Dullin - Brassenburg

MSX

Consejos y Trucos

UN LIBRO DATA BECKER

EDITADO POR FERRE MORET, S.A.

Dullin - Brassenburg

MSX

Consejos y Trucos

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

Este libro ha sido traducido por D. Juan Jiménez Molina, doctor ingeniero industrial y experto conocedor del MSX.

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

ISBN 84-86437-39-3

Depósito legal **B-1529/86**

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, responsabilidad jurídica o cualquier otra responsabilidad, sobre las consecuencias atribuibles a datos erróneos. El autor les agradecerá en todo momento la comunicación de posibles fallos.

Preámbulo

¡Por fin está aquí!

La norma MSX para ordenadores personales. Para nosotros, un deseo ansiado se ha hecho realidad. Por fin existen más de 20 ordenadores, cuyo Soft - y Hardware son compatibles, o sea, intercambiables entre sí. La ventaja consiste en que -en el propio ordenador MSX- sólo se pueden aplicar sin problema, programas que hayan sido escritos con otros ordenadores MSX. Por esto, podrá existir ilimitadamente Software para el usuario dentro de un lapso de tiempo muy breve.

Las siglas MSX significan "Microsoft Extended Basic". Es obvio, que la prestigiosa empresa Microsoft ha desarrollado el Basic para este ordenador. El Basic MSX se distingue por su diversidad de comandos útiles. No faltan ni comandos para el apoyo de la gráfica y sonido, ni aquellos que facilitan la edición, como por ejemplo "TRON" o "TROFF".

Para que la programación con estos ordenadores le sea más agradable, hemos escrito este libro. Aquí encontrará muchos "trucos y consejos" útiles, que podrá aplicar efectivamente en sus programas. Además obtendrá información importante sobre la construcción interna y el lenguaje máquina de estos ordenadores.

Le deseamos mucha diversión y éxito al leer este libro.

Los autores

INDICE

CAPITULO IV : SONIDO CON EL CHIP DE SONIDO

CAPITULO V: LENGUAJE MAQUINA

CAPITULO VI : RUTINAS DEL SISTEMA

CAPITULO VII : PEEKS Y POKES

CAPITULO VIII : BASIC INTERNO

CAPITULO IX : VARIOS

CAPITULO X : PROGRAMAS

APENDICE

CAPITULO I : LA NORMA MSX

1.1 ¿Qué es una norma?

Nadie se para a pensar, si un televisor, radio, u otros aparatos eléctricos que haya comprado, pueden ser conectados a la red eléctrica de su casa. Es natural, que el enchufe del aparato eléctrico entre en el enchufe hembra. Solo en viajes al extranjero se puede pasar por momentos de asombro. Aquí se puede comprobar que, o la corriente suministrada tiene otra tensión, o el enchufe no entra. Entonces queda claro, si existe o no, al menos a nivel nacional, un acuerdo entre empresas sobre el desarrollo de una norma.

Una norma generalizada es casi natural en otros sectores técnicos. Pero en el sector de ordenadores personales, no es éste el caso, a causa del rápido desarrollo y, principalmente, por motivos de competencia. Para el usuario es frecuentemente complicado, si no imposible, equipar un ordenador con aparatos periféricos segun su elección. Frecuentemente sólo se pueden aplicar sin gran complicación los aparatos de la misma empresa que ha fabricado el ordenador.

Justamente cuando se trata de Software, este hecho es enojoso. Se oye de verdaderamente buenos programas, por ejemplo procesadores de texto, pero que no funcionan en el propio ordenador. Por ello, hay que llegar a un compromiso sobre el programa a utilizar.

Por lo tanto, la unión de más de veinte empresas para confeccionar la norma MSX es un desarrollo conveniente y agradable para el consumidor.

1.2 La norma

- Microprocesador de 8-Bit Zilog Z80A
- Chip de Vídeo Texas Instruments TMS 9918A
- Chip de Audio General Instruments AY-3-8910
- 32 KByte ROM
- Microsoft Extended Basic programado en el ROM
- Mínimo de 8 KByte RAM, preferiblemente 16 Kbyte
- Representación en pantalla de 40 caracteres por línea
- 16 diferentes colores
- Interfase para casette
- Canal de ampliación para módulo de memoria RAM
- Dos entradas para joystick
- Teclado específico para cada país con cinco teclas de función (con 4 ocupaciones)
- Cuatro teclas para mando del cursor
- Salida de monitor

Vamos a exponer brevemente las funciones de los módulos más importantes de estos ordenadores:

El procesador Z80 A

Como Vd. seguramente sabe, cada ordenador posee un microprocesador, que puede llamarse el "cerebro" del ordenador. Este IC (circuito integrado) se llama CPU (Central Processing Unit) o unidad central. La CPU ejecuta comandos de máquina, gobierna la elaboración en el ordenador y los aparatos externos (periferia). La unidad central es el módulo más importante en un ordenador.
Los ordenadores MSX disponen de un procesador Z80 A, que también se aplica en muchos otros ordenadores. El Z80 A es una unidad central muy potente que comprende más de 600 comandos que, en ordenadores MSX, son ejecutados con gran rapidez.

El fabricante de este microprocesador es la empresa Zilog situada en Silicon Valley, en California.

El Chip de Vídeo TMS 9918A de Texas Instruments

El Chip de Vídeo tiene como finalidad la producción de una imagen sobre un monitor o televisor. Además es responsable de la gráfica. También administra los diferentes niveles de Sprites, así cómo la RAM de vídeo de 16 K. Por su existencia, el procesador Z80A, que en algunos ordenadores también tiene que realizar estos trabajos, es descargado.

El Chip de Sonido AY-3-8910 de General Instruments

El Chip es, como ya lo dice el nombre, responsable del sonido, o sea, de la música. El Chip de sonido permite una amplitud de sonidos de 8 octavas. Muchos órganos disponen de una amplitud menor. Además se han incluido en él tres canales de sonido regulables. De esta forma es relativamente fácil tocar a tres voces. Además, este Chip es responsable del color del sonido, o sea, puede imitar instrumentos y ruidos hasta la perfección.

El Microsoft Extended Basic

El MSX Basic dispone de más de 100 comandos. Entre ellos existen comandos que permiten la entrada y la edición confortable de programas, así como comandos que activan especialmente los Chips de sonido o Vídeo. Naturalmente existen también comandos para cargar y memorizar sobre casette, disco, o para el mando de una impresora. Especialmente hay que mencionar las posibilidades de mando de los Interrupt.
En resumen, el MSX Basic está bien ideado. Por ello es agradable trabajar con él. Ampliaciones complicadas, como por ejemplo en el Commodore 64, no son necesarias con ordenadores MSX.

Mínimo de 8 K, preferiblemente 16 K RAM

Los ordenadores MSX más pequeños disponen de 8 K RAM. En esta norma se condicionan 16 K RAM. La tendencia se dirige claramente hacia las versiones de 64 K, o sea, en estos ordenadores se dispone de 80 K RAM sumando el Vídeoram.

1.3 LOS INTERFASES DE LOS ORDENADORES MSX

Naturalmente, los ordenadores MSX no pueden pasar sin aparatos periféricos, al igual que otros ordenadores. Por este motivo es conveniente saber de que conexiones se dispone para que aparatos.

Televisor:

La norma MSX permite la conexión de aparatos de televisión a través de la entrada de la antena. El port de entrada del ordenador corresponde a un conector normal de TV con 75 ohmios de resistencia. Lamentablemente, la imagen del televisor es más bien de calidad mediocre.

Monitor:

Los ordenadores MSX poseen una conexión para monitor, la cual transmite la imagen a la pantalla sin transformación a una frecuencia portadora. Las conexiones correspondientes 1 a 8 están normalizadas, pero no el tipo de conector. De esta forma, se depende de adaptadores, que se pueden obtener, sin problemas, en tiendas especializadas.
Adicionalmente se pueden obtener conexiones RGB. Con este tipo de conexión la imagen se divide en los tres colores básicos (rojo, verde, azul) y se vuelven a componer nuevamente en el monitor.

Además se transmite una señal de luminancia y un estado RGB. Así se consigue una calidad de imagen muy buena.

Cassette:

Para la memorización y la carga de programas en casette, se ha previsto un conector DIN de 5 polos en la parte posterior del ordenador. Las señales procedentes del ordenador se transmiten al Pin 4. El "Pin" es lo contrario de un pivote, o sea, un contacto en una clavija. La clavija y el conector forman una unidad enchufable. El traspaso de datos desde el casette, también llamado Datasette o grabador de datos, se realiza a través del Pin 5. Para parar o arrancar el motor del magnetófono se puede emitir una señal a través de los Pins 6 ó 7. Todos los Pins restantes están conectados a tierra y no tienen ninguna función especial.

Joystick:

Los ordenadores MSX disponen de dos conexiones previstas para joysticks y para Paddles. Lo que muchos no saben, es que estas conexiones también valen para otros aparatos, como por ejemplo mando de relés. Pero esto exige unos ciertos conocimientos electrónicos.

Discos:

A los Pins del cartucho ROM se conectan las unidades de discos a través de un interfase. La gran parte de fabricantes de MSX ya ofrecen aparatos de este tipo. Según nuestra impresión, se impondrá el disco de la empresa Sony. Es una unidad extremadamente rápida y sólida de 3 1/2 pulgadas. Las 3 1/2 pulgadas indican el tamaño de los discos. Estos discos caben perfectamente en el bolsillo del pantalón, y están protegidas contra el polvo por un cierre especial.

Como la norma MSX no es aplicable a aparatos periféricos, es conveniente informarse bien antes de adquirir aparatos de este tipo.
Una posibilidad especial respecto a la memorización y carga de datos con ordenadores MSX, será en un futuro cercano la placa de imagen. La placa de imagen trabaja con técnica digital pura, y permite la memorización de cantidades inimaginables de datos. Además, el ordenador puede gobernar esta placa sin grandes problemas. Para tener una idea de la capacidad de memoria:
En una placa de imagen se podrían memorizar unos 200 microfilms.
Muchas empresas trabajan en el desarrollo de estos aparatos.

Impresora y Plotter:

La norma MSX prevé un interfase paralelo Centronics para la conexión de impresoras y plotters. Este interfase ya es hace tiempo un tipo de conexión normalizada para impresoras y plotters. De este modo existe una amplia oferta de estos aparatos.
El interfase Centronics tiene 36 polos, de los cuales la mayoría no están ocupados. Los datos hacia la impresora o plotter se transmiten a través de los Pins 2 hasta 9 (8 Bit), y a la vez existen algunas conexiones para el mando. El resto de los Pins no está conectado, o está conectado a tierra.

1.4 ESPECIALISTAS ENTRE LOS ORDENADORES MSX

La norma MSX ha determinado un concepto básico que han de cumplir las empresas que quieran contarse entre los proveedores de ordenadores MSX. Lo que no está determinado es la especialización en los diferentes sectores específicos. Así es posible que algunos ordenadores que cumplan esta norma, posean algunas cualidades no especificadas en ella.

Por ejemplo, la empresa Yamaha ofrece un ordenador MSX, que es "musical" a gran escala. Este ordenador se ofrece en el marco de un sistema de música CX-5. El Set básico está compuesto por un ordenador MSX, un módulo FM SFG-01, el miniteclado YK-10 con 44 teclas o un teclado normal con 49 teclas. Con este juego se dispone de un sintetizador Preset con acompañamiento automático, secuenciador y división manual. Es decisiva la oferta de Software, presentada en forma de módulo de ampliación ROM y que se puede introducir en el Slot de expansión. Esta boca se encuentra en la parte inferior del ordenador y no es compatible con el MSX. Para quien quiera ocuparse un poco de la música, se le puede sugerir este ordenador. Posee cualidades de las cuales no disponen ni siquiera sintetizadores de tamaño considerable.

También hay que mencionar el ordenador MSX de la empresa Sanyo. Para este ordenador se ha previsto un lápiz óptico que, junto con el Software adecuado, ofrece unas posibilidades asombrosas. Se pueden visualizar imagenes de vídeo, de televisores y cámaras de vídeo sobre el monitor y manipularlas con el lápiz óptico. Con un poco de fantasía y destreza se pueden conseguir efectos fantásticos.

También el Hit Bit de Sony es un especialista. Ofrece un programa fijo de direcciones con bloc de anotaciones y calendario de fechas, el cual puede ser manejado fácilmente a través de un menú.
Como Vd. ha visto, existen algunos desarrollos interesantes en el sector de ordenadores MSX.

CAPITULO II : GRAFICA CON EL TMS9918A

2.1 INTRODUCCION

Antes de ocuparnos de las cualidades específicas de cada ordenador MSX en el campo de la gráfica, vamos a ver cómo se elaboran las gráficas por un ordenador.

Como Vd. sabe, la imagen de un monitor, o de un televisor, está compuesta por miles de puntos independientes. El ordenador tiene que memorizar cada uno de estos puntos. La información sobre un punto a memorizar es:

- estado del punto, o sea, posicionado o no posicionado

- color del punto

Como la imagen se modifica constantemente, estas informaciones han de memorizarse en el RAM (memoria modificable de escritura/lectura) de su ordenador.

¿Cómo se memorizan estas informaciones necesarias?

Para determinar si un punto está posicionado o no, sólo hace falta un Bit. El Bit es la unidad de información más pequeña. Electrónicamente corresponde a los estados CONECTADO o DESCONECTADO. En el ordenador, estos dos estados se representan por medio de las dos cifras del sistema binario o dual: 0 y 1. Para cada punto de la imagen se memoriza un 1, si éste está posicionado, o un 0, si no lo está.

Por ejemplo, si una imagen de televisor de 640*400 puntos es memorizada, serían necesarios 640*400 = 256000 puntos.

Además hay que memorizar la información del color.

La necesidad de espacio para el color depende directamente de la cantidad de los colores a disponer. A cada color disponible se le asigna un código determinado. Este número se memoriza, junto con la información sobre si está posicionado o no, para cada punto. Como la necesidad de espacio está relacionada con el tamaño del código de color, y con ello con la cantidad de los diferentes colores, la memorización del color necesita más espacio que la información sobre el estado del punto.
Para que el RAM no se ocupe preferentemente con códigos de color, son aconsejables limitaciones. Para ello hay dos posibilidades:

1. Si se trata de una gran cantidad de puntos, la cantidad de colores se reduce radicalmente. A veces disponemos en el modo de mayor resolución sólo de dos colores: un color determinado para el punto y un color para el fondo. Para diseños con muchos detalles, como por ejemplo esquemas de conexión, es necesaria esta representación de alta resolución. Si se necesitan más colores, se procede de la siguiente forma:

2. Si se han de representar muchos colores, hay que realizarlo, si se dispone de la misma capacidad de memoria, a coste de la resolución. Esto significa, que en este tipo de resolución se resumen una cantidad de los puntos más pequeños (Pixel) para formar un punto mayor. Este ''punto grande'' sólo puede ser o no posicionado con todos los Pixels que lo forman. La capacidad así ahorrada, se puede utilizar para la memorización del color de ese punto grande.

También se pueden aplicar combinaciones de las dos posibilidades citadas.

Los principios expuestos sobre la memorización de las gráficas se utilizan tanto en la memorización de texto como en la representación de gráficas de alta resolución. En este capítulo vamos a tratar los métodos utilizados por ordenadores MSX para cada modo de imagen.

Veamos ahora otras cualidades gráficas especiales de los ordenadores MSX. Como ya hemos mencionado en el primer capítulo, la elaboración de la imagen se produce por un Chip especial: el VDP (Video Display Prozessor). El VDP determinado por la Norma MSX es el procesador de vídeo TMS9918A, producido por la empresa Texas Instruments. Este procesador sobresale por su cualidades excepcionales:

- Resolución de 256 * 192 puntos sobre un TV normal

- 15 diferentes colores y transparente

- Representación de hasta 32 Sprites

- Simulación 3D por medio de superposición de 35 niveles

- Direccionado previsto por la RAM de vídeo autónoma

- 4 modos diferentes de imagen, que son:

- Modo de texto: 40 caracteres de anchura, 256 caracteres diferentes

- Gráfica 1: 32 caracteres de anchura, Sprites y colores

- Gráfica 2: gráfica de alta resolución con 256 * 192 puntos

- Modo Multicolor: 64 * 48 puntos con 16 colores

Los 4 modos serán tratados ampliamente. Además se ha dedicado un párrafo propio a los Sprites. En esta ocasión vamos a exponer brevemente el quinto punto:

Cada ordenador MSX posee, además del RAM convencional que se utiliza para la memorización de programas, etc., una RAM de vídeo, abreviado VRAM. Esta VRAM sirve únicamente al VDP, para memorizar los datos necesarios para la elaboración de la imagen. La ventaja de este método es, por una parte, que no se utiliza para gráfica ningún Byte de la "vallosa" memoria RAM y, por otra parte, que sólo para la memorización de gráficas se dispone de 16 Kilobyte (= 16 * 1024 Byte =

16* 1024 * 8 Bits). Además, la administración interna de la VRAM se realiza independientemente por el VDP, por lo cual el procesador tiene más tiempo para otras cosas.
Ya que el acceso (leer o escribir) a las dos RAMs se ejecuta de diferente manera, existen comandos de BASIC especiales para la VRAM:

>VPOKE< : escribir valores en la VRAM y
>VPEEK< : leer valores de la VRAM

Con estos dos comandos es posible una manipulación directa de la VRAM, y de esta forma también de la imagen en la pantalla actual.

¡Introduzca Vd!:

```
SCREEN 0: FOR I=0 TO 255:VPOKE I,I: NEXT
```

Antes de tratar detalladamente los diferentes modos gráficos, vamos a ver, a raíz de algunos ejemplos, las cualidades de los ordenadores MSX en el campo de la gráfica:

Primer programa gráfico

```
10 SCREEN 2
20 FOR I=0 TO 255 STEP 2: REM variar paso
30 'LINE (0,96)-(I,96+95*SIN(I/33)):REM variar 33
31 'LINE (0,96)-(I,96+95*SIN(I/33)),I MOD 16: REM variar 16
32 'LINE (255-I,96)-(I,96+95*SIN(I/33):REM variar 33; añadir
   I MOD 15
33 'LINE (255-I,96-95*COS(I/33))-(I,96+95*SIN(I/33):REM modi
   ficar 33 con COS o SIN
34 'LINE (255-I,96+95*SIN(I/17))-(I,96+95*SIN(I/34)): REM mo
   dificar 34 o 17 con COS o SIN
35 'LINE (255-I,191)-(I,96+95*SIN(I/33)):REM modificar 33
36 'LINE (255,96-95*SIN(I/33))-(I,96+95*SIN(I/33)): REM modi
   ficar 33
37 'LINE (255,96+95*SIN(I/33))-(I,96+95*SIN(I/33)),I MOD 16:
   REM modificar 33
```

```
38 'LINE (128-127*SIN(I/33),96)-(I,96+95*SIN(I/33))
39 'LINE (128-127*SIN(I/33),96-95*SIN(I/33))-(I,96+95*SIN(I/
   33))
40 NEXT
50 WAIT &HAA,&H40: REM esperar a CAP desconectado
```

Este programa contiene diferentes variaciones de dibujos. Todas las líneas están protegidas por él en forma de líneas REM. Si quiere comprobar alguna de estas posibilidades, sólo tiene que borrar el REM en esta línea. La tecla CAP pulsada visualiza la imagen en la pantalla hasta que, pulsando nuevamente la misma tecla, vuelva al modo normal.

Una especialidad del Basic MSX son los subcomandos o comandos macro que se aplican para gráfica y sonido, y que pueden ser representados como Strings:

```
10 SCREEN 2
20 S=2
30 A$ = "c=c;s=s;nr10u10nr10e5r10nd10g5d10e5u10"
40 FOR S=1 TO 39 STEP 2
50 C=SMOD13+3
60 PSET (100-S*1.4,120+S*1,4)70 DRAW A$
80 NEXT
90 WAIT &HAA;&H40
```

2.2 EL MODO DE TEXTO

El modo de texto permite, como máximo, la representación de 40 caracteres por línea con 24 líneas por pantalla. Se puede disponer de 256 diferentes caracteres, entre otros ''letras no inglesas''. Así se pueden visualizar, sin modificar el juego de caracteres, las letras alemanas sobre la pantalla. En el modo de texto sólo se pueden elegir dos colores: el color de fondo, que se utiliza al mismo tiempo para los márgenes laterales, y el color de la escritura.

¿Cómo se memoriza internamente el texto de pantalla?

Al tratarse del modo de texto, se memoriza independientemente cada uno de los 256 caracteres posibles. Un carácter de este tipo está compuesto por 8*8 puntos, aunque en el modo de texto sólo se indican 6 puntos en sentido horizontal.

Veamos un ejemplo:

```
1 ***
2*   *
3*   *
4*   *
5* * *
6*  *
7 ** *
8
 12345678
```

Partiendo de la representación en matrices, para un punto posicionado se utiliza un 1, y para un punto no posicionado un 0. La serie así obtenida se interpreta como número binario (ver capítulo 5).

```
&B01110000
&B10001000
&B10001000
&B10001000
&B10101000
&B10010000
&B01101000
&B00000000
```

De esta forma obtenemos 8 Bytes. Un Byte es un número binario de 8 Bits. Su valor está comprendido entre 0 y 255. Así se puede definir exactamente cada carácter. Cada uno de los 256 caracteres posibles se representa de este modo por los 8 Bytes. Para memorizar el juego de caracteres completo, son necesarios 8*256=2024 bytes = 2K (kilobyte).

Este sector de memoria, que en el sistema MSX se encuentra en la RAM de vídeo, se denomina generador de caracteres.

El siguiente programa lee el juego de caracteres del Basic MSX:

```
10 SCREEN 0
20 ST=BASE(2)
30 FOR I=S TO ST+256*8-1
40 PRINT RIGHT$ ("0000000"+BIN$ (VPEEK(I)),8)
50 NEXT
```

La línea 40 es el corazón del programa:

>VPEEK(I)< lee un Byte de la VRAM. Con >BIN$< se tranforma el Byte leído en número binario a, y con >RIGH$< se completan los 0 que puedan faltar al principio, de forma que la longitud total de String sean 8 caracteres. >BASE(2)< en la línea 20 suministra además la dirección inicial del juego de caracteres.

Al transcurrrir el programa, verá Vd. pasar por la pantalla el juego de caracteres total en formato grande. En ello, el 1 representa un punto posicionado, y el 0 ningún punto.

Como Vd. ve, los diferentes caracteres no están separados entre sí, o sea, después del último Byte de un carácter sigue el primero del siguiente carácter. Para diferenciar los caracteres entre sí, cada uno recibe un número, el así denominado código de carácter. El código de carácter indica de qué carácter se trata en el generador de caracteres.

Los Bytes 0 hasta 7 corresponden al carácter 0, los Bytes 8 hasta 15 al carácter 2, los Bytes 16 hasta 23 al carácter 3, etc.

Esto significa:

El carácter con el código N, está memorizado desde el Byte (N*8) hasta el Byte (N*8+7) del generador de caracteres.

Para la numeración existe el bastante divulgado código ASCII (American Standard Code for Information Interchange).

Este código determina el significado de los caracteres de códigos 0 hasta 127. Los caracteres con los códigos 128 hasta 255 están ocupados en el MSX con caracteres gráficos propios y letras extranjeras. Pero comprobemos lo dicho. El siguiente programa edita, después de introducir el código, el carácter correspondiente.

```
 5 CLS
10 SCREEN 1
20 ST=BASE(7)
30 LOCATE 3,3:INPUT" Codigo de caracter: ";ZC
40 LOCATE 0,6
50 FOR I=ST+ZXC*8 TO ST+ZC*8+7
60 PRINT RIGHT$ ("00000000"+BIN$(VPEEK(I),8)
70 NEXT
80 LOCATE 0,16:PRINT CHR$(ZC)
90 GOTO 30
```

Como Vd. ve, el carácter leído directamente con >VPEEK<, es idéntico al originado con >CHR$(ZC)<. >PRINT CHR$< edita el carácter correspondiente al código indicado. Lo contrario al comando >CHR$< es la función >ASC<. >ASC(A$)< edita el código del carácter de A$.

De esta forma queda clara la construcción del generador de caracteres. De la misma manera que es posible leer el contenido del generador de caracteres con la función >VPEEK<, es posible escribir en él con el comando >VPOKE<. Esto significa, que el juego de caracteres puede ser modificado según deseos y necesidades propios.

Modificación del juego de caracteres

El carácter de más fácil representación es el espacio libre. En el espacio libre no hay ningún punto posicionado en la matriz de 8*8, por lo cual es representado por 8*0 Bytes. Vamos a modificar ahora este carácter:
El código ASCII del espacio libre (ó Space) es 32, tal como se obtiene con >PRINT ASC(" ")<. Ahora vamos a posicionar el punto de la esquina superior izquierda de este espacio libre. La fila superior es representada por el primer de los 8 Bytes, ó sea:

```
SCREEN 0
VPOKE BASE(2)+32*8, &B10000000
```

En lugar de &B10000000 podemos escribir el valor decimal 128, que es calculado con >VAL("&B10000000"). Inmediatamente obtenemos un espacio libre modificado, en el cual el punto superior izquierdo está posicionado. Esta modificación se aplica a todos los espacios libres existentes en la pantalla. El estado original se puede reponer con:

```
VPOKE BASE(2)+32*8,0
```

Programa para la composición de juegos de caracteres propios

```
10 REM generador de caracteres
20 SCREEN 0:COLOR 15,4,4:DEFINT A-Z
30 BA=BASE(PEEK(&HFCAF)*5+2):REM direccion basica
40 XA=5:YA=7:REM posicion de la letraOk
50 EI=0: REM determinar aparato de introduccion
60 ON STRIG GOSUB 470: REM definir salto de Interrupt
70 STRIG(EI) ON: REM conectar Interrupt
80 AU$=" ":EI$=CHR$(1)+CHR$(64+10): REM Desconect$, conect$
90 ON KEY GOSUB 700,800:REM definir saltos de Interrupt
100 KEY 1,"next":KEY (1) ON
110 KEY 2,"Fin ":KEY (2) ON:REM programar teclas de funcion;
 permitir Interrupt
120 REM
130 REM emitir caracteres
140 REM
200 CLS:LOCATE 4,4:PRINT"caracter:";
210 A$=INKEY$:IF A$="" THEN 210: REM esperar introduccion
220 PRINT A$:REM emitir letras
230 BZ=BA+ASC(A$)*8:REM direccion basica de caracter
240 FOR I=BZ TO BZ+7
250 BY=VPEEK(I)
260 P=128
270 LOCATE XA,YA+I-BZ
280 FOR J=0 TO 7
290 IF (BY AND P)=0 THEN C$=AU$ ELSE C$=EI$
300 PRINT C$;:REM editar matriz de caracter
310 P=P/2
320 NEXT
330 NEXT
340 X=0:Y=0
350 LOCATE XA+X,YA+Y,1
360 R=STICK(EI)
```

```
370 IF R=0 THEN 360: REM control del cursor mediante joystic
k/teclas de cursor
380 IF R=8 OR R=1 OR R=2 THEN Y=Y-1
390 IF R>3 AND R<7 THEN Y=Y+1
400 IF R>1 AND R<5 THEN X=X+1
410 IF R>5 AND R<=8 THEN X=X-1
420 IF X>7 THEN X=X-1:BEEP
430 IF X<0 THEN X=X+1:BEEP
440 IF Y>7 THEN Y=Y-1:BEEP
450 IF Y<0 THEN Y=Y+1:BEEP
460 GOTO 350
470 REM STRIG rutina Interrupt
480 BY=VPEEK(BZ+Y) XOR2^(7-X)
490 IF (BY AND 2^(7-X))=0 THEN PRINT AU$ ELSE PRINT EI$:REM
modificar matriz
500 LOCATE XA+X,YA+Y
510 VPOKE BZ+Y,BY
520 RETURN
700 REM Key 1 rutina Interrupt
710 KEY(1) ON: LOCATE,,0:GOTO 200
800 REM Key 2 rutina Interrupt
810 LOCATE,,0:REM desconectar cursor
820 DEFUSR1=&H139D:REM direccion de salto para ocupacion ori
ginal de Key
830 X=USR1(1)
840 CLS
850 END
```

Descripción del programa:

Este programa es interesante porque funciona tanto en el modo >SCREEN 0< como en el modo >SCREEN 1<. Además cabe destacar que es gobernado por interrupt. Como Vd. sabe, en los ordenadores MSX se ha previsto gobierno por interrupt.

Línea 30:

Aquí se calcula la dirección básica de cada carácter de la tabla respecto al modo correspondiente.

Línea 40:

Se determina la posición x ó y del carácter.

Línea 50:

En EI se determinan los aparatos de introducción.
EI=0 : teclado, o sea, teclas de cursor + tecla espaciadora (fuego)
EI=1 : joystick 1
EI=2 : joystick 2

Línea 60:

Esta línea define la dirección de salto para un salto de interrupt, iniciado desde el disparador. Por lo tanto, si se pulsa el disparador, el programa sigue en la línea 470.

Línea 70:

Desconecta el interrupt del disparador.

Línea 80:

En AU$ (Desconecta$) se memoriza un espacio libre.

En EI$ (Conecta$) se memoriza un carácter especial en forma de un círculo invertido. AU$ se utiliza en lugar de un punto no posicionado, y EI$ en lugar de uno posicionado.

Línea 90:

Define dos saltos hacia Interrupt por medio de teclas de función Key 1 y Key 2.

Línea 100,110:

La tecla de función 1 es ocupada con "next" y el Interrupt es desconectado.
La tecla de función 2 se ocupa con "final". También aquí se desconecta el Interrupt.

Línea 200:

Aquí comienza la parte del programa para la introducción de caracteres. Por lo que resta, esta línea es de fácil comprensión.

Línea 210:

En esta línea se memoriza en A$ el carácter que fue pulsado en el teclado.

Línea 220:

Visualiza sobre la pantalla el carácter pulsado.

Línea 230:

En esta línea se calcula la dirección básica del carácter (BZ). El carácter de la dirección básica se calcula de la dirección básica (BA) y el código ASCII de las letras contenidas en A$ multiplicado por ocho. El ocho posiciona el contador al principio del carácter correspondiente del generador de caracteres.

Línea 240:

Este bucle permite la lectura de Byte en Byte de un carácter de la tabla de caracteres.

Línea 250:

Lee una fila de 8 puntos de este carácter en BY. (Bytematriz).

Línea 260:

Aquí comienza la rutina, que edita los caracteres ampliados sobre la pantalla.
En P (Potencia) se memoriza el valor 128 para posteriores aplicaciones.

Línea 270:

Aquí se calcula la posición actual para la edición del carácter sobre la pantalla.

Línea 280:

Bucle para la lectura Bit a Bit de un Byte.

Línea 290:

Si el Bit está posicionado, el carácter especial es editado en EI$, si no un espacio libre (AU$).

Línea 300:

Edita el carácter correspondiente.

Línea 310:

P, cuyo valor siempre corresponde al Bit actual, es dividido por dos. De esta forma surge una fila de números que es la siguiente:
128,64,32,16,8,4,2,1
Esta fila corresponde al valor de los Bits de izquierda a derecha.

Línea 350:

Esta línea posiciona el cursor en la posición actual sobre el carácter grande visualizado en la pantalla y lo conecta.

Línea 360 hasta 460:

Aquí se realiza el movimiento del lápiz de juego y consulta. Con esta rutina es posible moverse con el joystick libremente a través del ''carácter grande''. Aquí comienza el programa principal.

Línea 470,520:

Hacia aquí se salta después de originar un interrupt con el disparador o la tecla espaciadora. En BY se posiciona o se borra un carácter, según el estado anterior, en la "matriz grande" pulsando el disparador.

Línea 500:

Coloca el cursor nuevamente en la posición arriba modificada dentro del ''carácter grande''.

Línea 510:

Escribe el Bit modificado (=1 punto) en la tabla

Línea 700,710:

Entrada para los Interrupts originados por KEY 1. Origina la elección y la impresión de pantalla del siguiente carácter.

Línea 800 hasta 850:

Rutina, hacia la cual se salta después de pulsar Key 2 a través de un Interrupt. Desconecta el cursor y restablece la ocupación original de las Keys En las líneas siguientes se finaliza el programa.

Línea 830:

Activa la rutina para la ocupación estándar de las Keys.

Después de iniciar el programa, determine el carácter a modificar por medio de la tecla correspondiente. Después se edita el carácter en tamaño grande y el cursor surge sobre la pantalla. Ahora se puede mover el mismo por medio de las teclas del cursor o el joystick a través de la matriz 8*8. Pulsando la tecla espaciadora, o el disparador del joystick, se invierte el punto que se encuentre debajo del cursor en ese momento, o sea, se activa si no lo estaba, o viceversa. La modificación efectuada es aplicada inmediatamente, tanto en la representación ampliada como en el carácter original. Con la tecla de función "F1" puede Vd. modificar otros caracteres, con F2 puede finalizar el programa. Tenga en cuenta al modificar letras, que el modo de texto los caracteres tiene una anchura de 6 puntos, en diferencia al modo gráfico 1.
Con esto nos hemos extendido suficientemente sobre el juego de caracteres.

Caracteres de mando

Merecen mención las siguientes peculiaridades. Los caracteres con los códigos de 0 hasta 31 no se pueden producir directamente con un solo comando >CHR$<. Esto es necesario, ya que los códigos, que son menores que 32, son interpretados normalmente como caracteres de mando. También éstos están normalizados por el código ASCII. Todos los códigos de mando puede ser producidos con la tecla CTRL en combinación con otras determinadas teclas. Además se dispone de algunas teclas para caracteres de mando importantes.
Con el siguiente programa Vd. puede encontrar los códigos de las combinaciones CTRL + tecla:

```
10 A$=INKEY$ : IF A$="" THEN 10
20 PRINT ASC(A$) : GOTO 10
```

Vale la siguiente asignación:

Código	CTRL+ tecla	Función
0	arroba	-
1	A	Edición de caracteres gráficos
2	B	Salto de palabra hacia atrás (BACK)
3	C	Interrupción de la entrada (desconectar AUTO)
4	D	-
5	E	Borra todo hasta el final de la línea
6	F	Salto de palabra hacia delante (FORWARD)
7	G	Pitido
8	H	Backspace (Tecla BS)
9	I	Salto del tabulador (tecla TAB)
10	J	Avance de línea (Line Feed)
11	K	Cursor Home (Tecla Home)
12	L	Borrar pantalla (tecla shift-HOME)
13	M	Retorno del carro "CR" (tecla RETURN)
14	N	Salto hacia el final de la línea
15	O	-
16	P	-
17	Q	-
18	R	Modo Inest conectado /descon. (tecla Inest)
19	S	-
20	T	-
21	U	Borrar línea completamente
22	V	-
23	W	-
24	X	Tecla Select
25	Y	-
26	Z	-
27	paréntesis	Escape (tecla ESC)(paréntesis abierto)
28	U:(shift)	cursor izquierda (tecla cursor)
29	corchete	cursor derecha (tecla cursor)
30	^ onda	cursor arriba (tecla cursor)
31	subrayar	cursor abajo (tecla cursor)
127		tecla Delete

Por lo tanto, los caracteres de mando se pueden producir en parte con las teclas especiales, pero también con >PRINT CHR$(...)<, por ejemplo con >PRINT CHR$(7)<. De esta forma, con >PRINT CHR$(7)< no se editan caracteres sobre la pantalla para códigos menores que 32. Pero como estos caracteres existen, a pesar de todo, hay otra posibilidad de visualizarlos sobre la pantalla.
Aquí un ejemplo:

El carácter con el código N (N es menor que 32) se edita a través de

PRINT CHR$(1);CHR$(64+N).

Por lo tanto, >CHR$(1) es aquí un carácter de mando, el cual comunica al ordenador que el código siguiente es uno de los 32 caracteres inferiores. Es interesante que el carácter con el código 0 no está ocupado en el juego de caracteres, por lo tanto es un espacio libre.

Además del código 0 existe otro de especial importancia:

El código 235

```
10 SCREEN 0 : WIDTH 36
20 LOCATE 0,5
30 FOR I=5 TO 20
40 PRINT STRING$ (36,255)
50 NEXT
60 LOCATE 0,0 : PRINT "el cursor tiene el codigo 255"
70 FOR I=0 TO 34 : LOCATE I,0,1
80 FOR W=0 TO 150 : NEXT W
90 NEXT
```

La explicación de lo expuesto es la siguiente:
El cursor es representado por el carácter con el código 255. El sistema operativo modifica constantemente la definición del carácter 255, según el carácter que se encontraba inicialmente en la posición del cursor. Corresponde a la representación inversa del carácter determinado. Con la inversión, los puntos posicionados se borran y viceversa. La representación de los caracteres invertidos será tratada en el capítulo sobre el modo gráfico 1.

¿Qué es una tabla de nombres?

Para originar la imagen completa, no sólo basta con la memorización del juego de caracteres. Además, hay que determinar qué carácter ha de surgir y en qué posición de la pantalla.

Este problema lo resuelve la así denominada tabla de nombres o de patrones. Todas las posiciones posibles sobre la pantalla son numeradas. Se comienza en la esquina izquierda superior y se numera, carácter por carácter, hasta llegar a la esquina derecha inferior. Con este método queda determinada cualquier posición de la pantalla.
Como la imagen en el modo 1 está compuesta por 40 columnas y 24 líneas, la tabla de nombres ha de tener una extensión de 40*24=960 Bytes. El Byte 0 de la tabla de nombres contiene el código del carácter que se encuentra en la esquina superior izquierda de la pantalla.

Byte 1 contiene el código del siguiente carácter, etc. Según esto, el código del primer carácter se encuentra en la segunda fila en el Byte 40 de la tabla. Con >SCREEN 0< obtenemos la dirección de inicio de la tabla de nombres por medio de >BASE(0)<. En el caso normal se obtiene la dirección 0.
De esta forma la serie de comandos:

```
LOCATE X,Y: PRINT CHR$(64);
```

puede ser sustituidos por:

```
VPOKE X+40*Y,64
```

Esto sólo vale con >WIDTH 40<.

Con ayuda del comando >VPOKE<, con frecuencia se pueden realizar ediciones sobre la pantalla con mayor rapidez. Internamente todos los comandos >PRINT< son ejecutados como comandos >VPOKE<, los cuales por su parte están formados por comandos I/O (ver capítulo 3).

Resumamos:

SCREEN 0	
Dirección inicial de la tabla de	BASE(0)=0
Longitud de la tabla	40*24=960
Dirección inicial del generador de caracteres	BASE(2)=&H800
Longitud de la tabla	256*8=2048Bytes

DIVISION DE LA MEMORIA

Tabla de nombre	&H0000 - &H03BF	(casi 1 K)
libre	&H03C0 - &H07FF	(más de 1 K)
Generador de caracteres	&H0800 - &H3FFF	(2K)
libre	&H1000 - &H3FFF	(12 K !!)

Como Vd. ve, menos de la cuarta parte de los 16 K VRAM están ocupados. Para aprovechar esta capacidad, trataremos sobre los registros del VDP.

Registros del VDP

El TMS 9918A posee 8 registros, 7 registros de escritura (¡no legibles!) y un registro de lectura (¡no transmisible!).
Nos podemos imaginar un registro, como una memoria para uno o varios Bytes, igual que una sola posición de memoria. Tal posición de memoria, que está asignada fijamente a un módulo, se denomina frecuentemente registro. Un registro "almacena" información importante para su módulo correspondiente.

El VDP tiene el deber de producir la imagen sobre la pantalla. Para ello son necesarias informaciones sobre la dirección inicial del generador de caracteres y la dirección inicial de la tabla de nombres. Si nos encontramos en el modo de texto, entonces estas informaciones y las referentes al color, son suficientes para la generación de la imagen.
Para que la imagen sea producida correctamente por el VDP, hay que transmitir estas informaciones. Todos los registros del VDP son registros de 8 Bit, de los cuales no todos los 8 Bits se utilizan.

La tabla de nombres se memoriza en el registro 2. Para ello es codificada previamente. La dirección real de memoria de la tabla de nombres en el modo de texto, la obtenemos con >BASE(0)<. En el caso estándar es 0.

(En el caso de que Vd. no conozca los sistemas hexadecimales y binarios, es conveniente que lea primero el capítulo 5 sobre sistemas de números).

Básicamente, las direcciones de la RAM de vídeo están memorizadas en el sector 0 hasta &H3FFF. Para memorizar &H3FFF, la mayor dirección VRAM, son necesarios 14 Bits (=2+3*4, ¿por qué?). De estos 14 Bits sólo los superiores, los de mayor valor, son memorizados en los registros.

En el caso de la tabla de nombres, sólo se memorizan los 4 Bits superiores, o sea, los Bits 10 hasta 13. El Bit 10 tiene el valor 2^10 = 1024. Esto significa, que la dirección de la tabla de nombres puede ser desplazada a pasos de 1 KByte (1 k =1024 Byte).

A continuación vamos a conocer la posibilidad de aprovechar, por medio de una modificación del VDP, la capacidad que queda libre en el VRAM. Por ello es posible memorizar simultáneamente 14 pantallas, e interconectar entre una y otra.

Como la dirección original de la tabla de nombres es 0, >BASE(0)< y también >VDP(2)< suministran el valor 0. El generador de caracteres ocupa las direcciones hasta &0FFF. El sector situado por encima está libre. Ahora vamos a desplazar la tabla de nombres hacia la dirección &H200.
Para ello puede introducir simplemente:

```
BASE(2) = &H2000
```

o, si quiere modificar directamente el registro 2 del VDP:

```
VDP(2) = &H2000/2^9
```

Por medio de la división por 2^9 sólo se tienen en cuenta los 4 Bits superiores (10 hasta 13).
Lamentablemente, con estos comandos sólo conseguimos un efecto muy interesante, pero no el deseado.

En la imagen ahora visualizada, que puede tener un aspecto muy desastroso, no podemos ni mover el cursor, ni hacer cualquier otra operación. Por lo tanto, vuelva al punto de partida:

```
VDP(2) = 0 ó BASE(0) = 0
```

No se deje perturbar por el hecho de que no puede reconocer lo que está introduciendo. Todas estas introducciones han ido a parar a la pantalla original, y no a la que se está visualizando en este momento.

Aquí se ve claramente que el VDP opera independientemente del BASIC. Cuando se modifican sus registros, entonces origina la imagen correspondiente. En nuestro caso, el sistema operativo no ''sabía'' que la tabla de nombres se había desplazado. Por lo tanto, todas la indicaiones se realizaban hacia la tabla de nombres antigua, aun cuando ésta no era utilizada por el VDP. Es usual que el comando >SCREEN< comunique al sistema operativo las informaciones necesarias. En ello se utiliza la dirección introducida con el comando >BASE<.

Por lo tanto, introduzca:

```
BASE(0) = &H2000
SCREEN 0
```

Ahora, la tabla de nombres se encuentra en la dirección indicada, como puede comprobar con el comando >PRINT HEX&(BASE(0))<. Este método tiene la desventaja, de que el comando de pantalla correspondiente es borrado por el comando >SCREEN<. Si se han de utilizar las diferentes pantallas simultáneamente, ha de ser posible cambiar sin perder el contenido de las mismas.

La dirección, en la cual el sistema operativo memoriza la tabla de nombres, es &HF922.
Si adicionalmente colocamos esta dirección en el valor nuevo, obtendremos nuevamente el cursor en la imagen nueva.

El siguiente pequeño programa va a mostrar la posibilidad de utilizar varias pantallas en el modo directo, o sea, al programar. Con él, Vd. puede, entre otras cosas, depositar diferentes partes de un listado en diferentes pantallas, e interconectar entre unas y otras, según necesidad.

```
9999 REM antes KEY1, introducir "RUB 10000"+CHR$(13)
10000 REM conmutador de página
10010 A$=INKEY$:IF A$="" THEN 10010
10020 A=VAL(A$)
10030 IF A=0 and A$<>"0" THEN A=(ASC(A$) AND &B11011111)
      -55
10040 IF A>13 OR A<0 THEN 10010
10050 POKE &HF923, (A+2+2*(A<2))*4
10060 VDP(2)=A+2+2*(A<2)
10070 LOCATE 0,20
```

Después de introducir el programa y:

```
KEY 1, "RUN 10000" + CHR$(13)
```

puede activar la pantalla deseada pulsando F1 y una de las teclas 0 hasta 9 y A, B, C ó D, de las cuales 0 es la pantalla estándar. En la primera utilización de una pantalla, hay que borrarla casi siempre con SHIFT+HOME.

Explicación del programa:

Línea 10010: Trae tecla pulsada
Línea 10020: Averigua el valor si es número
Línea 10030: Averigua el valor si es letra
Línea 10040: Comprueba la fiabilidad
Línea 10050: Escribe la dirección de la tabla de nombres
para el sistema operativo
&HF922 contiene el Low Byte (=0),y
&HF923 contiene el High Byte de la dirección.

Asignación de A a la dirección:

A	Dirección de la tabla de nombres	A	Dirección de la tabla de nombres
1	&H0400	7	&H2400
2	&H1000	8	&H2800
3	&H1400	9	&H2000
4	&H1800	10(A)	&H3000
5	&H1000	11(B)	&H3400
6	&H2000	12(C)	&H3800
		13(D)	&H3000

Línea 10060 : Cargar el registro 2 de vídeo con el valor correspondiente. En ello +2*(A<2) origina un -2 si A<2, con lo cual se consigue la asignación.

Línea 10070: Posiciona el cursor al final de la pantalla.

Naturalmente puede utilizar estas rutinas también en sus programas. Si casualmente hubiera en memoria otros programas en BASIC, las líneas se pueden cambiar de numeración a partir de 10000.

Al modificar las direcciones básicas tenga en cuenta la diferencia de >VDP< al comando >BASE<.
El comando >VDP< escribe únicamente el valor indicado en el registro correspondiente. La modificación es anulada con el comando >SCREEN< y se vuelve al punto de partida. Por lo contrario, con el comando >BASE< no sólo se escriben las modificaciones en los registros del Vídeo Chip, sino que también son memorizadas en la RAM, y así se toman como punto de partida en el siguiente comando >SCREEN<. Una modificación a través del comando >BASE< es válida hasta que es modificada por un nuevo comando >BASE<. Las modificaciones con el >VDP< son sobreescritas con >SCREEN< y también con >BASE<.

Como Vd. sabe, por medio de >VDP< no sólo se puede escribir en un registro, sino también leer su contenido. Pero como esto no es posible con el TSM9918A, excepto con el registro 9, se memorizan en el RAM los valores de registro actuales. A partir de la dirección &HF3DF están los valores de registro del Vídeo Chip comenzando con 0, o sea, en lugar de:

```
PRINT VDP(N)
```

se puede escribir:

```
PRINT PEEK (&HF3DF + N)
```

En el modo de texto existen otros registros VDP de importancia:
En el registro 4 se memorizan los 3 Bits MSB (Most Significant Bits = Bits de mayor valor) de la dirección inicial del generador de caracteres. También existe la posibilidad de interconectar entre diferentes juegos de caracteres, o incluso entre los juegos de caracteres y la pantalla. Esto lo describiremos más exactamente en el capítulo sobre el modo gráfico I.

Colores en el modo de texto

Ahora vamos a profundizar en el tema de los colores:
Los colores se memorizan en el registro 7. Como existen 16 diferentes colores, sólo se pueden memorizar 2 colores en un Byte. En ello, los 4 MSBs (Bit 4 y 7) del registro 7 determinan el color de fondo, y los 4 LSBs (Least Significant Bits = Bits de menor valor, Bits 0 hasta 3) el color de la escritura.

En el modo de texto, el color del marco es igual al color de fondo. En el modo de texto es imposible conseguir colores diferentes sobre la pantalla. Tampoco es posible la representación inversa de números o letras. Pero como esto es muy útil para una representación clara, por ejemplo en menús, presentaremos un programa con el cual es posible esta representación inversa de caracteres.

Para ello se modifica primeramente el juego de caracteres. Los caracteres con los códigos 0 hasta 127 se conservan. Los códigos de 128 hasta 255 corresponden a los 128 caracteres inferiores, sólo que están invertidos. Si se ha de representar un carácter invertido, entonces sólo hay que sumar 128 a su código el indicar el código resultante. Pero ahora veamos el programa que manipula el juego de caracteres de dicha manera:

```
10 BA = BASE(2) + 128*8
20 FOR I=BA TO BA +128*8-1
30 VPOKE I,VPEEK (I-128*8) XOR 255
40 NEXT
```

Ahora se pueden editar estos caracteres inversos por medio de >VPOKE<. La edición normal con

```
LOCATE X,Y:PRINT CHR$(Z);
```

ha de sustituirse por:

```
VPOKE BASE(0) + Y+X*40,Z+128
```

para obtener el carácter inverso. Como este método sería muy complejo para escribir palabras o frases enteras, hemos escrito la siguiente rutina máquina. Con ayuda de esta rutina máquina, es posible la edición con el comando >PRINT<, sólo que todo se representa inverso.

```
 10 REM BASIC Cargador Caracteres Inversos (Patch Invers)
 20 CLEAR 200, &HF370
 30 FOR I=&HF370 TO &HF37A
 40 READ A$
 50 POKE I, VAL("&H"+A$)
 60 NEXT
 70 DATA E1,F1,FE,20,38,02,F6,80,F5,E5,C9
 80 POKE &HFDA5,&H70
 90 POKE &HFDA6,&HF3
100 REM Activar con POKE &HFDA4,&HC3
110 REM Desconectar con POKE &HFDA4,&HC9
120 REM No utilizar en el modo directo
```

Este programa carga una ampliación del sistema, la cual es escrita en el sector del sistema MSX. Con >POKE &HFEE4,&HE1< se conecta la edición inversa de caracteres, y con >POKE &HFEE4,&HC9< es desconectada. Naturalmente el juego de caracteres ha de ser "tratado" naturalmente antes de la utilización de este programa, o sea, los códigos 128 hasta 255 han de contener las matrices inversas de los caracteres con los códigos 0 hasta 127.

Para "especialistas", aquí el listado máquina del Patch:

```
FDA4 C370F3   JP    &HF370     ;Edición Patch sobre la
                                 pantalla
F370 E1       POP   HL         ;Dirección de salto hacia atrás
F371 F1       POP   AF         ;Código de caracteres
F372 FE20     CP    &H20       ;¿Carácter de mando ?
F374 3802     JR    CN &HF378  ;Sí, entonces no transformar
```

```
F376 F680     OR    &H80       ;Código = Código + 128
F378 F5       PUSH  AF         ;Código de carácter sobre pila
F379 E5       PUSH  HL         ;Dirección de salto hacia atrás
                                sobre pila
F37A C9       RET              ;Seguir en el programa ROM
```

Aquí un ejemplo:

```
10 POKE &HFDA4, &HC3: PRINT "Inverso"
20 PRINT "Sigue inverso"
30 POKE &HFDA4,&HC9:PRINT"Ahora normal"
40 PRINT  "sigue normal, hasta que se ordene POKE &HFEE4,
   &HC3"
```

Ya que de momento estamos tratando un diseño de pantalla de alto nivel, vamos a seguir un poco con el tema.

"Ventanas" con MSX

Aunque el sistema MSX no prevé ventanas, se pueden producir bastante bien. Un Window (en inglés = ventana) es un sector de la pantalla exáctamente definido. Sobre un Window se puede editar independientemente del resto de la pantalla. Puede tomar cualquier forma rectangular.

¿Cómo se puede definir un Window para un ordenador MSX?

Primeramente se podría utilizar el comando >WIDTH<. Este comando determina la anchura del sector de edición. Pero al ejecutar este comando, se ejecuta automáticamente también un CLS (Clear Screen). A causa de esto, los contenidos de otras ventanas se borrarían.
El borrado automático de la pantalla después del comando >WIDTH< se puede evitar con un comando >POKE< determinado. La posición, en la cual está memorizado el ancho de pantalla actual, tiene la dirección &HF3B0. Modificando esta memoria con >POKE< se puede regular la longitud de las líneas de pantalla, sin que se ejecute un CLS>

Con esto se ha realizado el primer paso hacia la técnica Window.

Para seguir limitando el sector de pantalla, vamos a ocuparnos de la visualización de la ocupación de las Keys sobre la pantalla. Esta indicación de las Keys puede ser conectada y desconectada desde el BASIC.
La línea, en la cual se efectua la indicación, está depositada en la memoria &HF3B1. El valor aquí memorizado, corresponde a la última línea utilizada por la rutina de edición por pantalla, cuando KEY está desconectada (off). Si KEY está conectada (on), entonces el valor es el número de la línea que está debajo de la última línea utilizada. Pruebe Vd.:

```
POKE &HF3B1,15
```

Primero no cambia nada. Pero si Vd. mueve el cursor hacia el margen inferior de la pantalla, comprobará que se queda "atascado" en la línea 15 (si KEY off, sino en la 14). De esta forma se protege la parte inferior de la pantalla. También con el Scrolling se mueve sólo la parte superior hacia arriba. De esta forma puede determinar la última línea de un Window.
Pero esto aun no es todo. Existe otra memoria que influye en gran medida en el formato de la pantalla. Esta memoria tiene la dirección &HF3DE. Este Byte se utiliza principalmente como Flag (en inglés = bandera, indicación) para comprobar si la indicación Key está conectada o desconectada.

```
KEY OFF: PRINT PEKK (&HF3DE)
```

resulta 0 como valor y

```
KEY ON: PRINT PEEK (&HF3DE)
```

tiene 255 como resultado.

Si esta memoria se coloca directamente con POKE a 0 o 255, el estado correspondiente surge sólo después de introducir CLS.
Con el comando >POKE< también es posible memorizar otros valores diferentes de 0 y 255. Pruebe Vd.:

```
CLS: POKE &HF3B1,20: KEY ON
y luego
POKE &HF3DE,250
```

Si ahora corre con el cursor hacia abajo, no pasará Vd. de la quinta línea sobre la edición Key. Por lo contrario, si introduce:

```
POKE &HF3DE, 2
```

puede llegar hasta dos líneas por debajo de la edición Key. Por lo tanto, vale:

```
&HF3B1 - Línea de la edición Key
&HF3DE - Key on/ off diferencia Flag en el complemento de
         dos hacia la línea de la edición Key
```

La última línea de pantalla efectiva la obtenemos por:

```
PRINT PEEK(&HF3B1) + PEEK(&HF3DE) + 256*PEEK(&HF3DE)>127)
```

A través de manipulaciones hábiles de estas memorias, es posible una configuración de pantalla con Windows.

2.3 EL MODO "GRAFICA 1"

En este modo se indican, en diferencia al modo de texto, todos los puntos de la matriz 8*8 del generador de caracteres. Con esto se reduce a 32 el número máximo de caracteres por línea. El número de líneas sigue siendo 24. Adicionalmente, este modo permite la utilización simultánea de los 16 colores. También los colores de fondo y del margen lateral pueden ser diferentes. Otra característica importante de este modo es la posible utilización de Sprites.

La dirección básica del generador de caracteres para la gráfica 1 la obtenemos con >BASE(7)<, y la dirección de la tabla de nombres con >BASE(5)<. Estas tablas son análogas a las formadas en el modo de texto (ver allí). La tabla de nombres ha de ser adaptada naturalmente al nuevo formato de indicación. Su longitud es por lo tanto 32*24 = 768 = &H300 Bytes. Los registros del VDP 2 y 4 se utilizan igual que en el modo de texto.

Vamos a ver ahora el tratamiento de los colores:
Con el comando >COLOR< se pueden determinar independientemente los colores de escritura, fondo y margen lateral. El registro VDP 7 memoriza el color del margen lateral en los 4 Bits inferiores. El color de fondo y el color de la escritura se determinan en el modo gráfica 1 a través de una tabla de colores. Por lo tanto, los 4 Bits superiores del registro 7 están sin utilizar.
La dirección de inicio de la tabla de colores se puede determinar con >BASE(6)<. Los 8 Bits superiores de la dirección de la tabla de colores están memorizados en el registro VDP 3. Por esto, la tabla de colores ha de ser desplazada a pasos de &H40=2^7=2^(14-7).

¿Cómo está construida una tabla de colores?

Al conectar el ordenador, Vd. obtendrá simultáneamente un color para la escritura y otro para el fondo. Pero es posible utilizar simultáneamente los 16 colores.

Para ello, cada Byte de la tabla de colores contiene información sobre el color de fondo y de la escritura. Igualmente que en el modo de texto al registrar en el registro VDP 7, los cuatro Bits inferiores son responsables del color de fondo y los cuatro superiores del color de la escritura. Todos los caracteres posibles del generador de caracteres están divididos en grupos de 8 caracteres consecutivos. Por medio de la inscripción de los valores correspondientes en la tabla de colores, se puede asignar a cada uno de estos grupos un color independiente de los demás grupos. El color del carácter con los códigos 0 hasta 7 es, por lo tanto, determinado por el primer Byte de la tabla de colores. Los códigos 8 hasta 15 por el segundo Byte, etc. Por lo general vale:
El color del carácter con el código N, es determinado por el Byte 1+INT(N/8) de la tabla. De esta forma, la tabla de colores tiene una longitud de 32 Bytes.

Si el color se determina desde el BASIC con el comando >COLOR<, entonces todos los 32 Bytes de la tabla de colores son ocupados con el mismo valor, por ejemplo, >COLOR 15,4,4<. Con >VPOKE< existe la posibilidad de colorear diferentemente los grupos. Como ejemplo veamos cómo resaltan todas las letras mayúsculas por medio de la representación inversa.
Las letras mayúsculas ocupan los códigos desde &H41 (=A) hasta &H5A (=Z). Por lo tanto, hay que ocupar los Bytes desde INT(&H41/8)+1 hasta INT (&H5A/8)+1, o sea, los Bytes 9 hasta 12 con los valores de color inversos.

```
FOR I = BASE(6) + 8 TO BASE(6) + 11: VPOKE I,15+4*16:NEXT
```

Naturalmente también aquí se puede interconectar entre diferentes tablas de colores.

La división de la RAM de vídeo después de la conexión es la siguiente:

```
Generador de caracteres : $H0000 - &H07FF
                  libre : &H0800 - &H17FF
          Tabla de nombres : &H1800 - &H1AFF
       Atributo de Sprite : &H1B00 - &H1B7F
                  libre : &H1B80 - &H1FFF
          Tabla de colores : &H2000 - &H201F
                  libre : &H2020 - &H37FF
        Patrón de Sprite : &H3800 - &H3FFF
```

(Las tablas de Sprites se tratarán en el capítulo 2.6)

Lamentablemente, no es posible el cambio con el comando >BASE< en el modo gráfico.
La rutina que ejecuta el comando >BASE<, tiene un error en nuestro ordenador Sony. Escriba Vd., para comprobarlo, una segunda tabla de colores a partir de &H2040:

```
FOR I=&H2040 TO &H105F : VPOKE I,&H4F: NEXT
```

Cambie Vd. primeramente a la nueva tabla de colores con el comando >VDP<.

```
&H2040/&H40 = 129, o sea,

VDP(3) = 129
```

Ahora obtendrá Vd. los nuevos colores. Con

```
VDP(3) = 128
```

se puede volver hacia el estado anterior. El mismo resultado se debería obtener con:

```
BASE(6) = &H2040
```

¡Pero éste no es el caso! El siguiente truco nos solucionará el problema:

```
DEF USR 9 = &H7B
```

Después de esta introducción, todos los comandos >BASE< que conciernen al modo gráfico I, pueden ser corregidos por medio del anexo >X=USR)(1)<. El resultado correcto lo obtenemos por:

```
BASE(6) = &H2040:X=USR9(1)
```

Lo mismo vale para los comandos >BASE(5)=...< hasta >BASE(9)=...<.
La ejecución de la función >BASE< , en la forma >...=BASE(..)< es correcta para todos los valores y no se puede añadir >X=USR9(1)<.

La conmutación con >BASE< o >VDP< se puede ahora utilizar para memorizar juntamente, como en el modo de texto, diferentes imágenes y visualizarlas a voluntad.

Se han de elaborar dos imágenes diferentes, con distintas tablas de colores y, como especialidad, con diversos generadores de caracteres. De esta forma podemos utilizar diferentes tipos de ecritura.
Elabore Vd. un plano de ocupación de memorias, en las cuales se pueden albergar estas informaciones. Tenga en cuenta que han de desplazarse:

- La tabla de nombres — a pasos de &H400
- El generador de caracteres — a pasos de &H800
- Las tablas de colores — a pasos de &H040
- La tablas de atributos de los Sprites — a pasos de &H080
- La tabla de muestra de los Sprites — a pasos de &H800

Sugerencia:

```
&H0000 - &H0800 Generador de caracteres 1
&H0800 - &H1000 Generador de caracteres 2
&H1000 - &H1300 Tabla de nombres 1
&H1300 - &H1320 Tabla de colores 1
```

```
   &H1320 - &H1400
   &H1400 - &H1700 Tabla de nombres 2
   &H1700 - &H1720 Tabla de colores 2
   &H1720 - &H1B00 libre
   &H1B00 - &H1B80 Atributo de Sprite
   &H1B80 - &H3800 libre
   &H3800 - &H4000 Muestra de Sprite

10000 A$ = INKEY$: IF A& ="" THEN 10000
10010 A =VAL(A$):IF A>1 THEN 10000
10020 DEFUSR 9=&H7B
10030 BASE(7) = &H800*A: USR9(1):REM Generador de caracteres
10040 POKE &HF925,&HB*A:REM Byte alto
10050 BASE(5) = &H1000+&H400*A: X=USR9(1):  REM Tabla de nom
      bres
10060 POKE &HF923, &H10+&H4*A: REM Byte alto
10070 BASE(6) = &H1300+&H400*A: X=USR9(1): REM Tabla de colo
      res
```

El programa funciona análogamente al programa del capítulo anterior. Las direcciones BASIC del generador de caracteres son:

&HF924 (LOW) y &HF925 (HIGH).

En esta dirección hay que escribir adicionalmente la dirección inicial del generador de caracteres. Naturalmente, antes hay que definir el segundo juego de caracteres.
Pruebe Vd. este pequeño programa:

```
1 VDP(2) = VDP(2) XOR 1: GOTO 1
```

XOR 1 origina que el Bit 0 del registro VDP 3 sea invertido. De esta forma se cambia constantemente entre la tabla de nombres actual y la tabla de nombres situado por encima (distancia &H400). Pero este cambio no funciona lo suficientemente rápido, para conseguir una solapación de las imágenes.
Para conseguir esto, tenemos que "colgarnos" de la rutina interna de interrupt. Esta rutina se activa 50 veces por segundo.

Permite, entre otras cosas, la programación de interrupts en BASIC.
Para poder insertar nuestras propias rutinas, utilizamos el segundo Patch de la rutina de interrupt. Su dirección es &HFD9F:

```
FD9F C360F3   JP    F360 ; Interrupt Patch
F360 F5       PUSH  AF ; salvar acumulador
F361 21F3E1   LD    HL,&HF3E1 ; Dirección del contenido de
registro 2
F364 7E       LD    A,(HL) ; leer valor del registro VDP 2
F365 EE01     XOR   01 ; invertir Bit, o sea, cambiar
F367 47       LD    B,A ; valor nuevo
F368 DE02     LD    C,2 ; ha de cargarse hacia el registro 2
F36A CD6047   CALL  &H9947 ; lleva VDP(C) = B en
F36D F1       POP   AF ; traer acumulador
F36E C9       RET   ; seguir en la ROM
```

El cargador BASIC de este programa es:

```
10 REM Interrupt conmutador de pantalla
20 CLEAR 200,&HF300
30 FOR I=&HF360 TO &HF36E
40 READ A$:POKE I,VAL("&H"+A$):NEXT
50 DATA F5,21,E1,F3,7E,EE,01,47
60 DATA 0e,02,CD,47,00,F1,C9
70 POKE &HFDA0,&H60
80 POKE &HFDA1,&HF3
90 REM activo con POKE &HFD9F,&HC3
100 REM desactivado con POKE &HFD9F,&HC9
```

Con este programa se consigue un cambio de la tabla de nombres. Naturalmente, también se pueden cambiar las otras tres tablas. Para ello son necesarias las siguientes modificaciones:

El número de registro correspondiente ha de introducirse como segundo código en la línea 60 (allí 02).
El segundo hasta cuarto código de la línea 50 representa el comando máquina LD HL,&HF3E1.

Esta es la dirección RAM, en la cual está memorizado el valor actual del registro VDP 2. Es necesario memorizar adicionalmente en la RAM todos los contenidos del registro de escritura del VDP, ya que los mismos no pueden ser leídos directamente. Por lo tanto, edite con:

```
PRINT VDP(2)
```

el valor del registro 2, así este contenido es determinado internamente por lectura de la posición de memoria &HF3E1. Similarmente, el valor del registro 0 y el valor del registro N están memorizados en la dirección &HF3DF y (&HF3DF+N) respectivamente.

De esto se deduce, que el tercer y cuarto número hexadecimal de la línea 50, son modificados según la dirección del registro. Primero se memoriza el Byte bajo, en nuestro caso &HE1, y después el Byte alto &HF3.

Con una programación hábil, Vd. puede conseguir, por ejemplo, diferentes formas de escritura simultáneamente sobre la pantalla, como cursiva y escritura gruesa. También es posible subrayar las palabras con este sistema.
El mismo método se puede utilizar para representar más de 32 (64) Sprites simultáneamente.
Pero de momento basta con "cambiar de un lado a otro". Al final del capítulo queremos presentarle un programa, con el cual se puede producir una copia en papel del contenido actual de la pantalla:

```
20000 REM SUB Copia del texto en papel
20010 DEFINT A-Z
20020 SN=PEEK(&HFCAF)
20030 IF SN>1 THEN RETURN
20040 BA=BASE(SN*5)
20050 AS=40-8*SN
20060 FOR Z=0 TO 23
20070 FOR S=0 TO AS-1
20080 BY=VPEEK (BA+Z*AS+S)
```

```
20090 IF BY>31 THEN LPRINT CHR$(BY);
20100 NEXT
20110 LPRINT
20120 NEXT
20130 RETURN
```

Lo especial de este programa es que funciona tanto con >SCREEN 0< como con >SCREEN 1<. El programa determina automáticamente el modo que está conectado. Para ello se consulta la memoria &HFCAF. Esta contiene el número del modo de pantalla actualmente conectado.
Hemos confeccionado este programa a propósito como subprograma, para que pueda insertarlo en sus programas.
En la línea 20 se obtiene el número actual >SCREEN< con >PEEK(&HFCAF)<. Esta no es la única posibilidad de determinar el modo. Como el VDP origina la imagen, también en los registros de éste han de estar memorizadas las cifras codificadas del modo.
Para este fin, entre otras cosas, se utilizan los registros VDP 0 y 1. También se denominan registros de comando. Para la selección del modo son responsables los Bits 3 y 4 del registro 1 , así como el Bit 6 del registro 0. Los Bits 3 y 4 también se denominan M1 y M2, y el Bit 6 del registro 0 también es denominado M3.

Entonces vale lo siguiente:

	M1	M2	M3
Modo de texto	1	0	0
Gráfica I	0	0	0
Gráfica II	0	0	1
Multicolor	0	1	0

M1 : Bit 3 del registro 1
M2 : Bit 4 del registro 1
M3 : Bit 6 del registro 0

2.3 El modo gráfico II

El modo gráfico II permite la máxima resolución de 256*192 puntos. En él, todos los 16 colores pueden utilizarse, así como los Sprites. Básicamente, el modo gráfico II es similar al modo I. Pero en el modo II se ha previsto un generador de caracteres mayor, de forma que para cada una de las 768 (32*24) posiciones de caracteres sobre la pantalla, se puede definir un propio carácter, diferente de todos los demás. Esto significa, que cada uno de los 49152 puntos de imagen puede ser posicionado o borrado. Además existe la posibilidad de determinar diferentes colores para un carácter de 8*8. Para cada Byte (8 puntos) de cada carácter compuesto por 8 Bytes, se pueden elegir dos colores. Con ello, el generador de caracteres y la tabla de colores tienen una longitud de &H1800 Bytes.

División VRAM

Igualmente que el modo gráfico I, la tabla de nombres en el modo II, está compuesta por 768 inscripciones, que corresponden exactamente a las 768 posiciones en la pantalla. Al tener los nombres del modo I exactamente una longitud de 8 Bits, se pueden direccionar como máximo 256 diferentes definiciones de caracteres, de la forma expuesta en el último capítulo.

Ahora en el modo gráfico II se han de direccionar 768 diferentes definiciones de caracteres. Para ello se divide la pantalla en tres partes iguales con 256 posiciones de carácter cada una. De la misma forma se dividen las tablas de nombres, color y caracteres. Entonces, el tercio superior de las tablas corresponde al tercio superior de la pantalla, el tercio central de las tablas corresponde a la parte central de la pantalla, y de forma idéntica el tercio inferior. Con ayuda de este truco es posible representar 768 caracteres. La desventaja de este sistema consiste en que, los tercios son independientes entre sí. No se puede representar directamente una definición de carácter del primer tercio en otro tercio de la pantalla.

Pero como veremos, esto no tiene mucha importancia.

Veamos un ejemplo:

En la tabla de nombres, en la posición 300, o sea, en el segundo tercio, se encuentra en valor 10. Por consiguiente, en la posición 300 de la pantalla (línea 9,columna 12) se visualiza el carácter con el código 10, o sea, el décimo carácter del tercio central(!) del generador de caracteres. Los colores del carácter son determinados por los Bytes 10*8 hasta 10*8+7 del segundo tercio de la tabla de colores.

	TABLA DE NOMBRES	GENERAD. CARAC.	TABLA DE COLOR	PANTALLA
1	0 : 256	0 : 2047	0 : 2047	Línea 0 : : : Línea 7
2	256 : 300=(10) : : 511	2048 : -->2047+10*8 a 2047+10*8+7 : 4095	2048 : -->2047+10*8 a 2047+10*8+7 : 4095	Línea 8 : en la línea 9 columna 12, se visualiza el carácter co- rrespondiente
3	512 : 767	4096 : 6143	4095 : 6143	en el color correspondien- te

A cada 8 puntos, o sea, un Byte del generador de caracteres corresponden 2 colores (2*4 Bit= 1 Byte) de la tabla de colores. En ello, el valor de los cuatro Bits inferiores determina el color de un punto posicionado, y el valor de cuatro Bits superiores determinan el color de un punto borrado.
Mire con atención el siguiente programa, el cual aclarará el asunto:

```
10 SCREEN 2
20 READ X,Y: REM posición
30 NT=BASE(10):REM tabla de nombres
40 FT=BASE(11):REM tabla de colores
50 ZG=BASE(12):REM generador de caracteres
60 BN=(32*Y+X)*8:REM número de Byte en la tabla de colores y
de caracteres
70 FOR I=0 TO 7
80  READ  B,VF,HF:REM  Byte, color de primer plano, color de
fondo
90 VPOKE ZG+BN+I,B:REM definición de carácter
100 VPOKE FT+BN+I,HF*16+VF: REM colores
110 NEXT I
120 N=(32*Y+X) MOD 256: REM nombre de carácter
130 FOR I=NT TO NT+767
140 VPOKE I,N
150 NEXT I
160 IF INKEY$=""THEN 160
170 DATA 9,12: REM posición
180 DATA &B00010000,3,15
190 DATA &B00101000,3,15
200 DATA &B01000100,3,15
210 DATA &B10010010,3,15
220 DATA &B01000100,3,15
230 DATA &B00101000,3,15
240 DATA &B00010000,3,15
250 DATA &B00010000,11,3
```

Los valores en las líneas DATA 170-250 pueden ser modificados según voluntad. Observe qué es lo que sucede. Primero se traspasa la definición de carácter a la >VRAM< a través del búcle en la línea 70 hasta 110. Tenga en cuenta, que aquí no hay que utilizar la tabla de nombres. En cuanto se conecta el modo >SCREEN 2<, la rutina de sistema correpondiente numera cada tercio de la tabla de nombres de 0 a 255.
Desde este momento, la tabla de nombres es inalterable. Esto significa, que todas las modificaciones en la pantalla se realizan directamente en el generador de caracteres. Esta es la diferencia básica entre el modo gráfico II con respecto al modo de texto o al modo gráfico I. En este último, en el

caso normal se trabaja con una tabla de colores fija. Unicamente se modifica continuamente la tabla de nombres.

En el modo de gráfica II casi siempre se mantiene sin modificar la tabla de nombres, y todas las modificaciones se realizan directamente en el generador de caracteres y en la tabla de colores. Las líneas 120 hasta 150 del anterior programa demuestran que es posible la modificación de la tabla de nombres , y las consecuencias que tiene. La tabla de nombres total es llenada con el nombre del caracter anteriormente definido. A causa de esto, el tercio superior e inferior se quedan vacíos, ya que la definición sólo vale para el tercio central. Este, por lo contrario, es llenado completamente con el carácter definido.
Pruebe Vd. el siguiente programa:

```
5 REM Desgarrador
10 SCREEN 2
20 NT=BASE(10): REM tabla de nombres
30 S=7: GOSUB 210: REM dibujar
40 GOSUB 170: REM Lío
50 S=20:GOSUB 210
60 GOSUB 120: REM Numerar
70 S=11: GOSUB 210
80 GOSUB 170
90 GOSUB 120
100 IF INKEY$="" THEN 100
110 END
120 FOR J=0 TO 2
130 FOR I=0 TO 255
140 VPOKE NT+J*256+I,I
150 NEXT I,J
160 RETURN
170 FOR I=NT TO NT+767
180 VPOKE I,RND(1)*256
190 NEXT I
200 RETURN
210 FOR I=10 TO 80 STEP S
220 CIRCLE (128,96),I
230 NEXT
240 RETURN
```

Piense Vd. en lo que ocurre internamente, al transcurrir el programa sobre la pantalla. Como Vd. ve, las rutinas de comando de gráfica BASIC parten siempre de una tabla de nombres numerada. La sucesión de la tabla de nombres nunca es modificada. Si hacemos ésto, como por ejemplo con la rutina a partir de la línea 170, entonces la imagen se descompone. Si se sigue dibujando sin numerar de nuevo, entonces no se forman de ninguna manera círculos (línea 50). Sólo después de una nueva numeración (línea 60) la imagen es perfectamente visible. Por lo tanto, no es aconsejable modificar en la tabla de nombres si se han de utilizar rutinas BASIC. Todas las modificaciones deberían efectuarse directamente en el generador de caracteres y en la tabla de colores.

La división estandar del VRAM en el modo de alta resolución es:

&H0 - &H1800	Generador de caracteres
&H1800 - &H1B00	Tabla de nombres
&H1B00 - &H1B80	Atributo Sprite
&H1B80 - &H2000	Libre
&H2000 - &H3800	Tabla de colores
&H3800 - &H4000	Muestra de Sprite

La dirección básica de la tabla de nombres es determinada, como es normal, por el registro VDP 2. Para la dirección básica del generador de caracteres y la tabla de colores vale excepcionalmente la siguiente regla:

Como ambas tablas tienen una extensión de 6K, pueden ser desplazadas a pasos de 8K, quiere decir que pueden comenzar en la dirección 0 ó en la dirección &H2000. Para distinguir entre estas dos posibilidades se escoge el Bit más elevado utilizado. En ello, 1 significa a partir de la dirección &H2000 y 0 a partir de la dirección 0. En la tabla de colores este es el Bit 7 del registro 3, y en el generador de caracteres es el Bit 7 del registro 4. Todos los Bits de menor valor han de posicionarse, además, a 1, si no surgen efectos bastante extraños, que sólo son aprovechables en pocos casos.

Con la división estandar , el VRAM está casi completamente aprovechado. Pero es posible albergar , por ejemplo, dos tablas de nombres o varias tablas de atributos. Si se

prescinde de la utilización de Sprites, entonces aumenta el número. Un desplazamiento es posible, como se ha visto, a través de la modificación del registro VDP con el comando >VDP< o con el comando >BASE<. Lamentablemente, también aquí existe un error. Para que >BASE< se ejecute correctamente con >SCREEN 2< conectada, hay que activar inmediatamente después la rutina de sistema a partir de la dirección &H007E. Ella posiciona los registros VDP con los valores correctos. Por lo tanto:

```
DEF USR8 = &H7E
BASE(...)=....: X=USR8(1)
```

El uso de este truco sólo es necesario, cuando >SCREEN 2< está conectada, y el contenido actual de la imagen no se va a modificar. Si esto no tiene importancia, entonces también es posible:

```
BASE(..) = ....:SCREEN 2 posible
```

Las direcciones del interpretador BASIC para la tabla de nombres y generador de caracteres son &HF922/3 y &HF924/5. Para que entienda y pueda aprovechar mejor las cualidades gráficas de su ordenador, le presentaremos algunos programas de aplicación.

Editor gráfico

A continuación sigue un programa con el cual, sin mucho despliegue, Vd. puede componer gráficas directamente en la pantalla. La utilización del programa se basa en un concepto de menú. Esto significa, que Vd. mueve constantemente un símbolo en este programa, entre otras cosas, una flecha con ayuda del joystick o de las teclas del cursor sobre la pantalla. Todas las funciones del programa son activadas posicionando el símbolo sobre el lugar marcado y pulsando el disparador o la tecla espaciadora. En el margen inferior de la pantalla se ven los 16 colores. Elija con la flecha el color actual de la escritura. Al pulsar el disparador, es elegido el color hacia el cual señale la flecha.

Junto con la escala de colores se ven las abreviaciones Cu. y Ra. Estas están escritas con el color actual. Eligiendo Cu. de la forma conocida (posicionar y fuego), obtendrá una C en lugar de la flecha. Eligiendo un color con la C (=cursor) la flecha vuelve a resurgir con el color elegido. Con la elección de la abreviación Ra., obtendrá una R como símbolo para el color del margen y puede elegir el color de éste.

En la parte superior de la pantalla obtendrá la siguiente indicación:

Pun/Lin/Rec/Pal/Tex

Pun - Modo de punto

Este es señalizado por una flecha. Si se encuentran en el campo de caracteres, entonces al pulsar el disparador se posiciona un punto con el color actual.

Lin - Modo de Líneas

Si se coloca con el símbolo delante de Lin y pulsa luego el disparador, obtendrá como símbolo una flecha con un agujero en la punta. Con ella se pueden dibujar líneas. Marque pulsando el disparador, los puntos inicial y final de la línea. Al pulsarlo por segunda vez, ambos puntos quedarán unidos por una línea. Todos los modos siguen vigentes hasta que se elija uno nuevo.

Rec - Rectángulo

El Modo Rec es simbolizado por el símbolo rectangular. Analógicamente a Lin, se marcan dos puntos. Pero en este modo, se dibuja el rectángulo que tiene como diagonal a la línea que une ambos puntos.

Pai - Paint

Después de elegir Pai, obtendrá un rectángulo coloreado. Un sector cubierto por el color actual será cubierto por el nuevo color actual. Tenga cuidado de que esté verdaderamente cerrado el sector a colorear, ya que, de lo contrario, será coloreada la pantalla en su totalidad.

Tex - Modo de texto

Después de la elección de Tex, obtendrá una T como cursor, con el cual se pueden editar letras o caracteres. Después de pulsar el disparador se espera a la pulsación de una tecla normal. El carácter correspondiente a esta tecla surge en la posición actual del cursor.

```
10 DEFINT A-Z
20 EI=1:REM JOY
30 CC=1:REM SPRITE CURSOR COLOR
40 OPEN "grp:" FOR OUTPUT AS #1
50 SCREEN 2,0:COLOR 15,4,4
60 FOR I=0 TO 2
70 READ CH$:CH=ASC(CH$):BA=&H1BBF+8*CH
80 FOR J=0 TO 7
90 A$=A$+CHR$(PEEK(BA+J)):NEXT J
100 READ N:SPRITE$(N)=A$:A$=""
110 NEXT I
120 FOR I=0 TO 7:READ A:A$=A$+CHR$(A):NEXT
130 SPRITE$(0)=A$
140 A$=CHR$(ASC(A$))+CHR$(&H48)+CHR$(&H48)+RIGHT$(A$,5)
150 SPRITE$(3)=A$
160 SPRITE$(4)=CHR$(&H7F)+STRING$(6,&H41)+CHR$(&H7F)
170 SPRITE$(5)=STRING$(8,&H7F)
180 X=120:Y=80:XA=X:YA=Y
190 READ XL,XR,YO,YU
200 READ JO,JU
210 LINE (XL,YO+JO)-(XR,YU-JU),2,B
220 FOR I=0 TO 15:LINE (16+I*8,YU-1)-(22+I*8,YU+7),I,BF:NEXT
230 SF=15:HF=4:GOSUB 780
240 ON STRIG GOSUB 370,370,370,370,370:STRIG(EI) ON
250 PSET (16,YO):PRINT#1,"Pun/Lin/Rec/Pai/Tex/"
260 PUT SPRITE 0,(X,Y),CC,M
270 RI=STICK(EI):IF RI=0 THEN 270
280 IF RI>1 AND RI<5 THEN X=X+XP:XP=XP+2 ELSE XP=1
290 IF RI>3 AND RI<7 THEN Y=Y+YP:YP=YP+2 ELSE YP=1
300 IF RI>5 AND RI<9 THEN X=X-XM:XM=XM+2 ELSE XM=1
310 IF RI=1 OR RI=2 OR RI=8  THEN Y=Y-YM:YM=YM+2 ELSE YM=1
320 IF X>XR THEN X=XR
330 IF X<XL THEN X=XL
340 IF Y>YU THEN Y=YU
350 IF Y<YO THEN Y=YO
360 GOTO 260
```

```
370 REM strig
380 YM=0:YP=0:XM=0:XP=0
390 IF Y>YO+JO AND Y<YU-JU THEN 610
400 IF Y<=YO+JO THEN 520
410 PO=INT((X-16)/8)
420 IF PO<0 THEN RETURN
430 IF PO>15 THEN 490
440 IF M<>1 AND M<>2 THEN SF=PO:GOSUB 780:RETURN
450 IF M=1 THEN COLOR ,,PO
460 IF M=2 THEN CC=PO
470 GOSUB 780
480 M=MA:RETURN
490 IF PO=16 THEN RETURN
500 IF PO<23 THEN MA=M:M=INT((PO-14)/3):RETURN
510 RETURN
520 REM linea superior de comandos
530 PO=INT((X-16)/32)
540 NF=0:RF=0:LF=0:PF=0:TF=0
550 IF PO<1 THEN NF=-1:M=0:RETURN
560 IF PO<2 THEN LF=-1:M=3:RETURN
570 IF PO<3 THEN RF=-1:M=4:RETURN
580 IF PO<4 THEN PF=-1:M=5:RETURN
590 IF PO<5 THEN TF=-1:M=6:RETURN
600 RETURN
610 REM activar punto
620 IF M=1 OR M=2 THEN RETURN
630 IF NF=-1 THEN 700
640 IF TF THEN GOSUB 720:RETURN
650 IF LF=1 THEN LINE (XA,YA)-(X,Y),SF:LF=-1:RETURN
660 IF LF=-1 THEN XA=X:YA=Y:LF=1
670 IF RF=1 THEN LINE (XA,YA)-(X,Y),SF,B:RF=-1:RETURN
680 IF RF=-1 THEN XA=X:YA=Y:RF=1
690 IF PF THEN PAINT (X,Y),SF:RETURN
700 PSET (X,Y),SF
710 RETURN
720 REM TEXTO DESCONECTADO
730 POKE &HF3F8,PEEK(&HF3FA):POKE &HF3F9,PEEK(&HF3FB)
740 PSET(X,Y),0
```

```
750 A$=INKEY$:IF A$="" THEN 750
760 PRINT #1,A$;
770 RETURN
780 REM SALIDA SHBC
790 LINE (150,184)-(198,192),HF,BF
800 COLOR SF,HF
810 PSET (152,184),SF:PRINT#1,"Ra.Cu."
820 RETURN
830 DATA R,1,C,2,T,6
840 DATA &B01111100
850 DATA &B01111000
860 DATA &B01111000
870 DATA &B01111100
880 DATA &B01001110
890 DATA &B00000111
900 DATA &B00000011
910 DATA &B00000000
920 DATA 0,249,0,185
930 DATA 10,4
```

Descripción del programa

Línea 40:

Edición de gráfica: Abre datos para la edición de texto sobre la pantalla gráfica.

Línea 60:

Bucle para la definición de tres Sprites que han de representar letras

Línea 70:

El carácter es leído en CH$ y el código ASCII correspondiente es memorizado en CH. BA contiene la dirección básica del carácter actual en la ROM del generador de caracteres (dirección inicial &H1BBF).

Línea 80:

Bucle para lectura de 8 Bytes por carácter.

Línea 90:

Transformación de la definición de carácter en un String (A$) para una definición de Sprite.

Línea 100:

Leer número de Sprite, definir Sprite y borrar A$.

Línea 120,130:

Definir una flecha como Sprite.

Línea 140,150:

Produce la flecha con el agujero en el centro, que indica "Lin".

Línea 160:

Origina el cuadrado vacío que indica "Rec"

Línea 170:

Origina el cuadrado relleno para "Pal".

Línea 180:

X,Z:Posición de inicio del Sprite
XA,YA: antigua posición para dibujar la línea/rectángulo

Línea 190:

Lee la limitación de la pantalla : XL = Izquierda, XR = derecha, YO = arriba, YU = abajo

Línea 200:

Diferencias de la pantalla total al campo de dibujo JO = arriba, JU = abajo

Línea 210:

Dibuja el marco.

Línea 220:

Dibuja los cuadrados coloreados en el margen inferior de la pantalla.

Línea 230:

SF = Color de la escritura y HF = color de fondo se posicionan a sus valores estándar.

Línea 240:

Definición de Interrupt para disparador o tecla espaciadora y permitir Interrupt

Línea 260:

M = número de Sprite actual
CC = Color del cursor
Indicación del Sprite de cursor correspondiente.

Línea 270 hasta 360:

Gobierno del Sprite de cursor, en lo cual XP, YP, XM e YM modifican las direcciones de movimiento. La velocidad aumenta cuanto más tiempo se mantenga la dirección.

Línea 370:

Rutina de Interrupt para tratamiento del "String".

Línea 380:

Reposicionar a 0 la velocidad del Sprite del cursor.

Línea 390:

Consulta, si el Sprite del cursor está en el campo de dibujo.

Línea 400:

Consulta, si el Sprite del cursor está por encima del campo de dibujo, después se bifurca a la interpretación de las series de comandos.

Línea 410:

Determinar la posición en la fila inferior.

Línea 420:

Si está a la izquierda de los campos de color, entonces no válido.

Línea 430:

Si está a la derecha de los campos de color, entonces seguir en la línea 490

Línea 440:

Posicionar el color de la escritura por Sprite del cursor (SF=P0, color de escritura = campo de color). Salto para la edición shbc.

Línea 450:

Posicionar color del margen.

Línea 460:

Posicionar color del Sprite del cursor.

Línea 470:

Saltar hacia la edición del color.

Línea 480:

Activar Sprite antiguo.

Línea 490:

Si posición no es válida en el campo de comandos inferior, no pasa nada.

Línea 500:

Reconocimiento de comando para "Cu y RA".

Línea 530:

Comprobar la posición dentro de la fila de comandos.

Línea 540:

NF = Modo normal; LF = Modo de línea; RF = modo rectangular; PF = modo de Paint; TF = modo de texto.

Línea 550:

Cuando el Sprite del cursor esté sobre Pun, posicionar punto.

Línea 560 hasta 590:

Si está "Lin" entonces activar Sprite no. 3, posicionar flag de modo.

Línea 620:

Excluir Sprites que no están permitidos (C y R).

Línea 630:

Con NF =-1 hacia 700. Si se trabaja en el modo normal, se posiciona inmediatamente un punto.

Línea 640:

Si se trabaja en el modo de texto, saltar hacia 720.

Línea 650:

Unir segundo punto de la línea X,Y con el primer punto XA,XY.

Línea 660:

Registra el primer punto y posiciona LF a 1, para que la próxima vez se produzca la unión.

Línea 670,680:

La misma técnica que en las líneas 650,660.

Línea 690:

Colorear superficie

Línea 700:

Posicionar punto

Línea 720:

Edición de texto

Línea 730:

Borrar buffer de caracteres

Línea 740:

Determinar la posición actual

Línea 750:

Consulta al teclado

Línea 760:

Edición por fichero sobre la pantalla.

Línea 780:

Ediciones de escritura y color de fondo y de margen

Línea 800:

Posicionar color

Línea 810:

Editar las palabras de comando "Ra,Cu" inferiores.

Línea 830 hasta 930:

Definición de Sprite para la flecha.

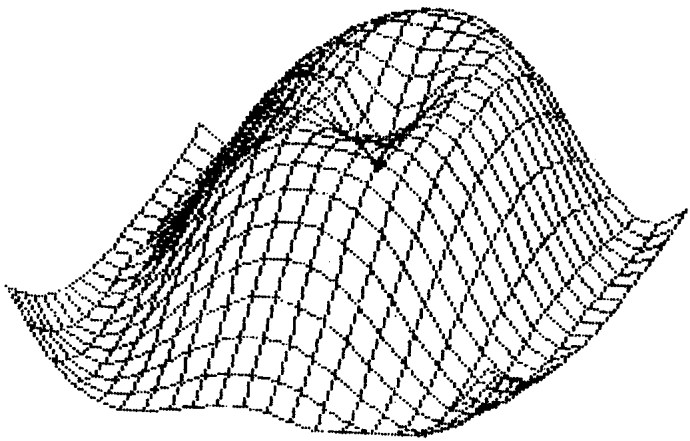
Gráfica 3D

Una de las aplicaciones más excitantes y cautivadoras de la gráfica es la representación de cuerpos o funciones tridimensionales. Lamentablemente, la elaboración de estas imágenes tarda, algunas veces, horas e incluso días.

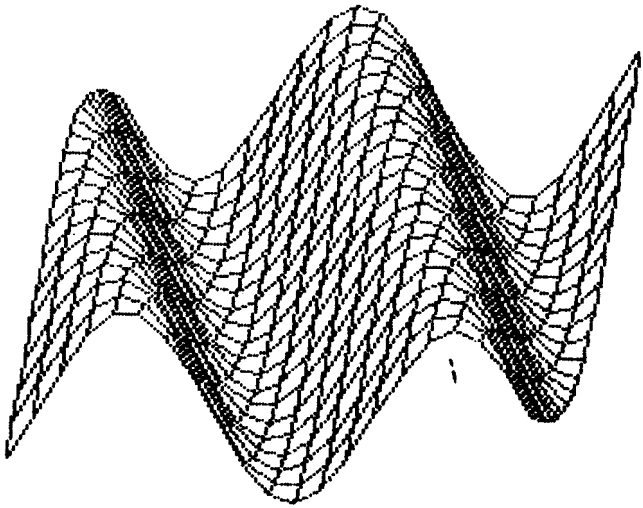
El siguiente programa necesita, en su versión básica, sólo pocos minutos para dibujar una imagen completa.
La representación de funciones 3D consiste en reducirla, por medio de cortes paralelos a una representación bidimensional. La función 3D es cortada a lo largo del eje XY. La línea formada al cortar se muestra normal. El siguiente corte se realiza un poco hacia atrás. La línea resultante es dibujada desplazada respecto a la línea anterior, para que surja el efecto 3D. Este proceso se repite hasta que el sector a dibujar cubra todo el eje Z. Con este método se consigue una gráfica de líneas 3D.
A veces, esto no es suficiente para conseguir una representación realista. Esto sólo se consigue con un recubrimiento de malla. Un ejemplo para una malla es un globo terráqueo, en el cual se marcan los paralelos y los meridianos. El siguiente programa dibuja una gráfica enmallada de una función cualquiera para un intervalo cualquiera. La anchura de malla influye terminantemente en la velocidad de dibujo. Una malla muy amplia se dibuja rápidamente, pero transmite sólo una imagen aproximada de la función. Esto está motivado en el método de la gráfica de malla:

En la gráfica de malla no se calculan todos los puntos de la curva, por eso la ventaja de la velocidad con repecto al método convencional de dibujo. Sólo se calculan los nudos de la red y luego se unen éstos a través de líneas. En principio se puede dibujar cualquier curva. Pero se ha de corregir el sector de validez, quiere decir, los límites sobre los ejes entre los que ha de dibujarse, para obtener imágenes aprovechables.

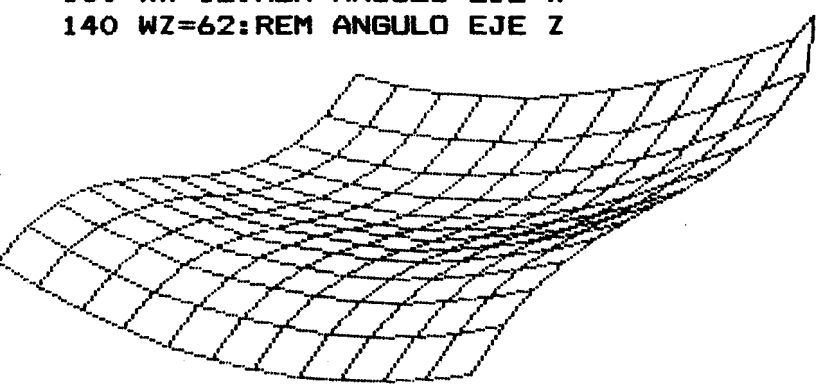
En muchas funciones será molestoso que se dibuje toda la función, incluso las partes tapadas por otras. En realidad, solo vemos la parte del globo dirigida hacia nosotros, pero no la parte posterior, a no ser que éste esté hecho de cristal. Esto significa, que Vd. obtiene la imagen de una función de cristal cubierta con una malla. A veces, esta superposición se puede eliminar en parte modificando algo el ángulo de vista. Una solución general para el problema de esta superposición no es posible con nuestros medios.



```
40 DEFFNF (X,Z)=SIN(SQR(X*X+Z*Z))
90 DATA -5,5,-1.9,1.9,-2,4
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM ANGULO EJE X
140 WZ=52:REM ANGULO EJE Z
```



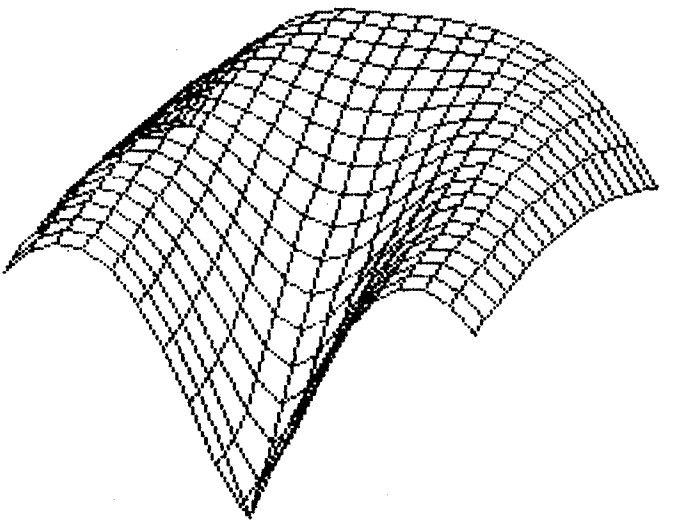
```
40 DEFFNF (X,Z)=SIN(X-Z)
90 DATA -5,5,-1.9,1.9,-2,2
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM ANGULO EJE X
140 WZ=62:REM ANGULO EJE Z
```



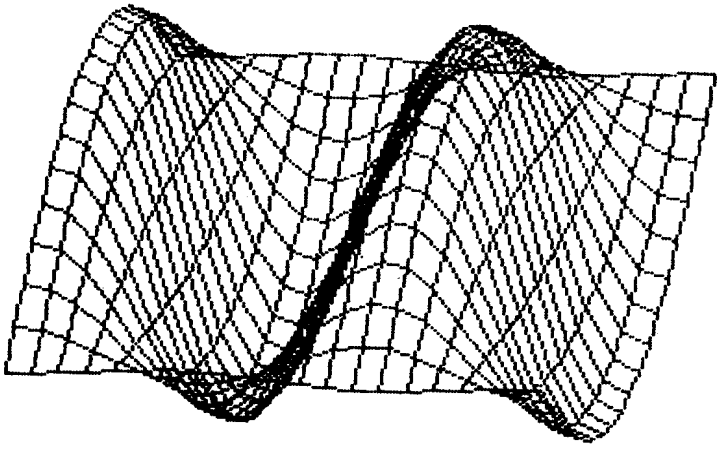
```
40 DEFFNF (X,Z)=(X-Z)^2+2*X-.05*Z*Z*Z+3
90 DATA -400,400,-500000,500000,-170,170
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM ANGULO EJE X
140 WZ=32:REM ANGULO EJE Z
```

```
10 ' 3-D Dibujo grafico de funciones de Holger Dullin
20 CLEAR 500,&HEFFF
30 DEFUSR1=&HF000
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
50 SCREEN 2
60 AP=18:REM cantidad de puntos
70 DIM X(AP+1),Y(AP+1)
80 READ XA,XE,YA,YE,ZA,ZE:REM limitacion de los ejes
90 DATA -2.1,2.1,0,1.3,-1.6,.1
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM angulo eje X
140 WZ=52:REM angulo eje Y
150 DE=3.141529#/180
160 ZX=COS(WZ*DE)^2
170 XX=COS(WX*DE)^2
180 ZP=SIN(WZ*DE)^2
190 XY=SIN(WX*DE)^2
200 XO=XA/(XA-XE)*256
210 YO=YE/(YE-YA)*192
220 J=0
230 FOR Z=ZE TO ZA STEP -(ZE-ZA)/AP
240 I=0
250 FOR X=XE TO XA STEP -(XE-XA)/AP
260 Y=FNF(X,Z)
270 XK=XX*MX*X-ZX*MZ*Z
280 YK=MY*Y-ZY*MZ*Z-XY*MX*X
290 XK=XK+XO:YK=YO-YK
300 IF I=0 THEN 320
310 LINE (XK,YK)-(X(I),Y(I))
320 I=I+1
330 IF J=0 THEN 350
340 LINE (XK,YK)-(X(I),Y(I))
350 X(I)=XK
360 Y(I)=YK
370 NEXT X
380 J=J+1
```

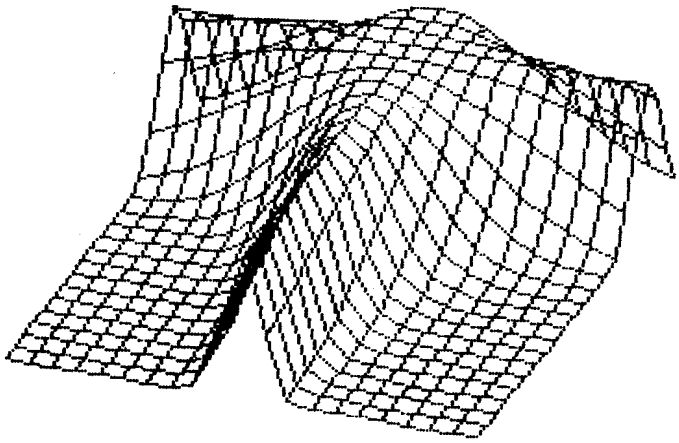
```
390 NEXT Z
400 A$=INKEY$:IF A$="" THEN 400
410 IF A$<>"j" AND A$<>"J" THEN END
420 X=USR1(1)
430 END
```



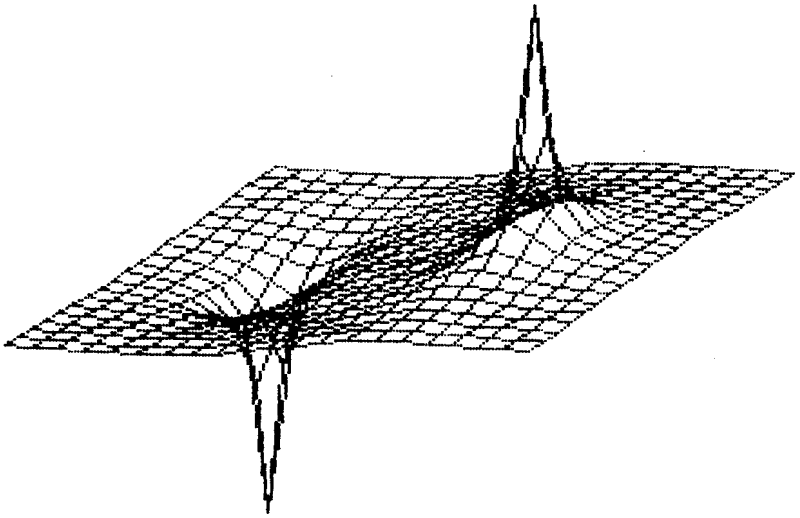
```
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
90 DATA -2.1,2.1,0,1.3,-1.6,.1
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM ANGULO EJE X
140 WZ=52:REM ANGULO EJE Z
```



```
40 DEFFNF(X,Z)=SIN(X)*COS(Z)
90 DATA -4.8,4.8,-1,1,-1.6,1.6
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=170/(ZE-ZA)
130 WX=10:REM ANGULO EJE X
140 WZ=62-WX/2:REM ANGULO EJE Z
```



```
40 DEFFNF(X,Z)=EXP(-X*X*Z*Z)
90 DATA -3,3,-1.2,1.2,-.5,3
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM ANGULO EJE X
140 WZ=52:REM ANGULO EJE Z
```



```
40 DEFFNF(X,Z)=X/SQR((1-X*X)^2+
   (2*X*Z/300)^2+1E-03)
90 DATA -2,2,-7,7,-300,300
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM ANGULO EJE X
140 WZ=42:REM ANGULO EJE Z
```

Descripción del programa gráfica 3D

Línea 20:

Reservar espacio para programa de copia en papel

Línea 30:

Determinar la dirección de inicio de la copia en papel

Línea 40:

Definición de la función

Línea 60:

AP determina la cantidad de cuadrados de malla, que se han de formar a lo largo de un eje.
Valores altos (hasta máximo 30) producen una mejor imagen A cambio de esto, se tarda más en realizarla.
Valores AP pequeños (hasta mínimo 10) suministran rápidamente el transcurso aproximado de la función.

Línea 70:

Reserva de las variables de campo, que memorizan las coordenadas de los puntos dibujados últimamente.

Línea 80,90:

Aquí, en las líneas DATA situadas por debajo, se leen las limitacines de los ejes. Ellas determinan el sector de la función que ha de representarse.
Si no se consigue ninguna imagen satisfactoria, han de modificarse los valores. Si Vd. no tiene ninguna idea sobre la elección de estos valores para una función desconocida, utilice primeramente valores grandes.

Línea 100 hasta 120:

Factor de escala para la tranformación a coordenadas de pantalla para los ejes X, Y y Z. Las cifras indicadas sólo han (y habrían) de cambiarse en casos excepcionales.

Línea 130:

WX es el ángulo que comprende el eje X con la horizontal. Este ángulo es muy decisivo para el aspecto de la imagen.

Línea 140:

WZ es el ángulo que comprende el eje Z con la horizontal. También este ángulo es muy importante para el aspecto de la imagen.

Línea 150:

Factor de transformación de grados a radianes.

Línea 160 hasta 190:

En función de los ángulos WZ y WX de los ejes, aquí se determinan los factores de alargamiento y reducción para las diferentes direcciones de los ejes. Estos factores originan que, distancias que en la realidad se observan bajo un ángulo agudo, sean reducidas debidamente en el dibujo.

Línea 200,210:

XO e YO son sumandos que se utilizan para transformar las coordenadas reales en coordenadas de pantalla. En ello se tienen en cuenta los límites fijados.

Línea 220:

I=0 significa, que en este momento se dibuja la primera línea en el sentido X.

Línea 230:

Por medio del bucle Z, la imagen real es cortada en rodajas. La línea que se forma al cortar, es la que se dibuja.

Línea 240:

I es el contador que contiene el número del cuadrado de malla actual.

Línea 250:

A través del bucle X se dibuja una línea del "corte Z".

Línea 260 hasta 290:

Cálculo del valor de la función.
Transformación de 3D a 2D.
Transformación a coordenadas de pantalla.

Línea 300:

Si el comienzo de la línea (I=0), no dibujar línea de unión (horizontal).

Línea 310:

Dibujar línea de unión horizontal.

Línea 330:

Si es la primera línea, entonces no dibujar línea de unión vertical.

Línea 340:

Dibujar líneas de unión verticales.

Líneas 350,360:

Dibujar coordenadas de malla.

Líneas 390 hasta 430:

Con introducción de J, copia en papel (programa de copia ha de haberse cargado anteriormente), si no final.

Copia de gráfica en papel

Para poder copiar sobre papel los dibujos elaborados con el editor de gráfica y el plotter de función 3D, le vamos a presentar un programa para copia en papel.
El programa funciona sin modificación alguna sobre la impresora EPSON FX-80 y compatibles. Para la adaptación a otras impresoras hay que modificar únicamente la secuencia de mando, suponiendo que exista un modo de Bit de 8 puntos.
Para la copia en papel se lee directamente el generador de caracteres en el modo de alta resolución. Como Vd. sabe, 8 Bits consecutivos representan un carácter, los 8 siguientes otro carácter, etc. El problema consiste ahora, en que la impresora imprime los 8 puntos uno debajo del otro, y no como la memoria de pantalla, cada punto al lado del otro.

Ejemplo:

Códigos de memoria de pantalla

00010000 = 16
00101000 = 40
01000100 = 68

```
 10111010  = 186
 01000100  = 68
 00101000  = 40
 00010000  = 16
 11111110  = 254
 !!!!! !!
.!! ! 17!
17!147! !
  !! !! !
 41!85! 0    código de Impresora
   !  !
  85  41
```

El carácter es editado sobre la pantalla por medio de:

```
FOR I=32 TO 39: READ A
VPOKE I,A:NEXT
DATA 16,40,68,16,68,40,16,254
```

Para la edición por Impresora hay que enviar los valores formados por columnas.

```
LPRINT CHR$ (27);"K";CHR$(8);CHR$(0);
FOR I=0 TO 7:READ A
LPRINT CHR$(A);:NEXT
DATA 17,41,85,147,85,41,17,0
```

El primer comando >LPRINT< es la serie de códigos de mando para la EPSON FX-80, la cual anuncia la edición de 8 códigos gráficos en el modo de patrón de Bit.

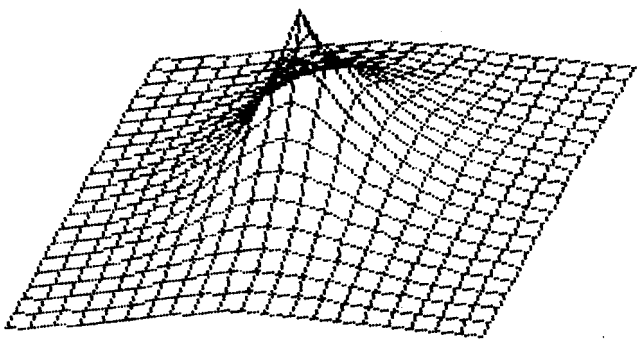
El programa de copia en papel ha de transformar paso a paso el juego total de caracteres de pantalla según la prescripción anterior, para poder enviar los valores por columna. Como esto en BASIC puede tardar unas horas, es apropiada una solución en lenguaje máquina.

```
 F000           10                ; Hardcopy
 F000           20  PRINT   EQU   &H00A5
 F000           30            ORG   &HF000
?F000 210000    40            LD    HL,TABLE1 ; Tabla caracteres
de control
 F003 7E        50            LD    A,(HL)
 F004 47        60            LD    B,A ; Cantidad caracteres de
control
 F005 23        70  NESTE1    INC   HL
 F006 7E        80            LD    A,(HL)
 F007 CDA500    90            CALL  PRINT
 F00A 10F9      100           DJNZ  NESTE1
 F00C 210000    110           LD    HL,&H0000 ; Tabla de nombres
 F00F 0618      120           LD    B,24 ; Contador de lineas
 F011 C5        130 NEZEIL    PUSH  BC
 F012 E5        140           PUSH  HL
?F013 210000    150           LD    HL,TABLE2 ; Tabla caracteres
de control
 F016 7E        160           LD    A,(HL)
 F017 47        170           LD    B,A ; Cantidad caracteres de
control
 F018 23        180 NESTE2    INC   HL
 F019 7E        190           LD    A,(HL)
 F01A CDA500    200           CALL  PRINT
 F01D 10F9      210           DJNZ  NESTE2
 F01F E1        220           POP   HL
 F020 0620      230           LD    B,32 ; Contador de columnas
 F022 C5        240 NESPAL    PUSH  BC
 F023 E5        250           PUSH  HL ; Puntero de nombre
 F024 CDA50B    260           CALL  &H0BA5 ; Copiar en RAM la
definición de caracteres
 F027 0608      270           LD    B,8 ; Contador de Bytes
 F029 C5        280 NEBYTE    PUSH  BC
 F02A 2118FC    290           LD    HL,&HFC18
 F02D 0608      300           LD    B,8 ; Contador de Bytes
 F02F 97        310           SUB   A ; Borrar acumulador
 F030 CB06      320 NEBIT     RLC   (HL) ; Bit al acarreo
 F032 17        330           RLA   ; Acarreo al acumulador
 F033 23        340           INC   HL
 F034 10FA      350           DJNZ  NEBIT
 F036 CDA500    360           CALL  PRINT
 F039 C1        370           POP   BC ; Contador de Bytes
 F03A 10ED      380           DJNZ  NEBYTE
 F03C E1        390           POP   HL ;Puntero tabla de nombres
 F03D 110800    400           LD    DE,8
 F040 19'       410           ADD   HL,DE
```

```
 F041 C1       420           POP  BC ; Contador de columnas
 F042 10DE     430           DJNZ NESPAL
 F044 3E0A     440           LD   A,10 ; Line Feed
 F046 CDA500   450           CALL PRINT
 F049 3E00     460           LD   A,13
 F04B CDA500   470           CALL PRINT
 F04E C1       480           POP  BC ; Contador de líneas
 F04F 10C0     490           DJNZ NEZEIL
 F051 C9       500           RET
 F052          510                ; Tablas caracteres de
control
**** Línea 40 : TABLE1=&HF052
 F052 03       520  TABLE    DB   3 ; Longitud de la tabla
 F053 1B       530           DB   27 ; Código ESC
 F054 41       540           DM   "A" ; Avance de línea
 F055 08       550           DB   8 ; 8/72 Inch
**** Línea 150 : TABLE2=&HF056
 F056 05       560  TABLE2   DB   5 ; Longitud  de secuen-
cias siguientes
 F057 1B       570           DB   27 ; Código ESC
 F058 2A       580           DM   "*" ; Bit Image Selection
 F059 04       590           DB   4 ; Gráficos CRT
 F05A 0001     600           DW   256 ; 256 Bytes por línea
```

```
Programa: harcop
Inicio: &HF000  Final: &HF05B
Longitud: &H5C Bytes
Errores: 0
Tabla de variables:
PRINT  00A5    NESTE1 F005
NEZEIL F011    NESTE2 F018
NESPAL F022    NEBYTE F029
NEBIT  F030    TABLE1 F052
TABLE2 F056
```

```
10 CLEAR 200,&HEFFF:MAXFILES=1
20 FOR I=&HF000 TO &HF05B
30 READ A$:W=VAL("&H"+A$)
40 S=S+W:POKE I,W:NEXT
50 IF S<>8670 THEN BEEP:PRINT"Fehler in DATAs":END
60 PRINT"Todo correcto !"
70 DEFUSR1=&HF000
80 DATA 21,52,F0,7E,47,23,7E,CD
90 DATA A5,00,10,F9,21,00,00,06
100 DATA 18,C5,E5,21,56,F0,7E,47
110 DATA 23,7E,CD,A5,00,10,F9,E1
120 DATA 06,20,C5,E5,CD,A5,0B,06
130 DATA 08,C5,21,18,FC,06,08,97
140 DATA CB,06,17,23,10,FA,CD,A5
150 DATA 00,C1,10,ED,E1,11,08,00
160 DATA 19,C1,10,DE,3E,0A,CD,A5
170 DATA 00,3E,0D,CD,A5,00,C1,10
180 DATA C0,C9,03,1B,41,08,05,1B
190 DATA 2A,04,00,01
```



```
40 DEFFNF(X,Z)=XP(-SQR(X*X+Z*Z))
90 DATA -3,,-0.9,0.9,-2,2
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM ANGULO EJE X
140 WZ=52:REM ANGULO EJE Z
```

2.5 El modo Multicolor

Este modo ofrece la visualización de 64*48 cuadrados en color. Cada cuadrado está compuesto por 4*4 puntos. El color de cada uno de estos cuadrados puede ser cualquiera de los 16 posibles. De esta forma se pueden utilizar simultáneamente todos los 16 colores. Los Sprites están a la total disposición.
La tabla de nombres es la misma que en ambos modos gráficos. Está compuesta por 768 inscripciones, con lo cual el nombre no señala hacia la tabla de colores sino sólo hacia el generador de caracteres. Por lo tanto, el color es determinado por el generador de caracteres. Generalmente una definición de carácter tiene una longitud de 8 Bytes, y por lo tanto, un nombre señala sobre un sector de 8 Bytes del generador de caracteres.

Sólo 2 de estos 8 Bytes determinan la visualización sobre la pantalla. Estos dos Bytes determinan 4 colores, en lo cual cada color cubre un cuadrado del tamaño de 4*4 puntos. Los 4 MSBs (de mayor valor 4-7) del primer Byte determinan el color del cuarto superior izquierdo de la posición actual de dibujo. Los 4 Bits de menor valor determinan el color del cuarto superior derecho. El segundo Byte define correspondientemente los colores de los cuartos derecho e izquierdo inferiores.
Las posiciones del segundo Byte afectado dentro de este sector de 8 Bytes sobre el cual señala el nombre, depende de la posición del nombre en la tabla de nombres, o sea, de la posición de pantalla afectada. Para los nombres de la primera línea (0+31) se utilizan los dos primeros Bytes del sector de 8 Bytes. La siguiente línea de nombres (32-63, corresponde a la línea de pantalla 2) utiliza los Bytes 3 y 4. La siguiente línea utiliza los Bytes 5 y 6 , y finalmente la última línea los Bytes 7 y 8. Este esquema se prosigue para el total de la pantalla.
La ocupación de los registros VDP y la utilización del comando >BASE< es análogo a lo hasta ahora aprendido. Por lo demás queda poco que decir sobre este modo. La resolución es demasiado baja para que con ella se puedan producir gráficas interesantes.

Unicamente en conjunto con los Sprites se puede utilizar el mutlicolor como fondo. Veamos, por lo tanto, los Sprites.

2.6 Sprites

Las posibilidades de representación de Sprites en el MSX son bastante buenas. Se pueden programar hasta 32 Sprites. Con ayuda de estos Sprites se puede representar movimientos de manera excelente sobre la pantalla.
En principio, un Sprite es sólo un carácter sobredimensional, que puede ser visualizado a través de comandos especiales sobre una pantalla gráfica. Un Sprite grande está formado por 16*16 puntos. Al igual que un carácter, el Sprite es definido en una así denominada tabla de patrones de Sprites. La posición del Sprite corresponde a las coordenadas de su esquina izquierda superior. A través de la modificación de las coordenadas de posición se puede mover el patrón de Sprite por toda la pantalla. Una representación de movimientos suaves y tranquilos, que en gráfica de alta resolución es difícil por motivos de tiempo, se puede programar aquí fácilmente.

Pero aún no se han agotado las posibilidades de los Sprites en ordenadores MSX. La visualización de los Sprites se efectua sobre los así denominados planos de Sprites. En cada uno de los 32 planos superpuestos, se puede visualizar un Sprite. Lo especial en este método consiste en el ''estar detrás'': un Sprite que está visualizado en un nivel de atrás, es tapado por otro Sprite que se encuentre en un nivel delantero, si los dos se solapan. Con ésto se pueden simular representaciones en 3D (sin perspectiva verdadera). La partes importantes, como por ejemplo figuras móviles, vehículos, etc., son representadas en niveles delanteros (cifras inferiores). De esta forma, estos Sprites tapan todos los demás, lo cual corresponde a la realidad. Imágenes, que en realidad sean observadas de mayor distancia, se dibujan en los niveles situados más atrás. En el último plano (=31) se dibujan, por ejemplo, nubes o cosas similares. Detrás del último plano de Sprites se encuentra el plano de pantalla propiamente dicho.

Esto significa que los Sprites tapan la indicación estandar de la pantalla.
La utilización de Sprites es idéntica en los modos 1,2 y 3. En el modo 0 (=modo de texto) no se pueden utilizar Sprites. El plano, en el cual se han de visualizar Sprites, es determinado por el primer parámetro del comando >PUT SPRITE<. Los Sprites se pueden utilizar en cuatro modificaciones diferentes. Existe, como ya se ha dicho, la posibilidad de utilizar Sprites de 16*16 o de 8*8 puntos. Independientemente de la cantidad de puntos se pueden representar Sprites a tamaño normal (=tamaño original) o ampliado (= doble tamaño). En el caso de la ampliación no (!) aumenta la cantidad de los puntos. Sólo que cada punto se visualiza cuatro veces mayor que anteriormente. La elección del modo de visualización se efectua con el segundo parámetro del comando >SCREEN<. En él significan:

Parámetro	Puntos	Ampliación	Tamaño real
0	8*8	normal	8*8
1	8*8	ampliado	16*16
2	16*16	normal	16*16
3	16*16	ampliado	32*32

La definición de un Sprite de 8*8 puntos ya la hemos tratado en principio en el capítulo sobre el modo de texto. Un Sprite se escribe en forma de matriz de Bits, en lo cual cada línea horizontal de 8 puntos representa un Byte (=8 Bit). El valor del Byte es igual al valor del número binario que se obtiene, si se escriben "unos" para los puntos posicionados y "ceros" para los no posicionados.
Así, un Sprite de 8*8 es caracterizado (como un carácter) por una serie de 8 cifras. En el caso de definiciones de carácter teníamos que POKEar estas cifras directamente en la VRAM. Para el caso de los Sprites existe un comando BASIC: >SPRITE $(...)=...<.
Quizás se extrañe Vd., por qué se trata aquí de un comando String. Pero para la elaboración interna, tanto Strings como cadenas de números son iguales, lo que quiere decir que un String se memoriza como cadena de números. Por ejemplo, "A" = CHR$(65) se memoriza internamente como Byte del valor 65.

La ventaja, al efectuar las definiciones de Sprites con Strings, es que para la manipulación de Strings existen una cantidad de comandos, utilizables todos para luego, con >SPRITE$<, poder manipular el patrón Sprite.
Observemos el siguiente Sprite:

```
1 ********     &HFF
2 *** ***      &HEE
3 ******       &HFC
4 * ***        &HB8
5 ******       &HFC
6 *** ****     &HEF
7 **   **      &HC6
8 *    * *     &H85
  12345678
```

Este Sprite se ha de definir como patrón de Sprite 1:

1a. Posibilidad:

```
10 SPRITE$(1) = CHR$(&HFF)+CHR$(&HEE)+CHR$(&HFC)+CHR$(&HB) +
   CHR$(&HFC)+CHR$(&HEF)+CHR$(&HC6)+CHR$(&H85)
```

2a. Posibilidad:

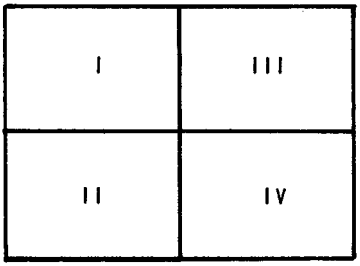
```
10 FOR I=0 TO 7: READB$:A$=A$+CHR$(VAL("&H"+B$)):NEXT
20 SPRITE$(1) = A$
30 DATA FF,EE,FC,B8,FC,EF,C6,85
```

La segunda posibilidad, que de momento parece más complicada, tiene la ventaja de que la definición de Sprite está memorizada en A$, o sea, que sigue estando a disposición y puede ser modificada posteriormente.
En Sprites de 8*8 se pueden definir hasta 256 (!!) (números 0-255) patrones de Sprites diferentes. De estos 256 se pueden visualizar 32 simultáneamente.

Tenga en cuenta, que similar a la primera posibilidad, también funciona algo como:

SPRITE$ (1) = "1AXz1,Bz"

Los Sprites 16*16 se definen en principio idénticamente. Para ello, la matriz 16*16 se divide en cuatro matrices 8*8, que se indican luego una detrás de otra. Se aplica el siguiente orden:



En total se pueden definir 64 patrones de Sprite 16*16 diferentes. Para que el diseño de Sprites no se tenga que efectuar laboriosamente sobre el papel, le presentamos un editor de Sprites.

Editor de Sprites

La función del editor de Sprites es igual a la del generador de caracteres. La diferencia principal consiste en la diferenciación entre los Sprites de 8*8 y 16*16, y en las funciones de usuario diferentes, que pueden ser activadas por medio de las teclas de función.

Primero, Vd. elige entre 8*8 o 16*16 puntos, después entre normal o ampliado.
Después, Vd. puede moverse con un joystick (teclas de cursor) a través de una matriz y posicionar o reposicionar puntos pulsando el disparador (tecla espaciadora).

Junto con la matriz de definición verá Vd. indicados los valores correspondientes al patrón en cuestión. Estos tiene que utilizarlos Vd. por columnas de arriba hacia abajo para la definición de Sprite en su programa. En ello, la columna izquierda contiene los valores para el primer y segundo cuadrante y la columna derecha contiene los valores para el tercer y cuarto cuadrante en forma hexadecimal (basado en el patrón de Sprite 16*16 de la página anterior).
Estos valores han de introducirse en forma >CHR$< (posibilidad 1 : algunas páginas hacia delante) o en líneas >DATA< (posibilidad 2: en el mismo lugar). Si Vd. no está familiarizado con números hexagesimales, lease el capítulo sobre lenguaje máquina. Aquí encontrará, bajo el título sistemas de números, las informaciones a este respecto.

El Sprite en tamaño original lo verá indicado sobre la matriz de edición.

Aquí ahora la ocupación de teclas de función:

Key 1: Norm (Modo normal)
Si Vd. ha pulsado esta tecla, puede posicionar o borrar puntos con el joystick dentro de una matriz
antes seleccionada, pulsando el disparador.

Key 2: Lion (Modo on Line)
Pulsando la segunda tecla de función conecta el modo Line on. En este modo puede dibujar líneas dentro de la matriz de Sprite pulsando el disparador. Vd. no está obligado más a colocar puntos, sino que puede dibujar líneas completas.

Key 3: Liof (Modo Line off)
Esta tecla origina la conexión del modo Line off. Este es exactamente lo contrario de "Lion", o sea, Vd. puede borrar líneas completas.

Key 4: prnt (Modo de Printer)

Al pulsar esta tecla, el Sprite dibujado por Vd. es imprimido con sus correspondientes valores (&H).

Key 5: Final

Esta tecla finaliza el programa.

```
10 DEFINT A-Z
20 SCREEN 2
30 A$="S"
40 INPUT"8x8 Sprite  (s/n) ";A$
50 IF (ASC(A$)AND&B11011111)=ASC("S") THEN M=0 ELSE M=2
60 A$="N"
70 A$="N":INPUT"ampliado (s/n)";A$
80 IF (ASC(A$)AND&B11011111)=ASC("S") THEN M=M+1
90 SCREEN 1,M:COLOR 15,4,4
100 MK=INT(M/2)
110 BA=BASE(9)
120 XA=1:YA=6:QZ=1
130 EI=1
140 ON STRIG GOSUB 530,530,530,530,530
150 STRIG(EI) ON
160 AU$=" ":EI$=CHR$(42)
170 ON KEY GOSUB 700,620,660,850,740
180 KEY 1,"norm":KEY (1) ON
190 KEY 2,"Lion":KEY (2) ON
200 KEY 3,"Liof":KEY (3) ON
210 KEY 4,"prnt":KEY (4) ON
220 KEY 5,"final:KEY (5) ON
230 CLS:PUT SPRITE 0,(24,0)
240 FOR QZ=1 TO 1+3*MK
250 FOR Y=0 TO 7
260 GOSUB 580
270 NEXT Y,QZ
280 GOSUB 700
290 X=0:Y=0:QZ=1
300 LOCATE XA-8*(QZ>2)+X,YA-8*(QZ=2 OR QZ=4)+Y,1
310 R=STICK(EI)
320 IF R=0 THEN 310
330 STRIG(EI) OFF
340 IF R=8 OR R=1 OR R=2 THEN Y=Y-1
350 IF R>3 AND R<7 THEN Y=Y+1
360 IF R>1 AND R<5 THEN X=X+1
370 IF R>5 AND R<=8 THEN X=X-1
380 IF X<8 THEN 410
```

```
390 IF M<2 THEN X=7:BEEP:GOTO 410
400 IF QZ=1 OR QZ=2 THEN X=0:QZ=QZ+2 ELSE X=7:BEEP
410 IF X>=0 THEN 440
420 IF M<2 THEN X=0:BEEP:GOTO 440
430 IF QZ=3 OR QZ=4 THEN X=7:QZ=QZ-2 ELSE X=0:BEEP
440 IF Y<8 THEN 470
450 IF M<2 THEN Y=7:BEEP:GOTO 470
460 IF QZ=1 OR QZ=3 THEN Y=0:QZ=QZ+1 ELSE Y=7:BEEP
470 IF Y>=0 THEN 500
480 IF M<2 THEN Y=0:BEEP:GOTO 500
490 IF QZ=2 OR QZ=4 THEN Y=7:QZ=QZ-1 ELSE Y=0:BEEP
500 IF LF=0 THEN STRIG(EI) ON:GOTO 300
510 IF LF=-1 THEN BY=VPEEK(BA+(QZ-1)*8+Y) OR 2^(7-X):GOSUB 5
60:GOTO 300
520 BY=VPEEK(BA+(QZ-1)*8+Y) AND (NOT 2^(7-X)):GOSUB 560:GOTO
 300
530 REM Strig
540 BY=VPEEK(BA+(QZ-1)*8+Y) XOR2^(7-X)
550 LOCATE ,,0
560 IF (BY AND 2^(7-X))=0 THEN PRINT AU$; ELSE PRINT EI$;
570 VPOKE BA+(QZ-1)*8+Y,BY
580 LOCATE XA+8*(1+MK)+3-3*(QZ>2),YA+Y-8*(QZ=2 OR QZ=4)
590 PRINTRIGHT$("00"+HEX$(BY),2);
600 LOCATE XA-8*(QZ>2)+X,YA-8*(QZ=2 OR QZ=4)+Y,1
610 RETURN
620 REM Lion: posicionar línea
630 LF=-1
640 A$="dibujar linea          ":GOSUB 800
650 RETURN
660 REM liof: borrar línea
670 LF=1
680 A$="borrar línea           ":GOSUB 800
690 RETURN
700 REM funcionamiento normal
710 LF=0
720 A$="funcionamiento normal ":GOSUB 800
730 RETURN
```

```
740 REM final
750 LOCATE ,,0
760 DEFUSR1=&H139D
770 X=USR1 (1)
780 CLS
790 END
800 XP=POS(0):YP=CSRLIN
810 LOCATE 5,1,0
820 PRINTA$;
830 LOCATE XP,YP,1
840 RETURN
850 REM Print
860 LOCATE ,,0:TB=BASE(5)
870 FOR Z=YA TO YA-1+8*(1+MK)
880 FOR S=0 TO 31
890 LPRINT CHR$(VPEEK(TB+32*Z+S));
900 NEXT S
910 LPRINT:NEXT Z
920 LOCATE ,,1:RETURN
```

Descripción del programa:

La construcción del editor de Sprites es igual a la del editor de caracteres. La diferencia esencial consiste en la diferenciación de 8*8 y 16*16 para el tamaño de Sprite y en las funciones adicionales, que permiten un manejo confortable.

A base de las variables adicionales, se ven claramente las diferencias más importantes.
M contiene el modo de visualización de Sprite (0 hasta 3) y MK es 0 con 8*8 y 1 con 16*16 puntos para la matriz de Sprite.
La variable QZ es importante. Contiene el número del subbloque 8*8 en el tamaño de Sprite 16*16.
A base de QZ, X e Y se puede calcular siempre la posición real de pantalla.

En parte resultan estructuras relativamente complicadas, como >...+8*(QZ>2)...<. Si Vd. desconoce el uso de expresiones aritméticas, pruebe lo siguiente:

PRINT 1=3 o PRINT 4>2.

Una afirmación cierta origina el valor -1, una afirmación no cierta el valor 0.

Basándose en este truco funciona la diferenciación de la representación de Sprites 8*8 y 16*16. ¡Introdúzcase en esta técnica de programación! Puede ser muy útil, y abreviar programas enormemente.

Para la diferenciación de los diferentes modos de funcionamiento se utiliza la variable LF. 0 significa normal, 1 borrar línea y -1 posicionar línea.
En todos los modos de funcionamiento, excepto "normal", se desconecta el Interrupt STRIG, ya que no se necesita.

¿Cómo se mueve un Sprite?

La posición de un Sprite se determina por las coordenadas de la esquina superior izquierda. Estas coordenadas se fijan por medio de >PUT SPRITE<. El formato completo del comando es:

PUT SPRITE plano, (coordenada x, coordenada y), color, número de patrón.

El número de patrón es el número asignado con
>SPRITE$ (número de patrón)=...<.

Ahora es posible la formación de imágenes completas con Sprites. Un programa de demostración para esto sería demasiado amplio, a causa de las muchas definiciones de Sprites. Tenga en cuenta lo siguiente en sus programas:

CONSEJOS:

- Elementos de imagen, mayores de 16*16 puntos, se pueden producir por medio de varios Sprites visualizados contiguamente. Naturalmente, al mover un Sprite gigante, hay que ejecutar varios comandos >PUT SPRITES< inmediatamente.

- Sprites de varios colores se representan generalmente por medio de la superposición de varios colores diferentes. También aquí hay que ejecutar tres comandos idénticos >PUT SPRITE<, por ejemplo, para el movimiento de un Sprite tricolor (compuesto por 3 Sprites superpuestos exactamente). Las coordenadas sólo hay que indicarlas para el primer comando, en caso de que los tres comandos estén situados uno detrás del otro. Si, además, concuerdan el número de patrón y de plano, se puede escribir >PUT SPRITE n<, donde n es el número de patrón y de plano.

En una línea horizontal se pueden visualizar como máximo 4 Sprites diferentes. Con 5 o más se visualizan los 4 de los planos inferiores.

- Una aplicación interesante de los Sprites es la ''simulación de cine''. Con 64 Sprites 16*16 diferentes se pueden representar, mediante una programación hábil, muchos movimientos.

Para ello hemos elaborado un programa, que origina un "elefante" andando. En él no sólo se mueve un patrón sino que, por medio de la unión de diferentes patrones, se transmite la sensación de movimiento (principio de las películas animadas).

Programa: Manada de elefantes

Definiciones de Sprite (16*16)

Sprite 1:

1234567812345678=8+8=16

	&H00	1	0000000000000000	&H00	
	&H0C	2	0000110000000100	&H04	
	&H12	3	0001001000000010	&H02	
A$(1)	&H13	4	0001001111110001	&HF1	A$(3)
	&H1A	5	0001101000001101	&H0D	
	&H3E	6	0011111000000011	&H03	
	&H40	7	0100000010000001	&H81	
	&H89	8	1000100110000001	&H81	
	&H56	1	0101011010000001	&H81	
	&H90	2	1001000010000001	&H81	
	&H57	3	0101011100001001	&H09	
A$(2)	&H94	4	1001010001001011	&H4B	A$(4)
	&H52	5	0101001010111010	&HBA	
	&HF2	6	1111001010001010	&H8A	
	&H02	7	0000001010001010	&H8A	
	&H07	8	0000011110011110	&H9E	

Sprite 2:

1234567812345678=8+8=16

	&H00	1	0000000000000000	&H00	
	&H0C	2	0000110000000100	&H04	
	&H12	3	0001001000000010	&H02	
B$(1)	&H13	4	0001001111110001	&HF1	B$(3)
	&H1A	5	0001101000001101	&H0D	
	&H3E	6	0011111000000011	&H03	
	&H40	7	0100000010000001	&H81	
	&H89	8	1000100110000001	&H81	
	&H56	1	0101011010000001	&H81	
	&H90	2	1001000010000001	&H81	
	&H57	3	0101011100001001	&H09	
B$(2)	&H94	4	1001010001001010	&H4A	B$(4)
	&H52	5	0101001010111010	&HBA	
	&HF2	6	1111001010001010	&H8A	
	&H05	7	0000010100010100	&H14	
	&H0F	8	0000111100111100	&H3C	

Sprite 3:

1234567812345678=8+8=16

	&H00	1	0000000000000100	&H04	
	&H0C	2	0000110000000010	&H02	
	&H12	3	0001001011100010	&HE2	
C$(1)	&H17	4	0001011100010001	&H11	C$(3)
	&H1A	5	0001101000001101	&H0D	
	&H1E	6	0001111000000011	&H03	
	&H20	7	0010000000000001	&H01	
	&H49	8	0100100110000001	&H81	
	&H56	1	0101011010000001	&H81	
	&H91	2	1001000100000001	&H01	
	&H96	3	1001011000001010	&H0A	
C$(2)	&HA4	4	1010010001001010	&H4A	C$(4)
	&HA2	5	1010001010111010	&HBA	
	&HF2	6	1111001010001001	&H89	
	&H01	7	0000000101000101	&H45	
	&H03	8	0000001111001111	&HCF	

Programa:

```
10 DEFINTA-Z
20 SCREEN 2,2
30 FOR I=1 TO 4:FOR J=0 TO 7:READ A$:A=VAL("&H"+A$)
40 A$(I)=A$(I)+CHR$(A):NEXT J,I
50 DATA 00,0C,12,13,1A,3E,40,89
60 DATA 56,90,57,94,52,F2,05,0F
70 DATA 00,04,02,F1,0D,03,81,81
80 DATA 81,81,09,4A,BA,8A,14,3C
90 SPRITE$(1)=A$(1)+A$(2)+A$(3)+A$(4)
100 B$(1)=A$(1)
110 B$(2)=LEFT$(A$(2),6)+CHR$(&H2)+CHR$(&H7)
120 B$(3)=A$(3)
130 B$(4)=A$(4):MID$(B$(4),4,1)=CHR$(&H4B):B$(4)=LEFT$(B$(4)
,6)+CHR$(&H8A)+CHR$(&H9E)
140 SPRITE$(2)=B$(1)+B$(2)+B$(3)+B$(4)
150 SPRITE$(0)=B$(1)+B$(2)+B$(3)+B$(4)
160 FOR J=0 TO 31:READ A$:A=VAL("&H"+A$)
170 C$=C$+CHR$(A):NEXT J
180 DATA 00,0C,12,17,1A,1E,20,49
190 DATA 56,91,96,A4,A2,F2,01,03
200 DATA 04,02,E2,11,0D,03,01,81
210 DATA 81,01,0A,4A,BA,89,45,CF
220 SPRITE$(3)=C$
230 '
240 'MOVIMIENTO DEL SPRITE
250 '
260 X=256:Y=20
270 FOR N=0 TO 3:X=X-1
280 FOR Z=1 TO 15 STEP 2
290 PUT SPRITE Z,(((X+20*Z)MOD287)-31,Y+(ZMOD 4)*30-2*(N=2))
,Z,N
300 NEXT Z,N
310 IF X>-31 THEN 270
320 GOTO 260
```

Descripción del programa

Línea 30:

Bucles para leer las líneas DATA en el orden A$(1) hasta A$(4).
Leer líneas DATA con entrada "&H" automática.

Línea 40:

Unión de los valores indicados en las líneas DATA de A$(1) hasta A$(4).

Línea 50 hasta 80:

Líneas DATA para Sprite 1

Línea 90:

Construir SPRITE$(1)

Línea 100 hasta 130:

Construir Sprite 2 y 0 de partes del Sprite 1.

Línea 140:

Definir Sprite 2 del String leído.

Línea 150:

Definir Sprite 0

Línea 160:

Bucle para leer Sprite 3.
Leer líneas DATA con introducción "&H" automática.

Línea 170:

Unión de los valores indicados en las líneas DATA de de C$(1) hasta C$(4).

Línea 180 hasta 210:

Líneas DATA para Sprite 3

Línea 220:

Definir Sprite 3 con SPRITE$ (3).

Línea 260:

Determinar coordenadas X e Y para la posición

Línea 270:

Determinar el orden de los diferentes Sprites en un bucle X determina el sentido de movimiento sobre la pantalla.

Línea 280:

Editar 8 Sprites sobre la pantalla con diferentes colores

Línea 290:

Activa los Sprites. Determina el sentido de movimiento, cantidad y orden de los Sprites con ayuda de los bucles en las líneas 270 y 280

Línea 310:

Cuando el contador de sentido de movimiento X es mayor que 31, cuando se ha sobrepasado el margen izquierdo de la pantalla, se sigue en la línea 270.

Línea 320:

De otra forma, continuar en la línea 260.

Sprites Interno

A continuación vamos a tratar brevemente la elaboración interna de los Sprites en la VRAM y los registros VDP relacionados con ello.

Así como hay una tabla de caracteres o de patrones, existe también una tabla de patrones de Sprites. Tiene una extensión de 2048 (=2 KByte) y puede ser desplazada a pasos de 2K. Existe la posibilidad de interconectar entre varias tablas de Sprites por medio de un Interrupt (ver capítulo Modo gráfico I).
La tabla de patrones de Sprites está dividida en 256 bloques de 8 Bytes cada una (= patrón de Sprite 8*8). Con Sprites 8*8, el número de patrón es igual al número del bloque de 8 puesto en la tabla y que contiene la definición. Con Sprite 16*16, se ha de multiplicar el número del patrón por 4, para obtener el número de bloque de la definición.

En principio también se pueden definir Sprites por medio de >VPOKE<. La dirección inicial de la tabla de patrones de Sprites es determinada por el registro VDP 6. Este registro contiene los 3 MSB's de la dirección de la tabla de patrones de Sprites.

La dirección inicial también puede determinarse por medio de >BASE(PEEKC(&HFCAF)*5+4)< para todos los modos. Al conectar el ordenador, la tabla de patrones de Sprites ocupa en todos los modos posibles la dirección &H3800 hasta &H3FFF. Con ello, el valor de registro VDP es 6. La segunda tabla es la así denominada tabla de Atributos de Sprite (SAT). Contiene informaciones que son fijadas por >PUT SPRITE<. Como existen 32 planos, la SAT esta dividida en 32 bloques. Cada bloque tiene una longitud de 4 Bytes. El número de bloque (0-31) corresponde con el número de plano, o sea, inscripciones en el bloque 5 corresponden al plano 5.

Por lo tanto, la SAT es desplazable a pasos de 128. El registro VDP 5 contiene los 7 Bits superiores de la dirección de 14 Bits de la tabla. Con >BASE< se puede determinar las dirección inicial de la SAT en todos los modos con >BASE(PEEK(&HFCAF)*5+3)<.

Al conectar resulta el valor &H1B00. La tabla ocupa, por lo tanto, el sector de &H1B00 hasta &H1BFF, el registro VDP 5 contiene &H1B00/2^(14-7)=&H36.

Las Inscripciones dentro de un bloque de 4 Bytes tienen el siguiente significado:

Byte 0 - coordenadas Y
Byte 1 - coordenadas X
Byte 2 - número de patrón
Byte 3 - color, entre otras cosas

Las coordenadas se refieren siempre a la esquina superior izquierda de los Sprites.

También son posibles coordenadas Y negativas (representación con el complemento 2: 256-valor absoluto del número), lo cual significa que la parte superior del Sprite no es visualizada. Por decirlo así, ha desaparecido detrás del marco lateral. De esta forma es posible dejar entrar lentamente los Sprites en la pantalla. Con >PUT SPRITE< se pueden indicar coordenadas Y entre -31 y +192. Con la coordenada X no es tan fácil, ya que todos los 256 posibles valores de Byte se pueden utilizar como coordenada normal. Para posibilitar esto, se utiliza el Bit 7 del cuarto Byte de un bloque. Este se llama el EC (Early Clock) Bit. Si es 0, se encuentra en el estado normal. Si tiene el valor 1, entonces todas las posiciones X de los Sprites están desplazadas 32 puntos hacia la izquierda. Entonces, por medio de las coordenadas x entre 0 y 32, se puede conseguir que el Sprite entre en la pantalla por la izquierda.

Con el comando >PUT SPRITE< se pueden introducir directamente valores entre -32 y +255.

El tercer Byte contiene un indicador hacia la tabla de patrones de Sprites. Con Sprites 8*8 es igual al número de Sprite del BASIC.
Los Bits 0-3 del cuarto Byte, finalmente, contienen el código para el color del Sprite. Para determinar el modo de indicación de Sprite, se han previsto dos Bits del registro VDP 1:

Bit 0 Mag - Magnification - ampliación
Bit 1 Sizw - Size - tamaño

	0	1
MAG	normal	ampliado
SIZE	8*8	16*16

Si surgen más de 4 Sprites en una línea horizontal, entonces en el registro VDP 8, el cual se denomina generalmente el registro de estado (read only), se posiciona el Bit 6. En los Bits 0 hasta 4 se memoriza el número del quinto Sprite que se encuentre en el primer plano no visualizado.

Finalmente existe el así denominado Coincidence Flag, el cual indica que se solapan como mínimo dos Sprites. El C-Flag es el Bit 5 del registro de estado. A través de este Bit es posible la función del Interrupt Basic "ON SPRITE". Con él se pueden elaborar inmediatamente collsiones de Sprites.

Capitulo III: E/S

3.1 Reflexiones generales sobre E/S

¿Qué sería un ordenador sin periferia, o sea, sin pantalla, sin teclado, cassette, impresora, etc.?

Las posibilidades de un ordenador se pueden aprovechar cuando existe una periferia adecuada, sobre todo un teclado y una pantalla. Para ello, el ordenador posee diferentes (así denominadas) interfases. En el MSX son, entre otras:

Clavija de monitor
Monitor RGB
Interfase para impresora
Port para joystick
Cartridge Slots

Primero vamos a presentar brevemente un aparato periférico MSX:

El cassette:

Sobre el manejo de un magnetófono para cassette no hay que perder muchas palabras.
Los comandos estándar >CSAVE< y >CLOAD< sirven para memorizar y cargar programas.
Los comandos >BLOAD< y >BSAVE< se utilizan para memorizar contenidos de memoria. Esta aplicación surge casi siempre al utilizar programas de máquina.

Interesantes son los comandos >SAVE< y >LOAD<.

Son comandos generales, que sirven para la transmisión de datos con ayuda de ficheros.
La cantidad de los ficheros a utilizar simultáneamente se fija por medio de >MAXFILES<. Con este comando se reservan sectores en la memoria para la administración de los ficheros.

Lo especial en >LOAD< y >SAVE< es, que no sólo son aplicables para el cassette. Teóricamente se puede utilizar cualquier aparato periférico como dirección fuente o de meta para los datos a recibir o a transmitir. Para ello es posible indicar el aparato en el argumento de estos comandos. Todos los comandos de tratamiento de ficheros (>OPEN, PRINT≠, CLOSE<) pueden utilizarse en la forma arriba descrita.
Por medio de la definición de otra meta, la corriente total de datos es conducida hacia el aparato indicado. De esta forma se puede transmitir hacia los siguientes aparatos:

CAS - cassette
CRT - pantalla
GRP - pantalla gráfica
LPT - impresora

La técnica de la edición de datos a través de >PRINT≠< ya ha sido tratada varias veces en los programas gráficos del capítulo anterior.

Aunque la periferia es un tema muy importante, frecuentemente se descuida. La programación E/S depende bastante del Hardware y por ello es difícilmente accesible. Muchas ideas sólo se pueden realizar con ayuda del lenguaje máquina. A continuación trataremos los aspectos básicos de la programación E/S en ordenadores MSX.

Interfases de pantalla

La señal, que puede ser obtenida a través de la clavija de antena, es compuesta, por un modulador incluído, a base de las informaciones procedentes del VDP. La señal de sonido del PSG (Programable Sound Generator) también se ha elaborado aquí.

La salida de monitor utiliza las mismas informaciones básicas, sólo que la señal no es modulada, es decir, que solamente puede ser recibida por un monitor y no por un televisor. La salida RGB suministra una señal para monitores de alta resolución de colores formada por las mismas

Informaciones básicas. Es interesante saber que la salida RGB contiene una señal en estereo. Los canales derecho e izquierdo se pueden activar independientemente. Como aquí no se puede programar nada con medios sencillos, vamos a pasar al segundo aparato periférico de importancia.

Teclado:

Desde el punto de vista electrónico, el teclado es una red rectangular, formada por hilos cruzados. En cada uno se encuentra una tecla. En este sentido se habla de una matriz de teclado. La consulta al teclado se realiza 50 veces por segundo mediante el interrupt interno. Para ello, se manda una corriente a través del conductor horizontal (Bit=1). En los conductores verticales se comprueba si existe una corriente. Si uno de estos conductores lleva corriente (Bit=1), entonces la tecla que se encuentre sobre el nudo de éste y el conductor vertical correspondiente está pulsada. Esto significa, que el circuito eléctrico se ha cerrado. De esta forma se comprueban constamente las teclas que están pulsadas.

¿Cómo se programa este proceso?

Primero se necesita un comando de edición (OUT) que pase una corriente a través de un conductor horizontal, e inmediatamente después un comando de entrada (>INP<) que confirme la tecla que esté pulsada. El IC, que ejecuta la consulta al teclado, es el PPI (Programable Periphal Interface). La norma MSX exige el PPI 8255 de la empresa Intel.

El Port de salida, que manda la corriente hacia el teclado, es el Port C (el PPI). Para diferenciar las diferentes salidas se ha asignado un número a cada Port E/S.

El Port C del PPI tiene el número &HAA.
Los cuatro Bits inferiores del Port C del PPI determinan el conductor horizontal a través del cual se ha de mandar una corriente. Los cuatro Bits superiores no se han de modificar. Entonces el comienzo del programa sería:

```
10 CLS
20 Z=INP(&HAA):REM leer Port C
30 Z=Z1 AND &HF0 : REM conservar los 4 Bits superiores y
   borrar los 4 inferiores.
40 LOCATE 0,0
```

Ahora sigue la consulta de los 9 posibles conductores:

```
50 FOR I=0 TO 8
60 OUT &HAA,Z+I:REM consultar el conductor I
```

Ahora se lee el valor del conductor vertical a través de >INP<. Los conductores procedentes del teclado forman el Port B del PPI. Este tiene el número &HA9.

```
70 PRINT RIGHT$ ("0000000"+BIN$(INP(&HA9)),8)
80 NEXT I
90 GOTO 20
```

Deje transcurrir el programa y obtendrá la matriz del teclado sobre la pantalla. Un 0 indica una tecla pulsada. Inserte ahora las líneas 45 y 46.

```
45 A$=INKEY$:IF A$="" THEN PRINT " " ELSE PRINT A$
46 LOCATE 0,1
```

y puede llenar la matriz de teclado. Entonces surge la siguiente imagen:

7	6	5	4	3	2	1	0
8	9	-	=	U			;
'		,	.	/		A	B
C	D	E	I	G	H	I	J
K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z
SHIFT	CTRL	GRAPH	CAP	CODE	F1	F2	F3
F4	F5	ESC	TAB	STOP	BS	SELECT	RETURN
SPACE	HOME	INS	DEL				

Tenga en cuenta que la tecla consultada por >INKEY$< frecuentemente sólo coincide con la matriz pulsando una sola tecla.
Esto significa que los caracteres especiales con >SHIFT,CTRL,GRAPH, etc.< son reconocidos e interpretados especialmente por la rutina de consulta. Por lo contrario, si queremos programar nuestra propia consulta al teclado, tenemos que basarnos en los datos leídos con el anterior programa. Como se ha mencionado, esta pregunta se puede ejecutar a través de un Interrupt interno, por lo cual podemos aprovechar estos datos, que están memorizados en el así denominado buffer NEWKEY. Entonces nuestro programa se simplifica:

```
10 CLS
20 LOCATE 0,0
30 FOR I=&HFBE5 TO &HFBED
40 PRINT RIGHT$("0000000"+BIN$(PEEK(I)),8)
50 NEXT I:GOTO 20
```

¿Cómo utilizamos ahora esta matriz de Bits para conseguir todas las teclas pulsadas?
El programa que ejecute tal consulta al teclado será aprovechado en el siguiente capítulo para programar un sonido polifónico (quiere decir a varias voces).

Dentro de la ROM están todas las tablas necesarias para la decodificación del teclado. Observemos la tabla de la ocupación del teclado sin >SHIFT< ni otras teclas especiales. Se encuentra en la dirección &HODA5. A partir de esta dirección volvemos a encontrar los códigos ASCII de los caracteres, que son producidos pulsando simplemente una tecla:

```
FOR I=0 TO 5: FOR J=7 TO 0 STEP-1:PRINT CHR$(PEEK(&HDA5+I*8+
J));:NEXT:PRINT:NEXT
```

Estas tablas se utilizan internamente para determinar el código ASCII perteneciente a la tecla pulsada en ese momento. Entonces, los códigos ASCII se depositan en el así denominado buffer KEYBUF y están allí a disposición.
La siguiente subrutina lee el Keybuffer y lo prepara para una nueva introducción inmediata:

```
10 KB=VAL("&HFBF0"):REM dirección inicial KEYBUF
20 GOTO 50: KEYBUF inicialización
30 A=PEEK(&HF3F8)-&HF0:IF A=0 THEN 30: REM cantidad de
teclas pulsadas
40 FOR I=KB TO KB+A-1:PRINT CHR$(PEEK(I);:NEXT:REM editar el
contenido del buffer
50 POKE &HF3F8,&HF0:POKE &HF3F9,&HFB: REM init buffer
60 POKE &HF3FA,&HF0:POKE &HF3FB,&HFB
70 POKE &HF3F7,1: REM desconectar tiempo de espera para
repetición
80 PRINT:GOTO 30
```

Es interesante el comando >POKE< en la línea 70.

Si eliminamos esta línea, entonces no se registra ''mantener una tecla pulsada''. Normalmente, hasta que la función continua de una tecla sea vigente, se tiene en cuenta un cierto tiempo de espera. Este contador de tiempo de espera (REPCNT-Repeat Counter) es el valor en la dirección &HF3F7. Si siempre es posicionado de nuevo al valor 1, es posible la repetición de la tecla con un tiempo de espera mínimo.
Hasta aquí el aparato E/S teclado.

Más arriba no hemos tenido en cuenta los cuatro Bits superiores del Port C. Observemos primero el Bit 6.

El Bit 6 del Port C del PPI indica el estado del diodo CAP. El significado del Bit 6 es el siguiente:

Bit 6=1 : El diodo CAP no está encendido
Bit 6=0 : El diodo CAP está encendido

Pruebe Vd:

Desconectar el diodo CAP y entonces:

```
PRINT HEX$ (INP(&HAA))
```

Desconectar el diodo CAP y entonces:

```
PRINT HEX$ (INP(&HAA))
```

El desconectado del diodo CAP se consigue con

```
OUT &HAA,INP(&HAA) OR 216
```

o más fácil con

```
OUT &HAB,13
```

La segunda versión representa un comando de gobierno sobre el Port &HAB del PPI. El comando es:
"Posiciona a 1 el Bit 6 del Port C."
El desconectado del diodo CAP se consigue con:

```
OUT &HAA,INP(&HAA) AND NOT 2^6
```

o a través del comando de gobierno

```
OUT &HAB,12
```

el cual significa "Posiciona a 0 el Bit 6 del Port C".

Tenga en cuenta, que sólo es afectado el diodo luminoso. El cambio de letras mayúsculas a minúsculas no es modificado. Si este cambio se ha de realizar desde el programa, hay que proceder de la siguiente manera:

1) Simular ''Pulsación de tecla CAP'' por el programa.

```
DEF USR1=&HF36:X=USRS1(0)
```

Este llamamiento al programa de máquina cambia el estado del CAP en cada llamamiento, o sea, corresponde exactamente a pulsar la tecla CAP.

2) Comprobar si está conectada la escritura mayúscula o minúscula:

```
F=PEEK (&HFCAB): IF F=0 THEN PRINT "MINUSCULA" ELSE PRINT "MAYUSCULA".
```

3) Conectado normal de la escritura mayúscula y del diodo CAP:

```
POKE &HFCAB,0:OUT &HAB,13
```

En relación al diodo CAP también es posible, por ejemplo, un tiempo de espera por medio del comando >WAIT<. El comando >WAIT< ejecuta prácticamente el comando >INP< y compara el valor obtenido con el indicado. Se ejecuta un >XOR< con el segundo valor y después un >AND< con el primer valor indicado. Si el resultado es 0, se sigue esperando, de otra forma sigue el programa.
Acordemos ahora, que un diodo CAP encendido indica el estado de espera. A través de

```
WAIT &HAA,2^6
```

se interrumpe el programa hasta que se pulsa la tecla CAP, lo cual apaga el diodo CAP.

Por lo contrario, si se quiere esperar hasta que el diodo sea apagado, hay que invertir el correspondiente Bit por medio de >XOR<.

```
WAIT &HAA,2^6,2^6
```

Hasta aquí la tecla CAP.

Obeservemos ahora la función del Bit 7 del Port C.

Seguramente que Vd. conoce el así llamado sonido de pulsación de tecla "clic". Con el comando >SCREEN< o con

```
POKE &HF3DB,0
```

se puede eliminar este sonido.

Aunque este clic es un "tono", no es producido excepcionalmente por el PSG (Sound Chip). Además existe la posibilidad de situar este clic en la salida Audio por medio del PPI. Esto se efectua simplemente por el posicionado y reposicionado de un conductor, que está unido con esta salida. El cambio de un estado al otro produce un movimiento de la membrana del altavoz, o sea, el clic es audible.
Es interesante la posibilidad de producir no solo este clic, sino también oscilaciones de la membrana a través de cambios rápidos del estado del conductor, de forma que se oiga un tono.

Primeramente desconecte el ruido de pulsación de tecla con >SCREEN,,0< o con >POKE &HF3DB,0<. Después introduzca lo siguiente:

```
OUT &HAB,15
```

Aunque se ha desconectado el ruido, éste es audible al pulsar la tecla Return.

Este ruido nos dice que la membrana del altavoz está "tensada". Al repetir la ejecución del mismo comando no se oye nada, ya que no se ha modificado el estado de la membrana. Con

```
OUT &HAB,14
```

se vuelve a activar la tensión del altavoz a 0. Como la membrana estaba tensada a causa del comando >OUT &HAB,15<, ahora también es audible un tono.
El estado actual del así denominado conductor Software Sound se obtiene por medio de un comando >INP<. El estado del conductor corresponde al Bit 7 del Port C del PPI. Este Port tiene la dirección &HAA. Para observar exclusivamente el Bit 7 utilizamos los siguientes comandos lógicos:

```
PRINT INP (&HAA) AND 2^7
```

Si obtiene 0 después de la introducción de la anterior línea, entonces el conductor estaba en 0. Si se obtiene 128, entonces el estado era 1.

Como un tono se obtiene por medio de oscilaciones, por ejemplo de una cuerda o una membrana de altavoz, un cambio rápido de estado del conductor Software Sound origina un tono.

```
10 FOR I=0 TO 100
20 OUT &HAB,15
30 OUT &HAB,14
40 NEXT
```

Insertando una línea 25 en la cual se encuentre, por ejemplo, un búcle de espera, un >REM<, o un comando >PRINT<, se puede reducir la velocidad de sucesión de los "clics". De esta forma se reduce la frecuencia con la cual oscila la membrana del altavoz. Menos oscilaciones, o sea, menor frecuencia significa un tono más bajo. Muchas oscilaciones o mayor frecuencia representan un tono más alto.

Los dos últimos Bits del Port C son importantes en relación con el cassette:

Bit 4 gobierna el motor del cassette:

```
OUT &HAB,8 : conecta motor
OUT &HAB,9 : desconecta el motor
```

Finalmente, el Bit 5 se utiliza para memorizar informaciones en el cassette. >OUT &HAB,11< conecta la tensión de magnetización y >OUT &HAB,10< la desconecta.

3.2 División de la memoria

Uno de los deberes más importantes del PPI en el sistema MSX es la administración de la división de memoria.
Como Vd. sabe, existen diferentes versiones dentro de la norma MSX. En el mercado se pueden adquirir diferentes capacidades RAM y/o ROM's adicionales con programas de aplicación listos, o ampliaciones de BASIC para fines especiales. Además existe la posibilidad general de ampliar la capacidad ROM o RAM del ordenador por medio de módulos enchufables.
Para normalizar todas estas posibilidades y hacerlas compatibles, se dispuso una asignación variable de los sectores de memoria. La programación de las diferentes posibilidades se efectua a través del Port A del PPI.
El sector de direcciones de la CPU cubre las direcciones desde 0 hasta &HFFFF. Estos 64 K están divididos en cuatro "Pages" (páginas) de 16 K cada una. A cada una de estas cuatro páginas se le puede asignar un sector de memoria, o sea, por ejemplo ROM para las direcciones 0 hasta &H4000, cartucho de discos &H4000-&H8000, etc.. Al asignar una memoria a una página existen cuatro posibilidades. Estas se denominan los cuatro Slots de un ordenador (por: Cartridge Slot = Clavija de módulo enchufable).
Esto significa:

Slot 0 comprende en todo caso la ROM BASIC incorporada (como mínimo 32 K).

SI existen más ROM's fijas, éstas están asignadas casi siempre al Slot 0. SI no, en el Slot 0 superior se encuentra el sector RAM para el usuario.

Slot 1 es la memoria de un posible módulo enchufado.

Slot 2 es un sector RAM, que es utilizado como mínimo en el sector inferior hasta &H8000 por todas las versiones de 64 K.

Slot 3 es similar al Slot 1, un módulo de ampliación de memoria. Slot 3 se utiliza frecuentemente para el uso del cassette.

A cada página de 16 K del sector de direcciones de la CPU se puede asignar un Slot independiente de las otras páginas.

Dirección	Slot 0	Slot 1	Slot 2	Slot 3
0-&H3FFF	Sistema oper. ROM	lugar de enchufe 1	(RAM)	lugar de enchufe 2
&H4000-&H7FFF	BASIC ROM		(RAM)	
&H8000-&HBFFF	Progr. personal ROM (Sony)		RAM	
&HC000-&HFFFF	(RAM)		RAM	

¿Qué podemos hacer con estas informaciones?

Primeramente es posible determinar la configuración actual de memoria de su ordenador.
Para ello introduzca:

```
PRINT BIN$(INP(&HA8))
```

Divida el número binario obtenido en 2 grupos de 2 Bits comenzando por la derecha. Determine el valor de cada grupo (el valor está entre 0 y 3). Por ejemplo con el Sony Hit Bit:

```
10 10 00 00
 2  2  0  0
```

Esto significa que se ha seleccionado RAM en las dos páginas superiores del Slot 2, o sea en el sector &HFFFF-&H0000, y ROM en las dos páginas inferiores del Slot 0 (Slot de sistema). Por lo tanto, el Port A da constantemente informaciones sobre la configuración de la memoria. Es interesante la posibilidad de manipular la configuración de memoria desde el BASIC a través del comando >OUT< al Port A. Para esto son necesarias lógicamente las versiones de 32 K, y mejor 64 K.

Como una desconexión de la ROM desde el BASIC tiene como consecuencia un despido inmediato del ordenador (por ejemplo, >OUT &HAB,&HCC<), vamos a dejar libre un sector de experimentación. Desplazamos el comienzo del BASIC hacia &HC000 (casi versión 16K) y así dejamos libre la página 3 (&H8000-&HC000) para nuestros experimentos. El desplazamiento del sector BASIC se realiza por medio de la modificación de la dirección &HF677, la cual contiene el inicio BASIC.

```
POKE &HC000,0:POKE &HF677,&HC0:NEW
```

Esta modificación no se nota inicialmente, sólo que ahora no se pueden cargar programas muy largos. Para observar los contenidos de la memoria sirve un minimonitor. Naturalmente también se puede utilizar el monitor del capítulo 5.

```
10 FOR I=&H8000 TO &HB100 STEP 8
20 PRINT RIGHT$("000"+HEX$(I),4);" ";
30 FOR J=0 TO 7
40 BY=PEEK (I+J)
50 PRINT RIGHT$ ("0)+HEX$(BY),2);" ";
60 BY = BY AND 127
70 IF BY<320R BY>126 THEN BY = 46
```

```
 80 A$=A$+CHR$(BY)
 90 NEXT J
100 PRINT A$;
110 A$="":NEXT I
```

Ahora podemos asignar otro Slot al sector de $H8000 hasta $HC000 (página 3).

```
OUT &HA8, (INP(&HA8) AND &HCF) OR (3*16)
```

En ello, el 3 representa el número de Slot deseado.
Como ya se ha mencionado, por ejemplo, en el Sony Hit Bit el programa incluido para el "Personal Data Bank" se encuentra en el Slot 0.
La ocupación de memoria para otros ordenadores la puede Vd. tomar de los manuales correspondientes.
Por lo tanto, cambie el Slot 0 por la página 3. Cambie entonces las direcciones en la línea 10 por &H810 y &H8470. Obtendrá como contenido de memoria las anotaciones del Personal Data Bank. También es posible la lectura de la ampliación de discos de forma similar.

Ahora algo que sólo es posible con las versiones de 64 K RAM. Quizás se ha preguntado Vd. el por qué se ha comprado un ordenador de 64 K si sólo quedan 28 K (indicación al conectar) para programas BASIC.

¿Dónde están los otros KBytes?

Estos son superpuestos costantemente por el sistema Slot 0, o sea, por la ROM. Esto significa, que estos 32 K RAM, que ocupan las direcciones &H0 hasta &H8000 en el Slot 2, no son accesibles desde el BASIC. A pesar de todo, su decisión a favor de 64 K fue correcta. El funcionamiento con discos sólo es posible con la versión de 64 K RAM. Además, con lenguaje máquina se puede utilizar el sector total.
Finalmente existe la posibilidad de copiar los 32 K ROM, que no son modificables, en la RAM situada por debajo. Entonces se cambia a Slot 2, o sea, a la RAM, y el funcionamiento del ordenador sigue normal.

La ventaja decisiva en todo esto es, que todo el BASIC y el sistema operativo puede ser modificado a gusto por medio de comandos >POKE<, ya que ahora se encuentra en la RAM. De esta forma podemos elaborar nuestro propio BASIC, o modificar el existente.

Primeramente el programa, el cual copia el contenido ROM del Slot 2 hacia la RAM del Slot 2.

```
F000 210080      10          LD    HL,&H8000
F003 16A0        20          LD    D,&HA0
F005 1EAA        30          LD    E,&HAA
F007 0EA8        40          LD    C,&HA8
F009 F3          50          DI
F00A 2B          60 NEXT     DEC   HL
F00B ED51        70          OUT   (C),D
F00D 7E          80          LD    A,(HL)
F00E ED59        90          OUT   (C),E
F010 77         100          LD    (HL),A
F011 7C         110          LD    A,H
F012 B5         120          OR    L
F013 20F5       130          JR    NZ,NEXT
F015 C9         140          RET
```

```
Programa : ramrom
Comienzo : &HF000  final : &HF015
Longitud : &H16 Bytes
Errores : 0
Tabla de variables :
NEXT F00A
```

El cargador BASIC:

```
10 CLEAR 200,&HEFFF
20 FOR I=&H7000 TO &HF015
30 READ A$:A=VAL("&H"+A$):POKE I;A:NEXT
40 DEF USR1=&HF000
50 X=USR1(1)
60 END
70 DATA 21,00,80,16,A0,1E,AA,DE
80 DATA A8,F3,2B,ED,51,7E,ED,59
90 DATA 77,7C,B5,20,F5,C9
```

Este programa parte de la siguiente configuración (Sony Hit Bit):

	Slot 0	Slot 2
&H0000 hasta &H7FFF	ROM	RAM
&H8000 hasta &HFFFF	-	RAM

Para otras divisiones se han de modificar los valores.

Página	0	1	2	3
A0 corresponde	ROM Slot 0	ROM Slot 0	RAM Slot 2	RAM Slot 2
AA corresponde	RAM Slot 2	RAM Slot 2	RAM Slot 2	RAM SLot 2
			Usuario	RAM

Después de que Vd. haya dejado transcurrir este programa, no se ve ninguna alteración. Pero la diferencia es importante: Primero conecte la configuración anterior con >OUT &HA8,&HA0<. Intente después modificar la ocupación del teclado a través de >POKE &HDA5,32<.
La dirección &HDA5 es el código ASCII que está asignado a la tecla "0". ¡Pero con la introducción de arriba no se modifica nada en la ocupación del teclado! Ahora cambie a RAM con >OUT &HA8,&HAA< y vuelva a intentar >POKE &HDA5,32<. Ahora pulse la tecla "0".
Obtendrá un espacio libre (código ASCII 32). Con >POKE &HDA5,49< el "1" se pone en el "0", etc.
Ahora es mucho más fácil de producir el teclado español de máquina de escribir (si aún no existe, como es el caso en algunos modelos). Experimente un poco con las tablas de matriz del teclado:

&HDA5 - &HDD4	sin Shift
&HDD5 - &HED4	con Shift
&HE05 - &HE34	con Graph
&HE35 - &HE5D	con Graph Shift
&HE65 - &HE94	con código
&HE95 - &HEC4	con código Shift
&H1033 - &H104A	carácter de mando; las tres filas inferiores de la matriz de teclado
&H104B - &H105A	Tabla para bloque de cifras, si éste existe
&H1061 - &H1066	Tabla de las teclas, sobre las cuales actúa la tecla de carácter especial
&H1067 - &H106C	Carácter producido con tecla especial y una tecla permitida
&H106D - &H1072	como arriba con Shift tecla especial
&H1073 - &H1078	como arriba con código tecla especial
&H1079 - &H107E	como arriba con código Shift tecla especial
&H107F - &H109D	tabla de los caracteres especiales, que también están afectados por la fijación CAP
&H109E - &H10BC	resultado de caracteres especiales con CAP conectado

Si alguna vez, al modificar el teclado, se enredase Vd. sin salvación, puede volver siempre a la ROM y al estado original por medio de >OUT &HA8,&HA0<.
Pero no sólo se puede modificar ahora la tabla del teclado, sino que las definiciones originales de caracteres también se encuentran en la ROM. El problema hasta ahora era, que cada juego de caracteres modificado era borrado por medio de un comando >SCREEN<. Normalmente, con el comando >SCREEN< se copia el contenido de la ROM en la VRAM. La tabla de caracteres ROM se encuentra a partir de la dirección &H1BBF hasta &H23BE.
Vamos a pokear una raya en el espacio libre:

```
POKE &H1BBF+32*8+7,255
```

Hasta ahora no ocurre nada. Aún tenemos que comunicar al sistema que la copia se encuentra ahora en el Slot 2. Esta información está en la dirección &HF91F.

```
POKE &HF91F,2
```

La dirección de la copia está en las direcciones &HF929/21. En nuestro caso sigue siendo correcta, o sea, &H1BBF.
Ahora introduzca >SCREEN 0< y obtendrá la raya deseada en el espacio libre. Con ayuda de la dirección &HF91F y &HF920/21 se puede copiar en la VRAM una tabla alternativa de juego de caracteres, cuando esté conectada a la ROM.

Quizás le moleste el signo de interrogación en el comando >INPUT<.
Ahora podemos sobreescribirlo simplemente con otro carácter, por ejemplo Space (código ASCII= 32):

```
POKE &H23D0,32
```

Pruebe: INPUT A$
En el lugar de la interrogación ha surgido un espacio libre. También es interesante >POKE &H23D0,29<. Con esto, la introducción empieza exactamente en la posición actual.

3.3 El VDP como aparato E/S

Desde el BASIC es posible el acceso a los registros VDP y a la RAM de vídeo con >VDP<, >VPOKE< y >VPEEK<. Pero internamente, el VDP y la VRAM son tratados como aparatos E/S.
Las direcciones del Port &H98 y &H99 se refieren al VDP. Las siguientes informaciones son especialmente interesantes para programadores en lenguaje máquina, ya que permiten un acceso al VRAM con la máxima velocidad.
Existen cuatro opciones básicas:

1 - Escribir datos en la VRAM
2 - Leer datos de la VRAM
3 - Escribir los contenidos de registros del VDP
4 - Leer el registro de estado

- 1 : La transferencia de datos de la CPU hacia la VRAM a través del VDP utiliza un registro de direcciones de 14 Bit autoincrementable. Con cada acceso, o sea, cada lectura o escritura de la VRAM, la dirección de este registro interno de VDP es aumentada. Para posicionar la dirección a un valor determinado, son necesarios dos traspasos de datos. Primero se editan los 5 Bits inferiores de la dirección (Byte bajo) a través del Port &H99. Después siguen inmediatamente los Bits restantes (el Byte alto). En el segundo traspaso, para indicar que se fija una dirección VRAM, el Bit 6 tiene que ser 1 y el Bit 7 ha de ser 0.

Ejemplo:

Traspasar dirección &H2030 del VDP:

Byte bajo: &H30
Byte alto: &H20

```
OUT &H99,&H30
OUT &H99,&H20 OR 2^6
```

Después de que se ha comunicado la dirección, a través del Port &H98 se puede escribir el valor en esta dirección (>OUT<).
Por ejemplo >OUT &H98,20< escribe el valor en las direcciones actuales. Con el acceso a través del Port &H98 se ha aumentado automáticamente la dirección a &H2031. Un nuevo >OUT &H988,...< escribiría, por lo tanto, el valor en esta dirección. Si se han de escribir muchos datos en direcciones consecutivas en la VRAM, entonces la función de autoincremento del VDP aumenta considerablemente la velocidad. En lugar de tres Bytes sólo se ha de transmitir un Byte.

- 2 : El proceso de escritura de direcciones es similar, sólo que esta vez los Bits 6 y 7 han de estar a 0 en la segunda transmisión. Para >BY = VPEEK(&H1234)< se puede escribir lo siguiente:

```
OUT &H99,&H34
OUT &H99,&H12
BY = INP(&H98)
```

Tambien aquí se aumentó la dirección a &H1235 por medio de >INP (&H98)<.

- 3 : Primero se transmiten a través del Port &H99 los datos a escribir en los registros correspondientes. Después se transmite el número de registro a través del mismo Port, para lo cual el Bit 7 ha de estar posicionado.
Para VDP(e) = 4 se puede escribir:

```
OUT &H99,4
OUT &H99,3 OR 2^7.
```

- 4 : Para preguntar por el registro de estado sólo se ha de ejecutar un comando >BY = INP(&H99)<. Este corresponde al comando >BY = VDP(8)<.

El uso de estas operaciones desde el BASIC es posible, pero a veces trae consigo complicaciones, ya que el Interrupt puede interferir la transmisión de datos.

Impresora:

Para la comunicación con la impresora son reponsables los Ports &H90 y &H91.
Primero se transmiten los datos con >OUT< a través del Port &H91. Por medio de >INP(&H90)< se espera hasta que el LSB sea 0, lo cual significa que la impresora está lista para la recepción. Después se manda también la señal Strobe a través del Port &H90.

PSG:

También el comando >SPUND< se puede representar como combinación de comandos >OUT/INP<. >OUT &HA0, registro< determina el registro afectado.
>OUT &HA1,valor< escribe el valor en el registro anteriormente determinado. Además el PSG posee otros dos Ports E/S, a los cuales está conectado el joystick.
Estos Ports son seleccionados por medio de los números de registros 14 (Port A) y 15 (Port B). Después se puede leer su contenido por medio de la dirección &HA2 del Port.

Capítulo IV : Sonido con el Chip de Sonido

4.1 Introducción

Para conseguir sacar música del generador de sonido del ordenador MSX, son necesarios algunos conocimientos generales. Estos conocimientos vamos a transmitirlos brevemente:

El sonido

Bajo sonido entendemos todos los procesos de oscilaciones audibles. Para nosotros son audibles cuando se encuentran en un sector de 16 hasta 20000 Hz. Por debajo del sonido audible se encuentra el infrasonido, por encima el Ultrasonido. El sonido puede dividirse en ruidos, mezclas de tonos, sonidos y tonos. Un sector del sonido, con una distribución diferente según cultura y época, se denomina música.

El tono

Bajo tono se entiende en la física el tono sinoidal puro, que, a excepción de la música electrónica (música sintética), no se encuentra en la práctica musical. Lo que nosotros denominamos tono en nuestro lenguaje, en la acústica (estudio del sonido) es denominado timbre de sonido.

El timbre de sonido

El timbre de un sonido se forma por medio de la superposición de una oscilación básica con oscilaciones superiores ''armónicos'', las cuales son múltiplos enteros de la oscilación básica. Un instrumento, como por ejemplo el piano o la guitarra, no produce nunca un tono tal como se ha definido anteriormente, sino que produce timbres.

De esta forma surge el timbre de sonido caracteristico de un intrumento, por medio de las osciaciones superiores.

Esta estructura de osciaciones superiores es frecuentemente muy complicada, también porque cambia durante una interpretación. Por ejemplo, la pulsación de una tecla de piano es rica en osciaciones superiores, en cambio, al decrecer el tono, éste es pobre en osciaciones superiores.

Como ya hemos mencionado, con aparatos electrónicos, y nuestro ordenador es uno de ellos, se pueden producir osciaciones puras, o sea, tonos físicamente puros. Estos tonos suenan algo "pobres". Para producir un tono lleno, el Chip de sonido dispone de algunas funciones, las cuales nos permiten modificar el sonido de tal forma que la impresión sea igual a un timbre de sonido. El PSG (Programable Sound Generator) ofrece las siguientes posibilidades de modificación:

Modificación de la altura del tono:

Se puede producir un tono, desde el sector inferior audible, unos 27 Hz, hasta el ultrasonido (no audible).

Modificación de la frecuencia de ruido

Modificación del volumen

Efectos de sonido

Los efectos de sonido de consiguen por medio de diferentes patrones de la modificación de volumen.

Tres canales de tonos

Se pueden hacer sonar simultáneamente tres canales de tonos y ruidos.

Para la programación del Sound Chip se dispone de los siguientes comandos:

4.2 El comando >PLAY<

Con este comando se puede interpretar piezas musicales después de indicar la altura del tono, velocidad, duración, volumen, pausas y forma de ejecución. La modificación de cada uno de los parámetros se consigue por medio de los así denominados "subcomandos".

Ejemplo: Hey Jude/Beatles

```
 10 T=80
 20 T$="t=t;18":PLAY T$,T$+"14",T$
 30 A1$="o5c4"
 40 B1$="o5c"
 50 C1$="r4"
 60 A2$="o4a2r8ao5cdo4g2r4ga"
 70 B2$="o3fffacccc"
 80 C2$="o3fcfcfcfcecececef"
 90 A3$="b-4o5f4r8fecdc16o4b-16a2r8o5c"
100 B3$="eb-b-gfccc8f8"
110 C3$="gcb-cb-cb-cfcfcfcfc"
120 A4$="dd4dg16fe16e16f16dc2o4fgao5d"
130 B4$="o4b-b-b-b-ffcf"
140 C4$="o2b-fb-fb-fb-fo3cfcfcfcf"
150 A5$="dcr8co4b-aeff4c2."
160 B5$="b-b-ecaaa2"
170 C5$="o3ccccccccfcfco2f2"
180 PLAY A1$,B1$,C1$
190 PLAY A2$,B2$,C2$
200 PLAY A3$,B3$,C3$
210 PLAY A4$,B4$,C4$
220 PLAY A5$,B5$,C5$
```

Descripción del programa

Línea 10:

En esta línea se determina la velocidad con la cual se ha de interpretar la pieza. >T< puede obtener valores de 32 hasta 255. Los valores de 32 hasta 255 son iguales a la cantidad de notas negras que han de interpretarse en un minuto.

Línea 20:

Para mantener variable el subcomando >Tn< se fija T=T en el String. Después de la introducción ha de seguir un punto y coma.
>L8< y >L4< son subcomandos, que representan la longitud de las notas a interpretar. En el subcomando >Ln< pueden estar contenidos valores entre 1 y 64. Estos valores tienen el siguiente significado:

1 = una nota entera
2 = una nota media
3 = un tercio de nota
4 = un cuarto de nota

*
*
*

Como en la pieza surgen muchas notas negras y corcheas, hemos memorizado los subcomandos >L4< (negras) y >L8< (corcheas) en T$. De esta forma se consigue una manera de introducción menos complicada en A$, B$ y C$. Como en la variable T$ se han fijado las longitudes frecuentemente utilizadas (>Ln<), no son necesarios los comandos en este sentido en los Strings de voces. Este truco es útil, ya que las líneas de String son más claras, y la entrada es más sencilla.

Línea 30 hasta 50:

En estas líneas se encuentra el compás de entrada para las tres voces. La variable >O< representa la octava. El subcomando >On< puede contener valores de 1 hasta 8. 04 es la así denominada octava central en el piano. En la música también se llama la octava sub-una.
El subcomando >C4< representa el tono DO con una duración de una nota negra.
En la línea 50 está >R4< para una pausa equivalente a una nota negra.

Línea 60 hasta 170:

Contiene en A$ la voz alta, en B$ la voz media y en C$ la voz baja. Para garantizar la sincronización de las diferentes voces es aconsejable utilizar una línea para cada compás..

Línea 180 hasta 190:

Aquí se activan los tonos contenidos en las variables de String en el orden deseado.

4.3 El comando >SOUND<

Con este comando se puede escribir directamente en el registro PSG. De esta forma se pueden conseguir efectos y sonidos especiales. Solo con el comando >SOUND< es posible aprovechar todas las posibilidades de sonido del PSG. Una ventaja del comando >SOUND< consiste en que el tono producido, o el sonido, suena hasta que el registro correpondiente sea escrito de nuevo con otra instrucción >SOUND<.

El PSG (Programable Sound Generator) dispone de 13 registros en total:

Bit	7	6	5	4	3	2	1	0
Registro 0	Low Byte del canal A							
Registro 1	0	0	0	0	High Byte can.A			
Registro 2	Low Byte del canal B							
Registro 3	0	0	0	0	High Byte can.B			
Registro 4	Low Byte del canal C							
Registro 5	0	0	0	0	High Byte can.C			
Registro 6	0	0	0	Frecuencia de ruido				
Registro 7	0	0	GC	GB	GA	TC	TB	TA
Registro 8	0	0	0	La	Volumen A			
Registro 9	0	0	0	La	Volumen B			
Registro 10	0	0	0	La	Volumen C			
Registro 11	Patrón modif. de vol. y fre. LB							
Registro 12	Patrón modif. de vol. y fre. HB							
Registro 13	0	0	0	0	Patrón de vol.			

Abreviaciones:

LB - Low Byte (ver capítulo: lenguaje máquina)
HB - High Byte
GC,GB,GA - Conectar ruido para canal A,B,C
LA - Al posicionar valor 16 : modificación del patrón de volumen a base de la envolvente posicionada
Fre. - Frecuencia

Registro 0 hasta 5:

En estos registros se puede memorizar la frecuencia de los tonos en el orden Low Byte (Byte de valor menor = registro 0), High Byte (Byte de valor mayor = registro 1).
Para calcular la frecuencia de los tres registros, se toma como base la frecuencia interna del ordenador (3,5 MHz).

$$\frac{3\ 579\ 545\ \text{(Hz)}}{32*\text{Frecuencia de salida (Hz)}} = 256*(\text{datos del reg. 1,3,5}) + (\text{datos del registro 0,2,4})$$

La fórmula se puede simplificar:

$$\frac{111\ 860{,}8\ \text{(Hz)}}{\text{frecuencia de salida (Hz)}} = 256*(\text{datos del registro 1,3,5}) + (\text{datos del registro 0,2,4})$$

Si, por ejemplo, queremos producir el tono de cámara "A" (440Hz), tendremos que realizar el siguiente cálculo:

$$\frac{111869{,}8\ \text{(Hz)}}{440\ \text{(Hz)}} = 254 = 256*0+254$$

Ahora colocamos estos valores calculados en los dos registros que son responsables de la frecuencia de tono de un canal, por ejemplo registro 0 y 1. Estos valores son 0 para el registro 1 y 254 para el registro 0.
Con el comando >SOUND< podemos traspasar ambos valores.

```
Sound 0,254
Sound 1,0
```

Son posibles frecuencias desde el sector inferior audible hasta el ultrasonido.
En los registros 0,2,4 pueden escribirse valores de 0 hasta 255.
Los registros 1,3,5 pueden contener valores de 0 hasta 15.

Registro 6:

Este registro determina la frecuencia del ruido. Esta se puede calcular por medio del siguiente método:

$$\text{Valor de registro 6} = \frac{3\ 579\ 545\ \text{(MHz)}}{32 * \text{frecuencia del ruido (Hz)}}$$

El valor puede oscilar entre 0 y 31.
Simplificando el cálculo:

$$\text{Valor} = \frac{111\ 860{,}8\ \text{(Hz)}}{\text{Frecuencia de ruido (Hz)}}$$

Si queremos producir una frecuencia de ruido de, por ejemplo 8000 Hz, tenemos que proceder de la siguiente forma:

$$\frac{111\ 860.8\ \text{(Hz)}}{8000\ \text{(Hz)}} = 14\ \text{(valor)}$$

Registro 7

Este registro elige un canal para la producción del tono y ruido. Se pueden traspasar valores de 0 hasta 63.

	Ruido			Tono		
Canal:	C	B	A	C	B	A
Valor:	32	16	8	4	2	1

Si, por ejemplo, queremos ocupar el canal A con el tono, canal B sólo con el ruido, y canal C con tono y ruido, tenemos que efectuar el siguiente cálculo:

Max.- (canal+canal+canal) = valor

Max.= valor máximo representable en es registro (63)

Para nuestro ejemplo:

63-(1+16+32+4) = 10

Este valor lo escribimos en el registro 7.

>SOUND 7,11<

Para avanzados sólo queda por mencionar, que este registro está orientado Low. Esto significa, que el estado 1 de un Bit significa DESCONECTADO y 0 significa CONECTADO.
Por lo contrario, exiten registros orientados High, en los cuales el estado 1 significa CONECTADO, y el estado 0 significa DESCONECTADO. El mando de registro orientado High es la formal normal.

Registro 8-10:

Estos registros permiten la regulación del volumen en el sector de valores de 0 hasta 15.

Registro 8 = canal A
Registro 9 = canal B
Registro 10 = canal C

Si el valor 16 se inscribe en un registro, entonces, con ayuda de los registros 11,12 y 13 se determina un transcurso del volumen.

Registro 11 y 12:

Aquí se indica la frecuencia para la modificación del patrón de volumen. Con esto, por ejemplo, se puede superponer un tono con el patrón formando un estado de oscilación, cuyo sonido tiene un carácter "extraterrestre".
Para el cálculo del periodo de la modificación de volumen se utiliza la siguiente expresión:

$$\frac{3\,579\,545\ (\text{Hz})}{532 * \text{periodo (Hz)}} = 256*(\text{datos de R12}) + (\text{datos de R11})$$

R = registro

Simplificando la expresión:

$$\frac{6728{,}5\ (\text{Hz})}{\text{periodo (Hz)}} = 256 * (\text{datos de R12}) + (\text{datos de R11})$$

Es posible un sector de frecuencia de 0,1 hasta 7000 Hz.
Si queremos realizar las modificaciones de frecuencia con una frecuencia de 20 Hz, tenmemos que realizar los siguientes cálculos:

$$\frac{6728{,}5\ (\text{Hz})}{20\ (\text{Hz})} = \text{Ca. } 336 = 256 * 1 + 80$$

En el registro 11 se escribe, por lo tanto, 80 y en el registro 12 un 1.

Para mejor comprensión de los parámetros necesarios para la formación de tonos, sigue a continuación un programa de ejemplo.
Por favor, introdúzcalo y pulse >RUN<.

Programa : curva senoidal

```
10 REM Curva senoidal
20 SCREEN 2:COLOR 1,15,7
30 OPEN"grp:" FOR OUTPUT AS #1
40 LINE (15,10)-(245,180),1,B
50 LINE (235,95)-(20,95),8
60 LINE (20,20)-(20,150),1
70 PSET (80,2),0
80 PRINT#1,"curva senoidal"
90 PSET(23,20),0
100 PRINT#1,"y"
110 PSET (237,95),0
120 PRINT#1,"X"
130 PSET (30,160),0
140 PRINT#1,"X=tiempo"
150 PSET (30,150),0
160 PRINT#1,"Y=elongación"
170 PSET (60,30),0
180 PRINT#1,"longitud de onda"
190 PSET(210,70),0
200 PRINT#1,"Amp"
210 PSET (30,170),0
220 PRINT#1,"Amp=Amplitud"
230 LINE (60,40)-(60,100),3
240 LINE (185,40)-(185,100),9
250 LINE (60,40)-(185,40)
260 LINE (220,50)-(220,65),10
270 LINE (220,80)-(220,95),10
280 PSET(20,125),0
290 FOR X=20 TO 225 STEP 3
300 Y=90+SIN(X/20)*40
310 LINE -(X,Y),2
320 NEXT X
330 IF INKEY$="" THEN 330
```

Descripción del programa:

Después de iniciar el programa, se dibuja una curva senoidal sobre la pantalla. Como ya hemos mencionado, un tono está compuesto por oscilaciones. Una de tales oscilaciones ha de representar la imagen sobre la pantalla.

Explicación de términos:

Elongación:

La amplitud sobre el eje Y desde el punto 0, se denomina elongación.

Duración del tono:

La duración del tono se indica por medio de los valores sobre el eje X.

Amplitud:

La amplitud representa la mayor elongación de una curva. Cuan mayor es la amplitud de una oscilación, mayor volumen tendrá el tono y viceversa.

Longitud de onda:

Una longitud de onda es un patrón de oscilación que se repite periódicamente.

Registro 13:

Este registro se utiliza para la elección de un determinado patrón para la variación del volumen. Estos patrones también se denominan envolventes, ya que envuelven, por decirlo así, al tono base con una determinada oscilación y magnitud de amplitud y a la magnitud de la amplitud propia. De esta forma se modifica el volumen en un modo prefijado por la forma de la envolvente.

El registro 13 nos permite la elección de en total 8 envolventes diferentes, las cuales todas ellas varían el transcurso del volumen de forma algo diferente. Las formas diferentes de envolventes las encontrará en su manual.
Ahora introduzca el siguiente programa e inícielo con >RUN<.

Programa:

```
10 MAXFILES=1
20 OPEN "grp:" FOR OUTPUT AS #1
30 DEFINT X,Y:DEFSNG S
40 SCREEN 2: COLOR 1,15,7
50 LINE (15,10)-(245,180),1,B
60 LINE (20,15)-(20,170),8
70 PRESET (17,180)
80 PRINT#1,"Envolvente (diente de sierra)"
90 PRESET (21,15)
100 PRINT#1,"y=Volumen"
110 PRESET (170,170)
120 PRINT#1,"x=tiempo"
130 PRESET (238,95)
140 PRINT#1,"X"
150 LINE (236,95)-(20,95),8
160 LA=50:REM longitud de onda de la envolvente
170 AN=200/LA: REM cantidad de dientes sobre el eje X
180 FY=60/LA:REM factor de ampliacion del eje Y
190 FOR I=1 TO AN STEP 2
200 LINE -(20+LA*I,95-LA*FY),2:REM flanco
210 LINE -(20+LA*(I+1),95),2:REM diagonal
220 NEXT I
230 FOR X=20 TO 236 STEP 2
240 R=(X-20) MOD (LA*2):REM resto de LA * 2
250 AM=R*SGN(LA-.1-R)-LA*2*(R>=LA)
260 SY=SIN((X-20)/3):REM amplitd del seno
270 Y=95-SY*AM*FY:REM y amplitud total
280 LINE -(X,Y),1
290 NEXT X
300 CLOSE #1
310 IF INKEY$="" THEN 310
```

Descripción del programa

Este programa le mostrará como funciona la envolvente. Hemos elegido una oscilación triangular (posibilidad 14 con subcomando en el manual).

Si ahora modifica las siguientes líneas, obtendrá un patrón tal como la posibilidad 12 del manual:

```
120 LINE -(20+LA*(I-1),95-LA),2:REM flanco
130 LINE -(20+LA*I1,95),2:REM diagonal
160 AM=LA-(X-20)MOD LA:REM amplitud actual de la envolvente
```

Modificando las siguientes líneas obtendrá la posibilidad 8:

```
130 LINE -(20+LA*I,95-LA*FY),2:REM flanco
140 LINE -(20+LA*(I+1),95),2: REM diagonal
170 R=(X-20) MOD (LA*2):REM resto de LA*2
180 AM=R*SGN(LA-.1-R)-LA*2*(R>=LA)
```

Compruebe también diferentes valores para las siguientes variables:

LA - longitud de onda de la envolvente
AN - Cantidad de dientes sobre el eje X
FY - Factor de ampliación del eje Y
SY - Amplitud del seno
Y - Amplitud total

Por medio de las envolventes son posibles algunas manipulaciones. Intente Vd. conseguir algunos tonos o ruidos interesantes probando.

Programa: Organo

El siguiente programa realiza el teclado de un organo. En él vale:

Pulsar tecla : A W S E D F G Y H U J K

Resultan los tonos : Do Do≠ Re Re≠ MI Fa Sol Sol≠ La B SI D

Lo especial en éste programa es que se pueden tocar acordes (tres tonos simultáneamente).

```
10 DEFUSR1=&HD12
20 SCREEN 0,,0
30 DEFINT A-Z
40 KB=&HFBF0
50 PP=&HF3F8
60 GP=&HF3FA
70 DIM F(25)
80 FOR I=65 TO 90
90 READ F(I-65):NEXT
100 SOUND 7,63:FOR I=8 TO 10:SOUND I,10:NEXT
110 P=10:GOSUB 190
120 A=PEEK(PP)-&HF0-P:IF A<=0 THEN GOSUB 190:GOTO 170
130 IF A>3 THEN A=3
140 GOSUB 190
150 FOR I=0 TO A-1:T=(PEEK(KB+I)AND&B11011111)-65
160 SOUND 2*I,F(T)MOD256:SOUND 2*I+1,F(T)/256:NEXT
170 SOUND 7,64-2^(A)
180 GOTO 120
190 P=10-P
200 POKE PP,&HF0+P:POKE PP+1,&HFB
210 POKE GP,&HF0+P:POKE GP+1,&HFB
220 POKE &HF3F7,1
230 FOR W=0 TO 3:X=USR1(1):NEXT
240 RETURN
250 REM A-H
260 DATA 1710,0,0,1357,1439,1281,1141,1016
270 REM I-P
280 DATA 0,906,855,1,0,0,1,1
290 REM Q-Z
300 DATA 0,0,1524,1209,960,0,1615,0,1078,0
```

Descripción de programa : Organo

Línea 10:

Dirección inicial de la rutina que ejecuta la consulta al teclado. Normalmente, ésta se activa automáticamente por medio del interrupt.

Línea 40:

KB es la dirección inicial de la memoria del Key buffer.

Línea 50:

PP es la así denominada dirección Put Point, en la cual se encuentra la dirección del final actual del Keybuffer.

Línea 60:

GP es la dirección Get Point, en la cual se encuentra la dirección del inicio actual del Keybuffer.

Línea 70:

F contiene la frecuencia asignada a cada tecla correspondiente.

Línea 80,90:

En estas líneas se asigna la frecuencia a cada una de la teclas descritas anteriormente (Líneas DATA 160 hasta 300). 0 está para una tecla no ocupada.

Línea 100:

Todos los canales conectados, y volumen para todos los canales a 10.

Línea 110:

P es la diferencia del primer al segundo buffer utilizado. Son necesarios dos Keybuffers para que sea registrada una nueva pulsación de tecla en un buffer, mientras que el otro es iniciado. El subprograma a partir de la línea 190 inicializa un Keybuffer y activa la consulta por el teclado.

Línea 120:

Comienzo del programa principal.
A contiene la cantidad de las teclas pulsadas.
SI A=0, entonces consulta por el teclado (GOSUB 1900) y todos los canales desconectados (GO TO 170).

Línea 130:

Tener en cuenta las 3 siguientes teclas pulsadas.

Línea 140:

Consulta al teclado

Línea 150:

Leer la tecla pulsada segun orden.

Línea 160:

Posicionar la frecuencia respecto a la tecla correspondiente.

Línea 170:

Y conectar los correspondientes canales.

Línea 180:

Volver al comienzo del programa pricipal.

Línea 190:

Subprograma de consulta al teclado.
Con cada consulta cambia P de 10 a 0 y viceversa. De esta forma se cambia constantemente entre ambos buffers.

Línea 200,210:

Borrar Keybuffer y posicionar a dirección inicial.

Línea 220:

Desconectar el retraso de repetición.

Línea 230:

Consultar tres veces al teclado.

Programa : Sintetizador

A continuación sigue un programa de sintetizador, con el cual se pueden aprovechar, y comprobar fácilmente, todas la posibilidades >SOUND< de los ordenadores MSX.

El programa es manejado confortablemente por medio del joystick. Es posible andar libremente sobre el menú visible sobre la pantalla. Si quiere modificar un valor, pulse el disparador. Ahora puede disminuir o aumentar los valores por movimiento del joystick hacia la izquierda o la derecha. Para indicación de este modo surge una estrella en la izquierda sobre el término elegido. Para volver al modo normal se ha de pulsar nuevamente el disparador. Para indicación al abandonar el modo de introducción desaparece la estrella inmediatamente de la pantalla.

```
10 REM Sintetizador
20 DEFINT A-Z
30 DEFSNG W,D,M,F
40 DIM W(7,3)
50 DIM X(8,4),Y(8,4)
60 SCREEN 1:WIDTH 29
70 TF#=1789772.5#
80 MF(2)=&HFFF:MF(5)=&H1F:MF(6)=2^16-1
90 PRINT"    canal 1  canal 2  canal 3"
100 PRINT"Tono:desc.    desc.    desc."
110 FOR I=1 TO 3:W(1,I)=1:NEXT:REM desconectado
120 PRINT:PRINT"Fre:"
130 FOR I=1 TO 3:W(2,I)=300:NEXT:REM frecuencia prefijada
140 PRINT:PRINT    "Rui: desc.    desc.    desc."
150 FOR I=1 TO 3:W(3,I)=1:NEXT:REM desconectado
160 PRINT:PRINT"Vol:"
170 FOR I=1 TO 3:W(4,I)=10:NEXT:REM volumen prefijado
180 PRINT:PRINTSTRING$(29,"-")
190 PRINT"         ruido"
200 PRINT:PRINT"Fre:"
210 W(5,1)=10:REM frecuencia de ruido prefijada
220 PRINT:PRINT"         envolvente"
230 PRINT:PRINT"Fre:"
240 W(6,1)=10
250 PRINT:PRINT"Num:"
260 W(7,1)=9:REM prefijacion del numero de envolvente
270 FOR I=1 TO 4
280 FOR J=1 TO 3
290 X(I,J)=-4+9*J
300 Y(I,J)=2*I
310 NEXT J,I
320 X(5,1)=5:Y(5,1)=14
330 X(6,1)=5:Y(6,1)=18
340 X(7,1)=5:Y(7,1)=20
350 FOR I=5 TO 7
360 FOR J=2 TO 3
370 X(I,J)=0:Y(I,J)=0
380 NEXT J,I
```

```
390 Y=4
400 FOR X=1 TO 3
410 LOCATE X(Y,X)-1,Y(Y,X),0
420 PRINTW(Y,X);
430 NEXT
440 X=1:Y=7:LOCATE X(Y,X)-1,Y(Y,X),0
450 PRINTW(Y,X);
460 SOUND 13,W(7,1)
470 Y=2
480 FOR X=1 TO 3
490 GOSUB 1120:NEXT
500 X=1:Y=5:GOSUB 1190
510 SOUND 7,63:R7=63
520 FOR X=1 TO 3:SOUND 7+X,W(4,X):NEXT
530 X=1:Y=6:GOSUB 1250
540 X=1:Y=1
550 ON STRIG GOSUB 700,700,700,700,700
560 EI=1
570 STRIG(EI) ON
580 GOTO 600
590 IF DX=0 AND DY=0 THEN 620
600 LOCATE X(Y,X),Y(Y,X),1
610 DX=0:DY=0
620 R=STICK(EI)
630 IF R=8 OR R=1 OR R=2 THEN DY=-1
640 IF R>3 AND R<7 THEN DY=1
650 IF R>1 AND R<5 THEN DX=1
660 IF R>5 AND R<=8 THEN DX=-1
670 IF X(Y+DY,X)=0 THEN DY=0
680 IF X(Y,X+DX)=0 THEN DX=0
690 X=X+DX:Y=Y+DY:GOTO 590
700 REM Strig
710 IF Y<>1 AND Y<>3 THEN 790
720 W(Y,X)=1-W(Y,X)
730 LOCATE X(Y,X),Y(Y,X),0
740 IF W(Y,X)=0 THEN PRINT"Cone.";ELSE PRINT "Desc.";
750 BI=X-(Y=3)*3-1
760 R7=(R7 AND (NOT2^BI)) OR (W(Y,X)*2^BI)
```

```
770 SOUND 7,R7
780 GOTO 1100
790 IF Y<>4 AND Y<>7 THEN 950
800 IF LF THEN LF=0:RETURN ELSE LF=-1
810 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT"*";
820 FOR WA=0 TO 1000:NEXT
830 A=PEEK(&HF3E8)
840 IF (A AND 16) =0 THEN 1100
850 R1=STICK(EI):IF R1=0 THEN 830
860 IF R1>1 AND R1<5 THEN D=1
870 IF R1>5 AND R1<9 THEN D=-1
880 W=W(Y,X)+D
890 IF W>(16+2*(Y=7)) OR W<0 THEN 830
900 W(Y,X)=W
910 IF Y=4 THEN SOUND 7+X,W ELSE SOUND 13,W
920 LOCATE X(Y,X)-1,Y(Y,X),0
930 PRINTW;
940 GOTO 830
950 REM Frequenzen
960 IF VF THEN VF=0:RETURN ELSE VF=-1
970 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT"*";
980 FOR WA=0 TO 1000:NEXT
990 DP=-1:DM=1:D=0
1000 A=PEEK(&HF3E8)
1010 IF (A AND 16) =0 THEN 1100
1020 R1=STICK(EI):IF R1=0 THEN DP=-1:DM=1:D=0:GOTO 1000
1030 IF R1>1 AND R1<5 THEN D=D+DP:DP=DP*2
1040 IF R1>5 AND R1<9 THEN D=D+DM:DM=DM*2
1050 W=W(Y,X)+D
1060 IF W<1 OR (W>MF(Y)) THEN 990
1070 W(Y,X)=W
1080 IF Y=2 THEN GOSUB 1120 ELSE IF Y=5 THEN GOSUB 1190 ELSE
 GOSUB 1250
1090 GOTO 1000
1100 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT" ";
1110 LOCATE X(Y,X),Y(Y,X),1:RETURN
1120 REM Frecuencia Reg Calc.
1130 W=W(2,X)
```

```
1140 F=TF#/16/W
1150 SOUND (X-1)*2,W-INT(W/256)*256
1160 SOUND (X-1)*2+1,INT(W/256)
1170 GOSUB 1310
1180 RETURN
1190 REM ruido Reg Calc.
1200 W=W(5,1)
1210 F=TF#/16/W
1220 SOUND 6,W
1230 GOSUB 1310
1240 RETURN
1250 REM Frec.envolvente Reg. Calc.
1260 F=TF#/256/W
1270 SOUND 11,W-INT(W/256)*256
1280 SOUND 12,INT(W/256)
1290 GOSUB 1310
1300 RETURN
1310 A$=LEFT$(STR$(F)+"      ",7)
1320 LOCATE X(Y,X)-1,Y(Y,X),0
1330 PRINTA$;
1340 RETURN
```

Descripción del programa:

Línea 70:

FF≠ contiene la frecuencia interna media del ordenador.

Línea 80:

MF contiene los valores máximos de registro para:

Frecuencia de tono	MF(2)
Frecuencia de ruido	MF(5)
Frecuencia de la envolvente	MF(6)

Línea 90 hasta 260:

Construcción de pantalla y definición de los datos pertenecientes a los correspondientes campos.
W(I,J) contiene los datos de la línea I de introducción y de la columnas J.
Con Desconectado/Conectado, el valor 1 significa desconectado, el 0 conectado. Con todos los demás valores W(I,J) corresponde a los contenidos de los registros.

Líneas 270 hasta 380:

Para posibilitar el salto con joystick entre los diferentes campos, se asigna a cada campo la posición correspondiente de la línea I y de cada columna J sobre la pantalla. 0 significa campo no válido.

Línea 390 hasta 530:

Aquí se transforman los valores W(I,J) prefijados al conectar a los valores de frecuencia o volumen, y son visualizados sobre la pantalla.

Línea 390 hasta 430:

Edición del volumen

Línea 440 hasta 460:

Edición y posicionado del número de envolvente

Línea 470 hasta 490:

Edición y posicionado de las frecuencias de tonos

Línea 500:

Posicionar y editar frecuencia de ruido

Línea 510:

Conectar canales

Línea 520:

Posicionar valores de volumen

Línea 530:

Posicionar y editar frecuencias de envolventes

Línea 600:

Visualizar cursor

Línea 620:

¿Es movido el joystick? No, entonces esperar.

Línea 630 hasta 660:

Fijar la dirección

Línea 670 hasta 680:

Comprobar si el campo de entrada está permitido en la dirección solicitada. Si no X(..)=0, entonces no ejecutar el movimiento.

Línea 690:

Posicionar nuevo campo de introducción y comenzar de nuevo.

Línea 710:

Si no se introduce en la línea 1 (Tono desc./conec.) o línea 3 (ruido conect./ desc.), entonces seguir.

Línea 720:

Cambiar valor (estado conect./ desc. o 0/1)

Línea 730:

Desconectar cursor

Línea 740:

Indicar nuevo estado

Línea 750:

Calcular número de Bit para el registro >SOUND<.

Línea 760:

Calcular Byte para el registro 7.

Línea 780:

Final String

Línea 790:

Si no se introduce en la línea 4 (volumen) ó 7.(número de envolvente).

Línea 800:

Si el interrupt se activa al "hacer fuego" la segunda vez, reposicionar LF y no hacer caso al interrupt. Si no, posicionar LF, para que el siguiente interrupt cause el final del modo de introducción.

Línea 810:

Editar estrella como señal de introducción

Línea 820:

Esperar hasta que el disparador no esté pulsado.

Línea 830,840:

¿Se ha pulsado el botón la segunda vez? Si éste es el caso, final de la introducción.

Línea 850 hasta 880:

Determinar dirección del joystick

Línea 890:

Test de límites, 0-16 de volumen y 0-14 para el número de la envolvente.

Línea 900 hasta 930:

Posicionar nuevo valor y editar

Línea 950:

Suponer modificación de la frecuencia

Línea 960 hasta 1090:

Análoga a las líneas 800 hasta 930.
La diferencia consiste, en que la velocidad de modificación aumenta al mantener una dirección. Para ello se multiplican DP o DM por 2.

Línea 1100:

Final de la rutina String. La estrella se borra

Línea 1110:

Y el cursor se indica nuevamente sobre el campo de entrada

Las siguientes líneas contienen subprogramas. Su función queda clara en las explicaciones de las líneas REM.

Capítulo V : Lenguaje de máquina

5.1 ¿Por qué lenguaje máquina?

La mayoría de los ordenadores personales están equipados con BASIC. Como Vd. habrá notado, este lenguaje no es muy difícil de aprender. Especialmente el MSX BASIC sobresale con su gran variedad de comandos. Se tiene la impresión de que con este BASIC no queda nada sin cumplir, y que todos los problemas de programación se pueden resolver bien. Introduzca lo siguiente y observe el tiempo:

```
 5 SCREEN 2
10 HL=&H2000
20 A=1
30 VPOKE HL,A
40 HL=HL+1
50 IF HL<&H3000 THEN 20
60 IF INKEY$="" THEN 60
70 RETURN
```

Inicie el programa con >RUN< y observe lo que sucede. Si quiere volver a la introducción, pulse cualquier tecla.

El siguiente programa carga el programa máquina con la misma finalidad que el programa BASIC:

```
 10 CLEAR 200, &HEFFF
 20 FOR I=&HF000 TO &HF014
 30 READ A
 40 POKE I,A
 50 NEXT I
 60 DEFUSR1=&HF000
 70 END
 80 DATA &HCD,&H72,&H00,&H21,&H00,&H20,&H3E,&H01
 90 DATA &HCD,&H4D,&H00,&H23,&H3E,&H30,&HBC
100 DATA &H20,&HF5,&HCD,&H9F,&H00,&HC9
```

Cargue ahora el programa máquina con >RUN<, llame al programa así cargado por medio de >X=USR1(1)< y se quedará asombrado.

Como habrá visto, los programas transcurren en:

- BASIC : 43 segundos
- Programa máquina : menos de 1 segundo.

La longitud es de:

- Programa BASIC : 97 Bytes
- Lenguaje máquina : 21 Bytes, desde &HF000 hasta &HF014.

Similitud de los programas:

BASIC	Lenguaje Assembler
5 SCREEN 2	- CALL &H0072
10 HL=&H2000	- LD HL, &H2000
20 A=1	- LD A,1
30 VPOKE HL,A	- CALL &H004D
40 HL=HL+1	- INC HL
50 IF HL<&H3000THEN 20	- LD,A,&H30
	- CP H
	- JR NZ,-11
60 IF INKEY$="" THEN 20	- CALL &H009F
70 RETURN	- RET

Explicación:

Línea 5: CALL &H0072 activa la rutina que conecta el modo gráfico >SCREEN 2<.

Línea 10: Aquí se posiciona el valor para la variable HL, o el registro HL se coloca al comienzo de la memoria de color.

Línea 20: En esta línea se memoriza el valor del color a editar (1=negro).

Línea 30: Aquí se escribe el valor A en la memoria de pantalla, lo cual origina que se visualize una raya negra.

Pruebe Vd. simplemente diferentes valores para las direcciones HL en la memoria de color (HL ha de estar entre &H2000 y &H3800), y para A el código del color.

Línea 40: Aquí, la variable HL, la cual contiene la dirección en la memoria de color, es aumentada en una, para que toda la pantalla sea completada.
(Inglés: INCrease: aumentar)

Línea 50: Consulta si HL es mayor que &H3000, o sea, si se ha llegado al final del sector de la memoria del color. Esta consulta en lenguaje máquina ha de dividirse en tres comandos:
LD (inglés load = cargar); cargar A con el valor comparativo (=High Byte de &H3000).
CP H: comparación con H, el High Byte de HL (CP: inglés compare = comparar).
JR (inglés jump relative : salto relativo); Nz (inglés non zero : no es cero). Por lo tanto, se puede decir:
"Salta, si no es cero" a la distancia indicada.

Línea 60: Activar rutina de consulta al teclado con CALL &H009F. Cuando la tecla sea pulsada, se efectúa el salto hacia atrás.

Línea 70: RET finaliza el subprograma.

A continuación veremos como ejemplo un listado de ensamblador:

Listado Assembler para programa máquina

```
Dirección código  No. línea  Comando Assembler; comentario

F000 CD7200          5     CALL &H0072 ; rutina para el co-
                                         nectado del modo
                                         gráfico >SCREEN 2<
F003 210020          10    LD   HL,&H2000; inicio memoria
                                          de color
F006 3E01            20    LD   A,&H01 ; Color de dibujo
                                         negro (=1)
F008 CD4D0           30    CALL &H004D ; rutina para editar
                                         un carácter sobre
                                         la pantalla
F00A 23              40    INC  HL ; Aumentar dirección de
                                       memoria de color
F00B 3E30            50    LD   A, &H30
F00D BC              60    CP   H ; H>&H30 ?
F00E 20F5            70    JR   NZ,&HF003 ; no, otra vez
F011 CD9F00          80    CALL &H009F ; rutina para con-
                                         sulta del teclado
F014 C9              90    RET ; vuelta al BASIC
```

Las ventajas del lenguaje máquina se ven claramente. Son la velocidad con la cual transcurren los programas y el ahorro considerable de memoria. Algunos problemas se pueden resolver efectivamente con el lenguaje máquina, por ejemplo, procesado de texto, sucesión rápida de imágenes en gráficas (juegos, construcción). También algunas rutinas, especialmente para cálculos matemáticos extremadamente exactos, sólo se pueden realizar en lenguaje máquina.
Estas han sido algunas de las ventajas del lenguaje máquina.

Estos ejemplos deberían ser suficientes para demostrar la necesidad de la programación en el lenguaje máquina, también en ordenadores con un BASIC muy avanzado, como son los ordenadores MSX. Pero también se ha de decir que este lenguaje tiene una gran desventaja.
El lenguaje máquina es el lenguaje del microprocesador (CPU) del ordenador y es, de esta forma, el lenguaje más directo de máquina. Pero tal orientación hacia la máquina exige del programador una forma de pensar muy abstracta para poder entender el lenguaje. Esto se fundamenta en que la CPU sólo entiende números, o sea, un programa máquina es simplemente una serie de números y no una sucesión de términos. En esta forma, la programación en lenguaje máquina para programas extensos sería casi imposible. Por este motivo se inventó un tipo de lenguaje intermedio, el cual hace que el lenguaje máquina sea más claro. Este lenguaje se llama Assembler. El lenguaje Assembler asigna a cada código máquina (o sea a un número) una serie de símbolos. Estos símbolos están compuestos por:

1. Palabra de comando: es casi siempre una abreviación de la palabra inglesa para el comando, llamada también llamada Mnemónic.

2. Operandos: indican direcciones, constantes, etc. (referidos a la palabra de comando).

De esta forma se simplifica la elaboración de un programa de máquina a una escritura en lenguaje Assembler. Este lenguaje Assembler es traducido automáticamente, por medio de un programa Assembler, al código máquina. Además existe un Disassembler, programa que traduce las series de números del lenguaje máquina al lenguaje Assembler. En el ejemplo anterior ha conocido Vd. un listado de Assembler.

5.2 Sistemas de números

Como un ordenador sólo trabaja internamente con números, concretamente con el 0 y el 1 (ordenador digital), es necesario que el programador en lenguaje máquina sepa algo sobre los sistemas de números. Son de importancia los siguientes sistemas de números:

1. El sistema decimal
2. El sistema dual
3. El sistema hexadicmal

Sistemas de números son series de cifras según un determinado esquema. Cada número puede ser transformado a otros sistemas de números. En todos los sistemas, el valor de la cifra aumenta de derecha a izquierda.
Para explicar los demás sistemas de números, vamos a partir del conocido sistema decimal.

El sistema decimal

```
Decenas de mil  Centenas  Decenas  Unidades  - Valor
      7             3         5         6    - cifras
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
¡El valor de la cifra aumenta de derecha a izquierda!
```

Potencia	Número	Denominación
10^0	1	U-nidades
10^1	10	D-ecenas
10^2	100	C-entenas
10^3	1000	U-nidades de M-il
10^4	10000	Decenas de mil
10^6	1000000	Millón

Pero el número decimal 1335 también se puede escribir de otra forma:

1335 significa : 1UM + 3C + 3D + 5U - el valor más bajo (uni-
dades) a la derecha.

435 significa : 4C + 3D + 5U - extremo derecha

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

1335 también es $1*10^3 + 3*10^2 + 3*10^1 + 5*10^0$

Una potencia elevada a 0 es igul a 1.

por ejemplo: $10^0 = 1$, $2^0 = 1$, $x^0 = 1$

El sistema binario

El sistema dual está formado con la misma estructura. La diferencia consiste únicamente en que, el valor de cada cifra no está representado por potencias de base 10, sino por potencias de base dos.
La base del sistema dual es el 2.

Binario 10101101 = decimal 173

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ - valor de la cifra

1 0 1 0 1 1 0 1 - cifra

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

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

Hasta ahora ha visto la transformación del sistema dual al sistema decimal. Este proceso es naturalmente reversible. Para explicación de la inversión, veamos el número 173.
Pensemos qué potencia de base 2 está contenida justamente en este número. Para ayuda: en principio se puede aplicar el sistema binario a números con n cifras. En el sector de ordenadores se utilizan principalmente números binarios de 8 cifras. Pueden surgir las siguientes potencias:

potencias de 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
valores tranformados	128	64	32	16	8	4	2	1

Por lo tanto, en este casó la potencia mayor sería 2^7 = 128. Ahora formamos la diferencia entre 173 y 128. El resultado es 45. Con este resto procedemos de la misma forma que anteriormente. Buscamos nuevamente la mayor potencia de 2 que quepa en este resto. Con la tabla se puede calcular y resulta : 2^5 = 32. A continuación formamos nuevamente la diferencia: (42-35=13).

El sistema descrito se aplica hasta que el resto sea 0.

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

Hemos determinado las siguientes potencias de 2:

2^7, 2^5, 2^3, 2^2, 2^0

Debajo de cada potencia determinada escribimos un 1 y un 0 debajo de las que falten:

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

1 0 1 0 1 1 0 1 = 173

Por lo tanto, el número decimal 173 es representado en el sistema binario por 10101101. A continuación vamos a simbolizar números binarios por la anteposición de &B.

por ejemplo : 173 = &B 10101101

Bit y Byte

Un Bit es la unidad de información más pequeña, de la cual se forman todas las demás informaciones. Bit es la abreviación de "Binary Digit", lo cual significa tanto como cifra binaria. Se habla de un Bit posicionado cuando tiene el estado 1, o de un Bit reposicionado cuando se encuentra en estado 0.
Los ordenadores MSX tiene un procesador de 8 Bit, o sea, que puede elaborar números duales de 8 Bits de longitud, lo cual corresponde del 0 al 255 en números decimales.

Número binario:

1 0 1 1 0 1 1 1

p r p p r p p p p = Bit posicionado; r = Bit reposicionado

7 6 5 4 3 2 1 0 Número del Bit.

A cada Bit, a cada cifra de un número binario, se ha asignado un número de Bit. El Bit con el menor valor, o sea, el que se encuentra a la derecha, tiene el número 0. De la derecha a la izquierda se numera sucesivamente. El número de Bit corresponde a la potencia binaria, que representa el valor de la misma. En un ordenador a veces es útil imaginarse los estados de Bits como un conmutador.

Conmutador conectado = 1
Conmutador desconectado = 0

Con un número de 8 conmutadores se pueden representar valores de 0-255, o sea, 256 estados de conmutación.
Ocho conmutadores (Bits) agrupados se denominan un Byte. Un Byte puede ser depositado por el ordenador en una posición de memoria. Pero, ¿cómo se memorizan números que sean mayores que 255? Para esta finalidad se divide el número en dos partes: el Byte bajo, y el Byte alto).

Estos Bytes se depositan en dos espacios de memoria contiguos.

Los Bytes alto y bajo se pueden calcular de la siguiente forma:

Número dividido por 256 = (Byte alto) + resto
El resto de la división corresponde al Byte bajo.

Para recordar: El número 255 es el valor máximo representable en un Byte, ya que está formado por 8 Bits.

Ejemplo: El número 34065 ha de dividirse en un Byte bajo y uno alto:

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

133 = Byte alto
17 = Byte bajo

Las fórmulas generales en BASIC son:

1. HB = INT (número/256) HB = Byte alto (High Byte)
   LB = número - HB*256 LB = Byte bajo (Low Byte)

Esta fórmula se puede aplicar con números de cualquier tamaño.

2. HB = número U 256 HB = Byte alto (High Byte)
   LB = número MOD 256 LB = Byte bajo (Low Byte)

La segunda fórmula se aplica con números que están en el sector de -32768 hasta 32767.

De esta forma un número que se encuentre en el sector de 256 hasta 65535, y se tenga que depositar en la memoria, sólo necesita 2 Bytes.
Para la representación más simple de números, que se han de memorizar de esta forma, es conveniente la introducción de otro sistema de números.

El Sistema Hexadecimal

La base del sistema hexadecimal es el 16.
Para recordar:
La base del sistema decimal es 10.
La base del sistema binario es 2.

Para representar cifras, cuyo valor es mayor que 10, en el sistema hexadecimal se utilizan las letras A hasta F.

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

Sistema hexadecimal:
0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10,11,12,13.....

Primero transformamos los números hexadecimales a números decimales:

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

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

Otro ejemplo:

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

&H1A3E = 4096 + 2560 + 48 + 14
&H1A3E = 6718

Ahora sigue la transformación de números decimales a hexadecimales:

El proceso de transformación es idéntico al descrito en las páginas anteriores. Supongamos que el número decimal 45380 se ha de transformar al sistema hexadecimal:

1. Paso: Buscamos la máxima potencia de 16 que puede caber en este número (para ayuda se encuentran los números en la tabla anterior de transformación: hexadecimal en decimal).

2. Paso: Dividimos nuestro número (45380) por este valor (4096), y obtenemos así nuestro número hexadecimal:

```
45380/4096 = 11 resto 324
decimal = 11 => hexadecimal B
```

3. Paso: Ahora se sigue el mismo proceso con el resto (324). Este lo dividimos ahora por el siguiente número potencia de 16 que quepa, en este caso el 256.

```
324/256 = 1 resto 68
decimal 1 => hexadecimal 1
```

Los cálculos así descritos se prosiguen hasta que el resto de la división sea 0.

```
68/16=4 resto 4
decimal = 4  => hexadecimal 4
4/1=4 resto 0
decimal =4 => hexadecimal 4
```

Nuestro número transformado es &HB144.

La ventaja del sistema hexadecimal consiste en que se puede leer directamente el High y el Low Byte. Para &H3ABF vale:

- El High Byte está compuesto por las dos primeras cifras hexadecimales (3 y A) . Tiene el valor decimal (3*16^1 + 10*16^0)=58.

- El Low Byte está compuesto por las dos últimas cifras hexadecimales (B y F). Tiene el valor decimal (11*16^1 + 15*16^0)=191.

Introduzca lo siguiente:

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

En las dos direcciones &H1D y &H1E se encuentra la dirección de salto, hacia la cual se deriva el sistema operativo cuando se ha de activar una rutina, por ejemplo en un módulo enchufable. Para una dirección de salto es posible un valor de 0 hasta 65535 (o sea, hasta &HFFFF). Este número es memorizado con ayuda de los Bytes alto y bajo. Vamos a calcular ahora la dirección de salto. Con el comando BASIC anterior obtenemos el valor 23 en la dirección &H1D y el valor 2 en la dirección &H1E. Por lo tanto, la dirección de salto decimal resulta 2*256+23=535.

Ahora vamos a efectuar el mismo cálculo en el sistema hexadecimal:

23=&H17 y 2=&H2 como Vd. puede comprobar fácilmente. El valor de la dirección de salto se obtiene simplemente escribiendo el Byte alto y el bajo contiguos: 535 = &H217.

Por lo tanto, es igual de fácil que dividir un número hexadecimal en Byte alto y bajo. Por lo general, el Byte bajo de un número está en la dirección inferior de memoria, y a continuación está el Byte alto.

Aquí ha conocido Vd. la primera ventaja del sistema hexadecimal. Además se puede realizar fácilmente la transformación del sistema binario al sistema hexadecimal. Para ello se divide el número binario en dos bloques de 4 Bits cada uno. El bloque de los Bits 0 hasta 3 se llama Low Nibble, y el otro bloque del 4 al 7 se llama High Nibble. Cada Nibble corresponde exactamente a una cifra hexadecimal.

Esto es obvio, ya que un número binario de 4 Bits sólo puede llegar al valor 15 (15=8+4+2+1). Pero todos los valores de 0 hasta 15 pueden ser representados por una cifra hexadecimal (0,1,...9,A,B,C,D,E,F). Veamos un ejemplo:

```
1 1 0 1   1 0 0 1
N. alto   Nibble bajo
8+4+1      8+1
  13        9
  &HD       &H9
```

Por lo tanto: &B11011001 = &HD9

Con un poco de práctica se puede leer directamente el número hexadecimal perteneciente a un número dual de 4 Bits. En ello ha de apoyarle la siguiente tabla:

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

De forma similar se lleva a cabo la transformación de hexadecimal a binario. Cada cifra hexadecimal es sustituida por la combinación correspondiente de 4 Bits, por ejemplo, &HC7=&B1100 0111.

La comprensión de la transformación entre los diferentes sistemas de números es la base para la programación en lenguaje de máquina e igualmente es indispensable para una programación avanzada en BASIC.

5.3 El procesador Z80A

Construcción de la CPU

Los ordenadores MSX poseen una CPU Z80A (unidad central). Recordamos que la CPU puede ser denominada el "cerebro" del ordenador. Por lo tanto, la importancia del MPU (MPU : en inglés Micro Prozessing Unit = microprocesador) no es cuestionable.

En este capítulo vamos a tratar sobre la construcción y la función de los diferentes elementos contenidos en la CPU.

1. CU (CU: inglés Control Unit - unidad de control)
   Todos los procesos en un ordenador son controlados y gobernados por la CU.

2. Bus de control
   El bus de control es el "brazo largo" de la CU. A través de él, se gobiernan y se controlan los módulos exteriores de la CPU.

3. Apuntador de pila SP (SP: en inglés Stack Pointer)
   Con ayuda del SP se memorizan provisionalmente datos y direcciones de salto de subprogramas en la RAM. Como en el SP se memorizan direcciones, éste es un registro de 16 Bits.

4. Indicador de programa PC (PC: en inglés Program Counter - en realidad contador de programa)
   El PC señala hacia la dirección de memoria, en la cual se encuentra el comando a ejecutar.

5. Registro B hasta L (registro registrar)
   La CPU posee varios registros, en los cuales se pueden memorizar datos.

6. Flags (Flag: del Inglés flag - bandera; en este caso, indicador o señalización)
   Los Flags se utilizan como indicadores de determinados sucesos que se originan en la CPU con ciertas operaciones de cálculo. Pueden estar posicionados (bandera arriba) o no posicionados (bandera abajo).

7. Acumulador (del latín: accumulare : acumular)
   El acumulador es el registro más importante de la CPU. También se le puede denominar registro calculador.

8. ALU (ALU: en Inglés Arithmetical Logical Unit - Unidad aritmética lógica o unidad de cálculo).
   La ALU lleva a cabo todas la operaciones aritméticas y lógicas. Dependiendo del resultado de las operaciones, se influencian los Flags.

11. Registro de desplazamiento
    El registro de desplazamiento lleva a cabo todas las rutinas de rotación y de desplazamiento.

Como ya se ha mencionado en el punto 5, la CPU dispone de varios registros. Para la comprensión de sus funciones los hemos dividido en 5 grupos:

1. El acumulador
2. Los Flags
3. Los "seis enlazables" registros de 8 Bits
4. Los "cuatro inseparables" registros de 16 Bits
5. Registros de Interrupt/Refresh

El acumulador

El acumulador, o el registro A es el registro más importante del Z80. La mayoría de los comandos lógicos y aritméticos utilizan este registro. Al ejecutar un comando de comparación se compara siempre con el contenido del acumulador. Como todos los registros, excepto SP,PC,IX e IX, el registro A es un registro de 16 Bit.

Los Flags

El registro Flag o F, es de 8 Bits (como A,B,C,D,E,H y L). Pero tiene otras funciones diferentes a éstos. En el registro, Flag se utilizan los diferentes Bits como indicadores para determinados sucesos, que surgen en operaciones del ALU (unidad de cálculo). Los diferentes Bits del registro F tienen el siguiente significado:

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

- C - Acarreo Carry
- N - Sustracción
- P/V - Paridad/Desbordamiento
- H - Semiacarreo
- Z - Cero
- S - Signo

C Flag (Bit 0)
Si en una adición o sustracción se ha de efectuar un acarreo, entonces este Bit es posicionado, si no es reposicionado.

Flag N y H (Bit 1, Bit 4)
Estos flags son utilizados internamente por el Z80. Para nuestros fines no tienen ninguna importancia.

Flag P/V (Bit 2)

Este Flag tiene una función doble. Es posicionado cuando surge un acarreo (inglés: overflow), si no es reposicionado.

Además indica la paridad (P) de un Byte.

Z Flag (Bit 6)
Este flag es posicionado cuando el resultado de una resta es 0, si no es reposicionado. En una comparación, este Bit es posicionado si existe igualdad.

S Flag (Bit 7)
Si el resultado de una suma o resta es mayor que 127, se posiciona este Bit. Como veremos más tarde, en la aritmética de la CPU, números negativos representan a aquellos Bytes que son mayores que 127.

Los Bits 3 y 5 del Registro de Flags no se utilizan.

Los registros de 8 Bits "seis enlazables"

A este grupo pertenecen seis registros de 8 Bits:
B,C,D,E,H,L
Estos registros están capacitados para formar pares de registros, para poder representar un registro de 16 Bits. En C,E,L se memoriza el Low y en B,D,H el High Byte.

B/C (Byte Computer)
El registro B o el par de registros BC se utiliza frecuentemente como contador para, por ejemplo, bucles.

El par de registros DE queda para libre disposición.
Este par se utiliza frecuentemente para la memorización provisional de direcciones o datos.

H/L (High/Low)
El par de registros HL se utiliza frecuentemente para la memorización de direcciones.

La habituación a la denominación de los registros de esta forma es útil, ya que algunos comandos utilizan los registros de la forma descrita.

En principio también se pueden utilizar como contadores los registros L o E.
Una particularidad del Z80 es que todos los registros nombrados, existen duplicados con la misma función. Este segundo juego de registros está disponible. Pero sólo es utilizable un juego.

Los registros de 16 Bits "los cuatro inseparables"

A este grupo pertenecen los cuatro registros de 16 Bit:
SP,PC,IX,IY
El registro SP es un registro fijo de 16 Bit, o sea, no se puede dividir en dos registros de 8 Bits. El Stack Pointer señala hacia la dirección correspondiente en la memoria, en la cual están memorizadas direcciones de salto atrás o datos memorizados provisionalmente. La dirección se refiere a un espacio de memoria situada en un sector de la RAM denominado Stack o pila. La utilización del Stack para la memorización de datos es de la siguiente manera:
Al conectar el ordenador, el SP es posicionado a la dirección más alta en el Stack (&HF000). Si ahora quiere colocar un Byte sobre el Stack, entonces SP es disminuido automáticamente en 1, y este Byte es memorizado en la dirección que indica el SP. Por lo tanto, siempre señala hacia la última inscripción en la pila. Al "traer del Stack", el proceso se invierte. Primero se lee el Byte de la dirección sobre la cual señala el SP, el cual se aumenta después en 1. De esta forma es posible combinar subprogramas a voluntad.

El PC es un registro especial. Desde el programa no puede ser ni escrito ni modificado. El PC es administrado internamente e indica siempre hacia la dirección del comando actual.

Los registros IX/IY se utilizan principalmente para la memorización de direcciones o direcciones relativas. También estos registros pertenecen, como todos los indicados bajo 2.5, a los registros de 16 Bit.

En estos no es posible acceder por separado al High o Low Byte (como en BC,DE,HL). La utilización del registro de índices es similar a la del par de registros HL. La diferencia la veremos en el direccionado indexado.

Registros de Interrupt y Refresh

Estos dos registros estan asignados a la CU.

Registro I o de Interrupt
(del Inglés: Interrupt = Interrumpir)
Si surge un Interrupt, o sea, una interrupción del programa, entonces este registro de 8 Bit contiene la parte superior de la dirección hacia la cual se ha de desviar. La parte inferior es suministrada por el módulo del ordenador que ha iniciado el Interrupt.

Registro R o de Refresh
(del Inglés: resfresh = refrescar)
Este registro es utilizado como contador por el Hardware para refrescar, en espacios periódicos, el contenido de la memoria dinámica. De esta forma se quiere evitar que se pierdan informaciones memorizadas. Cargando continuamente el mismo contenido de memoria en un tiempo muy breve, se evita la pérdida de datos.

Una ejecución de un comando por la CPU sería de la siguiente manera:
El Byte en la dirección, hacia la cual señala el PC, es leído, y el PC es aumentado en uno, o sea, que señala hacia el siguiente Byte. El Byte leído es interpretado como comando. Después se leen posibles datos pertenecientes al comando (el Pc es aumentado nuevamente). Después sigue la ejecución del comando, y el proceso comienza de nuevo.

Después de haber conocido la CPU Z80A, vamos a tratar sobre la introducción de programas de máquina.

5.4 Introducción de programas máquina

Para poder comprobar inmediatamente los comandos del Z80, tenemos que saber cómo se introduce y memoriza un programa máquina desde el BASIC. Similarmente al BASIC, donde se asigna un número de línea a un comando, a cada comando de máquina se le asigna una dirección.

BASIC		Lenguaje máquina		
No. de línea	Comando	Dirección	Comando	Código
9	HL=HL+1	&HF009	INC HL	&H23
10	RETURN	&HF00A	RET	&HC9

- En el BASIC se asigna un número de línea a cada comando.

- En el lenguaje máquina, un comando está asignado a una dirección.

Por lo tanto, un programa de máquina es una serie de códigos de comandos, los cuales se encuentran en direcciones contiguas de la memoria.
Desde el BASIC tenemos la posibilidad de escribir los códigos en las direcciones determinadas por medio del comando >POKE<. La activación del programa máquina se efectúa con el comando >USR<. Antes hay que determinar la dirección de inicio del programa de máquina con >DEFUSR<. La dirección inicial es casi siempre la dirección de la memoria que contiene el primer código de máquina. Para que nuestro programa de máquina no sea sobreescrito por descuido, hemos de reservar su sector de memoria con el comando >CLEAR<. Reservaremos frecuentemente el sector de &HF000 hasta &HF37F por medio del comando >CLEAR 200, &HEFFF<. De esta forma disponemos de &H380 Bytes (corresponde más o menos a 1K) para programas máquina. Un programa BASIC típico para cargar programas máquina, tiene la siguiente estructura:

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

En las líneas DATA se encuentran los códigos que forman el propio programa de máquina. La dirección final (V=variable; esta abreviación la escribiremos en el futuro siempre detrás de palabras que sean variables) tiene que ser naturalmente mayor que &HEFFF, y la dirección inicial menor que &HF380. La activación de un programa cargado se efectúa con >X=USR1 (parámetro)<. Normalmente utilizaremos la dirección inicial &HF000. La dirección final (V) resulta de la dirección inicial (V) más la longitud del programa en Bytes, menos 1. La longitud de un programa es igual a la cantidad de inscripciones en las líneas DATA.

Con el MSX BASIC es posible, a través de los comandos >DEFUSR No.< y >USR No. (parámetro), definir hasta 10 diferentes llamamientos para programas máquina. Nosotros utilizaremos >USR1<. El parámetro entre paréntesis situado detrás del comando >USR< no tiene importancia para nosotros. Hay que escribir un número cualquiera o una variable.

Para la introducción de pequeños programas se puede utilizar el siguiente programa BASIC:

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

Vd. introduce directamente los códigos hexadecimales, y el programa efectuará el POKEado para Vd. El símbolo (&H) no es necesario que lo introduzca.

Después de haber conocido la introducción de programas máquina, vamos a ver los comandos del Z80.

Observación: En la explicación de los comandos trabajaremos con analogías de los comandos en BASIC. Para ello nos imaginamos un registro en BASIC como una variable con el mismo nombre (registro HL en lenguaje máquina corresponde a una variable HL en BASIC).

5.5 Los comandos

Los comandos del Z80 se pueden dividir en 5 grupos:

1. Transferencia de datos
2. Elaboración de datos y tests
3. Saltos
4. Comandos de gobierno
5. Entrada y salida

Transferencia de datos

Estos comandos se utilizan para el traspaso de datos. Los datos pueden ser traspasados de:

a) Registro a registro
   Esto corresponde a una asignación en el BASIC, como por ejemplo A=B o SP=HL. El comando máquina tiene el siguiente formato:
   LD A,B (LD-carga)

b) Registro a lugar de memoria
   Al traspasar datos de un registro a una posición de memoria, el comando BASIC >POKE dirección de memoria, variavariable por ejemplo >POKE &HF000,HL<, corresponde al comando de lenguaje máquina LD (&HF000),HL.

c) Posición de memoria a registro
   El traspaso de datos de la memoria a un registro, por ejemplo (LDH,&Hf005), corresponde al comando BASIC:
   >H=PEEK (&HF005)<.

Elaboración de datos y tests

Los comandos para la elaboración de datos pueden subdividirse en 5 grupos:

- operaciones aritméticas (por ejemplo Adición, Sustracción)
- operaciones lógicas (por ejemplo AND, OR)
- Comandos de contar (INCrease = aumentar. DECrease = reducir).
- Manipulación de Bits (SET, RESet)
- Cambio y desplazamiento de Bits (Rotate = rotar, shift= desplazar)

Al ejecutar estos comandos se modifican los contenidos de los registros y de las memorias (en el RAM). Muchos comandos son similares a los del BASIC:

Assembler	BASIC
SUB A,B (sustracción)	A=A-B
ADD HL,BC (adición)	HL=HL+BC
AND C	A=A AND C
OR HL	A=A OR PEEK(HL)

Se comprueban Bits aislados en los registros o en las posiciones de memoria (Comando BIT), o se comparan contenidos de registros o memorias con el acumulador (Comando CP= compare). Según el resultado de estos tests, los Flags correspondientes del ALU se posicionan o se borran en el registro F.

Saltos

Con estos comandos es posible introducir bifurcaciones en el programa máquina.
Se diferencian tres tipos de salto:

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

Un salto se considera condicionado cuando la decisión sobre si se ha de saltar o no, depende del estado de un Flag. Un salto condicionado, o sea, uno que depende del estado de un

Flag es , por ejemplo, JR NZ,&HF003.

Analogías:

Assembler	BASIC
JP	GOTO
CALL	GOSUB
RET	RETURN
JR	--- (similar al bucle FOR-NEXT)

Comandos de mando

Con estos comandos se puede, por ejemplo, interrumpir un programa. Con estos comandos también es posible el gobierno de interrupts.

Comandos de entrada/salida (Input/Output)

Los comandos E/S sirven para la comunicación con los aparatos de entrada y salida. Dependiendo de la dirección en cuestión del Port E/S, con estos comandos se pueden solucionar los más diversos problemas. Por medio de estos comandos se tratan frecuentemente:

PPI (Programable Peripheral Interface)
PSG (Programable Sound Generator)
VDP (Vídeo Display Controller) y con él los periféricos: teclado, altavoces, monitor, impresora y el cassette.

5.6 Un ejemplo de programación en lenguaje máquina

Vamos a desarrollar con Vd. paso a paso un programa máquina. Este programa tendrá la misma finalidad que el programa BASIC descrito en el capítulo sobre gráficas para la representación inversa de caracteres.
La finalidad de este programa es invertir los primeros 128 caracteres del generador de tal forma, que éstos existan inversos y normales. Este problema vamos a solucionarlo primeramente en BASIC, pero de un forma que se asemeja bastante a la programación en lenguaje máquina.
Para la solución del problema necesitamos las siguientes informaciones:

Dirección básica del generador de caracteres:

La dirección básica la obtenemos con >PRINT BASE(2)<.
En forma hexadecimal es &H0800.

La dirección inicial del primer carácter que ha de representarse inverso es (espacio libre):

Como queremos tratar todos los caracteres, a partir del espacio libre hasta el código 128, necesitamos su dirección. Como Vd. sabe por el capítulo de gráficas, la posición dentro del generador de caracteres se calcula con:

Posición = ASCII*8

El código ASCII del espacio libre es 32. El cálculo es:

32*8=256

Este valor se suma ahora a la dirección inicial, para obtener la dirección en la cual se encuentra el espacio libre.

2048+256=2304=&H900

De esta forma sabemos la dirección actual de memoria a partir de la cual tenemos que leer. Esta dirección la memorizamos en una variable (HL).

```
HL=2304 o HL=&H900
```

Dirección de la memoria en la cual se ha de memorizar el primer carácter inverso (espacio libre invertido).

Las matrices invertidas han de tener los códigos del carácter normal+128. Por lo tanto, el espacio libre invertido tiene el código 128+32=160. De esta forma, la dirección inicial del primer espacio libre es:

```
2048+160*8=3328 o &HD00
```

El valor &HC00 también se memoriza en una variable (DE):

```
DE = 3328 o DE = &HD00
```

Contador para la cantidad de los 96 caracteres a transformar:

Se han de invertir los caracteres con los códigos 32-127. Estos son 96 caracteres.
Como cada carácter esta representado por 8 Bits, tenemos que recorrer el bucle 96*8=768 veces para leer y escribir. Este valor lo memorizamos en otra variable más (BC).

```
BC = 768 o BC = &H300
```

Por lo tanto, las primeras líneas de nuestro programa son:

```
10 HL = &H900
20 DE = &HD00
30 BC = &H300
```

Programación máquina:

En la programación máquina, los parámetros antes calculados no se memorizan en variables sino en registros. Lamentablemente, para este lenguaje de programación sólo son aptos unos pocos registros, los cuales son: HL, DE y BC. En ellos podemos memorizar los valores antes calculados.
El comando correspondiente es:

```
LD registro, valor
```

(Ld= Inglés load = cargar)

Carga el registro X con el valor Y.
En el lenguaje Assembler nuestras líneas tienen el siguiente formato:

```
10 LD   HL,&H900
20 LD   DE,&HD00
30 LD   BC,&H300
```

Ahora seguramente se preguntará Vd., ¿qué es lo que esto tiene en común con los valores en las líneas DATA. Como se ha dicho, un programa máquina es una sucesión de números. Para la traducción de los comandos Assembler a los códigos máquina existen dos posibilidades. El método más confortable es la traducción automática:
El Assembler traduce el lenguaje Assembler, más comprensible para el programador, a los códigos. Pero si no existe ningún Assembler, se puede transformar el lenguaje Assembler a códigos máquina por medio de tablas relativamente simples. El programa máquina se memoriza en forma de códigos en líneas DATA.

por ejemplo: línea 10 del programa máquina:

```
10 LD  HL,&H900        resultan los códigos 21 00 09
```

La línea DATA correspondiente es:

```
100 DATA &H21,&H00,&H09
```

En ello, el código 21 representa al comando Assembler LD HL, valor.
Si leemos los dos últimos valores de la línea DATA de la forma convencional, primero el Byte bajo y después el Byte alto, obtendremos en valor inicial del juego estándar de caracteres.

```
LB HB HB LB                    LB = Low BYte
00 09=09 00 = &H0900=2304      HB = High Byte
```

El mismo procedimiento se utiliza para las dos otras líneas:

```
20 LD DE,&HD00 resulta 11 00 0D
```

En ello, el código 11 representa: LD DE, valor
El código 000D para &HD00

```
30 LD BC,&H300 resulta 01 00 03
```

En ello, el código 01 representa: LD BC, valor
El código 0003 para &H300

Para llevar a cabo el siguiente paso, volvemos al BASIC:

Con >A=VPEEK (HL)< leemos una fila de puntos en forma de un Byte de los caracteres estándar, y los escribimos, después de haberlos invertido con >A=A OR 255<, con >VPOKE DE,A< en la dirección prevista para los caracteres inversos.

La siguientes líneas BASIC son:

```
40 A=VPEEK(HL) : REM leer
50 A=A XOR 255 : REM invertido
60 VPOKE DE,A  : REM escribir
```

En el lenguaje máquina necesitamos dos rutinas de máquina para leer y escribir una posición de memoria. La rutinas se llaman con

```
CALL dirección
```

Naturalmente han de conocerse las direcciones y funciones de estas rutinas. En el capítulo rutinas del sistema podrá encontrar con seguridad algunas de estas direcciones y funciones. Las siguientes líneas de programa:

```
40 CALL &H004A corresponde a los códigos CD004A
50 XOR  255    corresponde a los códigos EEFF
60 EX   DE,HL  corresponde a los códigos EB
70 CALL &H004D corresponde a los códigos CD004D
80 EX   DE,HL  corresponde a los códigos EB
```

La línea 40 llama la rutina que lee un Byte de la VRAM. La dirección del Byte a leer es determinada por el contenido del registro HL. La rutina memoriza el valor leído en el acumulador. Por lo tanto, en la próxima línea (línea 50) se puede disponer en el acumulador del valor que fue leído en el BASIC por medio de >VPEEK (HL)<.

La línea 50 en lenguaje máquina es casi idéntica a la línea BASIC. Representa un Byte inverso, lo cual está en lugar de una fila de ocho puntos de la definición de caracteres.

Línea 60: Como para la rutina llamada en la siguiente línea también hay que traspasar al registro HL la dirección del Byte a escribir, y nosotros queremos escribir en la dirección final, la cual está memorizada en DE, hay que cambiar DE y HL anteriormente. Esto lo lleva a cabo el comando EX (como >SWAP<).

En la línea 70 se activa la rutina para la escritura de un Byte en la VRAM. De nuevo, HL recibe la dirección de la posición de memoria a escribir. El valor que se ha de memorizar en este lugar, ha de estar contenido en el acumulador (A) al efectuar el llamamiento. Este es el caso en nuestro programa.

Todos los comandos lógicos conciernen siempre al acumulador, o sea, XOR 255 invierte el contenido del mismo, aun cuando A esté expresamente indicada.

Línea 80: Para volver al estado original para el siguiente llamamiento de la rutina VPEEK (Línea 40), se cambia nuevamente con EX.

Los siguientes pasos de programa incrementan más uno los valores memorizados en HL y DE. Con esto se consigue que, una detrás de otra, se lean y se escriban todas las posiciones de memoria importantes para nuestro programa.
Pero como sólo queremos modificar 96 caracteres, ha de existir un contador que cuente hacia 0 y que finalice el programa con una comparación, cuando se hayan modificado todos los caracteres. Este contador es BC. BC se disminuye cada vez menos 1.
La última línea representa la comparación, si BC=0.

```
 70 HL=HL+1 : REM posición de memoria siguiente
 80 DE=DE+1 : REM posición de memoria siguiente
 90 BC=BC-1 : REM siguiente carácter
100 IF BC<>0 THEN : REM si se han leído todos los caracte-
res, final
```

Lenguaje máquina:

En el lenguaje máquina, el comando

```
INC registro
```

aumenta más 1 el contenido del registro. INC significa en inglés increase = incrementar.
El comando

```
DEC registro
```

realiza lo contrario. Reduce en uno el contenido de un registro. DEC significa en inglés decrease = disminuir. Por lo tanto, las siguientes líneas del programa máquina tienen el siguiente formato:

```
 90 INC HL corresponde al código 23
100 INC DE corresponde al código 13
110 DEC BC corresponde al código 0B
```

Los siguientes comandos realizan la comparación y el bucle en la línea 100 de nuestro programa BASIC.

```
LD A,B
```

Carga el acumulador (A) con el valor B, o sea, con el High Byte del registro BC.

```
OR C
```

OR C se refiere, como todos los comandos lógicos (ver arriba), al acumulador. En realidad OR significa:
Une el contenido del acumulador con el contenido del registro C por medio de "o" y memoriza el resultado nuevamente en A:

```
A=A OR C
```

El contenido original de A es, a causa del comando anterior, el contenido de B. Entonces en realidad se unen B y C por medio de "o". El rodeo sobre el acumulador es necesario, ya que los comandos lógicos utilizan siempre el acumulador.
El resultado de la combinación lógica "OR" tiene la propiedad de estar activados todos aquellos bits, que estaban activados en cualquiera de los dos valores de la combinación. Esto significa, que el resultado de la unión es cero, cuando ambos valores unidos (aquí B y C) son cero. Por lo tanto, si el acumulador es 0 después de OR C, entonces tambien BC=0, y el bucle se ha de finalizar.
Los comandos aritméticos/lógicos influyen todos en el Flag Z. Este Flag indica si el resultado de un comando era 0 (Z=zero) o no (NZ=non zero). Con esto puede seguir un salto condicionado:

JR NZ, dirección de salto o JR NZ, distancia

(JR = Jump Relativ = salto relativo)
(NZ = Non Zero = no cero)

Esta orden de salto significa tanto como "salta si no es cero".
La dirección de salto se indica por la diferencia de la dirección del comando siguiente al comando de salto, hasta la dirección final (quiere decir, 256 - valor absoluto). No vamos a detallar el cálculo de este valor. Para nuestro caso el Byte de la distancia de salto es &HEF. Si la condición no se cumple, y por lo tanto se posiciona un Flag Z, se elabora el siguiente comando.
Este es el comando RET (como RETURN) que origina el retorno al BASIC.

```
120 LD A,B          corresponde al código 78
130 OR C            corresponde al código B1
140 LR NZ,$-17      corresponde al código 20EF
```

El listado completo del programa BASIC:

```
 10 HL=2304              o   10 HL=&H900
 20 DE=3328              o   20 DE=&HD00
 30 BC=778               o   30 BC=&H300
 40 A=VPEEK(HL)
 50 A=A XOR 255
 60 VPOKE DE,A
 70 HL=HL+1
 80 DE=DE+1
 90 BC=BC-1
100 IF BC<>0 THEN 40
```

Y ahora el cargador BASIC, el cual carga el programa máquina con la misma finalidad que el anterior:

```
  5 SCREEN 0
 10 CLEAR 200, &HEFFF
 20 FOR I=&HF000 TO &HF01A
 30 READ A$
 40 POKE I,VAL("&H"+A$)
 60 NEXT
 70 DEF USR1=&HF000
 80 LOCATE 0,7:END
 90 DATA 21,00,08,11,00,0C,01,00
100 DATA 04,CD,4A,00,EE,FF,EB,CD
110 DATA 4D,00,EB,,13,0B,78,B1
120 DATA 20,EF,C9
```

Inicie el programa con >RUN< y llámelo con >X=USR1(1)<.

El listado Assembler es el siguiente:

```
Direc. Código   No. línea   Comando Assembler; Comentario

F000 210009         10      LD    HL,&H900 ; dirección ini-
cial generador de caracteres
F003 11000D         20      LD    DE,&HD00;
F006 010003         30      LD    BC,&H300;
F009 CD004A         40      CALL  &H004A ;Leer Byte de direc-
ción HL de la VRAM
F00C EEFF           50      XOR   255 ; invertido
F00E EB             60      EX    DE,HL ; dirección final
después de HL
F00F CD004D         70      CALL  &H004D ; escritura de un
Byte en la dirección HL en la VRAM
F013 EB             80      EX    DE,HL ; reponer el estado
original
F014 23             90      INC   HL ; incrementar HL más 1
F015 13             100     INC   DE ; incrementar DE más 1
F016 0B             110     DEC   BC ; disminuir BC menos 1
F017 78             120     LD    A,B ; cargar Byte alto de
BC en el acumulador
F018 B1             130     OR    C ; combinar el Byte bajo
BC con OR
F019 20EF           140     JR    NZ,$-17 ; si no es cero,
adelante
F01A C9             150     RET ; de otro modo volver al
BASIC
```

Explicación del listado Assembler

Dirección:

Bajo esta columna están escritas las direcciones de las memorias, en las cuales han de memorizarse los códigos respectivos del programa máquina.
Las direcciones están escritas en el sistema hexadecimal, al igual que los códigos.

Código:

Aquí se encuentran los códigos de los diferentes comandos, igualmente en el sistema hexadecimal. Dos cifras juntas resultan un Byte. La cantidad de Bytes de un comando se puede determinar así fácilmente.

Por ejemplo: línea 10 y 20

Dirección Código

F000 210009

F003 11000D

La cantidad de códigos en la línea 10 son tres, que son: 21=Byte 1, 00=Byte 2 y 09=Byte 3. Por lo tanto, este comando necesita tres Bytes. La dirección de la siguiente línea nos indica lo dicho (F003). Los códigos también los encontrará en las líneas DATA del programa BASIC.

No. de línea:

Aquí se encuentra el número de línea, también en forma hexadecimal.

Comando Assembler:

Bajo este término se encuentran los Mnemonics (palabras comando) con los operandos correspondientes (por ejemplo direcciones, constantes, etc.)
por ejemplo: CALL &H0072

CALL = Mnemónic
&H0072 = Operando (en este caso una dirección)

Comentario:

Aquí se pueden incluir explicaciones para mejor comprensión del programa máquina.

5.7 El Monitor

Un medio indispensable para la formación de programas máquina, es el así denominado programa de monitor. Un monitor se utiliza para la visualización y modificación de contenidos de memoria. Los sectores de memoria pueden ser memorizados en cassette, o cargados de la misma.
Los monitores profesionales disponen de otras funciones más:

- Disassembler, con el cual se pueden traducir al lenguaje Assembler los contenidos de memoria que representan programas.

- Función de búsqueda, con la cual se pueden buscar determinadas series de Bytes en la memoria.

- Función de inicio, con la cual se pueden llamar programas máquinas para tests.

- Función de desplazamiento que puede desplazar y/o copiar contenidos de memoria.

- Miniassembler, con cuya ayuda se pueden traducir comandos Assembler a sus códigos.

En el siguiente programa se incluyen algunas de estas posibilidades.
Después del inicio del programa, una estrella indica que se espera una introducción. Todas las posibilidades del monitor han de activarse por medio de la introducción de una letra:

m - monitor: indicar sector de memoria
S - Save
L - Load
a - modificación de contenidos de memoria
g - iniciar programa máquina
r - seleccionar Slot

Las funciones en detalle:

Monitor (m)

m llama a la función base del programa monitor, que es la visualización de los contenidos de memoria. Inmediatamente después de la m hay que introducir las direcciones inicial y final del sector que ha de visualizarse. La introducción de cada número se confirma con RETURN. La edición de los contenidos de memoria se hace en el mismo formato que un HEX DUMP (edición de los contenidos de memoria en forma hexadecimal).

En primer lugar está la dirección actual, después los valores que estén en ésta y las 7 direcciones siguientes, todos en forma hexadecimal.
Al final de la línea se encuentra la representación de caracteres de los ocho últimos códigos, lo cual se interpreta por los códigos ASCII.
Códigos que sean mayor que 127, se reducen a 128. Los caracteres de mando se editan en forma de puntos.

Save (S)

Aquí hay que indicar las direcciones inicial y final del sector a memorizar. Después se ha de introducir el nombre bajo el cual se quiere memorizar.

Load(L)

Detrás de la L ha de indicarse el nombre de los datos a cargar.

LLamada del programa máquina (g)

Después de la introducción de la letra g se indica la dirección inicial de programa a llamar.

Modificación de contenidos de memoria (a)

Se ha de introducir la dirección del primer Byte a modificar. Después se edita siempre la dirección actual y Vd. ha de introducir el valor deseado.
e finaliza la introducción.

Elección de Slot (r)

El número del Slot deseado ha de ser introducido (desde 0 hasta 3). A partir de entonces se indican los contenidos del Slot introducido. El sector a partir de &HF000 sólo puede leerse con Slot 2.

```
10 CLEAR 200,&HEFFF:MAXFILES=2
20 OPEN "CRT:" FOR OUTPUT AS #1
30 OPEN "LPT:" FOR OUTPUT AS #2
40 DEFSNG A-Z
50 KN=1:REM Numero de canal
60 AN=6
70 FOR I=1 TO AN:READ BF$(I):NEXT
80 DATA m,a,s,l,g,r
90 FOR I=&HF000 TO &HF022:REM Leer Mapro para Slot Read
100 READ A$:POKE I,VAL("&H"+A$):NEXT
110 DATA CD,8A,2F,EB,C1,E1,E5,C5,D5,CF,2C,CD,1C,52,FE,04
120 DATA D2,5A,47,E3,CD,0C,00,CD,CF,4F,E1,C1,D1,E5,C5,21
130 DATA F6,F7,C9
140 DEFUSR1=&HF000
150 SCREEN 0:WIDTH 38
160 LOCATE 10,2:PRINT"M o n i t o r"
170 ON KEY GOSUB 880,920,960,1000
180 KEY 1,"I.on":KEY(1) ON
190 KEY 2,"Help":KEY(2) ON
200 KEY 3,"Fin ":KEY(3) ON
210 KEY 4,"Stop":KEY(4) ON
220 LOCATE 0,5
230 Y=CSRLIN:FOR I=1 TO AN:PRINT BF$(I);" : ":NEXT
240 LOCATE ,Y
250 LOCATE 5:PRINT"Indicar sector de memoria"
260 LOCATE 5:PRINT"Modificar sector de memoria"
270 LOCATE 5:PRINT"Grabar sector de memoria"
280 LOCATE 5:PRINT"Cargar sector de memoria"
290 LOCATE 5:PRINT"Ejecutar programa maquina"
300 LOCATE 5:PRINT"Seleccionar Slots"
310 PRINT
320 PRINT"Key 1 : Con./desc. impresion"
330 PRINT"Key 2 : Indicar los comandos"
340 PRINT"Key 3 : Fin del programa"
350 PRINT"Key 4 : Parar proceso actual"
360 PRINT
370 PRINT:LOCATE 0,CSRLIN-1:PRINT"* ";
380 A$=INKEY$:IF A$="" THEN 380
```

```
390 PRINTA$;" ";
400 FOR I=1 TO AN:IF (ASC(A$)OR2^5)<>ASC(BF$(I)) THEN NEXT
410 ON I GOSUB 440,610,690,740,780,830,870
420 GOTO 370
430 A$=""
440 REM Monitor
450 GOSUB 1040:REM Tomar entradas
460 IF (SN<>2) AND ((E-2^16*(E<0))>=49152!) THEN RETURN
470 FOR I=S TO E STEP 8
480 PRINT#KN,RIGHT$("000"+HEX$(I),4);" ";
490 A$=""
500 FOR J=0 TO 7
510 REM Leer Byte
520 B=USR1(I+J),SN
530 PRINT#KN,RIGHT$("0"+HEX$(B),2);" ";
540 C=B AND 127
550 IF C<32 OR C=127 THEN C=46
560 A$=A$+CHR$(C)
570 NEXT J
580 PRINT#KN,A$
590 NEXT I
600 RETURN
610 REM Cambiar
620 INPUT S
630 PRINT"&H";RIGHT$("000"+HEX$(S),4);" : ";
640 INPUT W$
650 IF (ASC(W$)OR2^5)=ASC("e") THEN RETURN
660 W=VAL(W$)
670 POKE S,W
680 S=S+1:GOTO 630
690 REM Grabar
700 GOSUB 1040
710 INPUT"Nombre :";N$
720 BSAVE "CAS:"+N$,S,E
730 RETURN
740 REM Cargar
750 INPUT"Nombre :";N$
```

```
760 BLOAD "CAS:"+N$
770 RETURN
780 REM Inicio de programa
790 INPUT S
800 DEFUSR2=S
810 S=USR2(1)
820 RETURN
830 REM Seleccion de Slot
840 INPUT "N. de Slot : ";SN
850 IF SN>3 THEN 840
860 RETURN
870 BEEP:LOCATE 0:PRINT"     ";:RETURN
880 REM Impresora con./desc.
890 IF KN=1 THEN KN=2 ELSE KN=1
900 IF KN=1 THEN KEY 1,"I.on" ELSE KEY 1,"I.out"
910 RETURN
920 REM Help
930 FOR I=1 TO AN+4:PRINTCHR$(13):NEXT
940 LOCATE ,CSRLIN-AN-3
950 RETURN 230
960 REM Fin
970 DEFUSR1=&H3E:X=USR1(1):REM Ocupacion estandar de Key
980 CLS
990 END
1000 REM Stop
1010 PRINT
1020 BEEP
1030 RETURN 370
1040 REM Introduccion inicio/fin
1050 Y=CSRLIN
1060 INPUT S
1070 LOCATE 14,Y:INPUT E
1080 RETURN
```

Descripción del programa:

La gran parte del monitor está documentado por líneas REM. El programa máquina leído en las líneas 90 hasta 120 transforma el comando >USR< en un comando >PEEK<, en el cual se puede indicar adicionalmente el número de Slot. La lectura de las direcciones a partir de &HC000 sólo es posible en el Slot no. 2. El formato del comando es:

USR1 (dirección), número de Slot

El listado Assembler para la ampliación lo encontrará en los Restos.

El bucle principal del programa lo forman las líneas 370 hasta 420. Son posibles otras introducciones por medio de Interrupt con las teclas de función.
También es de importancia que todos los números se pueden introducir tanto en forma hexadecimal como en forma decimal.

6.1 La utilización de las rutinas del sistema

El siguiente capítulo quiere exponer las posibilidades de uso de rutinas en lenguaje máquina. A continuación de este texto encontrará una lista con las rutinas más importantes. Muchas de estas rutinas se pueden utilizar de manera simple, mientras que otras requieren buenos conocimientos en la programación con lenguaje máquina. En este sentido queremos recomendar el ''libro para lenguje máquina para MSX'', el cual ofrece una introducción sencilla al lenguaje máquina, y para avanzados ofrece muchas indicaciones y consejos para la programación en este lenguaje.
En el caso más fácil, una rutina de sistema puede ser llamada directamente:

Ejemplo: Esperar a una pulsación de tecla

La dirección de la rutina es &H009F. Como no hay que acarrear parámetros, sólo hay que ocupar un comando >USR< con la dirección inicial. O sea:

```
DEFUSR1 = &H009F
```

Si ahora se ha de esperar a la pulsación de una tecla, sólo hay que llamar la rutina:

```
X = USR1(1)
```

Si hay que acarrear parámetros a una rutina, es necesario un miniprograma máquina, el cual ha de leer el valor.

Ejemplo: LD (HL), E con elección de Slot

Esta rutina es necesaria para, por ejemplo, escribir valores en los sectores escondidos de la RAM (&H0 hasta &H7FFF) sin tener que seleccionar los mismos por medio de >OUT<.

Se han de traspasar tres valores diferentes:

HL - Dirección
E - Valor
A - Número de Slot

Esto se efectúa con el miniprograma Assembler:

```
LD HL, dirección
LD E, valor
LD A, Slot
CALL &H0014
RET
```

Partiendo de las listas de comandos de los comandos de carga del Z80, Vd. puede determinar fácilmente los códigos.

```
LD HL,nn        código: 21 n  n
LD E,n          código: 1E n
LD A,Slot       código: 3E n
CALL &H0014     código: CD 14 00
RET             código: C9
```

El código del comando CALL es &HCD y el del comando RET es &HC9.
El cargador BASIC tiene la siguiente forma:

```
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF00A:READ A: POKE I,A:NEXT
30 DEFUSR1=&HF000
40 DATA &H21,0,0&H1E,0,&H3E,0&HCD,&H14,&H00,&HC9
```

Los valores a traspasar (dirección, valor, Slot) son indicados primeramente por ceros.
Entonces la llamada de las rutinas es la siguiente:

AD contiene la dirección
WE contiene el valor
SL contiene el Slot

```
100 POKE &HF001,AD-INT(AD/256)*256
110 POKE &HF002,INT(AD/256)
120 POKE &HF004,WE
130 POKE &HF006,SL
140 X=USR1(1)
```

Por medio de los comandos >POKE< se escriben los valores actuales en los lugares correctos en el programa máquina. Al llamar al programa máquina son cargados y acarreados de acuerdo con los comandos Assembler.

SI un valor, que se ha determinado por medio de una rutina de máquina, ha de ser pasado al BASIC, se ha de memorizar en una dirección acordada previamente.

La rutina STRING devuelve 0 ó 255 al acumulador. Después de llamar la rutina, el acumulador se escribe, por ejemplo, en la memoria &HF200 (en el sector reservado por >CLEAR<). Para ello se utiliza el comando:

```
LD (dirección),A
```

el cual tiene el código 32. Con la dirección &HF200 de traspaso acordada:

```
LD (&HF200),A   código: 32 00 F2
```

Después se cambia al BASIC por medio de RET, donde con >PEEK(&HF200)< se puede cargar el valor determinado.

6.2 Las rutinas

Dirección &H0000: RESET

La llamada de esta rutina origina lo mismo que un conectado/desconectado del ordenador, o la pulsación de la tecla RESET.

Dirección &H0008: RST &H08-Test del Byte siguiente

Se comprueba si el Byte escrito en la dirección HL es igual al Byte escrito detrás del comando RST &H08. Si no es igual, se edita un "Syntax error". Si no, se bifurca después a la rutina RST &H10 después de la dirección &H4666.

Dirección &H000C: LD A,(HL) con elección de Slot

Al acarrear, A contiene el número de Slot deseado (0-3). Como resultado se obtiene en el acumulador y en el registro E el valor inscrito en la dirección HL del Slot deseado.

Dirección &H0014: LD (HL),E con elección de Slot

Al acarrear, A contiene el número de Slot, HL la dirección y E el valor a inscribir.

Dirección &H0018: RST &H18 - edición de caracteres

Edita el carácter contenido en el acumulador sobre el aparato actual. Normalmente es la pantalla. Cargando la memoria &HF416 con un valor <> 0 se elige la impresora.

Dirección &H0020: RST &H20 - comparación de HL con DE

El contenido del registro DE es restado del registro HL y, según el resultado, se modifican los Flags. HL y DE no (!) se modifican.

Dirección &H0028: RST &H28 - test del tipo de variable

Determina el tipo de variable actual, con lo cual después de la devolución vale:

Carry = 0	(NC)	Tipo 8	DEL
Carry = 1	(C)	Tipo 2,3 ó 4 que son	
Sign Flag = 1	(M)	Tipo 2	INT
Zero Flag = 1	(Z)	Tipo 3	String
Sign Flag = 0	(P)	Tipo 4	SNG

Dirección &H0038: RST &H38 - Salto de Interrupt con INT MODE

Este es el punto de salto para la rutina que es llamada 50 veces por segundo por el Interrupt estándar.

Dirección &H003E: Ocupación estándar de las teclas

Esta rutina ocupa las teclas de función con sus palabras originales, existentes al conectar el ordenador.

Dirección &H0047: Registro VDP Write

El registro VDP con el número indicado en el registro C es escrito con el valor contenido en el registro B, o sea VDP(C)=B.

Dirección &H004A: RAM de vídeo - Read

El Byte situado en la dirección HL de la RAM de vídeo, es cargado en el acumulador por medio de esta rutina.
BASIC: A = VPEEK(HL).

Dirección &H004D: RAM de vídeo - Write

El valor del acumulador es memorizado por medio de esta rutina en la dirección de la RAM de vídeo.
BASIC: VPOKE HL,A

Dirección &H005F: Select SCREEN

Aquí, el modo SCREEN es cambiado al valor (0,1,2 o 3) contenido en el acumulador.
BASIC: SCREEN A

Dirección &H0093: PSG registro Write

Esta rutina escribe el valor contenido en el registro E en el registro PSG con el número A. La programación del PSG en lenguaje máquina es muy importante para la producción de sonidos complejos. Esta rutina correponde al comando BASIC >SOUND A,E<.

Dirección &H0096: PSG registro Sound

Después de llamar esta rutina, el acumulador contiene el valor del registro PSG con el número que el acumulador recibe antes del llamamiento de esta rutina.

Dirección &H009F: Espera a pulsar tecla

Esta rutina espera hasta que se pulse una tecla. El código ASCII correspondiente a esta tecla es registrado y me-

morizado en el acumulador. Después se realiza el salto hacia atrás.

Dirección &H00AE: Introducción hasta CR a partir del comienzo de línea

Con esta rutina Vd. obtiene una línea completa introducida. Como la línea de introducción puede tener una longitud de hasta 255 caracteres, ha de memorizarse en la RAM. Esto se efectúa a partir de la dirección &HF55E. Por lo tanto, en el sector &HF55E hasta &HF65D se ha memorizado la última introducción. Al volver de esta rutina, HL contiene la dirección inicial, menos uno, de este buffer de introducción.

Dirección &H00C0: BEEP

Edita un BEEP.
BASIC: BEEP

Dirección &H00C3: Clear Screen

Borra todo el sector de visualización de la pantalla en todos los modos.
BASIC: CLS

Dirección &H00C6: Posicionar cursor

Por medio de esta rutina, el cursor es posicionado en la posición HL. En este sentido vale:

línea H
columna L

BASIC: LOCATE L,H

Dirección &H00CC: KEY OFF

Desconecta la indicación KEY.

Dirección &H00CF: KEY ON

Conecta la indicación KEY.

Dirección &H00D5: Consulta STICK

Esta rutina suministra, después del traspaso de A en la forma convencional al acumulador, el valor de la dirección respectiva (ver manual).

0- teclado
1- joystick 1
2- joystick 2

BASIC: A= STICK(A)

Aunque esta rutina corresponde exactamente al BASIC, es muy importante ya que, justamente en juegos, el joystick es muy importante, para una consulta rápida.

Dirección &H00DB: Consulta STRIG

Esta rutina suministra después del traspaso A (ver arriba):
0 en el acumulador cuando el disparador (SPACE) no está pulsado.
255 en el acumulador cuando el disparador (SPACE) está pulsado.

Dirección &H0132: CAP Conectado/Desconectado

Conecta o desconecta CAP.
A = 0 - CAP conectado
A<>0 - CAP desconectado

Dirección &H0135: Software Sound

Esta rutina posiciona el conductor EXT SOUND, que está conectado directamente al altavoz, en low o high:

A = 0 - Conductor en low, B<>0 - Conductor en high

Una ejecución rápida de conectar y desconectar origina un tono. De otra forma es audible un clic al llamar la rutina.

Dirección &H013E: Estado VDP - Read

Esta rutina devuelve al acumulador el valor actual del estado del registro VDP.

Dirección &H2F8A: CINT

Convierte FAC al formato "Integer" y comprueba el tamaño. Esta rutina edita en HL el valor, y en el acumulador la cifra característica (=2).

Dirección &H2FB2: CSNG

Convierte FAC al formato "exactitud simple".

Dirección &H2F99: Copia HL en el FAC

Carga el valor del registro HL en el FAC y carga la cifra característica con 2.

Dirección &H303A: CDBL

Convierte FAC al formato "exactitud doble".

Dirección &H4055: Syntax Error (ERR=2)

Dirección &H4067: Overflow (ERR=6)

Dirección &H406A: Missing Operand (ERR=24)

Dirección &H406D: Typ Mismatch (ERR=13)

Dirección &H406F: Indicación de error

Indica el error correspondiente al número traspasado al registro E.

Dirección &H400B: GET PAR

Rutina de Interrupt BASIC, la cual lee una dirección de 16 Bits y un valor de 8 Bits a continuación, separados

DE = valor 8 Bit
BC = valor 16 Bit

Dirección &H521C: GET BYTE

Lee un valor de 16 Bit, el cual es devuelto al acumulador y al registro E.

Dirección &H542F: GET ADR

Lee un valor de 16 Bit que es devuelto al registro DE.

Dirección &H5439: GET ADR entre paréntesis

Lee un valor de 16 Bit escrito entre paréntesis.

Dirección &H0103: Tamaño de las coordenadas de punto texto

Al traspasar las coordenadas para >SCREEN 2 o 3<, en lo cual la coordenada X es traspasada al registro DE, esta rutina comprueba si se trata de coordenadas admisibles.

C=0 : coordenadas no permitidas
C=1 : coordenadas permitidas

Si >SCREEN 3< (Multicolor) está conectado, las coordenadas se dividen por 4, ya que en este modo un punto corresponde a 4*4 puntos en el modo >SCREEN 2<.

Dirección &H0111: Cálculo de la dirección de punto

Al traspasar coordenadas X a C y coordenadas Y a E, se calcula la dirección de punto y es traspasada al registro HL y a la dirección &HF92A/B. El valor del Bit, que corresponde al punto a posicionar, es memorizado en la dirección &HF02C.

Dirección &H0114: Leer dirección de punto

La dirección del último punto calculado es cargada en el registro HL y el valor del Bit del punto es cargado en el acumulador.

Dirección &H0117: Escribir dirección de punto

El valor actual HL es memorizado como dirección del punto (&H92A/B) y el contenido del acumulador como valor del Bit del punto (&HA92??)

Dirección &H011A: Test código de color

Se comprueba si el código de color contenido en el acumulador es admisible (<16). Si esto es así, C es reposicionado y el color se memoriza como color actual de punto en la dirección &HF3F2.

6.3 Modificaciones del sistema

Finalmente vamos a exponer las posibilidades de modificar el sistema operativo o el interpretador BASIC. Los programadores que han elaborado el sistemo aperativo MSX, han previsto una técnica que en castellano se podría decir "patchear" (inglés patch: tapar, poner parche).

El problema general en una modificación del sistema es que la memoria ROM no es modificable. Solo es posible la intervención en partes de la RAM. Para que sea posible el acceso a una mayor parte de sectores, se bifurca desde las rutinas importantes de la ROM, antes de la apropia rutina, hacia un subprograma en la RAM. Si el sistema se encuentra en el estado original, todas las direcciones hacia las cuales se ha de saltar, están cargadas con el código &HC9, el código para RET. Esto significa, que la activación de esta dirección no tiene ninguna influencia, y el programa ROM es proseguido sin alteraciones. Este sector "Patch" se encuentra en el sistema RAM desde &HFD9A hasta &HFFC9.
Si queremos modificar una rutina, existe la posibilidad de posicionar el "Patch" en las direcciones de salto en la RAM pertenecientes a esta rutina. Esto significa, que el comando RET es sustituido ("tapado") por los códigos de otros comandos, por ejemplo por un comando de salto. Para cada intervención Patch se dispone de 5 Bytes para la modificación. De ninguna forma se pueden modificar más, ya que si no sería manipulada la siguiente rutina. Para ver más claro este método de la modificación del sistema, veamos un ejemplo:

Se quiere modificar el comando >INPUT< del BASIC de tal forma que no se indique ningun símbolo de interrogación. La rutina Input comienza en la dirección &H23CC. Traduzca VD. con el Disassembler a partir de la dirección &H23CC:

```
23CC CDE0F0     CALL &HFDE0
23CF 3E3F       LD   A,&H3F
23D1 DF         RST  &H18
23D2 3E20       LD   A,&H20
23D4 DF         RST  &H18
23D5 .
 .   .
 .   .
```

El primer comando es el salto hacia el sector de RAM de Patch. Si queremos modificar esta rutina de interrupt, disponemos de las direcciones &HFDE0 hasta &HFDE4. El código para "?" es cargado con el próximo comando y editado por medio de RST &H18. Estos dos comandos se han de saltar. El programa ha de proseguir en la dirección &H23D2. El Patch tendría la siguiente configuración:

```
Dirección inicial del Patch: &HFDE0
POP AF ; Traer dirección de salto atrás
JP &H23D2 ; Saltar símbolo de interrogación

Los códigos resultan de: F1,C3,D2,23
```

El siguiente miniprograma inicia el Patch:

```
10 POKE &HFDE1,&HC3
20 POKE &HFDE2,&HD2
30 POKE &HFDE3,&H23
40 REM activación:
50 POKE &HFDE0,&HF1
60 REM desconectar con &HFDE0,&HC9
```

Es importante que la dirección inicial del Patch, en este ejemplo &HFDE0, sea cargada la última con el valor nuevo. Para anular esta modificación y volver al estado inicial, hay que escribir RET, o sea, &HC9 en el mismo lugar.

El siguiente programa edita con >Print< simultáneamente sobre la impresora y sobre la pantalla.

El programa modifica la rutina RST &H18 de tal forma que es llamada dos veces. Primero para la edición por pantalla y la segunda vez por impresora.

```
 10 CLEAR 200,&HF300
 20 POKE &HFEE5,0
 30 POKE &HFEE6,&HF3
 40 REM INIT con POKE &HFEE4,&HC3
 50 REM desconectar con POKE &HFEE4,&HC9
 60 REM no utilizar en el modo directo
 70 FOR I=&HF300 TO &HF30B
 80 READ A$
 90 POKE I,VAL ("&H"+A$)
100 NEXT
110 DATA F5,3A,61,F6,32,15,F4,F1
120 DATA CD,63,1B,C9
```

Listado Assembler

```
FEE4   JP &H3000;RST &H18 Patch

F300 F5      PUSH AF ; ASCII salvar código
F301 3A61F6 LD  A,(&HF661) ; POS
F304 3215F4 LD  (&HF415),A; cargado en LPOS
F307 F1      POP AF ; tomar código ASCII
F308 CD631B CALL &H363 ; edición por impresora
F30B C9      RET ; seguir con edición por pantalla
```

Capítulo VII : Peeks y Pokes

7.1 Introducción

En los capítulos anteriores hemos utilizado frecuentemente PEEKs y POKEs. A continuación vamos a explicar que son PEEKs y POKEs básicamente. Después seguirá una lista de las direcciones POKE más importantes con su respectiva función y aplicación.

Como Vd. sabe, los ordenadores MSX disponen de 32 K ROM. Allí está contenido el sistema operativo, el cual le permite programar de la forma más simple. Al transcurrir estos programas internos, escritos completamente en lenguaje máquina, resultan muchas informaciones que han de ser memorizadas. Son estas por ejemplo:

- el modo de pantalla elegido
- los colores elegidos
- Informaciones sobre interrupts del BASIC conectados (ON STRIG< etc.)
- Longitud de las líneas de pantalla >WIDTH<

De la necesidad de memorizar estas informaciones, resultan la siguientes consecuencias:

- Como en la ROM (memoria fija) no se pueden memorizar informaciones, se ha de reservar para esto una parte de la RAM. En el MSX BASIC este sector RAM se encuentra desde &HF380 hasta &HFFF.

- Como a nivel del lenguaje máquina no existen variables similares a las conocidas en el BASIC, se utilizan una o varias células de memoria para la memorización de las informaciones.

- Estos sectores de memoria determinados por el sistema, son utilizados por el propio sistema y modificados por él mismo en determinados casos. Pero como se encuentran en la RAM, pueden ser leídos con >PEEK< y modificados con >POKE<.

Con ayuda del PEEK y POKE son posibles cosas que desde el BASIC no están previstas. Como un >POKE< (el PEEK no) interviene directamente en el transcurso del sistema, el uso inadecuado de este comando puede traer consigo el despido del ordenador, o por lo menos una elaboración errónea.

7.2 Los comandos

>PEEK dirección< lee el valor (1 Byte) contenido en la dirección indicada.

>POKE dirección, valor< escribe el valor indicado (1 Byte) en la dirección indicada.

Frecuentemente, dos memorias contiguas forman una pareja, que ha de ser leída/escrita simultáneamente. En ello, la memoria con la dirección inferior contiene el Byte bajo (ver capítulo 4) y la memoria con la dirección superior contiene el Byte alto. Para determinar el valor de ambas memorias, ha de multiplicarse por 256 el valor del Byte alto (HB), a cuyo resultado ha de sumarse el valor del Byte bajo (LB). Para la división de un número de 2 Bytes, léase el capítulo 4.
A continuación una lista de las direcciones POKE más importantes con su función y su aplicación.

7.3 Las direcciones

F3AE LINL40 Longitud de LINeas modo de texto (máx. 40 caracteres).
Si se elige >SCREEN 0<, entonces >WIDTH< es colocado a este valor.

Ejemplo: >POKE &HF3AE,2:SCREEN 0<

F3AF LINL32 Longitud de LINeas en el modo gráfico I (máx. 32 caracteres).
Si se elige >SCREEN 1<, entonces >WIDTH< es colocado a este valor.
Es interesante, que las dos direcciones anteriores permiten valores mayores que los valores máximos. La introducción de tales valores permite la posibilidad de salirse lateralmente de la pantalla con el cursor.

F3B0 LINLEN LENGTH LINéas (longitud)

Contiene la longitud actual de una línea de pantalla (>WIDTH<). Este valor puede ser modificado directamente por medio de >POKE<, lo cual corresponde al comando >WIDTH< sin (!) >CLS<. Por medio de >SCREEN< , la longitud modificada por medio de >WIDTH< es reposicionada al estado original.

F3B1 CRTCNT Cathode Ray Tube CouNT - Contador de líneas de pantalla

Cantidad de líneas por pantalla. Esto significa al mismo tiempo, que en la última línea, cuyo número está memorizado aquí, se efectúa la edición de la ocupación del teclado.

F3B2 TABCNT TABulator CouNTer

Contiene la cantidad de espacios libres que se editan con TAB.

F3B3/4 TXTNAM Tabla de nombres

Dirección estándar de la tabla de nombres en el modo de texto.

F3B7/8 TXTCGP TeXT Character Generator Patten

Dirección estándar del generador de patrones de caracteres en el modo de texto.

F3BD/E T32NAM Modo de texto (máx. 32 caracteres) tabla de nombres
Dirección estándar de la tabla de nombres en el modo gráfica I.

F3BF/C0 T32COL Modo de texto (máx. 32 caracteres) tabla de color

F3C1 T32CGP Modo de texto (máx. 32 caracteres) Character Generator Patten

F3C3/4 T32ATR Modo de texto (máx. 32 caracteres) Tabla de ATRibutos de Sprites

F3C5/6 T32PAT Modo de texto (máx. 32 caracteres) Tabla de PATterns

F3C7/8 GRPNAM Tabla de nombres gráfica

F3C9/A GRPCOL Tabla de COLor GRáfica

F3CB/C GRPCGp GRaPhIk Character Generator Patten

F3CD/E GPPATR GRaPhIk tabla de ATRibutos

F3CF/D0 GRPPAT GRaPhIk tabla de PATterns

F3D1/2 MLTNAM Tabla de nombres MuLTIcolor

F3D5/6 MLTCGP MuLTIcoulor Charakter Generator Pattern

F3D7/8 MLTATR MuLTIcoulor Tabla de ATRibutos

F3D9/A MLTPAT MuLTIcoulor Tabla de PATterns

F3DB CLIKSW CLick Switch

Flag 0 significa desconectado el sonido de pulsación de tecla

F3DC CSRY Posición Y CurSoR

F3DD CSRX Posición X CurSoR

Las posiciones del cursor se refieren siempre a la definición actual >WIDTH<.

F3DE FNDFLG FuNction Display FLaG

Utilización ver capítulo gráfica

F3DF hasta F3E6 RG0SAV-RG7SAV SAVe ReGistro n

Aquí se memoriza el contenido de los registros 0 hasta 7 del VDP. Los registros VDP 0 hasta 7 son ''sólo registros de lectura''. Para que pueda ser consultado su valor actual, se ha memorizado aquí. Estas memorias no deberían modificarse por >POKE<, ya que esto originaría trastornos.

F3E8 TRGFLG TRIGger FLag

El Bit 4 de este Byte es 0, cuando es pulsado el disparador.

F3E9 FORCLR FORground CoLoR

Color actual de la escritura

F3EA BAKCLR BACKground CoLoR

F3EB BDRCLR BorDeR CoLoR

F3F2 ATRBYT ATtRIBute BYTe
Color actual para la gráfica de alta resolución.

F3F6 SCNCNT SCaN CouNTer

Contador que cuenta de 3 hasta 0 al transcurrir el Interrupt. Con 0 son consultados los Interrupts del teclado y del STRIG.

F3F7 REPCNT REPeat CouNTer

Contador de repetición para la función de repetición del teclado. Normalmente está posicionado en 13 si la tecla está pulsada. Evita que una tecla sea registrada varias veces. Si se desea una función de repetición inmediata, cargar REPCNT con 1. Ver programa consulta del teclado en el capítulo sobre E/S.

F3F8 PUTPNT PUT PoINTer

Señala sobre la dirección, en la cual se ha de memorizar provisionalmente la nueva introducción por el teclado.

F3F7 GETPNT GET PoINTer

Señala sobre el inicio actual del buffer de teclado desde la última consulta del buffer. La diferencia de PUTPNT hasta GETPNT indica la cantidad de teclas pulsadas (entretanto) .

F414 ERRNUM ERRor NUMber
Número del último error surgido

F415 LPTPOS Line PrinTer POSition

Posición del cabezal de impresión en la línea actual

F416 PRTFLG PRint FLaG

SI PRTFLG obtiene un 1, entonces la rutina general de edición OUTDO, que es llamada por RST &H18, edita por la impresora.

F417 NTMSXP NoT MSX Printer

Corresponde a la posible introducción con el comando >SCREEN< sobre impresora MSX o no MSX.

F418 RAWPRT RAW PRint

SI RAWPRT es desigual a 0, entonces se efectúa una impresión sin preparación previa (TAB,LPOS,etc.)

F41C/D CURLIN CURrant LINe

Número de línea actual

F663 VARTYP Número característico de variable

F676/7 BASSTA BASic Dirección Inicial

F6AA AUTOFL AUTO FLag

1 significa AUTO conectado
0 significa AUTO desconectado.

F6AB/C Línea AUTO actual

F6AD/E Paso a paso para AUTO

F5B3 ERRLIN ERRor LINe

Contiene la línea en la cual surgió el último error.

F5C2 VARTAB Dirección Inicial de la TABla de VARiables.

F6C4 VTBTAB TaBlas de Variables (Campos Arrays), dirección inicial de la tabla

F7C4 TROFLG TROn FLaG

0-TROFF; 1-TRON

F85F MAXFIL Cantidad MAXFILes

F91F CGSLOT CHraracter Generator SLOT

Número de Slot en el cual se encuentra una copia del juego de caracteres.

F929/1 CGPNT Character Generator PoINTer

Dirección de la copia CG

F922/3 NAMBAS actual Dirección BASica de la tabla de nombres

F924/5 CAPBAS Character Generator Dirección BASica actual

F926/7 PATBAS Sprite PATtern Dirección BASica actual

F928/9 ATRBAS Sprite ATtRIbut Tabla de dirección BASica

FBB0 ENSTOP ENable Stop

SI ENSTOP = 1, entonces es posible un arranque en caliente, o sea, una interrupción que lleva inmediatamente al modo de introducción.
Escriba un programa que está protegido contra toda interrupción (>ON STPO< etc.) y pulse, mientras el programa transcurre SHIFT+CODE+GRAF+CTRL, y se encontrará simultáneamente en el modo de introducción. Protección ante esto por >POKE &HFBB0,0<.

FBB1 BASROM Programa BASIc en la ROM

Cuando BASROM <>0, esto significa normalmente que un programa ROM BASIC transcurre y por este motivo es imposible cualquier interrupción por CTRL-C o CTRL-STOP.
Esta protección también se puede utilizar para programas BASIC propios >POKE &HFBB1,1<.

FBCC GODSAV CODe SAVe

corresponde a la variable TIME

FC9E TIMER

corresponde a la variable TIME

FCA9 CSRSW CurSoR ShoW - Cursor conectado/desconectado

FCAA CSTYLE Cursor STYLE

Cursor dividido (Insert-) conectado/desconectado

FCAB CAPST CAP STatus

Escritura mayúscula/minúscula conectada/desconectada

FCAC Tecla especial para letras especiales ha sido pulsada/no pulsada.

FCAF SCRMOD SCReen MODus

modo actual de pantalla

FCB0 OLDSCR OLD SCRee

modo anterior (texto), cuando la gráfica está conectada.

CAPITULO VIII : BASIC INTERNO

8.1 INTRODUCCION

El siguiente capítulo va a estar dedicado a la memorización interna del BASIC y de las variables. Este tema representa un punto intermedio entre lenguaje máquina y programación pura en BASIC.

Lenguaje máquina en el sentido de que el interpretador BASIC está escrito en lenguaje máquina, y las estructuras internas de programas y variables BASIC son comprendidas y elaboradas por el interpretador.

BASIC naturalmente, porque se trata de la estructura de programas BASIC. Para programación simple en BASIC no es necesario el conocimiento de las estructuras internas. Como ya es frecuente, por medio del aprovechamiento de estas estructuras se pueden escribir programas que en realidad "no son posibles". A ellas pertenecen por ejemplo:

- Producción de líneas BASIC desde el BASIC. De esta forma son posibles, por expresarlo así, programas que se "autoescriben".

- Ayudas de programación, como DUMP (edición de todas las variables y sus valores), XREF (índice de todas las variables y de la línea de programa en la cual son utilizadas).

- Se puede programar un REM Killer. Tal programa borra todos los comentarios en un programa y junta, en condiciones determinadas, varias líneas cortas formando líneas más largas. De esta forma, un programa terminado se puede acelerar.

Estos ejemplos han de ser suficientes, para demostrar lo que se puede hacer con las siguientes informaciones.

8.2 MEMORIZACION DE LINEAS BASIC

Como en los programas BASIC se trata de datos variables, estos son memorizados en la RAM. La dirección inicial del BASIC RAM en las versiones de 32 K y más, es &H8001, en las versiones de 16 K es &HC001. Básicamente, la dirección inicial de la RAM de BASIC se encuentra en las direcciones &HF676/77 en forma de Byte bajo y alto. La siguiente línea BASIC edita la dirección inicial:

```
PRINT HEX$ (PEEK(&HF676)+256*PEEK(&HF677))
```

Para observar la construcción interna de un programa BASIC, necesitamos un programa que nos muestre exactamente el contenido de las memorias. Este programa se denomina monitor (ver capítulo 5). Para que Vd. pueda comprender lo expuesto, debería copiar el programa de monitor. Además introduzca la siguiente línea como primera línea del monitor:

```
1 REM Libro Data Becker
```

Esta línea va a ser investigada. Inicie Vd. el monitor e indique &H800 como dirección de inicio y &H8017 como dirección final. Vd. obtendrá la siguiente expresión sobre la pantalla:

```
8000 00 18 80 01 00 8F 4C 69 ......Li
8008 62 72 80 20 44 61 74 61 bro Data
8010 20 42 65 63 6B 65 72 00  Becker
```

El significado de los diferentes Bytes es:

Dirección	Byte	Significado
&H8000	&H00	Byte cero simboliza el comienzo BASIC
&H8001	&H18	
&H8002	&H80	Los dos siguientes Bytes determinan la dirección de memoria, en la cual comienza la siguiente línea. Como siempre está en primer lugar el Byte bajo, en nuestro caso esto significa que la siguiente línea está memorizada a partir de la dirección &H8018 (dirección de encadenamiento).
&H8003	&H01	A la dirección de encadenamiento sigue
&H8004	&H00	nuevamente el número de línea representado por dos Bytes, aquí 1
&H8005	&H8F	Aquí comienza la primera línea del programa. &H8F es la representación codificada (Token) para el comando REM. Ahora sigue la observación después del REM. Comienza con el código de la primera letra:
&H8006	&H4C	Letra L
&H8007	&H69	Letra i
&H8008	&H62	Letra b
&H8009	&H72	Letra r
&H800A	&H80	Letra o
&H800B	&H20	espacio libre
&H800C	&H44	Letra D
&H800D	&H61	Letra a
&H800E	&H74	Letra t
&H800F	&H61	Letra a
&H8010	&H20	espacio libre
&H8011	&H42	Letra B
&H8012	&H65	Letra e
&H8013	&H63	Letra c
&H8014	&H6B	Letra k
&H8015	&H65	Letra e
&H8016	&H72	Letra r
&H8017	&H00	Este Byte 0 significa, que aquí ha finalizado la línea actual.

Borre ahora la línea 1 y pruebe otra línea:

```
2 PRINT "Data Becker"
```

Introduza nuevamente &H8000 y &H8017 para el monitor. La construcción es similar.

&H800 es el Byte de inicio. A continuación sigue la dirección de encadenamiento de 2 Bytes, después el número de línea (2 Byte), aquí 2, después sigue &H91, el Token para PRINT. A continuación siguen los símbolos de comillas (&H22) y después el texto a editar (Data Becker). Después del texto siguen nuevamente comillas. Al final de la línea está, como siempre, el Byte 0.
Los 2 Bytes siguientes representan la dirección de encadenamiento de la siguiente línea. En ello, el Byte bajo está en la dirección &H8014. A mano de estas direcciones es posible penetrar en el programa BASIC. Para que la técnica quede clara, a continuación un pequeño programa:

```
10 VA=&H8001 : REM dirección inicial BASIC
20  ZN=PEEK(VA+2)+256*PEEK(VA+3): REM la siguiente dirección de encadenamiento
30  VA=PEEK(VA)+256*PEEK(VA+1):REM la siguiente dirección de encadenamiento
40 IF VA=0 THEN END
50 PRINT ZN: REM editar número de línea
60  GOTO  30:  REM  repetir  proceso  con  nueva dirección de encadenamiento.
```

Si en su ordenador el sector BASIC no comienza a partir de la dirección &H8001, tiene que modificar la línea 10.
Como resultado obtendrá, uno detrás de otro, todos los números de líneas (como con List, pero sin contenido).
La línea 30 lee el número de línea siguiente a la dirección de encadenamiento.

En la línea 40, a base de la dirección de encadenamiento antigua, que señala siempre hacia la siguiente dirección, esta última es leída. Si para el valor de una dirección de encadenamiento se obtiene 0, esto significa que el programa BASIC ha finalizado en este momento. Un programa se finaliza siempre por 3 Bytes 0. El primer Byte 0 indica el final de la primera línea de programa y los dos siguientes representan la dirección de encadenamiento.
El final de programa se fija en la línea 50.
Ya hemos visto el "esqueleto" de todos los programas BASIC:

```
1 Byte 0    marca de partida BASIC
2 Bytes     primera dirección de encadenamiento
2 Bytes     primer número de línea
.
.
.
1 Byte 0    marca de final de línea
2 Bytes     segunda dirección de encadenamiento
2 Bytes     segundo número de línea
.
.           contenido de línea
.
1 Byte 0    final de línea
2 Byte 0    para final del programa
```

Los así denominados TOKEN juegan un papel muy importante repecto al contenido de línea.

8.3 Token

Un Token es un código que sustituye a un comando BASIC, o sea, que internamente nunca se memoriza el comando literalmente. Esto necesitaría demasiado espacio. Por este motivo, después de finalizar la introducción/modificación de una línea, se reconocen todas las palabras de comando y se sustituyen por sus repectivos códigos (Token).

Existe un truco muy simple para determinar la relación entre palabras de comando y Token. Introduza Vd.:

```
1 REM Hallo
```

Como hemos visto con ayuda del monitor, el Token para REM está inmediatamente después del número de línea, o sea, en la dirección &H8005 (BASIC inicio = &H8000). >PRINT PEEK (&H8005) resulta 143=&H8F. Sustituyamos este Token por otro >POKE &H8005,&H91<. Si ahora introduce >LIST 1< obtendrá:

```
1 PRINT Hallo
```

```
>POKE &H8005,&H84, resulta
```

```
1 DATA Hallo
```

O sea, &H84 es el Token para el comando >DATA<. Para que no tenga que comprobar todos laboriosamente, a continuación la lista de los Token de todas las palabras de comando BASIC.

DEX	HEX	COMANDOS	DEZ	HEX	COMANDOS
129	81	END	130	82	FOR
131	83	NEXT	132	84	DATA
133	85	INPUT	134	86	DIM
135	87	READ	136	88	LET
137	89	GOTO	138	8A	RUN
139	8B	IF	140	8C	RESTORE
141	8D	GOSUB	142	8E	RETURN
143	8F	REM	144	90	STOP
145	91	PRINT	146	92	CLEAR
147	93	LIST	148	94	NEW
149	95	ON	150	96	WAIT
151	97	DEF	152	98	POKE
153	99	CONT	154	9A	CSAVE
155	9B	CLOAD	156	9C	OUT
157	9D	LPRINT	158	9E	LLIST
159	9F	CLS	160	A0	WIDTH
161	A1	ELSE	162	A2	TRON
163	A3	TROFF	164	A4	SWAP
165	A5	ERASE	166	A6	ERROR
167	A7	RESUME	168	A8	DELETE
169	A9	AUTO	170	AA	RENUM
171	AB	DEFSTR	172	AC	DEFINT
173	AD	DEFSNG	174	AE	DEFDBL
175	AF	LINE	176	B0	OPEN
177	B1	FIELD	178	B2	GET
179	B3	PUT	180	B4	CLOSE
181	B5	LOAD	182	B6	MERGE
183	B7	FILES	184	B8	LSET
185	B9	RSET	186	BA	SAVE
187	BB	LFILES	188	BC	CIRCLE
189	BD	COLOR	190	BE	DRAW
191	BF	PAINT	192	C0	BEEP

DEZ	HEX	COMANDOS	DEZ	HEX	COMANDOS
193	C1	PLAY	194	C2	PSET
195	C3	PRESET	196	C4	SOUND
197	C5	SCREEN	198	C6	VPOKE
199	C7	SPRITE	200	C8	VDP
201	C9	BASE	202	CA	CALL
203	CB	TIME	204	CC	KEY
205	CD	MAX	206	CE	MOTOR
207	CF	BLOAD	208	D0	BSAVE
209	D1	DSKO$	210	D2	SET
211	D3	NAME	212	D4	KILL
213	D5	IPL	214	D6	COPY
215	D7	CMD	216	D8	LOCATE
217	D9	TO	218	DA	THEN
219	DB	TAB(	220	DC	STEP
221	DD	USR	222	DE	FN
223	DF	SPC(	224	E0	NOT
225	E1	ERL	226	E2	ERR
227	E3	STRING$	228	E4	USING
229	E5	INSTR	230	E6	'(REM)
231	E7	VARPTR	232	E8	CSRLIN
233	E9	ATTR$	234	EA	DSKI$
235	EB	OFF	236	EC	INKEY$
237	ED	POINT	238	EE	>
239	EF	=	240	F0	<
241	F1	+	242	F2	-
243	F3	*	244	F4	/
245	F5	^	246	F6	AND
247	F7	OR	248	F8	XOR
249	F9	EQV	250	FA	IMP
251	FB	MOD	252	FC	U

En esta lista surgen algunos comandos que Vd. no conocerá. Se trata de comandos que se utilizan cuando su ordenador disponga de una unidad de discos conectada.

Como Vd. ve, todos los Token son mayores que &H80=128. Casi todos los números hasta 255 están ocupados, pero aún faltan algunas funciones, por ejemplo >PEEK<.
Introduzca Vd. >POKE &H8005,255:POKE &H8006,&H97< y >LIST<.

En lugar del comando >REM<, que estaba originalmente en estas líneas, ha surgido la función >PEEK<. Esto significa que las funciones del BASIC MSX están codificadas por 2 Token. Primero el Token con el valor 255 indica que ahora sigue una función. Después sigue el Token propiamente dicho de la función. A continuación la lista de los Token de función (antes ha de estar siempre 255).

DEZ	HEX	COMANDOS	DEZ	HEX	COMANDOS
129	81	LEFT$	153	99	SPACE$
130	82	RIGTH$	154	9A	OCT$
131	83	MID$	155	9B	HEX$
132	84	SGN	156	9C	LPOS
133	85	INT	157	9D	BIN$
134	86	ABS	158	9E	CINT
135	87	SQR	159	9F	CSNG
136	88	RND	160	A0	CDBL
137	89	SIN	161	A1	FIX
138	8A	LOG	162	A2	STICK
139	8B	EXP	163	A3	STRIG
140	8C	COS	164	A4	PDL
141	8D	TAN	165	A5	PAD
142	8E	ATN	166	A6	DSKF
143	8F	FRE	167	A7	FPOS
144	90	INP	168	A8	CVI
145	91	POS	169	A9	CVS
146	92	LEN	170	AA	CVD
147	93	STR$	171	AB	EOF
148	94	VAL	172	AC	LOC
149	95	ASC	173	AD	LOF
150	96	CHR$	174	AE	MKI$
151	97	PEEK	175	AF	MKS$
152	98	VPEEK	176	B0	MKD$

Como Vd. ve, en los Token de función aún queda espacio. Probando un poco se pueden hacer cosas interesantes utilizando los Token ''vacíos''.

Protección contra listado

Hemos encontrado un Token con el cual se puede conseguir una fácil protección contra listados. Partamos de nuevo de la línea >1 REM Hallo<. Introduzca lo siguiente:

```
POKE &H8005,255
POKE &H8006,52
```

Ahora liste el programa:
Al iniciar el listado se ejecuta automáticamente un >CLS<. Si a distancias uniformes se insertan en un programa líneas >REM< manipuladas de tal forma (especialmente después de partes importantes), es muy difícil conseguir un listado completo sobre la pantalla. Hacia estas líneas no se puede saltar, ya que esto originaría un Syntax Error. Estas líneas se saltan con:

```
RENUM
5 GOTO 20
```

El burlador de programas puede borrar simplemente estas líneas. Este caso lo trataremos en los Restos (Capítulo IX).

Aquí otra aplicación

Programa: Generador de líneas DATA

Datos (aquí palabras) introducidos a través del teclado han de escribirse en el programa por medio de líneas >DATA<. De esta forma se podría escribir, por ejemplo, un fichero de direcciones, cuyos datos (!) no están memorizados en un cassette sino directamente en el programa.
La líneas >DATA< a modificar han de existir previamente en el programa. Supongamos, que la línea DATA tiene el número 1000 y que contiene 15 puntos como datos provisorios. Entonces funciona el siguiente programa:

```
 10 INPUT"palabra",A$
 20 L=LEN(A$)
 30 IF L>15 THEN 10
 40 VA=&H8001
 50 ZN=PEEK(VA+2)+256*PEEK(VA+3)
 60 IF ZN=1000 THEN 100
 70 VA=PEEK(VA)+256*PEEK(VA+1)
 80 IF VA=0 THEN PRINT "Falta linea 1000":END
 90 GOTO 50
100 AD=VA+4:REM dirección inicial de la línea DATA 1000
110 IF  PEEK(AD)<>&H84  THEN PRINT "linea 1000 no es linea
    DATA":LIST 160
120 FOR I=1 TO L
130 IF PEEK(AD+I)<>46 THEN PRINT "pocos puntos en la linea
    DATA 1000":LIST 1000
140 POKE POKE AD+I,ASC (MID$(A$,I,1))
150 NEXT
160 FOR I=L+1 TO 15
170 IF PEEK (AD+I)<>46 THEN 130
180 POKE AD+I,32:NEXT
1000 DATA ..............
```

Después de transcurrir el programa, el valor introducido por Vd. se encontrará en lA línea >DATA< 1000. Las líneas hasta 90 las hemos comentado ya.
La línea 110 comprueba el Token >DATA<.
En el primer bucle (línea 120 hasta 150), la palabra es POKEada letra por letra en la línea (línea 140). Antes se comprueba si en el lugar correspondiente hay un punto (línea 130).
El bucle en las líneas 160 hasta 180 completa el resto de la línea con espacios libres.

Con los Tokens se han comentado todos los Bytes posibles mayores que &H80.
Bytes con valores entre 32 y 127 representan en la representación interna del BASIC siempre sus códigos ASCII, quiere decir, que están en lugar del carácter correspondiente. Hay que tener especialmente en cuenta los códigos de caracteres de gobierno desde 0 hasta 31. No todos estos códigos cumplen una función determinada.
Aquí un resumen:

Código	Función
11	Símbolo para números octales (&O)
12	Símbolo para números hexadecimales (&H)
17	Cifra 0
18	Cifra 1
19	Cifra 2
20	Cifra 3
21	Cifra 4
22	Cifra 5
23	Cifra 6
24	Cifra 7
25	Cifra 8
26	Cifra 9
27	número 10
28	Símbolo para números de 2 Bytes (INT)
29	Símbolo para números de 4 Bytes (SGN)
31	Símbolo para números de 8 Bytes (DBL)

Todos estos códigos están relacionados de una u otra manera con la memorización de números. De la memorización de números y Strings en programas y en variables vamos a ocuparnos ahora.

8.4 REPRESENTACION DE NUMEROS

En un ordenador existen cuatro métodos diferentes de representación de datos:

1. Integer	INT - Número entero
2. Single Precision	SNG - Exactitud simple
3. Double Precision	DBL - Exactitud doble
4. Strings	STR - Alfanumérico

Integer

Vamos a ocuparnos primero de la primera forma, o sea, de la representación de números. El primer método, la representación de números enteros, ya la ha conocido Vd. Un número Integer es dividido en el Byte bajo y en el Byte alto. El Bit 7 del Byte alto se utiliza como signo. 0 significa números positivos y 1 significa números negativos. Los números negativos se representan en un complemento de dos. Esto significa, que el valor absoluto es complementado y sumado 1. De esta forma, constantes o variables pueden obtener valores enteros desde -32768 hasta +32767.

decimal	binario	hex
-32768	1 000 0000 0000 0000	80 00
-32767	1 000 0000 0000 0001	80 01
-32766	1 000 0000 0000 0010	80 02
-32765	1 000 0000 0000 0011	80 03
	...	
-2	1 111 1111 1111 1110	FF FE
-1	1 111 1111 1111 1111	FF FF
0	0 000 0000 0000 0000	00 00
1	0 000 0000 0000 0001	00 01
2	0 000 0000 0000 0010	00 02
	...	
32766	0 111 1111 1111 1110	7F FE
32767	0 111 1111 1111 1111	7F FF

Para diferenciar internamente las diferentes formas de representación, cada una de ellas tiene asignada una cifra característica, la cual es igual a la cantidad de Bytes que son necesarios para la memorización del tipo correspondiente. Para memorizar un número Integer son necesarios 2 Bytes; por lo tanto la cifra característica para constantes Integer o variables es 2. Las cifras características de las variables actuales están memorizadas siempre en la dirección &HF663.

Representación con coma flotante

Los números de coma flotante se memorizan en el sistema operativo MSX de una forma poco común para este tipo de ordenadores: El formato BCD.

En el formato BCD (Binario Codifica Decimal) se memorizan las diferentes cifras de un número decimal. La ventaja consiste en una cantidad exacta de valores que se elaboran. Tambien el sector de validez en el sistema MSX es mayor (10^-64 hasta 10^62) que el sistema convencional de memorización (1o^32 hasta 10^31). Lamentablemente, los números en el formato BCD no se pueden elaborar internamente

con tanta rapidez. Por este motivo existen dos exactitudes: SGN con 6 valores y DBl con 14 valores.
La representación con exactitud simple (SGN) cumplirá las necesidades en casi todos los casos.

¿Cómo se memoriza un número en el formato BCD?

Antes de ocuparnos de la memorización de cifras, vamos a tratar sobre la representación exponencial de números, tal como es conocido por las calculadoras de bolsillo. Si un número se ha de representar de forma exponencial, primero se ha de determinar cuántas veces cabe la base del sistema decimal, o sea, el 10, en dicho número. A continuación el número se divide en dos partes. Una parte contiene todas las cifras del número (mantisa), en el cual la coma siempre está detrás de la primera cifra. La segunda parte indica cuantas veces cabe el 10 en el número original, o expresado de otra forma, cuantas cifras hay que desplazar la coma en la primera parte (exponente). Se escribe de la siguiente forma:

```
    27 = 2.7*10^2
  3956 = 3.956*10^4
```

Con el acuerdo, de que los exponentes negativos significan un desplazamiento de la coma hacia la izquierda, se pueden representar todos los números:

```
   0.21 = 2.1*10^-1
 0.0051 = 5.1*10^-3
     -9 = -9.0*10^0
```

Con este acuerdo, cada número se puede dividir en una mantisa (parte con cifras, solo una cifra delante de la coma) y un exponente. La forma de escritura exponencial tiene la ventaja que números muy grandes y muy pequeños se pueden escribir con un espacio mínimo:

```
0.00000000000000735 = 7.35*10^-15
   6390000000000000 = 6.39*10^15
```

Al operar con estos números se aplican las reglas comunes del cálculo exponencial.

Adición / Sustracción

Sólo se pueden sumar números con el mismo exponente. Si los exponentes de los sumandos son diferentes, entonces hay que transformar el número con el exponente más pequeño al mayor. Si entonces los exponentes son iguales, se pueden sumar o restar simplemente las mantisas.

Ejemplo:

```
      57 + 0.31
5.7*10^1 + 3.1*10^-1
5.7*10^1 + 0.031*10^1
    5.731*10^1
      57.31
```

Multiplicación / División

En estas reglas de cálculo, se multiplan o se dividen las mantisas y después se suman o se restan los exponentes.

Ejemplo:

```
      0.13 * 20
1.3*10^-1 * 2.0*10^1
  1.3*2.0 * 10^1(-1+1)
      2.6 * 10^0
      2.6


       25 / 0.05
2.5*10^1 / 5.0*10^-2
 2.5/5.0 * 10^(1-(-2))
      0.5 * 10^3
      5.0 * 10^2
      500
```

Si queremos traspasar este método a un microprocesador, nos encontramos con la dificultad de memorizar estos números con este formato. En el formato BDC se memoriza cada cifra de la mantisa por sí sola.
Una cifra del sistema decimal tiene los valores 0 hasta 9=&B1001. Por lo tanto son necesarios 4 Bits para la memorización de una cifra. Con un Byte = 8 Bits se pueden codificar, por lo tanto, dos cifras, en lo cual el High Nibbel (Bit 4 hasta 7) corresponde a la primera y el Low Nibble (de 0 hasta 3) corresponde a la segunda cifra.

Ejemplo:

```
Decimal      : 27
High Nibble  : 2 = &B10
Low Nibble   : 7 = &H111
Formato BCD  : &B0010 0111 = &H27=39
```

A causa de las propiedades especiales del sistema hexadecimal, en el cual 4 Bits corresponden a una cifra hexadecimal, el número hexadecimal que corresponde al número decimal, es exactamente el valor BCD codificado.

Valor BCD de 63 es &H63 = &B0110 00111 = 99

La cantidad de cifras que se pueden memorizar es teóricamente ilimitada. Es determinada por la cantidad de Bytes necesarios para su memorización. Al representar números con exactitud simple (SNG), la cantidad de Bytes necesarios para la memorización de las cifras de la mantisa son 3 Con esto, los números de este tipo tienen una exactitud de 6 cifras, 2 por cada Byte.
El número 123794 se memoriza en el formato BCD en forma de una serie de 3 Bytes:

&H12, &H37, &H94

Si se elige la doble exactitud, entonces se dispone de 7 Bytes, lo cual permite una cantidad de 17 cifras.
Con esto queda claro como se memoriza internamente la mantisa de números de coma flotante representados exponencialmente.
Al codificar el exponente se ha elegido otro camino: Es representado directamente por el valor de un Byte. Pero aún nos faltan las informaciones sobre

- el signo de la mantisa
- el signo del exponente

que han de incluirse en este Byte. El Bit 7 nos da la información sobre el signo del número (de la mantisa):

0 = positivo
1 = negativo

Los 7 Bits restantes representan el valor del exponente, en lo cual se suma &H41 al valor real:

Valor Bit 6-0	Valor exponente
&H7F	&H3E = 62
&H7E	&H3D = 61
•	•
•	•
•	•
&H42	&H01 = 1
&H41	&H00 = 0
&H40	-&H01 =-1
•	•
•	•
•	•
&H02	-&H3F = -63
&H01	-&H40 = -64
&H00	significa número = 0

Por lo tanto, el exponente puede tomar valores entre -64 y +62, un sector de números que es suficiente para cualquier cálculo.
Según se ha acordado, en un número con el valor 0 se pone el exponente a 0.

Comprobemos esta forma de representación en un programa BASIC. Para ello se utiliza el comando >VARPTR<. Esta función nos indica la dirección en la memoria, en la cual se encuentra codificado el valor de la variable en la forma antes descrita. Para administrar las variables sigue, inmediatamente después del programa BASIC, un sector donde se memoriza el valor y el nombre de cada variable utilizada. En la dirección que indica la función >VARPTR< se encuentra el primer Byte del número codificado, el cual es, según acuerdo, el exponente. Después siguen, segun el tipo, los 3 (SGN) ó 7 (DBL) Bytes que representan las cifras.

```
10 X! = 43546!
20 AD = VARPTR (X!)
30 PRINT HEX$ (PEEK(AD))
40 FOR I=AXD+1 TO AD+3
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

```
10 X# = .012342349#
20 AD = VARPTR (X#)
30 PRINT HEX$ (PEEK(AD))
40 FOR I=AD+1 TO AD+6
50 PRINT RIGHT$ ("0"+HEX$(PEEK(I)),2);
60 NEXT
```

Sustituya la X por diferentes números y observe la edición respectiva del programa.

Después de haber comentado los números, vamos a tratar sobre otro tipo de variable totalmente diferente: la variable de String:

8.5 Strings

En un String se memorizan datos alfanuméricos, o sea, datos como letras o cifras. A cada carácter se le ha asignado un código. Los códigos MSX corresponden -para los códigos de 0 hasta 127- a los así denominados American Standard Code for Information Interchange (ASCII).

Por lo tanto, para memorizar un Byte se necesita exactamente un Byte. Un "String" (del inglés: String = cuerda, cadena) es una cadena de códigos de caracteres sucesivos. Como un carácter corresponde a un Byte, un String es memorizado en una serie de memorias consecutivas de la RAM. Para asignar un String determinado a una variable determinada, son necesarias dos informaciones, las cuales forman el denominado String Descriptor:

1. La dirección de la primera posición de memoria que contiene el primer código de carácter del String.

2. La longitud del String, o sea, la cantidad de Bytes que forman la cadena de caracteres.

Estos dos datos se memorizan junto con el nombre de variable en el sector de variables BASIC. La propia cadena se encuentra en otro lugar, o en el programa mismo o en el sector de String especialmente reservado. El tamaño de este sector se puede determinar por medio del comando >CLEAR<.
En el BASIC podemos volver a utilizar de nuevo el comando >VARPTR< para leer la dirección del descriptor de String de una variable. El descriptor de String consta de tres Bytes:

1. Byte : longitud del String
2. + 3. String : dirección inicial del String

La función >VARPTR< indica la dirección del primer Byte del descriptor de String.
Pruebe Vd. el siguiente programa:

```
10 X$ = "CCCadena de CaracTTTeres"
20 AD=VARPTR (X$)
30 LA=PEEL(AD)
40 ST=PEEK(AD+1)+256*PEEK(AD+2)
50 FOR I=ST TO ST+LA-1
60 PRINT CHR$ (PEEK(I));
70 NEXT
```

8.6 Tabla de variables del BASIC

A continuación trataremos brevemente como se administran las variables BASIC con ayuda de la ya mencionada página de variables que sigue inmediatamente al programa.
En la dirección &HF6C2/3 se encuentra la dirección inicial de la tabla de variables.
Se aplica el siguiente esquema en la tabla de variables:

1. Byte : Cifra característica (al mismo tiempo la cantidad de Bytes)
2. Byte : Nombre de la variable
3. Byte : Representación en código ASCII a partir de
4. Byte : Valor de la variable

En detalle vale lo siguiente:

Tipo INT

02 Cifra característica INT
.. Nombre de la variable
.. en el código ASCII
Byte bajo
Byte alto

Tipo SGN

04 Cifra característica SGN
.. Nombre
..
4 Bytes para el valor

```
Tipo DBL

08   Cifra característica DBL
..   Nombre
..
 8 Bytes para el valor

Tipo String

03   Cifra característica String
..   Nombre
..
 3 Bytes para el descriptor de String
        Byte 1 : longitud
        Bytes 2 y 3 : dirección
```

Con esto podemos confeccionar un programa simple para editar todas las variables utilizadas.

Este programa lo puede combinar con sus propios programas y activarlo para fines de pruebas. No debería ser problema ampliar el programa de tal forma, que también se indiquen los valores de las variables.

```
1000  REM Variables DUMP
1010  AD=PEEK(&HF6C2)+256*PEEK(&HF6C5): REM comienzo de
      la tabla de variables
1020  AE=PEEK(&HF6C4)+256*PEEK(&HF6C3):REM final de la
      tabla de variables
10030 TY=PEEK(AD): REM cifra característica
10040 REM editar nombre
10050 PRINT CHR$ (PEEK(AD+1));CHR$(PEEK(AD+2));
10060 REM editar tipo
10070 IF TY=2 THEN PRINT "%"; ELSE IF TY=3 THEN PRINT "$"
      ;ELSE IF TY=4 THEN PRINT "!"; ELSE PRINT''≠'';
10080 PRINT
10090 AD=AD+3: REM inicio del valor de variable
10100 AD=AD+TY:REM inicio de la siguiente variable
10110 IF AD>AE THEN END ELSE 10030
```

CAPITULO IX : VARIOS

9.1 EL COMANDO DEEK

En algunos "dialectos" del BASIC existe el comando >DEEK<. >DEEK< lee el valor de dos Bytes de dos posiciones de memoria sucesivos. Corresponde exactamente a los comandos >PEEK (dirección)+256*PEEK(dirección +1)<. El comando >DEEK< ha de implementarse ahora en el BASIC MSX con ayuda del comando >USR<. La dirección del Byte bajo, de la memoria a leer, ha de traspasarse. Al principio del programa se comprueba, si el parámetro traspasado es del tipo correcto. Si éste no es el caso, se ha de saltar para editar "Typ Mismatch Error". Para ello se carga el registro E con el número del error respectivo y se bifurca hacia &H406F (rutina para la edición de error).

```
CP 2; tipo 2
JR Z,OK ; si, entonces OK
LD E,13 ; no, entonces
JP &H406F ; Typ Mismatch Error
OK...
```

Desde luego, para cada indicación de error existe una dirección propia de entrada (salto) que ejecuta la carga del registro E. Para el tipo "Typ Mismatch Error" es la dirección &H406D. Con esto se simplifica el programa:

```
CP 2 ; tipo 2 ?
JP NZ,&H406D ; no, entonces "Typ Mismatch Error"
   .
   .
```

De esta forma, la solución del control de tipos tiene una desventaja: Si el usuario indica desde el BASIC un número de tamaño correcto, pero de tipo falso (por ejemplo, 1000 puede ser memorizado con tipo integer pero tambien como tipo SGN o DBL).

Esto traería consigo un "Typ Mismatch Error". Pero es posible transformar tal número en un número Integer. En el BASIC ésto lo realiza la función >CINT<.
Por lo tanto, si al principio del programa llamamos la rutina >CINT<, entonces esta tranformación se ejecuta automáticamente. El error indicado sólo surge entonces en la introducción de Strings. Además >CINT< comprueba si en la introducción se trata de un número de tamaño adecuado, si no se edita la indicación de error "Overflow".
Si Vd. necesita variables de otro tipo, utilice las rutinas CSNG (&H2FB2) y CDBL (&H303A). Observe la tabla al final del libro, en la cual se exponen todas las rutinas tratadas y muchas más rutinas de sistema.

Ahora la ejecución del comando >DEEK<.

```
10 ' CALL &H2F8A ; CINT convierte a INT
20 ' LD HL,(&HF7F8) ; valor de parámetro traspasado=di-
   rección
30 ' LD E,(HL) ; Byte bajo
40 ' INC HL
50 ' LD D,(HL) ; Byte alto
```

En la línea 20, el valor de parámetro a acarrear, o sea, la dirección a partir de la cual se ha de leer el valor de Bytes, es leído de las direcciones de acarreo &HF7F8 y &HF7F9. En las líneas 30 hasta 50 se carga en el registro DE el valor de 2 Bytes que está en esta dirección.
Ahora tenemos que devolver al BASIC el valor determinado. Para ello se ha de cargar en la dirección &HF7F8/9 el valor a traspasar. Además, el acumulador ha de contener la cifra característica, y HL ha de ser cargado con la dirección &HF7F6.

```
60 ' LD (&HF7F8),DE ; resultado de "DEEK"
70 ' LD HL, &HF7F6
80 ' RET
```

Traduzca Vd. el programa al Assembler y compruebe la nueva función.

```
PRINT USR1(2)
```

resulta 370, el mismo resultado que

```
PRINT PEEK(2)+256*PEEK(3).
```

Por lo tanto, esto significa que a partir de la dirección 2 se encontrará el valor de 2 Byte 370 ó &H207.

```
F000            10          ; comando DEEK
F000            20          ; da  valor de 2 Bytes a partir
dirección transmitida
F000 CD8A2F     30    CALL &H2F8A ; CINT
F003 2AF8F7     40    LD   HL,(&HF7F8) ; valor de parámetro
transmitido = dirección
F006 5E         50    LD   E,(HL) ; leer Byte bajo
F007 23         60    INC  HL
F008 56         70    LD   D,(HL) ; leer Byte alto
F009 ED53F8F7   80    LD   (&HF7F8),DE ; resultado de DEEK
F00D 21F6F7     90    LD   HL,&HF7F6
F010 C9        100    RET
-------------------------------------------
Programa : deek
Inicio : &HF000     Fin : &HF010
Longitud : &H11 Bytes
Errores : 0
```

```
10 REM comando DEEK
20 CLEAR 200,&HEFFF
30 FOR  I=&HF000 TO &HF010:READ A$
40 POKE I,VAL("&H"+A$):NEXT
50 DEFUSR1=&HF000
60 DATA CD,8A,2F,2A,F8,F7,5E,23
70 DATA 56,ED,53,F8,F7,21,F6,F7,C9
```

9.2 El comando UPPER

Con la ayuda del lenguaje máquina es posible realizar, por ejemplo, el comando >UPPER< conocido en algunos dialectos del BASIC/ El comando >UPPER< transforma a mayúsculas, todas las letras minúsculas de un String.

```
10 CLEAR 200,&HEFFF:MAXFILES=1
20 FOR I=&HF000 TO &HF021:READ A$
30 W=VAL("&H"+A$):S=S+W
40 POKE I,W:NEXT
50 IF S<>3814 THEN PRINT"Error en DATAs":END
60 PRINT"Todo correcto !"
70 DEFUSR1=&HF000
80 DATA FE,03,C2,6D,40,1A,B7,C8
90 DATA 47,62,6B,23,7E,23,66,6F
100 DATA 7E,FE,7B,30,06,FE,61,38
110 DATA 02,E6,DF,77,23,10,F1,3E
120 DATA 03,C9
```

```
F000            10                      ; Comando UPPER
F000            20              ORG     &HF000
F000            30  TYPERR      EQU     &H406D ; Typ mismatch Error
F000 FE03       40              CP      3 ; ¿Variable de cadena?
F002 C26D40     50              JP      NZ,TYPERR ; No, entonces
Typ mismatch error
F005 1A         60              LD      A,(DE) ; Longitud de strings
F006 B7         70              OR      A ; Activar Flag Z si A=0
F007 C8         80              RET     Z ; Terminado si longitud=0
F008 47         90              LD      B,A ; Longitud como contador
de bucles
F009 62        100              LD      H,D ; Dirección de
descriptor
F00A 6B        110              LD      L,E ; hacia HL
F00B 23        120              INC     HL
F00C 7E        130              LD      A,(HL) ; Dirección
F00D 23        140              INC     HL ; del
F00E 66        150              LD      H,(HL) ; Comienzo del string
F00F 6F        160              LD      L,A ; hacia HL
F010 7E        170  SCHLEI      LD      A,(HL) ; si código ASCII
F011 FE7B      180              CP      123 ; >= que ASCII("z")+1
?F013 30FE     190              JR      NC,OK ;entonces no convertir
F015 FE61      200              CP      97 ; si < ASCII("a")
?F017 38FE     210              JR      C,OK ; entonces no convertir
F019 E6DF      220              AND     &B11011111 ; Conversión
**** línea 190 : OK=&HF01B Offset 6
**** línea 210 : OK=&HF01B Offset 2
F01B 77        230  OK          LD      (HL),A ; Grabar ASCII de
nuevo
F01C 23        240              INC     HL ; Dirección del siguiente
código
F01D 10F1      250              DJNZ    SCHLEI ; Repetir hasta final
del string
F01F 3E03      260              LD      A,3 ; Código tipo del string
F021           270                      ; DE contiene todas las
direcciones de descriptor
F021 C9        280              RET
```

```
Programa: upper
Inicio: &HF000    Final: &HF021
Longitud: &H22 Bytes
Errores: 0
Tabla de variables:
TYPERR 406D   SCHLEI F010
OK     F01B
```

9.3 El comando SLOT PEEK

A continuación vamos a ver la 'teoría' respecto al comando Slot PEEK del Monitor. El siguiente texto es un extracto del ''MSX lenguaje máquina'' de DATA BECKER.

Un programa BASIC es traducido, después de su introducción por el programador, a un código intermedio. Por ejemplo, las palabras de comando no se memorizan como tales, sino por medio de un código abreviado (1 Byte de longitud; >128): el así denominado Token. Si el interpretador reconoce un Token, entonces salta hacia la rutina asignada a este Token (=comando). Allí se leen los parámetros que puedan estar escritos detrás del comando, e igualmente se verifican. Vamos a tratar esto detalladamente:

Tomemos como ejemplo el comando >POKE<. Destrás del comando se encuentran dos parámetros separados por un punto y coma: la dirección y el valor. Supongamos, que se ha reconocido el Token para >POKE< (=152) y que el programa se desvía hacia la rutina >POKE< (&H5423). El registro HL señala, en la interpretación BASIC, siempre hacia el lugar del programa BASIC que se está elaborando en este momento. HL s utiliza como BASIC Program Pointer. Hay que asegurar que este Pointer no se "pierda", ya que si no la interpretación no sería correcta.
Para la lectura de los parámetros nombrados existen rutinas especiales en el interpretador, que podemos naturalmente utilizar.
La rutina activada con el comando >POKE< se llama GETADR. Esta lee un parámetro Integer. Al llamar esta rutina, HL ha de cargarse con el Pointer BASIC actual. Expresiones como 1237 se tratan igual que expresiones complejas como:

```
ABS (INT(VAL(RIGHT$(STRING$(4,"0")+HEX$(x),4)))).
```

El valor de 2 Bytes calculado es devuelto al registro DE.

El Pointer BASIC HL señala, en nuestro caso, hacia la siguiente coma después del final de la rutina. La rutina GETADR tiene la dirección &H542F.
A continuación se comprueba si sigue una coma. Para ésto también existe, naturalmente, una rutina de sistema: RST &H08. Este reinicio sirve para la comprobación de un carácter cualquiera. Para ello, el código ASCII del carácter ha de estar junto con DB detrás del comando RST &H08. Por lo tanto, para el comando <POKE<:

```
        .
        .
RST &H08 ; Comprobación de ","
DB  &H2C ; Código de ","
        .
        .
```

Si no se encuentra el carácter, se edita un "Syntax Error", sino un salto hacia atrás. La rutina que se activa entonces para leer el valor de 8 Bit, se llama GETBYT. El valor de 1 Byte es devuelto al acumulador. Después de elaborar esta rutina, el registro HL señala como BASIC Pointer hacia el final del comando. Con ayuda de estas rutinas es posible ampliar el comando >USR< de forma tal que se pueden acarrear cuantos parámetros se quieran. El siguiente programa la utiliza para producir una función >PEEK< modificada, en la cual se indica también el número de Slot, además de la dirección. De esta forma existe la posibilidad de leer módulos de ampliación o módulos solapados incorporados.
La elaboración propiamente dicha se lleva acabo por la rutina a partir de &H000C (Slot RD). Si se acarrea el número de Slot al acumulador y la dirección al registro HL, entonces esta rutina devuelve en el acumulador el valor del Byte leído.
El Basic Pointer puede ser leído de la pila por medio de dos veces "POP" y un programa propio. Es importante que la pila, sobre todo la dirección de salto hacia atrás, sea reconstruída correctamente.

Aquí el listado Assembler:

```
F000            10                      ; Slot PEEK
F000            20                      ; Formato :  USR(dirección),
(Slot)
F000            30              ORG     &HF000
F000            40      ILLQUA  EQU     &H475A
F000            50      CINT    EQU     &H2F8A
F000            60      GETBYT  EQU     &H521C
F000            70      SLOTRD  EQU     &H000C
F000            80              RET     ;
F000 CD8A2F     90              CALL CINT ;  Convierte en INTEGER
y carga valor hacia HL
F003 EB         100             EX      DE,HL
F004 C1         110             POP     BC ;Dirección de salto atrás
F005 E1         120             POP     HL ; Puntero BASIC
F006 E5         130             PUSH    HL ; Stack re-
F007 C5         140             PUSH    BC ; poner
F008 D5         150             PUSH    DE ; Valor de la dirección
F009 CF         160             RST     &H08 ; Test por
F00A 2C         170             DB      &H2C ; ASCII(",")
F00B CD1C52     180             CALL    GETBYT ; Toma número de Slot
F00E FE03       190             CP      3 ; ¿número Slot >= 3 ?
F010 D25A47     200             JP      NC,ILLQUA ; sí,  entonces
Illegal Quantity
F013 E3         210             EX      (SP),HL ; Cambiar dirección
a leer con puntero BASIC
F014 CD0C00     220             CALL    SLOTRD ; Slot Read
F017 CDCF4F     230             CALL    &H4FCF ;  Cargar  acumulador
en FAC
F01A E1         240             POP     HL ; Puntero BASIC
F01B C1         250             POP     BC ; Salto hacia atrás
F01C D1         260             POP     DE ; viejo puntero BASIC
F01D E5         270             PUSH    HL ; nuevo puntero BASIC
F01E C5         280             PUSH    BC ; Salto hacia atrás
F01F 21F6F7     290             LD      HL,&HF7F6 ;Dirección inicial
FAC
F022            300                      ; A ya contiene tipo 2 (vea
rutina a partir de &H4fcf)
F022 C9         310             RET
```

```
Programa: sltrd
Inicio: &HF000    Final: &HF022
Longitud: &H23 Bytes
Errores: 0
Tabla de variables:
ILLQUA 475A    CINT    2F8A
GETBYT 521C    SLOTRD 000C
```

9.4 El comando CLEAR

En nuestro ordenador MSX no funcionaba el comando CLEAR sin problemas:

```
10 CLEAR 200, &HF000
20 INPUT B$
30 FOR I=32 TO 110
40 FOR J=32 TO II
50 A$=A$+CHR$(J)
60 NEXT J
70 PRINT B$
80 A$=""
90 NEXT I
100 END
```

Después de introducir una palabra (mínimo 2 caracteres), este programa debería editar repetidamente la misma palabra. Si su ordenador pone dificultades (por ejemplo, en lugar de ''árbol'' surge "IIIIII"), entonces ponga después de cada comando >CLEAR< un comando >MAXFILES<. O sea:

```
10 CLEAR 200, &HF000:MAXFILES = 1
```

Ahora no surgirán problemas.
El error consiste en que no todos los indicadores para el sector de variables de String BASIC se posicionan correctamente. Con >MAXFILES< se corrige todo.

9.5 Protección contra listado

Una protección simple se consigue bloqueando el List Patch:

```
POKE &HFF89, &HDD
POKE &HFF8A, &HE1
```

Los códigos significan POP IX, o sea, el salto hacia atrás hacia la rutina de listado es tomado de la pila. Con el siguiente comando RET se salta inmediatamente hacia el bucle del interpretador.
Con >POKE &HFF89,&HC9< se anula la protección contra listado.

Otra protección, sobre todo en combinación con lo expuesto en el capítulo 8, es la pregunta por la longitud del programa. Si se intenta visualizar la línea, que al listar origina el CLS, entonces la longitud del programa varía, y esto puede comprobarse. A continuación sigue la autodestrucción del programa. Para ello se necesita la siguiente línea:

```
1 A=FRE(0):IF A<>&H1111 THEN END
```

El valor correcto de la longitud, que se posiciona en lugar de &H1111, se puede determinar con la siguiente prueba:

Introducir RUN y PRINT HEX$(A)

El valor obtenido sustituye a &H1111.
Si ahora modifica cualquier cosa en las siguientes líneas del programa, entonces el programa es finalizado.

Más efectiva es la protección, si el END se sustituye por un NEW (o mejor por un RESET).

9.6 BASIC español

A continuación siguen algunos consejos que sólo tienen sentido si la ROM ha sido copiada en la RAM.

```
POKE &HD4A,1
```

Esta línea origina, que la función de repetición de las teclas actúe sin retardo. También son posibles otros valores diferentes a 1. El valor estándar es 13.

En la RAM podemos realizar todas las palabras de comando BASIC y todas las indicaciones de error en español u otro idioma.
La tabla de las palabras de comando se encuentra a partir de la dirección &H3AF".

Con:

```
POKE &H3AB6,ASC("A")
POKE &H3AB7,ASC("D")
POKE &H3AB8,&HC5 ASC("E")+128
```

CLOAD es transformado a CLADE.
Para cargar un programa, ahora hay que introducir CLADE en lugar de CLOAD. CLOAD origina un Syntax Error. También con List obtendrá siempre CLADE en lugar de CLOAD.
Al código del último carácter de una palabra Key hay que sumar siempre 128. La tabla de palabras Key está ordenada alfabéticamente, en lo cual no se indica la primer letra. En lugar de la primera letra se encuentra un Token. Un Byte 0 significa el comienzo de la siguiente letra de comienzo. Observe Vd. la estructura de la tabla con el monitor (comienzo &H3A72).

La tabla de las indicaciones de error se encuentra a partir de la dirección &H3D75. Aquí están memorizadas las indicaciones de error por medio de sus códigos ASCII. Las diferentes indicaciones están separadas por Bytes 0.
Introduzca lo siguiente:

```
POKE &H3D8D,ASC("f")
POKE &H3D8E,ASC("a")
        .       l
        .       t
        .       a
```

Ahora obtendrá en lugar de "Syntax Error" un "Syntax falta".

9.7 INPUT sin ?

Si se ha copiado la ROM en la RAM, este problema está solucionado. También existe la posibilidad de sobreescribir la interrogación con un Patch.

```
10 FOR I=&HFDE0 TO &HFDE3
20 READ A
30 POKE I,A
40 NEXT
50 DATA &H21,&HD2,&H23,&HE3
```

Listado Assembler:

```
FDE0 210223     LD  HL,&H23D2; Input Patch &H23D2 es la
                               nueva dirección de salto
                               hacia atrás
FDE3 E3         EX  (SP),HL; sobre pila
FDE4 C9         RET
```

El estado anterior se puede conseguir con

```
POKE &HFDE0,&HC9
```

CAPITULO X: PROGRAMAS

10.1 El Generador de Menú

El Generador de Menú ofrece la posibilidad de construir fácilmente aquellos programas en los cuales se trabaje frecuentemente con Menús.
El generador de menú propiamente dicho está contenido en las líneas 50 hasta 480 y 10000 hasta el final.
Las líneas DATA al final del programa pueden ser modificadas según sus necesidades. El símbolo de exclamación (!) señaliza la última línea DATA.
La primera línea DATA (10270) ha de ser siempre menú principal, seguida de la línea 10280 DATA 0. A partir de esta línea Vd. puede introducir sus opciones del menú. El orden es el siguiente:

Después de DATA 1 se encuentra el submenú de la primera opción del menu principal (aquí: Imprimir).
Después de DATA 2 se encuentra el submenú de la segunda opción , etc. Si se ha asignado un submenú a cada opción del menu principal, entonces sigue el submenú (nivel 2) de la primera opción del primer nivel (aquí: a FORMAT pertenece DATA 6 y las siguientes) etc.

Lo especial es que no tiene que tener en cuenta las longitudes de los diferentes menús. La introducción de la forma descrita es más que suficiente.

Si quiere introducir su propio menú, tiene que escribir, naturalmente, las rutinas correspondientes. Para ello se utilizan las líneas 1000-9999.
La integración del generador de menú es esquemáticamente siempre igual (ver línea 1010 hasta 1040). En las líneas de meta en >ON GOSUB< se encuentra o la llamada del siguiente menú (según el mismo esquema), o una rutina ejecutiva. Esta debería finalizarse con >RETURN<.

El programa se basa en la administración de las opciones de menú por medio de un campo de indicadores MD(MP,I). MD(,) contiene informaciones sobre:

0 - RUCK : Número del último menú
1 - NEXT : Número del siguiente menú
2 - TAMAÑO : Cantidad de las inscripciones en el menú
3 - DEFAULT : Opción últimamente elegida en este menú

Para activar los puntos correspondientes, el Pointer de menú MP y la cifra característica de la información (0-3) se utilizan como indicador para MD. El String de menú y los campos de indicadores se producen todos por medio de las líneas 10200 hasta 10260.

```
10 REM Generador de menu
20 CLEAR 200,&HEFFF:MAXFILES=0
30 KEY OFF:SCREEN 0:WIDTH 38
40 DEFINT A-Z
50 GOSUB 10000 : REM Inicializacion
60 GOSUB 1000 : REM Menu principal
70 CLS: END
170 '
180 REM Subprogramas para menu
190 '
200 REM Margen
210 CLS:PRINT"Generador de menus    ";M$(MP)
220 PRINTSTRING$(38,"-")
230 LOCATE 0,21:PRINTSTRING$(38,"-");
240 PRINT"  >RETURN< para seleccion del inverso"
250 NU=MD(MP,RU):IF NU>1 THEN NU=1
260 PRINT"o >BS< para ";EX$(NU+1);
270 RETURN
290 '
300 REM Construir menu
310 GOSUB 200: REM Margen
320 WA=MD(MP,DL):REM Seleccion es Default
330 LOCATE 0,4
340 FOR I=1 TO MD(MP,GR):IF I=WA THEN POKE &HFDA4,&HC3:REM r
epresentacion inversa del punto seleccionado
350 PRINTI;:POKE IN,AU:PRINT" - ";:IF I=WA THEN POKE IN,AN
360 PRINT M$(I+MD(MP,NX)-1):POKE IN,AU:REM emitir punto del
menu
370 NEXT
380 PRINT:PRINT:PRINT"Su eleccion :";:POKE IN,AN
390 PRINTWA;:POKE IN,AU
400 LOCATE POS(0)-3
410 EI$=INKEY$:IF EI$="" THEN 410
420 IF EI$=ZR$ THEN RETURN
430 IF EI$=CHR$(13) THEN MD(MP,DL)=WA:MP=MD(MP,NX)+WA-1:RETU
RN
440 IF EI$=CHR$(28) OR EI$=CHR$(31) OR EI$=CHR$(32) THEN WA=
WA+1+(WA=MD(MP,GR))*MD(MP,GR):GOTO 480
450 IF EI$=CHR$(29) OR EI$=CHR$(30) THEN WA=WA-1-(WA=1)*MD(M
P,GR):GOTO 480
```

```
460 N=VAL(EI$):IF (N<1) OR (N>MD(MP,GR)) THEN 410
470 WA=N
480 POKE IN,AN:PRINTWA;:POKE IN,AU:GOTO330
990 '
1000 REM Menu principal
1010 GOSUB 300:REM Indicar menu
1020 IF EI$=ZR$ THEN RETURN
1030 ON WA GOSUB 1050,1310,1400,1490,1600
1040 GOTO 1010
1050 REM 1 Submenu nivel 1
1060 GOSUB 300:REM Construir menu
1070 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1080 ON WA GOSUB 1100,1170,1240
1090 GOTO 1050
1100 REM Submenu nivel 2
1110 GOSUB 300
1120 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1130 ON WA GOSUB 1140,2160,1150:RETURN:REM Fin de esta parte
 del menu
1140 REM
1150 REM
1160 RETURN
1170 REM Submenu nivel 2
1180 GOSUB 300
1190 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1200 ON WA GOSUB 1210,2260,1220:RETURN:REM Fin de esta parte
 del menu
1210 REM
1220 REM
1230 RETURN
1240 REM Submenu nivel 2
1250 GOSUB 300
1260 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1270 ON WA GOSUB 1280,2360,1290:RETURN:REM Fin de esta parte
 del menu
1280 REM
1290 REM
1300 RETURN
```

```
1310 REM 2 Submenu nivel 1
1320 GOSUB 300: REM Construir menu
1330 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1340 ON WA GOSUB 1360,3200,3300
1350 GOTO 1310
1360 REM Submenu nivel 2
1370 GOSUB 300
1380 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1390 ON WA GOSUB 3150,3160,3170:RETURN:REM Fin de esta parte
 del menu
1400 REM 2 Submenu nivel 1
1410 GOSUB 300:REM Construir menu
1420 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1430 ON WA GOSUB 1450,4200,4300
1440 GOTO 1400
1450 REM Submenu nivel 2
1460 GOSUB 300
1470 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1480 ON WA GOSUB 4150,4160,4170:RETURN:REM Fin de esta parte
 del menu
1490 ' etc.
1500 ' todas las lineas del programa de
1510 ' 1000 - 10000 solo sirven de ejemplo
1520 ' En estas lineas habra que
1530 ' confeccionar luego el programa
1540 ' propiamente dicho
1550 ' El programa ahora presente
1560 ' solo representa el esqueleto de
1570 ' cualquier programa que utilice
1580 ' menus extensos
1600 RETURN
2160 REM
2260 REM
2270 RETURN
2360 REM
2370 RETURN
3150 REM
3160 REM
```

```
3170 RETURN
3200 REM
3300 RETURN
4150 REM
4160 REM
4170 RETURN
4200 REM
4300 RETURN
10000 REM Init
10010 REM Cargar juego de caracteres inverso
10020 BA=BASE(2)+128*8
10030 FOR I=BA TO BA+128*8-1
10040 VPOKE I,VPEEK(I-128*8) XOR 255:NEXT
10050 REM Cargar patch inverso
10060 FOR I=&HF370 TO &HF37A:READ A$
10070 POKE I,VAL("&H"+A$):NEXT
10080 DATA E1,F1,FE,20,38,02,F6,80,F5,E5,C9
10090 POKE &HFDA5,&H70:POKE &HFDA6,&HF3
10100 IN=&HFDA4:AN=&HC3:AU=&HC9:REM Con/desc variables para
modo inverso
10110 ZR$=CHR$(8):REM Volver al codigo de teclado
10120 FOR I=0 TO 2:READ EX$(I):NEXT
10130 DATA Fin programa
10140 DATA Menu principal
10150 DATA Ultimo menu
10160 DIM M$(100),MD(100,3)
10170 RU=0:NX=1:REM md(i,ru) senala al menu anterior, md(i,n
x) al siguiente
10180 GR=2:DL=3:REM md(i,gr) tamano menu, md(i,dl) ultimo pu
nto seleccionado (Default)
10190 N=-1:I=-1:REM Contador
10200 I=I+1:READ M$(I):REM Leer punto del menu
10210 MD(I,RU)=N:MD(I,DL)=1:REM Ultimo menu y Default
10220 IF M$(I)="!" THEN I=I-1:MD(MD(I,RU),GR)=I-MD(MD(I,RU),
NX)+1:RETURN:REM Constatar fin de todos los menus
10230 IF LEN(M$(I))>2 THEN 10200:REM Si todavia no menu nuev
o (m$=zahl)
10240 N=VAL(M$(I)):MD(N,NX)=I:I=I-1:REM Activar valores para
 menu siguiente
```

```
10250 IF MD(I,RUCK)>=0 THEN MD(MD(I,RUCK),GR)=I-MD(MD(I,RU),
NX)+1:REM Tamano del ultimo menu
10260 GOTO 10200
10270 DATA "Menu principal
10280 DATA 0
10290 DATA imprimir
10300 DATA mostrar
10310 DATA editar
10320 DATA buscar
10330 DATA servicio
10340 DATA 1
10350 DATA formato
10360 DATA paginas
10370 DATA normal
10380 DATA 2
10390 DATA formateado
10400 DATA 40 columnas
10410 DATA 32 columnas
10420 DATA 3
10430 DATA bloque
10440 DATA borrar
10450 DATA 4
10460 DATA palabra buscada
10470 DATA numero pagina
10480 DATA 5
10490 DATA Floppy
10500 DATA cassette
10510 DATA 6
10520 DATA formato estandar
10530 DATA entrada especial
10540 DATA sin formulario
10550 DATA !
```

10.2 Textomat mini

Una de las aplicaciones más importantes del ordenador es el procesado de textos. Para transmitirle una idea de las cualidades de los ordenadores MSX en el campo de los ordenadores personales, hemos escrito un programa para elaboración de textos con el MSX.
El manejo del programa es en principio igual que si se trabajara sobre la pantalla BASIC (al programar). Esto significa, que >INS<, >DEL< y >BS< actúan en el BASIC siempre sobre una línea, en el programa de texto sobre un párrafo completo.
Un párrafo se origina pulsando simultáneamente >SHIFT< y >RETURN<. Una flecha pequeña señaliza el final del párrafo. Si han de escribirse espacios libres en el texto, éstos han de producirse también con las teclas >SHIFT< y >RETURN<. La misma comnbinación de teclas borra también el resto de una línea.

Las teclas de cursor y >RETURN< (sin >SHIFT<) se utilizan para moverse libremente en el texto. Con >CTRL<+ Cursor Vd. puede saltar pantalla a pantalla a través del texto.
Al final del texto se ha de introducir siempre >GRAPH"+P< (ha de estar sólo en una línea). Este carácter, que tiene el mismo aspecto que el cursor, finaliza la edición de texto al imprimir.
Con <>ESC<, Vd. abandona el programa. Si al volver a iniciar el programa, Vd. introduce "no" en la función de borrado, entonces se cosnerva el texto anterior.
Además, por medio de las teclas de función se pueden llamar las rutinas Impresión/ Save/ Load.

Con Save/Load se pueden memorizar/cargar programas. Al imprimir, el formateo de las páginas se indica por medio de LR,AS,OB,AZ (línea 2510,2520). En un caso dado se han de modificar estos valores.
A continuación del listado BASIC encontrará el listado Assembler de las rutinas de máquina utilizadas en el programa de texto.

```
10 ' Minitextomat para MSX de H. Dullin
20 CLEAR 200,&HE6FF:REM Reservar memoria para textos
30 GOSUB 2000:REM Inicializacion
40 INPUT "Borrar memoria ?";A$
50 IF (ASC(A$)OR2^5)=ASC("s") THEN X=USR7(1):CLS ELSE X=USR1
(TB)
60 A$=""
70 LOCATE 0,0,1
80 ' Bucle principal de entrada
90 A$=INKEY$:IF A$="" THEN 90
100 GOSUB 120:REM Utilizar entrada
110 GOTO 80 :REM Para nueva entrada
120 REM Utilizar entrada
130 AF=0:REM Flag del Scroll
140 IF A$>=" " THEN 290:REM Entrada de caracteres estandar
150 A=ASC(A$):REM Codigo de control
160 IF A>26 THEN ON A-26 GOTO 230,390,430,510,580:REM Teclas
 Escape y de cursor
170 IF A=13 THEN 650:REM Return
180 IF A=11 THEN LOCATE 0,0:RETURN:REM Home
190 IF A=12 THEN X=USR1(TB):ZU=0:LOCATE 0,0,1:RETURN:REM Shi
ft Home
200 IF A=8 THEN A$=CHR$(29):GOSUB 430:X=USR3(TB+(ZU+CSRLIN)*
SM+POS(0)+1):X=USR1(TB+ZU*SM):RETURN:REM Back Space
210 IF A=18 THEN POKE &HFCAA,1+(PEEK(&HFCAA)=1):LOCATE ,,1:R
ETURN:REM Tecla Insert
220 RETURN: REM Volver a entrada
230 REM Fin de programa
240 CLS:LOCATE 0,0,0
250 ON ERROR GOTO 0:REM Desconectar On Error
260 POKE &HFCAA,0:REM Desactivar Flag Insert
280 END
290 REM Entrada de letras estandar
300 IF ASC(A$)=127 THEN X=USR3(TB+(ZU+CSRLIN)*SM+POS(0)+1):X
=USR1(TB+ZU*SM):RETURN: REM Delete
310 IF POS(0)=SM-1 AND CSRLIN=23-ZO THEN AF=-1:REM Averiguar
 si se presisa Scroll
```

```
320 IF PEEK(&HFCAA)=1 THEN GOSUB 750: REM Si Insert conectad
o, 1600
330 POKE TB+(ZU+CSRLIN)*SM+POS(0),ASC(A$): REM Grabar codigo
 de letras
340 PRINTA$;:REM Emitir letras
350 IF POS(0)<>0 THEN RETURN:REM Terminado si no Scroll
360 IF ZU+CSRLIN+1>=ZM THEN BEEP:AF=-1:ZU=ZU-1:REM Test memo
ria llena
370 IF AF THEN ZU=ZU+1:X=USR1(TB+ZU*SM): REM Scroll
380 RETURN
390 REM Cursor derecha
400 IF POS(0)=SM-1 AND CSRLIN=23-ZO THEN AF=-1:A$=CHR$(13)+C
HR$(10):REM Si Scroll
410 PRINTA$;
420 GOTO 350
430 REM Cursor izquierda
440 IF POS(0)=0 AND CSRLIN=0 THEN LOCATE SM-1:GOTO 460:REM S
i Scroll haca atras
450 PRINTA$;:RETURN
460 REM Scroll hacia atras
470 IF ZU=0 THEN LOCATE 0:BEEP:RETURN
480 ZU=ZU-1:REM Disminuir contador de lineas de acarreo
490 X=USR1(TB+ZU*SM):REM Indicar imagen
500 RETURN
510 REM Cursor arriba
520 IF (PEEK(&HFBEB)AND2)=0 THEN 550:REM CTRL pulsada ?
530 IF CSRLIN=0 THEN 460:REM Scroll hacia atras
540 PRINTA$;:RETURN
550 REM CTRL cursor arriba
560 ZU=ZU-24+ZO:IF ZU<0 THEN ZU=0:REM Si principio de texto
570 X=USR1(TB+ZU*SM):RETURN
580 REM Cursor abajo
590 IF (PEEK(&HFBEB)AND2)=0 THEN 620:REM CTRL ?
600 IF CSRLIN=23-ZO THEN AF=-1:PRINTCHR$(10);:GOTO 360:REM S
croll
610 PRINTA$;:RETURN
```

```
620 REM CTRL Cursor abajo
630 J=ZU+24-ZO:IF J<ZM-24+ZO THEN ZU=J ELSE ZU=ZM-24+ZO:REM
Final del texto ?
640 X=USR1(TB+ZU*SM):RETURN
650 REM Return => Cursor abajo
660 IF (PEEK(&HFBEB)AND1)=0 THEN 690:REM SHIFT pulsada ?
670 IF CSRLIN=23-ZO THEN AF=-1:PRINTA$;CHR$(10);:GOTO 360:RE
M Scroll ?
680 PRINTA$;CHR$(10);:RETURN
690 REM Shift RETURN
700 I=TB+(ZU+CSRLIN)*SM:REM Direccion inicial de la actual l
inea de la memoria
710 FOR J=I+POS(0) TO I+SM-1:POKE J,32:NEXT:REM Borrar resto
 de la linea
720 POKE I+POS(0),208:REM Shift Return grabar caracter (flec
ha)
730 GOSUB 670:X=USR1(TB+ZU*SM):REM Ejecutar Return e indicar
 texto
740 RETURN
750 REM INSERT
760 X=USR2(TB+(ZU+CSRLIN)*SM+POS(0)-1):REM Asignar espacio l
ibre de la memoria
770 X=USR1(TB+ZU*SM):REM Indicar texto desplazado
780 RETURN:REM Para emision normal de caracteres
1000 REM salida de impresion
1010 CLS
1020 Y$=STRING$(OB,10):REM String para avance de linea princ
ipio de linea
1030 AA=TB:REM Direccion de memoria
1040 ZZ=0:REM Contador de lineas
1050 X$=STRING$(LR,32):REM Crear string para margen izquierd
o
1060 REM Principio de pagina
1070 LPRINT Y$;:REM Avance de linea
1080 REM Emision de linea
1090 Z$="":REM Linea actual
1100 VP=VARPTR(Z$)
```

```
1110 POKE VP,AS:REM Longitud de la linea
1120 POKE VP+1,AA-INT(AA/256)*256:REM Direccion de la linea
en la memoria de texto
1130 POKE VP+2,INT(AA/256)-256*(AA<0)
1140 AA=AA+AS:REM Direccion hacia linea siguiente
1150 PO=INSTR(Z$,E$):REM GRAPH P ?, significa final de texto
1160 IF PO<>0 OR AA>&HF1FF THEN LPRINT STRING$(PZ-OB-AZ,10):
POKE &HF3DE,257-ZO:X=USR1(TB):RETURN 80:REM Si final de text
o
1170 PO=INSTR(Z$,CR$):REM SHIFT RETURN en la linea ?
1180 IF PO=0 THEN 1240: REM No
1190 REM Shift Return
1200 POKE VP,PO-1:REM Linea de impresion hasta SHIFT RETURN
1210 AA=AA-AS+PO:REM Direccion hacia caracter tras Shift Ret
urn
1220 IF PEEK(AA)<>32 THEN 1310 ELSE AA=AA+1:GOTO 1220:REM Re
correr espacios siguientes
1230 GOTO 1310
1240 REM Buscar espacio que precede a palabra siguiente
1250 IF PEEK(AA)=32 THEN AA=AA+1:GOTO 1310
1260 I=0:REM Contador hasta espacio
1270 AA=AA-1:I=I+1:REM Buscar atras en la linea actual
1280 IF PEEK(AA)<>32 THEN 1270:REM Si no espacio, continuar
buscando
1290 POKE VP,PEEK(VP)-I:REM Imprimir linea hasta la ultima p
alabra "adaptable"
1300 AA=AA+1:REM Direccion hacia principio de palabra siguie
nte
1310 LPRINTX$;Z$:REM Emitir margen izquierdo y linea
1320 ZZ=ZZ+1:REM Incrementar contador de lineas
1330 IF ZZ=AZ THEN LPRINT STRING$(PZ-OB-AZ,10):GOTO 1060:REM
 Fin de pagina
1340 GOTO 1080:REM Linea siguiente
1350 REM Grabar
1360 X=USR4(1):REM Borrar Keys
1370 POKE &HF3DE,0:POKE &HF3B1,25
1380 LOCATE 0,22,1
1390 INPUT"Nombre :";N$
```

```
1400 BSAVE "CAS:"+N$,TB,&HF1FF
1410 LOCATE 0,0,1
1420 POKE &HF3DE,257-ZO:POKE &HF3B1,23
1430 X=USR5(1):REM Indicar Keys
1440 RETURN 80
1450 REM Cargar
1460 X=USR4(1):REM Borrar Keys
1470 POKE &HF3DE,0:POKE &HF3B1,25:REM AMPLIAR VENTANA DE TEX
TO
1480 LOCATE 0,22,1
1490 INPUT"Nombre :";N$
1500 BLOAD "CAS:"+N$
1510 LOCATE 0,0,1
1520 POKE &HF3DE,257-ZO:POKE &HF3B1,23
1530 X=USR5(1):REM Indicar Keys
1540 RETURN 80
2000 REM Inicializacion
2010 SCREEN 0
2020 COLOR 14,4,4
2030 DEFINT A-Z
2040 TB=&HE700:REM Direccion basica de texto
2050 SM=36:WIDTH SM:REM Anchura de lineas
2060 ZM=INT((&HF1FF-TB)/SM):REM Cantidad max. de lineas
2070 ZO=3:REM Offset de lineas margen inferior
2080 POKE &HF3B1,23:REM Ultima linea de pantalla (emision Ke
y)
2090 ON STOP GOSUB 230
2100 STOP ON
2110 ON ERROR GOTO 230
2120 REM Cargar Mapro para indicacion de texto
2130 FOR I=&HF300 TO &HF326:READ A$:POKE I,VAL("&H"+A$):NEXT
2140 DATA 23,23,5E,23,56,3A,B0,F3
2150 DATA 4F,06,00,CD,32,0C,21,01,01,F5
2160 DATA E5,D5,CD,F2,0B,D1,C5,CD
2170 DATA 45,07,C1,E1,23,F1,3D,20
2180 DATA EE,CD,ED,09,C9
2190 DEFUSR1=&HF300
2200 REM Cargar Mapro para Insert
```

```
2210 FOR I=&HF330 TO &HF371:READ A$:POKE I,VAL("&H"+A$):NEXT
2220 DATA 23,23,7E,23,66,6F,E5,11
2230 DATA FF,F1,EB,B7,ED,52,E3,C1,E5
2240 DATA 3E,D0,ED,B1,D1,E5,B7,ED
2250 DATA 52,E3,7E,FE,20,28,1A,E5
2260 DATA 21,FF,F1,3A,B0,F3,5F,16
2270 DATA 00,E5,19,D1,EB,03,ED,B8,47,23,3E,20
2280 DATA 77,23,10,FC,E1,C1,54,5D
2290 DATA 1B,EB,ED,B8,C9
2300 DEFUSR2=&HF330
2310 REM Cargar Mapro para Delete/Backspace
2320 FOR I=&HF2B0 TO &HF2F0:READ A$:POKE I,VAL("&H"+A$):NEXT
2330 DATA 23,23,7E,23,66,6F,E5,11
2340 DATA FF,F1,EB,B7,ED,52,E3,C1,E5
2350 DATA 3E,D0,ED,B1,D1,B7,ED,52
2360 DATA 44,4D,62,6B,1B,ED,B0,2B
2370 DATA 36,20,E5,3A,B0,F3,47,7E
2380 DATA 23,FE,20,20,11,10,F8,EB,21
2390 DATA FF,F1,B7,ED,52,44,4D,E1
2400 DATA 00,EB,ED,B0,C9,E1,C9
2410 DEFUSR3=&HF2B0
2420 REM Borrar cargar memoria Mapro
2430 FOR I=&HF280 TO &HF28D:READ A$:POKE I,VAL("&H"+A$):NEXT
2440 DATA 21,00,E7,36,20,11,01,E7
2450 DATA 01,FF,1A,ED,B0,C9
2460 DEFUSR7=&HF280
2470 ON KEY GOSUB 1000,1350,1450
2480 KEY 1,"impr":KEY (1) ON
2490 KEY 2,"Save":KEY (2) ON
2500 KEY 3,"load":KEY (3) ON
2510 LR=10:AS=60:REM Margen izquierdo, cantidad columnas par
a imprimir
2520 OB=5:AZ=60:PZ=72:REM Formulario principio superior, can
tidad lineas de impresion, cantidad lineas por pagina impres
a
2530 KEY ON:CLS:LOCATE 0,0,1:POKE &HF3DE,256-ZO+1:REM Offset
 final de imagen hacia linea Key
```

```
2540 E$=CHR$(219):CR$=CHR$(208):REM Final caracteres (Graph
P) y caracter Shift Return
2550 DEFUSR4=&HB15:REM Borrar Keys
2560 DEFUSR5=&HB2B:REM Indicar Keys
2570 RETURN
```

```
F000          10                ; Rutina Insert
F000          20  ENDE    EQU   &HF1FF
F000          30                ; Transmisión: Dirección RAM
F000 23       40          INC   HL
F001 23       50          INC   HL
F002 7E       60          LD    A,(HL)
F003 23       70          INC   HL
F004 66       80          LD    H,(HL)
F005 6F       90          LD    L,A
F006 E5       100         PUSH  HL ; Actual dirección RAM
F007 11FFF1   110         LD    DE,ENDE ; Final de texto
F00A EB       120         EX    DE,HL
F00B B7       130         OR    A ; Borrar acarreo
F00C ED52     140         SBC   HL,DE ; Cantidad hasta final
de texto
F00E E3       150         EX    (SP),HL
F00F C1       160         POP   BC ; Cantidad
F010 E5       170         PUSH  HL ; Dirección RAM
F011 3ED0     180         LD    A,208 ; Shift Return
F013 EDB1     190         CPIR  ; buscar
F015 D1       200         POP   DE ; Dirección RAM
F016 E5       210         PUSH  HL ; Dirección de Shift
Return +1 = Dirección de meta final
F017 B7       220         OR    A ; Acarreo = 0
F018 ED52     230         SBC   HL,DE ; Cantidad  de Bytes a
desplazar
F01A E3       240         EX    (SP),HL
F01B 7E       250         LD    A,(HL)
F01C FE20     260         CP    32 ; ¿Espacio?
?F01E 28FE    270         JR    Z,OK ;  sí, entonces hay más
sitio
F020 E5       280         PUSH  HL ; Dirección de meta final
F021 21FFF1   290         LD    HL,ENDE
F024 3AB0F3   300         LD    A,(&HF3B0) ; Width
F027 5F       310         LD    E,A
F028 1600     320         LD    D,0
F02A E5       330         PUSH  HL ; Final de texto
F02B 19       340         ADD   HL,DE ; más Width
F02C D1       350         POP   DE
F02D EB       360         EX    DE,HL
F02E 03       370         INC   BC
```

```
 F02F EDB8    380          LDDR  ; Desplazar  texto restante
completo
 F031 47      390          LD    B,A
 F032 3E20    400          LD    A,&H20
 F034 77      410  NEXT    LD    (HL),A ; Borrar línea libera
da
 F035 23      420          INC   HL
 F036 10FC    430          DJNZ  NEXT
 F038 E1      440          POP   HL
**** línea 270 : OK=&HF039 Offset 19
 F039 C1      450  OK      POP   BC
 F03A 54      460          LD    D,H
 F03B 5D      470          LD    E,L
 F03C 1B      480          DEC   DE
 F03D EB      490          EX    DE,HL
 F03E EDB8    500          LDDR  ; desplazar línea hasta Shi
ft Return en 1
 F040 C9      510          RET
```

```
Programa: insert
Inicio: &HF000    Final: &HF040
Longitud: &H41 Bytes
Errores: 0
Tabla de variables:
ENDE   F1FF   NEXT   FF8A
OK     F039
```

```
F000          10                ; Rutina Delete
F000          20                ; TRANSMISION:DIRECCION RAM
F000          30  ENDE    EQU   &HF1FF
F000 23       40          INC   HL
F001 23       50          INC   HL
F002 7E       60          LD    A,(HL)
F003 23       70          INC   HL
F004 66       80          LD    H,(HL)
F005 6F       90          LD    L,A
F006 E5       100         PUSH  HL ; Dirección RAM
F007 11FFF1   110         LD    DE,ENDE
F00A EB       120         EX    DE,HL
F00B B7       130         OR    A
F00C ED52     140         SBC   HL,DE ; Cantidad hasta final
de texto
F00E E3       150         EX    (SP),HL
F00F C1       160         POP   BC ; Cantidad
F010 E5       170         PUSH  HL ; Dirección RAM
F011 3ED0     180         LD    A,208 ; Shift Return
F013 EDB1     190         CPIR  ; buscar
F015 D1       200         POP   DE
F016 B7       210         OR    A
F017 ED52     220         SBC   HL,DE ; Cantidad  de Bytes a
desplazar
F019 44       230         LD    B,H
F01A 4D       240         LD    C,L
F01B 62       250         LD    H,D
F01C 6B       260         LD    L,E
F01D 1B       270         DEC   DE ; -1 = Dirección de meta
F01E EDB0     280         LDIR
F020 2B       290         DEC   HL ; -1 = vieja dirección de
Shift Return
F021 3620     300         LD    (HL),&H20 ; Borrar
F023 E5       310         PUSH  HL
F024 3AB0F3   320         LD    A,(&HF3B0) ; Width
F027 47       330         LD    B,A
F028 7E       340  PRUEF  LD    A,(HL)
F029 23       350         INC   HL
F02A FE20     360         CP    32 ; ¿Espacio?
?F02C 20FE    370         JR    NZ,ZRUCK ; No, entonces fin
F02E 10F8     380         DJNZ PRUEF ; Comprobar línea comp
```

```
leta
 F030 EB      390          EX   DE,HL ; si línea sólo con es
pacios, entonces borrar
 F031 21FFF1  400          LD   HL,ENDE
 F034 B7      410          OR   A
 F035 ED52    420          SBC  HL,DE
 F037 44      430          LD   B,H
 F038 4D      440          LD   C,L ; Cantidad
 F039 E1      450          POP  HL ; Dirección viejo Shift R
eturn
 F03A 2B      460          DEC  HL
 F03B EB      470          EX   DE,HL
 F03C EDB0    480          LDIR  ; Borrar línea posiblemente
liberada
 F03E C9      490          RET
**** línea 370 : ZRUCK=&HF03F Offset 11
 F03F E1      500  ZRUCK   POP  HL
 F040 C9      510          RET
```

```
Programa: delete
Inicio: &HF000    Final: &HF040
Longitud: &H41 Bytes
Errores: 0
Tabla de variables:
ENDE   F1FF   PRUEF  F028
ZRUCK  F03F
```

```
F000          10                ; Rutina Visualización texto
F000          20                ; TRANSMISION: DIRECCION RAM
de la primera letra a indicar
F000          30  ENDE    EQU   &HF1FF
F000 23       40            INC   HL
F001 23       50            INC   HL
F002 5E       60            LD    E,(HL)
F003 23       70            INC   HL
F004 56       80            LD    D,(HL)
F005 3AB0F3   85            LD    A,(&HF3B0) ; Width
F008 4F       90            LD    C,A
F009 0600     100           LD    B,0 ; BC es contador de colu
mnas
F00B CD320C   110           CALL  &H0C32 ; Carga cantidad de
líneas por pantalla en acumulador
F00E 210101   120           LD    HL,&H0101 ; columna/línea
F011 F5       130 NEXT      PUSH  AF
F012 E5       140           PUSH  HL
F013 D5       150           PUSH  DE
F014 CDF20B   160           CALL  &H0BF2 ; Calcular dirección
de VRAM a partir columna/línea (HL)
F017 D1       170           POP   DE ; Dirección RAM
F018 C5       180           PUSH  BC ; Caracteres por línea
F019 CD4507   190           CALL  &H0745 ; Rutina carga de blo
ques RAM ->VRAM
F01C C1       200           POP   BC ; Cantidad por línea
F01D E1       210           POP   HL ; Posición de cursor
F01E 23       220           INC   HL ; hacia línea siguiente
F01F F1       230           POP   AF ; Contador de líneas
F020 3D       240           DEC   A ; reducir
F021 20EE     250           JR    NZ,NEXT ; todavía no cero,
entonces continuar
F023 CDED09   260           CALL  &H09ED ; Volver a visualizar
cursor
F026 C9       270           RET   ; volver al BASIC
```

```
Programa: visual
Inicio: &HF000    Final: &HF026
Longitud: &H27 Bytes
Errores: 0
Tabla de variables:
ENDE   F1FF   NEXT  F011
```

```
F000            10                  ; Borrar sector de texto
F000            20              LD  HL,&HE700 ; Principio sector
de texto
F003 3620       30              LD  (HL),32 ; borrar
F005 1101E7     40              LD  DE,&HE701 ; Dirección meta
F008 01FF1A     50              LD  BC,&H1AFF ; Cantidad=&Hf200-
&He701
F00B EDB0       60              LDIR  ; Borrar sector
F00D C9         70              RET
```

```
Programa: borrad
Inicio: &HF000    Final: &HF00D
Longitud: &HE Bytes
Errores: 0
```

Tabla de transformación Decimal-Hexadecimal-Binario

Decimal	Hex	Binario	Decimal	Hex	Binario
0	&H00	&B00000000	26	&H1A	&B00011010
1	&H01	&B00000001	27	&H1B	&B00011011
2	&H02	&B00000010	28	&H1C	&B00011100
3	&H03	&B00000011	29	&H1D	&B00011101
4	&H04	&B00000100	30	&H1E	&B00011110
5	&H05	&B00000101	31	&H1F	&B00011111
6	&H06	&B00000110	32	&H20	&B00100000
7	&H07	&B00000111	33	&H21	&B00100001
8	&H08	&B00001000	34	&H22	&B00100010
9	&H09	&B00001001	35	&H23	&B00100011
10	&H0A	&B00001010	36	&H24	&B00100100
11	&H0B	&B00001011	37	&H25	&B00100101
12	&H0C	&B00001100	38	&H26	&B00100110
13	&H0D	&B00001101	39	&H27	&B00100111
14	&H0E	&B00001110	40	&H28	&B00101000
15	&H0F	&B00001111	41	&H29	&B00101001
16	&H10	&B00010000	42	&H2A	&B00101010
17	&H11	&B00010001	43	&H2B	&B00101011
18	&H12	&B00010010	44	&H2C	&B00101100
19	&H13	&B00010011	45	&H2D	&B00101101
20	&H14	&B00010100	46	&H2E	&B00101110
21	&H15	&B00010101	47	&H2F	&B00101111
22	&H16	&B00010110	48	&H30	&B00110000
23	&H17	&B00010111	49	&H31	&B00110001
24	&H18	&B00011000	50	&H32	&B00110010
25	&H19	&B00011001	51	&H33	&B00110011

Tabla de transformación Decimal-Hexadecimal-Binario

Decimal	Hex	Binario	Decimal	Hex	Binario
52	&H34	&B00110100	78	&H4E	&B01001110
53	&H35	&B00110101	79	&H4F	&B01001111
54	&H36	&B00110110	80	&H50	&B01010000
55	&H37	&B00110111	81	&H51	&B01010001
56	&H38	&B00111000	82	&H52	&B01010010
57	&H39	&B00111001	83	&H53	&B01010011
58	&H3A	&B00111010	84	&H54	&B01010100
59	&H3B	&B00111011	85	&H55	&B01010101
60	&H3C	&B00111100	86	&H56	&B01010110
61	&H3D	&B00111101	87	&H57	&B01010111
62	&H3E	&B00111110	88	&H58	&B01011000
63	&H3F	&B00111111	89	&H59	&B01011001
64	&H40	&B01000000	90	&H5A	&B01011010
65	&H41	&B01000001	91	&H5B	&B01011011
66	&H42	&B01000010	92	&H5C	&B01011100
67	&H43	&B01000011	93	&H5D	&B01011101
68	&H44	&B01000100	94	&H5E	&B01011110
69	&H45	&B01000101	95	&H5F	&B01011111
70	&H46	&B01000110	96	&H60	&B01100000
71	&H47	&B01000111	97	&H61	&B01100001
72	&H48	&B01001000	98	&H62	&011000010
73	&H49	&B01001001	99	&H63	&B01100011
74	&H4A	&B01001010	100	&H64	&B01100100
75	&H4B	&B01001011	101	&H65	&B01100101
76	&H4C	&B01001100	102	&H66	&B01100110
77	&H4D	&B01001101	103	&H67	&B01100111

Tabla de transformación Decimal-Hexadecimal-Binario

Decimal	Hex	Binario	Decimal	Hex	Binario
104	&H68	&B01101000	130	&H82	&B10000010
105	&H69	&B01101001	131	&H83	&B10000011
106	&H6A	&B01101010	132	&H84	&B10000100
107	&H6B	&B01101011	133	&H85	&B10000101
108	&H6C	&B01101100	134	&H86	&B10000110
109	&H6D	&B01101101	135	&H87	&B10000111
110	&H6E	&B01101110	136	&H88	&B10001000
111	&H6F	&B01101111	137	&H89	&B10001001
112	&H70	&B01110000	138	&H8A	&B10001010
113	&H71	&B01110001	139	&H8B	&B10001011
114	&H72	&B01110010	140	&H8C	&B10001100
115	&H73	&B01110011	141	&H8D	&B10001101
116	&H74	&B01110100	142	&H8E	&B10001110
117	&H75	&B01110101	143	&H8F	&B10001111
118	&H76	&B01110110	144	&H90	&B10010000
119	&H77	&B01110111	145	&H91	&B10010001
120	&H78	&B01111000	146	&H92	&B10010010
121	&H79	&B01111001	147	&H93	&B10010011
122	&H7A	&B01111010	148	&H94	&B10010100
123	&H7B	&B01111011	149	&H95	&B10010101
124	&H7C	&B01111100	150	&H96	&B10010110
125	&H7D	&B01111101	151	&H97	&B10010111
126	&H7E	&B01111110	152	&H98	&B10011000
127	&H7F	&B01111111	153	&H99	&B10011001
128	&H80	&B10000000	154	&H9A	&B10011010
129	&H81	&B10000001	155	&H9B	&B10011011

Tabla de transformación Decimal-Hexadecimal-Binario

Decimal	Hex	Binario	Decimal	Hex	Binario
156	&H9C	&B10011100	182	&HB6	&B10110110
157	&H9D	&B10011101	183	&HB7	&B10110111
158	&H9E	&B10011110	184	&HB8	&B10111000
159	&H9F	&B10011111	185	&HB9	&B10111001
160	&HA0	&B10100000	186	&HBA	&B10111010
161	&HA1	&B10100001	187	&HBB	&B10111011
162	&HA2	&B10100010	188	&HBC	&B10111100
163	&HA3	&B10100011	189	&HBD	&B10111101
164	&HA4	&B10100100	190	&HBE	&B10111110
165	&HA5	&B10100101	191	&HBF	&B10111111
166	&HA6	&B10100110	192	&HC0	&B11000000
167	&HA7	&B10100111	193	&HC1	&B11000001
168	&HA8	&B10101000	194	&HC2	&B11000010
169	&HA9	&B10101001	195	&HC3	&B11000011
170	&HAA	&B10101010	196	&HC4	&B11000100
171	&HAB	&B10101011	197	&HC5	&B11000101
172	&HAC	&B10101100	198	&HC6	&B11000110
173	&HAD	&B10101101	199	&HC7	&B11000111
174	&HAE	&B10101110	200	&HC8	&B11001000
175	&HAF	&B10101111	201	&HC9	&B11001001
176	&HB0	&B10110000	202	&HCA	&B11001010
177	&HB1	&B10110001	203	&HCB	&B11001011
178	&HB2	&B10110010	204	&HCC	&B11001100
179	&HB3	&B10110011	205	&HCD	&B11001101
180	&HB4	&B10110100	206	&HCE	&B11001110
181	&HB5	&B10110101	207	&HCF	&B11001111

Tabla de transformación Decimal-Hexadecimal-Binario

Decimal	Hex	Binario	Decimal	Hex	Binario
208	&HD0	&B11010000	234	&HEA	&B11101010
209	&HD1	&B11010001	235	&HEB	&B11101011
210	&HD2	&B11010010	236	&HEC	&B11101100
211	&HD3	&B11010011	237	&HED	&B11101101
212	&HD4	&B11010100	238	&HEE	&B11101110
213	&HD5	&B11010101	239	&HEF	&B11101111
214	&HD6	&B11010110	240	&HF0	&B11110000
215	&HD7	&B11010111	241	&HF1	&B11110001
216	&HD8	&B11011000	242	&HF2	&B11110010
217	&HD9	&B11011001	243	&HF3	&B11110011
218	&HDA	&B11011010	244	&HF4	&B11110100
219	&HDB	&B11011011	245	&HF5	&B11110101
220	&HDC	&B11011100	246	&HF6	&B11110110
221	&HDD	&B11011101	247	&HF7	&B11110111
222	&HDE	&B11011110	248	&HF8	&B11111000
223	&HDF	&B11011111	249	&HF9	&B11111001
224	&HE0	&B11100000	250	&HFA	&B11111010
225	&HE1	&B11100001	251	&HFB	&B11111011
226	&HE2	&B11100010	252	&HFC	&B11111100
227	&HE3	&B11100011	253	&HFD	&B11111101
228	&HE4	&B11100100	254	&HFE	&B11111110
229	&HE5	&B11100101	255	&HFF	&B11111111
230	&HE6	&B11100110			
231	&HE7	&B11100111			
232	&HE8	&B11101000			
233	&HE9	&B11101001			

COMMODORE

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.

64 en el campo de la Técnica y la Ciencia. 361 págs. P.V.P. 2.800,- 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.

Todo sobre el Floppy 1541. Precio venta 3.200 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 Fast Tape. Además otras indicaciones y programas de utilidad (ajuste de cabezales, altavoz de control).

El Manual del Cassette. 196 pág. P.V.P. 1.600,- 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 simulador de paso a paso escrito en BASIC.

Lenguaje máquina para Commodore 64. 1984, 201 pág. P.V.P. 2.200,- ptas.

CONSEJOS Y TRUCOS, con más de 70.000 ejemplares vendidos en Alemania, es uno de los libros más vendidos de DATA BECKER. Es una colección muy interesante de ideas para la programación del Commodore 64, de POKEs y útiles rutinas e interesantes programas. Todos los programas en lenguaje máquina con programas cargadores en Basic.

64 Consejos y Trucos. 1984, 364 pág. P.V.P. 2.800,- ptas.

Este libro, contiene muchos interesantes programas de aprendizaje para solucionar problemas, descritos detalladamente y de manera fácilmente comprensible. Temas: 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.

Manual escolar para su Commodore 64. 389 págs. P.V.P. 2.800,- ptas.

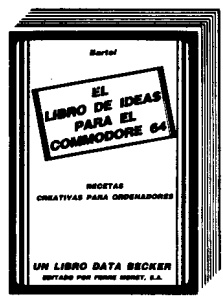
En el libro de los robots se muestran las asombrosas posibilidades que ofrece el CBM 64, para el control y la programación, presentadas con numerosas ilustraciones e intuitivos ejemplos. El punto principal: Cómo puede construirse uno mismo un robot sin grandes gastos. Además, un resumen del desarrollo 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.

Robótica para su Commodore 64. 340 págs. 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.

Mantenimiento y reparación del Floppy 1541. 325 págs. P.V.P. 2.800,- ptas.

Este es el libro que buscaba: un diccionario general de micros que contiene toda la terminología informática de la A a la Z y un diccionario técnico con traducciones de los términos ingleses de más importancia - los DICCIONARIOS DATA BECKER prácticamente son tres libros en uno. La increíble cantidad de información que contienen, no sólo los convierte en enciclopedias altamente competente, sino también en herramientas indispensables para el trabajo. El DICCIONARIO DATA BECKER se edita en versión especial para APPLE II, COMMODORE 64 e IBM PC.
El diccionario para su Commodore 64. 350 pág. P.V.P. 2.800,- ptas.



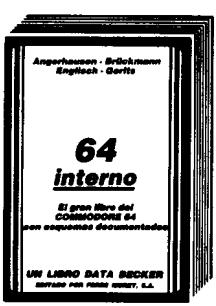
Casi todo lo que se puede hacer con el Commodore 64, está descrito detalladamente en este libro. Su lectura no es tan sólo tan apasionante como la de una novela, sino que contiene, además de listados de útiles programas, sobre todo muchas, muchas aplicaciones realizables en el C64. En parte hay listados de programas listos para ser tecleados, siempre que ha sido posible condensar «recetas» en una o dos páginas. Si hasta el momento no sabía que hacer con su Commodore 64, ¡después de leer este libro lo sabrá seguro!
El libro de ideas del Commodore 64. 1984, más de 200 pàginas, P.V.P. 1.600,- 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.
Lenguaje máquina para avanzados CBM 64. 1984, 206 pàg. P.V.P. 2.200 ptas.

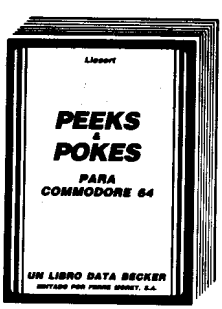
Este libro ofrece una amplia práctica introducción en el importante tema de la gestión de ficheros y bancos de datos, especialmente para los usuarios del Commodore 64. Con muchas interesantes rutinas y una confortable gestión de ficheros.
Todo sobre bases de datos y gestión de ficheros para Commodore-64. 221 págs. P.V.P. 2.200,- ptas.

Gráficos para el Commodore 64 es un libro para todos los que quieren hacer algo creativo con su ordenador. El contenido abarca desde los fundamentos de la programación de gráficos hasta el diseño asistido por ordenador (CAD).
Gráficos para el Commodore 64. 295 pags. P.V.P. 2.200,- ptas.

Para los usuarios que posean un VIC-20, C-64 o PC-128 este libro contiene gran cantidad de consejos, trucos, listados de programas, así como información sobre Hardware, tanto si usted dispone de una impresora de margarita o de matriz, como si tiene un Plotter VC-1520, el GRAN LIBRO DE IMPRESORAS constituye una inestimable fuente de información.
Todo sobre impresoras. 361 págs. P.V.P. 2.800,- ptas.



Con más de 60.000 ejemplares vendidos, ésta es la obra estándar para el COMMODORE 64. Todo sobre la tecnología, el sistema operativo y la programación avanzada del C-64. Con listado completo y exhaustivo de la ROM, circuitos originales documentados y muchos programas. ¡Conozca su C-64 a fondo!
64 interno. 1984, 352 pág. P.V.P. 3.800,- 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. 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 de interrupt. Además una introducción al lenguaje máquina. Muchos programas ejemplo.
PEEKS y POKES. 177 pág. P.V.P. 1.600,- ptas.

Este libro presenta una detallada e interesante introducción a la teoria, conceptos básicos y posibilidades de uso de la inteligencia artificial (1A). Desde un resumen histórico sobre las máquinas «pensantes» y «vivientes» hasta programas de aplicación para el Commodore 64.

Inteligencia artificial. 395 págs. 2.800,- ptas.

64, Consejos y Trucos vol. 2 contiene una gran profusión de programas, estímulos y muchas rutinas útiles. Un libro que constituye una ayuda imprescindible para todo aquél que quiera escribir programas propios con el COMMODORE.

Consejos y Trucos, Commodore 64. Vol. 2. 259 págs. 2.200,- ptas.

Este libro ofrece al programador interesado una introducción fácilmente comprensible para los tan extendidos Assembler PROFI-ASS, SM MAE y T.E-X.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.

El Ensamblador. 250 páginas. 2.200,- ptas.

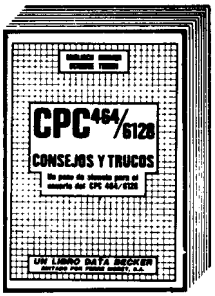
El libro de Primicias del Commodore 128 no ofrece solamente un resumen completo de todas las características y rendimientos del sucesor del C-64 y con ello una importante ayuda para su adquisición. Muestra, además, todas las posibilidades del nuevo equipo en función de sus tres modos de operación.

Todo sobre el nuevo Commodore 128. 250 págs. P.V.P. 2.200,- ptas.

El libro Commodore 128-Consejos y Trucos es un filón para cualquier poseedor del C-128 que desee sacar más partido a su ordenador. Este libro no sólo contiene gran cantidad de programas-ejemplo, sino que además explica de un modo sencillo y fácil la configuración del ordenador y de su programación.

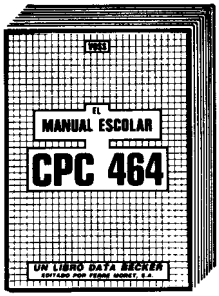
Commodore 128-Consejos y Trucos. 327 págs. 2.800,- ptas.

AMSTRAD



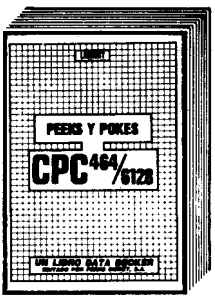
Ofrece una colección muy interesante de sugerencias, ideas y soluciones para la programación y utilización de su CPC-464: Desde la estructura del hardware, sistema de funcionamiento - Tokens Basic, dibujos con el joystick, aplicaciones de ventanas en pantalla y otros muchos interesantes programas como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

CPC-464 Consejos y Trucos. 263 págs. P.V.P. 2.200,- ptas.



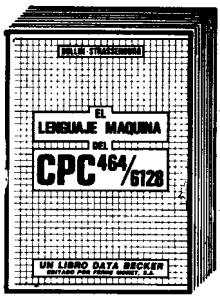
Escrito para alumnos de los últimos cursos de EGB y de BUP, este libro contiene muchos programas para resolver problemas y de aprendizaje, descritos de una forma muy compleja y fácil de comprender. Teorema de Pitágoras, progresiones geométricas, escritura cifrada, crecimiento exponencial, verbos irregulares,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

CPC-464 El libro del colegio. 380 págs. P.V.P. 2.200,- ptas.



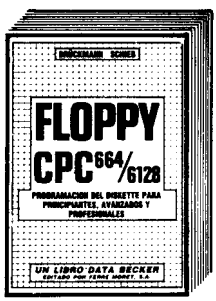
PEEKS, POKES y CALLS se utilizan para introducir al lector de una forma fácilmente accesible al sistema operativo y al lenguaje máquina del CPC. Proporciona además muchas e interesantes posibilidades de aplicación y programación de su CPC.

PEEKS y POKES del CPC 464/6128. 180 pág. P.V.P. 1.600,- ptas.



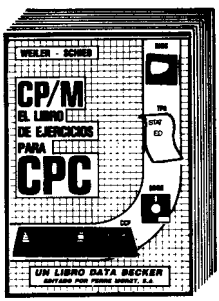
El libro del lenguaje máquina para el CPC 464/6128 está pensado para todos aquellos a quienes no les resulta suficiente con las posibilidades y rapidez del BASIC. Se explican aquí detalladamente las bases de la programación en lenguaje máquina, el funcionamiento del procesador Z-80 con sus respectivos comandos así como la utilización de las rutinas del sistema con abundantes ejemplos. El libro contiene programas completos de aplicación tales como Ensamblador. Desensamblador y Monitor, facilitando de esta manera la introducción del lector en el lenguaje máquina.

El Lenguaje Máquina del CPC 464/6128. 330 pág. P.V.P. 2.200,- ptas.



El LIBRO DEL FLOPPY del CPC lo explica todo sobre la programación con discos y la gestión relativa de ficheros mediante el floppy DDI-1 y la unidad de discos incorporada del CPC 664/6128. La presente obra, un auténtico stándart, representa una ayuda incomparable tanto para el que desee iniciarse en la programación con discos cómo para el más curtido programador de ensamblados. Especialmente interesante resulta el listado exhaustivamente comentado del DOS y los muchos programas de ejemplo, entre los que se incluye un completo paquete de gestión de ficheros.

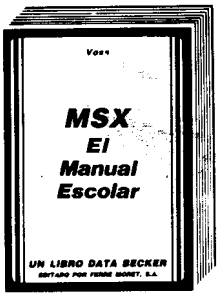
El Libro del Floppy del CPC. 353 pág. P.V.P. 2.800,- ptas.



¡Dominar CP/M por fin! Desde explicaciones básicas para almacenar números, la protección contra la escritura, o ASCII, hasta la aplicación de programas auxiliares de CP/M, así como «CP/M interno» para avanzados, cada usuario del CPC rápidamente encontrará las ayudas e informaciones necesarias, para el trabajo con CP/M. Este libro tiene en cuenta las versiones CP/M 2.2, así como CP/M Plus (3.0), para el AMSTRAD CPC 464, CPC 664 y CPC 6128.

CP/M. El libro de ejercicios para CPC. 260 pág. P.V.P. 2.900,- ptas.

MSX



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 el Manual Escolar. 389 págs. P.V.P. 2.800,- 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.

MSX Programas y Utilidades, 1985, 194 pág. P.V.P. 2.200,- 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 Gráficos y Sonidos, 250 págs. P.V.P. 2.800,- ptas.

Este libro contine una colección sin igual de trucos y consejos para todos los ordenadores con la nueva norma MSX. No sólo contiene las recetas completas, sino también los conocimientos básicos necesarios.

MSX - Consejos y Trucos. 288 págs. P.V.P. 2.200,- ptas.

El libro del Lenguaje Máquina para el MSX está creado para todos aquellos a quienes el BASIC se les ha quedado pequeño en cuanto a rendimiento y velocidad. Desde las bases para la programación en Lenguaje Máquina, pasando por el método de trabajo del Procesador Z-80 y una exacta descripción de sus órdenes, hasta la utilización de rutinas del sistema todo ello ha sido explicado en detalle e ilustrado con múltiples ejemplos en este libro.
El libro contiene, además, como programas de aplicación, un ensamblador un desensamblador y un monitor.

MSX Lenguaje Máquina. 306 págs. 2.200,- ptas.

ZX SPECTRUM

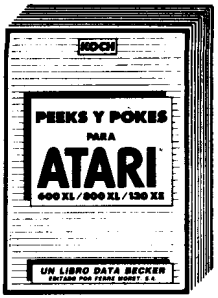
Una interesante colección de sugestivas ideas y soluciones para la programación y utilización de su ZX ESPECTRUM. Aparte de muchos peeks, pokes y USRs hay también capítulos completos para, entre otros, entrada de datos asegurado sin bloqueo de ordenador, posibilidades de conexión y utilización de microdrives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

ZX Spectrum Consejos y Trucos, 211 pág. 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 cuadráticas, movimiento pendular, estructura de moléculas, cálculo de interés y muchas cosas más.

ZX Spectrum el Manual Escolar. 389 págs. P.V.P. 2.200,- ptas.

ATARI



Tan interesante como el tema, es el libro que explica de forma fácilmente comprensible el manejo de Peeks y Pokes importantes, y representa un gran número de Pokes con sus posibilidades de aplicación, incluyendo además programas ejemplo. Al lado de temas como lo son la memoria de la pantalla, los bits y los bytes, el mapa de la memoria, la tabla de modos gráficos o el sonido, también se detalla de forma magnífica la estructura del ATARI 600XL/800XL/130XE.

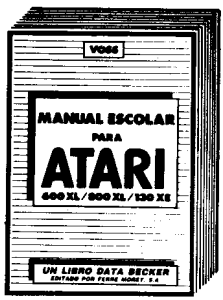
Peeks y Pokes para ATARI 600XL/ 800XL/130XE. 251 pág. P.V.P. 2.200, ptas.

Una lograda introducción al sugestivo tema de los «juegos estratégicos». Desde juegos sencillos con estrategia fija a juegos complejos con procedimientos de búsqueda hasta programas con capacidad de aprendizaje —muchos ejemplos interesantes, escritos por supuesto de forma fácilmente comprensible. Con programas de juegos ampliamente detallados: NIM con un montón, bloqueo, hexapawn, mini-damas y muchos más.

Juegos estratégicos y cómo programarlos en el ATARI 600XL/ 800XL/130XE. 181 pág. P.V.P. 1.600,- ptas.

Jugar a aventuras con éxito y programarlas uno mismo - todo lo verdaderamente importante sobre el tema, lo contiene este guía fascinante que te lleva a través del mundo fantástico de las aventuras. El libro abarca todo el espectro, hasta las más sofisticadas aventuras gráficas llenas de trucos, acompañándolas siempre de numerosos programas ejemplo. Sin embargo la clave —al margen de muchas aventuras para teclear— es un generador de aventuras completo, mediante el cual la programación de aventuras se convierte en un juego de niños.

Aventuras - y cómo programarlas en el ATARI 600XL/800XL/130XE. 284 pág. P.V.P. 2.200,- ptas.



Muchos programas interesantes de soluciones de problemas y de aprendizaje, descritos de forma amplia y comprensible, y adecuados sobre todo para escolares. ¡Aquí el aprendizaje intensivo se convierte de una tarea divertida! Al margen de temas como los verbos irregulares, o las ecuaciones de segundo grado, un resumen corto de las bases del tratamiento electrónico de datos, y una introducción a los principios del análisis de problemas, completan este libro que debería obrar en posesión de cualquier escolar.

El libro escolar para ATARI 600XL/ 800XL/130XE. 389 pág. P.V.P. 2.800,- ptas.

OTROS TITULOS

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. 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 informàtica. Abarca desde los métodos de programación clásicos a los más modernos.

Metodología de la Programación. 250 págs. P.V.P. 2.200,- ptas.

La técnica y programación del Procesador Z80 son los temas de este libro. Es un libro de estudio y de consulta imprescindible para todos aquellos que poseen un Commodore 128, CPC, MSX u otros ordenadores que trabajan con el Procesador Z80 y desean programar en lenguaje máquina.

El Procesador Z80. 560 pág. P.V.P. 3.800,- ptas.

El tema de este libro es la técnica y programación de los procesadores de la familia 68000. Es una obra de consulta indispensable, un manual para todo programador que quiera utilizar las ventajas del 68000.

Técnica y programación para el procesador 68000. 518 págs. P.V.P. 3.800,- ptas.

EL CONTENIDO:

Este libro contine una colección sin igual de trucos y consejos para todos los ordenadores con la nueva norma MSX. No sólo contiene las recetas completas, sino también los conocimientos básicos necesarios.

Extracto del contenido:

— La norma MSX
— Programación de gráficas
— Generador de caracteres
— Ventanas
— Copiador de gráficos y texto
— Gráfica tridimensional
— Editor de Sprites
— Módulos de entrada/salida
— Programación de sonido
— Programación de un sintetizador y miniórgano
— Lenguaje máquina
— Monitor
— Utilización de rutinas de sistema
— Variable muda
— Generador de menú
— Minitextomat
y mucho más

ESTE LIBRO HA SIDO ESCRITO POR:

Holger Dullin y Hardy Brassenburg son estudiantes de Informática y Biologia, programadores profesionales y autores de numerosos libros (Lenguaje máquina para MSX y CPC).

ISBN 84-86437-39-3