LIESERT

PEEKS Y POKES

CPC 464/6128

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

LIESERT

PEEKS Y POKES

CPC 464/6128

UN LIBRO DATA BECKER

EDITADO POR FERRE MORET, S.A.

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

Depósito legal B-42.502/85

ISBN 84-86437-34-2

Este libro ha sido traducido por Dn. Joachim Hommen, ingeniero electrónico y experto conocedor del CPC 464/6128.

Advertencia importante

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

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

INDICE

PROLOGO

¿Se han preguntado alguna vez de que manera funciona su CPC 464? ¿Han tenido en algun momento la curiosidad de saber que es exactamente un sistema operativo y que funciones realiza? Para estas preguntas un manual no puede dar amplias respuestas (aunque estuviera escrito extensamente). Muchos de los trucos y manipulaciones que pueden facilitar la labor del programador, han sido posibles únicamente mediante una ojeada tras las bambalinas. Por ello el presente libro debe explicar como funciona su ordenador, que ocurre en su interior si pulsa la tecla RETURN, etc.

El título PEEKS and POKES puede dar la impresión de que en este libro solo se trata de estos 2 comandos. Afortunadamente no es así (esto resultaría demasiado aburrido incluso para mí). La elección de este título se basa en el de su antecedente denominado "PEEKS and POKES para el Commodore 64" y se quisieron mantener el concepto general así como la forma de presentación. Tampoco el Commodore 64 se programa únicamente con PEEKS y POKES. Finalmente los 2 comandos mencionados tienen la cualidad de posibilitar el fácil acceso al sistema operativo y al lenguaje máquina, objetivos ambos de nuestro libro.

Se ofrecen asimismo todos los trucos correspondientes, incluso en forma doble. Después de la amplia presentación de un consejo, al final del capítulo encontrarán un resumen, que les permita en un repaso futuro evitar la extensa lectura de todas las explicaciones.

Otro objetivo de nuestro libro es la ampliación de los espartanos comandos gráficos del CPC, de forma fácil y comprensible. No es necesario ser un especialista informático para comprender perfectamente esta lectura. Si todavía les parece el lenguaje

máquina intrincado y laborioso, encontrarán aquí un curso de iniciación que les permitirá comenzar a desentrañar los aparentes misterios de este forma de programación, pudiendo a continuación optar por aprender Ensamblador, Pascal o cualquier otro lenguaje.

Sólo me queda desearles que disfruten de la lectura de este libro y de las pruebas correspondientes.

Hans Joachim Lisart
Muenster, noviembre 1984

1. ¿COMO FUNCIONA UN ORDENADOR ?

En los siguientes apartados conocerán el CPC y su funcionamiento. Todos aquellos que se encuentren suficientemente formados en la técnica de ordenadores pueden evitar la lectura de estas primeras páginas. Los "supercracks" me perdonarán eventuales simplificaciones que se han efectuado en aras de hacer esta lectura mas comprensible.

1.1 LOS ORDENADORES TAMBIEN TIENEN BUSES

En principio algo fundamental. Cada microprocesador puede direccionar un área de memoria determinada, es decir puede hacer referencia a una cierta cantidad de células de memoria o bytes. La capacidad de direccionamiento depende de la cantidad de líneas de direcciones que posea el procesador. Cada línea de dirección representa un bit (espero que recuerde el significado del bit, de su manual del CPC) y puede presentar 2 estados: 0 y 1.
El microprocesador Z80, que representa el "cerebro" de su CPC tiene 16 líneas de dirección, que en su conjunto componen lo que se denomina "Bus de dirección". De esta forma puede direccionar 2^16 = 65.536 = 64 K de células de memoria.
Es posible que haya notado, que el CPC posee 64 K RAM y 32 K ROM = 96 Ks disponiendo de esta forma de mayor número de direcciones de memoria que posibilidades de utilización. De que manera es que puede funcionar, les explicaré mas adelante.
Además del bus de dirección existen 2 buses adicionales en su Z80, por un lado el bus de control que es la denominación del conjunto de todas las líneas de control, que entre otras cosas conmutan por ejemplo la memoria de entrada a salida.
Existe además el bus de datos, mucho más importante. Este posee 8 líneas y su tarea es transportar datos (para ser mas exactos 8 bits = 1 byte) en el interior del ordenador.
El bus de datos, así como el bus de direcciones estan conectados

con la totalidad de los restantes componentes del ordenador a los que hace referencia el microprocesador. Es decir con RAM, ROM y otros chips.
Toda vez que el microprocesador ejecuta un comando refiriéndose a datos en el exterior del Z80, envía en primer lugar la dirección de memoria (que puede asimilarse con el número de teléfono) al bus de direcciones. El chip de memoria reconoce el byte al que se hace referencia. A continuación el microprocesador envía los datos al bus de datos, donde los recibe la memoria o esta envía el contenido de una célula de memoria a través del bus al Z80. Así de fácil es el trabajo del ordenador.

Todo esto es válido para todos los procesadores de 8 bits (8 bits por el ancho del bus de datos). A partir de ahora nos ocuparemos también de las características especiales de su CPC.

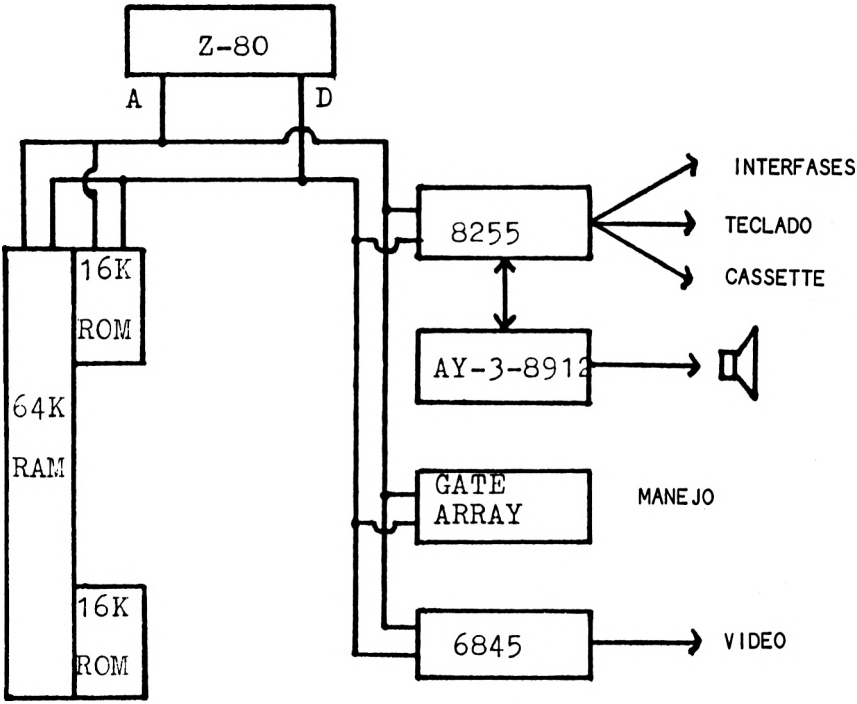
1.2 EL HARDWARE DEL CPC

No teman, tampoco aquí profundizaremos demasiado en la técnica. Para la mejor comprensión de los siguientes capítulos, es muy útil saber algo sobre el interior del CPC.
Al final de el presente apartado encontrarán un esquema de bloque de su ordenador de forma muy simplificada (dib. 1). Como pueden comprobar en el dibujo, el ROM y el RAM estan posicionados parcialmente lado a lado, es decir ocupan las mismas direcciones. En el momento de acceder a un byte no se pueden obtener 2 valores diferentes (sólo hay un bus de datos). En alguna parte se debe definir a que se refiere: RAM o ROM. Según las necesidades del procesador se "desconecta" una o la otra parte del bus de datos. Esto funciona porque el procesador utiliza las diferentes áreas también para diferentes tareas. Siempre que se necesitan determinados datos el Z80 conmuta previamente con tiempo.

Desde el BASIC solamente puede hacerse referencia al RAM a través de PEEK y POKE. Con un pequeño truco se puede llegar asimismo al ROM - de que manera lo explicaremos mas adelante.

Una parte del RAM ocupa la memoria de la pantalla, de donde el chip denominado 6845 obtiene sus informaciones. Este integrado se ocupa de convertir los datos del video-RAM en una señal video para el monitor.

Aparte del 6845 hay algunos chips más. El AY-3-8912 (no teman no es necesario memorizar estas combinaciones de memoria), es responsable del sonido, es decir que convierte los comandos SOUND en tonos audibles.
El 8912 por razones técnicas no esta directamente conectado al Z80, sino que obtiene sus datos del componente interfase 8255 (vean dib. 1).
Este circuito integrado transmite los datos del procesador a los dispositivos periféricos como cassettes, interfases y teclado (que para el Z80 representa también un dispositivo externo, sólo casualmente se encuentra en la misma caja).
Si el Z80 por ejemplo, quiere obtener información del teclado, se lo comunica al chip interfase. Este se "fija" si hay una tecla pulsada y cual es. Esto se comunica al Z80 a través del bus de datos. Si se quiere enviar un byte a través de un interfase, el procesador lo transmite al 8255. Aquí se mantiene hasta que el dispositivo de referencia está disponible para recibirlo.
Otro chip maneja procedimientos especiales internos. Este chip no es de relevancia para nosotros.



DIB. 1 LA ARQUITECTURA DEL CPC 464

1.3 DISTRIBUCION DE MEMORIA

Como ya saben del último capítulo el CPC tiene 64 K de RAM, de las cuales sólo están disponibles 42,5 K para la programación BASIC. Se preguntará con razón donde quedan las 21,5 K restantes.

A primera vista parece posible reservar una memoria mayor para el BASIC. Sin embargo el ordenador necesita también posiciones de memoria para funciones internas. En primer lugar se puede mencionar el Video-RAM que tiene la función de almacenar el contenido de la pantalla. Siempre que se quiere activar o desactivar un punto en la pantalla el procesador altera el valor correspondiente en el Video-RAM. El chip responsable de la generación de señales de video comprueba a intervalos regulares los puntos que deben activarse. De esta manera el procesador no debe ocuparse más de lo necesario con la generación de la imagen. El Video-RAM ocupa 16 Kilobytes (que por esta razón no pueden utilizarse para el BASIC) y se encuentra en el área de 49.152 hasta 65.536, es decir ocupa las mismas direcciones que el Interpretador BASIC.
Si restamos las 16 K de las 64 K totales del RAM, nos quedan 48 K y en nuestro cálculo faltan aún aproximadamente 5,5 K.

Cada procesador necesita una cierta cantidad de bytes en la memoria, llamado Stack o 'lote' donde poder almacenar temporalmente sus datos "personales". Así por ejemplo el Z80 tiene que almacenar la dirección actual al llamar una subrutina, para poder volver a la misma al final de la rutina. Esto se realiza con la ayuda del Stack (vean también capítulo 1.4). El Stack, se encuentra en el área de 48.896 hasta 49.151 y ocupa exactamente 256 bytes. Nunca debería intentar acceder a ésta área, mediante el POKE. El resultado podría ser una caída total del sistema, de la cual probablemente sólo podrá salir desactivando el ordenador.

Para poder trabajar tanto con el RAM como con el ROM, las posiciones de memoria 0 - 63 del RAM representan una copia de los ROM bytes paralelos. Si el Z80 está trabajando en el RAM y necesita datos del ROM puede conmutar entre ambas áreas mediante las rutinas en estos 64 bytes. Además encontrará más rutinas en el RAM entre memoria BASIC y Video-RAM. Nunca deberían intentar modificar ninguno de los valores mediante POKE. En la mayoría de los casos el ordenador se 'cuelga', lo que significa que no reacciona ya a ninguna pulsación de tecla, etc., no quedándoles otra salida que apagar el ordenador.

Todavía no basta con esto. El sistema operativo y el interpretador, necesitan una cierta cantidad de posiciones de memoria, por ejemplo para almacenar resultados intermedios de operaciones aritméticas, intercambiar datos entre ambos programas ROM, almacenar temporalmente las entradas del teclado, etc.

En este sentido el buffer de teclado es muy interesante, facilitando la entrada de caracteres antes que el BASIC pueda aceptarlos. Esto pueden comprobarlo ustedes mismos. Entre el siguiente "Programita" y arránquelo:

```
1 FOR I = 1 TO 10000: NEXT I
```

Si el programa se ejecuta entre una secuencia cualquiera de comandos. Hasta que no se provoca un BREAK no se visualiza ninguna letra en la pantalla. En el momento que el bucle finaliza después de 10000 vueltas, éstos caracteres aparecen en su pantalla. Mientras que el programa se ejecutaba el sistema operativo ha almacenado hasta 20 caracteres. De ésta manera ya puede preparar la entrada para el siguiente comando INPUT si se trabaja con cálculos muy extensos. Un BREAK de un programa BASIC en cambio borra completamente el buffer del teclado.

Por otro lado se almacenarán asimismo los caracteres al ejecutar comandos largos (por ejemplo LOAD etc., con ciertas restricciones).

Si quiere profundizar un poco en la arquitectura de memoria de su CPC debe consultar el anexo de este libro. Allí encontrará un plano de ocupación de memoria con indicación de direcciones de las diversas áreas.

1.4 POINTER Y STACKS

Estas dos denominaciones las encontrará frecuentemente.
Pointer (del inglés apuntador), apunta a posiciones determinadas de la memoria y puede denominarse asimismo vector. En las posiciones señaladas por este apuntador, pueden encontrarse informaciones determinadas o subrutinas. El apuntador de cursor, por ejemplo, señala la posición en la memoria de pantalla donde se encuentra en este momento el cursor, e indica las células de memoria en las que se guarda el carácter siguiente.
Los apuntadores a subrutinas se han creado para hacer más flexible el procedimiento de programas en lenguaje máquina. Un programa puede seleccionar el vector correcto mediante una tabla de apuntadores de subrutinas, según sus necesidades, en el caso PRINT#8 se selecciona por ejemplo el apuntador octavo para el manejo de la impresora (aunque el comando PRINT posiblemente no trabaja de esta manera, es un ejemplo que nos parece bastante gráfico).

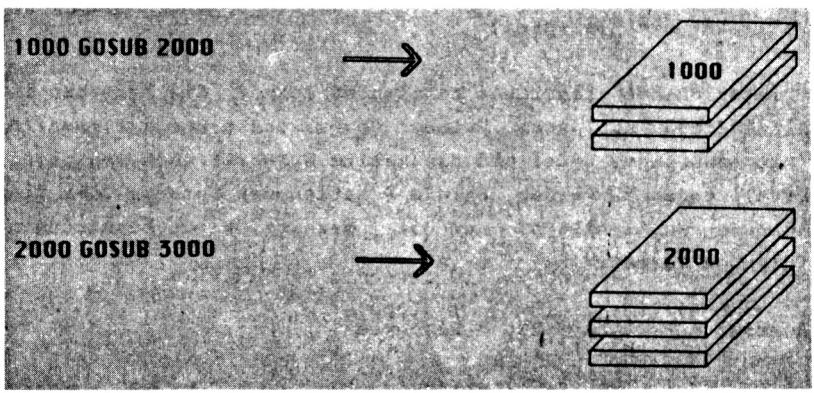
Los Pointer tienen siempre un formato determinado. Se componen en general de 2 bytes, el primero de los cuales es el LOWBYTE (byte de menor valor) y el segundo el HIGHBYTE (byte de mayor valor). Para obtener la posición del cursor o la dirección a la que se apunta se utiliza la siguiente fórmula:

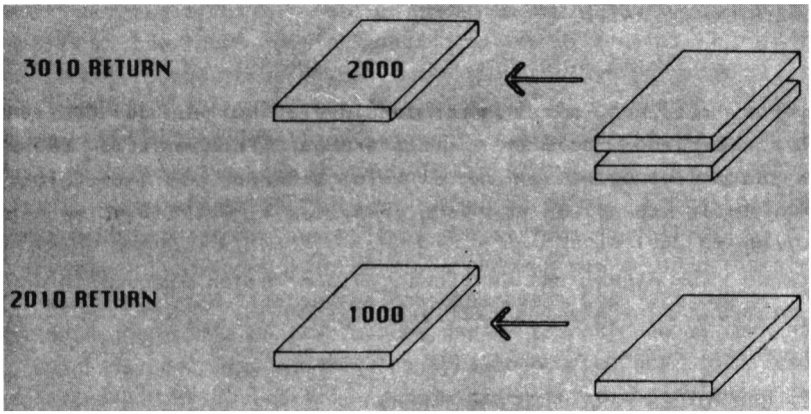
Dirección = Lowbyte + 256 * Highbyte

Si trabaja en sistema Hexadecimal posicione simplemente el Lowbyte

a la derecha y el Highbyte a la izquierda obteniendo de esta manera la dirección hexadecimal de 4 dígitos.
Este posicionamiento de Lowbyte delante del Highbyte en la memoria, es una de las características comunes a los ordenadores.
Los apuntadores de un byte señalan dentro de un área de memoria determinada, a la posición actual (0-255) y se suman a la dirección base.

Un Stack (del inglés lote), tiene la misión de almacenar datos temporalmente y devolverlos en el momento oportuno, en el orden inverso al que fueran almacenados. Como en una pila auténtica siempre puede cogerse únicamente el elemento superior (último añadido) y agregar datos nuevos de la misma forma. Esto se utiliza en especial para subrutinas. Al llamar la subrutina, la posición actual del programa se almacena temporalmente en el Stack y en el RETURN, el ordenador extrae esta dirección para continuar la ejecución del programa.
Una vez almacenados los datos en el Stack, es imprescindible que al extraerlos podamos asegurar que cada subrutina bifurca a la dirección correcta antes de hacer referencia al siguiente elemento de la pila.
Un ejemplo:





El procedimiento es el mismo tanto en programas en lenguaje máquina como en BASIC. De la misma manera las direcciones de retorno en bucles del tipo FOR-NEXT, se almacenan en el Stack. Vemos ahora la razón por la cual subrutinas y bucles no deben superponerse. Si el retorno de la subrutina 3000 se produjera antes que el retorno de 2000, el ordenador obtendría del Stack direcciones de retorno falsas (2000 está en la posición superior de la pila).

Resumen : Apuntador
Dirección = Lowbyte + 256 * Highbyte
Lowbyte = Dirección menos INT(Dirección / 256) * 256
Highbyte = INT(Dirección / 255)
Los apuntadores normalmente se componen de 2 bytes, en el orden LOW/HIGH

2. SISTEMA OPERATIVO E INTERPRETE

Antes de ocuparnos con mayor profundidad de las posibilidades del CPC, deberíamos aclarar algunas expresiones especiales. No es vergonzozo desconocer que hace un sistema operativo. ¿Qué cree Ud? Exactamente de este y otros temas nos ocuparemos en los siguientes capítulos.

2.1 UTILIDADES PARA PEQUEÑAS TAREAS

Cada ordenador lo necesita, conocen muchos nombres para él, pero sólo pocos principiantes saben lo que se esconde detrás - nos referimos al sistema operativo.
Como ya lo dice el nombre, el sistema operativo es imprescindible para el funcionamiento de su ordenador. Para por ejemplo , enviar un byte del procesador a la impresora, cada vez se necesitan pequeños programas máquina. La transmisión, desde luego, se produce de forma automática, pero el interface debe programarse y controlarse, para que puedan descubrirse rápida y seguramente errores etc.
En general el sistema operativo maneja los recursos entre ordenador y periferia (el teclado, impresora, etc). Para cada dispositivo existen subrutinas específicas, que pueden ser utilizadas por el interpretador de BASIC y sus programas. Para enviar caracteres a los periféricos el BASIC debe preparar los datos y bifurcar a continuación a la rutina correspondiente.

Seguramente ya habrán oido algo sobre otros sistemas operativos como CP/M o MS/DOS (para mencionar los 2 mas conocidos) asi como de "sistemas operativos standard".
Estos sistemas estan concebidos de manera que ya no bifurcan a subrutinas, sino que transmiten números de comandos y datos a

través de registros especiales. De esta manera el sistema operativo, que debe ser alterado para los diferentes tipos de ordenadores, puede trabajar con los mismos códigos en todos ellos. Debe decirse que el lenguaje máquina tiene un desagradable efecto secundario. Con frecuencia al adaptar rutinas, se alteran al mismo tiempo las direcciones de entrada de las subrutinas subsiguientes, dado que raramente se mantiene el número de bytes. Esto significa que en cada ordenador, las rutinas del sistema operativo, tienen diferentes direcciones y todos los programas que acceden a estas rutinas deben modificarse en consecuencia. Con el código de comando, se amplía en cambio con nuevas operaciones el conjunto de comandos del lenguaje máquina (y esto ocurre para todos los ordenadores). Esto implica que los productores de software no necesitan volver a crear sus programas para cada nuevo ordenador.

2.2 MUY POTENTE EN LENGUAJES: INTERPRETADOR

El nombre inglés de este programa en lenguaje máquina contenido en el ROM, ya dice mucho sobre su tarea. El interpretador (significado castellano de "Interpreter") convierte comandos BASIC en acciones del procesador. El Z80 en principio sólo puede tratar su lenguaje máquina individual (diferente para cada tipo de microprocesador). Los comandos BASIC sólo significan para el Z80 secuencias de caracteres más o menos concatenadas. Si a éstos caracteres se agrega un ENTER, el interpretador BASIC comienza su primera fase de traducción.
En esta fase el procesador se ocupa de reconocer las letras y cifras como números de línea, comandos BASIC y datos. Los comandos se convierten en un número especial de código, por el cual el interpretador puede seleccionar la subrutina de tratamiento correspondiente.

Hasta este punto solamente se ha codificado la secuencia de caracteres, que no se encuentra aún en la memoria del programa. Para ello es posible que se daba proporcionar espacio (si la línea ha sido agregada posteriormente). A continuación se almacena finalmente.
Todos los procedimientos explicados hasta el momento se ejecutan en un tiempo casi imperceptible para el programador, entre la pulsación de la tecla ENTER y la aparición del cursor en la siguiente línea.

La segunda parte de la traducción se inicia con el comando RUN. El interpretador extrae entonces de la memoria los comandos BASIC y los datos uno a uno ejecutándolos seguidamente. Si encuentra por ejemplo un comando PRINT, arranca la subrutina de lenguaje máquina PRINT, que lee las variables induciendo al sistema operativo a visualizar caracteres, etc.
Dado que los comandos contienen todos los códigos especiales no es necesario que después del RUN se reconozca la secuencia de caracteres P R I N T, sino solamente que se busque el apuntador de subrutina correspondiente. Con esta manera de trabajar, se ahorra mucho tiempo.

2.3 INTERRUPCIONES CON SISTEMA

Lo que entre los humanos se mira como falta de educación, en los ordenadores es de buen tono. Exactamente la interrupción facilita muchas aplicaciones extraordinarias que nunca se podrían realizar sin este detalle.
Tanto el interpretador BASIC como el sistema operativo son programas de lenguaje máquina. Ambos se arrancan en el momento de encender el ordenador y se ejecutan hasta el momento en que se llama un nuevo programa en lenguaje máquina. Si esto se realiza mediante un comando CALL, el ordenador retorna al BASIC una vez finalizada la rutina.

Como ya saben de la programación BASIC, un ordenador (excepto los sitemas de multiprocesador), siempre puede ejecutar únicamente un programa a la vez. Interpretador y sistema operativo, son 2 programas diferentes, que solamente han de trabajar simultaneamente para realizar determinadas tareas. ¿De que manera se resuelve este problema?
La posibilidad más simple de poder ejecutar 2 programas casi al mismo tiempo, es la llamada interacción de programas. Siempre que el BASIC finaliza una parte de su trabajo, el mismo activa el sistema operativo y viceversa. Esto ocurre por ejemplo cuando se quiere acceder a un dispositivo periférico. El BASIC unicamente proporciona la información, que el sistema operativo debe enviar al dispositivo. Este hecho incluye por ejemplo al teclado, que se leerá unicamente si el sistema operativo esta en funcionamiento en este momento. No obstante, durante la ejecución del programa, por lo menos la tecla ESC debería tener una acción especial (BREAK). Para resolver este problema los productores de ordenadores crearon el INTERRUPT (del Inglés "Interrupción"). Cada 1/50 de segundo el procesador interrumpe el programa máquina que se ejecuta (BASIC, sistema operativo o rutina propia) bifurcando a las subrutinas de comprobación de teclado etc. Si el ordenador "descubre" la pulsación de la tecla ESC, se interrumpe el proceso que se ejecuta en ese momento. En el caso que se hubiera pulsado otra tecla, esto se almacena en el buffer del teclado.

Al usuario puede parecerle que el teclado se comprueba continuamente, porque incluso el más rápido de los operadores, no puede introducir más de 15 caracteres por segundo. En cambio para el microprocesador el tiempo entre 2 interrupciones parece muy largo, dado que trabaja con un tiempo de reloj de aproximadamente 4.000.000 de pulsaciones por segundo y un comando máquina necesita para su ejecución un promedio de 8 a 10 de estas pulsaciones. De manera que el procesador puede ejecutar miles de instrucciones antes de que una interrupción le perturbe su trabajo. Después de comprobar el teclado, el procesador continúa su proceso en la posición en la que fuera interrumpido.

Durante diversos comandos, se desactiva el interrupt, como por ejemplo parcialmente en operaciones de cassette/diskette. Esto se puede asimismo reconocer al utilizar colores alternados que en este momento dejan de parpadear. También la alternación de colores se maneja a través de interrupciones, produciéndose un cambio de color en cada interrupción.

También en el BASIC pueden utilizarse interrupciones, porque con los comandos AFTER y EVERY pueden confeccionarse rutinas de interrupción propias, como subrutinas BASIC. Este procedimiento funciona en principio igual que en lenguaje máquina.
Si el interpretador encuentra un AFTER o EVERY se inicializa un Timer que después de un determinado tiempo provoca una interrupción como la alarma de un despertador. El interpretador suspende su trabajo y ejecuta la rutina INTERRUPT.

Una señal de interrupción puede generarse asimismo por otros componentes del ordenador, para indicar por ejemplo que se deben recibir datos de un interface. El BASIC reacciona unicamente ante señales TIMER.

Un tipo especial de interrupción representa el RESET, que provoca que el Z80 arranque el programa a partir de la célula de memoria 0 de la misma manera que al encender el ordenador, sin tener en cuenta lo que haya ocurrido hasta este momento. A partir de célula 0, no comienza otra cosa que la rutina de inicialización, que

borra la memoria y que realiza otros importantes ajustes básicos. Idénticos resultados se obtienen al pulsar SHIFT-CTRL-ESC y mediante un CALL 0 desde el BASIC - ¡pero cuidado! los datos importantes deben ser grabados previamente al cassette/diskette.

2.4 NO SOLO PARA ESPECIALISTAS: COMANDOS ESPECIALES

Imagínese la siguiente situación: en una de las innumerables revistas de ordenadores encuentra un superprograma para copiar. Acaba de teclear el listado de 20 K, pero la prueba del programa acaba con un ERROR.
Tiene que comprender el procedimiento del programa para eliminar errores si no quiere comparar cada letra del listado con el original. ¡Si no fuera por estos tontos comandos POKE, que ningún programador normal utiliza! Es ya hora de levantar el obstáculo que representan instrucciones de este tipo.

2.4.1 PEEK & POKE

Veamos en primer lugar el comando POKE. Su sintaxis debería resultarle familiar: POKE dirección, byte. La dirección puede estar entre 0 y 65.535, el byte entre 0 y 255. La función de este comando es almacenar el byte bajo la dirección indicada. Esto puede servirnos para muchas cosas, dependiendo de la dirección se puede llenar la pantalla, almacenar un comando máquina y mucho más. Con este comando podemos provocar desastres en el ordenador, porque tanto el sistema operativo como el interpretador tienen que almacenar determinados datos en determinadas posiciones. Mediante el comando PEEK podemos ver estos datos. La sintaxis debería resultarles asimismo conocida: PRINT PEEK (dirección), nos visualiza el byte almacenado bajo la dirección indicada.

Es importante saber que PEEK es una función y por ello sólo puede utilizarse dentro de una asignación (A=PEEK...) o bien en otra expresión.
Ambos comandos tienen en común el hecho de que su utilidad se dirige según la dirección a la que queremos acceder. Por ello se recomienda al utilizar estos comandos, observar el plano de ocupación de memoria para saber en qué área se está trabajando.
En la mayoría de los casos es posible de esta manera ver la función.

2.4.2 CALL

Este comando en principio es interesante unicamente para el programador de lenguaje máquina: CALL dirección. Se utiliza para la llamada de programas en lenguaje máquina.
En el comando CALL la dirección indica el byte con el que debe comenzar la ejecución del programa. Después de finalizar la rutina máquina, el interpretador retorna al BASIC como desde una subrutina.
Se pueden asimismo agregar datos al comando CALL, que serán tratados por el programa en lenguaje máquina.

2.4.3 UNA PEQUEÑA INCURSION POR LA ARITMETICA BINARIA

A los comandos explicados se añaden algunos más que seguramente ya conocen, pero cuyas múltiples aplicaciones posiblemente no les hayan sido aclaradas. En primer lugar debemos mencionar el AND, OR, XOR y NOT. Hasta el momento solamente han utilizado construcciones IF-THEN, por ejemplo de la siguiente forma:
IF A=0 AND B=0 THEN 100
Normalmente estos comandos han sido pensados para la interrelación lógica de variables y números. Para ello debe recordarse que el

ordenador trata también las comparaciones como números. Prueben los siguientes comandos:

```
PRINT (1=2)
PRINT (1=1)
```

Una comparación con el resultado "verdadero" nos proporciona -1, un resultado "falso" 0. En el sistema binario un -1 se representa así: 1111 1111. Si el bit del extremo izquierdo no se interpreta como signo, la misma combinación nos da un resultado de 255. ¿Pero que relación tiene esto con los comandos BASIC?

Una construcción IF-THEN se abandona siempre cuando el resultado del término es 0. Sería posible asimismo la siguiente secuencia de comandos:

```
IF 3 * A THEN 110
```

Los resultados de las comparaciones individuales se interrelacionan y el resultado final determina la acción siguiente del programa. Para comprender el procedimiento de las interrelaciones hacemos una pequeña incursión en la aritmética binaria.

AND, OR, XOR y NOT son operadores relacionales booleanos, que se utilizan para la interrelación de estados lógicos. Como saben estos estados lógicos se representan fácilmente con bits (0 para "falso" y 1 para "verdadero").

Cada vez se interrelacionan 2 bits. El resultado lo pueden comprobar en las tablas siguientes.

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

Como pueden ver el resultado en todos los casos es 1 si ambos bits de entrada tienen el valor 1. Las 2 funciones pueden expresarse en palabras. En el caso de AND, el resultado es 1 si bit 1 Y bit 2 son 1, en el caso de OR si bit 1 O bit 2 son 1, el resultado es también 1.

En el OR exclusivo (XOR) el resultado es siempre 1 cuando uno o el otro operando contienen 1, pero nunca si ambos contienen 1.
La función NOT invierte simplemente el bit de entrada.

NOT	0	1
	1	0

Lamentablemente para nosotros los programadores de BASIC, queda aún un problema. En BASIC no podemos utilizar los bits individuales, porque trabajamos con números duales. Para calcular que resultado nos proporciona la expresión 45 AND 123 hemos de proceder de la siguiente manera:

1. Convertir los números al sistema dual. Ya saben como realizarlo, del manual CPC.

Los números 45 y 123 en sistema binario son los siguientes:
45 = 00101101 123 = 01111011

2. Interrelacionar los números duales bit por bit.
En nuestro ejemplo 45 AND 123 nos proporciona el resultado:

```
   00101101
AND 01111011
------------
   00101001
```

3. Convertir el resultado al sistema decimal.
00101001 = 41

Naturalmente lo pueden hacer más sencillo entrando PRINT 45 AND 123. Pero de la manera anterior se puede ver la relación lógica. Preguntarán ahora, con razón, para que nos sirve. Aparte de la interrelación de comparaciones estos comandos muchas veces se utilizan para modificar determinados bits. Mediante una relación AND con 254 se desactivará el bit del extremo derecho, mediante una interrelación OR con 1 se puede activar nuevamente este bit. Inténtelo con números de su elección.

2.4.4 CONEXION AL EXTERIOR: COMANDO PORT

A través de los conectores posteriores de su CPC, puede comunicar su ordenador con dispositivos periféricos. Para que esto sea posible asimismo desde el BASIC, existen comandos especiales.
INP (Port) extrae un byte del PORT (puerto) indicado entre paréntesis. El número de puerto no se refiere a direcciones de memoria, existe para ello un sistema de direccionamiento propio, comprensible únicamente con conocimientos detallados de lenguaje máquina. No obstante explicaré los comandos correspondientes simplemente para que conozcan su funcionamiento.
Con OUT Port, Byte se puede enviar un byte al puerto. Como pueden observar ambos comandos tienen una cierta similitud con PEEK y POKE.
Nos queda un comando misterioso: WAIT Port,X,Y.
Este comando tiene la misión de obligar al ordenador a ESPERAR, y esto no le gusta nada. Esta espera se consigue mediante la interrelación consecutiva de bytes. Si el interpretador encuentra un comando WAIT, lee en primer lugar un byte del puerto indicado. El mismo se relaciona con un operador XOR con el número Y contenido en el comando. El resultado de esta operación, se relaciona mediante el operador AND con el valor X contenido en el comando. Si el resultado final es 0, el interpretador repite el procedimiento de lo contrario contrario continúa con el comando siguiente.
Existe otra variante del comando WAIT, no utilizando el argumento Y. En este caso el interpretador espera hasta que el byte del puerto indicado relacionado AND con X proporcione un resultado diferente de 0.

2.5 COMANDOS QUE NO SE ENCUENTRAN EN EL MANUAL

Lo que normalmente se conoce únicamente de otros fabricantes de ordenadores también ha ocurrido en el CPC. Un programador ha confeccionado un interpretador, lo ha depurado de errores y después de todo este trabajo el autor del manual se olvida de describir alguno de los comandos.

El comando se llama MOD. El nombre proviene de "Módulo", una expresión seguramente conocida por ustedes de las matemáticas. Una función módulo nos proporciona el resto de una división. Ejemplos:

```
10 / 4 = 2.5 ó  2 resto 2
10 MOD 4 = 2
11 MOD 4 = 3 (11/4 = 2 resto 3)
```

Puede utilizarse el comando MOD en lugar del signo de división "/" y de esta manera se obtiene el resto de la división. MOD funciona únicamente con números y variables del tipo INTEGER (-32768 hasta 32767).

Con el comando MOD se puede programar fácilmente el algoritmo de Euclides (vean capítulo 14). El comando opuesto al MOD es la INTEGER-DIVISION, que se representa por la barra invertida y nos proporciona únicamente resultados enteros, 11 dividido 4 = 2. También la INTEGER-DIVISION se ha pensado únicamente para valores enteros.

En las presentaciones del CPC se ha promocionado la posibilidad de conectar ROM adicionales cuyos programas (por ejemplo juegos, lenguajes de programación, businessoftware) pueden arrancarse con un comando BASIC. Este comando no se encuentra en el manual y sin embargo existe. A diferencia de los comandos restantes, este se compone de un único carácter que se forma al teclear el a de arrobas al unísono de la tecla "SHIFT".

En la pantalla se visualiza una raya vertical (la cual puede utilizarse asimismo para gráficos). Esta línea indica al ordenador que se quiere referenciar otra área de memoria. Pero no sabe aún cual ROM. Para ello los ROM contienen un llamado LABEL, un tipo especial de nombre, por el cual el sistema operativo lo reconoce. Dado que en su CPC sin ampliaciones tiene únicamente el BASIC-ROM, el único nombre que debe utilizarse es la palabra 'BASIC'. Compruébelo y el BASIC arranca nuevamente con el mensaje 'BASIC 1.0'. Todos los datos se borran entonces.
Un área ROM puede contener varios LABELS, que trabajan como comandos adicionales. Del mismo modo el floppy controller contiene una ampliación de set de comandos.

El BASIC del CPC está extraordinariamente autodocumentado, tiene una función que indica la dirección a partir de la cual se almacena una variable. Para un programador BASIC esto no es de mucha importancia, pero podrá utilizar esta función en el momento que quiere cambiar a lenguaje máquina.
La función se ejecuta con el a de arrobas, PRINT &a proporciona la dirección de la variable a (si la misma existe, de lo contrario se visualiza 'IMPROPER ARGUMENT').

Otro comando particular es la función DEC$ (x,y). Se menciona únicamente 1 vez, como familiar del comando BIN$. En el ROM existe realmente esta función, como hemos podido comprobar al listar la tabla de palabras de comando. Sin embargo parece que no existe ninguna subrutina para este comando en el interpretador, porque como único resultado se visualiza 'SYNTAX ERROR'. Es posible que el programador del interpretador tuviera previsto utilizar esta función como opuesta a BIN$ y HEX$, pero posteriormente encontrara mas conveniente la utilización de prefijos & y &X.

3. LA MEMORIA

El CPC en cuanto a capacidad de memoria pertenece al rango de ordenador profesional dentro de los domésticos. Además de sus 64 K de RAM, tiene 32 K ROM disponibles. Mediante accesorios especiales el área de memoria se puede ampliar enormemente. Como funciona y que puede hacerse con todo esto, deben comprenderlo en los siguientes capítulos.

3.1 PROTEGER MEMORIA

Para el BASIC tenemos - como he explicado anteriormente - aproximadamente 42,5 Kilobytes de memoria libre disponible. Los programadores en lenguaje máquina no pueden hacer uso de esta ventaja porque al interpretador de BASIC no le gustan los programas en lenguaje máquina, ya que hace uso libre de la memoria. Si se genera una nueva línea BASIC o una variable, el interpretador sobrescribe el área libre (para sí) después del programa con los datos nuevos. De esta manera, las posiciones de memoria no se llenan de arriba hacia abajo. Al principio de la memoria BASIC se encuentran las líneas de programa, los strings se posicionan al final del área reservada. Por en medio se mueven las restantes variables.
En algún momento estas áreas 'collsionan' en la mitad de la memoria; entonces el CPC visualiza un 'MEMORY FULL ERROR'. Esto significa que nunca puede decirse con exactitud cuales de los bytes de la memoria están libres y se quedan libres durante el procedimiento de un programa BASIC.

Por ello hay que reservar un número de bytes determinado para programas en lenguaje máquina. Esto se realiza mediante el comando MEMORY, cuyo funcionamiento es verdaderamente simple. El final de la memoria BASIC está señalado por un apuntador por el cual se orienta el interpretador. El comando MEMORY no hace otra cosa que

modificar este apuntador según nuestra necesidad. Si se posiciona en una dirección mas baja el BASIC llega antes al final de memoria.

Este apuntador puede encontrarlo en las posiciones de memoria &AE7B y &AE7C. Puede alterarse también mediante comandos POKE, en lugar de MEMORY.

El comando '? HIMEN ' tiene la función inversa, indicandonos la actual posición del apuntador. Una vez encendido el ordenador, está posicionado en 43.903. Este número marca el final del área de memoria BASIC, que comienza en byte 368.
El comando MEMORY se ejecuta únicamente utilizando valores entre 368 y 43.903, de lo contrario se visualiza un 'MEMORY FULL ERROR'. Esto evita la sobrescritura de datos del sistema operativo. Además el MEMORY evita la destrucción no deseada de programas BASIC. En caso de que el apuntador se posicione demasiado bajo, con riesgo de destruir un programa ya confeccionado, se visualiza también el 'MEMORY FULL ERROR'. Este bloqueo se puede evitar (poco razonable) posicionando el apuntador mediante POKE.

En caso de que se quieran reservar 256 bytes para un programa en lenguaje máquina, solo debe introducirse MEMORY 43.647, pudiendo grabarse entonces el programa en lenguaje máquina a partir de la célula de memoria 43.648 (MEMORY y HIMEN se refieren al último byte de la memoria BASIC).

Resumen: Proteger memoria

Con el comando MEMORY se puede modificar el final del área de memoria BASIC. La memoria BASIC se mueve entre las direcciones 368 y 43.903.
Con HIMEN puede leerse el apuntador actual al final del BASIC.

3.2 ¿COMO FUNCIONA EL BANKSWITCHING?

Como ya saben de los capítulos anteriores, en algunas áreas de la memoria el RAM y el ROM ocupan idénticas direcciones. El procesador sólo puede trabajar con una de las 2 partes de memoria, lo cual significa que debe poseer la capacidad de conmutar entre RAM y ROM. En un circuito lógico especial se analiza la dirección de memoria que llega al bus de direcciones y a continuación se accede al componente de memoria correspondiente.
Este circuito puede influirse desde el exterior, lo que quiere decir que el procesador puede determinar mediante comandos de entrada y salida (similares a INP y OUT) como se decodifican las direcciones. En algún momento se accede únicamente a los componentes RAM mientras que en los otros casos el RAM se desconecta del bus de direcciones y se accede al ROM. Además las 2 áreas ROM pueden conmutarse separadamente. De esta manera es posible que solamente este activo el sistema operativo mientras que se accede al Video-ram. El circuito lógico funciona entonces como un desvío.

También podemos conmutar desde el BASIC al ROM; el resultado sin embargo en la mayoría de los casos es que el ordenador queda 'colgado'. Inténtenlo. Con OUT &7F82,&82 se pueden activar las 2 áreas ROM.

Como habrá notado, la conmutación de las áreas de memoria desde el BASIC no es demasiado aconsejable. Este hecho no es extraño porque con el bankswitching el Z80 se vuelve loco por el cambio de memoria. En muchos casos el interpretador trabaja en el RAM y si conmuta al ROM, el procesador no puede encontrar su programa principal. La consecuencia inmediata es la caída brusca del sistema.

3.3 LECTURA DEL ROM

En ocasiones puede ser útil e incluso necesario leer datos desde el ROM, por ejemplo el set de caracteres (que a propósito se encuentra en el área &3800 hasta &3FFF). Desde el BASIC esto no es posible. La función PEEK solamente se refiere al RAM y la conmutación de RAM a ROM termina casi siempre con la pérdida del programa confeccionado. Teóricamente sería posible escribir un programa BASIC para la lectura del ROM pero el CPC reacciona de una forma muy alérgica. De todas formas listaré este programa porque,

...

a. aclara el principio y

b. ya no hace falta explicar el programa en lenguaje máquina correspondiente dado que es casi una traducción exacta.

El programa BASIC es el siguiente:

```
1 OUT &7F82,&82
2 A=PEEK(X): REM X=dirección deseada
```

El equivalente en lenguaje máquina es el siguiente:

```
1 DATA &01,&82,&7F,&ED,&49
2 DATA &1A,&32,&7F,&AB,&C9
3 MEMORY &AB6F: FOR I=&AB70 TO &AB79
4 READ A: POKE I,A: NEXT: END
5 REM X=dirección deseada
6 CALL &AB70,X
7 A=PEEK(&AB7F): RETURN
```

Para la utilización de la rutina máquina debe decirse que las líneas 1 a 4 inicializan el programam, lo que significa que aquí se cargan en la memoria los comandos máquina (mediante el bucle POKE). Esta parte se ejecuta unicamente una vez al principio y a continuación el programa máquina se encuentra en la memoria y puede ejecutarse. Solamente los valores en las líneas DATA representan los comandos (10 bytes).

Si quiere ahora leer un byte desde el ROM tiene que indicar a la rutina máquina la dirección del mismo. La dirección se almacena en la variable X y a continuación mediante GOSUB 6 se arranca la segunda parte del programa.
Línea 6 recibe la llamada mediante CALL &AB70,X. Debido a que programas en lenguaje máquina no pueden pasar datos al BASIC de forma directa, nos ayudamos almacenando el valor buscado del ROM en una posición especial de la memoria, de donde lo podemos extraer mediante PEEK (&AB7F). Esto se realiza en línea 7.
Naturalmente pueden alterarse según necesidades propias los nombres de variables y los números de línea.

3.4 AMPLIACIONES DE MEMORIA

Muchas veces se lee en las promociones del CPC, que la memoria del equipo puede ampliarse mediante circuitos adicionales, que podrían ser por ejemplo ROMS con programas almacenados como tratamiento de texto u otros pero es posible asimismo conectar espacio de memoria RAM adicional.

Es fundamental que también estas tarjetas de memoria se conecten al bus de direcciones mediante el circuito lógico. Por ello es posible acceder a estas áreas de memoria como el ROM incorporado mediante el bankswitching. Para el Z80 esto no establece diferencia porque solamente tiene que modificar los números correspondientes en el comando OUT.
El dispositivo floppy incorpora asimismo un ROM de ampliación, en el cual se encuentran almacenados los comandos para su manejo.

4. TRUCOS PARA LA PANTALLA

Por las posibilidades gráficas del CPC podemos asimismo contar este equipo entre los profesionales. Los comandos BASIC incorporados ya permiten una amplia utilización, pero existen aún posibilidades adicionales que vamos a comentar en los siguientes apartados.

4.1 MANEJO DE PANTALLA MEDIANTE COMANDOS CHR$

Ya habrán visto la tabla de caracteres especiales en el capítulo 9 del manual. Todos estos caracteres pueden utilizarse como comandos adicionales para el manejo de la pantalla a través de los cuales es posible el acceso directo a diversas subrutinas del sistema operativo. El interpretador BASIC no incluye subrutinas propias para tales funciones, simplemente envía el carácter especial correspondiente al sistema operativo.

Abajo se han listado unicamente caracteres especiales que pueden utilizarse realmente con sentido. ¿Para que les sirve un segundo comando LOCATE que además es más complicado de manejar que la instrucción BASIC?
Todos los caracteres especiales tienen en común, el que pueden llamarse mediante PRINT CHR$(X). El comando PRINT se ocupa de que el carácter llegue a la rutina correcta del sistema operativo, la función CHR$ solamente nos ayuda a indicar el código como número, ya que de lo contrario nos veríamos obligados a utilizar un símbolo gráfico, que no funciona en todos los casos, porque el sistema operativo trata los caracteres gráficos y los caracteres de control de diferente manera.

Con el carácter especial Nro. 1 (SOH) puede visualizar otros caracteres de control. Como ya saben caracteres cuyos códigos ASCII sean menores que 32 no pueden visualizarse, se envían al sistema operativo como caracteres de control. Si al código se

agrega un CHR$ (1), se visualiza un gráfico; por ejemplo para LF (Line feed o avance de línea) una flecha hacia abajo. Intente por una vez PRINT CHR$(1) CHR$(10).
SOH solamente se refiere al código siguiente. Para cada carácter a visualizar tiene que agregar un nuevo SOH.

También el código de control número 2 tiene un efecto muy interesante. Con STX (así se llama) se puede desconectar el cursor, lo cual funciona unicamente durante la ejecución de un programa. Si el BASIC trabaja en modalidad directa después de cada visualización de READY, conecta el cursor nuevamente. Con el siguiente ejemplo funciona:

```
10 PRINT CHR$(2)
20 INPUT "TEXTO";A$
30 PRINT CHR$(3)
```

Como pueden observar en el comando INPUT ya no aparece el cursor como de costumbre.
El comando PRINT en línea 30 provoca exactamente lo contrario; con él se conecta nuevamente el cursor (lo que nos podríamos ahorrar en el ejemplo dado que después del final del programa se conecta automáticamente).

El carácter especial número 7 ya lo conocen del manual, por ello no insistimos con él (produce el beep).

Veamos ahora los caracteres que mueven el cursor en la pantalla. CHR$(8) actúa como la tecla con la flecha hacia la izquierda, CHR$(9) mueve el cursor una posición a la derecha, CHR$(10) mueve hacia abajo y CHR$(11) hacia arriba. Esto puede resultar muy útil en caso de que tengan que visualizar valores indexados o potencias. Con los códigos descritos pueden mover el cursor a otra línea, imprimir en esa posición la potencia o el índice y de la misma manera volver a la línea antigua. A simple vista el CHR$(12) nos parece que sobra. Con él puede borrarse la pantalla igual que con el comando CLS. Se nos presentan nuevas posibilidades si

Incorporamos uno o varios caracteres de control en strings BASIC corrientes. Pruebe lo siguiente:

```
A$=CHR$(12)+"ENCABEZADO"
```

Cada vez que visualice A$ (simplemente con PRINT) en primer lugar se borra la pantalla y a continuación se visualiza el texto. Generalmente pueden incorporarse en los strings todos los caracteres de control de la manera arriba indicada.

Otro carácter de control para el cursor es el CR (CHR$(13)). Con él se puede posicionar nuevamente el cursor al principio de la línea. CR significa 'Carriage return' que en castellano no quiere decir otra cosa que 'retorno del carro'. Esta denominación procede aún de la época en que los ordenadores se manejaban a través de telex como terminal. Con CR el ordenador podía mover el cabezal del telex al borde izquierdo de la línea.
La función CHR$(16) la conocen también. Con ella pueden simular la tecla CLR desde el BASIC y borrar el carácter debajo del cursor. Inténtelo:

```
10 CLS
20 LOCATE 20,10
30 PRINT "ALGUN-TEXTO";
40 WHILE INKEY$="": WEND
50 PRINT CHR$(8) CHR$(8) CHR$(8) CHR$(8) CHAR$(8) CHR$(16)
```

No hace falta explicar el programa, arránquelo, pulse cualquier tecla y verá el resultado.
A simple vista no parece tener sentido, pero queda a su elección hacer uso fructífero de las nuevas posibilidades.

Para borrar líneas enteras de la pantalla se han previsto los caracteres especiales 17 a 20. Reemplacen en el ejemplo el último carácter de control de línea 50 por un CHR$(17).

Si arranca a continuación el programa se borrarán todos los caracteres desde el comienzo de la línea hasta la posición del cursor. Si reemplazan el CHR$(17) por CHR$(18) funcionará exactamente a la inversa, se borrarán los caracteres desde la posición del cursor hasta el principio de la línea.
Los códigos de control 19 y 20 funcionan de forma similar, únicamente que el carácter de control 19 borra todos los caracteres desde el comienzo de la pantalla hasta la posición del cursor. CHR$(20) borra todo a partir del cursor hasta el final de la pantalla (esquina inferior derecha). Puede probar agregando al programa ejemplo unos cuantos comandos PRINT para llenar más líneas.

Ahora dejamos el manejo del cursor y nos dedicamos al siguiente carácter de control, que es un bombón especial. Con PRINT CHR$(21) pueden desconectar la pantalla de texto lo que no quiere decir otra cosa que se suprimirán todas las salidas de pantalla que no procedan de comandos gráficos. Esto puede utilizarse para entradas de passwords como en el ejemplo siguiente:

```
10 PRINT "ENTRAR PASSWORD POR FAVOR!" CHR$(21)
20 INPUT A$
30 IF A$="PASSWORD" THEN GOTO 1000 ELSE NEW
1000 PRINT CHR$(6)
.......
```

Después de la ejecución de la línea 10 se suprime la visualización de texto, lo que incluye asimismo los caracteres que entren a través del teclado (vean línea 20), de modo que nadie puede leer su password sin autorización. El comando INPUT funciona normalmente. Línea 1000 devuelve su pantalla a la modalidad de texto.

También el carácter SYN (código 22) tiene un efecto interesante. Si se continúa con un '1', se activa la modalidad transparente. En esta modalidad, la posición de pantalla en la cual se quiera visualizar un carácter, no se borra previamente como de costumbre. El carácter anterior permanece y el nuevo carácter se superpone.

Esto puede utilizarse por ejemplo para el subrayado en la pantalla (lo que en raros casos es posible en los ordenadores), moviendo el cursor después de la impresión del texto a la posición inicial, y activando la modalidad transparente mediante "PRINT CHR$(22) CHR$(1);" imprimiendo a continuación caracteres de subrayado (se logra mediante SHIFT y 8). PRINT CHR$(22) CHR$(0) conmuta a modalidad normal.

Muy útil resulta el carácter de control 24. Este simplemente invierte el color PAPER y PEN. El resultado es la representación inversa del carácter, lo que significa que las letras que se visualizan en amarillo con fondo azul (o claro sobre oscuro), se visualizan ahora en azul sobre fondo amarillo (oscuro sobre fondo claro), lo que nos puede parecer mas usual (¿o escriben ustedes con lápiz blanco en papel negro?).

El último carácter de control, sirve una vez más para el manejo del cursor. Con PRINT CHR$(30), el cursor se posiciona en la esquina superior izquierda de la pantalla. Esta función se denomina asimismo 'Home'.

Existe un pequeño inconveniente en la utilización de los caracteres de control, si el comando PRINT se transfiere al cursor de gráficos mediante TAG se visualizarán únicamente caracteres de gráficos, las funciones de los códigos quedan desactivadas hasta el TAGOFF.

Resumen: Códigos CHR$

Código	Función
1	Impresión de caracteres de control
2/3	Cursor de texto activado/desactivado
7	Beep
8	Cursor un carácter hacia atrás
9	Cursor un carácter hacia delante
10	Cursor una línea hacia abajo
11	Cursor una línea hacia arriba
12	Borrar pantalla

13	Cursor en comienzo línea
16	Idem tecla CLR
17	Borrar línea hasta cursor
18	Borrar línea a partir del cursor
19	Borrar pantalla hasta cursor
20	Borrar pantalla a partir del cursor
21/6	Pantalla texto desactiva/activa
22	Modalidad transparente activada/desactivada
24	Inversión
30	Home

4.2 VIDEO-RAM INTERNO

¿Qué sabe hasta el momento del Video-RAM? Supongo que no demasiado. Este tema trataremos en las líneas siguientes. En los capítulos anteriores ya han leído que los bytes en el Video-Ram representan los puntos de la pantalla. La primera pregunta es la siguiente: "¿Cómo se ordenan los puntos en la memoria?" Para poder responder haremos un pequeño experimento. En primer lugar desconectamos el CPC y volvemos a arrancarlo, para que todos los datos internos se reinicialicen. A continuación entramos:

```
MODE 2
FOR I=49152 TO 65535: POKE I,255: NEXT
```

Con esta secuencia de comandos se inicializan con 1 todos los bits del video-ram y de este modo activamos la totalidad de puntos de la pantalla. Después de pulsar la tecla RETURN verán como se activan en primer lugar los puntos superiores de todas las líneas de texto y a continuación todos los puntos por debajo, etc, hasta que la pantalla queda totalmente llena.
Imaginen ahora todas las 25 líneas de texto unidas en una única línea, obtendrán un total de 8 líneas de puntos (porque cada carácter en modalidad 2 se compone de 8 X 8 puntos). Si dividimos

los 16 Kilobytes (este es el tamaño de la memoria de pantalla) por 8, resultan 2 K ó 2048 bytes para cada línea. Agrupando ahora las 8 líneas de puntos, tendremos la organización interna de la pantalla en la memoria. Aún con esto no queda aclarado de que manera se representa un punto individual.
Si - como en modalidad 2 - sólo se debe diferenciar entre 2 colores (punto activado o desactivado), nos basta con un bit para cada punto (con valor 0 para desactivado y 1 para activado). Para 640 X 200 puntos serán necesarios exactamente 128.000 bits o 16.000 bytes. 16 Kilobytes son 16384 bytes, es decir 384 bytes más de los que necesita la pantalla. Para ello hay 2 explicaciones. Por un lado para el procesador y para el chip TV es más sencillo calcular con Kilobytes enteros (normalmente también preferimos utilizar en nuestros cálculos números enteros). Además el procesador necesita bytes de reserva para algunas operaciones como por ejemplo el scrolling horizontal (lo aclararemos más adelante). Por ello al final de cada bloque de 2 K, hay 48 bytes sin utilizar.

En modalidad 1 hay 4 colores disponibles, y para ello se necesitan 2 bits por punto, cuyas combinaciones posibles son 00, 01, 10, 11. Debido a que en este caso se necesitan el doble de bits podemos representar únicamente la mitad de los puntos, que ahora tienen el doble de ancho. De esta manera para los 320 X 200 puntos se necesitan asimismo 16000 bytes.

En modalidad 0 hay 16 colores disponibles. Para poder diferenciar los 16 tipos de puntos correspondientes necesitamos medio byte para cada punto. Dado que se ha duplicado nuevamente la cantidad necesaria de bits, hay que dividir por 2 la cantidad de puntos y nos quedan solamente 160 X 200 puntos.

Todavía no queda claro el misterio del Video-ram. En modalidad 2 está bien claro que bit representa cada punto. El primer byte del Video-ram determina el aspecto de los primeros 8 puntos. Con POKE 49152,X puede almacenar cualquier configuración de bits que se representa igual en la pantalla (mientras el CPC permanezca en

modalidad 2). Inténtelo con los valores 1, 2, 4, 8, 16, 32, 64 128, 240, 15, 170 e imagínese cada vez la representación binaria de estos números. Podrán comprobar que los puntos corresponden exactamente al número binario.

En modalidad 1 cada punto necesita 2 bits. Sería conveniente que cada 2 bits correlativos uno indicara el valor y el otro el color activado, pero lamentablemente no está pensado de esta manera. El chip que produce la señal TV, en primer lugar diferencia la parte izquierda y derecha del byte del Video-ram (medio byte se llama asimismo NIBBLE). Escribimos ahora el nibble derecho debajo del nibble izquierdo como en nuestro ejemplo (solo exponemos los números de bit, ustedes deberían anotar los valores de cada uno):

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

Cada par de bits encolumnados indica el color para un punto. Con el byte 1111 0000 los 4 puntos tendrían el color 01 (se lee desde abajo hacia arriba). Con el valor 11111010 tendríamos el color 3 (binario 11) para el primer y tercer punto y el segundo y el cuarto punto tendrían el color 1. Compruébelo con comandos POKE.

En modalidad 0 se procede de forma similar. Escriba esta vez los bits como en el ejemplo siguiente:

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

Como ya saben un byte en esta modalidad representa 2 puntos. Lean nuevamente las combinaciones de bit desde abajo hacia arriba y obtendrán el número de color del punto. De su manual pueden extraer la tabla de colores (en el caso de que no hayan alterado esta con comandos INK).

4.3 GRAFICOS OCULTOS

En la programación de gráficos, textos etc. se quiere en muchos casos confeccionar en primer lugar la imagen completa de la pantalla, antes de visualizarla, o se desean visualizar 2 gráficos independientemente. Para ello existen 2 posibilidades. Si las representaciones no se superponen basta con visualizar una de ambas en color oscuro (con INK 2,0) y cuando esta ya esté completa conmutar el color a claro (por ejemplo con INK 2,24). Al mismo tiempo la otra imagen se debería oscurecer de igual modo.

Este procedimiento funciona únicamente si las 2 imágenes no se superponen y si el CPC se encuentra en otra modalidad que 2 (en modalidad 2 no se puede utilizar un color de reserva para conmutar).

Puede resultar más sencillo indicar al CPC la utilización de una segunda memoria de pantalla. Este procedimiento en cambio cuesta mucha memoria BASIC. La memoria de pantalla logicamente solo puede desplazarse a áreas de memoria que no sean utilizadas por el sistema operativo. Además el Video-ram se puede desplazar únicamente en intervalos de 16 Ks, es decir en las direcciones iniciales 0, 16384, 32768 y 49152. La última posibilidad se ha elegido ya al conectar el ordenador. En las áreas 0-16383 y 32768 hasta 49151 tanto el sistema operativo como el interpretador tienen almacenados datos de importancia. Nos queda por esto únicamente el área a partir de 16384.
Lamentablemente esto significa que tenemos que restringir la memoria BASIC a menos de 16 Ks. mediante MEMORY 16383. Por esta razón este método sólo sirve para programas relativamente pequeños.

Con CALL &BC06,&40 se puede desplazar la memoria de pantalla a 16384, CALL &BC06,&C0 nos devuelve al estado inicial.

Todos los comandos de gráficos y otros comandos de salida se refieren siempre únicamente a la memoria de la pantalla que está

activa en este momento, lo que significa que puede trabajar normalmente. Esta circunstancia puede modificarse. En algunos casos puede ser necesario trabajar en la imagen todavía oculta mientras que la otra queda visible. Felizmente existe una célula de memoria en el RAM en la cual el sistema operativo se almacena la dirección de pantalla. Si quiere visualizarse un texto o un punto de gráficos, el ordenador no se orienta por la dirección real sino por el valor de la célula de memoria mencionada. De esta manera podemos engañar al sistema operativo con una memoria de pantalla no existente en realidad. La célula de memoria mencionada tiene la dirección &B1CB. Mientras que en ella se encuentra el mismo valor que se agrega al comando CALL, la visualización de pantalla funciona normalmente.
CALL &BC06,&40: POKE &B1CB,&C0 activa el Video-ram a partir de la dirección 16384, pero las salidas se producen aún en la memoria antigua.
El comando POKE puede utilizarse naturalmente sólo, es importante que solamente se POKEAN &40 ó &C0 porque de lo contrario se 'cuelga' el ordenador.

Han de tener en cuenta una particularidad del CPC. Si se quiere realizar un scroll de la pantalla, no se desplaza el contenido del video-ram byte por byte en la memoria como puede parecer, sino que para el chip TV cambia el inicio del video-RAM, que comienza por ejemplo en 53248 (para escoger un valor cualquiera) en lugar de en 49152 como es habitual. Mediante una lógica de direcciones los bytes que "se caen por detrás" se agregan nuevamente en 49152, lo que a primera vista no cambia nada para nosotros. Si POKEAMOS la pantalla sin embargo, verán que los puntos deseados no aparecen en el extremo superior izquierdo sino en alguna otra parte de la pantalla. Para el ordenador (y asimismo para nosotros) este método tiene la ventaja de que el scrolling se realiza más rápido.
Debido a esta técnica particular es conveniente dar un comando MODE al inicio del programa BASIC en cada memoria de pantalla. De esta forma se reinicializan los apuntadores. No es suficiente el borrado con CLS o CLG. Recomendamos la no utilización de la técnica de scroll.

Resumen: Desplazar pantalla

CALL &BC06,&40 conmuta área 16384 hasta 32767 como nuevo Video-RAM (proteger antes con MEMORY).
CALL &BC06,&C0 conmuta a área inicial.
En la célula de memoria &B1CB el sistema operativo almacena, que Video-RAM está activado.

4.4 ALMACENAR PANTALLAS

Como saben el BASIC del CPC tiene un comando especial para grabar diversas áreas de memoria. Con él puede grabarse asimismo la memoria de pantalla en cassette/diskette y cargarla nuevamente desde allí. Los comandos son:

```
SAVE "!TV",B,49152,16384
LOAD "!TV",49152
```

Puede modificar naturalmente el nombre del fichero a su necesidad pero nunca deberían olvidar el signo de exclamación, porque de lo contrario los mensajes del sistema operativo estropean el contenido de la pantalla.

Asimismo es posible almacenar únicamente áreas parciales. Para ello debe calcularse para cada línea de puntos la dirección inicial y final de la memoria. Con los conocimientos del capítulo 4.2 no deberían tener problemas en realizarlo. Cada línea se grabará individualmente con un comando SAVE propio en al cassette/diskette y puede ser nuevamente cargada con LOAD. Esto sólo funciona si no se ha utilizado el scrolling (el problema lo conocen del capítulo 4.3).

4.5 SCROLLING

Todo aquel que posee un ordenador se ha sentido atraído por el fenómeno 'Scrolling'. Si el cursor llega al borde inferior de la pantalla todo el contenido de la pantalla se mueve en sentido ascendente para obtener espacio para una nueva línea. Hay 2 cosas que resultan interesantes para el programador de BASIC, por un lado está la forma en que se genera el sroliing y por el otro el hecho de que el scrolling no se limita únicamente a la dirección ascendente. Con esta técnica pueden lograrse efectos interesantes.

Análogamente al caso usual el texto de la pantalla puede moverse asimismo hacia abajo, si en la línea superior de la pantalla se envía el carácter de control 'cursor hacia arriba' (CHR$(11)). A continuación puede llenar la línea obtenida de esta forma, mediante el comando PRINT. En dirección vertical puede también moverse el contenido completo de la pantalla de un solo golpe. Para ello son necesarios 2 comandos OUT, con los cuales se modifica el offset para el comienzo del Video-RAM. Como ya han leído en capítulos anteriores un scrolling no provoca una alteración del contenido de la memoria de pantalla, sino únicamente el desplazamiento de la dirección inicial en la longitud de una línea de pantalla. Este desplazamiento se denomina 'offset'. Afortunadamente este offset se puede modificar en intervalos más pequeños que una línea completa. La modificación mínima es de 2 bytes, que según la modalidad de pantalla corresponde a medio carácter, un carácter o dos caracteres. De esta manera puede también realizarse un scrolling vertical.

El controlador de video 6845 posee diversos registros en los cuales se almacena este offset. Mediante comandos OUT es posible grabar en estos registros pero la lectura de los mismos no es posible a través de INP. Los comandos son los siguientes:

```
OUT &BC00,13: OUT &BD00,offset
```

En el caso normal (si no se ha efectuado aún ningún scrolling), el

offset tiene el valor 0. A través del incremento de este número se desplaza hacia la izquierda y disminuyendo el desplazamiento es hacia la derecha. Si desde el centro quiere desplazar hacia la izquierda y hacia la derecha se recomienda borrar la pantalla antes del primer comando PRINT desplazando todo hacia el centro mediante OUT &BC00,13: OUT &BD00,20.

Puede desplazarse una línea entera mediante un bucle como el siguiente:

```
OUT &BC00,13: FOR I=0 TO 40 : OUT &BD00,I: NEXT
```

El primer comando OUT sólo sirve para la elección del registro correspondiente del 6845, por ello no es necesario repetirlo dentro del bucle. Debido a que el sistema operativo utiliza también registros del controlador de video es posible que el número de registro ya no sea correcto. De esta manera no se evita el OUT &BC00,13 delante de cada scrolling.

Resumen: Scrolling

El scrolling hacia abajo puede realizarse a través de la salida del carácter $11 en la línea superior de pantalla. Para el scrolling vertical se utiliza la secuencia de comandos
OUT &BC00,13: OUT &BD00,offset
Donde offset indica la cantidad de bytes que se quieren desplazar dividido por 2.

4.6 "SCROLLING" ALTERNATIVO

En las siguientes líneas les voy a describir una particularidad del CPC que a poseedores de otros ordenadores les deja maravillados. Además del scrolling normal donde simplemente se modifica el apuntador a la memoria de pantalla, existe otro método

con el cual es posible desplazar la imagen completa (incluído el borde) en las cuatro direcciones.

En este caso también juegan un papel importante los registros del 6845. Ahora no se modifica el apuntador de la memoria de pantalla sino la señal video para el monitor. Esta señal video aparte de la información de la imagen para el monitor contiene asimismo señales de sincronización que indican el final de una línea de la pantalla. Si esta señal se envía con retraso el monitor no puede visualizar la información de imagen en la posición correcta de la pantalla. El resultado es una imagen desplazada.

El 6845 ofrece la posibilidad de determinar directamente el tiempo de las señales de sincronización mediante comandos OUT. Antes de un desplazamiento en dirección horizontal hay que ejecutar un OUT &BC00,2. Con el segundo comando (OUT &BD00,X) se ajusta la sincronización. Normalmente X tiene el valor 46, y se pueden utilizar valores entre 0 y 63, la utilización de valores mayores 'cuelga' el ordenador, debiendo en este caso desconectarlo y encenderlo nuevamente.
Si los valores de sincronización son más pequeños se produce desplazamiento hacia la derecha y si son mayores que 46 se desplaza hacia la izquierda. La secuencia de comandos completa para el desplazamiento en una posición a la izquierda es la siguiente:

```
OUT &BC00,2: OUT &BD00,47
```

Del mismo modo se procede para provocar desplazamiento vertical. Los comandos son:

```
OUT &BC00,7: OUT &BD00,X
```

X puede estar en el ámbito de valores entre 0 y 38, el valor normal es 30. Si X es menor que 30, la imagen se desplaza hacia abajo, si es mayor se desplaza hacia arriba.

Con estos trucos pueden realizarse efectos muy bonitos. Por ejemplo es posible visualizar "puntos negativos" en un juego desplazando la imagen. Naturalmente pueden combinarse ambas posibilidades de desplazamiento.

Resumen: Desplazamiento de imagen

Desplazamiento horizontal: OUT &BC00,2: OUT &BD00,X
Desplazamiento vertical: OUT &BC00,7: OUT &BD00,Y

4.7 NUEVAMENTE: MANEJO DEL CURSOR

El comando INKEY$ frente al comando INPUT tiene una desventaja fundamental, porque el cursor no se visualiza en la pantalla. Afortunadamente podemos activar y desactivar el cursor en cada situación de programa en el entorno de BASIC. Para ello se llaman 2 rutinas del sistema operativo. El siguiente programa ejemplo nos muestra el procedimiento:

```
10 CALL &BB81 : REM Activar cursor
20 WHILE INKEY$="" : WEND: REM Esperar pulsación tecla
30 CALL &BB84 : REM Desactivar cursor
40 ..... Siguiente programa
```

Todo esto no funciona en modalidad directa, porque el sistema operativo toma el control sobre el cursor automáticamente después de cada visualización 'READY'.

Resumen: Cursor Activado/Desactivado

Cursor visible : CALL &BB81
Cursor invisible : CALL &BB84

5. GRAFICOS

Si quiere visualizar algo más que líneas y puntos en su pantalla, en principio el CPC no se lo permite. El Interpretador BASIC como de costumbre no ofrece funciones para circunferencias, rectángulos, etc. Afortunadamente estas funciones pueden reemplazarse mediante simples subrutinas.

5.1 CARACTER DE CONTROL DE GRAFICOS

En el capítulo 4.1 hemos omitido expresamente un carácter de control, que está directamente relacionado con los gráficos de alta resolución. CHR$(23) activa las diferentes modalidades del lápiz gráfico de color. En el caso normal (es decir si no se ha utilizado aún el CHR$(23)), se activan los puntos de los comandos gráficos sin tener en cuenta el estado anterior de los puntos. El CPC incorpora además otras técnicas. Si antes de activar un punto se envía el comando PRINT CHR$(23) CHR$(1), los nuevos puntos se interrelacionan XOR con el valor antiguo. El efecto de la interrelación XOR es que el resultado sólo da 1 si ambos bits de entrada son diferentes. Con otras palabras: XOR activa un punto únicamente si el punto antiguo y el punto nuevo tienen estados diferentes. Si se ha dibujado una línea desde el punto (0,0) hasta el punto (100,100) y se vuelve a realizar por segunda vez en modalidad XOR el resultado es que la línea se borra, porque los puntos de los cuales se compone la línea original y los puntos de los cuales se contruye la segunda línea son iguales. Si para un punto visible tomamos el valor 1 se interrelaciona siempre 1 y 1-XOR obtenemos como resultado un 0, es decir se borran todos los puntos. Si se quiere activar una línea en posiciones donde anteriormente no había nada se activan los puntos, porque el valor inicial y el punto nuevo son diferentes. Para ello un programa ejemplo:

```
10 MODE 2: PRINT CHR$(23) CHR$(1): REM Modalidad XOR
20 FOR I=100 TO 200 STEP 2
30 MOVE 10,I : DRAW 100,I: NEXT: REM Dibujar cuadrado
40 FOR J=1 TO 2
50 WHILE INKEY$="": WEND
60 FOR I=130 TO 180 STEP 2
70 MOVE 60,I: DRAW 150,I: NEXT I,J: REM Segundo cuadrado
```

En primer lugar se habrá preguntado que hace el STEP 2. Este comando es importante porque en dirección vertical se pueden generar 200 puntos diferentes, y sin embargo las coordenadas para ellos se encuentran en el área 0 a 399. Por ello DRAW 100,100 y DRAW 100,101 generan la misma línea. Esto debe evitarse en modalidad XOR, porque 2 líneas superpuestas se borrarán.

El resto del programa supongo que lo pueden comprender ustedes mismos. Se dibujan en total 3 cuadrados uno grande y 2 pequeños donde el último borra su anterior. Veánlo ustedes mismos.

Existen otras 2 modalidades que son AND y OR. Ambas funcionan análogamente que la modalidad XOR, se utiliza únicamente otra interrelación. Con AND se activa solamente un punto si ya había uno anteriormente en la misma posición. Esto puede utilizarse en las modalidades de multicolor para cambiar colores. Se activan puntos en otro color en la misma posición que puntos activos, el color nuevo solamente aparece donde anteriormente había colores diferentes a cero. Esta modalidad se activa mediante PRINT CHR$(23) CHR$(2).

La modalidad OR (PRINT CHR$(23) CHR$(3)) actúa como la modalidad transparente para textos. PUntos nuevos se posicionan simplemente sobre los antiguos. Si un punto nuevo tiene el color cero (negro), el antiguo no se borra.

Resumen: Modalidades de gráficos

```
Normal: CHR$(23) CHR$(0)
XOR:    CHR$(23) CHR$(1)
AND:    CHR$(23) CHR$(2)
OR:     CHR$(23) CHR$(3)
```

5.2 RECTANGULO, CAJA

En los siguiente apartados queremos presentar una serie de subrutinas que pueden utilizarse como comandos gráficos adicionales. Comenzamos con las figuras más simples: caja y rectángulo.

Una caja según nuestra definición debe ser únicamente una superficie demarcada por cuatro líneas en forma rectangular. Hemos de dibujar estas 4 líneas mediante DRAW, y ya tenemos la caja. Esto se realiza mediante la siguiente subrutina:

```
65500 REM Caja: X1, Y1, X2, Y2, F
65501 MOVE X1,Y1: DRAW X1,Y2,F
65502 DRAW X2,Y2,F: DRAW X2,Y1,F
65503 DRAW X1,Y1,F: RETURN
```

Esta y las siguientes subrutinas están confeccionadas de tal manera que pueden agregarse al final de sus programas mediante Merge. Además se utilizan variables para pasar los parámetros necesarios.
Para determinar el aspecto de la caja solamente se debe determinar la esquina superior izquierda y la esquina inferior derecha. Estas coordenadas deben encontrarse en las variables X1 e Y1 para el punto superior izquierdo y en X2,Y2 para el otro punto antes de llamar a la subrutina. Además debe determinar el número de lápiz de color que se desea utilizar.

La primera línea de nuestra subrutina posiciona el cursor de gráficos en el punto inicial de nuestra caja y dibuja la primera

línea. Con los 3 siguientes comandos DRAW se dibujan las líneas restantes. El RETURN provoca el retorno al programa principal.
Una típica llamada de esta rutina es la siguiente:

```
X1=100: Y1=200: X2=200: Y2=100: F=1: GOSUB 65500
```

El siguiente programa debe dibujar un rectángulo, que solamente se diferencia de la caja por el relleno entre las líneas. Para ello dibujamos simplemente muchas líneas subsiguientes hasta que el rectángulo tiene el tamaño deseado. El listado:

```
65505 REM RECTANGULO: X1, Y1, X2, Y2, F
65506 FOR II=Y1 TO Y2 STEP 2*SGN(Y2-Y1)
65507 MOVE X1,II: DRAW X2,II,F
65508 NEXT II: RETURN
```

También en este caso solamente hemos de indicar dos puntos; exactamente como con la caja. En el comando FOR encuentran nuevamente el STEP 2 que es importante para la modalidad XOR.
Además se debe multiplicar el intervalo por -1 si Y2 es menor que Y1. Esto se realiza mediante SGN (Y2-Y1).
En línea 65507 en primer lugar se posiciona nuevamente el cursor de gráficos al borde izquierdo del rectángulo y a continuación se dibuja una línea hacia la derecha. En el siguiente paso del bucle FOR-NEXT se desplaza esta línea en un punto hacia abajo, de manera que poco a poco se llena el rectángulo.

5.3 TODO CON SENOS Y COSENOS

Las dos siguientes rutinas ya las conoce Vd. en forma algo más sencilla del manual CPC. Pero se puede hacer mucho más de los aburridos círculos y discos.

Empezemos primero por las rutinas de círculo. En los dos casos tiene que indicarse el centro (x1,y1) y el radio (r1), así como el color (f). Después de la llamada al subprograma, un bucle FOR-NEXT dibuja el perímetro del círculo (0 - 360 grados) y calcula mediante las funciones seno y coseno la distancia del punto a colocar al centro. Un disco se obtiene si no se emplea PLOTR, sino que se traza una línea del centro al margen.

Si se escoge un radio muy grande, puede suceder que no aparezcan dentro del disco puntos del color deseado, sino que permanezcan oscuros. Esto se puede evitar cuando se inserta en el comando FOR-NEXT un STEP 0.5 (en caso necesario tomar amplitudes de paso aún más pequeñas).
Aquí presentamos las dos rutinas:

```
65510 REM Circulo:x1,y1,r1,f
65511 DEG: r2= r1:FOR ll = 0 TO 359
65512 xx= COS(ll)*r1: yy= SIN(ll)*r2
65513 MOVE x1,y1:PLOTR xx,yy,f
65514 NEXT ll: RETURN

65515 REM Disco: x1,y1,r1,f
65516 DEG: r2= r1: FOR ll = 0 TO 359
65517 xx= COS(ll)*r1: yy= SIN(ll)*r2
65518 MOVE x1,y1: DRAWR xx,yy,f
65519 NEXT ll: RETURN
```

Llamadas típicas son:

```
x1= 320: y1= 200: r1= 100: f= 1: GOSUB 65510
```

```
x1= 100: y1= 100: r1= 30: f= 1: GOSUB 65515
```

¿Sabía Vd. que con sólo algunos pequeños cambios es posible también dibujar ellpses?. Ya se habrá dado cuenta de que las dos variables de radio de la línea 65512, aparecen innecesarias a primera vista. Si R1 y R2 tienen diferentes valores, se obtiene una ellpse. ¡Inténtelo!. R1 es el radio en la dirección X, R2 indica la distancia en la dirección Y.
Para poder tratar todo de forma más sencilla, inserte por favor las siguientes líneas:

```
65520 REM Ellpse: x1,y1,r1,r2,f
65521 DEG: FOR ll = 0 TO 359
65522 GOTO 65512
```

Mediante el "GOTO 65512" de la línea 65522 se utiliza el subprograma para el círculo, ya que es idéntico hasta las primeras dos líneas. De esta forma puede ahorrarse memoria (y trabajo de tecleo).

Con una pequeña modificación del programa de disco podemos producir estrellas con cualquier número de rayos. En el fondo también el disco es una clase de estrella, ya que se traza a cada uno de los 360 puntos del márgen un rayo desde el punto medio (ver figura). Debido a que los rayos están tan juntos unos de otros, aparece ante nuestros ojos como una superficie. Sólo tenemos pues que reducir la cantidad de líneas. Esto lo podemos lograr con un comando STEP; se dibuja por ejemplo sólo cada décimo rayo. Para repartir los rayos uniformemente, se divide el número 360 por su cantidad y se inserta el resultado como STEP.

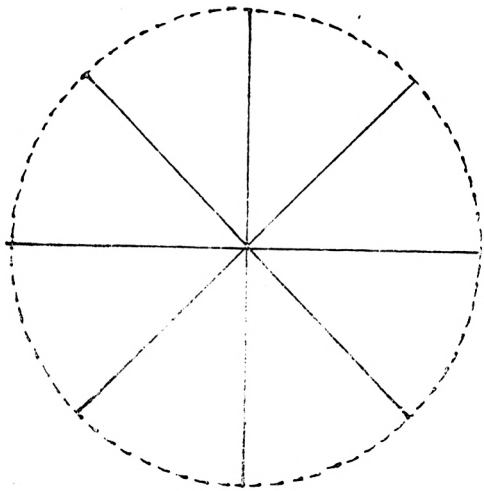
También en la línea 65527 encontrará otra vez un GOTO de ahorro. Para que aparezcan también todos los rayos, tiene que alargarse el bucle FOR-NEXT hasta 360.

También para la estrella tienen que depositarse en las conocidas variables sólo el centro, el radio y el color. Además la variable EC contiene la cantidad de rayos deseada.

A continuación presentamos el listado del programa y un ejemplo de llamada:

```
65525 REM ESTRELLA: X1,Y1,R1,EC,F
65526 DEG: R2=R1: FOR I1=0 TO 360 STEP 360/EC
65527 GOTO 65517

X1=100: Y1=100: R1=30: EC=12: F=1: GOSUB 65525
```



Dib. 2 Estrella

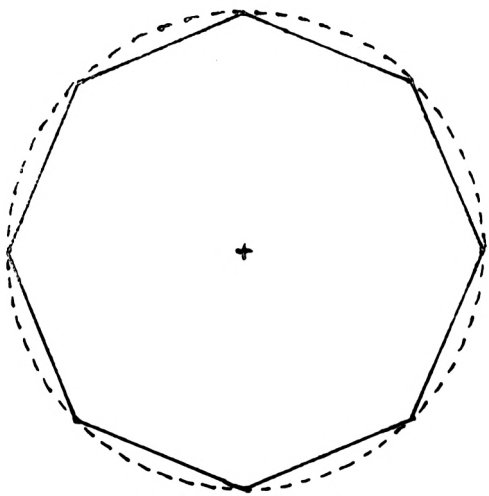
La última subrutina dibuja diversos tipos de polígonos. Para el principio de funcionamiento deberían repasar nuevamente la última subrutina de la estrella. Si se unen todas las puntas de la estrella mediante líneas, resulta un polígono. Esto es lo que hace nuestra rutina. Como en el caso de la estrella, los vértices se calculan partiendo de los 360 puntos de la circunferencia y a

continuación uniendo estos vértices tenemos los lados del polígono. Además el cursor debe posicionarse en el punto inicial, porque de lo contrario se dibujará involuntariamente una línea desde la posición del cursor hasta el punto inicial del polígono. En este caso debe incrementarse el bucle a 370.

```
65530 REM POLIGONO: X1,Y1,R1,EC,F
65531 DEG: MOVE X+R1,Y: FOR II=0 TO 370 STEP 360/EC
65532 XX=COS(II)*R1: YY=SIN(II)*R1
65533 DRAW X1+XX,Y1+YY,F: NEXT II: RETURN
```

Si lo desean pueden dibujarse también estrellas o polígonos ovoidales utilizando un segundo radio. También pueden dibujarse semicircunferencias u otros sectores de circunferencia, por ejemplo variando el bucle FOR-NEXT sólo entre 180 y 360. No hay límites para la fantasía.

Dib. 3 Polígono



5.4 ¿ PARA QUE EL TEST DE PIXEL ?

A simple vista los comandos TEST y TESTR parecen no tener mucho sentido. Para saber únicamente qué color posee un punto, con mirar la pantalla es suficiente. No obstante existen aplicaciones inimaginables sin el comando TEST.

En este sentido recuerdo los SPRITES de Commodore y los SHAPES de los ordenadores Apple. Para los principiantes que no están tan familiarizados con el mundo de los ordenadores, diremos que en ambos casos se trata de figuras que pueden estar predefinidas por el programador, como las letras y que pueden visualizarse en la pantalla con un único comando. Algo parecido podemos programar en BASIC para el CPC.
Nuestro programa debe contener 2 funciones. Por un lado hemos de copiar áreas de la pantalla trasladándolas a otras posiciones.
El principio de la subrutina es bastante simple. Como en las rutinas de la caja y el rectángulo en 5.1, se indican 2 puntos que marcan el área a copiar. A continuación mediante 2 bucles FOR-NEXT se repasan todos los puntos del rectángulo y el resultado de los comandos TEST se utiliza como número de color para el nuevo punto a activar. Si el resultado de TEST es por ejemplo 0, se activará un punto con el color 0, que significa finalmente que aparece un punto oscuro. De esta manera se copia toda el área pixel por pixel.

a segunda rutina debe visualizar una figura en la pantalla que no se ha visto anteriormente, lo que significa que no se copiará sino que se confeccionará nuevamente. Para ello naturalmente en alguna parte se debe definir el aspecto de la figura. Esto se realiza en el programa BASIC mediante líneas DATA. Para cada línea de puntos que debe visualizar nuestra subrutina en la pantalla, tiene que existir un string en el cual se indican los colores de los pixels a activar. La longitud de los strings indica cada vez cuantos puntos se deben activar en una línea. El string contiene un carácter para cada punto, que representa un número hexadecimal indicando el color por el mismo número. La rutina tiene que saber

además cuando finaliza el dibujo de la figura. Para ello se agrega un string con longitud cero y si la subrutina encuentra este string "vacío" termina su ejecución.
Finalmente necesitamos la posición de destino de la figura. Las coordenadas necesarias se transmiten simplemente por medio de variables.

```
999 REM COPIAR AREAS: X1,Y1,X2,Y2,XS,YS
1000 FOR II=Y1 TO Y2 STEP -2: FOR JJ=X1 TO X2
1010 PLOT XS+JJ-X1,YS+II-Y1,TEST(JJ,II)
1020 NEXT JJ,II: RETURN

1049 REM PLOTEAR AREA EN LA PANTALLA: XS,YS
1050 ZZ=0
1060 READ AA$: LL=LEN(AA$): IF LL=0 THEN RETURN
1070 FOR II=0 TO 11-1: AA=VAL("&"+MID$(AA$,II,1))
1080 PLOT XS+II*M,YS-ZZ*2,AA: NEXT II: ZZ=ZZ+1: GOTO 1060
```

La primera subrutina en X1,Y1,X2 y Y2 necesita las coordenadas del área a copiar, XS e YS indican la esquina superior izquierda del área de destino.

En la segunda rutina no es necesario determinar tantos valores. En XS e YS se transmiten las coordenadas de la esquina superior izquierda del área de destino de la pantalla. En el comando PLOT se deberían fijar exactamente en la duplicación de la variable ZZ, que indica el número de la línea que se está dibujando. Si no se procede de esta forma se dibujarían 2 de nuestras líneas DATA superpuestas (el CPC duplica las coordenadas Y). Este hecho debe considerarse también al indicar la altura de la figura. Para cada línea de puntos a visualizar se deben contar 2 puntos en dirección Y. En dirección X ocurre algo similar. En este caso se multiplica a través de la variable M, que contiene un número diferente según la modalidad de pantalla que se utilice. En modalidad 0 M debe contener el valor 4 porque en este caso 4 coordenadas X se refieren al mismo punto, en modalidad 1 el factor a utilizar es 2 y en modalidad 2 el valor será 1.

Para aclarar la utilización de esta subrutina proporcionamos a continuación un programa ejemplo:

```
100 MODE 2
110 XS=320: YS=200: M=1: GOSUB 1050: END
120 DATA "10000100001100010000010000011111 1"
130 DATA "10000100010010010000000000100001"
140 DATA "10000100100001010000010000100001"
150 DATA "10000100100001010000010000100001"
160 DATA "11111100111111010000010000100001"
170 DATA "11000100110001011000011000 110001"
180 DATA "11000100110001011000011000 0110001"
190 DATA "11000100110001011000011000 0110001"
200 DATA "11000100110001011111101111101111111"
210 DATA ""
```

Estas líneas las debe agregar al listado del programa. En las líneas DATA se indica la configuración de puntos para la palabra HALLO. Como pueden comprobar un 1 representa un punto activado y un 0 punto no activado. Si se trabaja en otra modalidad pueden utilizarse varios colores al unísono, por ejemplo "F" para color 15.

5.5 SISTEMAS DE COORDENADAS

El siguiente apartado será de interés para matemáticos y estudiantes. Los primeros sabrán perdonarme si explico cosas tan sencillas como sistemas de coordenadas en forma poco académica, pero los últimos podrán comprender mejor de esta manera temas con los que tal vez estén aún poco familiarizados.

Cada vez que quiera visualizarse una función de forma gráfica se deben calcular los valores de las funciones en coordenadas de pantalla. El comando ORIGIN les puede ahorrar algo de su trabajo. Repasemos el procedimiento con el ejemplo de una curva de seno.

Como ya saben los valores de la función en el seno, se encuentran entre -1 y +1. Por ello el eje de las 'X' se debería posicionar exactamente en el centro de la pantalla.

El eje de las 'Y' en la mayoría de los casos en las funciones trigonométricas, se posiciona en el borde izquierdo. De esta manera obtenemos para el punto 0 de nuestro sistema, las coordenadas de pantalla (0,199). Si introducimos ORIGIN 0,199 todos los comandos gráficos sucesivos, se referirán a este nuevo punto 0. PLOT 100,-1 nos posiciona un punto directamente debajo del eje X.

Si han experimentado ya con el comando ORIGIN y han olvidado los valores pertinentes pueden visualizarlos nuevamente a través de su pantalla mediante la siguiente línea:

```
PRINT UNT(PEEK(&B328)+256*PEEK(&B329))
```

De esta manera se obtiene el valor para la dirección X. Para el eje Y solamente hemos de modificar las direcciones &B32A y &B32B.

Si introducen ahora los valores de la función del seno en los comandos de gráficos (por ejemplo PLOT X,SIN(X)), su curva de seno

tendría únicamente un pixel de altura. Por ello el comando correcto es el siguiente:

```
PLOT X,SIN(X)*199
```

Las coordenadas X pueden calcularse con un factor de multiplicación, si el área comprendida no es idéntica a las coordenadas de gráficos.

Ofrecemos a continuación una subrutina que traza un sistema de coordenadas en la pantalla, según los valores introducidos para el área de X e Y.

```
1000 INPUT "X MINIMO "; XA
1010 INPUT "X MAXIMO "; XE
1020 INPUT "Y MINIMO "; YA
1030 INPUT "Y MAXIMO "; YE
1040 CLG
1050 MX=(XE-XA)/639: X= -XA/MX
1060 MY=(YE-YA)/399: Y= -YA/MY
1070 ORIGIN X,Y
1080 MOVE XA/MX,0: DRAW XE/MX,0: REM EJE Y
1090 MOVE 0,YA/MY: DRAW 0,YE/MY: REM EJE X
1100 RETURN
```

El cometido de las primeras 5 líneas no lo comentaremos ya que creemos es suficientemente claro.
En las siguientes 2 líneas se calcula la escala para los ejes X e Y así como las coordenadas para el punto cero (X,Y).
La escala se calcula mediante la diferencia de mínimo y máximo del área correspondiente, dividido por la cantidad total de puntos posibles. La diferencia indica cuantos puntos se desean; que se proyectan luego mediante la división a los puntos posibles.
Siempre que se desee calcular una coordenada de pantalla de un valor de función o de un valor X, solamente hemos de dividir entre la escala correspondiente. De la misma forma se procede en el cálculo de las nuevas coordenadas del punto cero.

La línea 1080 traza el eje Y y la línea 1090 el eje X. También aquí se encuentra nuevamente la escala.

Pueden afinar el programa introduciendo denominaciones de ejes, puntos, áreas, etc. mediante los comandos TAG y PRINT. Para ello naturalmente hay que "Plotear" los puntos correspondientes de la escala.

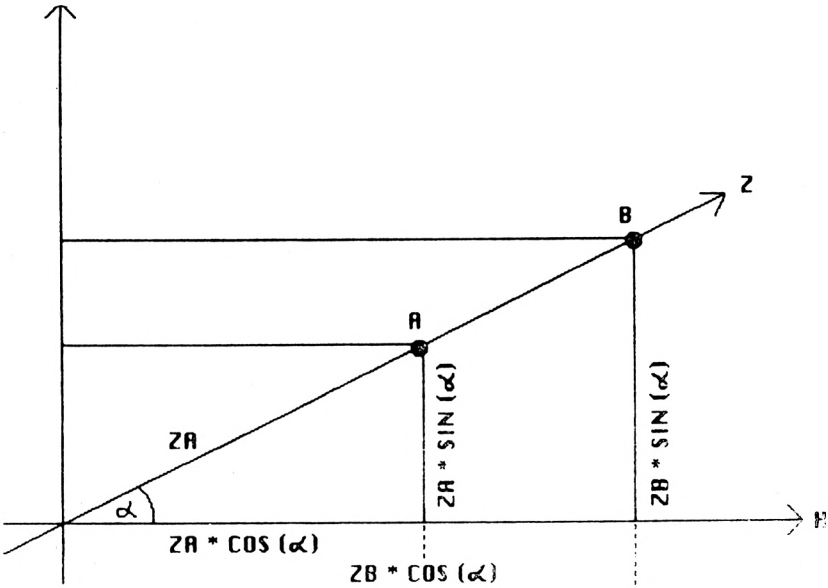
5.6 GRAFICOS 3D

Sin duda el tipo más interesante de gráficos corresponde a los gráficos tridimensionales, pero son asimismo los más laboriosos. Queremos no obstante aclarar un poco esta técnica mediante un pequeño programa.

Veamos en primer lugar el sistema de coordenadas. A diferencia de la representación normal de funciones, en este caso contamos con 3 ejes: X,Y,Z.
El eje Z, por necesidades de representación, se debe posicionar inclinado en el espacio (vean dibujo siguiente), el ángulo (y con él la perspectiva) los pueden definir ustedes mismos.

Dado que en el comando PLOT no pueden indicarse 3 coordenadas, debemos calcular el gráfico en 2 dimensiones. Esto se denomina "proyección"; trataremos únicamente la proyección simple paralela, en la cual no existe ningún punto de alineación. En la realidad las líneas paralelas se proyectan -como el nombre indica- de forma paralela y no se unen hacia el infinito.

Como pueden comprobar en el dibujo, hay que desplazar un punto según la coordenada Z más o menos hacia la derecha. En el eje Z de nuestro ejemplo, hay 2 puntos dibujados, que corresponden a las coordenadas (0,0,Za) y (0,0,Zb). Como pueden ver las coordenadas de la proyección pueden calcularse mediante la adición de



Dibujo 4

Sistema de coordenadas para gráficos 3D.

Za*cos(a) y Zb*sin(a). Las fórmulas de proyección son las siguientes:

x= 0 + za*cos(a) y= 0 + za*sin(a)

o de forma generalizada:

x= xa + za*cos(a) y= ya + za*sin(a)

Debido a que estas coordenadas 2D no siempre corresponden a las coordenadas de la pantalla, introducimos además el correspondiente factor de ampliación (SX,SY línea 100 del listado de programa). El origen de las coordenadas se traslada mediante ORIGIN (320,200) para que el dibujo aparezca en el centro de la pantalla.

Dado que debemos calcular únicamente una de las 3 coordenadas originales, pues las otras 2 están ya predefinidas, utilizaremos solamente 2 bucles FOR-NEXT anidados en nuestro programa. Además por 2 razones que explicaré más adelante, plotearemos la función desde atrás hacia delante.

En el siguiente programa, todos los valores que se modifican repetidas veces, se introducen mediante comandos INPUT. El programa trabaja en modalidad 2 porque no se trata de representar muchos colores sino de utilizar la resolución máxima. Introduzca el programa y arránquelo tecleando los valores siguientes:
W=45, SX=0.5, SY=0.5, XS=2, ZS=20

```
10 INPUT "ANGULO "; W: DEG: Z1=COS(W): Z2=SIN(W)
20 INPUT "AMPLIACION X "; SX
30 INPUT "AMPLIACION Y "; SY
40 INPUT "INTERVALO X "; XS
50 INPUT "INTERVALO Z "; ZS
60 MODE 2: ORIGIN 320,200
70 FOR Z=180 TO -180 STEP -ZS
80 FOR X=-180 TO 180 STEP XS
90 Y=SIN(X)*COS(Z)*100: REM TERMINO DE FUNCION
```

```
100 BX=SX*(X+Z*Z1): BY=SY*(Y+Z*Z2)
110 PLOT BX,BY,1
120 NEXT X,Z
```

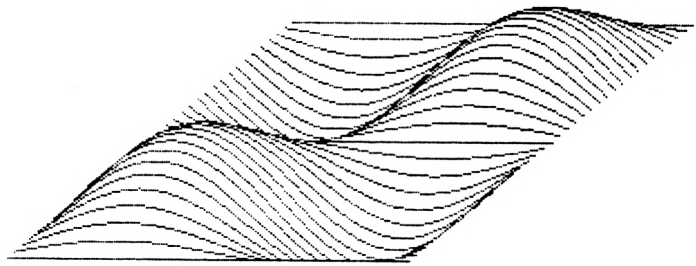
Es posible que haya notado algo al ejecutar el programa. Existen diversos puntos que deberían taparse ópticamente por porciones superpuestas del dibujo. Este problema se denomina superposición de imágenes y puede resolverse si todos los puntos que se encuentren por debajo del punto recién activado se borran. Para ello modifique la línea 110 de la siguiente forma:

```
110 PLOT BX,BY,1: MOVE BX,BY-2: DRAW BX,-200,0
```

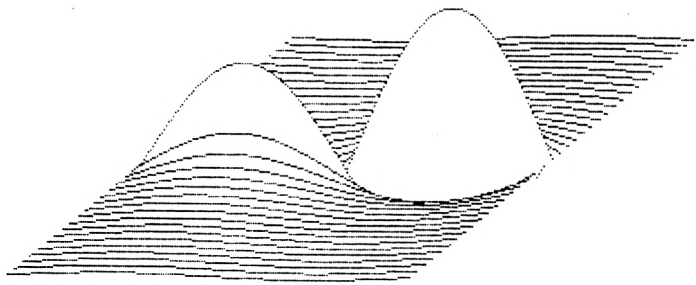
Permítanme explicarlo con un ejemplo. Supongamos que nuestra función dibuja una superficie totalmente plana. Esta superficie se traza de forma inclinada desde abajo hacia arriba en el espacio. El punto más cercano es el punto más bajo y por ello no puede tapar a ningún otro. Supongamos ahora que la coordenada Y de la primera línea de puntos se incrementa en 1, en este caso se superpone exactamente a la línea superior. Si incrementamos más aún las coordenadas se producirá mayor superposición. Resulta que todos los puntos por debajo de cada línea inferior, deben borrarse. Lo mismo ocurre si la línea inferior representa una curva u otra figura. Para comprender mejor esto, fíjense en el borrado al realizar el gráfico.

Explicaremos un poco más las diferentes líneas del programa. Línea 10 calcula los valores del seno y coseno del ángulo W. Estos valores pueden utilizarse como constantes dado que quedan inalterados, lo que ahorra mucho tiempo de cálculo. Los números en los bucles FOR-NEXT (líneas 70 y 80) pueden modificarse para visualizar porciones menores o mayores de la función.
Seguramente habrá notado el '*100' en la línea 90. Como los valores de función siempre serán menores que 1, tenemos que ampliar estos valores proporcionalmente por separado.

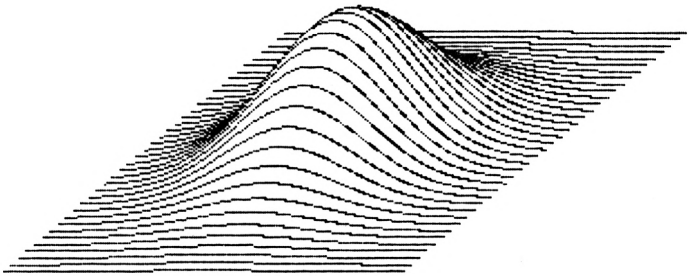
Intenten probar ahora con intervalos, factores de ampliación y funciones diferentes.



dibujo 5 y = sin(x) * sin(z) * 140

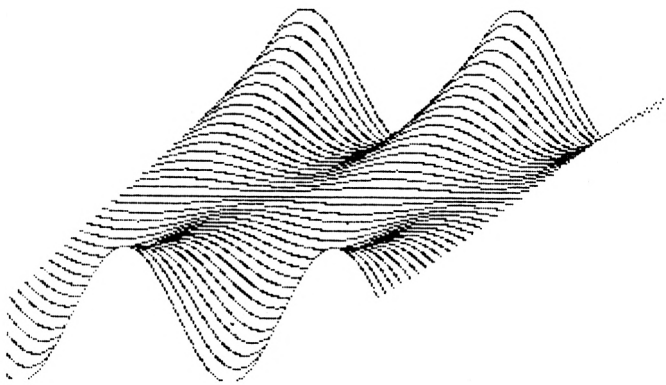


dibujo 6 y = sin(x) * 2 / (z / 8 + 0.5) * 80



dibujo 7

```
y = exp(-(x*x+z*z)/2)
    /sqr(2*π)*300
```



dibujo 8

```
y = sin(x/30)*(exp(z/90)
    -1/exp(z/90))*10
```

6. APLICACIONES DE GRAFICOS

Con las ayudas de los últimos capítulos ya pueden confeccionarse gráficos interesantes, pero hasta el momento no hemos hablado de gráficos específicos para aplicaciones. Si Ud. cuenta entre los usuarios que utilizan profesionalmente su CPC, le interesará mucho el capítulo 6.1. Para los usuarios de Hobby y para los artistas está pensado el programa de dibujo del capítulo 6.2.

6.1 DIVERSOS DIAGRAMAS

Se supone que todos los propietarios de ordenadores al ver los múltiples cálculos de la noche de elecciones han pensado en confeccionar gráficos de este tipo, o es posible que quiera representar las estadísticas de ventas por medio de un diagrama de barras. Esto puede realizarse fácilmente con el CPC. En principio una barra no es otra cosa que un rectángulo, como lo hemos programado en el capítulo 5.1. Solamente hemos de modificar nuestros valores a un sistema de coordenadas. El cálculo de los valores originales en coordenadas de la pantalla, ya se ha explicado en capítulo 5.5; este método lo utilizaremos nuevamente.

Para el área X no necesitamos indicar máximo y mínimo sino únicamente la cantidad de barras a visualizar en la variable XE. Este número se multiplica por 10 para reservar los puntos para el ancho de la barra (una barra ha de ser más ancha que un pixel; vean línea 60040).

La tabla W (0....XE) debe contener todos los valores. El máximo valor se calcula en el primer bucle FOR-NEXT. Dado que la numeración de elementos de la tabla comienza con 0 hemos de restar 1 de la cantidad XE pues en general humanamente se comienza a contar con 1.

En las fórmulas de escala seguramente se dará cuenta de las constantes de puntos disminuída, y por ello de la superficie disminuída de nuestro diagrama. Esto es necesario para dejar un poco de espacio de la pantalla, para descripciones.

Dado que los valores de estadísticas de ventas nunca serán negativos podemos ahorrarnos los mínimos en ambas áreas (Xa e Ya).

La línea 60040 borra la pantalla y posiciona el origen de las coordenadas en el punto (50,30). De esta manera queda espacio resarvado para descripciones a la izquierda y por debajo de las barras. En la línea 60050 se dibuja el borde del área de barras; que funciona casi igual que en el sistema de coordenadas.
En las líneas 60060 y 60070 puede encontrar el programa del rectángulo de forma ligeramente modificada. Dado que queremos dibujar barras verticales es aconsejable y más rápido utilizar el DRAW para las líneas en los rectángulos.
Además se han reemplazado algunas variables, directamente por expresiones aritméticas. Para ello queda por decir que la constante 10 indica la cantidad de unidades de escala disponibles para una barra mas un espacio. El ancho real se indica con "+8" en la línea 60070 (nuevamente en unidades de escala, no en cantidad de pixels). Si quieren espacios más grandes o más pequeños sólo deben alterar este valor.

```
60000 REM DIAGRAMA DE BARRAS: XE, W(0...XE-1), F
60010 XE=XE-1: YE=0
60020 FOR II=0 TO XE: YE=MAX(YE,W(II)): NEXT
60030 MX=(XE+1)*10/584: MY=ZE/364
60040 CLG: ORIGIN 50,30
60050 MOVE -5,0: DRAW -5,YE/MY+5: MOVE -5,0: DRAW (XE+1)*10/MX+5,0
60060 FOR II=0 TO XE: FOR JJ=II*10/MX TO (II*10+8)/MX
60070 MOVE JJ,0: DRAW JJ,W(II)/MY,F: NEXT JJ,II
60080 RETURN
```

El segundo tipo de diagramas parece mucho más complicado pero también en este caso pueden utilizarse subrutinas antiguas modificadas. Nos referimos a diagramas de 'pastel', en los cuales se asigna un segmento del círculo más grande o más pequeño según los valores a representar.

El procedimiento para confeccionar diagramas de este tipo es muy simple. Se trata de dibujar un segmento de círculo para cada valor, todos los segmentos forman el círculo completo. Para poder diferenciar los distintos segmentos se utilizan variedad de colores.
A lo mejor ya se ha preguntado como se programa un segmento de círculo, pues hasta el momento sólo hemos trabajado con circunferencias y círculos enteros. Para la siguiente explicación vuelvan a repasar el capítulo 5.3 En el listado línea 65516 ven la secuencia de comandos 'FOR I = 0 to 359'. Como ya se ha explicado en su momento, se repasa toda la circunferencia desde 0 hasta 359. Si entramos en cambio FOR I=180 TO 270, obtenemos un sector de circunferencia correspondiente a la cuarta parte. Como pueden ver puede seleccionarse libremente el punto de comienzo y de final del segmento de circunferencia.
Existe no obstante otro problema, los lectores despiertos se habrán dado cuenta que nuestra rutina original dibuja la circunferencia matemáticamente correcta partiendo del 'Este' en sentido inverso al de las agujas del reloj. Nuestra costumbre de lectura es exactamente a la inversa. Esto puede solucionarse intercambiando las funciones seno y coseno de las coordenadas. De esta forma se comienza el dibujo en el punto 12 horas del reloj (en la posición más alta) y se mueve en dirección de lectura en el sentido de las agujas del reloj. Presentamos a continuación el listado:

```
60100 REM DIAGRAMA DE PASTEL: X1, Y1, R, XE, W(0...XE-1),
      F(0...XE-1)
60110 WB=0: DEG: FOR II=0 TO XE-1
60120 WA=WB: WB=WB+W(II)*3.6: S(II)=(WB-WA)/2
60130 FOR JJ=WA TO WB
60140 XX=SIN(JJ)*R: YY=COS(JJ)*R
```

```
60150 MOVE X1,Y1: DRAWR XX,YY,F(II): PLOTR 0,0,1
60160 NEXT JJ,II: RETURN
```

Esta subrutina espera también diversos parámetros en la variable de traspaso (vean lista en 60100). X1 y X2 indican el punto medio de la circunferencia, R el radio de la misma. La función de XE ha quedado igual, los valores que indican el tamaño de los segmentos del círculo están incluídos como porcentajes en la tabla W(0...XE-1). Si un campo debe ocupar el 20 por ciento del círculo, la variable correspondiente ha de contener un 20.
Además se puede indicar un color individual (F(II)) para cada segmento.

Este programa contiene también 2 bucles anidados. El exterior se ocupa de que se utilicen todos los valores, el interior repasa todos los ángulos del segmento.

En línea 60120 se calculan los ángulos inicial y final de los segmentos. Nos permitimos además algún lujo. En la tabla S(0...XE-1) (la que debe DIMensionarse antes de utilizar la subrutina) se almacena el ángulo que divide en 2 el segmento. Si se quiere describir el segmento puede posicionarse el cursor de gráficos en este ángulo y visualizar un texto en la posición correcta, mediante TAG. En este caso deben calcularse asimismo las coordenadas con SIN y COS, pero debería ampliarse el radio para que el texto no toque la circunferencia.

Dentro de la rutina del círculo es nuevo el comando PLOTR. Este dibuja un borde alrededor del diagrama para prever el caso de que el color del segmento sea negro. De lo contrario puede resultar la impresión de una estrella.
El resto del programa ya les es familiar por tanto no se hacen más aclaraciones.

Deberían tener en cuenta que la suma de los valores en la tabla W debe ser exactamente 100, porque de lo contrario puede ocurrir que sólo se dibuje una parte del círculo o que el ordenador se "tragara" un segmento.

6.2 EL PROGRAMA PARA LOS 'ARTISTAS'

El siguiente programa está pensado para aquellas personas que experimentan mucho con los gráficos y que quieren utilizar imágenes prefabricadas en otros programas propios. Es demasiado laborioso confeccionar imágenes completas mediante comandos PLOT y DRAW; es más fácil "pasear" con el cursor por la pantalla y activar o desactivar puntos según sus necesidades. Esto se puede realizar mediante el siguiente programa:

```
10 MEMORY 16383: CALL &BC06,&40: MODE 2: X=320: Y=200: M=1
20 A$="": WHILE A$="": A$=INKEY$: WEND: PLOT X,Y,F
30 IF A$=CHR$(240) AND Y<398 THEN Y=Y+2
40 IF A$=CHR$(243) AND X<639 THEN X=X+1
50 IF A$=CHR$(241) AND Y>1 THEN Y=Y-2
60 IF A$=CHR$(242) AND X>0 THEN X=X-1
70 IF A$=CHR$9224) THEN F=1: BORDER 24: M=0
80 IF A$=CHR$(13)  THEN F=0: BORDER 24,0: M=0
90 IF A$=" " THEN M=1: BORDER 0
100 IF A$="S" THEN CALL &BC06,&C0: MODE 2: INPUT "FILE NAME"; F$:
    SAVE F$,B,16384: CALL &BC06,&40: GOTO 20
110 IF A$="L" THEN CALL &BC06,&C0: MODE 2: INPUT "FILE NAME"; F$:
    LOAD F$,16384: CALL &BC06,&40: GOTO 20
120 IF A$="C" THEN CLG: GOTO 20
130 IF A$="X" THEN CALL &BC06,&C0: END
140 IF M=1 THEN F=TEST(X,Y)
150 PLOT X,Y,1: GOTO 20
```

Para el gráfico se utiliza la segunda memoria de pantalla a partir de 16384, de esta manera se asegura que eventuales entradas de texto no estropeen la imagen.

Al dibujar existen 3 modalidades que se hacen notar por diferentes colores del borde. La primera modalidad la llamaremos modalidad

cursor, porque puede moverse el cursor gráfico a través de la pantalla sin alterar los puntos superpuestos por el mismo. El borde en este caso es oscuro. Si el mismo es de color claro se activa el punto debajo del cursor gráfico, si parpadea se borra el punto correspondiente. Cualquiera de esta modalidades tiene asignada una tecla mediante la cual puede llamarse. Para la modalidad cursor es la tecla de espacio, los puntos se activan al pulsar COPY y ENTER lleva el programa a modalidad de borrado.

Pero esto no es todo. Con la tecla S se puede grabar el gráfico en cassette/diskette. Para ello se conmuta a la memoria de pantalla original, donde puede introducirse el nombre del fichero, grabandose a continuación (vean línea 100). De forma similar funciona la carga de una imagen ya confeccionada pulsando L (línea 110). En ambas líneas se encuentran comandos MODE con los cuales se borra la pantalla y se evitan los efectos de scrolling al conmutar.

La tecla C borra completamente la pantalla. Finalmente nos queda el X, con la cual puede finalizarse el programa sin pérdida de la imagen. Si a continuación se quiere modificar alguna cosa y volver a extraer el gráfico original se debe teclear CALL &BC06,&40: GOTO 20.

Aclaremos un poco más las líneas del programa. Línea 10 inicializa la pantalla y posiciona las coordenadas en sus valores iniciales. En la siguiente línea se extrae un carácter del teclado y se activa el nuevo pixel (resultado del anterior paso del bucle). A continuación se comprueba la pulsación de tecla en diferentes construcciones IF-THEN.

La variable F indica el color para el comando PLOT de la línea 20. Si está activada la modalidad de cursor (M=1) F no se determina por una constante sino que se determina por el color del pixel original. De esta manera la imagen se mantiene invariable. El comando PLOT se posiciona después del INKEY$ para mantener el cursor de gráficos en la pantalla el máximo tiempo posible. El cursor mismo se genera mediante el comando PLOT en línea 150.

Naturalmente este pequeño programa no les presenta un super software, porque esto excedería los límites de este libro. Sin embargo queremos darles la base para desarrollos propios y una pequeña ayuda para los primeros tiempos.
Imágenes que se han almacenado con el programa pueden cargarse en la memoria de pantalla normal mediante el comando LOAD "!Nombre",49152.

7. PROGRAMACION DE INTERRUPCIONES

La programación de interrupciones es una de las particularidades sobresalientes del BASIC del CPC. Lamentablemente no se describe en el manual el funcionamiento de las interrupciones ni se explican los correspondientes comandos. Intentaremos aclarar estos extremos en los siguientes apartados.

7.1 ¿COMO FUNCIONA UNA INTERRUPCION BASIC?

Generalmente una interrupción de lenguaje máquina no se diferencia de una interrupción BASIC. En ambos casos un circuito (casi siempre el TIMER) produce una señal de interrupción que suspende el programa que se ejecuta en ese momento (una vez ejecutado el comando activo en el momento de producirse la interrupción) y provoca una bifurcación a una subrutina especial. En el Z80 todo esto se resuelve a base de Hardware, lo que significa que no son programas sino circuitos especiales los que se ocupan del procedimiento, por el contrario en BASIC se trata de programas elaborados en lenguaje máquina que se responsabilizan de la ejecución correcta; además una interrupción BASIC solamente puede provocarse por uno de los 4 TIMERS (0-3) pero no por otro circuito (por ejemplo una interfase).
Si se produce una señal de interrupción el BASIC comprueba en primer lugar si se debe finalizar la ejecución de algún comando. Por ello no es posible interrumpir instrucciones INPUT o similares en tiempo de proceso.
También podría darse que la rutina de interrupción hubiera sido bloqueada por un comando DI (Disable Interrupt). En ambos casos el interpretador BASIC almacenará temporalmente la petición de la interrupción ejecutándola cuando sea posible.
Finalmente comprueba que rutina corresponde al TIMER recién procesado, arrancándola como una subrutina normal.

Todavía no son estas todas las tareas que debe tener en cuenta el CPC relacionadas con las interrupciones. Si se han definido modificaciones de colores mediante el comando INK esta alteración de colores la provoca un TIMER especial, que puede inicializarse a través de SPEED INK. Siempre que se desee cambiar el color el sistema operativo modifica un byte en el chip responsable del manejo de colores.
Otra tarea importante es la comprobación del estado del teclado, que se realiza como de costumbre con una interrupción, que provoca que cada 1/50 de segundo el componente interfase envíe el código de la tecla pulsada al procesador.

7.2 COMANDOS DE INTERRUPCION

Después de toda esta teoría investigaremos ahora con mayor profundidad los comandos de interrupciones. En primer lugar mencionaremos los comandos que activan la interrupción BASIC: AFTER y EVERY.
Aclaremos para los usuarios que no dominen el inglés, que AFTER significa 'después' y EVERY es 'cada'.
Ambos comandos tienen tareas similares y su sintaxis es la misma:

```
AFTER X,Y,GOSUB Z
EVERY X,Y,GOSUB Z
```

Tanto AFTER como EVERY inicializan un TIMER, cosa similar a activar un despertador a una hora determinada. A diferencia de nuestra "herramienta de despertar" solamente debemos indicar en cuantos 1/50 de segundo se debe provocar la interrupción.

Este valor podemos determinarlo en el parámetro X. Un AFTER 200... provoca una interrupción después de 200/50 = 4 segundos.
Ya hemos llegado a la única diferencia entre AFTER y EVERY, un comando AFTER provoca una única interrupción mientras que el EVERY inicializa el TIMER en modalidad continua, los que significa que cada vez que termina un ciclo del tiempo indicado, comienza un ciclo nuevo. De esta manera por ejemplo cada 10 segundos puede extraerse un valor desde el teclado mientras que casi al mismo tiempo se pueden realizar los cálculos correspondientes.

El segundo parámetro (Y) indica cuál de los 4 TIMER se desea utilizar. Cuanto mayor sea el número mayor significado tiene la interrupción. TIMER 3 por ejemplo puede interrumpir TIMER 2 pero no a la inversa.
El resto del comando se explica por sí sólo ya que se indica únicamente el número de línea a bifurcar en el caso de una interrupción.

Los 2 comandos DI y EI se explican rápidamente. DI es la abreviación para 'Disable Interrupt' lo que significa que a partir del momento en que el comando se ejecuta, no se tiene en cuenta ninguna otra interrupción, auque tenga un TIMER de prioridad superior. El interpretador se limita a almacenar las peticiones de interrupción para ejecutarlas posteriormente, en el momento en que se produce un EI o un comando RETURN. EI es la abreviación para 'Enable Interrupt', y significa que se atienden todas las peticiones de interrupción que estuvieran almacenadas así como las que se produzcan posteriormente.
DI y EI se utilizan principalmente en rutinas que trabajan con comandos gráficos, para evitar que se modifique el cursor de gráficos a través de una segunda interrupción.

Para comprender mejor el sistema de prioridades y del bloqueo de interrupciones presentamos a continuación un pequeño programa ejemplo que contiene 2 rutinas de interrupción; el programa principal (línea 30) es un bucle sin fin.

```
1 CLS: PRINT T; "   SEGUNDOS'
2 LOCATE 6,5: PRINT "SEGUNDOS"
10 EVERY 50,0 GOSUB 100
20 EVERY 50,1 GOSUB 200
30 GOTO 30
100 T=T+1: LOCATE 1,1: PRINT T
101 WHILE INKEY$="": WEND
102 RETURN
200 X=X+1: LOCATE 1,5: PRINT X
201 RETURN
```

Ambas rutinas de interrupción incrementan en 1 un contador por cada segundo entero. La rutina a partir de línea 200, tiene una prioridad mayor porque se provoca con TIMER 1, por ello la subrutina a partir de línea 100, necesita más tiempo para su ejecución pues espera la pulsación de una tecla (línea 101). Arranque el programa y verá que la primera rutina continuamente se interrumpirá con la segunda (el contador de segundos superior se altera después de una pulsación de tecla, el inferior no se influye con la pulsación).
Si ahora arranca nuevamente el programa, también el segundo contador de segundos se actualiza al pulsar una tecla, porque la segunda subrutina tiene que esperar su liberación mediante el RETURN al final de la primera rutina.

El último comando en relación con las interrupciones, se llama REMAIN. Tiene 2 formas posibles:
a. REMAIN (Y)
b. PRINT REMAIN (Y) o A=REMAIN (Y) etc.

En todos los casos provoca que se reinicialice el TIMER Y, lo que significa que no puede provocar más interrupciones.
Si se utiliza el REMAIN como función (formato b), se indica adicionalmente cuantos 1/50 de segundo hubieran quedado hasta la siguiente interrupción.

7.3 IDEAS PARA LA PROGRAMACION DE INTERRUPCIONES

El ejemplo que se utiliza con mayor frecuencia para una rutina de interrupciones, es el reloj, que se visualiza continuamente en algun punto de la pantalla. Para ello se llama cada segundo entero (EVERY 50...) a una subrutina que incrementa en 1 el número de segundos y que actualiza también horas y minutos. El problema es que el reloj después de cada operación de cassette/diskette no funciona correctamente (se retrasa), porque estos comandos suspenden momentáneamente el TIMER.

¿Pero como funcionaría en un juego de adivinanzas, en el cual no se trata únicamente de encontrar la respuesta correcta sino que se tiene en cuenta también la velocidad? El procedimiento podría ser el siguiente:

1. Se le hace una pregunta al candidato "en el teclado".
2. Se le dan varias respuestas posibles codificadas. Este código puede solicitarse a través de INKEY$.
3. Mediante un comando AFTER se interrumpe el bucle INKEY$ después de un tiempo determinado (por ejemplo 10 segundos). El candidato obtiene puntos positivos o negativos según su respuesta y el tiempo que ha necesitado para darla.

El punto 2 no se puede programar mediante un INPUT porque una interrupción no interrumpe nunca un comando en ejecución - tampoco el INPUT.

Otra aplicación sería intercambiar periódicamente 2 caracteres de la pantalla. De esta manera se pueden visualizar movimientos fácilmente. El primer carácter, por ejemplo representa un PAC-MAN con boca abierta, en el segundo la boca está cerrada. Si se intercambian ambos caracteres periódicamente dará la impresión del movimiento de masticación.

Del mismo modo se podría por ejemplo mover un coche sobre la pantalla con velocidad constante, mediante interrupciones.

Hay muchas aplicaciones de este tipo, dejamos a su imaginación la utilización de estas posibilidades.

8. SOUND

Muchos especialistas catalogan el CPC como 'Superordenador'. Cuenta con detalles muy realizados en la generación de sonidos. Mi opinión es que hay que escuchar la programación de sonidos para poder evaluarla, y por ello aquí no encontrarán demasiadas explicaciones. En cambio quiero insistir en la recomendación de leer cuidadosamente las aclaraciones del manual, de las cuales describiré únicamente las aplicaciones resultantes.

8.1 MINI-SINTETIZADOR

Las instrucciones ENT y ENV ofrecen al programador un sinúmero de posibilidades de modificación de tonos e incluso de simulación de instrumentos. Mediante estos 2 comandos pueden alterarse todos los factores necesarios menos la tonalidad. Abajo ofrecemos un programa con el cual pueden modificarse todos los parámetros mediante la pulsación de tecla. Esto permite experimentos rápidos con la curva envolvente.

```
10 MODE 2: WINDOW #1,2,46,2,7: REM MENU Y VENTANAS DE TRABAJO
20 WINDOW #2,1,40,9,23: REM VENTANA ENV
30 WINDOW #3,42,80,9,23: REM VENTANA ENT
40 MOVE 0,272: DRAW 639,272
50 MOVE 0,32: DRAW 639,32
60 MOVE 320,32: DRAW 320,272
70 MOVE 368,272: DRAW 368,399
80 LOCATE 49,2: PRINT "MINI-SINTETIZADOR VERSION 1.0"
90 LOCATE 49,4: PRINT CHR$(164)" 1985 DATA-BECKER GMBH"
100 LOCATE 49,6: PRINT "AUTOR: HANS JOACHIM LIESERT"
110 LOCATE  2,25:  PRINT "1 = INTERVALO              2  =  TAMAÑO
    INTERV    3 = LONGITUD DE PAUSA"
120 LOCATE #2,3,2: PRINT #2,"ENvelope Volume"CHR$(10)
130 PRINT #2,"          1          2          3"CHR$(10)
140 PRINT #2,"    1     0          0          0"CHR$(10)
150 PRINT #2,"    2     0          0          0"CHR$(10)
160 PRINT #2,"    3     0          0          0"CHR$(10)
170 PRINT #2,"    4     0          0          0"CHR$(10)
180 PRINT #2,"    5     0          0          0"
```

```
190 LOCATE #3,3,2: PRINT #3,"ENvelope Tone"CHR$(10)
200 PRINT #3,"          1          2          3"CHR$(10)
210 PRINT #3,"    1     0          0          0"CHR$(10)
220 PRINT #3,"    2     0          0          0"CHR$(10)
230 PRINT #3,"    3     0          0          0"CHR$(10)
240 PRINT #3,"    4     0          0          0"CHR$(10)
250 PRINT #3,"    5     0          0          0"
260 DATA "q","2","w","3","e","r","5","t","6","y","7","u","i"
270 DIM T$(12): FOR I = 0 TO 12: READ T$(I): NEXT
280 DIM V(5,3),T(5,3)
290 SPEED KEY 255,10: ENV 1: ENT 1
300 LOCATE 1,1: C=1
1000 CLS #1: PRINT #1,"MENU PRINCIPAL"CHR$(10)
1010 PRINT #1,"TECLA ESPACIO = TOCAR MELODIA"
1020 PRINT #1,"V             = MODIFICAR ENV"
1030 PRINT #1,"T             = MODIFICAR ENT"
1040 A$="": WHILE A$="": A$=INKEY$: WEND
1050 IF A$=" " THEN 1100
1060 IF A$="V" THEN 1200
1070 IF A$="T" THEN 1300
1080 GOTO 1040
1100 CLS #1: PRINT #1,"TECLADO"CHR$(10)
1110 PRINT #1," 2 3   5 6 7"
1120 PRINT #1,"Q W E R T Y U I"CHR$(10)
1125 PRINT #1,"TECLA ESPACIO = MENU"
1130 A$="": WHILE A$="": A$=INKEY$: WEND
1140 IF A$=" " THEN 1000
1150 FOR I = 0 TO 12: IF A$<>T$(I) THEN NEXT I: GOTO 1130
1160 PER=ROUND(125000/440/2^(I/12))
1165 C=C*2: IF C=8 THEN C=1
1170 SOUND C,PER,0,0,1,1
1180 GOTO 1130
1200 CLS #1: PRINT #1,"MODIFICAR ENV"CHR$(10)
1210 PRINT #1,"TECLEAR 0 PARA FINAL"CHR$(10)
1220 INPUT #1,"QUE ELEMENTO (LINEA, COLUMNA)";Z,S
1230 IF Z*S=0 THEN 1000
1240 INPUT #1,"VALOR";V(Z,S)
1250 LOCATE #2,S*10,4+2*Z: PRINT #2,V(Z,S); "    "
```

```
1260 ENV 1,V(1,1),V(1,2),V(1,3),V(2,1),V(2,2),V(2,3),V(3,1),
     V(3,2),V(3,3),V(4,1),V(4,2),V(4,3),V(5,1),V(5,2),V(5,3)
1270 GOTO 1200
1300 CLS #1: PRINT #1,"MODIFICAR ENT"CHR$(10)
1310 PRINT #1,"TECLEAR 0 PARA FINAL"CHR$(10)
1320 INPUT #1,"QUE ELEMENTO (LINEA, COLUMNA)";Z,S
1330 IF Z*S=0 THEN 1000
1340 INPUT #1,"VALOR"; T(Z,S)
1350 LOCATE #3,S*10,4+2*Z: PRINT #3,T(Z,S);"    "
1360 ENT -1,T(1,1),T(1,2),T(1,3),T(2,1),T(2,2),T(2,3),T(3,1),
     T(3,2),T(3,3),T(4,1),T(4,2),T(4,3),T(5,1),T(5,2),T(5,3)
1370 GOTO 1300
```

Después de arrancar aparece la máscara de pantalla con el menú principal (arriba a la izquierda). Mediante pulsación de tecla se selecciona la parte de programa deseada. Mientras que los parámetros están en 0, no tiene sentido tocar una melodía. Por ello se debería entrar al menos la curva envolvente de volumen, introduciendo el elmento a modificar y su nuevo valor, alterando inmediatamente el contenido de la matriz. Todos los parámetros se transmiten directamente a los comandos de curva envolvente y por ello puede tranferirlos fácilmente a programas propios.

Algunas aclaraciones sobre este programa:

Las líneas 10 a 270 confeccionan la máscara de pantalla y las 3 ventanas. 2 ventanas se utilizan para visualizar los parámetros ENV y ENT y en la ventana restante se tratan todas las entradas e instrucciones de funcionamiento. En línea 260 se introduce la ocupación del teclado para los diversos tonos en la tabla t$. Mediante comparación puede asignarse a cada tecla el índice de campo como "número de tono", que sirve para el cálculo del período de tono.

En línea 280 se dimensionan las tablas de parámetros para ENV y ENT. La línea 290 desactiva la función REPEAT y borra curvas envolventes anteriores.

En línea 1000 comienza el programa en sí, visualizando en primer lugar el menú. En las líneas siguientes se extrae un carácter del teclado (A$), por medio del cual se puede bifurcar a las diversas porciones del programa.

Las líneas 1100 hasta 1180 representan la porción del programa "tocar melodía". Es muy interesante en especial el bucle FOR-NEXT en línea 1150, que continúa únicamente si no se encuentran coincidencias entre A$ y T$(I). De lo contrario se calcula de I (= número tecla/tono) el período de tono (PER) para el comando SOUND.
En línea 1200 comienza la porción de programa de modificación, para ENVelope-Volume. Gracias a la técnica 'window' se pueden solicitar los datos necesarios mediante comandos INPUT. La última parte (a partir de línea 1300) sirve para modificar los parámetros ENT y es casi idéntico a la parte ENV.

8.2 COMO SE "PLANIFICA" UN TONO

Cómo ya saben del manual del CPC se pueden imitar las características de sonido de diversos instrumentos mediante las curvas envolventes. Para lograr esto además de una gran dosis de paciencia para experimentar, hace falta el conocimiento de las curvas envolventes de tonos "reales". Es sabido que el tono de una trompeta se inicia y acaba con la misma velocidad, mientras que una campana tiene un inicio rápido y el sonido se continúa. Un piano tiene también un inicio fuerte y el sonido se continúa, la única diferencia es la modalidad de tono, porque suena de una forma más compleja. En las líneas siguientes queremos intentar simular diversos instrumentos mediante la modificación de los parámetros de la curva envolvente.
Una campana de cristal tiene un tono muy limpio lo que significa un tono de frecuencia constante, y por ello el tono no se modifica mediante los parámetros ENT. El volumen en cambio tiene que subir velozmente al punto más alto y bajar lentamente a 0. Ambas matrices de parámetros deberían tener el siguiente aspecto:

1	15	1	1	0	1
1	0	1	1	0	1
1	0	1	1	0	1
12	-1	8	1	0	1
2	-1	20	1	0	1

o en el comando ENV de esta forma:

```
ENV 1, 1,15,1, 1,0,1, 1,0,1, 12,-1,8, 2,-1,20
```

Una campana de metal suena un poco ronca, esto en el CPC se puede lograr alterando continuamente la frecuencia mediante la curva envolvente del tono. El comando correspondiente sería:

```
ENT -1, 1,1,3, 1,-1,2, 1,0,1, 1,1,3, 1,-1,3
```

Lamentablemente la generación de tonos del CPC tiene una desventaja, no hay posibilidad de alterar la tonalidad. A diferencia de otros ordenadores el CPC no puede producir tonos sordos como el de una flauta, sino solamente sonidos brillantes. Por ello para algunos instrumentos no hay posibilidad de imitación (por ejemplo las maderas). Tampoco se pueden reproducir los típicos sonidos metálicos de las trompetas (especialmente desde el BASIC). No obstante describiremos algunas curvas envolventes que se aproximan a estos sonidos.

Una armónica debido a la forma en que esta construída así como a la manera en que se ejecuta, tiene una elevación y un descenso lentos del volumen, el tono a su vez no es muy limpio sino un poco ronco, que obtenemos mediante:

```
ENV 1, 7,2,1, 1,1,1, 1,0,1, 1,0,1, 15,-1,1
ENT -1, 1,0,3, 1,-1,1, 1,0,2, 1,0,1, 1,1,1
```

Todos los instrumentos que producen los tonos mediante vibración de cuerdas, tienen un curva envolvente de volumen característica. Después de la elevación rápida del tono, desciende un poco y acaba de descender con mucha lentitud (comando ENV 1, 1,15,1, 1,-3,2, 1,0,1, 1,0,1, 12,-1,4). Con diferentes curvas envolventes de tono, pueden imitarse diversos instrumentos de cuerdas. Un tono similar al del piano se obtiene mediante:

```
ENT -1, 1,1,3, 1,-1,3, 1,0,3, 1,1,3, 1,1,3, 1,-1,3
```

El sonido distorsionado de un Banjo (debemos decir que no muy bien logrado) se obtiene con:

```
ENT -1, 1,2,1, 1,0,2, 1,0, 2, 1,-2,1, 1,0,4
```

Esto es solamente una pequeña muestra de las posibilidades ya que se podrían además imitar otros sonidos que los de los instrumentos. ¿Que les parece una batería que puede programarse mediante ruidos cortos con fuertes elevaciones y descensos rápidos? O bien ¿que les parece producir tonos de fantasía? Su creatividad no tiene límites.

Resumen: Curvas envolventes

```
Campana:              ENV 1, 1,15,1, 1,0,1, 1,0,1, 12,-1,8 2,-1,20
Campana de metal:     ENT -1, 1,1,3, 1,-1,3, 1,0,1, 1,1,3, 1,-1,3
Armónica:             ENV 1, 7,2,1, 1,1,1, 1,0,1, 1,0,1, 15,-1,1
                      ENT -1, 1,0,3, 1,-1,1, 1,0,2, 1,0,1, 1,1,1
Instr. de cuerda:     ENV 1, 1,15,1, 1,-3,2, 1,0,1, 1,0,1, 12,-1,4
Piano:                ENT -1, 1,1,3, 1,-1,3, 1,0,3, 1,1,3, 1,-1,3
Banjo:                ENT -1, 1,2,1, 1,0,2, 1,0,2, 1,-2,1, 1,0,4
```

9. BASIC Y SISTEMA OPERATIVO

Las rutinas del interpretador y del sistema operativo pueden ser muy útiles. Ya conocen una parte de ellas.

9.1 ¿COMO SE ALMACENAN LINEAS BASIC?

Seguramente ya se habrá preguntado cómo se almacena en la memoria una línea BASIC. Como ya saben del capítulo 2 todas las palabras de comando como PRINT, SIN, SAVE, etc, obtienen un código especial llamado TOKEN.
Con éste método se ahorra mucho espacio de memoria (para PRINT en lugar de 5 códigos de letra se utiliza solamente un byte TOKEN). Además el interpretador al ejecutar el programa no se ve obligado a analizar secuencias de letras.

Los nombres de variables y textos en cambio se almacenan como códigos ASCII. Las interrelaciones del tipo *, /, AND, etc. así como caracteres de relación ("=" etc.) tienen asimismo un código TOKEN.
Los números se almacenan en un formato muy complejo, que se altera según el tipo (entero, real) y el tamaño del mismo.

En el interior de la memoria los TOKEN pueden reconocerse porque se componen de bytes de valores superiores a 127 (excepto que se trate de un número). Los nombres de variables y cadenas de caracteres se definen con caracteres con valores inferiores a 128. De esta manera el interpretador solamente necesita una tabla de códigos.

Hagamos una pequeña prueba, borre un programa eventualmente presente con NEW y pulse la siguiente línea (carácter por carácter exactamente igual):

```
100 PRINT "test"
```

Este "programa" lo observaremos a continuación en la memoria. Para ello teclee en modalidad directa:

```
FOR I=368 TO 383: PRINT PEEK (I),: NEXT
```

Nuestra memoria de BASIC comienza en el byte 368 y la secuencia anterior de comandos nos visualiza en la pantalla los primeros 16 bytes:

```
13  0  100  0  191  32  34  116  101  115  116  34  0  0  0  0
```

Los primeros 4 bytes representan 2 números de 16 bits en formato pointer. Los 2 primeros indican la longitud de la línea (13 bytes). Si el interpretador busca una línea determinada (por ejemplo en el caso GOTO) comprueba en primer lugar si el primer número de línea es la correcta. Si no encuentra el destino, se suma la longitud de línea a la dirección actual. De esta manera el interpretador llega a la línea siguiente para comprobarla.

En los siguientes 2 bytes, se encuentra el número de línea. A continuación se encuentra el primer TOKEN; en nuestro caso 191, que representa un PRINT. 32 y 34 son los códigos ASCII para espacio en blanco y comillas respectivamente. Como no es difícil de adivinar los 4 bytes siguientes representan los códigos para "test". Nuestra línea finaliza con el 34 para las últimas comillas. Dado que no se han almacenado más líneas de programa, los últimos 4 bytes contienen un 0.

Esta estructura puede manipularse un poco. Si el interpretador encuentra una línea con el número 0 en el comienzo de la memoria del programa, esto no se listará aunque se ejecute normalmente. Sólo las bifurcaciones a esta línea no funcionan aun cuando no puedan borrarse, ya que el BASIC no acepta un 0 como número de línea. Si se quiere proteger contra listado la primera línea de un programa BASIC solamente han de modificarse los bytes 370 y 371 a ceros mediante POKE. Inténtenlo.

Este procedimiento no puede utilizarse en líneas que no se encuentren al principio del texto BASIC. El número de línea puede ponerse a ceros, pero la línea se listará normalmente.

Posiblemente conoce de otros ordenadores los comandos RENEW u OLD que sirven para reconstruír programas borrados mediante NEW. Esto funciona porque algunos ordenadores (por ejemplo Commodore) no llenan la memoria con ceros sino que borran únicamente los apuntadores para que el interpretador ignore un programa. Nuestro CPC no trabaja con esta técnica, por ello, no puede incorporarse un comando de este tipo mediante el procedimiento Do-it-yourself.

Resumen: Formato de una línea BASIC

Todos las instrucciones y comandos se almacenan como TOKEN, textos y nombres de variables en código ASCII. Los primeros 2 bytes indican la longitud de la línea, los siguientes 2 el número de línea. El número de línea puede modificarse mediante POKE, lo cual permite una protección de listado para la primera línea alterando su número a ceros.

9.2 GARBAGE COLLECTION

¿Ha oído alguna vez algo sobre una recogida de basuras (esta es la traducción de garbage collection)? Si es así seguramente sólo referido a ordenadores, porque lo que se anuncia aquí como un familiar de los basureros es una particularidad muy útil. Para poder comprender esto son necesarios algunos conocimientos más.

Si el interpretador trabaja con variables de string produce 'basuras' en grandes cantidades. Cada vez que se modifica de alguna manera un string (aunque sea únicamente una letra), este se confecciona nuevamente por completo, mientras que la cadena de caracteres antigua queda inalterada y tampoco se sobrescribe.

El string antiguo solamente ocupa memoria sin ninguna utilidad. En algún momento se ocuparán todos los bytes aunque sólo una parte del espacio de memoria sigue conteniendo datos realmente válidos. El resto es basura. Si ahora se quiere confeccionar otro string el ordenador ha de ordenar en primer lugar su memoria. Este procedimiento se denomina 'garbage collection'. Muchos fans de ordenadores lo conocen de sobras porque en muchos casos el ordenador requiere mucho tiempo para ejecutarlo. Un pequeño ejemplo lo aclara mejor:

```
DIM a$(8000)
FOR i = 0 TO 8000: a$(i)=CHR$(1): NEXT i
```

Con estos comandos hemos llenado bastante la memoria. Mediante PRINT FRE ("") podemos provocar un garbage collection (no mediante FRE (0)).
El problema es que la ejecución de esta función a simple vista muy sencilla tarda varios minutos, dado que el interpretador de las 8001 cadenas de caracteres (todas idénticas) ha de borrar 8000. También sería inútil almacenar 8000 strings iguales si una cadena puede servir como descripción de todas las restantes.

Para evitar que uno de sus programas se "cuelgue" temporalmente por una garbage collection, debería incorporar en porciones de programa con muchos strings un FRE ("") de tiempo en tiempo, por ejemplo en forma F=FRE(""). Si únicamente se tienen que quitar pequeñas cantidades de 'basura' el garbage collection no funciona mucho más rápido, pero en este caso no molesta tanto.

9.3 ATENCION: ERROR!

Según la antigua regla "ningún programa sin errores", también el ROM-BASIC contiene un error, que no se percibe con facilidad. Lamentablemente se esconde además en los REM-statements de modo que muchas veces no nos damos cuenta después de una ejecución de programa sin errores.

Si en una línea REM aparece uno de los caracteres de control "flecha a la derecha" (tecla TAB) o "raya vertical" (tecla SHIFT mas @), el interpretador se lía; pueden aparecer errores impredecibles. Con un poco de suerte el error se limita a una bifurcación a línea 32511. Si esta línea no existe se interrumpe a continuación el procedimiento de programa. También es posible que se produzca un borrado de porciones del programa o las líneas correspondientes se hacen invisibles, lo que significa que no puede llegarse ya a ellas mediante GOTO, GOSUB, RUN o LIST, el REM no obstante bifurca aún a las líneas antiguas.

Los resultados del error REM son muy diversos y probablemente también dependen del resto del programa así como de la posición del comando REM. Por ejemplo puede darse también un cambio de la modalidad de pantalla. Posiblemente estos problemas pueden producirse también con otros caracteres de control no mencionados arriba.

Probablemente investigando con mayor profundidad este fenómeno pueda realizarse una segunda protección de listado o resulten - como en el HP 41- comandos sintéticos; instrucciones no previstas por el constructor. Les agradeceré nos comunique sus experiencias referentes al tema.

9.4 COMANDOS DESCONOCIDOS

De forma semejante a los comandos desconocidos del capítulo 2.5 quiero presentar aquí 2 comandos particulares del BASIC y del sistema operativo, que no figuran en el manual.

La primera particularidad se refiere al editor, que puede utilizarse por ejemplo en el manejo del cursor, tecla COPY y entrada de línea. Si ya han entrado algunos caracteres de una nueva línea y quieren tratar y corregir un carácter de una línea antigua mediante EDIT, en primer lugar se debe posicionar el corsor en la posición correcta. Todos los caracteres que teclee ahora se introducen desplazándose los antiguos de forma correspondiente. Esto puede ser muy molesto si se quieren sobrescribir caracteres antiguos. Esta introducción puede desconectarse fácilmente pulsando al mismo tiempo CTRL y TAB. Otra pulsación de las teclas la vuelve a conectar.

La segunda particularidad se refiere al interpretador BASIC. Si se refiere por primera vez a una tabla (por ejemplo A(1)) sin haberla dimensionado previamente, el interpretador BASIC automáticamente realiza un pre-dimensionado con 11 elementos. Esto reemplaza el comando DIM A(10).

Debemos decir que ahorrarse el comando DIM sólo sirve si el campo realmente debe contener 11 elementos. Para cada elemento dimensionado se reserva un espacio de memoria, que no puede ser utilizado por otros datos. Además hace más difícil la lectura de un programa, si de repente aparece un campo que no ha sido expresamente dimensionado con anterioridad.

9.5 MANIPULACIONES DEL BASIC

Como cualquier otro programa el interpretador del BASIC necesita también unos cuantos bytes de memoria para almacenar sus datos internos. Dado que estos bytes están accesibles a través de POKE podemos provocar un caos en el BASIC.

Veamos por ejemplo la protección de programa incorporado en el BASIC. Si se quiere cargar un programa protegido del cassette/diskette, el interpretador lo nota a través de un byte especial. Si se ha llegado al final del programa este byte se comprueba y si se encuentra protegido, se ejecuta automáticamente un NEW. La célula de memoria &AE45 contiene este byte. Si tiene un valor diferente de cero se protege el programa residente actualmente en memoria. Esta protección puede activarse manualmente mediante POKE &AE45,1. POKE &AE45,0 provoca lo contrario.

El byte con la dirección &AC00 contiene otra particularidad útil. Según su contenido al entrar líneas de programa, se borran espacios en blanco sobrantes o se mantienen (caso normal). POKE &AC00,1 activa esta función útil que ahorra espacios de memoria.

En capítulo 3.1 hemos estudiado el procedimiento del comando HIMEM, que reserva un área por encima de la memoria BASIC. De la misma manera puede desplazarse hacia arriba el comienzo del programa BASIC. Para ello no existe un comando específico pero se nos abren nuevas posibilidades.
Las células de memoria &AE81 y &AE82 contienen el apuntador al comienzo del programa BASIC. Normalmente apuntan a la dirección 367, es decir un byte antes del programa. Si alteramos este apuntador la memoria en sí no se altera pero el interpretador busca su programa en otra dirección. El BASIC ignora todas las partes del programa que se encuentren por delante de este nuevo punto de arranque.

De esta manera se pueden esconder partes de programa, conociendo las direcciones donde terminan las diversas líneas.

Es muy fácil hacer desaparecer todo el programa. Los registros &AE83 y &AE84 contienen el apuntador al final del programa. Mediante los comandos

```
POKE &AE81,PEEK(&AE83): POKE &AE82,PEEK(&AE84): NEW
```

el punto de arranque para el interpretador se desplaza en la memoria detrás del programa. El comando NEW es necesario para borrar las variables antiguas que podrían ser mal interpretadas por el BASIC como líneas de programa; nuestro programa no se altera porque los apuntadores se han modificado anteriormente.
Ahora puede cargarse un nuevo programa en la memoria y tratarlo sin influír en el programa antiguo. Solamente se borran las variables.

```
POKE &AE81,111: POKE &AE82,1
```

nos devuelve al estado inicial

Programas escondidos de esta manera tienen una particularidad interesante. Si los comandos para alterar el apuntador, se ejecutan durante el procedimiento de un programa, esto casi no influye en la ejecución de las líneas subsiguientes. Lo que no funciona son los comandos de bifurcación, porque el interpretador utiliza los apuntadores como puntos de orientación para la búsqueda de una línea.

Resumen: Manipular el BASIC.

&AE45 sirve como byte de protección de programa.
POKE &AC00,1 inicializa la modalidad de comprimido, que significa todos los caracteres en blanco sobrantes, se borran (se desactiva mediante POKE &AC00,0). Los bytes &AE81 y &AE82 apuntan al byte antes del comienzo del programa, &AE83 y &AE84 al final del programa.

9.6 OTROS TRUCOS

Los valores de los comandos como SPEED XXXX o similares se olvidan rápidamente, al experimentar con ellos. Conozco por lo menos una solución para ello en el comando SPEED INK. Los dos parámetros de esta instrucción están almacenados por el BASIC en los 2 bytes de dirección &B1D7 y &B1D8, de donde los extrae el sistema operativo cuando los necesita (en cada cambio de color). Lo que puede hacer el sistema operativo fácilmente podemos hacerlo nosotros desde el BASIC mediante el comando PEEK.

Seguramente ya habrán utilizado caracteres definidos por ustedes mismos. Posiblemente les haya ocurrido lo mismo que a mí, que me cansé de recalcular cada vez los 8 bytes de una matriz de caracteres, aunque solo se quiera modificar un único punto. En este caso también hay una solución. Los caracteres definibles CHR$(240) hasta CHR$(255) están almacenados en el RAM desde &AB80 hasta &ABFF. Para cada carácter hay 8 bytes reservados que pueden leerse fácilmente mediante PEEK. De esta manera solamente debe calcularse nuevamente el byte que contiene el punto a alterar.

La dirección de un carácter puede determinarse mediante la siguiente fórmula:

Dirección = 43904 + (X - 240) * 8

X representa el número del carácter deseado en el área de 240 hasta 255.

Resumen: trucos para el sistema operativo.

Los parámetros del comando SPEED INK pueden extraerse de los bytes &B1D7 y &B1D8.
Las matrices de caracteres de los caracteres definibles se encuentran en el área &AB80 hasta &ABFF.

10. ACCESORIOS Y SU FUNCIONAMIENTO

Cada ordenador necesita accesorios (periféricos) para poder trabajar debidamente. Algunos constructores como por ejemplo IBM suministran cada periférico separadamente, como ocurre en el caso del IBM PC con respecto al teclado y el monitor. Afortunadamente el CPC es diferente, ya que trae incluso su cassette/diskette y un monitor incorporados en su configuración base. También para el CPC pueden adquirirse más accesorios.

10.1 EL FLOPPY

¿Pertenece Ud. a los impacientes para los cuales es excesivo el tiempo de carga de un programa desde el cassette? En este caso necesita un floppy. Los conocedores de los ordenadores hace tiempo saben la enorme facilidad de trabajo incluso de un floppy lento a diferencia de un cassette. Una pequeña comparación de tiempos nos lo puede demostrar. Para cargar un programa de 20 K, el cassette con SPEEDLOAD necesita 2 minutos 27 segundos mientras que el floppy lo hace en 9 segundos.

Esta no es la única ventaja de un dispositivo floppy. Dado que en un diskette las diferentes pistas magnéticas (hay 40) pueden accederse directamente como en un tocadiscos mediante el movimiento del brazo y de la cabeza de lectura, es posible utilizar el llamado procedimiento de acceso directo. Esto significa que el ordenador no tiene que buscar en todos los ficheros desde el principio para encontrar los datos correctos, sino casi puede decirse "extrae el 3er. byte del fichero de direcciones". El ordenador en principio solo acepta la traducción BASIC original de nuestra instrucción pero esta la ejecuta a alta velocidad. De esta manera el diskette casi puede utilizarse como una ampliación de memoria.

Un floppy tiene además otras ventajas (por ejemplo existe un índice con todos los ficheros almacenados en el diskette) pero también pequeñas desventajas. En primer lugar es muy caro y además necesita un circuito electrónico especial. Este controlador se suministra con el primer floppy que se compre (y por ello es asimismo más caro). El circuito puede controlar 2 dispositivos floppy y por ello puede conectar el segundo al primero. Sin controlador no funciona nada.
La caja de la electrónica de control contiene además un ROM que amplía la sentencia de comandos de su CPC por los comandos necesarios de diskette.

10.2 LA IMPRESORA

Una de las múltiples ventajas de su ordenador es la interfase de impresora incorporada. A diferencia del floppy no necesita software de ampliación y tampoco controlador, todo esto ya está incorporado. Se trata de una interfase llamada CENTRONICS, que significa que está adaptado a las interfases del productor de impresoras CENTRONICS. La mayoría de los productores han optado por esta interfase. Por ello pueden conectar directamente la mayoría de las impresoras corrientes del mercado. Nos encontramos nuevamente con un pequeño problema. El CPC transmite únicamente los 7 bits inferiores del código ASCII de 8 bits y de esta manera les "roba" la mitad de los caracteres representables. Esto les molestará menos de lo que piensan porque los códigos 0 hasta 127 contienen todos los caracteres importantes y los códigos perdidos 128 hasta 255 en la mayoría de los casos están ocupados con caracteres gráficos.

Muchas veces ocurre que el código ASCII de la impresora no coincide con el del ordenador. En este caso tiene que programar una tabla de adaptación. Esto suena más difícil de lo que realmente es. Introduzca los códigos de la impresora que correspondan a los caracteres del CPC uno por uno en una tabla (0 hasta 127) de variables enteras. Si ahora se quieren imprimir datos se deben enviar las cadenas de caracteres letra por letra mediante PRINT #8,CHR$(X). X es el código ASCII del caracter deseado. Si se quieren enviar los códigos corregidos debe utilizarse en lugar de X, la expresión ARRAY (X). En este caso no se imprime el código del ordenador sino el código alternativo de la tabla.

10.3 EL JOYSTICK

Como muchos usuarios probablemente habrá intentado alguna vez confeccionar un programa con comprobación de JOYSTICKS. En la mayoría de los casos habrán valorado las diferentes posiciones mediante construcciones IF-THEN, por ejemplo de la siguiente forma:

```
IF JOY(0) = 1 THEN 100
IF JOY(0) = 2 THEN 200
IF JOY(0) = 4 THEN 300
.....
```

Este método es muy lento y laborioso. Mejor resultado se obtiene con el siguiente programa (explicación abajo):

```
10 A = LOG(JOY(0)) / LOG(2)
20 ON A GOTO 100, 200, 300, ....
```

Línea 20 bifurca a diferentes porciones del programa según el valor de la variable A. Si se asignara a la variable el valor JOY, el comando ON no funcionaría correctamente porque JOY no suministra valores consecutivos para las diversas direcciones (como 1, 2, 3, etc), sino sus potencias en 2 (1, 2, 4, 8, 16, 32). De estas potencias se calcula el logaritmo para la base 2, que a continuación genera los números deseados.

El funcionamiento de un JOYSTICK es muy simple. Se compone de 5 ó 6 teclas, 1 ó 2 se utilizan como tecla de disparo, las restantes están posicionadas debajo de la palanca de mando en las 4 diferentes direcciones de movimiento. Según la posición de la palanca se activa la tecla o el pulsador correspondiente -el CPC lo registra y proporciona el valor correspondiente.
Existen naturalmente diferencias entre los diversos JOYSTICKS que ofrece el mercado. Ejemplares simples y baratos trabajan con contactos al pulso (como las conocen los antiguos propietarios del ZX81), otros trabajan con micro-switches que se hacen notar muchas veces por un pequeño 'click'.
Al comprar un JOYSTICK debería tener en cuenta que el mismo tenga cantos redondeados, porque de lo contrario se cansa uno muy pronto al jugar. Se puede decir que todos los JOYSTICKS ATARI compatibles funcionan con el CPC.

11. ALGO SOBRE INTERFASES

En la parte posterior del CPC se encuentran diversas conexiones y conectores. Alguna vez se habrá preguntado como funciona todo esto y que se esconde detrás del ordenador, esto lo explicaremos en este capítulo.

11.1 PEQUEÑO INVENTARIO DE INTERFASES

Antes de ocuparnos de las diversas conexiones deberíamos aclarar el significado de INTERFASE.

Una interfase representa una conexión hacia un periférico. Las conexiones internas del ordenador, se 'cortan' en la parte posterior, donde deben conectarse los cables de los periféricos que establecen de esta forma la comunicación entre los mismos y el interior del ordenador. Debido a estos 'cortes' es posible el acceso al 'corazón' del ordenador. El mejor ejemplo es el "Expansion Connector", que se denomina por la parte posterior del ordenador "Floppy Disk". Se trata en realidad de una conexión de ampliación que no sólo establece las conexiones necesarias hacia el floppy sino que también tiene previstas las conexiones de módulos ROM adicionales. Los ROM son componentes de memoria y por ello pertenecen al área de acceso inmediato del procesador.
Debido a esto el conector de expansión se compone "solamente" del bus de direcciones, el bus de datos y el bus de control del Z80 además de algunas señales de control adicionales.

Junto a este se encuentra el conector de impresora, que se adapta a la norma CENTRONICS. El puerto de impresora a su vez esta conectado al bus de datos del procesador pero no con el bus de direcciones ni el de control. En lugar de esto últimos, tiene algunas líneas de control del componente de puerto 8255.

Es lamentable que justamente en este caso los diseñadores del CPC

comenzaran a ahorrar. En lugar de la transmisión tradicional de 8-bits en el caso del CPC se ha omitido el octavo bit (¿pero dónde está el ordenador perfecto?)

La conexión de JOYSTICK solamente es una ampliación del teclado, el que está en verdad conectado con diversos componentes del ordenador (explicación más adelante).

También las conexiones para el monitor y amplificador de estéreo cuentan bajo la denominación común de interfases, aunque no trabajan directamente con el procesador.

11.2 COMO FUNCIONA UNA INTERFASE

El funcionamiento de una interfase generalmente es igual en todos los ordenadores. El procesador envía la información a través del componente interfase, que se ocupa de llamar al dispositivo periférico al que se transmiten los datos. Para ello puede ser necesario sincronizar ambos dispositivos. En este caso se intercambian impulsos a través de líneas de control especiales que indican la preparación para la transmisión.

Antes de la transmisión de datos el procesador debe indicar si se quieren recibir o transmitir datos. Según la modalidad deseada el puerto se conmuta para entrada o salida.

En algunos casos no se utilizan componentes de interfase especiales, lo que también ocurre con el CPC. En este caso el propio procesador ejecuta todas las operaciones necesarias (lo que naturalmente se ejecuta algo más lento). El Hardware se limita en ese caso a la adaptación de las diferentes señales.

Los programas necesarios para el manejo de un interfase ya se encuentran almacenados en el ROM y pueden utilizarse mediante

comandos BASIC; una programación eficiente de interfase (con pocas excepciones) solamente es posible en lenguaje máquina.

11.3 SU INTERFASE PERSONAL

¿Cuenta usted entre los inmejorables especialistas de Hardware que piensan siempre en modificaciones, soldadores calientes y montajes varios? Los representantes de esta especie de 'Homo-electronicus' muchas veces desean leer datos de dispositivos propios (por ejemplo termómetro, u otros). Para estas tareas se ofrece teóricamente el interfase CENTRONICS y el conector de expansión, que en la práctica tienen otro destino.

Lo mismo puede decirse para el conector del JOYSTICK, que normalmente se utiliza únicamente para juegos, por ello podemos utilizarlo asimismo para aplicaciones serias.

El 'User-port' (como lo denomina el letrerito de la parte posterior) puede controlarse muy bien desde el BASIC mediante las funciones JOY e INKEY. De esta forma se han dado todas las condiciones previas para el contacto del ordenador hacia el mundo exterior.

En los Pins 1 a 7 (vean manual, anexo V) pueden conectarse 2 interruptores en cada uno. El primer interruptor de cada pin se conecta con COMMON (pin 8), el segundo con COM2 (pin 9). Con estos últimos pins el ordenador diferencia entre JOYSTICK 0 y 1. Al cerrar un contacto la entrada (1 hasta 7) se conecta a uno de las señales COMMON. Para el CPC esto es lo mismo que una pulsación de tecla o una acción del JOYSTICK, lo que puede comprobarse mediante los comandos mencionados.

A las entradas del puerto del JOYSTICK pueden conectarse tantos interruptores como se deseen. Queda a su criterio que dispositivos utilizar (relé, transistores, etc.).

11.4 LECTURA DEL TECLADO

Para completar la información explicaremos en este apartado como funciona la lectura del teclado.
Como pueden comprobar al contar las teclas de teclado del CPC, este se compone de 73 teclas (SHIFT solo se cuenta una vez), que se comprueban en intervalos fijos de tiempo (1/50 segundo). Para ello el teclado está repartido eléctricamente en 10 columnas, que puede activar el Z80 una por una, enviando solamente el número de la columna al componente puerto 8255.

Si hay una tecla pulsada de la columna seleccionada se desactiva un bit a ceros, de lo contrario queda en 1. De esta manera obtenemos hasta 8 bits por columna componiendo un byte entero, que será devuelto por el sound chip (sí, el chip de sonido) a través del 8255 al Z80. El procesador puede comprobar fácilmente mediante los bits desactivados, cuales teclas de la columna están pulsadas. Este método a través del soud-chip que a primera vista parece algo complicado, se ha elegido porque este componente está equipado con un puerto adicional y no era conveniente pasar la lectura del teclado que es relativamente lenta a puertos de acceso más directos.

Este principio de la lectura a través de columnas se utiliza en casi todos los ordenadores con pequeñas diferencias.

12. CASSETTE Y TECLADO

El manejo del cassette se ha tratado muy brevemente en el manual del CPC. Queremos por esto insistir ahora en ello así como en la lactura del teclado, que puede ofrecer posibilidades desconocidas.

12.1 COMO SE CONFECCIONAN FICHEROS

Ya habrá leído en el manual del CPC la descripción del tipo de fichero "ASCII". Lamentablemente no se ha explicado que no solamente pueden generarse listados para programas de tratamiento de texto (mediante SAVE 'nombre',A), sino que el CPC también tiene capacidad de almacenar variables de string y variables aritméticas en el cassette. Quiero asimismo enseñarles la forma de leer listados ASCII desde el BASIC.

El nombre de 'fichero ASCII' proviene de que todos los datos están almacenados como secuencia de códigos ASCII (o bytes). La única diferencia con los ficheros de programa estriba en que no se copia simplemente una porción de memopria en la cinta, sino que las diversas entradas se acaban con CHR$(13). No se diferencia si los datos provienen de variables o no.

La grabación y lectura de estos ficheros se asemeja mucho con la salida de pantalla; para ello existen los comandos PRINT ≠9 e INPUT ≠9. También en la salida a pantalla se utiliza el CHR$(13) para la separación de las diversas informaciones. El carácter de control en este caso indica al sistema operativo que se debe mover el cursor a la siguiente línea.

Supongamos que se han llenado 2 variables con datos que se desean mantener hasta el dia siguiente. En este caso se pueden aplicar los ejemplos de programa que siguen:

```
10 REM CONFECCIONAR FICHERO
20 OPENOUT "TEST"
30 PRINT #9, A$
40 PRINT #9, B
50 CLOSEOUT
```

```
10 REM RELEER FICHERO
20 OPENIN "TEST"
30 INPUT #9, A$
40 INPUT #9, B
50 PRINT A$, B
60 CLOSEIN
```

El primer programa graba las variables A$ y B en el cassette. Con el segundo programa se pueden leer los contenidos de ambas variables. Los nombres de variable no son significativos, pueden utilizar X$ e Y por ejemplo. En cambio los tipos de variable deben ser correctos, ya que de lo contrario puede ocurrir un 'TYPE MISMATCH ERROR'.

Al grabar listados de ASCII se generan cadenas de caracteres, que puede leer el BASIC en forma de strings. Como ejemplo presentamos el siguiente programa:

```
10 INPUT "CUANTAS LINEAS?"; A: A=A-1
20 DIM A$(A)
30 OPENIN "NOMBRE FICHERO"
40 FOR I=0 TO A
50 INPUT #9, A$(I)
60 NEXT
70 CLOSEIN
```

Después de ejecutar este programa las líneas de un listado ASCII se encuentran en una tabla de string, donde puede continuar su tratamiento. Lamentablemente todo el sistema tiene un problema. El BASIC utiliza también comas para la separación de strings. Por ello todas las líneas que contienen una coma (pueden ser muchas) se separan en la posición de la coma y el resto de la línea ocupa un segundo string. Esto puede solucionarse mediante programación apropiada, es decir utilizando en vez del comando INPUT el comando LINE INPUT. Esta forma especial solamente reconoce un CHR$(13) como marca de final de línea.

En el ejemplo anterior se debían indicar manualmente el número de líneas que contenía el listado. Si se hubiera equivocado en este extremo, se podría producir un mensaje "EOF met", si el programa intentara leer más datos que los presentes. Este error puede evitarse ya que en el BASIC existe una función que indica el final de un fichero, el EOF (End of File). PRINT EOF proporciona el valor cero si quedan aún datos, al final del fichero el resultado es -1. Nuestro programa por ello puede modificarse de la siguiente manera:

```
10, 20, 30 Y 70 no se modifican
40 WHILE NOT (EOF)
50 LINE INPUT #9, A$(I): I=I+1
60 WEND
```

Mediante la construcción WHILE-WEND se leerán todos los datos presentes en el fichero.
Lamentablemente puede ocurrir un SUBSCRIPT OUT OF RANGE ERROR si el listado se compone de más líneas que los strings dimensionados.

En la confección de ficheros puede grabarse en primer lugar la cantidad de datos que contendrá en forma de variable normal seguida por el fichero real de datos. Cuando se lea este fichero, podrá utilizarse el primer dato (número de datos) para dimensionar los campos y continuación se leerán los datos evitando el error antes mencionado.

Las 32 posiciones de memoria en las áreas &B807 hasta &B816 y &B84C hasta &B85B nos ofrecen una aplicación muy exótica. Aquí el sistema operativo almacena los nombres de los ficheros INPUT (B807) y de los ficheros OUTPUT (B84C) en forma de cadenas ASCII. Mediante la siguiente línea podemos leer el último nombre de fichero OUTPUT:

```
FOR I=&B84C TO &B85B: PRINT CHR$(PEEK(I)); : NEXT
```

¿Qué les parece si equipamos los programas con una rutina de comprobación que evite la modificación del nombre?. Para ello hemos de leer el nombre del fichero de la forma indicada al principio del programa. Si la comprobación es incorrecta borraremos el programa mediante NEW, acompañado tal vez por un comentario.

Para saber en que velocidad de grabación se está trabajando basta teclear PRINT PEEK(&B8D1). Si el resultado es 6, la grabación es lenta, si es 12 se ha indicado SPEED WRITE 1.

Resumen: Ficheros en cassette

Con LINE-INPUT pueden leerse también strings conteniendo comas. La función EOF indica el final del fichero. Mediante ambos comandos es posible fácilmente tranferir listados ASCII a tablas BASIC para tratarlos posteriormente. De las posiciones de memoria &B807-&B816 y &B84C-&B85B puede extraerse el nombre del último fichero. El byte &B8D1 indica la velocidad de grabación.

12.2 INKEY DE DIFERENTE FORMA

Es probable que al confeccionar su primer juego BASIC necesite una forma de lectura del teclado por la cual se registren pulsaciones múltiples de tecla.

Afortunadamente existe para esto una función BASIC. INKEY (X) (sin el "$") indica si la tecla X está pulsada. Dado que esta función trabaja independientemente del buffer de teclado no se devuelve el primer carácter del buffer sino únicamente se comprueba el contacto eléctrico de la tecla. Como ejemplo para la aplicación podría servir el listado siguiente:

```
10 CLS
20 IF INKEY(69) = 0 THEN LOCATE 1,5: PRINT "Tecla a pulsada"
30 IF INKEY(36) = 0 THEN LOCATE 1,10: PRINT "Tecla l pulsada"
40 LOCATE 1,5: PRINT SPACE$(20): REM Borrar línea 5
50 LOCATE 1,10: PRINT SPACE$(20) :REM Borrar línea 10
60 GOTO 20
```

Arranque el programa y pulse a un tiempo las teclas a y l, comprobará que aparecen ambos mensajes en la pantalla. En una programación similar con INKEY$ esto no sería posible, porque con INKEY$ sólo se lee un carácter.

Mencionaremos un pequeño truco para la lectura del teclado. En muchos programas se incorporan bucles especiales que interrumpen la ejecución del programa hasta que el usuario pulsa una tecla para continuar. Un bucle de este tipo puede ser el siguiente:

```
WHILE INKEY$="" : WEND
```

Lo mismo hace una rutina en lenguaje máquina en el ROM que puede llamarse mediante CALL &BB18.

Resumen: Lectura del teclado

El comando INKEY puede utilizarse para la lectura de varias teclas a un tiempo.
CALL &BB18 espera, hasta que se pulsa una tecla cualquiera.

13. INTRODUCCION AL LENGUAJE MAQUINA DEL Z80

En muchas publicaciones se encuentran programas para copiar escritos en lenguaje máquina, un lenguaje que al principiante le resulta un total misterio. Debemos decir que el lenguaje máquina no es tan sencillo como el BASIC, pero en cambio trabaj mucho más rápido y ofrece al programador experto incalculable cantidad de posibilidades. Ofrecemos aquí la base para la programación en este lenguaje. Después de la lectura del presente capítulo serán capaces de comprender el funcionamiento general de programas en lenguaje máquina y podrán decidir si desean profundizar en este lenguaje. Si no les atrae el lenguaje máquina, no importa, los conocimientos adquiridos podrán utilizarlos para otras tareas y tanto el PASCAL como el LOGO tampoco son inadecuados para hacer funcionar su ordenador.

13.1 ¿QUE ES EL LENGUAJE MAQUINA?

Como ya saben el lenguaje máquina es la única posibilidad de programar el procesador sin necesidad de compilador ni de interpretador. Por ello este lenguaje permite tan inmensas velocidades de ejecución. El lenguaje máquina contiene diversos comandos de los cuales pueden componerse todas las operaciones complejas del BASIC o de otros lenguajes. Los comandos máquina pueden dividirse en 3 grupos. Los comandos más fáciles de comprender para un programador BASIC son los comandos de bifurcación con los cuales es posible bifurcar dentro de la memoria de forma similar que con GOTO y GOSUB. Otros comandos provocan manipulaciones de datos (por ejemplo adiciones, interrelaciones, etc.). El último grupo es el de operaciones que

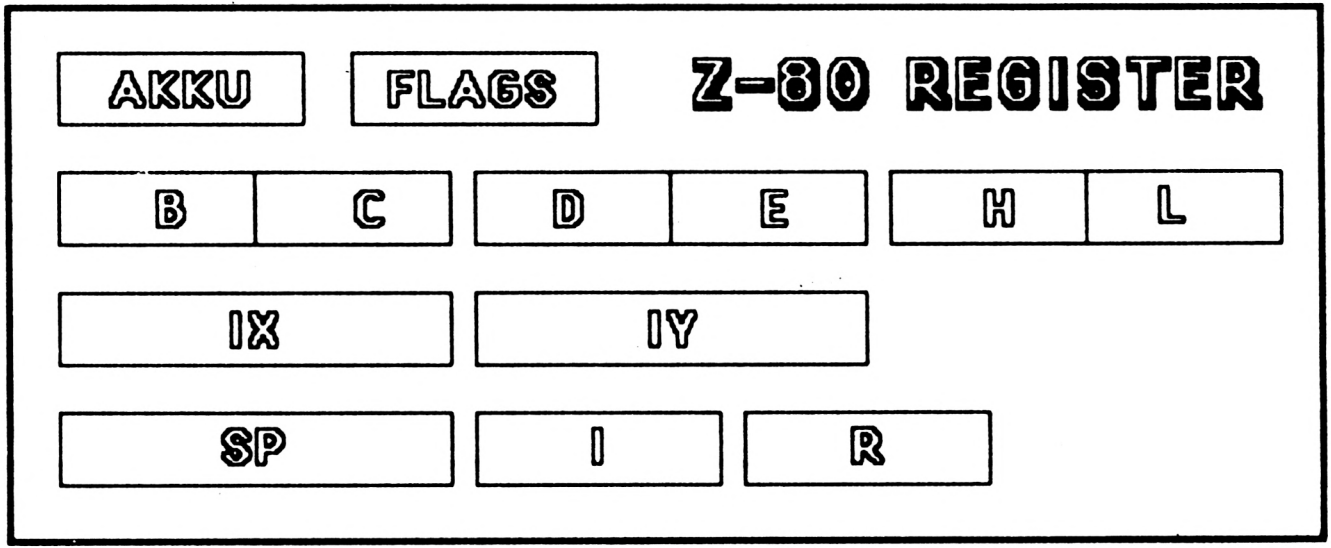
desplazan los datos dentro de la memoria de una posición a otra. Podemos decir fundamentalmente que para el microprocesador no existen variables, sólo reconoce las células normales de memoria y los registros internos. El programador ha de ocuparse de diferenciar entre datos y bytes de programas. En general las manipulaciones de datos sólo pueden realizarse en los registros internos.
Un comando en lenguaje máquina se compone de un código de operación (OP CODE), que puede decirse indica el "número" del comando. Este OP-CODE puede componerse de un máximo de 3 bytes, y además puede continuarse con un máximo de 2 bytes de datos. De esta manera los comandos del Z80 teóricamente tienen una longitud de hasta 5 bytes. En la práctica sólamente son 4, porque la utilización de OP-CODE de 3 bytes está reservada únicamente a comandos con un máximo de 1 byte de datos.

13.2 EL CICLO

Todos los componentes del ordenador se orientan por un pequeño y humilde cuarzo que marca el ciclo de proceso (4 megahercios= 4000000 pulsaciones o ciclos por segundo), necesario para sincronizar los diferentes componentes integrados. Si no se realizara esta tarea podría pasar por ejemplo que un componente de memoria enviara datos al procesador sin que el mismo estuviera dispuesto para recibirlos. Incluso un microprocesador tan rápido necesita un poco de tiempo para el tratamiento de datos.

13.3 LA ARQUITECTURA DEL Z80

Todo microprocesador posee registros internos en los cuales se ejecutan las operaciones. El registro más importante es el llamado acumulador en el cual se ejecutan la mayoría de las interrelaciones aritmáticas y lógicas.



akku = acumulador Dibujo 9
Z-80 Register = Registros Z-80

El acumulador (abreviado ACU o A) es un registro de 8 bits, que puede contener y tratar 1 byte (realmente el ACU mismo no trabaja sino que almacena resultados). La mayoría de los comandos aritmáticos necesitan 2 operandos (p. ej. la adición de 2 números). El primer operando se encuentra en el ACU antes de la ejecución del comando, el segundo proviene de otro registro del procesador o de la memoria. Después de una adición el ACU contiene el resultado.

Otro registro que lleva el nombre F, almacena diferentes flags, que reflejan determinados estados del procesador. Mediante los flags puede comprobarse por ejemplo si el contenido del ACU es cero. Existen otros 6 registros de 8 bits, con una particularidad especial. Cada 2 de estas células de memoria componen un registro de 16 bits. La pareja HL es casi un acumulador de 16 bits, que significa que se ocupa de las mismas tareas que el acumulador normal, en este caso para 16 bits en vez de 8 y de esta manera es más fácil el tratamiento de números con mayor cantidad de dígitos. Otras parejas de registros son BC y DE. IX e IY son 2 registros de índice (con 16 bits cada uno), que señalan como apuntador a determinadas células de la memoria. Mediante estos apuntadores pueden manipularse fácilmente grupos enteros de datos, el procedimiento lo explicaremos más adelante.

El registro de 16 bits SP tiene una tarea especial, apunta al elemento superior de la pila (SP significa STACK POINTER). Cuando se almacena o extrae algo del Stack el ZZ80 actualiza el apuntador de modo que apunte a la nueva posición.

Finalmente quedan los registros I y R reservados para el control del Hardware.

Para todos los registros A hasta L existe un segundo registro que puede intercambiarse con el registro original. Sólo es posible trabajar con un grupo de registros cada vez y por ello en la mayoría de los casos los registros secundarios sirven como memoria temporal. Estos registros de reserva se marcan en los comandos con un apóstrofe después del comando (en este libro por razones técnicas con comillas).

Ya conocen los registros disponibles del Z80 para el lenguaje máuina. En el apartado siguiente explicaremos el procedimiento de un comando máquina dentro del procesador.

13.4 EL FUNCIONAMIENTO DEL Z 80

Supongamos que en la memoria de su ordenador reside un programa en lenguaje máquina esperando su ejecución. De alguna forma el microprocesador ha de enterarse dónde se encuentra este programa. Para ello existe un registro especial de 16 bits en el Z80, llamado contador de programa (inglés 'Programm Counter' = PC), que contiene la dirección del siguiente comando a ejecutar. Si se desea ejecutar este comando el procesador extrae el byte de la dirección de memoria indicada. Este byte se mantiene en el procesador y el contador de programa se incrementa en 1 para obtener la dirección del byte siguientes. Al mimso tiempo se decodifica el OP-CODE (de este se trata en el byte) lo que significa que el Z80 comprueba de cual de los múltiples comandos se trata. Algunos comandos tienen un OP-CODE, compuesto de varios bytes (de lo contrario sólo podrían reconocerse 256 comandos). En este caso se extraen los bytes necesarios uno por uno de la memoria, igual que el primero (el PC siempre apunta a la posición actual de memoria porque se incrementa después de cada byte). También es posible que se continúe con uno o 2 bytes de datos los cuales tambieén se extraen de la memoria sin necesidad de decodificarlos. Estos bytes llegan a determinados registros (por lo cual ya podría haber finalizado el comando) o quedan disponibles en alguna parte del procesador para su tratamiento.
Si aún hay que modificar los datos (por ejemplo mediante adición u otros) esta operación tiene lugar en este momento y el resultado se almacena nuevamente (por ejemplo en el ACU). Finalizado de esta forma el comando puede arrancarse el siguiente comando.

Todos estos procedimientos necesitan un poco de tiempo. En el caso normal cada uno de los pasos descritos como "extraer OP-CODE", "decodificar OP-CODE", "ejecutar comando" y "almacenar resultado" necesita un ciclo de reloj. Por ello muchas veces se unen 4 de estos ciclos para un ciclo de máquina. Los comandos más complejos pueden necesitar para su ejecución varios de estos ciclos de máquina.

Resumen: El PC siempre apunta a la posición de memoria que contiene el siguiente byte a trabajar. OP-CODES y datos se leen uno por uno. El OP-CODE se decodifica y a continuación se ejecuta el comando.

13.5 SISTEMA HEXADECIMAL

Siempre que se trabaja en lenguaje máquina se encuentran representaciones de números en sistema hexadecimal. Este sistema a diferencia de nuestro sistema decimal posee 16 símbolos (0-9 y A-F para los valores 10 a 15). Se utiliza tanto porque la conversión de binario a hexadecimal es muy sencilla. Otra ventaja del sistema hexadecimal es que un número Hex. representa exactamente medio byte (el valor más elevado representable con 2 posiciones es hexadecimal FF, en combinación binaria 1111 1111 que es el mayor valor que puede contener un byte). Cada vez se utiliza medio byte transformándolo en un dígito hexadecimal. La tabla siguiente muestra valores de conversión:

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

Del byte 1010 1011 obtenemos el número hexadecimal AB (porque 1010=A y 1011=B). Naturalmente esto funciona asimismo a la inversa.
Para la conversión de números hexadecimales en números decimales se tranforma cada dígito a su equivalente decimal. El dígito del extremo derecho se multiplica por 16^0=1, el siguiente por 16^1=16, el tercero por 16^2=256, etc. Los productos obtenidos se suman proporcionando el resultado decimal. Un ejemplo:

ABCD (los dígitos son 10,11,12,13)
= 10*16^3 + 11*16^2 + 12*16^1 + 13*16^0
= 10*4096 + 11*256 + 12*16 + 13*1
= 43981

Para la conversión inversa a la anterior (dec-hex) puede dividir el número decimal entre 16 y anotar el resto obtenido de la división. El resultado nuevamente se divide entre 16, etc hasta

que arroje un resultado 0. Un ejemplo:

```
  207 / 16 =   12 Rest 15 -) F
   12 / 16 =    0 Rest 12 -) C
=) 53000(10) = CF08(16)
```

Existen calculadoras que tienen funciones especiales para la conversión de la base. Un Assembler o un Hexmonitor potentes ofrecen también esta función.

El CPC posee asimismo una función BASIC que permite el tratamiento de números hexadecimales, indicando el número hexadecimal mediante un ampersand "&" (para ser exactos &H) delante del número y queda tranformado el mismo al sistema decimal.

13.6 ARITMETICA BINARIA

13.6.1 ADICION

Para empezar hemos de decir que la adición binaria funciona exactamente como la adición decimal aunque con otro sistema de numeración.
La suma de 2 ceros o de un cero y un 1 (indiferente del orden), no necesitan explicación, se suman normalmente. Si en cambio queremos sumar 1 y 1, hay un problema, en decimal el resultado sería 2, pero el 2 no existe en el sistema binario, por lo cual debemos hacer un desplazamiento a la posición siguiente (como en el caso del 9 en el sistema decimal):

```
  0    0    1    1
 +0   +1   +0   +1
---  ---  ---  ---
  0    1    1   10
```

Igualmente de fácil es la interrelación de bytes enteros. Se suman simplemente cada posición (teniendo en cuenta eventualmente un desplazamiento):

```
  01101101 = 109
 +00001001 =   9

     1  1     (desbordamiento)
-----------   -----
  01110110 = 118
```

Para comprenderlo mejor hemos señalado el desbordamiento producido.
Si se deben sumar 2 unos y además tenemos un arrastre anterior (1+1+1=3) el resultado es 11 (claro)

Intente la siguiente adición:

```
 10010011
+11011111

1  11111   (desbordamiento)
----------
101110010
```

Tenemos ahora 9 bits en el resultado. El noveno bit se denomina 'Carry' o bit de desbordamiento, e indica que la adición de 2 números de 8 bits ha superado el valor posible para un byte (0-255) lo que nos lleva a la adición de 16 bits. Ningún ordenador tiene suficiente con 8 bits para la representación numérica porque en la mayoría de los casos los valores pueden ser mayores. Es por esto que un procesador de 8 bits (como el 6510) solamente puede tratar 8 bits a un tiempo. Si el número se compone por ejemplo de 2 bytes, la adición se debe realizar uno a uno. Excepto el bit CARRY la adición se realiza con las dos partes del número de forma independiente, el CARRY se necesita solamente para tratar números mayores. Su tarea es almacenar temporalmente el desbordamiento de la última posición del primer byte hacia la primera posición del segundo byte. Un ejemplo:

```
 00110101 10010011
+10011011 11011111

 1111111   11111   (desbordamiento)
------------------
11010001 01110010
```

La parte de la derecha de la adición ya la conoce del ejemplo anterior.

13.6.2 SUSTRACCION

Cuando el ordenador debe sustraer una cantidad de otra, en primer lugar busca el equivalente negativo de la segunda cantidad (es decir multiplicando por -1) y a continuación los suma. Se procede de esta manera porque una adición y una negación pueden componerse con circuitos electrónicos (como AND, OR, XOR, NOT) pero no una sustracción.
Para representar un número negativo, se desplaza el área de 1 byte de 0-255 hacia -127 hasta +127. El bit de valor máximo (bit 7) actúa como signo del número. Si está en 1 se trata de un número negativo y si es 0 el byte es positivo. Para la negación no es posible activar únicamente el bit 7. Un ejemplo explica este problema:

```
  00000001
+ 10000001
-----------
  10000010
```

En sistema decimal, esta operación sería 1 + (-1) = -2, lo que no proporcionaría el resultado correcto, hemos de buscar otro camino. Un byte puede multiplicarse por -1 simplemente operando con el concepto de "complemento a dos". De esta manera se invierten todos los bits y se suma 1.

```
Un ejemplo:                 01011011
          inversión         10100100
                          +        1
                          -----------
                            10100101
```

Utilizando este esquema calculando en sistema binario 1 - 1, se obtiene el resultado correcto:

```
                            00000001
                          + 11111111
                            11111111      (desbordamiento)
                          -----------
                           100000000
```

Como pueden ver aparentemente se produce un desbordamiento, pero también en esto la sustracción se comporta de forma diferente. En nuestro caso podemos simplemente ignorarlo. Si sustrayéramos 16 bits nuestro carry bit sobrante se ocuparía de que las posiciones del segundo byte se pusieran a ceros. Esto es importante porque en el caso de números negativos de 16 bits se invierten las 16 posiciones. Como número de 2 bytes el -1 se representaría así: 11111111 11111111. Si faltara el bit carry nuestro resultado sería 11111111 00000000 lo que es incorrecto.
Afortunadamente la programación de una sustracción no es tan complicada, porque los comandos de sustracción del Z80 ya incluyen la conversión al complemento a 2.

13.6.3 MULTIPLICACION

Aunque le parezca increíble, el lenguaje máquina del Z80 sólo posee 2 comandos de cálculo que son la adición y la sustracción. Las restantes operaciones aritméticas se confeccionan utilizando estos comandos básicos, en la mayoría de los casos como subrutinas.
Ya que no queremos profundizar demasiado en el lenguaje máquina (para ello hay mejor y más amplia literatura), les ofrecemos únicamente el algoritmo más simple para la multiplicación, que los profesionales no suelen utilizar dada su escasa eficiencia. Para calcular el producto de X * n basta sumar X n veces. Naturalmente sólo funciona con números enteros. Para fracciones decimales hay procedimientos más complicados en los cuales se interrelacionan los números por ejemplo posición por posición y no como enteros, pero el principio funciona de forma similar.

Para comprenderlo mejor veamos este ejemplo:

4 * 3 = 4 + 4 + 4 = 12

13.6.4 DIVISION

También para la división existe un procedimiento muy simple. Para dividir X por n se resta sucesivamente n de X. La cantidad de sustracciones posibles hasta que n sea mayor que X, nos proporciona el resultado de la división.
Veamos este ejemplo:

```
    10 / 3 = ?
    10 - 3 = 7      Registro de cuenta   = 1
     7 - 3 = 4      Registro de cuenta   = 2
     4 - 3 = 1      Registro de cuenta   = 3
=) 10 / 3 = 3  Resto = 1
```

Estos métodos pueden utilizarse porque el lenguaje máquina permite velocidades inmensas. Las calculadoras de bolsillo funcionan con el mismo principio. Cada vez que se pulsa una tecla de cálculo se ejecuta un pequeño programa máquina (naturalmente con los mencionados laboriosos algoritmos).
De las cuatro operaciones básicas de cálculo pueden asimismo componerse funciones más complejas (por ejemplo potencias, senos, etc.). De esta manera puede expresarse cada operación matemática mediante minúsculas operaciones AND, OR, XOR y NOT (porque pueden construirse la adición y la sustracción a partir de estas operaciones).

13.7. ¿COMO FUNCIONAN LAS COMPARACIONES?

En BASIC las comparaciones no representan nada extraordinario. ¿Pero cómo pueden generarse en lenguaje máquina? Veamos en principio un ejemplo:
A=B (=) A - B = 0
Como pueden ver una comparación de 2 números (A y B) puede transformarse de manera muy fácil. Para el ordenador esta forma tiene la ventaja de que aparece un cero en el lado derecho de la

ecuación. El cero es el único valor que el microprocesador puede comprobar si está o no contenido actualmente en el registro de cálculo interno (llamado ACU). Para ello se interrelacionan simplemente todos los bits mediante OR -por ejemplo de esta manera:

Bit 7 OR Bit 6 OR Bit 5 OR Bit 4 OR Bit 3 OR Bit 2 OR Bit 1 OR Bit 0

Si todos los 8 bits del ACU contienen un 0 el resultado de esta cadena de interrelaciones es 0, de lo contrario es 1 (lo que significa que hay por lo menos un bit en 1). De esta manera el microprocesador puede indicar si el registro de cálculo (donde casi siempre se encuentra el resultado de la última operación) es igual o desigual a cero -ya está, las primeras 2 comparaciones se han realizado. Para la comparación A igual B o bien A diferente B solamente hemos de restar los dos números para comprobar a continuación si el contenido del ACU es cero, lo cual puede comprobarse con el flag Z (Z significa ZERO=0). Si este flag está activado (1) el resultado de la última operación era cero. Si Z=0 entonces el último resultado era diferente de cero. El flag no se limita únicamente al acumulador, pero por otro lado tampoco todos los comandos alteran los flags. En el anexo pueden comprobar la lista de comandos y que influencia tienen sobre los flags.
Ahora volvemos a las comparaciones.
En el caso de "mayor" y "menor" procedemos casi de la misma manera que con "igual". Después de la sustracción comprobamos si el número en el ACU es menor o mayor que cero, reconociéndolo por el bit de signo:

A Mayor que B <=> A - B Mayor que 0 (verdadero si bit 7 = 0)
A Menor que B <=> A - B Menor que 0 (verdadero si bit 7 = 1)

Muchos comandos transfieren el bit del signo al flag S (S significa "SIGN" = signo). Aquí puede comprobarse fácilmente mediante comandos especiales. Otro flag se denomina P/V (dado que en este libro casi no lo utilizamos, lo llamamos simplemente P).

Este flag tiene 2 funciones, por un lado puede indicar la paridad (par o impar; su significado no es importante en este momento), por otro lado indica después de algunas operaciones aritméticas si se ha modificado erróneamente el bit del signo (lo que tampoco es importante de momento, porque únicamente queremos dar una visión general del lenguaje máquina).

13.8 EL PRIMER PROGRAMA

Después de haber conocido las decisivas bases de la programación máquina comencemos ahora con el programa más simple. Confeccionemos un programa de adición para 2 valores de 8 bits.

En primer lugar hemos de indicar al programa de alguna manera cuales son los números a sumar. En lenguaje máquina no existe un comando INPUT y por ello debemos almacenar los números en algún lugar de la memoria, desde donde el programa pueda extraerlos. Esta será la primera tarea que deba realizar nuestro programa. El comando "LD A,(nnnn)" funciona casi como un PEEK, extrayendo el byte de la dirección nnnn y llevándolo al ACU.

También sabemos donde se encuentra el segundo byte, pero no podemos cargarlo al procesador con LD B,(nnnn) pues este comando no existe en lenguaje máquina. Sin embargo es posible utilizar la pareja de registros HL como apuntador a nuestro byte. Para ello hemos de almacenar la dirección en el registro deseado mediante LD HL,nnnn. Tengan en cuenta que la expresión "nnnn" no debe ir entre paréntesis, indicando que esta expresión debe cargarse directamente en HL y que no representa la dirección del valor deseado.

Con el siguiente comando sumaremos finalmente ambos bytes. El comando es "ADD A,(HL)" y produce la adición del valor contenido en el ACU con el byte cuya dirección se almacena en HL. El resultado se almacena nuevamente en el ACU. Si la suma de ambos números excede el área de valores de 1 byte (0-255), el Z80 lo indica cargando un 1 en el bit carry (desbordamiento).

Dado que en el acumulador no nos sirve para nada el resultado, lo trasladamos con otro comando a una posición de memoria de donde podemos leerlo mediante PEEK.

Esto se realiza mediante "LD (nnnn),A". Este comando funciona exactamente igual que "LD A,(nnnn)", solamente en sentido inverso.

Finalmente la subrutina se termina con RET (como con el RETURN de BASIC). Para ello debe saber que el interpretador trata las llamadas de programas máquina como subrutinas, mediante CALL; el último comando de una rutina en lenguaje máquina siempre debe ser RET, de lo contrario el ordenador se "cuelga".

Hasta ahora hemos comenzado a programar sin preocuparnos por las células de memoria con las cuales queremos trabajar. En nuestro ejemplo pueden elegir las direcciones casi al azar, únicamente deben asegurarse de que las mismas no las utiliza ya el ordenador (esto puede evitarlo con el comando MEMORY). Una advertencia: limite la memoria BASIC a &AAFF y de esta manera puede utilizar todos los bytes desde &AB00 hasta &AB7F. Cargaremos nuestro programa a partir de &AB00 en la memoria, los valores a sumar y el resultado lo almacenaremos al final (anterior) área BASIC. El programa se verá:

Dirección	Comando	Comentario
AB00	LD A,(AB7F)	Carga primer número
AB03	LD HL,AB7E	Almacena segundo número en AB7E
AB06	ADD A,(HL)	Suma ACU y segundo número
AB07	LD (AB7D),A	Almacena resultado
AB0A	RET	Final del programa

Puede ver por las direcciones que cada comando ocupa diferente cantidad de bytes. Los comandos que contengan una dirección necesitan al menos 3 bytes, otros como por ejemplo "ADD A,(HL)" son comandos de 1 byte.

Antes de poder 'POKEAR' los bytes a la memoria, hemos de saber que aspecto tienen. En la tabla de comandos Z80 encontrará el Low-Code para cada comando. Para "LD A,(nnnn)" es 3A más 2 bytes de direcciones.

Hay que tener en cuenta que el Low-byte de la dirección va siempre delante del High-byte. El comando completo representado en números es:

```
3A   7F   AB
```

A continuación los códigos para el resto del programa:

```
21   7E   AB   (LD HL,AB7E)
86             (ADD A,(HL))
32   7D   AB   (LD (AB7D),A)
C9             (RET)
```

Estos valores deben POKEARSE a las posiciones correspondientes de memoria. Esto puede realizarse mediante el siguiente programa BASIC.

```
10 MEMORY &AAFF
20 FOR I = &AB00 TO &AB0A: READ A: POKE I,A: NEXT
30 DATA &3A,&7F,&AB,&21,&7E,&AB,&86,&32,&7D,&AB,&C9
```

La rutina máquina puede arrancarse con CALL &AB00. Previamente deben almacenarse los 2 números a sumar mediante POKE &AB7F,1er número y POKE &AB7E, 2do número. Después del CALL puede obtener el resultado mediante PRINT PEEK(&AB7D). Para poder probar diferentes valores agregue las siguientes líneas al programa BASIC anterior:

```
40 INPUT "Número 1, Número 2": Z1,Z2
50 POKE &AB7F, Z1: POKE &AB7E, Z2
60 CALL &AB00: PRINT PEEK(&AB7D): GOTO 40
```

La substracción se programa de igual forma, solamente debe reemplazar el comando ADD por SUB (HL). En la línea DATA cambia por esto el séptimo valor de &86 a &96.

13.9 ¿COMO SE PROGRAMA UN BUCLE?

Si quiere repetir un procedimiento múltiples veces en BASIC, tiene 2 formas de programarlo: FOR-NEXT y WHILE-WEND. Existe una tercera posibilidad pero no es tan cómoda y por ello casi no se utiliza. Se trata de incrementar en 1 un contador después de cada ciclo y bifurcar al inicio mediante IF-THEN si es necesario aún otro ciclo. Aunque nos parezca muy incómodo este procedimiento, no nos queda otra posibilidad de utilizar esta técnica para programar en lenguaje máquina. En vez de una variable utilizamos un registro del procesador y controlamos el bucle contando desde arriba hacia abajo, el resto funciona igual. El siguiente programa no hace otra cosa que esperar 255 ciclos (porque el bucle es vacío). No hemos indicado un programa de carga en BASIC porque el procedimiento no puede visualizarse. El lenguaje máquina trabaja con gran rapidez de manera que el bucle se ejecuta en fracciones de segundo. El programa:

```
AB00   LD  B,FF       Carga longitud del bucle
AB02   DEC B          Decrementar B en 1
AB03   JP  NZ,AB02    Bifurcar si diferente de 0
AB06   RET            Final del programa
```

El primer comando carga la longitud del bucle al registro B. Este número está incorporado fijo en el programa posicionado directamente después del OP-CODE. Seguramente se preguntarán porqué se utiliza el registro B para el contador. Y bien, el ACU está predestinado para operaciones aritméticas y en una forma similar existen comandos de contador especiales para el registro B.

El comando DEC (Decrement = decrementar en 1) sustrae 1 de B, si el resultado es cero el Zeroflag contendrá un 1 y de lo contrario quedará en cero. Este comportamiento lo utiliza el siguiente paso. JP NZ bifurca únicamente a la dirección indicada si el flag Z contiene 0, lo que significa que la bifurcación de retorno solamente se realiza si no hemos llegado a ejecutar el final del

bucle. SI Z contiene 1 el Z80 continúa con el siguiente comando que en nuestro caso es RET, por lo cual termina la rutina.

13.10 OTRAS RUTINAS ARITMETICAS

13.10.1 ADICION CON 16 BITS

Con 8-bits -como ya hemos visto- sólo se puede operar con valores inferiores a 255. 16 bit representan números del área de hasta 32767. El Z 80 pertenece a los pocos procesadores de 8 bits que ofrecen comandos para la aritmética de 16 bits y posee también los necesarios registros de 16 bits.

Para la adición de 16 bits debería cargarse el primer número en la pareja de registros DE. Mediante el comando "LD DE,(nnn)" se carga el byte nnn en el registro D, E contiene el valor de la posición de memoria con la dirección nnn+1. También en esste caso se utiliza el "formato pointer" low-high. El mismo comando existe también para HL.
El comando de adición de 16 bits se llama "ADD HL,DE". Con este comando se suman los 2 números contenidos en DE y HL almacenandose el resultado en HL, desde dónde se lleva a la memoria mediante "LD (nnn),HL".
El programa completo es el siguiente:

```
AB00  LD DE,(AB7E)
AB04  LD HL,(AB7C)
AB07  ADD HL,DE
AB08  LD  (AB7A),HL
AB0B  RET
```

Tengan en cuenta que cada comando LD trata 2 bytes a la vez. A continuación el programa se carga:

```
10 DATA &ED,&5B,&7E,&AB
11 DATA &2A,&7C,&AB
12 DATA &19
13 DATA &22,&7A,&AB
14 DATA &C9
20 FOR I = AB00 TO &AB0B: READ a: POKE I,a: NEXT
30 IMPUT " Número 1, Número 2"; z1,z2
40 POKE &AB7E, z1 AND 255: POKE &AB7F, INT(z1/256)
50 POKE &AB7C, z2 AND 255: POKE &AB7D, INT(z2/256)
60 CALL &AB00: PRINT PEEK(&AB7A) + 256 * PEEK(&AB7B)
70 GOTO 50
```

También aquí pueden ustedes experimentar con diferentes valores.

13.10.2 MULTIPLICACION

Para multiplicar 2 valores, hemos de rallzar sumas sucesivas (como hemos visto ya en el capítulo 13.6.3). En BASIC este algoritmo podría representarse:

```
10 IMPUT " Número 1, Número 2" ; z1, z2: e=0
20 FOR I = 1 TO z1
30 e = e + z2: NEXT
40 PRINT "Resultado "; e
```

Como pueden comprobar, el algoritmo puede programarse con un bucle y una única edición, ambas operaciones han sido ya estudiadas en lenguaje máquina.
La multiplicación de 2 números de 8 bits tiene como resultado un número de 16 bits. Por ello se ofrece la modificación de la rutina de adición de 16 bits adecuadamente.
Dentro del bucle se debe encontrar únicamente el comando de adición. Los restantes comandos sirven para la preinstalación de registros o bien para almacenar el resultado. Con el primer comando (vean listado) se carga el número 1 al acumulador. En principio

debe cargarse en registro B, pero debido a que no podemos cargarlo directamente desde la memoria en B, almacenamos el número primeramente en el ACU para llevarlo con el segundo comando al registro B. De esta manera queda determinada la longitud del bucle. El otro número de 8 bits se suma continuamente a la pareja de registros HL y por ello llega al registro E a través del mismo "desvío".
Debido a que ADD HL,DE también suman siempre el registro D, debe inicializarse este registro a ceros. También HL han de inicializarse a ceros antes de comenzar el bucle. Esto lo realizamos con los 2 comandos siguientes.

A conticuación comienza el bucle con el comando ADD. Dado que la pareja de registros HL sólo se altera mediante el comendo de adición, el mismo contiene siempre el último resultado parcial.
Probablemente se podrán explicar ustedes mismos el resto del programa máquina. DEC B y JP NZ,AB0D marcan el final del bucle, LD (AB7C),HL almacena el resultado y RET termina el programa. Aquí el listado:

```
AB00  LD   A,(AB7F)      número 1
AB03  LD   B,A           hacia B
AB04  LD   A,(AB7E)      número 2
AB07  LD   E,A           hacia E
AB08  LD   D,00          borrar D
AB0A  LD   HL,0000       borrar HL
AB0D  ADD  HL,DE         sumar
AB0E  DEC  B             decrementar
AB0F  JP   NZ,AB0D       bifurcar si diferente de 0
AB12  LD   (AB7C),HL     almacenar resultado
AB15  RET                retorno al BASIC
```

Naturalmente para ejecutar este programa necesitamos un programa de carga en BASIC:

```
10 DATA    &3A,&7F,&AB
11 DATA    &47
12 DATA    &3A,&7E,&AB
```

```
13 DATA     &5F
14 DATA     &16,&00
15 DATA     &21,&00,&00
16 DATA     &19,&05
17 DATA     &C2,&0D,&AB
18 DATA     &22,&7C,&AB
19 DATA     &C9
20 FOR I=&AB00 TO &AB15: READ a: POKE I,a: NEXT
30 INPUT "Número 1, Número 2"; z1,z2
40 POKE &AB7F, z1: POKE &AB7E, z2
50 CALL &AB00: PRINT PEEK(&AB7C) + 256 * PEEK(&AB7D)
60 GOTO 30
```

13.11 RUTINAS UTILES EN LENGUAJE MAQUINA

En este capítulo queremos presentarles algunas rutinas máquina que no sirven para aprender lenguaje máquina, sino que tienen utilidad práctica.

Comencemos con algo destructivo, el borrado de áreas enteras de memoria. En el BASIC podemos resolverlo con un bucle FOR-NEXT y con comandos POKE pero esto demora relativamente bastante. Esta tarea podemos realizarla con una rutina mucho más rápida en lenguaje máquina. Sin embargo hay una restricción que solamente podemos borrar un máximo de 256 bytes por ciclo (si no queremos utilizar una rutina excesivamente compleja).

Similar al comando SAVE para determinadas áreas de memoria tenemos que indicar a nuestro programa solamente la dirección inicial y la cantidad de bytes a borrar.

HL sirve como apuntador a los bytes a borrar y se incrementa en cada ciclo del bucle. El registro B contiene como es usual la longitud del bucle (y así la cantidad de bytes a borrar). En el bucle se carga en primer lugar un cero en el byte al que apunta HL. A continuación se actualizan ambos registros y si es necesario JP NZ,(xxxx) bifurca nuevamente al comienzo del bucle.

En principio nuestro programa finalizaría ya en este punto pero queremos incluír un poco de lujo. Al final del bucle HL contiene la dirección del byte que se borraría en el siguiente ciclo, si el bucle no hubiera finalizado. Esta dirección se lleva nuevamente a la posición de memoria de donde el primer comando extrae la dirección de inicio. Si ahora quiere continuar borrando no debe pokear la dirección inicial sino puede limitarse al comando CALL.

Un procedimiento similar puede utilizarse para la longitud del

bucle. Dado que la posición de memoria para este número no se modifica mediante el programa y el valor puede siempre permanecer igual, también podemos ahorrarnos este POKE después de la primera llamada. Si quiere borrar 100 bytes en intervalos de 10 utilice la siguiente secuencia de comandos:

```
POKE &AB7F, 10: POKE &AB7D, Lowbyte: POKE &AB7E, Highbyte
FOR I = 1 TO 10: CALL &AB00: NEXT
```

Esta manera no tiene mucho sentido porque se podría realizar mejor con una llamada y longitud del bucle 100. Pero si incluimos más comandos en el bucle FOR-NEXT (por ejemplo esperar pulsaciones de tecla) podríamos construir un borrado por intervalos de la memoria de pantalla. Además puede realizarse de esta manera un borrado de bloques grandes de memoria.

Hay otro punto a mencionar referente a la longitud del bucle. Si se llama a una rutina con longitud de bucle 0, se borran 256 bytes, porque antes del primer JP (antes de la primera finalización posible) se decrementa el B. 0 - 1 da para el Z80 (y todos los restantes procesadores) 255 (vean la representación binaria y sumen 255 y 1). Por ello el bucle no ha finalizado y se borrarán otros 255 bytes. El programa es el siguiente:

```
AB00  LD   HL,(AB7D)
AB03  LD   A,(AB7F)
AB06  LD   B,A
AB07  LD   (HL),0
AB09  INC  HL
AB0A  DEC  B
AB0B  JP   NZ,AB07
AB0E  LD   (AB7D),HL
AB11  RET
```

```
10 DATA     &2A,&7D,&AB
11 DATA     &3A,&7F,&AB
12 DATA     &47
13 DATA     &36,&00
14 DATA     &23
15 DATA     &05
16 DATA     &C2,&07,&AB
17 DATA     &22,&7D,&AB
18 DATA     &C9
20 FOR I = &AB00 TO &AB11: READ a: POKE I,a: NEXT
30 INPUT "Dirección inicial"; a
40 POKE &AB7D, a-INT(a/256): POKE &AB7E, INT(a/256)
50 INPUT "Longitud"; b
60 POKE &AB7F, b
70 CALL &AB00: GOTO 20
```

La segunda rutina sirve para copiar bloques de memoria. Dado que podemos utilizar un comando Z80 especial este programa es más corto que el anterior y la restricción a 256 puede olvidarse.

El comando especial se denomina LDIR y copia sin nuestra ayuda bloques enteros de memoria de cualquier longitud. Esta longitud debe estar almacenada previamente en la pareja de registros BC. Además el Z80 tiene que encontrar la dirección inicial del área a copiar en HL y la dirección del área de destino en DE.
Ahora puede accionar LDIR, que copia el byte que se direcciona en HL a la dirección que indica BE. A continuación se incrementan HL y DE apuntando de esta manera al siguiente byte. El Z80 decrementa además BC. Mientras que BC sea diferente de 0 el comando LDIR se repite automáticamente. Como podemos ver un único comando reemplaza un bucle entero.

El resto ya no necesita explicación. Este programa ha sido confeccionado para un área de memoria un poco diferente. Por ello no influye la rutina del borrado y puede utilizarse al mismo tiempo con ella.

```
AB20  LD    HL,(AB6E)
AB23  LD    DE,(AB6C)
AB27  LD    BC,(AB6A)
AB2B  LDIR
AB2D  RET
```

```
10 DATA     &2A,&6E,&AB
11 DATA     &ED,&5B,&6C,&AB
12 DATA     &ED,&4B,&6A,&AB
13 DATA     &ED,&B0
14 DATA     &C9
20 FOR I = &AB20 TO &AB2D: READ a: POKE I,a: NEXT
30 INPUT "Area fuente   "; a
40 POKE &AB6E, a-INT(a/256): POKE &AB6F, INT(a/256)
50 INPUT "Area destino "; b
60 POKE &AB6C, b-INT(b/256):  POKE &AB6D, INT(b/256)
70 INPUT "Longitud ";c
80 POKE &AB6A, c-INT(c/256): POKE &AB6B, INT(c/256)
90 CALL &AB20: GOTO 30
```

13.12. LAS POSIBILIDADES DE DIRECCIONAMIENTO

Si han observado la lista de comandos seguramente habrán notado que el operando no se compone de registros como A, B, C, etc. sino que hay también paréntesis, etc. En un comando como ADD A,B se ve claramente su tarea; suma los registros A y B. ¿Pero que significa ADD A,(HL)?

En general debería memorizarse que una expresión entre paréntesis representa la dirección de un byte en la memoria y no el operando en sí mismo. "(nn)" significa "contenido de la posición de memoria nnnn" (en la forma abreviada una letra representa un byte; nn representa 2 bytes = 16 bits y 4 dígitos hexadecimales). Este método se denomina también 'direccionamiento absoluto'. Si se omiten los paréntesis se utilizan los valores directamente como operandos. ADD A,n suma el byte nn (que se encuentra directamente detrás del OP-CODE en la memoria, casi una constante) al ACU. Esta forma de direccionamiento se denomina 'direccionamiento directo'.
Otro método es el denominado 'direccionamiento indirecto' en el cual el operando no indica la dirección de los datos a tratar, sino el lugar donde se encuentra esta dirección. El comando ADD A,(HL) opera de manera que el procesador toma en primer lugar el valor contenido en HL. Este valor se utiliza como apuntador a la posición de memoria en la cual está contenido el byte que finalmente se suma al ACU. 'Muy laborioso' podrá pensar ahora, pero 'muy listo' dijeron los que desarrollaron el Z80. Mediante este direccionamiento indirecto puede accederse a bloques enteros de bytes como en una tabla.
Una modalidad del método recién explicado es el "direccionamiento indirectamente indexado". El operando "(IX+d)" significa en este caso que al valor del registro IX se suma la constante dd y el resultado de esta adición señala la dirección definitiva.
La última modalidad de direccionamiento de la familia es el "direccionamiento relativo" que puede encontrarse únicamente en determinados tipos de bifurcaciones. De forma similar que en el direccionamiento indexado sigue una constante después del OP-CODE.

Esta se suma al contador de programa. Si el bit 7 del byte contiene 1 se tratará como número negativo (lo mismo puede decirse para el direccionamiento indirectamente indexado). El resultado es una bifurcación de retroceso. Dado que el contador de programa después de la lectura de la constante apunta ya al comando siguiente puede bifurcarse un máximo de 129 pasos hacia delante y un máximo de 127 pasos hacia atrás.

13.13 LOS COMANDOS DEL Z-80

Observando el anexo con los OP-CODES del Z-80 (donde se listan todos los comandos con sus respectivos códigos y los flags influídos) podrán comprobar que existen varios grupos de comandos que solamente se diferencian por los operandos (es decir el modo de direccionamiento). Esto grupos se describen a continuación. No es necesario memorizar todas las descripciones; solamente sirven como visión general sobre las grandes posibilidades del lenguaje máquina. Si más adelante programan ustedes en este lenguaje pueden utilizar la tabla de OP-CODES para la elección de sus comandos.

Dado que los comandos para diferentes operandos funcionan siempre igual, los últimos se indican mediante letras de generalización (ejemplo X, n etc.).

ADC A,X
El operando X se suma en el ACU. Un eventual desbordamiento se lleva con el Bit-Carry. El resultado se almacena en el ACU, un nuevo desbordamiento se almacena en el Bit-Carry. Después de la operación se activan en consecuencia los flags según el contenido del ACU.

ADC HL,X
Este comando funciona igual que el anterior, únicamente operando con 16 bits, por ello el operando (ahora un registro de 16 bits) y el bit Carry se suman en el par de registros HL. El resultado se almacena en HL y se activan los flags correspondientes.

ADD A,X
El operando se suma directamente al ACU, y el resultado queda en el ACU. No se tiene en cuenta el Bit-Carry, pero después de la adición se activan todos los flags correspondientes dependiendo del resultado.

ADD HL,X
Como ADC HL,X, el Bit-Carry no se tiene en cuenta para la adición.

ADD IX,X
El operando se suma con el registro índice IX, el resultado se almacena nuevamente en el registro y se actualiza el Bit-Carry. Los restantes flags no se alteran.

ADD IY,X
Como ADD IX,X, únicamente utilizando el registro de índice IY.

AND X
El operando se interrelaciona (AND) con el ACU, el resultado se almacena nuevamente en el ACU. Se actualizan los flags correspondiendo con el resultado. El Bit-Carry permanece con 0 (esta interrelación no produce desbordamiento).

BIT N,X
Se comprueba el bit N del operando X, es decir que se complementa y transfiere al flag Z. Si el bit elegido contiene un 1 el flag Z tendrá 0, de lo contrario Z=1. Los flags S y P obtienen valores casuales.

CALL NN
Se llama a la subrutina que se inicia en dirección nnnn (similar al GOSUB).

CALL CONDICION,NN
Se bifurca a la subrutina que se inicia en nnnn si se cumple la condición. Como condición se pueden comprobar cada uno de los 4 flags S, Z, P y C (carry). Según el valor se bifurca o se contiúa con el comando siguiente del programa.

CCF
Complementa el Bit-Carry (1->0, 0->1)

CP X

El operando X se compara con el ACU, lo que significa que se sustrae, el resultado no se almacena aún en el ACU, por ello el ACU no se altera. Si el operando es menor que el ACU, se activa el flag S con 1, si el operando es mayor o igual S=0, si ambos valores son iguales, el flag Z=1 si son diferentes Z=0.

CPD

El contenido de HL se utiliza como apuntador a una célula de memoria que se compara con el ACU. Si ambos valores son iguales, el flag Z=1, si el ACU es mayor o igual entonces S=0, si el ACU es menor que el byte de memoria, S=1. A continuación el valor de HL se decrementa con BC (en 1). Si el contenido de BC es 0 entonces el flag P=0, de lo contrario P=1, BC funciona como contador de bytes.

CPDR

Funciona como CPD, pero la ejecución se repite automáticamente hasta que se verifica la igualdad Z=1 o BC=0 (flag P en 0).

CPI

Como CPD, pero en este caso HL no se decrementa sino que se incrementa en 1.

CPIR

Como CPDR, pero incrementando HL en 1.

CPL

Invierte los valores de todos los bits del ACU (0 se convierte en 1 y viceversa).

DAA

Corrige eventuales dígitos BCD erróneos en el ACU después de una operación aritmética (ADC, ADD, DEC, INC, NEG, SBC, SUB) así como los valores de los flags.

DEC X

Los operandos se decrementan en 1 almacenándose nuevamente.

Los flags se activan según el resultado, excepto en el caso de los registros BC, DE, HL, IX, IY y SP.

DI
Una vez ejecutado este comando el Z-80 ignora todas las peticiones de interrupción que se produzcan (interrupciones con máscara).

DJNZ e
B se decrementa en 1. Siempre que el contenido de B sea diferente de 0 se produce la bifurcación relativa. e se suma a la dirección del siguiente comando y de esta manera el programa continúa su ejecución en esta nueva dirección.

EI
Después de ejecutar este comando se liberan las máscaras de interrupción, lo que significa que el procesador bifurca a rutinas especiales después de una petición de interrupción.

EX X,Y
Se intercambian los 2 operandos proporcionados. X puede representar también (SP), en cuyo caso el intercambio se realiza entre el segundo operando con el elemento superior de la pila.

EXX
Los registros BC, DE y HL se intercambian con los registros secundarios.

HALT
EL procesador detiene la ejecución de programas hasta que se solicita una interrupción .

IM N
Seleccionar modalidad de interrupción.
Modo 0: El componente que pide la interrupción ha de tener preparado un comando para la ejecución en el bus de datos.
Modo 1: El Z-80 bifurca a dirección 0038 (hexadecimal) y ejecuta allí una rutina de manejo de interrupciones.
El CPC trabaja en esta modalidad. No deberían modificar nunca la

modalidad de interrupción porque de lo contrario su ordenador no trabajará correctamente.

IN X,(C)
Registro C indica el puerto desde el cual se leerá un byte al registro X.

IN A,(N)
El ACU se carga con el byte del puerto N.

INC X
Como DEC pero el operando se incrementa en 1.

IND
El registro HL sirve como apuntador a la posición de la memoria en la cual se carga un byte desde el puerto. El registro C indica el número del puerto, además se decrementan HL y B. Si B=0 entonces Z=1, los restantes flags (excepto el Carry) tienen valores al azar.

INDR
Como IND pero el comando se repite hasta que B=0.

INI
Como IND, pero HL se incrementa.

INIR
Como INDR pero HL se incrementa.

JP NN
El programa bifurca a dirección NNNN (similar GOTO).

JP CONDICION,NN
Bifurca si se cumple la condición (para condición vean CALL).

JP X
Bifurca a la dirección contenida en el operando X (HL, IX, IY).

JR N y JR CONDICION,N
Como los correspondientes comandos JP pero se bifurca de forma relativa, es decir el byte siguiente al OP-CODE (N) se suma a PC.

LD X,Y
Este grupo se compone del mayor número de comandos, todos los cuales siguen un principio. El contenido del segundo operando se traspasa al primer operando. Esto por ejemplo puede significar que el contenido del ACU se carga en la posición NNNN, o viceversa. Además existen comandos de carga para los registros de 16 bits. Si quiere traspasarse un registro de 16 bits a la memoria el lowbyte se almacena en NNNN y el highbyte en NNNN + 1. Este formato se utiliza siempre en la técnica de ordenadores aunque en el primer momento parece un poco extraño (memorice: primero low entonces high).

LDD
HL y DE sirven como apuntadores a bytes de la memoria. El contenido de la posición direccionada por HL se traspasa a la posición direccionada por DE. A continuación se decrementan HL, DE y BC. Si BC=0 flag P=0. De esta manera BC puede utilizarse como contador.

LDDR
Como LDD pero en este caso el comando se repite automáticamente hasta que BC=0.

LDI
Como LDD pero los registros HL y DE se incrementan.

LDIR
Como LDDR pero incrementando los registros HL y DE.

NEG

El byte del ACU se invierte, es decir se multiplica por -1. El resultado se almacena en el ACU y los flags se actualizan. P=1 si el contenido anterior del ACU era 80 (hexadecimal), C=1 si el ACU tenía contenido 0.

NOP

Este comando no realiza ninguna función, sólo sirve para generar pequeños tiempos de espera.

OR X

El ACU se interrelaciona OR con el operando. El resultado se almacena nuevamente en el ACU y los flags se actualizan. El flag Carry se pone a ceros porque en la interrelación OR no hay desbordamiento.

OTDR

HL sirve como apuntador a una posición de memoria cuyo contenido se envía al puerto (C). Se decrementan además HL y B repitiéndose esta operación hasta que B=0. Excepto el bit Carry y Z (que se activa a 1) los restantes flags tienen valores al azar.

OTIR

Como OTDR pero HL se incrementa.

OUT (C),X

El operando X se envía a un puerto cuyo número se indica en el registro C.

OUT (N),A

El ACU se envía al puerto N.

OUTD

HL sirve como apuntador a una posición de memoria cuyo contenido se envía al puerto (C) decrementándose HL y B. Si B=0 el flag Z=1. Los restantes flags (excepto el Carry) obtienen valores al azar.

OUTI
Como OUTD pero incrementando HL.

POP X
La pareja de registros del operando se carga de la pila y el apuntador de pila se actualiza. AF significa ACU y flags.

PUSH X
La pareja de registros del operando se lleva a la pila (stack) y el apuntador de stack se actualiza (stack pointer). AF significa ACU y flags.

RES N,X
Se borra el bit N del operando X.

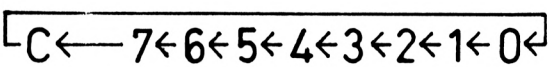
RET y RET CONDICION
Estos comandos provocan el retorno de una subrutina (similar RETURN del BASIC). Además puede indicarse una condición, es decir que el retorno se realiza únicamente si un determinado flag tiene un valor.

RETI
Provoca retorno de rutina de interrupción (similar RET).

RETN
Provoca retorno de una rutina NMI.

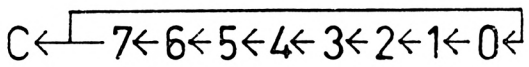
RL X y RLA
Se produce una rotación hacia la izquierda de los bits del operando. El bit Carry se desplaza al bit 0 y el bit 7 al bit carry. (ver dib.) Loas comandos RL alteran todos los flags.
El comando RLA (no RL A) representa una excepción. Este trabaja como RL A pero influye únicamente sobre el flag Carry.

Dib. 9



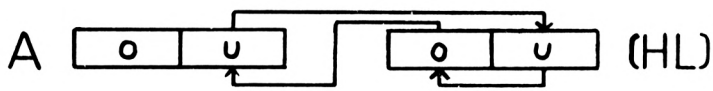
RLC X y RLCA
El operando rota hacia la izquierda y además se transfiere el bit 7 al Carry (ver dib.).
Excepto RLCA todos los comandos RLC alteran los flags.

Dib. 10



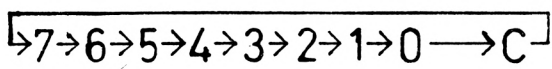
RLD
Este comando realiza una rotación BCD es decir la porción izquierda de la posición de memoria, cuya dirección está almacenada en HL, se traspasa a la porción derecha del ACU, la porción derecha de la posición de memoria se desplaza a la izquierda y la porción derecha original del ACU se traspasa a la porción derecha de la posición de memoria (ver dib.). Se alteran los flags S, Z y P.

Dib. 13



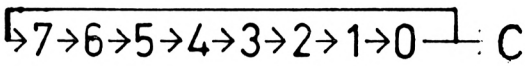
RR X y RRA
Estos comandos funcionan como RL X y RLA, solamente que el operando se rota hacia la derecha.

Dib. 11



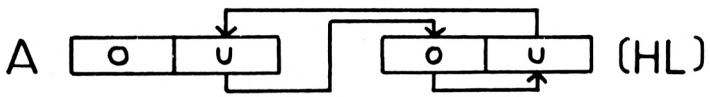
RRC X y RRCA
Como RLC X y RLCA pero rotando hacia la derecha.

Dib. 12



RRD
Como RLD pero con rotación a la derecha. El nibble inferior del ACU (un nibble es medio byte) de sedplaza al nibble superior de la posición de memoria direccionada por HL. El nibble superior de la posición de memoria se desplaza a la posición inferior y la posición inferior del nibble de la posición de memoria se desplaza al nibble inferior del ACU.

Dib. 14



RST N
En la dirección N (definida fija) se arranca una subrutina.

SBC A,X
El operando X se sustrae del ACU teniendo en cuenta un eventual desbordamiento indicado por el bit Carry. El resultado se almacena nuevamente en el ACU y un nuevo desbordamiento se señala en el Carry. Después de la ejecución los flags tienen los valores correspodiendo al contenido del ACU.

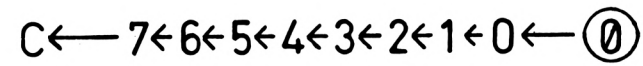
SBC HL,X
Este comando funciona como el anterior pero provoca una adición de 16 bits. A continuación se sustrae el operando (ahora un registro de 16 bits) y el bit Carry de la pareja de registros HL. El

resultado se almacena nuevamente en HL y los flags se modifican debidamente.

SCF
Se activa el bit Carry con 1.

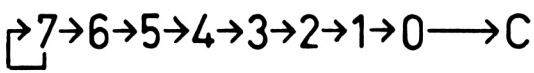
SET N,X
El bit N del operando X se activa con 1.

SLA X
El operando se desplaza hacia la izquierda, bit 0 se llena con 0 y bit 7 se desplaza al bit Carry. (ver dib.) Los flags se alteran según el resultado.



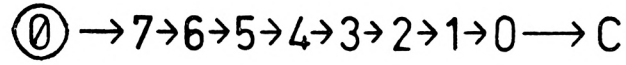
Dib. 15

SRA X
El operando se desplaza hacia la derecha, bit 7 (el signo) se mantiene, bit 0 se lleva al bit Carry (ver dibujo). Se alteran todos los flags.



Dib. 16

SRL X
El operando se desplaza hacia la derecha, el bit 7 se pone a ceros y el bit 0 se lleva al Carry (ver dib.). Se alteran todos los flags.



Dib. 17

SUB X
El operando se sustrae del ACU, los flags se alteran según el resultado y éste se almacena nuevamente en el ACU.

XOR X
El operando se interrelaciona XOR con el ACU y el resultado se almacena nuevamente en el ACU. Los flags se actualizan, mientras el bit Carry se borra (ya que una interrelación XOR no provoca desbordamiento).

13.14 Z-80 OP-CODES

En la siguiente tabla se listan todos los OP-CODES de los comandos del Z-80. La columna "Flags" indica además todos los flags que se alteran con el comando. En los diversos comandos puede ocurrir que los flags se alteren al azar después de la ejecución. Estos casos puede extraerlos de las descripciones detalladas de los comandos.

Mnemónico	OP-CODE	Flags	Comentario
ADC A,A	8F	S Z P C	Suma Carry y A al A
ADC A,B	88	S Z P C	..... al B
ADC A,C	89	S Z P C	..... al C
ADC A,D	8A	S Z P C	..... al D
ADC A,E	8B	S Z P C	..... al E
ADC A,H	8C	S Z P C	..... al H
ADC A,L	8D	S Z P C	..... al L
ADC A,n	CEnn	S Z P C	..... al byte indicado
ADC A,(HL)	8E	S Z P C	..... a dirección en HL
ADC A,(IX+d)	DD8Edd	S Z P C	..... a dirección IX + D
ADC A,(IY+d)	FD8Edd	S Z P C	..... a dirección IY + D
ADC HL,BC	ED4A	S Z P C	Suma Carry y HL a BC
ADC HL,DE	ED5A	S Z P C	..... a DE
ADC HL,HL	ED6A	S Z P C	..... a HL
ADC HL,SP	ED7A	S Z P C	..... a SP
ADD A,A	87	S Z P C	Suma ACU a ACU
ADD A,B	80	S Z P C	..... a B
ADD A,C	81	S Z P C	..... a C
ADD A,D	82	S Z P C	..... a D
ADD A,E	83	S Z P C	..... a E
ADD A,H	84	S Z P C	..... a H
ADD A,L	85	S Z P C	..... a L
ADD A,n	C6nn	S Z P C	..... al byte indicado
ADD A,(HL)	86	S Z P C	..... a dirección en HL
ADD A,(IX+d)	DD86dd	S Z P C	..... a dirección IX + D
ADD A,(IY+d)	FD86dd	S Z P C	..... a dirección IY + D

ADD HL,BC	09	C	Suma HL a BC
ADD HL,DE	19	C	..... a DE
ADD HL,HL	29	C	..... a HL
ADD HL,SP	39	C	..... a SP
ADD IX,BC	DD09	C	Suma IX a BC
ADD IX,DE	DD19	C	..... a DE
ADD IX,HL	DD29	C	..... a HL
ADD IX,SP	DD39	C	..... a SP
ADD IY,BC	FD09	C	Suma IY a BC
ADD IY,DE	FD19	C	..... a DE
ADD IY,HL	FD29	C	..... a HL
ADD IY,SP	FD39	C	..... a SP
AND A	A7	S Z P C=0	Interrelaciona AND ACU con Akku
AND B	A0	S Z P C=0	..... con B
AND C	A1	S Z P C=0	..... con C
AND D	A2	S Z P C=0	..... con D
AND E	A3	S Z P C=0	..... con E
AND H	A4	S Z P C=0	..... con H
AND L	A5	S Z P C=0	..... con L
AND n	E6nn	S Z P C=0	..... con byte indicado
AND (HL)	96	S Z P C=0	..... con dirección en HL
AND (IX+d)	DD96dd	S Z P C=0	..... con dirección IX + D
AND (IY+d)	FD96dd	S Z P C=0	..... con dirección IY + D
BIT 0,A	CB47	Z	Comprueba bit 0 del ACU
BIT 0,B	CB40	Z	Comprueba bit 0 de B
BIT 0,C	CB41	Z	..... de C
BIT 0,D	CB42	Z	..... de D
BIT 0,E	CB43	Z	..... de E
BIT 0,H	CB44	Z	..... de H
BIT 0,L	CB45	Z	..... de L
BIT 0,(HL)	CB46	Z	..... de dirección en HL
BIT 0,(IX+d)	DDCBdd46	Z	..... de dirección IX + D
BIT 0,(IY+d)	FDCBdd46	Z	..... de dirección IY + D
BIT 1,A	CB4F	Z	Comprueba bit 1 del ACU
BIT 1,B	CB48	Z	Comprueba bit 1 de B
BIT 1,C	CB49	Z	..... de C
BIT 1,D	CB4A	Z	..... de D
BIT 1,E	CB4B	Z	..... de E
BIT 1,H	CB4C	Z	..... de H

BIT 1,L	CB4D	Z	..... de L
BIT 1,(HL)	CB4E	Z	..... de dirección en HL
BIT 1,(IX+d)	DDCBdd4E	Z	..... de dirección IX + D
BIT 1,(IY+d)	FDCBdd4E	Z	..... de dirección IY + D
BIT 2,A	CB57	Z	Comprueba bit 2 del ACU
BIT 2,B	CB50	Z	Comprueba bit 2 de B
BIT 2,C	CB51	Z	..... de C
BIT 2,D	CB52	Z	..... de D
BIT 2,E	CB53	Z	..... de E
BIT 2,H	CB54	Z	..... de H
BIT 2,L	CB55	Z	..... de L
BIT 2,(HL)	CB56	Z	..... de dirección en HL
BIT 2,(IX+d)	DDCBdd56	Z	..... de dirección IX + D
BIT 2,(IY+d)	FDCBdd56	Z	..... de dirección IY + D
BIT 3,A	CB5F	Z	Comprueba bit 3 del ACU
BIT 3,B	CB58	Z	Comprueba bit 3 de B
BIT 3,C	CB59	Z	..... de C
BIT 3,D	CB5A	Z	..... de D
BIT 3,E	CB5B	Z	..... de E
BIT 3,H	CB5C	Z	..... de H
BIT 3,L	CB5D	Z	..... de L
BIT 3,(HL)	CB5E	Z	..... de dirección en HL
BIT 3,(IX+d)	DDCBdd5E	Z	..... de dirección IX + D
BIT 3,(IY+d)	FDCBdd5E	Z	..... de dirección IY + D
BIT 4,A	CB67	Z	Comprueba bit 4 del ACU
BIT 4,B	CB60	Z	Comprueba bit 4 de B
BIT 4,C	CB61	Z	..... de C
BIT 4,D	CB62	Z	..... de D
BIT 4,E	CB63	Z	..... de E
BIT 4,H	CB64	Z	..... de H
BIT 4,L	CB65	Z	..... de L
BIT 4,(HL)	CB66	Z	..... de dirección en HL
BIT 4,(IX+d)	DDCBdd66	Z	..... de dirección IX + D
BIT 4,(IY+d)	FDCBdd66	Z	..... de dirección IY + D
BIT 5,A	CB6F	Z	Comprueba bit 5 del ACU
BIT 5,B	CB68	Z	Comprueba bit 5 de B
BIT 5,C	CB69	Z	..... de C
BIT 5,D	CB6A	Z	..... de D
BIT 5,E	CB6B	Z	..... de E
BIT 5,H	CB6C	Z	..... de H

BIT 5,L	CB6D	Z	..... de L
BIT 5,(HL)	CB6E	Z	..... de dirección en HL
BIT 5,(IX+d)	DDCBdd6E	Z	..... de dirección IX + D
BIT 5,(IY+d)	FDCBdd6E	Z	..... de dirección IY + D
BIT 6,A	CB77	Z	Comprueba bit 6 del ACU
BIT 6,B	CB70	Z	Comprueba bit 6 de B
BIT 6,C	CB71	Z	..... de C
BIT 6,D	CB72	Z	..... de D
BIT 6,E	CB73	Z	..... de E
BIT 6,H	CB74	Z	..... de H
BIT 6,L	CB75	Z	..... de L
BIT 6,(HL)	CB76	Z	..... de dirección en HL
BIT 6,(IX+d)	DDCBdd76	Z	..... de dirección IX + D
BIT 6,(IY+d)	FDCBdd76	Z	..... de dirección IY + D
BIT 7,A	CB7F	Z	Comprueba bit 7 del ACU
BIT 7,B	CB78	Z	Comprueba bit 7 de B
BIT 7,C	CB79	Z	..... de C
BIT 7,D	CB7A	Z	..... de D
BIT 7,E	CB7B	Z	..... de E
BIT 7,H	CB7C	Z	..... de H
BIT 7,L	CB7D	Z	..... de L
BIT 7,(HL)	CB7E	Z	..... de dirección en HL
BIT 7,(IX+d)	DDCBdd7E	Z	..... de dirección IX + D
BIT 7,(IY+d)	FDCBdd7E	Z	..... de dirección IY + D
CALL nn	CDnnnn		Llama subrutina -
auf			nnnn
CALL C,nn	DCnnnn		..... si Carry=1
CALL M,nn	FCnnnn		..... si S=1 (menos)
CALL NC,nn	D4nnnn		..... si Carry=0
CALL NZ,nn	C4nnnn		..... si Z=0
CALL P,nn	F4nnnn		..... si S=0 (más)
CALL PE,nn	ECnnnn		..... si P=1
CALL PO,nn	E4nnnn		..... si P=0
CALL Z,nn	CCnnnn		..... si Z=1
CCF	3F	C	Complementa flag Carry
CP A	BF	S Z P C	Compara A con A
CP B	B8	S Z P C	..... con B
CP C	B9	S Z P C	..... con C
CP D	BA	S Z P C	..... con D
CP E	BB	S Z P C	..... con E

CP H	BC	S Z P C	..... con H
CP L	BD	S Z P C	..... con L
CP n	FEnn	S Z P C	..... con byte indicado
CP (HL)	BE	S Z P C	..... con dirección en HL
CP (IX+d)	DDBEdd	S Z P C	..... con dirección IX + D
CP (IY+d)	FDBEdd	S Z P C	..... con dirección IY + D
CPD	EDA9	S Z P	Compara y decrementa
CPDR	EDB9	S Z P	Compara bloque y decrementa
CPI	EDA1	S Z P	Compara e incrementa
CPIR	EDB1	S Z P	Compara bloque e incrementa
CPL	2F		Complementa ACU
DAA	27	S Z P C	Adaptación BCD del ACU
DEC A	3D	S Z P	Decrementa A
DEC B	05	S Z P	..... B
DEC BC	0B		..... BC
DEC C	0D	S Z P	 C
DEC D	15	S Z P	..... D
DEC DE	1B		..... DE
DEC E	1D	S Z P	..... E
DEC H	25	S Z P	..... H
DEC HL	2B		..... HL
DEC IX	DD2B		 IX
DEC IY	FD2B		..... IY
DEC L	2D	S Z P	 L
DEC SP	3B		..... SP
DEC (HL)	35	S Z P	..... dirección en HL
DEC (IX+d)	DD35dd	S Z P	..... dirección IX + D
DEC (IY+d)	FD35dd	S Z P	..... dirección IY + D
DI	F3		Bloquea interrupciones
DJNZ e	10ee		decrementa y bifurca por ()0
EI	FB		liberar interrupciones
EX AF,AF"	08		intercambia A/flags con registro secundario
Zweitre			
EX DE,HL	EB		intercambia DE y HL
EX (SP),HL	E3		intercambia byte del stack con HL
Stapelbyte			
EX (SP),IX	DDE3		 IX
EX (SP),IY	FDE3		..... IY
EXX	D9		intercambia BC, DE, HL con registro secundario

HALT	76		Detiene Z80 hasta interrupt
IM 0	ED46		Modalidad interrupt 0
IM 1	ED56		Modalidad interrupt 1
IM 2	ED5E		Modalidad interrupt 2
IN A,(C)	ED78	S Z P	Lee byte de puerto C en A
IN A,(n)	DBnn		..... de Puerto n
IN B,(C)	ED40	S Z P	..... en B
IN C,(C)	ED48	S Z P	..... en C
IN D,(C)	ED50	S Z P	..... en D
IN E,(C)	ED58	S Z P	..... en E
IN H,(C)	ED60	S Z P	..... en H
IN L,(C)	ED68	S Z P	..... en L
INC A	3C	S Z P	Incrementa A
INC B	04	S Z P	..... B
INC BC	03		..... BC
INC C	0C	S Z P	 C
INC D	14	S Z P	..... D
INC DE	13		..... DE
INC E	1C	S Z P	..... E
INC H	24	S Z P	..... H
INC HL	23		..... HL
INC IX	DD23		 IX
INC IY	FD23		..... IY
INC L	2C	S Z P	 L
INC SP	33		..... SP
INC (HL)	34	S Z P	..... dirección en HL
INC (IX+d)	DD34dd	S Z P	..... dirección IX + D
INC (IY+d)	FD34dd	S Z P	..... dirección IY + D
IND	EDAA	Z	Lectura con decremento
INDR	EDBA	Z=1	Lectura bloque con decremento
INI	EDA2	Z	Lectura con incremento
INIR	EDB2	Z=1	Lectura bloque con incremento
JP C,nn	DAnnnn		Bifurca a nn, si C=1
JP M,nn	FAnnnn		..... a nn, si S=1
JP NC,nn	D2nnnn		..... a nn, si C=0
JP nn	C3nnnn		..... a nn
JP NZ,nn	C2nnnn		..... a nn, si Z=0
JP P,nn	F2nnnn		..... a nn, si S=0
JP PE,nn	EAnnnn		..... a nn, si P=1
JP PO,nn	E2nnnn		..... a nn, si P=0

JP Z,nn	CAnnnn		..... a nn, si Z=1
JP (HL)	E9		..... a dirección en HL
JP (IX)	DDE9		..... a dirección IX + D
JP (IY)	FDE9		..... a dirección IY + D
JR C,e	38ee		Bifurque relativo, si C=1
JR e	18ee		..... a PC+e
JR NC,e	30ee		..... si C=0
JR NZ,e	20ee		..... si Z=0
JR Z,e	28ee		..... si Z=1
LD A,A	7F		Carga ACU con ACU
LD A,B	78		..... con B
LD A,C	79		..... con C
LD A,D	7A		..... con D
LD A,E	7B		..... con E
LD A,H	7C		..... con H
LD A,I	ED57	S Z P	..... con I
LD A,L	7D		..... con L
LD A,n	3Enn		..... con byte indicado
LD A,R	ED5F	S Z P	..... con R
LD A,(BC)	0A		..... con dirección en BC
LD A,(DE)	1A		..... con dirección en DE
LD A,(HL)	7E		..... con dirección en HL
LD A,(IX+d)	DD7Edd		..... con dirección IX + D
LD A,(IY+d)	FD7Edd		..... con dirección IY + D
LD A,(nn)	3Annnn		..... con dirección nn
LD B,A	47		Carga B con ACU
LD B,B	40		..... con B
LD B,C	41		..... con C
LD B,D	42		..... con D
LD B,E	43		..... con E
LD B,H	44		..... con H
LD B,L	45		..... con L
LD B,n	06nn		..... con byte indicado
LD B,(HL)	46		..... con dirección en HL
LD B,(IX+d)	DD46dd		..... con dirección IX + D
LD B,(IY+d)	FD46dd		..... con dirección IY + D
LD BC,nn	01nnnn		Carga BC con bytes indicados

LD BC,(nn)	ED4Bnnnn	..... con nnnn y nnnn + 1
LD C,A	4F	Carga C con ACU
LD C,B	48	..... con B
LD C,C	49	..... con C
LD C,D	4A	..... con D
LD C,E	4B	..... con E
LD C,H	4C	..... con H
LD C,L	4D	..... con L
LD C,n	OEnn	..... con byte
Byte		indicado
LD C,(HL)	4E	..... con dirección en HL
LD C,(IX+d)	DD4Edd	..... con dirección IX + D
LD C,(IY+d)	FD4Edd	..... con dirección IY + D
LD D,A	57	Carga D con ACU
LD D,B	50	..... con B
LD D,C	51	..... con C
LD D,D	52	..... con D
LD D,E	53	..... con E
LD D,H	54	..... con H
LD D,L	55	..... con L
LD D,n	16nn	..... con byte
		indicado
LD D,(HL)	56	..... con dirección en HL
LD D,(IX+d)	DD56dd	..... con dirección IX + D
LD D,(IY+d)	FD56dd	..... con dirección IY + D
LD DE,nn	11nnnn	Carga DE con byte
		indicado
LD DE,(nn)	ED5Bnnnn	..... con nnnn y nnnn + 1
LD E,A	5F	Carga E con ACU
LD E,B	58	..... con B
LD E,C	59	..... con C
LD E,D	5A	..... con D
LD E,E	5B	..... con E
LD E,H	5C	..... con H
LD E,L	5D	..... con L
LD E,n	1Enn	..... con byte
Byte		indicado
LD E,(HL)	5E	..... con dirección en HL
LD E,(IX+d)	DD5Edd	..... con dirección IX + D
LD E,(IY+d)	FD5Edd	..... con dirección IY + D

LD H,A	67	Carga H con ACU
LD H,B	60	..... con B
LD H,C	61	..... con C
LD H,D	62	..... con D
LD H,E	63	..... con E
LD H,H	64	..... con H
LD H,L	65	..... con L
LD H,n	26nn	..... con byte indicado
LD H,(HL)	66	..... con dirección en HL
LD H,(IX+d)	DD66dd	..... con dirección IX + D
LD H,(IY+d)	FD66dd	..... con dirección IY + D
LD HL,nn	21nnnn	Carga HL con bytes indicados
LD HL,(nn)	2Annnn	..... con nnnn y nnnn + 1
LD I,A	ED47	Carga I con ACU
LD IX,nn	DD21nnnn	Carga IX con bytes indicados
LD IX,(nn)	DD2Annnn	..... con nnnn y nnnn + 1
LD IY,nn	FD21nnnn	Cagra IY con bytes indicados
LD IY,(nn)	FD2Annnn	..... con nnnn y nnnn + 1
LD L,A	6F	Carga L con ACU
LD L,B	68	..... con B
LD L,C	69	..... con C
LD L,D	6A	..... con D
LD L,E	6B	..... con E
LD L,H	6C	..... con H
LD L,L	6D	..... con L
LD L,n	2Enn	..... con byte indicado
LD L,(HL)	6E	..... con dirección en HL
LD L,(IX+d)	DD6Edd	..... con dirección IX + D
LD L,(IY+d)	FD6Edd	..... con dirección IY + D
LD R,A	ED4F	Carga R con ACU
LD SP,HL	F9	Carga SP con HL
LD SP,IX	DDF9	..... con IX
LD SP,IY	FDF9	..... con IY
LD SP,nn	31nnnn	..... con bytes indicados
LD SP,(nn)	ED7Bnnnn	..... con nnnn y nnnn + 1

LD (BC),A	02	Carga dirección de BC con ACU
LD (DE),A	12	Carga dirección de DE con ACU
LD (HL),A	77	Carga dirección de HL con ACU
LD (HL),B	70	..... con B
LD (HL),C	71	..... con C
LD (HL),D	72	..... con D
LD (HL),E	73	..... con E
LD (HL),H	74	..... con H
LD (HL),L	75	..... con L
LD (HL),n	36nn	..... con byte indicado
LD (IX+d),A	DD77dd	Carga dirección IX + D con ACU
LD (IX+d),B	DD70dd	..... con B
LD (IX+d),C	DD71dd	..... con C
LD (IX+d),D	DD72dd	..... con D
LD (IX+d),E	DD73dd	..... con E
LD (IX+d),H	DD74dd	..... con H
LD (IX+d),L	DD75dd	..... con L
LD (IX+d),n	DD36ddnn	..... con byte indicado
LD (IY+d),A	FD77dd	Carga dirección IY+D con ACU
LD (IY+d),B	FD70dd	..... con B
LD (IY+d),C	FD71dd	..... con C
LD (IY+d),D	FD72dd	..... con D
LD (IY+d),E	FD73dd	..... con E
LD (IY+d),H	FD74dd	..... con H
LD (IY+d),L	FD75dd	..... con L
LD (IY+d),n	FD36ddnn	..... con byte indicado
LD (nn),A	32nnnn	Carga célula nnnn con ACU
LD (nn),BC	ED43nnnn	..... con C y nnnn+1 con B
LD (nn),DE	ED53nnnn	..... con E y nnnn+1 con D
LD (nn),HL	22nnnn	..... con L y nnnn+1 con H
LD (nn),IX	DD22nnnn	..... y nnnn+1 con IX

LD (nn),IY	FD22nnnn		..... y nnnn+1 con IY
LD (nn),SP	ED73nnnn		..... y nnnn+1 con SP
LDD	EDA8	P	Cargar y decrementar
LDDR	EDB8	P=0	Cargar bloque y decrementar
LDI	EDAO	P	Cargar e incrementar
LDIR	EDBO	P=0	Cargar bloque e incrementar
NEG	ED44	S Z P C	Invierte ACU
NOP	OO		Espera (operación nula)
OR A	B7	S Z P C	Interrelación OR ACU con ACU
OR B	BO	S Z P C	..... con B
OR C	B1	S Z P C	..... con C
OR D	B2	S Z P C	..... con D
OR E	B3	S Z P C	..... con E
OR H	B4	S Z P C	..... con H
OR L	B5	S Z P C	..... con L
OR n	F6nn	S Z P C	..... con byte indicado
OR (HL)	B6	S Z P C	..... dirección en HL
OR (IX+d)	DDB6dd	S Z P C	..... dirección en IX+D
OR (IY+d)	FDB6dd	S Z P C	..... dirección en IY+D
OTDR	EDBB	S Z P	Salida bloque y decrementa
OTIR	EDB3	S Z P	Salida bloque e incrementa
OUT (C),A	ED79		Envía a puerto C el ACU
OUT (C),B	ED41		..... el B
OUT (C),C	ED49		..... el C
OUT (C),D	ED51		..... el D
OUT (C),E	ED59		..... el E
OUT (C),H	ED61		..... el H
OUT (C),L	ED69		..... el L
OUT (n),A	D3nn		Envía ACU a puerto n
OUTD	EDAB	S Z P	Salida y decremento
OUTI	EDA3	S Z P	Salida e incremento
POP AF	F1		Extrae ACU y flags del Stack
POP BC	C1		Extrae BC del Stack
POP DE	D1		Extrae DE del Stack
POP HL	E1		Extrae HL del Stack
POP IX	DDE1		Extrae IX del Stack
POP IY	FDE1		Extrae IY del Stack
PUSH AF	F5		Lleva ACU y flags al

Stack		Stack
PUSH BC	C5	Lleva BC al Stack
PUSH DE	D5	Lleva DE al Stack
PUSH HL	E5	Lleva HL al Stack
PUSH IX	DDE5	Lleva IX al Stack
PUSH IY	FDE5	Lleva IY al Stack
RES 0,A	CB87	Borra bit 0 del ACU
RES 0,B	CB80	Borra bit 0 de B
RES 0,C	CB81	..... de C
RES 0,D	CB82	..... de D
RES 0,E	CB83	..... de E
RES 0,H	CB84	..... de H
RES 0,L	CB85	..... de L
RES 0,(HL)	CB86	..... de dirección en HL
RES 0,(IX+d)	DDCBdd86	..... de dirección IX+D
RES 0,(IY+d)	FDCBdd86	..... de dirección IY+D
RES 1,A	CB8F	Borra bit 1 del ACU
RES 1,B	CB88	Borra bit 1 de B
RES 1,C	CB89	..... de C
RES 1,D	CB8A	..... de D
RES 1,E	CB8B	..... de E
RES 1,H	CB8C	..... de H
RES 1,L	CB8D	..... de L
RES 1,(HL)	CB8E	..... de dirección en HL
RES 1,(IX+d)	DDCBdd8E	..... de dirección IX+D
RES 1,(IY+d)	FDCBdd8E	..... de dirección IY+D
RES 2,A	CB97	Borra bit 2 del ACU
RES 2,B	CB90	Borra bit 2 de B
RES 2,C	CB91	..... de C
RES 2,D	CB92	..... de D
RES 2,E	CB93	..... de E
RES 2,H	CB94	..... de H
RES 2,L	CB95	..... de L
RES 2,(HL)	CB96	..... de dirección en HL
RES 2,(IX+d)	DDCBdd96	..... de dirección IX+D
RES 2,(IY+d)	FDCBdd96	..... de dirección IY+D
RES 3,A	CB9F	Borra bit 3 del ACU
RES 3,B	CB98	Borra bit 3 de B
RES 3,C	CB99	..... de C
RES 3,D	CB9A	..... de D

RES 3,E	CB9B	..... de E
RES 3,H	CB9C	..... de H
RES 3,L	CB9D	..... de L
RES 3,(HL)	CB9E	..... de dirección en HL
RES 3,(IX+d)	DDCBdd9E	..... de dirección IX+D
RES 3,(IY+d)	FDCBdd9E	..... de dirección IY+D
RES 4,A	CBA7	Borra bit 4 del ACU
RES 4,B	CBA0	Borra bit 4 de B
RES 4,C	CBA1	..... de C
RES 4,D	CBA2	..... de D
RES 4,E	CBA3	..... de E
RES 4,H	CBA4	..... de H
RES 4,L	CBA5	..... de L
RES 4,(HL)	CBA6	..... de dirección en HL
RES 4,(IX+d)	DDCBddA6	..... de dirección IX+D
RES 4,(IY+d)	FDCBddA6	..... de dirección IY+D
RES 5,A	CBAF	Borra bit 5 del ACU
RES 5,B	CBA8	Borra bit 5 de B
RES 5,C	CBA9	..... de C
RES 5,D	CBAA	..... de D
RES 5,E	CBAB	..... de E
RES 5,H	CBAC	..... de H
RES 5,L	CBAD	..... de L
RES 5,(HL)	CBAE	..... de dirección en HL
RES 5,(IX+d)	DDCBddAE	..... de dirección IX+D
RES 5,(IY+d)	FDCBddAE	..... de dirección IY+D
RES 6,A	CBB7	Borra bit 6 del ACU
RES 6,B	CBB0	Borra bit 6 de B
RES 6,C	CBB1	..... de C
RES 6,D	CBB2	..... de D
RES 6,E	CBB3	..... de E
RES 6,H	CBB4	..... de H
RES 6,L	CBB5	..... de L
RES 6,(HL)	CBB6	..... de dirección en HL
RES 6,(IX+d)	DDCBddB6	..... de dirección IX+D
RES 6,(IY+d)	FDCBddB6	..... de dirección IY+D
RES 7,A	CBBF	Borra bit 7 del ACU
RES 7,B	CBB8	Borra bit 7 de B
RES 7,C	CBB9	..... de C
RES 7,D	CBBA	..... de D

RES 7,E	CBBB					..... de E
RES 7,H	CBBC					..... de H
RES 7,L	CBBD					..... de L
RES 7,(HL)	CBBE					..... de dirección en HL
RES 7,(IX+d)	DDCBddBE					..... de dirección IX+D
RES 7,(IY+d)	FDCBddBE					..... de dirección IY+D
RET	C9					Retorno de subrutina
RET C	D8					Retorno si C=1
RET M	F8					..... si S=1
RET NC	D0					..... si C=0
RET NZ	C0					..... si Z=0
RET P	F0					..... si S=0
RET PE	E8					..... si P=1
RET PO	E0					..... si P=0
RET Z	C8					..... si Z=1
RETI	ED4D					Retorno de la interrupción
RETN	ED45					..... de rutina NMI
RL A	CB17	S	Z	P	C	Rotación izquierda Carry - A
RL B	CB10	S	Z	P	C	..... y B
RL C	CB11	S	Z	P	C	..... y C
RL D	CB12	S	Z	P	C	..... y D
RL E	CB13	S	Z	P	C	..... y E
RL H	CB14	S	Z	P	C	..... y H
RL L	CB15	S	Z	P	C	..... y L
RL (HL)	CB16	S	Z	P	C	..... y dirección en HL
RL (IX+d)	DDCBdd16	S	Z	P	C	..... y dirección IX+D
RL (IY+d)	FDCBdd16	S	Z	P	C	..... y dirección IY+D
RLA	17				C	..... y ACU
RLC A	CB07	S	Z	P	C	Rotación izquierda del ACU
RLC B	CB00	S	Z	P	C	..... de B
RLC C	CB01	S	Z	P	C	..... de C
RLC D	CB02	S	Z	P	C	..... de D
RLC E	CB03	S	Z	P	C	..... de E
RLC H	CB04	S	Z	P	C	..... de H
RLC L	CB05	S	Z	P	C	..... de L
RLC (HL)	CB06	S	Z	P	C	..... de dirección en HL
RLC (IX+d)	DDCBdd06	S	Z	P	C	..... de dirección IX+D
RLC (IY+d)	FDCBdd06	S	Z	P	C	..... de dirección IY+D
RLCA	07				C	..... de ACU
RLD	ED6F	S	Z	P		Rotación izquierda decimal

RR A	CB1F	S	Z	P	C	Rotación derecha de Carry
A						y A
RR B	CB18	S	Z	P	C	..... y B
RR C	CB19	S	Z	P	C	..... y C
RR D	CB1A	S	Z	P	C	..... y D
RR E	CB1B	S	Z	P	C	..... y E
RR H	CB1C	S	Z	P	C	..... y H
RR L	CB1D	S	Z	P	C	..... y L
RR (HL)	CB1E	S	Z	P	C	..... y dirección en HL
RR (IX+d)	DDCBdd1E	S	Z	P	C	..... y dirección IX+D
RR (IY+d)	FDCBdd1E	S	Z	P	C	..... y dirección IY+D
RRA	1F				C	..... y ACU
RRC A	CB0F	S	Z	P	C	Rotación derecha del ACU
RRC B	CB08	S	Z	P	C	..... de B
RRC C	CB09	S	Z	P	C	..... de C
RRC D	CB0A	S	Z	P	C	..... de D
RRC E	CB0B	S	Z	P	C	..... de E
RRC H	CB0C	S	Z	P	C	..... de H
RRC L	CB0D	S	Z	P	C	..... de L
RRC (HL)	CB0E	S	Z	P	C	..... de dirección en HL
RRC (IX+d)	DDCBdd0E	S	Z	P	C	..... de dirección IX+D
RRC (IY+d)	FDCBdd0E	S	Z	P	C	..... de dirección IY+D
RRCA	0F				C	..... de ACU
RRD	ED67	S	Z	P		Rotación derecha decimal
RST 00	C7					Llamada subrutina en 00
RST 08	CF					..... en 08
RST 10	D7					..... en 10
RST 18	DF					..... en 18
RST 20	E7					..... en 20
RST 28	EF					..... en 28
RST 30	F7					..... en 30
RST 38	FF					..... en 38
SBC A,A	9F	S	Z	P	C	Sustracción Carry y A
A						de A
SBC A,B	98	S	Z	P	C	..... B del ACU
SBC A,C	99	S	Z	P	C	..... C del ACU
SBC A,D	9A	S	Z	P	C	..... D del ACU
SBC A,E	9B	S	Z	P	C	..... E del ACU
SBC A,H	9C	S	Z	P	C	..... H del ACU
SBC A,L	9D	S	Z	P	C	..... L del ACU

SBC A,n	DEnn	S	Z	P	C	.....	byte indicado del ACU
SBC A,(HL)	9E	S	Z	P	C	.....	dirección en HL de A
SBC A,(IX+d)	DD9Edd	S	Z	P	C	.....	dirección IX+D del
Akku						ACU	
SBC A,(IY+d)	FD9Edd	S	Z	P	C	.....	dirección IY+D del
Akku						ACU	
SBC HL,BC	ED42	S	Z	P	C	Sustracción C y BC de HL	
SBC HL,DE	ED52	S	Z	P	C	.....	DE de HL
SBC HL,HL	ED62	S	Z	P	C	.....	HL de HL
SBC HL,SP	ED72	S	Z	P	C	.....	SP de HL
SCF	37				C=1	Activa bit Carry en 1	
SET 0,A	CBC7					Activa bit 0 del ACU	
SET 0,B	CBC0					Activa bit 0 de B	
SET 0,C	CBC1					.....	de C
SET 0,D	CBC2					.....	de D
SET 0,E	CBC3					.....	de E
SET 0,H	CBC4					.....	de H
SET 0,L	CBC5					.....	de L
SET 0,(HL)	CBC6					.....	de dirección en HL
SET 0,(IX+d)	DDCBddC6					.....	de dirección IX+D
SET 0,(IY+d)	FDCBddC6					.....	de dirección IY+D
SET 1,A	CBCF					Activa bit 1 del ACU	
SET 1,B	CBC8					Activa bit 1 de B	
SET 1,C	CBC9					.....	de C
SET 1,D	CBCA					.....	de D
SET 1,E	CBCB					.....	de E
SET 1,H	CBCC					.....	de H
SET 1,L	CBCD					.....	de L
SET 1,(HL)	CBCE					.....	de dirección en HL
SET 1,(IX+d)	DDCBddCE					.....	de dirección IX+D
SET 1,(IY+d)	FDCBddCE					.....	de dirección IY+D
SET 2,A	CBD7					Activa bit 2 del ACU	
SET 2,B	CBD0					Activa bit 2 de B	
SET 2,C	CBD1					.....	de C
SET 2,D	CBD2					.....	de D
SET 2,E	CBD3					.....	de E
SET 2,H	CBD4					.....	de H
SET 2,L	CBD5					.....	de L
SET 2,(HL)	CBD6					.....	de dirección en HL
SET 2,(IX+d)	DDCBddD6					.....	de dirección IX+D

SET 2,(IY+d)	FDCBddD6	..... de dirección IY+D
SET 3,A	CBDF	Activa bit 3 del ACU
SET 3,B	CBD8	Activa bit 3 de B
SET 3,C	CBD9	..... de C
SET 3,D	CBDA	..... de D
SET 3,E	CBDB	..... de E
SET 3,H	CBDC	..... de H
SET 3,L	CBDD	..... de L
SET 3,(HL)	CBDE	..... de dirección en HL
SET 3,(IX+d)	DDCBddDE	..... de dirección IX+D
SET 3,(IY+d)	FDCBddDE	..... de dirección IY+D
SET 4,A	CBE7	Activa bit 4 del ACU
SET 4,B	CBEO	Activa bit 4 de B
SET 4,C	CBE1	..... de C
SET 4,D	CBE2	..... de D
SET 4,E	CBE3	..... de E
SET 4,H	CBE4	..... de H
SET 4,L	CBE5	..... de L
SET 4,(HL)	CBE6	..... de dirección en HL
SET 4,(IX+d)	DDCBddE6	..... de dirección IX+D
SET 4,(IY+d)	FDCBddE6	..... de dirección IY+D
SET 5,A	CBEF	Activa bit 5 del ACU
SET 5,B	CBE8	Activa bit 5 de B
SET 5,C	CBE9	..... de C
SET 5,D	CBEA	..... de D
SET 5,E	CBEB	..... de E
SET 5,H	CBEC	..... de H
SET 5,L	CBED	..... de L
SET 5,(HL)	CBEE	..... de dirección en HL
SET 5,(IX+d)	DDCBddEE	..... de dirección IX+D
SET 5,(IY+d)	FDCBddEE	..... de dirección IY+D
SET 6,A	CBF7	Activa bit 6 del ACU
SET 6,B	CBFO	Activa bit 6 de B
SET 6,C	CBF1	..... de C
SET 6,D	CBF2	..... de D
SET 6,E	CBF3	..... de E
SET 6,H	CBF4	..... de H
SET 6,L	CBF5	..... de L
SET 6,(HL)	CBF6	..... de dirección en HL
SET 6,(IX+d)	DDCBddF6	..... de dirección IX+D

SET 6,(IY+d)	FDCBddF6		..... de dirección IY+D
SET 7,A	CBFF		Activa bit 7 del ACU
SET 7,B	CBF8		Activa bit 7 de B
SET 7,C	CBF9		..... de C
SET 7,D	CBFA		..... de D
SET 7,E	CBFB		..... de E
SET 7,H	CBFC		..... de H
SET 7,L	CBFD		..... de L
SET 7,(HL)	CBFE		..... de dirección en HL
SET 7,(IX+d)	DDCBddFE		..... de dirección IX+D
SET 7,(IY+d)	FDCBddFE		..... de dirección IY+D
SLA A	CB27	S Z P C	Desplaza Carry y ACU izquierda
SLA B	CB20	S Z P C	..... y B izquierda
SLA C	CB21	S Z P C	..... y C izquierda
SLA D	CB22	S Z P C	..... y D izquierda
SLA E	CB23	S Z P C	..... y E izquierda
SLA H	CB24	S Z P C	..... y H izquierda
SLA L	CB25	S Z P C	..... y L izquierda
SLA (HL)	CB26	S Z P C	..... y dirección en HL izquierda
SLA (IX+d)	DDCBdd26	S Z P C	..... y dir. IX+D izquierda
SLA (IY+d)	FDCBdd26	S Z P C	..... y dir. IY+D izquierda
SRA A	CB2F	S Z P C	..... y ACU derecha
SRA B	CB28	S Z P C	..... y B derecha
SRA C	CB29	S Z P C	..... y C derecha
SRA D	CB2A	S Z P C	..... y D derecha
SRA E	CB2B	S Z P C	..... y E derecha
SRA H	CB2C	S Z P C	..... y H derecha
SRA L	CB2D	S Z P C	..... y L derecha
SRA (HL)	CB2E	S Z P C	..... y dir. en HL derecha
SRA (IX+d)	DDCBdd2E	S Z P C	..... y dir. IX+D
rechts			derecha
SRA (IY+d)	FDCBdd2E	S Z P C	..... y dir. IY+D
rechts			derecha
SRL A	CB3F	S Z P C	..... y ACU derecha
SRL B	CB38	S Z P C	..... y B derecha
SRL C	CB39	S Z P C	..... y C derecha
SRL D	CB3A	S Z P C	..... y D derecha
SRL E	CB3B	S Z P C	..... y E derecha
SRL H	CB3C	S Z P C	..... y H derecha

SRL L	CB3D	S Z P C	..... y L derecha
SRL (HL)	CB3E	S Z P C	..... y dir. en HL derecha
SRL (IX+d)	DDCBdd3E	S Z P C	..... y dir. IX+D
rechts			derecha
SRL (IY+d)	FDCBdd3E	S Z P C	..... y dir. IY+D
			derecha
SUB A	97	S Z P C	Sustracción ACU de ACU
SUB B	90	S Z P C	..... B de ACU
SUB C	91	S Z P C	..... C de ACU
SUB D	92	S Z P C	..... D de ACU
SUB E	93	S Z P C	..... E de ACU
SUB H	94	S Z P C	..... H de ACU
SUB L	95	S Z P C	..... L de ACU
SUB n	D6nn	S Z P C	..... byte indicado
SUB (HL)	96	S Z P C	..... dirección en HL
SUB (IX+d)	DD96dd	S Z P C	..... dirección IX+D del
			ACU
SUB (IY+d)	FD96dd	S Z P C	..... dirección IY+D del
Akku			ACU
XOR A	AF	S Z P C=0	Interrelaciona XOR A y A
XOR B	A8	S Z P C=0	..... y B
XOR C	A9	S Z P C=0	..... y C
XOR D	AA	S Z P C=0	..... y D
XOR E	AB	S Z P C=0	..... y E
XOR H	AC	S Z P C=0	..... y H
XOR L	AD	S Z P C=0	..... y L
XOR n	EEnn	S Z P C=0	..... y byte
			indicado
XOR (HL)	AE	S Z P C=0	..... y dirección en HL
XOR (IX+d)	DDAEdd	S Z P C=0	..... y dirección IX+D
XOR (IY+d)	FDAEdd	S Z P C=0	..... y dirección IY+D

14. TRUCOS Y FORMULAS EN BASIC

Con los siguientes trucos y fórmulas pueden resolverse problemas delicados programando con mayor sencillez, pueden evitarse errores y ahorrar mucho tiempo. Comencemos con fórmulas matemáticas.

Posiblemente ya han observado que en el BASIC del CPC (como en otros ordenadores) sólo existen comandos para el logaritmo natural (LOG) y el logaritmo de base 10 (LOG 10). Afortunadamente hay una fórmula simple con la cual pueden calcularse logaritmos en base n. Es la siguiente:

```
LOG n (X) = LOG (X)/LOG (n)
```

También con las inversiones de las funciones trigonométricas el interpretador BASIC tiene problemas. Faltan el ARCOSENO y el ARCOCOSENO. Ambos pueden calcularse a través del ARCOTANGENTE (ATN) como sigue:

```
ARCSEN = ATN  (X / SQR(1-X*X))
ARCCOS = -ATN (X / SQR(1-X*X)) + PI/2
```

Para reducir fracciones debe calcularse el máximo común divisor de 2 números. Para ello se utiliza muchas veces el algoritmo de Euclides, que tiene como ventaja que trabaja muy rápido.
Las variables P y Q contienen los números cuyos M.C.D. quieren calcularse. A continuación se asigna el resto de la división P/Q a la variable R. P contiene el valor antiguo de Q, a Q se le asigna el valor de R. Este procedimiento se repite hasta que el resto quede a ceros. En este caso P contiene el M.C.D. buscado.
Esta explicación un poco árida la veremos mejor en un ejemplo:

```
*=========================================*
*  P *  Q *  R *                         *
*-------------*                         *
* 56 * 21 * 14 *     Valores iniciales   *
* 21 * 14 *  7 *     1er. paso           *
* 14 *  7 *  0 *     2do. paso           *
*  7 *  0 *    *     P contiene M.C.D.   *
*=========================================*
```

El listing es bastante corto:

```
10 P = Número 1: Q = Número 2: r = 1
20 WHILE r diferente 0: r=P MOD Q: P=Q: Q=r: WEND
```

El r=1 es importante porque de lo contrario la condición se cumpliría a continuación de WHILE, antes de ejecutar el bucle por primera vez.

Si en cualquier momento quiere saber cuantas posiciones tiene un número delante de la coma, utilice la fórmula siguiente:

```
S = INT(LOG 10(ABS(X)) + 1)
```

La función ABS también les permite utilizar números negativos porque LOG 10 está definido únicamente para valores positivos.

La siguiente fórmula les facilita un número al azar del área a hasta b, no (cómo de costumbre) del área 0 hasta b:

```
X = INT(RND(1)*(b-a+1)+a)
```

La siguiente advertencia se relaciona también con las matemáticas, aunque relativamente. La conversión de base de números hexadecimales y binarios mediante & y &X tiene un pequeño inconveniente. Si los números son mayores que &7FFF es decir si el bit de mayor valor (número 15) tiene un 1, obtenemos un valor negativo. En este caso sumen 65536 y obtendrán el "resultado real".

Muchos programas de tratamiento de texto se lucen porque un texto introducido puede formatearse de diferentes maneras. Entre estas posibilidades se encuentra también la visualización centrada del texto. Lo mismo puede realizarse mediante una simple fórmula BASIC:

```
PRINT TAB ((longitud de línea - LEN(TEXTO$))/2); TEXTO$
```

La longitud de línea varía según la modalidad, texto contiene la línea a visualizar.

El último truco se refiere a los strings. Se trata de una particularidad desagradable del CPC, la que comparte con otros interpretadores BASIC. Si intenta visualizar el valor ASCII de un string vacío (por ejemplo ASC("")), el ordenador reacciona con un mensaje "IMPROPER ARGUMENT". Esto puede evitarse mediante:

```
PRINT ASC(A$+CHR$(0))
```

De esta forma queda resuelto el problema.

ANEXO

1. PLANO DE OCUPACION DE MEMORIA

En las páginas siguientes, listamos las funciones de todos los bytes de memoria (conocidas desde un principio). Naturalmente puede existir algún error y por ello debe mirarse con cuidado al experimentar en la memoria con PEEK o POKE, no por ello dejen de probar con sus nuevas ideas -graben sus programas previamente!.

0000 - 3FFF	ROM Sistema operativo
0000 - 003F	Copia de ROM para Bankswitching
0040 - 013F	Buffer de entrada y área de trabajo
0170 - AB7F	Memoria BASIC
3800 - 3FFF	Generador de caracteres en ROM
AB80 - ABFF	Generador de caracteres definibles
AC00	Flag para borrar caracteres vacíos
AC01 - AC03	Bifurcación ampliación para modo READY
AC04 - AC06	Bifurcación ampliación para tratamiento ERROR
AC07 - AC09	Bifurcación ampliación ejecución comandos
AC0A - AC0C	Bifurcación ampliación cálculo función
AC0D - AC0F	Bifurcación ampliación extraer constante
AC10 - AC12	Bifurcación ampliación entrada línea BASIC
AC13 - AC15	Bifurcación ampliación para LIST
AC16 - AC18	Bifurcación ampliación conversión números
AC19 - AC1B	Bifurcación ampliación para operadores
AC1C	Flag para AUTO
AC1D - AC1E	Número línea AUTO
AC1F - AC20	Intervalo para AUTO
AC21	Ultimo número stream
AC22	Canal de entrada
AC23	Ultima posición de la impresora
AC24	WIDTH
AC25	Ultima posición en cassette

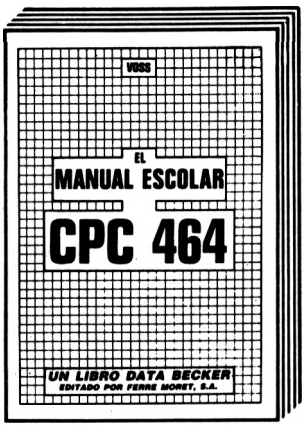
AC26	Flag para FOR-NEXT
AC27 - AC2B	Memoria intermedia para FOR
AC2C - AC2D	Dirección para NEXT
AC2E - AC2F	Dirección para WEND
AC34 - AC35	Dirección para ON-BREAK
AC38 - AC43	Secuencia tono 0
AC44 - AC4F	Secuencia tono 1
AC50 - AC5B	Secuencia tono 2
AC5C - AC6D	Area para interrupt 0
AC6E - AC7F	Area para interrupt 1
AC80 - AC91	Area para interrupt 2
AC92 - ACA3	Area para interrupt 3
ACA4 - ADA3	Buffer de entrada
ADA6 - ADA7	Dirección del error
ADA8 - ADA9	Apuntador programa BASIC después de un error
ADAA	Número del error
ADAB - ADAC	Apuntador programa BASIC después de interrupt
ADAD - ADAE	Dirección de línea interrumpida
ADAF - ADB0	Dirección para ON ERROR
ADB1	Flag: ON-ERROR activo
ADB2	Estado de canal
ADB3	ENT
ADB4	ENV
ADB5 - ADB6	Período
ADB7	Período de zumbido
ADB8	Volumen
ADB9 - ADBA	Longitud
ADBB - ADBC	ENV y ENT
ADCB - ADCF	Memoria temporal aritmética coma flotante
ADD0 - AE03	Tabla para variables escalares
AE04 - AE05	Tabla FN
AE06 - AE0B	Tabla Array
AE0C - AE25	Tipo variables pre-definidas A hasta Z
AE2D	Marca separación para INPUT
AE2E - AE2F	Dirección para READ
AE30 - AE31	Dirección para DATA
AE32 - AE33	Memoria para apuntador en BASIC stack
AE34 - AE35	Dirección actual de comando
AE36 - AE37	Dirección actuaL de línea de programa
AE38	Flag para TRACE

AE3F - AE40	Dirección inicial comando LOAD
AE41	Flag para CHAIN MERGE
AE42	Tipo fichero
AE43 - AE44	Longitud fichero
AE45	Flag protección programa
AE46 - AE78	Area de trabajo conversión a ASCII
AE72 - AE73	Dirección CALL
AE74	Modalidad Bankswitching para CALL
AE75 - AE76	Registro HL para CALL
AE77 - AE78	Registro SP para CALL
AE79	Intervalo tabulador
AE7B - AE7C	Apuntador HIMEM
AE7D - AE7E	Apuntador al final área libre de RAM
AE7F - AE80	Apuntador a comienzo área libre de RAM
AE81 - AE82	Apuntador a comienzo programa BASIC
AE83 - AE84	Apuntador a final programa
AE85 - AE86	Apuntador a inicio variables
AE87 - AE88	Apuntador a inicio tabla
AE89 - AE8A	Apuntador a final tabla
AE8B - B08A	Stack para BASIC (FOR, GOSUB, etc.)
B08B - B08C	Stack pointer BASIC
B08D - B08E	Apuntador a inicio string
B08F - B090	Apuntador a final string
B09A - B09B	Apuntador a stack del string
B09C - B0B9	Stack del string
B0BA - B0BC	Descriptor de string
B0C1	Tipo variable
B0C2 - B0C3	Direcciones diversas
B100 - B1AB	Area de trabajo para manejo sistema operativo
B1C8	Modalidad pantalla
B1CA	Offset pantalla
B1CC - B1D6	Diversas tareas
B1D7 - B1D8	Tiempos de parpadeo
B1D9 - B20B	Diversos registros para colores intermitentes
B20C	Ventana actual
B20D - B276	Parámetros para ventana
B285 - B286	Posición cursor (línea, columna)
B287 - B28B	Diversos registros para ventanas
B28C - B327	Diversos registros para pantalla

B328 - B329	ORIGIN-X
B32A - B32B	ORIGIN-Y
B32C - B4DD	Diversos registros para salida gráficos
B4DE - B550	Diversos registros para comprobación teclado
B551 - B7FF	Diversos registros para SOUND
B800 - B8DC	Diversos registros para cassette
B807 - B816	Nombre del fichero input
B84C - B85B	Nombre del fichero output
B8D1	Velocidades de grabación
B8DD	Flag insert
B8E4 - B8E7	Número al azar
B8E8 - B8F6	Memoria intermedia aritmética coma flotante
B8F7	Flag para DEG o RAD
BF00 - BFFF	Stack procesador
C000 - FFFF	Video-RAM
C000 - FFFF	ROM interpretador
C000 - FFFF	ROM de ampliación

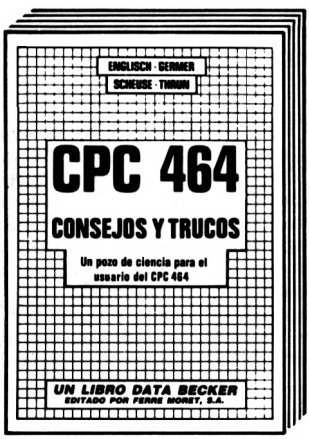
64 EN EL CAMPO DE LA TECNICA Y LA CIENCIA, 296 pág.
P.V.P. 2.800,- ptas.

Ofrece un campo fascinante y amplio de problemáticas científicas. Para esto el libro contiene muchos listados interesantes: Análisis de Fornier y síntesis, análisis de redes, exactitud de cálculo, formateado de números, cálculo del valor PH, sistemas de ecuaciones diferenciales, modelo ladrón presa, cálculo de probabilidad, medición de tiempo, integración, etc.



CPC-464 EL LIBRO DEL COLEGIO
P.V.P. 2.800,- ptas.

Escrito para alumnos de los últimos cursos de EGB y de BUP, este libro contiene muchos programas para resolver problemas y de aprendizaje, descritos de una forma muy compleja y fácil de comprender. Teorema de Pitágoras, progresiones geométricas, escritura cifrada, crecimiento exponencial, verbos irregulares, igualdades cuadráticas, movimiento pendular, estructura de moléculas, cálculo de interés y muchas cosas más.



CPC-464 CONSEJOS Y TRUCOS
P.V.P. 2.200,- ptas.

Ofrece una colección muy interesante de sugerencias, ideas y soluciones para la programación y utilización de su CPC-464: Desde la estructura del hardware, sistema de funcionamiento - Tokens Basic, dibujos con el joystick, aplicaciones de ventanas en pantalla y otros muchos interesantes programas como el procesamiento de datos, editor de sonidos, generador de caracteres, monitor de código máquina hasta listados de interesantes juegos.

ROBOTICA PARA SU COMMODORE 64, 230 pág.
P.V.P. 2.800,- ptas.

En el libro de los robots se muestran las asombrosas posibilidades que ofrece el CBM 64, para el control y la programación, presentadas con numerosas ilustraciones e intuitivos ejemplos. El punto principal: Cómo puede construirse uno mismo un robot sin grandes gastos. Además, un resumen del desarrollo histórico del robot y una amplia introducción a los fundamentos cibernéticos.
Gobierno del motor, el modelo de simulación, interruptor de pantalla, el Port-Usuario cómodo del modelo de simulación, Sensor de infrarrojos, concepto básico de un robot, realimentación unidad cibernética, Brazo prensor, Oír y ver.

MANUAL ESCOLAR PARA SU COMMODORE 64, 351 pág.
P.V.P. 2.800,- ptas.

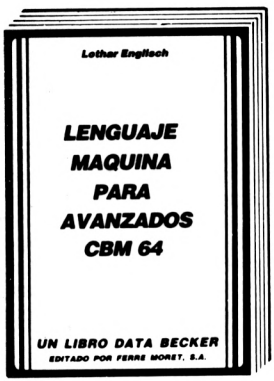
Este libro, escrito especialmente para escolares de grado medio y superior, contiene muchos interesantes programas de aprendizaje para solucionar problemas, descritos detalladamente y de manera fácilmente comprensible. Facilitan un aprendizaje intensivo y ameno, con, entre otros, los siguientes temas: Teorema de pitágoras, progresiones geométricas, palanca mecánica, crecimiento exponencial, verbos irregulares, ecuaciones de segundo grado, movimientos de péndulo, formación de moléculas, aprendizaje de vocablos, cálculo de interés y su capitalización. Una corta repetición de los elementos BASIC más importantes y una introducción a los rasgos esenciales del análisis de problemas, entre otros. completan el conjunto.

PEEKS y POKES, 177 pág.
P.V.P. 1.600,- ptas.

Con importantes comandos PEEK y POKE se pueden hacer también desde el Basic muchas cosas, para las que se necesitarían normalmente complejas rutinas en lenguaje máquina. Este libro explica de manera sencilla el manejo de PEEKs y POKEs. Con una enorme cantidad de POKEs importantes y su posible aplicación. Para ello se explica perfectamente la estructura del Commodore 64: Sistema operativo, interpretador, página cero, apuntadores y stacks, generador de caracteres, registros de sprites, programación de interfaces, desactivación del interrupt. Además una introducción al lenguaje máquina. Muchos programas ejemplo.

LENGUAJE MAQUINA PARA
COMMODORE 64, 1984, 201 pág.
P.V.P. 2.200,- ptas.

¡Por fin una introducción al código máquina fácilmente comprensible! Estructura y funcionamiento del procesador 6510, introducción y ejecución de programas en lenguaje máquina, manejo del ensamblador, y un atractivo muy especial: ¡un simulador de paso a paso escrito en BASIC!



LENGUAJE MAQUINA PARA AVANZADOS
CBM 64, 1984, 206 pág.
P.V.P. 2.200 ptas.

¿Ud. ha logrado iniciarse en código máquina? Entonces el «nuevo English» le enseñará cómo convertirse en un profesional. Naturalmente con muchos programas ejemplo, rutinas completas en código máquina e importantes consejos y trucos para la programación en lenguaje máquina y para el trabajo con el sistema operativo.

EL ENSAMBLADOR

Este libro ofrece al programador interesado una introducción fácilmente comprensible para los tan extendidos Assembler PROFI-ASS, SM MAE y T.EX.ASS. con consejos y trucos de gran utilidad, indicaciones y programas adicionales. Al mismo tiempo sirve de manual orientado a la práctica, con aclaraciones de conceptos importantes e instrucciones.
250 páginas. 2.200,- ptas.

TODO SOBRE EL FLOPPY 1541,
P.V.P. 3.200,- ptas.

La obra Standard del floppy 1541, todo sobre la programación en disquettes desde los principiantes a los profesionales, además de las informaciones fundamentales para el DOS, los comandos de sistema y mensajes de error, hay varios capítulos para la administración práctica de ficheros con el FLOPPY, amplio y documentado Listado del Dos. Además un filón de los más diversos programas y rutinas auxiliares, que hacen del libro una lectura obligada para los usuarios del Floppy.

MANTENIMIENTO Y REPARACION DEL FLOPPY 1541,
200 pág.
P.V.P. 2.800,- ptas.

Saberse apañar uno mismo, ahorra tiempo, molestias y dinero, precisamente problemas como el ajuste del floppy o reparaciones de la platina se pueden arreglar a menudo con medios sencillos. Instrucciones para eliminar la mayoría de perturbaciones, listas de piezas de recambio y una introducción a la mecánica y a la electrónica de la unidad de disco, hay también indicaciones exactas sobre herramientas y material de trabajo. Este libro hay que considerarlo en todos sus aspectos como efectivo y barato.

EL MANUAL DEL CASSETTE, 190 pág.
P.V.P. 1.600,- ptas.

Un excelente libro, que le mostrará todas las posibilidades que le ofrece su grabadora de casettes. Describe detalladamente, y de forma comprensible, todo sobre el Datassette y la grabación en cassette. Con verdaderos programas fuera de serie: Autostart, Catálogo (¡busca y carga automáticamente!), backup de y a disco, SAVE de áreas de memoria, y lo más sorprendente: un nuevo sistema operativo de cassette con el 10-20 veces más rápido FastTape. Además otras indicaciones y programas de utilidad (ajuste de cabezales, altavoz de control).

ZX SPECTRUM EL MANUAL ESCOLAR
P.V.P. 2.200,- ptas.

Escrito para alumnos de los últimos cursos de EGB y de BUP, este libro contiene muchos programas para resolver problemas y de aprendizaje, descritos de una forma muy completa y fácil de comprender. Teorema de Pitágoras, progresiones geométricas, escritura cifrada, crecimiento exponencial, verbos irregulares, igualdades cudráticas, movimiento pendular, estructura de moléculas, cálculo de interés y muchas cosas más.

ZX SPECTRUM CONSEJOS Y TRUCOS, 211 pág.
P.V.P. 2.200,- ptas.

Una interesante colección se sugestivas ideas y soluciones para la programación y utilización de su ZX ESPECTRUM. Aparte de muchos peeks, pokes y USRs hay también capítulos completos para, entre otros, entrada de datos asegurado sin bloqueo de ordenador, posibilidades de conexión y utilización de microdrives y lápices ópticos programas para la representación de diagramas de barra y de tarta, el modo de utilizar óptimamente ROM y RAM.

METODOLOGIA DE LA PROGRAMACION
P.V.P. 2.200,- ptas.

El primer libro recomendado para escuelas de enseñanza de informática y para aquellas personas que quieren aprender la programación. Cubre las especificaciones del Ministerio de Educación y Ciencia para Estudios de Informática. Realizado por un alto mando del ejército Español, un Dr. Ingeniero y Diplomado en Informática y profesor de la UNED y por un oficial técnico especialista en informática de gestión. Utilizado en todos los institutos politécnicos del ejército español. Es un seguro candidado a ediciones en lengua inglesa, alemana y francesa. Es el primer libro que introduce a la lógica del ordenador. Es un elemento de base que sirve como introducción para la programación en cualquier otro lenguaje. No se requieren conocimientos de programación ni siquiera de informática. Abarca desde los métodos de programación clásicos a los más modernos.

MSX_EL MANUAL ESCOLAR
P.V.P. 2.800,- ptas.

Escrito para alumnos de los últimos cursos de EGB y de BUP, este libro contiene muchos programas para resolver problemas y de aprendizaje, descritos de una forma muy completa y fácil de comprender. Teorema de Pitágoras, progresiones geométricas, escritura cifrada, crecimiento exponencial, verbos irregulares, igualdades cuadráticas, movimiento pendular, estructura de moléculas, cálculo de interés y muchas cosas más.

MSX GRAFICOS Y SONIDOS, 250 pág.
P.V.P. 2.800,- ptas.

Las computadoras MSX no sólo ofrecen una relación precio/rendimiento sobresaliente, sino que también poseen unas cualidades gráficas y de sonido excepcionales. Este libro expone las posibilidades de los MSX de forma completa y fácil. El texto se completa con numerosos y útiles programas ejemplo.

MSX PROGRAMAS Y UTILIDADES, 1985, 194 pág.
P.V.P. 2.200,- ptas.

El libro contiene una amplia colección de importantes programas que abarcan, desde un desensamblador hasta un programa de clasificaciones deportivas. Juegos superemocionantes y aplicaciones completas. Los programas muestran además importantes consejos y trucos para la programación. Estos programas funcionan en todos los ordenadores MSX, así como en el SPECTROVIDEO 318 328. ESTRACTO DEL CONTENIDO: Volcado memoria hexadecimal. Editor gráficos. Editor de sonido. Escritura de ordenador. Lista referencia de variables. Calendario. Desensamblador. ADMINISTRACION de una colección de discos L.P. HOLLOW - JUEGO DE LAS CEREZAS. DIAGRAMAS DE BARRAS. TABLAS DEPORTIVAS.

TODO SOBRE EL NUEVO COMMODORE 128

P.V.P. 2.200,- ptas.

El Libro de Primicias del Commodore 128 no ofrece solamente un resumen completo de todas las características y rendimientos del sucesor del C-64 y con ello una importante ayuda para su adquisición. Muestra, además, todas las posibilidades del nuevo equipo en función de sus tres modos de operación.

Entre otros se describen el hardware, los modos de operación: modo 64, modo 128 y modo CP/M, las configuraciones de memoria, la disposición de la página cero, trabajos con dos pantallas, modo de 80 caracteres, Basic V 7.0: comandos de gráficos y sonidos, comandos de control, periféricos rápidos (1571) etc.

RESPUESTA COMERCIAL

F.D. Autorización 6975
(B.O. de Correos N.º 80 de 26-7-85)

HOJA PEDIDO DE LIBRERIA

NO NECESITA SELLOS

A franquear en destino

FERRE MORET, S.A.

Apartado N.º 551. F.D.
08080 BARCELONA

RESPUESTA COMERCIAL

F.D. Autorización 6975
(B.O. de Correos N.º 80 de 26-7-85)

HOJA PEDIDO DE LIBRERIA

NO NECESITA SELLOS

A franquear en destino

FERRE MORET, S.A.

Apartado N.º 551. F.D.
08080 BARCELONA

Puesta al día de datos

EDITORIAL FERRER MORET, S.A. mantiene vivo y amplía el contenido informativo de sus libros y programas, mediante el envío de un servicio de puesta al día, junto con una síntesis noticiosa de la actualidad y perspectivas de la realidad informática española.
Agradecemos cualquier sugerencia o crítica que desee formular y que nos ayude a mejorar las ediciones. Muchas gracias.

¿Qué añadiría?
.....

.....

¿Qué suprimiría?
.....

.....

Observaciones
.....

.....

Título del libro
.....

Nombre
.....

Dirección Tfno.
.....

Código Postal y Población Provincia
.....

UN SERVICIO GRATUITO

Información

FERRE MORET, S.A. cuenta con un amplio fondo de libros y Software y mantiene un servicio de información por correo sobre las novedades que edita.
Agradecemos nos indique los temas que representan para Vd. mayor interes.

Libros

☐ ATARI	☐ MSX	☐ AMSTRAD
☐ COMMODORE	☐ LENGUAJES	☐ APPLE
☐ SINCLAIR	☐ IBM	☐ SOFTWARE

Si está interesado en recibir alguno de estos servicios, rellene y envíe la tarjeta correspondiente; no necesita franqueo. Muchas gracias.

DESEO RECIBIR EL LIBRO .....

EL PROGRAMA .....

☐ Adjunto cheque ☐ Contra reembolso
.....

Nombre
.....

Dirección Tfno.
.....

Código Postal y Población Provincia
.....

UN SERVICIO GRATUITO

EL CONTENIDO:

PEEKS, POKES y CALLS se utilizan para introducir al lector de una forma fácilmente accesible al sistema operativo y al lenguaje máquina del CPC. Proporciona además muchas e interesantes posibilidades de aplicación y programación de su CPC.

Extracto del contenido:

— Arquitectura Hardware del CPC
— Sistema operativo e interpretador
— PEEK & POKE - CALL
— Aritmética binaria
— Protección memoria
— Bankswitching - Lectura ROM
— Video-RAM - Gráficos - Scrolling
— Interrupciones BASIC
— Almacenamiento de sentencias BASIC
— Garbage collection
— Estructura y funcionamiento del Z-80
— Posibilidades de direccionamiento
— Rutinas máquinas de utilidad

ESTE LIBRO HA SIDO ESCRITO POR:

Hans Joachim Liesert, autor de literatura informática, que explica de forma fácilmente comprensible hasta los temas más complicados.

ISBN 84-86437-34-2

Liesert / Peeks, Pokes

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/