas en lenguaje máquina, hemos escrito un Ensamblador Z80.
El Ensamblador crea los códigos máquina (código objeto, o bien, programa objeto) correspondientes al programa escrito en lenguaje ensamblador (Programa fuente). Así, p.ej., se realizan automáticamente los cálculos de las distancias de los saltos. De este modo, nos ahorramos el trabajo pesado de la traducción a mano, de la consulta del código de operaciones, etc.
Para los programas Ensamblador Z80 rigen determinadas convenciones.

Una línea en ensamblador tiene el siguiente aspecto:

Etiqueta Orden Operando ; Comentario

Puesto que queremos utilizar el editor del Basic para la introducción de programas, cada instrucción en ensamblador tendrá asignado un número de línea.

A continuación definiremos el formato de entrada del ensamblador. Para evitar errores en la utilización del ensamblador son de gran importancia los párrafos que vienen a continuación. Por favor, estúdielos a fondo.

Label:

Puede haber una etiqueta (label) al inicio de una línea. Una etiqueta es una variable. La longitud del nombre de la variable (nombre del Label) no puede abarcar más de 6 caracteres. Los nombres de la etiquqta deben comenzar con una letra. Los comandos ensamblador no pueden utilizarse como nombres de etiqueta. Mediante el uso de etiquetas se simplifica la programación de saltos:

```
          .
          .
ANF Comando       : ANF es una etiqueta
          .
          .
        JR ANF    : Salto a ANF
          .
          .
```

El Ensamblador calcula automáticamente la distancia correcta.

Comando (Mnemonico):

El comando deberá seguir a la etiqueta en el caso de que exista. La etiqueta y el comando deberán estar separados uno del otro por espacios en blanco. El mnemonico deberá ser una palabara de comando válida en lenguaje ensamblador. En las listas de comandos utilizamos continuamente palabras de comando válidas, p.ej.: LD, ADD, INC, etc.

Operando:

El operando viene a continuación de la palabra de comando, separado de nuevo por un espacio en blanco. En los saltos, la dirección de salto puede indicarse como etiqueta (ver arriba). Naturalmente, la existencia de esta etiqueta es importante.
En lugar de constantes o distancias, pueden colocarse nombres de variables o etiquetas.
¡Dentro del Operando jamás pueden haber espacios en blanco!

Comentario:

Al final de la línea puede haber un comentario, separado del resto por un espacio en blanco o por un punto y coma. Todos los caracteres a continuación de un punto y coma no son tenidos en cuenta en la traducción. El comentario es una ayuda útil para la posterior comprensión de los programas.

Durante la traducción, el ensamblador crea un listado ensamblador, que puede ser editado en la impresora o en el monitor. Además, el código así creado puede ser memorizado en casette o diskette.
El listado ensamblador así creado presenta la siguiente estructura:

```
&HDir. &HCodig Num.Lin. Label Comando Operando; Comentario
                BASIC

A003  36CC      50        SEGUIR LD (HL),Bitmat ; DIRECCION..
A005  23        60               INC HL         ; Incremen. HL
```

A parte de los comandos Z80, el ensamblador también conoce una serie de pseudocomandos. Estas son instrucciones dadas al ensamblador, como p.ej. END. Para el Ensamblador, END significa: no seguir buscando otros comandos y concluir la traducción.
Otra instrucción importante es EQU (ingl. equal:igual). Con EQU se define el valor de una Variable:

```
Nombre de la Variable EQU valor
```

(asigna al nombre de la variable precedente el valor situado detrás de la instrucción EQU.)
La instrucción ORG (organización) indica en qué dirección ha de ser situado el programa. Generalmente, utilizaremos &HF000 como dirección de inicio.
Se ha llegado a los siguientes acuerdos para la indicación de números.
Los números hexadecimales se reconocen por llevar delante "&H".
Los números binarios se reconocen por llevar delante "&B".
Los números octales se reconocen por llevar delante "&O".
Cualquier número que no vaya precedido por uno de estos signos será interpretado como decimal.

Los convenios estándard para el Ensamblador Z80 son: H al final de número hexadecimal B al final de un número Binario y O al final de un número Octal. Sin embargo, nosotros utilizaremos la convención mencionada arriba con &H y &B.

Pruebe usted mismo el Ensamblador introduciendo un programa pequeño.

Independientemente del programa Ensamblador, se puede introducir el programa fuente ensamblador (Programa de partida). El primer programa del capítulo 1.2 presentaría la siguiente estructura:

```
10 ' ORG &HF000
20 ' LD HL,0 ; Start memoria de pantalla
30 ' SEGUIR LD A,&H2A ; = 42 = ASCII de "*"
40 ' CALL &H07CD ; corresponde al comando VPOKE
50 ' INC HL ; aumentar dirección
60 ' LD A,4
70 ' CP H ;¡ H > 4 ?
80 ' JR NZ, SEGUIR ; no, entonces, de nuevo
90 ' RET ; retorno al BASIC
100 ' END
```

Para la introducción, usted puede utilizar tanto minúsculas como mayúsculas. Internamente, la forma de escritura no se tiene en cuenta, se convierte a mayúsculas directamente. Observe que detrás de cada número de línea, separado por un espacio en blanco, ha de indicarse un "'". Si usted se olvida de este signo, la línea en cuestión no puede ser traducida después por el lenguaje ensamblador, y aparece el mensaje de error:

''Falta el ' en la línea ...''

Mediante la línea 10, el espacio de memoria del programa se determina a partir de &HF000.

En el bucle del programa hemos utilizado la etiqueta "SEGUIR" como dirección del salto. Por lo demás, utilizamos los comandos normales en ensamblador.
Si una línea viene seguida de un comentario, éste estará separado por un punto y coma. Es importante que haya un espacio en blanco delante del punto y coma. Para el ensamblador, los espacios en blanco significan que p.ej. finaliza una etiqueta y que a continuación le sigue el comando. Por este motivo, entre etiqueta, comando, operando y comentario, siempre deben haber (!!) espacios en blanco, y p.ej., dentro de un operando, nunca pueden utilizarse (!!) espacios en blanco.
Ejemplo:

```
LD( HL )        ¡¡¡ERROR!!!

LD (HL)         ¡¡¡CORRECTO!!!
```

Al final del programa deberá darse el pseudocomando END. Este comando significa para el ensamblador que la traducción puede finalizar en este punto. El ensamblador cubre a partir de la línea 10000 y la línea 1. Como consecuencia, estas líneas no pueden ser utilizadas por el programa fuente.
Grabe el programa fuente introducido como fichero ASCII, o sea, mediante >SAVE"CAS:nombre"<. Como la carga de ficheros ASCII dura más tiempo que la de los programas almacenados con el método normal, cargue primero el Ensamblador. A continuación, cargue el programa fuente con >MERGE"CAS:nombre"<.

Luego, inicie el programa simplemente con >RUN<.
El Ensamblador pregunta cual es el nombre del programa, si se desea un listado ensamblador de la traducción, y si se ha de imprimir este Listado. Usted puede elegir las respuestas mediante >RETURN<. Escoja primero las entradas estándar.

Ahora comienza realmente la traducción:
En pantalla aparece el listado ensamblador que usted ya conoce. En caso de error, aparecen delante de cada línea las correspondientes advertencias. Al final del listado se indican, en caso de que existan, las etiquetas y variables no definidas. A continuación se indica el nombre del

programa, dirección de inicio, dirección final, la longitud del programa y el número de errores. Los errores pueden ser corregidos en la correspondiente línea BASIC.
Al final del listado aparece una tabla que comprende, siguiendo el orden de aparición, todas las etiquetas y variables con sus respectivos valores. Finalmente, el Ensamblador pregunta si ha de ser grabado el código de la máquina.
Si se introduce "s", se grabará el código creado como fichero binario con el nombre antes introducido. Tras el ensamblaje, el programa en lenguaje máquina se hallará en la memoria en la posición dada, y podrá ser llamado utilizando la tecla de función 1. >DEFUSR> ha asignado con anterioridad la dirección de inicio al comando >USR1<. La tecla de función 1 es además cubierta con "X=USR1(1)"+RETURN.

Ejemplo: listado ensamblador completo

```
F000            10          ORG  &HF000
F000 210000     20          LD   HL,0 ; inicio
memoria de pantalla
F003 3E2A       30   SEGUIR  LD   A,&H2A ; =42=ASCII de "*"
F005 CDCD07     40          CALL &H07CD ; corresponde al
comando VPOKE
F008 23         50          INC  HL   ; aumentar Dirección
F009 3E04       60          LD   A,4
F00B BC         70          CP     H ; H > 4 ?
F00C 20F5       80          JR   NZ,SEGUIR ; no entonces,
de nuevo
F00E C9         90          RET ; retorno al BASIC
------------------------------------------------------------
Programa :DEMO
Inicio: &HF000    End: &HF00E
Longitud : &HF Bytes
Errores : 0
Tabla de Variables::
SEGUIR  F003
¿GRABAR (s/n)?
```

Al copiar el listado, intente comprender la estructura básica del ensamblador, partiendo de las explicaciones que se dan a continuación del listado.

Para facilitarle la introducción de programas fuente más largos, hemos escrito un programa que modifica tanto el comando >AUTO<, que además del número de línea, también proyecta el "'".

```
10 REM AUTO con proyección de '
20 CLEAR 200,&HEFFF
30 POKE &HFF0D,&H20
40 POKE &HFF0E,&HF3
50 DEFINT I
60 DI=&HF320-&H4137
70 FOR I=&H4137 to &H415E:POKE I+DI,PEEK(I):NEXT
80 POKE &H4148+DI,&H16+6
90 I=I+DI
100  READ A$:IF A$<>"#" THEN  POKE I,VAL("&H"+A$):I=I+1:GOTO
100
DATA 3E,27,DF,3E,20,DF,E1,C3,5F,41
120 DATA ##
130 REM activación con
140 REM POKE &HFF0C,&HC3
150 REM desactivación de REM con POKE &HFF0C,&HC9
```

Una vez cargar con >RUN< la ampliación, pruebe la activación: POKE &HFF0C,&HC3. Si desea utilizar de nuevo el comando >AUTO<, introduzca >POKE &HFF0C,&HC9<.Esta ampliación es colocada partir de &HF320 en adelante. Con ello, este espacio ya no queda libre para programas propios en lenguaje máquina.

```
1 CLEAR 200,&HEFFF:MAXFILES=0:GOTO 10000
10000 ' *** Z80 Ensamblador de Holger Dullin 1985 ***
10010 GOSUB 13720
10020 LOCATE 4,4:PRINT"Z 8 0 - E n s a m b l a d o r"
10030 LOCATE 3,8:INPUT"Nombre del programa";NA$
10040 LOCATE 19,11:PRINT"s";
10050 LOCATE 3,11:INPUT"Listado (s/n) ";A$
10060 IF (ASC(A$)OR32)=110 THEN LF=0:GOTO 10100 ELSE LF=-1
10070 LOCATE 19,13:PRINT"n";
10080 LOCATE 3,13:INPUT"Impresora (s/n) ";A$
10090 IF (ASC(A$)OR32)=106 THEN POKE &HFEE4,&HC3
10100 CLS:REM Inicio Ensamblaje ****
10110 NZ=FNDK(BP):BP=BP+2
10120 ZN=FNDK(BP)
10130 IF ZN>9999 THEN PRINT"falta END":GOTO 10590
10140 BP=BP+2
10150 IF (FNDK(BP)<>&H8F3A) OR (PEEK(BP+2)<>&HE6) THEN PRIN
T"El ' falta en LÍnea";ZN:BP=NZ:FZ=FZ+1:GOTO 10110
10160 BP=BP+3
10170 POKE VP,NZ-BP-1
10180 POKE VP+1,BP-INT(BP/256)*256
10190 POKE VP+2,INT(BP/256)-256*(BP<0)
10200 ZA$=Z$:BP=NZ
10210 FOR I=0 TO 3:A$(I)="":NEXT
10220 PO=INSTR(Z$,";")
10230 IF PO=0 THEN BE$="":GOTO 10260
10240 BE$=RIGHT$(Z$,LEN(Z$)-PO+1)
10250 POKE VP,PO-1
10260 J=0:Z$=USR9(Z$)
10270 IF LEFT$(Z$,1)=" " THEN Z$=RIGHT$(Z$,LEN(Z$)-1):GOTO
10270
10280 PO=INSTR(Z$," ")
10290 IF Z$="" THEN J=J-1:GOTO 10360
10300 IF PO=0 THEN 10350
10310 A$(J)=(LEFT$(Z$,PO-1)):Z$=RIGHT$(Z$,LEN(Z$)-PO)
10320 IF Z$="" THEN J=J-1:GOTO 10360
```

```
10330 IF J>2 THEN ERROR 60
10340 J=J+1:GOTO 10270
10350 A$(J)=(Z$)
10360 IF J>2 THEN ERROR 60
10370 GOSUB 10790
10380 REM Salida ********************
10390 IF FI THEN 10520
10400 IF NOT LF THEN LOCATE 5,3:PRINTZN:GOTO 10480 ELSE IF
POS(0)=0 THEN PRINT" "; ELSE
10410 PRINTRIGHT$("000"+HEX$(MP),4);" ";
10420 IF LP=0 THEN 10450
10430 FOR I=1 TO LP:PRINTRIGHT$("0"+HEX$(PW(I)-256*(PW(I)<0
)),2);
10440 POKE MP+I-1,PW(I)-256*(PW(I)<0):NEXT I
10450 PRINTTAB(15);USING"####";ZN;
10460 PRINTTAB(21);LB$;TAB(28);BF$;TAB(33);OA$;:IF BE$<>""
THEN PRINT" ";
10470 PRINTBE$
10480 MP=MP+LP+DS:IF MP+4>&HF31F THEN PRINT:PRINT"Memoria l
lena":PRINT:GOTO 13090
10490 LP=0:DS=0
10500 LB$="":LA$="":BF$="":OA$="":OP$="":BE$=""
10510 GOTO 10110
10520 FOR J=LP TO 1 STEP -1
10530 PW(J+1)=PW(J):NEXT
10540 PW($)=WI:LP=LP+1
10550 IF NOT DF THEN 10580
10560 IF LP=3 THEN PW(4)=PW(3)
10570 PW(3)=DW:LP=LP+1
10580 FI=0:DF=0:GOTO 10400
10590 REM Fin Programm **************
10600 IF NOT LF THEN LOCATE 0,9
10610 PRINTSTRING$(36,"_"):IF UP=0 THEN 10650
10620 FOR I=0 TO UP-1
10630 PRINT"Label indefinido ";UL$(I);" en";UD(I,0);" / Dir
ecciõn &H";RIGHT$("000"+HEX$(UD(I,1)),4)
```

```
10640 FZ=FZ+1:NEXT
10650 PRINT:PRINT"Programa :";NA$
10660 PRINT"Inicio : &H";RIGHT$("000"+HEX$(MS),4);"    Fin
: &H";RIGHT$("000"+HEX$(MP-1),4)
10670 PRINT"Longitud: &H";HEX$(MP-MS);" Bytes"
10680 PRINT"Errores :";FZ
10690 IF LI=0 THEN 10730
10700 PRINT"Tabla de Variables :"
10710 FOR I=0 TO LT-1
10720 PRINTLEFT$(LT$(I)+"          ",7);RIGHT$("0000"+HEX$(WL
(I)),4),:NEXT
10730 PRINT:POKE &HFEE4,&HC9
10740 LOCATE 20,CSRLIN:PRINT"n";
10750 LOCATE 0,CSRLIN:INPUT"Grabación(s/n)";A$
10760 IF (ASC(A$)OR32)<>106 THEN 10780
10770 BSAVE "CAS:"+NA$,MS,MP
10780 DEFUSR1=MS:POKE &HFEE4,&HC9:KEY 1,"X=USR1(1)"+CHR$(13
):KEY ON:END
10790 J=0:REM Interpretación *********
10800 BF$=LEFT$(A$(J)+"     ",4)
10810 PO=INSTR(T1$,BF$)
10820 IF PO<>0 THEN LP=0:GOTO 11390
10830 PO=INSTR(T2$,BF$)
10840 IF PO<>0 THEN LP=1:GOTO 11080
10850 PO=INSTR(T3$,BF$)
10860 IF PO<>0 THEN LP=2:PW(1)=&HED:GOTO 11110
10870 PO=INSTR(T4$,BF$)
10880 IF PO<>0 THEN 11140
10890 IF J>0 THEN ERROR 60
10900 IF A$(0)="" THEN RETURN
10910 A$=A$(0):GOSUB 13210
10920 IF NO THEN ERROR 60
10930 LB$=LA$:WE=MP
10940 LT$(LT)=LA$:WL(LT)=MP:LT=LT+1
10950 FOR I=0 TO UP:IF LA$=UL$(I) THEN 10980 ELSE 10960
10960 NEXT
```

```
10970 J=J+1:GOTO 10800
10980 ON UD(I,2) GOTO 10990,11010
10990 AD=UD(I,1)-1:ZI=WE:GOSUB 13660
11000 PW(1)=OS:GOTO 11030
11010 PW(2)=INT(WE/256)
11020 PW(1)=WE-PW(2)*256
11030 IF LF THEN PRINT"**** LInea ";UD(I,0);" : ";UL$(I);"=
&H";HEX$(WE);:IF UD(I,2)=1 THEN PRINT" Offset ";HEX$(OS) EL
SE PRINT
11040 FOR K=1 TO UD(I,2):POKE UD(I,1)+K-1,PW(K)-256*(PW(K)<
0):NEXT
11050 FOR K=I TO UP-1:UL$(K)=UL$(K+1)
11060 FOR C=0 TO 2:UD(K,C)=UD(K+1,C):NEXT C,K
11070 UP=UP-1:I=I-1:GOTO 10960
11080 REM 1- Byte sin Operando ------
11090 IF A$(J+1)<>""THEN ERROR 62
11100 PW(1)=W2((PO-1)/4):RETURN
11110 REM 2- Byte sin Operando -----
11120 IF A$(J+1)<>"" THEN ERROR 62
11130 PW(2)=W3((PO-1)/4):RETURN
11140 REM pseudo comandos -------------
11150 J=J+1:OP$=A$(J):OA$=OP$
11160 ON (PO-1)/4 GOTO 11210,11250,11270,11290,11310,11360
11170 REM EQU
11180 IF LA$="" THEN ERROR 63
11190 A$=OP$:GOSUB 13420
11200 WL(LI-1)=WE:RETURN
11210 REM ORG
11220 IF OP$="" THEN ERROR 64
11230 A$=OP$:GOSUB 13350
11240 LP=0:MP=WE:MS=MP:RETURN
11250 REM END
11260 GOTO 10590
11270 REM DB
11280 A$=OP$:GOSUB 13610:RETURN
11290 REM DW
```

```
11300 A$=OP$:GOSUB 13420:RETURN
11310 REM DM
11320 IF LEFT$(OP$,1)<>CHR$(34) OR RIGHT$(OP$,1)<>CHR$(34)
THEN ERROR 66
11330 ZW$=MID$(OP$,2,LEN(OP$)-2)
11340 LP=LEN(ZW$)
11350 FOR I=1 TO LP:PW(I)=ASC(MID$(ZW$,I,1)):NEXT:RETURN
11360 REM DS
11370 A$=OP$:GOSUB 13420
11380 DS=WE:LP=0:RETURN
11390 REM Otras utilizaciones de comandos -
11400 J=J+1:OP$=A$(J)
11410 OA$=OP$:ZW=(PO+3)/4
11420 IF OP$="" AND BF$<>"RET " THEN ERROR 64
11430 GOSUB 13090
11440 PI=INSTR(OP$,"IX")
11450 IF PI<>0 THEN WI=&HDD:IR$="IX":GOTO 11520
11460 PI=INSTR(OP$,"IY")
11470 IF PI<>0 THEN WI=&HFD:IR$="IY":GOTO 11520
11480 ON ZW GOTO 12620,11940,11920,12060,12060,12100,12220,
12240,12330,12310,12370,12420,12420,12510,12550
11490 IF ZW<24 THEN 11630
11500 IF ZW<32 THEN 11790
11510 GOTO 11850
11520 REM Comandos indexados -------
11530 FI=-1
11540 IF (NOT QF) OR (PI-PA<>1) THEN MID$(OP$,PI,2)="HL":GO
TO 11600
11550 IF LEFT$(QI$,3)<>IR$+"+" THEN IF BF$="JP  " THEN 1159
0 ELSE ERROR 60
11560 A$=RIGHT$(QI$,LEN(QI$)-3)
11570 DI$=A$:GOSUB 13610
11580 DF=-1:DW=WE
11590 OP$=LEFT$(OP$,PA)+"HL"+RIGHT$(OP$,LEN(OP$)-PZ+1)
11600 IF (INSTR(OP$,"IX")=0)AND(INSTR(OP$,"IY")=0)THEN 1162
0
```

```
11610 IF (OP$=("HL,"+IR$))AND(BF$="ADD ") THEN OP$="HL,HL"
ELSE ERROR 61
11620 GOSUB 13090:GOTO 11480
11630 REM comandos aritméticos/lógicos
11640 IF NOT KF THEN A$=O1$:GOTO 11660
11650 IF O1$<>"A" THEN 11710 ELSE A$=O2$
11660 LP=1:CO=ZW-16
11670 GOSUB 13260
11680 IF RF THEN PW(1)=128 OR (CO*8) OR RR:RETURN
11690 PW(1)=&HC6 OR (CO*8)
11700 GOSUB 13610:RETURN
11710 IF O1$<>"HL" THEN ERROR 61
11720 A$=O2$:GOSUB 13290
11730 IF NOT RF THEN ERROR 61
11740 IF BF$="ADD " THEN CO=9:LP=1:GOTO 11780
11750 PW(1)=&HED:LP=2
11760 IF BF$="ADC " THEN CO=&H4A:GOTO 11780
11770 IF BF$="SBC " THEN CO=&H42 ELSE ERROR 60
11780 PW(LP)=CO OR (DD*16):RETURN
11790 REM comandos de rotación/desplaamiento
11800 LP=2:PW(1)=&HCB
11810 IF KF THEN ERROR 61
11820 A$=OP$:GOSUB 13260
11830 IF NOT RF THEN ERROR 61
11840 PW(2)=(8*(ZW-24)) OR RR:RETURN
11850 REM Comandos de manipulación de Bits--
11860 LP=2:PW(1)=&HCB
11870 A$=O2$:GOSUB 13260
11880 IF NOT RF THEN ERROR 61
11890 BB=ASC(O1$)-48
11900 IF (0>BB) OR (7<BB) OR (LEN(O1$)<>1) THEN ERROR 61
11910 PW(2)=(64*(ZW-31))OR(BB*8)OR RR:RETURN
11920 REM Saltos relativos --------
11930 LP=1:PW(1)=&H10:A$=OP$:GOTO 12010
11940 LP=1
11950 IF NOT KF THEN CC=3:A$=OP$:GOTO 12000
```

```
11960 A$=O1$:GOSUB 13320
11970 IF (NOT CF) OR (CC>3) THEN ERROR 61
11980 CC=CC OR 4
11990 A$=O2$
12000 PW(1)=CC*8
12010 IF LEFT$(A$,1)<>"$" THEN GOSUB 13420:LP=LP-2:IF I>LT
THEN WE=MP:GOTO 12030 :ELSE 12030
12020 WE=MP+VAL(RIGHT$(A$,LEN(A$)-1)
12030 LP=LP+1:AD=MP:ZI=WE
12040 GOSUB 13660
12050 PW(2)=OS:RETURN
12060 REM Saltos------------------
12070 ZW=1:LP=1
12080 IF BF$="RET " THEN CO=0 ELSE CO=&B100
12090 GOTO 12130
12100 IF OP$="(HL)" THEN LP=1:PW(1)=&HE9:RETURN
12110 CO=2
12120 ZW=0:LP=1
12130 IF BF$="RET " THEN IF OP$="" THEN 12150 ELSE 12170 EL
SE
12140 IF KF THEN 12170
12150 PW(1)=&HCO OR CO OR 1 OR (ZW*8)
12160 A$=OP$:GOTO 12200
12170 A$=O1$:GOSUB 13320
12180 IF NOT CF THEN ERROR 61
12190 PW(1)=&HCO OR CO OR (CC*8):A$=O2$
12200 IF BF$="RET " THEN RETURN
12210 GOSUB 13420:RETURN
12220 REM Comandos de contador---------
12230 ZW=0:GOTO 12250
12240 ZW=1
12250 IF KF THEN ERROR 61
12260 LP=1:A$=OP$:GOSUB 13260
12270 IF RF THEN PW(1)=4 OR (RR*8) OR ZW:RETURN
12280 GOSUB 13290
12290 IF NOT RF THEN ERROR 61
```

```
12300 PW(1)=3 OR (DD*16) OR (ZWI*8):RETURN
12310 REM Comandos de la pila----------
12320 CO=&HC1:GOTO 12340
12330 CO=&HC5
12340 A$=OP$:DR$(3)="AF":GOSUB 13290:DR$(3)="SP"
12350 IF NOT RF THEN ERROR 61
12360 LP=1:PW(1)=CO OR (DD*16):RETURN
12370 REM Restart ------------------
12380 A$=OP$:GOSUB 13610
12390 ZW=WO/8
12400 IF ZW<>INT(ZW) OR ZW>7 THEN ERROR 65
12410 LP=1:PW(1)=&HC7 OR (ZW*8):RETURN
12420 REM Comandos I/O --------------
12430 IF NOT (KF AND QF) THEN ERROR 61
12440 IF BF$="IN  " THEN ZW=0 ELSE ZW=1:SWAP O2$,O1$
12450 IF QI$<>"C" THEN 12490
12460 A$=O1$:GOSUB 13260
12470 IF (NOT RF) OR (QI$<>"C") THEN ERROR 61
12480 LP=2:PW(1)=&HED:PW(2)=64 OR (RR*8) OR ZW:RETURN
12490 LP=1:A$=QI$:GOSUB 13610
12500 PW(1)=&HDB XOR (ZW*8):RETURN
12510 REM Modos de interrupción------
12520 IF OP$<>"0" AND OP$<>"1" AND OP$<>"2" THEN ERROR 65
12530 LP=2:PW(1)=&HED
12540 PW(2)=&H46 OR ((VAL(OP$)-(OP$<>"0"))*8):RETURN
12550 REM Comandos de intercambio---
12560 LP=1
12570 IF OP$="(SP),HL" THEN PW(1)=&HE3:RETURN
12580 IF OP$="DE,HL" THEN PW(1)=&HEB:GOTO 12610
12590 IF OP$="AF,AF'" THEN PW(1)=&H8:RETURN
12600 ERROR 61
12610 IF FI THEN ERROR 61 ELSE RETURN
12620 REM Comandos de carga---------
12630 IF NOT KF THEN ERROR 61
12640 A$=O1$:GOSUB 13260
12650 IF RF THEN 12850
```

```
12660 GOSUB 13290
12670 IF RF THEN 12750
12680 A$=O2$:GOSUB 13290
12690 IF RF THEN 12730
12700 SWAP O1$,O2$
12710 A=0:GOSUB 12930
12720 IF NF THEN ERROR 61 ELSE RETURN
12730 IF NOT QF THEN ERROR 61
12740 ZW$=O2$:ZF=1:GOTO 12790
12750 IF OP$="SP,HL" THEN LP=1:PW(1)=&HF9:RETURN
12760 IF QF THEN ZW$=O1$:ZF=0:GOTO 12790
12770 A$=O2$
12780 LP=1:CO=1:GOTO 12820
12790 A$=QI$
12800 IF ZW$="HL" THEN LP=1:CO=&HA:GOTO 12820
12810 LP=2:PW(1)=&HED:CO=&H4B
12820 CO=CO AND NOT (ZF*8)
12830 PW(LP)=CO OR (DD*16)
12840 GOSUB 13420:RETURN
12850 ZZ=RR:A$=O2$:GOSUB 13260
12860 IF NOT RF THEN 12890
12870 LP=1:PW(1)=64 OR (ZZ*8) OR RR
12880 IF PW(1)=&H76 THEN ERROR 61 ELSE RETURN
12890 A=1:GOSUB 12930
12900 IF NOT NF THEN RETURN
12910 LP=1:PW(1)=6 OR (RR*8)
12920 A$=O2$:GOSUB 13610:RETURN
12930 REM Carga especial 8-Bits-----
12940 NF=0
12950 IF O1$<>"A" THEN NF=-1:RETURN
12960 IF QF THEN 13020
12970 IF O2$="I" THEN ZW=0:GOTO 13000
12980 IF O2$="R" THEN ZW=1:GOTO 13000
12990 NF=-1:RETURN
13000 CO=&H47:LP=2:PW(1)=&HED
13010 PP=(A*2) OR ZW:GOTO 13070
```

```
13020 IF QI$="BC" THEN ZW=0:GOTO 13060
13030 IF QI$="DE" THEN ZW=1:GOTO 13060
13040 LP=1:PW(1)=&H32 OR (A*8)
13050 A$=QI$:GOSUB 13420:RETURN
13060 CO=2:LP=1:PP=(ZW*2) OR A
13070 PW(LP)=CO OR (8*PP):RETURN
13080 REM Subprogramas **************
13090 REM Descomponer operando------
13100 PO=INSTR(OP$,",")
13110 IF PO=0 THEN O1$=OP$:KF=0:GOTO 13140
13120 KF=-1
13130 O1$=LEFT$(OP$,PO-1):O2$=RIGHT$(OP$,LEN(OP$)-PO)
13140 PA=INSTR(OP$,"("):PZ=INSTR(OP$,")")
13150 IF PA=0 THEN QF=0:QI$="":GOTO 13190
13160 IF PA>PZ THEN ERROR 61
13170 QF=-1
13180 QI$=MID$(OP$,PA+1,PZ-PA-1)
13190 RETURN
13200 REM Comprobación etiqueta----
13210 LC=ASC(A$)
13220 IF LC<65 OR LC>90 THEN NO=-1:RETURN
13230 IF LEN(A$)>6 THEN PRINT"Label demasiado largo":A$=LEF
T$(A$,6)
13240 LA$=A$:NO=0
13250 RETURN
13260 REM Comprobación registro-----
13270 FOR I=0 TO 7:IF RG$(I)=A$ THEN RF=-1:RR=I:RETURN ELSE
 NEXT
13280 RF=0:RETURN
13290 REM Comprobación registro doble-----
13300 FOR I=0 TO 3:IF DR$(I)=A$ THEN RF=-1:DD=I:RETURN ELSE
 NEXT
13310 RF=0:RETURN
13320 REM Comprobación condición----
13330 FOR I=0 TO 7:IF CD$(I)=A$ THEN CF=-1:CC=I:RETURN ELSE
 NEXT
```

```
13340 CF=0:RETURN
13350 REM Comprobación números------
13360 WE=VAL(A$)
13370 LC=ASC(A$)
13380 IF WE=0 AND LC<>30 AND (LC>57 OR LC<48) THEN ERROR 66
13390 IF WE>=0 THEN RETURN
13400 IF LEFT$(A$,2)<>"&H" THEN ERROR 65
13410 RETURN
13420 REM Evaluación de la expresión
13430 GOSUB 13200
13440 IF NO THEN GOSUB 13350:GOTO 13480
13450 FOR I=0 TO LT:IF LT$(I)<>LA$ THEN NEXT
13460 IF I>LT THEN 13540
13470 WE=WL(I)
13480 WH=INT(WE/256)
13490 WO=WE-WH*256
13500 LP=LP+2
13510 PW(LP-1)=WO
13520 PW(LP)=WH
13530 RETURN
13540 IF LF THEN PRINT"?";
13550 UL$(UP)=A$:UD(UP,0)=ZN
13560 UD(UP,1)=MP+LP-FI-DF
13570 UD(UP,2)=2+(BF$="DJNZ" OR BF$="JR  ")
13580 UP=UP+1
13590 WE=0
13600 GOTO 13480
13610 REM Evalación de la expresión < 256 --
13620 GOSUB 13420
13630 LP=LP-1
13640 IF WH<>0 THEN PRINTWH:ERROR 65
13650 RETURN
13660 REM calcular Offset ----------
13670 OS=ZI-AD
13680 OS=OS-2
```

```
13690 IF OS>129 OR OS<-126 THEN ERROR 65
13700 IF OS<0 THEN OS=256+OS
13710 RETURN
13720 REM ***** Inicialización ******
13730 DEFINT A-Z:Z$=";":PO=0:J=0
13740 VP=VARPTR(Z$)
13750 DEF FNDK(I)=VAL("&H"+HEX$(PEEK(I)+256*PEEK(I+1)))
13760 SCREEN 0:WIDTH 39:COLOR 1,14,14:KEY OFF
13770 T1$="LD  JR  DJNZCALLRET JP  INC DEC PUSHPOP RST IN
OUT IM  EX  ADD ADC SUB SBC AND XOR OR  CP  RLC RRC RL  RR
 SLA SRA *** SRL BIT RES SET "
13780 T3$="CPD CPDRCPI CPIRIND INDRINI INIRLDD LDDRLDI LDIR
NEG OTDROTIROUTDOUTIRETIRETINRLD RRD "
13790 DATA A9,B9,A1,B1,AA,BA,Ad,B2,A8,BB,A0,B0,44,BB,B3,AB,
A3,4D,45,6F,67
13800 T2$="CCF CPL DAA DI  EI  EXX HALTNOP RLA RLCARRA RRCA
SCF "
13810 DATA 3F,2F,27,F3,FB,D9,76,00,17,07,1F,0F,37
13820 T4$="EQU ORG END DB  DW  DM  DS  "
13830 DIM A$(3),LT$(50),WL(50),UL$(50),UD(50,2)
13840 DIM W2(12),W3(20)
13850 FOR I=0 TO 20:READ A$:W3(I)=VAL("&H"+A$):NEXT
13860 FOR I=0 TO 12:READ A$:W2(I)=VAL("&H"+A$):NEXT
13870 BP=FNDK(FNDK(&HF676)):MP=&HF000:MS=MP
13880 DIM RG$(7),CD$(7),DR$(3)
13890 FOR I=0 TO 7:READ RG$(I):NEXT
13900 FOR I=0 TO 7:READ CD$(I):NEXT
13910 FOR I=0 TO 3:READ DR$(I):NEXT
13920 DATA B,C,D,E,H,L,(HL),A
13930 DATA NZ,Z,NC,C,PO,PE,P,M
13940 DATA BC,DE,HL,SP
13950 ON STOP GOSUB 10780:STOP ON
13960 ON ERROR GOTO 14080
13970 FOR I=&HF374 TO &HF37F:READ A$:POKE I,VAL("&H"+A$):NE
XT
13980 DATA F5,3A,61,F6,32,15,F4,F1
```

```
13990 DATA CD,63,1B,C9
14000 POKE &HFEE5,&H74:POKE &HFEE6,&HF3
14010 FOR I=&HF351 TO &HF373:READ A$:POKE I,VAL("&H"+A$):NE
XT:DEFUSR9=&HF351
14020 DATA FE,03,C2,6D,40,1A,FE,00
14030 DATA C8,47,62,6B,23,7E,23,66
14040 DATA 6F,7E,FE,7B,30,06,FE,61
14050 DATA 38,02,E6,DF,77,23,10,F1
14060 DATA 3E,3,C9
14070 RETURN
14080 REM Traamiento erores**********
14090 IF ERR<60 THEN POKE &HFEE4,&HC9:KEY ON:ON ERROR GOTO
0
14100 BEEP:FZ=FZ+1
14110 IF NOT LF THEN LOCATE 5,3:PRINTZN:PRINT
14120 IF ERR=60 THEN PRINT"Syntax Error";
14130 IF ERR=61 THEN PRINT"Syntax Error en el operando";
14140 IF ERR=62 THEN PRINT"Operando supérfluo";
14150 IF ERR=63 THEN PRINT"Falta el Label";
14160 IF ERR=64 THEN PRINT"Falta el operando";
14170 IF ERR=65 THEN PRINT"Valor inadmisible";
14180 IF ERR=66 THEN PRINT"Carácter inadmisible";
14190 PRINT" von";ERL;
14200 PRINT TAB(30);ZN,ZA$
14210 RESUME 10490
```

Descripción del programa:

Línea 1:

La posición de memoria RAM de &HF000-&HF31F se reserva para el programa en lenguaje máquina, luego, se salta el programa fuente existente en las líneas 2 hasta 9999.

Línea 10010:

Bifurcación a la parte del programa ''Inicialización'', es decir, estructura de la tabla de comandos y similares (ver línea 13720 y siguientes.).

Línea 10020-10090

Menu: son determinados el LF (List Flag (V)) y los periféricos.

Línea 10100-10160:

El BP indica la dirección actual en el programa fuente BASIC (BP:BASIC Programm Pionter). Al inicio de una línea se indica la longitud de las mismas como Byte bajo y alto FNDK(BP) lee el valor de 16-Bits en la dirección BP y BP+1. El valor corresponde a la dirección inicial de la siguiente línea NZ (V). BP es incrementando a 2 y es leído el número de línea ZN (V). Si es mayor que 9999, se finaliza la traducción. En la línea 10150 se comprueba si el signo "'" se halla al inicio de la línea. Si no está ahí, aparece un mensaje de error y el programa pasa a tratar la línea siguiente.

Línea 10170-10190:

Con esta parte del programa se rellena Z$ (V) con la línea actual. Para mantener bastante alta la velocidad del ensamblador mediante un cambio de los indicadores de la cadena Z$ (V) en la tabla interna de variables interna del ensamblador.

Línea 10200-10360:

En primer lugar, es almacenado en BE$ (V) el comentario, si existe, luego, la línea que queda es cortada cada vez que aparece un espacio en blanco, y cada uno de los segmentos es almacenados en A$(J) (V). Si la línea tiene más de tres segmentos (Etiqueta, orden, operando), es decir, J>2, aparece un "Syntax error".

Línea 10370:

Aquí se efectúa el salto al Subprograma, el cual lleva a cabo la interpretación.

Línea 10380-10580:

Salida: Si el Flag FI (V) (Flag para comandos indexados) es activado, se salta primero a la rutina especial a partir de la línea 10520. De lo contrario, se proyecta la línea ensamblador completa.
Antes de saltar de nuevo al comienzo para traducir la línea siguiente, son desactivadas las variables importantes, y el MP (V) (contador del programa en lenguaje máquina es incrementado en el valor de la longitud del comando LP (V).

Línea 10590-10780:

Al final del programa se indican las etiquetas definidas, el nombre del programa, la dirección de inicio y la de llegada, la longitud, el número de errores y la tabla de variables.
A partir de la línea 10770 se graba el código objeto.

Línea 10790-10860:

Aquí se comprueba si en A$(J) (V) hay un comanando válido. Si el comando es efectivamente válido, se bifurca a la parte del programa donde estos comandos son traducidos.

Línea 10870-10880:

Si no se ha detectado ningún comando, se comprueba, si se trata de un pseudocomando.

Línea 10890-11070:

Si se trata de una etiqueta, ésta es registrada en la tabla de etiquetas, y se le asigna como valor el MP (V) (contador del programa en lenguaje máquina) (línea 10910-10940). En las líneas sucesivas hasta 11070, se comprueba se esta etiqueta fue utilizada anteriormente, siendo aún en aquel momemto una etiqueta indefinido. Si es así, se guarda el valor correspondiente y la etiqueta es anulada de la tabla de las etiquetas indefinidas UL$(I) (V). Si se trata de una etiqueta no válida, (es decir, no comienza por una letra), aparece el mensaje de error "Syntax error" (línea 10920).

Línea 11110-11130:

Aquí son tratados los comandos de 2-Bytes sin operando. El primer código de operaciones siempre es &HED (PW(1)=&HED). El segundo Byte del código de operaciones resulta de la posición del comando en el T3$ (V) y W3(I) (V).

Línea 11140-11380:

Aquí son traducidos todos los pseudocomandos.

Línea 11390-13070:

Si el comando no se encuentra en ninguno de los grupos mencionados anteriormente es utilizado en esta parte del programa.

Línea 11430:

Aquí se descompone el operando.

Línea 11440-11470:

Si se trata de un comando de direccionamiento indexado, se efectúa una bifurcación a la línea 11520.

Línea 11480-11510:

Aquí se salta a la correspondiente rutina de tratamiento de comandos según la posición en TI$ (V).

Línea 11520-11620:

En los comandos indexados, IX, IY, o bien, IX+d y IY+d son sustituidos por HL, el Flag indexado FI (V) y el Flag de desplazamiento DI (V) son activados respectivamente. A continuación, se lleva a cabo la evaluación normal de los comandos a partir de la línea 11480. Una vez realizada la interpretación, la modificación se efectúa de nuevo en sentido inverso y se proyecta el código del comando indexado (análogo al de HL).

Línea 11630-11780:

Aquí son interpretados los comandos aritméticos (de 8 y 16 Bits).

Línea 11790-11840:

Comandos de rotación y de desplazamiento

Línea 11850-11910:

Comandos de manipulación de Bits

Línea 11920-12050:

Saltos relativos (JR y DJNZ)

Línea 12060-12210:

Otros saltos (JP,RET,CALL)

Línea 12220-12300:

Comandos de contador (INC,DEC)

Línea 12310-12620:

(ver líneas REM)

Línea 12630-13070:

Comandos de carga

Es imposible explicar cada una de las rutinas, pues ello abarcaría demasiada extensión, pero cojamos como ejemplo la rutina para los comandos de manipulación de Bits:

Línea 11860:

El LP (V) (longitud del comando, es decir, cantidad de valores que han de ser movidos mediante POKE) es 2. El primer valor PW(1) (V) es &HCB. Esto es válido para todos los comandos de Bits.

Línea 11870:

02$ (V) (la parte del operando situada detrás de la coma) es almacenado, para su entrega al subprograma a partir de la línea 13260, en A$ (V). El subprograma determina si se trata de uno de los registros A,B,C,D E,H,L, o de (HL).

Línea 11880:

Si no es ninguno de ellos (RF=0), aparece el mensaje de error "Syntax error en operando". De lo contrario, es devuelto el código del registro en RR y es activado RF.

Línea 11890:

A BB se le asigna el valor del número situado delante de la coma (o del número de Bit).

Línea 11900:

Aquí se comprueba si la cifra se halla entre 0 y 7. Si no tiene ninguno de estos valores, aparece de nuevo "Syntax error en el operando".

Línea 11910:

Finalmente, es guardado el código de operación en PW(2) (V). El código de operación está constituído de la siguiente forma (ver tabla):

01 BB RR - para comandos Bit
10 BB RR - para comandos RES
11 BB RR - para comandos SET

A partir de (ZW-31)*64 se obtienen los Bits 7 y 6 del código de operación. Aquí, ZW (V) es la posición del comando en T1$ (V). BB*8 representa los Bits 5 a 3, y RR, los Bits 2 a 0. RR ha sido determinado por el subprograma y corresponde al código de registro. Una vez calculado

el código de operación, se pasa a la proyección mediante >RETURN<. Las demás rutinas funcionan de manera similar.

En las líneas 13080-13190 se hallan

subprogramas utilizados frecuentemente:

Línea 13090-14150:

En primer lugar se comprueba si el operando OP$ (V) contiene una coma. Si es así, es descompuesto en O1$ (V) (parte anterior a la coma) y O2$ (V) (parte posterior a la coma), y KF (V) es colocado a -1 (=verdadero). Además se comprueba si aparecen paréntesis. En caso afirmativo, se almacena el contenido del paréntesis en Q1$ (V) y QF (F) es colocado a (=-1).

Línea 13200-13250:

Aquí se comprueba si hay una etiqueta permitido en A$ (V).

Línea 13260-13280:

Estas líneas verifican si hay un registro (A, B, C, D, E, H, L, (HL)) en A$ (V).

Línea 13290-13310:

Aquí se comprueba si en A$ (V) hay alguno de los pares de registros BC, DE, HL, SP.

Línea 13320-13340:

Comprueba si en A$ (V) hay una condición C, NC, Z, NZ, PO, PE, P, M.

Línea 13350-13410:

Comprueba si A$ (V) es un número, y devuelve el valor de ese número.

Línea 13420-13600:

Estas líneas proporcionan el valor de los 2-Bytes de A$ (V). A$ (V) puede ser tanto un número como una variable (etiqueta).

Línea 13610-13650:

Se determina el valor del byte (byte bajo) de A$ (V).

Línea 13660-13710:

Calcula el desplazamiento para saltos relativos.

Línea 13720-14390:

Inicialización: Se crean las cadenas y campos de datos utilizados para la comparación. VP (V) indica la dirección en la que es almacenada la longitud de la cadena Z$ (V).
FNDK(X) señala el valor de 16-Bits de dos posiciones de memoria situadas una a continuación de la otra.
BP contiene la dirección de inicio de la línea siguiente a la línea 1.
MP es colocado a &HF000.

Línea 13970-14000:

En estas líneas se llevan a cabo las modificaciones del sistema operativo que hacen posible las salidas de pantalla y monitor.

Línea 14010-14060:

Aquí se define la función >USR9<; se convierten a mayúsculas las minúsculas que hay en la cadena.

Línea 14080-14210:

Tratamiento de errores: Al aparecer un "error normal" (ERR<60), se modifica la salida a impresora, proyectándose el correspondiente mensaje de error.

Lista de Variables
(SUB significa : subprograma)

A- entrega al SUB "carga especial de 8-Bits"
A$- entrega a diversos subprogramas
AD- entrega a SUB "calcular desplazamiento": dirección de salto
BB- código del número de Bit en comandos de manipulación de Bits
BF$- palabra de comando ensamblador
BE$- comentario de la línea ensamblador
BP- apunatdor de programa BASIC
CC- código condicional para saltos
CF- activado (es decir =-1), cuando se ha encontrado la condición, desactivado (es decir =0); cuando no, devolución de SUB "cond test"
CO- utilizado para crear los códigos de operación de los correspondientes comandos.
DI$- contiene la distancia en los comandos indexados.
DI- activado en comando indexado con indicación de distancia, sinó, desactivado.
DW- valor de la distancia dada (complem. a dos)
DS- contiene el número de posiciones de memoria reservadas por un comando DS.
FZ- número de errores
I,J,C- contadores para bucles, etc.

FI- activado en comandos indexados, sinó, desactivado.
PI- posición del registro índice (IX o IY) en el operando.
IR$- si se trata de un direccionamiento indexado, IR$ contiene, o bien, IX, o IY
QF- activado si hay paréntesis en el operando, sinó, desactivado.
QI$- contiene el contenido del paréntesis del operando (en caso de que exista parntesis)
KI- activado si hay comas en el operando, sinó, desactivado.
LC- código ASCII del primer carácter de una etiqueta que ha de ser comprobada (SUB "Label Test").
LA$- restitución del nombre de la etiquetá del SUB-Label Test.
LB$- nombre actual de la etiqueta
LF- activado, si se desea un listado, sinó, desactivado.
LP- longitud del comando correspondiente (longitud del código objeto)
LT- apuntador de espacio libre en la tabla de etiquetas (LT$) (Pointer de tabla de Labels)
MP- Contador del programa en lenguaje máquina: indica la posición de memoria en la que es almacenado el siguiente código máquina
MS- dirección de partida del programa máquina
NA$- nombre del programa
NF- activado cuando hay un direccionamiento directo en un comando de carga, sinó, desactivado (SUB carga especial de 8-Bits)
NO- no Label Flag: activado cuando el test de etiqueta da negativo, sinó, desactivado; devolución de SUB label-Test
O1$- parte del operando anterior a la coma
O2$- parte del operando posterior a la coma
OF- desplazamiento calculado de SUB Offset
OP$- operando de trabajo
OA$- operando, original para salida
PO- posición en la línea de una palabra de com-

probación.

PA- posición de "abrir parentesis" en el operando

PZ- posición de "cerrar parentesis" en el operando

RF- activado cuando SUB ha detectado uno de los registros A,B,C,D,E,H,L (HL), sinó, desactivado.

RR- Código del registro; devolución de SUB

T1$- Test varios: contiene todas las palabras de comando que aparecen con un operando

T2$- Test 1 Byte: contiene todas las palabras de comando que solamente aparecen sin operando, y que poseen un código deoperación de 1 Byte

T3$- Test &HED: Contiene todas las palabras de comando que aparecen solamente sin operando, que poseen un código de operación de de 2-Bytes y cuyo primer Byte es &HED

T4$- Test pseudo: contiene todos los pseudocomandos.

UP- Apuntador a tablas de etiquetas indefinidas: indica el siguiente espacio libre en la tabla UL$ o UD$

VP- Indicador de variables: indica la dirección de Z$ en la tabla de variables interna

WE- valor de una expresión, devolución de SUB "valores" o SUB "test de numeros"

WH- Byte alto del valor

WI- primer Byte del código de operación en direccionamiento indexado.

WO- Byte bajo del valor

Z$- contiene la línea actual a tratar

ZA$- contiene la línea actual (original)

ZN- número actual de línea

ZI- entrega a la dirección objeto de SUB "calcular offset"

ZW1- funciones diversas de memoria intermedia

ZW$- funciones diversas de memoria intermedia

```
 LT$(50)- tabla de etiquetas
  WL(50)- valor de las etiquetas de la tabla de valores
 UL$(50)- tabla de las etiquetas indefinidas
UD(50,2)- datos para etiquetas indefinidas
  (I,0) : número de línea donde se detecta
   (I1) : dirección del valor colocado posteriormente
          mediante POKE
  (I,2) : Tipo, es decir, 16-Bits (=2) o offset (=1)
  W2(12)- código de operación de los comandos de
          1 Byte (T2$)
  W3(20)- código de operación de los comandos de
          2-Byte (T3$)
  RG$(7)- tabla de registros: B,C,D,E,H,L,(HL),A
  CD$(7)- tabla de condiciones: NZ,Z,NC,C,PO,PE,A,M
  DR$(3)- tabla de registros dobles: BC,DE,HL,SP
```

4.10 COMANDOS DE ROTACION Y DE DESPLAZAMIENTO

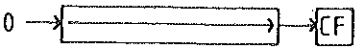
¿Qué se entiende por desplazamiento de las cifras de un número?

```
   4   3   2   1   0
10  10  10  10  10

       3   7   3   0

   3   7   3   0   0 : ¡desplazamiento hacia la
                         izquierda!

           3   7   3 : ¡desplazamiento hacia la derecha!
```

En el sistema decimal, un desplazamiento hacia la izquierda ocasiona una multiplicación por 10 (la base del sistema decimal), y un traslado hacia la derecha ocasiona una división entre 10. (Un desplazamiento de las cifras hacia la izquierda significa un desplazamiento de la coma una posición hacia la derecha).
En el sistema binario, un desplazamiento hacia la izq. o derecha significará respectivamente una multiplicación por 2 o una división entre 2. En BASIC no existe ningún equivalente directo para estos comandos. (será pues, la multiplicación o la división por o entre 2).
El Z80 posee 76 comandos de este tipo, de los cuales, la mayoría utilizan el direccionamiento implícito, indirecto o indexado. Existen varias formas de rotación y de desplazamiento. En primer lugar, distinguiremos entre las operaciones rotar y desplazar

Desplazar: Al correr hacia derecha e izquierda, el contenido del registro es movido Bit a Bit en la dirección correspondiente. El Bit que sobra al realizarse el desplazamiento es recogido en el Flag de acarreo. El espacio que ha quedado libre al otro lado del Byte es rellenado con un 0.

SLA- desplazamiento izquierda aritmético



SRL-desplazamiento derecha lógico

ESQ. 7

Al aplicar el comando SRL a números con signo, se origina un error. El Bit 7, el Bit de signo; es corrido al lugar del Bit 6. En el lugar del Bit 7 se coloca un 0. De ésto resulta que un número negativo (Bit 7=1) se convierte en positivo (Bit 7=0). El comando SRA evita este error. En este comando, el Bit incluído a la izquierda es idéntico al Bit de signo. Es 0 cuando el Bit izquierdo era =0 (+), y 1, cuando el Bit izquierdo era =1 (-). Puesto que este comando tiene en cuenta el significado aritmético del séptimo Bit, se le denomina comando de desplazamiento aritmético, y no lógico.

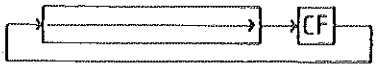
SRA-desplazamiento derecha aritmético

ESQ. 8

Rotar: Contrariamente a la operación de desplazamiento, en la rotación, el Bit que entra es, o bien, el que ha caído fuera al otro extremo, o el Bit de acarreo
En el Z80 se dan dos tipos de rotación:

Rotación de 8-bits (sin Carry)
Rotación de 9-Bits (con Carry)

En una rotación de 9-Bits hacia la derecha se corren todos los ocho Bits una posición hacia la derecha. El Bit que sale del extremo derecho va a parar al Carry. El Bit que entra por el extremo izquierdo corresponde al antiguo contenido del Carry (antes de adoptar el contenido del nuevo Bit que ha entrado en él). Puesto que aquí rotan los 8-Bits del Byte y el Carry (¡el Bit 9!), a este tipo de rotación se la denomina: rotación de 9-Bits.

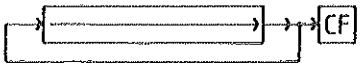


RR-Rotación a la derecha a través del Carry

RL- Rotación a la izquierda a través del Carry

ESQ. 9 Rotación de 9-Bits

En la rotación de 8-Bits, sólo rotan los 8 Bits del registro. En el Carry sólo se registra el Bit que sale. Sin embargo, el anterior contenido del Carry no rota con el resto de los Bits. El Bit que sale es recogido de nuevo por el otro extremo del registro.



RRC - Rotación hacia la derecha

RLC - Rotación hacia la izquierda

Por otro lado, también existen dos comandos especiales para la rotación de cifras (=bloques de 4 Bits) en formato BCD: RLD y RRD (D: Digit-cifra) hacen rotar dos cifras de la posición de memoria que indica HL, y la cifra dada mediante la mitad inferior del acumulador (ver capítulo 5.4).
Por lo general, los comandos de rotación y traslado tienen un código de operaciones de 2 Bytes. El primer Byte del código de operaciones siempre es &HCB. (En los comandos de direccionamiento indexado, &HCB es el 2. Byte, ya que en este tipo de direccionamiento, el primer Byte es, o bien, &HDD ó &HFD. (Excepción: RRD/RLD comienzan por &HED). Puesto que los comandos de rotación son utilizados a menudo para la aritmética, se establecieron otros cuatro comandos. Estos se refieren unicamente al Acumulador y poseen un código de operaciones de 1 Byte. Son exactamente la mitad de largos y el doble de rápidos que los comandos estándar:

"normal"	"Especial-Acumulador"
RLC A	RLCA
RRC A	RRCA
RL A	RLA
RR A	RRA

Los Flags S y Z son influenciados en la forma habitual mediante los comandos normales de rotación y de desplazamiento. El Flag P/V indica la paridad. El contenido del Carry es el correspondiente Bit que es desplazado fuera del Byte. Los comandos especiales para el acumulador sólo modifican el Flag C, mientras los Flags S, Z y P permanecen inalterados. Los comandos de rotación BCD RLD/RRD sólo influencian a los Flags S, Z y P en la forma antes citada, pero no al Carry.

Ejemplos:

```
SRL C           C:&H36

     00110110          :&H36
 0--> 0011011--> 0    al Carry
     00011011          :registro C después de la
                         ejecución
                  0    :Carry después de la ejecución
```

SRL ocasiona una división entre 2: &H36^2=&H1B

```
SRA (HL)      HL:&HB100
              posición de memoria &HB100:&HC2

      11000010      :&HC2
      *1100001--> 0 :Carry
      11000010    0 :CF:(HL) después de la
                       ejecución=&HE1
```

(* Bit 7 queda en esta posición)

Como complemento a dos, significa:

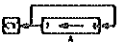
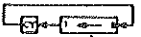
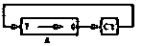
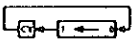
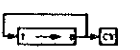
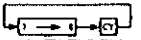
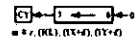
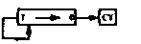
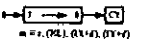
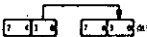
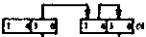
```
&HC2 = -62
&HE1 = -31
```

El comando SRA efectúa correctamente la partición de los números con signo. De lo contrario, SRL (HL) habría tenido &H61 =97 como resultado. Sin embargo, ésta no es la mitad de -62, sinó de 194, que corresponde a &HC2 como número carente de signo.

```
RLC D
    D: &HE4    Carry=1

       &HE4= &B11100100
Carry nuevo <--11100100 <--1=Carry viejo
              11001001
Contenido de D después de la ejecución: &HC5
       Carry = 1
```

De todos modos, &HC5 no es el doble de &HE4. El motivo es debido a que un Bit roto hacia el Carry. Así pues, &H1C5 debe ser el doble de &HE4. Ello no es del todo correcto, pues el Carry viejo (=1) entró por el otro extremo. De este modo, &H1C9-1=&H1C8 es el doble de &HE4.
Si han de rotar números compuestos por varios Bytes, el Bit salido del Byte que acaba de rotar es introducido por RLC o RRC mediante el Carry, hacia el interior del siguiente Byte. (ver programa al final del capítulo).

```
RRA
     Acumulador : &H76

     &H76=&B011101110
          &B*01110110 -> Carry
(* aquí "rota hacia dentro" el viejo Bit 0)

               Acumulador : &B00111011      CF=0
 Contenido del Acumulador : &H3B
                    &H3B*2 = &H76
```

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		‡	•	•	•	0	0	00 000 111	1	1	4	Rotate left circular accumulator
RLA		‡	•	•	•	0	0	00 010 111	1	1	4	Rotate left accumulator
RRCA		‡	•	•	•	0	0	00 001 111	1	1	4	Rotate right circular accumulator
RRA		‡	•	•	•	0	0	00 011 111	1	1	4	Rotate right accumulator
RLC r		‡	‡	P	‡	0	0	11 001 011 00 000 r	2	2	8	Rotate left circular register r
RLC (HL)		‡	‡	P	‡	0	0	11 001 011 00 000 110	2	4	15	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)		‡	‡	P	‡	0	0	11 011 101 11 001 011 ← d → 00 000 110	4	6	23	
RLC (IY+d)		‡	‡	P	‡	0	0	11 111 101 11 001 011 ← d → 00 000 110	4	6	23	
RL m		‡	‡	P	‡	0	0	010				Instruction format and states are as shown for RLC,m. To form new OP-code replace 000 of RLC,m with shown code
RRC m		‡	‡	P	‡	0	0	001				
RR m		‡	‡	P	‡	0	0	011				
SLA m		‡	‡	P	‡	0	0	100				
SRA m		‡	‡	P	‡	0	0	101				
SRL m		‡	‡	P	‡	0	0	111				
RLD		•	‡	P	‡	0	0	11 101 101 01 101 111	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected
RRD		•	‡	P	‡	0	0	11 101 101 01 100 111	2	5	18	

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Aplicación:

La aplicación estándar de los comandos de rotación y de desplazamiento son la multiplicación y división binárias. A continuación, escribiremos un programa para la multiplicación de dos números de 8-Bits. Observemos para ello primero la multiplicación en el sistema decimal:

```
  101*29
  ------
    202
     909
  ------
    2929
```

En el sistema binario,esta multiplicación se realiza de la siguiente forma:

```
101 = &H65 = &B01100101
 29 = &H1D = &B00011101

01100101*00011101
------------------
         0
          0
           0
      01100101
       01100101
        01100101
                 0
          01100101
------------------
     101101110001 = &HB71 = 2929
```

La particularidad de la senciliez de multiplicar en el sistema binario radica en que sólo se multiplica con 0 (=>resultado = 0) o con 1 (=>resultado = factor).

De este modo, la multiplicación de cualquier número por otro es el resultado de una adición normal y un desplazamiento. En la multiplicación escrita, el desplazamiento se efectúa escribiendo, una debajo de otra, cada línea de resultado, desplazándola una posición a la izquierda. De este modo se obtiene el siguiente programa:

```
10 ' ORG &HF000
20 ' LD B,B
30 ' LD A,(FAK1)
40 ' LD E,A
50 ' LD A,(FAK2)
60 ' LD D,0
70 ' LD HL,0
80 ' MULTIP RRCA
90 ' JR NC,NOADD
100 ' ADD HL,DE
110 ' NOADD SLA E
120 ' RL D
130 ' DJNZ MULTIP
140 ' LD (RESULT),HL
150 ' RET
160 ' FAK1 DB 101
170 ' FAK2 DB 29
180 ' RESULT DB 2
190 ' END
```

Ante usted tiene el programa origen para la multiplicación. Al final del programa aparecen dos nuevos pseudocomandos, DS y DB.

DB - Define Byte

DB ocasiona que el Byte indicado detrás del comando DB sea guardado en la dirección actual.

DS - Define Storage

DS reserva la cantidad de posiciones de memoria indicada detrás del comando DS.

El Indicador de dirección es simplemente incrementado en el número indicado. La memoria así reservada es utilizada en nuestro caso para almacenar el resultado.
Introduzca el programa y déje que el ensamblador lo traduzca.
Para comprender perfectamente el método de funcionamiento del programa, tendrá sentido simular el programa una vez ''a mano''. Para ello, coja como valores de partida, los números dados en el ejemplo, y juegue a ser un "procesador":
Cada comando se ejecuta por separado en el papel. El resultado, o sea, los registros y los contenidos de los Flags serán escritos paso por paso. Una vez concluída la simulación completa, su resultado debería darle naturalmente otra vez 2929=&H1B7.

4.11 COMANDOS DE MANIPULACION DE BITS

En el capítulo 4.8 se mostró como se pueden utilizar las operaciones lógicas para activar o desactivar Bits sueltos o grupos de Bits en el Acumulador. Sin embargo, es útil poder activar o desactivar un Bit deseado en un registro o posición de memoria deseados. Debido a que ésto supone el tener que ejecutar un elevado número de comandos, la mayoría de las CPUs disponen para ello de muy pocos o ningun comando. A este respecto, el Z80 está muy bien "surtido". Incluyendo los comandos de comprobación de Bits, posee 120 comandos para la manipulación de Bits.
Los comandos de comprobación de Bits verifican si está activado o desactivado un Bit determinado en un registro o en una posición de memoria. Según el resultado de la comprobación se activará o desactivará el Flag de cero. El Carry no es influenciado, el Flag S y el Flag P/V quedan indeterminados después de la ejecución (!). Los dos comandos para activar (SET) y desactivar (RES) Bits no ejercen ninguna influencia sobre los Flags.
Todos los comandos de Bits comienzan con el código de operación &HCB (a excepción, como siempre, de los comandos de direccionamiento indexado). El segundo código de operación resulta del número del Bit y del código del registro.
Para acceder al Byte afectado se dispone de los siguientes direccionamientos:

- Implícito : registros A, B, C, D, E, H, L
- Indirecto : (HL)
- Indexado : (IX+d) y (IY+d)

Formato:

BIT b,r	BIT b,(HL)	BIT b,(IX+d)	Bit b,(IY+d)
SET b,r	SET b,(HL)	SET b,(IX+d)	SET b,(IY+d)
RES b,r	m= r, (HL)	RES b,(IX+d)	RES b,(IY+d)

b=número de Bit

El número del Bit b, es codificado de la siguiente forma:

0- 000	4- 100
1- 001	5- 101
2- 010	6- 110
3- 011	7- 111

Todos estos comandos también son considerados como comandos de direccionamiento de Bits debido a que el correspondiente Bit es indicado en el código de operación.

Ejemplos:

```
BIT 6,B        B:&H33

            &B00110011 =&H33
              *
              76543210 -número de Bit

           *:El Bit número 6 es 0.
           Puesto que el Bit 6=0, el Flag Z es activado a 1
           después de la ejecución:

              B=&H33       Flag: S Z V C
                                 X 1 X     X=Flag S, V son
                                             desconocidos

RES 1,(HL)    HL:&HA975
              &H1975=&H23

            &B00100011  =&H23
                   *
              76543210  -núm. de Bit

           *:el Bit número 1 es desactivado.
```

```
          &B00100001  =&H21

   Posición de memoria &HA975 después de la ejecución:&H21

         Flags: S Z V C
                          -ninguna influencia
0
SET 7,C    C:&H7F

           &B01111111  =&H7F
              *
            76543210  -núm. de Bit

           *-Bit 7 es activado.

             &B11111111  =&HFF

           Flag C después de la ejecución es:&HFF

           Flags: S Z V C
                          -ninguna influencia
```

Analogías en BASIC

Intentemos ejecutar el comando SET b,A en BASIC:
El Bit b ha de ser activado. Con el comando >OR< tenemos la posibilidad de activar determinados Bits. El Bit b tiene el valor de posición 2^b. Se obtiene:

```
   SET b,A          BASIC: A=A OR (2^b)
```

Para RES se obtiene algo similar:

```
   RES b,A          BASIC: A=A AND (255-2^b)
```

Los comandos especiales SCF y CCF:

Puesto que el Bit 0 se utiliza con bastante frecuencia en el registro F (el Carry), existen para ello dos comandos especiales.
SCF pone el Carry en el valor 1.
CCF complementa el valor del Carry F, es decir, de C=0 se pasa a C=1 y viceversa.
Estos son los únicos comandos con los que se puede influir directamente sobre los Flags.
El Carry puede ser borrado con los comandos lógicos.
Encontrará los comandos CCF y SCF en la lista de comandos al final del capítulo siguiente.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r_b}$	•	‡	X	X	0	1	11 001 011 01 b r	2	2	8	
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	•	‡	X	X	0	1	11 001 011 01 b 110	2	3	12	
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)_b}$	•	‡	X	X	0	1	11 011 101 11 001 011 ← d → 01 b 110	4	5	20	
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)_b}$	•	‡	X	X	0	1	11 111 101 11 001 011 ← d → 01 b 110	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b r	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b 110	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	11 011 101 11 001 011 ← d → [11] b 110	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	11 111 101 11 001 011 ← d → [11] b 110	4	6	23	
RES b, m	$s_b \leftarrow 0$ m≡r, (HL), (IX+d), (IY+d)							[10]				To form new OP-code replace [11] of SET b,m with [10]. Flags and time states for SET instruction

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

b	Bit Tested
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Notes: The notation s_b indicates bit b (0 to 7) of location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, ‡ = flag is affected according to the result of the operation.

4.12 COMANDOS DE CONTROL

Los comandos de control modifican o influyen sobre el modo operativo o el funcionamiento de la CPU.

El comando NOP

NOP significa No Operation. Así pues, el coamndo NOP carece de función. Aunque parezca paradójico, tiene su justificación. Por un lado, el comando NOP puede ser utilizado para un retardo intencionado; en el ordenador MSX, un comando NOP dura aproximadamente un microsegundo, y por otro lado, este comando puede utilizarse como reserva de espacio en programas. De este modo resulta más fácil la búsqueda o rectificación de errores. El código de operación de NOP es &H00, es decir que si el programa es ejecutado por descuido en un área borrada, no corre el riesgo de ser dañado o modificado debido a que NOP no ocasiona modificaciones.

El comando STOP

Este comando interrumpe las operaciones de la CPU durante el tiempo necesario hasta que se ejecute un Reset o un interrupt.

Comandos de control interrupt

Una interrupción sirve principalmente para la ejecución de proceso, importante en el ordenador. Una interrupción es el aviso que da un componente del ordenador sobre la entrada a un estado, como p.ej., la espera de los periféricos de entrada/salida a la entrada de datos. La CPU se encarga de tratar esos avisos según su importancia. Un programa que se esté ejecutando normalmente puede ser interrumpido por un interrupt. Estas interrupciones juegan un papel importante en la entrada y salida de datos. Los ordenadores MSX ofrecen la posibilidad de programar interrupciones también a partir del BASIC (ON INTERVAL, ON STOP, etc.). En estos comandos, el interrupt es activado por el reloj interno del

procesador.Si se requiere una interrupción, el programa bifurca a la dirección de inicio de un subprograma que ejecutará las acciones correspondientes de la respectiva interrupción. De este Programa Servicio Interrupción se puede saltar de nuevo al programa principal mediante el comando RETI (Return Interrupt).
Se distingue entre interrupciones con máscara e interrupciones sin máscara. Las últimas se ejecutan bajo cualquier condición. Tienen máxima prioridad. El comando RETN permite retornar desde un Non-Mascerable-Interrupt (NMI).

DI Disable Interrupt y EI

El comando DI provoca que desde el instante en que se inicia su ejecución, no se tengan en cuenta las interrupciones con máscara. Las interrupciones permanecen así bloqueados hasta que el comando EI (Enable Interrupt) las permite de nuevo.

El Z80 posee tres Modos de Interrupción: IM 0, IM 1, IM 2

IM 0 (Modo Interrupción 0)

Con el IM 0 se puede pasar del modo 1 al modo 0.
Después de una interrupción, el procesador espera en ese modo el comando de un periférico.

IM 1

Este es el modo estándar que aparece al conectar el ordenador.
Cuando se produce una interrupción en este modo, se bifurca automáticamente a la dirección &H30.

IM 2 (Vector-Interrupt)

En IM 2 se bifurca a una dirección existente en una tabla de función.

Mnemonic	Symbolic Operation	Flags						Op-Code	No. of Bytes	No. of M Cycles	No. of T States	Comments
		C	Z	P/V	S	N	H	76 543 210				
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	‡	‡	P	‡	•	‡	00 100 111	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	•	•	•	•	1	1	00 101 111	1	1	4	Complement accumulator (one's complement)
NEG	$A \leftarrow 0 - A$	‡	‡	V	‡	1	‡	11 101 101 01 000 100	2	2	8	Negate acc. (two's complement)
CCF	$CY \leftarrow \overline{CY}$	‡	•	•	•	0	X	00 111 111	1	1	4	Complement carry flag
SCF	CY ← 1	1	•	•	•	0	0	00 110 111	1	1	4	Set carry flag
NOP	No operation	•	•	•	•	•	•	00 000 000	1	1	4	
HALT	CPU halted	•	•	•	•	•	•	01 110 110	1	1	4	
DI	IFF ← 0	•	•	•	•	•	•	11 110 011	1	1	4	
EI	IFF ← 1	•	•	•	•	•	•	11 111 011	1	1	4	
IM 0	Set interrupt mode 0	•	•	•	•	•	•	11 101 101 01 000 110	2	2	8	
IM 1	Set interrupt mode 1	•	•	•	•	•	•	11 101 101 01 010 110	2	2	8	
IM2	Set interrupt mode 2	•	•	•	•	•	•	11 101 101 01 011 110	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

4.13 COMANDOS DE ENTRADA Y SALIDA

Los comandos E/S (entrada/Salida) son un grupo de comandos del Z80 a menudo descuidados. Ello es debido a que la eficacia y el modo de funcionamiento de los comandos E/S dependen en gran medida del hardware. Cada ordenador utiliza para comunicarse con los dispositivos periféricos un comando especial E/S IC como mínimo. Su ordenador posee un PPI (Programmable Peripherical Interface (8255)). También se utilizan otras denominaciones, como PIO, PIA. Una parte de estos comandos E/S se refieren a este componente, que luego ejecuta independientemente la comunicación con el teclado, casette y control de memoria. Además, el PSG (Programmable Sound Generator) encargado de la generación de sonido y del port del Joystick, y el VDP (Video Display Processor), que produce la imagen del monitor, también están unidos al procesador mediante conectores E/S. Como usted ve, todos los elementos requeridos para los comandos E/S son partes importantes del ordenador. La conexión entre estos elementos que trabajan con tanta independencia (es decir p.ej., el VDP produce de manera continuada e independiente de la CPU la señal de salida del video) y el Z80, se realiza a través de los citados comandos E/S. Con ayuda de ellos se transfieren comandos y datos a los elementos del IC o se reciben de él. Los comandos E/S se encargan entonces de la comunicación real con los periféricos.

Para esta conexión, en el "lenguaje del ordenador" se utiliza a menudo el concepto "interfase". Como hemos descrito, existen pues interfases internos (generalmente: elemento procesador -E/S) e interfases externos (generalmente: elemento periférico E/S).

Un comando E/S coloca (o escribe) simplemente un valor en el respectivo interfase. Si hay algún aparato conectado, el valor es recibido y se ejecuta la consiguiente acción. Normalmente existen, en el caso normal, un total de 256 direcciones E/S distintas posibles. La dirección (también dirección del port E/S) determina a que "periférico" se han de enviar los datos. En el curso del libro comentaremos la importancia de algunos ports E/S.

De lo anterior, usted puede deducir que un comando completo necesita dos actos:

- La dirección del port E/S
- El valor de los datos que son "enviados", o bien, el registro donde deberán ser almacenados los datos recibidos.

La dirección del port estará siempre entre paréntesis. En el comando IN, los datos son leídos del port y almacenados en el registro indicado. El comando OUT "envía" los datos indicados al port correspondiente.
Los comandos E/S se presentan en dos tipos de direccionamiento:

Direccionamiento inmediato

Formato:

OUT (n),A IN A, (n)

Direccionamiento indirecto

Formato:

OUT (C),r IN r,(C)

Estos comandos tienen en el BASIC del MSX los comandos análogos >INP< y >OUT<. Su método de funcionamiento es igual al de los comandos en código máquina, pero la realización de muchas de las funcioens sólo es posible a partir del código máquina.
Además, aún existen otros cuatro comandos para la entrada/salida de bloques. Se utilizan de manera similar a los comandos de transferencia de bloque: HL indica la dirección correspondiente en la memoria, C, la dirección del port, y B, la longitud del bloque.

Los comandos E/S comienzan con el código &HED y poseen un código de 2 Bytes. Unicamente los comandos de direccionamiento inmediato IN A,(n) y OUT (n),A son representados por un código de 1 Byte.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011 ← n →	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	•	‡	P	‡	0	‡	11 101 101 01 r 000	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	‡ ①	X	X	1	X	11 101 101 10 100 010	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1 ①	X	X	1	X	11 101 101 10 110 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	‡ ①	X	X	1	X	11 101 101 10 101 010	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL -1 Repeat until B = 0	X	1 ①	X	X	1	X	11 101 101 10 111 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011 ← n →	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	•	•	•	•	11 101 101 01 r 001	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	‡ ①	X	X	1	X	11 101 101 10 100 011	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1 ①	X	X	1	X	11 101 101 10 110 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	‡ ①	X	X	1	X	11 101 101 10 101 011	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL -1 Repeat until B = 0	X	1 ①	X	X	1	X	11 101 101 10 111 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Aplicación:

Para ilustrar los comandos de E/S, los utilizaremos para la producción de sonidos. Seguramente, ya habrá notado que al accionar una tecla de función de su ordenador MSX se producen unos "crujidos". Como caso excepcional, tales crujidos, aunque se trata de "tonos", no los produce el PSG (Sound Chip). Es bastante más probable que el crujido se emita al canal audio a través del PPI (Chip programable E/S). Esto ocurre sencillamente al activar y desactivar una línea que se halle conectada a esta salida. El paso de un estado a otro ocasiona el movimiento de la membrana del altavoz, y de ahí, que se oiga un crujido.
Así pues, es interesante esta posibilidad que permite, no sólo producir ese crujido, sinó también el hecho de, que mediante una rápida y sucesiva activación y desactivación de la línea sea posible hacer vibrar el altavoz de tal modo que nos permita oir un sonido.

Desconecte primero el sonido de la acción del teclado mediante >SCREEN,,0< o >POKE &HF3DB,0<. Luego, introduzca lo siguiente:

```
OUT &HAB,15
```

A pesar de haber sido desconectado el sonido, usted oirá un crujido al accionar la tecla Return. Este ruído nos demuestra que la membrana del altavoz es ahora "tensada". Si vuelve a ejecutar el mismo comando, ya no oirá nada pues el estado de la membrana no varía.
Mediante

```
OUT &HAB,14
```

la tensión del altavoz es colocada de nuevo a cero. Puesto que la membrana estaba tensada mediante el comando >OUT &HAB,15<, también se oirá aquí un crujido.

Mantendremos el estado actual de la denominada línea de sonido software mediante un comando >INP<. El estado de la línea corresponde al Bit 7 del Port C del PPI. Este Port tiene la dirección &HAA. Para considerar exclusivamente al Bit 7, utilizaremos los comandos lógicos (¡se acuerda?):

```
PRINT INP(&HAA) AND 2^7
```

Si tras introducir la línea anterior usted obtiene como resultado 0, es debido a que la conducción era 0. Si se obtiene 128, es que la conducción se hallaba en el estado 1.

Puesto que un tono suena a consecuencia de las vibraciones de una cuerda o de la membrana de un altavoz, la rápida activación y desactivación de la línea de sonido software provoca la aparición de un tono.

```
10 FOR I=0 TO 100
20 OUT &HAB,15
30 OUT &HAB,14
40 NEXT
```

Añadiendo una línea 25, en la que p.ej. haya un bucle de espera, un comando >REM< o un comando >PRINT<, puede ser reducida la velocidad que provoca la aparición de los "crujidos". De este modo disminuye la frecuencia a la que vibra la membrana del altavoz. Pocas vibraciones, o sea, frecuencia baja, producen un tono bajo. Muchas vibraciones, o sea, frecuencia alta, producen un tono elevado.
En BASIC, este tipo de producción de sonidos es realmente sólo un juego, pues no es posible producir sonidos de más de 250 Hz. Escribiremos pues un programa en código máquina para esta operación:

La estructura del programa es similar a la del programa en BASIC.

Un bucle externo con un contador determina la duración del tono. Dentro de este bucle se deberá activar y desactivar la línea del sonido a distancias regulares.
La duración determinada por un bucle de espera establece la frecuencia del tono.
Análogamente al BASIC, para la producción del crujido utilizaremos también el comando OUT (C),r.

```
10 ' ; oscilación rectangular
20 ' LD C,&HAB ; dirección del port
30 ' LD B,&HF ; "sonido en High"
40 ' LD DE,(LONGIT) ; longitud del tono
50 ' TON OUT (C),B ; línea a 1
60 ' DEC DE ; disminuir el contador
70 '  LD  HL (FREWA) ; contador para el bucle de espera que
determina la frecuencia
80 ' WART1 DEC HL ; disminuir contador
90 ' LD A,H ; comprobar si
100 ' OR L ; contador=0
110 ' JR NZ,WART1 ; si negativo, seguir esperando
120 ' DEC B ; "sonido en Low"
130 ' OUT (C),B ; línea a 0
140 ' LD HL,(FREWA) ; contador
150 ' WART2 DEC HL ; segundo
160 ' LD A,H ; bucle de espera
170 ' OR L ; como arriba
180 ' JR NZ,WART2
190 ' INC B ; vuelta a "sonido en High"
200 ' LD A,D ; comprobar si
210 ' OR E ; contador para longitud=0
220 ' JR NZ,TON ; si negativo, producir sig. ondulación
230 ' RET ; si positivo, saltar hacia atrás
240 ' LONGIT DW 440 ; duración del tono = 1 seg
250 ' FREWA DW 155 ; frecuencia = 440 Hz
```

El programa se explica en detalle a continuación.

Para comprender como se realizan los valores para el "buvcle de longitud" y el bucle de espera de la frecuencia, deberemos profundizar un poco más. El Z80 descompone internamente en operaciones los comandos en código máquina programados por nosotros. Una sucesión perfectamente establecida de estas operaciones provoca en conjunto la ejecución de los comandos. Las acciones tales como la escritura/lectura de la memoria y la escritura/lectura de los periféricos de E/S, forman parte, entre otras, de las operaciones elementales. Una operación de este tipo, cerrada en sí misma lógicamente, se denomina Ciclo de máquina.
Un comando se compone de uno hasta cinco ciclos máquina (M1 - M5). Cada ciclo máquina tiene una duración de 3 hasta 6 períodos. La duración de uno de estos períodos se establece a partir de la frecuencia de impulsos, con la que funciona la CPU. El generador de impulsos es un elemento importante del ordenador. Garantiza el funcionamiento sincronizado de todos los procesos a pesar de la elevada velocidad a la que se realizan. Todos los procesos que tienen lugar en el ordenador se guían por estos "impulsos". La norma del MSX establece una frecuencia de impulsos de 3.58 Megaherzios (MHz). Esto significa que el periodo, es decir, la duración de una oscilación del generador de impulsos es:

$$\frac{1}{3.58\ \text{MHz}} = \frac{1}{3580000\ \text{Hz}} = 0.00000028\ \text{seg} = 0.28\ \text{Microseg.}$$

A este espacio se la califica como frecuencia del reloj (T). Observemos un ejemplo de ejecución de un comando:

El ciclo M1 también recibe el nombre de Instructionfetch (engl. fetch:coger).Se compone como mínimo de 4 ciclos de impulsos. El funcionamiento es el siguiente:

M1 Instructionfetch
   T1 : sacar el PC al bus de dirección
   T2 : aumentar el PC

T3 : almacenar en la memoria intermedia el valor del bus de datos (corresponde al valor de la dirección a la que indicaba el PC en T1)

T4,5,6 : ejecutar las operaciones dependientes del comando

Todos los demás ciclos-M necesarios tienen entre 3 y 5 ciclos-T. No vamos a ocuparnos con más detalle de su exacta estructura. Lo importante es que podemos calcular con exactitud la duración del proceso a partir del dato de los ciclos de impulsos necesarios para un comando. Los comandos más cortos sólo poseen el ciclo-M1, que a su vez, se compone de 4 ciclos de impulsos (p.ej. LD r,r'). En una frecuencia de impulsos de 3.58 MHz, la duración de la ejecución de este comando es de 4*0.28 microseg = 1.12 microseg; o bien, expresado de otro modo, en un segundo pueden ser ejecutados casi un millón de estos comandos.
En las listas de comandos hallará el número necesario de ciclos-M y T (No. of T Cycles) para cada comando. Con estos datos, intente calcular la duración de un funcionamiento de 155 veces a través del bucle de espera que determina la frecuencia.

Solución:

Comando:	Ciclos-T:
WART DEC HL	6
LD A,H	4
OR L	4
JR NZ,WART	12 (7)
	26

Duración=(155*26-5)*0.28 microseg.
= 11127 microseg.
= 1.127 millseg.

Puesto que para una oscilación total, el bucle de espera realizará dos recorridos, la duración de la vibración del tono producido será aprox. de 1.127+1.127=2.254 millseg. Por lo tanto, la frecuencia del tono es:

$$\frac{1}{2.254\text{mseg}} = \frac{1}{0.002254\text{seg}} = 443.7\ \text{Hz}$$

Este cálculo es aproximado debido a que la duración de ejecución de los comandos fuera de los bucles no ha sido tenida en cuenta:

```
1. DEC DE            6
   LD HL,(FREWE)    16
   DEC B             4
   OUT (C),B        12
                    --
                    38

2. LD HL,(FREWA)     16
   INC B              4
   LD A,D             4
   OR E               4
   JR NZ,TONO        12
   OUT (C),B         12
                     --
                     52
```

D1=(38+N1*26-5)*T=26*N1*T+33*T
D2=(52+N2*26-5)*T=26*N2*T+47*T

$$T: \text{Duración de un ciclo} = \frac{1}{3.579545}\ \text{microseg.}$$

N: Número de pasadas del bucle de espera

La frecuencia (F) de un tono es igual al valor recíprocode la duración de la oscilación (D).
La duración total de la oscilación resulta de la suma de los dos tiempos indicados arriba:

$$D=D1+D2$$

1/F=26*T*(N1+N2)+80*T
y continuando el cálculo

$$N1+N2 = \frac{1/F-80*T}{26*T} = \frac{1}{F*26*T} - \frac{80}{26}$$

$$\text{Con } T = \frac{1}{3.579545}\text{microseg} = \frac{1}{3579545} \text{ obtenemos}$$

$$N1+N2 = \frac{3579545}{F*26} - 3.08$$

Vamos a probar nuestra fórmula:

La frecuencia deberá ser de 440 Hz.

$$N1+N2 = \frac{3579545}{440*26} - 3.08$$

$$N1+N2 = 309.8$$

Puesto que en el programa de arriba no se hace distinción entre N1 y N2 (allí FREWA), será válida la igualdad N1=N2= FREWA=155=310/2.
Para este valor, el bucle grande (DE como contador) realizará el recorrido 440 veces (=frecuencia). El contador (LONGIT) estará colocado a 440, lo cual origina un tono de 1 seg de duración.
Si usted prueba el programa, verá que el tono producido se oye de una forma algo peculiar, no correspondiendo al tono deseado de 440 Hz (=diapasón normal A). El tono se escucha con interrupciones regulares de tiempo. ¿Cómo es posible producir estas rápidas y continuadas interrupciones siendo como es el programa un programa cerrado en sí mismo?

¿Sabía usted que...
Cada programa (!) es interrumpido a cada 50avo. segundo por un interrupt? Es decir que en se bifurca un determinado punto del sistema operativo donde se comprueba, por así decirlo, si ha sido pulsada, p.ej., una tecla (STOP!) o el disparador del Joystick (ON STRIG). El reloj interno no se detiene en este caso (por ello habrá que dividir el valor de la varia le TIME entre 50, para poder así obtener el tiempo en segundos). Así pues, esta interrupción, generalmente tan útil, distorsiona nuestro tono de diapasón normal.
Para evitar la aparición de interrupciones no deseadas, existen los comandos DI (Disable Interrupt) y EI (Enable Interrupt) anteriormente descritos. DI evita que se bifurque a la rutina del interrupt, y EI provoca que ésta sea utilizada de nuevo.
Añadiremos pues el comando DI en la línea 45 a nuestro programa. De este modo evitaremos que se originen interrupciones durante la total duración del tono. En la línea 225 añadiremos el comando EI para volver al estado original.
Realice usted el ensamblaje de nuevo, y de este modo, obtendremos por fin el resultado deseado.
El hecho de conectar los interrupts presenta un pequeño inconveniente:
Con el interrupt conectado, es posible interrumpir todos (!) los programas -también los programas en código máquina-, pulsando al mismo tiempo las teclas de función SHIFT-, CTRL-, CODE-, y GRAPH-. Con el interrupt desconectado, ésto no es posible, lo que también puede ser ventajoso. Si p.ej., su programa se encuentra en un bucle sin fin, después del comando DI, sólo queda la tecla de función RESET para interrumpir el programa. Así pues, sería conveniente quitar el comando DI al hacer pruebas, puesto que en caso de aparecer errores, quedaría la posibilidad de evitar que el ordenador se bloquee pulsando las teclas de función "CTRL + SHIFT + CODE + GRAPH".
Modifique el programa de modo que los dos bucles de espera puedan funcionar también con valores distintos:

```
      70 ' LD HL,(FREWA1)
      140 ' LD HL,(FREWA2)

      250 ' FREWA1 DW  155
      255 ' FREWA2 DW  155
```

Tras pulsar >RETURN< se obtiene el listado completo del programa ensamblador:

```
 F000            10                ; oscilación rectán-
 gular
 F000 0EAB       20          LD    C,&HAB
 F002 060F       30          LD    B,&HF
?F004 ED5B0000   40          LD    DE,(LONGIT)
 F008 F3         50          DI
 F009 ED41       60  TONO    OUT   (C),B
 F008 1B         70          DEC   DE
?F00C 2A0000     80          LD    HL,(FREWA1)
 F00F 2B         90  WART1   DEC   HL
 F010 7C         100         LD    A,H
 F011 B5         110         OR    L
 F012 20FB       120         JR    NZ,WART1
 F014 05         130         DEC   B
 F015 ED41       140         OUT   (C),B
?F017 2A0000     150         LD    HL,(FREWA2)
 F01A 2B         160 WART2   DEC   HL
 F01B 7C         170         LD    A,H
 F01C B5         180         OR    L
 F01D 20FB       190         JR    NZ,WART2
 F01F 04         200         INC   B
 F020 7A         210         LD    A,D
 F021 B3         220         OR    E
 F022 20E5       230         JR    NZ,TONO
 F024 FB         240         EI
 F025 C9         250         RET
**** Línea  40  : LONGIT.=&HF026
 F026 9001       260 LONGIT.  DW    400
**** Línea  80  : FREWA1=&HF028
 F028 AC00       270 FREWA1 DW    172
```

```
**** Línea 150  : FREWA2=&HF02A
 F02A AC00       280 FREWA2  DW    172
```

```
Programa : Tono
Inicio : &HF000        Fin : &HF02B
Longitud : &H2C Bytes
Errores : 0
Tabla de Variables:
TONO   F009   WART1   F00F
WART2  F01A   LONGIT.   F026
FREWA1 F028   FREWA2  F02A
```

Observemos el listado del programa ensamblador:

En la línea 40 se realiza una referencia a la etiqueta LONGIT, que no vuelve a aparecer hasta la línea 260. Por ello es almacenado de momento ED5B0000 como código. En la traducción de la línea 206, el ensamblador encuentra la etiqueta LONGIT y señala que ésta ya ha aparecido antes en la línea 40. El código ED5B0000 es colocado correctamente de forma automática. Esto también es posible para JR, CALL y JP. Este problema aparece cuando en el programa se efectúa un salto hacia adelante.
Es necesario hacer el tratamiento de los saltps hacia adelante de este modo, ya que el ensamblador es de una sola pasada. Esto significa que el ensamblador sólo busca una sóla vez en el programa fuente.
Por el contrario, un ensamblador de dos pasos busca en su primer recorrido unicamente todas las variables y etiquetas, asignando valores a cada uno. Y es en el segundo paso, cuando tiene lugar la traducción. Los grandes ensambladores profesionales realizan más de dos pasadas. Para nuestros fines, el ensamblador de un paso nos es más útil ya que es casi dos veces más rápido que el ensamblador de dos pasos.
Para conseguir una utilización cómoda de la rutina, escriba por favor también un programa en BASIC que calcule, mediante la introducción de frecuencia, duración del tono y relación High-Low, los valores correspondientes para el programa en código máquina, y los escriba en las posiciones respectivas con >POKE<. La relación High-Low es la relación que mantienen entre sí los dos valores de los bucles de espera.

Mientras la suma de los valores sea constante, el tono mantendrá la misma frecuencia base. Si se cambia la relación, varía también la tonalidad del sonido, y el volumen del tono obtenido.

Programa de utilidad en BASIC

```
10 REM oscilación rectangular
20 CLEAR 200,&HEFFF
30 CLS
40 INPUT "frec";F
50 INPUT "duracion";D
60 INPUT "hi/lo";P
70 L=D*F
80 REM Longitud
90 PPOKE &HF026,L-INT(L/256)*256
100 POKE &HF027,INT(L/256)
110 W=3579545#/2/F/26-80/26
120 W2=W/(1+P)
130 W1=W2*P
140 REM Frecuencia
150 POKE &HF028,W1-INT(W1/256)*256
160 POKE &HF029,INT(W1/256)
170 POKE &HF02A,W2-INT(W2/256)*256
180 POKE &HF02B,INT(W2/256)
190 X=USR1(1)
200 GOTO 40
```

CAPITULO V : P R O G R A M A C I O N

5.1 EL DESENSAMBLADOR

Los ordenadores MSX poseen una ROM de 32k. Estos 32Kilobytes contienen rutinas del sistema. Las 16K ROM superiores (&H4000 hasta &H7FFF) contienen el BASIC, las 16K (&H0 hasta &H3FFF) inferiores , el sistema operativo del ordenador. La memoria ROM alberga muchas rutinas de gran interés para el programador en código máquina.
Para analizar estas rutinas necesitamos el desensamblador como un "instrumento" más.
El desensamblador interpreta los Bytes de un espacio dado como si fuesen códigos máquina y traduce los números correspondientes a los comandos en lenguaje ensamblador. De este modo, el desensamblador constituye la parte opuesta del ensamblador. Con el desensamblador podemos realizar la traducción inversa a códigos ensamblador de programas desconocidos en código máquina que aparecen como cargadores BASIC (Líneas DATA) una vez han sido cargados. También es posible traducir de este modo las rutinas internas del ordenador. Se puede sacar mucho partido de estos programas elaborados por "profesionales". Además, también podemos aplicar las rutinas a nuestros propios programas.
Para probar el desensamblador, inicielo con >RUN<, e introduzca &H146A como dirección de partida y &H146F como dirección final.
Obtendrá el siguiente cuadro.

```
146A 7C        LD  A,H
146B 92        SUB D
146C C0        RET NZ
146D 7D        LD  A,L
146E 93        SUB E
146F C9        RET
```

Esta rutina del sistema sirve para comparar los registros HL y DE. Aquí, el Z80 no tiene previsto para ello ningún comando directo. Pero, como esta operación se necesita muy a menudo resultados muy útiles, ha sido convertida en una rutina RESTART, es decir, que puede ser llamada a través de un comando RST. Obsérvelo usted mismo en la dirección &H20 (=dirección de bifurcación del comando RST &H20):

```
0020 C36A14        JP  &H146A
```

Intente determinar como son influenciados los Flags para los casos HL>DE, HL<DE y HL=DE.

Una vez realizado el desensamblaje del área que usted ha definido, el programa queda a la espera de una instrucción (pulsar una tecla de función). Si usted pulsa "E" o "e", el programa finaliza, de lo contrario, salta de nuevo al MENU. ¡Ponga atención a la descripción del programa al introducir el desensamblador!.

```
10 CLEAR 200,&HF000:MAXFILES=0
20 SCREEN 0:COLOR 1,14,14:KEY OFF
30 GOTO 990
40 LOCATE 1,4:PRINT"Z 8 0 - D E S E N S A M B L A D O R"
50 E$="":LOCATE 3,7:INPUT"Impresora (s/n)";E$
60 LOCATE 3,10:INPUT"Dirección de INICIO :&H";A$
70 GOSUB 900:AN=A
80 LOCATE 3,12:INPUT"Dirección final  :&H";A$
90 GOSUB 900:EN=A
100 IF AN>EN THEN CLS:GOTO 40
110 IF E$="s" OR E$="S" THEN POKE &HFEE4,&HC3
115 IF E$="s" OR E$="S" THEN POKE &HFEE4,&HC3
120 PC=AN
130 CLS
140 AD=PC
150 PRINTRIGHT$("000"+HEX$(AD),4);"  ";
160 IL=0
170 GOSUB 940
180 GOSUB 300
190 IF IL THEN 600
200 IF LEFT$(PR$,3)="RST" AND (W=&HCF) THEN PR$=PR$+" /DB:n
"
210 IF INSTR(PR$,"n")<>0 THEN 700
220 IF INSTR(PR$,"e")<>0 THEN 820
230 PO=INSTR(PR$," ")
240 IF PR$="" THEN PR$="???"
250 IF PO=0 THEN PRINTTAB(20);PR$;:GOTO 270
260 PRINTTAB(20);LEFT$(PR$,PO-1);TAB(25);RIGHT$(PR$,LEN(PR$
)-PO);
270 PRINT
280 IF PC<=EN THEN 140 ELSE POKE &HFEE4,&HC9
285 A$=INKEY$:IF A$="" THEN 285
290 IF A$<>"E" AND A$<>"e" THEN CLS:GOTO 50
295 POKE &HFEE4,&HC9:KEY ON:END
300 REM Interpretar
310 IF (W=&HDD OR W=&HFD) AND NOT IL THEN 490
```

```
320 IF W=&HED THEN 460
330 IF W=&HCB THEN 410
340 GOSUB 540
350 ON C1 GOTO 370,380,360
360 PR$=BF$(W):RETURN
370 IF W=&H76 THEN PR$="ALTO":RETURN
380 PR$="LD "+RT$(C2)+","+RG$:RETURN
390 IF C2=0 OR C2=1 OR C2=3 THEN A$=" A," ELSE A$=" "
400 PR$=AL$(C2)+A$+RG$:RETURN
410 REM Comandos con primer Byte &HCB
420 GOSUB 940
425 IF IL THEN DI=W:GOSUB 940
430 GOSUB 540
440 IF C1=0 THEN PR$=RS$(C2)+" "+RG$ ELSE PR$=BI$(C1)+STR$(
C2)+","+RG$
450 RETURN
460 REM Comandos con primer Byte &HED
470 GOSUB 940
480 IF W<&H40 OR W>&HBF THEN PR$="???":RETURN ELSE GOTO 360
490 REM Comandos Indexados
500 IL=-1
510 IF W=&HDD THEN I$="IX" ELSE I$="IY"
520 GOSUB 940
530 GOTO 300
540 REM descomponer códigos
550 C1=(W AND &B11000000)/64
560 C2=(W AND &B111000)/8
570 C3=W AND &B111
580 RG$=RT$(C3)
590 RETURN
600 REM indexado
610 PO=INSTR(PR$,"HL")
620 IF PO=0 THEN PR$="???":GOTO 230
630 IF INSTR(PR$,"(HL)")<>0 THEN 670
640 IF PR$="EX DE,HL" THEN PR$="???":GOTO 230
```

```
650 IF PR$="ADD HL,HL" THEN PR$="ADD "+I$+","+I$:GOTO 230
660 PR$=LEFT$(PR$,PO-1)+I$+RIGHT$(PR$,LEN(PR$)-PO-1):GOTO 2
00
670 IF LEFT$(PR$,2)="JP" THEN 660
680 IF PC-ADR<3 THEN GOSUB 940:DI=W
685 IF DI>127 THEN DI$=STR$(DI-256) ELSE DI$="+"+RIGHT$(STR
$(DI),LEN(STR$(DI))-1)
690 I$=I$+DI$:GOTO 660
700 REM sustituir n
710 PO=INSTR(PR$,"nn")
720 IF PO<>0 THEN 770
730 PO=INSTR(PR$,"n")
740 GOSUB 940
750 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("00"+HEX$(W),2)+RIGHT$(
PR$,LEN(PR$)-PO)
760 GOTO 230
770 GOSUB 940:LB=W
780 GOSUB 940
790 WE=W*256+LB
800 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("0000"+HEX$(WE),4)+RIGH
T$(PR$,LEN(PR$)-PO-1)
810 GOTO 230
820 REM sustituir e
830 PO=INSTR(PR$,"e")
840 GOSUB 940
850 IF W>127 THEN W=W-256:REM Complemento de dos.
860 W=W+2
870 A$="$"+STR$(W)+" >"+"&H"+RIGHT$("0000"+HEX$(PC+W-2),4)
880 PR$=LEFT$(PR$,PO-1)+A$+RIGHT$(PR$,LEN(PR$)-PO)
890 GOTO 230
900 REM Conversión hex -> dec
910 IF A$="" THEN A=0:RETURN
920 A=VAL("&H"+A$)
930 RETURN
940 REM leer Byte
950 W=PEEK(PC)
```

```
960 PC=PC+1
970 PRINTRIGHT$("00"+HEX$(W),2);" ";
980 RETURN
990 REM Inicialización Variables
1000 DIM RT$(7),RS$(8),BI$(3),AL$(7),BF$(255)
1010 FOR I=0 TO 7:READ RT$(I):NEXT
1020 FOR I=0 TO 7:READ RS$(I):NEXT
1030 FOR I=1 TO 3:READ BI$(I):NEXT
1040 FOR I=0 TO 7:READ AL$(I):NEXT
1050 FOR I=0 TO &H7F:READ BF$(I):NEXT
1060 FOR I=&H80 TO &H9F:BF$(I)="":NEXT
1070 FOR I=&HA0 TO &HFF:READ BF$(I):NEXT
1080 GOTO 1400
1090 REM Tabla de comandos
1100 DATA B,C,D,E,H,L,(HL),A
1110 DATA RLC,RRC,RL,RR,SLA,SRA,???,SRL
1120 DATA BIT,RES,SET
1130 DATA ADD,ADC,SUB,SBC,AND,XOR,OR,CP
1140 DATA NOP,"LD BC,nn","LD (BC),A",INC BC,INC B,DEC B,"LD
 B,n",RLCA
1150 DATA "EX AF,AF'","ADD HL,BC","LD A,BC",DEC BC,INC C,DE
C C,"LD C,n",RRCA
1160 DATA DJNZ e,"LD DE,nn","LD (DE),A",INC DE,INC D,DEC D,
"LD D,n",RLA
1170 DATA JR e,"ADD HL,DE","LD A,(DE)",DEC DE,INC E,DEC E,"
LD E,n",RRA
1180 DATA "JR NZ,e","LD HL,nn","LD (nn),HL",INC HL,INC H,DE
C H,"LD H,n",DAA
1190 DATA "JR Z,e","ADD HL,HL","LD HL,(nn)",DEC HL,INC L,DE
C L,"LD L,n",CPL
1200 DATA "JR NC,e","LD SP,nn","LD (nn),A",INC SP,INC (HL),
DEC (HL),"LD (HL),n",SCF
1210 DATA "JR C,e","ADD HL,SP","LD A,(nn)",DEC SP,INC A,DEC
 A,"LD A,n",CCF
1220 DATA "IN B,(C)","OUT (C),B","SBC HL,BC","LD (nn),BC",N
EG,RETN,IM 0,"LD I,A"
```

```
1230 DATA "IN C,(C)","OUT (C),C","ADC HL,BC","LD BC,(nn)",,
RETI,,"LD R,A"
1240 DATA "IN D,(C)","OUT (C),D","SBC HL,DE","LD (nn),DE",,
,IM 1,"LD A,I"
1250 DATA "IN E,(C)","OUT (C),E","ADC HL,DE","LD DE,(nn)",,
,IM 2,"LD A,R"
1260 DATA "IN H,(C)","OUT (C),H","SBC HL,HL","LD (nn),HL",,
,,RRD
1270 DATA "IN L,(C)","OUT (C),L","ADC HL,HL","LD HL,(nn)",,
,,RLF
1280 DATA ,,"SBC HL,SP","LD (nn),SP",,,,
1290 DATA "IN A,(C)","OUT (C),A","ADC HL,SP","LD SP,(nn)",,
,,
1300 DATA LDI,CPI,INI,OUTI,,,,,LDD,CPD,IND,OUTD,,,,
1310 DATA LDIR,CPIR,INIR,OTIR,,,,,LDDR,CPDR,INDR,OTDR,,,,
1320 DATA RET NZ,POP BC,"JP NZ,nn",JP nn,"CALL NZ,nn",PUSH
BC,"ADD A,n",RST &H00
1330 DATA RET Z,RET,"JP Z,nn",->,"CALL Z,nn",CALL nn,"ADC A
,n",RST &H08
1340 DATA RET NC,POP DE,"JP NC,nn","OUT (n),A","CALL NC,nn"
,PUSH DE,"SUB n",RST &H10
1350 DATA RET C,EXX,"JP C,nn","IN A,(n)","CALL C,nn",->,"SB
C A,n",RST &H18
1360 DATA RET PO,POP HL,"JP PO,nn","EX (SP),HL","CALL PO,nn
",PUSH HL,"AND n",RST &H20
1370 DATA RET PE,JP (HL),"JP PE,nn","EX DE,HL","CALL PE,nn"
,->,"XOR n",RST &H28
1380 DATA RET P,POP AF,"JP P,nn",DI,"CALL P,nn",PUSH AF,"OR
 n",RST &H30
1390 DATA RET M,"LD SP,HL","JP M,nn",EI,"CALL M,nn",->,"CP
n",RST &H38
1400 ON STOP GOSUB 295
1410 STOP ON
1420 ON ERROR GOTO 295
1430 POKE &HFEE5,&H70:POKE &HFEE6,&HF3
1440 FOR I=&HF370 TO &HF37B:READ A:POKE I,A:NEXT
```

```
1450 GOTO 40
1460 DATA &HF5,&H3A,&H61,&HF6
1470 DATA &H32,&H15,&HF4,&HF1
1480 DATA &HCD,&H63,&H1B,&HC9
```

EXPLICACION:

Línea 10-130:

Menú: introducción de la dirección de inicio y de la dirección de final. Determinar si se desea salida a impresora o a monitor

En las líneas 140-290:

está el bucle principal del programa

Línea 150:

Aquí es proyectada la dirección actual

Línea 170:

leer y proyectar el siguiente Byte

Línea 180:

saltar al subprograma que ejecuta la interpretación

Línea 190:

cuando es activado IL (=-1), saltar para el tratamiento de comandos indexados

Línea 200:

Tratamiento de los comandos RST, que tienen datos a continuación

Línea 210:

Bifurcación, cuando el comando contiene números

Línea 220:

Bifurcación, cuando el comando contiene distancias relativas

Línea 230-270:

salida formateada

Línea 280:

Si aún no es final, volver al principio del bucle principal

Línea 295:

Fin del programa

En las líneas 300-530:

están los subprogramas para la interpretación

Línea 310:

Bifurcación para el tratamiento de comandos indexados

Línea 320:

Bifurcación para el tratamiento de comandos que comienzan con &HED

Línea 330:

Bifurcación para el tratamiento de comandos que comienzan con &HCB

Línea 340:

Saltar al subprograma que descompone W en C1

(Bit 6,7), C2 (Bit 5-3) y C3 (Bit 2-0)

Línea 350:

En C1=0 y C1=3 a la línea 360, es decir, leer los comandos de la tabla, sinó, línea 370: comandos de formato LD r,r', o bien, línea 390: comandos lógicos aritméticos de 8-Bits

Línea 360:

determinar PR$ a partir de la tabla

Línea 370/380:

Comandos de formato LD r,r' y STOP

Línea 390/400:

comandos Aritmético-lógicos

En las líneas 410-450:

tiene lugar el tratamiento de comandos que comienzan con el código &HCB

Línea 420:

leer el siguiente Byte

Línea 425:

leer otro Byte más en comandos Indexados

Línea 430:

descomponer en C1, C2, C3

Línea 440:

crear PR$ para comandos de rotación o de desplazamiento (C1=0) y para comandos de manipulación de Bits

En las líneas 460-480:

tiene lugar el tratamiento de los comandos que empiezan por el código &HED

Línea 470:

leer el siguiente Byte

Línea 480:

si el código es válido, obtener PR$ de la tabla (Línea 360)

En las líneas 490-530:

tiene lugar el primer tratamiento de los comandos indexados de la línea 310

Línea 500:

activar el Flag

Línea 510:

almacenar en I$ el registro ("IX" o bien "IY")

Línea 520:

leer el siguiente Byte

Línea 530:

comenzar de nuevo la Interpretación (a partir de la Línea 300)

Línea 540-590:

Subprograma: SUB descomponer el código, W es descompuesto en C1 (Bit7,6), C2 (Bit 5-3) y C3 (Bit 2-0). RG$ contiene el registro correspondiente a C3

Línea 600-690:

comandos Indexados segundo tratamiento (salto de la Línea 190). Comprobar si son admitidos los comandos Indexados. Si afirmativo, sustituir HL por I$. Si es necesario indicar la distancia, las líneas 680/690 leen la distancia

Línea 700-810:

Si PR$ contiene una "n", se coloca aquí un número para "n"

Línea 730-760:

sustituir números de 1 Byte ("n")

Línea 770-810:

sustituir números de 2 Bytes ("nn")

Línea 820-890:

sustituir Offset ("e")

Línea 850/860:

calcular Offset

Línea 870/880:

colocar Offset

Línea 900-930:

Subprograma: SUB Conversor Hex-Dec

Línea 940-980:

Subprograma: SUB leer y sacar el siguiente Byte

Línea 990-1080:

Inicialización: estructura de las tablas

Línea 1090-1390:

Líneas-DATA

Línea 1400-1480:

Inicialización de las rutinas STOP y ERROR y lectura de las rutinas de salida para Impresora y monitor a la vez

Lista de Variables

A- Devolución de SUB"Hex.-Dec." interpretación del valor de A$ como cifra hexadecimal.
A$- Entrada de una cifra Hex/ entrega a SUB"Hex.-Dec."
AD- Dirección del primer código del comando actual
AN- Traducción de la dirección de inicio
C1- Bit 7 y 6 del Byte a tratar
C2- Bit 5 hasta 3 del Byte a tratar
C3- Bit 2-0 del Byte a tratar
DI- Distancia en comandos indexados
DI$- Distancia de la cadena para salida
E$- Entrada de la cadena (s/n)
EN- Traducción de la dirección final
I$- Contiene el actual registro índice
IL- Activado, cuando hay direccionamiento indexado, sinó, desactivado
LB- Almacenamiento intermedio del Low Byte en cifras de 2-Bytes
PC- El indicador de programa indica hacia las direcciones de la posición actual
PO- Posición de n, nn, e, HL..., en PR$
PR$- PR$ contiene el comando ensamblador traducido
RG$- Registro: devolución del subprograma, descomponer el código, RG$ contiene el registro correspondiente a C3
W- código leído
WE- Valor de una cifra de 2-Bytes ("nn")

Tablas

RT$(7) - Registro
RS$(7) - Comandos de rotación y desplazamiento
BF$(3) - Comandos de manipulación de Bits
AL$(7) - Comandos aritméticos o lógicos
BF$(255)- 0 hasta &H3F: Comandos que tienen como

Byte 1, el número del contador de las variables del campo
&H40-&HBF: Comandos que comienzan con &HED y que tienen como Byte 2, el número del contador de las variables &HBF-&HFF: Comandos que tienen como Byte 1, el número del contador de las variables

5.2 RUTINAS DEL SISTEMA

Una de las rutinas del sistema más importantes, es la que sirve para mostrar un signo en pantalla. Puede ser llamada con RST &H18. Esta rutina muestra el signo correspondiente al valor contenido en el acumulador.
Escriba un programa que muestre el juego de caracteres de los ordenadores MSX (códigos 32 a 255).

Solución:

```
10 ' ORG &HF000
20 ' LD B,223 ; Contador=255-32
30 ' LD A,32
40 ' proyectar BUCLE RST &H18 ; CHR$(A)
50 ' INC A
60 ' DJNZ BUCLE
70 ' RET
80 ' END
```

En relación con la proyección de signos, el ensamblador posee el pseudocomando DM. El comando DM va seguido de una palabra, entre comillas, que actúa como operando. Los códigos ASCII de las letras de la palabra son depositados mediante DM a partir de la dirección actual. Observe el programa siguiente:

```
10 ' ORG &HF000
20 ' LD HL,PALABR ; direción de la palabra a proyectar
30 ' BUCLE LD A,(HL) ; cargar en el acumulador con el código
ASCII de la letra correspondiente
40 ' INC HL ; colocar el indicador en la siguiente letra
50 ' RST &H18
60 ' OR A ; colocar los Flags
70 ' JR NZ,BUCLE ; aún no 0, entonces, letra siguiente
80 ' RET
90 ' PALABR DM "COMPUTER"
100 ' DB 0
110 ' END
```

El Byte a ceros binarios producido por DB 0 al final de la palabra (Línea 100), sirve para reconocer el final de la palabra a proyectar
Con ayuda de las rutinas Print (para la impresión), escribiremos ahora un programa monitor.

Rutina Monitor

Ya conocemos el método de trabajo del ensamblador. Pero aún existen algunos otros medios de ayuda para mejorar el trabajo con el lenguaje máquina. El denominado Monitor, entre otros, forma parte de ellos.
Aquí, naturalmente, no se trata del monitor (pantalla) de su ordenador, sinó de un programa, con el que usted puede, p.ej., comprobar el contenido de la memoria (ingl. to monitor: verificar), o modificarlo. Un cómodo programa monitor ofrece también la posibilidad de almacenar, cargar o iniciar programas en código máquina. A continuación, escribiremos un programa de este tipo en código máquina.
De este modo, ''matamos dos pájaros de un tiro'':
Usted aprende otras técnicas de programación y como resultado, obtiene un sencillo programa monitor.

Como ya hemos mencionado, la misión fundamental de un programa monitor consiste en mostrar el contenido de la memoria. En BASIC, ésto se puede realizar con >PEEK<.
Escriba un programa que, al introducir la dirección de inicio (V) y la dirección final (V), muestre los contenidos de memoria existentes entre ambas direcciones. Para la salida, utilice el formato habitual para un Hex-Dump (volcado de memoria en forma hexadecimal), es decir:

Volcado de memoria de la dirección &00 hasta &30:

```
0000 F3 C3 D7 02 8F 1B 98 98 sCW.?...
0008 C3 83 26 00 C3 B6 01 00 C.&.C6..
0010 C3 86 26 00 C3 D1 01 00 C.& CQ..
0018 C3 45 1B 00 C3 17 02 00 CE..C...
0020 C3 6A 14 00 C3 5E 02 00 Cj..C^..
0028 C3 89 26 A1 13 00 00 00 C.&!....
0030 C3 05 02 00 00 00 00 00 C.......
```

Su programa debería producir la misma imagen que nuestro programa. Es imprescindible que los códigos estén correctamente situados en su sucesión.
Observe, que la expresión del código ASCII viene dada a la derecha del verdadero volcado de memoria. Los códigos superiores a 127 son reducidos antes en 128. Los códigos no representados (0-31), han sido sustituidos por puntos.

Solución:

```
10 ON STOP GOSUB 170:STOP ON
20 ON KEY GOSUB 160:KEY (1) ON
30 KEY OFF:SCREEN 0
40 INPUT"Inicio &H";A$
50 ST=VAL("&H"+A$)
60 INPUT "Final &H";A$
70 EN=VAL("&H+A$)
80 FOR I=ST TO EN STEP 8:A$=""
90 PRINT RIGHT$("000"+HEX$(I),4);" ";
100 FOR J=0 to 7:W=PEEK(I+J)
110 WP=W AND &H7F
120 IF WP<32 OR WP=127 THEN WP=46
130 A$=A$+CHR$(WP)
140 PRINT RIGHT$("0"+HEX$(W),2);" ";
150 NEXT:PRINT A$;:NEXT:PRINT:GOTO 40
160 J=7:I=EN:RETURN
170 KEY ON:CLS:END
```

Con este programa usted puede verificar las memorias ROM y RAM del ordenador. Introduzca la siguiente línea en su programa monitor: 1 REM Esta es la primera línea.

Comprobemos el contenido de la memoria desde &H8000 hasta &H8030. En la representación ASCII de los contenidos de la memoria, reconocemos la primera línea, es decir, aparece de nuevo el comentario ''Esta es la primera línea''. A partir de &H8000 son depositados los programas BASIC en la memoria. Directamente a continuación del programa BASIC hay una página administrada internamente con todas las variables utilizadas en el programa, donde el valor de las variables numéricas está alamcenado directamente, y la dirección y la longitud de las variables en cadena . Allí, las variables han sido depositadas siguiendo su orden de aparición en el programa.
Ahora, vamos a escribir un programa en código máquina que ejecute para nosotros la misma operación (e incluso más).

Puesto que nosotros sacamos los contenidos en números hexadecimales, necesitaremos primero un subprograma que saque un Byte como número hexadecimal. Este Byte es transferido al acumulador.

Ejemplo: A= 63 = &H3F = &B 0011 1111

F corresponde a los 4 Bits inferiores (High Nibble).
3 corresponde a los 4 Bits superiores (Low Nibble).

Primero, es mostrado el High Nibble. Para ello, desplazaremos el contenido del Acumulador 4 veces hacia la derecha (Rotación de 8-Bits). El resultado de este desplazamiento es &B 1111 0011. A continuación, son borrados con AND los 4 Bits superiores. Finalmente, el acumulador contiene el valor &B 0000 0011=3. Este valor (3) es el que deberá ser mostrado.
El código ASCII de 3 es 51. Para obtener el valor 51 en el acumulador, deberemos sumar 48 al contenido del Acu (=3). Después, se llama a la rutina-PRINT (RST &H18). Para mostrar el Low Nibble, borraremos los 4 Bits superiores del antiguo contenido del acumulador. Tras la suma de 48 obtenemos 63.

Queremos proyectar F (&HF=15). El código ASCII de F es 70. Esto significa, que cuando el número hexadecimal a sacar es mayor que 9 (o sea, que se expresa en forma de letra), habrá que sumar además 7 antes de llamar a la rutina.
Intente escribir la rutina para la salida de un Byte en forma hexadecimal.

Listado del Ensamblador de SALHEX (salida hexadecimal)

```
F000           10         ORG  &HF000
F000           30 ; SALHEX
F000           40 ; muestra el acumulador en forma hexadec.
F000           50 ; registro E es modificado
F000 5F        60  SALHEX LD   E,A ; guardar acumulador
F001 0F        70         RRCA  ;rotación
F002 0F        80         RRCA  ;del acumulador
F003 0F        90         RRCA  ;4 Bits a
F004 0F       100         RRCA  ;la derecha
F005 E60F     110         AND  &B1111 ; borrar Bits 4-7
F007 CD0000   120         CALL CONV ; mostrar High Nibble
F00A 7B       130         LD   A,E ; conten. antig. del Acu
F00B E60F     140         AND  &B1111 ; borrar Bits 4-7
F00D CD0000   150         CALL CONV ; mostrar Low Nibble
F010 C9       160         RET ; final SALHEX
F011          170 ; Rutina CONV
F011                  180 ; muestra la correspondiente cifra
hexadecimal del contenido del acumulador.
**** Línea 120  : CONV=&HF011
**** Línea 150  : CONV=&HF011
F011 FE0A     190  CONV   CP   &HA ; valor de la cifra <10
F013 38FE     200         JR   C,NUMERO ;si, entonces sumar
F015 C607     210         ADD  A,7 ; sumar 7 para letras
**** Línea 200  : NUMERO=&HF017
F017  C630       220  NUMERO  ADD A,48 ; =código ASCII de la
cifra-hexadecimal
```

```
F019 CD5ABB  230          RST &H18
F01C C9      240          RET   ; Fin CONV

Programa: SALHEX
Inicio : &F000    Fin : &F01C
Longitud : 001D
Errores: 0
Tabla de variables:
SALHEX A000  CONV   A011  NUMERO   A017
```

Pasemos a la verdadera rutina monitor:

La dirección de inicio y de final viene dada a partir del BASIC.

```
10 ' ORG &HF000
20 ' INICIO DS 2
30 ' FINAL DS 2
```

Ahora, carguemos HL con la dirección de partida y mostrémosla:

```
40 ' LD HL,(INICIO) ; Indicador
50 ' WEI16 LD A,H ; mostrar High Byte
60 ' CALL SALHEX
70 ' LD A,L ; mostrar Low Byte
80 ' CALL SALHEX
```

Después deberá aparecer un espacio en blanco:

```
90 ' LD A,&20 ; ASCII del espacio en blanco:
100 ' RST &H18
```

A continuación, serán mostrados los valores de las 8 siguientes posiciones de memoria:

```
110 ' LD B,8 ; contador
120 ' WEI LD A,(HL) ; cargar el Byte en el acumulador
```

```
130 ' CALL SALHEX ; y mostrar
140 ' LD A,&H20 ; mostrar espacio
150 ' RST &H18  ; en blanco
160 ' INC HL
170 ' DJNZ WEI
```

Acto seguido, aparece un espacio en blanco y son mostrados los últimos 8 Bytes como signos-ASCII. De los códigos mayores que 127, se resta 128 (se retira el Bit 7). Para los códigos menores que 32 (signo de control), es mostrado un punto (ASCII= 46).

```
180 ' LD A,&H20
190 ' RST &H18 ; espacio en blanco
200 ' LD DE,8
210 ' OR A ; Carry = 0
220 ' SBC HL,DE ; decrementar indicador a 8
230 ' LD B,8
240 ' WEIAS LD A,(HL) ; cargar el acumulador con el Byte
250 ' INC HL ; incrementar el indicador
260 ' RES 7,A ; convertir signo gráfico en ASCII
270 ' CP &H20 ; mayor que 32 ??
280 ' JR NC,PR ; sí, entonces proyectar
290 ' CP 127 ; =DEL
300 ' JR NC,PR
310 ' LD A,46 ; ASCII para punto
320 ' PR RST &H18 ; proyectar signo
330 ' DJNZ WEIAS
```

Para llegar al comienzo de la siguiente línea, son enviados los códigos 13 y 10:
(CHR$(13) = Carriage Return)
(CHR$(10) = Line Feed-avance de líneas)

```
340 ' LD A,13 ; Carriage Return
350 ' RST &H18 ; emitir
360 ' LD A,10 ; avance de línea
370 ' RST &H18 ; emitir
```

Ahora, se comprueba si ya se ha llegado al final:

```
380 ' PUSH HL ; guardar Indicador
390 ' LD DE,(FINAL)
400 ' OR A ; Carry = 0
410 ' SBC HL,DE ; Indicador-Final<=0
420 ' POP HL ; coger Indicador (¡¡no influencia los Flags!!)
430 ' JR C,WEI16 ; HL-DE<0, entonces, continuar
440 ' JR Z,WEI16 ; HL-DE=0, entonces, continuar
450 ' RET ; HL-DE>0, entonces, final
460 ' ;Final Monitor
```

Ahora, aún ha de ser añadida la rutina Salhex, y nuestro programa ya puede funcionar:

```
470 ' ;Salhex
480 ' ; muestra el acumulador en forma hexadecimal
490 ' ; modificación del registro E
500 ' SALHEX LD E,A ; guardar acumulador
510 ' RRCA ; rotación
520 ' RRCA ; del acumulador
530 ' RRCA ; 4 Bits a
540 ' RRCA ; la derecha
550 ' AND &B1111 ; borrar Bits 4-7
560 ' CALL CONV ; mostrar High Nibble
570 ' LD A,E ; antiguo contenido acumulador
580 ' AND &B1111 ; borrar Bit 4-7
590 ' CALL CONV ; mostrar Low Nibble
600 ' RET ; Final Salhex
610 ' ; Rutina Conv
620 ' ; muestra la correspondiente cifra del contenido del
acumulador
630 ' CONV CP &HA ; valor de la cifra <10
```

```
640 ' JR C,NUM ;sí, entonces ir a NUM
650 ' ADD A,7 ; sumar 7 para letras
660 ' NUM ADD A,48 ; =código ASCII de la cifra hexadecimal
670 ' RST &H18
680 ' RET ; Final Conv
690 ' END
```

5.3 EL SIMULADOR A PASO

En este capítulo le presentaremos otro importante medio de ayuda para la realización de programas en código máquina: el simulador a paso.
Una de las dificultades que presenta la programación en el lenguaje ensamblador radica en el hecho de que no existe ninguna posibilidad de verificar fácilmente los programas creados. Un error que, en según que casos, en BASIC sólo hubiera originado un "Syntax Error" o similar, en un programa en lenguje máquina, casi siempre, provoca un bloqueo del sistema operativo. De este modo, se prolonga demasiado el tiempo de desarrollo para un programa, y la búsqueda de un error resulta dificultosa. Para encontrar el error en el programa, sería útil ejecutar el programa en lenguaje máquina paso a paso, rectificando para su corrección, los contenidos de los respectivos registros. Sin el simulador, éste pesado trabajo sólo es realizable a mano. El programa simulador que le ofrecemos a continuación, realizará por usted ese trabajo penoso. Tras iniciar el programa, usted podrá elegir entre salida a monitor, o a impresora y monitor. También es posible dejar esta decisión para después. Finalmente, se comienza con la simulación a partir de la dirección 0.

```
SZ H PNC A  B C  D E  H L  I X  I Y
00000000 00 0000 0000 0000 0000 0000
00000000 00 0000 0000 0000 SP : F290
000 F3         DI
```

En este lugar, el cursor indica que se espera la entrada de datos. Se preveen las siguientes entradas:

ESC
SELECT
RETURN
"d" o bien, "D"
CAP

Al pulsar la tecla ESC, usted dará por finalizado el programa.
Al pulsar la tecla RETURN se continúa la simulación. Aparecen entonces los contenidos eventualmente modificados de los registros, y el próximo comando:

```
SZ H PNC A  B C  H L  I X  I Y
00000000 00 0000 0000 0000 0000
00000000 00 0000 0000 SP : F290
0001 C3D702     JP   &H02D7
```

La tecla "I" o "i" efectúa la pregunta de si la salida ha de ser a impresora.
La tecla SELECT tiene una función importante:
Mediante SELECT, el programa pasa al modo Editor. En este modo, usted tiene la posibilidad de modificar a su gusto los contenidos de los registros, así como la dirección del programa. Para ello, será imprescindible que siga uno a uno los siguientes pasos:

Una vez pulsada la tecla de función SELECT, el cursor se colocará directamente debajo del S (=Signal Flag) de la última salida de registro. Los contenidos mostrados en esta línea superior corresponden al juego de registros utilizado en ese momento. Usted podrá moverse libremente por dentro de esa línea con las teclas del cursor o con CTRL-B y CTRL-F, e introducir los contenidos de los registros deseados en los puntos previamente dados en forma hexadecimal (se puede escribir en mayúsculas y en minúsculas). La introducción para el registro de Flag deberá ser naturalmente en forma binaria, es decir, deberá ser realizada con 0 y 1. Una última advertencia:
Al inicio de la línea, se halla el registro Flag en número binario. Su significado está encima de su respectivo Bit:

S- Flag de signo
Z- Flag de cero
H- Flag de media transferencia
P- P/V como Flag de paridad o de desbordamiento
N- Flag de sustracción BCD
C- Flag de acarreo

A continuación están el acumulador A (8-Bits) y los registros universales B hasta L, agrupados como registros de 16-Bit. Al final de esta línea están los registros IX y IY. Una vez concluida la introducción de esta línea, usted deberá pasar (¡IMPORTANTE!) mediante RETURN a la siguiente línea. Para ello, no utilice por favor las teclas del cursor u otras. La consecuencia de una manipulación de error sería que los registros no serían cargados con los valores deseados.
Tras finalizar con RETURN la modificación de la línea superior (=serie de registros actual), podrá proceder de igual modo con la segunda línea (=segunda serie de registros, no utilizada de momento).

SI no desea efectuar ninguna modificación, podra pulsar RETURN en cuanto lo desee. La estructura de la segunda línea es análoga a la de la primera. De cualquier modo, los registros índice sólo están presentes una sola vez en el Z80. En su lugar, sin embargo, se halla el indicador de la pila SP en la segunda línea. Procure que en la simulación, el SP nunca sea menor que &HF259. De lo contrario, el programa en lenguaje máquina, que lleva a cabo la simulación, sería destruido, lo cual originaría el bloqueo del ordenador (en ese caso, debería ser modificado el SP). Sin embargo, ello sólo debería ocurrir al principio de una simulación, ya que sinó, se descontrola la sucesión de las direcciones de retorno. SI ha concluido con esta línea, finalice de nuevo con RETURN (!), y el cursor aparecerá al comienzo de la línea que contiene el comando traducido. En esta línea, puede usted cambiar ahora el PC, es decir, las cuatro primeras cifras hexadecimales de la línea. La dirección que usted ha dado aquí se considerará, al pulsar de nuevo RETURN, como dirección de continuación de la simulación. Con ello finaliza el modo Editor, introducido mediante la tecla SELECT, y la simulación continúa.
La última función de servicio es la tecla CAP. En este programa carece de finalidad, es decir que su misión principal ya no consiste en el cambio de escritura mayúscula a minúscula.

Más bien, decide sobre la simulación real o pseudosimulación. Simulación real significa que todos los comandos son realmente ejecutados. Sin embargo, ello puede provocar consecuencias fatales en algunos comandos. En estos comandos "peligrosos" se engloban todos aquellos que modifican realmente la configuración del sistema, p.ej, LD (&HF3DB),A. Los contenidos modificados de los registros podrían cambiar importantes informaciones utilizadas por el interpretador de BASIC o el sistema operativo, y ocasionar así errores. También los comandos de E/S, en caso de ser "simulados realmente", pueden provocar el bloqueo del ordenador debido a que son responsables, entre otros, de la elección de las memorias ROM y RAM. Así mismo, se recomienda prudencia en los comandos que afectan directamente al SP.

Por este motivo, antes de que el comando sea ejecutado, usted puede decidir cada vez, mediante la tecla CAP, si desea o no realmente que se ejecute dicho comando. La luz encendida del diodo CAP significará entonces que todos los comandos son ejecutados (simulación real). Si usted apaga el diodo pulsando la tecla CAP, se anula la ejecución de los comandos.
Observemos como ejemplo la simulación del comando RST &H20 (ver desensamblador). Coloque el PC en &H15BB y conecte la simulación real. Compare su proyección con la que aparece aquí impresa.

```
15BB 110001    LD   DE,&H0100
SZ H PNC A  B C  D E  H L   IX   IY
00000000 00 0000 0000 0000 0000 0000
00000000 00 0000 0000 0000 SP : F290
15BB E7        RST  &H20
SZ H PNC A  B C  D E  H L   IX   IY
00000000 00 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
0020 C36A14    JP   &H146A
SZ H PNC A  B C  D E  H L   IX   IY
```

```
00000000 00 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
146A 7C        LD    A,H
SZ H PNC A  B C  D E  H L   IX   IY
00000000 00 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
146B 92        SUB   D
SZ H PNC A  B C  D E  H L   IX   IY
10111011 FF 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
146C C0        RET   NZ
SZ H PNC A  B C  D E  H L   IX   IY
10111011 FF 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F290
15BF 3804      JR    C,$6 >&H15C5
```

Primero se carga DE con &H100. Observe el cambio en la indicación debajo de DE. A continuación tiene lugar el salto RST &H20. Después de este comando, SP ha disminuido, pues la dirección de retorno (&H15BF) había sido puesta en la pila. Mediante JP &H146A se salta a la rutina de comparación. Allí se realiza la sustracción HL-DE. Puesto que en nuestro caso DE es mayor que HL, se efectúa un retorno mediante RET NZ. Como era de esperar, se ha activado el Flag de acarreo (=1), para demostrar que DE es mayor que HL.

Pruebe ahora la misma simulación a partir de la dirección &H15BB. Pero antes, cargue HL y DE con valores distintos (&H101, &H1000 &H80), y observe el efecto producido sobre los Flags. Por último, el comando RET o RET NZ finaliza la rutina. La dirección de retorno es recogida de la pila y la ejecución del programa se reanuda mediante el comando siguiente a RST &H20.

Si sus programas o comandos están poco claros, utilice el simulador para verificar su modo de funcionamiento.

Antes de iniciar la explicación sistemática del simulador, queremos mencionar algunos datos sobre su modo de funcionamiento.

El núcleo del simulador está compuesto por un programa en lenguaje máquina, que es cargado al final del programa. Hacia la mitad del programa, por así decirlo, es introducido mediante POKE el comando a ser ejecutado (Línea 1850). Primero son cargados los registros con los respectivos valores. Tras la ejecución del comando se escriben de nuevo en la memoria los contenidos, modificados en su caso, de los registros, para ser pasados al BASIC. El simulador llama a este programa mediante >X=USR8(1)< en la línea 455. Para comprender mejor todo el proceso, le presentamos en la página siguiente el listado ensamblador del programa en lenguaje máquina.

Basándose en el listado, piense cuál sería el modo de funcionamiento del programa. Para ello, seguramente necesitará practicar un poco de "acrobacias con la pila". Si desea efectuar la simulación, desconecte la simulación real de los comandos que influyen sobre el SP (Indicador de la pila), y realice la modificación necesaria pulsando la tecla SELECT. Con el fin de que el programa sea lo más corto posible, los contenidos de los registros no serán escritos en la memoria con los comandos LD, sinó que serán leídos y escritos mediante los comandos PUSH y POP. Para ello, se colocará antes el SP al principio del espacio reservado para el traspaso (SPUELO-SP traspaso Low Adress).
Con ayuda de este método pueden ser ejecutados todos los comandos, hasta incluso los comandos para saltos. En un comando de salto, tendría lugar una bifurcación a partir, del programa en lenguaje máquina, y el retorno al BASIC sería imposible.
Debido a ello, los comandos de saltos son tratados por completo en BASIC. Las manipulaciones necesarias de la pila y del PC son llevadas a cabo a partir del BASIC. Observe al escribir, que la última parte del simulador del ya conocido

desensamblador, presenta ligeras variaciones.

```
 F000              10                ; Simulador
 F200              20        ORG  &HF200
?F200 ED730000     30        LD   (SPREAL),SP ; salvar el
 verdadero SP
?F204 310000       40        LD   SP,SPUELO ; SP para traspa-
 so de registros
 F207 F1           50        POP AF
 F208 C1           60        POP BC
 F209 D1           70        POP DE
 F20A E1           80        POP HL
 F20B D9           90        EXX
 F20C 08          100        EX  AF,AF'
 F20D DDE1        110        POP IX
 F20F FDE1        120        POP IY
 F211 F1          130        POP AF
 F212 C1          140        POP BC
 F213 D1          150        POP DE
 F214 E1          160        POP HL
?F215 ED7B0000    170        LD  SP,(SPSIMU) ; coger simula-
 ción SP
 F219 0000        180 COMAND DW  0 ; Lugar para comando
 F21B 0000        190        DW  0 ; Lugar para comando
?F21D ED730000    200        LD  (SPSIMU),SP ; grabar simu-
 lación SP
?F221 310000      210        LD  Sp,SPSIMU
 F224 E5          230        PUSH HL
 F225 D5          240        PUSH DE
 F226 C5          250        PUSH BC
 F227 F5          260        PUSH AF
 F228 FDE5        270        PUSH IY
 F22A DDE5        280        PUSH IX
 F22C 08          290        EX   AF,AF'
 F22D D9          300        EXX
 F22E E5          310        PUSH HL
 F22F D5          320        PUSH DE
 F230 C5          330        PUSH BC
```

```
 F231 F5          340         PUSH AF
?F232 ED7B0000    350         LD   SP,(SPREAL) ; repetir el
 verdadero SP
 F236 C9          360         RET
**** Línea  30  : SPREAL=&HF237
**** Línea  350 : SPREAL=&HF237
 F237             370  SPREAL DS   2
**** Línea  40  : SPUELO=&H239
 F239             380  SPUELO DS   20
**** Línea  170 : SPSIMU=&HF24D
**** Línea  200 : SPSIMU=&HF24D
**** Línea  210 : SPSIMU=&HF24D
?F24D 0000        390  SPSIMU DW   PILA
 F24F             400          DS   4 ; Lugar para desborda-
 miento de la pila del simulador
**** Línea  390 : PILA=&HF253
 F253             410  PILA   DS   50 ; lugar siguiente para
 la pila
------------------------------------------

Programa:SIMULA
Inicio : &HF200     Final : &HF284
Longitud : &H85 Bytes
Errores : 0
Tabla de Variables:
COMAND F219   SPREAL F237
SPUELO  F239   SPSIMU F24D
PILA    F253
```

```
10 CLEAR &H200,&HEFFF:MAXFILES=0
20 SCREEN 0:COLOR 1,14,14:KEY OFF:WIDTH 37
30 LOCATE 0,3:PRINT"Z 8 0  S I M U L A D O R  -  M S X" ' d
e Holger Dullin 1985
40 GOTO 1890
50 LOCATE 3,7
60 INPUT"Impesora (s/n)";E$:GOTO 460
70 REM Bucle principal **************
80 IF E$="s" OR E$="S" THEN POKE &HFEE4,&HC3
90 FOR I=S4 TO S4+3:POKE I,0:NEXT:QQ=0
100 GOSUB 1020:REM Desenmblar
110 POKE &HFEE4,&HC9:LOCATE ,,1
120 A$=INKEY$:IF A$="" THEN 120 ELSE LOCATE ,,0
130 IF A$=CHR$(27) THEN 2520:REM ESC para Final
140 IF A$=CHR$(24) THEN 570:REM SELECT para Modificaciones
150 IF A$="i" OR A$="I" THEN 60
160 IF A$<>CHR$(13) THEN 120:REM RETURN para Continuación
170 IF PEEK(&HFCAB)<>255 THEN 460:REM Simulación real sólo
cuando CAP conectado
180 IF PO=0 THEN B$=PR$:GOTO 210
190 B$=LEFT$(LEFT$(PR$,PO-1)+"    ",4)
200 REM Comandos de salto tratados separadamente
210 IF B$="JP  " OR B$="JR  " THEN GOSUB 790:GOTO 460
220 IF B$<>"RST " THEN 240
230 A$=PC:PC=VAL(MID$(PR$,5,4)):GOTO 310
240 IF B$<>"DJNZ" THEN 280
250 B=PEEK(S1+15):B=B-1-256*(B=0):POKE S1+15,B:IF B<>0 THEN
 GOSUB 870:GOTO 460 ELSE 460
260 IF B<>0 THEN GOSUB 870
270 GOTO 460
280 IF B$<>"CALL" THEN 370
290 A$=PC:GOSUB 790
300 REM colocar en la pila la dirección de retorno
310 SP=PEEK(S3)+256*PEEK(S3+1)
320 SP=SP-1:POKE SP,INT(A$/256)-256*(A$<0)
330 SP=SP-1:POKE SP,A$-256*INT(A$/256)
```

```
340 POKE S3,SP-INT(SP/256)*256
350 POKE S3+1,INT(SP/256)-256*(SP<0)
360 GOTO 460
370 IF LEFT$(B$,3)<>"RET" THEN 455
380 GOSUB 720
390 IF FL=0 THEN 460
400 REM coger de la pila la dirección de retorno
410 SP=PEEK(S3)+256*PEEK(S3+1)
420 PC=PEEK(SP+1)*256+PEEK(SP)
430 SP=SP+2
440 POKE S3,SP-INT(SP/256)*256
450 POKE S3+1,INT(SP/256)-256*(SP<0)
451 GOTO 460
455 IF PEEK(&HFCAB)=255 THEN X=USR8(1):REM Simulación real
cuando CAP conect.
460 REM Salida registros
470 IF E$="s" OR E$="S" THEN POKE &HFEE4,&HC3
480 PRINT"SZ H PNC A  B C  D E  H L   IX   IY "
490 FOR Q=1 TO 0 STEP -1
500 PRINTRIGHT$("0000000"+BIN$(PEEK(S1+Q*12)),8);
510 PRINT" ";RIGHT$("0"+HEX$(PEEK(S1+1+Q*12)),2);
520 FOR K=2 TO 6 STEP 2
530 PO=K+Q*12:GOSUB 890:NEXT
540 IF Q=1 THEN PO=8:GOSUB 890:PO=10:GOSUB 890 ELSE PRINT"
SP :";:PO=20:GOSUB 890
550 PRINT:NEXT Q
560 GOTO 80
570 REM cambiar Registros SELECT
580 LOCATE 0,CSRLIN-3
590 LINE INPUT Z$
600 Q=12:GOSUB 920
610 Q=0:PO=28
620 GOSUB 970
630 PO=33:GOSUB 970
640 LINE INPUT Z$
650 Q=0:GOSUB 920
```

```
660 Q=10:PO=33:GOSUB 970
670 LINE INPUT Z$
680 PC=VAL("&H"+LEFT$(Z$,4))
690 LOCATE 0,CSRLIN-1
700 GOTO 80
710 REM Subprogramas **********************
720 REM SUB Comprobación de condiciónOk
730 A$=MID$(PR$,PO+1,2)
740 FOR I=0 TO 7:IF A$<>CO$(I) THEN NEXT
750 IF I=8 THEN FL=1:RETURN
760 BI=((I\2)*2-(I>3)-((I=4)OR(I=5)))
770 IF (PEEK(S1+12)AND2^BI)/2^BI=-(IMOD2=1) THEN FL=-1 ELSE
 FL=0
780 RETURN
790 REM SUB Activar PC en los saltos
800 GOSUB 720
810 IF FL THEN 870
820 IF FL=0 THEN RETURN
830 A$=MID$(PR$,PO+2,2)
840 IF A$="HL" THEN PC=PEEK(S1+18)+256*PEEK(S1+19):RETURN
850 IF A$="IX" THEN PC=PEEK(S1+8)+256*PEEK(S1+9):RETURN
860 IF A$="IY" THEN PC=PEEK(S1+10)+256*PEEK(S1+11):RETURN
870 PC=VAL(RIGHT$(PR$,6)):RETURN
880 REM SUB Salida Registro 16 Bits
890 PRINT" ";RIGHT$("000"+HEX$(256*PEEK(S1+PO+1)+PEEK(S1+PO
)),4);
900 RETURN
910 REM SUB pasar los registros modificados al programa en
lenguaje máquina
920 POKE S1+1+Q,VAL("&H"+MID$(Z$,10,2))
930 POKE S1+Q,VAL("&B"+MID$(Z$,1,8))
940 FOR PO=13 TO 23 STEP 5:GOSUB 970:NEXT
950 RETURN
960 REM SUB A partir de la entrada, determinar los valores
de los registros tras la modificación
```

```
970 WE=VAL("&H"+MID$(Z$,PO,4))
980 POKE S1+Q+(PO\5)*2-1,INT(WDSKO$/256)-256*(WE<0)
990 POKE S1+Q+(PO\5)*2-2,WE-INT(WE/256)*256
1000 RETURN
1010 REM Desensamblador **********************
1020 AD=PC
1030 PRINTRIGHT$("000"+HEX$(AD),4);" ";
1040 IL=0
1050 GOSUB 1830
1060 GOSUB 1170
1070 IF IL THEN 1480
1080 IF LEFT$(PR$,3)="RST" AND (W=&HCF) THEN PR$=PR$+" /DB:
n"
1090 IF INSTR(PR$,"n")<>0 THEN 1590
1100 IF INSTR(PR$,"e")<>0 THEN 1710
1110 PO=INSTR(PR$," ")
1120 IF PR$="" THEN PR$="???"
1130 IF PO=0 THEN PRINTTAB(15);PR$;:GOTO 1150
1140 PRINTTAB(15);LEFT$(PR$,PO-1);TAB(20);RIGHT$(PR$,LEN(PR
$)-PO);
1150 PRINT SPACE$(36-POS(0))
1160 RETURN
1170 REM Interpretar
1180 IF (W=&HDD OR W=&HFD) AND NOT IL THEN 1370
1190 IF W=&HED THEN 1340
1200 IF W=&HCB THEN 1280
1210 GOSUB 1420
1220 ON C1 GOTO 1240,1260,1230
1230 PR$=BF$(W):RETURN
1240 IF W=&H76 THEN PR$="ALTO":RETURN
1250 PR$="LD "+RI$(C2)+","+RG$:RETURN
1260 IF C2=0 OR C2=1 OR C2=3 THEN A$=" A," ELSE A$=" "
1270 PR$=AL$(C2)+A$+RG$:RETURN
1280 REM Comandos con inicio &HCB
1290 GOSUB 1830
1300 IF IL THEN DI=W:GOSUB 1830
```

```
1310 GOSUB 1420
1320 IF C1=0 THEN PR$=RS$(C2)+" "+RG$ ELSE PR$=BI$(C1)+STR$
(C2)+","+RG$
1330 RETURN
1340 REM Comandos con inicio &HED
1350 GOSUB 1830
1360 IF W<&H40 OR W>&HBF THEN PR$="???":RETURN ELSE GOTO 12
30
1370 REM Comandos indexados
1380 IL=-1
1390 IF W=&HDD THEN I$="IX" ELSE I$="IY"
1400 GOSUB 1830
1410 GOTO 1170
1420 REM descomponer código
1430 C1=(W AND &B11000000)/64
1440 C2=(W AND &B111000)/8
1450 C3=W AND &B111
1460 RG$=RT$(C3)
1470 RETURN
1480 REM indexado
1490 PO=INSTR(PR$,"HL")
1500 IF PO=0 THEN PR$="???":GOTO 1110
1510 IF INSTR(PR$,"(HL)")<>0 THEN 1550
1520 IF PR$="EX DE,HL" THEN PR$="???":GOTO 1110
1530 IF PR$="ADD HL,HL" THEN PR$="ADD "+I$+","+I$:GOTO 1110
1540 PR$=LEFT$(PR$,PO-1)+I$+RIGHT$(PR$,LEN(PR$)-PO-1):GOTO
1080
1550 IF LEFT$(PR$,2)="JP" THEN 1540
1560 IF PC-ADR<3 THEN GOSUB 1830:DI=W
1570 IF DI>127 THEN DI$=STR$(DI-256) ELSE DI$="+"+RIGHT$(ST
R$(DI),LEN(STR$(DI))-1)
1580 I$=I$+DI$:GOTO 1540
1590 REM sustituir n
1600 PO=INSTR(PR$,"nn")
1610 IF PO<>0 THEN 1660
```

```
1620 PO=INSTR(PR$,"n")
1630 GOSUB 1830
1640 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("00"+HEX$(W),2)+RIGHT$
(PR$,LEN(PR$)-PO)
1650 GOTO 1110
1660 GOSUB 1830:LB=W
1670 GOSUB 1830
1680 WE=W*256+LB
1690 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("0000"+HEX$(WE),4)+RIG
HT$(PR$,LEN(PR$)-PO-1)
1700 GOTO 1110
1710 REM sustituir e
1720 PO=INSTR(PR$,"e")
1730 GOSUB 1830
1740 IF W>127 THEN W=W-256:REM Complemento a dos
1750 W=W+2
1760 A$="$"+STR$(W)+" >"+"&H"+RIGHT$("0000"+HEX$(PC+W-2),4)
1770 PR$=LEFT$(PR$,PO-1)+A$+RIGHT$(PR$,LEN(PR$)-PO)
1780 GOTO 1110
1790 REM Conversión hexadecimal -> decimal
1800 IF A$="" THEN A=0:RETURN
1810 A=VAL("&H"+A$)
1820 RETURN
1830 REM Leer Byte
1840 W=PEEK(PC)
1850 POKE S4+QQ,W:QQ=QQ+1
1860 PC=PC+1
1870 PRINTRIGHT$("00"+HEX$(W),2);
1880 RETURN
1890 REM Inicialización de las Variables *********
1900 DIM RT$(7),RS$(8),BI$(3),AL$(7),BF$(255)
1910 FOR I=0 TO 7:READ RT$(I):NEXT
1920 FOR I=0 TO 7:READ RS$(I):NEXT
1930 FOR I=1 TO 3:READ BI$(I):NEXT
1940 FOR I=0 TO 7:READ AL$(I):NEXT
```

```
1950 FOR I=0 TO &H7F:READ BF$(I):NEXT
1960 FOR I=&H80 TO &H9F:BF$(I)="":NEXT
1970 FOR I=&HA0 TO &HFF:READ BF$(I):NEXT
1980 GOTO 2300
1990 REM Tablas de comandos
2000 DATA B,C,D,E,H,L,(HL),A
2010 DATA RLC,RRC,RL,RR,SLA,SRA,???,SRL
2020 DATA BIT,RES,SET
2030 DATA ADD,ADC,SUB,SBC,AND,XOR,OR,CP
2040 DATA NOP,"LD BC,nn","LD (BC),A",INC BC,INC B,DEC B,"LD
 B,n",RLCA
2050 DATA "EX AF,AF'","ADD HL,BC","LD A,BC",DEC BC,INC C,DE
C C,"LD C,n",RRCA
2060 DATA DJNZ e,"LD DE,nn","LD (DE),A",INC DE,INC D,DEC D,
"LD D,n",RLA
2070 DATA JR e,"ADD HL,DE","LD A,(DE)",DEC DE,INC E,DEC E,"
LD E,n",RRA
2080 DATA "JR NZ,e","LD HL,nn","LD (nn),HL",INC HL,INC H,DE
C H,"LD H,n",DAA
2090 DATA "JR Z,e","ADD HL,HL","LD HL,(nn)",DEC HL,INC L,DE
C L,"LD L,n",CPL
2100 DATA "JR NC,e","LD SP,nn","LD (nn),A",INC SP,INC (HL),
DEC (HL),"LD (HL),n",SCF
2110 DATA "JR C,e","ADD HL,SP","LD A,(nn)",DEC SP,INC A,DEC
 A,"LD A,n",CCF
2120 DATA "IN B,(C)","OUT (C),B","SBC HL,BC","LD (nn),BC",N
EG,RETN,IM 0,"LD I,A"
2130 DATA "IN C,(C)","OUT (C),C","ADC HL,BC","LD BC,(nn)",,
RETI,,"LD R,A"
2140 DATA "IN D,(C)","OUT (C),D","SBC HL,DE","LD (nn),DE",,
,IM 1,"LD A,I"
2150 DATA "IN E,(C)","OUT (C),E","ADC HL,DE","LD DE,(nn)",,
,IM 2,"LD A,R"
2160 DATA "IN H,(C)","OUT (C),H","SBC HL,HL","LD (nn),HL",,
,,RRD
2170 DATA "IN L,(C)","OUT (C),L","ADC HL,HL","LD HL,(nn)",,
,,RLF
```

```
2180 DATA ,,"SBC HL,SP","LD (nn),SP",,,,
2190 DATA "IN A,(C)","OUT (C),A","ADC HL,SP","LD SP,(nn)",,
,,
2200 DATA LDI,CPI,INI,OUTI,,,,,LDD,CPD,IND,OUTD,,,,
2210 DATA LDIR,CPIR,INIR,OTIR,,,,,LDDR,CPDR,INDR,OTDR,,,,
2220 DATA RET NZ,POP BC,"JP NZ,nn",JP nn,"CALL NZ,nn",PUSH
BC,"ADD A,n",RST &H00
2230 DATA RET Z,RET,"JP Z,nn",->,"CALL Z,nn",CALL nn,"ADC A
,n",RST &H08
2240 DATA RET NC,POP DE,"JP NC,nn","OUT (n),A","CALL NC,nn"
,PUSH DE,"SUB n",RST &H10
2250 DATA RET C,EXX,"JP C,nn","IN A,(n)","CALL C,nn",->,"SB
C A,n",RST &H18
2260 DATA RET PO,POP HL,"JP PO,nn","EX (SP),HL","CALL PO,nn
",PUSH HL,"AND n",RST &H20
2270 DATA RET PE,JP (HL),"JP PE,nn","EX DE,HL","CALL PE,nn"
,->,"XOR n",RST &H28
2280 DATA RET P,POP AF,"JP P,nn",DI,"CALL P,nn",PUSH AF,"OR
 n",RST &H30
2290 DATA RET M,"LD SP,HL","JP M,nn",EI,"CALL M,nn",->,"CP
n",RST &H38
2300 ON STOP GOSUB 2520
2310 STOP ON
2320 ON ERROR GOTO 2520
2330 POKE &HFEE5,&H70:POKE &HFEE6,&HF3
2340 FOR I=&HF370 TO &HF37B:READ A$:POKE I,VAL("&H"+A$):NEX
T
2350 DATA F5,3A,61,F6,32,15,F4,F1
2360 DATA CD,63,1B,C9
2370 FOR I=0 TO 7:READ CO$(I):NEXT
2380 DATA NC,"C,",PO,PE,NZ,"Z,","P,","M,"
2390 FOR I=&HF200 TO &HF236:READ A$:POKE I,VAL("&H"+A$):NEX
T
2400 DATA ED,73,37,F2,31,39,F2,F1
```

```
2410 DATA C1,D1,E1,D9,08,DD,E1,FD
2420 DATA E1,F1,C1,D1,E1,ED,7B,4D
2430 DATA F2,00,00,00,00,ED,73,4D
2440 DATA F2,31,4D,F2,E5,D5,C5,F5
2450 DATA FD,E5,DD,E5,08,D9,E5,D5
2460 DATA C5,F5,ED,7B,37,F2,C9
2470 DEFUSR8=&HF200
2480 S1=&HF239:S2=&HF24C:S3=&HF24D:S4=&HF219
2490 FOR I=S1 TO S2:POKE I,0:NEXT
2500 POKE S3,&H90:POKE S3+1,&HF2
2510 GOTO 50
2520 POKE &HFEE4,&HC9:ON ERROR GOTO 0:LOCATE 0,24,0:END
```

Descripción del programa

Línea 10-60:

Inicialización

Línea 70-560:

Bucle principal

Línea 80:

Desconectar impresora, cuando E$="s"

Línea 90:

Borrar área de traspaso de comandos (es decir, comando NOP)

Línea 100:

Desensamblar comando, proyectar, introducir con POKE el código en el programa en lenguaje máquina

Línea 110:

Desconectar impresora

Línea 120-160:

Bucle de entrada con evaluación

Línea 170:

No ejecutar comando cuando CAP esté desconectado

Línea 180-190:

Palabra de comando a B$

Línea 200-450:

Simular comandos de salto desde el BASIC

Línea 210:

Comando JP/JR

Línea 230:

Comandos RST

Línea 250:

Incrementar comando DJNZ con B

Línea 290:

Comando CALL

Línea 300-360:

Aquí se ejecuta la corrección de la pila, SP y PC para saltos a Subprogramas (CALL y RST)

Línea 380-390:

Comando RET

Línea 400-450:

Corrección de la pila, SP y PC en retornos (RET)

Línea 455:

Simulación real a través de un programa en lenguaje máquina en caso de que CAP esté conectado y no se esté simulando ningún comando de salto.

Línea 460-560:

Salida formateada de los contenidos de los registros

Línea 570-700:

Rutina de entrada, cuando son modificados los registros

Línea 710-1000:

Subprogramas

Línea 720-780:

SUB (Subprograma) coloca FL a:
1 -, cuando no se presenta ninguna condición
0 -, cuando no se cumple la condición
-1 -, cuando se cumple la condición

Línea 760:

Calcula el núm. de Bit del correspondiente Flag en el registro de Flags.

Línea 790-860:

Aquí se activa el PC en función de FL. Los saltos indirectos son tratados en las líneas 830-860

Línea 870:

Coloca el PC en la dirección dada

Línea 880-900:

Muestra un registro de 16-Bits. La posición de los registros resulta de la sucesión de los comandos PUSH o POP del programa en lenguaje máquina

Línea 910-950:

Aquí se escriben los registros estándar en las respectivas posiciones de la memoria tras la modificación

Línea 960-1000:

Tras una modificación, puede ser colocado de nuevo un registro deseado con esta rutina

Línea 1010-1880:

Desensamblador (ver descripción allí)
Se realizaron las siguientes modificaciones respecto al desensamblador original:
- El formato de salida ha sido ligeramente variado (Línea 1180)
- Al leer los códigos, éstos son introducidos mediante POKRE inmediatamente en la posición correspondiente en el programa en lenguaje máquina (Línea 1850). También está modificado el formato de la salida de códigos (Línea 1870)

Línea 1890-2340:

Inicialización
Hasta Línea 2340 como desensamblador

Línea 2350-2540:

Lectura de las variables adicionales necesarias para el Simulador y lectura del programa de simulación en lenguaje máquina.

Lista de Variables

A- Devolución de SUB"Hex.-Dec."Valor de A$ interpretado como cifra hexadecimal
A$- Traspasos de SUBs
AD- Dirección del primer código del comando actual
AS- Dirección de salto
B- Valor del registro B
B$- Palabra de comando
BI- NÚmero de Bit
C1- Bit 7 y 6
C2- Bit 5 hasta 3
C3- Bit 2-0
DI- Distancia en comandos indexados
DI$- Distancia del string para salida
E$- Entrada string (s/n)
FL- Flag para comprobación de condición
I$- contiene registro Indice actual
IL- activado, cuando hay direccionamiento indexado, sinó, desactivado
K- Contador
LB- Memorización intermedia del Low Byte en cifras de 2-Bytes
PC- El indicador de programa señala hacia las direcciones de la posición actual
PO- Posición de n, nn, e, HL... en PR$
PR$- PR$ contiene el comando Assembler traducido
Q- Contador
QQ- Contador
RG$- Registro: descomponer devolución del código SUB RG$ contiene el registro subordinado a C3
S1- Traspaso registro dirección de Inicio
S2- Fin traspaso registro dirección de Inicio
S3- Dirección de traspaso de la pila
S4- Dirección de traspaso código del comando

SP- Dirección de la pila
W- Código leído
WE- Valor de un núm. de 2-Bytes ("nn")
Z$- Línea de entrada

Tablas

RT$(7) - Registros
RS$(7) - Comandos de rotación y desplazamiento
BF$(3) - Comandos de manipulación de Bits
AL$(7) - Comandos aritméticos o lógicos
BF$(255)- 0 hasta &H3F: Comandos que comienzan como Byte 1 y tienen el núm. del contador de las variables de campo
&H40-&H8F: Comandos que comienzan con &HED y tienen como Byte 2, el núm. del contador de las variables de campo
&H8F-&HFF: Comandos que tienen como Byte 1, el núm. del contador como variable de campo
CO$(7) - Condición

5.4 EL COMANDO >USR<

En este capítulo aprenderemos la utilización del comando USR en el BASIC del MSX.
Contrariamente a la llamada habitual con >CALL< o >SYS< (p.ej. en el C64), se ha implementado en el comando del BASIC MSX un pase directo de parámetros. Para sacar provecho a esta posibilidad, deberemos primero familiarizarnos con la representación interna de números y datos alfanuméricos (=string).
Existen cuatro tipos distintos de representación de datos en el ordenador:

1. Integer	INT - número entero
2. Single Precision	SNG - precisión simple
3. Double Precision	DBL - precisión doble
4. Strings	STR - alfanumérico

En primer lugar, nos ocuparemos sólo de los tres primeros tipos, es decir, de la representación de números. El primer tipo, el de la representación de números enteros, usted ya lo conoce. Un número entero es descompuesto en Low Byte y en High Byte. El Bit 7 del High Byte se utiliza como signo (+,-). Cero significa números positivos, y 1, negativos. Los números negativos son representados en el complemento a dos, es decir, se complementa la cantidad del número y se suma 1 (ver capítulo 4.5 Comandos Aritméticos). De este modo, las constantes o variables de números enteros pueden obtener valores de números enteros desde -32768 hasta +32767.

decimal	binario	hex
-32768	1 000 0000 0000 0000	80 00
-32767	1 000 0000 0000 0001	80 01
-32766	1 000 0000 0000 0010	80 02
-32765	1 000 0000 0000 0011	80 03
	...	
-2	1 111 1111 1111 1110	FF FE
-1	1 111 1111 1111 1111	FF FF
0	0 000 0000 0000 0000	00 00
1	0 000 0000 0000 0001	00 01
2	0 000 0000 0000 0010	00 02
	...	
32766	0 111 1111 1111 1110	7F FE
32767	0 111 1111 1111 1111	7F FF

Para distinguir interneaente los distintos tipos de representación, a cada uno le corresponde un código de tipo. Este código es igual al número de Bytes necesarios para el almacenado del tipo correspondiente. Para memorizar un número entero, se necesitan 2 Bytes; así pues, el índice de tipo para constantes o variables enteras será 2. Este índice de tipo es cargado en el acumulador por la rutina del sistema del interpretador BASIC, que ejecuta el comando >USR<. Además, el signo característico de la variable actual siempre es memorizado en la dirección &HF663. De este modo, es posible comprobar si el parámetro dado corresponde al tipo correcto. Los dos Bytes son transferidos a la dirección &HF7F8 y &HF7F9.
Con estas informaciones, disponemos ahora en el programa en lenguaje máquina, del respectivo valor dado en BASIC. Mediante esta posibilidad, podemos realizar ampliaciones sencillas de comandos.

En el ensamblador se definió la función FNDK, que lee el valor de 16-Bits de dos posiciones de memoria sucesivas. En algunos "Dialectos" de BASIC existe el comando >DEEK<, que ejecuta esta función. Con ayuda del comando >USR< ha de ser implementado ahora el comando >DEEK< en el BASIC del MSX. También ha de ser transferida la dirección del Byte bajo de

las posiciones de memoria que han de ser leídas. Al principio del programa se comprueba si el parámetro transferido corresponde al tipo correcto. Si éste no es el caso, se deberá saltar a la salida "Typ Mismatch Error". Para ello, el registro E es cargado con el número del correspondiente error y es bifurcado hacia &H406F (Rutina para salida de error).

```
CP 2 ; Tipo 2
JR Z,OK ; si, entonces OK
LD E,13 ; no, entonces
JP &H406F ; Typ mismatch Error
OK ...
```

Además, para cada mensaje de error, también existe una dirección de salto propia, que resuelve la carga del registro E. Para el "Typ mismatch Error" es la dirección &H406D. Con ésto, el programa se reduce a:

```
CP 2 ; Tipo 2 ?
JP NZ,&H406D ; no, entonces "Typ mismatch Error"
   .
   .
```

Mediante este procedimiento, la solución de los controles de tipos presenta un inconveniente: Si el usuario introduce, a partir del BASIC, un número de tamaño correcto, pero de tipo equivocado -p.ej. 1000 puede ser memorizado como número entero, pero también como Tipo SNG o DBL-, ello conduciría a un "Typ mismatch Error". Sin embargo, es posible convertir un número de este tipo en un número entero. En BASIC, ésto es resuelto mediante la función >CINT<.
Si por lo tanto, llamamos al inicio del programa en lenguaje máquina a la rutina >CINT<, esta conversión se realiza de forma automática. De este modo, sólo en el caso de introducción de strings (cadenas), puede presentarse un error del tipo descrito arriba. Además, la Rutina >CINT< comprueba si

en la introducción, se trata de un número de tamaño correcto, de lo contrario, aparece el mensaje de error "Overflow".
Si usted necesita variables de otros tipos, utilice las rutinas CSNG (&H2FB2) y CDBL (&H303A). Observe la tabla existente al final del libro, en la que hay representadas todas las rutinas del sistema tratadas, además de muchas otras.

A continuación, la ejecución del comando >DEEK<.

```
10 ' CALL &H2F8A ; CINT convertido a partir de INT
20 ' LD HL,(&HF7F8) ; valor parámetro transferido=
Dirección
30 ' LD E,(HL) ; Low Byte
40 ' INC HL
50 ' LD D,(HL) ; High Byte
```

En la línea 20 es leído a partir de las direcciones transferidas &HF7F8 y &HF7F9, el valor del parámetro transferido, o sea, la dirección a partir de la cual ha de ser leído el valor de 2-Bytes. En las líneas 30 hasta 50 es cargado en el registro DE, el valor de 2-Bytes existente en estas direcciones.
Ahora, tenemos que pasar de nuevo al BASIC el valor obtenido. Para ello, deberá ser cargado en la dirección &HF7F8/9, el valor a transferir. Además, el acumulador deberá contener el índice de tipo, y el registro HL deberá ser cargado con la dirección &HF7F6.

```
60 ' LD (&HF7F8),DE ; resultado de "DEEK"
70 ' LD HL,&HF7F6
80 ' RET
```

Ensamble usted el programa y pruebe la nueva función:

```
?USR1(2)
```

da como resultado 370, el mismo resultado que

```
?PEEK(2)+256*PEEK(3)
```

Esto significa pues, que a partir de la dirección 2, se halla el valor de 2-Bytes 370 o bien, &H207.
Con ésto, hemos aprendido a pasar a números enteros. Las otras dos representaciones de números son básicamente distintas.

Los números de coma flotante son memorizados en el sistema operativo del MSX en una forma poco habitual para este tipo de ordenadores: En el formato BCD.
En el formato BCD (decimal codificado binario) son almacenadas cada una de las cifras de un Número decimal. La ventaja radica en una cantidad perfectamente determinada de posiciones que son tratados. El rango de valor en el sistema del MSX es mayor (10^-64 hasta 10^62) que en el método de memorización habitual (10^-32 hasta 10^31). Desgraciadamente, los números en formato BCD no pueden ser tratados tan rápidamente. Por este motivo, existen dos precisiones distintas: SGN con 6 posiciones y DBL con 14 posiciones.
La representación con precisión simple (SGN) será suficiente para cubrir las exigencias de casi todos los casos.

¿Cómo se almacena un número en formato BCD?

Antes de ocuparnos de la memorización de cifras, describiremos la representación exponencial de los números, como se conoce p.ej. en las calculadoras de bolsillo. Si un número ha de ser respresentado en forma exponencial, primero, se determina el número de veces que la base del sistema decimal, o sea, el diez, está contenida como factor en el número. Finalmente, el número es descompuesto en dos cantidades. Una parte contiene todas las cifras del número (mantisa), donde la coma siempre está detrás de la primera cifra. La segunda parte indica el número de veces que el diez cabe en el número original, o bien, expresado de otro modo, cuántos lugares ha de ser corrida la coma en la primera parte (exponente). Se escribirá lo siguiente:

27=2.7*10^2
3956=3.956*10^4

Basándonos en el convenio de que el exponente negativo significa un corrimiento de la coma en dirección contraria, o sea, hacia la derecha, es posible representar todos los números:

0.21=2.1*10^-1
0.0051=5.1*10^-3
-9=-9.0*10^0

Con esta convención, cada número puede ser descompuesto en una mantisa (parte de cifras situada sólo un lugar antes de la coma) y en un exponente. La forma de escritura exponencial presenta la ventaja de poder almacenar números muy grandes o muy pequeños con un esfuerzo relativamente pequeño:

0.00000000000000735=7.35*10^-15
6390000000000000=6.39*10^15

Para el cálculo mediante estos números, son válidas las reglas habituales de cálculo exponencial.

Adición/Sustracción

Sólo se pueden sumar números de igual exponente. Si los exponentes de los respectivos sumandos son distintos entre sí, entonces, el número de menor exponente es convertido al de mayor exponente. Una vez que los dos exponentes son iguales, se procede simplemente a la suma o sustracción de las mantisas.

Ejemplo:

57 + 0.31
5.7*10^1 + 3.1*10^-1
5.7*10^1 + 0.031*10^1
5.731*10^1

Multiplicación/División

En estos dos tipos de operaciones aritméticas, las mantisas son multiplicadas o divididas, y a continuación, son sumados o restados los exponentes.

Ejemplo:

```
     0.13 * 20
1.3*10^-1 * 2.0*10^1
  1.3*2.0 * 10^1(-1+1)
      2.6 * 10^0
      2.6
```

```
      25 / 0.05
2.5*10^1 / 5.0*10^-2
 2.5/5.0 * 10^(1-(-2))
      0.5 * 10^3
      5.0 * 10^2
      500
```

Si queremos traspasar este proceso a un microprocesador, será necesaria la operación de almacenar los nÚmeros en este formato. En el formato BCD, cada cifra de la mantisa es almacenada por separado.
Una cifra del sistema decimal posee valores que van desde 0 hasta 9=&B1001. Así pues, se necesitan cuatro Bits para el almacenado de una cifra. SegÚn ésto, con un Byte = 8 Bits, pueden ser codificadas dos cifras , donde el High Nibble (Bit 4 hasta 7) corresponde a la primera cifra, y el Low Nibble (Bit 0 hasta 3), a la segunda.

Ejemplo:

```
Decimal     : 27
High Nibble: 2 =&B10
Low Nibble : 7=&B111
Formato BCD: &B0010 0111=&H27=39
```

Debido a las características especiales del sistema hexadecimal, donde 4 Bits corresponden a una cifra hexadecimal, el valor BCD codificado siempre es el valor del número decimal interpretado como número hexadecimal.

El valor BCD de 63 es &H63=&B0110 0011=99

El número de cifras a almacenar es teóricamente ilimitado. Se determina a través del número de Bytes utilizados para la memorización. En la representación de números con precisión simple (SNG), el número de Bytes, utilizados para la memorización de las cifras de la mantisa, asciende a tres. Con ésto, los números de este tipo presentan una precisión de 6 cifras, concretamente, 2 por cada Byte.
El número 123794 es almacenado en formato BCD como secuencia de 3 Bytes:

```
&H12, &H37, &H94
```

Si se elige la precisión doble, habrá 7 Bytes a nuestra disposición, lo cual hace posible la obtención de un número de 14 cifras.
Con ésto queda claro el proceso interno de memorización de la mantisa de números de coma flotante en representación exponencial.
Para la codificación del exponente se ha seguido un camino distinto: éste queda determinado directamente por el valor de un Byte. Además, aún se ha de colocar en este Byte información sobre:

-el signo de la mantisa
-el signo del exponente

El Bit 7 muestra, siguiendo el método habitual, el signo del número:

0- positivo
1- negativo

Los demás 7 Bit representan el valor del exponente, donde al valor real se le ha sumado &H41:

Valor Bit 6-0	Valor Exponente
&H7F	&H3E = 62
&H7E	&H3D = 61
.	.
.	.
.	.
&H42	&H01 = 1
&H41	&H00 = 0
&H40	-&H01 =-1
.	.
.	.
.	.
&H02	-&H3F =-63
&H01	-&H40 =-64
&H00	significa número = 0

O sea, el exponente puede adoptar valores que van desde -64 hasta +62, un rango de números más que suficiente para cualquier cálculo.
Según la convención, para un número con valor 0, su exponente será colocado a 0.

Comprobemos está forma de representación con un programa BASIC. Para ello, utilizaremos el comando >VARPTR<. Esta función nos da la dirección en la memoria donde está codificado el valor de la variable en la forma descrita arriba. Para la administración de variables, a continuación del programa BASIC sigue inmediatamente un espacio, donde es almacenado el nombre y el valor de cada variable utilizada. En la dirección dada por la función >VARPTR<, se halla el primer Byte del número codificado, el cual según la convención, es el exponente.

A continuación están, según el tipo, los 3 (SNG) o bien, 7 (DBL) Bytes, que representan a las cifras.

```
10 X!=43546!
20 AD=VARPTR(X!)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+3
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

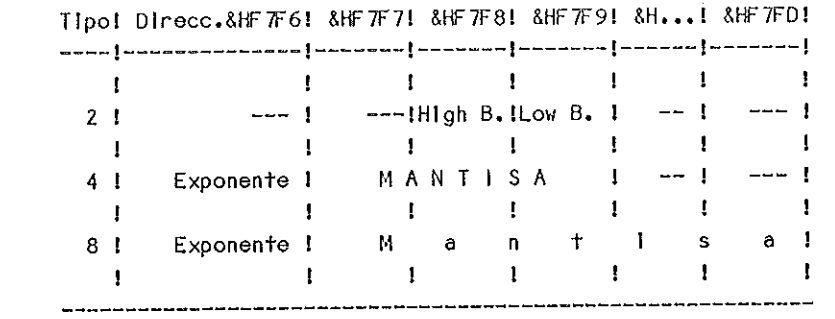
```
10 X#=.012342349#
20 AD=VARPTR(X#)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+6
50 PRINT RIGHT$("0"+HEX$(PEEK(I),2);
60 NEXT
```

Déle valores distintos a X y observe las respectivas salidas del programa.

¿Cómo son transferidos los números de este tipo de la función >USR<?

Basicamente, los números son tratados internamente, son depositados en una memoria intermedia especial en la RAM. A este espacio se le denomina acumulador de coma flotante o, abreviado, FAC. En el MSX BASIC, el FAC está entre la dirección &HF7F6 y &HF7FD. El sistema posee, cómo mínimo, dos de estas memorias, donde p.ej., se almacenan en cada una de ellas, los dos sumandos de una adición. El resultado de una operación de cálculo es básicamente depositada de nuevo en el FAC antes citado. Además, existe otra posición de memoria importante, donde es colocado el código de tipo de las respectivas variables. Esta tiene la dirección &HF663. La rutina del sistema >USR< carga este código de tipo en el acumulador.

Además, el registro HL es cargado con la dirección de inicio del FAC, o sea, con &HF7F6.
Si deseamos utilizar el parámetro transferido, tendremos que utilizar para nuestros fines el contenido del FAC. El formato de memoria exacto en el FAC es el siguiente:



Si al final del programa ha de ser devuelto un valor como resultado, deberemos cargar de nuevo el registro y la memoria según la forma antes descrita, o sea:

Acumulador : código de tipo
Registro HL : dirección de inicio FAC=&HF7F6
FAC : Valor codificado en la forma correspondiente

Para que usted se haga una idea de la aritmética en el ordenador, observe el siguiente programa. Aquí, es importante el comando DAA (Comparación decimal del acumulador), en la Línea 80. Prescinda de él y observe lo que ocurre cuando usted transfiere p.ej. 123456789 como número, o sea, >?USR1(123456789)<.

```
F000          10              ; por 2
F000 CD3A30   20        CALL  &H303A ; convierte en formato
DBL
F003 21FDF7   30        LD    HL,&HF7FD ; Fin FAC
F006 0607     40        LD    B,7 ; cantidad de Bytes
F008 B7       50        OR    A ; borrar Carry
F009 7E       60  NEXT  LD    A,(HL) ; valor del Byte
```

```
F00A 8F       70        ADC   A,A ; duplicar
F00B 27       80        DAA    ; comparación BCD
F00C 77       90        LD    (HL),A ; grabar de nuevo
F00D 2B      100        DEC   HL ; repetir direcc. del Byte
F00E 10F9    110        DJNZ  NEXT ; siguiente 7 veces
F010 3E08    120        LD    A,8 ; código de tipo
F012 21F6F7  130        LD    HL,&HF7F6 ; dirección FAC
F015 C9      140        RET
```

```
Programa :POR 2
Inicio : &HF000    Final : &HF015
Longitud :&H16 bytes
Errores : 0
Tabla de Variables :
NEXT  F009
```

El comando DAA provoca que los errores aparecidos tras las operaciones aritméticas (ADD, SUB, ADC y SBC) con números codificados en formato BCD, sean corregidos.

Ejemplo:

```
          7+7 = 14  decimal
&H07+&H07 = &H0E<>&H14
```

El comando DAA corrige el resultado erróneo &H0E a &H14. Además, son activados los Flags que corresponden al resultado real. Este comando puede ser utilizado tanto después de los comandos de adición, como de los de sustracción. Internamente, ésto se consigue con ayuda del Flag de sustracción (N). Este Flag es colocado a 1 después de cada sustracción, y a 0 después de cada adición.
Como programadores, nosotros no necesitamos ocuparnos de este Flag, pues el comando DAA ya se ocupa de él automáticamente. También el segundo Flag interno, el Flag de acarreo medio (H) es utilizado por el comando DAA. El Flag H es activado cuando tiene lugar un acarreo del Bit 3 al Bit 4, es decir, cuando se realiza un acarreo de una cifra BCD a otra.

Ejecute usted mismo el programa de la página anterior con un simulador, para poder así observar el método de funcionamiento del comando DAA. Para ello, no ejecute el comando LD (HL),A (o sea, desconecte CAP).

Además, haremos otra vez alusión a los comandos de rotación RRD y RLD de formato BCD. En estos comandos se efectúa una rotación de cuatro Bits entre las dos cifras (de 4 Bits cada una), en la posición de memoria a la que señala el registro HL, y una cifra en la mitad inferior (Bit 0-30) del acumulador.

Ejemplo:

A: &H23 (HL)=&H45 (Byte en dirección HL)

Después de RRD

A: &H25 (HL)=&H34

Después de RLD

A: &H24 (HL)=&H53

Una vez comentado el traspaso de los números, pasemos a ocuparnos de otro tipo de variables completamente distinto: la variable de cadena:

En una cadena (string) se almacenan datos alfanuméricos, es decir, signos tales como letras o cifras. A cada signo le corresponde un código. Para los códigos de 0 hasta 127, el código de signos del MSX corresponde ampliamente al denominado American Standard Code for Information Interchange (ASCII).
Para memorizar un signo se necesita exactamente un Byte. Un "String" (Ingl. string: cadena) es simplemente una cadena de sucesivos códigos de signos. Puesto que un signo corresponde a un Byte, un string se memorizará en una serie de posiciones de memoria sucesivas en la RAM. Para hacer corresponder a una variable determinada una cadena determinada, serán necesarias dos informaciones, las cuales,

componen el denominado String Descriptor:

1. La dirección de la primera posición de memoria, que contiene el primer código del signo de la cadena.

2. La longitud de la cadena, o sea, el número de Bytes que forman la cadena de signos.

Estos dos datos son almacenados en el espacio de variables BASIC junto con el nombre de la variable. La verdadera cadena de signos está en otro lugar. O bien, en el mismo programa, o en el espacio reservado especialmente a las cadenas. El tamaño de este espacio puede ser determinado mediante el comando >CLEAR<.
En BASIC podemos utilizar de nuevo el comando >VARPTR< para leer la dirección del descriptor de la cadena de una variable. El descriptor de la cadena consta de tres Bytes:

1. Byte : Longitud de la cadena
2.+3. Byte : Dirección de inicio de la cadena

La función >VARPTR muestra la dirección del primer Byte del descriptor de la cadena.
Pruebe usted el siguiente programa:

```
10 X$="CCCadena de SignOOOs"
20 AD=VARPTR(X$)
30 LA=PEEK(AD)
40 ST=PEEK(AD+1)+256*PEEK(AD+2)
50 FOR I=ST TO ST+LA-1
60 PRINT CHR$(PEEK(I));
70 NEXT
```

Por lo que se refiere al comando >USR<: Puesto que la longitud del descriptor de la cadena abarca tres Bytes, las cadenas alfanuméricas tienen el 3 como código de tipo. Este 3 es traspasado al acumulador y a la dirección &HF663. La dirección en la que está el descriptor de la cadena, o sea, el número que obtiene la función >VARPTR< en el BASIC, es transferida al registro DE. Para devolver la cadena al BASIC se procede exactamente de igual modo.
Con estas informaciones, es posible realizar, p.ej. el comando >UPPER<, conocido en algunos dialectos del BASIC. El comando >UPPER< convierte en mayúsculas todas las minúsculas de una cadena. La técnica de esta conversión ya la hemos estudiado en la aplicación de los comandos lógicos en el capítulo 4.8. Utilizando este programa, escriba usted un programa que realice el comando >UPPER< como comando >USR< en el BASIC del MSX.

```
 F000              10                ; Comando - UPPER
 F000              20             ORG &HF000
 F000 6D40         30  TYPERR  EQU  &H406D ; Typ mismatch E-
rror
 F002 FE03         40          CP   3 ; variable de cadena?
 F004 C26D40       50          JP   NZ,TYPERR ; no, entonces
Typ mismatch Error
 F007 1A           60          LD   A,(DE) ; Longitud de la
cadena
 F008 B7           70          OR   A ; activar Flag-Z cuando
A=0
 F009 C8           80          RET  Z ; Finalizado, cuando
Longitud=0
 F00A 47           90          LD   B,A ; Longitud como conta-
dor de Bucles
 F00B 62          100          LD   H,D ; direcc. descriptor
 F00C 6B          110          LD   L,E ; a HL
 F00D 23          120          INC  HL
 F00E 7E          130          LD   A,(HL) ; dirección
 F00F 23          140          INC  HL ; del
 F010 66          150          LD   H,(HL) ; inicio del string
 F011 6F          160          LD   L,A ; a HL
 F012 7E          170  BUCLE   LD   A,(HL) ; cuando código
ASCII
 F013 FE7B        180          CP   123 ; >= que ASCII ("z")
+1
?F015 30FE        190          JR   NC,OK ; entonces, no con-
vertir
 F017 FE61        200          CP   97 ; cuando < ASCII("a")
?F019 38FE        210          JR   C,OK ; entonces, no con-
vertir
 F01B E6DF        220          AND  &B11011111 ; Conversión
**** Línea  190  : OK=&HF01D Offset 6
**** Línea  210  : OK=&HF01D Offset 2
 F01D 77          230  OK      LD   (HL),A ; grabar ASCII de
nuevo
 F01E 23          240          INC  HL ; dirección del có-
digo siguiente
```

```
 F01F 10F1        250          DJNZ BUCLE ; repetir hasta
final del string
 F021 3E03        260          LD   A,3 ; Código de tipo para
String
 F023             270               ; DE contiene la antigua
dirección del descriptor
 F023 C9          280          RET
------------------------------------------------------------

Programa :UPPER
Inicio : &HF000      Final ; &HF023
Lngtd : &H24 Bytes
Errores : 0
Tabla de Variables:
TYPERR 406D   BUCLE F012
OK     F01D
```

Para cerrar este capítulo, le enseñaremos otros métodos, mediante los cuales, podremos ampliar el comando >USR< , de manera que nos permita transferir más parámetros. Sin embargo, para ello deberemos antes tratar algunos puntos relativos al método básico de funcionamiento del Interpretador BASIC.

Una vez que el programador introduce un programa BASIC en el ordenador, éste es traducido a un código intermedio. Por ejemplo, las palabras de comando no son almacenadas como tales, sinó que son sustituídas por un código abreviado (1 Byte de longitud; >128). A este código se le denomina Token. Cuando el interpretador reconoce un Token, salta a la rutina correspondiente a este Token (=comando). Allí son leídos los parámetros eventualmente existentes detrás del comando y se verifica si son correctos. A continuación nos ocuparemos en detalle de ello:

Cojamos como ejemplo el comando >POKE<. Detrás del comando >POKE< hay dos parámetros separados entre sí por una coma, la dirección y el valor. Supongamos de momento que el Token fue reconocido para >POKE<(=152) y que el programa bifurca hacia la rutina >POKE<(&H5423). En la interpretación en BASIC, el registro HL siempre señala en principio hacia la posición actualmente tratada en el programa BASIC. HL hace la función de puntero del programa BASIC. Se ha de garantizar siempre que este Puntero (Pointer) no se "pierda", pues de lo contrario no se podría realizar correctamente la interpretación.
Para leer los parámetros citados arriba, el interpretador posee rutinas especiales que, naturalmente, nosotros también podemos utilizar.
De momento, la rutina llamada en el comando >POKE<, recibe el nombre de GETADR. Lee un parámetro de valor entero. Antes de llamar a esta rutina, el registro HL deberá ser cargado con el puntero BASIC actual. Esta rutina evalúa de igual modo expresiones simples como 1237, o complejas, como

ABS(INT(VAL(RIGHT$(STRING$(4,"0")+HEX$(x),4)))).

El valor de 2-Bytes obtenido es devuelto al registro DE. En nuestro caso, al finalizar la rutina, el puntero BASIC señala hacia la coma siguiente. La rutina GETADR tiene la dirección &H542F.
A continuación, se comprueba si sigue una coma. Para ello, también existe aquí una rutina del sistema: El RST &H08. Este Restart sirve para verificar cualquier signo deseado. El código ASCII del signo deberá estar con DB directamente detrás del comando RST &H08. Así pues, para el comando >POKE<:

```
        .
        .
RST &H08 ; Verificación de ","
DB  &H2C ; Código para ","
        .
        .
```

Si no es encontrado el signo, aparece un "Syntax Error", de lo contrario, se efectúa el retorno.
La rutina que es llamada entonces para leer el valor de 8-Bits, recibe el nombre de GETBYT. El valor de 1-Byte es devuelto al acumulador. Una vez efectuada esta rutina, el registro HL señala, como puntero BASIC, al final del comando. Con ayuda de esta rutina es posible ampliar de tal modo el comando >USR<, que podemos transferir a voluntad un gran número de parámetros. El siguiente programa los utiliza para crear una función >PEEK< modificada, en la que, además de la dirección que ha de ser leída, también viene indicado el número del Slot. Con ésto, existe la posibilidad de leer módulos conectados de ampliación o solapados internamente.
La verdadera ejecución se lleva a cabo mediante la rutina a partir de &H000C (SLOT RD). Si el número del Slot es transferido al acumulador y la dirección al registro, esta rutina devuelve en el acumulador el valor del Byte leído.
En un programa de fabricación propia, el puntero BASIC puede ser leído de la pila mediante un "POP" doble. Es importante que la pila, y especialmente la dirección de retorno, sea reconstruida correctamente de nuevo.

A continuación, le mostramos el listado del ensamblador:

```
 F000            10                ; Slot PEEK
 F000            20                ; Formato: USR1 (direc-
ción),Slot
 F000            30          ORG   &HF000
 F000 5A47       40   ILLQUA EQU   &H475A
 F002 8A2F       50   CINT   EQU   &H2F8A
 F004 1C52       60   GETBYT EQU   &H521C
 F006 0C00       70   SLOTRD EQU   &H000C
 F008            80                ;
 F008 CD8A2F     90          CALL CINT ; convierte a
 INTEGER y carga el valor en HL
 F00B EB        100          EX   DE,HL
 F00C C1        110          POP  BC ; dirección de retorno
 F00D E1        120          POP  HL ; Puntero BASIC
 F00E E5        130          PUSH HL ; reconstruir
 F00F C5        140          PUSH BC ; pila
 F010 D5        150          PUSH DE ; valor de la direcc.
 F011 CF        160          RST  &H08 ;verificación de
 F012 2C        170          DB   &H2C ;ASCII (",")
 F013 CD1C52    180          CALL GETBYT ; coge núm. Slot
 F016 FE03      190          CP   3 ; ¿Núm. Slot >=3?
 F018 D25A47    200          JP   NC,ILLQUA ; sí, entonces
Illegal Quantity
 F01B E3        210          EX   (SP),HL ; cambiar la di-
rección a leer por el puntero BASIC
 F01C CD0C00    220          CALL SLOTRD ; slot Read
 F01F CDCF4F    230          CALL &H4FCF ; cargar Acumulador
en FAC
 F022 E1        240          POP  HL ; Puntero BASIC
 F023 C1        250          POP  BC ; Salto Retorno
 F024 D1        260          POP  DE ; Puntero BASIC antiguo
 F025 E5        270          PUSH HL ; Puntero BASIC nuevo
 F026 C5        280          PUSH BC ; retorno
 F027 21F6F7    290          LD   HL,&HF7F6 ; dirección
de inicio FAC
 F02A           300                ; A ya contiene el Tipo
2 ( ver rutina desde &H4FCF)
 F02A C9        310          RET
-------------------------------------------------
Programa :SLOTRD
Inicio : &HF000      Final : &HF02A
Long. : &H2B Bytes
Errores : 0
Tabla de Variables:
ILLQUA 475A    CINT   2F8A
GETBYT 521C    SLOTRD 000C
```

5.5 RUTINAS DEL SISTEMA

Dirección &H0000: RESET

La llamada de esta rutina provoca el mismo efecto que el conectar/desconectar del ordenador, o bien, como el pulsar la tecla RESET.

Dirección &H0008: RST &H08- comprueba el siguiente Byte

Se comprueba si el Byte situado en la dirección HL es igual al que hay detrás del comando RST &H08. En caso de desigualdad, aparece un "Syntax Error". De lo contrario, se bifurca hacia la rutina RST &H10 desde la dirección &H4666 (ver allí).

Dirección &H000C: LD A,(HL) con elección del Slot

En la transferencia, A contiene el número del Slot deseado (0-3). Como resultado, se obtiene en el acumulador y en el registro E el valor situado en la dirección HL del Slot deseado.

Dirección &H0014: LD (HL),E con elección del Slot

En la transferencia, A contiene el número del Slot, A, HL la dirección y E el valor a escribir.

Dirección &H0018: RST &H18- Emisión de un carácter

Muestra el carácter contenido en el acumulador en el periférico actual. Según el modelo estándar, es el del monitor. La impresora se selecciona al cargar la posición de memoria &HF416 con un valor <> 0.

Dirección &H0020: RST &H20- Comparación de HL con DE

El contenido del registro DE es restado del registro HL y, según el resultado, son influenciados los Flags. HL y DE no son (!) modificados.

Dirección &H0028: RST &H28- Comprobación del tipo de variable

Usted obtiene el tipo de la variable actual, con los siguientes significados:

Carry=0	(NC)	Tipo 8	DEL
Carry=1	(C)	Tipo 2,3 ó 4, es decir	
Sign Flag=1	(M)	Tipo 2	INT
Flag Cero=1	(Z)	Tipo 3	String
Sign Flag=0	(P)	Tipo 4	SNG

Dirección &H0038: RST &H38- Salto Interrupt en MODO 1

Este es el punto de entrada para la rutina, que en el Interrupt estándar es llamada 50 veces por segundo.

Dirección &H003E: Activación de las teclas de función

Esta rutina activa las teclas de función con sus palabras originales existentes al conectar el ordenador.

Dirección &H0047: Escritura del registro VDP

El registro VDP con el número dado en el registro C, recibe el valor contenido en el registro B, o sea, VDP(C)=B.

Dirección &H004A: Video RAM-Read

El Byte situado en la dirección HL en la RAM de Video es cargado mediante esta rutina en el acumulador. BASIC: A=VPEEK(HL).

Dirección &H004D: Video RAM-Write

El valor del acumulador es almacenado en la dirección HL de la RAM de Video mediante esta rutina. BASIC: VPOKE HL,A

Dirección &H005F: Select SCREEN

Aquí se conecta el Modo SCREEN al valor (0, 1,2 ó 3) contenido en el acumulador. BASIC: SCREEN A

Dirección &H0093: PSG Register Write

Esta rutina escribe en el registro PSG el valor contenido en el registro E con el número A. La programación del PSG en lenguaje máquina es muy importante para la producción de sonidos complejos. En BASIC, esta rutina corresponde al comando >SOUND A,E<.

Dirección &H0096: PSG Register Read

Tras llamar a esta rutina, el acumulador contiene el valor del registro PSG con el número que contenía el acumulador antes de ser llamada la rutina.

Dirección &H009F: Esperar a pulsación teclas

Esta rutina espera el tiempo necesario hasta que es pulsada una tecla. El código ASCII correspondiente a la tecla es registrado y almacenado en el acumulador. A continuación, se efectúa el retorno.

Dirección &H00AE: Entrada hasta CR desde el Inicio de la Línea

Mediante la llamada de esta rutina, usted obtiene la entrada de una línea completa. Puesto que una línea de entrada puede abarcar hasta 255 caracteres, deberá ser almacenada en la memoria RAM. Esto ocurre a partir de la dirección &HF55E. Por consiguiente, la última entrada estará registrada entre &HF55E y &HF65D. Al regreso de esta rutina, el registro HL contiene la dirección de inicio de este buffer de entrada menos 1.

Dirección &H00C0: BEEP

Saca un BEEP.
BASIC: BEEP

Dirección &H00C3: Clear Screen

Borra toda la memoria de pantalla en todos los Modos.
BASIC: CLS

Dirección &H00C6: colocar cursor

Mediante esta rutina, el cursor es colocado en la posición HL. Para ello es válido:

H- Línea
L- Columna

BASIC: LOCATE L.H

Dirección &H00CC: KEY OFF

Desconecta KEY.

Dirección &H00CF: KEY ON

Conecta KEY.

Dirección &H00D5: Consulta-STICK

Tras la transferencia de A, la rutina suministra el valor de la dirección correspondiente en el acumulador siguiendo el método habitual (ver manual).

0- Teclado
1- Joystick 1
2- Joystick 2

BASIC: A=STICK(A)
Aunque esta rutina corresponde exactamente al comando en BASIC, es importante, porque es precisamente en los juegos, donde adquiere importancia la rápida consulta del Joystick.

Dirección &H00D8: Consulta-STRIG

Tras la transferencia de A (ver arriba), esta rutina suministra:

0- cuando no se pulsa ninguna tecla (''botón de disparo'')
255- cuando se pulsa una tecla.

Dirección &H0132: CAP ON/OFF

Conecta/desconecta CAP.

A=0 CAP ON
A<> 0 CAP OFF

Dirección &H0135: Software Sound

Esta rutina coloca en low o en high la línea EXT SOUND, que lleva directamente al altavoz:

A=0 línea en low
A=<>0 línea en high

Una rápida y sucesiva conexión/desconexión lleva a la producción de un sonido. De lo contrario, al llamar a la rutina se produce un "crujido".

Dirección &H013E: VDP- Status Read

Esta rutina devuelve en el acumulador el valor actual del registro de estado VDP.

Dirección &H2F8A: CINT

Convierte FAC en "Formato Integer" y comprueba el tamaño. Esta rutina muestra en HL el valor y en el acumulador, el código de tipo (=2).

Dirección &H2FB2: CSNG

Convierte FAC en el formato ''simple precisión''.

Dirección &H2F99: copiar HL en FAC

Carga en el FAC el valor del registro HL y carga el código de tipo con 2.

Dirección &H303A: CDBL

Convierte FAC en el formato ''doble precisión''.

Dirección &H4055: Syntax Error (ERR=2)

Dirección &H4067: Overflow (ERR=6)

Dirección &H406A: Missing Operand (ERR=24)

Dirección &H406D: Typ Mismatch (ERR=13)

Dirección &H406F: Salida de errores

Muestra el error correspondiente al número transferido al registro E.

Dirección &H400B: GET PAR

Rutina del interpretador de BASIC que lee una dirección de 16 Bits y un valor de 8 Bits existente a continuación de ésta y separado de ella por una coma
Salida:
DE - valor de 8 Bits
BC - valor de 16 Bits

Dirección &H521C: GET BYTE

Lee un valor de 8 Bits, que es devuelto en el acumulador y en el registro E.

Dirección &H542F: GET ADR

Lee un valor de 16 Bits que es devuelto al registro DE.

Dirección &H5439: GET ADR entre paréntesis

Lee un valor de 16 Bits situado entre paréntesis.

5.6 MODIFICACIONES DEL SISTEMA

Para finalizar, aún conoceremos otra posibilidad de modificar el sistema operativo, o bien, el interpretador de BASIC. Los programadores que han creado el sistema operativo del MSX, preveyeron para ello una técnica denominada "enmascaramiento" (ingl to patch: enmascarar, parchear).
En una modificación del sistema, el problema básico consiste en que una memoria ROM no es modificable. Sólo es posible intervenir en las partes existentes en la memoria RAM. Para poder intervenir en muchas posiciones, antes de la rutina verdadera, se efectúa a partir de las rutinas ROM importantes una bifurcación a la RAM con un subprograma. Si el programa se halla en estado inicial, todas las direcciones referenciadas en la memoria RAM son cargadas con el código &HC9, el código para RET. Ello significa que la llamada de esta dirección no ejerce ninguna influencia y que el programa ROM se ejecuta sin modificaciones. Este ''área de parcheo'' se halla en la RAM del siatema desde &HFD9A hasta &HFFC9.
Si queremos cambiar una rutina, existe la posibilidad de colocar el "parche" en las direcciones de salto de la RAM correspondientes a la rutina. Es decir, que el comando RET es sustituído ("parcheado") por los códigos de otro comando, p.ej, de un comando de salto. Para cada salto parcheado hay disponibles 5 Bytes para la modificación. En ningún caso se deberán cambiar más, ya que entonces se empezaría a manipular la siguiente rutina. Observemos un ejemplo para aclarar este proceso de modificaciones en el sistema:

El ejercicio consiste en modificar el comando >INPUT< del BASIC de manera que no aparezca ningún interrogante. La rutina Input comienza en la dirección &H23CC. Mediante el desensamblador, traduzca a partir de la dirección &H23CC:

```
23CC  CDE0F0     CALL  &HFDE0
23CF  3E3F       LD    A,&H3F
23D1  DF         RST   &H18
23D2  3E20       LD    A,&H20
23D4  DF         RST   &H18
23D5  .
  .   .
  .   .
```

Como primer comando, usted verá el salto al área de parcheo en la memoria RAM. Si queremos modificar la rutina Input, dispondremos de las direcciones &HFDE0 hasta &HFD4. El código para "?" es cargado en el próximo comando y sacado mediante RST &H18. Se deberá saltar por encima de estos dos comandos. El programa deberá continuar a partir de la dirección &H23D2. El parcheado presentará el siguiente formato:

```
Dirección de partida del parche: &HFDE0
POP AF ; coger dirección de retorno
JP  &H23D2 ; omitir el interrogante

Los códigos serán: F1,C3,D2,23
```

El siguiente miniprograma instala el parche:

```
10 POKE &HFDE1,&HC3
20 POKE &HFDE2,&HD2
30 POKE &HFDE3,&H23
40 REM Activación
50 POKE &HFDE0,&HF1
60 REM Desconectar con &HFDE0,&HC9
```

Es importante que la dirección inicial del parcheo, en este ejemplo &HFDE0, sea cargada al final con el valor nuevo. Para hacer reversible la modificación y recuperar de nuevo el estado inicial, se deberá volver a escribir en esta posición: RET, o sea, &HC9.

La modificación >AUTO< del capítulo 4.9 emplea la misma técnica. Puesto que, sin embargo, el área a modificar se hallaba a mayor distancia del salto al parcheado, todo el espacio fue copiado y modificado en la memoria RAM.
El parcheo empleado corresponde al bucle de espera principal del ordenador, que se pondrá en funcionamiento cuando nos hallemos en el Modo de Entrada.

La salida a Impresora y a monitor también funciona siguiendo el mismo principio descrito arriba. El programa modifica la rutina RST &H18 hasta tal punto, que es llamada dos veces. Una vez para la salida a pantalla y otra, para la salida a Impresora.

CAPITULO VI : P E R S P E C T I V A S

Usted acaba de conocer las técnicas de programación fundamentales y los programas de ayuda para la elaboración de programas en lenguaje máquina.
La programación en lenguaje ensamblador es indispensable para problemas de programación de mayor envergadura. Los tiempos de desarrollo para el software, sin embargo, son mucho más largos que los de programas en lenguajes más elevados. Por este motivo, para una programación efectiva, son necesarios también buenos programas de desarrollo.
Comentaremos brevemente las características de tales programas. A un paquete de programas para el desarrollo de programas en lenguaje máquina le corresponde, como mínimo, un programa ensamblador y un extenso programa monitor.
El ensamblador es la condición previa para el desarrollo de programas más amplios. Además de los ya conocidos pseudocomandos, muchos ensambladores ofrecen posibilidades de simplificar aún más el desarrollo de un programa -aquí corresponde, p.ej., la definición de Macros-, de ensamblar condicionalmente y de acceder a programas externos o a variables.

Macros:

A menudo ocurre que una determinada sucesión de comandos se presenta varias veces en un programa. Mediante la utilización de Macros se evita que en tales casos, sea introducida continuamente la misma sucesión de comandos. Con ayuda de una Macrodefinición, se le puede asignar un nombre a una serie de comandos. Luego, en el Programa fuente se puede colocar el nombre del Macro en vez de la sucesión de comandos.
El ensamblador sustituye de manera automática la sucesión de comandos por el nombre del Macro correspondiente. Mediante la utilización de Macros, los programas fuente llegan a ser más claros y más cortos.

Ensamblaje condicionado:

En el ensamblaje condicionado se puede traducir determinadas partes del programa que dependen de una condición. El ensamblaje condicionado hace posible que un programa fuente normal, como una gestión de datos, sea escrita y luego recortada y adaptada para cada aplicación.

Programas externos y variables

En la programación en ensamblador es de gran importancia programar de forma estructurada. Ello significa que los grandes problemas se dividen en secciones más pequeñas, y cada sección del programa es realizada de forma independiente. A menudo, aparecen continuamente los mismos subprogramas: nosotros, p.ej. utilizábamos la rutina para la salida hexadecimal en distintos programas. En un ensamblador cómodo, estas rutinas a menudo necesarias y estas variables frecuentemente utilizadas forman una biblioteca de variables de programa. Las rutinas son reconocidas por su nombre en el programa fuente y luego, son cargadas automáticamente en el casette/diskette y añadidas al programa objeto.
El programa que realiza la combinación de distintos programas máquina también recibe el nombre de "Enlace" (ingl. link: enlazar). A menudo también está relacionado con él el denominado Relocator (reubicador), que corrige de nuevo las direcciones que cambian debido a la introducción adicional y al desplazamiento de los programas. Los programas que también contienen esta opción, por lo general, abarcan muchas posibilidades y son relativamente caros. La programación se realiza así de un modo mucho más cómodo y rápido. Para ello, muchos ensambladores poseen un editor propio, o sea que la entrada de los comandos en ensamblador ya no están asociados a un número de línea.

Aún existen otros programas auxiliares para el ensamblaje. La mayoría de ellos se integran en un programa monitor. Usted ya conoce las rutinas estándar de un programa monitor. Generalmente, el desensamblador está integrado en el programa monitor. Una de las características más importantes

del programa monitor está en sus muchas posibilidades de verificar programas.

La posibilidad de colocar un punto de ruptura es la más sencilla de las posibilidades de verificación. Las rutinas de comprobación amplias son recogidas a menudo en un denominado Debugger (eliminador de errores). El programa más importante a este respecto es el simulador paso a paso.

El hecho de poseer buenos programas de ayuda para el desarrollo de software, no lo es todo. Más importante aún es dar ese paso que nos lleva a la práctica de la programación. Este libro le ha proporcionado las técnicas básicas necesarias para la programación del Z80. Sólo mediante la práctica conocerá a fondo el lenguaje máquina. ¡Esperamos que se divierta con la elaboración de sus propios programas en código máquina!

Observación:

Si su ordenador no posee el diodo CAP, deberá ser cambiada la línea 120 del simulador de la manera siguiente:

```
120 LOCATE1:A$=INKEY:IF  A$="" THEN IF PEEK(&HFCAB)=255
THEN PRINT CHR$(13);";:GOTO 120 ELSE PRINT CHR$(13);"N";:GO
TO 120 ELSE LOCATE 0,,0
```

De este modo, la simulación real es indicada por "E" y la pseudosimulación, por "N".

TABLA DE CONVERSION DECIMAL - HEXADECIMAL - BINARIO

decimal	hex	binario	decimal	hex	binario
0	&H00	&B00000000	26	&H1A	&B00011010
1	&H01	&B00000001	27	&H1B	&B00011011
2	&H02	&B00000010	28	&H1C	&B00011100
3	&H03	&B00000011	29	&H1D	&B00011101
4	&H04	&B00000100	30	&H1E	&B00011110
5	&H05	&B00000101	31	&H1F	&B00011111
6	&H06	&B00000110	32	&H20	&B00100000
7	&H07	&B00000111	33	&H21	&B00100001
8	&H08	&B00001000	34	&H22	&B00100010
9	&H09	&B00001001	35	&H23	&B00100011
10	&H0A	&B00001010	36	&H24	&B00100100
11	&H0B	&B00001011	37	&H25	&B00100101
12	&H0C	&B00001100	38	&H26	&B00100110
13	&H0D	&B00001101	39	&H27	&B00100111
14	&H0E	&B00001110	40	&H28	&B00101000
15	&H0F	&B00001111	41	&H29	&B00101001
16	&H10	&B00010000	42	&H2A	&B00101010
17	&H11	&B00010001	43	&H2B	&B00101011
18	&H12	&B00010010	44	&H2C	&B00101100
19	&H13	&B00010011	45	&H2D	&B00101101
20	&H14	&B00010100	46	&H2E	&B00101110
21	&H15	&B00010101	47	&H2F	&B00101111
22	&H16	&B00010110	48	&H30	&B00110000
23	&H17	&B00010111	49	&H31	&B00110001
24	&H18	&B00011000	50	&H32	&B00110010
25	&H19	&B00011001	51	&H33	&B00110011

TABLA DE CONVERSION DECIMAL - HEXADECIMAL - BINARIO

decimal	hex	binario	decimal	hex	binario
52	&H34	&B00110100	78	&H4E	&B01001110
53	&H35	&B00110101	79	&H4F	&B01001111
54	&H36	&B00110110	80	&H50	&B01010000
55	&H37	&B00110111	81	&H51	&B01010001
56	&H38	&B00111000	82	&H52	&B01010010
57	&H39	&B00111001	83	&H53	&B01010011
58	&H3A	&B00111010	84	&H54	&B01010100
59	&H3B	&B00111011	85	&H55	&B01010101
60	&H3C	&B00111100	86	&H56	&B01010110
61	&H3D	&B00111101	87	&H57	&B01010111
62	&H3E	&B00111110	88	&H58	&B01011000
63	&H3F	&B00111111	89	&H59	&B01011001
64	&H40	&B01000000	90	&H5A	&B01011010
65	&H41	&B01000001	91	&H5B	&B01011011
66	&H42	&B01000010	92	&H5C	&B01011100
67	&H43	&B01000011	93	&H5D	&B01011101
68	&H44	&B01000100	94	&H5E	&B01011110
69	&H45	&B01000101	95	&H5F	&B01011111
70	&H46	&B01000110	96	&H60	&B01100000
71	&H47	&B01000111	97	&H61	&B01100001
72	&H48	&B01001000	98	&H62	&B01100010
73	&H49	&B01001001	99	&H63	&B01100011
74	&H4A	&B01001010	100	&H64	&B01100100
75	&H4B	&B01001011	101	&H65	&B01100101
76	&H4C	&B01001100	102	&H66	&B01100110
77	&H4D	&B01001101	103	&H67	&B01100111

TABLA DE CONVERSION DECIMAL - HEXADECIMAL - BINARIO

decimal	hex	binario	decimal	hex	binario
104	&H68	&B01101000	130	&H82	&B10000010
105	&H69	&B01101001	131	&H83	&B10000011
106	&H6A	&B01101010	132	&H84	&B10000100
107	&H6B	&B01101011	133	&H85	&B10000101
108	&H6C	&B01101100	134	&H86	&B10000110
109	&H6D	&B01101101	135	&H87	&B10000111
110	&H6E	&B01101110	136	&H88	&B10001000
111	&H6F	&B01101111	137	&H89	&B10001001
112	&H70	&B01110000	138	&H8A	&B10001010
113	&H71	&B01110001	139	&H8B	&B10001011
114	&H72	&B01110010	140	&H8C	&B10001100
115	&H73	&B01110011	141	&H8D	&B10001101
116	&H74	&B01110100	142	&H8E	&B10001110
117	&H75	&B01110101	143	&H8F	&B10001111
118	&H76	&B01110110	144	&H90	&B10010000
119	&H77	&B01110111	145	&H91	&B10010001
120	&H78	&B01111000	146	&H92	&B10010010
121	&H79	&B01111001	147	&H93	&B10010011
122	&H7A	&B01111010	148	&H94	&B10010100
123	&H7B	&B01111011	149	&H95	&B10010101
124	&H7C	&B01111100	150	&H96	&B10010110
125	&H7D	&B01111101	151	&H97	&B10010111
126	&H7E	&B01111110	152	&H98	&B10011000
127	&H7F	&B01111111	153	&H99	&B10011001
128	&H80	&B10000000	154	&H9A	&B10011010
129	&H81	&B10000001	155	&H9B	&B10011011

TABLA DE CONVERSION DECIMAL - HEXADECIMAL - BINARIO

decimal	hex	binario	decimal	hex	binario
156	&H9C	&B10011100	182	&HB6	&B10110110
157	&H9D	&B10011101	183	&HB7	&B10110111
158	&H9E	&B10011110	184	&HB8	&B10111000
159	&H9F	&B10011111	185	&HB9	&B10111001
160	&HA0	&B10100000	186	&HBA	&B10111010
161	&HA1	&B10100001	187	&HBB	&B10111011
162	&HA2	&B10100010	188	&HBC	&B10111100
163	&HA3	&B10100011	189	&HBD	&B10111101
164	&HA4	&B10100100	190	&HBE	&B10111110
165	&HA5	&B10100101	191	&HBF	&B10111111
166	&HA6	&B10100110	192	&HC0	&B11000000
167	&HA7	&B10100111	193	&HC1	&B11000001
168	&HA8	&B10101000	194	&HC2	&B11000010
169	&HA9	&B10101001	195	&HC3	&B11000011
170	&HAA	&B10101010	196	&HC4	&B11000100
171	&HAB	&B10101011	197	&HC5	&B11000101
172	&HAC	&B10101100	198	&HC6	&B11000110
173	&HAD	&B10101101	199	&HC7	&B11000111
174	&HAE	&B10101110	200	&HC8	&B11001000
175	&HAF	&B10101111	201	&HC9	&B11001001
176	&HB0	&B10110000	202	&HCA	&B11001010
177	&HB1	&B10110001	203	&HCB	&B11001011
178	&HB2	&B10110010	204	&HCC	&B11001100
179	&HB3	&B10110011	205	&HCD	&B11001101
180	&HB4	&B10110100	206	&HCE	&B11001110
181	&HB5	&B10110101	207	&HCF	&B11001111

TABLA DE CONVERSION DECIMAL - HEXADECIMAL - BINARIO

decimal	hex	binario	decimal	hex	binario
208	&HD0	&B11010000	234	&HEA	&B11101010
209	&HD1	&B11010001	235	&HEB	&B11101011
210	&HD2	&B11010010	236	&HEC	&B11101100
211	&HD3	&B11010011	237	&HED	&B11101101
212	&HD4	&B11010100	238	&HEE	&B11101110
213	&HD5	&B11010101	239	&HEF	&B11101111
214	&HD6	&B11010110	240	&HF0	&B11110000
215	&HD7	&B11010111	241	&HF1	&B11110001
216	&HD8	&B11011000	242	&HF2	&B11110010
217	&HD9	&B11011001	243	&HF3	&B11110011
218	&HDA	&B11011010	244	&HF4	&B11110100
219	&HDB	&B11011011	245	&HF5	&B11110101
220	&HDC	&B11011100	246	&HF6	&B11110110
221	&HDD	&B11011101	247	&HF7	&B11110111
222	&HDE	&B11011110	248	&HF8	&B11111000
223	&HDF	&B11011111	249	&HF9	&B11111001
224	&HE0	&B11100000	250	&HFA	&B11111010
225	&HE1	&B11100001	251	&HFB	&B11111011
226	&HE2	&B11100010	252	&HFC	&B11111100
227	&HE3	&B11100011	253	&HFD	&B11111101
228	&HE4	&B11100100	254	&HFE	&B11111110
229	&HE5	&B11100101	255	&HFF	&B11111111
230	&HE6	&B11100110			
231	&HE7	&B11100111			
232	&HE8	&B11101000			
233	&HE9	&B11101001			

Explicación de las tablas siguientes:

En la primera tabla hay unas flechas para los códigos &HCB, &HED, &HDD y &HFD. Ello significa lo siguiente:

&HCB: Si el primer código a traducir es &HCB, se deberá consultar la segunta tabla para el segundo código. Estos comandos son los comandos de rotación y de desplazamiento

&HED: Si el primer código a traducir es &HED, se deberá consultar la tercera tabla para el segundo código.

&HDD y &HFD: Si el primer código es &HED o &HFD, en tal caso, se trata de un comando de direccionamiento indexado. &HDD afecta el registro IX y &HFD, afecta al registro IY.
Los comandos de direccionamiento indexado no aparecen en otra tabla. Pueden ser obtenidos a partir de las tablas existentes siguiendo el procedimiento:
El segundo código es consultado en las tablas, como hacemos habitualmente. El comando obtenido deberá contener el registro HL. Si el registro HL no aparece en el operando, o si se ha detectado el comando EX DE,HL, entonces se trata de un comando no válido (es indicado en el desensamblador como ???). Si se trata de un comando válido, se deberá sustituir el registro HL por IX o IY.

De HL resulta IX o IY
De (HL) resulta (IX+d) o (IY+d), donde ésto es dado mediante el tercer código.

Estas reglas son válidas, a excepción del comando JP (HL), para todos los comandos que contienen HL. Una vez activado el registro índice, JP (HL) se convierte, aunque HL esté entre paréntesis, en JP (IX) o JP (IY), después de aplicar el registro índice.

	0	1	2	3	4	5	6	7
0	NOP	LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA
1	DJNZ of	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA
2	JR NZ,of	LD HL,nn	LD (nn),HL	INC HL	INC H	DEC H	LD H,n	DAA
3	JR NC,of	LD SP,nn	LD (nn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A
A	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A
B	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A
C	RET NZ	POP BC	JP NZ,nn	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST & 00
D	RET NC	POP DE	JP NC,nn	OUT (n),A	CALL NC,nn	PUSH DE	SUB n	RST & 10
E	RET PO	POP HL	JP PO,nn	EX (SP),HL	CALL PO,nn	PUSH HL	AND n	RST &20
F	RET P	POP AF	JP P,nn	DI	CALL P,nn	PUSH AF	OR n	RST & 30

	8	9	A	B	C	D	E	F
0	EX AF,AF	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	JR of	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR Z,of	ADD HL,HL	LD HL,(nn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR C,of	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C	RET Z	RET	JP Z,nn	⟶	CALL Z,nn	CALL nn	ADC A,n	RST & 08
D	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn	⟶	SBC A,n	RST & 18
E	RET PE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn	⟶	XOR n	RST & 28
F	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn	⟶	CP n	RST & 38

	0	1	2	3	4	5	6	7	...
0	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A	...
1	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A	...
2	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A	...
3									...
4	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A	...
5	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A	...
6	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A	...
7	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A	...
8	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A	...
9	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A	...
A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A	...
B	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A	...
C	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A	...
D	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A	...
E	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A	...
F	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A	...

	8	9	A	B	C	D	E	F
0	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
4	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

	0	1	2	3	4	5	6	7
4	IN B,(C)	OUT (C),B	SBC HL,BC	LD (nn),BC	NEG	RETN	IM 0	LD I,A
5	IN D,(C)	OUT (C),P	SBC HL,DE	LD (nn),DE			IM 1	LD A,I
6	IN H,(C)	OUT (C),H	SBC HL,HL	LD (nn),HL				RRD
7			SBC HL,SP	LD (nn),SP				
8								
9								
A	LDI	CPI	INI	OUTI				
	LDIR	CPIR	INIR	OTIR				

	8	9	A	B	C	D	E	F
4	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(nn)		RETI		LD R,A
5	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(nn)			IM 2	LD A,R
6	IN L,(C)	OUT (C),L	ADC HL,HL	LD HL,(nn)				RLD
7	IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(nn)				
8								
9								
A	LDD	CPD	IND	OUTD				
B	LDDR	CPDR	INDR	OTDR				

SOURCE

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

SOURCE

DESTINATION		REGISTER							IMM. EXT.	EXT. ADDR.	REG. INDIR.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
REGISTER	AF										**F1**
	BC								**01** n n	ED 4B n n	**C1**
	DE								**11** n n	ED 5B n n	**D1**
	HL								**21** n n	**2A** n n	**E1**
	SP				**F9**		DD F9	FD F9	**31** n n	ED 7B n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)		ED 43 n n	ED 53 n n	**22** n n	ED 73 n n	DD 22 n n	FD 22 n n			
REG. IND.	(SP)	**F5**	**C5**	**D5**	**E5**		DD E5	FD E5			

PUSH INSTRUCTIONS →

POP INSTRUCTIONS ↑

NOTE: The Push & Pop Instructions adjust the SP after every execution

		IMPLIED ADDRESSING				
		AF′	BC′, DE′ & HL′	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			**E8**		
REG. INDIR.	(SP)			**E3**	DD E3	FD E3

SOURCE

DESTINATION				
		REG. INDIR.		
		(HL)		
REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)⟵(HL) Inc HL & DE, Dec BC	
		ED B0	'LDIR,' – Load (DE)⟵(HL) Inc HL & DE, Dec BC, Repeat until BC = 0	
		ED A8	'LDD' – Load (DE)⟵(HL) Dec HL & DE, Dec BC	
		ED B8	'LDDR' – Load (DE)⟵(HL) Dec HL & DE, Dec BC, Repeat until BC = 0	

Reg HL points to source
Reg DE points to destination
Reg BC is byte counter

SEARCH LOCATION

REG. INDIR.	
(HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory to be compared with accumulator contents
BC is byte counter

SOURCE

	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

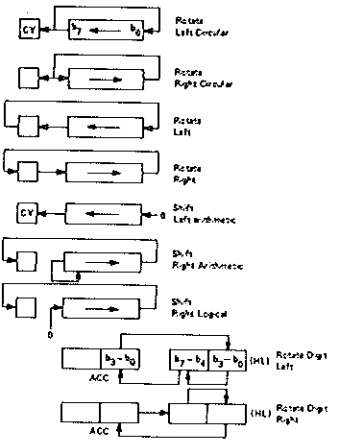
SOURCE

DESTINATION		BC	DE	HL	SP	IX	IY
'ADD'	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
INCREMENT 'INC'		03	13	23	33	DD 23	FD 23
DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

Source and Destination

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

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F



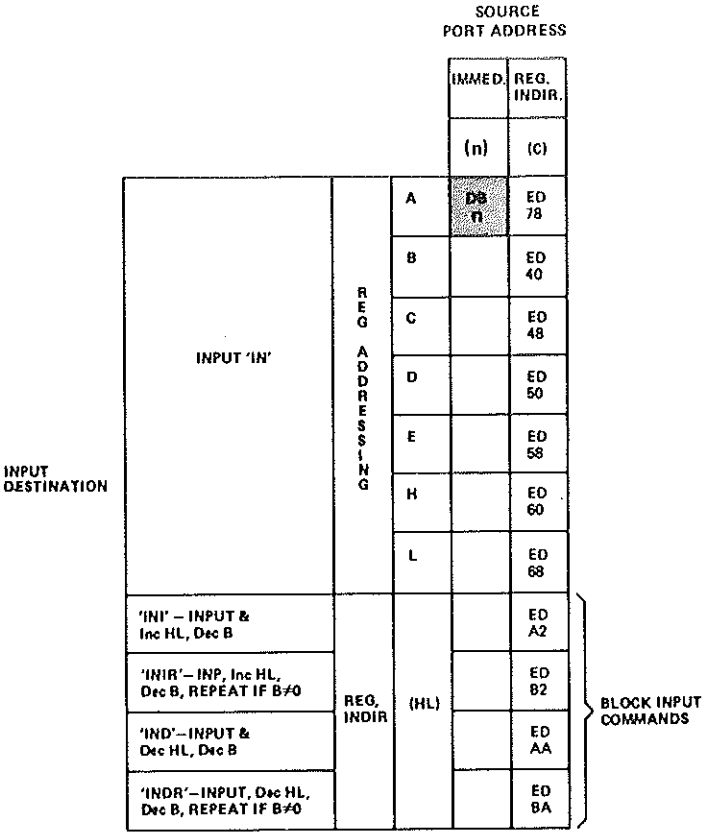
		REGISTER ADDRESSING							REG. INDIR.	INDEXED	
	BIT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

CONDITION

			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

NOTE—CERTAIN FLAGS HAVE MORE THAN ONE PURPOSE. REFER TO SECTION 6.0 FOR DETAILS

		OP CODE	
CALL ADDRESS	0000_H	C7	'RST 0'
	0008_H	CF	'RST 8'
	0010_H	D7	'RST 16'
	0018_H	DF	'RST 24'
	0020_H	E7	'RST 32'
	0028_H	EF	'RST 40'
	0030_H	F7	'RST 48'
	0038_H	FF	'RST 56'


SOURCE PORT ADDRESS
IMMED.
REG. INDIR.
(n)
(C)
INPUT DESTINATION
INPUT 'IN'
REG ADDRESSING
A
DB n
ED 78
B
ED 40
C
ED 48
D
ED 50
E
ED 58
H
ED 60
L
ED 68
'INI' – INPUT & Inc HL, Dec B
'INIR'– INP, Inc HL, Dec B, REPEAT IF B≠0
'IND'–INPUT & Dec HL, Dec B
'INDR'–INPUT, Dec HL, Dec B, REPEAT IF B≠0
REG, INDIR
(HL)
ED A2
ED B2
ED AA
ED BA
BLOCK INPUT COMMANDS

SOURCE

			REGISTER							REG. IND.
			A	B	C	D	E	H	L	(HL)
'OUT'	IMMED.	(n)	D3 n							
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED BB

BLOCK OUTPUT COMMANDS

PORT DESTINATION ADDRESS

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

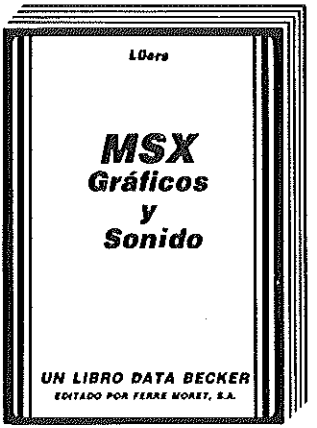
Instruction	C	Z	P/V	S	N	H	Comments
ADD A, s; ADC A,s	↕	↕	V	↕	0	↕	8-bit add or add with carry
SUB s; SBC A, s, CP s, NEG	↕	↕	V	↕	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	0	↕	P	↕	0	1	Logical operations
OR s; XOR s	0	↕	P	↕	0	0	And set's different flags
INC s	•	↕	V	↕	0	↕	8-bit increment
DEC m	•	↕	V	↕	1	↕	8-bit decrement
ADD DD, ss	↕	•	•	•	0	X	16-bit add
ADC HL, ss	↕	↕	V	↕	0	X	16-bit add with carry
SBC HL, ss	↕	↕	V	↕	1	X	16-bit subtract with carry
RLA; RLCA, RRA, RRCA	↕	•	•	•	0	0	Rotate accumulator
RL m; RLC m; RR m; RRC m SLA m; SRA m; SRL m	↕	↕	P	↕	0	0	Rotate and shift location m
RLD, RRD	•	↕	P	↕	0	0	Rotate digit left and right
DAA	↕	↕	P	↕	•	↕	Decimal adjust accumulator
CPL	•	•	•	•	1	1	Complement accumulator
SCF	1	•	•	•	0	0	Set carry
CCF	↕	•	•	•	0	X	Complement carry
IN r, (C)	•	↕	P	↕	0	0	Input register indirect
INI; IND; OUTI; OUTD	•	↕	X	X	1	X	Block input and output
INIR; INDR; OTIR; OTDR	•	1	X	X	1	X	Z = 0 if B ≠ 0 otherwise Z = 1
LDI, LDD	•	X	↕	X	0	0	Block transfer instructions
LDIR, LDDR	•	X	0	X	0	0	P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI, CPIR, CPD, CPDR	•	↕	↕	↕	1	X	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	•	↕	IFF	↕	0	0	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	•	↕	X	X	0	1	The state of bit b of location s is copied into the Z flag
NEG	↕	↕	V	↕	1	↕	Negate accumulator

The following notation is used in this table:

Symbol	Operation
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from into bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>
nn	16-bit value in range <0, 65535>
m	Any 8-bit location for all the addressing modes allowed for the particular instruction.

MSX EL MANUAL ESCOLAR
P.V.P. 2.800,- ptas.

Escrito para alumnos de los últimos cursos de EGB y de BUP, este libro contiene muchos programas para resolver problemas y de aprendizaje, descritos de una forma muy completa y fácil de comprender. Teorema de Pitágoras, progresiones geométricas, escritura cifrada, crecimiento exponencial, verbos irregulares, igualdades cuadráticas, movimiento pendular, estructura de moléculas, cálculo de interés y muchas cosas más.



MSX GRAFICOS Y SONIDOS, 250 pág.
P.V.P. 2.800,- ptas.

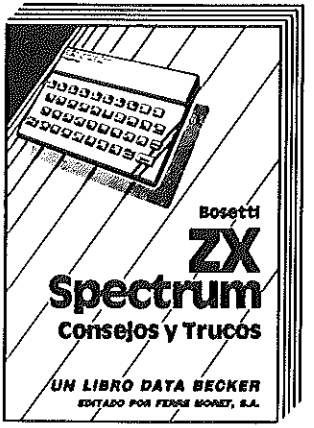
Las computadoras MSX no sólo ofrecen una relación precio/rendimiento sobresaliente, sino que también poseen unas cualidades gráficas y de sonido excepcionales. Este libro expone las posibilidades de los MSX de forma completa y fácil. El texto se completa con numerosos y útiles programas ejemplo.

MSX PROGRAMAS Y UTILIDADES, 1985, 194 pág.
P.V.P. 2.200,- ptas.

El libro contiene una amplia colección de importantes programas que abarcan, desde un desensamblador hasta un programa de clasificaciones deportivas. Juegos superemocionantes y aplicaciones completas. Los programas muestran además importantes consejos y trucos para la programación. Estos programas funcionan en todos los ordenadores MSX, así como en el SPECTROVIDEO 318 328. ESTRACTO DEL CONTENIDO: Volcado memoria hexadecimal. Editor gráficos. Editor de sonido. Escritura de ordenador. Lista referencia de variables. Calendario. Desensamblador. ADMINISTRACION de una colección de discos L.P. HOLLOW - JUEGO DE LAS CEREZAS. DIAGRAMAS DE BARRAS. TABLAS DEPORTIVAS.

ZX SPECTRUM EL MANUAL ESCOLAR
P.V.P. 2.200,- ptas.

Escrito para alumnos de los últimos cursos de EGB y de BUP, este libro contiene muchos programas para resolver problemas y de aprendizaje, descritos de una forma muy completa y fácil de comprender. Teorema de Pitágoras, progresiones geométricas, escritura cifrada, crecimiento exponencial, verbos irregulares, igualdades cudráticas, movimiento pendular, estructura de moléculas, cálculo de interés y muchas cosas más.



ZX SPECTRUM CONSEJOS Y TRUCOS, 211 pág.
P.V.P. 2.200,- ptas.

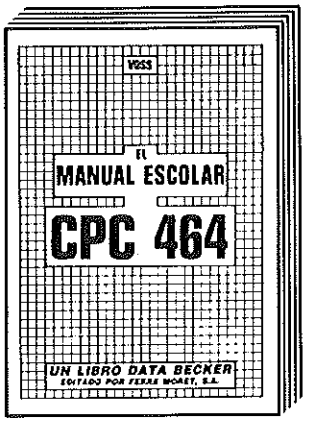
Una interesante colección se sugestivas ideas y soluciones para la programación y utilización de su ZX ESPECTRUM. Aparte de muchos peeks, pokes y USRs hay también capítulos completos para, entre otros, entrada de datos asegurado sin bloqueo de ordenador, posibilidades de conexión y utilización de microdrives y lápices ópticos programas para la representación de diagramas de barra y de tarta, el modo de utilizar óptimamente ROM y RAM.

METODOLOGIA DE LA PROGRAMACION
P.V.P. 2.200,- ptas.

El primer libro recomendado para escuelas de enseñanza de informática y para aquellas personas que quieren aprender la programación. Cubre las especificaciones del Ministerio de Educación y Ciencia para Estudios de Informática. Realizado por un alto mando del ejército Español, un Dr. Ingeniero y Diplomado en Informática y profesor de la UNED y por un oficial técnico especialista en informática de gestión. Utilizado en todos los institutos politécnicos del ejército español. Es un seguro candidado a ediciones en lengua inglesa, alemana y francesa. Es el primer libro que introduce a la lógica del ordenador. Es un elemento de base que sirve como introducción para la programación en cualquier otro lenguaje. No se requieren conocimientos de programación ni siquiera de informática. Abarca desde los métodos de programación clásicos a los más modernos.

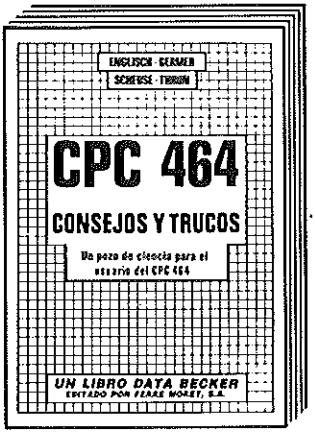
64 EN EL CAMPO DE LA TECNICA Y LA CIENCIA, 296 pág.
P.V.P. 2.800,- ptas.

Ofrece un campo fascinante y amplio de problemáticas científicas. Para esto el libro contiene muchos listados interesantes: Análisis de Fornier y síntesis, análisis de redes, exactitud de cálculo, formateado de números, cálculo del valor PH, sistemas de ecuaciones diferenciales, modelo ladrón presa, cálculo de probabilidad, medición de tiempo, integración, etc.



CPC-464 EL LIBRO DEL COLEGIO
P.V.P. 2.200,- ptas.

Escrito para alumnos de los últimos cursos de EGB y de BUP, este libro contiene muchos programas para resolver problemas y de aprendizaje, descritos de una forma muy compleja y fácil de comprender. Teorema de Pitágoras, progresiones geométricas, escritura cifrada, crecimiento exponencial, verbos irregulares, igualdades cuadráticas, movimiento pendular, estructura de moléculas, cálculo de interés y muchas cosas más.



CPC-464 CONSEJOS Y TRUCOS
P.V.P. 2.200,- ptas.

Ofrece una colección muy interesante de sugerencias, ideas y soluciones para la programación y utilización de su CPC-464: Desde la estructura del hardware, sistema de funcionamiento - Tokens Basic, dibujos con el joystick, aplicaciones de ventanas en pantalla y otros muchos interesantes programas como el procesamiento de datos, editor de sonidos, generador de caracteres, monitor de código máquina hasta listados de interesantes juegos.

TODO SOBRE EL FLOPPY 1541, 482 pág.
P.V.P. 3.200,- ptas.

La obra Standard del floppy 1541, todo sobre la programación en disquettes desde los principiantes a los profesionales, además de las informaciones fundamentales para el DOS, los comandos de sistema y mensajes de error, hay varios capítulos para la administración práctica de ficheros con el FLOPPY, amplio y documentado Listado del Dos. Además un filón de los más diversos programas y rutinas auxiliares, que hacen del libro una lectura obligada para los usuarios del Floppy.

MANTENIMIENTO Y REPARACION DEL FLOPPY 1541,
200 pág.
P.V.P. 2.800,- ptas.

Saberse apañar uno mismo, ahorra tiempo, molestias y dinero, precisamente problemas como el ajuste del floppy o reparaciones de la platina se pueden arreglar a menudo con medios sencillos. Instrucciones para eliminar la mayoría de perturbaciones, listas de piezas de recambio y una introducción a la mecánica y a la electrónica de la unidad de disco, hay también indicaciones exactas sobre herramientas y material de trabajo. Este libro hay que considerarlo en todos sus aspectos como efectivo y barato.

EL MANUAL DEL CASSETTE, 190 pág.
P.V.P. 1.600,- ptas.

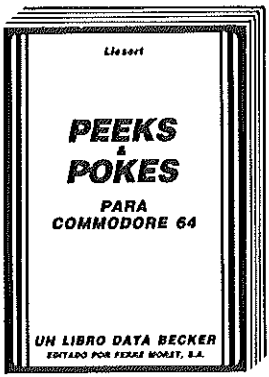
Un excelente libro, que le mostrará todas las posibilidades que le ofrece su grabadora de casettes. Describe detalladamente, y de forma comprensible, todo sobre el Datassette y la grabación en cassette. Con verdaderos programas fuera de serie: Autostart, Catálogo (¡busca y carga automáticamente!), backup de y a disco, SAVE de áreas de memoria, y lo más sorprendente: un nuevo sistema operativo de cassette con el 10-20 veces más rápido FastTape. Además otras indicaciones y programas de utilidad (ajuste de cabezales, altavoz de control).

ROBOTICA PARA SU COMMODORE 64, 230 pág.
P.V.P. 2.800,- ptas.

En el libro de los robots se muestran las asombrosas posibilidades que ofrece el CBM 64, para el control y la programación, presentadas con numerosas ilustraciones e intuitivos ejemplos. El punto principal: Cómo puede construirse uno mismo un robot sin grandes gastos. Además, un resumen del desarrollo histórico del robot y una amplia introducción a los fundamentos cibernéticos.
Gobierno del motor, el modelo de simulación, interruptor de pantalla, el Port-Usuario cómodo del modelo de simulación, Sensor de infrarrojos, concepto básico de un robot, realimentación unidad cibernética, Brazo prensor, Oír y ver.

MANUAL ESCOLAR PARA SU COMMODORE 64, 351 pág.
P.V.P. 2.800,- ptas.

Este libro, escrito especialmente para escolares de grado medio y superior, contiene muchos interesantes programas de aprendizaje para solucionar problemas, descritos detalladamente y de manera fácilmente comprensible. Facilitan un aprendizaje intensivo y ameno, con, entre otros, los siguientes temas: Teorema de pitágoras, progresiones geométricas, palanca mecánica, crecimiento exponencial, verbos irregulares, ecuaciones de segundo grado, movimientos de péndulo, formación de moléculas, aprendizaje de vocablos, cálculo de interés y su capitalización. Una corta repetición de los elementos BASIC más importantes y una introducción a los rasgos esenciales del análisis de problemas, entre otros, completan el conjunto.

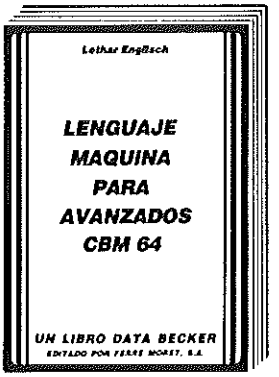


PEEKS y POKES, 177 pág.
P.V.P. 1.600,- ptas.

Con importantes comandos PEEK y POKE se pueden hacer también desde el Basic muchas cosas, para las que se necesitarían normalmente complejas rutinas en lenguaje máquina. Este libro explica de manera sencilla el manejo de PEEKs y POKEs. Con una enorme cantidad de POKEs importantes y su posible aplicación. Para ello se explica perfectamente la estructura del Commodore 64: Sistema operativo, interpretador, página cero, apuntadores y stacks, generador de caracteres, registros de sprites, programación de interfaces, desactivación del interrupt. Además una introducción al lenguaje máquina. Muchos programas ejemplo.

LENGUAJE MAQUINA PARA
COMMODORE 64, 1984, 201 pág.
P.V.P. 2.200,- ptas.

¡Por fin una introducción al código máquina fácilmente comprensible! Estructura y funcionamiento del procesador 6510, introducción y ejecución de programas en lenguaje máquina, manejo del ensamblador, y un atractivo muy especial: ¡un simulador de paso a paso escrito en BASIC!



LENGUAJE MAQUINA PARA AVANZADOS
CBM 64, 1984, 206 pág.
P.V.P. 2.200 ptas.

¿Ud. ha logrado iniciarse en código máquina? Entonces el «nuevo English» le enseñará cómo convertirse en un profesional. Naturalmente con muchos programas ejemplo, rutinas completas en código máquina e importantes consejos y trucos para la programación en lenguaje máquina y para el trabajo con el sistema operativo.

EL CONTENIDO

El libro del Lenguaje Máquina para el MSX está creado para todos aquellos a quienes el BASIC se les ha quedado pequeño en cuanto a rendimiento y velocidad. Desde las bases para la programación en Lenguaje Máquina, pasando por el método de trabajo del Procesador Z-80 y una exacta descripción de sus órdenes, hasta la utilización de rutinas del sistema, todo ello ha sido explicado en detalle e ilustrado con múltiples ejemplos en este libro.

El libro contiene, además, como programas de aplicación, un ensamblador, un desensamblador y un monitor.

¡Así es cómo se facilita el acceso al Lenguaje Máquina!

ESTE LIBRO LO HA ESCRITO:

Holger Dullin y Hardy Strassenburg, estudiantes universitarios (de Biología e Informática) y expertos programadores, que ya han conseguido prestigio como autores de libros especializados en ordenadores-Z-80.

ISBN 84-86437-27-10