Colección Electrónica

CODIGO DE MAQUINA PARA PRINCIPIANTES

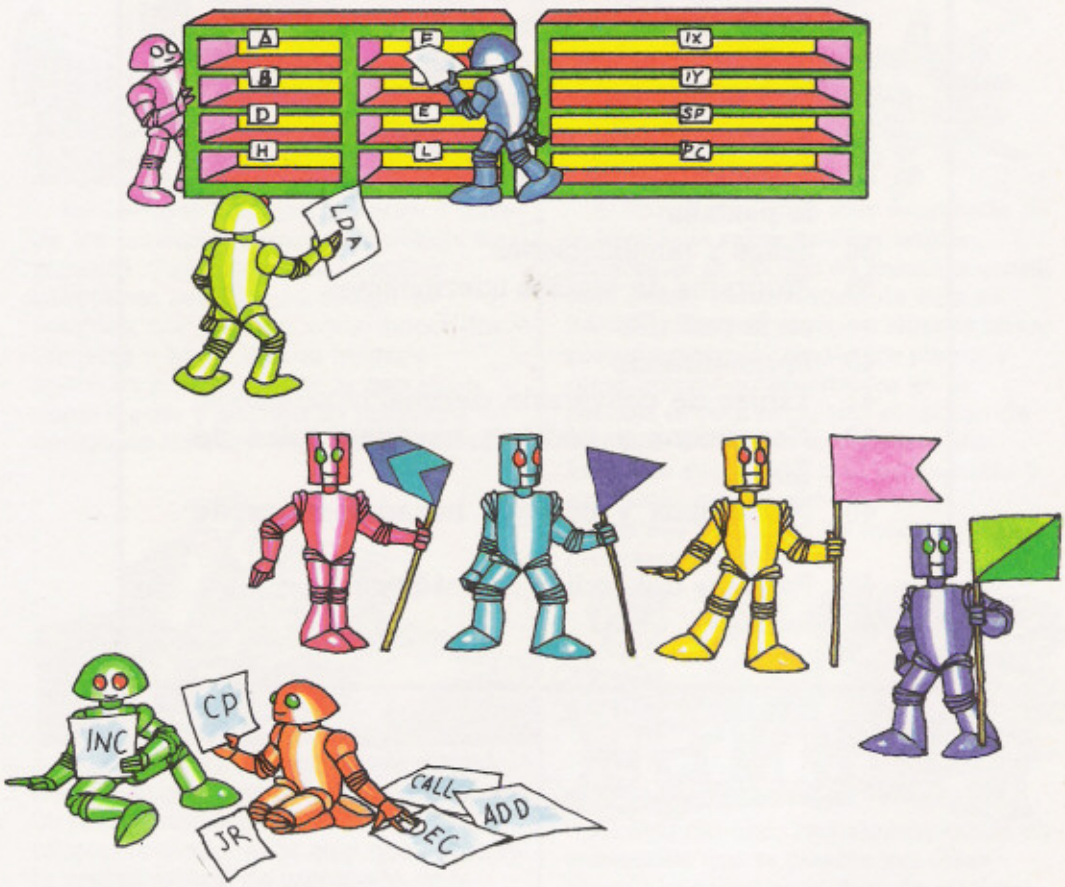
PARA EL Z80 Y 6502

Ediciones Plesa SM

Colección Electrónica

CODIGO DE MAQUINA PARA PRINCIPIANTES

Por Lisa Watts y Mike Wharton



Contenido

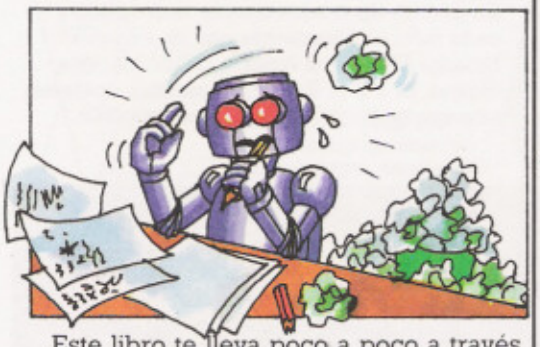
© 1983 Usborne Publishing Ltd.
© 1986 Publicaciones y Ediciones Lagos, S.A. (PLESA)
Sestao, 1. Pinto (Madrid)

Impreso en España, Printed in Spain, por MELSA
Depósito Legal: M-11473-1986 I.S.B.N.: 84-7374-147-1

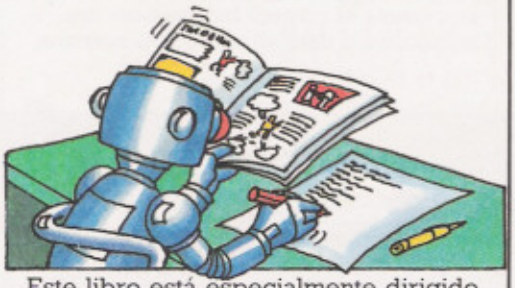
No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright.

Sobre este libro

Este libro es una sencilla guía que enseña paso a paso a programar en código de máquina. El código de máquina es el código en el que la computadora realiza todas sus funciones y los programas escritos en código de máquina se ejecutan mucho más rápidamente y ocupan menos memoria que los programas escritos en BASIC. No obstante, un programa en código de máquina es mucho más difícil de escribir y de entender que uno igual en BASIC.



Este libro te lleva poco a poco a través de los principios básicos del código de máquina. Te enseña cómo escribir programas sencillos en código de máquina; por ejemplo, cómo sumar dos números o hacer que un mensaje aparezca intermitente en la pantalla o cómo cargar y ejecutar un programa en código de máquina en tu computadora.



Este libro está especialmente dirigido a usuarios que posean computadoras con un microprocesador Z80 ó 6502.* El microprocesador es el chip que contiene la unidad central de procesado de la computadora, por lo que las computadoras con diferentes microprocesadores entienden diferentes códigos de máquina. Por tanto, todas las computadores con un microprocesador idéntico usan el mismo código de máquina.

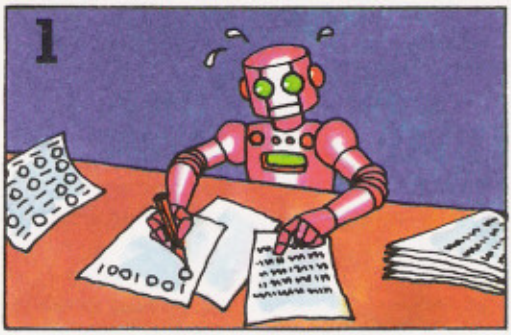
El código de máquina es difícil y requiere mucho trabajo, ya que posee muchas reglas que hay que seguir y gran cantidad de detalles que hay que recordar. No te asustes si al principio te resulta imposible de entender. Suele desmoralizar, ya que no te es posible leer y entender un programa en código de máquina —está compuesto únicamente de sucesiones de letras y números—. Además, los errores son muy difíciles de localizar y suelen tener consecuencias fatales si te las saltas. Debes tener especial cuidado, ser metódico y comprobar todo dos o tres veces.

A menos que estés muy interesado no te hace falta saber escribir largos programas en código de máquina —casi todo puede hacerse igual de bien en BASIC—. Sin embargo, en algunas tareas como hacer más rápidos los juegos y crear efectos sorprendentes en la pantalla te será necesario el código de máquina. Este libro te enseña cómo hacer tus programas más interesantes y excitantes incluyendo pequeñas subrutinas en código de máquina a tus programas en BASIC.

Al final del libro hay algunas tablas de conversión que te pueden ser útiles cuando escribas en código de máquina, así como una lista de palabras en código de máquina para explicar toda la jerga. También encontrarás multitud de problemas e ideas para escribir programas cortos con sus respuestas en la página 44.

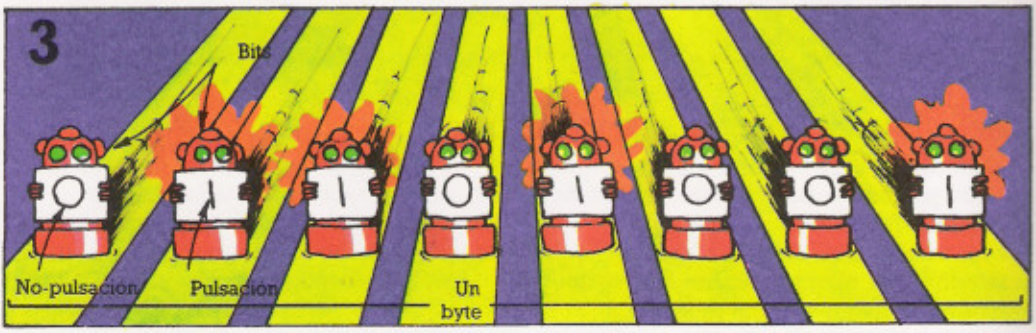
*El Spectrum y el ZX81 (Timex 2000 y 1000) usan el microprocesador Z80 y el VIC 20, el BBC, las computadoras Atari y el Oric usan el 6502. El Comodore 65 usa el 6510, pero entiende el código del 6502.

¿Qué es el código de máquina?



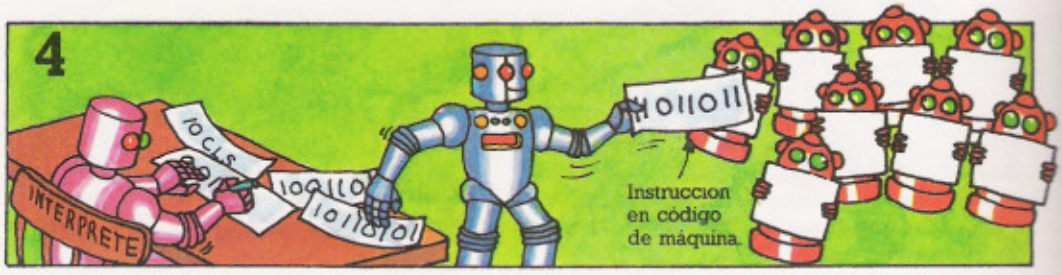
El código de máquina es el código en el que la computadora realiza todas sus funciones. Cuando das a la computadora un programa en BASIC, todas las instrucciones y datos son traducidos a código de máquina dentro de la computadora.

En código de máquina cada instrucción y cada dato se representa por un número binario. El sistema binario utiliza sólo dos dígitos, el 1 y el 0. Puedes escribir cualquier número en binario usando unos y ceros.*



Dentro de la computadora, los números binarios están representados por pulsaciones eléctricas, siendo la pulsación el 1 y la no-pulsación el 0. Las pulsaciones y no-pulsaciones se denominan «bits», que quiere decir dígitos binarios.

Los bits fluyen a través de la computadora en grupos de ocho, denominados «bytes». Cada byte de pulsaciones y no-pulsaciones representa el número binario para una instrucción o dato en código de máquina.



Cada una de las tareas que realiza una computadora, como sumar dos números o borrar la pantalla, supone una secuencia de varias instrucciones en código de máquina. Cuando introduces una instrucción en BASIC, un programa especial denominado «intérprete» traduce tu instrucción a instrucciones en código de máquina que la computadora entiende.

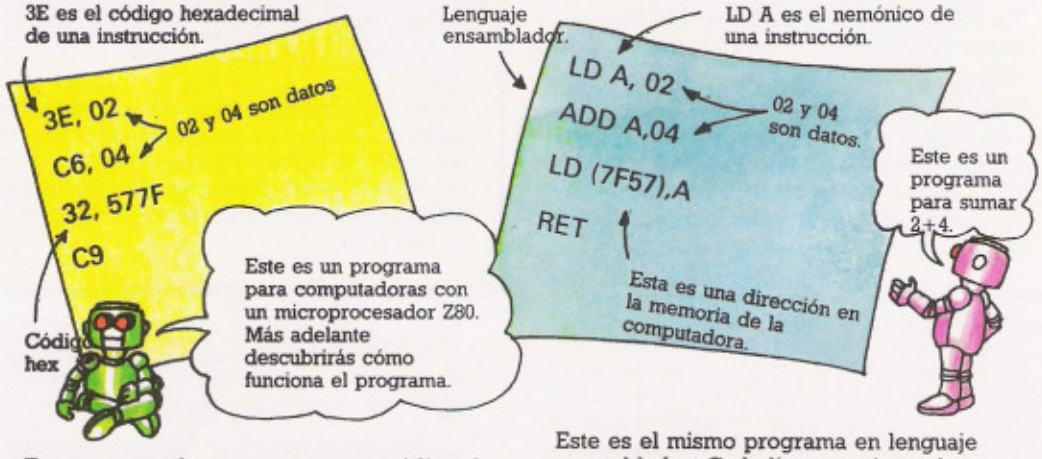
El término código de máquina se usa también para referirse a programas escritos en una forma que está mucho más cerca del código de la computadora que el BASIC. En un programa en código de máquina debes dar a la computadora todas las diferentes instrucciones que necesite para llevar a cabo una labor como borrar la pantalla.

*Puedes encontrar más sobre el sistema binario en la página 28.

Programar en código de máquina

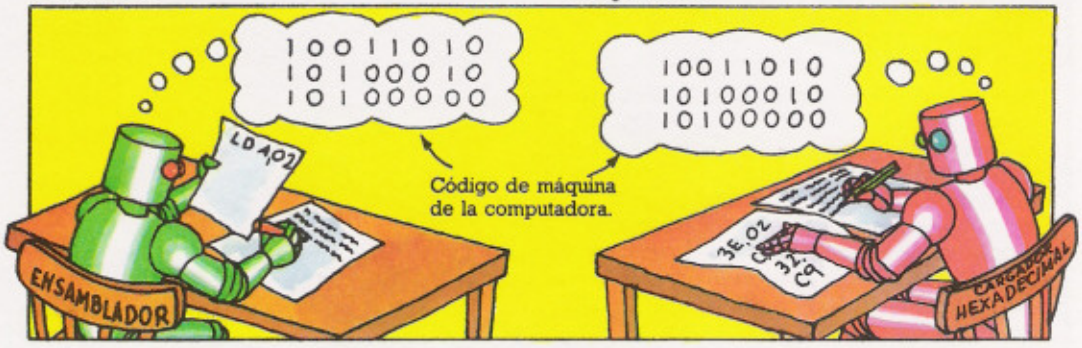
Existen varias formas de escribir programas en código de máquina. Podrías escribir todas las instrucciones en números binarios, pero esto resultaría realmente agotador. Para evitar esto puedes usar otro sistema conocido como hex —abreviatura de hexadecimal—. Una vez que te acostumbres, resulta mucho más fácil de usar que el binario.

Los programas en código de máquina también pueden escribirse en un código denominado «lenguaje ensamblador». En el lenguaje ensamblador cada instrucción se representa mediante un «nemónico» —palabra corta que suena como la instrucción que representa.



Esto es parte de un programa en código de máquina hexadecimal. El sistema de números hexadecimal posee dieciséis dígitos, por lo que usa los símbolos 0-9 y A-F para representar los números del 0 al 15. (Más adelante encontrarás más cosas sobre el sistema hexadecimal.) El número hexadecimal al principio de cada línea del programa es una instrucción (ej.: 3E). Es el equivalente hexadecimal del código binario de esa instrucción.

Este es el mismo programa en lenguaje ensamblador. Cada línea contiene el nemónico de una instrucción que equivale al número hexadecimal de la misma línea del programa de la izquierda. Por ejemplo, el nemónico L DA (viene de «load A» que quiere decir «cargar A») significa lo mismo que el número hexadecimal 3E. En ambos programas, cada línea contiene una instrucción que equivale a una sola instrucción en el código de máquina de la computadora.

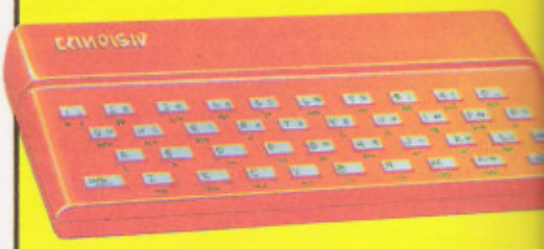


Para introducir en la computadora un programa en lenguaje ensamblador necesitas un programa especial conocido como «ensamblador» que traduce los nemónicos al código de máquina de la computadora. Algunas computadoras tienen incorporado un ensamblador, para otras puedes comprarlo en cinta y cargarlo en la memoria de la computadora. El ensamblador te permitirá escribir un programa en código de máquina usando los nemónicos del lenguaje ensamblador (son más fáciles de recordar que los números) que serán traducidos a hexadecimal antes de ser introducidos en la computadora. Algunas computadoras aceptan números hexadecimales, en otros tendrás que usar un pequeño programa llamado «cargador hexadecimal», que traduce los números hexadecimales. En la página 24 encontrarás un cargador hexadecimal que puedes usar para cargar los programas en código de máquina de este libro.

Empezar a conocer tu computadora

Cuando programas una computadora en código de máquina debes decirle lo que debe hacer en cada etapa: dónde buscar y almacenar los datos, cómo escribir en la pantalla, etc. (cuando trabajas en BASIC, unos programas especiales situados en el interior de la computadora se encargan de hacer todo esto por ti). Para saber darle a la computadora las instrucciones correctas en cada momento es conveniente saber qué está sucediendo dentro de tu computadora. Los dibujos de estas dos páginas te muestran las partes principales dentro de una computadora personal y sus funciones. En las páginas siguientes encontrarás más cosas sobre ellas.

Dentro de una computadora



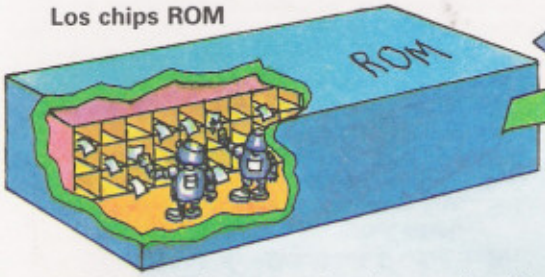
Dentro del teclado de una microcomputadora existe un tablero con un circuito impreso. Este posee pistas metálicas impresas a lo largo de los cuales fluye la corriente eléctrica. Unido al tablero del circuito impreso existen un serie de chips.

Función de los chips

Este dibujo muestra el trabajo realizado por los diferentes chips que hay dentro de la computadora. Los mensajes fluyen entre los chips en forma de bytes; es decir grupos de ocho señales de pulsaciones y no-pulsaciones, que representan los datos y las instrucciones.

Reloj

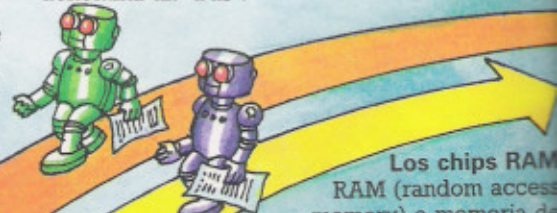
Los chips ROM



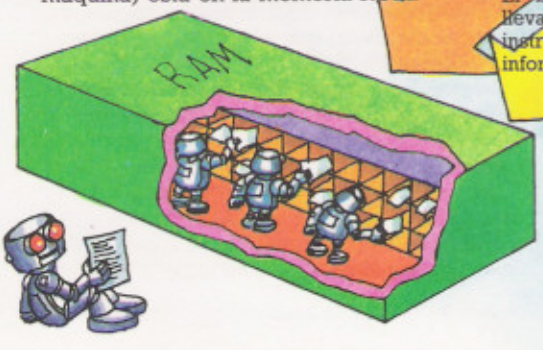
El "bus de direccionamiento" lleva direcciones de memoria.

Los bytes en el código de la computadora fluyen entre los chips a lo largo de las pistas del tablero del circuito impreso. Hay tres sistemas separados de pistas que llevan bytes para diferentes funciones. Cada sistema de pistas se denomina un "bus".

ROM (read only memory) es la memoria sólo de lectura. Las instrucciones en código de máquina que indican a la computadora lo que debe hacer están almacenadas en los chips ROM. Se denomina sólo de lectura, ya que lo único que puede hacer la computadora es leer la información que hay en el ROM; no puede almacenar nada en ella. En casi todas las computadoras personales el intérprete (el programa que traduce BASIC a código de máquina) está en la memoria ROM.

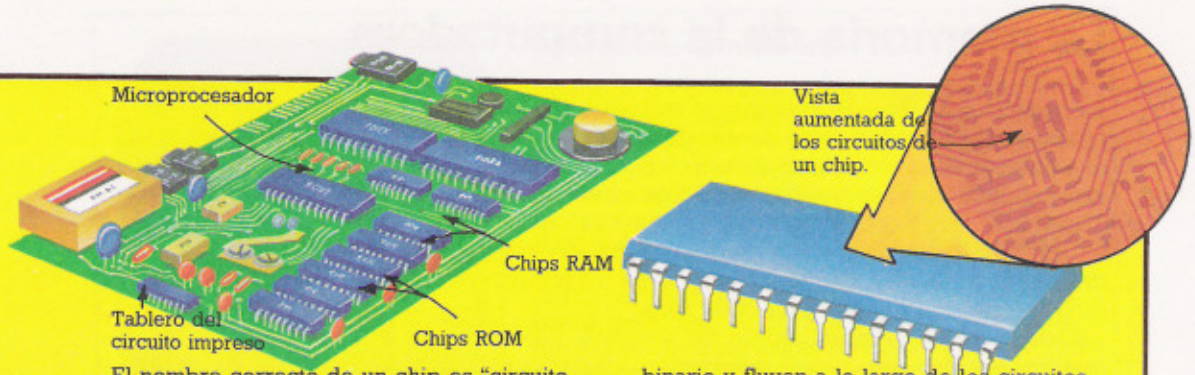


El "bus de datos" lleva bytes de instrucciones o información.



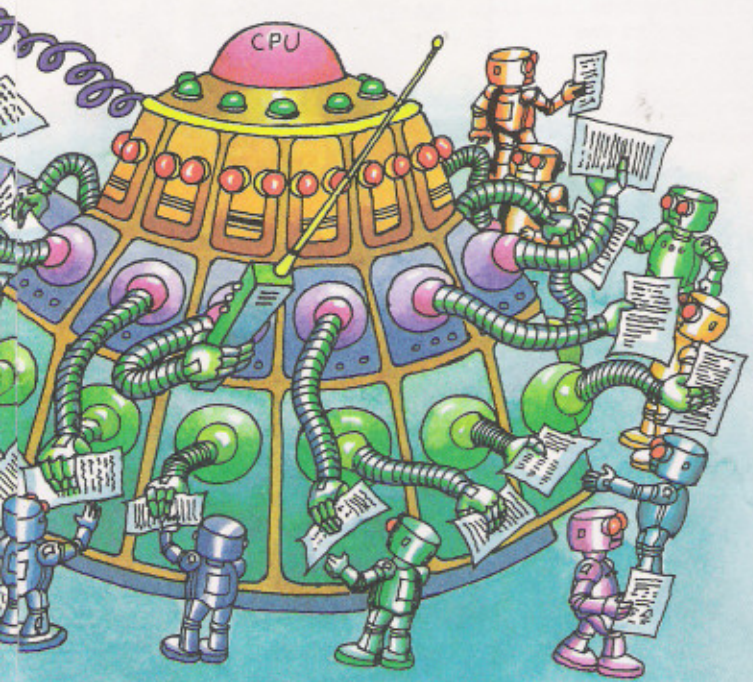
Los chips RAM

RAM (random access memory) o memoria de acceso directo. Es donde se almacenan los programas que introduces en la computadora mientras los está usando la computadora. Se denomina memoria de acceso directo porque la computadora puede acceder a cualquier información almacenada en cualquier parte de la memoria. Cuando apagas la computadora, la información almacenada en la memoria RAM desaparece inmediatamente.



El nombre correcto de un chip es "circuito integrado", ya que dentro de cada chip existen circuitos eléctricos microscópicos. Todo el trabajo de la computadora lo realizan chorros de pulsaciones que representan las instrucciones en código binario y fluyen a lo largo de los circuitos de los chips. Existen diferentes chips para realizar funciones diversas. El trabajo realizado por los diferentes tipos de chips se muestra en el dibujo inferior.

Reloj
Este es un reloj de cuarzo que emite millones de pulsaciones por segundo, regulando el flujo de pulsaciones en el interior de la computadora.



El microprocesador
El chip microprocesador posee la unidad central de procesado de la computadora, es decir, el CPU. Este es el lugar donde se realiza todo el trabajo de la computadora. El CPU realiza cálculos, compara datos, toma decisiones y coordina todas las demás actividades dentro de la computadora. La información que dice al CPU lo que debe hacer está almacenada en el ROM.

El microprocesador de tu computadora

Existen varios tipos diferentes de chips microprocesadores aunque todas las computadoras personales utilizan un reducido número. Los modelos más usados son el Z80, que encontramos en computadoras Sinclair (Timex) y el 6502, usado por el Oric, el BBC el Vic 20 y muchas otras. Otros microprocesadores pueden ser el 6809, usado en la computadora Dragón, y el 9900, usado en la TI99/4.

Cada modelo diferente de microprocesador usa diferentes instrucciones de código de máquina, pero todas las computadoras que poseen el mismo microprocesador entienden el mismo código de máquina. También existen varias versiones diferentes de cada chip. Por ejemplo, el Z80A es una versión más rápida del Z80, y el 6502A y el 6510 (usado en el Commodore 64) son versiones del 6502. Usan las mismas instrucciones de código de máquina que las que usa el chip del que fueron desarrollados. Este libro está escrito especialmente para computadoras con microprocesadores que entienden el código de máquina del Z80 y del 6502.

La memoria de la computadora

La manera más fácil de imaginar la memoria de la computadora es como un gran número de pequeñas cajas dentro de cada una de las cuales hay un byte, es decir, una instrucción o información en código de máquina. Cada caja de la memoria se denomina "ubicación" y cada ubicación viene determinada por un número llamado "dirección" que permite que la computadora encuentre cualquier caja en la memoria.

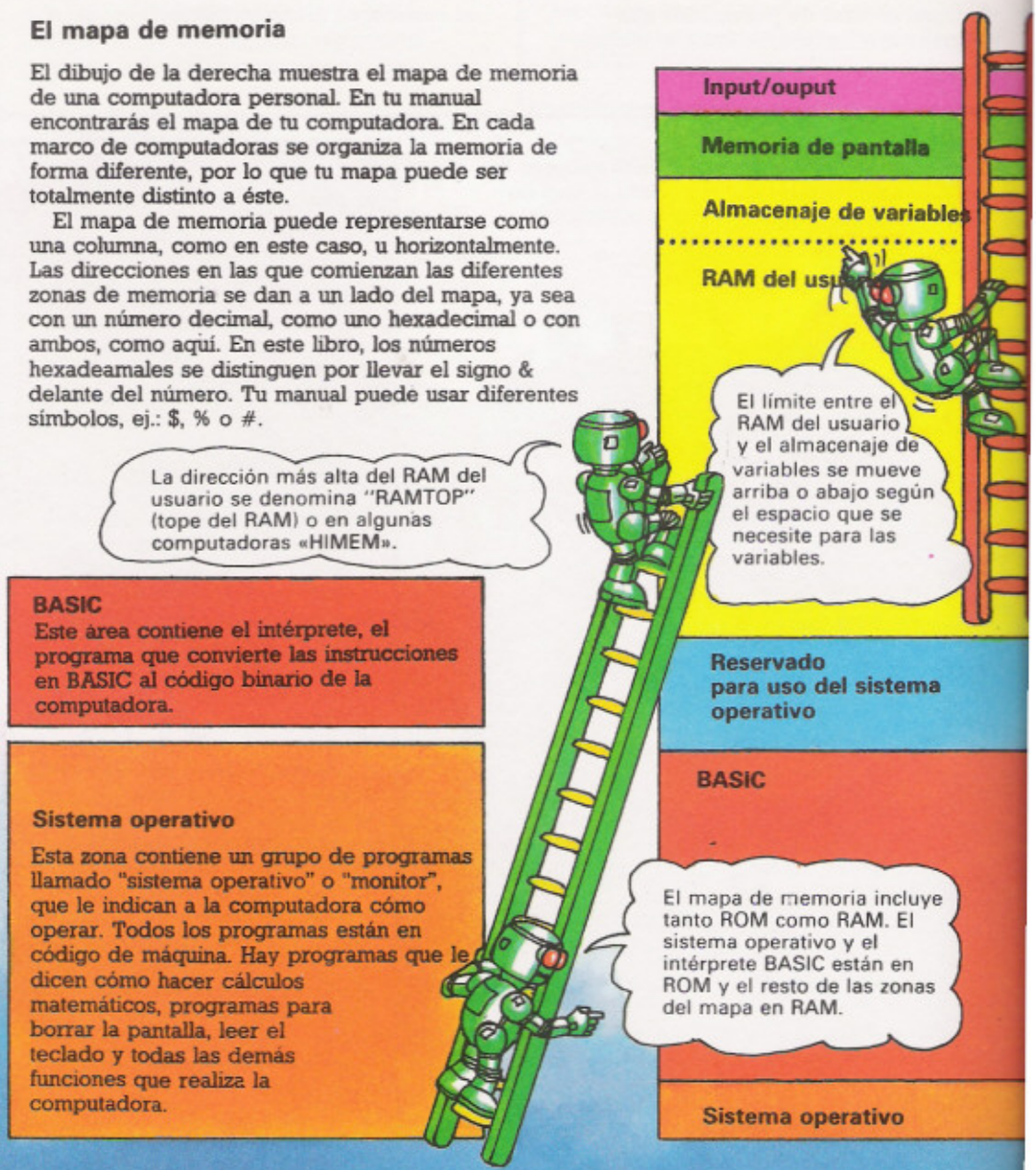
Las diferentes zonas de la memoria se usan para almacenar información para diferentes funciones. Se denomina "mapa de memoria" a la tabla que te indica las direcciones donde comienza cada zona.

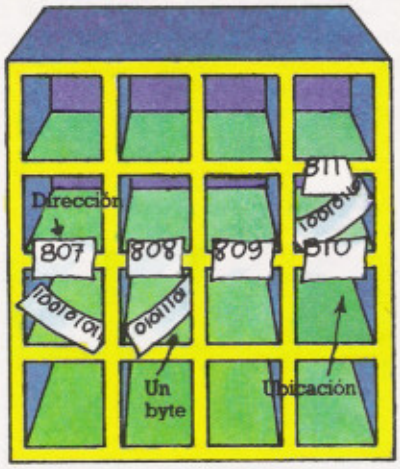
Cuando programas en código de máquina debes decirle a la computadora dónde encontrar o almacenar cada instrucción o información, dándole la dirección de una ubicación de memoria. Incluso tienes que decirle dónde almacenar el programa en código de máquina, por lo que te será necesario conocer el mapa de memoria.

El mapa de memoria

El dibujo de la derecha muestra el mapa de memoria de una computadora personal. En tu manual encontrarás el mapa de tu computadora. En cada marco de computadoras se organiza la memoria de forma diferente, por lo que tu mapa puede ser totalmente distinto a éste.

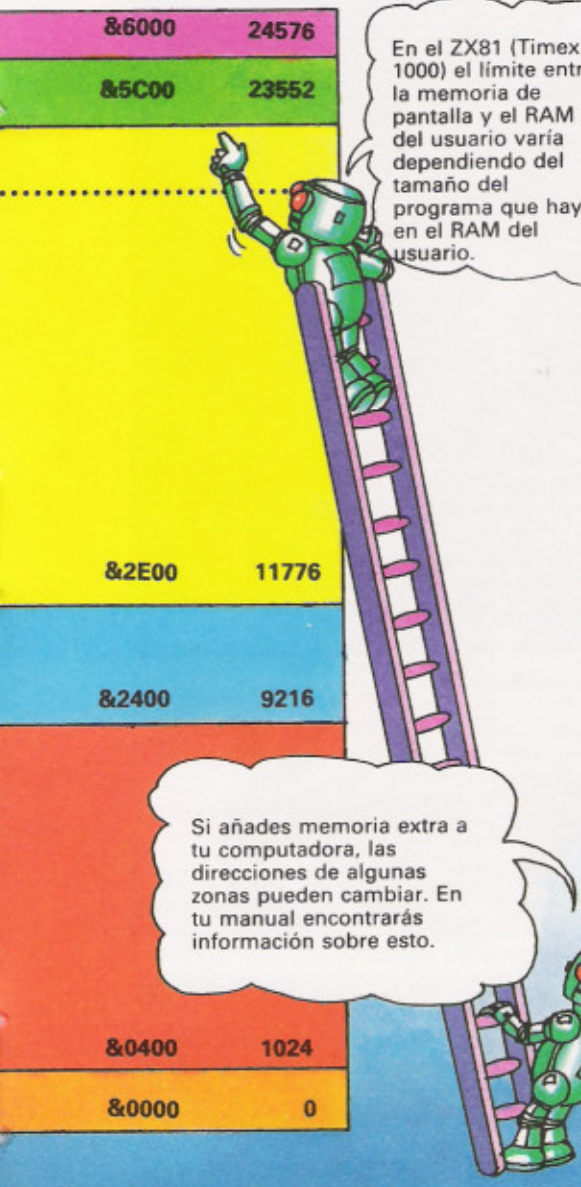
El mapa de memoria puede representarse como una columna, como en este caso, u horizontalmente. Las direcciones en las que comienzan las diferentes zonas de memoria se dan a un lado del mapa, ya sea con un número decimal, como uno hexadecimal o con ambos, como aquí. En este libro, los números hexadeamales se distinguen por llevar el signo & delante del número. Tu manual puede usar diferentes símbolos, ej.: $, % o #.





Direcciones de memoria

Dentro de la computadora, las direcciones de memoria están representadas por dos bytes de código de máquina, es decir, 16 pulsaciones y no pulsaciones o "bits". El máximo de memoria que puede tener una microcomputadora que use el microprocesador Z80 o el 6502 es de 64K (ROM y RAM combinados). Esto es debido a que el mayor número que puedes formar con 16 dígitos binarios es 65535, por lo que ésta es la dirección más alta que puede tener. Esta da 65535 ubicaciones numeradas del 0 al 65535. Cada ubicación contiene un byte, 1024 bytes forman un kilobyte (K) y 65536 equivalen a 64K (65536 ÷ 1024 = 64).



Input/output
Estas ubicaciones de memoria tienen relación con las conexiones de input/ output (entrada/salida) de la computadora.

Memoria de Pantalla
También conocida como archivo de visualización, es la parte de la memoria que contiene la información visualizada en la pantalla. Cualquier información almacenada en la memoria de pantalla queda visualizada automáticamente en la pantalla. Muchas microcomputadoras tienen un mapa de memoria de visualización en la que cada ubicación de la memoria de pantalla contiene la información para una determinada posición en la pantalla.

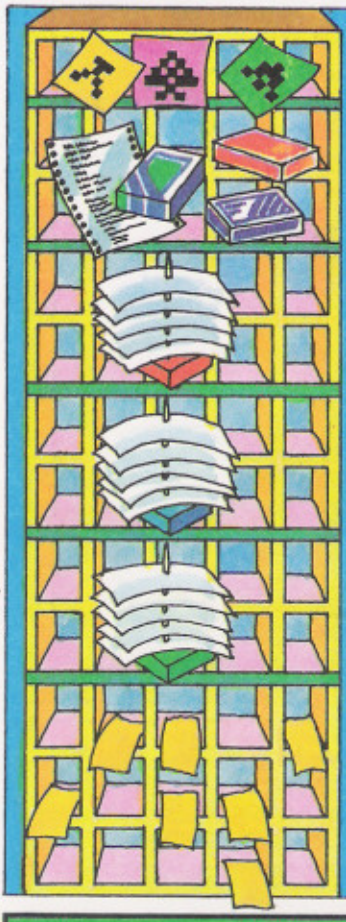
RAM del usuario
Esto es la zona donde se alacenan los programas que introduces. Los datos para las variables y las matrices se almacenan en la parte superior del RAM del usuario.

Reservado para uso del sistema operativo

Estas ubicaciones RAM las usa la computadora para seguir la pista de todo lo que sucede mientras ejecuta un programa. Por ejemplo, la información sobre la posición del cursor, el color en cada momento de la pantalla, qué tecla está siendo apretada y que número de línea está siendo ejecutada. Está dividido en pequeñas zonas para las diferentes funciones. Algunas computadoras tienen un segundo mapa de esta zona. En la página siguiente encontrarás más cosas sobre esta zona.

Dentro de la zona de trabajo de la computadora

Este dibujo es una representación de la zona de la memoria de la computadora reservada para uso del sistema operativo. Posiblemente encontrarás otro mapa detallado de esta zona en tu manual, o bien una lista de varias direcciones y sus usos. En algunas computadoras (ej: Sinclair/Timex), las ubicaciones usadas por el sistema operativo no están agrupadas sino que están distribuidas por toda la memoria.



Gráficas definidas por el usuario
Si creas tus propios caracteres gráficos se almacenarán aquí.

Buffers
Estos son almacenes temporales para contener los datos que llegan del teclado o que se mandan a una impresora o cassette.

Pila para lenguaje de máquina
También llamada pila del procesador: El CPU utiliza estas ubicaciones para almacenar direcciones, mientras está trabajando con un programa en código de máquina.

Pila BASIC
También llamada pila GOSUB. Se utiliza para almacenar los números de línea usados con las instrucciones GOSUB y GOTO del BASIC.

Pila de calculadora
Este el el almacén temporal del CPU para los números que se utilizan en operaciones.

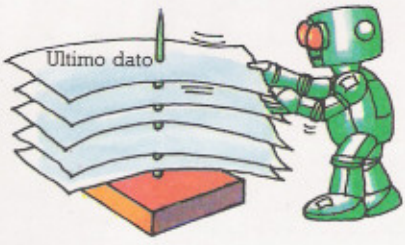
Sistema de variables
Son una serie de ubicaciones de memoria donde el CPU almacena información sobre lo que está sucediendo dentro de la computadora. Por ejemplo, existen ubicaciones para contener la posición actual del cursor en la pantalla, qué tecla está siendo apretada y la dirección de las zonas donde están almacenadas las variables.

Páginas de memoria

Para que a la computadora le resulte más fácil localizar datos en la memoria, ésta está dividida en "páginas". En una microcomputadora, una página consta de 256 ubicaciones, por lo que cuatro páginas son un kilobyte ($4 \times 256 = 1024$)

Las ubicaciones 0-255 suelen conocerse como página cero. Las diferentes zonas de la memoria de la computadora suelen comenzar al principio de una nueva página. Por ejemplo, en el mapa de memoria de la página anterior, el RAM del usuario comienza en la pág. 45, llamando a la primera página, página cero.

Más sobre pilas



La computadora usa las pilas para almacenar datos temporales de una forma muy particular. El último dato que se introduce es siempre el primero al que se accede. Esto se denomina almacenamiento LIFO (last in, first out), último en entrar, primero en salir.

Números hexadecimales

En los programas en código de máquina, los números y las direcciones se escriben siempre en hexadecimal. A continuación, te explicamos cómo convertir números decimales a hexadecimales y viceversa.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Esta tabla muestra los dígitos hexadecimales (0-9 y A-F) y sus valores decimales. Para los números superiores a 15 (F) se utilizan dos (o más) dígitos igual que en el sistema decimal se utilizan dos dígitos (o más) para números superiores a 9. El valor de cada dígito depende de su posición en el número.

Decimal			
1000s	100s	10s	1s
1	2	2	6

Hexadecimal		
256s	16s	1s
4	C	A

En el sistema decimal, el primer dígito a la derecha del número indica cuántas unidades hay, el segundo indica el número de decenas (10s), el tercero, las centenas... (10s; 10 ^{2}s), etc.

En un número hexadecimal, el primer dígito a la derecha indica el número de unidades, pero el segundo, el número de 16s y el tercero el número de 256s (16 ^{2}).

Convertir hexadecimales a decimales

Para convertir un número hexadecimal como 4CA a decimal, mira el valor decimal que corresponde a cada uno de los dígitos del número. Multiplica cada dígito por el valor de su posición en el número y suma los resultados.

256s	16s	1s
4	C	A
4	12	10 Valor decimal
×256	×16	×1
1024 +	192 +	10 = 1226

&4CA es 1226 en decimal.

¿Sabrías convertir &A7 a decimal y el número decimal 513 a hexadecimal? (la solución pág. 44).

Decimal a hexadecimal

Para convertir un número decimal como el 1226 a hexadecimal hay que dividir en primer lugar entre 256 para averiguar cuántos 256 hay. Después, divide el resto entre 16 para averiguar cuántos 16 hay, siendo el resto de esta operación el número de unidades. Convierte todos los resultados en un dígito hexadecimal.*

1226 ÷ 256 = 4.....4 es 4 en hexadecimal
resto 202

202 ÷ 16 = 12.....12 es C en hexadecimal
resto 10.....10 es A en hexadecimal

1226 es 4CA en hexadecimal

Convertir direcciones hexadecimales

En un dirección hexadecimal como 5C64, los dos dígitos a la izquierda indican en qué página (ver pág. anterior) se halla la ubicación, mientras el segundo par indica la posición en la página.

Hexadecimal a decimal

Dirección &5C64

Número de página = &5C = 92 decimal
Posición en la página = &6C = 100 decimal

92 × 256 = 23552 + 100
= 23652

Dirección hexadecimal 5C64 es 23652 decimal.

Para convertir una dirección hexadecimal a decimal, has de convertir cada pareja de dígitos en números decimales siguiendo los pasos indicados arriba. A continuación multiplica el número de página por 256 (hay 256 ubicaciones en una página) y suma el número de posición en la página.

Decimal a hexadecimal

Dirección decimal 23652

23652 ÷ 256 = 92
(número de página de memoria)
Resto 100 (posición en la página)

92 ÷ 16 = 5 Resto 12 = &5C
100 ÷ 16 = 6 Resto 4 = &64

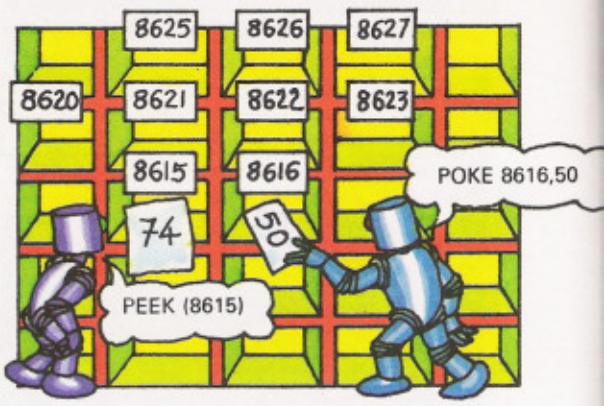
Dirección decimal 23652 es 5C64 en hexadecimal.

Para convertir una dirección decimal de hexadecimal tendrás que dividir entre 256 para encontrar el número de página de memoria. El resto da la posición en la página. A continuación convierte el resultado a dígitos hexadecimales como se indica arriba.

*Ver página 41 para saber cómo hacer esto en una computadora.

Peek y Poke

Las dos palabras BASIC, PEEK y POKE* te permiten ver los bytes almacenados en las ubicaciones de memoria y cambiarlos. PEEK y POKE se usan con direcciones de ubicaciones de memoria escritas en números decimales aunque en algunas computadoras también se usan hexadecimales. Recuerda que para introducir en la computadora números hexadecimales debes escribir un signo que puede ser &, # (llamado almohadilla) o $ delante del número. Compruébalo en tu manual, ya que varía en diferentes computadoras, incluso algunas sólo aceptan números decimales.



Puedes "ver" cualquier ubicación de la memoria de tu computadora, pero sólo puedes "introducir" (poke) nuevos bytes en las ubicaciones RAM, ya que en las ROM no se pueden cambiar los bytes.

Usar PEEK

```
PRINT PEEK (12345)
46
PRINT PEEK (720)
240
PRINT PEEK (8643)
0
LET A=PEEK (1024)
PRINT A
176
```

```
10 FOR J=700 TO 725
20 PRINT PEEK (J); ",";
30 NEXT J
```

```
RUN
36,27,234,56,21,0,0
0,0,45,32,67,121,45,
47,89,63,21,0,87,241,
202,225,63,87,16,
```

Estos son los equivalentes decimales de bytes en código de máquina.

Para decirle a la computadora que lea una ubicación de memoria se usa PEEK (puede ser otra instrucción en tu computadora) con la dirección de esa ubicación. Para ver el resultado en la pantalla usa PRINT PEEK, o almacena el resultado en una variable utilizando LET para, a continuación, escribir la variable como se indica a la izquierda.

Intenta escribir un programa corto usando un bucle FOR/NEXT como el que hay arriba en el centro para escribir los bytes de una serie de ubicaciones. Observa el mapa de memoria de tu computadora y experimenta con direcciones de diferentes partes de la memoria.

Poke

```
POKE 16763,60
PRINT PEEK (16763)
60
```

Esto le dice a la computadora que introduzca 60 en la ubicación 16763.

Usa PRINT PEEK para ver el resultado.

```
10 FOR A=16763
TO 16768
20 INPUT N
30 POKE A,N
40 NEXT A
```

Esto pone un número N en la ubicación A.

El dibujo superior te enseña cómo usar POKE. Puedes utilizarlo en cualquier parte del RAM, pero si introduces nuevos valores en una zona reservada para el sistema operativo puedes interferir con el funcionamiento de la computadora. Puedes volverla a su estado normal apagando y encendiendo de nuevo. Intenta escribir un programa corto como el superior para introducir varios números en ubicaciones del RAM del usuario.

Los números que introduzcas deben estar entre el 0 y 255, siendo éste el número máximo que puede obtenerse con ocho dígitos binarios (un byte de código de máquina).

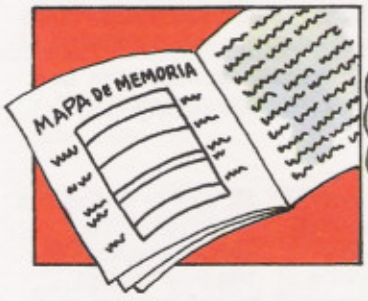
*Algunas computadoras usan instrucciones diferentes. Por ejemplo, el BBC utiliza una interrogación, ?. Compruébalo en tu manual.

Significado de los números

Cuando le dices a la computadora que escriba el contenido de una ubicación de memoria en la pantalla, el resultado es siempre un número decimal entre el 0 y el 255. Esto se debe a que cada ubicación de memoria puede contener un byte, y el valor más alto que puede obtenerse con ocho dígitos binarios es 255. Sólo existen 256 (0 a 255) bytes posibles de código de máquina, y cada byte puede tener diferentes significados para la computadora.

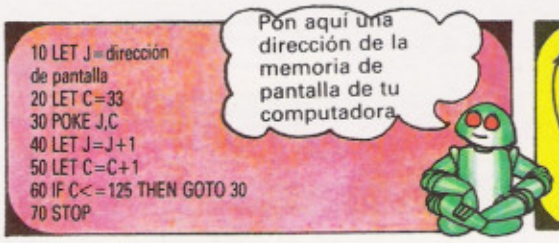
Por ejemplo, el número binario 00110000 (48 en decimal) podría ser el código de alguna instrucción, de alguna letra del teclado o parte de la dirección de otra ubicación de memoria (cada dirección se compone de dos bytes).

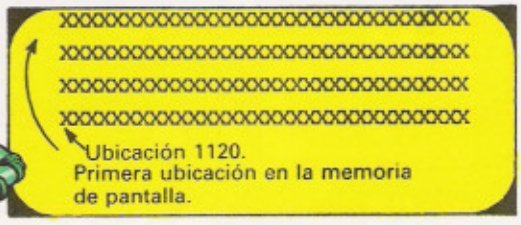
Busca en tu manual la dirección en el ROM del sistema operativo de tu computadora y prueba este programa. Los números que aparecen en la pantalla son los equivalentes decimales de bytes de código de máquina de uno de los programas del sistema operativo.



Ahora busca la memoria de pantalla de tu computadora e intenta introducir números con POKE en las ubicaciones de la memoria de pantalla. No necesitas usar PRINT PEEK, ya que los bytes almacenados en la memoria de pantalla se visualizan automáticamente en la pantalla. Esta vez, la computadora interpreta el número como el código de un carácter.

Muchas computadoras usan el código ASCII para decidir qué número representa a cada carácter. Pero algunas como el ZX81 (Timex 100) usan números diferentes. El VIC 20 posee unos números especiales llamados código de pantalla para visualizar caracteres en la pantalla. En el manual de tu computadora debe haber una lista de los códigos y los caracteres que representan.





Prueba algún programa como el que se muestra aquí para obtener la lista de caracteres de tu computadora. El programa usa códigos ASCII, comenzando con el 23, el código !, y terminando con el 90. El resto de los números entre el 0 y el 255 son para teclas especiales como la barra espaciadora y DELETE, o para escribir el alfabeto en inversa o intermitente, así como para caracteres de gráficos.

En casi todas las computadoras puedes escribir un carácter en una posición determinada de la pantalla calculando la dirección de la ubicación que corresponde a esa posición. Por ejemplo, si la memoria de pantalla comienza en la ubicación 1024 y la computadora puede visualizar 32 caracteres por línea, la dirección de la primera posición de la cuarta línea será 1024+(32×3); es decir, 1120. (La dirección 1024 se cuenta como cero.)

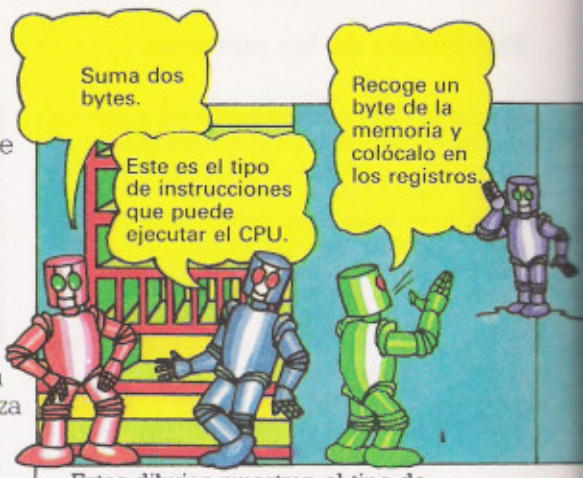
*En el Spectrum (Timex 2000), la información para cada posición de la pantalla se almacena en diferentes ubicaciones de memoria que no puedes alterar introduciendo nuevos códigos con POKE.

Dentro del CPU

Todo el trabajo de la computadora se realiza cogiendo de la memoria bytes de instrucciones y de datos y llevando a cabo las instrucciones en el CPU.

Existen tres zonas principales en el CPU: los registros en los que se contienen los bytes de datos mientras están siendo procesados; el ALU o unidad lógico/aritmética, en la que los bytes se suman, restan o comparan, y la unidad de control, que es la que organiza todas estas actividades.

La distribución de los registros en los chips Z80 y 6502 es diferente como podrás observar en los dibujos inferiores.



Estos dibujos muestran el tipo de instrucciones que puede ejecutar el CPU, muy sencillas. Puede recoger bytes de la memoria mandarlos a los registros, mover bytes de un registro a otro, procesarlos en el ALU y almacenar los resultados en la memoria. Incluso la labor más sencilla como puede ser sumar dos números y visualizar el

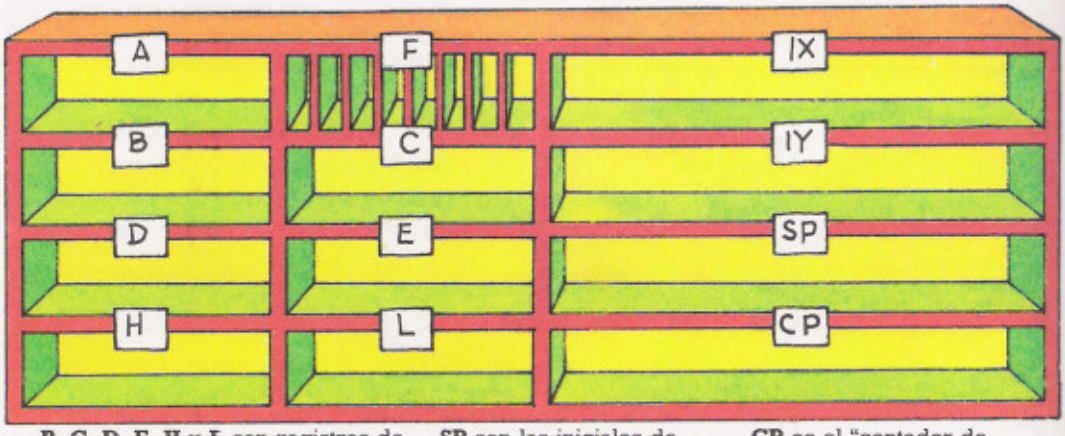
Los registros del Z80

La principal diferencia entre los chips Z80 y 6502 es que el primero posee más registros. Esto significa que se pueden almacenar bytes temporalmente en el CPU, mientras que en el 6502 tienen que ser mandados de vuelta a la memoria.

A significa "acumulador". Este es el registro más importante del CPU. Almacena bytes en su camino de ida o de vuelta a la unidad lógico-aritmética (ALU). Sólo puede contener un byte cada vez.

F es el "registro de bandera". Contiene ocho bits de los que sólo se usan seis. Cada bit actúa como una señal. Por ejemplo, la bandera de acarreo toma el valor 1 cuando la respuesta es mayor que 255; es decir, cuando no cabe en un byte y la bandera de signo muestra si un número es positivo o negativo.

IX y **IY** se denominan "registros índice". Pueden contener 16 bits y se usan en ciertas instrucciones para calcular la dirección de un byte en la memoria.



B, C, D, E, H y **L** son registros de uso general donde los bytes se almacenan en su camino de ida o de vuelta a la memoria. Cada uno puede contener un solo byte, pero pueden agruparse en parejas para contener dos bytes; ej.: BC, DE o HL.

SP son las iniciales de «stack pointer» en español indicador de pila. Es un registro de 16 bit donde se almacena la dirección del último dato de la pila —el lugar donde el CPU almacena datos temporales.

CP es el "contador de programa". Es un registro de 16 bit donde se encuentra la dirección del siguiente bit que va a ser recogido de la memoria. El número del contador de programa aumenta en uno cada vez que se ejecuta una instrucción.

resultado en la pantalla, engloba más de un centenar de pequeños pasos como éstos. El CPU puede realizar más de medio millón por segundo.

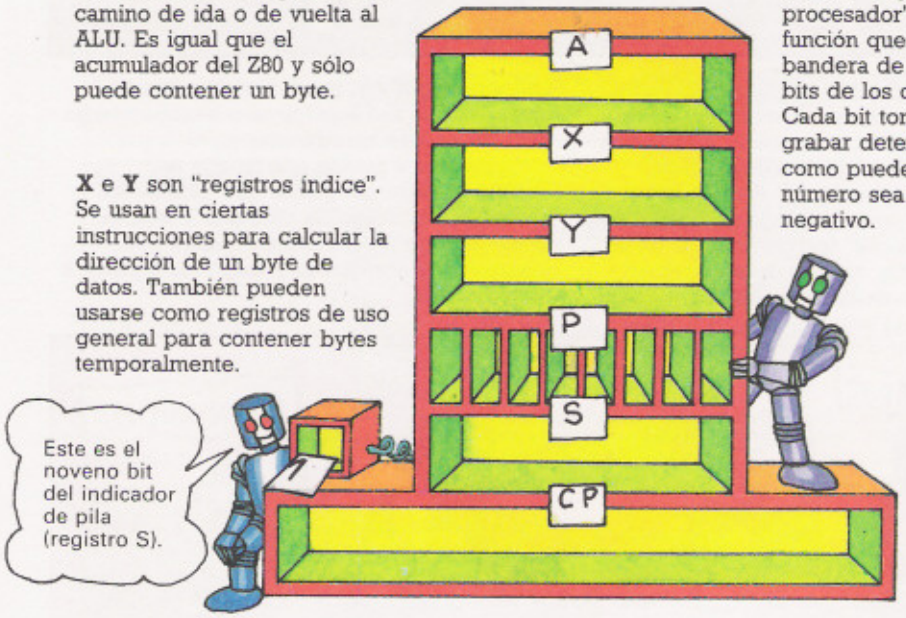
Para cada operación, la unidad de control recoge un byte de instrucción del ROM o del RAM, carga un byte de datos en los registros y a continuación realiza la operación especificada por la instrucción. En código de máquina puedes decirle al CPU lo que debe hacer con los bytes en los registros mientras que el ALU y la unidad de control llevan a cabo su labor automáticamente y no puedes decirles lo que deben hacer.

Los registros del 6502

Los principales registros de 6502 son iguales que los del Z80, aunque posean nombres diferentes.

A es el "acumulador" donde se almacenan los bytes en su camino de ida o de vuelta al ALU. Es igual que el acumulador del Z80 y sólo puede contener un byte.

X e **Y** son "registros índice". Se usan en ciertas instrucciones para calcular la dirección de un byte de datos. También pueden usarse como registros de uso general para contener bytes temporalmente.



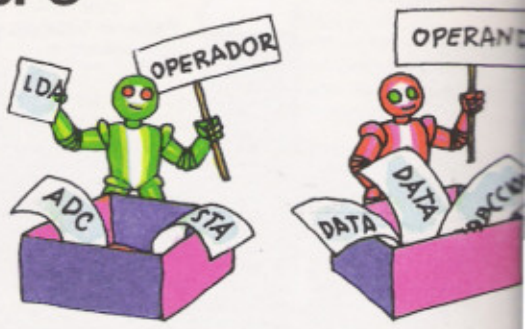
P es el "registro de status del procesador" y tiene la misma función que el registro de bandera de Z80. Contiene ocho bits de los que se usan siete. Cada bit toma el valor 1 para grabar determinada condición, como puede ser que un número sea positivo o negativo.

CP es el "contador de programa", que funciona de igual manera que el registro CP del Z80.

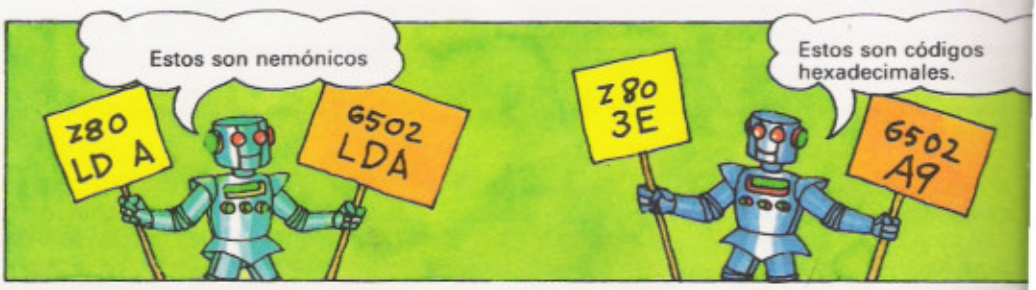
S es el "indicador de pila". Contiene la dirección del último dato de la pila —la zona especial del RAM donde el CPU almacena datos—. En el 6502, el indicador de pila es un registro de ocho bits. Para poder contener direciones existe un nuevo bit cuya posición permanente es 1, unido al registro. Este representa el número de página de la dirección, por lo que en el 6502 la pila está siempre en la página uno de la memoria. El número del indicador de pila da la posición en la página.

Dar instrucciones al CPU

Un programa en código de máquina consiste en una lista de instrucciones que le dicen al CPU lo que debe hacer con los bytes en los registros. Sólo puedes usar instrucciones que entienda el CPU, por lo que para las computadoras con microprocesador Z80 o Z80A debes usar instrucciones especiales para Z80 mientras que para computadoras con microprocesadores 6502, 6502A ó 6510 debes usar instrucciones especiales para 6502. En la parte de atrás de este libro hay una lista de instrucciones de Z80 y de 6502.

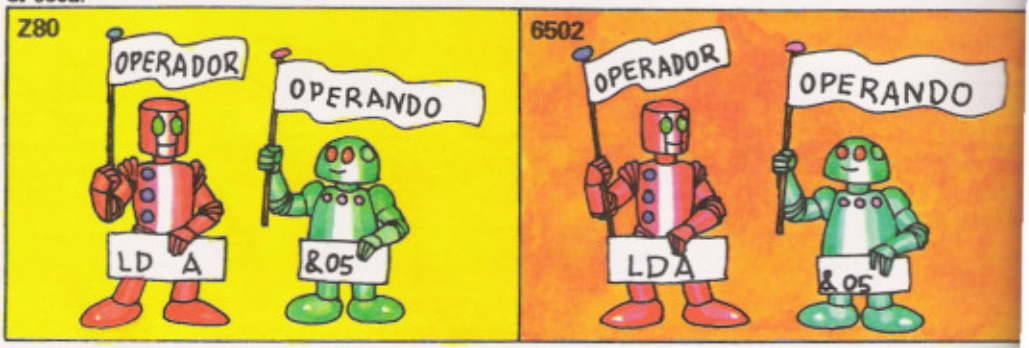


Las instrucciones en código de máquina están formadas por dos partes: un "operador" y un "operando". El operador le dice al CPU lo que debe hacer y el operando le dice dónde encontrar los datos con los que debe trabajar (la palabra operando significa "objeto sobre el que se realiza una operación"). Cada operador es una instrucción.



Los operadores pueden escribirse en forma de nemónicos, es decir, abreviaturas que simbolizan la función que realiza, o en forma de equivalentes hexadecimales de cada instrucción. Por ejemplo, LD A en el Z80 y LDA en el 6502 son nemónicos de "load a byte into the accumulador" (cargar un byte en el acumulador). Estos mismos operadores en hexadecimal serían 3E para el Z80 y A9 para el 6502.

Los nemónicos son mucho más fáciles de entender que sus equivalentes haxadecimales, pero no podrás introducirlos en tu computadora a menos que tengas un ensamblador(un programa que traduce los nemónicos a código de máquina*). Mucha gente escribe programas en código de máquina con nemónicos y luego los traduce a hexadecimal.



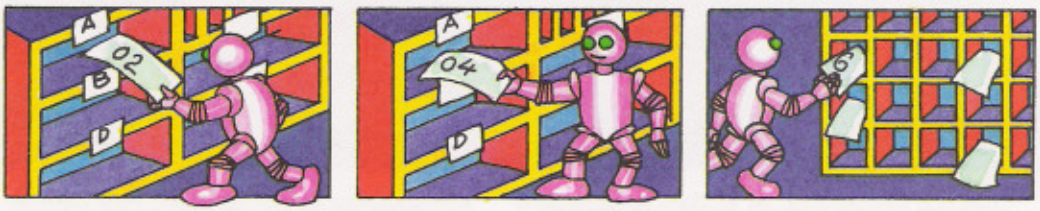
Aquí vemos dos instrucciones de código de máquina expresadas mediante nemónicos, una para el Z80 y otra para el 6502. Ambas le dicen a la computadora que cargue el número 05 (hexadecimal) en el acumulador (& es el signo que indica que es un número hexadecimal). En el código de máquina, los números siempre se escriben en forma hexadecimal. En el 6502, el número va precedido del signo # para indicar que es un dato.

*En la página 40 encontrarás más cosas sobre ensambladores.

Un programa simple

Aquí hay dos programas, uno para el Z80 y otro para el 6502, que dan instrucciones al CPU para que sume dos números. Ambos están escritos con nemónicos. Un programa con nemónicos se denomina programa en lenguaje ensamblador y uno con códigos hexadecimales se denomina programa en código de máquina. En la página siguiente te indicamos cómo traducir los programas a código de máquina y a continuación cómo cargar y ejecutar la versión para tu computadora.

Los programas para Z80 y para 6502 siguen los mismos pasos aunque las instrucciones en sí sean diferentes*. En el 6502, los datos sobre los que se van a realizar los cálculos han de estar en el acumulador. En el Z80 pueden estar en el el acumulador o si son números muy altos en la pareja de registros HL.



Para sumar dos números, cargas el primero en el acumulador. A continuación sumas el segundo número al que haya en el acumulador y almacenas el resultado en la memoria. Los nemónicos para estas instrucciones son los siguientes:

Nemónicos para Z80	Significado
LD A, número	Cargar un número en A. A significa "acumulador" y LD son las abreviaturas de "load" (cargar).
ADD A, número	Sumar un número a A (el acumulador).
LD (dirección) A	Cargar el contenido de A (el acumulador) sobre una determinada dirección. Las direcciones se ponen siempre entre paréntesis.

Nemónicos para 6502	Significado
LDA número	Cargar un número en A. A significa "acumulador" y LD es una abreviatura de "load" (cargar).
ADC número	ADC es el nemónico para la instrucción "sumar con acarreo". Le dice a la computadora que sume un número al acumulador y que ponga la bandera de acarreo si es necesaria en el registro de banderas. En la página 29 encontrarás más información sobre esto.
STA dirección	Almacenar A (es decir, el contenido del acumulador) en una determinada dirección, ST es la abreviatura de "store" (almacenar) y A la de "acumulador".

Ahora puedes completar los datos y las direcciones. En estos ejemplos, los programas suman 2 hexadecimal con 4 hexadecimal (que son lo mismo que el 2 y el 4 decimales) y almacenan el resultado en la ubicación de memoria 7F57 hexadecimal.

*De ahora en adelante, si tienes un Z80 puedes saltarte los programas para 6502 y si tu computadora usa instrucciones para 6502, entonces ignora los programas de Z80.

Traducir un programa a hexadecimal

La única manera de traducir nemónicos a sus códigos hexadecimales es buscar cada nemónico en una tabla. Al final de este libro encontrarás una tabla de nemónicos y códigos hexadecimales. No obstante debes tener cuidado ya que existen diferentes códigos hexadecimales para cada instrución según el tipo de operando, ya que éste puede ser un dato, una dirección o el nombre de un registro. Por ejemplo, aquí hay diferentes versiones de los operadores para cargar en el acumulador y sus códigos hexadecimales.

Z80		6502	
Nemónicos	**Códigos hexadecimales**	**Nemónicos**	**Códigos hexadecimales**
LD A, dato	3E, dato	LDA dato	A9 dato
LD A, dirección	3A, dirección	LDA dirección	AD dirección

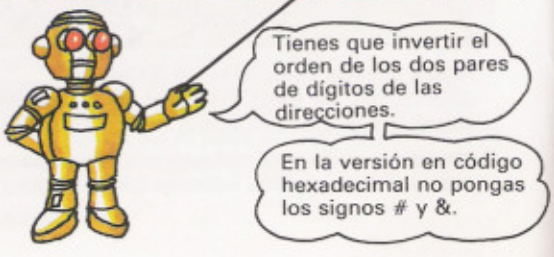
Cuando el operando es un dato se denomina "direccionamiento inmediato". Cuando es la dirección en la que se encuentra el dato, se denomina "direccionamiento absoluto". La lista de nemónicos y códigos hexadecimales de la parte final del libro incluye todas las instrucciones que se tratan en este libro. Si quieres escribir programas más complejos necesitarás una lista completa de los códigos de Z80 o de 6502 para lo cual te recomendamos algunos libros en la pág. 40.

Programa para sumar de Z80		Programa para sumar de 6502	
Nemónicos	**Códigos hexadecimales**	**Nemónicos**	**Códigos hexadecimales**
LD A, dato	3E, dato	LDA dato	A9 dato
ADD A, dato	C6, dato	ADC dato	69 dato
LD (dirección), A	32, dirección	STA dirección	8D dirección

Estos son los códigos hexadecimales para los programas para sumar de Z80 y de 6502. Las instrucciones en forma nemónica suelen llamarse códigos fuentes y aquéllos en forma hexadecimal, códigos objeto.

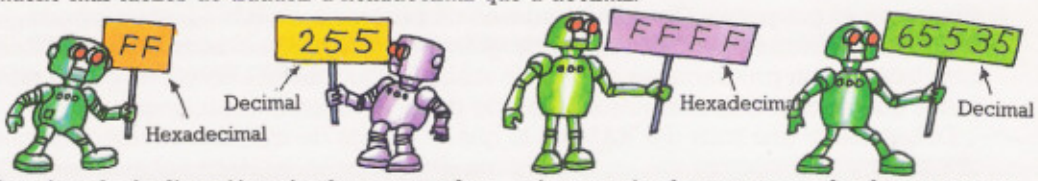
Programa para sumar con datos para Z80		Programa para sumar con datos para 6502	
Nemónicos	**Códigos hexadecimales**	**Nemónicos**	**Códigos hexadecimales**
LDA, &02	3E, 02	LDA #&02	A9 02
ADD A, &04	C6, 04	ADC #&04	69 04
LD (&7F57), A	32, 577F	STA &7F57	8D 577F

Ahora puedes completar los datos y direcciones. Esto se hace directamente —salvo para las direcciones—. En código de máquina tienes que invertir el orden de los pares de dígitos que forman una dirección. En la página opuesta encontrarás más información sobre esto.

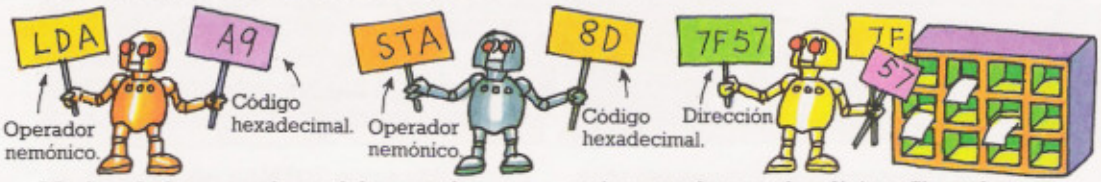


Más sobre código hexadecimales

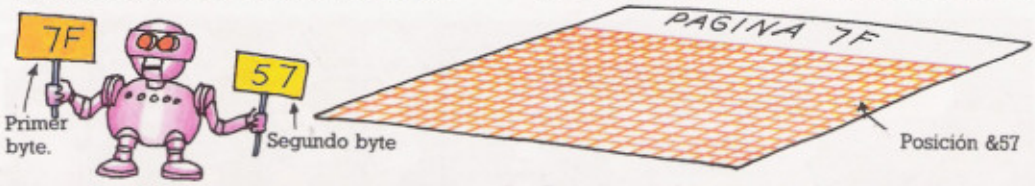
Los programas en código de máquina se escriben con números hexadecimales en lugar de números decimales debido a que los números binarios que entiende la computadora son mucho más fáciles de traducir a hexadecimal que a decimal.



Por ejemplo, la dirección más alta que puedes obtener con dieciséis dígitos binarios es 65535 en decimal y FFFF en hexadecimal y el número más alto que se puede obtener con un byte (ocho dígitos binarios) es 255 en decimal y FF en hexadecimal.*



Muchos de los operadores del grupo de instrucciones de la computadora miden un byte de largo, por lo que en hexadecimal cada operador son dos dígitos. Sin embargo, las direcciones usan dos bytes por lo que necesitan dos pares de dígitos hexadecimales.



La primera pareja de dígitos hexadecimales es el número de página de la memoria en la que se halla la dirección (ver página 10). La segunda pareja de dígitos es la posición en la página de la ubicación de memoria (una página = 256 ubicaciones de memoria).

Debido a la forma en que el CPU maneja las direcciones siempre debes darle en primer lugar la posición en la página (el segundo byte), seguido del número de página (el primer byte).

Observar un programa en código de máquina

Los programas en código de máquina que ves en las revistas suelen resultar muy complejos hasta que descubres cómo están presentadas. A continuación verás dos ejemplos de listado en código de máquina de las dos maneras en que suelen presentarse. (Ninguno de estos programas está completo, por lo que no funcionarán en una computadora.)

Forma hexadecimal

Cada par de dígitos es una instrucción, dato o parte de una dirección en hexadecimal.

Dirección hexadecimal

```
3A30  00 00 00 00 DB 01 CB 1F
3A38  CB 1F 40 07 3E 10 D3 00
3A40  CF 37 3F 21 2F 39 11 00
3A48  D0 01 C0 03 C5 0E 27 71
3A50  1A FE 58 28 28 CD C5 3A
3A56  CD A0 3A D3 01 4E 0D 79
3A60  FE 00 FA 88 3A F5 CD BD
3A68  08 FE 64 CA 06 3B F1 71
```

Este se halla en forma hexadecimal. Los primeros cuatro dígitos de cada línea son una dirección y el resto de los pares de dígitos son los códigos hexadecimales de instrucciones, datos y direcciones. El primer código de cada línea se almacena en la dirección del principio de línea. El resto de los códigos se almacena en las ubicaciones siguientes a dicha dirección.

Listado en lenguaje ensamblador

Dirección — Código hexadecimal — Nemónicos

```
0340  A2 00       LDX  #&00
0342  BD 4E 03    LDA  &034E,X
0345  9D C0 83    STA  &83C0,X
0348  E8          INX
0349  E0 0B       CPX  #&0B
034B  D0 F5       BNE  &F5
034D  00          BRK
```

Este listado incluye códigos hexadecimales y nemónicos. El primer número de cada línea es la dirección de memoria en la que se almacena el primer byte de cada línea. La siguiente columna contiene códigos hexadecimales para el programa seguidos de nemónicos.

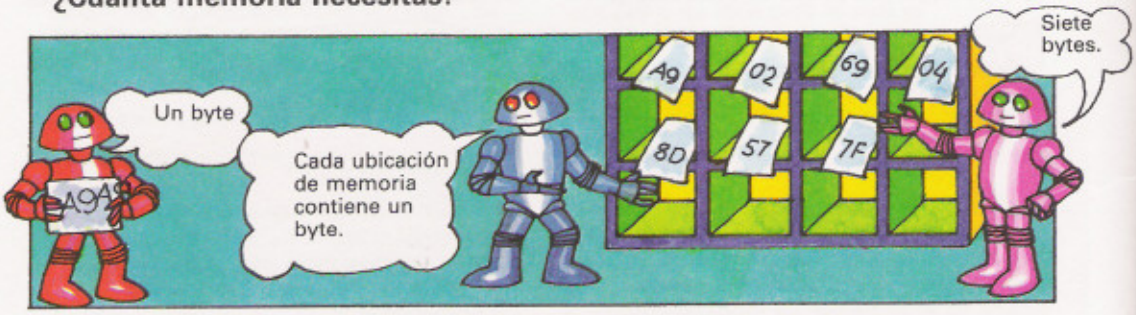
*En la página 28 encontrarás cómo convertir números binarios en decimales.

Encontrar RAM vacío

Antes de poder usar el programa para sumar de la página 18 hay varias cosas que debes hacer. Primero tienes que escoger una zona de la memoria en la que puedas almacenar el programa. Cuando introduces un programa en BASIC, el intérprete BASIC automáticamente almacena tu programa en el RAM del usuario. Cuando lo que introduces es un programa en código de máquina, no usas el intérprete, por lo que tienes que decirle a la computadora dónde debe almacenar el programa.

Debes elegir una zona del RAM en lo que tu código de máquina se interfiera con otra información almacenada en la memoria. Por ejemplo, no debes almacenar código de máquina en zonas reservadas para uso del sistema operativo, como pueden ser los sistemas de variables o las pilas. Si lo haces, el sistema dejará de funcionar, ya que tu código de máquina habrá sustituido información vital para que la computadora organice su trabajo. También debes tener cuidado de mantener tu código de máquina separado de cualquier programa en BASIC que hayas introducido. Si el sistema deja de funcionar, la única manera de poner la computadora bien es apagándola y volviéndola a encender, aunque en ese caso habrás perdido tu programa.

¿Cuánta memoria necesitas?



Es muy fácil calcular la longitud de un programa en código de máquina —debes contar el número de pares de dígitos hexadecimales (cada par ocupa un byte)—. Por ejemplo, el programa para sumar tiene siete bytes.

Casi todos los programas en código de máquina son breves, por lo que en principio con cien bytes de espacio de memoria tendrás suficiente para tus programas en código de máquina.

Encontrar RAM desocupado

El sitio más habitual para almacenar en código de máquina es en la parte superior del RAM del usuario el lugar donde se almacenan los programas en BASIC. Sin embargo, esto implica que debes tener cuidado para no mezclarlos con programas en BASIC. Para evitar esto puedes bajar el límite de la zona RAM del usuario. Esto crea una «tierra de nadie» sobre el RAM del usuario que la computadora no puede usar hasta que tú se lo ordenes al cargar tu programa en códhgo de máquina.

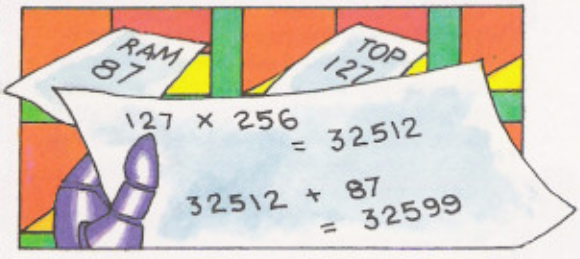
La parte superior del RAM del usuario se denomina RAMTOP o HIMEM, o simplemente límite de la memoria. En la página siguiente aprenderás cómo bajar el RAMTOP.

Bajar el tope del RAM del usuario

La computadora graba la dirección del RAMTOP en el sistema de variables, que tú puedes cambiar variando la dirección almacenada en el sistema de variables. La manera de hacer esto varía en las diferentes computadoras, pero a continuación te damos los principios que casi todas siguen. Sin embargo, debes comprobar en tu manual cómo hace tu computadora para variar el RAMTOP, ya que puede usar instrucciones diferentes o incluso tener un sistema más sencillo para guardar sitio para código de máquina.

La dirección del RAMTOP ocupa ubicaciones consecutivas en el sistema de variables, una para el número de página de la ubicación y la otra para la posición en la página. Busca en tu manual las direcciones de estas ubicaciones del sistema de variables (pueden estar como RAMTOP, HIMEM o simplemente tope del RAM del usuario). La computadora almacena los dos bytes de la dirección en orden inverso, en primer lugar la posición en la página y a continuación el número de página, por lo que la primera ubicación del sistema de variables contendrá el número de posición y el segundo la página.

```
PRINT PEEK (dirección)+PEEK
(dirección 2)*256
```



Puedes usar PRINT PEEK (o la instrucción que requiera tu computadora) de esta manera para leer en el sistema de variables y visualizar la dirección del RAMTOP. Complétalo con las direcciones de tu sistema de variables.

Esta instrucción convierte automáticamente los dos bytes de la dirección del RAMTOP en una dirección decimal multiplicando el número de página por 256 y sumando la posición en la página.

```
CLEAR ramtop dirección - 100      Spectrum
HIMEN ramtop dirección - 100      Oric
```

Muchas computadoras tienen su propia instrucción especial para cambiar la dirección del tope del RAM del usuario. Por ejemplo, la instrucción de Spectrum (Timex 2000) es CLEAR mientras que la del Oric es HIMEM. Estas instrucciones van seguidas de la dirección del tope del RAM del usuario menos el número de bytes de memoria que quieres reservar para el código de máquina tal y como se muestra arriba a la izquierda. Comprueba en tu manual cuál es la instrucción de tu computadora.

Estas instrucciones bajan el tope del RAM del usuario 100 ubicaciones, por lo que reservan una zona de 99 bytes para código de máquina comenzando en la dirección siguiente al RAMTOP. Puedes cambiar el número 100 si quieres reservar más espacio.

*Pasa la página para ver cómo bajar el tope del RAM en el VIC 20, y dónde almacenar código de máquina en el ZX81 (Timex 1000).

Truco para VIC 20

El VIC 20 no tiene ninguna instrucción especial para cambiar la dirección almacenada en el sistema de variables. Aquí tienes lo que debes hacer para bajar el tope del RAM del usuario en el VIC.

La dirección está contenida en los sistemas de variables 55 y 56. Recuerda la segunda localización de mantener el número de página.

```
POKE 56, PEEK (56)-1
```

Para bajar el tope del RAM del usuario 256 ubicaciones, es decir una página, utiliza las instrucciones directas que se muestran arriba. Estas hacen que la computadora lea la ubicación 56 (la que contiene el número de página). Resta 1 del valor contenido en dicha ubicación e introduce el nuevo valor en la ubicación 56. En otras palabras, reduce en 1 la parte de la dirección que corresponde al número de página. Para ver la nueva dirección del tope del RAM del usuario usa estas instrucciones: PRINT PEEK (55) + (56)*256.

Truco para ZX81

En el ZX81, el mejor lugar para almacenar programas en código de máquina es al principio del RAM del usuario. Para hacer esto, escribe un paso REM como primer paso del programa cargador hexadecimal que encontrarás en la página 24 y complétalo con tantos dígitos como bytes haya en tu programa en código de máquina.

```
5 REM 1234567
```

Siete bytes

Cada uno de los dígitos del paso REM ocupa una ubicación en la memoria. Ahora ya puedes introducir tus bytes de código de máquina en las ubicaciones reservadas por los dígitos del paso REM.

Para hacer esto necesitas saber la dirección en la que está almacenado el primer dígito. El RAM del usuario comienza en la ubicación 16509 y la computadora necesita dos bytes para almacenar el número del paso REM; una para NEWLINE y otro para grabar la longitud del paso, por lo que el primer dígito está en la ubicación 16514.

Otros lugares para almacenar código de máquina

Existen otros lugares de la memoria en los que puedes almacenar código de máquina, siempre que no los estés usando. Por ejemplo, si no tienes intención de grabar tu programa puedes almacenarlo en el buffer de cassette, o si no estás creando gráficos definidos por el usuario puedes almacenarlo en la zona reservada para este fin. Busca en tu manual las direcciones de esta zona del RAM.

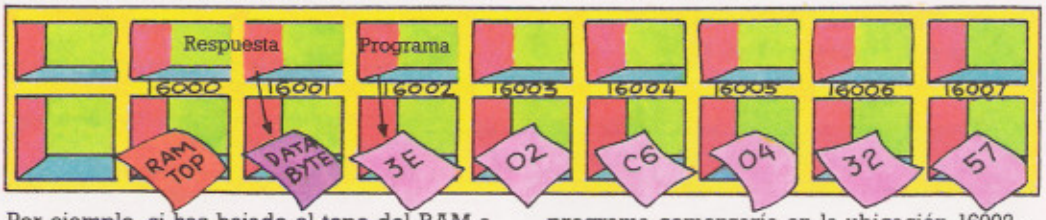
Quizá tu manual sugiera lugares adecuados de la memoria de tu computadora en los que puedas almacenar códigos de máquina. Busca también en revistas y libros, ya que existen muchos trucos.

Cargar y ejecutar un programa

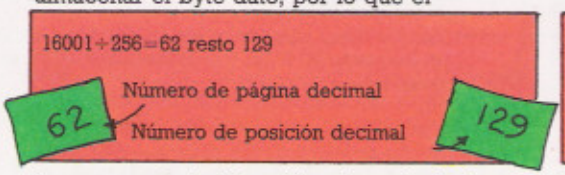
Las próximas páginas te muestran cómo cargar y ejecutar los programas para sumar de la página 18. Para introducir en la computadora un programa en código de máquina debes introducir cada byte en la zona de memoria que hayas escogido para almacenar código de máquina (ej.: por encima del RAMTOP). En muchas computadoras sólo puedes usar POKE con números decimales, por lo que necesitas un programa en BASIC denominado «cargador hexadecimal». El cargador hexadecimal convierte cada byte de código de máquina en un número decimal para a continuación introducirlo (poke) en la memoria. En la página siguiente encontrarás un cargador hexadecimal. Sin embargo, en primer lugar debes cambiar la dirección de la respuesta del programa para sumar, escogiendo una adecuada para tu computadora. Existe además una instrucción más (ver debajo) que debes añadir al programa.

Escoger una dirección para la respuesta

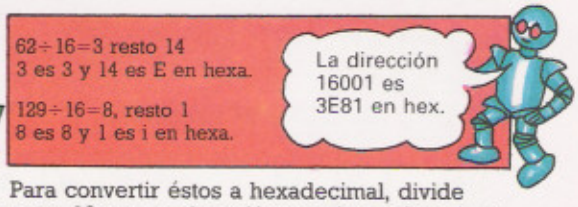
Los datos producidos por un programa en código de máquina, como puede ser el resultado de la suma en el programa para sumar, se denominan «bytes dato». Es importante almacenar estos bytes en un lugar donde no haya peligro de que se mezclen con el propio programa. El lugar más adecuado es justo al principio de la zona que has reservado para el código de máquina, es decir, justo antes del programa.



Por ejemplo, si has bajado el tope del RAM a la ubicación 16000, la primera dirección de la zona para código de máquina será la ubicación 16001. Aquí es donde debes almacenar el byte dato, por lo que el programa comenzaría en la ubicación 16002. Debes convertir la dirección del byte dato en un número hexadecimal para poder introducirlo en el programa.

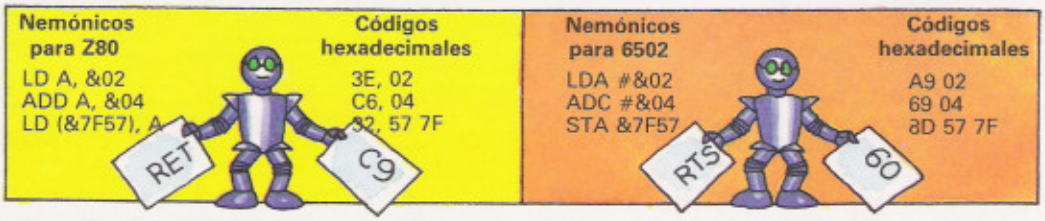


Para convertir la dirección a hexadecimal, divide entre 256. El cociente será el número de página decimal, y el resto será la posición en la página (ver página 11).



Para convertir éstos a hexadecimal, divide entre 16 y a continuación convierte el cociente y el resto en dígitos hexadecimales tal y como se indica arriba.

La instrucción retornar



Al final de los programas en código de máquina debes poner la instrucción RET (para el Z80) o RTS (para en 6502). Esto hará que la computadora deje de ejecutar el programa en código de máquina y vuelva donde comenzó el programa. Sin esta instrucción, la computadora seguiría intentando ejecutar una instrucción para cada byte que encontrase en la memoria y el sistema no tardaría mucho en dejar de funcionar.*

*En la página 35 hallarás más sobre la instrucción retorno.

Cargador hexadecimal

Este es el programa para cargar código de máquina en la memoria de la computadora. Para usarlo pon los códigos hexadecimales de tu programa en código de máquina en el paso 160, seguidos de la palabra END, como signo para indicar a la computadora que no hay más datos. En el paso 40, la computadora lee un par de dígitos hexadecimales, los convierte en números decimales entre los pasos 70 y 100 y a continuación introduce los números en la memoria en el paso 130.*

```
10 PRINT «ESCOGER UNA DIRECCION
   PARA LA RESPUESTA»
20 INPUT A
30 LET C=0
40 READ H$
50 IF H$=«END» THEN GOTO 180
60 IF LEN(H$)<>2 THEN GOTO 170
70 LET X=(ASC(H$)-48)*16
80 IF ASC(H$)>57 THEN LET X=
   (ASC(H$)-55)*16
90 LET Y=ASC(RIGHT$(H$,1))
100 IF Y<58 THEN LET X=X+Y-48
110 IF Y>57 THEN LET X=X+Y-55
120 IF X<0 OR X>255 THEN GOTO 170
130 POKE A+C,X
140 LET C=C+1
150 GOTO 40
155 REM DATOS DE EJEMPLO
160 DATA EF,F6,E2,A9,END
170 PRINT «DATO ERRONEO»
180 STOP
```

A es la dirección de la primera ubicación en la que quieres almacenar tu programa.

C es un contador.

Pone el primer par de dígitos hexadecimales del paso 160 en H$.

Comprueba si en H$ está la palabra END, la señal que indica el fin de los datos.

Comprueba si H$ contiene dos dígitos; si no los tiene pasa al paso 170.

Convierte el primer dígito hexadecimal en un número decimal y lo almacena en X.

Convierte el segundo dígito hexadecimal en un número decimal Y, y lo suma a X.

Busca datos erróneos comprobando que el número X está entre 0 y 255.

La primera vez, C=0, por lo que introduce X en la ubicación de memoria A.

Suma uno a C, lo que hará que el valor decimal del siguiente código hexadecimal entre en la ubicación de memoria A+1.

Vuelve a leer el siguiente código hexadecimal.

Pon tus códigos hexadecimales aquí, seguidos de la palabra END.

Escribe estas palabras si encuentra datos erróneos en los pasos 60 ó 120 y a continuación corta el programa.

Cómo funciona el cargador

Hex.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ASCII	48	49	50	51	52	53	54	55	56	57	65	66	67	68	69	70
	Menos 48										Menos 55					
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Hexa-decimal ASCII
3 = 51 -48 = 3
E = 69 -55 = 14
Decimal
x 16=48
48
14 +
62

En el paso 70, la computadora convierte el primer dígito de H$ a su código ASCII utilizando la palabra BASIC ASC. A continuación convierte el código ASCII a un valor decimal restándole 48; para los códigos superiores a 57, restando 55, tal y como se muestra en la tabla superior. A continuación multiplica por 16, ya que el primer dígito hexadecimal representa el número de 16s y pone la solución en X.

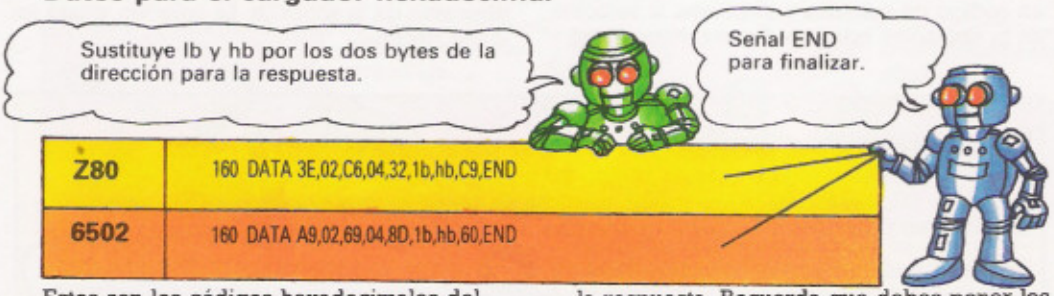
En el paso 90 usa el mismo método para convertir el dígito de la derecha en su código ASCII y almacenarlo en Y. En los pasos 100 y 110 convierte Y en un número decimal restando 48 y 55 como ya explicamos anteriormente y sumando a X. (Esta vez no multiplica por 16, ya que es el dígito que representa los 1s en el número hexadecimal.) El valor almacenado en X es el equivalente decimal del par de dígitos hexadecimales.

*Para el Spectrum (Timex 2000) cambia la instrucción ASC por CODE y pon cada par de códigos hexadecimales entre comillas. Ver pág. 48 para los cambios necesarios para el ZX81 (Timex 1000) y las computadoras Atari.

Uso del cargador

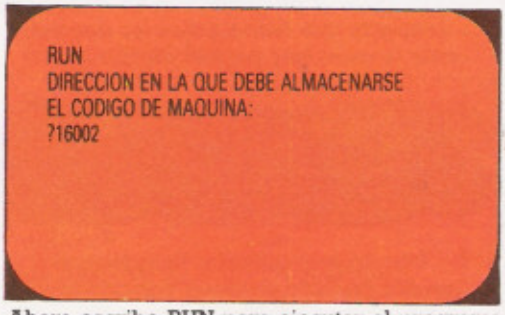
Ya puedes usar el cargador para probar el programa de sumar. Este programa no es muy interesante, pero es sencillo y sirve para comprender cómo funciona el código de máquina. Copia el listado del cargador hexadecimal. En el paso 160 sustituye los datos del ejemplo por códigos hexadecimales del programa para sumar tal y como se indica a continuación.

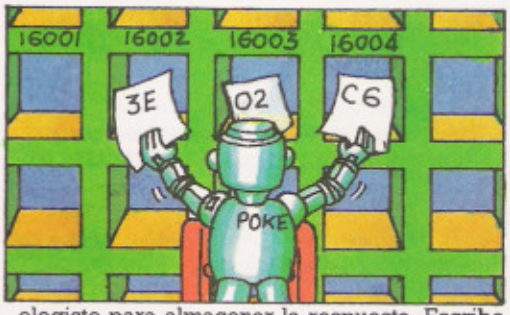
Datos para el cargador hexadecimal



Estos son los códigos hexadecimales del programa para sumar. Tienes que sustituir las letras lb y hb por los dos bytes de la dirección en la que quieras que se almacene la respuesta. Recuerda que debes poner los dos bytes en orden inverso, es decir, primero el byte de la posición en la página y a continuación el byte del número de página.

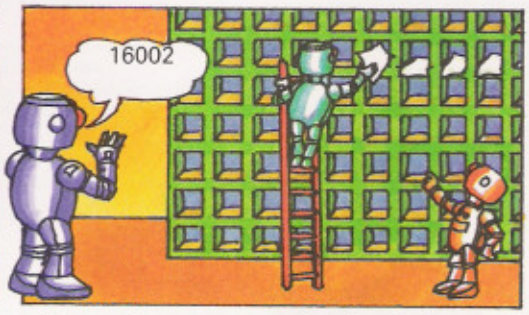
Ejecutar el cargador hexadecimal





Ahora escribe RUN para ejecutar el programa de cargador hexadecimal. Cuando te pida la dirección escribe como primera ubicación la que va exactamente después de la que elegiste para almacenar la respuesta. Escribe esta dirección en forma de número decimal, ya que será usada con la instrucción POKE.

Ejecutar el programa en código de máquina



La instrucción para decirle a la computadora que ejecute un programa en código de máquina varía según la computadora. Algunos utilizan CALL, otros PRINT USR o SYS con la dirección decimal de la ubicación donde se almacena el primer byte del programa.

Comprueba en tu manual cuál es la instrucción que necesitas. Cuando la computadora recibe esta instrucción va hasta la dirección y comienza a ejecutar las instrucciones en código de máquina.

Ver el resultado

```
PRINT PEEK(16001)
```

La computadora lleva a cabo las instrucciones en código de máquina y almacena la solución en la ubicación que le indicaste. Para ver el resultado tendrás que usar PRINT PEEK con la dirección de la solución. La solución estará en forma decimal.

```
PRINT PEEK(16001)
6
```

Programas para escribir

En este momento sabes suficiente código de máquina para escribir algunos programas sencillos. Encontrarás, a continuación, una lista de las cosas que debes recordar cuando escribes un programa en código de máquina. Las respuestas están en la página 44.

1. Intenta escribir un programa que sume 25 y 73 (decimales) y almacene la solución en la memoria.

2. Prueba a escribir un programa que sume 64, 12 y 14 (decimal) y almacene la solución en la memoria.

El programa para sumar sólo sumará números cuyo total sea inferior a 255. En la página 28 podrás ver cómo sumar números mayores.

Lista para el código de máquina

1. Escribe tu programa en lenguaje ensamblador y convierte cualquier dato a hexadecimal.

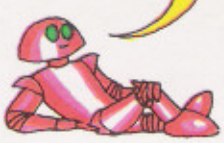
2. Busca el código hexadecimal para cada uno de los nemónicos (encontrarás una lista de nemónicos y códigos hexadecimales en la parte final del libro).

No olvides poner END detrás de la lista de códigos hexadecimales en el cargador hexadecimal.

3. Añade la instrucción retorno al final del programa (ver pág. 23).

4. Cuenta el número de bytes y reserva la correspondiente zona de RAM (ver páginas 20-22).

Haz una anotación de las direcciones de los bytes dato y de la dirección en la que has almacenado el programa.



5. Calcula qué ubicaciones de memoria necesitas para los bytes dato y convierte las direcciones a hexadecimal (ver página 23).

6. Completa el programa con las direcciones —recuerda que debes poner los dos bytes en orden inverso (ver páginas 18-19).

Antes de ejecutar el cargador hexadecimal, comprueba los códigos hexadecimales en el paso de datos.

7. Introduce el cargador hexadecimal (puedes grabarlo en cinta) y completa los códigos hexadecimales del paso 160 poniendo al final la señal END (ver pág. 24).

Si no funciona el programa, comprueba que has introducido los códigos hexadecimales correctos.

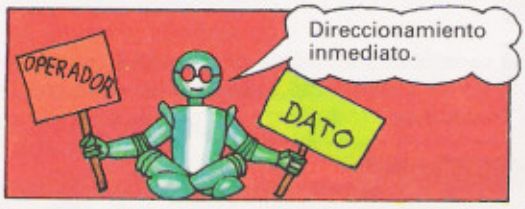
8. Ejecuta el cargador hexadecimal e introduce la dirección decimal de la primera ubicación en la que desees almacenar el código de máquina (ver página 25).

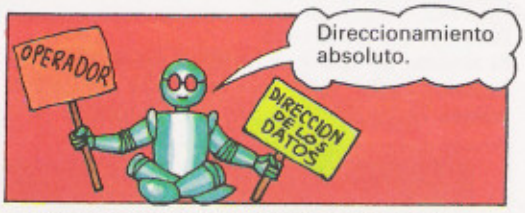
9. Ejecuta el programa en código de máquina utilizando la instrucción que requiera tu computadora con la dirección (en decimal) de la primera ubicación en la que esté almacenado el código de máquina (ver página 25).

Si cambias los datos del cargador hexadecimal tendrás que ejecutar otra vez el programa para introducir los nuevos bytes en la memoria.

Añadir bytes de la memoria

En el programa, los datos se incluían en el propio programa. Esto se denomina direccionamiento inmediato. En ocasiones querrás indicar a la computadora que haga algo con datos almacenados en su memoria. En este caso, el operando de la instrucción será una dirección que le diga a la computadora dónde encontrar los datos. Esto se denomina direccionamiento absoluto (o directo).





Estas son dos de las muchas formas en que puedes decirle a la computadora dónde encontrar los datos con los que trabajas. Las diferentes formas se denominan «modalidades de direccionamiento». Existe un diferente código hexadecimal para cada instrucción dependiendo de la modalidad de direccionamiento que uses.

Programa para sumar números de la memoria

Este es un programa para sumar dos números almacenados en la memoria. Compara los códigos hexadecimales de las instrucciones de este programa que utiliza direccionamiento absoluto, con los usados en el programa de sumar anterior que usaba direccionamiento inmediato.

Programa para Z80		
Nemónicos	**Códigos hexadecimales**	**Significado**
LD A, (dirección 1)	3A, dirección 1	Pon el número de la dirección 1 en el acumulador.
LD B, A	47	Pon el número del acumulador en el registro B.
LD A, (dirección 2)	3A, dirección 2	Pon el número de la dirección 2 en el acumulador.
ADD A, B	80	Suma el número del registro B al del acumulador.
LD (dirección 3), A	32, dirección 3	Almacena el contenido del acumulador en dirección 3.
RET	C9	Retorno

Para sumar dos números de la memoria, en primer lugar debes cargarlos en los registros. Para esto puedes usar el acumulador (A) y el registro B. No puedes cargar el registro B directamente desde la memoria, por lo que, en primer lugar, debes introducir el número en A y a continuación transferirlo a B.

Programa para 6502		
Nemónicos	**Códigos hexadecimales**	**Significado**
LDA dirección 1	AD dirección 1	Pon el número de la dirección 1 en el acumulador.
ADC dirección 2	6D dirección 2	Suma el número de la dirección 2 al del acumulador.
STA dirección 3	8D dirección 3	Almacena el contenido del acumulador en dirección 3.
RTS	60	Retorno.

Ejecutar el programa

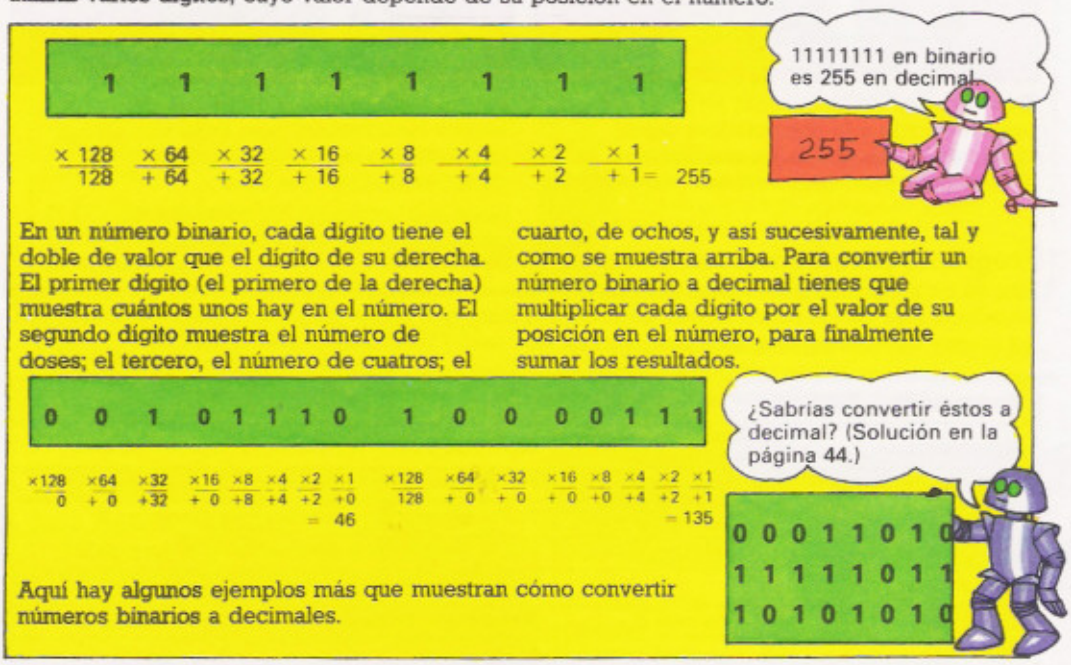
Para ejecutar este programa, sigue los pasos dados en la lista de la página anterior. Sin embargo, en primer lugar, debes introducir en la memoria los dos números que quieres sumar. Es conveniente elegir ubicaciones de memoria al principio de la zona que has reservado para el código de máquina, para así mantener estos bytes dato separados de las instrucciones. A continuación convierte las direcciones a hexadecimal e introdúcelas en el programa. Necesitarás una tercera dirección para la solución. Para ver la solución escribe PRINT PEEK (dirección 3).

Trabajar con números altos

Los programas de las páginas anteriores funcionan sólo con números cuya suma no superaba 255. Este es el número más grande que puedes representar con ocho bits en un registro o ubicación de memoria. Para trabajar con números más altos tendrás que conocer algo más sobre el sistema de números binarios y cómo usar la bandera de acarreo. En la página siguiente hay un programa en código de máquina para sumar números mayores.

Números binarios

El sistema de números binarios funciona de forma semejante a los números decimales y hexadecimales, salvo que sólo usan dos dígitos, el 0 y el 1. Para formar números mayores que 1 utilizas varios dígitos, cuyo valor depende de su posición en el número.

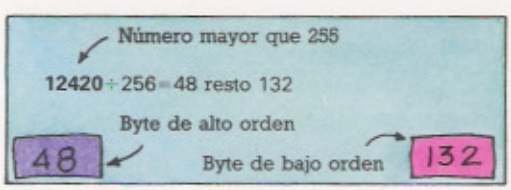


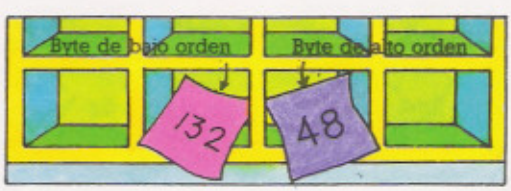
En un número binario, cada dígito tiene el doble de valor que el dígito de su derecha. El primer dígito (el primero de la derecha) muestra cuántos unos hay en el número. El segundo dígito muestra el número de doses; el tercero, el número de cuatros; el cuarto, de ochos, y así sucesivamente, tal y como se muestra arriba. Para convertir un número binario a decimal tienes que multiplicar cada dígito por el valor de su posición en el número, para finalmente sumar los resultados.

Aquí hay algunos ejemplos más que muestran cómo convertir números binarios a decimales.

Dar números grandes a la computadora

Dentro de la computadora, los números mayores que 255 se almacenan en dos bytes conocidos como «byte de orden alto» y «byte de orden bajo». El byte de orden alto indica cuántos 256s hay en el número y el byte de orden bajo es el resto. De igual forma que con las direcciones, la computadora siempre trata con el byte de bajo orden, por lo que tienes que almacenarlos en ese orden en la memoria.





Para dar a la computadora un número mayor que 255 tienes que calcular el valor de cada byte. Para ello, divide el número entre 256. El cociente será el valor decimal del byte de alto orden. El resto será el byte de bajo orden.

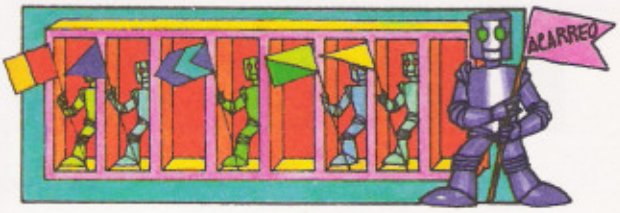
Si quieres usar el número como dato en un programa en código de máquina tendrás que convertir cada byte a hexadecimal. Para ello divide cada byte entre 16 y convierte el cociente y los restos en dígitos hexadecimales tal y como se indica en la página 11.

307; 21214; 759; 1023.

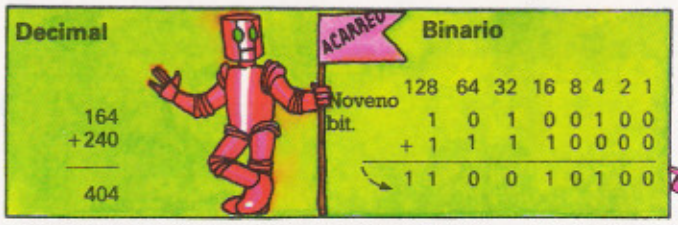
¿Cuáles son los bytes decimales de alto orden y de bajo orden de estos números? ¿Cuál es su valor hexadecimal? (La respuesta en la página 44.)

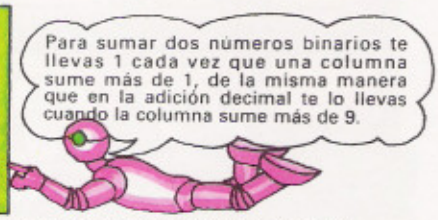
La bandera de acarreo

La bandera de acarreo es un simple bit del registro de banderas (también llamado registro de status del procesador) se utiliza cuando la solución de una operación es mayor que 255, por lo que no calzaría en un byte (ocho bits). Siempre que sucede esto, la computadora pone automáticamente un 1 en la bandera de acarreo. Esto se denomina «dar valor a la bandera de acarreo», mientras que hacerla 0 se denomina «borrar la bandera de acarreo».



Puedes pensar en la bandera de acarreo como el noveno bit que indica que un 1 binario ha sido llevado de la octava columna de un número. Por ejemplo, en la suma 164+240 (10100100+11110000 en binario) obtendríamos:

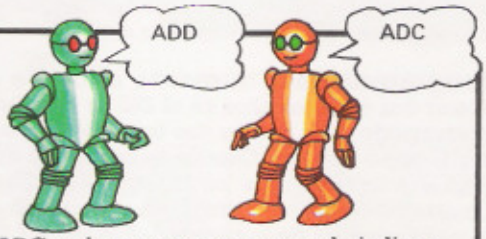




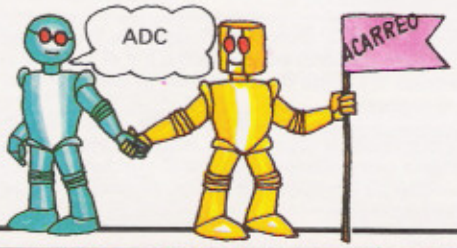
La solución a esta suma es 404, que requiere nueve bits en binario. El noveno bit indica cuántos 256s hay en el número. En la computadora estaría representado por el bit de la bandera de acarreo.

Acarreo en el Z80

El Z80 tiene dos diferentes funciones para sumas: ADD y ADC. ADD le dice a la computadora que sume dos números pero que ignore cualquier posible acarreo de cálculos anteriores. Si en la suma se produce un acarreo, la computadora dará valor a la bandera de acarreo, mientras que si no hay acarreo hará que la bandera valga 0.



ADC es la suma con acarreo y le indica a la computadora que sume dos números más la bandera de acarreo, para finalmente dar a ésta valor 1 ó 0, según el resultado. Si estás realizando una serie de cálculos, es mejor usar la instrucción ADD para la primera suma para estar seguro de que no incluye un acarreo de anteriores operaciones, y a continuación usar ADC por si hubo acarreo en la primera operación.



Acarreo en el 6502

El 6502 sólo tiene una instrucción para sumar: ADC; por lo que siempre incluye en los cálculos el contenido de la bandera de acarreo. Debido a esto es importante borrar la bandera de acarreo usando la instrucción CLC (clear carry flag: borrar bandera de acarreo) antes de hacer cualquier suma.

Programa para números altos

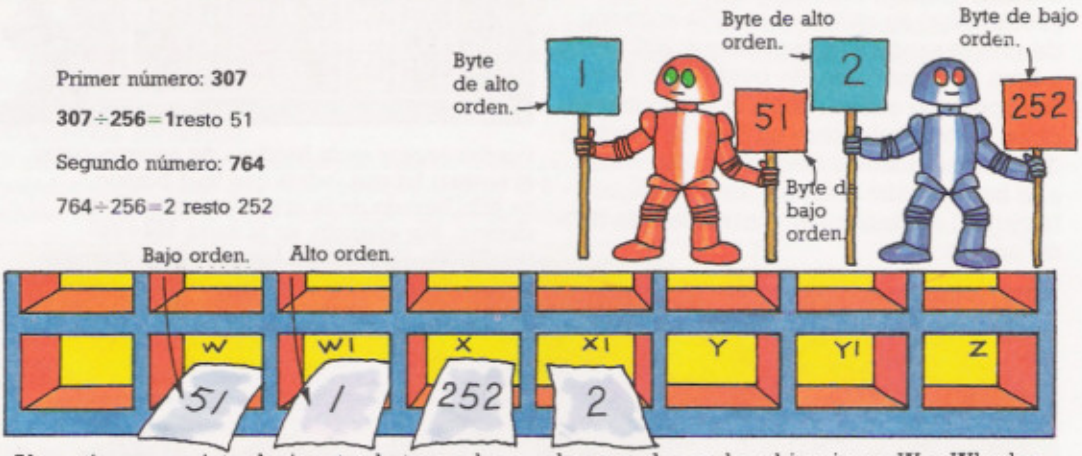
Antes de poder ejecutar los programas de estas dos páginas tendrás que calcular los bytes de alto y bajo orden para cada uno de los números que quieres sumar y, a continuación, introducirlos en la memoria. Por ejemplo, digamos que quieres sumar 307 y 764.

Primer número: **307**

307÷256=1resto 51

Segundo número: **764**

764÷256=2 resto 252



Ahora tienes que introducir estos bytes en las ubicaciones de memoria del principio de la zona que reservaste para código de máquina. Para cada número, el byte de bajo orden debe estar en la primera ubicación, seguida del byte de alto orden. En el dibujo superior, los dos bytes del primer número están almacenados en las ubicaciones W y W1 y los bytes del segundo número en las ubicaciones X y X1. Necesitarás tres ubicaciones Y, Y1 y Z para la solución (una para el byte de bajo orden, otra para el byte de alto orden y otra para el posible acarreo).

Programas para números altos de Z80

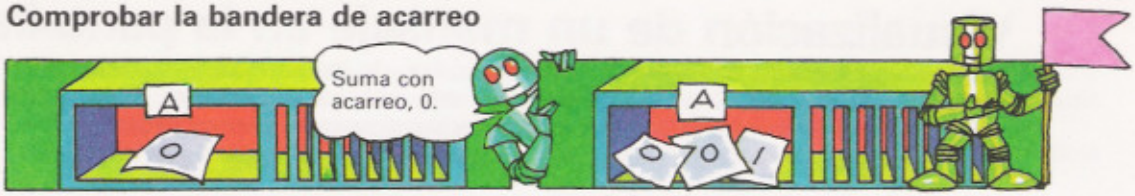
Sumar dos números altos en el Z80 es muy fácil, ya que puedes usar los registros por pares, conteniendo cada par los dos bytes del número. Puedes usar los registros H y L como un par y B y C como otro par. Cuando se utilizan de esta forma se les denomina HL y BC. Cuando no uses el acumulador usa los registros HL para sumar. Estos son los nemónicos y códigos hexadecimales para el programa. Al estudiar el programa te puede ayudar el ver los dibujos de la parte superior de la página.

Nemónicos	Códigos hexadec.	Significado
LD, HL, (dirección W)	2A, dirección W	Pon el byte de la dirección W (byte de bajo orden del primer número) en el registro L y el byte de la dirección W1 (byte de alto orden, primer número) en el registro H.
LD, BC, (dirección X)	ED 4B, dirección X Este operador usa dos bytes.	Pon el byte de la dirección X (byte de bajo orden, segundo número) en el registro C y el byte de la dirección X1 (byte de alto orden, segundo número) en el registro B.
ADD HL, BC	09	Suma los contenidos de HL y de BC y deja el resultado en HL. No suma la bandera de acarreo pero si es necesario la da valor.
LD (dirección Y), HL	22, dirección Y	Almacena el byte de bajo orden de la solución en la dirección Y y el byte de alto orden en la direc. Y1.
LD A, &0 ADC A, &0 LD(dirección Z), A	3E, 0 CE, 0 32 dirección Z	Ver página opuesta para saber cómo hace la computadora para comprobar la bandera de acarreo.
RET	C9	Retorno.

Ver enfrente cómo visualizar la solución de este programa.

Para ejecutar el programa tendrás que completar las direcciones hexadecimales W, X, Y y Z (no te olvides de invertir el orden de los pares de dígitos). Cuando uses los registros por pares sólo tendrás que especificar una dirección para cada par. La computadora automáticamente pone el byte de la siguiente dirección consecutiva de memoria en el otro registro del par.

Comprobar la bandera de acarreo

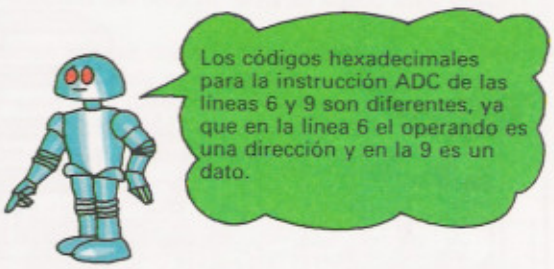


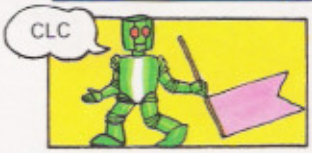
Las líneas 5-7 del programa para Z80 son para comprobar la bandera de acarreo. No puedes cargar el contenido de la bandera de acarreo directamente en un registro o en la memoria. La única manera de ver su valor es realizar otra suma. Para hacer esto pones 0 en el acumulador (línea 5) y le sumas 0 usando la instrucción sumar con acarreo. Si la bandera de acarreo tenía el valor 1 debido a cálculos anteriores, el acumulador contendrá ahora un 1 (el de la bandera de acarreo) que será almacenado en la dirección Z (línea 7).

Programa para números altos de 6502

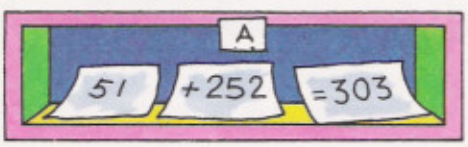
Este es un programa para sumar números mayores de 255 en el 6502. Antes de ejecutarlo debes calcular los bytes de bajo y alto orden de los dos números e introducirlos en la memoria de la forma descrita en la página opuesta.

Nemónicos	Códigos hex.
CLC	18
LDA dirección W	AD dirección W
ADC dirección X	6D dirección X
STA dirección Y	8D dirección Y
LDA dirección W1	AD dirección W1
ADC dirección X1	6D dirección X1
STA dirección Y1	8D dirección Y1
LDA #&0	A9 00
ADC #&0	69 00
STA dirección Z	8D dirección Z
RTS	60

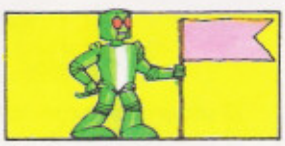




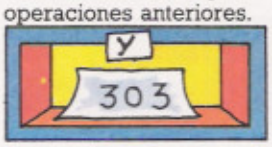
En primer lugar, el programa borra la bandera de acarreo por si ha tomado valor por operaciones anteriores.

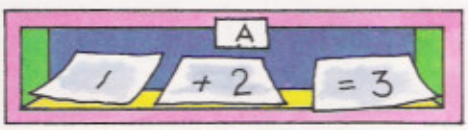


A continuación pone el byte de bajo orden del primer número en el acumulador y suma con acarreo el byte de bajo orden del segundo número (líneas 2 y 3).

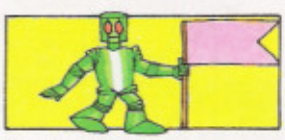


Si el resultado es mayor de 255, da el valor 1 a la bandera de acarreo.





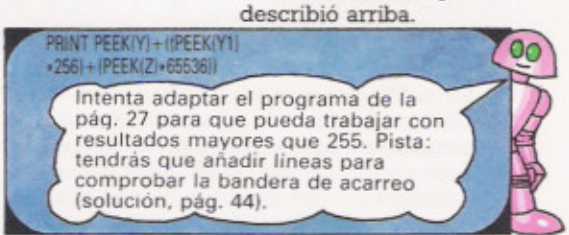
Almacena el resultado en la ubicación Y (línea 4). Después suma los dos bytes de alto orden y el acarreo (en caso de que lo haya) de la suma anterior. Almacena el resultado en la ubicación Y1 (línea 7).



Las líneas 8-10 comprueban si la bandera de acarreo tenía valor 1 siguiendo el mismo método que se describió arriba.

Ver la solución

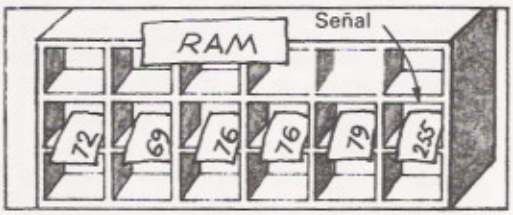
La solución se almacena en forma de tres bytes. El byte de bajo orden (ubicación Y) indica el número de unidades. El byte de alto orden (ubicación Y1) indica el número de 256s. Esta vez, el acarreo (ubicación Z) indica el número de 65536s. Para ver el resultado usa la instrucción que se indica a la derecha (sustituye Y, Y1 y Z por las direcciones de tu computadora).



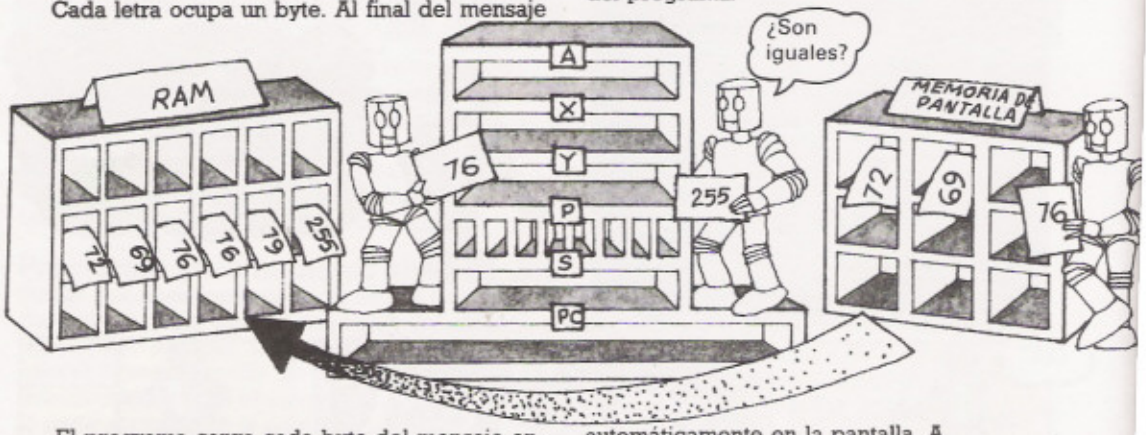
Visualización de un mensaje en la pantalla

El siguiente programa muestra cómo usar el código de máquina para visualizar un mensaje en la pantalla. El programa para Z80 está en la página opuesta, mientras que el de 6502 está en la página 34. Ambos programas siguen los mismos principios básicos, aunque el método de ambos microprocesadores es ligeramente diferente*.

Cómo funciona el programa

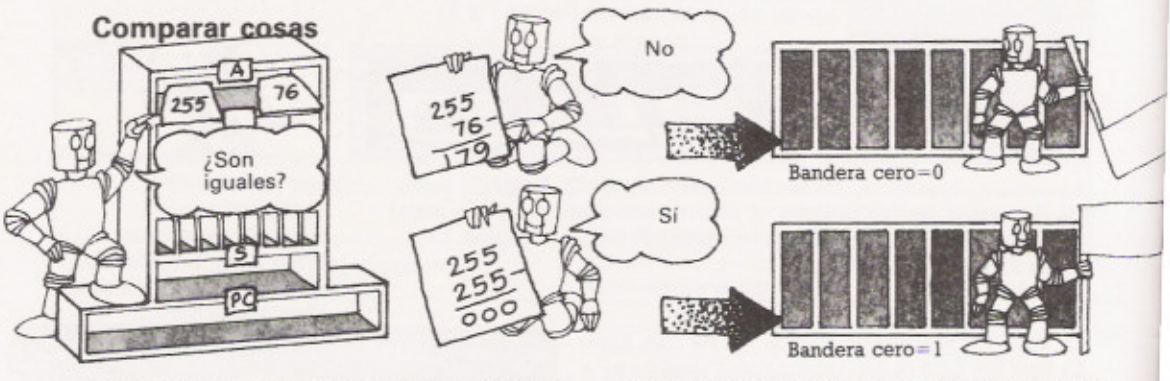


En primer lugar introduces el código de carácter para cada letra de tu mensaje en ubicaciones al principio de tu zona RAM libre. Cada letra ocupa un byte. Al final del mensaje introduce el código 255 como señal para indicar a la computadora que ése es el final del programa.



El programa carga cada byte del mensaje en el acumulador y lo compara con 255. Si el byte del mensaje no es igual a 255, lo almacena en la memoria de la pantalla y se visualiza automáticamente en la pantalla. A continuación, la computadora vuelve al principio del programa para buscar el siguiente byte del mensaje de la memoria.

Comparar cosas



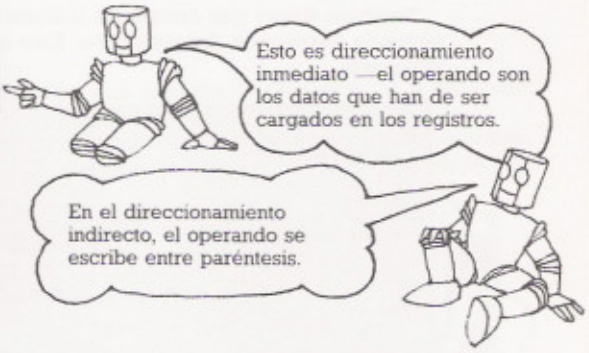
En el Z80 usa el operador CP y en el 6502 el CMP para decir a la computadora que compare un byte con el que haya en el acumulador. La computadora los compara restando uno del otro (esto es sólo una comprobación, los bytes no sufren ninguna alteración). Si el resultado es 0, los dos bytes son iguales y la bandera cero del registro de banderas toma el valor 0. Según cuál sea el valor de la bandera cero, puedes decirle a la computadora que vaya a una u otra parte del programa o que continúe con la siguiente instrucción.

*En el Spectrum (Timex 2000) no obtendrás un mensaje inteligible en la pantalla debido a la forma en que está organizada la memoria de pantalla.

Programa de mensajes para Z80

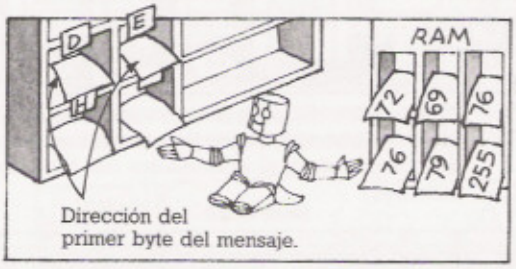
Estos son los nemónicos y codigos hexadecimales para Z80. Antes de ejecutar el programa, introduce tu mensaje en el RAM libre. A continuación completa las direcciones de las líneas 1 y 2 del programa. La última instrucción del programa indica a la computadora que salte hasta la tercera instrucción. Tendrás que introducir la dirección en la que está almacenada la tercera instrucción en la última línea del programa.

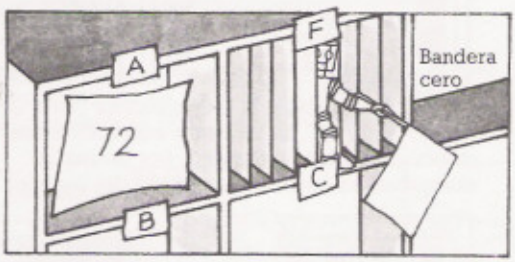
Nemónicos	Códigos hex.
LD HL, dirección pantalla	21, dirección pantalla
LD DE, dirección mensaje	11, dirección mensaje
LD A, (DE)	1A
CP,&FF	FE,FF
RET Z	C8
LD (HL),A	77
INC, DE	13
INC, HL	23
JP, dirección de la tercera instrucción	C3, dirección de la tercera instrucción



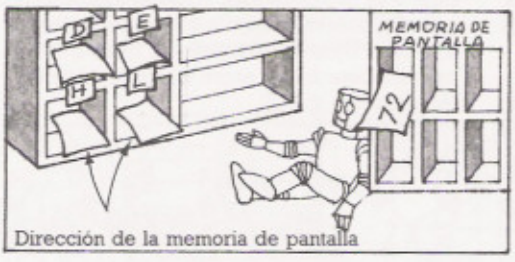
En este programa, los pares de registros HL y DE se utilizan como indicadores de las direcciones en las que la computadora debe almacenar o recoger datos. Esto se denomina «direccionamiento indirecto». Las instrucciones de las líneas seis y siete usan direccionamiento indirecto.

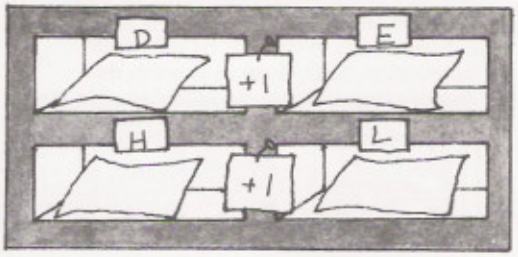
En las dos primeras líneas, la computadora pone la dirección de pantalla (la dirección en la que deben almacenarse los datos) en el par de registros HL y la dirección del mensaje (la dirección de la que deben recogerse los datos) en el par de registros DE.





LD A, (DE) le dice a la computadora que lea la dirección en DE y luego busque el byte de esa dirección y la ponga en el acumulador. Esto es direccionamiento indirecto. A continuación compara el byte del acumulador con &FF (el hexadecimal de 255). RET Z le dice a la computadora que retorne a BASIC si la bandera cero vale 1 (es decir, si el byte es igual a 255). Si la bandera cero vale 0, continuará con la siguiente instrucción.





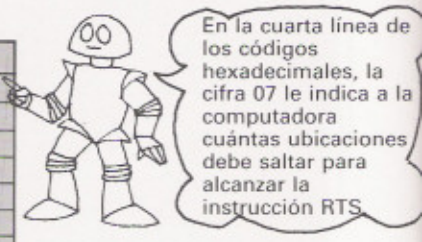
LD (HL), A también utiliza direccionamiento indirecto. Le dice a la computadora que lea la dirección en HL y almacene el contenido del acumulador (el byte del mensaje) en la ubicación de esa dirección. INC es el nemónico de «incremento» y significa aumento en uno. En las líneas siete y ocho, la computadora suma uno a las direcciones contenidas en DE y HL para que cuando vuelva a la construcción de la tercera línea recoja el byte de mensaje de la siguiente ubicación de memoria.

Programa de mensajes para 6502

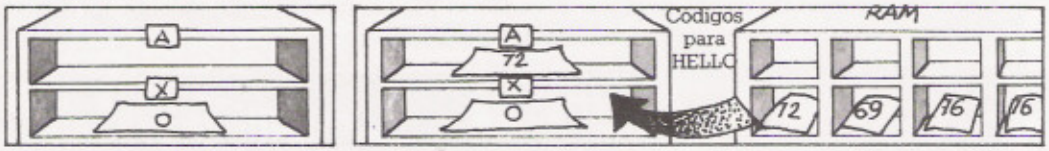
Estos son los nemónicos y códigos hexadecimales para el 6502. Antes de ejecutar el programa tendrás que introducir los códigos de los caracteres de tu mensaje en la zona de RAM libre, seguidos de un 255, es decir, de la señal de fin de mensaje. A continuación pon en la segunda línea del programa la dirección, en hexadecimal, de la primera ubicación en la que esté almacenado el mensaje. Pon una dirección en la memoria de pantalla de tu computadora en la quinta línea.

También tienes que completar la línea séptima con la dirección en la que va a almacenarse la segunda instrucción del programa. Esto hará que la computadora salte de vuelta para repetir el programa.

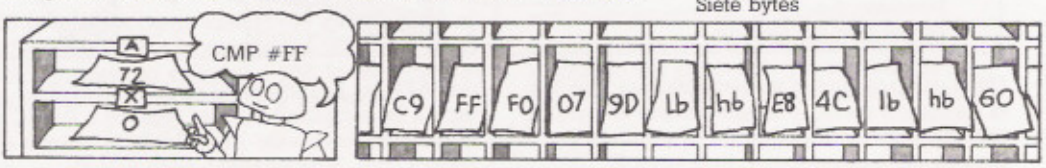
Nemónicos	Códigos hexadecimales
LDX#&00	A2 00
LDA dirección del mensaje	BD dirección del mensaje
CMP#&FF	C9 FF
BE Q a instrucción RTS	F0 07
STA dirección de pantalla	9D dirección de pantalla
INX	E8
JMP dirección 2.ª instrucción	4C dirección 2.ª instrucción
RTS	60



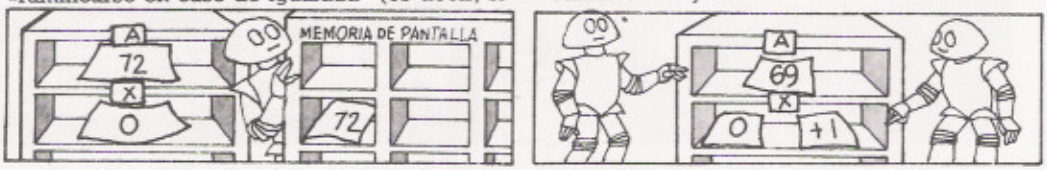
Este programa usa otra modalidad de direccionamiento denominada «direccionamiento indexado». En el direccionamiento indexado el contenido de los registros X o Y se suman al operando para dar la dirección en la que están almacenados los datos. Las líneas quinta y sexta usan direccionamiento indexado.



En la primera línea, la computadora pone un 0 en el registro X. La segunda instrucción usa direccionamiento indexado, por lo que la computadora suma el contenido del registro X a la dirección dada en la instrucción. El resultado será la dirección de los datos que tienen que ser cargados en el acumulador (un byte de mensaje).



CMP en la tercera línea hace que la computadora compare el byte del acumulador con &FF (hexadecimal de 255), es decir, con la señal de fin del mensaje. Si son iguales, da a la bandera cero el valor 1. La siguiente instrucción, BEQ, tiene la función de «ramificarse en caso de igualdad» (es decir, si la bandera cero es uno). En los códigos hexadecimales va seguido de un número que dice a la computadora cuántas ubicaciones saltar. Queremos que la computadora vaya a RTS si el byte de mensaje es igual a 255 y existen siete bytes entre la instrucción de ramificación y RTS.



A continuación, en la línea quinta, el programa usa direccionamiento indexado para almacenar el byte del acumulador (el byte de mensaje) en la dirección dada en la instrucción, más X.

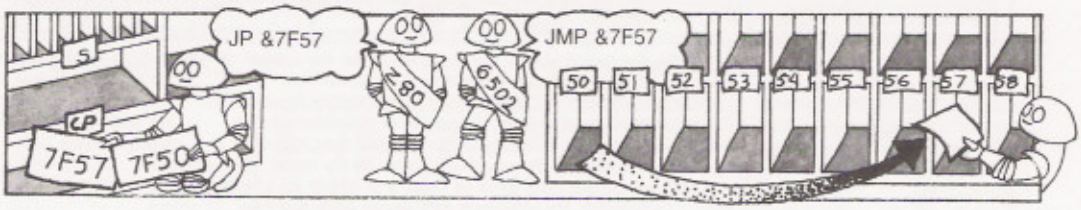
INX quiere decir «incremento de X» y hace que la computadora sume 1 al contenido del registro X. Después saltará de vuelta a la segunda instrucción. Esta vez, X es 1, por lo que carga el siguiente byte del mensaje en el acumulador y lo almacena en la siguiente ubicación de pantalla.

Saltos y ramificaciones

Se denomina ramificar a hacer que la computadora vaya a una instrucción en otra parte del programa. Existen tres tipos diferentes de ramificaciones: saltos, subrutinas y ramificaciones condicionales. En la ramificación condicional, la computadora lleva a cabo una comprobación y a continuación o bien se ramifica o bien continúa con la siguiente instrucción dependiendo del resultado de la comprobación. En la página siguiente hallarás más cosas sobre la ramificación condicionada. Los saltos simplemente le dicen a la computadora que vaya a una determinada dirección.

El contador del programa

El contador de programa es un registro especial de 16 bit que contiene la dirección de la siguiente instrucción que va a ejecutar la computadora. La computadora lee el número del contador de programa para, a continuación, dirigirse a la ubicación con esa dirección para recoger su siguiente instrucción. Tras este proceso, el contador aumenta en uno, por lo que dirige a la computadora a la siguiente ubicación de memoria.

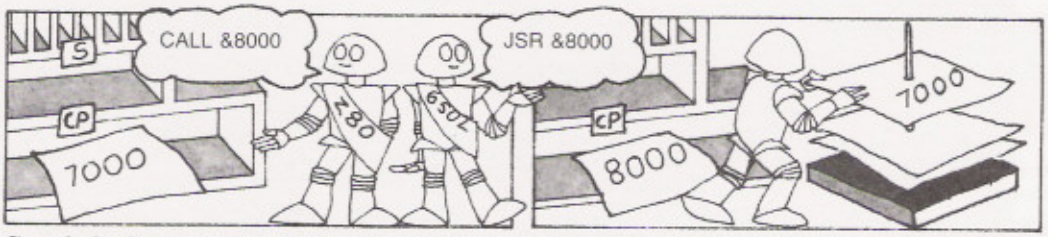


Salto

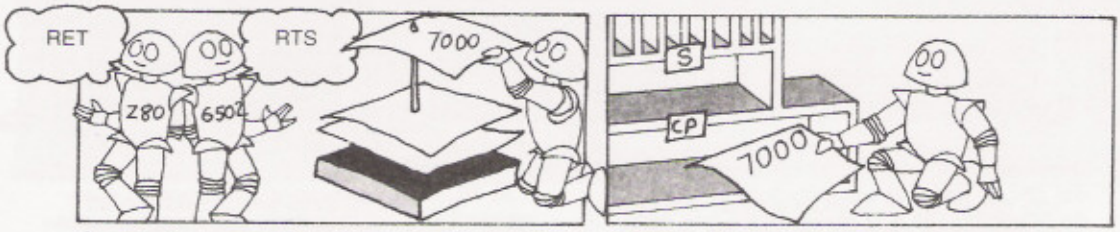
Cuando le dices a la computadora que salte o se ramifique a una determinada dirección, esa dirección se pone en el contador de programa y la computadora lleva a cabo secuencialmente todas las instrucciones a partir de esa dirección. Los operadores de Z80 y 6502 se muestran arriba.

Subrutinas

La instrucción «CALL dirección» en el Z80 y «JSR dirección» (saltar a la subrutina) en el 6502 le dicen a la computadora que vaya a una subrutina. Funciona de igual forma que en BASIC, por lo que al final de la subrutina necesitas una instrucción retorno (RET en el Z80 y RTS en el 6502).



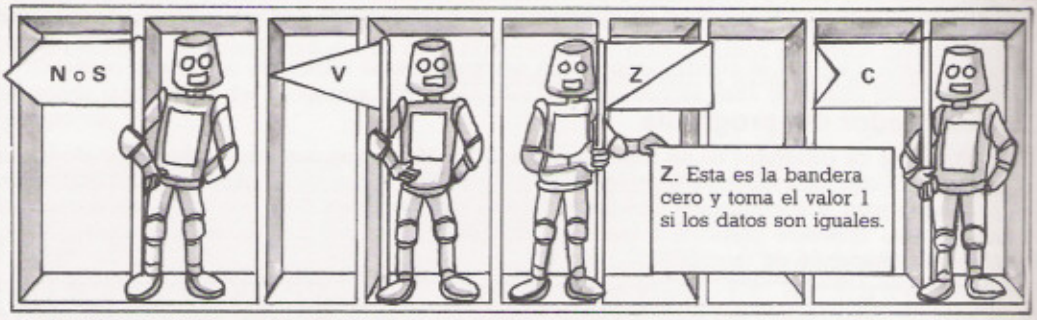
Cuando le dices a la computadora que vaya a una subrutina, la dirección de la subrutina se pone en el contador del programa. El contenido de contador (la dirección de la instrucción detrás de CALL o JSR) se almacena o «empuja» en la pila. La pila es una parte especial del RAM reservada para uso de la computadora (ver página 10).



Cuando la computadora llega a una instrucción RTS o RET al final de una subrutina saca el último dato de la pila y la pone en el contador del programa. Esta será la dirección de la instrucción posterior a la que le manda a la subrutina. Esto es también lo que sucede cuando le dices a la computadora que ejecute un programa en código de máquina.

Ramificaciones condicionales

En una ramificación condicionada, la computadora comprueba uno de los bits del registro de banderas según sea el resultado, o bien se ramifica o bien continúa con la siguiente instrucción. A están los bits del registro de banderas que puedes comprobar en las ramificaciones condicionada



N o S. Este es el bit de signo. Se denomina N en el 6502 y S en el Z80. Toma el valor 1 cuando el resultado de un cálculo es negativo y 0 para los resultados positivos.

V o P/V. Se denomina bit de superación de capacidad en el 6502. En el Z80 tiene dos funciones y se denomina paridad/superación de capacidad. Como bit de superación de capacidad toma el valor 1 cuando el resultado de un cálculo en notación complementaria a dos (ver página opuesta) resulta con un acarreo para el bit de signo.

Como bit de paridad toma el valor 1 si existe un número impar de unos en un byte y su función es para fines de comprobación.

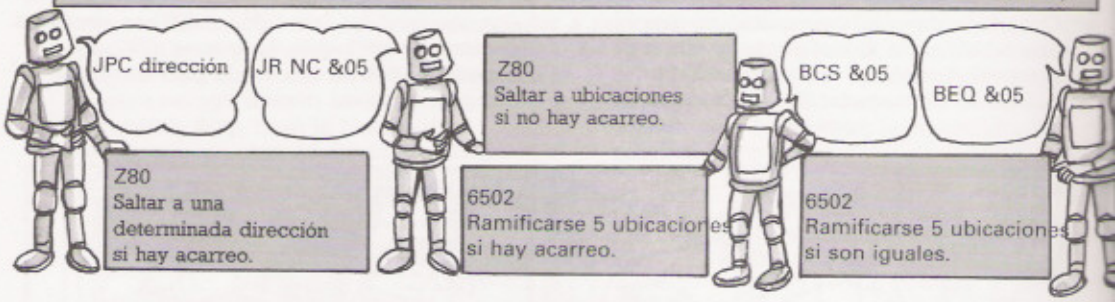
C. Esta es la bandera de acarreo. Toma el valor 1 cuando el total de una operación no encaja en un byte.

Algunas instrucciones, además de la instrucción de comparación, hacen que estas banderas sean automáticamente borradas, es decir, que tomen el valor 0. Por ejemplo, en el 6502 la instrucción DEC (decremento) afecta al signo y a la bandera cero.*

Operadores para ramificaciones condicionales

Estas son las instrucciones de ramificaciones condicionadas para comprobar cada bit.

Z80 Salto si...		6502 Ramificación si...	
JP C	Existe acarreo (C=1).	BCS	Existe acarreo (C=1).
JP NC	No hay acarreo (C=0)	BCC	No hay acarreo (C=0)
JP Z	Igual (Z=1)	BEQ	Igual (Z=1)
JP NZ	No igual (Z=0)	BNE	No igual (Z=0)
JP M	Negativo (S=1)	BMI	Negativo (N=1)
JP P	Positivo (S=0)	BPL	Positivo (N=0)
JP PO	Paridad impar (P/V=1)	BVS	Con superación de capacidad (V=1)
JP PE	Paridad par (P/V=0)	BVC	Sin superación de capacidad (V=0)

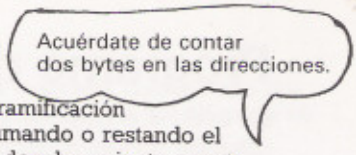


Después de la instrucción «JP comprobación» del Z80, le das a la computadora la dirección de la instrucción a la que quieres que salte. En el 6502 le das a la computadora un número que le indica cuántas ubicaciones tiene que saltar hacia delante o hacia detrás para encontrar la instrucción. Esto se denomina «direccionamiento relativo» y el número en sí, «desplazamiento.

El Z80 tiene también una instrucción para ramificación condicionada «JR comprobación» con la que se utiliza el desplazamiento en lugar de una dirección. JR quiere decir «jump (salto) relativo» y puede usarse sólo comprobando la bandera cero y la bandera de acarreo.

*Una lista completa de las instrucciones de tu microprocesador te indicará qué instrucciones afectan a qué banderas.

Calcular el desplazamiento



Cuando le das a la computadora un número de desplazamiento en una ramificación condicional, la computadora calcula la dirección a la que debe saltar sumando o restando el número de desplazamiento del contador del programa. Para calcular el desplazamiento cuenta el número de bytes hasta la instrucción a la que quieres que salte, incluyendo a ésta. Comienza en la instrucción siguiente a la de la ramificación condicionada y cuéntala como 0 (ya que el contador de programa ya indicará esa instrucción). Por ejemplo, aquí tienes dos programas breves de 6502 con los que aprenderás a calcular el desplazamiento (el método es igual en el Z80).

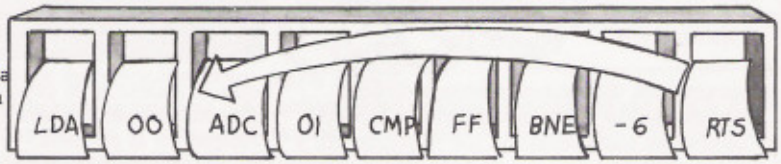
```
LDA dirección
CMP #&FF
BNE a RTS
STA dirección
RTS
```

En el ejemplo superior, para hacer que la computadora salte a la instrucción RTS el desplazamiento es 3.

En el ejemplo inferior, el desplazamiento para hacer que la computadora salte de vuelta a la instrucción ADC es −6.

```
LDA #&00
ADC #&01
CMP #&FF
BNE a ADC
RTS
```

Cuenta esta instrucción como 0.



Saltos hacia delante y hacia detrás

Para los saltos hacia delante simplemente convierte el desplazamiento en un número hexadecimal e introdúcelo en el programa. No obstante, para los saltos hacia detrás el desplazamiento es un número negativo y no hay manera de indicar números negativos en un número binario de ocho bit. Por ello, en estos casos se usa un sistema diferente conocido como «notación complementaria a dos». En esta notación, el bit de la izquierda se usa como signo de bit. Si este bit es 1, el número es negativo. Si es cero, el número será positivo.

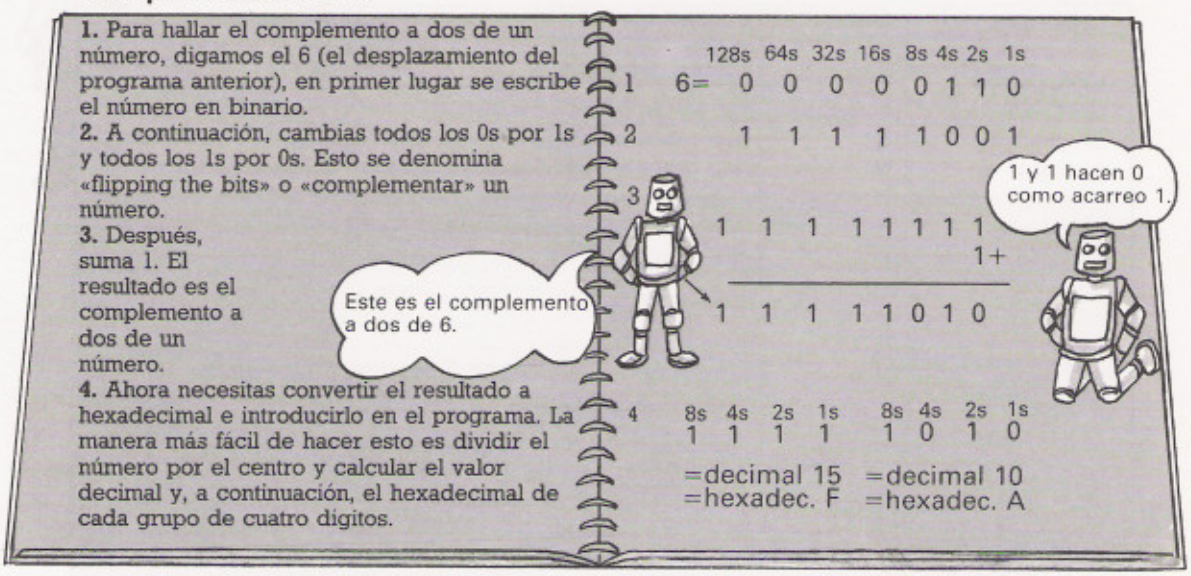
Complemento a dos

1. Para hallar el complemento a dos de un número, digamos el 6 (el desplazamiento del programa anterior), en primer lugar se escribe el número en binario.

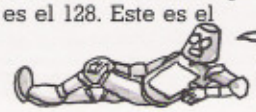
2. A continuación, cambias todos los 0s por 1s y todos los 1s por 0s. Esto se denomina «flipping the bits» o «complementar» un número.

3. Después, suma 1. El resultado es el complemento a dos de un número.

4. Ahora necesitas convertir el resultado a hexadecimal e introducirlo en el programa. La manera más fácil de hacer esto es dividir el número por el centro y calcular el valor decimal y, a continuación, el hexadecimal de cada grupo de cuatro dígitos.



La representación hexadecimal del complemento a dos de 6 es FA, por lo que éste es el número que insertarías para un salto hacia detrás en el programa anterior. En el complemento a dos, el número más alto que puedes representar es el 128. Este es el máximo desplazamiento hacia detrás que puedes realizar. El máximo desplazamiento hacia delante es 127, el máximo número que puedes formar con el octavo dígito binario convertido en 0 para indicar que es número positivo.



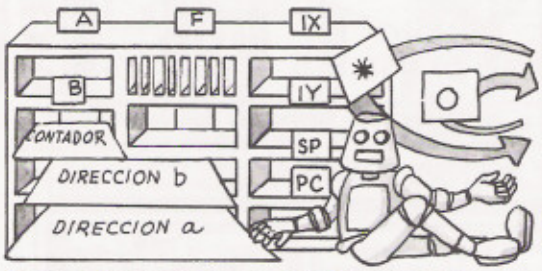
¿Sabrías calcular el hexadecimal del complemento dos de los números 12, 18 y 19? (La solución en la página 48.)

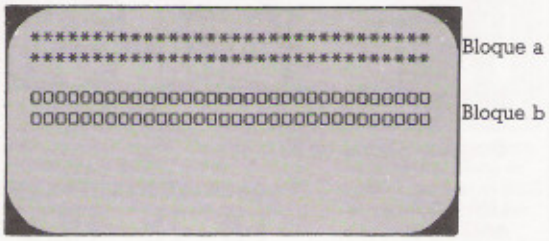
Programa de efectos intermitentes en pantalla

En estas dos páginas encontrarás un programa que visualiza alternativamente dos bloques creando un efecto en intermitencia luminosa. Muestra cómo funciona la animación a niveles sencillos. En la parte inferior está el programa para Z80, mientras que el de 6502 está en la página opuesta. Al final encontrarás unas guías sobre cómo ejecutar el programa en ambos microprocesadores.

Efectos intermitentes en el Z80

Dicho de una manera sencilla, el programa alterna la visualización de los dos bloques cargando un byte de cada bloque en los registros y almacenando el byte del bloque b en la dirección de pantalla para el bloque a y viceversa.



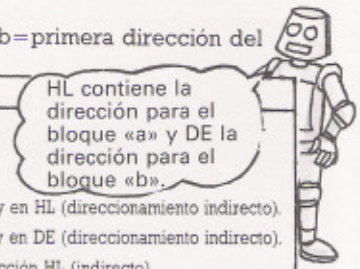


El programa usa direccionamiento indirecto. Las direcciones de pantalla del primer byte de cada bloque se almacenan en los registros HL y DE. La computadora lee las direcciones de estos registros cada vez que carga o almacena los bytes. Después de intercambiar los dos bytes, la instrucción INC (nemónico de incremento) suma uno a HL y a DE de tal forma que al repetirse el programa, éstas son las direcciones de los dos siguientes bytes de cada bloque en la pantalla.

El registro B contiene el número de bytes que son intercambiados cada vez que se repite el programa B disminuye en 1 por lo que actúa como un contador. Cuando B = 0 todos los bytes han sido intercambiados.

Programa para Z-80

n=número de bytes en un bloque; a=primera dirección del bloque a; b=primera dirección del bloque b.



Nemónicos	Códigos hex.	Significado
LD B, n	06, n	Contador.
LD HL, (dirección a)	21, dirección a	Pone dirección del bloque a en HL.
LD DE, (dirección b)	11, dirección b	Pone dirección del bloque b en DE.
LD C, (H)	4E	Carga C con el contenido de la dirección que hay en HL (direccionamiento indirecto).
LD A, (DE)	1A	Carga A con el contenido de la dirección que hay en DE (direccionamiento indirecto).
LD (HL), A	77	Almacena el contenido del acumulador en la dirección HL (indirecto).
LD A, C	79	Pone C (primer byte del bloque a) en el acumulador.
LD (DE), A	12	Almacena el contenido del acumulador en la dirección que hay en DE.
INC HL	23	Suma uno a HL y a DE.
INC DE	13	
DEC B	05	Decrimenta B, el contador.
LD A, & 00	3E, 00	Pone 0 en el acumulador.
CP B	B8	Compara B con el contenido del acumulador (0).
JR NZ a la instrucción 4	20, F3	Si B no es igual a cero, salta de vuelta a la ubicación &F3 para cargar los siguientes bytes en los registros. F3 es el hexadecimal del complemento a dos de 13 (ver página 37).
RET	C9	Retorno.

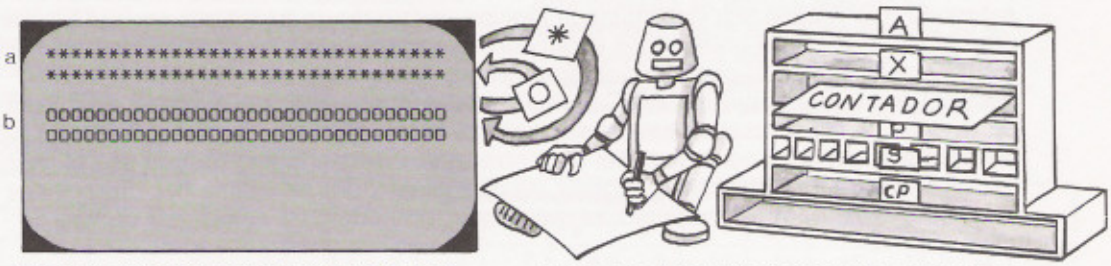
Completar los datos y direcciones

n (número de caracteres en un bloque). Para hallar n, multiplica el número de caracteres en una línea por el número de líneas en un bloque. Convierte a hexadecimal.

direcciones a y b. Si quieres intercambiar las dos líneas superiores de la pantalla con las dos siguientes, haz que la dirección a sea la primera dirección de la memoria de pantalla de tu computadora. La dirección b es la dirección a más el número de bytes que van a ser intercambiados. Convierte ambas direcciones a hexadecimal.

Efectos intermitentes para 6502

Este programa intercambia dos bloques byte a byte (es decir, carácter a carácter), comenzando con el último byte de cada bloque. Carga estos bytes en los registros y almacena el byte del bloque a en la ubicación de pantalla del bloque b y viceversa. A continuación se repite el programa para intercambiar el siguiente par de bytes.



Utiliza el direccionamiento indexado para encontrar la dirección para cada byte. El número total de bytes de un bloque se carga en el registro X. Después, para almacenar o cargar un byte, el número del registro X se suma a la dirección de inicio de cada bloque. La instrucción DEX (decremento de X) hace que la computadora reste 1 de X para que al repetirse el programa la computadora recoja el siguiente byte para visualizarlo.

Programa de efectos intermitentes para 6502

Mira la parte inferior de la página opuesta para saber cómo calcular los valores de n, a y b. Luego resta 1 de a y b para que cuando la computadora sume X obtenga la última dirección de cada bloque en lugar de la primera dirección de la siguiente línea.

Nemónicos	Códigos hexadecimales	Significado
LDX #n	A2 n	Cargar X con el número de bytes de un bloque.
LDA dirección a, X	BD dirección a	Pone en el acumulador el contenido de la ubicación de dirección a+X.
TAY	A8	Transfiere el contenido del acumulador al registro Y.
LDA dirección b, X	BD dirección b	Pone en el acumulador el contenido de la ubicación de dirección b+X.
STA dirección a, X	9D dirección a	Almacena el contenido del acumulador en la dirección a+X.
TYA	98	Transfiere el contenido del registro Y al acumulador.
STA dirección b, X	9D direcciones	Almacena el contenido del acumulador en la dirección b+X.
DEX	CA	Decrementa X. La bandera cero toma el valor 1 cuando X=0.
BNE a la instrucción dos	D0 EF	Ramifica de vuelta a la ubicación &EF si X no es igual a 0. EF es el hexadecimal del complemento a dos de 17 (ver página 37).
RTS	60	Retorno.

Cargar y ejecutar el programa para Z80 o para 6502

La mejor manera de ejecutar este programa es como una subrutina en código de máquina dentro del cargador hexadecimal. Para hacer esto, sigue estos pasos:

1. Copia el cargador hexadecimal y pon los códigos hexadecimales adecuados para tu computadora en el paso 160.

2. En el paso 180 necesitarás dos bucles para introducir los caracteres para la visualización en la memoria de pantalla. Por ejemplo, éstos son los pasos para dos filas de * (código 42) seguidos de dos filas de 0 (código 48) para una computadora con una pantalla de 40 columnas.

```
180 FOR J=0 TO 79
190 POKE primera dirección de pantalla+J,42
200 NEXT J
210 FOR J=80 TO 159
220 POKE primera dirección de pantalla+J,48
230 NEXT J
```

3. A continuación, añade los siguientes pasos al final del programa:

```
240 CALL dirección donde el código de máquinas está almacenado.
250 FOR K=1 TO 500
260 NEXT K          Cambia el número 500 por el que sea
270 GOTO 240        más adecuado para tu computadora.
```

4. Ahora escribe RUN para ejecutar el programa. El cargador hexadecimal introduce los códigos hexadecimales en la memoria y a continuación los códigos de visualización en la memoria de pantalla. El paso 240 la hace ir a la ubicación en la que está almacenado el programa en código de máquina para que ejecute sus instrucciones. Por sí solo, el programa en código de máquina sólo intercambia una vez, por lo que el paso 270 sirve para hacer que ejecute el programa una y otra vez creando un efecto de intermitencia. Necesitas el bucle de retardamiento debido a lo rápido que es el código de máquina.

Profundizando

Si quieres aprender más cosas sobre el código de máquina, lo mejor es que intentes escribir y probar tus propios programas y que estudies programas escritos por otras personas. Un interesante modo de usar el código de máquina es como una pequeña subrutina que realice una determinada función dentro de un programa en BASIC. Por ejemplo, el código de máquina es especialmente apropiado para distribuir datos o hacer representaciones en la pantalla debido a que es mucho más rápido y ocupa menos espacio de memoria que el BASIC. Este tipo de subrutinas los encontrarás con frecuencia en revistas. Si las subrutinas están escritas para tu computadora podrás ejecutarlas sin ninguna alteración. Si están escritas para otra marca de computadora que use el mismo microprocesador, tendrás que cambiar las direcciones del programa por direcciones en la zona de memoria de tu computadora que hayas escogido para almacenar código de máquina.

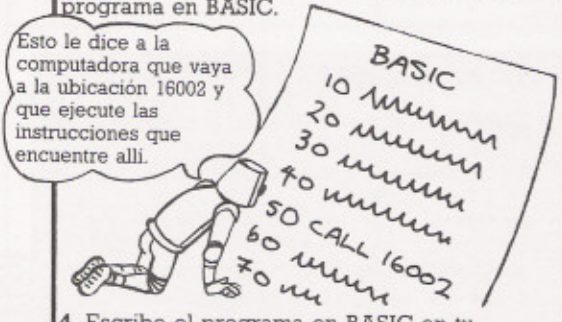
Subrutinas en código de máquina

Estos son los pasos que debes seguir para usar una subrutina en código de máquina en un programa en BASIC.

1. Reserva un espacio en la memoria para el código de máquina bajando el tope del RAM del usuario (ver página 20-22).

2. Pon los códigos para la subrutina en código de máquina en la línea 160 del programa cargador hexadecimal de la página 24. (Asegúrate de que hay una instrucción retorno al final del programa en código de máquina.) Si es necesario, añade líneas para introducir 2 bytes de datos, copia el cargador hexadecimal y ejecútalo.

3. Numera tu programa en BASIC con números más altos que el del último número de paso del cargador hexadecimal. En el punto del programa en el que quieres que la computadora ejecute el código de máquina, pon la instrucción que requiere tu computadora para ejecutar un programa en código de máquina como un paso más del programa en BASIC.



4. Escribe el programa en BASIC en tu computadora y escribe RUN. La computadora ejecutará las instrucciones en BASIC hasta que llega a la instrucción que le dice que ejecute el programa en código de máquina, tras lo cual irá a la dirección en la que está almacenado el código de máquina y ejecutará sus instrucciones. La instrucción retorno al final del código de máquina mandará a la computadora a la siguiente línea del programa en BASIC.

Uso de un ensamblador

Un ensamblador (un programa que te permite escribir un programa en código de máquina en nemónicos) hace mucho más fácil la programación en código de máquina. Puedes comprar un ensamblador en cinta para casi todas las computadoras personales, y algunas como la BBC tienen un ensamblador incorporado.

Con un ensamblador puedes escribir comentarios junto a los nemónicos para recordar qué hace cada línea. El ensamblador visualizará el programa en la pantalla en hexadecimal, con nemónicos y con los comentarios y direcciones en los que están almacenadas las instrucciones.

El ensamblador reserva automáticamente los pares de dígitos en la dirección y calcula la dirección o el desplazamiento para un salto. Algunos ensambladores te permiten usar nombres simbólicos para los datos de la misma manera que las variables en BASIC. Un buen ensamblador incluso te ayudará a buscar errores que podrás corregir con un editor.

Libros sugeridos

Existen multitud de libros sobre el código de máquina escritos especialmente para cada marca de microcomputadora. La mejor manera de escoger es leyendo artículos en las revistas de computadoras. También son interesantes los siguientes libros:

Programming the Z80 y *Programming the 6502*, ambos por Rodney Zaks y publicados por Sybex. Estas son guías detalladas con listas completas de todas las instrucciones para cada microprocesador. No resultan fáciles de leer para los principiantes pero son muy útiles como consulta.

VIC 20 Programmer's Reference Guide, publicado por Commodore.

6502 Machine Code for Beginners, por A. P. Stephenson, Newnes Microcomputer Books.

Tablas de conversión decimal/hexadecimal

Esta tabla convierte números hexadecimales entre 0 y FF en números decimales y viceversa.

Hexadecimal a decimal
Para convertir un número de hexadecimal a decimal busca la fila del primer dígito hexadecimal de tu número hexadecimal y la columna del segundo dígito hexadecimal. La casilla en la que se crucen será el equivalente decimal de tu número hexadecimal. Ej.: el hexadecimal A1 es el decimal 161.

Decimal a hexadecimal
Para convertir un número decimal en hexadecimal busca el número decimal en la tabla. Luego lee la fila en la que se halla para tener el primer dígito y la columna en la que se halla para tener el segundo. Ej.: 154 es 9A.

Primer dígito hexadecimal	Segundo dígito hexadecimal															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Conversión de direcciones

Para usar la tabla para convertir direcciones hexadecimales busca el equivalente decimal para el primer par de dígitos de la dirección. Este será el número de página. A continuación busca el equivalente decimal del segundo par de dígitos para tener la posición en la página. Multiplica el número de página por 256 y súmale la posición en la página.

Tabla de conversión del complemento a dos

Esta tabla da en hexadecimal el complemento a dos de números decimales entre −1 y −128. Para convertir un número en su complemento a dos, busca el número en la tabla y a continuación lee la fila en la que se halla para obtener el primer dígito hexadecimal y la columna para obtener el segundo dígito.

Primer dígito hexadec.	Segundo dígito hexadecimal															
	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
E	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
D	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
C	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
B	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
A	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
9	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128

Hacer conversiones con una calculadora

Cuando realizas conversiones con una calculadora, ésta te muestra el resto como un número decimal. Por ejemplo, si quieres convertir el decimal 134 a hexadecimal, divides entre 16 y conviertes el cociente y el resto en dígitos hexadecimales. Una calculadora te dará como solución 8,375.

Para transformar el resto en un número entero, resta el número que hay delante de la coma y multiplica el resultado por el número entre el que dividiste.

$8.375-8=0,375\times16=6$

Por lo que $134\div16=8$ con resto 6, por lo que 134 decimal es 86 en hexadecimal.

Nemónicos y códigos hexadecimales de Z80

En las próximas páginas encontrarás los nemónicos y códigos hexadecimales de las instrucciones tratadas en este libro. El término «direccionamiento implícito» usado en estas listas es simplemente el nombre de aquellas instrucciones que no requieren que se especifique ningún operando en el código hexadecimal. Existen algunas instrucciones no incluidas en estas listas, por lo que si deseas conocer más a fondo el código de máquina necesitarás una lista completa de las instrucciones de Z80 (ver página 40). Las siguientes abreviaturas se usan en las listas:

n =número
nn=número de dos bytes
r =registro
rr =par de registros
x=dirección
c=condición
d=desplazamiento

ADC A,n. Suma con acarreo un número n al acumulador. (Direccionamiento inmediato.)

ADC A,n	CE,n

ADC A,r. Suma con acarreo un registro r al acumulador. (Direccionamiento implícito.)

ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D

ADC HL, rr. Suma con acarreo el contenido del par de registros rr a HL. (Direccionamiento implícito).

ADC HL,BC	ED4A
ADC HL,DE	ED5A
ADC HL,HL	ED6A

ADD A,n. Suma un número n al acumulador. (Direccionamiento inmediato).

ADD,n	C6,n

ADD A,r. Suma el registro r al acumulador. (Direccionamiento implícito).

ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85

ADD EL,rr. Suma el contenido del par de registro rr a HL. (Direccionamiento implícito).

ADD HL,BC	09
ADD HL,DE	19
ADD HL,HL	29

CALL x. Ir a la subrutina que comienza en la dirección x.(Direccionamiento inmediato).

CALL x	CD x

CALL c,x. Ir a la subrutina que comienza en la dirección x dependiendo de la condición c. c puede ser Z (igual); NZ (no igual); c (acarreo); NC (no acarreo); PE (paridad par); PO (paridad impar); M (negativo), o P (positivo).(Direccionamiento inmediato).

CALL Z,x	CC,x
CALL NZ,x	C4,x
CALL C,x	DC,x
CALL NC,x	D4,x
CALL PE,x	EC,x
CALL PO,x	E4,x
CALL M,x	FC,x
CALL P,x	F4,x

CCF Complementar bandera de acarreo. (Direccionamiento implícito).

CCF	3F

CPn. Compara el contenido del acumulador con el dato n. (Direccionamiento inmediato).

CP n	FE n

CPr. Compara el contenido del registro r con el acumulador. (Direccionamiento implícito).

CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD

CP (HL). Compara el contenido del acumulador con el contenido de la dirección que hay en HL. (Direccionamiento indirecto).

CP (HL)	BE

DEC r. Decremento del registro r.(Direccionamiento implícito).

DEC A	3D
DEC B	05
DEC C	0D
DEC D	15
DEC E	1D
DEC H	25
DEC L	2D

DECrr. Decremento del par de registros rr. (Direccionamiento implícito).

DEC BC	0B
DEC DE	1B
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B

DEC (HL). Decremento del contenido de la dirección contenido en HL. (Direccionamiento indirecto).

DEC (HL)	35

INCr. Incremento del registro r. (Direccionamiento implícito).

INC A	3C
INC B	04
INC C	0C
INC D	14
INC E	1C
INC H	24
INC L	2C

INC rr. Incremento del par de registros rr. (Direccionamiento implícito).

INC BC	03
INC DE	13
INC HL	23

INC (HL). Incremento del contenido de la dirección que hay en HL. (Direccionamiento indirecto).

INC (HL)	34

JPx. Salto a la dirección x. (Direccionamiento inmediato).

JP x	C3 x

JP (rr). Salto a la dirección contenida en el par de registros rr. (Direccionamiento implícito).

JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9

JP c,x. Salto a la dirección x dependiendo de la condición c. c puede ser Z (igual); NZ (no igual); C (acarreo); NC (no acarreo); PE (Paridad par); PO (paridad impar; M (negativo); P (positivo). (Direccionamiento inmediato).

JP Z,x	CA,x
JP NZ,x	C2,x
JP C,x	DA,x
JP NC.x	D2,x
JP PE,x	EA,x
JP PO,x	E2,x
JP M,x	FA,x
JP P,x	F2,x

JR d. Salto relativo. Saltar d bytes (el desplazamiento). (Direccionamiento relativo).

JR d	18 d

JR c,d. Salto relativo. Saltar d bytes (el desplazamiento) dependiendo de la condición c. c puede ser NZ (no igual); Z (igual); NC (no acarreo, o C (acarreo). (Direccionamiento relativo).

JR NZ,d	20,d
JR Z,d	28,d
JR NC,d	30,d
JR C,d	38,d

LD r,n. Cargar el registro r con el dato n. (Direccionamiento inmediato).

LD A,n	3E,n
LD B,n	06,n
LD C,n	0E,n
LD D,n	16,n
LD E,n	1E,n
LD H,n	26,n
LD L,n	2E,n

LD rr, nn. Cargar el par de registros rr con el número de dos bytes nn. (Direccionamiento inmediato).

LD BC,nn	01,nn
LD DE,nn	11,nn
LD HL,nn	21,nn

LD A, (x). Cargar el acumulador con el contenido de la dirección x. (Direccionamiento absoluto).

LD A, (x)	3A, (x)

LD rr, (x). Cargar el par de registros rr con el contenido de las direcciones x y x+1. (Direccionamiento absoluto).

LD BC, (x)	ED4B, (x)
LD DE, (x)	ED5B, (x)
LD HL, (x)	2A, (x)

LD A,r. Cargar el acumulador con el contenido del registro r. (Direccionamiento implícito).

LD A,A	7F
LD A,B	78
LD A,C	79
LD A,D	7A
LD A,E	7B
LD A,H	7C
LD A,L	7D

LD B,r. Cargar el registro B con el contenido del registro r. (Direccionamiento implícito).

LD B,A	47
LD B,B	40
LD B,C	41
LD B,D	42
LD B,E	43
LD B,H	44
LD B,L	45

LD C,r. Cargar el registro C con el contenido del registro r. (Direccionamiento implícito).

LD C,A	4F
LD C,B	48
LD C,C	49
LD C,D	4A
LD C,E	4B
LD C,H	4C
LD C,L	4D

LD D,r. Cargar el registro D con el contenido del registro r. (Direccionamiento implícito).

LD D,A	57
LD D,B	50
LD D,C	51
LD D,D	52
LD D,E	53
LD D,H	54
LD D,L	55

LD E,r. Cargar el registro E con el contenido del registro r. (Direccionamiento implícito).

LD E,A	5F
LD E,B	58
LD E,C	59
LD E,D	5A
LD E,E	5B
LD E,H	5C
LD E,L	5D

LD H,r. Cargar el registro H con el contenido del registro r. (Direccionamiento implícito).

LD H,A	67
LD H,B	60
LD H,C	61
LD H,D	62
LD H,E	63
LD H,H	64
LD H,L	65

LD L,r. Cargar el registro L con el contenido del registro r. (Direccionamiento implícito).

LD L,A	6F
LD L,B	68
LD L,C	69
LD L,D	6A
LD L,E	6B
LD L,H	6C
LD L,L	6D

LD r, (rr). Cargar el registro r con el contenido de la dirección contenida en el par de registros rr. (Direccionamiento indirecto).

LD A, (BC)	0A
LD A,(DE)	1A
LD A,(HL)	7E
LD B,(HL)	46
LD C,(HL)	4E
LD D,(HL)	56
LD E,(HL)	5E
LD H,(HL)	66
LD L,(HL)	6E

LD (x),A. Almacena el contenido del acumulador en la dirección x. (Direccionamiento absoluto).

LD (x),A	32,x

LD (x),rr. Almacena el contenido del par de registros rr en las direcciones x y x+1. (Direccionamiento absoluto).

LD (x),BC	ED43,x
LD (x),DE	ED53,x
LD (x),HL	22,x

LD (rr),r. Almacena el contenido del registro r en la dirección contenida en el par de registros rr. (Direccionamiento indirecto).

LD (BC),A	02
LD (DE),A	12
LD (HL),A	77
LD (HL),B	70
LD (HL),C	71
LD (HL),D	72
LD (HL),E	73
LD (HL),H	74
LD (HL),L	75

LD (rr), n. Almacena el dato n en la dirección contenida en el par de registros rr. (Direccionamiento inmediato/indirecto).

LD (HL),n	36

RET. Retorno desde una subrutina. (Direccionamiento indirecto).

RET	C9

RET c. Retorno desde una subrutina dependiendo de la condición c. c puede ser Z (igual); NZ (no igual); C (acarreo); NC (no acarreo); PE (paridad par); PO (paridad impar); P (positivo); M (negativo). (Direccionamiento indirecto).

RET Z	C8
RET NZ	C0
RET C	D8
RET NC	D0
RET PE	E8
RET PO	E0
RET M	F8
RET P	F0

SBC A,n. Resta con acarreo el dato n del acumulador. (Direccionamiento inmediato).

SBC A,n	DE,n

SBC A,r. Resto con acarreo el contenido del registro r del acumulador. (Direccionamiento implícito).

SBC A,A	9F
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,H	9C
SBC A,L	9D

SBC HL,rr. Resta con acarreo el contenido del par de registros rr del par de registros HL. (Direccionamiento implícito).

SBC HL,BC	ED42
SBC HL,DE	ED52
SBC HL,HL	ED62

SBC A,(HL). Resta con acarreo el contenido de la dirección que hay en el par de registros HL del acumulador. (Direccionamiento indirecto).

SBC A,(HL)	9E

SCF. Dar valor a la bandera de acarreo. (Direccionamiento implícito).

SCF	37

SUB n. Resta el dato n del acumulador. (Direccionamiento inmediato).

SUB, n	D6, n

SUB r. Resta el contenido del registro r del acumulador.(Direccionamiento inmediato).

SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95

SUB (HL). Resta el contenido de la dirección que hay en HL del acumulador. (Direccionamiento indirecto).

SUB (HL)	96

Respuestas a los problemas

Consejo: una manera fácil de calcular el complemento a dos de un número es restarlo de 256 y convertir la solución a hexadecimal. Ej.: 256−6=250 que es FA en hexadecimal.

Página 11

&A7 en decimal es 167. 513 en hexadecimal es &201.

Página 26

1. 25+73 (25 es &19 y 73 es &49)

Z80 Nemónicos	Códigos hexadecim.	6502 Nemónicos	Códigos hexadecim.	Significado
LD A, &19	3E, 19	LDA #&19	A9 19	Pone &19 en el acumulador.
ADD A, &49	C6, 49	ADC #&49	69 49	Suma &49 al acumulador.
LD (dirección), A	32, dirección	STA dirección	8D dirección	Almacena el contenido del acumulador en una determinada dirección.
RET	C9	RTS	60	Retorno

2. 64+12+14 (64 es &40, 12 es &0C y 14 es &0E)

Z80 Nemónicos	Códigos hexadecim.	6502 Nemónicos	Códigos hexadecim.	Significado
LD A, &40	3E, 40	LDA #&40	A9 40	Pone &40 en el acumulador.
ADD A, &0C	C6, 0C	ADC #&0C	69 0C	Suma &0C al acumulador.
ADD A, &0E	C6, 0E	ADC #&0E	69 0E	Suma &0E al acumulador.
LD (dirección), A	32, dirección	STA dirección	8D dirección	Almacena el contenido del acumulador en una determinada dirección.
RET	C9	RTS	60	Retorno

Las respuestas a los problemas continúan en la página 48.

Nemónicos y códigos hexadecimales de 6502

Esta tabla muestra los nemónicos y los códigos hexadecimales de todas las instrucciones tratadas en este libro (y algunos más). Los nemónicos de cada instrucción se encuentran a la izquierda y sus códigos hexadecimales para las diferentes modalidades de direccionamiento se muestran a lo largo de la tabla. El direccionamiento a la página cero es igual que el direccionamiento absoluto en donde el operando es la dirección en la que están almacenados los datos, siendo ésta una dirección de la página cero (es decir, ubicaciones 0-255) de la memoria (ver página 10). Direccionamiento implícito es el término usado para describir instrucciones en las que no hace falta especificar un operando. Ej.: CLC. Existe un número de instrucciones que no se tratan aquí, por lo que si quieres conocer más a fondo el código de máquina necesitarás una lista completa de las instrucciones del 6502.

Modalidad de direccionamiento	Inmediato	Absoluto	Página cero	Indexado X	Indexado Y	Implícito	Relativo
Operando	Dato	Cualquier dirección	Direcciones en la página cero	Direcciones +registro X	Direcciones +registro Y	Ninguno	Desplazamiento
ADC Suma de acarreo, es decir, suma un byte más la bandera de acarreo al acumulador.	69	6D	65	7D	79		
BCC Ramificación si el acarreo está vacío.							90
BCS Ramificación si el acarreo tiene valor.							B0
BEQ Ramificación si iguales.							F0
BMI Ramificación si negativo.							30
BNE Ramificación si no iguales.							D0
BPL Ramificación si positivo.							10
BVC Ramificación si no hay desbordamiento.							50
BVS Ramificación si hay desbordamiento.							70
CLC Borrar bandera de acarreo.						18	
CMP Compara con el acumulador.	C9	CD	C5	DD	D9		
CPX Compara con el registro X.	E0	EC	E4				
CPY Compara con el registro Y.	C0	CC	C4				
DEC Decremento (restar 1 de) ubicación de memoria.		CE	C6	DE			
DEX Decremento (restar 1 de) registro X.						CA	
DEY Decremento (restar 1 de) registro Y.						88	
INC Incremento (sumar 1 a) ubicación de memoria.		EE	E6	FE			
INX Incremento (sumar 1 a) registro X.						E8	
INY Incremento (sumar 1 a) registro Y.						C8	
JMP Salto a la dirección especificado en el operando.		4C					
JSR Salto a la subrutina que comienza en la dirección especificada el operando.		20					
LDA Cargar el acumulador.	A9	AD	A5	BD	B9		
LDX Cargar registro X.	A2	AE	A6		BE		
LDY Cargar registro Y.	A0	AC	A4	BC			
RTS Retorno desde subrutina.						60	
SBC Resta con acarreo. Resta del acumulador y toma prestado de la bandera de acarreo.	E9	ED	E5	FD	F9		
SEC Dar valor a la bandera de acarreo.						38	
STA Almacena el acumulador en una determinada dirección.		8D	85	9D	99		
STX Almacena el registro X en una determinada dirección.		8E	86				
STY Almacena el registro Y en una determinada dirección.		8C	84				
TAX Transfiere el acumulador al registro X.						AA	
TAY Transfiere el acumulador al registro Y.						8A	
TXA Transfiere el registro X al acumulador.						8A	
TYA Transfiere el registro Y al acumulador.						98	

Observa que todas las instrucciones no pueden usarse en todas las modalidades de direccionamiento.

Palabras del código de máquina

Acumulador. El registro que contiene los bytes de información sobre los que se van a realizar operaciones lógicas o cálculos aritméticos.
Bandera. Un bit en el registro de banderas que se usa para indicar una cierta condición. Ej.: la presencia de un número negativo o de acarreo tras una suma.
Bandera cero. El bit en el registro de banderas que indica cuándo el resultado de una operación es 0, usándose también para indicar cuándo dos bytes son iguales.
Bandera de acarreo. del registro de banderas que toma el valor 1 cuando el resultado de una suma no cabe en ocho bits.
Bandera de signo. El bit del registro de banderas que se usa para indicar números positivos y negativos.
Binario. Un sistema de números que usa dos dígitos, el 0 y el 1, y en el que cada dígito del número tiene el doble de valor que el dígito a su derecha.
Bit. Una unidad simple del código de las computadoras, es decir, un 1 o un 0 que simbolizan una pulsación o un no-pulsación.
Borrar. Hacer que un bit, como puede ser cualquier bit del registro de banderas, valga cero.
Buffer. Una zona de la memoria de la computadora donde se almacenan temporalmente los datos en su camino hacia o desde su destino final.
Byte. Un grupo de ocho pulsaciones y no-pulsaciones (bits) que representan la información en el código de computadora.
Byte de alto orden. Los dos primeros dígitos de una dirección hexadecimal que representan el número de página de memoria en la que se halla la dirección.
Byte de bajo orden. Los dos dígitos hexadecimales de una dirección que dan la posición de dicha dirección dentro de una página de memoria.
Cargador hexadecimal. Un programa en BASIC que convierte los códigos hexadecimales de un programa en código de máquina en números decimales y los introduce en la memoria de la computadora.
Código fuente. Un programa escrito en lenguaje ensamblador o en cualquier lenguaje de alto nivel como puede ser el BASIC.
Código objeto. Un programa que ha sido traducido a código de máquina a partir de uno en lenguaje ensamblador o en cualquier lenguaje de alto nivel.
Complementar. También llamada «flipping the bits», es el proceso de transformar todos los 1 de un byte en 0 y todos los 0 en 1.
Complementa a dos. Un sistema de rotación usado para representar números negativos. Para hallar el complemento a dos de un número, complementas (es decir, conviertes todos los 1 en 0 y todos los 0 en 1) el binario de dicho número y las sumas 1.
Contador de programa. El registro que contiene la dirección de la siguiente instrucción que va a ser buscada.
Desensamblador. Un programa que puede visualizar en la pantalla el contenido de una serie de ubicaciones de memoria en lenguaje ensamblador. Puedes conseguir un desensamblador en cinta y es muy útil para encontrar errores en programas en código de máquina y para examinar los programas del ROM de tu computadora.
Desplazamiento. Un número usado en una instrucción, salto o ramificación, que le indica a la computadora cuántas ubicaciones debe saltar para encontrar la siguiente instrucción.
Dirección. Un número usado para identificar una ubicación en la memoria de la computadora.
Dirección absoluta. La dirección de un dato.
Direccionamiento a la página cero. Usado sólo en el 6502, es una modalidad de direccionamiento en la que el operando es una dirección de la página cero de la memoria (es decir, entre 0-255).
Direccionamiento absoluto. Una modalidad de direccionamiento en la que la isntrucción contiene la dirección del dato. También denominada direccionamiento indirecto.
Direccionamiento directo. Ver direccionamiento absoluto.
Direccionamiento implícito. Una modalidad de direccionamiento en la que se sobreentiende el operando y no es necesario especificarlo.
Direccionamiento indexado. Una modalidad de direccionamiento en la que el contenido de un registro índice se suma a la dirección dada en la instrucción para calcular la dirección real del dato.
Direccionamiento indirecto. Una modalidad de direccionamiento en la que el operando se usa como indicador (painter) de los datos. El operando puede ser una dirección o en el caso del Z80 un par de registros, y contiene las direcciones de los datos.
Direccionamiento inmediato. Modalidad de direccionamiento en la que el dato para una instrucción va incluido en la instrucción.
Direccionamiento relativo. Una modalidad de direccionamiento en la que la computadora calcula la dirección de la siguiente instrucción sumando un número denominado desplazamiento a la dirección del contador de programa.
Ensamblador. Un programa que convierte las instrucciones escritas en nemónicos en los códigos propios de la computadora.
Grupo de instrucciones. Todas las operaciones que pueden realizarse con un determinado microprocesador.
Hexadecimal. Un sistema numérico que usa

16 dígitos (los números 0 al 9 y las letras A-F). Cada dígito de un número tiene 16 veces el valor del dígito de su derecha.

HIMEM. La dirección más alta del RAM del usuario.

Indicador (pointes). Una ubicación de memoria (o par de registros) que contiene la dirección de un dato.

Indicador de pila. Un registro del CPU que contiene la dirección del último dato de la pila.

Instrucción. Una operación para ser llevada a cabo por la unidad central de procesado.

Intérprete. Un programa que traduce las instrucciones en BASIC (o cualquier otro lenguaje de alto nivel) al código propio de la computadora.

Introducir. Poner algo en la pila.

Lenguaje ensamblador. Una forma de programar la computadora usando códigos de letras llamadas nemónicos que representan las instrucciones en código de máquina.

LIFO. Iniciador de «lost in /first out» (último en entrar, primera en salir). Es el método usado por la computadora para almacenar información en la pila.

Memoria de pantalla. Las ubicaciones del RAM que se usan para contener la información que va a ser visualizada en la pantalla.

Microprocesador. El chip que contiene el CPU de la computadora y que lleva a cabo las instrucciones del programa y controla todas las demás actividades dentro de la computadora.

Modalidades de direccionamiento. Las distintas maneras en que puedes decirle a la computadora dónde encontrar los datos para trabajar en un programa en código de máquina.

Nemónico. Un código de letras usado en el lenguaje ensamblador para representar una instrucción en el código de computadora. La palabra nemónico significa «para ayudar a la memoria» y en el lenguaje ensamblador suena como la instrucción que representa.

Offset. Ver desplazamiento.

Operador. La parte de la instrucción que le dice a la computadora dónde encontrar los datos con los que trabajar.

Operando. La parte de la instrucción que le dice a la computadora lo que debe hacer.

Página. Una subdivisión de la memoria. En casi todas las computadoras, una página son 256 ubicaciones.

Página cero. Las primeras 256 ubicaciones de la memoria.

Pila. Una zona de la memoria usada por la computadora para almacenar temporalmente, y donde el último dato que se introduce es el primero en salir.

Pop. Remover un dato almacenado en la pila.

RAM del usuario. La parte del RAM donde se almacenan los programas en BASIC.

Ramificación. Una instrucción que le dice a la computadora que salte a otra línea del programa.

Ramificación condicional. Una instrucción que le dice a la computadora que salte a otra línea del programa dependiente del resultado de una comprobación.

RAMTOP. La dirección más alta del RAM del usuario.

Registro del status del procesador. Este es el nombre que tiene el registro de banderas del 6502 (el registro donde cada bit se usa para grabar una determinada condición dentro de la computadora).

Registros. Los lugares del CPU donde los bytes de instrucciones, datos y dirección están contenidos mientras la computadora trabaja con ellos.

Registros índice. Los registros usados en el direccionamiento indexado y también en el 6502 como registros de uso general.

Sacar. También denominado pop, consiste en remover un dato de la pila.

Salto. Una instrucción que le dice a la computadora que vaya a otra línea del programa.

Signo #. Este es el signo usado en algunas computadoras para señalar los números hexadecimales. En el microprocesador 6502 se usa para indicar los datos.

Signo &. Este es otro signo usado para señalar números hexadecimales.

Sistema de variables. Ubicaciones de memoria del RAM que contienen información sobre el estado en cada momento de la computadora.

Sistema operativo. Un grupo de programas escritos en código de máquina y almacenados en el ROM de la computadora, que le dicen cómo llevar a cabo las funciones que realiza.

Tope de la memoria. La dirección más alta del RAM del usuario.

Unidad lógico-aritmética (ALU). La zona dentro del CPU, donde se realizan las operaciones aritméticas y lógicas.

Respuesta a los problemas. Continuación

Página 28

00011010 es 26 en decimal.
11111011 es 251 en decimal.
10101010 es 170 en decimal.

	Decimal		Hexadecimal	
	Alto orden	Bajo orden	Alto orden	Bajo orden
307	1	51	&01	&33
21214	82	222	&82	&DE
759	2	247	&02	&F7
1023	3	255	&03	&FF

Página 31

Para adaptar el programa de la página 27 para soluciones mayores que 255 tendrás que borrar la instrucción retorno y añadir las siguientes líneas. Para ver el resultado usa esta instrucción: PRINT PEEK (dirección 3)+PEEK (dirección 4)×256.

Z80		6502		Significado
Nemónicos	**Códigos hexadecimal**	**Nemónicos**	**Códigos hexadecimales**	
LD A, &00	3E, 00	LDA #&00	A9 00	Pone 0 en el acumulador.
ADC A, &00	CE, 00	ADC #&00	69 00	Suma con acarreo 0 al acumulador.
LD (dirección 4),A	32, dirección 4	STA dirección 4	8D dirección 4	Almacena el contenido del acumulador en la dirección 4.
RET	C9	RTS	60	Retorno.

Página 37

El hexadecimal para el complemento a dos de 12 es &F4; 18 es &EE y 9 es &F7.

Indice

Cambios para el cargador hexadecimal

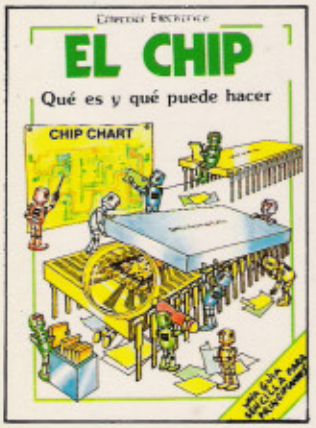
Cambia estos pasos para el ZXS1 (Timex 1000).

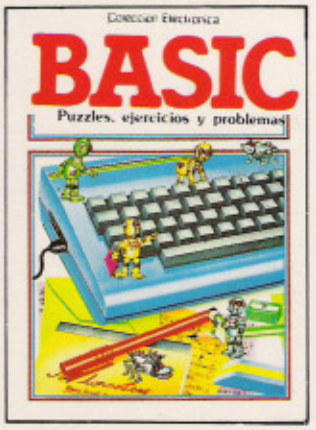
```
 40 INPUT H$
 70 LET X=
    (CODE (H$)-28)*16
 80 Delete
790 LET Y=CODE
    (H$(2 TO))-28
100 LET X=X+Y
110 Delete
155 Delete
160 Delete
```

Cambia estos pasos para el Atari.

```
 90 LET Y=ASC(A$(2))
```

Colección Electrónica





distribuidor exclusivo
cesma sa
C/ Aguacate, 25
28044 MADRID